UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

Facoltà di Ingegneria

Tesi di Laurea di Primo Livello in Ingegneria Gestionale

Problemi di cardinalità massima in grafi bipartiti

Relatore : Romanin Jacur Giorgio

Laureando : Kristiansen Maria Borup

Anno Accademico : 2011/2012

INDICE

SOMMARIO

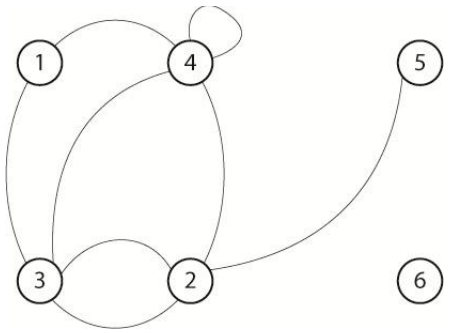
Questa tesi affronta lo studio dei problemi di matching di cardinalità massima in grafi bipartiti. Il lavoro si articola in una prima parte riguardante gli aspetti più generali e teorici dell'argomento trattato, contenente le definizioni e i teoremi fondamentali necessari per comprendere la logica che sta alla base del procedimento risolutivo di un problema di matching in un grafo bipartito. Successivamente ne viene presentato e risolto in modo analitico un esempio, utilizzando l'algoritmo per la costruzione di un matching di cardinalità massima, analizzando passo passo il procedere della risoluzione. L'ultima parte dell'elaborato, nonché il vero nucleo della tesi, presenta e analizza una procedura risolutiva per i problemi di matching in questione realizzata in linguaggio VBA (Visual Basic For Applications) e creata con "Visual Basic Editor" applicato a Microsoft Excel. Dapprima ne viene illustrata la struttura e successivamente ne viene analizzato il codice ad "alto livello", applicandola al problema prima risolto analiticamente e discutendone i risultati.

TEORIA DEI GRAFI

Alcune definizioni di grafo

Grafo
Un grafo, G=(V,E), è una coppia ordinata di insiemi: V è un insieme finito e non vuoto di elementi, mentre E è un insieme finito di coppie di elementi distinti di V.

Grafi non orientati
Un grafo non orientato è una coppia ordinata G=(V,E), dove V è un insieme finito ed E è una famiglia di coppie non ordinate di elementi di V.



Grafi, Vertici, Spigoli

L' insieme finito V è chiamato *vertice* di G e E è chiamato *spigoli* di G. L'insieme di spigoli è un sottoinsieme delle coppie di vertici: E⊆ $V \times V$. E siano n=|V| il numero di vertici di G e m=|E| il numero dei spigoli di G. Si indica tipicamente con V={1, ….., n} i vertici di un grafo e gli spigoli con {i,j}

Se in un grafo compare lo stesso spigolo due volte si parla di spigoli *multipli o paralleli*, cioè se due spigoli hanno gli stessi estremi (i,j)=(j,i). I grafi senza spigoli multipli si dicono *semplici*. Si dice che il grafo ha un *cappio* se esiste un lato (i,i), che in alcuni casi non è consentito.

Due vertici sono detti tra loro *adiacenti* se esiste un spigolo che li collega. Due spigoli che incidono sullo stesso vertice si dicono *adiacenti*.

Il numero di spigoli incidenti in un vertice *v* si dice *grado* del vertici, e viene indicato con $\delta(v)$. Un vertice *v* è *isolato* se non è adiacente a nessun altro vertice, cioè se $\delta(v) = 0$. Oppure si dice che un vertice è *pendente* se su di esso incide un solo spigolo, cioè $\delta(v) = 1$. La somma dei gradi di tutti i vertici è sempre un numero pari dato che ogni spigolo incide sempre in due vertici e quindi $\sum_{v \in V} \delta(v) = 2|E|$.

Il grafo G si dice completo se contiene tutti i possibili spigoli, cioè se E = $\{[i,j]: i,j \in V, i \leq j\}$.

Si dice che G è *pesato* sui vertici, se è specificata una funzione c: E→ $\mathbb{R}$.

Un *sottografo* di G=(V,E) è il grafo $G' = (V', E')$, dato che $V' \subseteq V e\ E' \subseteq E$. Se E' è l'insieme di tutti gli spigoli di G con entrambi gli estremi in V' si dice che G' è il sottografo di G *indotto* da V'. Un grafo parziale $G' = (V', E')$ di G=(V,E) è un sottografo con V'=V: ha gli stessi vertici di G ma solo un sottoinsieme di spigoli.

Cammini e cicli

Una sequenza di spigoli P=($(v_1,v_2),(v_2,v_3),\ldots,(v_k, v_{k+1})$) è un cammino (path) di G; due spigoli consecutivi nel cammino hanno in comune un estremo. Diremo che il cammino collega $v_1 \text{e } v_{k+1}$, visitando in sequenza i vertici intermedi $v_2, \ldots, v_k$. Se $v_1 = v_{k+1}$ il cammino può essere detto circuito,ciclo o cammino chiuso. Un cammino/ciclo si dice semplice se non usa mai due volte lo stesso spigolo. Un cammino è elementare se non visita mai due volte lo stesso vertice, a parte i vertici $v_1 = v_{k+1}$ nel caso del ciclo.

Il vertice *v* si dice connesso ad un altro vertice *w* se esiste un cammino che li collega. Per definizione, ogni vertice *v* è connesso a se stesso, anche se $[v, v] \notin E$.

Un grafo si dice connesso se i vertici *v* e *w* sono connessi per ogni v,w ∈ V. La relazione di connessione induce una partizione dei vertici del grafo, formato dalle sue componenti connesse, cioè dai sottoinsiemi massimali di vertici che inducono sottografi connessi.

Taglio e alberi

Un taglio di G è un insieme di lati del tipo $\delta(S) = \{[i,j] \in E: |S \cap \{i,j\}| = 1\}$, dove S è il sottoinsieme dei vertici che induce il taglio (il taglio contiene tutti i lati con un estremo in S e l'altro in V\S). È facile verificare che G è connesso se e solo se $\delta(S) \neq \emptyset\ \forall\ \emptyset \subset S \subset V$. Più in generale, il teorema di Menger afferma che, dati due vertici s e t, esistono k cammini disgiunti sui lati che li collegano se e solo se $|\delta(S)| \geq k\ \forall\ S \subset V\ tale\ che\ s \in S, t \notin S|$. Per ogni S⊆V, si usa la notazione $E(S) = \{[i,j] \in E: i \in S, j \in S\}$ per rappresentare l'insieme dei lati di G con entrambi gli estremi in S.

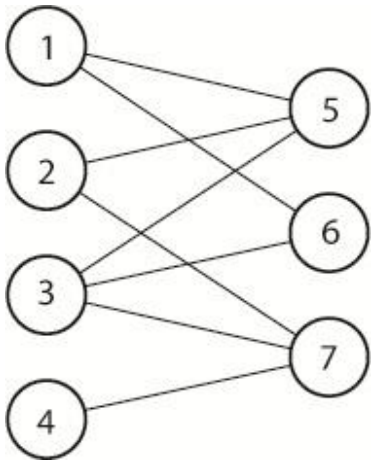
Un grafo parziale $G' = (V, E')$ di G si dice *foresta* se è aciclico, cioè se non contiene cicli. Una foresta è massimale se ogni lato in E\E' forma un ciclo con i lati in E'; in questo caso è facile vedere che G' e G hanno le stesse componenti connesse.

Una foresta massimale connessa, se esiste, è detta *spanning tree* o semplicemente *albero*. È facile verificare che ogni albero è formato da esattamente $|V| - 1$ spigoli. Inoltre, esiste un albero in G se e solo se G è connesso. In caso contrario ogni foresta massimale è formata da una collezione di "alberi parziali" (uno per ogni componente connessa).

L'albero di supporto è un sottografo di un grafo che sia un albero e abbia come vertici tutti i vertici del grafo originale. Il numero di alberi di supporto che si possono individuare in un grafo varia tra 1, se il grafo è un albero, a n, quando il grafo è completo.

Grafi bipartiti e planari

Un grafo si dice *bipartito* se esiste una partizione (V_1,V_2) di V tale che ogni lato $[i,j] \in E$ collega un vertice $i \in V_1$ ad un vertice $j \in V_2$. Si dimostra che G è bipartito se e solo se non contiene cicli dispari, cioè cicli formati da un numero dispari di lati.



Un grafo è *planare* se può essere disegnato sul piano senza che le linee che rappresentano gli spigoli si incrocino.

Un cammino si dice *euleriano* se copre una ed una sola volta tutti gli archi del grafo. Un grafo G è euleriano se esiste in G un ciclo euleriano.

Un cammino è *hamiltoniano* se visita una ed una sola volta tutti i vertici del grafo. Un grafo è *hamiltoniano* se contiene un ciclo hamiltoniano.

RAPPRESENTAZIONE DI GRAFI

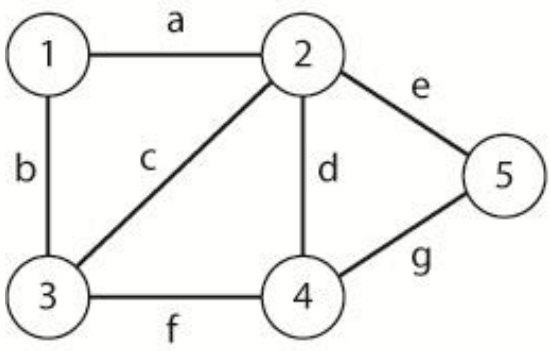
Un grafo può rappresentare qualcosa di fisico, come per esempio le strade, un relazione tra un lavoratore e il suo compito da svolgere o un albero genealogico. Esistono diversi modi per rappresentare un grafo: la rappresentazione grafica è quello più semplice da capire, ma per potere usare il calcolatore bisogna descrivere i grafi con opportune matrici.

Matrice di incidenza (matrice vertici-spigoli)

La matrice di incidenza D di un grafo non orientato G=(V,E) è la matrice |V|× |E| cioè una matrice $n \times m$ (Le righe corrispondono ai vertici e le colonne agli spigoli), definita così:

$$d_{ij} = \begin{cases} 1 \; se \; il \; j-esimo \; lato \; \grave{e} \; incidente \; nel \; vertice \; i \\ 0 \; altrimenti \end{cases}$$

La matrice di incidenza ha esattamente due elementi diversi da zero per ogni colonna.



	a	b	c	d	e	f	g
1	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0
3	0	1	1	0	0	1	0
4	0	0	0	1	0	1	1
5	0	0	0	0	1	0	1

Matrice di adiacenza(matrice vertici-vertici)

La matrice di adiacenza Q di un grafo semplice non orientato G=(V,E) è la matrice simmetrica $|V| \times |V|$, dove ogni riga e colonna rappresentano un nodo. La matrice si definisce cosi:

$$q_{ij} = \begin{cases} 1 \; se \; [i,j] \in E \\ 0 \; altrimenti \end{cases}$$

Nel caso che sto trattando , ossia grafi non orientati, la matrice è simmetrica rispetto alla diagonale (nei grafi orientati ciò non è vero).

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

Liste di adiacenza

Come si vede nel figura soprastante, nella matrice di adiacenza ci sono tanti posti vuoti; questo nel calcolatore corrisponde ad un grande spreco di memoria. Perciò si usa una scrittura più compatta chiamata *lista di adiacenza.* La lista di adiacenza è un vettore di liste che ha dimensione n e rappresenta i vertici del grafo; ogni lista associata rappresenta l'insieme dei vicini del nodo

Problemi di matching

Introduzione

Il termine accoppiamento o matching viene usato per indicare, in un grafo non orientato, un sottoinsieme di spigoli a due a due non adiacenti. I problemi di matching riguardano la determinazione di un accoppiamento che, soddisfacendo opportuni requisiti, ottimizza un determinato obiettivo (tipicamente: il massimo della cardinalità o il minimo dei costi).

Anche in questa circostanza, come nel caso dei problemi di flusso, gli esempi illustrativi che si è soliti dare hanno carattere prevalentemente didattico, in quanto l'utilizzo di metodologie tipiche del matching è assai ampio ed il matching appare spesso come sottoproblema nella soluzione di problemi più complessi di ottimizzazione combinatoria.

Quelli di matching sono infatti, almeno nelle loro formulazioni classiche qui considerate, problemi facili, in quanto risolvibili con algoritmi di complessità polinomiale. Ciò, evidentemente, costituisce un' incentivazione al loro uso, in particolare come punto di passaggio nella costituzione di soluzioni approssimate per problemi difficili.

Descriviamo ora alcuni esempi che possono servire come giustificazione delle terminologie adottate.

Esempio A: Siano dati n operai ed altrettanti compiti. Ogni operaio è in grado di svolgere solo alcuni dei compiti, quelli per i quali egli è idoneo. Si vorrebbe che ogni operaio fosse adibito ad uno e uno solo compito e che, simmetricamente, ogni compito fosse svolto da una ed una sola persona. Questo problema può essere formulato come problema di accoppiamento in un grafo bipartito nel modo seguente. Si consideri un grafo bipartito $G = (U \cup V, E)$ dove ciascun vertice di U rappresenta un operaio, ciascun vertice di V rappresenta un compito, mentre ciascun spigolo (u,v) in E rappresenta l'idoneità dell'operaio u a svolgere il compito v. Nel grafo appena introdotto si cerca se esiste un matching completo (tale cioè che ogni vertice sia incidente in uno spigolo del matching).

Esempio B: Una compagnia aerea dispone di una flotta di aeromobili e di un insieme di piloti. Per ogni aeromobile occorre decidere la coppia di piloti che lo possa guidare, in modo che i piloti di ciascun aereo siano in grado, per carattere, intelligenza, ecc.., di collaborare in maniera efficace tra di loro. Questo problema può essere formulato come problema di accoppiamento in un grafo non bipartito. In questo caso il modello è un grafo in cui i vertici corrispondono ai piloti, mentre esiste uno spigolo tra due vertici se e solo se i due piloti possono costituire un equipaggi (massima cardinalità) oppure quello di costituire un numero di equipaggi dato che comporti il minor costo possibile (supponendo che ad ogni coppia di piloti sia associato un costo).

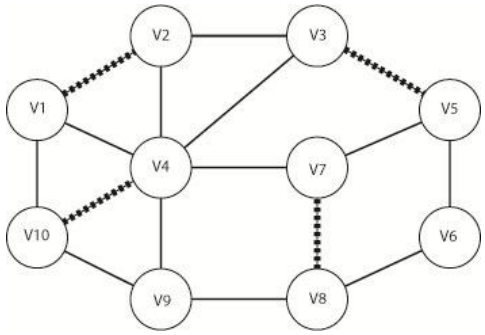
Gli esempi richiedono alcuni chiarimenti per quanto riguarda le definizioni, l'impostazione dei problemi e gli obiettivi che si possono perseguire.

Per quanto riguarda la terminologia, si ribadisce che la differenza tra gli esempi A e B è data dal fatto che il primo opera in un grafo bipartito, in quanto gli elementi descritti sono naturalmente suddivisi in due categorie, e gli spigoli, che esprimono "compatibilità" o legami di altro tipo tra soggetti di una categoria e dell'altra, collegano solo vertici di insiemi diversi; nel caso B gli elementi in gioco sono omogenei e il grafo rappresentativo è generale. Evidentemente ogni procedimento risolutivo per problemi di tipo B vale anche per quelli di tipo A, mentre non è detto che valga il viceversa.

Per quanto riguarda l'obiettivo da raggiungere, esso consiste, a secondo dei casi, nella determinazione di un insieme di spigoli di cardinalità massima oppure peso/costo minimo. Il grafo sottostante è in generale completo (cioè tale che ogni vertice è connesso ad ogni altro vertice) nei problemi di peso minimo, mentre è qualunque nei problemi di cardinalità.

Alcune definizioni

Un accoppiamento o matching M in un grafo non orientato G= (V,E), sia esso bipartito oppure no, è un insieme di spigoli a due e due non adiacenti. Un vertice v di G che sia estremo di uno spigolo in M si dice accoppiato, altrimenti si dice esposto. Un *camino alternante* di G è una catena elementare i cui spigoli appartengono alternativamente a M e a E\M. ad esempio, nella figura sotto i vertici v_7,v_5,v_3,v_4,v_{10} individuano, nell'ordine, un cammino alternante.



Un *cammino aumentante* è un cammino alternante in cui sono esposti il primo e l'ultimo vertice. È un camino aumentante, sempre riferito al figura sopra, il cammino individuato dai vertici v_6, v_8, v_7, v_4, v_{10}, v_9. In un cammino aumentante P vi è un numero dispari di spigoli e tutti gli spigoli in posizione dispari non stanno in M. Inserendo in M questi ultimi e togliendo gli spigoli di posto pari, si ottiene un nuovo insieme di spigoli M' che risulta ancora un accoppiamento e contiene uno spigolo in più:

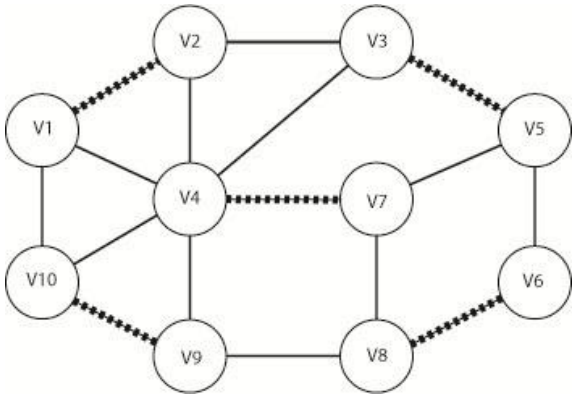
$$M' = M \backslash P \cup P \backslash M$$

Vale il seguente

Teorema di Berge: Un accoppiamento è di cardinalità massima se e solo se non esistono nel grafo cammini aumentanti rispetto a tale accoppiamento.

