OTTIMIZZAZIONE COMBINATORIA

Prof. Marco Trubian
6 CFU

Luca Cappelletti

Lecture Notes
Year 2017/2018

Magistrale Informatica
Università di Milano
Italy
9 dicembre 2018

Indice

1 Introduzione

L' **Ottimizzazione combinatoria** propone modelli di soluzioni ad innumerevole problemi, tra i quali vi sono:

Matching covers Consideriamo due insiemi A e B, di cardinalità n: ad ogni coppia di valori del prodotto cartesiano dei due insiemi è associato un valore positivo che descrive la compatibilità tra i due valori. Si vanno a scegliere n coppie, senza che gli elementi vengano ripetuti, in modo da massimizzare la compatibilità totale.

Set Covering Data una *matrice binaria* ed un vettore di costi associati alle colonne si va a realizzare il sottoinsieme di costo minimo che copra tutte le righe.

Set Packing Data una *matrice binaria* ed un vettore di valori associati alle colonne, si cerca il sottoinsieme di colonne di valore massimo tali che non coprino entrambe una stessa riga.

Set Partitioning Data una *matrice binaria* ed un vettore di cosi associati alle colonne, si cerca il sottoinsieme di colonne di costo minimo che copra tutte le righe senza conflitti.

Vertex Cover Dato un grafo non orientato $G = (V, E)$ si cerca il sottoinsieme di vertici di cardinalità minima tale che ogni lato del grafo vi incida.

Maximum Clique Problem Dato un grafo non orientato e una funzione peso definita sui vertici, si cerca il sottoinsieme di vertici fra loro adiacenti di peso massimo.

Maximum Independent Set Problem Dato un grafo non orientato e una funzione di peso definita sui vertici, si cerca il sottoinsieme di vertici fra loro non adiacenti di peso massimo.

Minimum Steiner Tree Dato un grafo non orientato e una funzione costo definita sui lati, si cerca un albero ricoprente di costo minimo.

Boolean satisfiability problem or SAT Data una forma normale congiunta (CNF), si cerca un assegnamento di verità alle variabili logiche che la soddisfi.

Versione pesata (MAX-SAT) Viene considerata anche una funzione peso associata alle formule che compongono la CNF. L'obbiettivo è massimizzare il peso totale delle formule soddisfatte.

2
Matching Covers

2.1 Matching

Definizione 2.1.1 (Matching o Accoppiamento). Dato un grafo $G = (V, E)$, un **matching** è un sottoinsieme $M \subseteq E$ di archi *a due a due non adiacenti.*

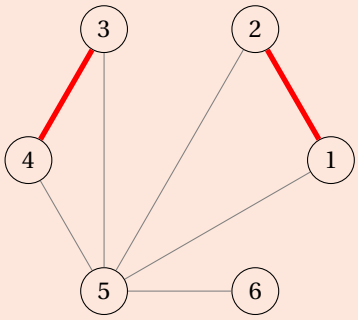


Figura 2.1: Un esempio di matching, in rosso e grassetto.

Definizione 2.1.2 (Matching massimo). Matching M^* di cardinalità massima.

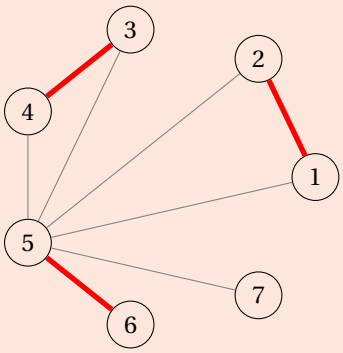


Figura 2.2: Un esempio di matching massimo, in rosso e grassetto.

Definizione 2.1.3 (Matching bipartito). Se il grafo G è **bipartito**, allora anche M si dice **bipartito**.

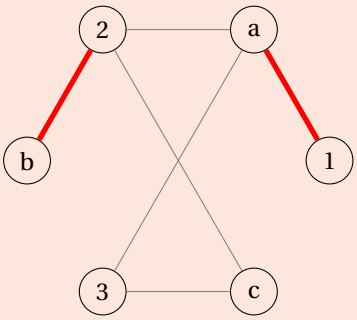


Figura 2.3: Un esempio di matching bipartito, in rosso e grassetto.

Definizione 2.1.4 (Matching perfetto). Se la cardinalità del matching è pari a metà del numero di vertici, allora si dice **perfetto**:

$$|M| = \frac{|V|}{2}$$

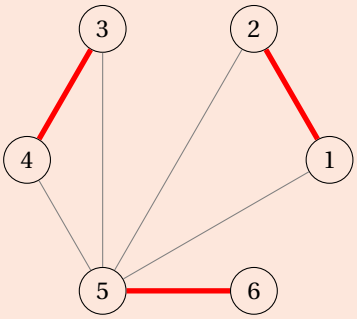


Figura 2.4: Un esempio di matching perfetto, in rosso e grassetto

Definizione 2.1.5 (Matching massimale). Un matching M si dice **massimale** se ogni elemento di $E \setminus M$ è adiacente ad almeno un elemento di M.

Un matching massimale **non** necessariamente è massimo, mentre un matching massimo è sempre massimale.

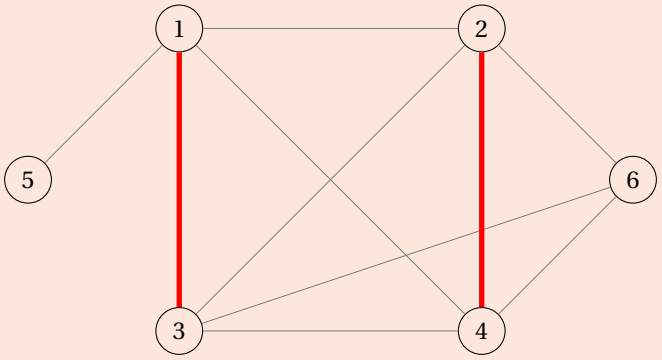


Figura 2.5: Esempio di matching massimale, in rosso e grassetto

2.2 Insieme stabile

Definizione 2.2.1 (Insieme stabile o indipendente). Dato un grafo simmetrico $G = (V, E)$, un qualunque sottoinsieme S di vertici si dice **indipendente** o **stabile** se esso è costituito da elementi *a due a due non adiacenti*.

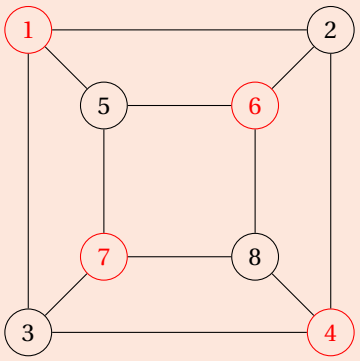


Figura 2.6: Esempio di insieme stabile

Definizione 2.2.2 (Insieme stabile massimo). Un insieme stabile S^* si dice **massimo** se $|S^*| \geqslant |S|$, per ogni insieme stabile S di G.

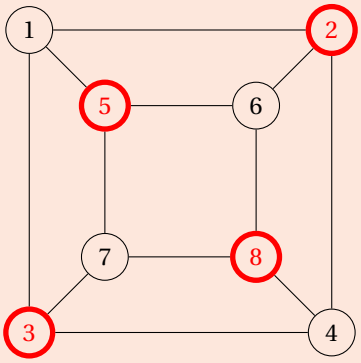


Figura 2.7: Esempio di insieme stabile, in rosso e grassetto

Definizione 2.2.3 (Insieme stabile massimale). Un insieme stabile S si dice **massimale** se ogni elemento di $V \setminus S$ è adiacente ad almeno un elemento di S.

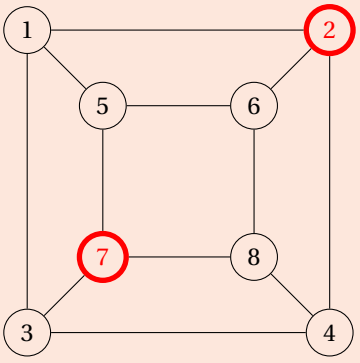
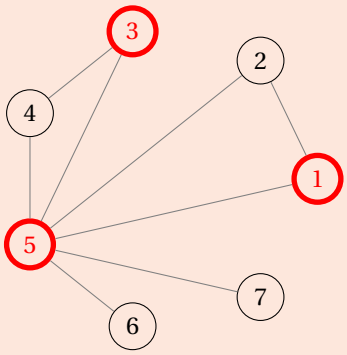


Figura 2.8: Esempio di insieme stabile, in rosso e grassetto

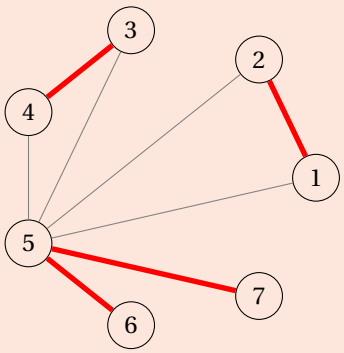
2.3 Copertura

Definizione 2.3.1 (Copertura). Dato un grafo simmetrico $G = (V, E)$, un qualunque sottoinsieme T di vertici *(F di archi)* tale che ogni arco di E *(vertice di V)* incide su almeno un elemento di T *(di F)* si dice **copertura**.

In particolare, l'insieme T è detto **insieme trasversale** o **vertex cover** mentre l'insieme F è detto **edge cover**.



(a) Vertex cover, in rosso



(b) Edge cover, in rosso

Figura 2.9: Esempi di edge e vertex cover

Definizione 2.3.2 (Copertura minima). Una copertura X^* si dice **minima** se:

$$\left|X^*\right| \leq |X| \quad \forall X \in G$$

Dove X sono gli insiemi di copertura del grafo G.

Definizione 2.3.3 (Copertura minimale). Una copertura X si dice **minimale** se $X \setminus \{x\}$ non è una copertura per ogni $x \in X$.

2.4 Disuguaglianze duali deboli

Teorema 2.4.1 (Disuguaglianze duali deboli). Definiamo per un grafo G:

$\alpha(G)$ la cardinalità dell'**insieme stabile massimo** del grafo G.

$\mu(G)$ la cardinalità del **matching massimo** del grafo G.

$\rho(G)$ la cardinalità dell'**edge cover minimo** di G.

$\tau(G)$ la cardinalità dell'**insieme trasversale minimo** di G.

Per un grafo G valgono le seguenti due disuguaglianze:

$$\alpha(G) \leqslant \rho(G)$$
$$\mu(G) \leqslant \tau(G)$$

L'**insieme stabile minimo di un grafo** è sempre di cardinalità minore dell'**edge cover minimo**. Inoltre, l'**insieme stabile massimo** è sempre minore o uguale dell'**insieme trasversale minimo**.

Dimostrazione 2.4.2 (Disuguaglianze duali deboli). Dato un grafo $G = (V, E)$, ne definiamo un **insieme stabile** X ed un **edge cover** Y.

Poiché Y è una *edge cover*, ogni elemento di X incide su almeno un elemento di y.

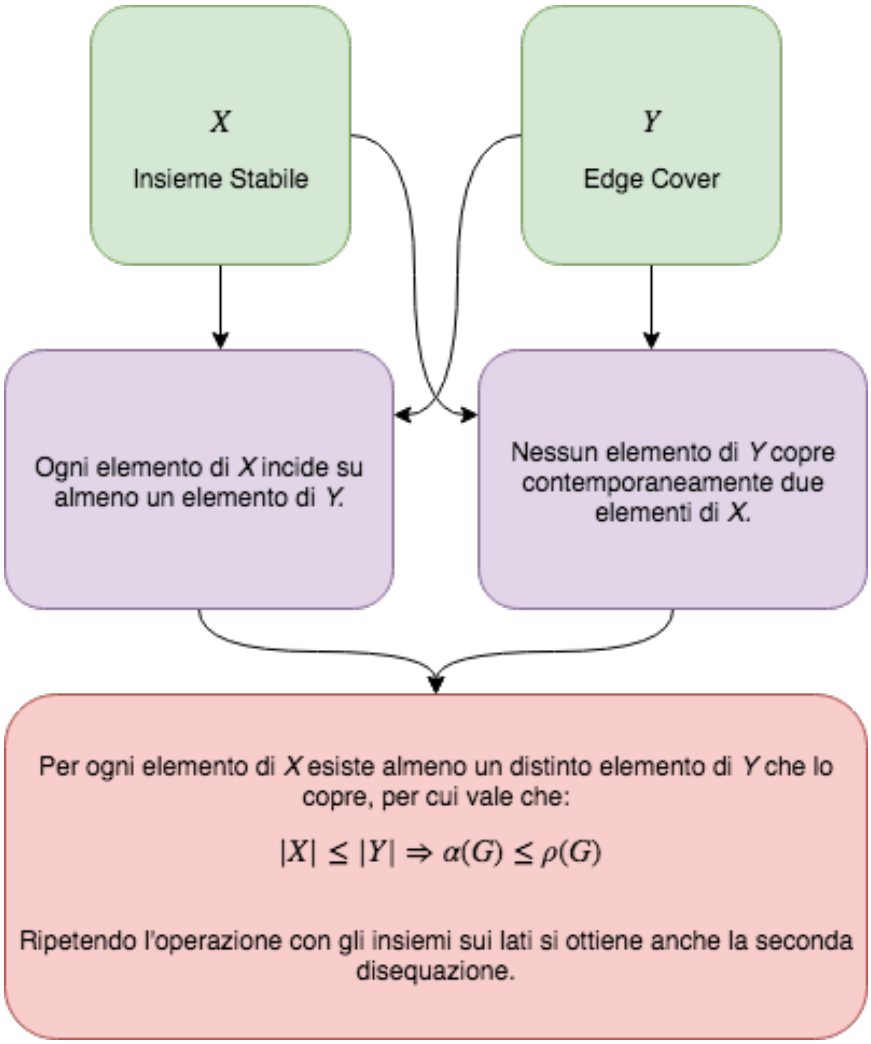
Nessun elemento di Y copre contemporaneamente due elementi di X, che altrimenti non sarebbe un *insieme stabile*.

Per ogni elemento di X esiste un distinto elemento di Y che lo copre, e quindi $|X| \leqslant |Y|$, che per gli insiemi ottimi X^* e Y^* risulta:

$$\alpha(G) \leqslant \rho(G)$$

Ripetendo l'operazione con gli insiemi sui lati E, si ottiene anche la seconda disequazione $\mu(G) \leqslant \tau(G)$. □

2.4.1 Diagramma della dimostrazione - Disuguaglianze duali deboli



2.5 Teorema di Gallai

Teorema 2.5.1 (Teorema di Gallai). Per ogni grafo G con n nodi si ha:

$$\alpha(G) + \tau(G) = n$$

Se inoltre G non ha nodi isolati

$$\mu(G) + \rho(G) = n$$

Cioè la somma della cardinalità di **insieme stabile massimo** e **insieme trasversale minimo** di un grafo G sono pari al numero dei suoi nodi.

Se non esistono nodi isolati, la stessa relazione vale anche per la somma della cardinalità del **matching massimo** e dell'**edge cover minimo** di un grafo G.

Dimostrazione 2.5.2 (Teorema di Gallai). *Iniziamo ottenendo la prima equazione:* sia S un **insieme stabile** di G. Allora $V \setminus S$ è un **insieme trasversale**.

In particolare, vale che $|V \setminus S| \geqslant \tau(G)$.

Se consideriamo l'insieme stabile massimo S^*, otteniamo:

$$\tau(G) \leqslant \left|V \setminus S^*\right| = n - \alpha(G) \quad \Rightarrow \quad \alpha(G) + \tau(G) \leqslant n$$

Analogamente, sia T un **insieme trasversale** di G. Allora $V \setminus T$ è un **insieme stabile**.

In particolare, $|V - T| \leqslant \alpha(G)$.

Se consideriamo l'insieme trasversale minimo T^*, otteniamo:

$$\alpha(G) \geqslant \left|V \setminus T^*\right| = n - \tau(G) \quad \Rightarrow \quad \alpha(G) + \tau(G) \geqslant n$$

Per le due disequazioni ottenute possiamo affermare che:

$$\alpha(G) + \tau(G) \leqslant n \quad \wedge \quad \alpha(G) + \tau(G) \geqslant n \quad \Rightarrow \quad \alpha(G) + \tau(G) = n$$

Procediamo a dimostrare la seconda equazione: sia G un grafo privo di nodi isolati e sia M^* il **matching massimo** di G. Indichiamo con V_{M^*} i nodi che sono estremi degli archi in M^*.

