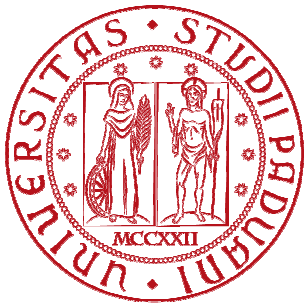
UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI



LAUREA TRIENNALE IN INGEGNERIA GESTIONALE

RICAVO DI ALBERI E CAMMINI OTTIMI SUI GRAFI: SOFTWARE SPECIFICI

RELATORE: Ch.mo Prof. GIORGIO ROMANIN JACUR

LAUREANDO: DAVIDE SFERRAZZA

ANNO ACCADEMICO 2010-2011

RINGRAZIAMENTI

I miei ringraziamenti più sinceri vanno al mio relatore prof. Giorgio Romanin Jacur per la disponibilità e i suggerimenti ricevuti durante lo svolgimento della tesi.

Inoltre desidero ringraziare la mia Famiglia per il sostegno, l'incoraggiamento e la fiducia dimostrata durante tutti questi miei anni di studio.

Infine ringrazio la mia fidanzata Valentina e i miei Amici tutti, per il loro autentico e caloroso supporto morale.

SOMMARIO

La teoria dei grafi, con i relativi problemi di alberi e cammini minimi, è ad oggi studiata ed applicata in molteplici campi scientifici e tecnici. Dall'ubicazione e la comunicazione si arriva a problemi in ambito informatico, economico, scientifico e sociale. La diffusione quindi della teoria pone l'attenzione sugli strumenti più adatti alla sua trattazione più efficace: la soluzione proposta dall'uso di software *open source*, programmi resi disponibili per diffusione sul Web e con lo scopo di uno sviluppo condiviso delle funzionalità, permette lo studio pratico e la risoluzione efficiente dei problemi suddetti. Nella prima parte di questo documento si presenta un'introduzione alla teoria dei grafi tramite le fondamentali definizioni, rappresentazioni e proprietà: di seguito si illustrano i problemi di albero e cammino minimi, le loro applicazioni e gli algoritmi più esaurienti, utilizzati inoltre nei software trovati.
Nella seconda parte si discutono i punti che hanno segnato la ricerca dei programmi, le loro caratteristiche principali, le variabili-qualità volute e le diverse fonti da cui sono stati reperiti. I software scelti vengono quindi descritti direttamente, nello specifico delle funzioni e delle peculiarità principali, passando per i livelli prestazionali di progetto, la conformazione strutturale e il layout grafico.

INDICE

Ricavo di alberi e cammini ottimi sui grafi: software specifici

1. Introduzione

La teoria dei grafi trova il suo importante spazio nel campo della Ricerca Operativa, in quanto modellizzazione efficace per scelte decisionali e problemi diffusi: maggior interesse si è sviluppato per la trattazione dei problemi di minimo (o di massimo) sotto opportuni vincoli posti dalla situazione presa in esame, dovuti a restrizioni ambientali, economiche o funzionali. L'utilizzo di modelli come i grafi dalla costruzione semplice e regolati da chiare regole e proprietà è coadiuvato dall'elaborazione di algoritmi risolutivi sempre più efficaci e di apprendimento intuitivo: ne emerge dunque un'ampia capacità di adattamento ad ambiti più o meno tecnici e scientifici.
L'applicabilità della teoria trova ormai diffusione a partire dalle problematiche d'ubicazione e comunicazione tradizionali fino a settori come il chimico, l'informatico, il sociologico ed il decisionale-organizzativo. Sorge dunque la necessità di sfruttare a pieno le potenzialità di questo modello, indagando sui mezzi sviluppati dalla tecnologia informatica ad oggi disponibile: lo scopo dell'analisi proposta è volto alla ricerca di software *open source*, accessibili e manipolabili liberamente, le cui funzionalità siano basate sulla teoria dei grafi. L'avvicinamento a questi elementi informatici è dato dai vantaggi di interattività ed immediatezza di risultati, così come dall'efficienza e la gestione dei dati inseriti.
Attraverso gli esempi dei programmi trovati, l'indagine presenta un orizzonte degli strumenti offerti sul Web per la risoluzione dei problemi di albero e cammino minimo, illustrandone tipologie di progetto, impostazioni e modalità di utilizzo. L'efficacia degli algoritmi, la chiarezza nel layout, il materiale a supporto e la facilità nella reperibilità sono alcune variabili principali con cui si è operata la ricerca: dopo l'individuazione preliminare degli elementi più appropriati, effettuata poi la cernita finale, si è provveduto ad un'analisi descrittiva dell'uso e delle caratteristiche proprie per singolo software.
Nel seguito si procede innanzitutto ad un'introduzione di base alla teoria dei grafi, con le necessarie definizioni e le rappresentazioni fondamentali, per poi concentrare l'attenzione direttamente sui problemi di albero e cammino minimi, con i loro campi di utilizzo e i più noti algoritmi risolutivi adottati. Infine maggior spazio verrà lasciato all'illustrazione dei software scelti, in tutti i loro aspetti tecnici e le loro limitazioni nell'implementazione pratica o di progetto.

2. Definizioni e proprietà fondamentali dei Grafi

Si rende necessario il richiamo di alcune definizioni e proprietà che stanno alla base della teoria dei grafi. A tali assunti così definiti verrà fatto riferimento, di qui in avanti, nelle analisi e considerazioni fatte.

Definizione 1. *Un* **grafo orientato** *G è una coppia (V, E), dove V è un insieme definito ed E è una relazione binaria su V. L'insieme V è chiamato l'***insieme dei vertici** *(o insieme dei nodi) di G ed i suoi elementi sono detti* **vertici** *(o* **nodi***). L'insieme E è chiamato l'***insieme degli spigoli** *(o* **insieme degli archi***) di G ed i suoi elementi sono detti* **spigoli** *(o* **archi***).*

Definizione 2. *In un* **grafo non orientato** *G = (V, E), l'insieme degli spigoli E è costituito da coppie non ordinate di vertici, piuttosto che da coppie ordinate; cioè, uno spigolo è un insieme {u, v}, dove u, v ∈ V e u ≠ v. Per convenzione si usa la notazione (u, v) per uno spigolo, piuttosto che la notazione insiemistica {u, v}; inoltre (u, v) e (v, u) sono considerati essere lo stesso spigolo. Si definisce* **aciclo** *(o* **loop***) uno spigolo in cui u = v.*

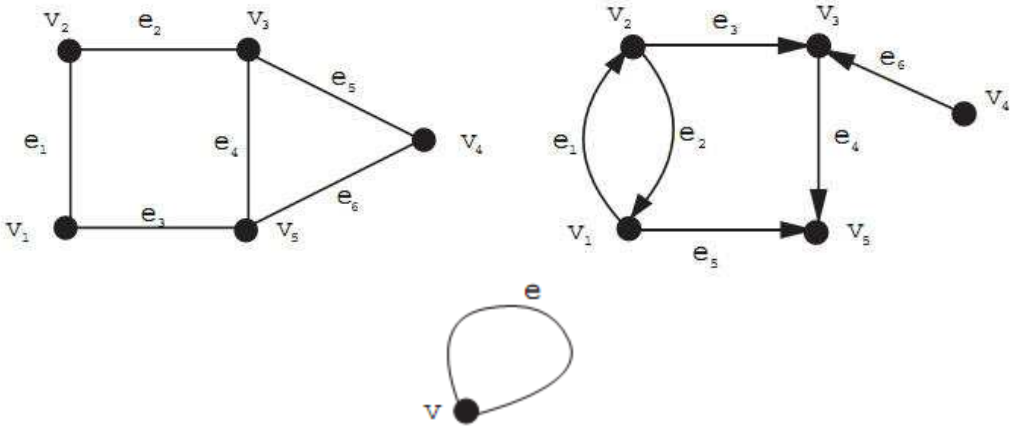
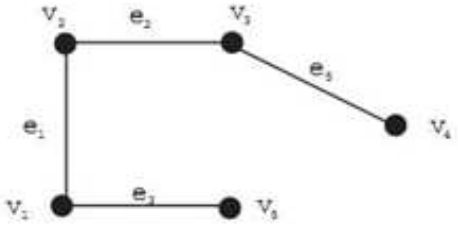


Figura 1: Grafo non orientato, grafo orientato e loop

Definizione 3. *Se (u, v) è uno spigolo di un grafo orientato, si dice che (u, v) è* **incidente** *o* **esce dal** *vertice u ed è* **incidente** *o* **entra nel** *vertice v. Se (u, v) è uno spigolo di un grafo non orientato, si dice che (u, v) è incidente sui vertici u e v.*

Definizione 4. Se (*u, v) è uno spigolo di un grafo G = (V, E), si dice che il vertice v è* **adiacente** *al vertice u. Quando il grafo è non orientato la relazione di adiacenza è simmetrica, mentre quando è orientato la relazione non è necessariamente simmetrica. Se v è adiacente ad u in un grafo orientato talvolta si scrive* $u \rightarrow v$.

Definizione 5. *Un* **cammino** *di* **lunghezza** *k da un vertice u ad un vertice u' in un grafo G = (V, E) è una sequenza* $\langle v_1, v_2, v_3, \ldots, v_k \rangle$ *di vertici tale che* $u = v_0$, $u' = v_k$ *e* $(v_{i-1}, v_i) \in E$ *per* $i = 1, 2, \ldots, k$. *La lunghezza di un cammino è il suo numero di spigoli. Il cammino* **contiene** *i vertici* $v_0, v_1, v_2, \ldots, v_k$ *e gli spigoli* $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$. *Se vi è un cammino p da u a u' , si dice che u' è* **raggiungibile** *da u tramite p, ciò talvolta si scrive come* $u \overset{p}{\rightsquigarrow} v$. *Un cammino è* **semplice** *se tutti i vertici sono distinti.*



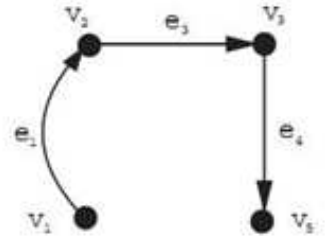


Figura 2: Cammino semplice *Figura 3: Cammino orientato da* v_1 *a* v_5

Definizione 6. *Si definisce la* **distanza di cammino minimo** $\delta(u, u')$ *da u a u' come il numero di spigoli del cammino più breve dal vertice u al vertice u', oppure* ∞ *se non esiste nessun cammino da u a u'. Un cammino di lunghezza* $\delta(u, u')$ *da u a u' è chiamato* **cammino minimo** *da u a u' .*

Definizione 7. *In un grafo orientato, un cammino* $\langle v_1, v_2, v_3, \ldots, v_k \rangle$ *forma un* **ciclo** *se* $v_0 = v_k$ *ed il cammino contiene almeno uno spigolo. Il ciclo è* **semplice**, *se* $v_1, v_2, v_3, \ldots, v_k$ *sono distinti. In un grafo non orientato, un cammino* $\langle v_1, v_2, v_3, \ldots, v_k \rangle$ *forma un* **ciclo** *se* $v_0 = v_k$ *e* $v_1, v_2, v_3, \ldots, v_k$ *sono distinti. Un grafo senza cicli è* **aciclico**. *Spesso si usano le prime lettere del nome inglese "direct acyclic graph" (dag) per indicare un grafo orientato aciclico.*

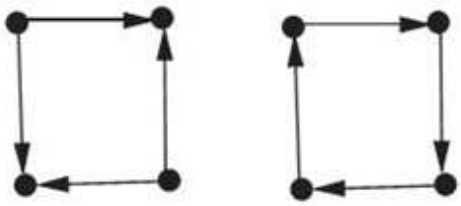


Figura 4: Ciclo e ciclo orientato

Definizione 8. *Un grafo non orientato è* **connesso** *se ogni coppia di vertici v, u ∈ V è collegata con un cammino. Le* **componenti connesse** *di un grafo sono le classi di equivalenza dei vertici sotto la relazione "raggiungibile da".*

Definizione 9. *Un grafo orientato è* **fortemente connesso** *se ogni coppia di vertici v, u ∈ V è collegata con un cammino. Le* **componenti fortemente connesse** *di un grafo sono le classi di equivalenza dei vertici sotto la relazione "mutuamente raggiungibile".*

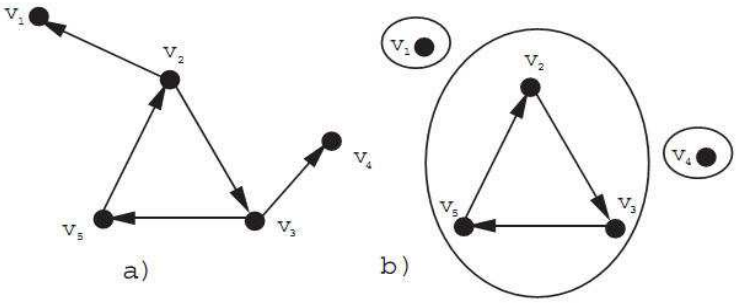


Figura 5: a) Grafo orientato b) Componenti fortemente connesse

Definizione 10. *Un grafo aciclico e non orientato è una* **foresta** *e un grafo connesso aciclico e non orientato è un* **albero** *(***libero***).*

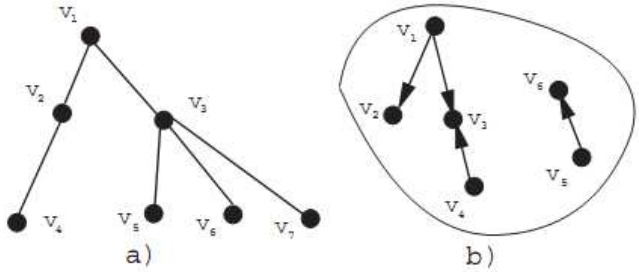


Figura 6: a) Albero su grafo aciclico, connesso e non orientato b) Foresta

Il seguente teorema raccoglie molte importanti proprietà che riguardano gli alberi.

Teorema 1. *Sia G = (V ,E) un grafo non orientato. Le seguenti affermazioni sono equivalenti.*

1. *G è un albero libero.*
2. *Due vertici qualsiasi in G sono connessi da un unico cammino semplice.*
3. *G è connesso, ma se un qualunque spigolo è rimosso da E, il grafo risultante è sconnesso.*
4. *G è connesso e |E| = |V | - 1.*
5. *G è aciclico e |E| = |V | - 1.*

6. G è aciclico, ma se un qualsiasi spigolo è aggiunto ad E, il grafo risultante contiene un ciclo..

Definizione 11. *Un* **albero radicato** *o (***arborescenza***) T è un grafo orientato senza cicli che soddisfa le seguenti tre proprietà:*

1. C'è soltanto un vertice, detto **radice**, *con nessuno spigolo entrante.*

2. Ogni vertice, escluso la radice, ha esattamente uno spigolo entrante.

3. C'è un (unico) cammino dalla radice ad ogni vertice.

Definizione 12. *Si consideri un vertice x di un albero radicato T con radice r; qualunque vertice y sull'unico cammino da r a x è chiamato* **antenato** *di x e se y è antenato di x, allora x è un* **discendente** *di y (ogni vertice è sia un antenato che un discendente di se stesso). Se y è un antenato di x, e $x \neq y$, allora y è un* **antenato proprio** *di x e x è un discendente proprio di y. Il* **sottoalbero con radice in x** è *l'albero indotto dai discendenti di x, radicato in x.*

Definizione 13. *Se l'ultimo spigolo di un cammino dalla radice r di un albero T ad un vertice x è (y, x), allora y è il* **padre** *di x e x è il* **figlio** *di y; la radice è l'unico vertice in T che non ha padre. Se due vertici hanno lo stesso padre, si dicono* **fratelli**. *Un vertice senza figli si dice* **foglia**. *Un vertice non foglia è un* **vertice interno**.

Definizione 14. *Il numero di figli di un vertice x in un albero T è chiamato il* **grado** *di x. La lunghezza del cammino dalla radice r ad un vertice x è la* **profondità** *di x in T; la profondità più grande di un qualsiasi vertice in T è l'***altezza** *di T.*

Definizione 15. *Dato un grafo orientato G = (V, E) ed un vertice $x \in V$, un* **albero dei cammini in avanti radicato in x** *è un albero $T_f(x) = (X, S)$ radicato in x tale che*

1. X è l'insieme dei discendenti di x in G;

2. $S \subseteq E$.

Allo stesso modo, un **albero dei cammini indietro radicato in x** *è un albero $T_b(x) = (X, S)$ radicato in x tale che*

1. X è l'insieme degli antenati di x in G;

*2. $S \subseteq E^R$, dove $E^R = \{ (y, x) \mid (x, y) \in E \}$ è detto l'***insieme degli spigoli inversi**.

Definizione 16. *Un albero dei cammini in avanti (indietro) T = (X, S) radicato in x è di* **lunghezza minima** *(o equivalentemente è un* **albero dei cammini minimi in avanti (indietro)***) se per ogni vertice $v \in X$, la profondità di v in T coincide con la lunghezza del cammino minimo da x a v in G.*

In un *problema di cammini minimi* viene fornito un grafo orientato e pesato G = (V, E), con una funzione peso $\omega : E \rightarrow \mathbb{R}$ che associa ad ogni spigolo un peso a valore nei reali.

Definizione 17. *Il* **peso** *di un cammino* $p = \langle v_1, v_2, v_3, \ldots, v_k \rangle$ *è la somma dei pesi degli spigoli che lo costituiscono:*

$$\omega(p) = \sum_{i=1}^{k} \omega(v_{i-1}, v_i).$$

Definizione 18. *Il* **peso di cammino minimo** *da u a v è definito come*

$$\delta(u, v) = \begin{cases} min\{\ \omega(p) : u \overset{p}{\rightsquigarrow} v\} \text{ se e solo se esiste un cammino da u a v,} \\ \infty \text{ altrimenti.} \end{cases}$$

Definizione 19. *Un* **cammino minimo** *(***Shortest Path***,* **SP***) dal vertice u al vertice v è definito come un qualunque cammino p con peso* $\omega(p) = \delta(u, v)$.

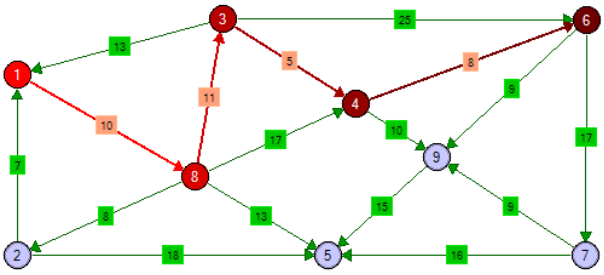


Figura 7: Cammino minimo (SP) segnato in rosso, dal nodo 1 al nodo 6

Definizione 20. *L'***arborescenza minima** *(***di costo minimo***) o* **albero dei cammini minimi** *in un grafo G = (V, E) orientato è l'arborescenza T di radice r secondo cui, per ogni* $v \in V$, *il cammino orientato da r a v in T è il cammino di costo minimo in G.*

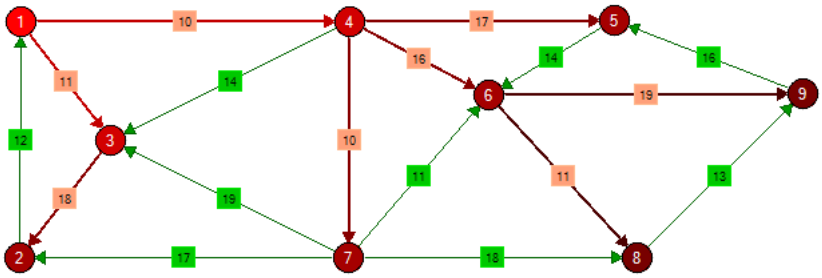


Figura 8: Arborescenza minima, segnata in rosso, che ha come radice il nodo 1

Definizione 20. *Un* **sottografo** *di un grafo G = (V, E) è un grafo G' = (V', E') con insieme dei vertici* $V' \subseteq V$, *insieme degli spigoli* $E' \subseteq E$ *e le cui relazioni di adiacenza per gli spigoli rimangono invariate rispetto alle rispettive in G.*

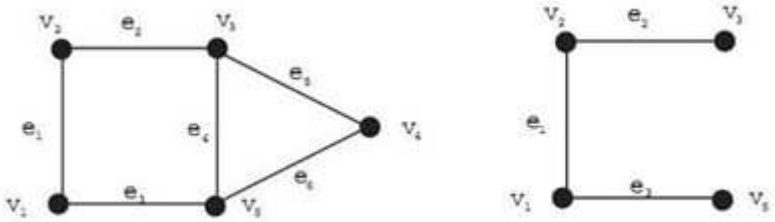


Figura 9: Grafo ed esempio di suo sottografo

In un *problema di albero minimo* viene fornito invece un grafo non orientato e pesato G = (V, E), con ancora una funzione peso $\omega : E \rightarrow \mathbb{R}$.

Definizione 21. Si definisce **albero ricoprente** i*n un grafo G = (V, E) connesso e non orientato un sottografo che è un albero composto da tutti i vertici di V e alcuni (oppure tutti) gli spigoli di E. In un grafo pesato, l'***albero ricoprente minimo** (**Minimum Spanning Tree**, **MST**) *è l'albero ricoprente il cui peso (definizione analoga al peso di un cammino) è inferiore o al massimo uguale al peso di ogni altro eventuale albero ricoprente.*

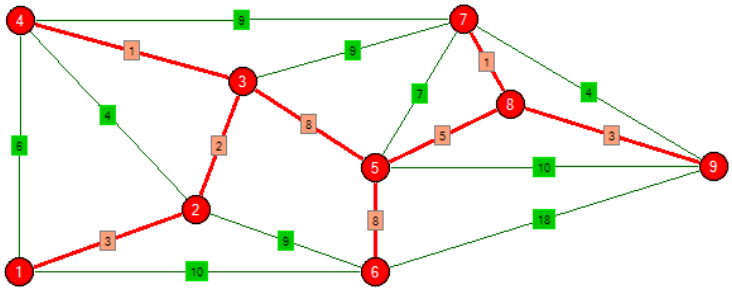


Figura 10: Albero minimo(MST) segnato in rosso, di un grafo pesato non orientato

2.1. Alcune osservazioni

I pesi degli spigoli possono essere interpretati come misure diverse da semplici distanze: essi sono usati spesso per rappresentare tempi, costi, penalità, perdite o qualunque quantità o misura di rapporto tra due elementi che si accumuli in maniera lineare e si ha interesse a minimizzare.
Pesi dal valore negativo si possono realizzare e sono valori che rappresentano un vantaggio, una remunerazione o un premio nell'attraversamento dell'arco.
Si osserva che un grafo non pesato può essere considerato come un grafo in cui ogni spigolo ha peso unitario.
In un grafo non orientato è possibile considerare uno spigolo come due archi orientati entranti e uscenti mutuamente sui due nodi dello spigolo, mantenendo per essi peso uguale al non orientato. Così facendo si può passare da un grafo non orientato ad una rappresentazione dello stesso come grafo orientato.

3. Rappresentazioni di un Grafo

Un grafo G = (V, E), con V insieme dei nodi ed E insieme degli archi, può essere rappresentato in molti modi diversi. Finora nella sezione precedente già sono state introdotte due modalità: la rappresentazione estensiva e la rappresentazione grafica. Si intende ora porre l'attenzione sulle differenze di applicazione e convenienza che contraddistinguono scelta ed utilizzo delle rappresentazioni possibili.

3.1. Rappresentazione estensiva

La più semplice e formale rappresentazione è l'estensiva, in cui vengono elencati tutti gli elementi dell'insieme V e tutti quelli dell'insieme E. Consideriamo questo esempio:

$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$
$E = \{e_1 = (v_1,v_2), e_2 = (v_1,v_3), e_3 = (v_3,v_2), e_4 = (v_2,v_4), e_5 = (v_3,v_5), e_6 = (v_4,v_5), e_7 = (v_4,v_6)\}$

Spesso, nel riportare la rappresentazione estensiva di un grafo, si identificano, per comodità, i vertici con i primi *n* numeri naturali. In questo caso la rappresentazione estensiva del grafo diventa:

V = {1, 2, 3, 4, 5, 6}
E = {(1, 2), (1, 3), (3, 2), (2, 4), (3, 5), (4, 5), (4, 6)},

e spesso ci si limita a fornire il valore di *n*, cioè il numero di nodi e la lista degli archi.

3.2. Rappresentazione grafica

Nella rappresentazione grafica il grafo viene individuato mediante un disegno nel quale ai nodi si fanno corrispondere tendenzialmente dei piccoli cerchi o dei simboli specifici identificativi, mentre agli archi dei tratti di retta o curva congiungenti tali nodi.

Già da questi due esempi si capisce che, benché concettualmente equivalenti, questi modi diversi di rappresentare un grafo possono avere caratteristiche molto differenti. Per esempio, la rappresentazione grafica è molto immediata e fornisce subito una visione intuitiva delle caratteristiche del grafo, e molta della terminologia introdotta finora è ispirata proprio da tale rappresentazione (si pensi ad esempio alla definizione di albero e foresta). Tuttavia, se, come spesso accade nella pratica, il grafo ha un alto numero di nodi (per esempio migliaia) si intuisce come la rappresentazione grafica non sia più di pratica utilità.
Sorge allora l'evidente esigenza di avere a disposizione diverse tipologie di rappresentazione di un grafo. La decisione di scegliere una forma piuttosto che un'altra andrà poi fatta di volta in volta in base al problema che si vuole risolvere, all'utilità, alle dimensioni del grafo, alle sue caratteristiche etc.

3.3. Matrice di adiacenza, matrice di incidenza e lista di adiacenza

Passiamo allora a descrivere altri tre modi molto utili e usati per rappresentare un grafo:

1. matrice di adiacenza;
2. matrice di incidenza;
3. lista di adiacenza.

La prima e la terza modalità sono le più diffuse per la memorizzazione di grafi.
La prima è preferibile nel caso di grafi con un elevato numero di archi rispetto al numero di nodi (per esempio, con m = numero archi e n = numero nodi, quando $m \approx n^2$) in quanto, pur essendo in questo caso particolare la prima e la terza modalità equivalenti dal punto di vista dell'efficienza computazionale, la rappresentazione in termini di matrice di adiacenza è più immediata e fornisce una rappresentazione del grafo più efficace.
La terza è preferibile nel caso di grafi "sparsi", in cui cioè i numero di archi è piccolo rispetto al numero di nodi (per esempio quando $m \approx n$, come negli alberi).
La seconda, pur se meno efficace delle altre dal punto di vista computazionale, è preferibile in alcune applicazioni della Ricerca Operativa per la sua corrispondenza ad alcune formulazioni standard di modelli di ottimizzazione e la conseguante facilità di analisi del modello.

3.3.1. Matrice di adiacenza (o connessione)

La matrice di adiacenza è basata su di un a matrice quadrata n x n. Il generico elemento (i, j) della matrice sarà pari a 1 se l'arco (i, j) del grafo esiste, sarà pari a 0 se l'arco (i, j) non esiste. In questo modo si possono memorizzare grafi sia orientati che non orientati.

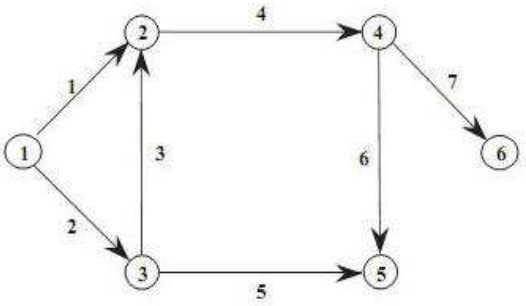
Nel caso di grafi non orientati la matrice di adiacenza risulta essere simmetrica, poiché (i, j) e (j, i) sono lo stesso arco.

3.3.2. Matrice di incidenza

Questa modalità di rappresentazione è basata su di una matrice *n* x *m*. La matrice ha quindi un numero di righe pari al numero di nodi e un numero di colonne pari al numero di archi.
Nel caso di grafi non orientati, il generico elemento (i, j) della matrice sarà pari a 1 se il j-esimo arco del grafo incide sul nodo i, sarà pari a 0 se l'arco j-esimo non incide sul nodo i.
Nel caso di grafi orientati, il generico elemento (i, j) della matrice sarà pari a -1 se l'arco j-esimo esce dal nodo i, sarà pari a 1 se l'arco j-esimo entra nel nodo i, oppure pari a 0 se l'arco j-esimo non incide sul nodo i.
Nella matrice di incidenza ogni colonna ha esattamente due elementi diversi da zero; essi sono in corrispondenza delle righe della matrice relative ai due nodi estremi dell'arco.

3.3.3. Lista di adiacenze

La rappresentazione per lista di adiacenze è basata su una lista di nodi del grafo, in cui per ogni nodo vengono elencati i nodi ad esso adiacenti.
Questa è in genere la più efficiente tra le modalità dal punto di vista computazionale, specialmente per grafo con pochi archi.