Dimostrazione: La necessita è ovvia. Se esiste un cammino aumentante l'accoppiamento non può essere di cardinalità massima. Per quanto riguarda la sufficienza, sia M un accoppiamento rispetto al quale non esistono cammini aumentanti e, per assurdo, supponiamo che esista un matching M* di cardinalità superiore a quella di M. Non è restrittivo supporre che |M*| = |M| +1. Si consideri $\Delta = M\backslash M^* \cup M^*\backslash M$ differenza simmetrica fra M e M*. Δ contiene quegli spigoli che stanno o in M o in M* ma non in entrambi: quindi contiene un numero dispari di spigoli. D'altra parte Δ contiene più spigoli di M* che non di M (esattamente uno in più). Si noti che ogni vertice del grafo originario ha grado al più 2 nel sottografo indotto da Δ. Di conseguenza le componenti connesse di Δ sono o dei cicli o dei cammini. I cicli in Δ devono contenere un numero pari di spigoli: tanti di M, altrettanti di M*. Essendo Δ di cardinalità dispari occorre quindi che esista almeno un cammino che contiene un numero dispari di spigoli e che inizia, e termina, con uno spigolo di M*. Questo cammino è evidentemente aumentante rispetto a M, contro l'ipotesi.

Nell'esempio precedente si vede che l'accoppiamento può essere migliorato proprio mediante il cammino aumentante da v_6 a v_9 menzionato sopra e il nuovo accoppiamento, rappresentato nel figura sottostante, è di cardinalità massima.

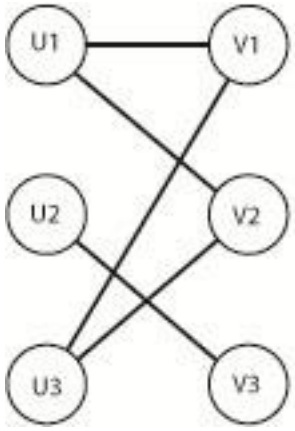


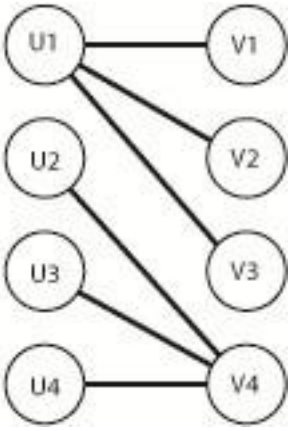
Che il matching rappresentato nel figura sopra sia di cardinalità massima, deriva immediatamente dal fatto che ogni vertice risulta accoppiato. Si osservi che un matching può essere di cardinalità massima anche se molti vertici rimangono esposti. Come caso estremo si consideri un grafo a stella S=(Vs,Es) dove Vs = {v_o, v_1, … ,v_n} ed Es = {(v_0 , v_j): j=1,2, …, n}
con almeno 3 vertici. In un grafo siffatto, un accoppiamento di cardinalità massima consiste evidentemente di un solo spigolo.

Problemi di cardinalità massima in grafi bipartiti

Sia G = (U ∪ V, E) un grafo bipartito, in generale non completo, in cui U abbia m vertici e V ne abbia n. Ricordiamo che un accoppiamento tra U e V è un qualsiasi insieme di spigoli di G a due a due non adiacenti.

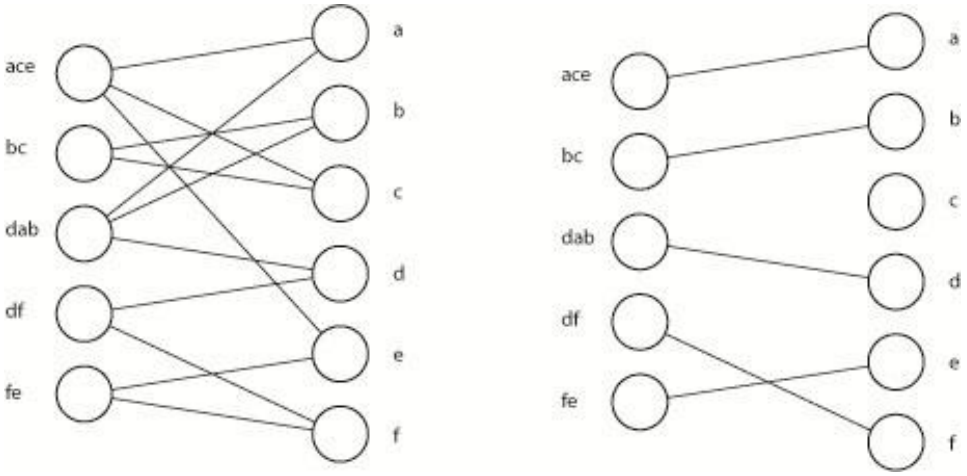
Diciamo accoppiamento completo di U in V un accoppiamento di G tale che ogni vertice di U sia incidente in uno spigolo dell'accoppiamento stesso. Ad esempio, nei grafi sotto, nel primo caso esiste un accoppiamento completo, nel secondo no.





Ad esempio, si considera la seguente applicazione: dato un insieme di parole, una cui sequenza, con possibili ripetizioni, deve essere trasmessa come messaggio, si vuole studiare la possibilità di

rappresentare ogni parola con una singola lettera in essa contenuta, in modo che le parole siano individuate univocamente. Si può far ricorso ad un grafo bipartito in cui i vertici del primo sottoinsieme sono le parole mentre i vertici del secondo sottoinsieme sono le lettere contenute nelle parole stesse. Nel grafo bipartito della figura che segue, il primo insieme di vertici è costituito dalle parole, mentre i vertici del secondo sottoinsieme sono le lettere. Ogni parola è collegata con



uno spigolo alle lettere che la compongono. Si vede che in questo caso esiste un accoppiamento completo fra l'insieme delle parole e quello delle lettere, come risulta dal grafo della figura a destra.

È evidente che per poter avere un accoppiamento completo di U in V è indispensabile che $|U| \leq |V|$. Una condizione necessaria e sufficiente è data dal teorema seguente. Per ogni sottoinsieme di vertici W, si indichi con G(W) l'insieme dei vertici che sono adiacenti ad almeno un vertice di W.

Teorema di Hall: Dato un grafo bipartito G=(U ∪ V,E) esiste un accoppiamento completo di U in V se e solo se per ogni insieme W di vertici di U si ha: |W|≤|G(W)|.

Dim: La condizione è evidentemente necessaria: se esiste un sottoinsieme W di vertici di U che è collegato ad un sottoinsieme di V di numerosità più piccola, non è possibile associare a tutti i vertici di W vertici distinti di V.

Per trovare che la condizione è sufficiente, procediamo per induzione su m=|U|. Se m=1 l'affermazione è ovviamente vera. Supponiamo che l'affermazione sia vera per tutti i grafi bipartiti in cui U ha cardinalità minore di un certo r. Si consideri ora un grafo in cui U ha cardinalità r. Si danno due casi alternativi:

A: Si supponga che per ogni W ⊂ U con |W|=k (dove k<r) si abbia |G(W)|≥k+1. In tal caso, se noi prendiamo un qualunque elemento di U e lo accoppiamo con un qualche elemento di V, la condizione originale rimane vera per l'insieme U' costituito dagli altri r-1 elementi di U. Questi r-1 vertici di U possono quindi essere accoppiati in virtù dell'ipotesi induttiva. Ciò conclude la dimostrazione in questo caso.

B: Si supponga che esista un sottoinsieme W di U con |W| = k e |G(W)|=k (dove k<r). Allora i vertici di W possono essere accoppiati con quelli di G(W) in virtù dell'ipotesi induttiva. Posto W' = U\W e h=|W'|=r-k, evidentemente W' deve essere tale che |G(W')\G(W)|≥h, altrimenti |G(U)|=|G(W) ∪ [G(W') \ G(W)]|≤ |G(W)|+|G(W') \ G(W)|<k+h=|U|, contro l'ipoteso del teorema. Ma allora, essendo h<r, per l'ipotesi induttiva, è possibile accoppiare tutti gli elementi di W' con elementi di G(W') \ G(W) e quindi in definitiva accoppiare tutti gli elementi di U con elementi di V.

Si noti che il procedimento per stabilire se in un grafo bipartito esiste un accoppiamento completo mediante la verifica della condizione suddetta non è efficiente. Infatti si dovrebbe ripetere la verifica della condizione un numero esponenziale di volte (una per ogni sottoinsieme di U). analogamente all'accoppiamento completo di U in V si può definire quello di V in U. Se poi U e V hanno la stessa cardinalità, si può porre il problema di individuare, se esiste un accoppiamento completo tra U e V. In questo caso si parla di matching perfetto tra U e V. Come conseguenza del teorema precedente si ha che una condizione sufficiente, anche se non necessaria, per l'esistenza di un accoppiamento completo tra U e V è data dal seguente

Corollario: Se ogni vertice in U è adiacente ad esattamente k (per qualche k≥1) vertici in V e viceversa, allora esiste un accoppiamento perfetto tra i due sottoinsiemi U e V.

Dimostrazione: In virtù del teorema precedente è sufficiente dimostrare che per ogni sottoinsieme W di vertici di U si ha: |W| ≤|G(W)|. Sia W un sottoinsieme generico di U. Indichiamo con F il sottoinsieme di spigoli incidente in W e con F' quello incidente in G(W). Per l'ipotesi del corollario, si ha |F|=k× |W| e |F'|=kx|G(W)|. Dal fatto che F⊆ F' consegue che kx|W| ≤ kx|G(W)| e quindi |W| ≤ |G(W)|.

Ad esempio, in una compagnia mista di ragazzi e ragazze, se ogni ragazza ha k amici e ogni ragazzo ha k amiche, è possibile che ad una festa ogni ragazza balli con un amico e che ogni ragazzo balli con una amica. Ciò non dipende dal valore k che rappresenta il numero di amici di ogni persona né dal numero totale delle persone (occorre però che le amicizia siano reciproche)

La ricerca di accoppiamenti completi rientra nella categoria dei problemi di cardinalità ottima: più in generale, si può cercare in un grafo bipartito non completo un accoppiamento di cardinalità massima. Il problema da risolvere può essere formulato come problema di PLI:

MAX $\sum_{(u_i,v_j)\in E} x_{ij}$

Soggetto a: $\sum_{j:(u_i,v_j)\in E} x_{ij} \leq 1 \quad \forall$ i = 1,2,…,m

$\sum_{i:(u_i,v_j)\in E} x_{ij} \leq 1 \quad \forall$ j = 1,2,…,

$x_{ij} \in \{0,1\} \quad \forall\, i,j : (u_i, v_j) \in E$

Tuttavia in pratica, si ricorre ai concetti e alle proprietà relative ai camini aumentanti descritte nel paragrafo precedente. La ricerca di tali cammini può essere effettuata mediante algoritmi di tipo polinomiale e risulta particolarmente agevole nel caso dei grafi bipartiti. In ogni caso si tratta, a partire da un vertice esposto, di verificare in maniera efficiente se esiste un cammino alternante che conduca ad un altro vertice esposto senza dover studiare singolarmente tutte le combinazioni di vertici esposti a due a due. Dato un accoppiamento M, inizialmente vuoto, la tecnica usata consiste nel costruire un albero aumentante, nel seguito indicato con T_r, avente come radice un vertice r esposto. La costruzione dell'albero procede sino a che si incontra un altro vertice esposto, il che consente di aumentare il matching, oppure si stabilisce che non esiste alcun cammino aumentante che parta dal vertice esposto r. La procedura si ripete per ogni vertice che risulti esposto. L'algoritmo si arresta quando tutti i vertici esposti sono stati esaminati (e nessuno di essi può essere accoppiato).

L'albero aumentante T_r è formato da più cammini alternanti aventi in comune la radice r: i cammini vengono accresciuti nel corso dell'algoritmo per aggregazione di spigoli e vertici. Essendo la radice un vertice esposto, è ovvio che gli spigoli man mano aggiunti sono, alternativamente, non

contenuti e contenuti nel matching corrente. I vertici via via inclusi sono classificati in due categorie: outer ed inner e, in corrispondenza, ricevono ciascuno una etichetta, O oppure I. Classificazione ed etichettatura sono legate al particolare albero che si sta costruendo ed al matching corrente. Sono outer la radice e tutte le vertice di posti dispari nei cammini alternanti: sono inner gli altri vertice. In altre parole, se si orientano tutti gli spigoli a partire dalla radice, i vertici inner sono vertici iniziali di spigoli del matching oppure vertici esposti, mentre i vertici outer (ad eccezione di r) sono vertici finali di spigoli del matching.

Descriviamo ora il procedimento più dettagliatamente: facciamo notare che i concetti di base che giustificano l'algoritmo sono comuni al caso bipartito e a quello non bipartito, anche se in quest'ultimo occorrerà tenere conto di particolari strutture che si possono presentare e obbligare a operare delle trasformazioni sul grafo che si studia.

Algoritmo (Matching di massima cardinalità in grafi bipartiti)

Passo 1. (Inizializzazione)

Sia M un matching iniziale qualunque(per esempio M = ∅). Ogni vertice v ∈ $U \cup V$ viene dichiarato non esplorato.

Passo 2. (Criterio di arresto)

Si cancellano tutte le etichette O e I eventualmente assegnate in precedenza. Ivertici etichettati con O vengono chiamati outer, invece i vertici etichettati con I vengono chiamati inner. Sono outer la radice e tutti i vertici di posti dispari nei cammini alternanti: sono inner gli altri vertici. Sia R l'insieme dei vertici esposti e non ancora esplorati. Se |R|≤ 1, allora l'algoritmo termina: il matching corrente è di cardinalità massima. Altrimenti sia r ∈ R. Si attribuisce etichetta O al vertice r. Si pone i=1, A_1={r},T_r=A_1 e si dichiara r esplorato.

Passo 3. (Inserimento di T_r di spigoli non in M)

Si pone i = i+1 (si noti che ora i è pari). Sia A_i l'insieme dei vertici adiacenti ai vertici di A_{i-1}che non siano stati inseriti in precedenza in T_r. Se A_i =∅ allora si va al passo 2. Altrimenti, per ogni vertici q di A_i, si inserisce in T_r, oltre a q, anche uno ed uno solo spigolo incidente in q ed in un vertice di A_{i-1}. Ogni vertice di A_i viene etichettato con I. Se in A_i vi è un vertice s esposto, allora si va al Passo 5.

Passo 4. (inserimento in T_r di spigoli in M)

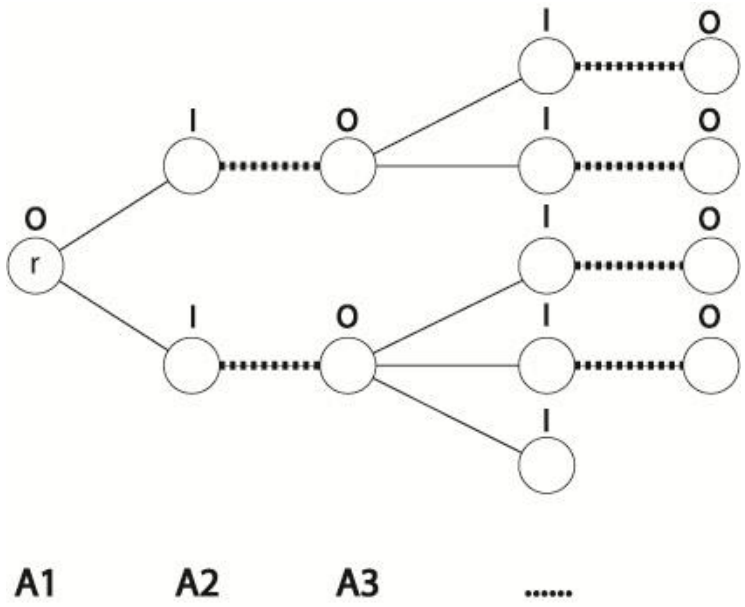
Si pone i = i+1 (si noti che ora i è dispari). Sia A_i l'insieme di vertici accoppiati, mediante spigoli in M, a vertici di A_{i-1}. Si accresce l'albero T_r inserendo, oltre ai vertici di A_i, anche i suddetti spigoli. Si attribuisce ad ogni vertice di A_i etichetta O. Si va al Passo3.

Passo 5. (Aumento del matching)

Nell'albero T_r vi è un unico cammino aumentante P_A da r al vertice esposto s. Si aumenta M, ponendo: M=M\P_A ∪ P_A\M. Si va al Passo 2.

Si osservi che i vertici di A_1 sono tutti outer cosi come quelli di A_3, A_5 ecc.. , mentre sono inner i vertici in A_2, A_4 ecc..; inoltre un vertice esposto (distinto da r) può essere individuato solo in un

insieme con indice pari. L'albero aumentante si ramifica solo usando spigoli che non stanno in M, e cioè a partire da vertici outer come è evidente dalla figura riportato sotto.



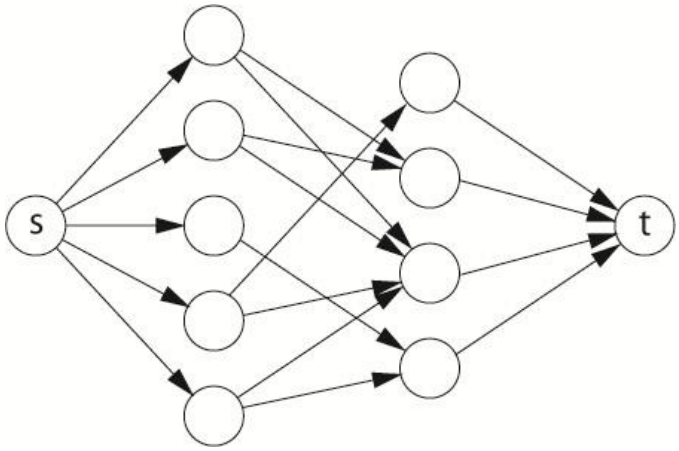
Se in un grafo bipartito G=(U∪V, E) supponiamo che la radice e i vertici outer stiano nel sottoinsieme U, i vertici inner stanno in V e viceversa. Ne consegue che un cammino aumentante collega sempre un vertici esposto in U con un altro pure esposto in V e che pertanto basta ricercare solo alberi aumentanti con radice in U (o, alternativamente, in V).