Sia H un **insieme minimale** di archi tale che ogni nodo in $V \setminus V_{M^*}$ è estremo di qualche arco in H. Segue che:

$$|H| = |V \setminus V_{M^*}| = n - 2\left|M^*\right|$$

Osserviamo che l'insieme $C = H \cup M^*$ è un edge-cover di G. Sicuramente, $|C| \geqslant \rho(G)$, quindi:

$$\rho(G) \leqslant |C| = \left|M^*\right| + |H| = \left|M^*\right| + n - 2\left|M^*\right| = n - \left|M^*\right| = n - \mu(G) \quad \Rightarrow \quad \rho(G) + \mu(G) \leqslant n$$

Sia C^* il **minimo edge-cover** su G, cioè tale che $|C^*| = \rho(G)$ e sia $K = (V, C^*)$ il sottografo indotto da C^*. Valgono quindi le seguenti proprietà:

1. K è un grafo aciclico.
2. Ogni cammino di K è composto al più da due archi.
3. K possiede $|V| = n$ vertici.
4. K possiede $|C^*| = \rho(G)$ lati.
5. K può essere decomposto in N componenti connesse aventi la forma di stella.

Consideriamo l'i-esima componente connessa di K e ne indichiamo con s_i il numero di nodi mentre con $s_i - 1$ il numero di archi:

$$n = \sum_{i=1}^{N} s_i \quad \text{e} \quad \rho(G) = \sum_{i=1}^{N} (s_i - 1) = n - N \Rightarrow N = n - \rho(G)$$

Sia M un matching con un arco per ogni componente di K. Si ottiene:

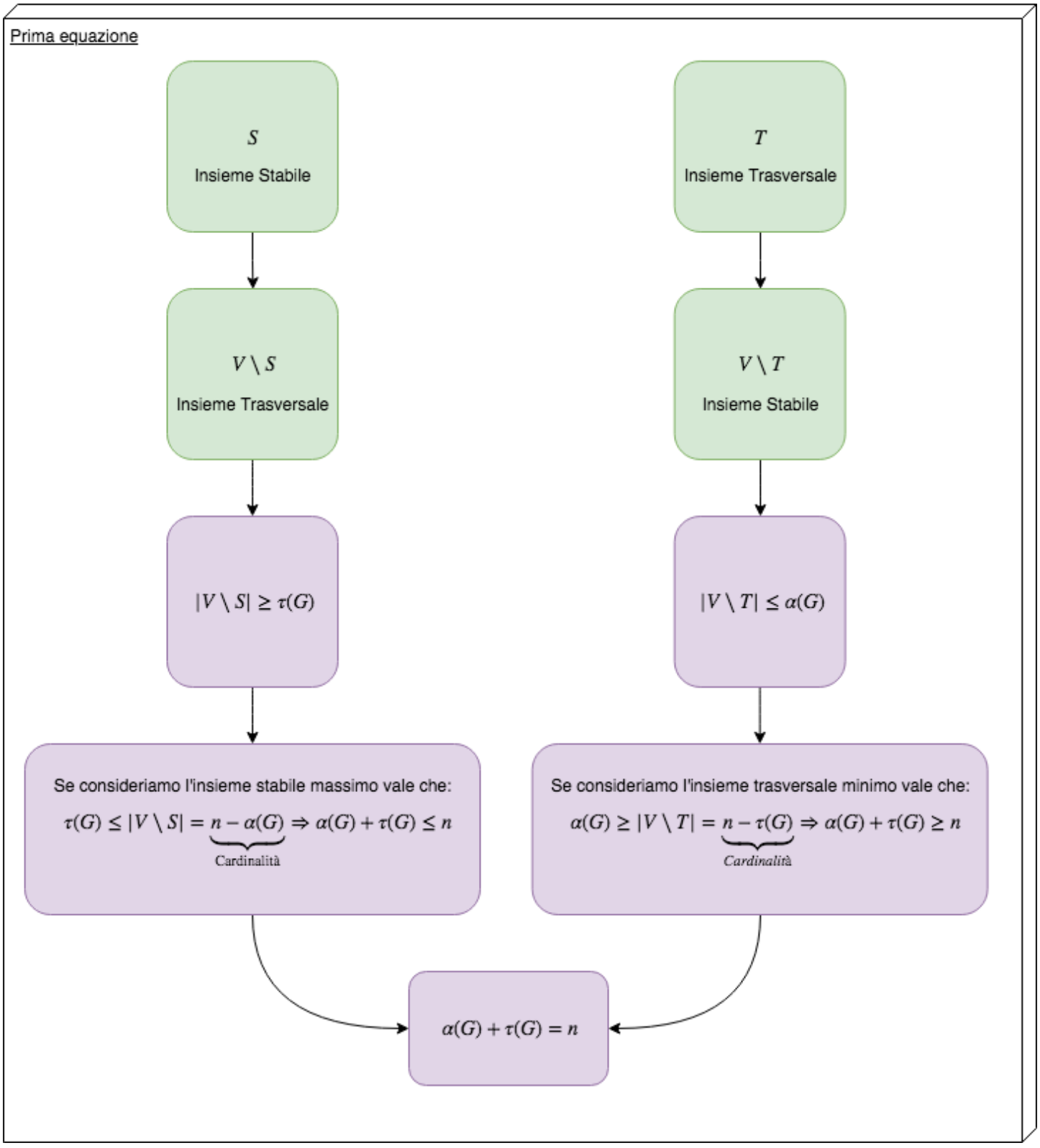
$$\mu(G) \geqslant |M| = n - \rho(G) \Rightarrow \rho(G) + \mu(G) \geqslant n$$

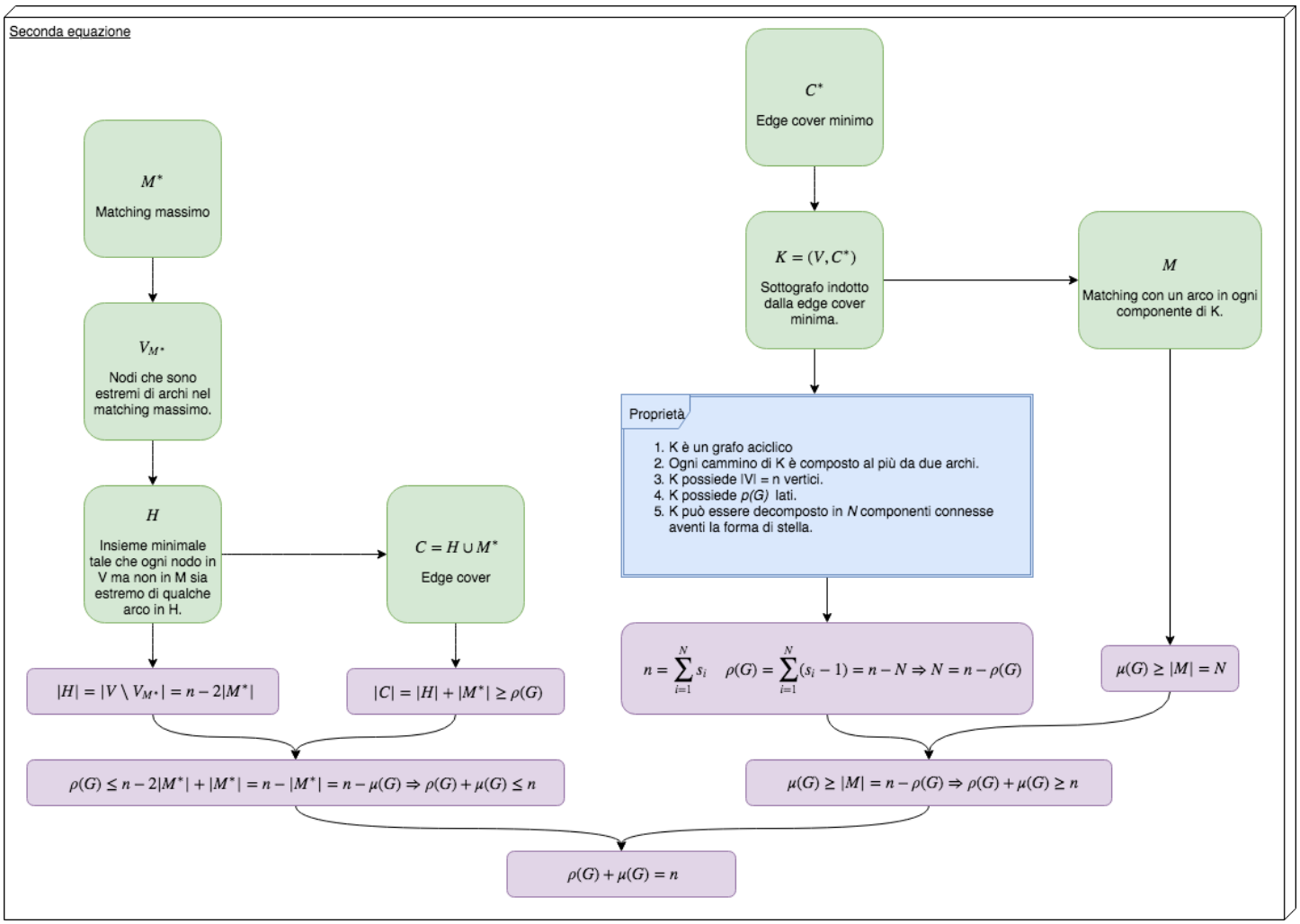
Possiamo quindi concludere che:

$$\rho(G) + \mu(G) \leqslant n \quad \wedge \quad \rho(G) + \mu(G) \geqslant n \quad \Rightarrow \quad \rho(G) + \mu(G) = n$$

□

2.5.1 Diagramma della dimostrazione - Teorema di Gallai




Seconda equazione
C^*
Edge cover minimo
M^*
Matching massimo
$K = (V, C^*)$
Sottografo indotto dalla edge cover minima.
M
Matching con un arco in ogni componente di K.
V_{M^*}
Nodi che sono estremi di archi nel matching massimo.
Proprietà
1. K è un grafo aciclico
2. Ogni cammino di K è composto al più da due archi.
3. K possiede |V| = n vertici.
4. K possiede $\rho(G)$ lati.
5. K può essere decomposto in N componenti connesse aventi la forma di stella.
H
Insieme minimale tale che ogni nodo in V ma non in M sia estremo di qualche arco in H.
$C = H \cup M^*$
Edge cover
$n = \sum_{i=1}^{N} s_i \quad \rho(G) = \sum_{i=1}^{N} (s_i - 1) = n - N \Rightarrow N = n - \rho(G)$
$\mu(G) \geq |M| = N$
$|H| = |V \setminus V_{M^*}| = n - 2|M^*|$
$|C| = |H| + |M^*| \geq \rho(G)$
$\rho(G) \leq n - 2|M^*| + |M^*| = n - |M^*| = n - \mu(G) \Rightarrow \rho(G) + \mu(G) \leq n$
$\mu(G) \geq |M| = n - \rho(G) \Rightarrow \rho(G) + \mu(G) \geq n$
$\rho(G) + \mu(G) = n$

2.6 Cammino alternante e aumentante

Sia M un matching di $G = (V, E)$.

Definizione 2.6.1 (Arco accoppiato). Un arco $(i, j) \in E$ si dice **accoppiato** se:

$$(i, j) \in M$$

Altrimenti è detto **libero**.

Definizione 2.6.2 (Vertice accoppiato). Un vertice $i \in V$ si dice **accoppiato** se su di esso incide un arco di M. Altrimenti si dice che **non incide**.

Definizione 2.6.3 (Cammino alternante). Un cammino P sul grafo G si dice **alternante** rispetto a M se esso è costituito alternativamente da archi accoppiati e liberi.

Definizione 2.6.4 (Cammino aumentante). Un cammino P *alternante* rispetto ad M che abbia entrambi gli estremi esposti si dice **aumentante**.

Teorema 2.6.5 (Estensione di un Matching tramite un cammino aumentante). Sia M un matching definito sul grafo G e sia P un cammino aumentante rispetto a M.

La **differenza simmetrica**:

$$M' = (M \setminus P) \cup (P \setminus M) = M \oplus P = M \Delta P$$

È un **matching** di **cardinalità** $|M| + 1$.

Dimostrazione 2.6.6 (Estensione di un Matching tramite un cammino aumentante). Dimostriamo che l'insieme M' definito tramite **diiffenza simmetrica** gode delle due proprietà descritte:

M' **è un matching:**

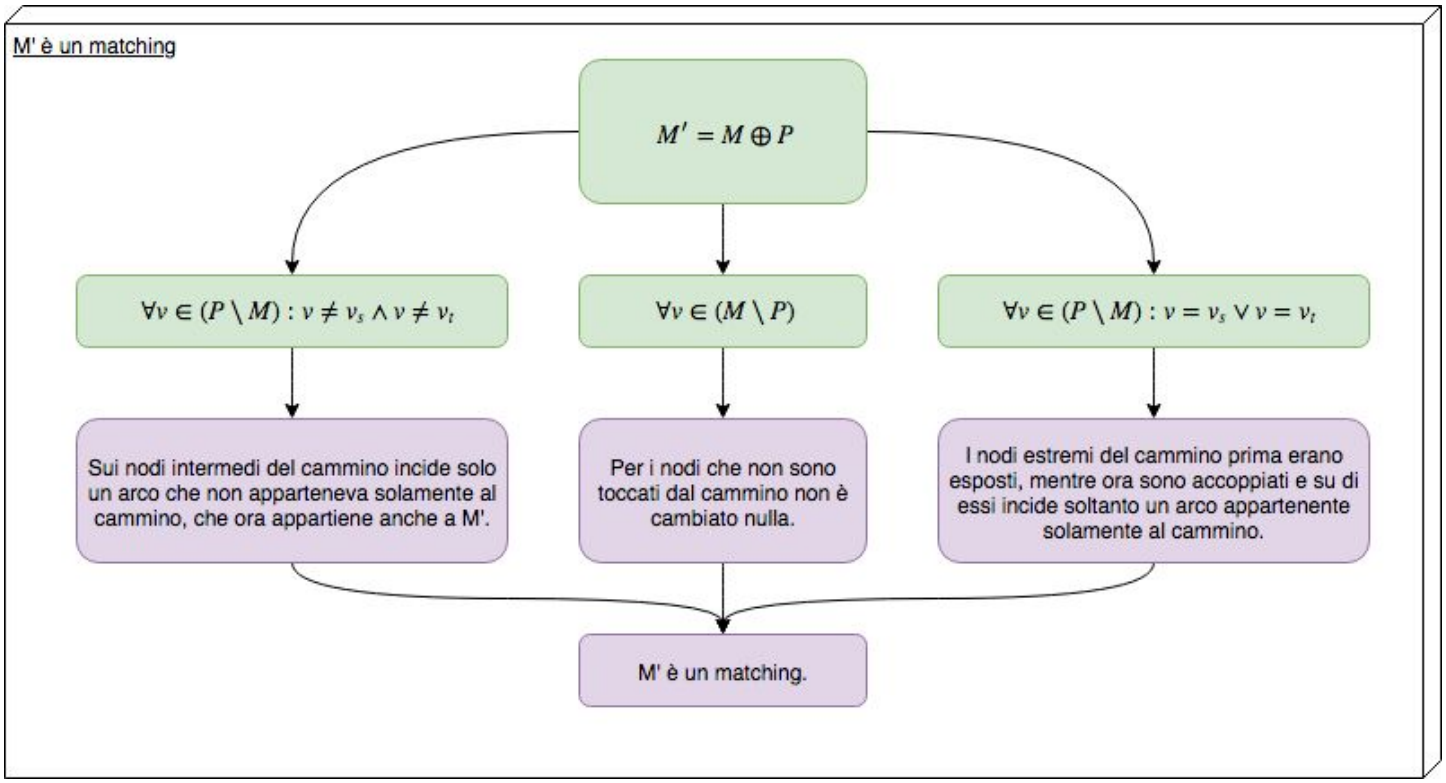
- $\forall v \in (M \setminus P)$ Per i nodi che non sono toccati da P non è cambiato nulla: su di essi incide un solo arco di M che ora appartiene anche ad M'.
- $\forall v \in (P \setminus M) : v \neq v_s \wedge v \neq v_t$ Sui nodi intermedi di P incide soltanto un arco di $P \setminus M$, e quindi di M'.
- $\forall v \in (P \setminus M) : v = v_s \vee v = v_t$ I nodi estremi di P prima erano esposti mentre ora sono accoppiati e su di essi incide soltanto un arco di $P \setminus M$.