Rappresentazione grafica

1 : 2 , 3	archi da 1 a 2 e da 1 a 3
2 : 4 .	arco da 2 a 4
3 : 2 , 5	archi da 3 a 2 e da 3 a 5
4 : 5 , 6	archi da 4 a 5 e da 4 a 6
5 :	nessun arco
6 :	nessun arco

Lista di adiacenze

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

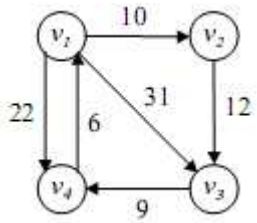
Matrice di connessione

$$\begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrice di incidenza

Figura 11: Esempi delle rappresentazioni di grafi sopra elencate

Può risultare conveniente anche definire la **matrice dei pesi**, matrice quadrata di ordine *n* x *n*. Ogni elemento (i, j) di tale matrice sarà il peso dell'arco (i, j), così che sia possibile con essa individuare sia la disposizione degli archi tra i nodi, sia i loro valori peso. Qualora l'arco (i, j) non fosse presente si pone al posto dell'arco il simbolo ∞.
In questo modo si possono memorizzare grafi sia orientati che non orientati: nel caso di grafi non orientati la matrice dei pesi risulta essere simmetrica, poiché (i, j) = (j, i) con medesimo peso.



$$\begin{bmatrix} \infty & 10 & 31 & 22 \\ \infty & \infty & 12 & \infty \\ \infty & \infty & \infty & 9 \\ 6 & \infty & \infty & \infty \end{bmatrix}$$

Figura 12: Esempio di grafo orientato e relativa matrice dei pesi

4. Problemi di calcolo su grafi orientati e non orientati

A questo punto si intende focalizzare l'attenzione sui problemi di *albero ricoprente minimo*, *cammino minimo* e *arborescenza minima* in un'analisi su grafi pesati, orientati e non orientati. L'importanza di questi problemi deriva dalla diffusa possibilità di accostamento della teoria dei grafi a problematiche dalla natura diversa. In ambiti scientifici, informatici, economici, sociali e decisionali questi tre problemi trovano molteplici applicazioni: nella tabella seguente sono indicati solamente alcuni degli esempi che si prestano ad una modellizzazione su grafi.

GRAFO	NODI	ARCHI
Comunicazione	telefoni, computers	cavi in fibre ottiche
Composti chimici	molecole	legami chimici
Meccanica	Giunti	aste e corpi rigidi, molle
Idraulica	serbatoi, impianti di sollevamento	tubazioni
Finanza	azioni, moneta corrente	transazioni
Trasporti	incroci stradali, aeroporti	strade, rotte aeree
Circuiti	porte, memorie, processori	cavi e fili elettrici
Software di sistema	funzioni	chiamate di funzioni
Internet	pagine web	hyperlinks
Giochi	posizioni dei pezzi	mosse consentite
Relazioni sociali	persone, attori, terroristi	amicizia, casts di film, associazioni
Reti di interazione di proteine	proteine	interazioni proteina-proteina
Reti di geni regolatori	Geni	interazioni tra geni regolatori
Reti neurali	neuroni	sinapsi
Malattie infettive	persone	contagi
Rete di energia elettrica	stazioni di trasmissione	cavi
Scheduling e Programmazione	compiti	vincoli di precedenza

Si procede ora con l'illustrazione delle caratteristiche, le applicazioni specifiche e gli algoritmi relativi ai problemi di albero, arborescenza e cammino minimi.

4.1. Albero ricoprente minimo

Il MST si dimostra importante in svariate applicazioni. Ecco alcuni esempi specifici:

- progettazione di reti:
  - telefoniche, elettriche, idrauliche, stradali, TV via cavo, computer;
- algoritmi di approssimazione per NP-hard problems (problemi "difficili nondeterministici in tempo polinomiale"):
  - travelling salesman problem (il problema del commesso viaggiatore), minimum Steiner tree problem (il problema dell'albero di Steiner);
- applicazioni indirette:
  - minimum bottleneck paths (cammini minimi con collo di bottiglia);
  - codici LDPC per la correzione di errori (trasmissione segnali numerici, sequenze simboli binari);
  - classificazione di immagine (processo di trasformazione di diversi insiemi di informazioni in un sistema di coordinate) con il concetto di entropia di Renyi;
  - apprendere i tratti caratterizzanti nel riconoscimento facciale in tempo reale;
  - ridurre il deposito di informazioni nella messa in sequenza degli aminoacidi in una proteina;
  - model locality dell'interazione delle particelle nei moti fluidi turbolenti;
  - protocollo di configurazione automatica di collegamento Ethernet per evitare cicli un una rete;
- analisi dei cluster, genetica e patologica;
- modellizzazione di rapporti e interazioni sociali.

L'attenzione è posta quindi sul problema dell'*albero ricoprent*e (o *di connessione*) *di peso minimo* (*MST*) che fa capo a grafi non orientati. Il MST da trovare, come già esposto dalle definizioni, dovrà essere un percorso che contiene tutti i vertici, dovrà essere privo di cicli e inoltre la somma dei costi di connessione dovrà essere la minima tra tutti gli alberi possibili.
I principali algoritmi per risolvere in un tempo accettabile tale problema sono caratterizzati da una complessità computazionale relativamente bassa. Risultano infatti più interessanti gli algoritmi utilizzabili su software e linguaggi di programmazione con la minore difficoltà possibile sia a livello di implementazione sia di finale comprensione e utilizzazione per l'utente.
Si presentano ora i due algoritmi comunemente usati per il calcolo del MST di un grafo, nonché quelli su cui si baseranno i software e i programmi di seguito: l'*algoritmo di Kruskal* e l'*algoritmo di Prim*. Entrambi fanno parte della cosiddetta categoria degli algoritmi *greedy*: essi cercano di ottenere una soluzione ottima da un punto di vista globale attraverso la scelta della soluzione più "golosa" ("aggressiva" o "avida", a seconda della traduzione preferita del termine *greedy* dall'inglese) ad ogni passo locale.
Come strumento didattico, gli algoritmi per trovare l'albero minimo forniscono un'esplicitazione grafica del fatto che gli algoritmi *greedy* possono dare soluzioni ottime in maniera comprovata.
Con questi algoritmi si ottiene dunque il MST, ma se un grafo ne possedesse molteplici potrebbero non trovare un MST comune, causa la diversa modalità di procedere. Essi inoltre possono venire utilizzati per determinare se un grafo è connesso o qual è il suo numero di componenti connesse.

4.1.1. *Algoritmo di Kruskal*

L'algoritmo prende il nome dallo statistico matematico Joseph B. Kruskal, e fu stilato nel 1956: si assume per semplicità che nel grafo considerato esista uno e unico MST: senza questa ipotesi non si preclude la possibilità di sviluppo dell'algoritmo però si evitano complessità nella formulazione ai fini dello scopo da raggiungere.

Il procedimento che porta alla scelta del MST del grafo non orientato è il seguente: gli archi vengono ordinati a seconda del loro peso; considerati i pesi dal minore al maggiore, vengono inseriti gli archi in tale ordine di peso nella soluzione che si sta costruendo, se non si evidenzia via via la formazione di cicli con archi precedentemente scelti; nel momento in cui non vi sia più possibilità di inserimento di archi, e ogni nodo è presente nell'albero trovato, si è giunti al MST del grafo.
Si nota come ad ogni step, nel caso di archi inseribili della soluzione con peso uguale, è indifferente la scelta di uno piuttosto che l'altro.
La presenza di eventuali pesi di valore negativo è tollerata: tali valori sono interpretati come vantaggi di percorrenza dell'arco e saranno preferibili ai pesi con valore positivo.

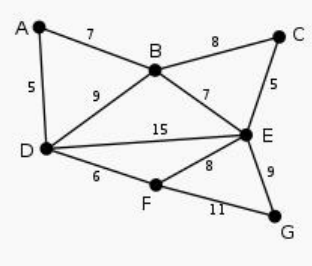
Per l'implementazione di questo algoritmo si può seguire questa serie di passi:

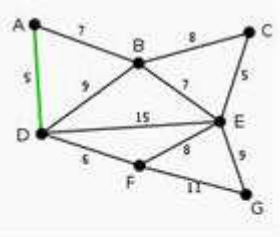
- si crei una foresta F (un insieme di alberi), dove ogni vertice del grafo in partenza è un diverso albero;
- si crei un insieme S contenente tutti gli archi del grafo;
- finché S è un insieme non vuoto e F non contiene un albero collegato:
    - rimuovere un arco con il peso minimo da S;
    - se tale arco collega due alberi diversi, allora aggiungerlo alla foresta, combinando i due alberi in un unico albero;
    - altrimenti scartare tale arco.

Al termine dell'algoritmo, la foresta ha un unico componente ed esso consiste nell'albero di peso minimo del grafo considerato.

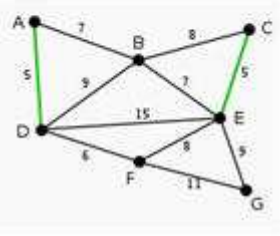
4.1.2. Esempio di applicazione di algoritmo di Kruskal

Passiamo allora ad un esempio applicativo, prendendo in considerazione il grafo di figura seguente:

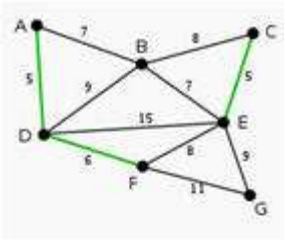




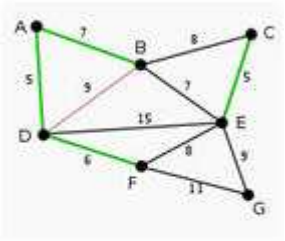
Vediamo come i pesi siano segnati a fianco dei rispettivi archi.
Gli archi dal peso minore sono (A, D) e (C, E), entrambi di costo 5. Arbitrariamente si sceglie (A, D), segnato in verde da inserire nel MST da costruire.



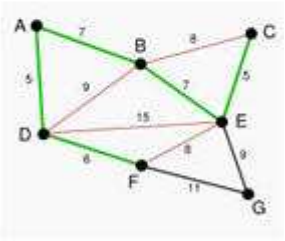
Ora è (C, E) l'arco dal peso minore: esso non forma nessun ciclo con (A, D), allora lo inseriamo nell'albero minimo come secondo arco e lo evidenziamo.



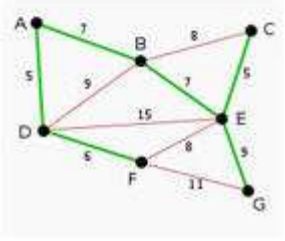
Allo stesso modo evidenziamo ora (D, F) di peso 6.



I prossimi archi da peso minimo sono (A, B) e (B, E), entrambi di peso 7. Arbitrariamente inserisco (A, B).
Osservo adesso l'arco (B, D): un percorso tra i nodi B e D risulta già esistente lungo il tragitto segnato in verde passante per A. Se inserissimo (B, D) si creerebbe un ciclo (ABD), dunque lo segnamo in rosso e lo scartiamo.



Si procede evidenziando (B, E) di peso 7.
Ora si nota come sia necessario escludere altri archi: (B, C) che formerebbe il ciclo (BCE); (E, F) che formerebbe (ABEFD);
(D, E) che formerebbe (ABED).



Infine si inserisce l'arco (E, G) di peso 9 e si osserva che (F, G) deve essere scartato altrimenti creerebbe il ciclo (ABEGFD). Otteniamo quindi il MST del grafo, poiché tutti i nodi sono stati collegati dall'albero color verde: la sua lunghezza è di 39.

4.1.3. Algoritmo di Prim

L'algoritmo di Prim è anche detto algoritmo di Jarník-Prim dal nome del matematico ceco Vojtěch Jarník, che lo sviluppò nel 1930, e dallo statunitense Robert C. Prim, informatico che indipendentemente lo ripropose nel 1957.
Per la descrizione dell'algoritmo si assume in ingresso che il grafo sia rappresentato utilizzando una lista di adiacenze degli archi, considerando il peso di ciascuno di essi, oppure la sua matrice di adiacenza.
Lo sviluppo della procedura che porta al MST è il seguente: si individua un nodo arbitrariamente, preso come nodo di partenza per la costruzione dell'albero minimo; da questo nodo si considerano gli archi che lo connettono ai nodi adiacenti e si sceglie quello dal peso minore (nel caso di archi dai pesi minimi uguali la scelta è arbitraria); si prosegue considerando sempre l'arco dal peso minore tra quelli che raggiungono nodi adiacenti ai nodi già inseriti nell'albero in costruzione, senza ammettere archi che creano cicli; nel momento in cui tutti i nodi sono connessi nell'albero di costo minimo la soluzione è stata raggiunta.
Sono tollerati anche per questo algoritmo pesi di archi negativi e si considera l'ipotesi semplificativa di esistenza e unicità del MST del grafo.

I passaggi utili all'implementazione sono i seguenti:

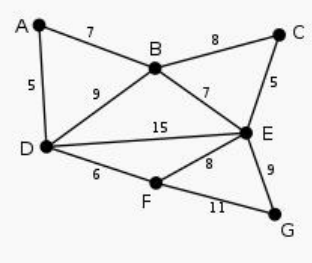
- in input si ha un grafo non orientato di insieme dei nodi V e degli archi E;

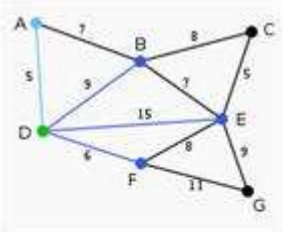
- si crei un insieme V' contenente un nodo x di partenza scelto arbitrariamente e un insieme E' vuoto;
- finché non si arriva ad avere V = V':
  - scegliere un arco (u, v) da E di peso minimo tale che u ∈ V' e v ∉ V'
    (se sono presenti più archi dallo stesso peso minimo la scelta è arbitraria);
  - si aggiunga v a V' e (u, v) a E'.

Gli insiemi V' ed E' contengono nodi e archi che compongono il MST del grafo.

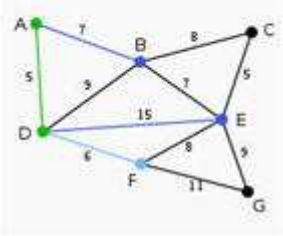
4.1.4. Esempio di applicazione di algoritmo di Prim

Applichiamo l'algoritmo di Prim al grafo visto nell'esempio precedente:

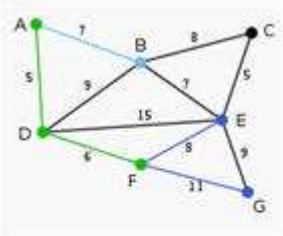




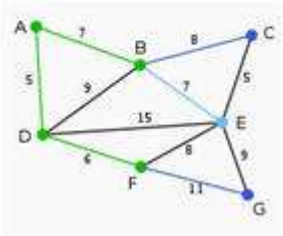
Scegliamo il nodo D arbitrariamente come nodo di partenza. I vertici adiacenti a D connessi da un singolo arco sono A, B e F. Di questi A è quello collegato dall'arco di peso minimo, allora lo scegliamo come secondo nodo e aggiungiamo nel MST in costruzione (A,D).



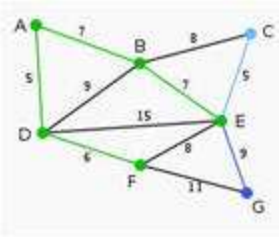
Il prossimo nodo da scegliere è il nodo più vicino ad A o D, considerati ambedue: B è distante 9 da D e 7 da A; E dista 15 da D; F dista 6 da D. Allora F è il più vicino a uno dei due nodi, lo inseriamo ed evidenziamo l'arco (D, F).



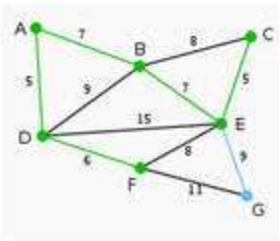
Si procede in questo modo: inseriamo B distante 7 da A, e quindi (A, B).



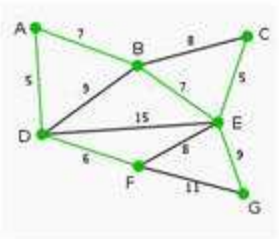
Ora possiamo scegliere tra C, E e G; C dista 18 da B; E dista 7 da B; G dista 11 da F. E quindi è il più vicino e va inserito, evidenziamo di conseguenza anche (B, E).



Gli unici nodi rimasti da inserire sono C e G: C dista 5 da E e G dista 9 da E. Scegliamo C ed evidenziamo l'arco (E, C).



Il nodo G è l'ultimo rimasto: esso dista 11 da F e 9 da E. E è il nodo più vicino, dunque inseriamo G e il nodo (E, G).



Adesso ogni nodo fa parte dell'albero di costo minimo segnato in verde. Il MST trovato ha una lunghezza di 39.

4.2. Cammino minimo e Arborescenza minima

Fra le applicazioni dei problemi più importanti e specifici si possono citare:

- instradamento di veicoli (routing);
- ottimizzazione del traffico su reti di comunicazione;
- ottimizzazione del traffico su reti di trasporto;
- alcuni problemi di controllo degli stock;
- problemi di gestione dei progetti;
- problemi di controllo risolti con tecniche di programmazione dinamica;
- alcuni problemi di elaborazione di segnali;
- problemi di codifica e decodifica di messaggi;
- problemi di allocazione temporale di attività (sequenziamento o scheduling);
- alcuni problemi di intelligenza artificiale;
- alcuni problemi di riconoscimento di immagini;
- alcuni problemi di riconoscimento vocale;
- alcuni problemi di verifica di correttezza di programmi per calcolatori;
- alcuni problemi di schematizzazione decisionale e relazioni sociali.

I grafi considerati di seguito sono, salvo diversa specificazione, grafi orientati. Nel caso della ricerca di cammini infatti è sempre possibile ricondursi con poca fatica a questo caso. Per fare questo è sufficiente sostituire ogni arco non orientato (e quindi percorribile in entrambi i versi) con due archi diretti in direzione opposta. Se esiste un cammino con le caratteristiche richieste nel grafo originario, allora esiste anche nel grafo trasformato e viceversa.
Il problema del *cammino di peso minimo* può essere enunciato nel seguente modo:

dati due nodi $s \in V$ e $t \in V$, trovare un cammino orientato
P* in G da s a t che abbia peso minimo.

In genere, dato un nodo *s* gli algoritmi per il calcolo di cammini minimi determinano non il (o uno dei) cammini minimi da *s* a un fissato nodo *t*, ma la cosiddetta *arborescenza minima* (*albero dei cammini minimi*): si tratta in effetti (vedi Definizione 20) di un sottografo di G che è un albero i cui nodi includono *s* e tutti i nodi da esso raggiungibili con un cammino orientato e tale che l'unico cammino da *s* a ogni altro nodo *t* dell'albero sia un cammino minimo da *s* a *t*. A prima vista questa può sembrare una complicazione non necessaria, ma in effetti, per come sono organizzati gli algoritmi non è così. In pratica, gli algoritmi in questione risolvono il seguente problema:

dato un nodo $s \in V$, trovare un albero dei cammini minimi
da *s* ad ogni nodo in V raggiungibile da *s*.

Esistono molti algoritmi per il calcolo dei cammini minimi, ognuno di essi ha le sue peculiarità e le sue limitazioni. Per esempio alcuni algoritmi funzionano solo su determinate classi di grafi (grafi con pesi non negativi, grafi aciclici, etc.), inoltre ci possono essere notevoli differenze nell'efficienza. In generale un algoritmo che funziona su una classe ampia di grafi sarà più complesso di un algoritmo studiato per una classe ristretta, tenendo conto delle peculiarità di quella classe.
Considereremo ora il principale algoritmo applicato nei software e nei programmi che si incontreranno in seguito per la ricerca del cammino minimo su grafi pesati.

4.2.1. Algoritmo di Dijkstra

Questo algoritmo prende il nome dall'informatico olandese Edsger Dijkstra che lo formulò nel 1956.
La sua applicazione si limita esclusivamente a grafi con archi di peso non negativo, mentre non vi sono restrizioni per quanto riguarda l'aciclicità. Più precisamente, l'algoritmo calcola il peso del cammino minimo da un nodo *s* a tutti gli altri nodi del grafo, costruendo contemporaneamente l'albero dei cammini minimi. Con questo processo è possibile anche individuare il cammino minimo da un nodo *s* di partenza ad un singolo nodo *t* d'arrivo, fermandolo una volta che il cammino minimo fino al nodo di destinazione scelto è stato determinato.
Il procedimento di calcolo dello SP si compone di più iterazioni per ognuna delle quali, preso un grafo orientato e pesato G = (V, E), l'insieme dei nodi V (a cui appartiene *s*) viene partizionato in due sottoinsiemi S e T. L'insieme S contiene il nodo *s* e tutti i nodi *j* per cui l'etichetta di distanza associata $d(j)$ è ottima: essa rappresenta da distanza di quel nodo, lungo un cammino, al nodo iniziale. In T vi sono i restanti nodi $V \setminus S$. A ogni iterazione k l'algoritmo sposta un nodo *h* da T a S, e aggiorna alcune etichette distanza $d(i)$; l'esecuzione termina quando T è vuoto.
L'inizializzazione dell'algoritmo prevede che esista sempre l'arco (s, i), per ogni $i = s$. Se qualche arco non è presente nel grafo originale, si può aggiungere un arco fittizio di peso pari a $+\infty$.
Per applicare la procedura ad un grafo in cui ho un nodo di partenza e un nodo d'arrivo si devono ricordare dunque le seguenti *regole* per procedere correttamente:

- ogni nodo all'inizio ha una distanza associata posta a $+\infty$;
- il nodo di partenza ha distanza associata posta a 0;
- ogni volta che si sceglie un nodo con distanza associata minore e lo si inserisce nel cammino minimo si aggiornano i nodi adiacenti;
- la distanza associata di un nodo è data dalla somma della distanza del nodo precedente con il peso del collegamento a quel nodo dal precedente;

- non si aggiornano le distanze associate dei nodi già considerati nel cammino minimo;
- le distanze associate definitive trovate indicano la distanza in peso di tali nodi a quello di partenza;
- quando si aggiorna la distanza minima di un nodo si mantiene quella minore.

Quindi scelto un nodo di partenza, l'algoritmo assegna dei valori iniziali di distanza dei nodi e prova a migliorarli (cioè minimizzarli). L'implementazione della ricerca iterativa si presenta così:

1. si assegna inizialmente ad ogni nodo un valore della distanza: per il nodo iniziale si pone valore 0, per tutti gli altri nodi a ∞;
2. si segnano tutti i nodi come non ancora visitati, e il nodo iniziale come nodo corrente;
3. per il nodo corrente si considerano i suoi nodi adiacenti non ancora visitati e si calcola la loro distanza *tentativo* (dal nodo iniziale). Se per esempio il nodo corrente è A che ha distanza associata 5, e vi è un nodo B connesso ad A da un arco di peso 3, la distanza al nodo iniziale di B passante per A sarà di 5+3 = 8. Se tale distanza fosse minore della distanza associata a B in precedenza (∞ all'inizio, 0 per il nodo iniziale), sostituire la distanza;
4. quando si è terminato di considerare tutti i nodi adiacenti al nodo corrente lo si segna come visitato. Un nodo visitato non verrà più preso in considerazione. La sua distanza associata è definitiva ed è minima;
5. se tutti i nodi sono stati visitati si ha finito. In caso contrario si imposta il nodo non ancora visitato con la minore distanza come nodo corrente e si procede dal punto 3.

4.2.2. *Esempio di applicazione di algoritmo di Dijkstra*

L'esempio sul grafo seguente permette di apprezzare l'esecuzione pratica dell'algoritmo di Dijkstra sul grafo di Figura 13:

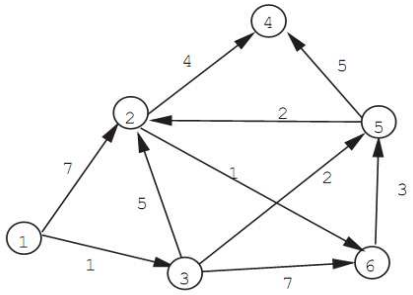


Figura 13: Esempio di grafo pesato ed orientato

Iterazione 0: si prenda il nodo 1 come nodo iniziale. Si assegnano quindi i valori delle distanze associate agli altri nodi. In particolare, ai nodi adiacenti si assegna il valore del peso dell'arco che li collega a 1 (al nodo 2 assegno distanza 7, al nodo 3 distanza 1), per gli altri valore +∞.
Iterazione 1: si segna dunque come visitato il nodo 3 che ha distanza minore dal nodo iniziale 1, e si riprendono le distanze associate ai nodi restanti aggiornandole: per il nodo 2 noto che la distanza dal nodo 1 passante per il nodo 3 diviene 6, minore del 7 che era prima, allora lo sostituisco; lo stesso per i nodi 6 e 7 con valori della distanza pari rispettivamente a 8 e a 3, entrambi minori di +∞. I nodo 4 conserva distanza +∞ in quanto non è raggiunto dal nodo 3.
Iterazione 2: preso come visitato il nodo 5 con distanza associata minore pari a 3, si ripete la procedura di aggiornamento delle distanze dei nodi non ancora visitati. Si considera sempre la

distanza nodo per nodo dal vertice iniziale 1, con il passaggio però per il nodo 5. Otteniamo che la distanza del nodo 2 ora è 5 (< 7) e la distanza del nodo 4 è 8 (< +∞).
Iterazione 3: si segna come visitato il nodo 2 con distanza 5: si ottiene che non ci sono variazioni sulla distanza del nodo 4, viceversa che il nodo 6 da una distanza di 8, passando per il nodo 3, cambia ad una distanza di 6 attraverso il nodo 2.
Iterazione 4: segnato il nodo 6 non si riesce da esso a raggiungere il nodo 4, che quindi mantiene distanza definitiva di 8, e viene segnato come visitato all'*iterazione* 5 successiva.

	nodo									
	2		3		4		5		6	
	d	J	d	J	d	J	d	J	d	J
It. 0	7	1	1	1	+∞	1(fitt.)	+∞	1(fitt.)	+∞	1(fitt.)
It. 1	6	3	**1**	**1**	+∞	1(fitt.)	3	3	8	3
It. 2	5	5			8	5	**3**	**3**	8	3
It. 3	**5**	**5**			8	5			6	2
It. 4					8	5			**6**	**2**
It. 5					**8**	**5**				

Figura 14: Tabella risolutiva dell'algoritmo di Dijkstra

Una comoda rappresentazione dell'evolvere dell'algoritmo è la forma tabellare di Figura 14 qui sopra presentata, in cui le righe rappresentano le iterazioni mentre le colonne rappresentano i nodi selezionati ad ogni iterazione. Per ciascun nodo *j* ci sono due colonne che riportano i valori della variabile $d^k(j)$ di distanza associata e della funzione $J^k(j)$ che indica il nodo precedente a quello considerato, attraversato dal cammino.
All'iterazione 0 viene assegnato come precedente il nodo 1 fittizio nel caso dei nodi non adiacenti al nodo iniziale. L'elemento segnato come visitato all'iterazione k-esima è rappresentato in grassetto e, nelle iterazioni successive, il valore della variabile corrispondente non viene più riportato. La colonna corrispondente al nodo 1 è omessa.

5. Software e Programmi

Dopo aver espresso formalmente i principi base e gli algoritmi necessari allo sviluppo dei problemi di albero, arborescenza e cammino minimi, si tratta ora di entrare prettamente nell'analisi d'utilizzo dei *software open source* trovati. La caratteristica *open source* dei software studiati li contraddistingue come programmi i cui autori ne favoriscono studio, modifiche e distribuzione ai programmatori indipendenti. I software, disponibili e scaricabili gratuitamente da differenti fonti presenti sul web (ossia siti propri descrittivi, *libraries*, siti di ricerca unicamente dedicati alla presentazione di liste di programmi etc.), consistono in applicazioni atte all'aiuto della comprensione e allo sfruttamento degli algoritmi illustrati in precedenza: essi sono utilizzati in quanto strumento potente di apprendimento e di risoluzione rapida di problemi non tipicamente semplici. La facilità nell'utilizzo pratico da parte dell'utente e l'interfaccia grafica attrattiva, dotata in alcuni casi di intuitiva animazione, rendono questi software di grande interesse per tali scopi.
La reperibilità sul Web di programmi per la risoluzione di problemi su grafi orientati o non orientati è buona, ma non certo dopo un'esplorazione di superficie: è chiaro che la specificità dell'argomento della teoria dei grafi rende poco elevato il numero di software sviluppati, nonostante all'inverso l'applicabilità a diversi campi e ambienti possa essere elevata. Dopo una ricerca più approfondita, si possono trovare valide proposte efficienti e ben progettate. Nel caso di applicazioni calate in contesti e campi specifici (scientifici, informatici, di programmazione o di relazione interpersonale ad esempio) le soluzioni software basate sulla teoria dei grafi possono essere molto soddisfacenti, anche se spesso non incluse nella categoria *open source*.

Si osserva dunque come emerga la tendenza ad una creazione di programmi o dalla più ampia adattabilità possibile a diversi settori o dallo sviluppo orientato all'immersione massima nello specifico settore di utilizzo, per gamma funzioni e terminologia: l'analisi si soffermerà nella descrizione di esempi di programmi del primo tipo.
Dopo un'attenta cernita si ottengono i software che verranno considerati e descritti in analisi: sono stati scelti i migliori per caratteristiche di applicazione e prestazioni di algoritmi, completezza sotto l'aspetto della risoluzione di problemi e topics differenti, completezza e funzionalità di comandi, semplicità di utilizzo e accessibilità di informazioni, chiarezza e cura di interfacce e diversi layouts grafici, applicabilità verso la più ampia finestra di settori etc.

6. Librerie, licenza e scaricamento

I software *open source* sono molto spesso reperibili sul Web in siti specializzati nella presentazione di liste di programmi scaricabili gratuitamente o in una *library* dedicata: una *library* (in italiano tradotto *libreria* invece di *biblioteca*, traduzione errata ma ormai di diffuso uso comune) è una vera e propria libreria di strutture dati, codici e funzioni a supporto di un particolare programma software. Lo scopo di queste librerie *online* è quello di fornire elementi ed entità di base che quindi non devono essere inseriti dall'utilizzatore: la libertà d'accesso e scaricamento delle componenti già disponibili di supporto ha la finalità di porre l'attenzione allo sviluppo e l'esecuzione del programma. Il software, una volta scaricato il file di setup d'installazione o direttamente il file d'esecuzione, funzionante grazie al corrispondente linguaggio di programmazione (Java, C, C++ etc.), può essere manipolato e modificato dall'utente secondo i suoi scopi, quindi immesso nuovamente nella rete e reso disponibile ad altri utenti. Così facendo, è agevolato e incoraggiato lo sviluppo condiviso a seconda di esigenze diverse delle funzionalità più specifiche del software.
L'unico restrizione a cui l'utente deve sottostare al momento del *download* dei files è la presa visione e quindi l'accettazione delle condizioni d'uso della *GNU Lesser General Public License*: formalmente GNU Library General Public License, si tratta della principale e più diffusa licenza sui software gratuiti. Il suo scopo primo è garantire la libertà d'uso e di manipolazione dei *software packages* riferiti a librerie appartenenti alla *Free Software Foundation*: questa fondazione si occupa di eliminare le restrizioni sulla copia, sulla redistribuzione, sulla comprensione e sulla modifica dei programmi per computer.
La richiesta d'accettazione delle condizioni avviene immediatamente dopo la richiesta di *download* e in molti casi una copia della licenza viene inviata in allegato al pacchetto di files. Ogni software descritto di seguito quindi è stato ottenuto ed installato con queste modalità.