Tra due vertici esposti possono esistere più cammini aumentanti, ma uno solo di questi è effettivamente utile per incrementare la cardinalità dell'accoppiamento: la procedura descritta in affetti individua uno solo tra più cammini di questo tipo, perche da ogni vertici si possono aggregare altri spigoli ed altri vertici solo se questi non sono già stati inclusi in precedenza nell'albero.

L'algoritmo, per un grafo bipartito G=($U \cup V, E$) di 2n vertici, ha complessità O(n^3). Infatti, supponendo di partire da un matching vuoto, il numero di alberi aumentanti che vanno costruiti è dell'ordine di n, numero di vertici di U e di V. in ogni iterazione poi, la costruzione dell'albero richiede O(n^2) passi.

Accoppiamenti bipartiti di cardinalità massima e problemi di flusso

Il problema della determinazione di un matching massima in un grafo bipartito G = ($U \cup V, E$) può essere affrontato utilizzando algoritmi di flusso massimo in reti opportune. A partire da G si costruisce una rete di flusso G' come segue. Si orientano tutti gli spigoli (u_i, v_j) da U verso V; si aggiunge poi un ulteriore nodo sorgente s e tutti gli m archi del tipo (s, u_i); quindi un nodo pozzo t e tutti gli n archi del tipo (v_j,t), come si può vedere nella figura sotto. A tutti gli archi di G' si attribuisce capacità (superiore) unitaria.



Si risolve quindi il problema di flusso massimo da s a t:

MAX $\sum_{i:u_i \in U} x_{s\,i}$

Soggetto a :

$x_{si} - \sum_{j:(u_i,v_j)\in E} x_{ij} = 0 \qquad \forall\, i\colon u_i \in U$

$\sum_{i:(u_i,v_j)\in E} x_{ij} - x_{jt} = 0 \qquad \forall\, j\colon v_j \in V$

$0 \leq x_{hk} \leq 1$ per ogni arco di G'.

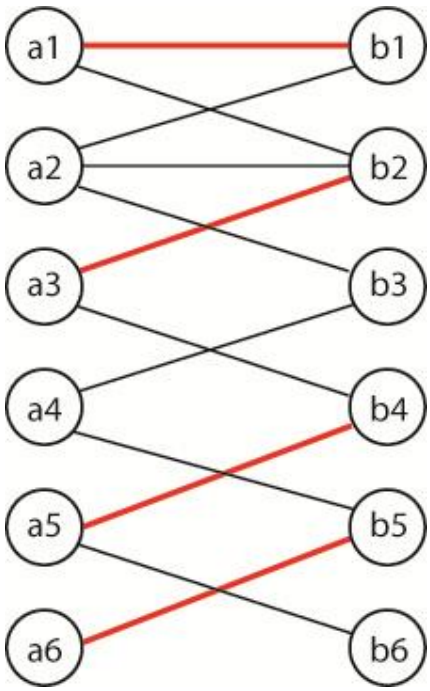
È evidente che il collo di bottiglia è costituito dal fatto che tra i due sottoinsiemi U e V di G non vi sono tutti i possibile spigoli. Si cercherà quindi di utilizzarne il maggior numero, individuando di fatto un accoppiamento di cardinalità massima.

Poiché un problema di massimo flusso è risolubile in O(n^3) passi, viene confermata la risolubilità in tempo O(n^3) del problema di matching di cardinalità massima su grafi bipartiti.

È inoltre interessante esaminare il problema duale che, in termini di flusso, corrisponde a trovare la sezione di capacità minima. La sezione di capacita minima può essere interpretata nel grafo originale come il sottoinsieme di nodi S⊆U∪V di cardinalità minima sufficiente a coprire tutti gli spigoli in E. In altre parole, un nodo u_i sta in S se e solo se il corrispondente arco (s, u_i) è attraversato dalla sezione minima; cosi pure un nodo v_j sta in S se e solo se il corrispondente arco (v_j,t) è attraversato dalla sezione minima. In tal modo tutti gli spigoli di E sono incidenti in almeno uno dei nodi di S.

RISOLUZIONE ANALITICA DI UN PROBLEMA

Sia dato il seguente grafo bipartito, in cui è evidenziato un matching iniziale



Passo 1.

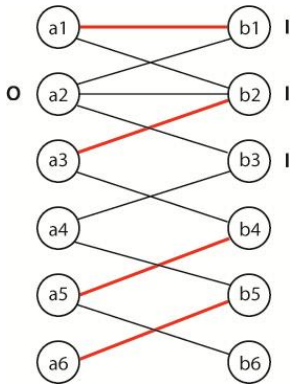
M={(a_1, b_1),(a_3, b_2),(a_5, b_4),(a_6, b_5)}

Passo 2.

R = $(\mathrm{a}_2, \mathrm{a}_4, \mathrm{b}_3, \mathrm{b}_6)$ è l'insieme dei vertici esposti non ancora esplorati. Si prende r = a_2 (r è la radice dell'albero aumentante $\mathrm{T_r}$) e si attribuisce a a_2 l'etichetta 0.
Si pone $\mathrm{i} = 1, \mathrm{A}_1 = \mathrm{A}_\mathrm{i} = \{\mathrm{a}_2\}$, $\mathrm{T_r} = \mathrm{A}_1 = \mathrm{a}_2$ e si dichiara a_2 esplorato.

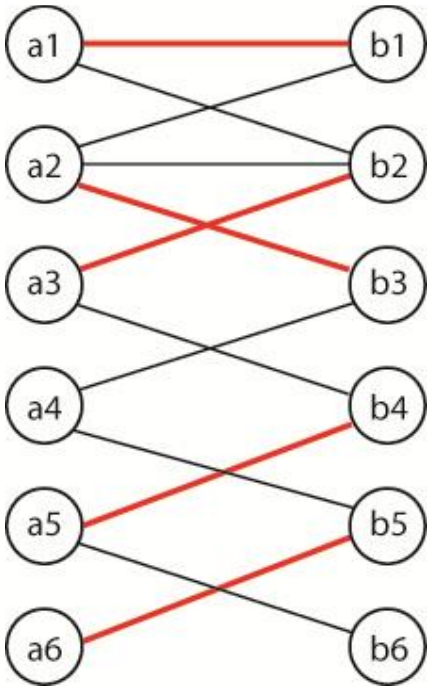
Passo 3.

Si pone $i = i + 1 = 2$. Sia $A_2 = \{b_1, b_2, b_3\}$ l'insieme dei vertici adiacenti ai vertici di A_1 che non sono ancora stati inseriti in T_r. In T_r si inseriscono i vertici di A_2 e per ogni vertice a_i di A_2 si inserisce anche un solo spigolo incidente sia in a_i che in un vertice di A_1. Quindi T_r diventa $T_r = \{(a_2, b_1)(a_2, b_2)(a_2, b_3)\}$. Si etichetta ogni vertice di A_2 con I. Siccome b_3 è un vertice esposto, si va al Passo 5



Passo 5.

Nell' albero ho un solo cammino aumentante $P_A = (a_2, b_3)$. Trovato il cammino aumentante posso aumentare M con la seguente formula $M = M \backslash P_A \cup P_A \backslash M$. Quindi il nuovo matching è $M' = \{(a_1, b_1), (a_2, b_3), (a_3, b_2), (a_5, b_4), (a_6, b_5)\}$. Si va al Passo 2.



Passo 2.

Si cancellano tutte le etichette O e I assegnate prima. L'insieme dei vertici esposti diventa $R = (a_4, b_6)$; poiché $R > 1$ il matching M' non è di cardinalità massima. Scegliendo $r = a_4$ e lo etichetto O. Si pone $i = 1$, $A_1 = a_4$ e $T_r = \{a_4\}$ e si dichiara a_4 esplorato.

Passo 3.

Ho $i = 2$, $A_2 = (b_3, b_5)$ e $T_r = \{(a_4, b_3), (a_4, b_5)\}$. Etichetto ogni vertice di A_2, con I.

Passo 4.

Pongo $i = 3$ e $A_3 = (a_2, a_6)$. A_3 è l'insieme dei vertici accoppiati, mediante spigoli in M', a vertici di A_2. Assegno ai vertici di A_3 l'etichetta O. Si fa crescere l'albero T_r inserendo i vertici di A_3 e i corrispondenti spigoli. T_r diventa $T_r = \{(a_4, b_3), (a_4, b_5), (b_3, a_2), (b_5, a_6)\}$.

Passo 3.

$i = 4$, $A_4 = (b_1, b_2)$ e $T_r = \{(a_4, b_3), (a_4, b_5), (b_3, a_2), (b_5, a_6), (a_2, b_1), (a_2, b_2)\}$

Passo 4.

$i = 5$, $A_5 = (a_1, a_3)$ e $T_r = \{(a_4, b_3), (a_4, b_5), (b_3, a_2), (b_5, a_6), (a_2, b_1), (a_2, b_2), (b_1, a_1), (b_2, a_3)\}$

Passo 3.

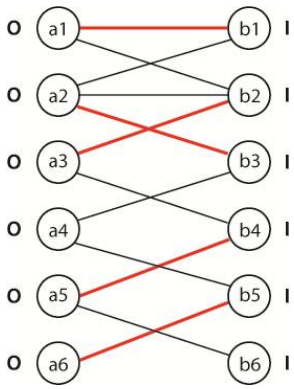
$i = 6$, $A_6 = (b_4)$ e $T_r = \{(a_4, b_3), (a_4, b_5), (b_3, a_2), (b_5, a_6), (a_2, b_1), (a_2, b_2), (b_1, a_1), (b_2, a_3), (a_3, b_4)\}$

Passo 4.

$i = 7,\ A_7 = (a_5)$ e
$T_r = \{(a_4, b_3), (a_4, b_5), (b_3, a_2), (b_5, a_6), (a_2, b_1), (a_2, b_2), (b_1, a_1), (b_2, a_3), (a_3, b_4), (b_4, a_5)\}$

Passo 3.

$i = 8,\ A_8 = (b_6)$ e
$T_r = \{(a_4, b_3), (a_4, b_5), (b_3, a_2), (b_5, a_6), (a_2, b_1), (a_2, b_2), (b_1, a_1), (b_2, a_3), (a_3, b_4), (b_4, a_5), (a_5, b_6)\}$.
b_6 è una vertice esposto quindi vado a Passo 5.

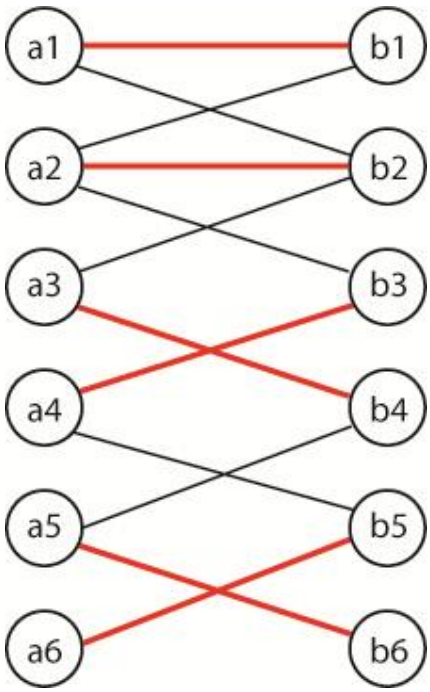


Passo 5.

Trovo il cammino aumentante $P_A = (a_4, b_3, a_2, b_2, a_3, b_4, a_5, b_6)$. Con $M' \backslash P_A = (a_1, b_1), (a_6, b_5)$ e $P_A \backslash M' = (b_4, a_3), (a_2, b_2), (a_4, b_3), (a_5, b_6)$ il nuovo matching risulta
$M" = \{(a_1, b_1), (a_2, b_2), (a_3, b_4), (a_4, b_3), (a_5, b_6), (a_6, b_5)\}$.

Passo 2.

Poiché $R = 0$, l'algoritmo termina e il matching $M"$ è di cardinalità massima.



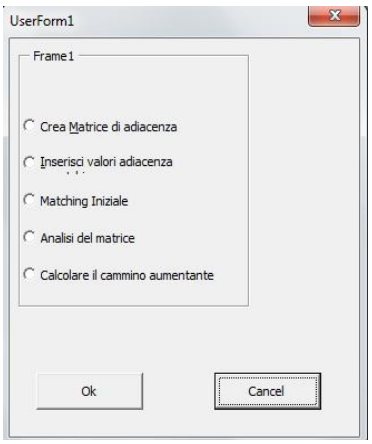
Introduzione al linguaggio VBA

Microsoft Excel è il *foglio elettronico* prodotto da Microsoft, dedicato alla produzione e alla gestione dei fogli elettronici. È parte della suite di software di produttività personale Microsoft Office, ed è disponibile per i sistemi operativi Windows e Macintosh. È attualmente il foglio elettronico più utilizzato.
Una delle potenzialità più importanti di Excel è la possibilità di scrivere delle *macro*, veri e proprie porzioni di codice che possono eseguire in automatico qualsiasi comando. Il linguaggio di programmazione delle macro di Excel è *Visual Basic for Applications* (VBA), un linguaggio di programmazione a oggetti (anche se non possiede tutte le caratteristiche della programmazione orientata agli oggetti) che utilizza la dot notation. Tramite il codice VBA è possibile scrivere macro che modificano le proprietà degli oggetti (ad esempio il colore di una cella), lanciare metodi (ad esempio l'aggiornamento di una tabella pivot) e reagire ad un evento (ad esempio eseguire un pezzo di codice quando viene modificato il contenuto di una cella).
Nonostante il suo stretto legame con *Visual Basic*, VBA non può essere usato per eseguire applicazioni stand-alone, ma è comunque possibile una certa interoperatività fra applicazioni (ad esempio è possibile creare un report in Word a partire da dati di Excel) grazie all'automazione (tecnologia COM, Component Object Model).
VBA è un linguaggio di programmazione ad *alto livello*. I principali oggetti di questo linguaggio sono subroutine e funzioni. La subroutine, chiamata anche procedura o macro, esegue automaticamente un insieme di operazioni, nella cartella, foglio e/o cella selezionate al momento del lancio.
La nostra procedura in linguaggio VBA è stata realizzata con la costruzione di alcune UserForm; esse sono delle interfacce che ci permettono la creazione e la manipolazione di finestre di dialogo personalizzate all'interno di programmi o procedure.
Utilizzando le UserForm, è possibile costruire finestre personalizzate per visualizzare dati, o richiedere all'utente la digitazione di valori, utilizzando la logica che abbiamo impostato per la corretta esecuzione del programma (i valori inseriti devono essere compatibili con quanto impostato nelle istruzioni del programma);
In sostanza le finestre di dialogo permettono al programma di interagire con l'utente in modo più "sofisticato" e forniscono uno strumento versatile per svolgere le normali funzioni di Input e Output. L'oggetto UserForm è una finestra di dialogo vuota e contiene una barra del titolo e un pulsante di chiusura; aggiungendo controlli a un oggetto di tipo UserForm è possibile personalizzare l'aspetto e la funzionalità della finestra di dialogo. Ogni oggetto UserForm possiede proprietà, metodi e risponde ad eventi, inoltre ogni oggetto UserForm incorpora un modulo nel quale l'utente può aggiungere i propri metodi e proprietà e nel quale può scrivere il codice che risponde ad eventi della finestra.

APPLICAZIONE DELL' ALGORITMO IN LINGUAGGIO VBA

Per realizzare l'algoritmo in VBA ho creato sei userform, due delle quali contengono il codice per calcolare i primi quattro passi dell'algoritmo, l' albero aumentante, il cammino aumentante ed il nuovo matching. Le altre quattro userform sono usate per aprire a loro volta altre userform o ottenere informazione dall'utente. Il primo foglio di Excel, chiamato Matrice, è usato per visualizzare la matrice adiacente, e viene modificato durante la elaborazione. Il secondo foglio, chiamato Risultati, è il foglio elettronico dove vengono visualizzati i risultati ottenuti dai varie passaggi. Il terzo foglio è riservato ad alcuni risultati intermedi.

Il programma parte premendo ctrl-c; la prima userform (userform 1) si apre.



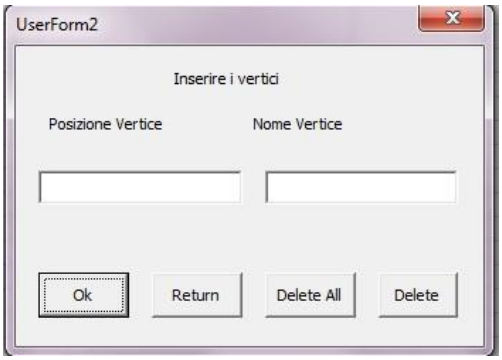
La userform 1 è una finestra nella quale poter scegliere l' operazione da eseguire. Si presenta con cinque pulsanti di opzione, ciascuno che esegue un azione diversa, e due pulsanti di comando. Le opzioni sono state pensate per essere utilizzate in sequenza, dalla prima all'ultima. Ora diamo uno sguardo al codice. Il codice è formato da una serie di enunciati If, che vengono eseguiti a patto che la condizione contenuta negli stessi sia vera. Premendo il pulsante di comando Cancel la userform 1 si chiude, invece premendo il pulsante di comando OK vengono effettuate le operazioni contenute nell'enunciato scelto. Di seguito il codice della userform 1

```
Private Sub CancelButton_Click()
UserForm1.Hide

End Sub
```

```
Private Sub OkButton_Click()

If MatriceAdiacenza.Value = True Then
UserForm2.Show
MatriceAdiacenza.Value = False
End If

If InsAdiacenza.Value = True Then
UserForm3.Show
InsAdiacenza.Value = False
End If

If MatchingInz.Value = True Then
UserForm4.Show
MatchingInz.Value = False
End If

If AnaliseButton.Value = True Then
UserForm5.Show
AnaliseButton.Value = False
End If

If PAButton.Value = True Then
UserForm6.Show
PAButton.Value = False
End If

End Sub
```

```
Private Sub OptionButton1_Click()

End Sub
```

Analizziamo in dettaglio la prima scelta, ossia la userform 2.