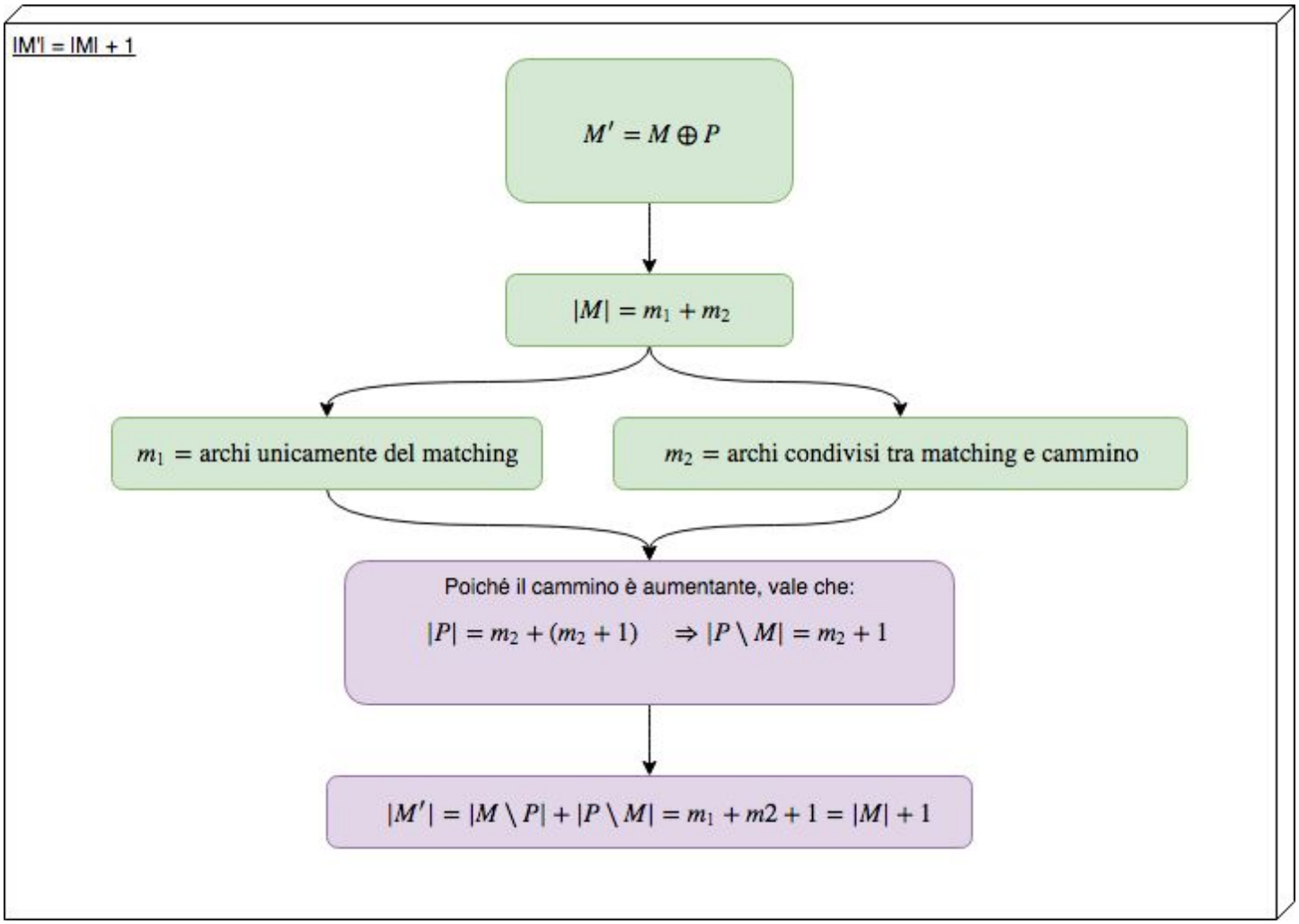
M' **ha un elemento in più di** M**:**

1. Sia $|M| = m_1 + m_2$ con m_1 = "Archi unicamente di M" ed m_2 = Archi condivisi tra il matching M ed il cammino P.
2. Poiché P è aumentante, $|P| = m_2 + (m_2 + 1)$ dove $m_2 + 1 = |P \setminus M|$.
3. $\left|M'\right| = |M \setminus P| + |P \setminus M| = m_1 + m_2 + 1 = |M| + 1$

□

2.6.1 Diagramma della dimostrazione - Estensione di un Matching tramite un cammino aumentante





Teorema 2.6.7 (Teorema di Berge). Un matching M di G è massimo **se e solo se** G non ammette cammini aumentanti rispetto a M.

Dimostrazione 2.6.8 (Teorema di Berge). La condizione sufficiente segue dal teorema precedente. Procediamo quindi a dimostrare la condizione necessaria.

Supponiamo che G ammetta un matching M' con un elemento in più di M. Mostriamo che allora esiste un cammino aumentante per M.

Consideriamo l'insieme di archi:

$$F = \left(M' \cup M\right) \setminus \left(M' \cap M\right)$$

e sia $G' \subseteq G$ con $E' = F$.

Analizziamo il grado di ciascun $v \in V'$, considerando tutti i casi possibili:

1. Un nodo su cui incide lo stesso arco appartenente sia ad M che ad M' è un nodo isolato su G' e quindi ha grado 0.
2. Un nodo su cui incide sia un arco di M sia un arco di M' è un nodo che ha grado 2 su G'.
3. Un nodo su cui incide un arco di M e nessun arco di M' o viceversa è un nodo che ha grado 1 su G'.
4. Un nodo esposto sia rispetto ad M che rispetto ad M' è un nodo isolato su G' e quindi ha grado 0.

Pertanto in G' nessun nodo ha un grado superiore a 2 e possiamo concludere che le componenti connesse di G' sono o nodi isolati o percorsi o cicli.

Nessun ciclo può essere dispari altrimenti ci sarebbero due archi dello stesso matching incidenti sullo stesso nodo e questo è impossibile.

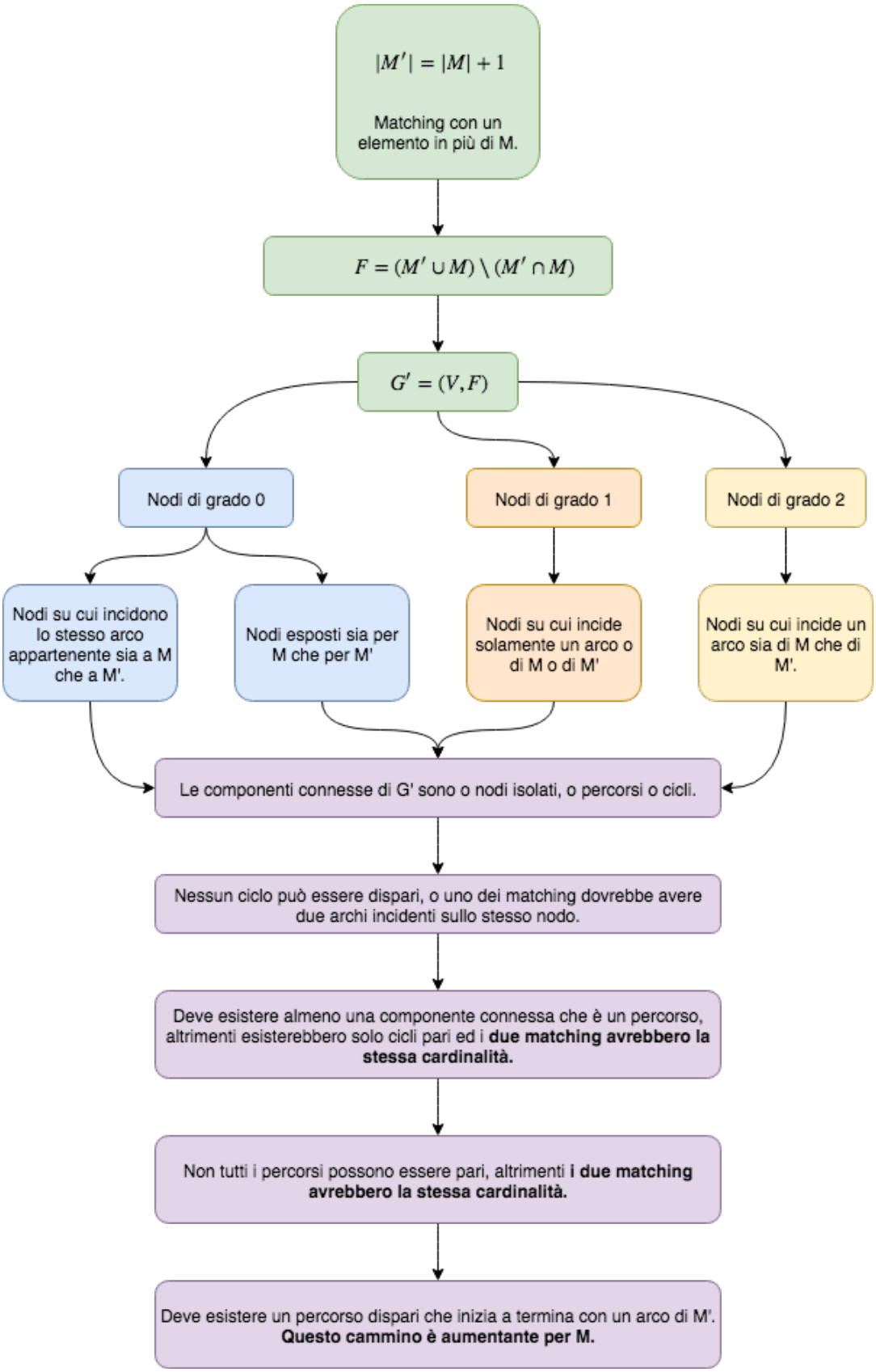
Non possono essere tutti cicli pari altrimenti $|M| = \left|M'\right|$. Deve esistere una componente connessa che è un percorso.

Non tutti i percorsi possono essere pari altrimenti, nuovamente, $|M| = \left|M'\right|$.

Quindi, senza perdita di generalità, possiamo assumere che esista un percorso dispari che inizia e termina con un arco di M'.

Questo percorso è aumentante per M. □

2.6.2 Diagramma della dimostrazione - Teorema di Berge



2.7 Teorema del cammino aumentante

Teorema 2.7.1 (Teorema del cammino aumentante). Sia v un vertice esposto in un matching M. Se non esiste un cammino aumentante per M che parte da v, allora esiste un matching massimo avente v esposto.

Dimostrazione 2.7.2 (Teorema del cammino aumentante). Sia M^* un matching massimo in cui v è accoppiato. Consideriamo $(M^* \cup M) \setminus (M^* \cap M)$: questo insieme non può contenere un cammino alternante con i vertici degli archi di M esposti, altrimenti sarebbe aumentante per esso.

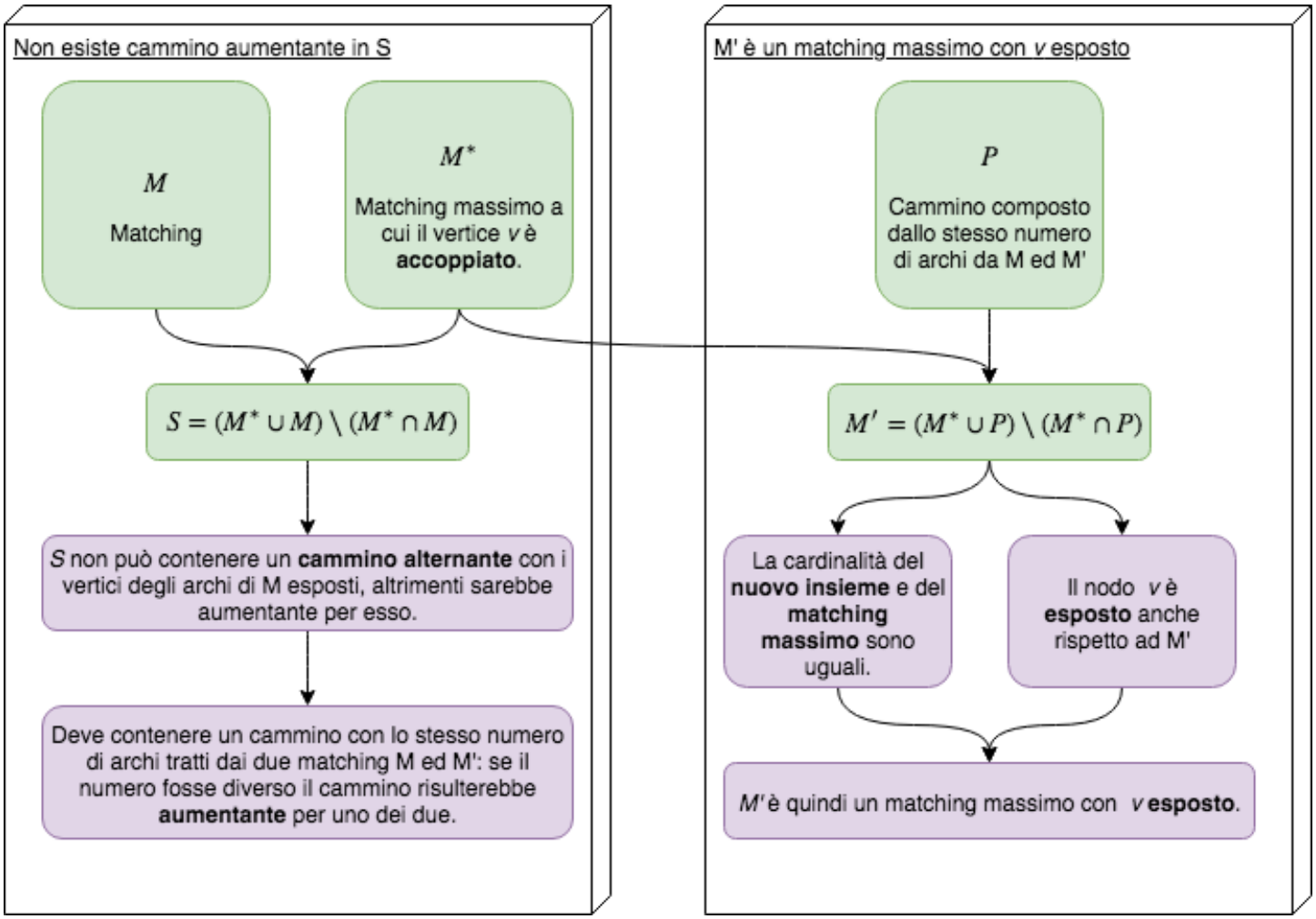
Deve però contenere un cammino composto dallo stesso numero di archi dei due insiemi, M e M^*: un cammino con un solo arco di un insieme, infatti, sarebbe aumentante per l'altro e viceversa.

Consideriamo quindi un cammino P composto da un ugual numero di archi dai due insiemi e consideriamo un nuovo matching $M' = (M^* \cup P) \setminus (M^* \cap P)$. Vanno osservate due proprietà:

1. La cardinalità del nuovo insieme e del matching massimo sono uguali:
$$\left|M'\right| = \left|M^*\right|$$
2. Il nodo v è esposto rispetto ad M'.

Pertanto abbiamo individuato un nuovo matching massimo con v esposto. □

2.7.1 Diagramma della dimostrazione - Teorema del cammino aumentante



2.8 Teorema di König

Teorema 2.8.1 (Teorema di König). Se $G = (X, Y, E)$ è un grafo bipartito, allora $\mu(G) = \tau(G)$.

Dimostrazione 2.8.2 (Teorema di König). Sia M^* un matching massimo, e siano:

1. X_1 un insieme dei nodi x di X **accoppiati** rispetto ad M^*
2. X_2 un insieme dei nodi x di X **esposti** rispetto ad M^*
3. Y_1 insieme dei nodi y di Y raggiungibili da x in X_2. Questi nodi, per definizione, sono **accoppiati** altrimenti M^* non sarebbe massimo.
4. $Y_2 = Y \setminus Y_1$

> **Definizione 2.8.3 (Nodo raggiungibile).** Un nodo $y \in Y$ è raggiungibile se esiste P alternante rispetto ad M^* da x in X_2 tale che l'ultimo arco non appartiene ad M^*.