7. Graph Magics

Graph Magics è uno strumento efficace e completo, che offre la possibilità di facile, veloce ed efficiente costruzione e manipolazione di grafi. Graph Magics possiede un sito internet, in quanto software acquistabile, fornito di specifiche d'utilizzo ed informazioni sia generali che puntuali: è presente una sezione dedicata in cui effettuare il download gratuito del software per un periodo mensile di prova reiterabile.

Caratteristiche principali:

- la facilità con cui i grafi sono costruiti: nodi e archi possono essere aggiunti con un semplice click; il riposizionamento o ridimensionamento dei nodi può essere fatto semplicemente con un 'click and drag' ('clicca, trascina e rilascia' un oggetto); si possono modificare i pesi e le portate degli archi (il programma ammette solamente valori non negativi) digitando direttamente sulla rappresentazione grafica, o usando

una tabella di valori; è presente ogni minima funzione inserita anche nei più diffusi software agenti su grafi, nonché ogni risolutore utile all'analisi obiettivo;
- sono presenti tre diversi layouts per la visualizzazione dei grafi. Tutte contengo lo stesso grafo in uso, ma la visualizzazione è indipendente in ogni layout, offrendo così la possibilità all'utente di vedere il grafo in diverse modalità;
- l'esecuzione in animazione e la visualizzazione passo dopo passo di ogni algoritmo per i grafi (trovare lo SP, il MST, altri problemi come quello del flusso massimo, del taglio minimo, i circuiti euleriani ed ha miltoniani etc.) sono supportate;
- con il comando *Graph generator* è possibile generare 4 diversi tipi di grafi con dimensioni e parametri differenti;
- sono disponibili funzioni per disporre grafi in schemi a cerchio, albero, griglia, grafi bipartiti;
- sono disponibili cinque forme differenti per i vertici: rettangolo, quadrato, cerchio, ellisse e rombo;
- le funzioni Avanti / Indietro, Taglia / Copia / Incolla e Zoom sono presenti;
- è possibile importare ed esportare grafi da files 'raw data' (files di dati / informazioni "grezze", non manipolate o processate) di diverse rappresentazioni (matrice di adiacenza, lista di adiacenza, lista di archi);
- sono supportati grafi contenenti fino a 1000 nodi, pesi e portare di archi da 1 a 2147483647 incluso;
- la visualizzazione grafica può essere messa a stampa o inserita in un file d'immagine.

Le caratteristiche migliori di questo programma sono il suo *Graph generator* e i suoi 17 algoritmi applicabili alla teoria dei grafi. I motivi principali per cui le qualità di questo particolare software possono rendersi utili sono molteplici: l'applicazione nella pratica di più ambiti (economia, scienze sociali, funzioni urbane etc.) per la risoluzione e l'analisi di problemi sui grafi (e i relativi algoritmi); come strumento per l'insegnamento e lo studio della teoria dei grafi; costruire, modificare e progettare grafi per scopi differenti dai sopra citati.
Si entra ora nel merito delle operazioni possibili che riguardano i problemi da risolvere nell'impostazione della ricerca.

7.1. *Visualizzazione*

Graph Magics permette di porre in interfaccia tre tipi di visualizzazioni. Sulla barra delle operazioni in alto vediamo: *View Graph*, *View Graph–Data* e *View Data*.

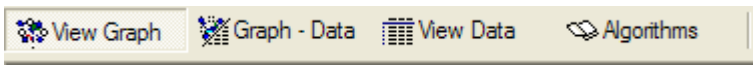


Figura 15: Opzioni View

L'opzione ***Algorithms*** permette di vedere una breve descrizione di ogni algoritmo utilizzabile con Graph Magics.

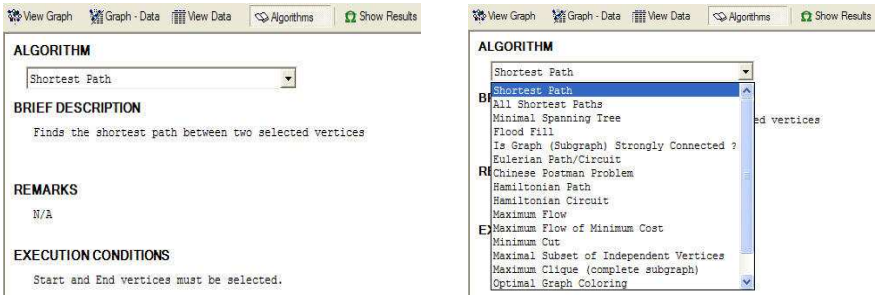


Figura 15.1: Opzione Algorithms

L'opzione ***View Graph*** mostra la rappresentazione grafica del grafo ed è impostata di default all'apertura del programma. ***View Data*** mostra le tabelle contenenti i valori degli archi in forma di matrici di adiacenze e peso. Vi sono due tabelle per tipo, una con tutti gli archi del grafo, l'altra con solo quelli di un nodo selezionato. ***View Graph-Data*** mostra sullo schermo entrambe le rappresentazioni accoppiate una sopra l'altra.

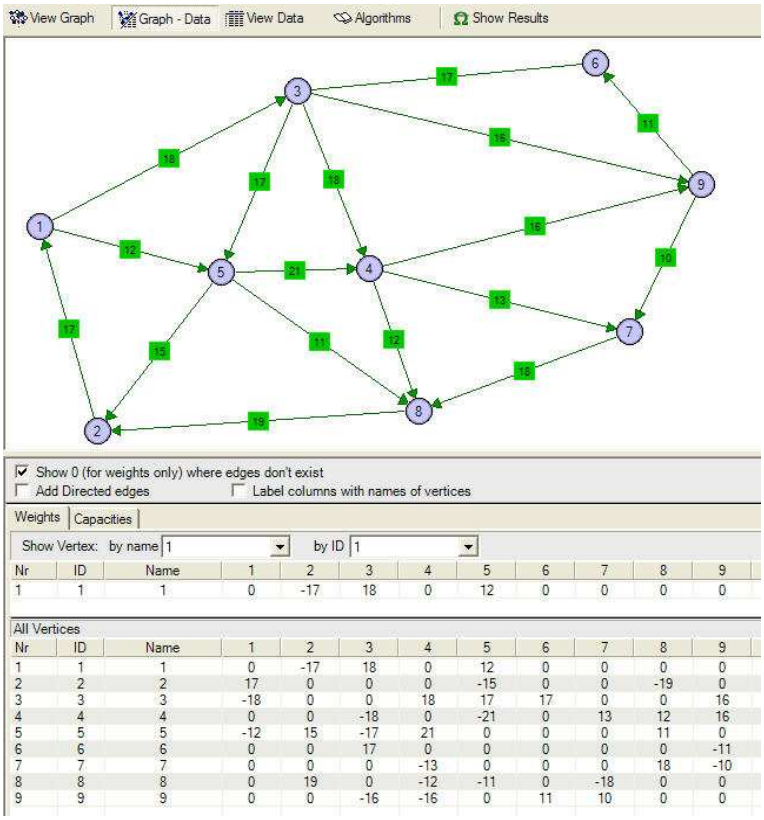


Figura 15.2: Visualizzazione in modalità View Graph-Data

Un grafo può essere importato da un file salvato in precedenza, aggiunto ad un grafo corrente in uso, o importato da file 'raw data'. Per la creazione di un nuovo grafo invece sono possibili due strade: la costruzione manuale da rappresentazione grafica od il *Graph generator*.

7.2. *Costruzione da rappresentazione grafica*

Dal momento in cui si usa la modalità di visualizzazione grafica per la costruzione del grafo vi sono solamente due opzioni da dover considerare: l'opzione *Pointer* e *Add Vertex*.

Lo stato **Pointer** viene inserito di default, e viene utilizzato per tutto tranne la creazione dei nodi (a forma di freccia nella figura). **Add Vertex** viene utilizzato invece unicamente per aggiungere i nodi che danno forma al grafo.
Per aggiungere un nodo come detto selezionando il tasto + (Add Vertex) sulla barra degli strumenti e, se l'opzione di layout è su Free, il nodo verrà formato ad un click del mouse.
Con il tasto destro del mouse, selezionato un vertice, si può creare un arco con un click in un altro vertice. Lo stesso si fa per eliminarne uno preesistente. Per archi orientati il procedimento è quasi lo stesso: per creare un arco da 1 a 2, selezionato 1, si tiene premuto il tasto X e si fa click su 2 con il tasto destro; per farlo apparire da 2 a 1,selezionato 1, premo Z; per inserire *archi doppi* premo S se voglio l'arco verso 2, altrimenti A.

Per archi orientati è possibile anche eseguire un solo click dopo aver selezionato il comando dalla toolbar.

Figura 16: Archi doppi con pesi differenti

È possibile selezionare uno o più nodi e archi con un click, passando allo stato Pointer, e lavorare con essi. Semplicemente trascinando i nodi selezionati si possono muovere a piacimento e premendo il tasto destro del mouse si rendono disponibili molte operazioni: variare la dimensione del nodo, il nome, la forma, eliminarlo, eliminare tutti i nodi, aggiungere acicli, presentare nodo in primo o secondo piano (in caso di sovrapposizioni grafiche), selezionare il nodo come "nodo di partenza" o "di arrivo" (Start Vertex, End Vertex), selezionare l'algoritmo scelto per l'esecuzione.
Alla modifica del layout dei vertici (**Vertices Layouts**) in cerchio, griglia, albero etc. si accede nella stessa maniera, oppure dalla barra degli strumenti dai simboli seguenti:

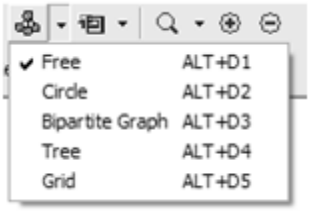


Figura 17: Menu opzione Vertices Layouts

Dall'icona visualizzata a destra di Vertices Layouts si può accedere a delle aree di lavoro (**Graph Layouts**), che contengono il grafo su cui si sta lavorando, e da cui si possono attuare modifiche indipendentemente.
Sulla stessa figura sono situati anche i simboli della funzione **Zoom**, presenti assieme alle funzionalità **Copia**, **Incolla**, **Avanti** e **Indietro** (Avanti / Indietro di un'operazione).
Tutte le funzioni e le operazioni sopra citate sono inoltre raggiungibili e riportate ordinatamente nella tradizionale sezione della barra degli strumenti in alto.

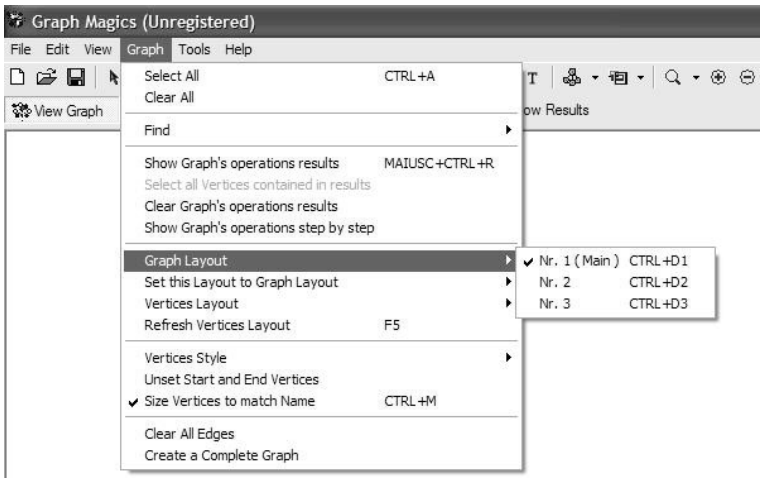


Figura 18: Menu opzioni sulla barra degli strumenti

7.3. Graph generator

Questo strumento consente di generare facilmente un grafo da parametri desiderati. Tale grafo poi può essere salvato in un file, aperto da Graph Magics come nuovo grafo, oppure inserito come modifica di un grafo corrente.
Si devono inserire i seguenti parametri per generare il grafo:

- **Numero di Vertici**
  - numero fisso;
  - random (in un intervallo selezionato).
- **Numero di Archi**
  - numero fisso;
  - random (in un intervallo selezionato).
- **Peso degli Archi**
  - numero fisso;
  - random (in un intervallo selezionato).
- **Tipo di Grafo**
  - generale;
  - bipartito;
  - albero (con la possibilità di scegliere il massimo numero di livelli);
  - albero binario;
  - archi orientati;
  - grafo connesso.

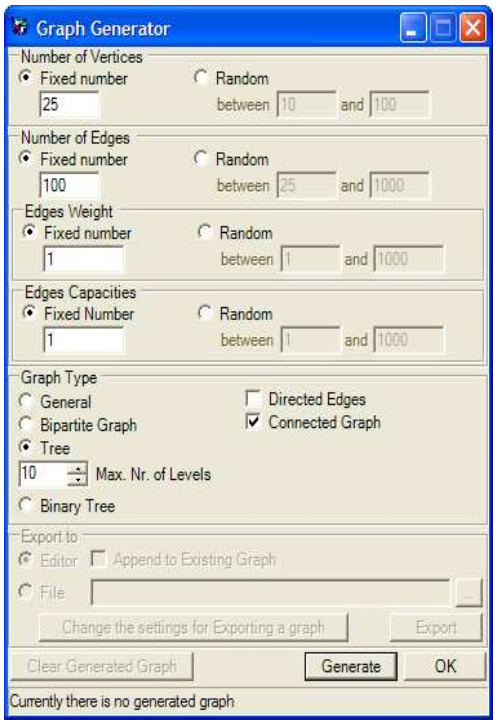
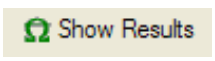


Figura 19: Finestra Graph Generator

Il *Graph generator* corregge automaticamente l'inserimento errato di parametri e quasi istantaneamente, in meno di 1-2 secondi, crea il grafo descritto, anche per grafi al limite massimo per numero di nodi e archi del programma.

7.4. Esecuzione degli algoritmi

Tra tutti gli algoritmi presenti in Graph Magics troviamo **Shortest Path** tra due vertici, **All Shortest Path** partendo da un vertice scelto e **Minimal Spanning Tree**: questi sono di interesse per la risoluzione dei problemi di cammino minimo, arborescenza minima e albero minimo. Il Minimal Spanning Tree è l'implementazione dell'algoritmo di Prim, applicato a grafi privi di orientazione: Graph Magics segnala l'errore se si tenta di applicare l'algoritmo a grafi con archi orientati. Lo Shortest Path lavora invece su grafi orientati (anche se è permesso applicarlo a grafi non orientati) con archi non negativi poiché è l'efficiente algoritmo di Dijkstra,: il processo di calcolo viene eseguito solo dopo aver posto due nodi come "nodo di partenza" e "nodo di arrivo" (Start e End Vertex). L'All Shortest Path, l'algoritmo di Floyd-Warshall, che ammetterebbe anche archi di peso negativo, necessita invece di un nodo di partenza per poter procedere al calcolo dei cammini minimi da quel nodo ai rimanenti. Il modo di procedere di questa procedura è il seguente: si ha in input l'insieme degli archi A (matrice N x N) e si assegna per ogni coppia di vertici *i* e *j* un valore A(*i*, *j*) pari alla lunghezza dell'arco che li connette, o il valore infinito se non esiste tale arco; ad ogni step si osserva se vi è un cammino da *i* a *j* passante per un vertice intermedio *k* che sia minore in valore del cammino già trovato da *i* e *j*.
Per eseguire gli algoritmi è necessario premere il tasto destro del mouse su un vertice o sullo sfondo bianco del grafo: apparso il menu popup, raggiungere l'opzione **Find** e selezionare l'algoritmo desiderato. Dopo che il programma ha eseguito l'algoritmo si può osservare il risultato nell'immediato selezionando dallo stesso menu **Show Graph's operations results**, oppure dalla toolbar **Show Results**.



La soluzione è apprezzabile sia graficamente con View Graph, con colorazione intuitiva che evidenzia archi e nodi interessati, sia in forma di View Data, con espressione tabellare del

risultato. È possibile a questo punto tornare a modificare il grafo originario, dopo essere usciti dalla situazione di Show Results, che comunque mantiene memoria del risultato dell'ultima esecuzione.
Graph Magics ammette la possibilità di esaminare lo svolgimento dell'algoritmo durante la sua ricerca alla soluzione: con un click sull'icona **Show Results step by step**, quando si esegue l'algoritmo scelto si assiste al susseguirsi di operazioni che portano a cammino, albero o arborescenza minima. Quest'ultimo strumento è interessante per quanto riguarda l'apprendimento del processo di calcolo e la verifica del metodo risolutivo seguito dall'algoritmo scelto.

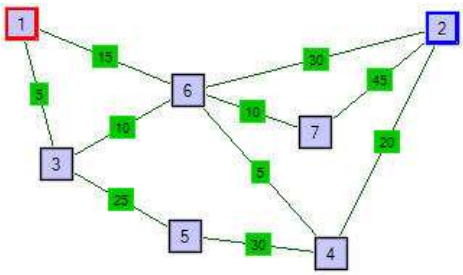
7.5. Esempio di calcolo

Esempio di calcolo step by step di *Shortest Path* tra due nodi di un grafo non orientato:

- i nodi da processare sono in grigio;
- i nodi contornati di rosso e blu sono rispettivamente nodo di partenza e arrivo;
- il nodo colorato di blu è il nodo corrente;
- i nodi già processati sono colorati in rosso;
- il numero su un nodo rappresenta la lunghezza del cammino minimo di quel nodo dal nodo di partenza, fino a quel punto dell'esecuzione.

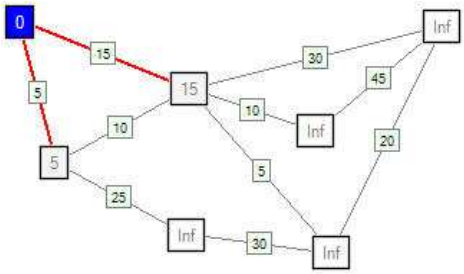
Ecco ora il grafo di partenza su cui calcolare il cammino di costo minimo dal nodo 1 al nodo 2, con una tabella riassuntiva dei valori passo per passo: in colonna troviamo i nodi, sulle righe se il nodo è stato processato oppure no, la sua distanza dal nodo di partenza e il nodo precedente sul cammino minimo.

Node	Processed	Distance from Source	Parent
1	false	0	null
2	false	Infinity	null
3	false	Infinity	null
4	false	Infinity	null
5	false	Infinity	null
6	false	Infinity	null
7	false	Infinity	null



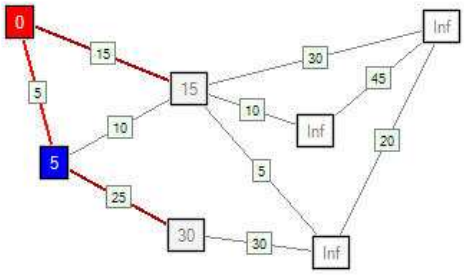
Tra tutti i nodi il nodo 1 ha ovviamente distanza minore da se stesso. Segno 1 come processato e aggiorno le distanze.

Node	Processed	Distance from Source	Parent
1	**true**	0	null
2	false	Infinity	null
3	false	**5**	**1**
4	false	Infinity	null
5	false	Infinity	null
6	false	**15**	**1**
7	false	Infinity	null



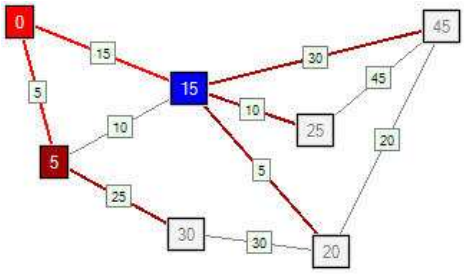
Tra tutti i nodi non processati il nodo 3 ha distanza minore da 1. Aggiorno le distanze di tutti i nodi e segno 3 come processato.

Node	Processed	Distance from Source	Parent
1	true	0	null
2	false	Infinity	null
3	**true**	5	1
4	false	Infinity	null
5	false	**30**	**3**
6	false	15	1
7	false	Infinity	null



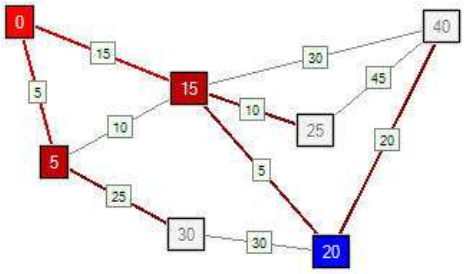
Il nodo 6, con distanza 15, è il nodo con distanza minore dal nodo 1. Lo segno e aggiorno i valori per i nodi ancora da processare adiacenti.

Node	Processed	Distance from Source	Parent
1	true	0	null
2	false	**45**	**6**
3	true	5	1
4	false	**20**	**6**
5	false	30	3
6	**true**	15	1
7	false	**25**	**6**



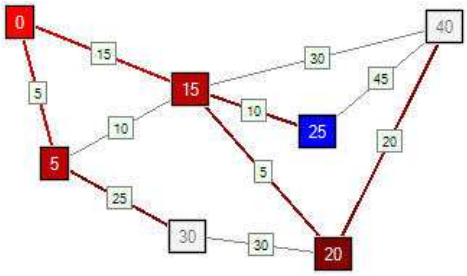
Il nodo 4 ha distanza minore ora da 1. Ancora proseguo, aggiornando le distanze dei rimanenti cammini agli archi.

Node	Processed	Distance from Source	Parent
1	true	0	null
2	false	**40**	**4**
3	true	5	1
4	**true**	20	6
5	false	30	3
6	true	15	1
7	false	25	6



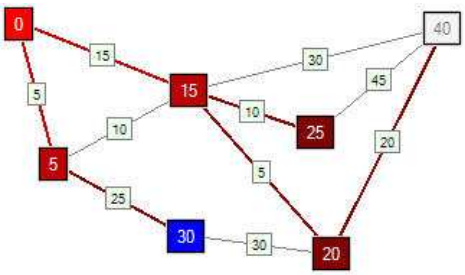
Adesso va processato il nodo 7 con distanza di 25.

Node	Processed	Distance from Source	Parent
1	true	0	null
2	false	40	4
3	true	5	1
4	true	20	6
5	false	30	3
6	true	15	1
7	**true**	25	6

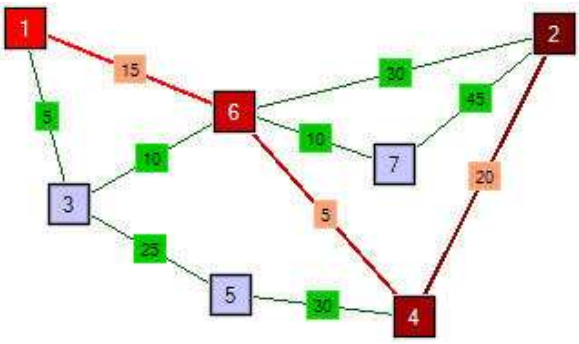


Tra i nodi 2 e 5 rimasti inserisco il nodo 5 con distanza 30.

Node	Processed	Distance from Source	Parent
1	true	0	null
2	false	40	4
3	true	5	1
4	true	20	6
5	**true**	30	3
6	true	15	1
7	true	25	6



Quindi considerato infine il nodo 2 unico rimasto e punto d'arrivo voluto: si ottiene il cammino di peso minimo dal nodo 1 al nodo 2, passante per i nodi 6 e 4, di peso totale pari a 40.



ALGORITHM

Shortest Path

RESULT (Time Elapsed - 0 sec.)

Start Vertex - 1
End Vertex - 2
Shortest distance - 40

Path:

#	Vertex	Total Distance
1	1	0
2	6	15
3	4	20
4	2	40

Figura 20: Soluzione grafica e testuale del problema del cammino di peso minimo

8. Peklo

Peklo è un software scaricabile da molti siti Web specializzati nella diffusione di programmi per le più diverse esigenze di utilizzo: Peklo è un editor grafico creato dal *toolkit* di funzioni di GTK+, una piattaforma multipla per la costruzione di interfacce grafiche per gli utenti, anch'essa disponibile online sotto licenza GNU. L'ambiente di lavoro così ottenuto è utile nella visualizzazione e nella comprensione di algoritmi applicati alla teoria dei grafi. È un software che offre un'efficiente costruzione per via grafica, buone capacità di manipolazione e complete metodologie di risoluzione dei problemi.

Caratteristiche principali:

- i grafi sono costruiti tramite rappresentazione grafica diretta: nodi e archi possono essere aggiunti, riposizionati con un click del cursore o cliccando e trascinando un oggetto; i pesi e le portate degli archi si possono modificare a piacimento, con ammessi anche valori negativi di peso; sono presenti molti tipi di algoritmi efficienti per la risoluzione di problemi e vi è immediatezza nell'utilizzo grazie a buone funzionalità e indicazioni per l'utente;
- il supporto di animazione step by step durante l'esecuzione dei molteplici algoritmi risolutivi per SP, MST e altri problemi sui grafi;
- sono disponibili funzioni per ridisporre i nodi del grafo, riassumere la sequenza di operazioni eseguite sul grafo, adattarne le dimensioni ed evidenziarne i parametri;
- è disponibile una finestra di impostazioni per la modifica grafica della rappresentazione: ridimensionamento e denominazione di nodi (necessariamente di forma circolare) e archi, colorazioni di oggetti e testi, riposizionamento di testo;
- le funzioni Avanti / Indietro, Taglia / Copia / Incolla sono presenti;
- è possibile effettuare il salvataggio di grafi, importare ed esportare a files, aprire esempi di grafo su cui lavorare anche con disposizione geometrica dei nodi;
- sono supportati grafi contenenti fino a 2147483647 nodi, pesi e portare di archi da -9×10^{98} a 9×10^{99} inclusi;
- la visualizzazione grafica può essere portata a stampa o inserita in un file d'immagine;
- non sono ammessi archi doppi, acicli, l'importazione di un grafo da matrici o liste di adiacenza, la creazione di grafi da parametri, una modifica del layout per quanto riguarda forme di nodi e archi già impostate.

Peklo offre all'utente la possibilità di lavorare in un'interfaccia votata alla semplicità d'utilizzo, al fine di prendere maneggevolezza con la risoluzione di esercizi e l'analisi di problemi sui grafi. Inoltre il programma è progettato per la dimostrazione delle procedure di calcolo degli algoritmi forniti: la visualizzazione a colori step by step consente di seguire chiaramente il susseguirsi delle operazioni fino alla soluzione, sia essa l'arborescenza,l'albero o il cammino di costo minimo.

8.1. Visualizzazione

Aprendo dal file eseguibile la finestra di Peklo si osserva in alto una tradizionale barra degli strumenti, a sinistra una lista di problemi da risolvere tra cui scegliere e un ampio spazio su cui si effettua la rappresentazione grafica del grafo.
Tra le icone nella barra si riconoscono le funzioni di **Nuovo Documento**, **Apri Documento**, **Salva** e **Salva con nome**, **Avanti** e **Indietro**. Da Nuovo Documento si può aprire un nuovo file su cui costruire un nuovo grafo, impostando alla finestra **Edit graph properties** se esso dovrà essere orientato o pesato.

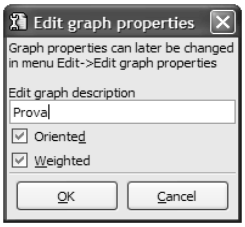


Figura 21: Finestra Edit Graph Properties di un nuovo documento

Dunque, dopo aver introdotto le caratteristiche del nuovo grafo, per inserire un nodo è sufficiente mantenere premuto il tasto "Shift" sulla tastiera e con un click del mouse selezionare la posizione desiderata a schermo. Una volta creato, il vertice verrà movimentato cliccandolo e trascinandolo nella nuova posizione. L'inserimento di un arco avviene invece mantenendo premuto il tasto destro del mouse a partire da un nodo fino al nodo d'arrivo.
Una piccola finestra popup appare dalla funzione **Properties** in alto, semplicemente passandoci sopra con il puntatore del mouse: da qui sarà possibile decidere sulle proprietà di nodi o archi selezionati. Per i nodi si può agire su nome, numero e coordinate, mentre per gli archi su pesi, capacità, descrizione, nodi connessi etc., da visualizzare e modificare. Un'altra funzione permette anche di scegliere un colore da applicare all'oggetto, sia nodo che arco.