La userform 2 si presenta come una finestra dotata di due text box nei quali inserire i dati che andremo a vedere e quattro pulsanti di comando. Viene usata per inserire i vertici della matrice adiacente: all'utente viene chiesto di inserire il nome e la posizione di ogni singolo vertice nelle rispettive due textbox. La subroutine per i quattro pulsanti di commando si presenta cosi:

```
Private Sub DeleteButton_Click()

Sheets("Matrice").Select
Dim N As String
Dim P As Integer
N = UCase(TextBox2.Text)
P = UCase(TextBox1.Text)

If Cells(P + 1, 1) = N And Cells(1, P + 1) = N Then
   Cells(P + 1, 1).Clear
   Cells(1, P + 1).Clear
End If


End Sub
```

```
Private Sub ReturnButton_Click()
UserForm2.Hide
End Sub
```

```
Private Sub DeleteAllButton_Click()
Sheets("Matrice").Select
Cells.ClearContents
End Sub
```

```
Private Sub OkButton_Click()

Sheets("Matrice").Activate
Sheets("Matrice").Select

Dim P As Integer
Dim N As String
N = UCase(TextBox2.Text)
P = UCase(TextBox1.Text)
Cells(1, P + 1) = N
Cells(P + 1, 1) = N
TextBox1.Text = ""
TextBox2.Text = ""


End Sub
```

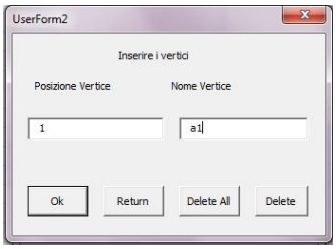
Il bottone Delete cancella, dopo che l'utente ha inserito nome e posizione, il vertice selezionato. La subroutine DeleteButton_Click() salva nelle due variabili, chiamate N e P, il nome e la posizione del vertice. Attraverso un If cerca nella Matrice di adiacenza se nella posizione indicata esiste il nome inserito dall'utente. Se la condizione dell'If è vera, viene cancellato il contenuto della cella. Siccome è una matrice di adiacenza deve sia trovare il nome nella primo riga e nella prima colonna. Il bottone delete all cancella tutto il contenute sul foglio matrice. Il bottone return chiude la Userform2. Premendo OK il nome del vertice viene inserito nella posizione indicata sul foglio Matrice. Anche qua viene salvato il nome e la posizione, inseriti dall'utente, nelle variabili string nominate N e P. Poiché è una matrice di adiacenza, sia sulla prima riga che sulla prima colonna deve essere riportato il nome del vertice. Questo lo fa il programma in automatico, grazie al codice

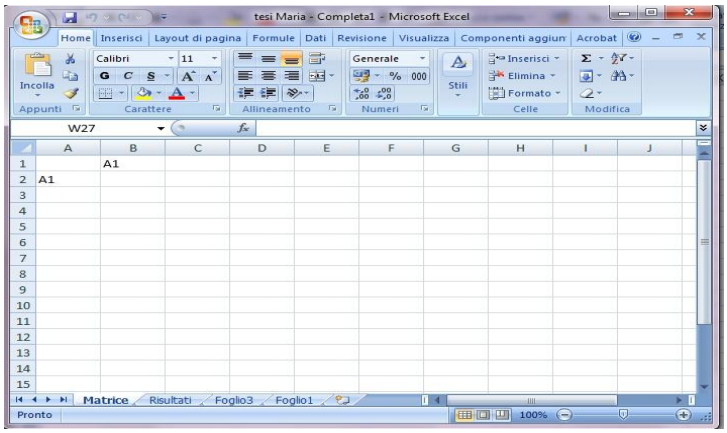
```
Cells(1, P + 1) = N
Cells(P + 1, 1) = N
```

Questo prima inserisce il nome del vertice nella prima riga sulla colonna indicata e dopo inserisce il nome sulla prima colonna nella riga indicata. Gli ultimi due comandi che si vedono cancellano il nome e la posizione scritta dall'utente nel textbox1 e 2 cosi da essere pronti per inserire una nuova posizione e nome.

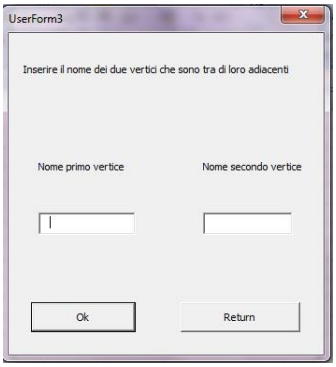
```
TextBox1.Text = ""
TextBox2.Text = ""
```

Nel nostro esempio il nome del primo vertice che voglio inserire è a_1 nella posizione uno. La Userform2 e il foglio matrice si presentano così per l'inserimento del primo vertice:





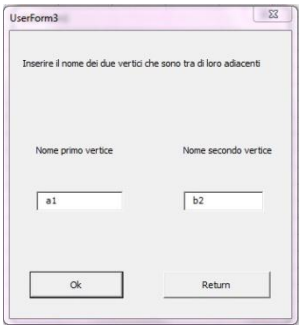
Dopo avere inserito tutti i vertici si passa alla seconda opzione presente nella userform1. Scegliendo la seconda opzione appare la userform3. Questa userform viene utilizzata per permettere all'utente di inserire le coppie di adiacenza. Viene chiesto di immettere i nomi dei due vertici tra loro adiacenti:

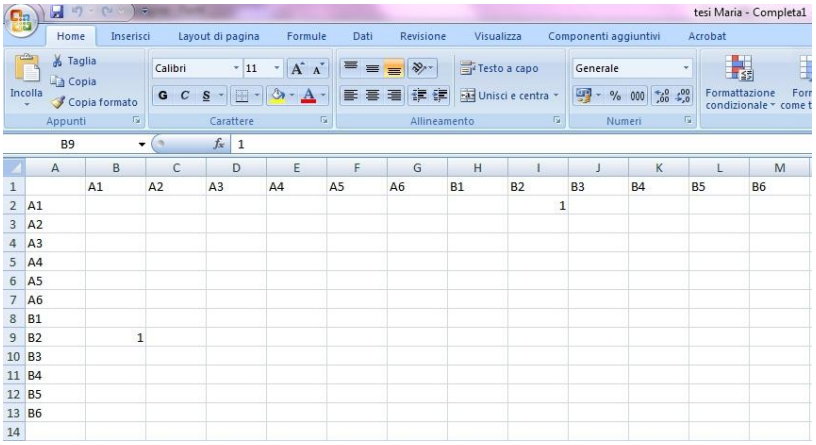



```
Private Sub OkButton2_Click()

Sheets("matrice").Activate
Sheets("Matrice").Select
Dim N1 As String
Dim N2 As String
N1 = UCase(TextBox1.Text)
N2 = UCase(TextBox2.Text)
Dim X As Integer
Dim Y As Integer

For X = 1 To 100
    If Cells(X, 1) = N1 Then
       For Y = 1 To 100
          If Cells(1, Y) = N2 Then
             Cells(X, Y) = "1"
             Cells(Y, X) = "1"
          End If
       Next Y
    End If
Next X

TextBox1.Text = ""
TextBox2.Text = ""

End Sub
```

```
Private Sub ReturnButton2_Click()
UserForm3.Hide
End Sub
```

La Subroutine OkButton2_Click() salva temporaneamente i nomi della coppia di adiacenza in due variabili string, nominate N1 e N2. Poi utilizzando un For annidato con due If ed un'altro For cerca sul foglio Matrice se sono presenti i due nomi inseriti nella userform3. Il primo If viene verificato se trova, sull'foglio matrice, il nome del primo vertice inserito. Se esiste il nome viene eseguito il secondo For e If. Allora il secondo If cerca nella prima riga se trova il secondo nome inserito nella userform3. Se la ricerca da esito positivo inserisce nella matrice il valore "1" in corrispondenza delle due celle appartenenti ad entrambi i vertici . Per esempio nel nostro esercizio a_1 è adiacente al vertice chiamato b_2. Quindi poiché una matrice di adiacenza è speculare rispetto alla diagonale viene inserito un "1" nella cella I2 e B9. Il userform3 e il foglio matrice si presenta così:





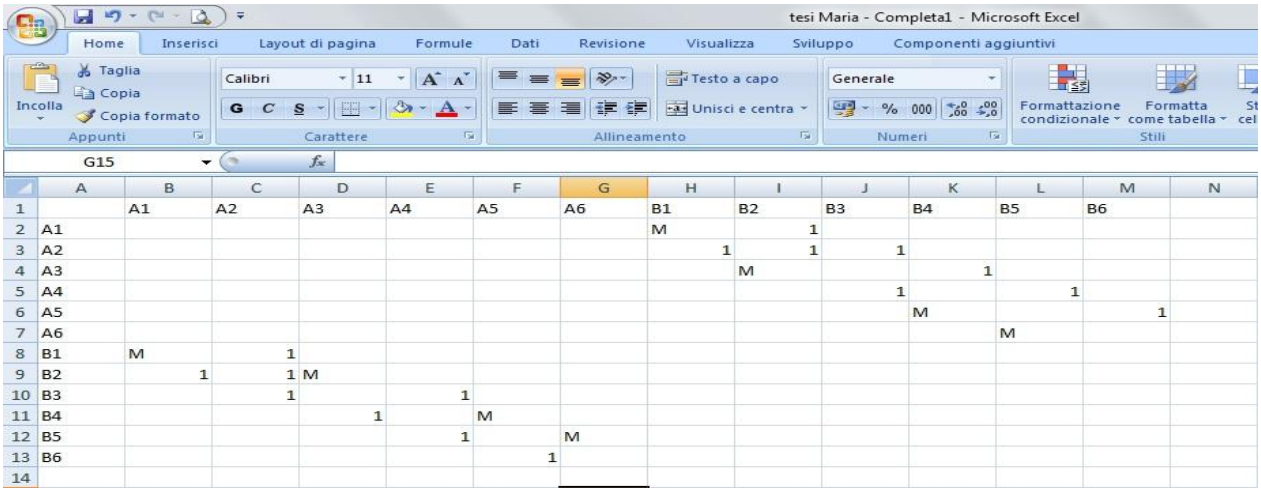
	A	B	C	D	E	F	G	H	I	J	K	L	M
1		A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6
2	A1								1				
3	A2												
4	A3												
5	A4												
6	A5												
7	A6												
8	B1												
9	B2		1										
10	B3												
11	B4												
12	B5												
13	B6												
14													

Dopo aver inserito tutte le coppie dei vertice di adiacenza si può passare alla quarta opzione sul userform1. Scegliendo la quarta opzione appare l'userform4.


```
Private Sub OkButton3_Click()

Sheets("matrice").Activate
Sheets("Matrice").Select
Dim v1 As String
Dim v2 As String
v1 = UCase(TextBox1.Text)
v2 = UCase(TextBox2.Text)
Dim Z As Integer
Dim Z1 As Integer

For Z = 1 To 100
    If Cells(Z, 1) = v1 Then
       For Z1 = 1 To 100
          If Cells(1, Z1) = v2 Then
             Cells(Z, Z1) = "M"
             Cells(Z1, Z) = "M"
          End If
       Next Z1
    End If
Next Z

TextBox1.Text = ""
TextBox2.Text = ""

End Sub

Private Sub ReturnButton3_Click()
UserForm4.Hide
End Sub
```

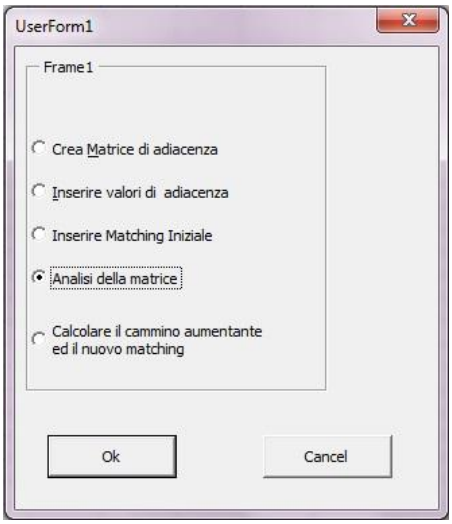
In questa fase si può inserire l'eventuale matching iniziale. Funziona in maniera similare alla userform3. Viene chiesto di immettere i nomi dei due vertici tra loro accoppiati: il codice salva temporaneamente i nomi della coppia in due diverse variabili, poi cerca nel foglio Matrice se sono presenti questi due nomi. Se la ricerca da esito positivo inserisce nella matrice il valore "M" in corrispondenza delle due celle appartenenti ad entrambi i vertici.
Una volta inseriti i vertici, i valori di adiacenza e l'eventuale matching, abbiamo creato una matrice di adiacenza , sulla quale vogliamo trovare un matching di cardinalità massima.

Nell'nostro esempio la matrice di adiacenza, che rappresenta il nostro grafo bipartito, dopo avere inserito tutti i vertici, le coppie adiacente e il matching iniziale si presenta così:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	
2	A1							M	1					
3	A2							1	1	1				
4	A3								M		1			
5	A4									1		1		
6	A5										M		1	
7	A6											M		
8	B1	M	1											
9	B2	1	1	M										
10	B3		1		1									
11	B4			1		M								
12	B5				1		M							
13	B6					1								
14														

L'quarta scelta attiva la userform 5, nella quale è presente il codice principale.



Il codice individua un vertice esposto e, partendo da questo, arriva a calcolare tutti gli insiemi dei vertici adiacenti, accoppiati e l'albero aumentante. Vediamo ora il codice un po' alla volta.

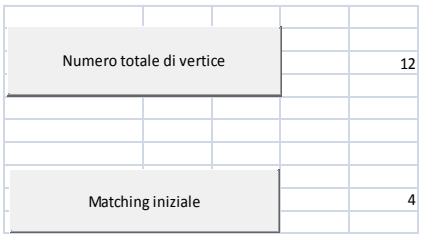
```
Private Sub NoButton_Click()
UserForm5.Hide
End Sub

Private Sub SiButton_Click()

Call SumVertice
Call MatchingIniziale

M = Worksheets("Risultati").Range("e10").Value
totv = Worksheets("Risultati").Range("e4").Value - 1

Dim R As Integer
Dim C As Integer
Dim RA As Integer
Dim CA As Integer
Dim RB As Integer
Dim CB As Integer
Dim i As Integer
Dim E As Integer
Dim g As Integer
Dim j As Integer

Dim InsA As String
Dim X As String
Dim Y As String

Dim TR_Matrice() As Variant
Dim Ins_A3() As Integer
Dim M1() As String
Dim V_Etichetta() As String
Dim V_Esplorati() As Boolean
Dim Tr_Inserito() As Boolean
Dim Ins_A1() As Integer
Dim Ins_R() As Integer

ReDim Preserve Ins_A3(totv)
ReDim Preserve Ins_A1(totv) As Integer
ReDim Tr_Inserito(0, totv) As Boolean
ReDim M1(totv, totv) As String
ReDim V_Etichetta(0, totv) As String
ReDim V_Esplorati(0, totv) As Boolean
ReDim Preserve Ins_R(totv)
ReDim Preserve TR_Matrice(totv, totv) As Variant


Worksheets("Matrice").Activate
Worksheets("Matrice").Select

For R = 0 To totv
    For C = 0 To totv
        M1(R, C) = Cells(R + 2, C + 2)
    Next C
Next R
```

La Private Sub NoButton_Click() contiene il codice per il bottone nominato No, utilizzato per chiudere la Userform5.

Scegliendo il bottone nominato Si viene eseguito il codice contenuto nella Subroutine chiamata Private Sub SiButton_Click().

Le prime due istruzioni che incontriamo usano la funzione predefinita di VBA Call, che richiama un'altra Subroutine e la esegue. Esegue le stesse azioni dell'utente che preme i pulsanti "numero totale di vertici" e " Matching iniziale" sul foglio 2. Le due subroutine dei due bottoni sono state scritte nel modulo 2, in modo che possano essere richiamate in tutte le altre subroutine.

Il codice relativo al Sub SumVertice() definisce una variabile inputrange come intervallo della prima colonna sul foglio Matrice, ossia la colonna dove sono scritti tutti i nomi dei vertici. Si usa poi un ciclo For Each, con al suo interno un comando If. Il codice funziona con questa logica: ogni cella contenuta in inputrange viene controllata e se questa è diversa da vuoto alla variabile sum viene sommato 1. Continua così fino a che raggiunge la fine del range. Poi nel foglio "Risultati" scrive il valore della variabile sum, ossia il numero totale dei vertici. In Sub MatchingIniziale() nella variabile inputrange1 viene assegnato il range sul foglio "Matrice" che contiene la matrice di adiacenza e scorre tutto; se trova la lettere M, somma 1 alla variabile sum. Il valore del variabile sum, che è la cardinalità del matching corrente, viene mostrato sul foglio "Risultati". Nel nostro esempio il numero di vertici è dodici e il numero di matching è quattro, come si vede nella figura.




```
Sub SumVertice()

Set inputrange = Sheets("Matrice").Range("A1:A100")
For Each cell In inputrange
   If cell.Text <> "" Then
     Sum = Sum + 1
     End If
     Next cell
     Worksheets("Risultati").Select
     Worksheets("Risultati").Range("e4").Value = Sum


End Sub
```

```
Sub MatchingIniziale()

Set inputrange1 = Sheets("Matrice").UsedRange
For Each cell In inputrange1
   If cell = "M" Then
     Sum = Sum + 1
     End If
     Next cell
     Worksheets("Risultati").Select
     Worksheets("Risultati").Range("e10").Value = Sum \ 2


End Sub
```

Torniamo al codice relativo alla Sub SiButton_Click(). Questa salva in un variabile chiamata M la cardinalità del matching e in totv il numero totale dei vertici. Si vede poi una serie del codice che contiene tutte le variabili che sono state utilizzate nel programma, che per semplicità possiamo dividere in tre gruppi:

- Il primo gruppo contiene variabili dichiarate come integer che hanno la funzione di contatore; per esempio in un ciclo For, per aumentare il numero delle colonne quando viene riportato il risultato su Excel, o come contare di un azione, quando accade qualcosa che fa aumentare la variabile.