Consideriamo un set di nodi Z definito come:

$$Z = \{z_1, z_2, \ldots, z_{\mu(G)}\} \quad \text{con} \quad \begin{cases} z_i = y_i & \text{se } y_i \text{ è raggiungibile} \\ z_i = x_i & \text{altrimenti} \end{cases}$$

Procediamo ora a dimostrare che il set Z è *trasversale*.

Iniziamo dimostrando che non esistono archi da nodi in X_2 verso nodi in Y non coperti da Z:

1. Non può esistere un arco non coperto da Z tra un nodo in X_2 e un nodo in Y_2, altrimenti il matching non sarebbe massimo.
2. Non può esistere un arco non coperto da Z tra un nodo in X_2 e un nodo in Y_1 perché i nodi in Y_1 sono raggiungibili e quindi l'arco necessariamente deve essere coperto.

Dimostriamo ora che non esistono archi da nodi in X_1 verso nodi in Y non coperti da Z:

Consideriamo un arco da X_1 a Y_2: se non fosse coperto, allora esisterebbe un nodo, estremo dell'arco del matching, raggiungibile da X_2 in Y_2. Ciò implicherebbe l'esistenza di un cammino aumentante ed il matching sarebbe pertanto non massimo.

Consideriamo ora un arco da X_1 a Y_1: se il nodo terminale non fosse coperto non sarebbe raggiungibile (per la definizione di Y_1 e di Z) e non apparterrebbe in primo luogo a Y_1, quindi l'arco non esisterebbe.

Pertanto, Z è un insieme trasversale di cardinalità pari a $\mu(G)$. □

Dimostrazione 2.8.4 (Teorema di König (Rizzi, 1999)). Dato un grafo G, per le disuguaglianze duali deboli (2.4.1), vale che: $\mu(G) \leqslant \tau(G)$. Per assurdo ipotizziamo che esistano grafi **bipartiti** per cui valga: $\mu(G) < \tau(G)$.

Tra tutti i possibili grafi che violano il teorema di König ne scegliamo uno come controesempio **minimo** (*significa che qualsiasi grafo più piccolo andrebbe a rispettare il teorema di König*):

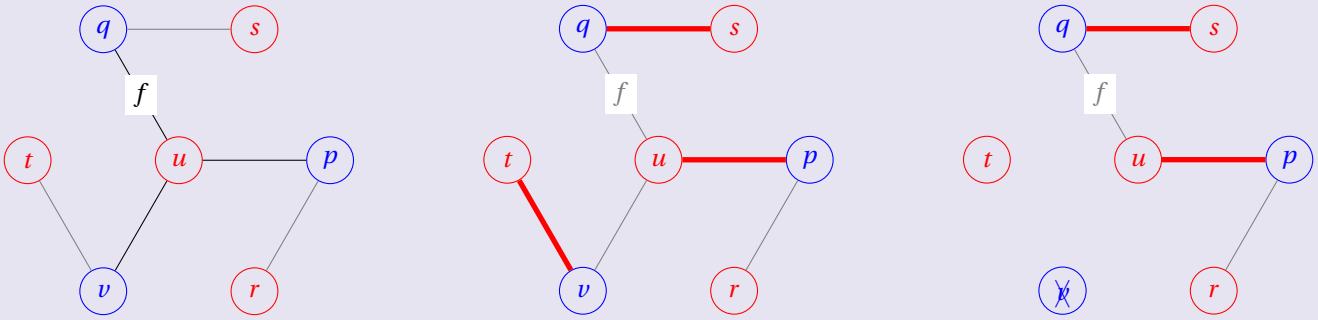
1. Il grafo G ha il più piccolo numero di vertici n.
2. Il grafo G, tra tutti i possibili controesempi con n vertici, ha anche il numero minimo di lati.

Ne segue che:

1. Il grafo G è **connesso**.
2. Il grafo G non è composto da un **ciclo pari** (cicli di lunghezza dispari non sono possibili in un grafo bipartito) o un **cammino**, poiché grafi di questo genere chiaramente rispetterebbero il teorema di König.

Ne segue che G ha un nodo u di grado almeno 3. Sia v uno dei nodi vicini di u. Sia M^*_G il matching massimo per il grafo e $M^*_{G\setminus v}$ il matching massimo per il grafo privato del nodo v.

NB: non è possibile costruire veramente il grafo G siccome stiamo procedendo per assurdo. L'immagine seguente serve per aiutare a visualizzare il procedimento, ma non è una reale rappresentazione del controesempio minimo G.



(a) Componente a stella di G. (b) Matching M^*_G, in rosso. (c) Matching $M^*_{G\setminus v}$, in rosso.

Figura 2.10: Matching massimi e componente a stella.

Siccome G è stato scelto come un **controesempio minimo**, il grafo $G \setminus v$ soddisfa il teorema di König (cioè modificando il grafo togliendo elementi si deve rientrare in un grafo che riespetta il teorema): $\mu(G \setminus v) = \tau(G \setminus v)$. Procediamo quindi **assumendo** che $\mu(G \setminus v) < \mu(G)$.

Allora possiamo realizzare una vertex cover $C_{G\setminus v}$ di cardinalità $\mu(G \setminus v)$ ed estenderla ad essere una cover di G aggiungendovi il vertice v: $C_G = C_{G\setminus v} \cup \{v\}$. Questo implica che:

$$\tau(G) \leqslant \mu(G \setminus v) + 1 \leqslant \mu(G) \quad \Rightarrow \quad \tau(G) = \mu(G)$$

Pertanto G rispetterebbe il teorema di König, una contraddizione con l'ipotesi iniziale per cui $\mu(G \setminus v) < \mu(G)$. Ne segue che $\mu(G \setminus v) = \mu(G)$: deve esistere quindi un matching massimo M^*_G di G in cui nessun lato è incidente con v.

Siccome il nodo u possiede grado almeno pari a 3, possiamo scegliere un lato $f = (u, q) : f \notin M_G$. Nuovamente per **minimalità** il nuovo grafo $G \setminus f$ soddisfa il teorema di König ed è possibile ottenere:

$$\mu(G) = \mu(G \setminus f) = \tau(G \setminus f)$$

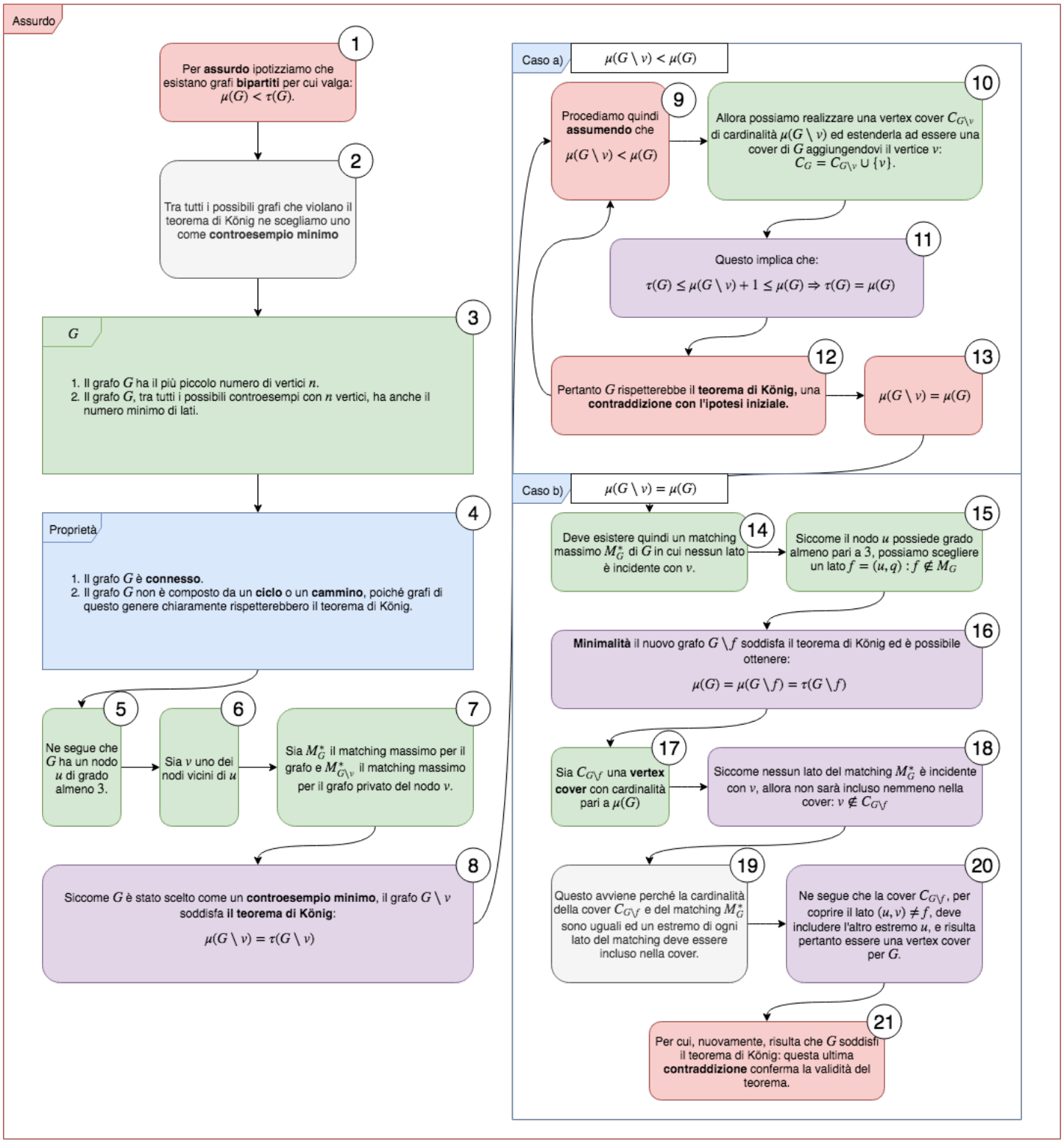
Sia $C_{G\setminus f}$ una **vertex cover** con cardinalità pari a $\mu(G)$: siccome nessun lato del matching M^*_G è incidente con v, allora non sarà incluso nemmeno nella cover: $v \notin C_{G\setminus f}$.

Questo avviene perché la cardinalità della cover $C_{G\setminus f}$ e del matching M^*_G sono uguali ed un estremo di ogni lato del matching deve essere incluso nella cover.

Ne segue che la cover $C_{G\setminus f}$, per coprire il lato $(u, v) \neq f$, deve includere l'altro estremo u, e risulta pertanto essere una vertex cover per G.

Per cui, nuovamente, risulta che G soddisfi il teorema di König: questa ultima contraddizione conferma la validità del teorema. □

2.8.1 Diagramma della dimostrazione - Teorema di König (Rizzi, 1999)



2.9 Teorema di Hall o dei matrimoni

Teorema 2.9.1 (Teorema di Hall). Sia $G = (V_1 \cup V_2, E)$ un grafo bipartito con $|V_1| \geqslant |V_2|$. Dato un insieme di vertici $J \subseteq V_2$, sia $\Gamma(J)$ il set dei vertici in V_1 che sono adiacenti a qualche vertice in J. Allora:

$$G \text{ ammette un } \textbf{matching completo} \Leftrightarrow |\Gamma(J)| \geqslant |J| \quad \forall J \subseteq V_2$$

Cioè in un **grafo bipartito** esiste un **matching completo** se e solo se ogni vertice della componente minore ha almeno un vertice incidente distinto della componente maggiore.

Dimostrazione 2.9.2 (Teorema di Hall). Sia M^* un **matching massimo** di G e J un sottoinsieme qualsiasi di V_2. Sia $E(J)$ l'insieme dei lati contenuti nel matching M^* che sono incidenti con un vertice in J. Allora, i vertici terminali dei lati in $E(J)$ che sono contenuti in V_1 formano un sottoinsieme di cardinalità $|J|$ di $\Gamma(J)$.

Per assurdo, supponiamo che la condizione $|\Gamma(J)| \geqslant |J|$ è valida e che non esista un matching completo $\left|M_G^*\right| < |V_2|$. Allora per il **Teorema di König** è possibile costruire una **vertex cover** di tutti i lati con cardinalità $|X| = \left|M_G^*\right| = \mu(G) < |V_2|$:

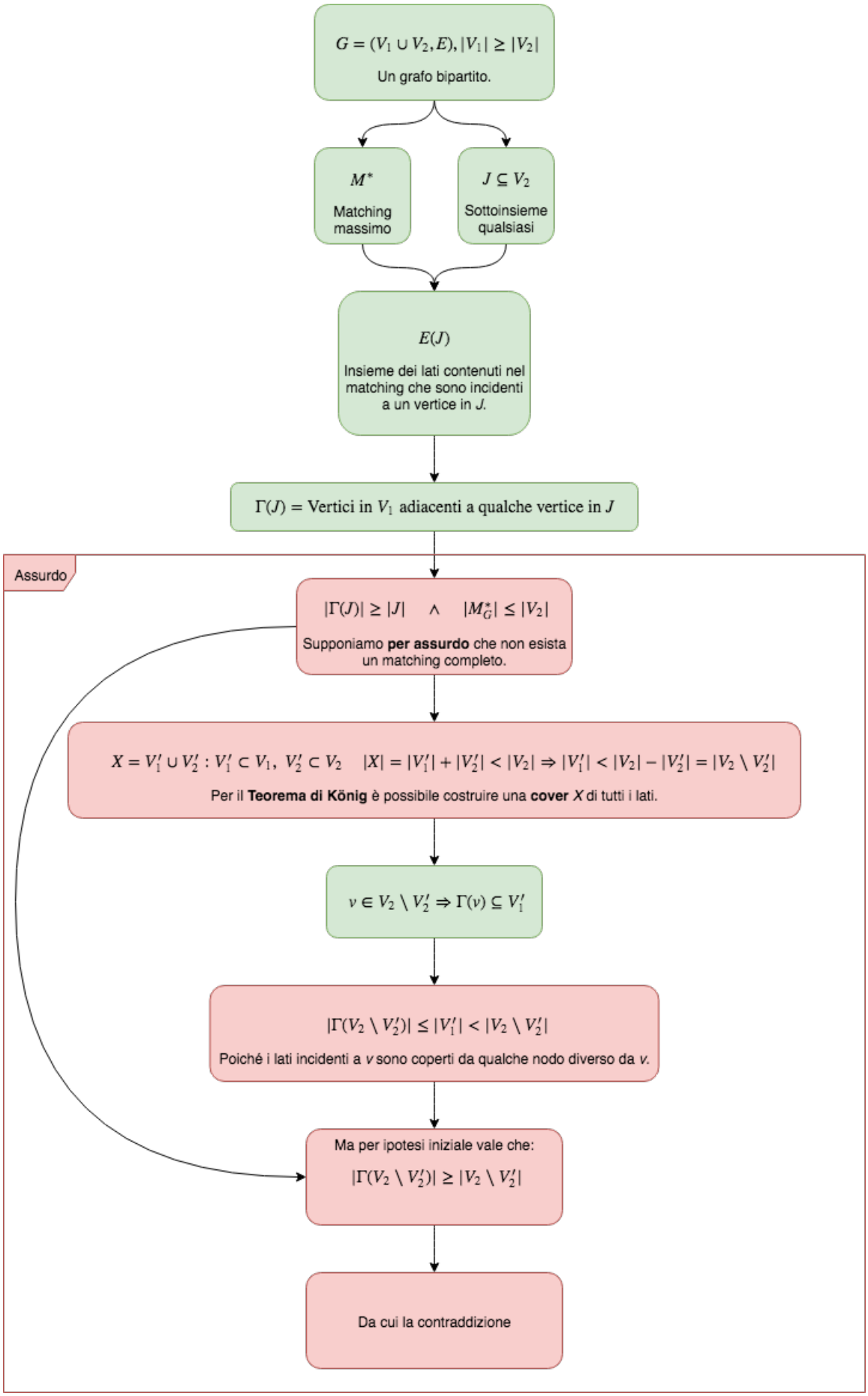
$$X = V_1' \cup V_2' : V_1' \subseteq V_1,\ V_2' \subseteq V_2 \quad |X| = \left|V_1'\right| + \left|V_2'\right| < |V_2| \Rightarrow \left|V_1'\right| < |V_2| - \left|V_2'\right| = \left|V_2 \setminus V_2'\right|$$

Sia ora $v \in V_2 \setminus V_2' \Rightarrow \Gamma(v) \subseteq V_1'$, poiché i lati incidenti a v devono esere coperti da qualche nodo $\neq v$, quindi:

$$\left|\Gamma(V_2 \setminus V_2')\right| \geqslant \left|V_2 \setminus V_2'\right| \quad \wedge \quad \left|\Gamma(V_2 \setminus V_2')\right| \leqslant \left|V_1'\right| < \left|V_2 \setminus V_2'\right|$$

da cui la contraddizione. □

2.9.1 Diagramma della dimostrazione - Teorema di Hall



2.10 Formula di Tutte-Berge e Teorema di Tutte

2.10.1 Formula di Tutte-Berge

Teorema 2.10.1 (Formula di Tutte-Berge). La dimensione di un matching massimo in un grafo G = (V, E) è pari a:

$$\frac{1}{2}\min_{U\subseteq V}\{|U| - \text{odd}\,(G\setminus U) + |V|\}$$

dove odd(H) è il numero di componenti connesse del grafo H con un numero dispari di vertici.