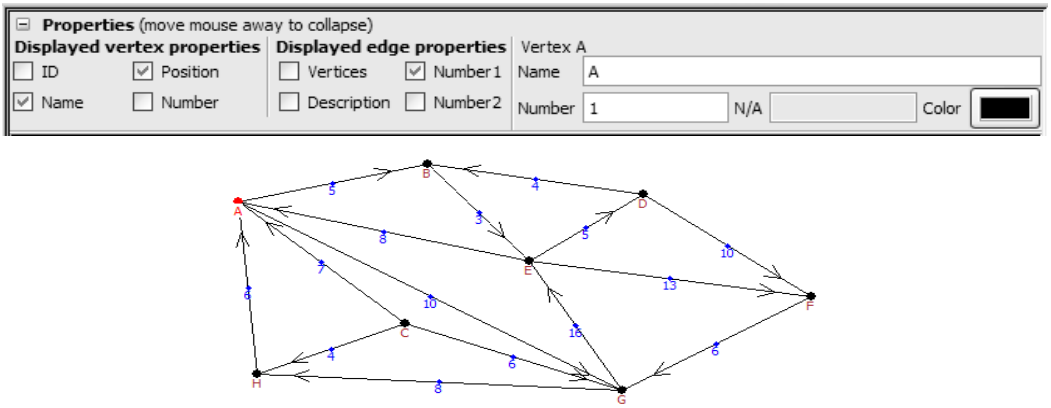


Figura 22: Finestra popup della funzione Properties

Dopo aver inserito un arco, per modificarne l'orientamento basta selezionarlo e premere insieme "Shift + Ctrl + E" da tastiera. Per eliminare un oggetto lo si seleziona e si preme "F8".

8.2. Funzioni

Sulla toolbar sono presenti altri comandi-funzione:

Figura 23: Comandi funzione della toolbar

Con il comando **Fit to window** si può adattare in proporzione la dimensione del grafo costruito a quella della finestra aperta del programma, così da avere sempre visibili tutti gli elementi del grafo.

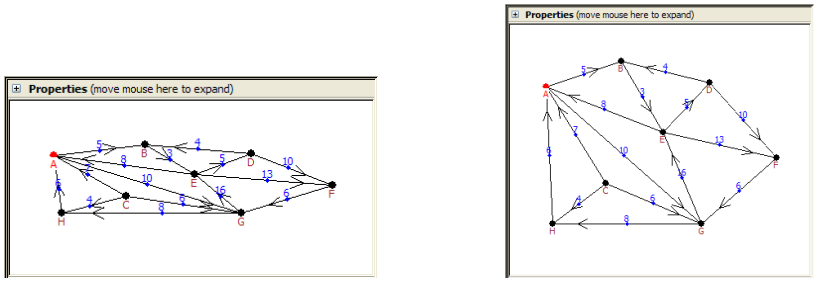


Figura 23.1: Esempio di opzione Fit to Windows su un grafo in due diverse finestre

Premendo su **Descriptions** appare nel layout l'elenco di descrizioni esplicite di testo date ai nodi, le descrizioni degli archi e i loro numeri caratterizzanti di peso, assegnati tramite la finestra delle Properties.

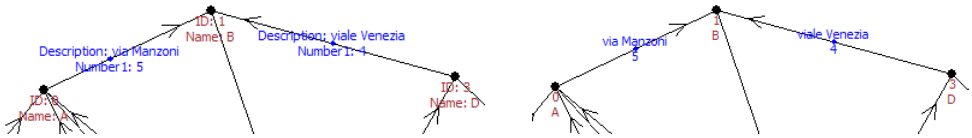


Figura 23.2: Grafo con e senza opzione Descriptions

Cliccando sull'icona **Reposition** si effettua un riposizionamento automatico dei vertici del grafo tramite l'algoritmo VIhope o di Kamada.

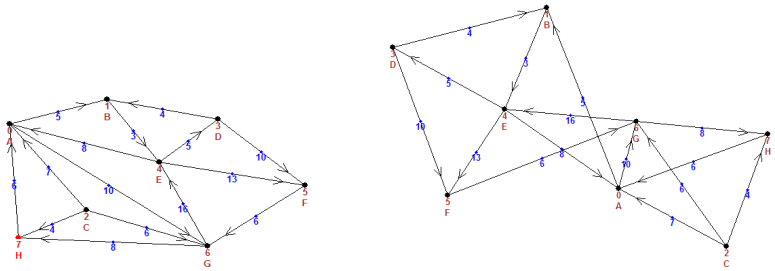


Figura 23.3: Applicazione della funzione Reposition al grafo di sinistra

Tramite **Log** si può consultare in ogni momento, in una finestra supplementare, l'elenco di operazioni svolte dalla creazione del grafo come creazione di oggetti, applicazione di algoritmi ed errori riscontrati, di cui viene tenuta memoria.

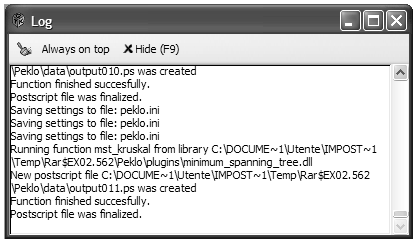


Figura 23.4: Finestra di Log

La funzione **Compare Mode** permette di aprire in una finestra separata il file in uso ed applicare su di esso diversi algoritmi risolutivi (tramite Change Library “Ctrl + L” si scelgono gli algoritmi a seconda del problema), per poi confrontarne i risultati ottenuti. È possibile anche aprire altri files, applicare gli algoritmi e quindi confrontarne il risultato coi risultati sulla finestra del grafo corrente.

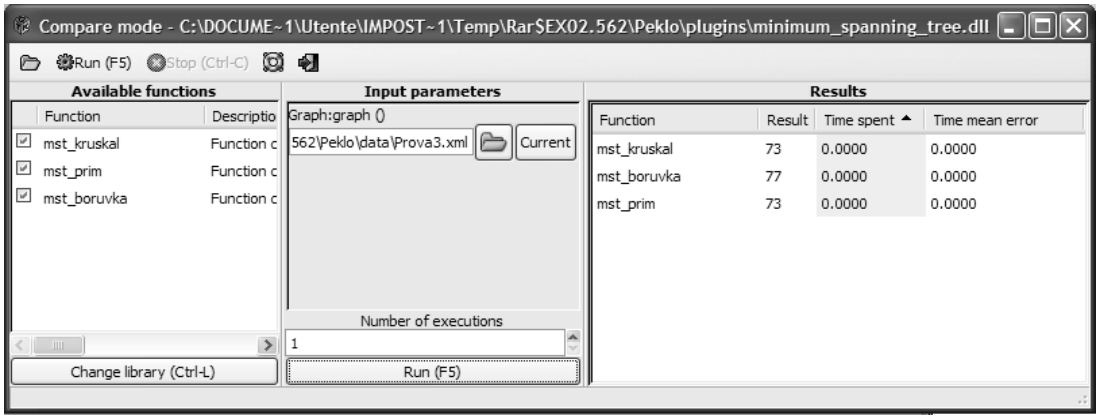


Figura 23.5: Finestra di Compare Mode

Per la manipolazione del layout grafico della rappresentazione Peklo rende disponibile, sotto alla voce **Tools,** una finestra di **Settings** da cui modificare: il diametro e lo spessore di nodi e archi, il tipo e la colorazione del testo da inserire, la forma delle frecce per l’orientamento degli archi. Tramite la finestra Settings si possono scegliere inoltre l’algoritmo e i parametri d’esecuzione per il riposizionamento dei vertici (Reposition). Nell’opzione **General** si possono inserire le opzioni di messaggio al compimento di un errore, “mostra e nascondi” automaticamente proprietà, attivazione pulsante “Next step”, nome del file, apertura dalla library. È presente anche un ulteriore punto d’accesso per la modifica delle denominazioni e dei numeri caratteristici di nodi e archi (numero di nodo, peso di arco etc.) tramite la sottofinestra **Templates**.

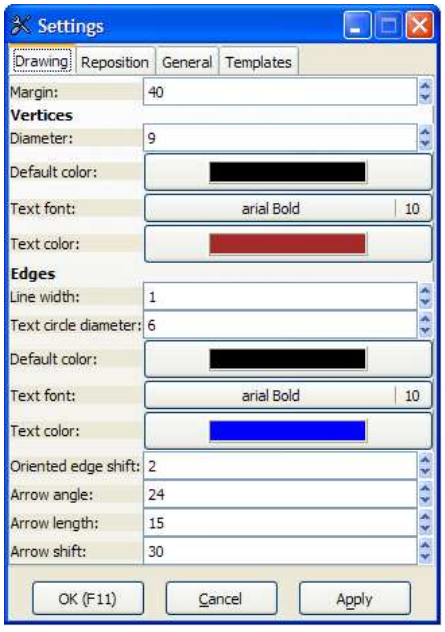
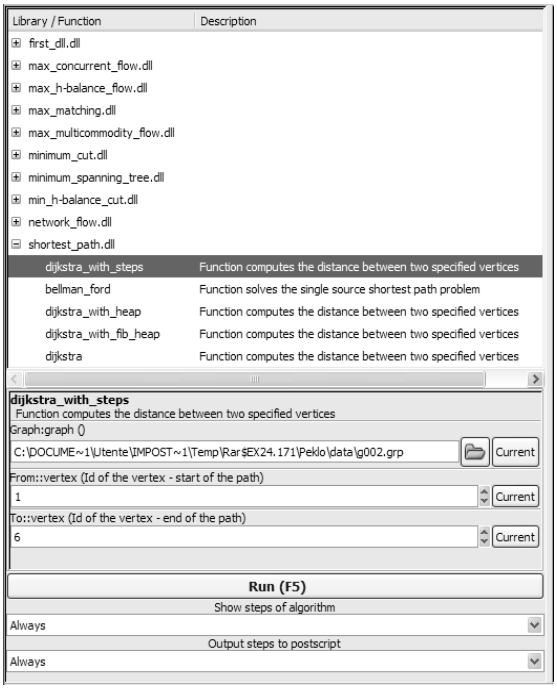


Figura 23.6: Finestra di Settings

8.3. *Algoritmi risolutivi*

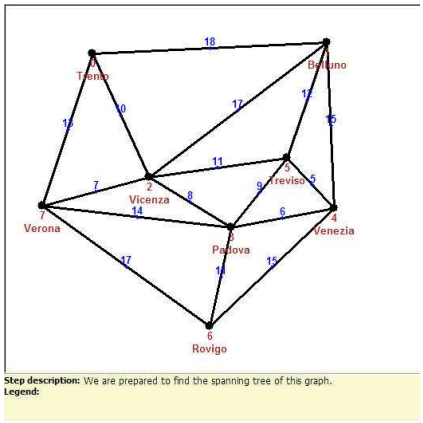


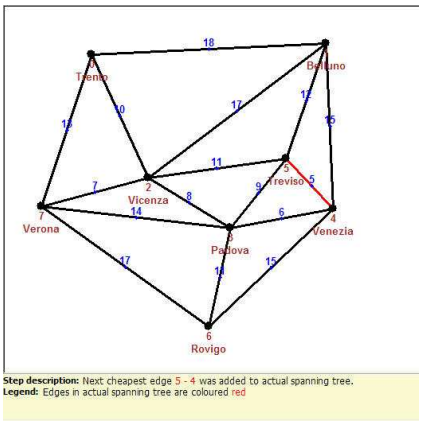
Nella sezione riservata alla risoluzione dei problemi è presentata un'ampia gamma di algoritmi: per ogni problema, estendendone il campo, si può scegliere l'algoritmo preferito. Per ogni algoritmo Peklo evidenzia brevemente la sua funzione e chiede su quale file applicarlo: è possibile infatti, oltre che sul grafo corrente, aprire direttamente un grafo da un file salvato e applicare su di esso il calcolo.
Dopo aver scelto l'algoritmo da eseguire, a seconda della sua procedura, si inseriscono i dati di partenza (per lo SP si richiedono nodo di partenza e di arrivo per esempio) e si innesca il calcolo con un click su **Run** o "F5": è quindi disponibile l'opzione di visualizzazione step by step del procedimento e spiegazione di ogni passo eseguito a nota di testo.
Per il problema del MST si utilizza la funzione di **minimum_spanning_tree** che offre l'uso di tre algoritmi: Kruskal, Prim e Borůvka. L'algoritmo di Borůvka segue la procedura seguente: da ogni nodo considera l'arco con minore peso che lo connette ad un altro nodo; prende allora in considerazione questi archi trovati per il MST, così che si formano uno o più alberi; nel caso gli archi trovati non formino un solo albero contenente tutti i nodi, vengono considerati gli archi di peso minimo che connettono gli alberi formati trovati, fino ad avere un unico albero che connette tutti i nodi. Borůvka in generale risulta meno efficiente di Kruskal e Prim, sia per precisione di soluzione che per tempo di risoluzione.
I problemi di cammino minimo si risolvono tramite **shortest_path**, ancora una volta con l'implementazione dell'algoritmo di Dijkstra: oltre al calcolo del cammino minimo tra due vertici, dopo opportuna impostazione dalla finestra delle Properties, è visibile su ogni nodo il costo aggiornato del tragitto minimo dal nodo di partenza. Nel caso di nodo non raggiungibile viene posto valore ∞. Dijkstra può essere applicato in Peklo anche a grafi non orientati, ma, per definizione, non con archi di peso negativo.
Il problema dell'arborescenza minima da una radice è supportato dall'algoritmo di Bellman-Ford: anch'esso ha in entrata un nodo di partenza e arrivo poiché, dopo aver calcolato dal nodo di partenza il percorso minimo ad ogni altro nodo, può anche evidenziarne solamente uno verso un nodo voluto. Vale anche per Bellman-Ford la presenza del valore numerico del peso del cammino minimo dalla radice su ogni nodo. Per definizione l'algoritmo ammette valori negativi dei pesi purché il grafo non presenti cicli negativi, cioè con archi la cui somma di peso sia negativa, ma in Peklo pesi negativi non sono supportati.
Allo stesso modo dell'algoritmo di Dijkstra, Bellman-Ford utilizza le indicazioni di relazioni fra archi, ma non procede con metodo *greedy*: l'algoritmo fa progressivamente decrescere il valore stimato del peso del cammino minimo dal nodo radice ad ogni nodo v in V (insieme dei vertici) finché non ottiene il reale e definitivo cammino minimo.

8.4. *Esempio di calcolo*

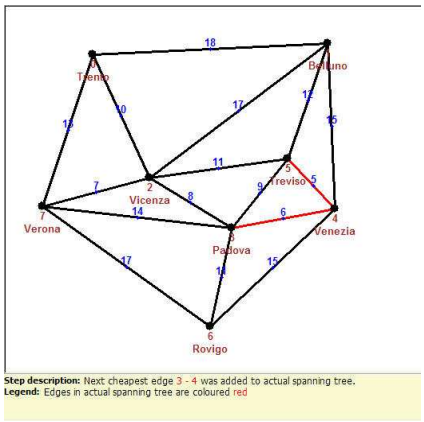
A questo punto si osservi un esempio di visualizzazione step by step accompagnata dai messaggi di testo sotto la rappresentazione grafica che descrivono le operazioni eseguite. Nel

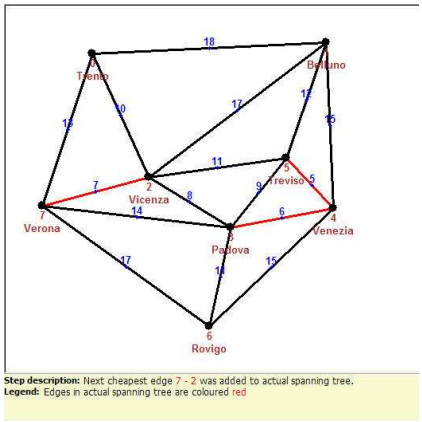
grafo non orientato che segue è rappresentata una schematizzazione delle distanze stradali che collegano alcune città del nord-est italiano: si vuole ricavare l'albero di peso minimo attraverso l'esecuzione dell'algoritmo di Kruskal.
Partendo dal grafo non orientato si individua il primo arco tra i nodi Venezia e Treviso, di peso minore a tutti pari a 5, e lo si inserisce nel MST in costruzione segnato in rosso.



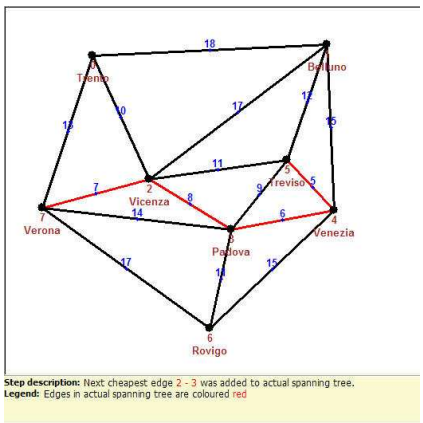


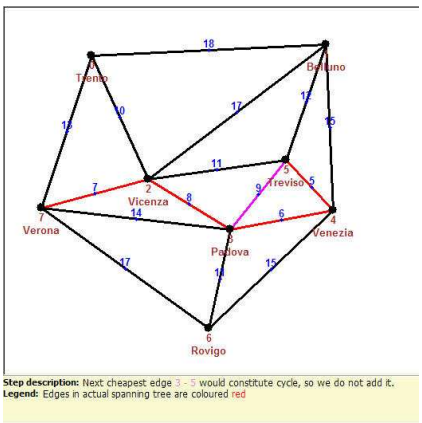
L'arco dal peso minore ora è quello tra Venezia e Padova, lungo 6. Lo si inserisce, come l'arco tra Verona e Vicenza di peso 7, poiché ciò non comporta formazione di cicli con archi già segnati.



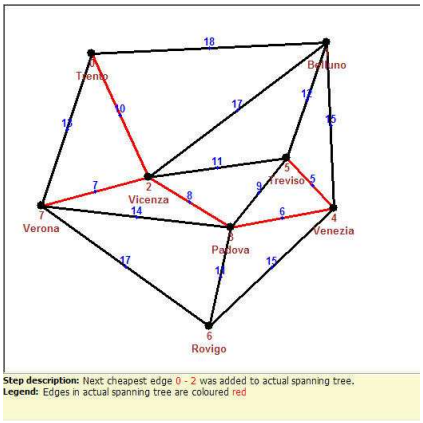


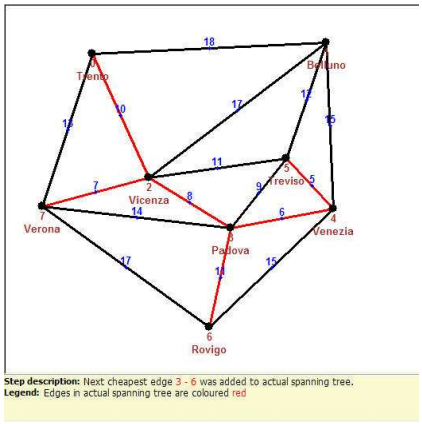
Si evidenzia l'arco tra Vicenza e Padova (peso 8), dopodiché si può notare come l'arco di peso minore fra i rimanenti è quello tra Padova e Treviso. Non può appartenere al MST però, ciò comporterebbe la formazione di un ciclo (Vicenza-Padova-Treviso).



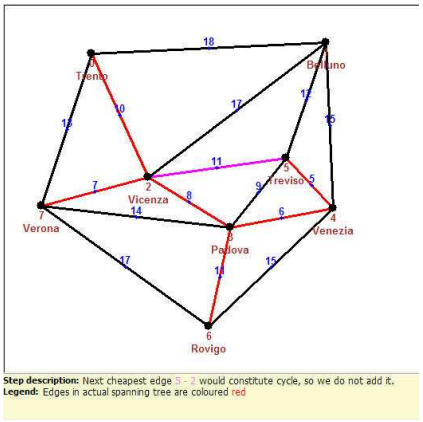


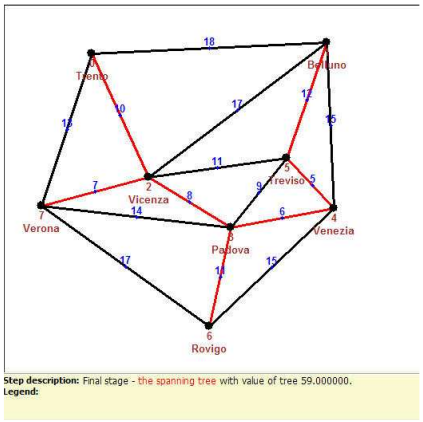
L’arco di peso 10 tra Vicenza e Trento è il minore a questo punto. Allo stesso modo si inserisce l’arco di peso 11 tra Padova e Rovigo.





Tra Vicenza e Treviso vi è l’arco di peso 11 minimo tra i rimanenti da considerare, però forma un ciclo nel MST costruito fin qui, quindi non viene inserito. Come ultimo, l’arco tra Treviso e Belluno conclude la ricerca dell’albero che collega tutti i nodi a più basso costo, cioè 59.





9. GOBLET

GOBLET è l’interfaccia grafica utilizzabile liberamente della libreria GOBLIN, libreria di classe C++ concentrata nell’ottimizzazione grafica e nei problemi di programmazione di rete. Il download del software è disponibile nel sito della libreria, assieme ad informazioni sulle specifiche di progetto, gli ultimi aggiornamenti ed un manuale di riferimento molto approfondito. GOBLET può essere usato per costruire e modificare grafi, eseguire algoritmi di risoluzione di problemi e osservare i risultati su rappresentazione grafica. Il software offre un’ampia gamma di funzionalità, un’interfaccia chiara e semplice da usare e un layout ben costruito. Molte funzionalità non fanno capo allo scopo della nostra ricerca, ma, anche se considerate in maniera marginale, concorrono a far comprendere le vaste potenzialità del programma. Le caratteristiche migliori presenti sono la multifunzionalità, l’intuitività d’utilizzo e la facilità di costruzione dei grafi.

Caratteristiche principali:

- i grafi sono costruiti tramite rappresentazione grafica: nodi e archi possono essere aggiunti, riposizionati e ridimensionati mediante semplice uso di mouse e quadro

comandi, i pesi degli archi sono modificabili e sono ammissibili valori negativi. L'utilizzo di svariate funzioni, algoritmi e opzioni a disposizione dell'utente rende GOBLET un software completo sia in esecuzione, per rispondere a problemi applicabili alla teoria dei grafi, sia nel controllo di forma, layout e impostazioni del software;

- sono disponibili funzioni per la disposizione dei nodi (cerchio, albero, griglia, allineamento etc.), funzioni per l'inserimento di grafi con caratteristiche predefinite (orientazione archi, tipo di grafo, aggiunte di cicli speciali, sottografi etc.) e per la manipolazione del layout (geometria da adattare, adattamento alla finestra, manipolazione dei simboli etc.);
- una sezione di testo è riservata alla descrizione delle caratteristiche e a messaggi utili sulle procedure eseguite, tenendone memoria;
- sono presenti le funzioni Avanti / Indietro, Taglia / Copia / Incolla, Zoom, Snapshot Image, Image Navigator;
- è possibile salvare e mandare a stampa files, inserire il grafo in un file immagine e scorrere, direttamente in GOBLET, le immagini già in memoria di grafi;
- sono supportati secondo le indicazioni grafi contenenti fino a 2147483647 nodi, anche se le funzioni basate sugli algoritmi risolutori sono state pensate per problemi di applicazione pratica fino a 10000 nodi. Non conviene lavorare ad un numero maggiore causa lentezza e imprecisione di tali funzioni.

9.1. Visualizzazione

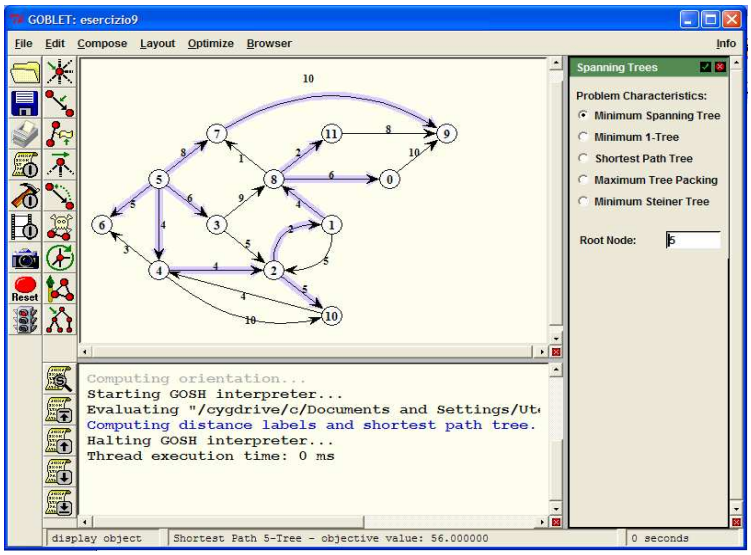


Figura 24: Visualizzazione in GOBLET di esempio svolto di problema di arborescenza minima su grafo pesato e orientato

All'apertura di GOBLET si apre lo spazio su cui si andrà a costruire il grafo, e una barra verticale su cui alcune icone permettono l'uso di diverse funzioni. Per facilitarne l'utilizzo, portando il cursore su ogni comando è possibile visualizzare su testo la rispettiva funzione. Partendo dai tradizionali **Apri**, **Salva**, **Stampa** sulla colonna più a sinistra, si prosegue con i comandi:

- **Visualizza / Aggiorna Messaggio**, agente sulla sezione di testo che descrive le procedure eseguite;
- **Visualizza / Modifica Oggetto**, per passare dalla modalità edit alla visualizzazione del grafo;
- **Image Navigator**, per la visione delle immagini create sui grafi costruiti;

- **Snapshot Image**, per effettuare istantanee del layout del grafo corrente costruito e passare alla modalità di visualizzazione a schermo *image navigation*;
- **Reset Browser**, per il reset del setting assegnato o della procedura in corso;
- **Stop / Restart Solver**, per fermare o far ripartire la risoluzione di un algoritmo.

Sulla colonna di destra dei comandi invece sono collocate tutte le funzionalità utili alla costruzione e alla manipolazione del grafo:

- **Inserisci Nodi**;
- **Inserisci Archi**;
- **Modifica Etichetta**, per modificare i pesi assegnati agli archi;
- **Muovi Nodi**;
- **Ridireziona Archi**;
- **Cancella Oggetti**;
- **Sort Incidences**, che evidenzia gli archi che entrano ed escono da un nodo;
- **Set Colours**, per assegnare con dei click su un nodo o arco il colore desiderato;
- **Set Predecessors**, comando che permette di cancellare o sostituire un arco che giunge nel nodo selezionato.

Per ognuna di queste opzioni, selezionare un'icona consente di inserire la modalità indicata ed applicarla nella sezione del grafo con un click dal mouse.

Sotto lo spazio di rappresentazione grafica ha sede una finestra in cui si possono visualizzare reports e messaggi per l'utente. Questa finestra funge da interfaccia di collegamento tra l'utente che costruisce e opera sul grafico e le operazioni di calcolo eseguite: alla fine dell'esecuzione vi compaiono messaggi di testo riguardanti risultato, prestazioni numeriche e di tempo, eventuali errori o problemi riscontrati. Anche a fianco della finestra dei messaggi sono collocate delle icone per velocizzarne l'uso: con **Search Messages** si può andare direttamente alla ricerca di un messaggio passato, rimasto in memoria; i comandi **Find Previous** e **Find Next** consentono di visualizzare il messaggio precedente o successivo al punto in cui ci si trova; **Find First** o **Last** il primo e l'ultimo messaggio tra tutti.

9.2. *Setting di funzioni e algoritmi*

Il primo passo nella costruzione di un grafo con GOBLET è sceglierne la tipologia. Dalla funzione **New**, tramite **File** sulla barra degli strumenti, si seleziona l'alternativa desiderata tra le varie proposte come *Mixed Graph*, la classe base per tutti i tipi di grafi, *Undirected Graph*, per grafi non orientati, *Sparse* o *Dense Graph*, per grafi a pochi o moti archi, etc.
Nella barra degli strumenti sono presenti tutti i comandi di cui l'utente dispone per la modifica del setting e del layout, le opzioni per la costruzione del grafo e l'esecuzione degli algoritmi. Alla voce **Edit** si trovano le funzioni di costruzione già espresse dalle icone illustrate in precedenza (Inserisci Nodi, Inserisci Archi etc.), inoltre, tra le altre, la funzione **Avanti** (*Undo Changes*), **Indietro** (*Redo Changes*) e **Constant Labels** che permette di assegnare un valore costante ai pesi di tutti gli archi in figura.
La parte dedicata alla finestra **Compose** tratta le opzioni riferite al grafo costruito o in costruzione. Tra quelle disponibili, ai fini della nostra analisi sui grafi, risultano interessanti le seguenti: **Orientation**, da cui si può rimpiazzare ogni arco non orientato con una coppia d'archi (archi doppi) oppure orientare ogni arco non orientato dal nodo dall'indice di

colorazione inferiore a quello d'indice superiore; **Subgraph**, usata per esportare un sottografo dal grafo considerato, indotto dalla colorazione di nodi, di archi etc.; **Add Random Arcs** aggiunge in maniera casuale uno specifico numero di archi al grafo corrente; **Random Generator** crea in maniera casuale etichette per i nodi e gli archi nel grafo esistente; altre funzioni che esulano dalla teoria dei grafi fin qui considerata, però degne di nota, sono **Node** e **Graph Splitting**, per dividere in gruppi di nodi il grafo.
Dalla finestra aperta su **Layout** sono accessibili le operazioni create per manipolare gli oggetti del grafo e le loro coordinate a schermo: con **Strip Geometry** si fanno traslare le coordinate dei nodi così che tutti siano nel quadrante positivo, al minimo le coordinate sono zero; **Scale Geometry** proporziona il corpo geometrico del grafo in una sezione delimitata; da **Node Grids** si può configurare e modificare la griglia a cui sottostà la disposizione di nodi e archi del grafo; **Fit to Window** adatta alla dimensione della finestra aperta del software il corpo del grafo; **Align Arcs** ridisegna gli archi così che i cicli e gli archi paralleli siano visibili; **Place Nodes** ridispone i nodi in layout prestabiliti come una griglia, un cerchio, per colore etc.; **Arc**, **Node Display** e **Layout Options** servono a modificare la visualizzazione di archi, nodi e del layout delle forme; **Zoom In** e **Zoom Out**.
Le funzioni per risolvere modelli e problemi basati su grafi sono stilate nel menu **Optimize**: **Spanning Trees** annovera, tra le altre, le opzioni *Minimum Spanning Tree* per il calcolo dell'albero minimo del grafo (basata sugli algoritmi di Kruskal e Prim) e *Shortest Path Tree* per trovare i cammini minimi a partire da un nodo sorgente (basato sugli algoritmi di Dijkstra e Bellman-Ford). Una finestra alla destra dello schermo (vedi Figura 24) permetterà di scegliere il risolutore ed inserire il nodo di partenza dell'esecuzione: se si inserisce il simbolo " * " invece di un nodo l'esecuzione parte dal nodo 0. In questa finestra poi vi sono possibilità di risoluzione o verifica di altri tipi di problemi: **Routing**, per trovare *cicli Euleriani*, *cammino critico* e lavorare col problema del *Travelling Salesman*; **Bipartitions**, per problemi come il *Maximum Erge Cut* o *Vertex Cover*; **Graph Partitions**, per la suddivisione in parti del grafo basata su diversi criteri; **Connectivity**, per calcolare le componenti connesse del grafo per un dato grado di connessione; **Ordering Problems**, utile in problemi di ordinamento di nodi topologico o secondo numerazione.
Da Optimize si accede anche ad alcune opzioni di configurazione per la risoluzione dei problemi: **Restart / Stop Solver**, per ripetere l'esecuzione con gli stessi parametri o fermare l'esecuzione; **Optimization Level**, restringe gli sforzi di calcolo in casi di problemi NP-difficili; **Method Options**, per configurare i vari risolutori di problemi; **Data Structures**, per selezionare alternative riguardo alla priorità delle code o le adiacenze dei nodi.
Resta da presentare solamente la sezione di comandi della voce **Browser** della barra: **Toggle Editor / Navigator** serve a passare da modalità editor a modalità display o modalità navigation (*image navigation*); **View / Update Messenger** permette di aprire la finestra dei messaggi sotto la rappresentazione grafica; **Tracing Options** configura il *tracing module*, cioè quanto spesso gli oggetti tracciati sono generati; **Browser Options** configura il browser, in particolare il trattamento del file e le caratteristiche della finestra; **Logging Options** specifica la quantità di informazioni da registrare che deve essere scritta dai risolutori di problemi.
Come già segnalato per la funzione Spanning Trees, per gran parte dei comandi sopraelencati, facenti essi parte della sezione Compose, Layout, Optimize o Browser, appare a destra della schermata grafica una sezione apposita su cui operare.