- Il secondo gruppo contiene variabili dichiarate String. Queste vengono usate per salvare i nomi dei vertici.

- Il terzo blocco è un serie di array sia monodimensionali che multidimensionali, dichiarati string, integer, boolean o variant, in base al tipo di dati che andranno a contenere. Inizialmente vengono definiti senza una dimensione, in maniera tale che possano essere ridimensionati in base alla dimensione della matrice che vado ad analizzare (la variabile totv). Quando viene salvato in totv il numero dei vertici totali, sottraggo un unità (nel nostro caso a totv viene assegnato il numero 11=12-1). Questo perche in un array, quando non specificato diversamente, il primo posto viene considerato come posizione zero. Ciò vuol dire che, se all'interno dell'argomento dell'array viene scritto 12 (Il nostro numero di vertici totali), l'array avrebbe in realtà 13 posti. Per evitare questo si deve sottrarre 1 dal numero totale dei vertici, in maniera tale da avere tanti posti nell'array uguali all'effettivo numero di vertici .

Procedendo incontriamo il primo ciclo For con annidato al suo interno un altro ciclo For; questo blocco di istruzioni non fa altro che copiare in una matrice M1(totv,totv) tutte le celle della matrice di adiacenza del foglio “Matrice”, in modo che l’analisi di quest’ultima venga effettuata sulla matrice M1, senza dover sempre riferirsi al foglio Excel. Poiché la prima colonna e riga della matrice di adiacenza sul foglio elettronico di Excel sono utilizzate per contenere i nomi dei vertici, la matrice M1 sarà scalata indietro di 2 posizioni rispetto d essa, cosi che la posizione (0,0) in M1 corrisponde alla cella (2,2) nella matrice di adiacenza sul foglio Excel. Nella matrice di adiacenza la diagonale deve sempre essere vuota.

Le due variabili C ed R corrispondono rispettivamente alle colonne ed alle righe (con la correzione delle due posizioni spiegata poco sopra). Vengono definite da 0 fino a totv, in modo da ricreare la matrice uguale alla matrice iniziale, senza il rischio di creare una matrice più grande sprecando memoria o una più piccola andando a perdere dati. Il primo ciclo For seleziona la prima riga della matrice, mentre Il ciclo For annidato parte dalla prima colonna di quella riga e comincia a copiare il valore delle celle dentro M1 e prosegue fino ad arrivare alla fine delle colonne (nel nostro caso 11). A questo punto il ciclo For principale passa alla seconda riga ed il suo ciclo annidato ricomincia a copiare ogni singola cella. Il processo continua fino a che anche il ciclo principale non arriva all’ultima riga.

Il Passo 2 dell’algoritmo “Matching di massima cardinalità in grafi bipartiti” è stato realizzato con una procedura, chiamata appunto Passo2, in cui troviamo una serie di cicli For e If. Vediamo il primo blocco di istruzioni.

```
Passo2:

i = 0

E = 0
For R = 0 To totv
   For C = 0 To totv
       If M1(R, C) = "M" Then
          E = E + 1
       End If
   Next C
      If E = 0 Then
        Ins_R(R) = 2
      End If
        If E <> 0 Then
          Ins_R(R) = 0
        End If
      E = 0
Next R

Worksheets("foglio3").Cells.ClearContents
Worksheets("Risultati").Range("I9").ClearContents

j = 0
For R = 0 To totv
    j = j + 1
    Worksheets("foglio3").Cells(1, j).Value = Ins_R(R)
Next R


E = 0
For R = 0 To totv
    If Ins_R(R) = 2 And V_Esplorati(0, R) = False Then
       E = E + 1
    End If
Next R

If E <= 1 Then
    Worksheets("Risultati").Select
    Worksheets("Risultati").Range("I9").Value = "Il matching di cardinalità massima è:  " & M
    Exit Sub
End If
```

La prima cosa che si vede è la variabile “i”, dichiarata all‘inizio come integer, cioè una variabile che può solo contenere numeri. Viene usata per attribuire l’indice agli insieme dei vertici adiacenti e accoppiati. Così quando sul foglio “risultati” viene mostrato un insieme nominato con un indice pari (es. A2 ,A4, A6 …) sappiamo che si tratta di un insieme di vertici adiacenti mentre quando nello stesso foglio incontro insiemi nominati con un indice dispari (es. A3, A5, A7…) so che si tratta di insiemi di vertici accoppiati. Unico insieme che non segue questa regola è l’insieme A1, che contiene il vertice esposto.

Dopodiché vediamo un ciclo For che contiene un altro For e tre enunciati If. Questo ha lo scopo di trovare tutti i vertici esposti. Il primo For, parte dalla prima riga; il secondo For scorre tutte le colonne della matrice M1, appartenenti alla prima riga, e se trova una cella che contiene un “M”, la variabile E viene aumentata di un unità. Se E risulta essere uguale a zero, cioè se non è stato trovato nessun “M” su tutte le colonne di quella riga, vuol dire che il vertice corrispondente a quella riga è un vertice esposto. Dentro all’array nominato Ins_R(), che è un array monodimensionale costituito da una riga e tante colonne quante sono i vertici, viene salvato un 2 nella posizione relativa al vertice esposto. Per esempio nel nostro esercizio scorrendo la riga relativa al vertice a_2 , non si trova nessun “M”, perciò è una vertice esposto. Quindi dentro Ins_R viene salvato un 2 nella posizione 1 dell’array, corrispondente al vertice A2. Se invece risulta essere E diverso da zero nel Ins_R viene assegnato uno zero. Questo accade per ogni vertice presente nella matrice M1.

Ora troviamo due istruzioni che cancellano i risultati intermedi presenti sul foglio3 e il contenuto della cella I9 sul foglio risultati, quest’ultima dedicata al valore della cardinalità massima. Il foglio 3 è riservato per scrivere l’array dei vertici esposti e la matrice dell’albero aumentante (sul quale torno dopo), questo perché utilizzo queste informazioni anche nella userform 6 per calcolare il cammino aumentante ed il matching nuovo. Infatti le variabili dichiarate all’interno di una userform non sono accessibili ad altre userform..

Il For successivo copia sul foglio 3 l’array dei vertice esposti. Come in figura.

0	2	0	2	0	0	0	0	2	0	0	2

Nell’esercizio abbiamo trovato quattro vertici esposti R = (a_2, a_4, b_3, b_6) . Nell’array sopra è appunto stato assegnato il valore 2 in quattro posti. Nella seconda cella ce un 2 che rappresenta il vertice a_2 nella quarta cella ce un 2 che rappresenta a_4 e lo stesso per b_3 e b_6

Successivamente incontriamo un'altra ciclo For annidato con un If. Il For conta semplicemente quanti vertici sono presenti nella Ins_R. Ogni volta che la condizione del If si verifica, la variabile contatore E viene aumentata di un’unità. La condizione dell’If è verificato ogni volta che nell’array Ins_R viene trovato un “2” e contemporaneamente questa vertice non è già stato esplorato. Nel nostro caso viene contato 4 vertici. L’ultimo If viene solo eseguita se il valore contenuto in E è minore o uguale a uno. Nel nostro caso E è uguale a quattro, quindi non viene verificata la condizione dell’If. Se avessi avuto solo un vertice o nessuno, la condizione dell’If sarebbe stata verificata e avrebbe stampato sul foglio risultati il valore della cardinalità massima. L’istruzione Exit Sub è un istruzione che permette al programma di uscire dal subroutine senza eseguire le eventuali istruzioni successive. Siccome un volta trovata il matching di cardinalità massima l’algoritmo termina, l’exit sub è stato inserito per permettere ciò.

Ora vediamo il secondo set di istruzioni per il Passo 2 dell'algoritmo.

```
i = i + 1
RA = 17
CA = 2

For C = 0 To totv
    V_Etichetta(0, C) = ""
Next C


Worksheets("Risultati").Range("InsA").ClearContents
Worksheets("Risultati").Range("Tr").ClearContents
Worksheets("risultati").Range("PA").ClearContents


For R = 0 To totv
    Ins_A3(R) = 0
Next R


For R = 0 To totv
    If Ins_R(R) = 2 And V_Esplorati(0, R) = False Then
        V_Esplorati(0, R) = True
        V_Etichetta(0, R) = "O"
        Tr_Inserito(0, R) = True
        Ins_A3(R) = 2
        InsA = Worksheets("Matrice").Cells(1, R + 2)
        Worksheets("Risultati").Cells(RA, CA - 1) = "Insieme A " & i & " è: "
        Worksheets("Risultati").Cells(RA, CA) = InsA
        Exit For
    End If
Next R
```

Prima incontriamo tre variabili, la "i" ,vista già prima, che ora viene aumentata di uno. Questo evento avrà luogo ogni volta che si eseguirà questo blocco di istruzioni. La variabile "i" viene aumentata ogni volta che si arriva ad una procedura che corrisponde ad un passo del algoritmo. Viene solo azzerato quando è stato trovato un nuovo matching o se nel passo3 risulta che l'insieme dei vertici adiacenti sia vuoto. Le variabili RA e CA rappresentano invece la riga e la colonna iniziali nel foglio "Risultati" dove verranno pubblicati i risultati dell'elaborazione. Proseguendo troviamo un ciclo For che ha come unica funzione quella di cancellare l'array chiamato V_Etichetta. V_Etichetta è un array monodimensionale, con una riga e tante colonne quante sono i vertici del matrice di adiacenza, che tiene conto dei vertici etichettati. Ogni volta che si trova un vertice adiacente a quello precedente si assegna a questo un I; lo stesso ai vertici accoppiati viene assegnato un O. L'array V_Etichetta deve essere cancellata ogni volta che si ritorna al passo 2 perché l'algoritmo inizia da capo.

Troviamo poi tre istruzioni che cancellano sul foglio risultati gli insiemi dei vertici adiacenti e accoppiati, albero aumentante e il cammino aumentante trovati in precedenza. Gli intervalli sul foglio risultati, dove vengono scritti questi risultati, sono stati nominati rispettivamente InsA, Tr e PA.

Procediamo con un'altro For che svolge la stessa funzione del For incontrato prima, cioè cancella il contenuto dell'array Ins_A3(). La differenza sta nel fatto che ora per cancellare sovrascrive le celle con una serie di zero; questo perché Ins_A3() è un array dichiarato Integer, quindi può solo contenere numeri. Quindi non si può sovrascrivere il numero prima contenuto con un spazio vuoto.

Il cuore vero e proprio del passo 2 dell'algoritmo è contenuto nel successivo ciclo For con annidato il comando if. Questo cerca nell'Ins_R, l' insieme trovato in precedenza contenente tutti i vertici esposti, un vertice esposto. La condizione dell'If è verificata quando viene trovato un vertice esposto, che non è ancora stato esplorato. V_Esplorato è un array che tiene conto dei vertici esposti che sono già stati esplorati. Secondo l'algoritmo se al Passo 3 viene trovato un insieme vuoto si ritornare al Passo 2, e si sceglie un nuova vertice esposto su cui fare l'analisi. Quindi, come si vede dopo, assegnando al V_Esplorato un True, se si deve ritornare al passo 2 non si corre il rischio che la programma fa l'analisi sullo stesso vertice esposto. Se la condizione è verificata viene eseguito l'istruzione del If che assegna all'array V_esplorati e Tr_Inseriti un True. Tr_Inseriti è l'array che tiene conto dei vertici già inseriti nell'albero aumentante; questo etichetta il vertice con un "O" e assegna un 2 alla posizione del vertice nel Ins_A3. L'istruzione Exit For fa si che esca dal ciclo For subito dopo aver trovato il primo vertice esposto. Se non era stato messo il exit for il for avrebbe continuato a girare fino ad avere trovato il ultimo vertice esposto. Quindi il programma sceglie sempre il primo vertice esposto che trova per fare l'analisi seguente, a differenza di quello che può avvenire facendo l'esercizio a mano. Nel nostro esempio essendo R = (a_2, a_4, b_3, b_6) viene scelto il vertice a_2 ,perche è il primo vertice esposto e non ancora esplorato che il ciclo trova, da cui iniziare l'analisi. Il foglio risultati si presenterà come nella figura seguente.

Numero totale di vertice	12
Matching iniziale	4
GLI INSIEMI DEI VERTICI ADIACENTI E ACCOPPIATI	
Insieme A 1 è:	**A2**

Dopo avere trovato un vertice esposto da cui iniziare l'analisi, si passa al Passo3. L'istruzione Gosub Passo3 fa eseguire il blocco di istruzioni denominato Passo3 e Exit Sub ha la funzione di uscire dal subroutine attuale. Vediamo il codice iniziale per il passo 3.

```
Passo3:

i = i + 1
For C = 0 To totv
   Ins_A1(C) = 0
Next C


E = 0
For R = 0 To totv
    If Ins_A3(R) = 2 Then
       For C = 0 To totv
          If M1(R, C) = "1" And Tr_Inserito(0, C) = False Then
             Ins_A1(C) = 2
             TR_Matrice(R, C) = "1"
             E = E + 1
          End If
       Next C
    End If
Next R

If E = 0 Then
   GoSub Passo2
   Exit Sub
End If
```

La prima sequenza di istruzioni aumenta la variabile “i” di un unita come accadeva nel passo 2,nel nostro caso la nostra variabile diventa 2. Il ciclo For cancella tutto il contenuto dell’array Ins_A1, il quale contiene i vertici adiacenti ai vertice trovati in precedenza salvati in Ins_A3. Serve che Ins_A1 viene cancellato ogni volta che si arriva ad Passo3 perche durante l’algoritmo l’ insieme dei vertice adiacenti si cambia e anche l’insieme dei vertici accoppiati.

Il secondo blocco di istruzioni è un For con annidato due if e un'altro For. Prima cerca nell’array contenente i vertici trovati nel passo precedenti un 2, se la ricerca dà esito positivo (quindi verifica la condizione del primo if) prosegue al secondo For. Nel nostro caso la Ins_A3 contiene solo il vertice esposto a_2, che è salvato nella posizione 1 del array. Il secondo For cerca per ogni colonna della riga di a_2,cioè la riga 1, dentro la matrice M1. Se trova un “1” e il vertice corrispondente non sia già stato inserito nell’albero aumentante, assegna un “2” nella posizione della colonna trovata nel Ins_A1. Nel nostro caso viene assegnata un 2 nelle posizioni corrispondenti a b_1, b_2, b_3 . L’array Tr_matrice è un array multidimensionale con tante colonne e righe quanti sono il numero dei vertici della matrice di adiacenza. Viene usato per l’inserimento di uno spigolo tra i vertici trovati. Nel nostro esempio viene assegnato un “1” nella cella (1,6),(1,7)(1,8) corrispondente agli spigoli tra $(a_2, b_1)(a_2, b_2)(a_2, b_3)$. Le istruzioni all’interno del primo If vengono eseguite solo se nell’Ins_A3 si trova il valore 2.

Se l’insieme dei vertici adiacenti è vuoto, si deve tornare al passo2 e scegliere un'altro vertice esposto dal quale far partire l’analisi. La variabile E viene aumentata di un unità ogni volta che si trova un vertice adiacente. Se E resta uguale a zero, ciò vuol dire che non sono stati trovati vertici adiacenti e si deve tornare al Passo2. L’ultima If ha la seguente funzione. GoSub Passo2 fa eseguire il Passo2 di nuovo e l’exit Sub fa si che si esce dalla procedura attuale. L’If non viene eseguita nel nostro esercizio, perche come detto prima trova tre vertici adiacenti al vertice esposto.

Proseguiamo nel codice.

```
RA = RA + 2

j = 0
For C = 0 To (totv)
    If Ins_A1(C) = 2 Then
       j = j + 1
       V_Etichetta(0, C) = "I"
       Tr_Inserito(0, C) = True
       InsA = Worksheets("Matrice").Cells(1, C + 2)
       Worksheets("Risultati").Cells(RA, CA - 1) = "Insieme A " & i & " è "
       Worksheets("Risultati").Cells(RA, CA - 1 + j) = InsA
    End If
Next C


GoSub Passo32
Exit Sub
```

Questa blocco di istruzioni porta sul foglio Matrice i nomi degli elementi dell’insieme dei vertici adiacenti ai vertici di A_{i-1}. Il blocco è composto da un ciclo for che contiene una variabile c e un istruzione If. La variabile C parte da zero fino al totv (uguale ad undici nel nostro caso) che aumenta di un unita ad ogni ciclo. L’If controlla una posizione nel Ins_A1 corrispondente al valore attuale di C. Se è presente un valore diverso da 2, l’istruzione dell’ If non viene eseguita, la variabile C viene aumentata di un’unità e riparte il ciclo. Altrimenti se trova un 2 nel Ins_A1, viene

eseguita l'istruzione dell'If. La variabile j è un variabile contatore che viene aumentata di un'unità ogni volta che si verifica la condizione del If. Nell'array V_Etichetta viene assegnato un "I" nella posizione indicata dalla variabile C, perche i vertici contenuti in Ins_A1 sono vertici adiacenti ai vertici contenuti in Ins_A3, e nel Tr_ inserito un True. Nella variabile InsA, che è un variabile dichiarato string, viene salvato il nome del vertice trovato sulla prima riga della Matrice di adiacenza nel foglio Matrice. Alla fine viene riportato il risultato sul foglio Risultati. Siccome j viene aumentata di un'unità ogni volta che il If si verifica, mi assicuro che i nomi dei vertici, riportati sul foglio Risultati, non vengono sovrascritti. Il foglio Risultati dopo questo procedimento si presenterà così.