Questa formula implica immediatamente una condizione per l'esistenza di un matching perfetto: il Teorema di Tutte.

2.10.2 Teorema di Tutte

Teorema 2.10.2 (Teorema di Tutte). Un grafo G = (V, E) contiene un matching perfetto se e solo vale che:

$$\text{odd}\,(G\setminus A) \leqslant |A| \quad \forall A \subseteq V$$

2.11 Proposizioni sugli alberi alternanti

Definizione 2.11.1 (Albero alternante). Un albero alternante è costruito a partire da un matching M di un grafo G = (V, E) ed un nodo $r \in V$ di M esposto.

Costruiamo iterativamente set di nodi A e B, in modo tale che ogni nodo A sia al termine di un cammino alternante di M di lunghezza dispari che inizia con r, mentre ogni nodo B al termine di quelli pari.

Tali insiemi possono essere costruiti iniziando da $A = \emptyset, B = \{r\}$ e seguendo la regola seguente:

$$(v,w) \in E, v \in B, w \notin A \cup B, (w,z) \in M \Rightarrow A' = A \cup \{w\}, B' = B \cup \{z\}$$

L'albero così costruito possiede due proprietà:

1. Ogni nodo dell'albero oltre alla radice è coperto da un lato di $M \cap E(T)$.
2. Per ogni nodo v dell'albero, il cammino nell'albero da v alla radice è alternante nel matching M associato all'albero.

Tali insiemi sono importanti poiché se identifichiamo un lato (v,w) tale che $v \in B$ e $w \notin A \cup B$ ed esposto, allora esiste un cammino M-alternante da r a v che assieme a (v,w) va a formare un cammino M-aumentante.

Definizione 2.11.2 (Estendere un albero alternante). Input: Un matching M' di un grafo G', un albero M'-alternante T di G', un lato (v,w) di G' tale che $v \in B(T)$, $v \notin V(T)$ e w sia M'-coperto.

Azione: Sia (w,z) un lato nel matching M' che comprende w, con z il nodo da aggiungere all'albero. L'albero viene aggiornato come:

$$E(T) = E(T) \cup \{(v,w),(w,z)\}$$

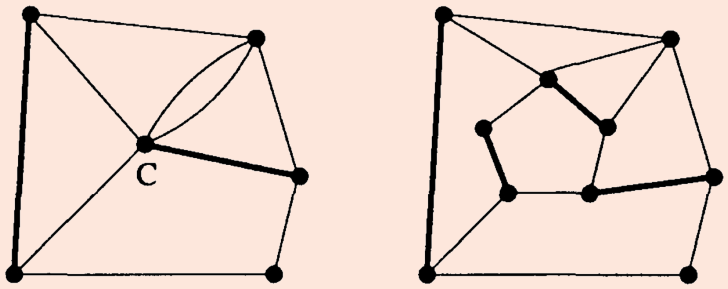


Figura 2.11: Extending

Definizione 2.11.3 (Ridurre e aggiornare un matching ed albero alternante). Input: Un matching M' del grafo G', un albero M'-alternante T, ed un lato (v,w) di G' tale che $v,w \in B(T)$.

Azione: Sia C il circuito formato da (v,w) insieme con il cammino in T da v a w. Aggiorniamo albero e matching come segue:

$$G' = G' \times C$$
$$M' = M \setminus E(C)$$
$$E(T) = E(T) \setminus E(C)$$

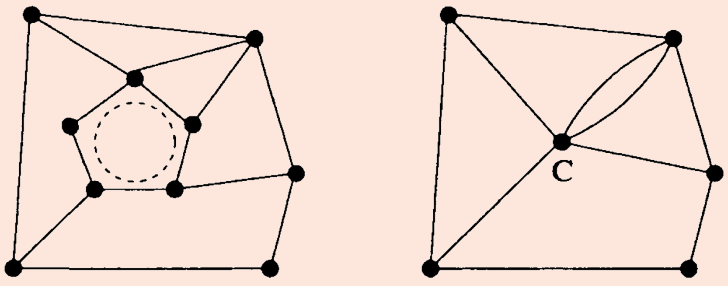


Figura 2.12: Shrinking

Definizione 2.11.4 (Aumentare un Matching). Input: Un matching M' di un grafo G', un albero M'-alternante T di G' con radice r, un lato (v,w) di G' tale che $v \in B(T)$, $v \notin V(T)$ e w sia M'-coperto.

Azione: Sia P un cammino ottenuto appendendo (v,w) al cammino da r a v in T. Il matching viene aggiornato come:

$$M' = \{M' \cup E(P)\} \setminus \{M' \cap E(P)\}$$

Definizione 2.11.5 (Albero alternante frustrato). Un albero alternante in un grafo G = (V, E) si dice frustrato se ogni lato di G avente un termine in $B(T)$ possiede l'altro termine in $A(T)$.

Proposizione 2.11.6 (Prima proposizione su AAF (5.6)). Sia G = (V, E) un grafo con un matching M ed un albero alternante frustrato T associato ad esso. Allora G non possiede un matching perfetto.

Dimostrazione 2.11.7 (Prima proposizione su AAF (5.6)). Ogni elemento di $B(T)$ è una componente dispari a nodo singolo in $G \setminus A(T)$.

Siccome $|A(T)| < |B(T)|$, per il teorema di Tutte G non possiede nessun matching perfetto. □

Proposizione 2.11.8 (Seconda proposizione su AAF (5.7)). Sia G = (V, E) un grafo bipartito, M un matching definito su G e T un albero M-alternante tale che nessun lato di G sia posto tra un nodo in $B(T)$ ed un nodo non in $V(T)$. Allora risulta che T è frustrato e di conseguenza G non ha un matching perfetto.

Dimostrazione 2.11.9 (Seconda proposizione su AAF (5.7)). Procediamo a mostrare che ogni lato avente un termine in $B(T)$ possiede l'altro termine in $A(T)$. Dalle ipotesi del teorema, l'unica possibile eccezione sarebbe un lato tra due nodi in $B(T)$. Ma questo lato, insieme con i lati che lo uniscono alla radice r, formerebbero un ciclo di lunghezza dispari, una cosa impossibile in un grafo bipartito.

Di conseguenza T è frustrato e quindi per la prima proposizione su AAF (5.6), il grafo G non possiede un matching perfetto. □

Definizione 2.11.10 (Pseudonodo). In riferimento ad un grafo G' ottenuto da G, vengono detti **pseudonodi** quei nodi che sono parte di G' ma non di G.

Proposizione 2.11.11 (Terza proposizione su AAF (5.8)). Sia G' un grafo derivato da G, sia M' un matching di G' e sia T un albero M'-alternante frustrato di G' tale che nessun elemento di $A(T)$ è uno pseudonodo.

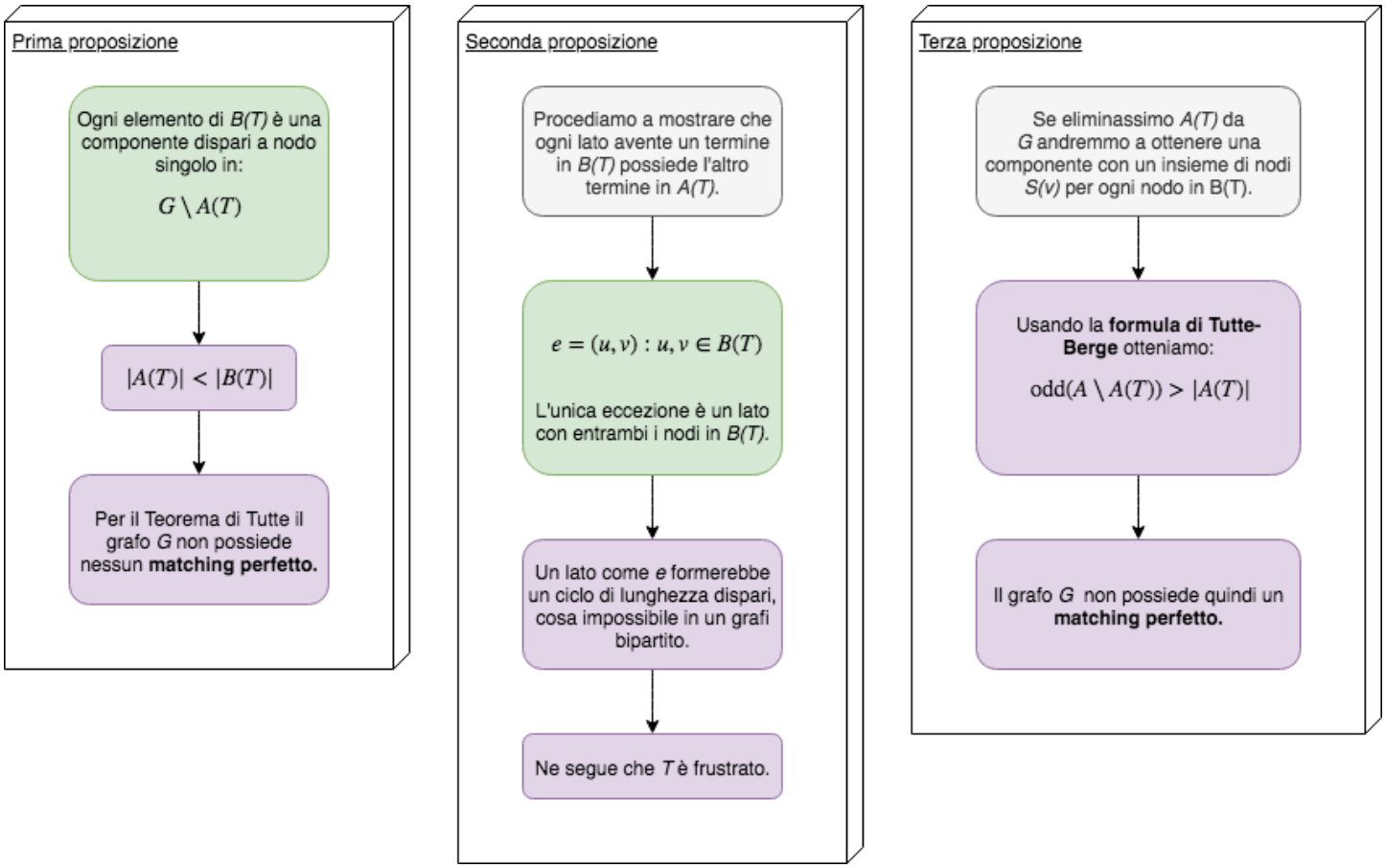
Se T è frustrato, allora G non possiede un matching perfetto.

Dimostrazione 2.11.12 (Terza proposizione su AAF (5.8)). Se eliminassimo $A(T)$ da G andremmo a ottenere una componente con un insieme di nodi $S(v)$ per ogni nodo $v \in B(T)$. Utilizzando quindi la formula di Tutte-Berge si ha che:

$$\text{odd}(G \setminus A(T)) > |A(T)|$$

Ne segue che G non possiede un matching perfetto. □

2.11.1 Diagramma della dimostrazione - Proposizioni sugli AAF

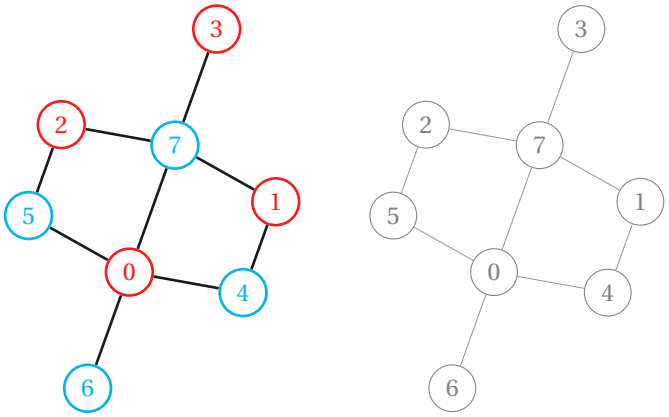


3

Algoritmo per il Matching Massimo

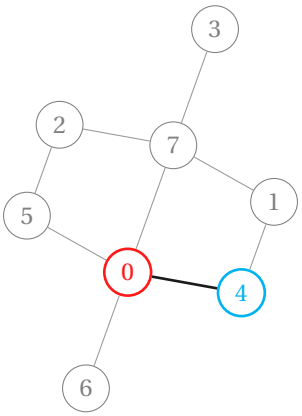
Questo algoritmo identifica un cammino aumentante ed iniziando da un matching banale identifica un matching massimo in un dato grafo bipartito.

3.1 Immagini step by step



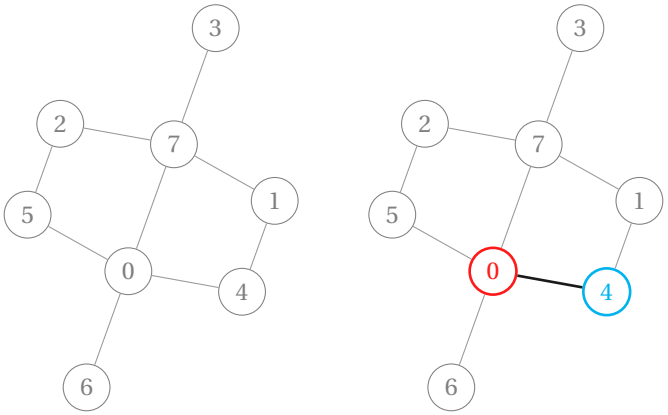
(a) Original graph (b) Initial Matching

Figura 3.1: Starting with empty matching.



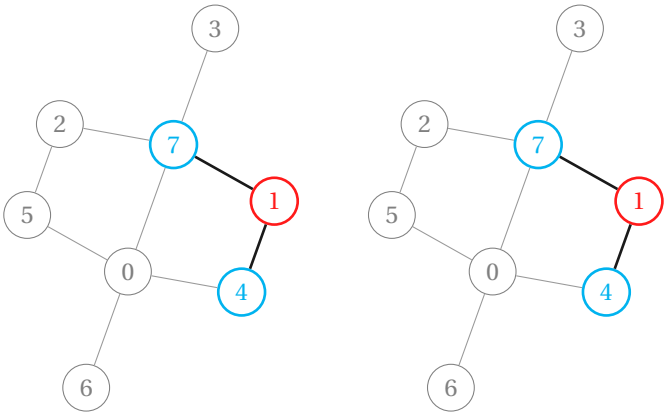
(a) Step 0

Figura 3.2: Searching for path starting from node 0.



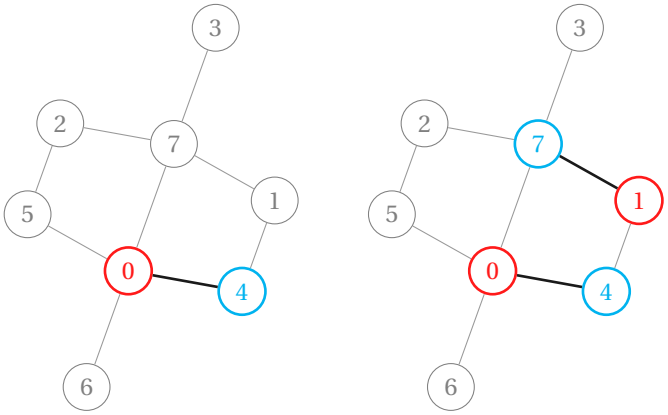
(a) Previous Matching (b) Adding edge 4->0.