9.3. Esempi di calcolo

Ecco due esempi di problemi di albero minimo e cammino minimo risolti con GOBLET.

La prima schermata offre il layout del calcolo del Minimum Spanning Tree di peso totale 244 sul grafo pesato non orientato in figura.

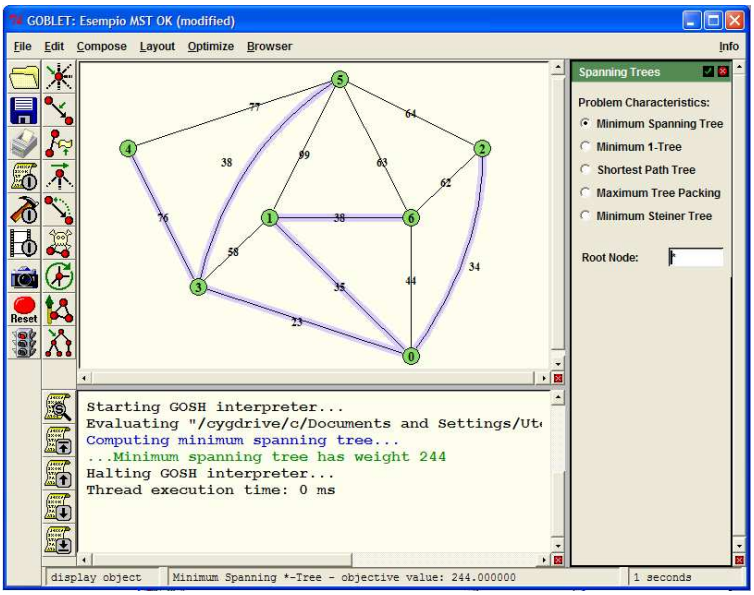


Figura 25: Esempio di risoluzione di problema di albero minimo con GOBLET

La seconda finestra qui di seguito mostra la visualizzazione dello Shortest Path dal nodo radice 10 a tutti gli altri nodi, per un grafo pesato e orientato.

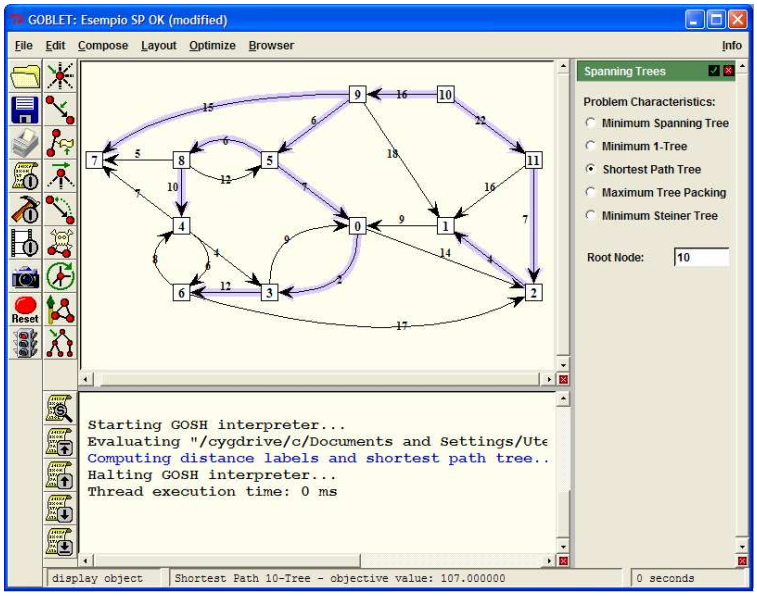


Figura 26: Esempio di risoluzione di problema di arborescenza minima con GOBLET

10. GraphThing

GraphThing è uno strumento pratico che permette di creare, manipolare e studiare i grafi. Dotato di un'interfaccia grafica semplificata, propone dei comandi d'uso intuitivo, votati all'essenzialità di funzioni utili e principali.
Questo software può essere anche utilizzato, a discrezione dell'utente, come programma base modificabile per lo sviluppo di software e progetti più complessi. Nella versione presentata, la più diffusa tra i siti utili nel download di software open source, GraphThing supporta semplici grafi e digrafi (grafi orientati), non ammettendo strutture ad archi multipli e acicli.

Ogni altra tipologia di grafo può essere invece sviluppata attraverso il software, le cui funzioni e caratteristiche principali sono qui presentate:

- aggiungere e cancellare vertici e archi direttamente da rappresentazione grafica cliccando da mouse; riposizionare oggetti con il 'click and drag'; assegnare un peso, che varia tra i valori 0 e 10000, ad archi sia orientati che non orientati;
- salvare e caricare files dei grafi costruiti; il numero massimo di nodi accettati non è definito, per le caratteristiche del software e per non caricare troppo l'esecuzione delle funzioni di risoluzione è consigliabile inserire un numero di nodi fino a 100;
- estrarre sottografi da grafi in costruzione o in uso; trovare il grafo inverso o complementare ad uno dato, cioè il grafo con gli stessi vertici del grafo di partenza, in cui però due nodi sono connessi da un arco solamente se non lo sono in quello di partenza; trovare il cosiddetto *line graph* di un grafo non orientato, cioè un grafo che rappresenta le adiacenze tra i nodi del grafo di partenza;
- sono disponibili per una creazione veloce dei modelli di grafi predefiniti e basati su diverse forme come grafo completo, a stella, a cerchio, null (senza archi) etc.;
- i risolutori permettono di trovare il MST e lo SP da un nodo ad un altro sia per grafi orientati che non orientati, con la presentazione tramite colorazione del risultato sul grafo costruito, e numero di peso totale minimo per SP; altre funzioni sono disponibili, tra queste *Connectivity* e *Eulericity*, al fine di sapere se il grafo è connesso o se è un grafo euleriano o semi-euleriano. Si dice euleriano un grafo che contiene un ciclo euleriano, cioè un percorso che passa per ogni nodo del grafo (cammino euleriano) e si conclude nel vertice di partenza. Un grafo semi-euleriano è un grafo che ammette un cammino euleriano, ma non il ciclo;
- visualizzare la matrice di adiacenza pesata del grafo rappresentato;
- tra le altre funzioni si citano il *Maximum network flow*, la polinomiale cromatica, il numero cromatico, *Breadth-First Search* e *Depth-First Search*;
- non è possibile modificare, se non con il riposizionamento degli oggetti, il layout grafico di archi e nodi: forme, colore e dimensione;
- in GraphThing è presente un'opzione Indietro, non sono presenti invece le funzioni Copia / Incolla, Avanti, Zoom, Stampa, Salva su file immagine.

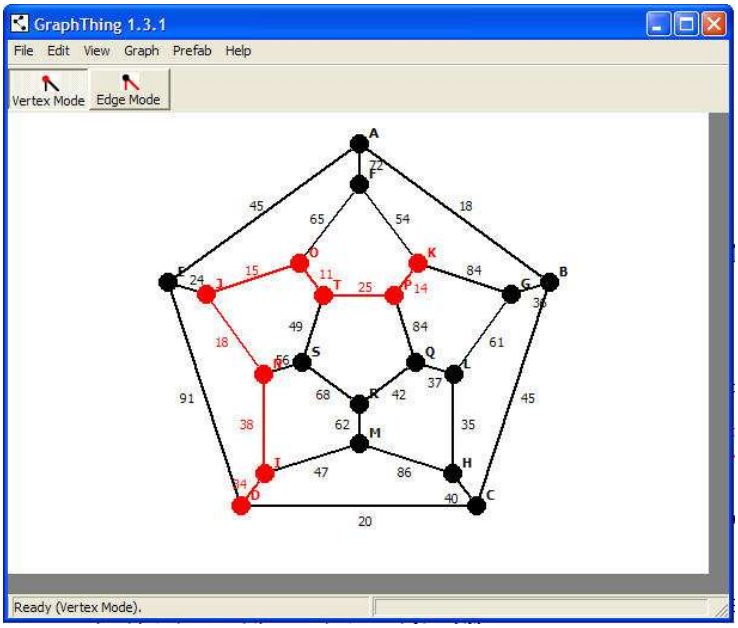
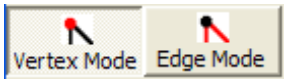
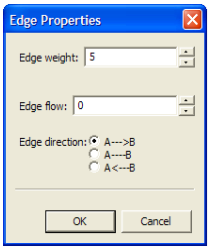


Figura 27: Visualizzazione in GraphThing di esempio svolto di problema di cammino minimo su grafo pesato non orientato

10.1. *Costruzione grafo*



Per la costruzione della rappresentazione grafica del grafo sono necessari solamente due comandi per due modalità: **Vertex Mode** ed **Edge Mode**.
Selezionando la modalità Vertex Mode, con un doppio click sullo sfondo vuoto si crea un nodo: questo nodo potrà essere riposizionando semplicemente cliccando e trascinando fino al punto desiderato, oppure cancellato premendo il tasto “canc” da tastiera dopo averlo selezionato. Per selezionare più di un nodo è sufficiente mantenere premuto il tasto “Shift” da tastiera.
Gli archi si inseriscono dopo aver cambiato modalità in Edge Mode: per far apparire l’arco si seleziona prima il nodo di partenza e poi il nodo d’arrivo; per cancellare un arco ancora lo si selezione e si preme “canc”; col tasto “Shift” è possibile selezionare più archi.



Se è necessario agire sulle proprietà di nodi e archi, con un doppio click sull’oggetto scelto si accede ai parametri modificabili relativi: di un nodo è possibile modificare il suo nome, di un arco invece si può cambiare il peso (alla creazione dell’arco viene assegnato il valore 1 di default), l’orientazione da un nodo ad un altro e l’*edge flow*, cioè un valore che è in entrata all’arco e non deve superarne il peso, usato nelle reti di flusso. Si considera bidirezionato un arco privo di orientazione.
Per generare un grafo GraphThing offre dei modelli predisposti di grafo da caricare e su cui l’utente può lavorare e apportare modifiche. Dalla barra degli strumenti alla voce **Prefab** appaiono per esempio: l’opzione **Complete** con cui, inserito un numero di nodi iniziale, si crea un grafo completo, in cui ogni nodo presenta un arco che lo connette agli altri nodi; l’opzione **Cycle** in cui, inserito un numero di nodi, questi andranno a disporsi su una circonferenza, mentre gli archi connetteranno un nodo solamente ai due nodi più prossimi; l’opzione **Hanoi** che presenta una rappresentazione di forma piramidale a più triangoli; l’opzione **Lattice** che dispone i nodi e gli archi su una griglia, senza ammettere archi sulle diagonali; l’opzione **Null** consente di inserire dei nodi disposti su una circonferenze senza presenza di archi; l’opzione **Platonic**, da cui si possono scegliere delle configurazioni che riprendono delle figure geometriche, come **Tetrahedric**, **Cubical**, **Octahedrical**, **Icosahedrical** e **Dodecahedrical**.
Si nota come sui modelli predefiniti per i quali è richiesto l’inserimento del numero dei nodi da costruire il numero ammissibile massimo è limitato caso per caso.

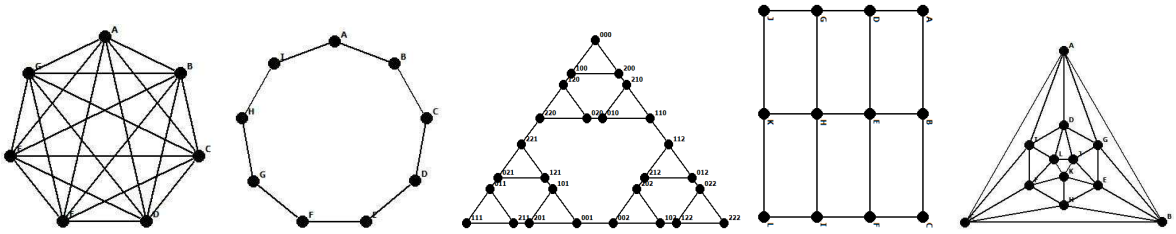


Figura 28: Alcuni esempi di modelli accessibili dall'opzione Prefab

10.2. *Funzioni e algoritmi*

Le funzioni e i risolutori di GraphThing sono accessibili dalla semplice barra degli strumenti in alto. Da **Edit** si trova l’opzione **Indietro** che permette di tornare alla modifica precedentemente effettuata e le opzioni che permettono la selezione degli oggetti: **Select All** evidenzia tutti gli oggetti costruiti; **Select None** deseleziona ciò che era evidenziato; **Invert Selection** per **Vertices**, **Edges** e **All** inverte la selezione di nodi archi o tutto ciò fino a quel punto selezionato.

Alla voce **View** si può effettuare la scelta di alcuni parametri da visualizzare o meno sulla rappresentazione, come le etichette nominali dei nodi per **Labels**, i pesi e i flussi degli archi per **Weights** e **Flows**.
Le opzioni ed i risolutori più interessanti per le analisi sul grafo si trovano nel menu della funzione **Graph**:

- **Clear** elimina tutto ciò che è stato creato fino a quel punto;
- **Complement** crea il grafo complementare al grafo in uso;
- **Line Graph** trova il grafo che rappresenta le adiacenze del grafo in uso;
- **Subgraph** permette di estrarre dal grafo un sottografo su cui poi lavorare;
- alla voce **Properties** vi sono le opzioni **Connectivity** e **Eulericity**;
- alla voce **Find** si trovano gli algoritmi per risolvere i problemi di **Minimum Spanning Tree** e **Shortest Path** tra due nodi, applicabili a ogni tipo di grafo;
- da **Statistics** si apre una finestra di più funzioni, delle quali la più interessante per l'analisi è l'**Adjacency Matrix**, che permette di visualizzare la matrice di adiacenza pesata del grafo costruito.

10.3. Esempi di calcolo

Ecco due esempi di problemi risolti con GraphThing: il primo è un grafo non orientato di cui è stato trovato l'albero minimo e resa visibile la matrice di adiacenza dei pesi.

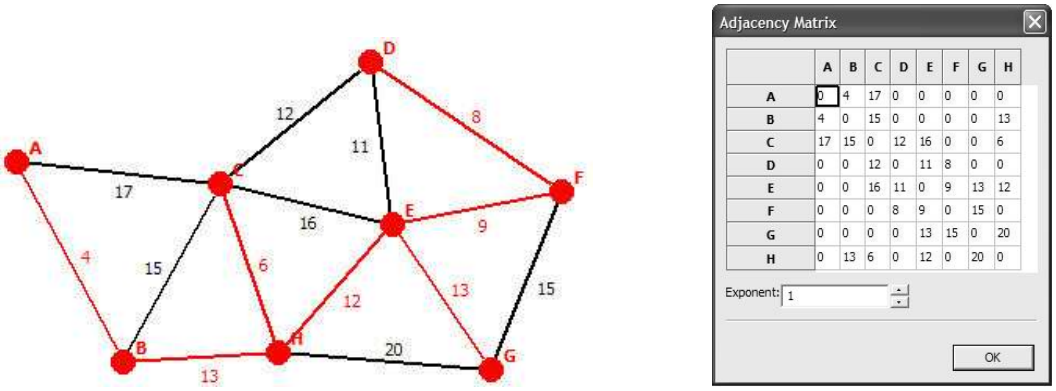


Figura 29: *Risoluzione di problema di albero minimo su grafo pesato non orientato, con relativa matrice di adiacenza dei pesi*

Il secondo esempio tratta il cammino di peso minimo trovato dal nodo A al nodo B del grafo orientato costruito in figura. Oltre alla soluzione grafica viene dato il risultato in termini di peso totale del cammino trovato (pari a 25) e viene costruita la matrice di adiacenza del grafo.

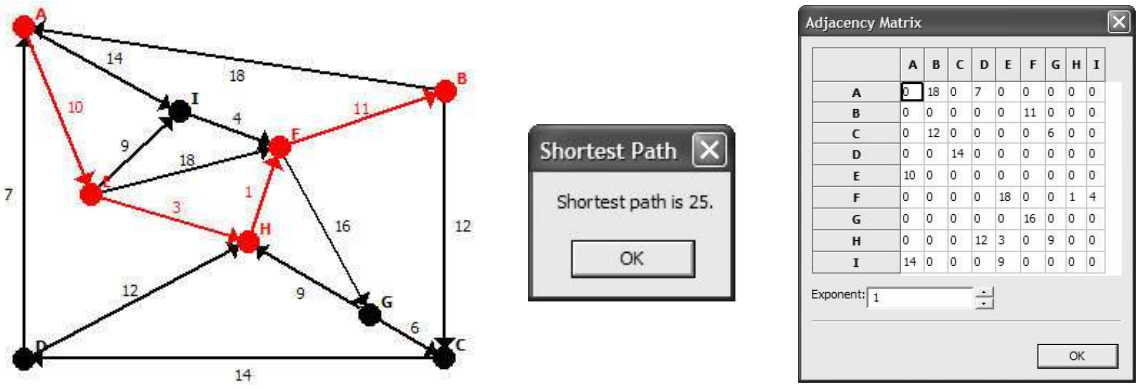


Figura 30: Risoluzione di problema di cammino minimo su grafo pesato e orientato, con relativa matrice di adiacenza dei pesi

11. JGraphEd

JGraphEd è un'applicazione creata come Java Graph Editor, per l'implementazione e l'utilizzo di grafi, e come struttura per il Graph Drawing. Nella pagina Web del software, figlia della pagina di Jonathan Harris, creatore del software, è possibile trovare informazioni chiare sulle funzionalità, le istruzioni d'uso, le impostazioni di download ed il *source code*. JGraphEd permette all'utente la facile costruzione e manipolazione di grafi aggiungendo e movimentando nodi e archi direttamente da rappresentazione grafica. Le molte funzionalità permettono una larga varietà di operazioni, tale da raggiungere e soddisfare le problematiche più diffuse applicate alla teoria dei grafi. La creazione di JGraphEd presuppone ancora come obiettivi lo studio diretto e pratico della teoria, la risoluzione di problemi e la costruzione di layout grafici. Come ogni software open source inoltre l'utilizzatore ha libertà di interazione, modifica e sviluppo, a patto venga reso poi di pubblica disponibilità il source code di tali modifiche.

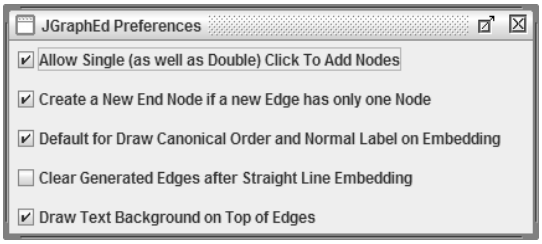
Caratteristiche principali:

- aggiungere, rimuovere, riposizionare e ridimensionare archi e nodi è possibile semplicemente utilizzando dei clicks da mouse; è possibile creare archi curvi, ortogonali o rettilinei;
- etichette di testo ed informazioni relative a nodi e archi sono presenti e possono essere modificate e visualizzate nel layout; è possibile modificare la colorazione degli oggetti;
- è possibile che archi siano generati da un algoritmo piuttosto che dall'utente, oltre alla creazione e disposizione random di nodi.
- selezionare, riproporzionare, ruotare o traslare il grafo o parti di esso sono operazioni disponibili;
- è permesso aprire e lavorare simultaneamente su finestre multiple e manipolare diversi grafi in maniera indipendente;
- diversi risolutori e algoritmi sono disponibili per garantire una copertura esauriente dei quesiti più diffusi sui grafi, e possono essere applicati a tutto il grafo oppure a sue parti: tra gli algoritmi sono evidenziati il risolutore *Minimum Spanning Tree* (basato sull'algoritmo di Prim) per il problema dell'albero minimo, e *Shortest Path For Two Nodes* (basato sull'algoritmo di Dijkstra) per il problema del cammino minimo tra due vertici;
- sono supportare le funzioni Avanti / Indietro, Apri, Salva e permette di salvare un grafo in un file immagine;
- non si possono modificare i pesi degli archi, né sono ammessi archi multipli o acicli;
- il numero massimo di nodi supportati non è segnalato, ma. date le caratteristiche grafiche e gli scopi del software, un limite ragionevole di funzionalità può essere considerato 1500-2000 nodi.

11.1. Visualizzazione

Una volta avviato JGraphEd, all'utente si presenta una barra degli strumenti con in evidenza molte funzionalità ed operazioni eseguibili ed uno spazio su cui si effettuerà la rappresentazione grafica. Per poter accedere alle suddette funzionalità e ai comandi dell'editor è necessario selezionare un file da aprire o tramite **Nuovo File** aprire una pagina vuota.

Già disponibili invece sono le opzioni **JGraphEd Help** e **Performances**. La prima rimanda ad una finestra in cui, tramite collegamento web, si possono



visualizzare istruzioni ed esempi utili d'utilizzo del software, la seconda invece permette l'accesso ad un menu popup in cui scegliere delle impostazioni generali: permettere la creazione di un nodo con un solo click da mouse; creare un nodo al rilascio del mouse alla fine di un arco che viene generato per trascinamento da un altro nodo; rappresentare il testo di etichette sopra gli archi nel caso di sovrapposizione etc.
Dopo aver aperto un file, quindi una sottofinestra bianca di rappresentazione, si può adeguarne la dimensione alla finestra in uso, per gestire lo spazio della rappresentazione a piacimento. È possibile a questo punto cominciare la costruzione della rappresentazione grafica del grafo voluto, anche a seconda delle impostazioni selezionate da Preferences: si consideri che con un click da mouse su un'area vuota si comanda l'inserimento di un nodo; per l'inserimento di un arco si clicca un nodo di partenza e si trascina il cursore o fino ad un nodo già esistente, o fino ad un punto vuoto dello sfondo, così da creare arco e nodo finale al rilascio del mouse; con un click in un nodo o un arco già esistente lo si seleziona, ed esso viene caratterizzato da un contorno con linea tratteggiata; per ricollocare un nodo è sufficiente selezionarlo e trascinarlo nella nuova posizione; è possibile curvare un arco a piacimento, selezionandolo e trascinando il suo punto di mezzeria; per orientare un arco è sufficiente ripetere l'operazione di creazione di tale arco, prendendo come nodo di partenza quello da cui l'orientazione è voluta uscente; non essendo possibili strutture come archi multipli o acicli, non sono possibili archi doppi, ma si può considerare un arco non orientato tra due nodi come due archi orientati vicendevolmente; il peso dell'arco corrisponde alla sua effettiva lunghezza nella rappresentazione grafica.

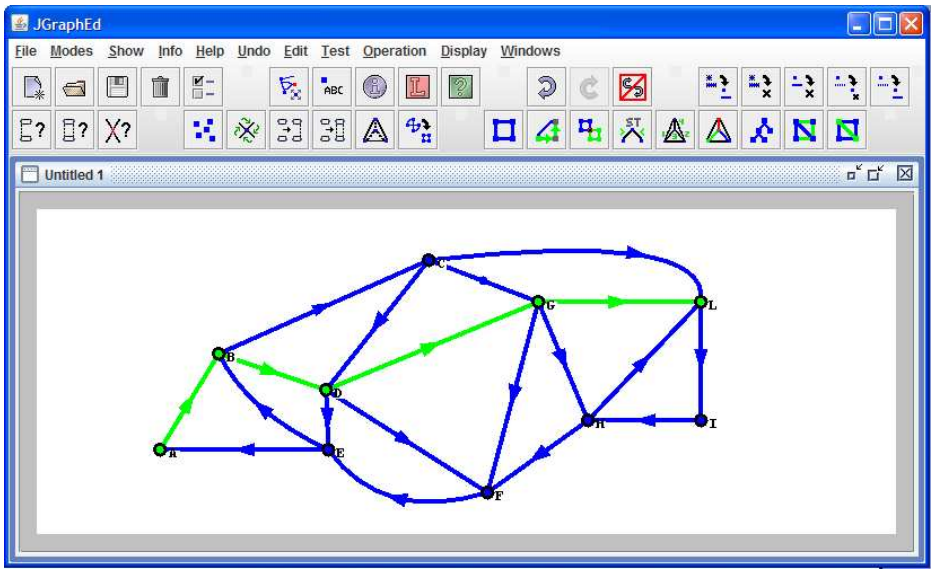


Figura 31: Visualizzazione in JGraphEd di esempio svolto di problema di cammino minimo su grafo non orientato.



Grazie a tre opzioni sulla barra l'utente può decidere di far inserire i nodi e gli archi del grafo allo stesso JGraphEd tramite algoritmi: l'opzione **Create Random Nodes** consente l'inserimento in una disposizione casuale di un numero scelto di nodi; l'opzione **Connect Graph** rende connesso il grafo rappresentato con archi che saranno segnati da una linea tratteggiata; l'opzione **Biconnect Graph** invece rende il grafo biconnesso, cioè aggiunge archi tratteggiati finché il grafo non è nello stato per cui eliminando un nodo rimarrebbe connesso.
Le proprietà e i parametri modificabili di nodi e archi sono accessibili con un click del

tasto destro del mouse sull'oggetto. Di un nodo sono evidenziate le coordinate e si può sceglierne nome e colore, mentre per un arco si possono vedere le coordinate dei nodi che connette, si può modificarne colore, orientazione e lo si può render rettilineo se curvilineo.

11.2. Comandi di funzioni e algoritmi

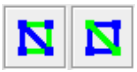
L'intera spiegazione su come sia possibile costruire un nuovo grafo o modificarne uno preesistente si basa sul fatto che di default JGraphEd sia in modalità Edit. Sono infatti possibili cinque modalità di stato diverse nel menu popup sulla barra degli strumenti:

- **Edit Mode** permette la costruzione e manipolazione del grafo, con aggiunta e modifica di nodi e archi, tutte le operazioni fin qui descritte;
- **Move Mode** permette il riposizionamento per traslazione dell'intero grafo semplicemente trascinandolo con il cursore;
- **Rotate Mode** consente di ruotare l'intero grafo attorno ad un polo posizionato a scelta, ancora con movimento del cursore;
- **Resize Mode** offre la possibilità di ridimensionare l'intero grafo, anche in maniera proporzionale, all'interno di una cornice.

Tra i comandi più utili nella fase d'utilizzo della modalità Edit vi sono quelli riferiti alle diverse tipologie di selezione: **Unselect All**, per deselezionare ciò che è evidenziato; **Remove Selected**, per cancellare ciò che è selezionato; **Remove All**, per cancellare tutto; **Remove Generated**, per rimuovere gli archi creati non dall'utente ma da funzioni del software; **Preserve Generated** trasforma gli archi tratteggiati generati dal software in archi a linea continua.
Oltre alle icone delle funzioni tradizionali **Apri**, **Salva**, **Chiudi Grafo Corrente**, **Avanti / Indietro** e **Abilita Avanti / Indietro** sulla toolbar sono presenti anche **Info**, **Log** e **Show**: con

Info si possono visualizzare in una finestra le caratteristiche del grafo rappresentato, come numero di nodi e archi, archi curvilinei, se il grafo è planare, numero di componenti connesse e biconnesse; l'opzione Log se selezionata apre una finestra che mostra l'elenco registrato in ordine cronologico delle esecuzioni degli algoritmi sul grafo, accompagnate dai rispettivi tempi; Show consente la scelta della visualizzazione dell'etichetta nome o delle coordinate dei nodi.

Per quanto riguarda le funzioni dei risolutori si notano tra i più semplici le verifiche di connettività, biconnettività e planarità del grafo: selezionando **Connectivity Test**, **Biconnectivity Test** e **Planarity Test** si ottiene la risposta al quesito corrispettivo.