Numero totale di vertice				12	
Matching iniziale				4	
GLI INSIEMI DEI VERTICI ADIACENTI E ACCOPPIATI					
Insieme A 1 è:	**A2**				
Insieme A 2 è	**B1**	**B2**	**B3**		

GoSub Passo32 fa eseguire il blocco di istruzioni denominato Passo32. Vediamolo:

```
Passo32:
   E = 0
    For C = 0 To totv
        If Ins_A1(C) = 2 And Ins_R(C) = 2 Then
            E = E + 1
        End If
    Next C

    If E = 0 Then
       GoSub Passo4
       Exit Sub
    End If


    If E <> 0 Then
       GoSub Passo5
       Exit Sub
    End If
```

Queste blocco di istruzioni ha lo scopo di controllare se nell'insieme dei vertici adiacenti è presente un vertice esposto: se questo è verificato si passa direttamente al Passo 5. Questo è stato realizzato con un ciclo For con al suo interno un If. L'If controlla la posizione nel Ins_A1 e nell'Ins_R corrispondente al valore della variabile C. Se in entrambi viene trovato un "2", allora la variabile contatore E viene aumentata di un'unità. Essendo Ins_A1 l'insieme dei vertici adiacenti e Ins_R l'insieme dei vertici esposti, se un vertice è presente sia nell'Ins_A1 che nell'Ins_R vuol dire che esso è esposto, allora E viene aumentato. Nel nostro caso avendo $A_2 = \{b_1, b_2, b_3\}$ e R = (a_2, a_4, b_3, b_6) si trova che b_3 è un vertice esposto. Quindi quando C arriva ad nove, la condizione del If è verificato e la variabile E diventa uno. L'If , che si trova subito dopo il For, viene eseguito solamente se non viene trovato un vertice esposto nell'insieme dei vertici esposti. L'istruzione dell'If fa eseguire il blocco di codice nominato Passo4 e termina l'attuale blocco di istruzione. Il secondo If viene eseguito ogni qual volta che si trova un vertice esposto nell'insieme dei vertici adiacenti. L'istruzione GoSub Passo5 esegue il blocco di codice nominato Pass5 e termina l'attuale blocco d'istruzione. Nel nostro esempio abbiamo trovato un vertice esposto b_3. L'If con la

condizione E=0 viene saltato e viene eseguito il secondo che ci manda al Passo 5. Ora vediamo Il codice relativo al passo 5. Il codice del passo 4 lo vedremo più avanti quando andremo ad utilizzarlo.

```
Passo5:

RB = 3
CB = 10

j = 0
For R = 0 To totv
    If Tr_Inserito(0, R) = True Then
       X = Worksheets("Matrice").Cells(1, R + 2)
         For C = 0 To totv
           If TR_Matrice(R, C) = "1" Or TR_Matrice(R, C) = "M" Then
              j = j + 1
              Y = Worksheets("matrice").Cells(1, C + 2)
              Worksheets("Risultati").Cells(RB, CB - 1 + j) = X & Y
           End If
         Next C
    End If
Next R


g = 0
j = 0
For R = 0 To totv
    j = 0
    g = g + 1
      For C = 0 To totv
          j = j + 1
          Worksheets("foglio3").Cells(3 + g, j).Value = TR_Matrice(R, C)
      Next C
Next R
```

Il blocco di istruzioni del passo5 ha la funzione di trascrivere sul foglio Risultati l'albero aumentante e sul foglio 3 l'array TR_Matrice. L'albero aumentante viene scritto come una serie di coppie di nodi, in modo da indicare anche lo spigolo che collega i due vertici. L'array TR_matrice viene usato come un risultato intermedio che ci serve quando entra in funzione la userform6.

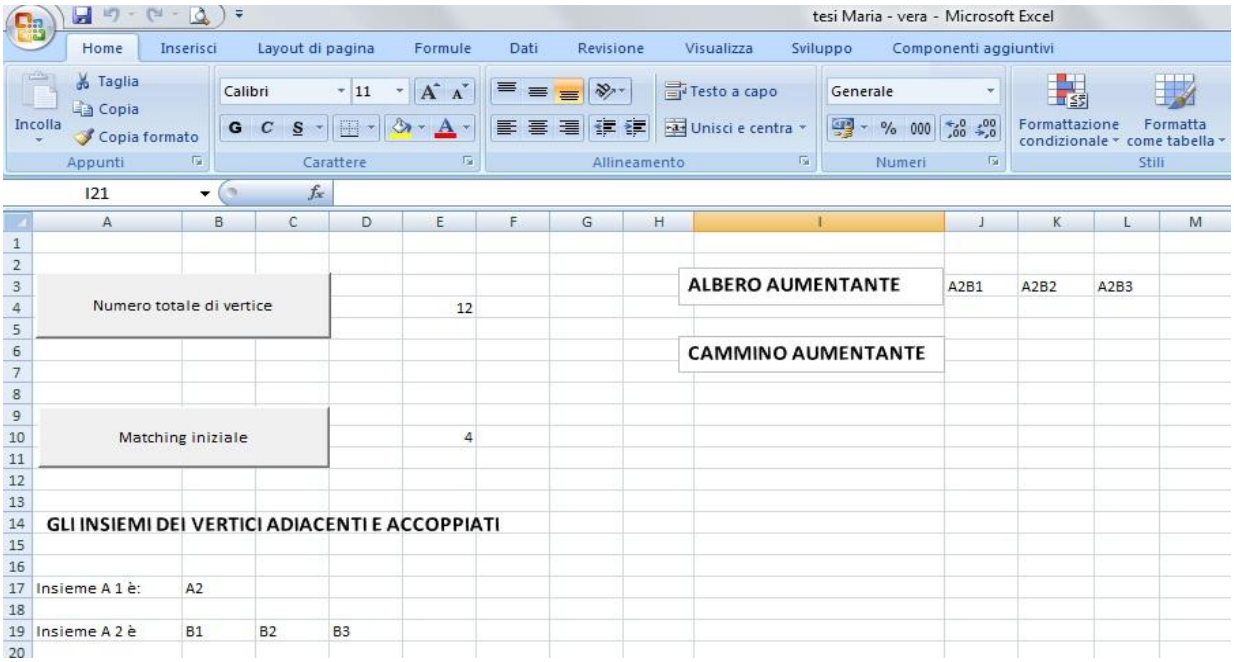
Le due variabili che si vedono all'inizio vengono solamente usate come indicatori di riga e di colonna sulle quali voglio iniziare a scrivere l'albero aumentante.

Il blocco di istruzioni che ora vediamo, è costituito da un For con al suo interno un if contenente un For ed un If . Il primo For cerca, attraverso un If, nell'array Tr_Inseriti un valore True. Tr_Inseriti è l'array che tiene conto di tutti vertici inseriti nel albero aumentante. Se viene trovato un True, nella posizione relativa al valore della variabile R, la condizione del If risulta verificate e si eseguono le sue istruzioni. Tutto il blocco di istruzioni all'interno del primo If viene eseguito ogni qualvolta che trovo in una posizione, uguale al valore di R, dell'array Tr_Inseriti un valore True. Per esempio nel nostro caso: nella prima posizione, la posizione corrispondente ad R=0, dell'array Tr_inseriti si trova un False e non viene soddisfatta la condizione del comando If. Questo accada perche la posizione zero dell'array corrisponde al vertice a_1 e esso non fa parte del nostro albero aumentante. Poi R viene aumentato di un'unità e l'If controlla ancora. Con R=1 nella Tr_inseriti troviamo un True. Questo perche la posizione 1 è la posizione corrispondente al vertice a_2, che fa parte del nostro albero aumentante. A questo punto viene eseguita l'istruzione del primo If. Viene salvato il nome del vertice in una variabile nominata X. Nel nostro caso abbiamo X uguale ad a_2. La variabile X è stata dichiarata come string all'inizio, così che possa contenere un nome. L'istruzione dell'If trova nella prima riga e colonna uguale alla posizione R+2 della matrice di adiacenza sul foglio matrice il nome del vertice, appunto perche nella Matrice di adiacenza la prima riga è stata usata per scrivere i nomi dei vertici. Successivamente inizia un'altro For con al suo interno il secondo If. La variabile c viene aumentata di un'unità ogni qual volta che il For fa un

ciclo. Nella matrice Tr_Matrice, per la riga corrispondente al valore di R, scorre tutte le colonne finché non trova un “1” o un “M”.
La condizione dell’If è verificata sia se si trova un “M” che un “1”. Tr_Matrice contiene i collegamenti dei vertice contenuti nell’albero aumentante, cioè “1” e “M” ,inseriti nei passi precendenti, rappresentanti gli spigoli. In Y viene salvato il nome del vertice che si trova nella posizione corrispondente al valore di C. Alla fine sul foglio Risultati viene portata la coppia dei vertici.

Nel nostro caso la condizione del secondo If viene verificata per la prima volta quando C è uguale a sei, la posizione del vertice b_1. Quindi In Y viene salvato il nome b_1 e sul foglio Risultati viene scritto a_2, b_1. Dopodiche la variabile C viene aumentata di un’unità, diventando sette, dove in Tr_inseriti si trova un altro “1”. In Y allora viene salvato il nome del vertice b_2 e sul foglio Risultati viene scritto a_2, b_2. Con C uguale ad otto si trova l’ultimo vertice, chiamato b_3, ed il risultato viene portato sul foglio Risultati.
Il For continua ad aumentare la variabile C fino ad arrivare al valore di totv (uguale ad 11). Finito il secondo ciclo For, la variabile R del primo ciclo For viene aumentata di un unità e ripete tutto per la terza riga. Nel nostro caso, dopo avere trovato a_2, il primo If non ha più esito positivo.

Il foglio Risultati si presenta cosi:



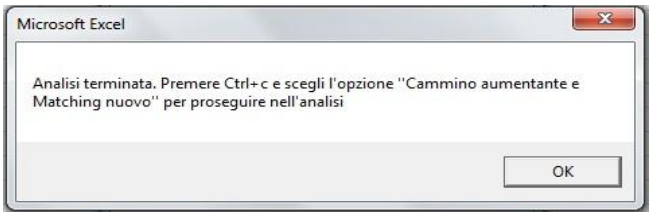
Vediamo l’Insieme A1 che è l’insieme dei vertici esposti, l’insieme A2 che è l’insieme dei vertici adiacenti al vertice esposto e l’albero aumentante. Poiché la variabile i aumenta ogni volta che inizio un nuovo passo dell’algoritmo, posso tenere conto dei vari insiemi: tutti i vertici contenuti negli insiemi A con indice dispari sono tutti outer e quelli contenuti negli insiemi con indice pari con tutti inner.

L’ultimo blocco di istruzioni non fa altro che portare sul foglio 3 la matrice Tr_Matrice. È solo una visualizzazione del contenuto di Tr_Matrice. È stato realizzata perché le variabili o array dichiarati dentro un Private sub() non possono essere richiamate dopo. Ciò significa che al di fuori dalla userform5 non esistono le sue variabili. Siccome mi serve Tr_Matrice nella Userform6 per calcolare il cammino aumentante, ho deciso di portarlo sul foglio 3 per poi riportarlo nella Userform6. Sono riuscita ad ottenere questo utilizzando un ciclo For annidato: sono presenti due

variabili contatore g e j. La prima viene usata per aumentare il numero delle righe e la seconda il numero delle colonne. Ogni volta che viene eseguito il For esterno, la variabile g viene aumentata di un'unità mentre la variabile j viene posta uguale ad zero. Pongo j uguale a zero, perché, rappresentando la posizione della colonna, voglio ad ogni nuova riga ricominciare dalla prima colonna. Nel For annidato, j viene aumentata di un'unità ogni volta che viene eseguito. Il For finisce quando le due variabili raggiungono il numero totale dei vertici meno uno (nel nostro caso uguale ad undici) .

```
MsgBox "Analisi terminata. Premere Ctrl+c e scegli l'opzione ''Cammino aumentante e "
Matching nuovo'' per proseguire nell'analisi"

UserForm5.Hide
UserForm1.Hide
Worksheets("Risultati").Select
Exit Sub
End Sub
```

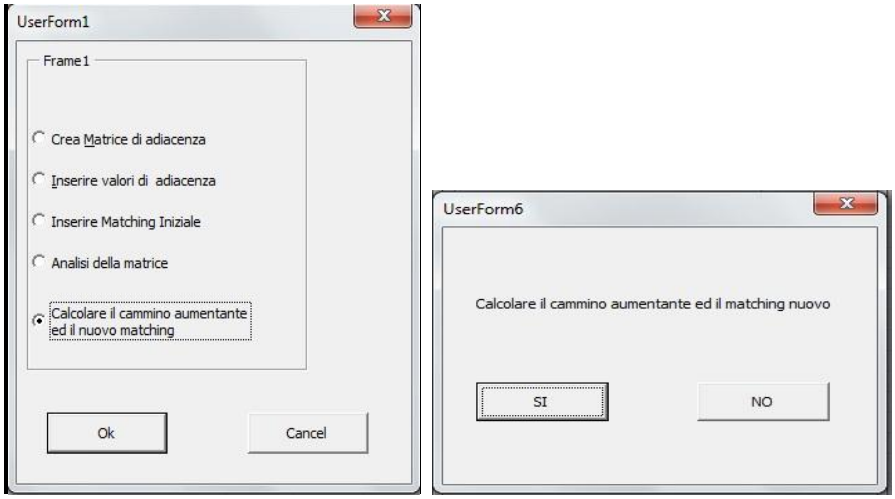
Le ultime istruzioni che vediamo fanno apparire un message box per aiutare l'utente a proseguire nell'analisi. La funzione message box è una funzione predefinita da VBA, che ha il solo scopo di mostrare un messaggio per l'utente, come nella figura seguente.



Troviamo poi le istruzioni per chiudere la Userform5 e la userform1. L'istruzione Worksheets("Risultati").select mostra la pagina che contiene i risultati, in modo che si possano valutare i risultati intermedi prima di proseguire nell'analisi.

Adesso dobbiamo calcolare il cammino aumentante ed il matching nuovo. Per fare ciò viene utilizzata la userform6. Come dice il Message box premendo ctrl+c appare nuovamente la Userform1.

Adesso però invece di scegliere la quarta opzione si deve scegliere la quinta, nominata "Calcolare il cammino aumentante ed il nuovo matching"



Cliccando ok appare la userform6. Quest'ultima è costituita da due bottoni di comando ed una casella di testo che chiede se si vuol calcolare il cammino aumentante ed il matching nuovo. Scegliendo il tasto no la Userform6 si chiude.

Il codice per calcolare il cammino aumentante ed il matching nuovo è contenuto dentro al bottone Si.

```
Private Sub SiButton1_Click()

Call SumVertice
totv = Worksheets("Risultati").Range("e4").Value - 1

Dim C As Integer
Dim R As Integer
Dim v As Integer
Dim v1 As Integer
Dim E As Integer
Dim RC As Integer
Dim CC As Integer
Dim j As Integer
Dim PaV As String

Dim TR_Matrice() As Variant
Dim Ins_R() As Integer
Dim PA() As Integer
Dim V_visti() As Boolean
Dim M_match() As String
Dim M1() As String
Dim M_match_nuovo() As String

ReDim Preserve M_match_nuovo(totv, totv) As String
ReDim M1(totv, totv) As String
ReDim M_match(totv, totv) As String
ReDim TR_Matrice(totv, totv) As Variant
ReDim Ins_R(totv) As Integer
ReDim Preserve PA(totv, totv) As Integer
ReDim Preserve V_visti(0, totv) As Boolean
```

Come nel Private Sub SiButton_Click() per la Userform5, anche qua sono inizialmente state dichiarate una serie di variabili, utilizzate sia come contatori che come array monodimensionali o multidimensionali, nei quali vengono salvati numeri,lettere o nel caso di un boolean True o False. Prima di tutto ho richiamato la funzione SumVertice, che avevamo già discusso all'inizio della Userform5, così che possa avere a disposizione il numero totale delle vertici. Nella variabile totv, che ha la stessa funzione della Userform5, viene salvato il numero totale dei vertici meno 1. (Sempre undici nel nostro caso)

```
Worksheets("foglio3").Activate
Worksheets("foglio3").Select

For R = 0 To totv
    For C = 0 To totv
        TR_Matrice(R, C) = Cells(R + 4, C + 1)
    Next C
Next R

For C = 0 To totv
    Ins_R(C) = Cells(1, C + 1)
Next C


Worksheets("Matrice").Activate
Worksheets("Matrice").Select

For R = 0 To totv
    For C = 0 To totv
      M1(R, C) = Cells(R + 2, C + 2)
    Next C
Next R

For R = 0 To totv
    For C = 0 To totv
        If M1(R, C) = "M" Then
        M_match(R, C) = "M"
        End If
    Next C
Next R
```

Il secondo blocco di istruzioni copia dal foglio3 la matrice Tr_Matrice e l'array Ins_R, dentro un'altra Matrice e array con gli stessi nomi delle precedenti. Dopodiché viene attivato e selezionato il foglio matrice, che contiene la matrice di adiacenza. Come nella userform precedente anche qua la matrice di adiacenza viene copiata dentro ad un matrice nominata M1. Siccome gli array Tr_matrice, Ins_R ed M1 hanno la stessa funzione e gli stessi valori come nella userform5 ho scelto di darli lo stesso nome, per sottolineare questo fatto.