Figura 3.3: Updated Matching.



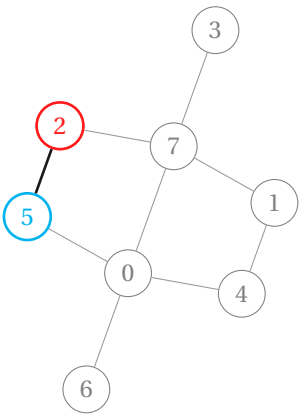
(a) Step 0 (b) Step 1

Figura 3.4: Searching for path starting from node 1.



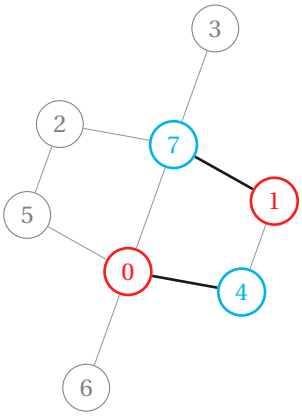
(a) Previous Matching (b) Adding edge 7->1.

Figura 3.5: Updated Matching.

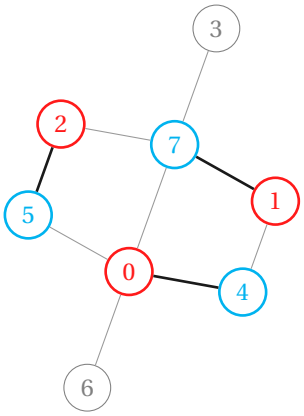


(a) Step 0

Figura 3.6: Searching for path starting from node 2.

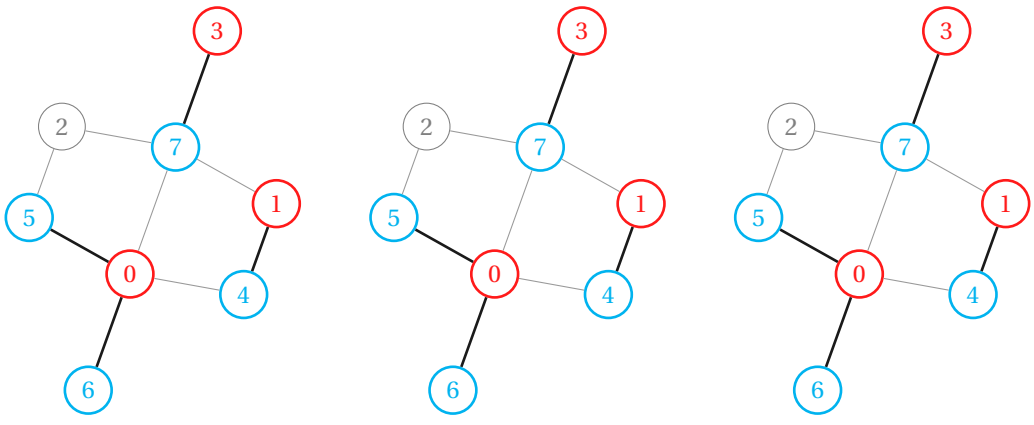


(a) Previous Matching



(b) Adding edge 5->2.

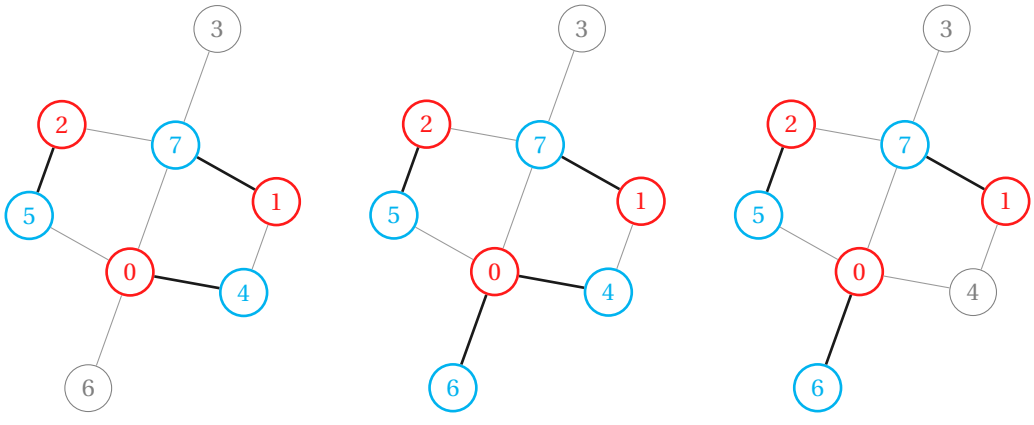
Figura 3.7: Updated Matching.



(a) Step 0 (b) Step 1 (c) Step 2

(d) Step 3 (e) Step 4 (f) Step 5

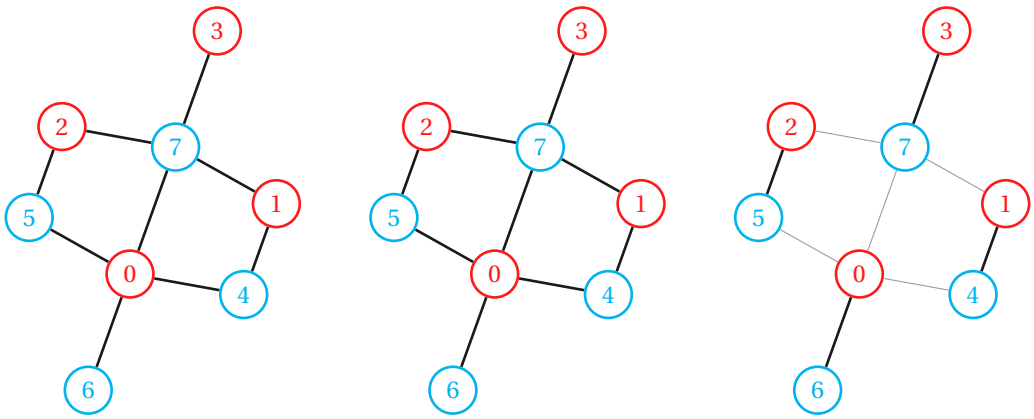
Figura 3.8: Searching for path starting from node 3.



(a) Previous Matching (b) Adding edge 6->0. (c) Removing edge 0->4.

(d) Adding edge 4->1. (e) Removing edge 1->7. (f) Adding edge 7->3.

Figura 3.9: Updated Matching.



(a) Initial Graph (b) Final Graph (c) Final Matching

Figura 3.10: Completed Algorithm.

3.2 Algoritmo Blossom

L'algoritmo procede derivando grafi da un grafo bipartito G iniziale: se riesce ad identificare un matching perfetto in un grafo derivato allora è noto che esiste un matching perfetto in G e se riusciamo ad identificare un certo tipo di AAF all'interno di un grafo derivato possiamo concludere che G non possiede un matching perfetto. I grafi derivati G' vengono ottenuti identificando circuiti dispari all'interno dell'albero alternante T ottenuto da G e rimuovendoli attraverso un procedimento detto di "shrinking" (questo tipo di circuito veniva chiamato blossom, da cui il nome dell'algoritmo.)

Teorema 3.2.1 (Algoritmo Blossom). L'algoritmo Blossom termina dopo $\mathcal{O}(n)$ passi di "augmentations", $\mathcal{O}(n^2)$ passi di "shrinking" e $\mathcal{O}(n^2)$ passi di estensione dell'albero. Inoltre, determina correttamente se il grafo G possiede un matching perfetto.

Dimostrazione 3.2.2 (Algoritmo Blossom). L'algoritmo inizia con un **matching** qualsiasi $M = M'$ di G, che non cessa di essere tale per tutto il procedimento.

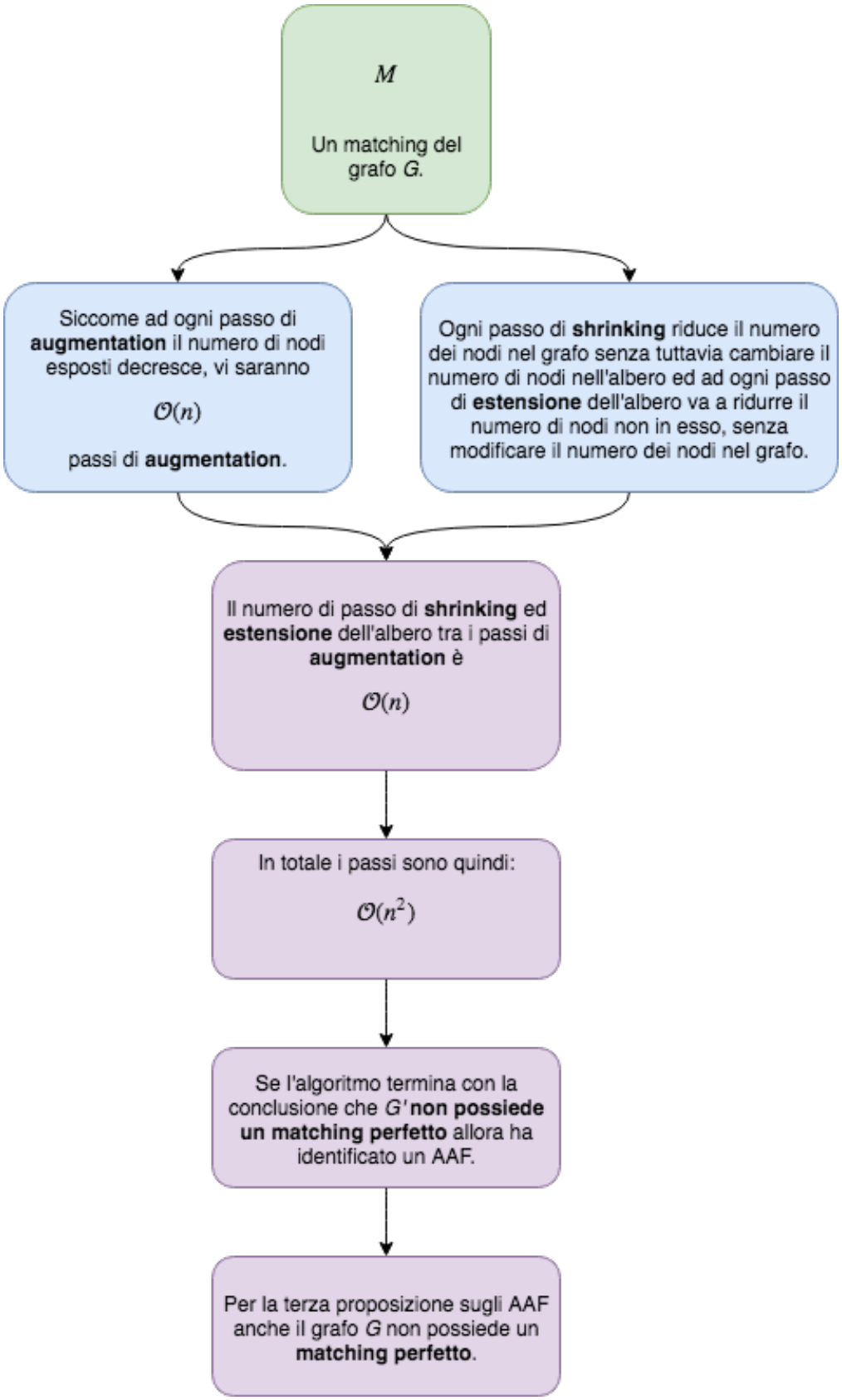
Siccome ad ogni passo di "**augmentation**" il numero di nodi esposti decresce, vi saranno $\mathcal{O}(n)$ passi di "**augmentation**".

Tra questi, ogni passo di "**shrinking**" riduce il numero dei nodi nel **grafo** G' senza tuttavia cambiare il numero dei nodi nell'**albero** T ed ogni passo di estensione dell'albero riduce il numero di nodi non in T non modificando il numero di nodi in G'.

Ne segue che il numero di passi di "**shrinking**" ed estensione dell'albero tra passi di "**augmentation**" è $\mathcal{O}(n)$ e quindi in totale sono $\mathcal{O}(n^2)$.

Infine, siccome ogni G' è un grafo derivato da G, se l'algoritmo termina con la conclusione che G' non possiede un matching perfetto, allora significa che ha trovato un albero con le proprietà descritte nella terza proposizione su AAF (5.8) e quindi G **non possiede un matching perfetto**. □

3.2.1 Diagramma della dimostrazione - Algoritmo Blossom



3.3 Il postino cinese (Chinese postman problem, CPP)

Sia G = (V, E) un grafo connesso e sia $w : E \to \mathbb{R}_0^+$ una lunghezza definita su G. Vogliamo trovare un percorso chiuso C di lunghezza minima $w(C)$ che contiene ogni lato di G almeno una volta.

Definizione 3.3.1 (Grafo e cammino euleriano). Un grafo è detto **euleriano** se è possibile tracciare un cammino che passa una sola volta per tutti gli archi del grafo. Un tale cammino è detto a sua volta euleriano.

Equivalentemente, un grafo G è euleriano se e solo se ogni vertice ha grado pari e un cammino euleriano può essere costruiti con complessità $\mathcal{O}(|E|)$.

Se il grafo G risulta euleriano allora la soluzione del CPP è banale: qualsiasi cammino euleriano andrebbe bene.

Altrimenti si procede come segue: sia X l'insieme di tutti i vertici di G con grado dispari. Aggiungiamo un insieme di lati E' a G tale che le seguenti tre condizioni siano soddisfatte:

1. Ogni lato $e' \in E'$ è parallelo a qualche lato $e \in E$. Estendiamo la distanza w a E' definendo $w(e') = w(e)$.
2. In V, E', solo i vertici di X hanno grado dispari.
3. La distanza $w(E')$ è minima: $w(E') \leqslant w(E'')$ per ogni insieme E'' che soddisfa le due condizioni precedenti.

Ne segue che $\left(V, E \cup E'\right)$ è un multigrafo euleriano, e ogni percorso euleriano introduce un cammino chiuso di lunghezza minima $w(E) + w(E')$ in G.

3.3.1 Applicazione su grafo orientato

Definizione 3.3.2 (Grafo orientato euleriano). Un grafo orientato è detto **euleriano** se tutti i vertici hanno grado entrante uguale al grado uscente.

Consideriamo un grafo orientato non euleriano G e sia $S(T)$ l'insieme di tutti i vertici che hanno un eccesso di archi entranti (o uscenti) rispetto a quelli uscenti (o entranti).

Indichiamo con $d^{+(i)}$ (o $d^{-(i)}$) l'eccesso di grado entrante (o uscente) nei nodi di $S(T)$. Per pareggiare il grado dei nodi $i \in S(T)$ devo aggiungere $d^{+(i)}$ (o $d^{-(i)}$) archi uscenti (o entranti). Poiché ogni arco contribuisce con un grado in uscita ed uno in ingresso, la somma totale dei gradi in uscita coincide con quella dei gradi in ingresso.

Come conseguenza la somma delle etichette $d^{(i)}$ dei vertici in S coincide con quella dei vertici in T. Calcolo il costo C_{ij} di un cammino di costo minimo fra ogni vertice $i \in S$ ed ogni vertice $j \in T$.

Costruisco una istanza di un problema di trasporto con gli insiemi S e T, dove le quantità prodotte dai vertici in S e da trasportare a quelli in T sono le etichette $d^{+(i)}$ mentre la domanda dei vertici in T sono i valori $d^{-(i)}$, ed il costo per unità di merce trasportata lungo l'arco (i, j) è C_{ij}. La soluzione del problema di trasporto indica gli archi da aggiungere.

Se la soluzione dice di trasportare k unità di merda da i a j allora aggiungiamo k archi paralleli ad ogni arco nel cammino di costo minimo da i a j.

3.4 Algoritmi primali-duali

Un algoritmo primale duale risolve problemi di programmazione lineare sfruttando la teoria della dualità e in particolare le condizioni di slackness complementare (o complementary slackness conditions, CSCs). L'algoritmo è inizializzato con una soluzione ammissibile nel problema duale e la corrispondente, in generale inammissibile, soluzione primale che soddisfino le CSCs.

Dopo ogni iterazione l'algoritmo mantiene una coppia della soluzione primale (inammissibile) e della soluzione duale (ammissibile), che soddisfino le CSCs.