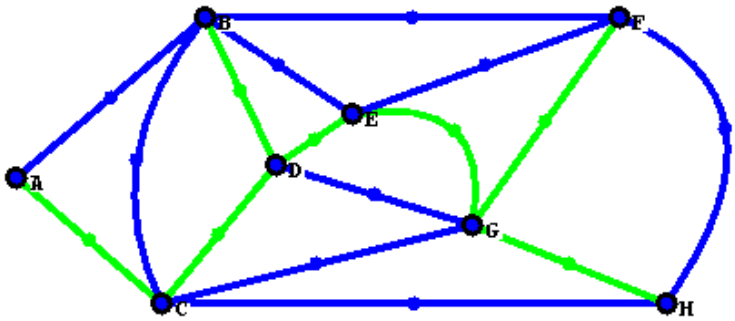


Tra i diversi comandi per la risoluzione di problemi sui grafi, tra cui si citano la *Depth First Search*, trovare le componenti biconnesse, problemi di ordinamento (*Canonical Ordering* e *Normal Labeling*) etc., vi sono anche le funzioni **Display Minimum Spanning Tree** e **Display Shortest Path For Two Nodes**. Minimum Spanning Tree implementa l'algoritmo di Prim per il calcolo dell'albero minimo per grafi non orientati, ma si può applicare anche a grafi orientati, a cui vengono eliminate le orientazioni degli archi. Il risultato si ottiene a schermo con la colorazione degli archi appartenenti alla soluzione. Nel caso in cui il grafo non sia connesso, la funzione esegue comunque il calcolo dopo aver generato gli archi necessari a renderlo connesso. Shortest Path offre un metodo risolutivo per trovare il cammino minimo tra due nodi, basato sull'algoritmo di Dijkstra per i cammini a radice singola. Anch'esso si può applicare a grafi orientati o non orientati e disegna direttamente sulla rappresentazione la soluzione: nel caso in cui un cammino da un nodo ad un altro non sia possibile, un messaggio di dialogo verrà visualizzato per l'utente.

In ogni momento è possibile ritornare alla situazione in cui il grafo si trovava prima dell'esecuzione di una funzione, grazie all'icona **Default**.

11.3. Esempi di calcolo

Sono presentati di seguito due esempi di calcolo: nel primo grafo non orientato si trova l'albero ricoprente minimo tramite la funzione Display Minimum Spanning Tree. Vengono riportate inoltre le informazioni di calcolo, con relativo tempo d'esecuzione, presenti nella finestra di Log: JGraphEd svolge innanzitutto un controllo sulla connettività del grafo, dopodiché ne calcola il MST evidenziato in verde.



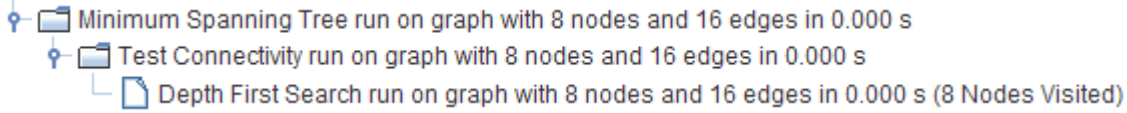
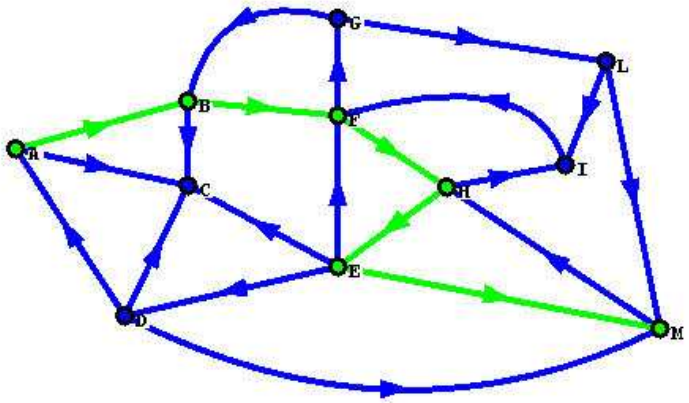


Figura 32: Risoluzione di problema di albero minimo su grafo non orientato e relativi dati d'esecuzione

Nel secondo esempio si calcola in grafo orientato il cammino di costo minimo che giunge al nodo M, partendo dal nodo A: la funzione Display Shortest Path For Two Nodes impiega circa 15 millisecondi per eseguire l'algoritmo e trova uno SP segnato in verde di lunghezza circa 583,762.



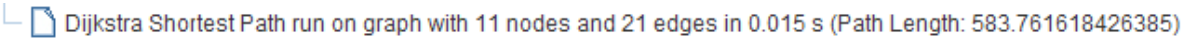


Figura 33: Risoluzione di problema di cammino minimo su grafo orientato e relativi dati d'esecuzione

12. MST algorithms

Il software MST algorithms è un elementare strumento per lo studio e la risoluzione di algoritmi per la ricerca dell'albero di peso minimo. Esso emerge, dopo la ricerca nei siti di software a libera diffusione, per la semplicità di setting e l'attenzione all'algoritmo. L'interfaccia è essenziale, così come le funzionalità per la costruzione in rappresentazione grafica e le opzioni di calcolo: fondamentalmente il software si focalizza sull'esecuzione degli algoritmi di Prim e Kruskal per il calcolo del MST e la loro visualizzazione. Non sono presenti impostazioni di layout da modificare, risolutori di problemi diversi, altri testers come potevano essere la verifica della connettività o la presenza di cicli euleriani, modelli predefiniti o modalità di inserimento automatico di archi e nodi etc. Osservate quindi queste caratteristiche, MST algorithms si presta bene ad una realtà di apprendimento pratico e specifico, grazie alla semplicità ed immediatezza d'utilizzo, e alla risoluzione di problemi di piccole dimensioni per quantità di nodi e archi.

Caratteristiche principali:

- la costruzione grafica consiste nella semplice creazione di nodi e archi pesati non orientati cliccando da mouse, con possibilità di riposizionamento; sono ammessi valori di pesi negativi per archi unicamente non orientati;
- vi sono due unici algoritmi risolutivi per l'unico problema dell'albero di peso minimo su grafo non orientato: l'implementazione dell'algoritmo di Prim e dell'algoritmo di Kruskal;
- si può visualizzare la soluzione dell'algoritmo su una finestra separata: è disponibile la visione dell'esecuzione step by step oppure in autonoma animazione;
- sono possibili il salvataggio e il caricamento di files;
- non è possibile intervenire sul layout grafico per modificare conformazione e colorazione degli oggetti; non sono ammesse strutture come archi multipli o acicli; funzioni come Zoom, Avanti / Indietro, Fit to window etc. e rappresentazioni a matrice sono assenti;
- non vi è limitazione sul valore del peso di un arco; non è indicata nemmeno per il numero di nodi, ma non è consigliabile lavorare con un numero superiore a 150: le caratteristiche fisse del layout precluderebbero la chiarezza e la semplicità, scopi base del software.

12.1. Costruzione del grafo

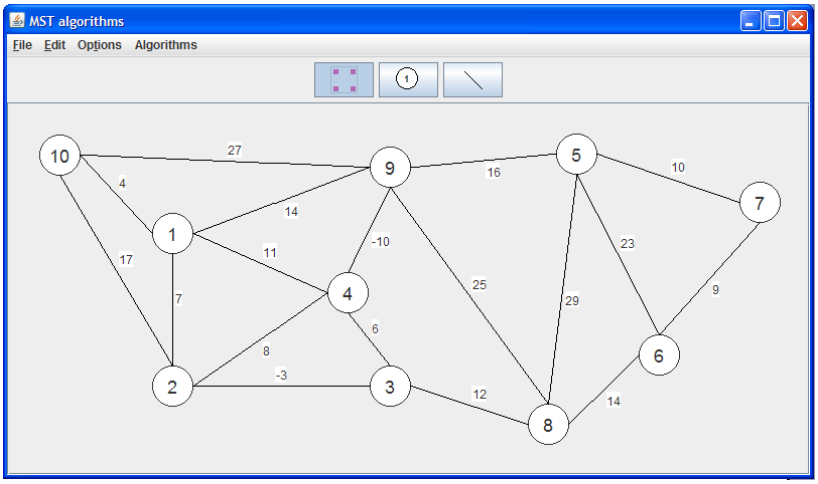


Figura 34: Visualizzazione MST Algorithms di grafo pesato non orientato

MST algorithms, dall'accesso al file d'esecuzione in linguaggio Java, permette di lavorare in due fasi: la costruzione grafica e la risoluzione dell'algoritmo scelto. Le due fasi sono autonome e interessano delle finestre distinte: è possibile infatti mandare in esecuzione un algoritmo, dopo aver costruito un grafo, e poi modificare il grafo senza produrre cambiamenti nella finestra di risoluzione.
La pagina di apertura del software è la pagina su cui si crea la struttura di nodi e archi. Dalla spoglia barra si notano tre comandi con icona che rappresentano le tre modalità di stato per la

manipolazione del grafo: il primo comando, preselezionato di default, permette di lavorare nella **modalità di movimentazione** nodi: è possibile selezionare e trascinare quindi un nodo fino alla posizione desiderata; il secondo comando ha come simbolo un nodo: sarà possibile in questa **modalità creare un nodo** nello spazio vuoto sottostante con un click; il terzo comando è il comando per l'**inserimento degli archi**: si può creare un arco cliccando dal tasto sinistro del mouse il nodo di partenza fino al nodo d'arrivo, rilasciando solo su quest'ultimo il tasto. Eseguita la creazione di un arco (strettamente non orientato) immediatamente verrà aperta una finestra da cui l'utente potrà inserire il valore del peso di tale nodo. Il peso viene visualizzato accanto al segmento dell'arco ed è possibile modificarne il valore ripetendo l'operazione di inserimento di arco tra gli stessi nodi, digitando poi un valore differente di costo.
I nodi sono rappresentati di forma circolare con colorazione blu e all'interno il numero progressivo di creazione: non è possibile modificarne né dimensioni né nome o colore. Gli archi sono fatti partire da un punto sul contorno del cerchio del nodo: vi sono in realtà solo quattro punti, visto questo confine come una circonferenza, posti a 0, 90, 180 e 270 gradi.
La cancellazione di un elemento appena costruito può essere fatta tramite i tasti "Ctrl+D" oppure alla voce **Delete** da **Edit** della barra degli strumenti. In effetti dalla toolbar è possibile **Salvare** o **Caricare**, da **File**, il grafo e unicamente far partire l'esecuzione degli algoritmi risolutori (da **Algorithms**). Nessuna altra funzione è supportata.

12.2. *Algoritmi*

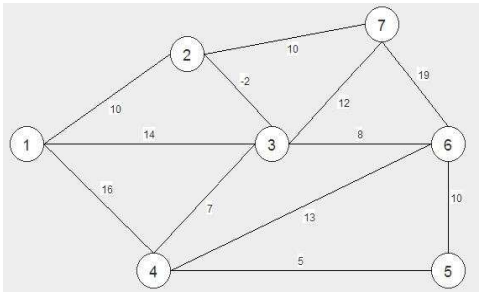
MST Algorithms permette l'esecuzione degli algoritmi di Prim e Kruskal dalla voce Algorithms. Per ogni algoritmo appaiono due voci tra cui scegliere: **Adjacency Lists** o **Adjacency Matrix**. Questa differenza sta nel basare il trattamento dei dati in input del grafo all'algoritmo scelto o sulla matrice di adiacenza o sulla lista di adiacenza. Sostanzialmente si può dire che l'utilizzo della lista di adiacenza è più indicato per grafi sparsi, in cui il numero di archi è significativamente minore del numero dei nodi, viceversa per la matrice di adiacenza.
Perché il risolutore possa procedere al calcolo del MST è necessario che il grafo sia connesso: a questo punto, scegliendo una delle quattro voci per la risoluzione, una nuova finestra con riportato lo schema del grafo costruito si apre presentando tre funzionalità. La prima, **Solve**,

consente di trovare e visualizzare con una colorazione evidente sul grafo il MST cercato; la seconda, **Step by Step**, permette con un click di visualizzare i vari archi scelti durante l'avanzare della procedura; infine **Animation** presenta un'animazione dei passi d'avanzamento dell'algoritmo e quindi della costruzione dell'albero minimo, di cui si può controllare la velocità.

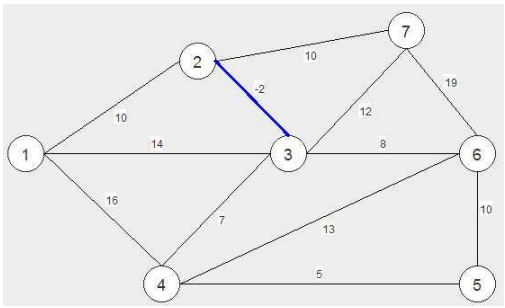
12.3. *Esempio di calcolo*

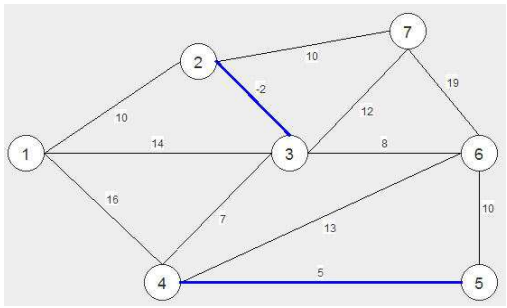
Si applica l'algoritmo di Kruskal al grafo non orientato rappresentato di seguito.

Quindi si sfrutta la funzionalità Step by Step per osservare la progressiva scelta degli archi da inserire nel MST in costruzione.

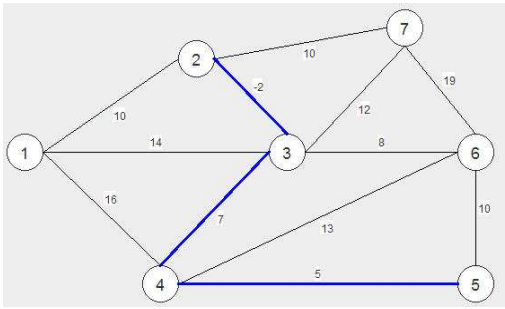


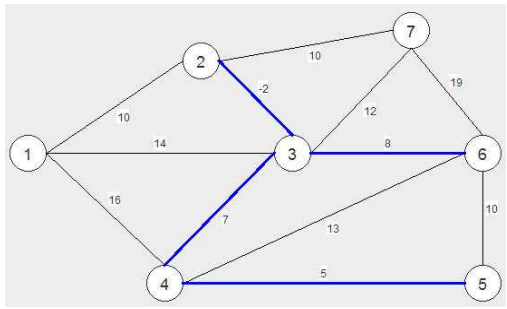
Riordinati gli archi in ordine crescente in base al loro peso, si prende l'arco 2-3 di peso minimo -2 come primo componente segnato in blu del MST da trovare. Dopodiché l'arco 4-5 ha peso minimo 5 tra quelli non ancora inseriti.



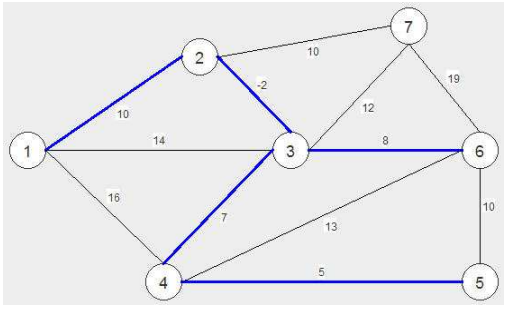


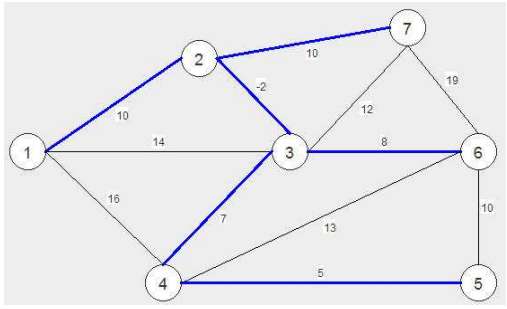
Vengono inseriti gli archi 3-4 e 3-6, rispettivamente di pesi 7 e 8.



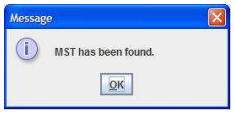


Ora si nota che gli archi 1-2, 2-7 e 5-6 hanno tutti lo stesso peso 10: l'arco 5-6 non può essere inserito poiché connette due nodi già toccati dal MST, si creerebbe un ciclo. La scelta tra 1-2 e 2-7 risulta quindi arbitraria. Si sceglie 1-2 e al passo successivo risulta che 2-7 è l'arco di peso minore tra i rimanenti.





L'albero evidenziato infine connette tutti i nodi del grafo, e quindi rappresenta l'albero di peso minimo 38.



13. MST Java V1.5

Il software MST Java è, come per MST algorithms, un semplice e basilare strumento per il calcolo dell'albero ricoprente minimo su grafi pesati non orientati: le funzionalità sono ridotte al minimo sufficiente e l'utilizzo comprensibile ed intuitivo. Non sono presenti altre opzioni se non quelle strettamente necessarie all'introduzione e salvataggio dei dati, poca attenzione è data allo sviluppo grafico, mentre emerge la stringente attinenza alla risoluzione del problema.

Caratteristiche principali:

- in input il software richiede un file di testo ASCII ordinario (esempio con programma Blocco note) con una struttura per l'inserimento su ogni riga del tipo: A B C. Gli archi e i rispettivi pesi sono inseriti digitando per ogni arco in posto A il primo nodo, in posto B il nodo connesso dall'arco al primo nodo, in posto C il valore del peso. I tre parametri sono separati da un semplice spazio. Terminato l'inserimento di un arco si procede con i seguenti sempre andando a capo riga per ognuno di essi. Sono ammessi valori negativi dei pesi;
- data la struttura in input da elenco su pagina di testo, sono possibili archi multipli con pesi differenti e acicli;
- il risolutore per il calcolo del MST è basato sull'algoritmo di Kruskal e i risultati ottenuti sono riportati in un report di testo tabellare in output. Nel caso di molteplici soluzioni dal peso equivalente il software restituisce uno degli alberi minimi possibili;
- è possibile salvare l'output dei risultati e caricare files da files di testo, ottenere una stilizzata rappresentazione grafica dell'albero ricoprente minimo su una finestra separata, anche se di scarsa utilità se non per un colpo d'occhio intuitivo;
- non sono disponibili altre funzioni o comandi per la risoluzione di problemi differenti, nessuna attenzione o possibilità di modifica verso il layout grafico o per altri tipi di rappresentazioni del grafo;
- in numero limite di archi e il valore massimo del peso non sono specificati, si rimanda alla necessità e la logica dell'utente vista la modalità di inserimento dati del software.

13.1. Comandi di funzioni

Figura 35: Visualizzazione in MST Java V1.5 di tabella dei risultati e finestra di Draw per esempio di problema di albero minimo su grafo non orientato, con documento in ingresso da Blocco Note

Nella pagina di apertura del programma si vedono alcuni comandi sulla barra in alto ed uno spazio bianco su cui verranno esposti i risultati. Con il tasto **Load** è possibile caricare il grafo costruito in file di testo, con **Save** invece salvare i risultati di un'esecuzione dell'algoritmo. **Clear** permette di cancellare i risultati mandati in output e visualizzati nelle finestra, mentre **Draw** consente di far apparire in una finestra separata la soluzione dell'albero minimo con una schematizzata rappresentazione grafica.
Sulla barra degli strumenti in alto da **File** si può accedere ancora ai tasti appena elencati, mentre da **Help** sono disponibili delle rapide istruzioni sul funzionamento del software.

13.2. *Esempio di calcolo*

Di seguito si esegue un esempio di calcolo dell'albero minimo utilizzando il software: il grafo pesato non orientato in figura deve essere ricondotto a dei parametri inclusi in una pagina di testo di Blocco note, per poter essere caricato in input a MST Java.

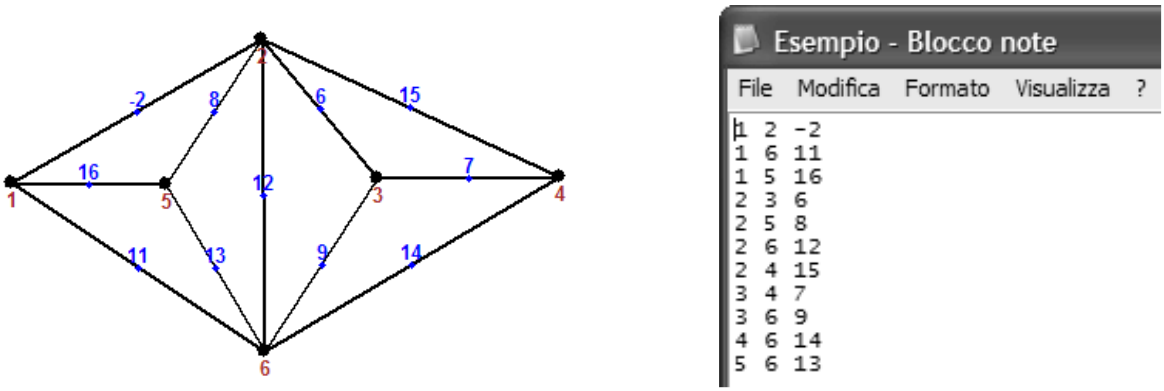


Figura 36: Esempio di compilazione di foglio Blocco Note con dati relativi al grafo di figura

Quindi, immediatamente dopo il caricamento, avviene in modo automatico il calcolo dell'albero ricoprente minimo: viene rappresentato schematicamente nella finestra di Draw e i suoi archi sono elencati in forma tabellare assieme al valore di peso totale minimo 28.

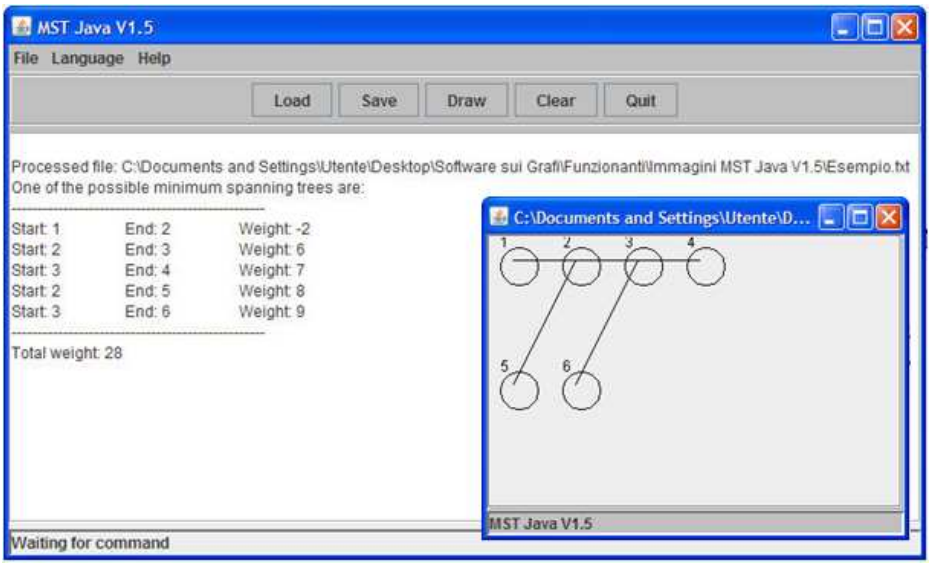


Figura 37: Risultati in forma tabellare e con finestra di Draw di problema albero minimo su grafo non orientato

14. DijkstraVis

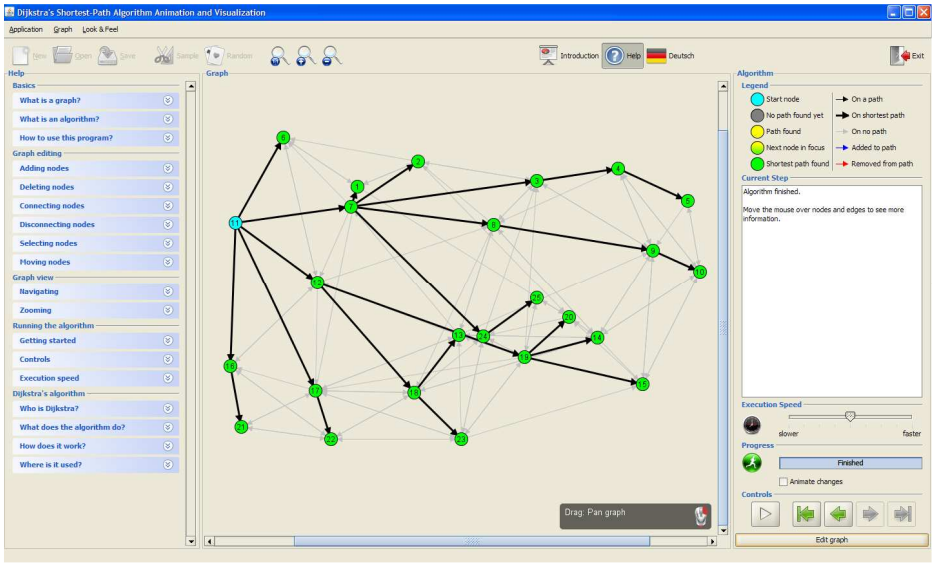


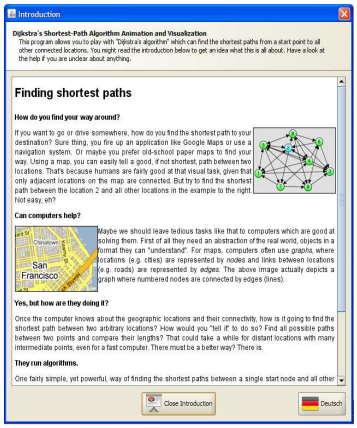
Figura 38: Visualizzazione in DijkstraVis di esempio svolto di problema di arborescenza minima su grafo orientato

Il programma ha come obiettivo primo la visualizzazione del processo di calcolo dei cammini minimi da radice unica con algoritmo di Dijkstra. La buona semplificazione d'uso ha l'intento di focalizzare l'attenzione sullo svolgimento della procedura risolutiva: passo dopo passo l'utente può visualizzare informazioni grafiche e di testo, nonché scorrere i singoli steps avanti o indietro. La complessità è mantenuta di basso livello per permettere un'interazione efficace anche con l'utente meno esperto. La funzione didattica ed espositiva, assieme alla possibilità di risoluzione di problemi anche non banali, risultano essere le caratteristiche distintive e gli scopi finali di DijkstraVis.

Caratteristiche principali:

- la costruzione del grafo è assistita da indicazioni in aiuto all'utilizzatore ed avviene per rappresentazione grafica diretta, con funzionalità di riposizionamento, selezione ed eliminazione di archi e nodi; gli archi sono rettilinei ed orientati, con ammessi archi bidirezionati ma non acicli; i pesi sono assegnati in base alla lunghezza dell'arco e non sono modificabili;
- sono disponibili modelli di grafi da caricare e su cui lavorare, ed una funzione di creazione di grafi di struttura casuale, a seconda della dimensione del numero di nodi voluto;
- la sezione dedicata all'esecuzione dell'algoritmo risolutivo è completa e ben concepita: la risoluzione è pensata sulla dimostrazione per via animata della procedura di calcolo sul grafo, con legenda chiara, messaggi di testo e comandi all'animazione;
- sono disponibili funzioni di caricamento e salvataggio di files, Zoom e una funzione Help di spiegazione per lo svolgimento delle operazioni di manipolazione grafica e riguardanti l'algoritmo di Dijkstra;
- il layout del grafo non è modificabile per forme e colori, e così ogni altra funzione non strettamente connessa con la costruzione e l'esecuzione dell'algoritmo risulta assente;
- il numero massimo consigliato di nodi con cui lavorare, in un grafo denso, è 500: non è un limite di progetto del software ma, visto il fine del programma, un limite dovuto all'efficacia e ai tempi di svolgimento di algoritmo e supporto di layout.

14.1. *Visualizzazione*



Il programma inizia con la visualizzazione di una finestra introduttiva in cui si presenta all'utente il problema dei cammini minimi e come un computer, tramite un software su cui è implementato il corretto algoritmo, possa essere utile nella risoluzione al problema. L'introduzione può essere chiusa e richiamata in seguito tramite la funzione **Introduction** sulla barra degli strumenti. Chiusa dunque questa finestra si accede alla finestra principale di DijkstraVis: sostanzialmente è suddivisa in una barra degli strumenti in alto, l'area grafica al centro per la costruzione del grafo e l'area dedicata allo svolgimento e alle informazioni dell'algoritmo di Dijkstra sulla destra.
Quindi è chiaro che si lavora sempre in una delle due modalità disponibili: **Edit**, per la costruzione della rappresentazione grafica, e **Algorithm**, per l'esecuzione dell'algoritmo. Quando si è nella modalità di costruzione e manipolazione la sezione relativa all'algoritmo è oscurata per non creare distrazione.
Le funzioni di **Zoom** sulla barra, al di là del riposizionamento e della manipolazione del grafo, sono le uniche atte alla modifica del layout.
Dalla voce **Help** si può far apparire una finestra in cui sono presenti informazioni utili all'utente sulla teoria dei grafi, sulle modalità di costruzione di un grafo e di applicazione dell'algoritmo, sulle opzioni di visualizzazione e l'algoritmo di Dijkstra in generale.