L'ultimo For annidato viene usato per creare un array M_match che contiene solo il matching. Ciò tiene solo conto dei vertici che sono tra loro accoppiati. Il For cerca nella matrice M1 per ogni riga, scorrendo tutte le colonne. Se la condizione del If è verificata, cioè se viene trovato un "M", nell'array M_match viene aggiunto un "M". La "M" viene aggiunto nella stessa posizione in cui è stata trovata nel M1. Alla fine si ottiene un matrice adiacente con solo gli accoppiamenti assegnati.

Adesso possiamo passare al codice relativo al calcolo del cammino aumentante. Il calcolo del cammino aumentante viene effettuato con 3 blocchi di istruzioni, in modo che si possa ciclare tra loro. Il cammino aumentante è per definizione un cammino alternante, che è un catena elementare i cui spigoli appartengono alternativamente al maching e non, in cui sono esposti il primo e l'ultimo vertice. Il codice segue questa definizione del cammino aumentante.

Il primo blocco di istruzioni relativo al calcolo del cammino aumentante, chiamato CalcPA, trova un vertice esposto da cui iniziare a calcolare il cammino aumentante.

```
CalcPA:

RC = 6
CC = 9

j = 0
v = 0
For C = 0 To totv
   If Ins_R(C) = 2 And V_visti(0, C) = False Then
      v = C
      j = j + 1
      PaV = Worksheets("Matrice").Cells(1, C + 2)
      Worksheets("Risultati").Cells(RC, CC + j) = PaV
      Exit For
   End If
Next C


GoSub CalcPA1
Exit Sub
```

Le variabili RC e CC sono variabili integer, alle quali è stata assegnata rispettivamente la riga e la colonna sul foglio Risultati nelle quali voglio iniziare a scrivere il camino aumentante. La variabile j è un contatore che aumenta il numero della colonna ogni volta che deve essere scritto un nuovo vertice. Nella variabile v invece viene salvata la posizione del vertice esposto. Il ciclo For inizia come sempre contando da zero fino al numero totale dei vertici. La condizione dell'If si verifica quando nell'insieme dei vertici esposti viene trovato un 2 e che non sia ancora stato inserito nell'array V_visti. L'array V_visti è un array monodimensionale usato per tenere conto dei vertici con cui si è già provato a calcolare un cammino aumentante, ma non andato a buon fine. Torniamo a parlare di questo array alla fine dei blocchi di codice per il calcolo del cammino aumentante. Se la condizione dell'If è vera, allora nella variabile v viene salvata la posizione corrispondente al valore di C del vertice esposto e j viene aumentata di uno. Nella variabile PaV, dichiarata string, viene salvato il nome del vertice esposto. Il nome viene trovato sulla Matrice di adiacenza, sul foglio Matrice, sulla prima riga e nella colonna C+2. Ci ricordiamo che le colonne ed righe della matrice di adiacenza, sono spostati di due rispetto all'array M1. Successivamente sul foglio Risultati viene riportato il nome del vertice esposto. Alla fine l'istruzione Exit For arresta il ciclo For. Grazie all'Exit For il ciclo For si arresta subito dopo avere trovato il primo vertice esposto nel insieme dei vertici esposti. Nel nostro caso avendo il Ins_R uguale ad (a_2, a_4, b_3, b_6), la condizione dell'If viene verificata con C =1. Quindi alla variabile v viene assegnato il valore uno. Nel PaV viene salvato il nome trovato sul foglio matrice nella cella (1,3) che appunto corrisponde ad a_2. Sul foglio Risultati viene puoi scritto a_2 e il ciclo termina. Il Gosub CalcPa1 porta ad eseguire il blocco di istruzioni denominate CalcPa1

```
CalcPA1:

E = 0
For R = 0 To totv
    If TR_Matrice(v, R) = "1" And V_visti(0, R) = False Then
       PA(v, R) = 2
       PA(R, v) = 2
       v1 = R
       E = E + 1
       j = j + 1
       PaV = Worksheets("Matrice").Cells(1, R + 2)
       Worksheets("Risultati").Cells(RC, CC + j) = PaV
           If Ins_R(v1) = 2 Then
              GoSub matching
              Exit Sub
           End If
    Exit For
    End If
Next R


If E = 0 Then
   V_visti(0, v) = True
     For C = 0 To totv
       PA(C, v) = "0"
       PA(v, C) = "0"
    Next C
    GoSub CalcPA
    Exit Sub
End If

GoSub CalcPA2
Exit Sub
```

Il secondo blocco di istruzioni per il calcolo del cammino aumentante trova il successivo vertice del cammino. Un cammino aumentante è un camino alternante con il vertice iniziale e finale esposti. In un cammino alternante gli spigoli appartengono alternativamente a M e a E\M. Con M uguale al matching ed E uguale agli spigoli presenti nel grafo. Utilizzando questa definizione ho strutturato i vari passaggi del mio programma. Siccome il cammino aumentante parte da un vertice esposto, il vertice collegato a lui può solo essere un vertice adiacente, ed un vertice esposto per definizione è un vertice che non appartiene ad un matching. Nell'For, l'If cerca nell'TR_Matrice un "1". In TR_Matrice viene solo cercato un "1" nella riga corrispondente alla posizione del vertice esposto trovato prima, e salvato in v. Se in una colonna viene trovato un "1" e il vertice corrispondente a tale colonna non è già stato analizzato, viene salvata un 2 nell'array PA in corrispondenza di tale riga e colonna. PA è un array multidimensionale, anche detto matrice, con tante righe e colonne quante sono i vertici della matrice di adiacenza. Nell'array PA viene creata una matrice di adiacenza contenente i vertici del cammino aumentante ed i loro spigoli. Siccome voglio creare una matrice di adiacenza viene assegnato un 2 sia nella posizione (R,v) che in (v,R). Alla variabile v1 viene assegnata la posizione del vertice adiacente. Si trova il nome di tale vertice e viene riportato sul foglio risultati. E' presente un'altra variabile all'interno dell'If, j che è un variabile contatore. Essa viene aumentata di un'unità ogni volta che si trova un vertice adiacente al vertice precedentemente salvato in v. Alla fine troviamo un If, annidato nel primo If. Ha la funzione di verificare se il vertice trovato è un vertice esposto, perché quando viene trovato un vertice esposto il calcolo del cammino aumentante si ferma e viene calcolato il matching nuovo. L'If vede se nell'insieme dei vertici esposti c'è il vertice appena trovato, e se si avvia il blocco di istruzioni contenuto nel matching e termina il sub CalcPA1. L'istruzione Exit For all'interno del primo If, ha lo scopo di uscire dal ciclo For subito dopo avere trovato un vertice adiacente. La programma sceglie sempre il primo vertice adiacente al vertice precedente.

Il blocco di istruzioni relativo all'ultimo If viene solamente eseguito se non è stato trovato nessun vertice adiacente al vertice salvato in v. Cioè se la variabile E rimane uguale ad zero vuol dire che

nell' albero aumentante non c'è un vertice adiacente al precedente, perciò il Gosub CalcPA fa eseguire nuovamente il blocco di istruzione nominato CalcPA.

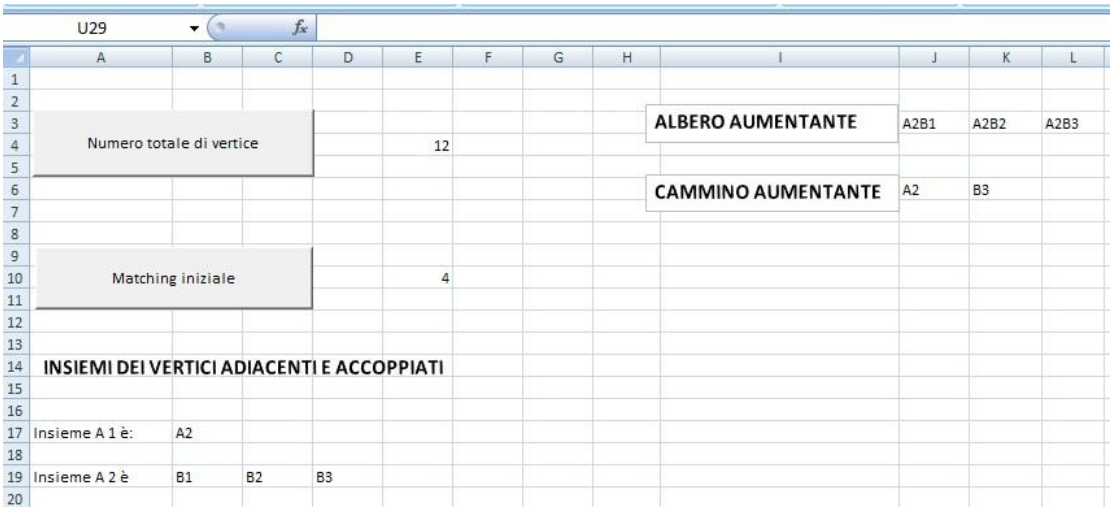
Nel nostro esempio in v è stata salvata la posizione del vertice a_2. Cercando un "1" nella Tr_Matrice il primo trovato è b_1. La posizione del $b_1$viene salvata nella variabile v1. Siccome $b_1$non è un vertice esposto il For viene terminato ed il programma avvia l'istruzione Gosub CalcPA2 che fa seguire il prossimo blocco di istruzioni.

```
CalcPA2:

 E = 0
For R = 0 To totv
    If TR_Matrice(v1, R) = "M" And V_visti(0, R) = False Then
       PA(v1, R) = 2
       PA(R, v1) = 2
       v = R
       E = E + 1
       j = j + 1
       PaV = Worksheets("Matrice").Cells(1, R + 2)
       Worksheets("Risultati").Cells(RC, CC + j) = PaV
     End If
Next R


If E = 0 Then
   V_visti(0, v1) = True
      For C = 0 To totv
         PA(C, v1) = "0"
         PA(v1, C) = "0"
      Next C
GoSub CalcPA
Exit Sub
End If

GoSub CalcPA1
Exit Sub
```

Come detto sopra un cammino aumentante varia tra spigoli appartenenti al matching e non appartenenti. Quindi dopo avere trovato un vertice adiacente al vertice esposto, dobbiamo trovare un vertice accoppiato al vertice adiacente. Il For funziona in modo simile a quello visto nel CalcPA1, ma invece di cercare un "1" cerca un "M", proprio perché voglio uno spigolo appartenente al matching. Se lo trova viene salvo un 2 nell'array PA, analogamente al CalcPA1. Alla variabile v viene ora assegnata la posizione del vertice accoppiato al vertice precedente. È stato usato la stessa variabile come al CalcPA, questo perché dopo avere trovato un vertice accoppiato al precedente si ritorna al blocco di istruzioni nominato CalcPA1. La variabile E viene aumentata di uno. Si trova il nome del vertice sul foglio matrice e viene riportato il risultato sul foglio Risultati. L'If viene solamente eseguita se E risulta uguale a zero, cioè se nella Matrice Tr_Matrice non esiste un M nella riga corrispondente alla posizione salvata in v1. Se l'If è verificato nell'array V_visti viene salvato un True. Così si può tenere conto dei vertici con i quali non si è riusciti a trovare un vertice accoppiato. Poi il For all'interno dell'If cancella la riga e colonna in cui è stato inserito un "2". Dopodiché si ritorna al CalcPA è si prova con un'altro vertice esposto. Se viene trovato un vertice accoppiato al precedente si ritorna al CalcPA1. Il programma continua a passare ad un altro finché non trova un'altro vertice esposto.

Nel nostro esempio, come detto prima, nel CalcPA è stato scelto a_2, perché è il primo vertice contenuto nell'insieme dei vertici esposti. Nel CalcPA1 è stato trovato nell'albero aumentante b_1, perche b_1è il primo dei vertici adiacenti ad a_2. In PA viene inserito un "2" nelle celle (1,6) e (6,1). b_1 non è esposto, quindi andiamo al CalcPA2. Qua cerca nell'albero se c'è un vertice accoppiato al b_1. Siccome il nostro albero aumentante è $T_r = \{(a_2, b_1)(a_2, b_2)(a_2, b_3)\}$ non esiste nessun vertice accoppiato a b_1. Allora nell'array V_visti viene assegnato un True nella posizione 6 e con il For vengono cancellate la riga e colonna in cui prima era stato inserito un "2". Ora si passa al CalcPA. Qua si prende ancora il vertice esposto a_2. Si passa al CalcPA1 che trova il secondo vertice

adiacente ad a_2, che è b_2. Ora si passa al CalcPa2. Nella Matrice dell'albero aumentante non viene trovato neanche per b_2, un "M". Quindi nel V_visti viene assegnato un True nella posizione relativa al vertice b_2 , la posizione 7, e si torna al CalcPA. In CalcPa viene sempre preso il vertice a_2. In CalcPA1 viene ora scelto il vertice b_3 . Tale vertice è un vertice esposto. Quindi si passa al blocco di istruzioni nominate matching e si termina la sub CalcPA1. Dopo tutti questi cicli sul foglio risultato viene scritto il cammino aumentante:



Una volta trovato il cammino aumentante si può allora calcolare il matching nuovo. Il mio codice relativo al calcolo del matching esegue la funzione data dall'algoritmo, M = $M \backslash P_A \cup P_A \backslash M$. Avendo ottenuto una matrice di adiacenza con i valori del cammino aumentante e una matrice con il matching attuale, si può sottrarre una dall'altra. Vediamo il codice.

```
matching:

For R = 0 To totv
   For C = 0 To totv
       If M_match(R, C) = "M" And PA(R, C) <> "2" Then
          M_match_nuovo(R, C) = "M"
            ElseIf PA(R, C) = "2" And M_match(R, C) <> "M" Then
               M_match_nuovo(R, C) = "M"
       End If
   Next C
Next R


For R = 0 To totv
    For C = 0 To totv
        If M_match(R, C) = "M" Then
           Worksheets("matrice").Cells(R + 2, C + 2).Value = "1"
        End If
        If M_match_nuovo(R, C) = "M" Then
           Worksheets("matrice").Cells(R + 2, C + 2).Value = "M"
        End If
    Next C
Next R



MsgBox "Matching nuovo trovato, per verificare se è di cardinalità massima: "
Premere ctrl+c e scegli l'opzione ''Analise della matrice'' "
UserForm6.Hide
UserForm1.Hide
Worksheets("Risultati").Select

Exit Sub
End Sub
```

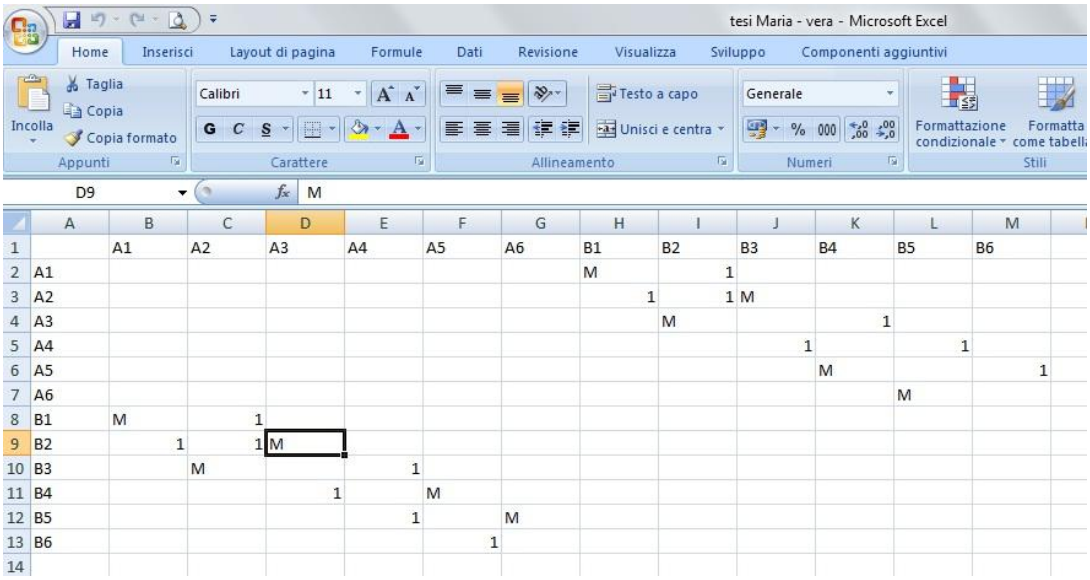
Il For annidato controlla nelle matrici, contenenti il matching corrente ed il cammino aumentante, tutte le celle. L'if viene verificato se è solo se nel M_match si trova un "M" e se nella stessa posizione, in cui è stato trovato un "M", della matrice PA non ce un 2. Se tale è vera nella matrice nominata M_match_nuovo viene assegnato un "M". M_match_nuovo è una matrice adiacente contenente il matching nuovo. Questo realizza M\PA, cioè trova le coppie di vertici che appartengono al matching, ma che non sono presenti nel cammino aumentante. Se l'If non viene

verificato, e solo se non viene verificato, passa all'istruzione Else If. L'istruzione Else If viene verificata ogni qual volta che nella matrice del cammino aumentante si trova un 2 e nella matrice del matching corrente non è presente un "M" nella stessa posizione. Anche qua se la condizione è vera, in M_match_nuovo viene assegnato un "M" nella posizione corrispondente. Questo realizza PA\M, cioè trova le coppie di vertici che appartengono al cammino aumentante, ma che non fanno parte al matching. Dopo questo ciclo For, in M_match_nuovo è così stato salvato il matching nuovo. Il For termina quando è stata controllata ogni riga e colonna delle matrici.

Il secondo For annidato che troviamo ha una struttura simile al precedente. La sua funzione è quella, attraverso due If, di cancellare il matching esistente e riportare il matching nuovo nella matrice di adiacenza sul foglio matrice. Il primo If cerca nella matrice con il matching corrente. Ogni volta che trova un "M", la posizione corrispondente sul foglio matrice viene sostituito con un "1". Lo sostituisco con un "1" perche lo spigolo tra i due vertice non viene tolto, viene solo tolto il matching.