L'algoritmo alterna le due iterazioni e riduce monotonamente l'inammissibilità primale fintanto che non raggiunge l'ammissibilità.

Iterazione primale: mantenendo la soluzione duale corrente fissata, si procede ad identificare una soluzione primale minimizzando l'inammissibilità primale tra le soluzioni ammissibili per le CSCs.

Iterazione duale: mantenendo fissata la soluzione primale corrente, si va a modificare la soluzione duale, mantenendola ammissibile e continuando a rispettare le CSCs.

Definizione 3.4.1 (Assegnamento parziale). Un **assegnamento parziale** è un assegnamento che rispetta i seguenti vincoli:

$$\sum_{j \in V_2} x_{ij} \leqslant 1 \quad \forall i \in V_1$$
$$\sum_{i \in V_1} x_{ij} \leqslant 1 \quad \forall j \in V_2$$

Definizione 3.4.2 (Metrica di inammissibilità primale). Il valore dell'inammissibilità primale è determinato dal numero di assegnamenti mancanti.

Definizione 3.4.3 (Cella ammissibile). Chiamiamo **cella ammissibile** della matrice ammissibile quelle per cui vale che:

$$\overline{c}_{ij} = 0$$

4

Algoritmo Ungherese

L'algoritmo ungherese (così chiamato perché basato sul lavoro di due matematici ungheresi), è un algoritmo primale-duale che serve ad identificare il matching di cardinalità massima e costo minimo in un grafo bipartito con matrice di incidenza pesata.

Complessità computazionale 4.0.1 (Algoritmo Ungherese). La complessità dell'algoritmo ungherese è pari a:

$$\mathcal{O}\left(n^3\right)$$

4.1 Preprocessing

4.1.1 Bilanciamento del grafo

Se il dato grafo bipartito non risultasse bilanciato, cioè $|V_1| \neq |V_2|$, si procede a inserire vertici nella partizione di cardinalità minore fino che esse risultano bilanciate. Non aggiungendo lati, non vengono modificati i matching.

4.1.2 Completare il grafo

Se il grafo dato non fosse completo, cioè alcuni lati fossero assenti tra le due partizioni, si procede ad aggiungere lati con un costo molto alto in modo tale che essi non siano scelti, ma che in ogni caso il grafo risulti completo.

Questi nuovi lati portano alla creazione di matching nuovi, ma di costo estremamente alto.

4.2 Il problema dell'assegnamento riformulato

Possiamo quindi riformulare il problema come quello di trovare il matching completo tra due set di vertici in un dato grafo bipartito pesato: ogni soluzione è rappresentata da una **matrice di assegnamento**, una matrice quadrata dove un set rappresenta le righe e l'altro le colonne.

4.3 Iterazione primale-duale

$$\begin{aligned} \min z = \sum_{i \in V_1} \sum_{j \in V_2} c_{ij} x_{ij} & \\ \sum_{j \in V_2} x_{ij} = 1 \quad & \forall i \in V_1 \\ \sum_{i \in V_1} x_{ij} = 1 \quad & \forall j \in V_2 \\ x_{ij} \geqslant 0 \quad & \forall i \in V_1, \forall j \in V_2 \end{aligned}$$

(a) Problema primale

$$\begin{aligned} \max w = \sum_{i \in V_1} u_i + \sum_{j \in V_2} v_j & \\ u_i + v_j \leqslant c_i j \quad & \forall i \in V_1, \forall j \in V_2 \end{aligned}$$

(b) Problema duale

Le variabili duali $\underline{\boldsymbol{u}}$ e $\underline{\boldsymbol{v}}$ non sono vincolate in segno.

Le variabili di scarto complementare duale sono:

$$\overline{c} = c_{ij} - u_i - v_j$$

Per garantire l'ottimalità della soluzione, le CSCs impongono che:

$$\overline{c}_{ij} x_{ij} = 0 \quad \forall i \in V_1, \forall j \in V_2$$

Iterazione primale Mantenendo fissati $\underline{\boldsymbol{u}}$ e $\underline{\boldsymbol{v}}$ e mantenendo pertanto fissato anche $\underline{\overline{\boldsymbol{c}}}$ si procede a determinare una soluzione $\underline{\boldsymbol{x}}$ massimizzando il numero di assegnamenti (cioè di variabili unitarie) utilizzando solamente celle ammissibili.

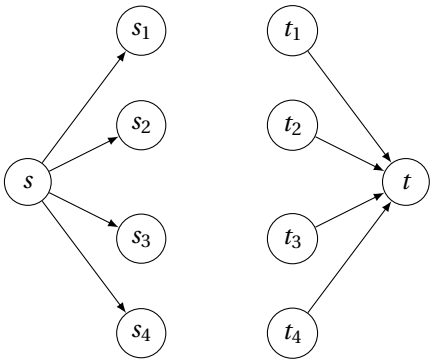
Iterazione duale Si procede ad aggiornare $\underline{\boldsymbol{u}}$ e $\underline{\boldsymbol{v}}$, mantenendo $\overline{c}_{ij}$ dove $x_{ij} = 1$ e rendendo alcune celle ammissibili inammissibili.

4.4 Pseudo codice

Algorithm 1: Algoritmo Ungherese

```
1  Step 1: Inizializzazione duale di u e v;
2  Step 2: Inizializzazione primale di x;
3  while x è inammissibile do
4      Step 3.1: Inizializzazione del cammino;
5      Path ← nil;
6      while (Path = nil) do
7          while (Path = nil) ∧ (L ≠ ∅) do
8              Step 3.2: Propagazione delle etichette;
9          end
10         if Path = nil then
11             Step 4: Iterazione duale, modifica di u e v;
12         end
13     end
14     Step 5: Iterazione primale, modifica di x;
15 end
```

4.5 Visualizzazione



(a) Grafo

	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	*nil*
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	*nil*
Mate	*nil*	*nil*	*nil*	*nil*	**0**

(b) Matching parziali

$\underline{\overline{\boldsymbol{c}}}$	t_1	t_2	t_3	t_4	$\underline{\boldsymbol{u}}$
s_1	15	22	13	4	0
s_2	12	21	15	7	0
s_3	16	20	22	6	0
s_4	6	11	8	5	0
$\underline{\boldsymbol{v}}$	0	0	0	0	**0**

(c) Duale

Figura 4.2: Valori iniziali

I valori in fondo a destra delle tabelle sono rispettivamente:

$$\min z = \sum_{i \in V_1} \sum_{j \in V_2} c_{ij} x_{ij} \qquad \max w = \sum_{i \in V_1} u_i + \sum_{j \in V_2} v_j$$

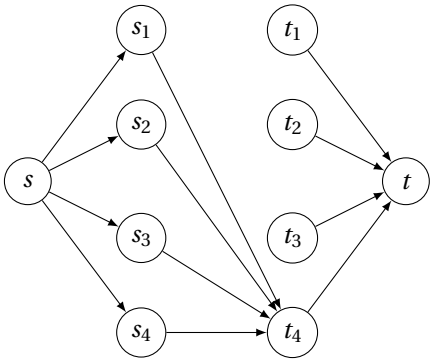
4.5.1 Primo step: Inizializzazione duale

Algorithm 2: Inizializzazione duale

```
for i ∈ V1 do
    u_i ← min_{j∈V1} {c_ij}
end
for j ∈ V2 do
    v_i ← min_{i∈V2} {c_ij − u_i}
end
```

Complessità computazionale 4.5.1 (Inizializzazione duale). Durante l'inizializzazione duale le variabili duali sono sottratte al valore minimo della propria riga che rende il vincolo duale attivo. Questo garantisce che la soluzione duale rimanga **ammissibile**.

$$\mathcal{O}\left(n^2\right)$$



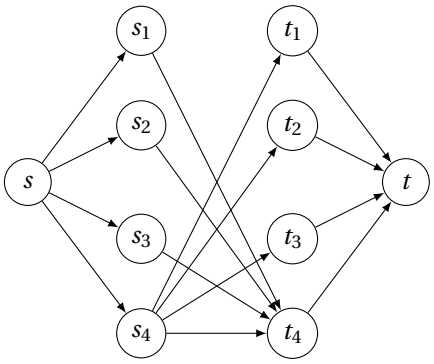
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	*nil*
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	*nil*
Mate	*nil*	*nil*	*nil*	*nil*	**0**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	11	18	9	0	4
s_1	5	14	8	0	7
s_2	10	14	16	0	6
s_3	1	6	3	0	5
s_4	0	0	0	0	**22**

(b) Duale

Figura 4.3: Operazione sulle righe



	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	*nil*
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	*nil*
Mate	*nil*	*nil*	*nil*	*nil*	**0**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Figura 4.4: Operazione sulle colonne

4.5.2 Secondo step: Inizializzazione primale

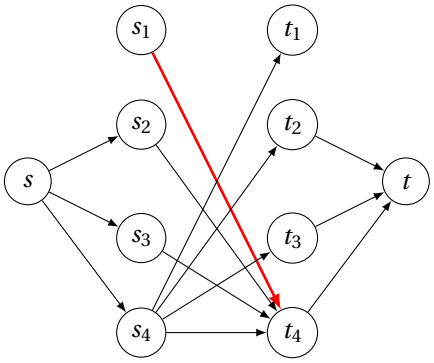
Algorithm 3: Inizializzazione primale

```
for v ∈ V1 ∪ V2 do
    Mate_v = nil;
end
Card ← 0 while (∃i, j : (c_ij − u_i − v_j = 0) ∧ (Mate_i = nil) ∧ (Mate_j = nil)) do
    x_ij ← 1;
    Card ← Card + 1;
    Mate_i ← j;
    Mate_j ← i;
end
```

Complessità computazionale 4.5.2 (Inizializzazione primale). Viene calcolato un **matching parziale massimale**, utilizzando unicamente le celle ammissibili. Questo richiede una scansione completa della matrice $n \times n$, per cui la complessità risulta:

$$\mathcal{O}\left(n^2\right)$$

Nel caso in esempio vi sono 7 celle ammissibili, cioè quelle che rendono attivi i vincoli (le celle a valore zero). Procedendo con la scelta in ordine lessicografico, viene scelta la prima e quarta colonna: $(1,4)$.



	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	*nil*
Mate	*nil*	*nil*	*nil*	1	**4**

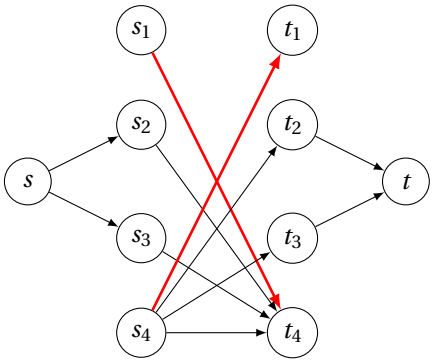
(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Figura 4.5: Primo aggiornamento

A questo punto abbiamo ancora tre celle ammissibili, pertanto la cella $(4,1)$ è scelta successivamente.



	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Figura 4.6: Secondo aggiornamento

Non vi sono più celle ammissibili su colonne o righe accoppiate ed il matching parziale corrente è massimale. Il valore di Card è pari a 2.

4.5.3 Test di ammissibilità primale

Consiste semplicemente nel verificare quanti lati sono stati inseriti nella soluzione primale: l'obbiettivo è Card = n, mentre ora siamo ancora a 2.

4.5.4 Terzo step: Ricerca di un cammino aumentante

Si procede ora a identificare un cammino aumentante, che coincide anche con un cammino alternante siccome il grafo è bipartito. Questo cammino deve attraverso cui una unità di flusso può andare dalla sorgente s al termine t.

Ogni volta che un cammino $s \rightarrow t$ viene identificato, la cardinalità del matching parziale corrente può essere aumentata di 1 attraverso una **iterazione primale**. Per identificare un cammino può essere necessario eseguire al più $\mathcal{O}(n)$ **iterazioni duali**, poiché ognuna di esse consente di raggiungere un nuovo nodo in V_2.

Il cammino parte da s: ogni nodo in V_1 e V_2 che può essere raggiunto è quindi etichettato: l'etichetta di un nodo è il suo predecessore. L è l'insieme delle etichette usate per generare gli altri.

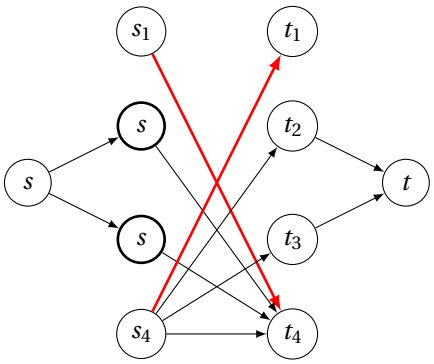
Il vettore $\boldsymbol{p}$ mantiene il valore minimo del costo ridotto per ogni colonna priva di etichetta tra le righe etichettate. Il vettore $\underline{\boldsymbol{\pi}}$ mantiene invece la riga corrispondente.

Step 3.1: Inizializzazione del cammino

Algorithm 4: Inizializzazione del cammino

```
1  L ← ∅;
2  for v ∈ V1 ∪ V2 do
3  |   Label_v = nil;
4  end
5  for j ∈ V2 do
6  |   p_j ← ∞;
7  |   π_j ← nil;
8  end
9  for i ∈ V1 : Mate_i = nil do
10 |   Label_i ← s;
11 |   L ← L ∪ {i};
12 |   for j ∈ V2 : Label_j = nil do
13 |   |   if c_ij − u_i − v_i < p_j then
14 |   |   |   p_j ← c_ij − u_i − v_i;
15 |   |   |   π_j ← i;
16 |   |   end
17 |   end
18 end
```

Vi sono due nodi non assegnati in V_1: $L = \{s_2, s_3\}$



	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{\boldsymbol{c}}$	t_1	t_2	t_3	t_4	$\underline{\boldsymbol{u}}$
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Figura 4.7: Identificazione dei nodi non marcati

Step 3.2: Propagazione delle etichette

Algorithm 5: Propagazione delle etichette

```
v ← L.pop();
if v ∈ V1 then
    Step 3.2.A: Propagazione da v ∈ V1 a V2;
else if Mate_v ≠ nil then
    Step 3.2.B: Propagazione da v ∈ V2 a V1;
else
    Path ← v
end
```

La propagazione termina quando un nodo non assegnato $v \in V_2$ viene etichettato. Ogni nodo viene inserito o estratto da L al più una volta.

Algorithm 6: Propagazione delle etichette da V_1 a V_2

```
for j ∈ V2 : (Label_j = nil) ∧ (c_vj − u_v − v_v = 0) do
    Label_j ← k;
    L ← L ∪ {j};
end
```

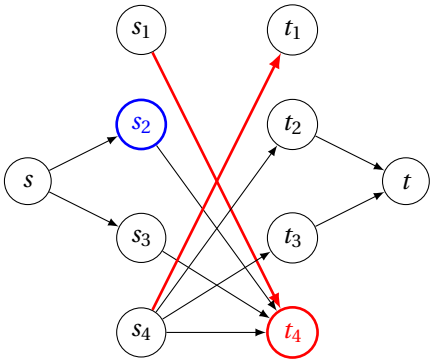
Step 3.2.A: Propagazione delle etichette da V_1 a V_2 La propagazione da $v \in V_1$ a V_2 avviene lungo i lati tali che:

1. Corrispondono a celle ammissibili.
2. Non appartengono al matching parziale corrente.

Algorithm 7: Propagazione delle etichette da V_2 a V_1

```
if Label_Mate_v = nil then
    Label_Mate_v ← v;
    L ← L ∪ {Mate_v};
    for j ∈ V2 : Label_j = nil do
        if c_Mate_v j − u_Mate_v − v_j < p_j then
            p_j ← c_Mate_v j − u_Mate_v − v_j;
            π_j ← Mate_v;
        end
    end
end
```

Step 3.2.B: Propagazione delle etichette da V_2 a V_1 La propagazione da $v \in V_2$ a V_1 avviene lungo i lati del matching parziale corrente.