14.2. *Costruzione di un grafo*

All'apertura del programma si è di default nella modalità Edit: nella parte inferiore dello spazio dedicato alla rappresentazione grafica sono presenti delle finestre semitrasparenti in cui vengono indicate le istruzioni d'uso di tasti di mouse e tastiera. A seconda degli elementi selezionati o da creare gli stessi tasti cambiano funzione. Per creare un nodo è necessario un doppio click con il tasto sinistro del mouse, per selezionarlo un solo click sul nodo. Per riposizionare un nodo soltanto vi si clicca con il tasto destro del mouse e lo si trascina mantenendo premuto, se invece si fa lo stesso su una zona priva di elementi, tutto il grafo trasla. Per creare un arco si tiene premuto invece il tasto sinistro dal nodo di partenza fino al nodo di arrivo. È ammessa la possibilità di creare archi bidirezionati attraverso l'inserimento di due archi con nodi di partenza e arrivo invertiti. Selezionando con il tasto destro su un arco è disponibile l'opzione Delete per cancellarlo, su un nodo invece si visualizza un menu in cui è possibile modificarne il nome, cancellare il nodo stesso, impostarlo come nodo di partenza o intermedio per l'applicazione poi dell'algoritmo. Passando con il cursore semplicemente sopra un oggetto appare una finestra semitrasparente che per i nodi ne identifica numero e nome, per gli archi lunghezza e nodi che connette.

DijkstraVis rende disponibili anche dei grafi preimpostati di esempio, utilizzabili e modificabili una volta selezionato dalla barra degli strumenti l'icona **Sample**. Il comando **Random** accanto a questa permette invece la creazione con posizionamento dei nodi e inserimento di archi casuali, per grafi di "taglia" più o meno grande (da 9 a 225 nodi).
L'importazione di un grafo creato in precedenza o il salvataggio di un file creato sottostanno ai comandi **Open** e **Save**.

14.3. *Algoritmo risolutivo*

Una volta conclusa la fase di costruzione e scelto un nodo come nodo di partenza per l'algoritmo, si può accedere alla sezione di calcolo tramite il comando **Run algorithm** in basso a destra nella schermata. Entrati nella modalità dell'algoritmo sono bloccate momentaneamente le modifiche al grafo. L'area Algorithm si resa quindi accessibile si compone di queste parti: una prima parte è occupata dalla legenda delle colorazioni assunte da nodi e archi fondamentale per seguire chiaramente i vari passi della procedura di calcolo; la seconda parte, denominata Current Step, fornisce in una finestra di testo la descrizione del singolo passo in cui l'algoritmo si trova e le informazioni relative alle operazioni eseguite; un cursore su una barra permette la scelta della velocità dell'animazione grafica; una barra permette di conoscere lo stadio visualizzato raggiunto sul totale del processo, il numero totale di steps è precalcolato in base al numero di nodi raggiungibili; infine gli ultimi comandi offrono la possibilità di avere immediatamente il risultato di ogni passo a schermo oppure l'animazione grafica: dell'animazione si possono controllare partenza e pause, farla avanzare e indietreggiare. Passando sui nodi con il cursore si visualizza se sono già stati processati e la distanza lungo il cammino minimo dal nodo di partenza.
Terminata l'esecuzione con il tasto Edit Graph si ritorna in modalità Edit.

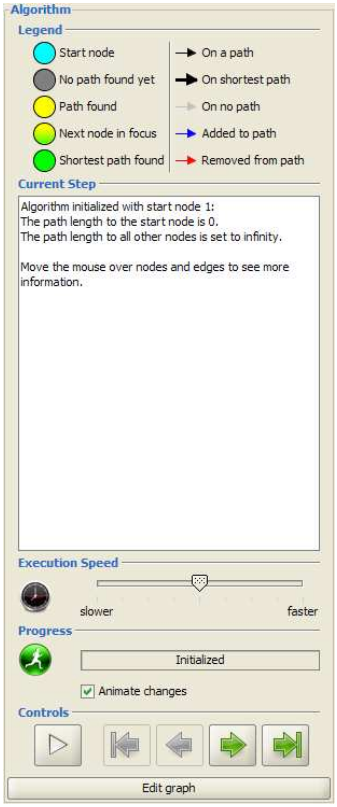
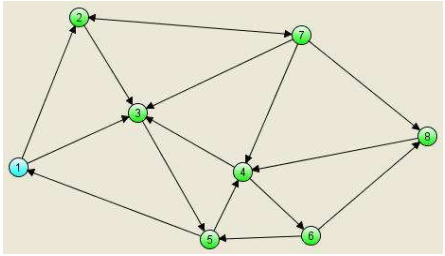


Figura 39: Finestra di messaggi e comandi dell'algoritmo

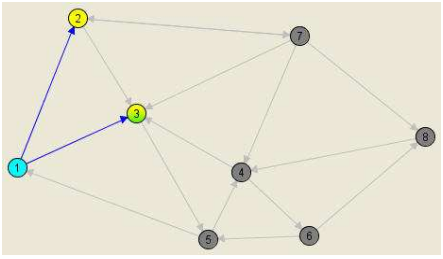
14.4. *Esempio di calcolo*

Creato un grafico orientato e con archi pesati secondo la loro lunghezza, si esegue l'algoritmo di Dijkstra per la ricerca dell'arborescenza minima da un nodo radice, con il supporto dell'animazione grafica.



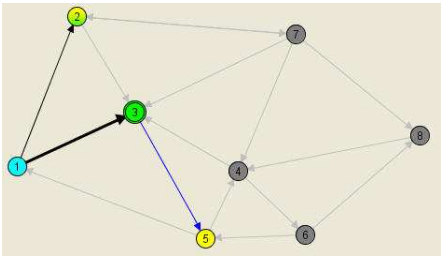
Partendo dal nodo radice 1 di colore azzurro, l'algoritmo procede considerando gli archi uscenti, segnati in blu, che lo collegano con i nodi adiacenti: questi, se esistenti, prendono il colore giallo; in giallo con sfumatura verde invece viene segnato il nodo con cammino da nodo 1 minimo tra quelli trovati. Il nodo con cammino minore da 1 in questione è il 3, che viene automaticamente inserito nell'arborescenza minima e colorato pienamente di verde. Assieme al nodo 3 anche l'arco che lo connette ad 1 viene evidenziato, e il nodo 3 sarà prossimo nodo considerato.
Da qui il procedimento si ripete, cercando per ogni nodo aggiunto all'arborescenza i nodi a questo adiacenti, e confrontando i cammini totali ai nodi raggiunti dal nodo radice.
I nodi e gli archi non ancora presi in considerazione sono di colore grigio.

A fianco di ogni figura, che rappresenta uno step dell'algoritmo, è visibile la finestra Current Step relativa in cui si descrivono le operazioni eseguite.



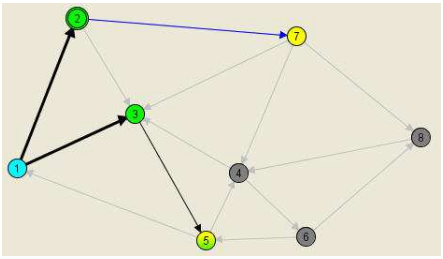
Expanding node 1
- 2 paths found

Node 3 has the minimum path length (138.7) of all non-green nodes. Any other path to node 3 visits another yellow node, and will thus be longer.
Node 3 will be colored green to indicate that the shortest path has been found.



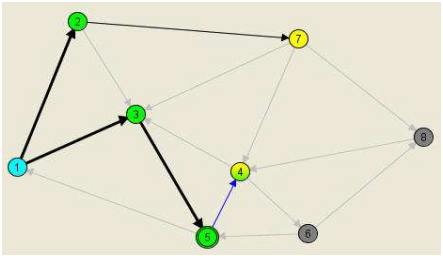
Expanding node 3
- 1 path found

Node 2 has the minimum path length (172.3) of all non-green nodes. Any other path to node 2 visits another yellow node, and will thus be longer.
Node 2 will be colored green to indicate that the shortest path has been found.



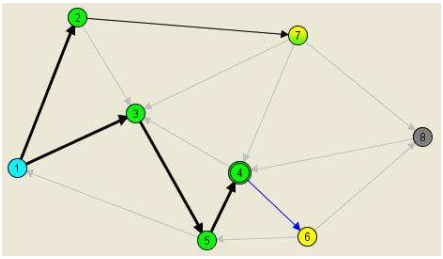
Expanding node 2
- 1 path found

Node 5 has the minimum path length (293.6) of all non-green nodes. Any other path to node 5 visits another yellow node, and will thus be longer.
Node 5 will be colored green to indicate that the shortest path has been found.



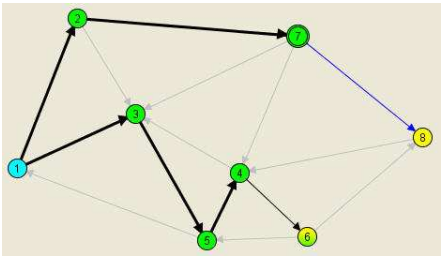
Expanding node 5
- 1 path found

Node 4 has the minimum path length (373.7) of all non-green nodes. Any other path to node 4 visits another yellow node, and will thus be longer.
Node 4 will be colored green to indicate that the shortest path has been found.



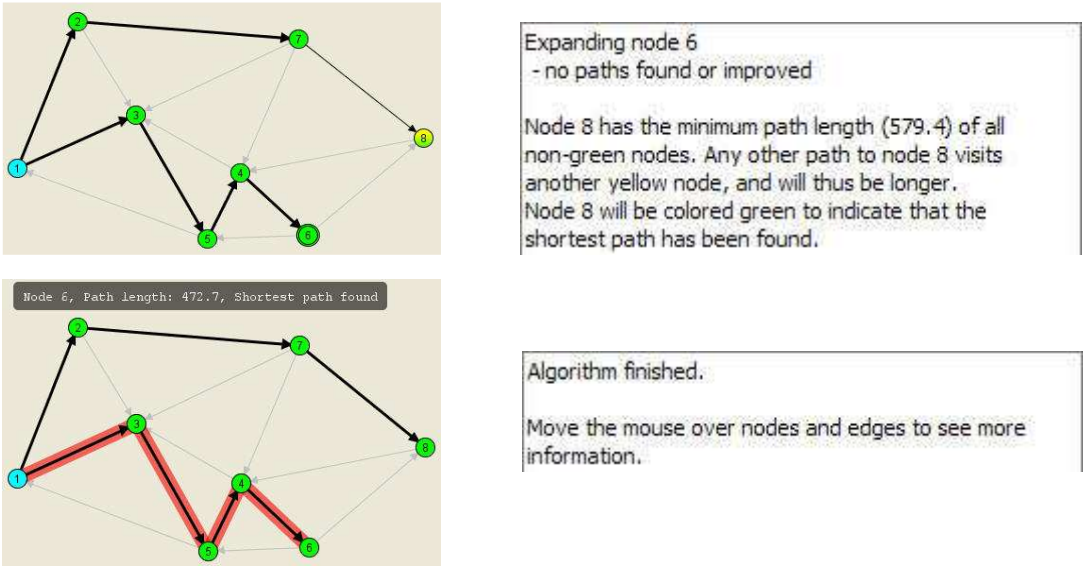
Expanding node 4
- 1 path found

Node 7 has the minimum path length (408.1) of all non-green nodes. There are no other arrows coming in to node 7.
Node 7 will be colored green to indicate that the shortest path has been found.



Expanding node 7
- 1 path found

Node 6 has the minimum path length (472.7) of all non-green nodes. There are no other arrows coming in to node 6.
Node 6 will be colored green to indicate that the shortest path has been found.



È possibile visualizzare in rosso il cammino minimo dalla radice per un nodo facente parte dell'arborescenza minima semplicemente muovendo il cursore del mouse sopra tale nodo: appare inoltre una finestra semitrasparente su cui vengono indicati nome del nodo, lunghezza del cammino e se un cammino è effettivamente stato trovato.
Nell'ultima figura dell'esempio è evidenziato il cammino minimo dal nodo radice 1 al nodo 6, di lunghezza 472,7.

15. PathFinder

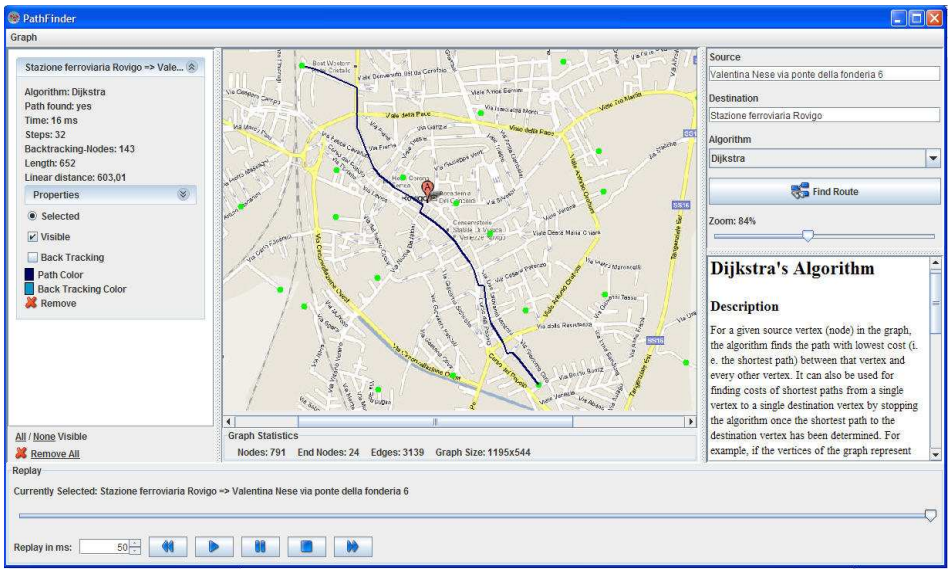


Figura 40: Visualizzazione in PathFinder di esempio svolto di problema di cammino minimo su sfondo di città di Rovigo

Il software ha come scopo l'individuazione del cammino minimo tra due nodi di un grafo. Non è però un tradizionale strumento di calcolo con grafi: PathFinder è ideato per lavorare su un file d'immagine, posto sullo sfondo della rappresentazione dei grafi da creare. La presenza dell'immagine (un percorso, una mappa, un'area geografica, uno schema etc.) in relazione alla quale il grafo viene costruito, permette una visualizzazione che in un certo senso è lo scopo di questo tipo di struttura: un impatto efficace di layout del file.

A discapito di ciò però risultano deboli alcune opzioni in fase di *editing* del grafo, sono accettati infatti solamente archi non orientati e pesati a seconda della loro effettiva lunghezza. L'algoritmo di Dijkstra è presente tra i risolutori del problema di cammino minimo. Si citano anche A*, Depth First e Breadth First tra gli algoritmi presenti.

Caratteristiche principali:

- è necessario importare un file immagine per poter procedere alla costruzione del grafo: tale grafo si compone di archi non orientati, di peso pari alla lunghezza, e nodi inseriti con dei clicks da mouse; non sono ammesse strutture come archi multipli e acicli; i nodi dei punti di interesse segnati con dei nomi al momento dell'esecuzione saranno gli unici visualizzati, il resto della rappresentazione sarà non visibile se non dopo aver mandato in esecuzione l'algoritmo e aver trovato il risultato;
- è disponibile la funzione Zoom, come le opzioni di caricamento e salvataggio di files;
- di ogni algoritmo è fornita una descrizione delle caratteristiche, proprietà ed implementazione;
- PathFinder offre l'animazione della procedura di calcolo sulla rappresentazione grafica, con la possibilità di controllarne visualizzazione, stato e velocità;
- il numero massimo di nodi e archi non è indicato, ma per la capacità funzionale e le finalità grafiche 1000 nodi è un buon numero limite.

15.1. Graph Builder

PathFinder presenta una finestra che si compone di quattro sezioni principali: al centro vi è lo spazio per la rappresentazione grafica e il layout del grafo; sulla destra una colonna in cui sono posizionati i comandi per l'esecuzione degli algoritmi risolutivi; in basso il controller dell'esecuzione animata; a sinistra la colonna in cui verranno visualizzati i risultati degli algoritmi in forma di dati di testo. Prima di entrare nello specifico delle funzioni della finestra principale, utilizzata in fase di risoluzione, si considera come un grafo può essere creato. Sulla barra in alto l'unica funzione accessibile è la funzione **Graph**: da Graph si può caricare un file già costruito e salvato oppure procedere alla costruzione di un nuovo grafo tramite **Open Graph Editor**. Una finestra separata mette a disposizione le seguenti funzioni: **Load Background** permette il caricamento di un'immagine da porre come sfondo; **Load Graph** carica un grafo già salvato sullo sfondo corrente scelto; **Save Graph** permette il salvataggio del file creato, pronto così per l'esecuzione degli algoritmi.

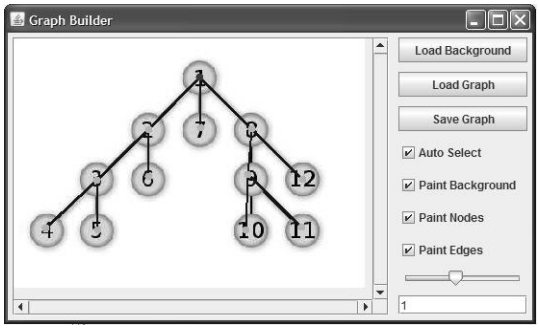


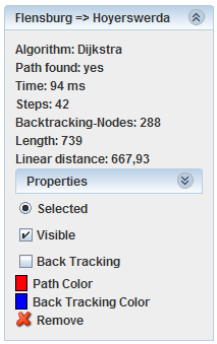
Figura 41: Finestra Graph Builder per la costruzione di un grafo *da Open Graph Editor*

La costruzione diretta del grafo si effettua tramite un semplice click per la creazione di un nodo, nel punto desiderato sullo sfondo. Un arco viene creato a partire dal nodo selezionato tramite un altro click, che produrrà la creazione vicendevole dell'arco e del suo nodo d'arrivo. Sono disponibili le opzioni da inserire: **Auto Select**, che mantiene selezionato l'ultimo nodo creato di volta in volta, e **Paint Background**, **Nodes** o **Edges**, per visualizzare o meno

sfondo, nodi o archi. Lo **Zoom** è regolabile tramite un comando scorrevole in una barra, e uno spazio bianco per il testo consente di inserite la denominazione dei nodi una volta selezionati.

15.2. Algoritmo risolutivo

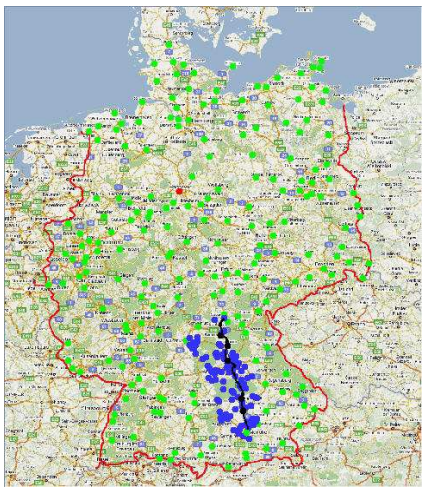
Una volta caricato un grafo, verranno visualizzati unicamente lo sfondo e i nodi a cui è stato assegnato un nome, che saranno anche gli unici a poter essere selezionati. Assieme al layout grafico sono disponibili le **Graph Statistics** quali numero di archi, nodi, nodi con nome e dimensione dell'immagine del grafo. Sulla colonna di destra è possibile quindi selezionare l'algoritmo di Dijkstra da una lista di risolutori, impostare il nodo di partenza e di arrivo, selezionando direttamente dal grafo o scrivendo il loro nome, e scegliere **Find Route**. Sotto lo Zoom, una sottofinestra mostra le caratteristiche dell'algoritmo scelto e consente di visualizzare messaggi di testo per l'utente in caso di svolgimento errato di operazioni.

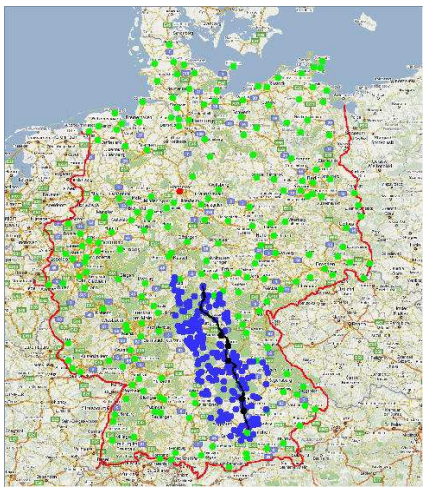


Se il grafo consente un cammino tra i nodi segnati, viene trovato un cammino minimo, e graficamente individuato. Si nota come i nodi senza nome sono attraversati ma non visualizzati. A questo punto assieme al risultato grafico, sulla colonna di sinistra un sottomenu elenca le caratteristiche dell'esecuzione e del cammino trovato: tempo d'esecuzione, nodi facenti parte del cammino minimo e nodi attraversati nel corso della procedura risolutiva. Inoltre PathFinder offre alcune opzioni di layout per rendere o meno visibile il cammino o i nodi attraversati (Backtracking-Nodes) e per la loro colorazione. Visualizzare contemporaneamente più cammini diversi è consentito e i sottomenu relativi restano disponibili.
Come già anticipato, la sezione in basso della finestra principale consente di rivedere il procedimento di calcolo seguito dall'algoritmo tramite un'animazione (**Replay**): se ne possono regolare velocità, pause e istante da visualizzare su una barra scorrevole. Agendo direttamente sulla rappresentazione, risulta uno strumento efficace e dal forte impatto visivo.

15.3. Esempio di calcolo

Ecco un esempio di calcolo del cammino minimo tra due nodi in un grafo. Il grafo è applicato su di una cartina della Germania: città di interesse sono contrassegnate da nodi color verde il cui nome si può leggere selezionandoli; le strade principali rappresentate sono riprodotte in una rete non visibile di archi e nodi. Dopo aver scelto di trovate il cammino minimo da München ad Hannover si innesca il calcolo tramite Find Route. Ottenuto il risultato (in cui sono resi evidenti in blu i nodi di Back Tracking attraversati) si utilizza la sezione Replay per visualizzare l'animazione della procedura di calcolo, di cui si presentano alcune fasi.





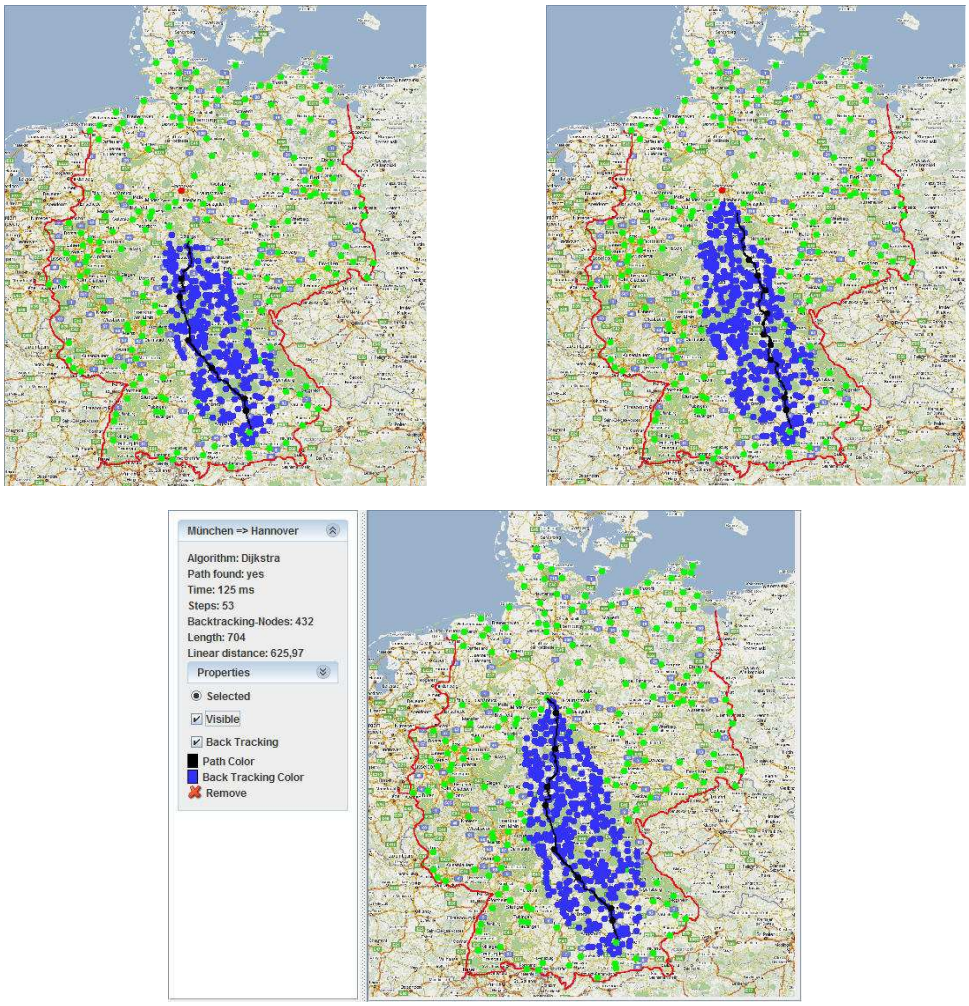


Figura 42: Risoluzione di problema di cammino minimo con nodi di Back Tracking visualizzati su sfondo di cartina della Germania

Nel sottomenu della soluzione sono indicati 125 ms per l'esecuzione, 53 passi eseguiti nel cammino (cioè i nodi dal nodo di partenza a quello d'arrivo) e 432 nodi attraversati durante l'algoritmo. La distanza lineare tra i due nodi è di 625,97, mentre la lunghezza totale del cammino minimo è 704.

16. Applets Java

Una *applet Java* è un'applicazione scritta in linguaggio Java che può essere inserita in pagine HTML e che può essere eseguita da un Web browser. Questo tipo di applicazioni sono utilizzate per aggiungere alla pagina Web effetti multimediali o di interattività non ottenibili altrimenti. La possibilità di manipolazione diretta e visualizzazione animata, aggiunte alla semplicità d'uso e la sveltezza d'esecuzione, rendono le *applet* un mezzo interessante nell'implementazione di algoritmi risolutivi nei problemi sui grafi.
Due esempi di *applets* ben costruiti e dalle funzionalità principali pari ad alcuni software scaricabili dalla rete sono presentati qui di seguito: le informazioni contenute nelle istruzioni d'uso sono puntuali e precise, i comandi intuitivi e le opzioni sufficientemente elaborate e varie. I due programmi si basano rispettivamente sugli algoritmi di Dijkstra e Kruskal: essi sono proposti per essere utili nell'acquisire familiarità con la teoria dei grafi, la costruzione in rappresentazione grafica punta in questa direzione, e come strumenti adatti all'apprendimento della procedura risolutiva degli algoritmi stessi.

I problemi trattati sono quindi di cammini e alberi minimi in grafi orientati o non orientati, su con si possono creare e risolvere esercizi.

16.1. Dijkstra's Shortest Path Algorithm

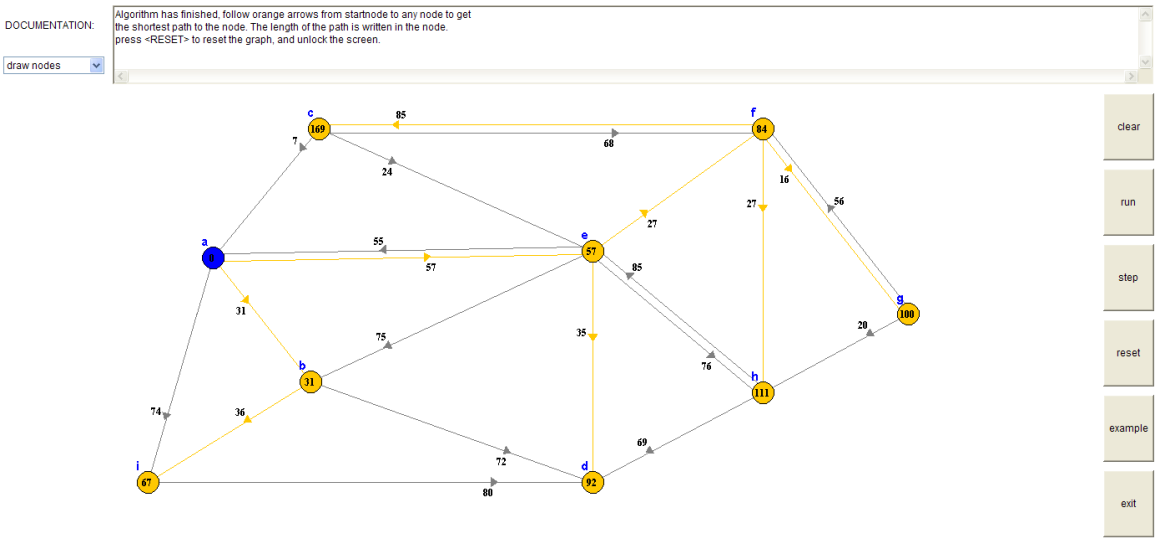


Figura 43: Visualizzazione di esempio svolto di problema di arborescenza minima su grafo pesato e orientato

Alla pagina Web relativa, la finestra che viene proposta in questa applet Java è semplificata più possibile per essere di facilmente comprensione per l'utilizzatore: nella parte centrale vi è lo spazio su cui si costruisce la rappresentazione grafica; in alto è presente la lista di tutte le funzionalità disponibili, con la descrizione tramite messaggio di testo di come utilizzarle e come procedere alla risoluzione nella sezione DOCUMENTATION; sulla destra i comandi principali che riguardano la risoluzione dell'algoritmo sono messi in evidenza.
L'algoritmo implementato è l'algoritmo di Dijkstra, utile nel problema della ricerca di tutti gli alberi di peso minimo a partire da un nodo scelto come radice.
Selezionando le funzionalità offerte nel menu a scomparsa in alto a destra, nella sezione dei messaggi di testo DOCUMENTATION vengono visualizzate le operazioni da compiere per poterle applicare. Le modalità di utilizzo per i comandi sono le seguenti:

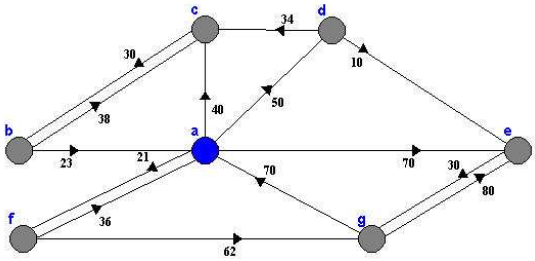
- **draw nodes**: con un click sulla sezione centrale viene creato un nuovo nodo. Il numero massimo di nodi è pari a 20;
- **remove nodes**: si rimuove un nodo con un click tenendo premuto il tasto "Ctrl";
- **move nodes**: per riposizionare un nodo si preme "Shift" e lo si trascina;
- **the startnode**: il nodo di partenza dell'algoritmo è impostato come il primo creato ed è contraddistinto da colore blu dagli altri in grigio. Per modificarlo, premere "Ctrl" e cliccando trascinare il cursore dal nodo di partenza corrente al nuovo nodo di partenza scelto;
- **draw arrows**: un arco orientato si crea con un click dal nodo di partenza, per trascinamento fino al nodo di arrivo. Archi doppi sono permessi, anche di peso differente, non sono permessi acicli;
- **change weights**: il peso dell'arco può essere modificato facendo scorrere il simbolo della freccia lungo l'arco. Sono permessi pesi di valore da 1 a 100;
- **remove arrows**: per rimuovere un arco si cambia il valore del suo peso a 0;

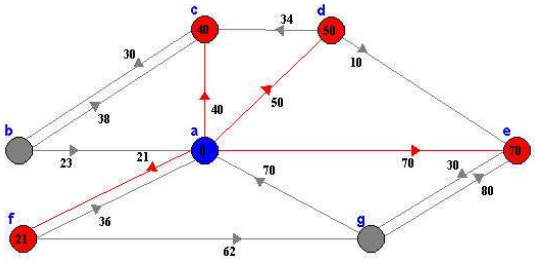
- **clear / reset**: con *clear* si rimuove la rappresentazione grafica costruita, con *reset* solo il risultato dell'esecuzione dell'algoritmo;
- **run algorithm**: si avvia l'esecuzione dell'algoritmo, la cui procedura è visibile passo dopo passo a schermo: i vari steps sono separati da 1 secondo nell'animazione, che si sviluppa grazie alla diversa colorazione di nodi e archi;
- **step through**: permette di seguire l'esecuzione per steps controllati dall'utilizzatore;
- **example**: fornisce all'utente un grafo precostruito su cui lavorare;
- **exit**: produce l'uscita dall'*applet*;
- **all items**: nella sezione DOCUMENTATION permette di visualizzare tutte le istruzioni dei comandi illustrati finora.