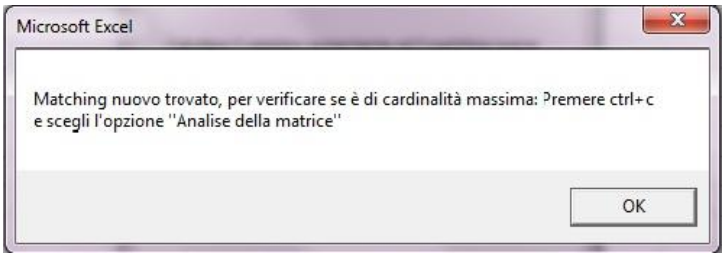
Il secondo If cerca nella matrice in cui è stato salvato il matching nuovo. Quando viene trovato un "M" nell' M_match_nuovo, sul foglio Matrice, nella stessa posizione, viene scritto un "M". Cosi sul foglio matrice ho ottenuto il nuovo Matrice di adiacenza.

Nel nostro esempio abbiamo $P_A = (a_2, b_3)$ e $T_r = \{(a_2, b_1)(a_2, b_2)(a_2, b_3)\}$. Quindi utilizzando $M = M \backslash P_A \cup P_A \backslash M$. Si trova il matching nuovo $M' = \{(a_1, b_1), (a_2, b_3), (a_3, b_2), (a_5, b_4), (a_6, b_5)\}$. Nel foglio matrice vengono allora sovrascritti il matching corrente e inserito quello nuovo. Qua il matching nuovo è uguale a quello precedente con l'aggiunta di (a_3, b_2). Quindi come si vede in figura nella cella tra i due vertici viene assegnato un "M". Si ha cosi ottenuto un nuova matrice di adiacenza, su quale si vuole provare se il matching trovato è di cardinalità massima. Se non lo è si cerca un altro matching.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1		A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6
2	A1							M	1				
3	A2							1	1	M			
4	A3								M		1		
5	A4									1		1	
6	A5										M		1
7	A6											M	
8	B1	M		1									
9	B2	1		1	M								
10	B3		M		1								
11	B4			1		M							
12	B5				1		M						
13	B6					1							
14													

A questo punto appare un message box che dice all'utente come proseguire.



Si fa partire la userform1 di nuovo è si sceglie ancora una volta la quarta opzione. Scegliendo l'opzione "Analisi della matrice" si fa apparire di nuovo la userform5.

Il programma parte,come spiegato sopra, con il passo 2: trovando il nuovo insieme delle vertice esposti, controlla se il matching è di cardinalità massima. Se lo è il programma si arresta, altrimenti viene selezionata il primo vertice esposto dell'insieme R e passa la Passo 3. Nel nostro esempio avendo trovato un nuovo matching $M' = \{(a_1, b_1), (a_2, b_3), (a_3, b_2), (a_5, b_4), (a_6, b_5)\}$ l'insieme dei vertici esposti si è ridotto a $R = (a_4, b_6)$. Il matching trovato non è di cardinalità massima, perché R> 1. Il programma sceglie il primo vertice dell'insieme R, che è a_4. Si attribuisce etichetta O al a_4. Sul foglio Risultati viene riportato il nome del vertice e si passa al Passo3. Al passo 3 si trovano due vertici b_3, b_5 adiacenti a a_4. Quindi l'insieme dei vertici adiacenti non è vuoto. Diversamente dalla prima interazione, ne b_3 ne b_5 sono vertici esposti. Quindi dopo averli etichettati con I e riportati sul foglio risultati, viene eseguito il blocco di istruzioni nominato Passo4, che prima avevamo saltato.

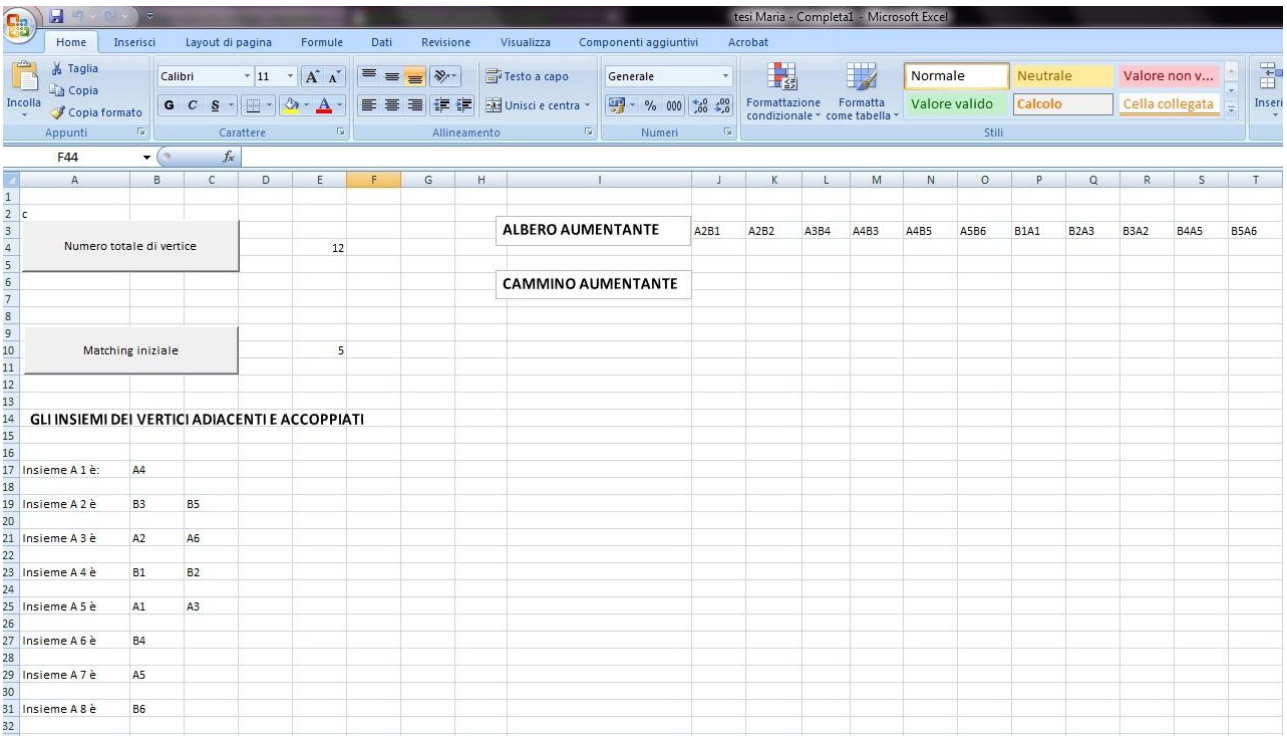
```
Passo4:

i = i + 1

For C = 0 To totv
    Ins_A3(C) = 0
Next C

For R = 0 To totv
    If Ins_A1(R) = 2 Then
        For C = 0 To totv
            If M1(R, C) = "M" And Tr_Inserito(0, C) = False Then
               Ins_A3(C) = 2
               TR_Matrice(R, C) = "M"
            End If
        Next C
    End If
Next R


RA = RA + 2

j = 0
For C = 0 To (totv)
    If Ins_A3(C) = 2 Then
       j = j + 1
       V_Etichetta(0, C) = "O"
       Tr_Inserito(0, C) = True
       InsA = Worksheets("Matrice").Cells(1, C + 2)
       Worksheets("Risultati").Cells(RA, CA - 1) = "Insieme A " & i & " è "
       Worksheets("Risultati").Cells(RA, CA - 1 + j) = InsA
    End If
Next C

GoSub Passo3
Exit Sub
```

Prima viene aumentata la variabile i, discussa in precedenza, di un'unità. La variabile i sarà sempre di numero dispari, così si può riconoscere quale sia l'insieme dei vertici accoppiati. Poi viene cancellato il contenuto del Ins_A3. L'array Ins_A3 è lo stesso che è stato usato al Passo2 per salvare il vertice esposto. Deve essere lo stesso perché, come vedremo, dopo il passo4 si torna al passo3.

I due For del passo4 funzionano similarmente a quelli del passo3, infatti hanno la stessa struttura. Il primo For annidato trova i vertici accoppiati, mediante spigoli in M, a vertici di Ins_A1. Dove Ins_A1 è l'insieme dei vertici adiacenti, trovati al passo3. Il For esterno viene usato dal primo if, che cerca nell'Ins_A1 se trova un uno. Se la condizione non viene verificata la variabile R viene aumentata di un'unità e si controlla la prossima posizione. Quando invece la condizione dell'If ha esito positivo, nell'Ins_A1 viene trovato un vertice nella posizione corrispondente al valore di R e viene effettuato il secondo For ed If. Si cerca nella matrice M1 un "M" che non sia già stata inserito nell'albero aumentante. Quando l'If ha esito positivo, nell'Ins_A3 viene inserito un 2 nella posizione C. Nell'Ins_A3 vengono cosi salvati i vertici accoppiati. A differenza del Passo3 nel Tr_Matrice viene salvato un "M" nella posizione (R,C).

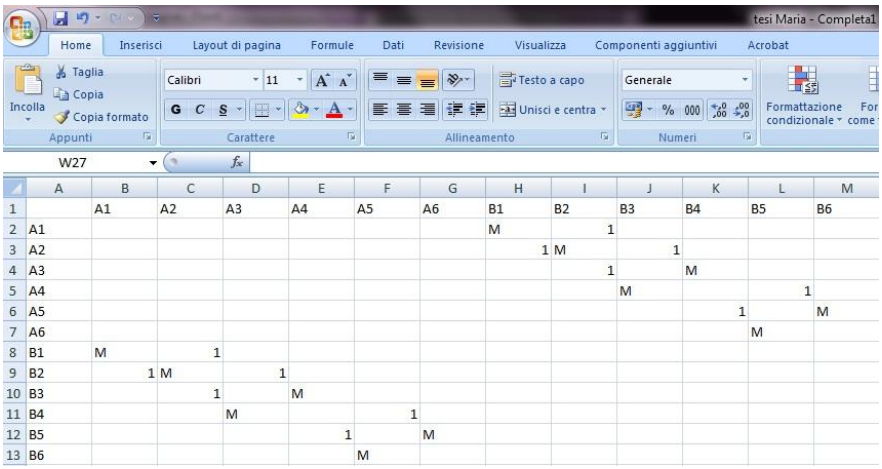
Il successivo For, con al suo interno un If, etichetta tutti i vertici nell'Ins_A3 con un O. Inserisce nell'albero aumentante tutti i vertici e riporta sul foglio risultati i vertici trovati. Gosub Passo3 fa eseguire il blocco di codice scritto nel Passo3. Come nell'algoritmo dopo il Passo4 si ritorna a fare il Passo 3.

Nel nostro caso con il Passo 4 si trovano due vertici adiacenti $a_2, a_6.$, dove a_2 è accoppiato con b_3 e a_6 è accoppiato con b_5. L'insieme dei vertici accoppiati è $A_3 = (a_2, a_6)$. L'albero aumentante diventa $T_r = \{(a_4, b_3), (a_4, b_5), (b_3, a_2), (b_5, a_6)\}$. Dopo si ritorna al Passo 3. Il programma continua a passare tra il Passo 3 ed il Passo 4, finché non trova un vertice esposto. Dopo 3 cicli viene trovato un vertice esposto b_6. Il foglio Risultati si presenta così:



Qui vediamo l'insieme A_i , trovato nei vari passaggi intermedi, e l'albero aumentante. Avendo trovato un vertice esposto, appare il message box che dice che ora si può calcolare il cammino aumentante ed il matching nuovo. Quindi scegliendo la quinta opzione nell'userform1, si fa partire l' userform6 di nuovo. Esso calcola come prima il cammino aumentante e il matching nuovo. Alla fine riporta sul foglio risultati il cammino aumentante e sostituisce nella matrice di adiacenza il matching nuovo.

Il foglio Risultati e Matrice appaiono così:



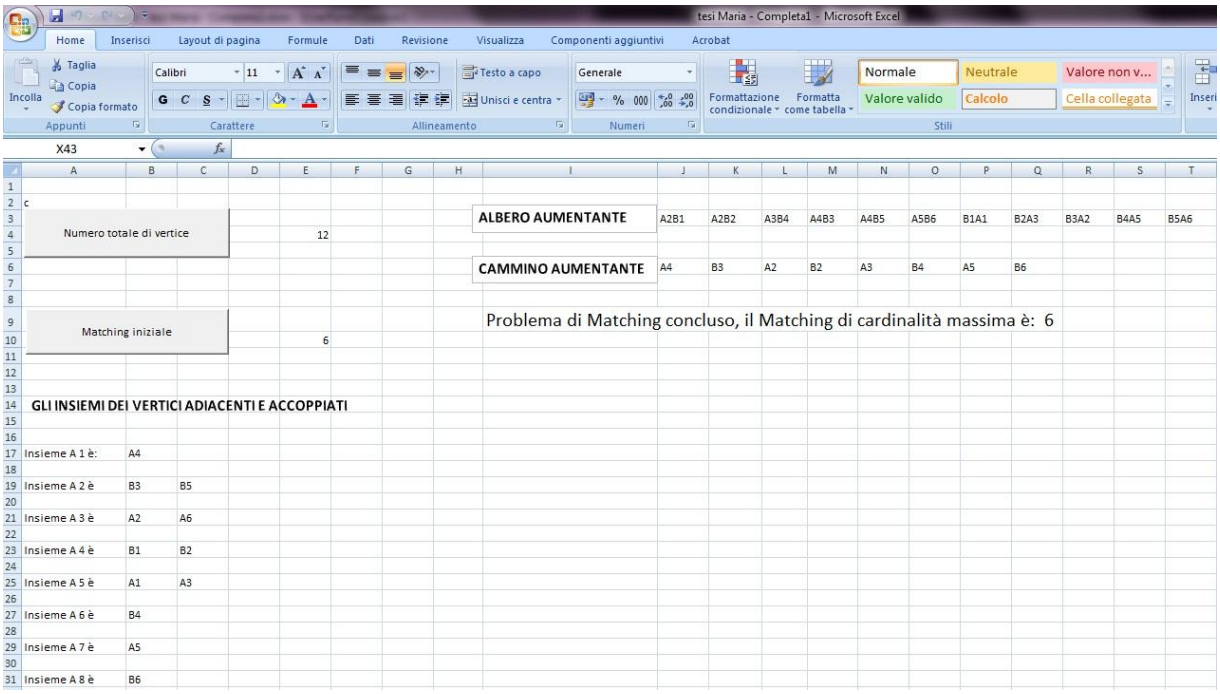
	A	B	C	D	E	F	G	H	I	J	K	L	M
1		A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6
2	A1							M	1				
3	A2							1	M		1		
4	A3								1		M		
5	A4									M		1	
6	A5										1		M
7	A6											M	
8	B1	M		1									
9	B2		1	M		1							
10	B3			1		M							
11	B4				M		1						
12	B5					1		M					
13	B6					M							

Si vede il matching nuovo $M'' = (a_1, b_1), (a_2, b_2), (a_3, b_4), (a_4, b_3), (a_5, b_6), (a_6, b_5)$ inserito nel matrice di adiacenza.

Sul foglio Risultati si vede ora anche il cammino aumentante.

ALBERO AUMENTANTE	A2B1	A2B2	A3B4	A4B3	A4B5	A5B6	B1A1	B2A3	B3A2	B4A5	B5A6
CAMMINO AUMENTANTE	A4	B3	A2	B2	A3	B4	A5	B6			

Ora siamo pronti per provare a verificare se il matching trovato è di cardinalità massima. Scegliendo ancora una volta la quarta opzione e facendo partire il codice relativo alla userform5. Sul foglio risultati appare la scritta che il matching è di cardinalità massima ed il programma si arresta. Abbiamo così risolto il nostro esercizio. Sulla schermata finale del foglio risultato vengono mostrati tutti i passaggi intermedi, l'albero aumentante, il cammino aumentante e la cardinalità massima.



CONCLUSIONE

Questa tesi ha voluto presentare ed analizzare un procedura realizzata in linguaggio VBA per la risoluzione di un problema di cardinalità massima in un grafo bipartito.
Ciò che si è voluto evidenziare è l'autonomia del processo seguito del programma, infatti l'intervento dell'utente utilizzatore è limitato solo all'inserimento dei dati iniziali (i vertici, gli spigoli ed il matching iniziale) ed alla selezione dell'operazione da far eseguire al programma (sequenza che però è obbligata).
Considerando questo fatto la procedura in linguaggio VBA è stata realizzata in modo che i vari passaggi dell'algoritmo venissero gestiti automaticamente del programma, infatti questo trova da solo un vertice esposto, trova da questo un matching nuovo, controlla se è di cardinalità massima; se non lo è in automatico parte ad analizzare un altro vertice esposto, fino a dare il risultato finale del matching di cardinalità massima.
In tutta questa elaborazione li programma comunica all'utente i vari risultati intermedi (albero aumentante, cammino aumentante, insieme dei vertici adiacenti e accoppiati), richiedendo a questo solo una conferma per proseguire.
In conclusione questa procedura agevola l'utente nella risoluzione del problema, sostituendo completamente il metodo di risoluzione analitico.
Abbiamo concluso questa digressione sulla risoluzione dei problemi di accoppiamento di cardinalità massima in grafi bipartiti e sulla procedura relativa realizzata in linguaggio VBA.

Bibliografia

Matteo Fischetti. Lezioni di Ricerca operativa. Edizioni Libreria Progetto Padova 1999. Pagine 111-119

John Green, Stephen Bullen, Rob Bovey, Michael Alexander. Excel 2007 VBA, Programmer's Reference. Wiley Publishing, Inc.

John Walkenbach. Excel 2007 Power Programming with VBA. John Wiley & Sons 2007

Giovanni Andreatta, Francesco Mason, Giorgio Romanin Jacur. Ottimizzazione su reti. Edizione Libreria Progetto Padova 1996. Pagine 115-148