Il layout grafico non è modificabile, i nodi sono rappresentati in grigio, tranne il nodo di partenza blu, e gli archi rettilinei e semplici frecce: è assegnato e visualizzato un nome ad ogni nodo, lettere dell'alfabeto, e viene visualizzato il peso a fianco di ogni arco. Viene data logicamente più importanza ai comandi di costruzione ed esecuzione, alla chiarezza e semplicità grafica piuttosto che a funzioni di manipolazione non strettamente fondamentali.
L'opzione salvataggio non è disponibile, per la costituzione strutturale dello strumento applet usato: si opera con un mezzo d'interazione, d'esecuzione su di una pagina Web.
Il *source code* utilizzato per la creazione dell'applet è reso visibile all'utente a fondo pagina.

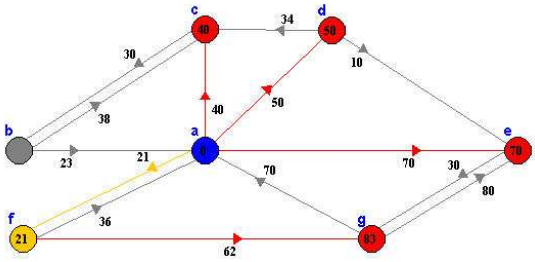
16.1.1. Esempio di calcolo

Di seguito è svolto un esempio d'esecuzione dell'algoritmo di Dijkstra su un grafo pesato e orientato: ecco la visualizzazione della sequenza delle varie fasi di avanzamento per trovare tutti i cammini minimi ai nodi dal nodo radice *a*, segnato in blu. La visualizzazione *step by step* è possibile grazie al comando *step through* ed è accompagnata dalla spiegazione delle operazioni eseguite, riportate nella sezione DOCUMENTATION.

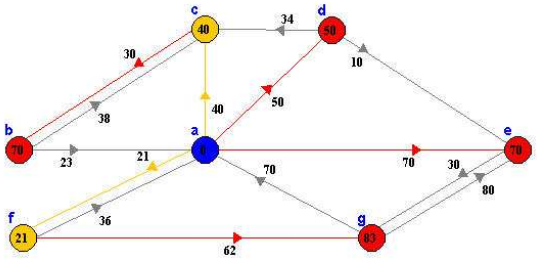




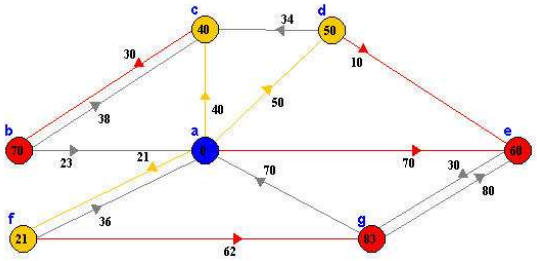
Al primo step sono evidenziati in rosso i nodi raggiungibili dal nodo di partenza *a*: in ogni nodo appare il peso totale del cammino dalla radice *a*. Qui il più breve ha peso totale 21 e porta al nodo *f* .



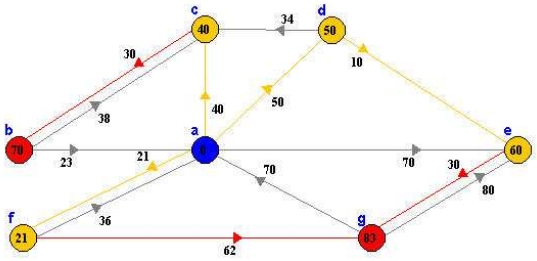
Il cammino minimo che porta ad *f* quindi è segnato in arancione e viene aggiunto l'arco che da *f* porta a *g*: in *g* si vede il peso totale del cammino che da *a* passa per *f* e arriva a *g* pari a 83. Ora il cammino di peso minore ha valore 40 e porta in *c*, che verrà evidenziato in arancione.



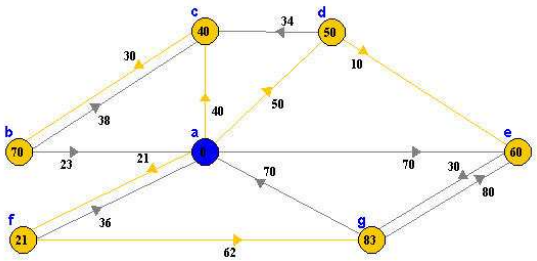
Dopo aver evidenziato *c* si aggiunge l'arco che da *c* raggiunge *b*, che aggiorna il valore del cammino da *a* a *b*: 70. Il cammino minore tra quelli presenti ora ha valore 50 ed è quello che raggiunge *d*.



Il nodo *d* e l'arco (*a*,*d*) sono aggiunti ai cammini definitivi trovati e viene aggiunto l'arco che da *d* porta ad *e*: si nota come il percorso da *a* ad *e* tramite *d* sia di peso totale 60, inferiore a quanto non fosse seguendo l'arco da *a* ad *e* (peso 70). Si sostituisce il valore in *e*, quindi se ne evidenzia il cammino tramite *d*.



Aggiungo ora l'arco (*e*,*g*) e valuto se il cammino da *a* passante per *d* ed *e* fino a *g* ha peso minore del cammino che da *a* raggiunge *g* attraverso *f*: 90>83, quindi si mantiene il valore precedente. Il cammino di peso minimo è comunque quello che porta a *b*, tramite *c*, di peso totale 70.



Infine, dopo aver evidenziato in arancione il cammino che porta a *b* tramite *c*, non resta che selezionare anche il cammino che da *a* porta a *g* tramite *f*, di valore minimo 83.
Ora si hanno tutti i cammini di peso minimo che partono dal nodo radice *a* e giungono a tutti gli altri nodi, coi relativi valori di peso.

16.2. Kruskal's Minimum Spanning Tree Algorithm

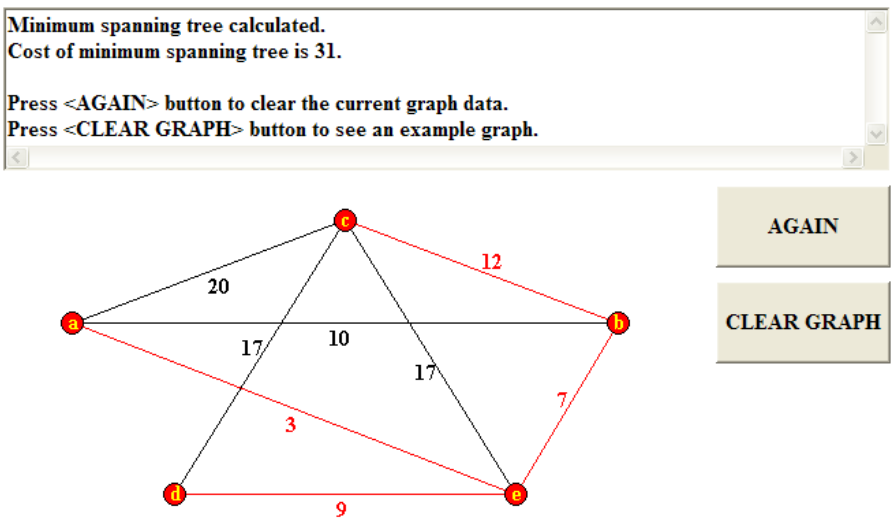


Figura 44: Visualizzazione di esempio svolto di problema di albero minimo su grafo pesato non orientato

Questa applet Java permette di cimentarsi nello svolgimento di semplici problemi di albero minimo su grafi pesati non orientati. Come per l'applet appena descritto, la dimensione massima ammissibile è di 20 nodi, quindi si parla di grafi semplici adatti ad uno studio didattico dell'algoritmo risolutivo di Kruskal.
La struttura della pagina di lavoro è composta da uno spazio centrale in cui avviene la rappresentazione grafica del grafo, una sezione di testo in alto in cui sono presenti le istruzioni d'utilizzo delle funzioni e i messaggi per l'utente, due tasti in evidenza a destra le cui funzioni cambiano a seconda dello stadio di lavoro a cui si giunge.
All'inizio l'applet presenta due opzioni sui comandi in rilievo a destra: **NEW GRAPH** e **EXAMPLE** consentono di decidere se voler costruire il grafo manualmente o caricare un grafo di esempio predefinito su cui lavorare. Una volta scelta l'opzione NEW GRAPH nella sezione di testo in alto appaiono le istruzioni per la costruzione grafica: l'aggiunta di un nodo si effettua tramite un click sullo spazio vuoto centrale, un arco invece cliccando e trascinando il cursore dal nodo di partenza fino al nodo di arrivo. La disposizione degli oggetti è libera, anche se non è permesso l'inserimento in caso di eccessiva vicinanza con oggetti già esistenti. Come nome del nodo è assegnata automaticamente una lettera dell'alfabeto, mentre per un arco è visibile il suo peso: di default è posto valore 5 di peso, ma cliccando sul numero è possibile modificarlo da 0 a 99 tramite una tastiera numerata che viene visualizzata. Non sono ammessi pesi negativi, come altrettanto archi multipli o acicli. Per riposizionare un nodo è sufficiente trascinarlo dopo aver contemporaneamente premuto il tasto "Shift", per eliminarlo invece si seleziona "Ctrl" e lo si clicca. Non è permesso con un comando cancellare un arco.

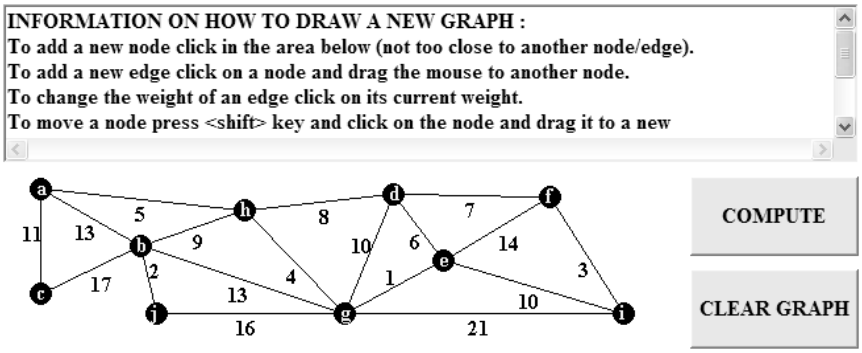
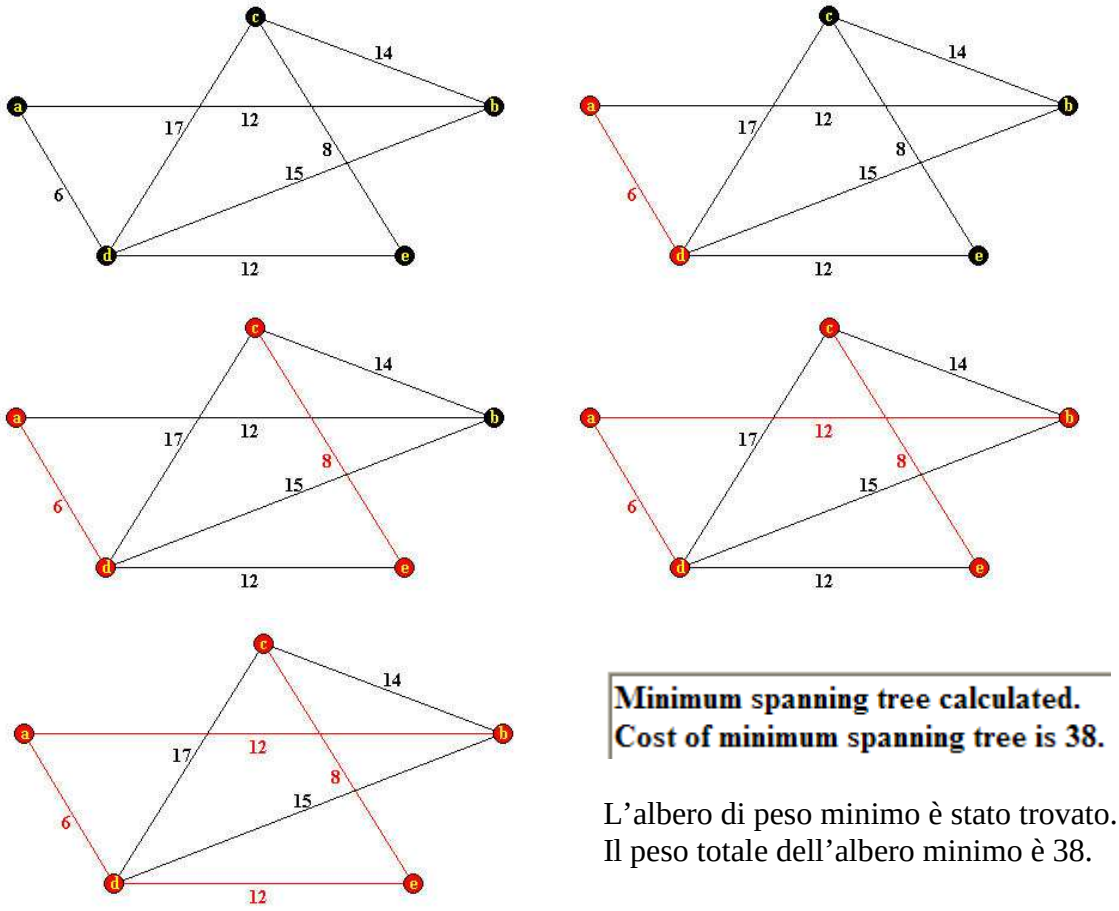


Figura 45: Visualizzazione della finestra di testo con istruzioni d'uso e tasti in rilievo dopo la scelta dell'opzione NEW GRAPH

Nel momento in cui si è passati alla modalità NEW GRAPH i tasti in rilievo cambiano in **COMPUTE** E **CLEAR GRAPH**: questi permettono di attivare l'esecuzione dell'algoritmo di Kruskal o di cancellare completamente il grafo fin lì costruito. Una volta scelto COMPUTE nella sezione di testo appare la spiegazione dell'operazione che si sta per eseguire, mentre il tasto COMPUTE diviene **NEXT STEP**, per il controllo dell'avanzamento *step by step* della procedura. L'animazione avviene tramite la colorazione degli archi da inserire nell'albero minimo soluzione, e si conclude con il messaggio di testo di trovato MST, con annesso peso totale. Nel caso il grafo non fosse connesso l'applet comunica che non è possibile trovare il MST.
Le funzionalità si esauriscono ancora un volta a quelle basilari descritte, non comprendendo opzioni di salvataggio, modifica di layout e rappresentazione, risoluzione di problemi differenti etc.

16.2.1. Esempio di calcolo

Si presenta un breve esempio di risoluzione del problema di albero minimo in un grafo pesato non orientato, tramite Kruskal's Minimum Spanning Tree Algorithm:



Minimum spanning tree calculated.
Cost of minimum spanning tree is 38.

L'albero di peso minimo è stato trovato.
Il peso totale dell'albero minimo è 38.

17. Conclusioni

Al termine di questa analisi di software *open source* basati sulla teoria dei grafi sono necessarie alcune considerazioni: emerge come, per la fase di ricerca, si debba porre a priori una linea che separi in due filoni gli elementi di interesse. In una prima classe si collocano i software che presentano una struttura strettamente attinente alla teoria dei grafi e ai problemi relativi di ottimizzazione di alberi e cammini minimi, caratterizzati da un'applicabilità a più ampio raggio. L'altra classe si distingue per lo sviluppo di software fortemente radicati nella realtà esecutiva in cui vengono impiegati, quindi ampiamente differenziati e specifici, forniti di apposita nomenclatura, funzionalità e layout. Secondo la scelta presa in origine d'analisi, si può osservare come la diffusione di software con caratteristiche attinenti alla teoria e alla rappresentazione di grafi sia di buon livello: effettuata un'indagine sul Web di profondità crescente, a seconda delle esigenze di propedeuticità, varietà di funzioni, layout, conformità agli scopi richiesti, si può giungere a programmi con buone qualità di visualizzazione ed esecuzione, nonché di semplicità di installazione.
Le fonti da cui reperire software ed informazioni sono molteplici: si passa dalle *libraries*, a siti specializzati nella diffusione di programmi, a siti propri di aziende operanti nel mercato. Per la quasi totalità dei casi, grazie alle licenze come la *GNU Lesser General Public License*, la fruibilità del *free download* è incoraggiata dagli stessi creatori, così come la reperibilità di informazioni sull'utilizzo delle loro funzioni. Lo scaricamento dei software trovati risulta inoltre facilitato dal fatto che un gran numero dei programmi si appoggi a linguaggi di programmazione diffusi nella maggioranza dei computers o allo stesso modo reperibili in versioni gratuite dal Web (come Java o C).

Molti software, ricavati nella fase superficiale della ricerca effettuata, dimostrano funzionalità limitate, spesso con algoritmi specifici per uno soltanto dei due problemi considerati: è il caso di alcuni degli ultimi programmi presentati in analisi (i.e. MST algorithms o MST Java V1.5), il cui primo scopo è offrire la possibilità di comprendere lo svolgimento dell'algoritmo risolutivo e lavorare con esso. Osservate le caratteristiche di *demonstration* svariati altri non sono stati inseriti nella trattazione.
Nonostante questo primo livello di strumenti limitati si individuano anche software dalle potenzialità molto elevate, sia per dimensione di grafi consentite che per possibilità di manipolazione grafica, modalità di esecuzione e molteplicità di opzioni: ovviamente i primi programmi descritti soddisfano esigenze più elevate di efficienza computazionale, con una gamma di funzioni ed impostazioni esauriente e potenzialità superiori alla sola esecuzione di algoritmi risolutivi per il MST o lo SP. Di questi le informazioni di progetto e i manuali d'utilizzo sono solitamente accessibili via Web, nel sito d'origine o nelle sezioni di *documentation* degli files scaricati.
La rappresentazione per via grafica, comune per altro a tutti i casi considerati, funge da buon compromesso tra la manipolazione diretta ed intuitiva e la possibilità di ottenere chiarezza nella costruzione dei grafi e nell'esecuzione algoritmica: gli algoritmi più diffusi ed efficienti sono ben implementati e spesso, come si è potuto constatare, si trovano ottime animazioni *step by step* utili in fase di comprensione e verifica del procedimento di calcolo.
In definitiva le caratteristiche sopra descritte pongono i software *open source* in un orizzonte di efficienza computazionale e condivisione libera per l'evoluzione funzionale, dimostrando delle potenzialità applicative sia per lo studio ad ampio raggio dei problemi che per lo sviluppo su specifici settori di utilizzo.

Bibliografia

[1] Matteo Fischetti, Lezioni di Ricerca Operativa, 1995, Edizioni Libreria Progetto, via Marzolo 28, Padova.

[2] ICS 161: Design and Analysis of Algorithms Lecture notes for February 6, 1996. Sito <http://www.ics.uci.edu/~eppstein/161/960206.html>. Novembre 2010.

[3] Facchinei F., Grafi: nozioni fondamentali, 2000. Sito <www.dis.uniroma1.it/~facchinei/cap6.pdf>. Novembre 2010.

[4] Facchinei F., Cammini minimi, 2000. Sito <www.dis.uniroma1.it/~facchinei/cap7.pdf>. Novembre 2010.

[5] Sassano A., Cammini minimi, 2005. Sito <www.dis.uniroma1.it/~sassano/Dispense/POR/cammini.pdf>. Novembre 2010.

[6] Oriolo G., Problema di cammino minimo su grafi, 2000. Sito <www.dis.uniroma1.it/~or/trasporti/grafi.pdf>. Novembre 2010.

[7] Quattrocchi G., Breve introduzione alla teoria dei grafi, 2010. Sito <http://www.dmi.unict.it/~gquattro/quattrocchigae/Didattica/formazione_discreta_2/teoria%20dei%20grafi%20no.pdf>. Novembre 2010.

[8] Casolo C., Introduzione alla teoria dei grafi, 2008. Sito <http://www.liceodavincitv.it/attivi/lauree/08-09/appuntigrafi.pdf>. Novembre 2010.

[9] Lodi A., Teoria dei Grafi, 2001. Sito <www.dmi.unict.it/nicosia/lectures/programmazione-scientifica/Grafi1.pdf>. Novembre 2010.

[10] Gallizia G., Appunti di Algoritmi e Strutture Dati, 2004. Sito <appunti.ccni.it/primo_anno/Algoritmi-StruttureDati/lezioni/grafi.pdf>. Novembre 2010.

[11] Free Software Foundation. Sito <http://www.fsf.org/>. Novembre 2010.

[12] GNU Operating System, GNU Lesser General Public License. Sito <http://www.gnu.org/>. Novembre 2010.

[13] The GTK+ Project. Sito <http://www.gtk.org/>. Novembre 2010.

[14] Sourceforge. Sito <http://sourceforge.net/>. Novembre 2010.

[15] Softpedia. Sito <http://www.softpedia.com/>. Novembre 2010.

[16] Brothersoft. Sito <http://www.brothersoft.com/>. Novembre 2010.

[17] Greedy Algorithms, Kruskal's Minimum Spanning Tree Algorithm applet. Sito <http://www.cs.man.ac.uk/~graham/cs2022/greedy/index.html>. Dicembre 2010.

[18] Liverani M., Un problema di cammino minimo ed una sua applicazione al World Wide Web, 1999. Sito <www.mat.uniroma3.it/users/liverani/doc/tesi_fabrianesi_sintesi.pdf>. Dicembre 2010.

[19] Kondo N., Masina A.P., Path Finder based on shortest path. Sito <http://samsajournal.org/samsa/abstracts/Masina.pdf>. Dicembre 2010.

[20] Rodrigues F., Applet Java for Kruskal's algorithm and Prim's algorithm, 2008. Sito <http://www.iut-info.univ-lille1.fr/~lebegue/ri/2007-2008/Timisoara/RODRIGUES%20Rapport.pdfwww.phys.psu.edu/~ralbert/phys597_09/c05_graph_alg.pdf>. Dicembre 2010.

[21] Sedgewick R., Minimum Spanning Tree, 2005. Sito <http://www.cs.princeton.edu/courses/archive/fall05/cos226/lectures/mst.pdf>. Dicembre 2010.

[22] Wikipedia, Drichel A., Prim's algorithm and Kruskal's algorithm images, 2008. Sito <http://en.wikipedia.org/ wiki/Main_Page>. Dicembre 2010.

[23] Graph Magics. Sito <http://www.graph-magics.com/>. Dicembre 2010.

[24] Sourceforge, Peklo download. Sito <http://sourceforge.net/projects/peklo/>. Dicembre 2010.
[25] GOBLIN: A Graph Object Library for Network Programming Problems. Sito <http://www.math.uni-augsburg.de/~fremuth/goblin.html>. Dicembre 2010.
[26] GraphThing. Sito <http://graph.seul.org/>. Dicembre 2010.
[27] JGraphEd. Sito <http://www.jharris.ca/JGraphEd/>. Dicembre 2010.
[28] Sourceforge, MST download. Sito <http://sourceforge.net/projects/mst-comparison/>. Dicembre 2010.
[29] Softpedia, MST 1.5 download. Sito <http://mac.softpedia.com/get/Utilities/MST.shtml>. Dicembre 2010.
[30] Uweschmidt, DijkstraVis. Sito <http://www.uweschmidt.org/projects/dijkstravis>. Dicembre 2010.
[31] Sourceforge, PathFinder download. Sito <http://sourceforge.net/projects/jpathfinder/>. Dicembre 2010.
[32] Dijkstra's Shortest Path Algorithm. Sito <http://www.dgp.toronto.edu/people/JamesStewart/270/9798s/Laffra/DijkstraApplet.html>. Dicembre 2010.
[33] Pathfinder Library for Processing. Sito <http://robotacid.com/PBeta/AILibrary/Pathfinder/index.html>. Dicembre 2010.
[34] YWorks. Sito <http://www.yworks.com/en/index.html>. Dicembre 2010.
[35] VisAnt. Sito <http://visant.bu.edu/>. Dicembre 2010.

Appendice

1. Pseudocodice Algoritmo Shortest Path Dijkstra:

Nelle specifiche del software Graph Magics è accessibile lo pseudocodice dell'algoritmo di Dijkstra utilizzato nell'implementazione e riportato di seguito:

```
# N – numero di nodi
# peso(i,j) – peso dell'arco dal nodo i al nodo j; posto ad infinito se tale arco non esiste
# dist(i) – cammino minimo dal nodo di partenza i
# parent(i) – nodo che precede i in un cammino minimo, usato per ricostruire il cammino

# inizializzare tutti i valori
For i = 1 to N
   dist(i) = infinity
   processato(i) = false
   parent(i) = null
End For
dist(source) = 0

While (non tutti i nodi sono stati processati)
   If (le distanze di tutti i nodi non ancora processati sono uguali ad infinito) Then
      nessun cammino from source to destination node esiste
      Exit
   End If

   Per i nodo non ancora processato con dist(i) che è la minore tra tutte quelle dei nodi non processati.
   If i = destination Then Exit While    # cammino minimo da nodo di  partenza a destinazione trovato
   Set processato(i) = true

   Per ogni nodo j non ancora processato    # i.e. per i nodi  per cui è  processato(j) = false
      If dist(i) + peso(i,j) < dist(j) Then
         # un cammino minore dal nodo di partenza a  j è stato trovato; aggiornare il cammino
         dist(j) = dist(i) + peso(i,j)
         parent(j) = i
      End If
   End For
End While

# il valore del cammino minimo è così  dist(destination)

# la ricostruzione del cammino è data di seguito
Set i = destination
While i != source
   Appendi i all'inizio del cammino corrente costruito
   i = parent(i)
End While
Appendi source all'inizio del cammino
```

2. Pseudocodice algoritmo All Shortest Path di Floyd-Warshall:

Nelle specifiche di Graph Magics è reso disponibile anche lo pseudocodice per l'implementazione dell'algoritmo di Floyd-Marshall:

```
# N – numero di nodi
# peso(i,j) – peso dell'arco dal nodo i a j; uguale ad infinito se tale nodo non esiste

For i = 1 to N
   For j = 1 to N
      A(i,j) = peso(i,j)

For k=1 to N       # k è il nodo intermedio
   For i=1 to N
      For j=1 to N
         # controllo se il cammino da i a j che passa attraverso k è più breve di quello già trovato
         If A(i,k) + A(k,j) < A(i,j) Then A(i,j) = A(i,k) + A(k,j)
```

3. Pseudocodice algoritmo Shortest path a una radice di Bellman-Ford:

Ecco l'esempio generico di pseudocodice, con pesi negativi ammessi e controllo di cicli non negativi:

```
# V – numero di nodi
# A – numero di archi
# dist(v) – cammino minimo dal nodo di partenza v
# parent(v) – nodo che precede v in un cammino minimo
# peso(u,v) – peso dell'arco dal nodo u al nodo v; posto ad infinito se tale arco non esiste

# inizializzare il grafo
For each vertice v in V
  If v is source Then
    dist(v) = 0
  Else
    dist(v) = infinity
    parent(v) = null

# relax gli archi ripetutamente
  For i from 1 to size(V) - 1
     For each arco (u,v) in A    #  (u,v) è l'arco da u a v
        u = source(u,v)
        v = destination(u,v)
        If  dist(v) + peso(u,v) < dist(v)
           dist(v) = dist(u) + peso(u,v)
           parent(v) = u

# controllo su cicli negativi
For each arco (u,v) in A
  u = source(v)
  v = destination(u,v)
  If dist(u) + peso(u,v) < dist(v)
    Error "Il grafo contiene un ciclo negativo"
```