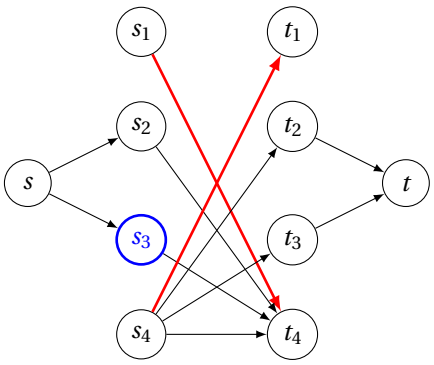
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Figura 4.8: Estraiamo $L = \{s_2, s_3\}$ ed inseriamo $L = \{s_3, t_4\}$



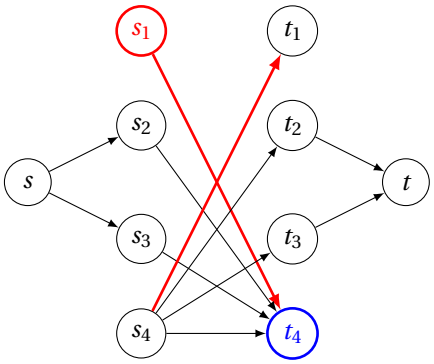
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Figura 4.9: Estraiamo $L = \{s_3, t_4\}$ ed inseriamo $L = \{t_4\}$



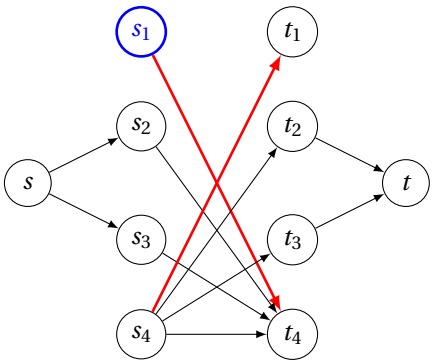
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Figura 4.10: Estraiamo $L = \{t_4\}$ ed inseriamo $L = \{s_1\}$



	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Figura 4.11: Estraiamo $L = \{s_1\}$ ed inseriamo $L = \{\}$

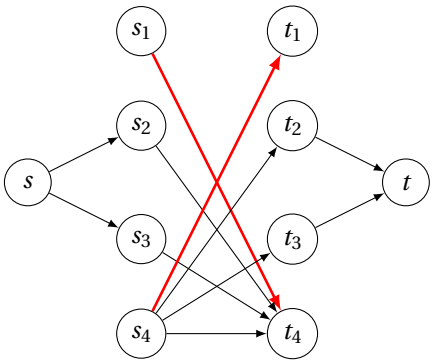
Riassumendo: Nessun cammino da sorgente s a destinazione t è stato identificato. I nodi s_1, s_2, s_3 e t_4 sono stati etichettati, mentre i nodi s_4, t_1, t_2 e t_3 non lo sono.

4.5.5 Step 4: Iterazione duale

Algorithm 8: Iterazione duale

$\delta \leftarrow \min_{j \in V_2} \{p_j : \text{Label}_j) nil\}$;
for $i \in V_1 : Label_i \neq nil$ **do**
  $u_i \leftarrow u_i + \delta$;
end
for $j \in V_2 : Label_j \neq nil$ **do**
  $v_j \leftarrow v_j - \delta$;
end
for $j \in V_2 : Label_j = nil$ **do**
  $p_j \leftarrow p_j - \delta$;
end
for $j \in V_2 : (Label_j = nil) \wedge (p_j = 0)$ **do**
  $\text{Label}_j \leftarrow (\pi_j, +)$;
  $L \leftarrow L \cup \{j\}$;
end

Il valore δ è il costo minimo ridotto nella sotto-matrice di righe etichettate e colonne non etichettate. Almeno una nuova cella diviene ammissibile ed è utilizzata per etichettare un nodo in più in V_2.



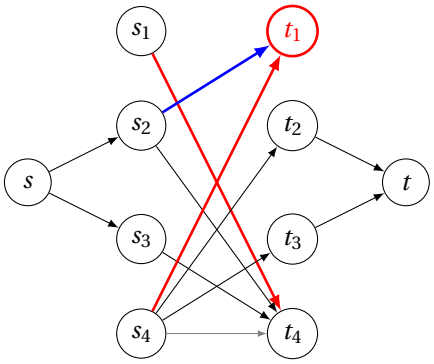
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	10	12	6	0	4
s_1	4	8	5	0	7
s_2	9	8	13	0	6
s_3	0	0	0	0	5
s_4	1	6	3	0	**32**

(b) Duale

Consideriamo i lati che uniscono i nodi etichettati in V_1 con i nodi non etichettati in V_2. Si identifica $\delta = 4$.



	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

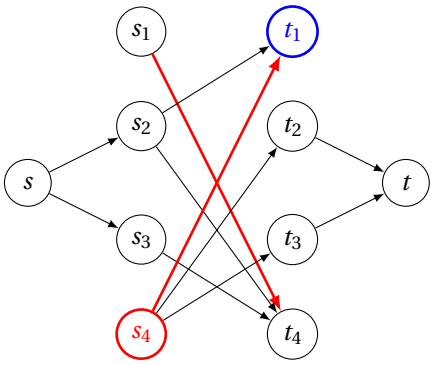
$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	6	8	2	0	8
s_1	0	4	1	0	11
s_2	5	4	9	0	10
s_3	0	0	0	4	5
s_4	1	6	3	−4	**40**

(b) Duale

Si va ad incrementare u_1, u_2 ed u_3 di δ, mentre si decrementa v_4 di δ.

La cella $(4,4)$ cessa di essere ammissibile, mentre la cella $(2,1)$ diviene ammissibile e si etichetta t_1 da s_2.

4.5.6 Step 3.2: Nuova propagazione delle etichette



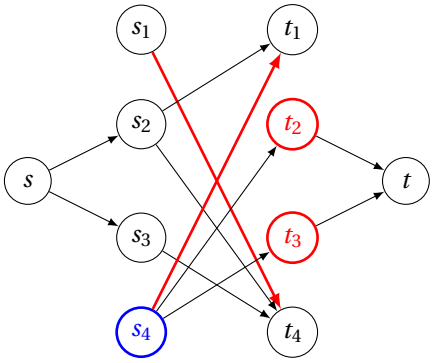
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	6	8	2	0	8
s_1	0	4	1	0	11
s_2	5	4	9	0	10
s_3	0	0	0	4	5
s_4	1	6	3	−4	**40**

(b) Duale

Figura 4.14: Estraiamo $L = \{t_1\}$ ed inseriamo $L = \{s_4\}$



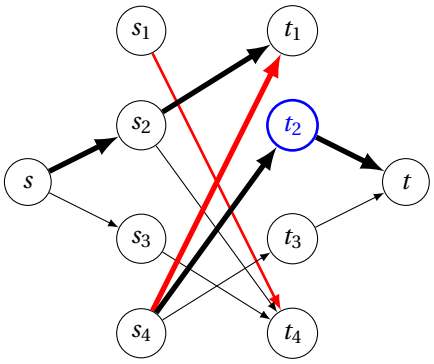
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	6	8	2	0	8
s_1	0	4	1	0	11
s_2	5	4	9	0	10
s_3	0	0	0	4	5
s_4	1	6	3	−4	**40**

(b) Duale

Figura 4.15: Estraiamo $L = \{s_4\}$ ed inseriamo $L = \{t_2, t_3\}$



	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	1
Mate	4	*nil*	*nil*	1	**10**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	6	8	2	0	8
s_1	0	4	1	0	11
s_2	5	4	9	0	10
s_3	0	0	0	4	5
s_4	1	6	3	−4	**40**

(b) Duale

Figura 4.16: Estraiamo $L = \{t_2, t_3\}$

Il nodo estratto t_2 non è assegnato: un cammino $s \to t$ è stato identificato.

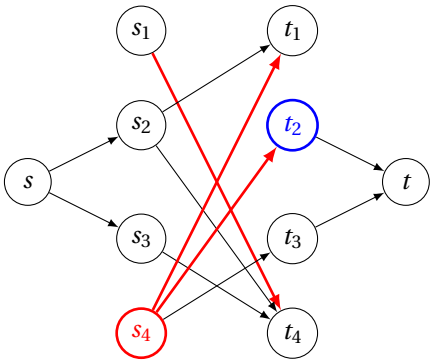
4.5.7 Step 5: Iterazione primale

Algorithm 9: Iterazione primale

```
1  j ← Path;
2  while j = s do
3      i ← Label_j;
4      Mate_j ← i;
5      Mate_i ← j;
6      x_ij ← 1;
7      z ← z + c_ij;
8      Card ← Card + 1;
9      j ← Label_i;
10     if Label_i ≠ s then
11         x_ij ← 0;
12         z ← z − c_ij;
13         Card ← Card − 1;
14     end
15 end
```

Complessità computazionale 4.5.3 (Iterazione primale). Il cammino è ricostruito a ritroso da t a s e contiene $\mathcal{O}(n)$ lati, pertanto l'**iterazione primale** ha complessità:

$$\mathcal{O}(n)$$



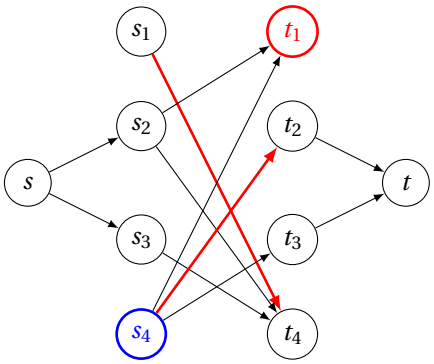
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	2
Mate	4	4	*nil*	1	**21**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	6	8	2	0	8
s_1	0	4	1	0	11
s_2	5	4	9	0	10
s_3	0	0	0	4	5
s_4	1	6	3	−4	**40**

(b) Duale

Figura 4.17: Il predecessore di t_2 è s_4, Card = 3



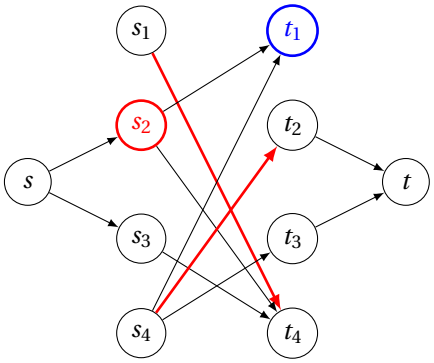
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	*nil*
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	2
Mate	*nil*	4	*nil*	1	**15**

(a) Matching parziali

$\overline{c}$	t_1	t_2	t_3	t_4	u
s_0	6	8	2	0	8
s_1	0	4	1	0	11
s_2	5	4	9	0	10
s_3	0	0	0	4	5
s_4	1	6	3	−4	**40**

(b) Duale

Figura 4.18: Il predecessore di s_4 è t_1, Card = 2



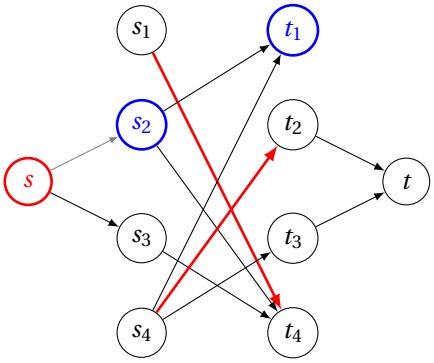
	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	1
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	2
Mate	2	4	*nil*	1	**15**

(a) Matching parziali

$\overline{\underline{c}}$	t_1	t_2	t_3	t_4	$\underline{u}$
s_0	6	8	2	0	8
s_1	0	4	1	0	11
s_2	5	4	9	0	10
s_3	0	0	0	4	5
s_4	1	6	3	−4	**40**

(b) Duale

Figura 4.19: Il predecessore di t_1 è s_2, Card = 3



	t_1	t_2	t_3	t_4	Mate
s_1	15	22	13	4	4
s_2	12	21	15	7	1
s_3	16	20	22	6	*nil*
s_4	6	11	8	5	2
Mate	2	4	*nil*	1	**27**

(a) Matching parziali

$\overline{\underline{c}}$	t_1	t_2	t_3	t_4	$\underline{u}$
s_0	6	8	2	0	8
s_1	0	4	1	0	11
s_2	5	4	9	0	10
s_3	0	0	0	4	5
s_4	1	6	3	−4	**40**

(b) Duale

Figura 4.20: Il predecessore di s_2 è s, Card = 3

Siccome Card < n è necessaria una nuova iterazione.

5

Minimum spanning trees (MST)

5.1 Minimum spanning trees

Problema 5.1.1 (Connector Problem). Dato un grafo connesso G = (V, E) ed un costo positivo $c_e \in \mathbb{R}^+ \forall e \in E$, trovare il sottografo connesso di G a costo minimo.

Lemma 5.1.2. Un lato $e = uv$ di G è un lato di un circuito in G se e solo se esiste un cammino in $G \setminus \{e\}$ da u a v.

Di conseguenza se eliminiamo un lato in un circuito di un grafo connesso, il grafo risultante rimane connesso ed ogni soluzione ottima al *connector problem* non avrà circuiti.

Definizione 5.1.3 (Albero e Foresta). Un grafo privo di circuiti è detto **foresta**, mentre una foresta connessa è detta **albero**.

È possibile risolvere il Connector Problem risolvendo il problema del Minimum Spanning Tree (MST):

Problema 5.1.4 (Minimum Spanning Tree Problem). Dato un grafo connesso G = (V, E) ed un costo $c_e \in \mathbb{R} \forall e \in E$, trovare l'albero ricoprente a costo minimo in G.

Lemma 5.1.5. Un sottografo ricoprente connesso di G è un albero ricoprente se e solo se ha esattamente $n-1$ lati.

5.2 Algoritmi che risolvono MST

5.2.1 Algoritmo di Kruskal

Sia $H = (V, F)$ una foresta ricoprente di G, con $F = \emptyset$ inizialmente.

A ogni passo viene aggiunto ad F il lato a costo minimo $e \notin F$ tale che H rimane una foresta.

Il procedimento si interrompe quando H diviene un albero ricoprente.

5.3 Kruskal step by step

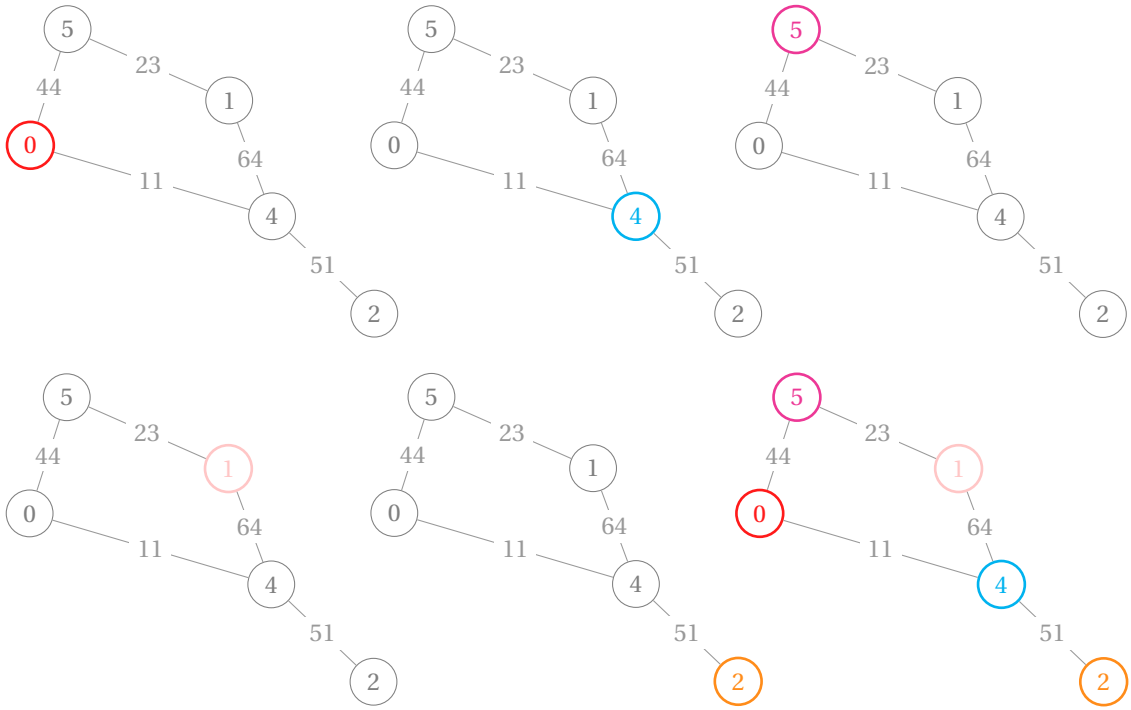


Figura 5.1: Initial conditions

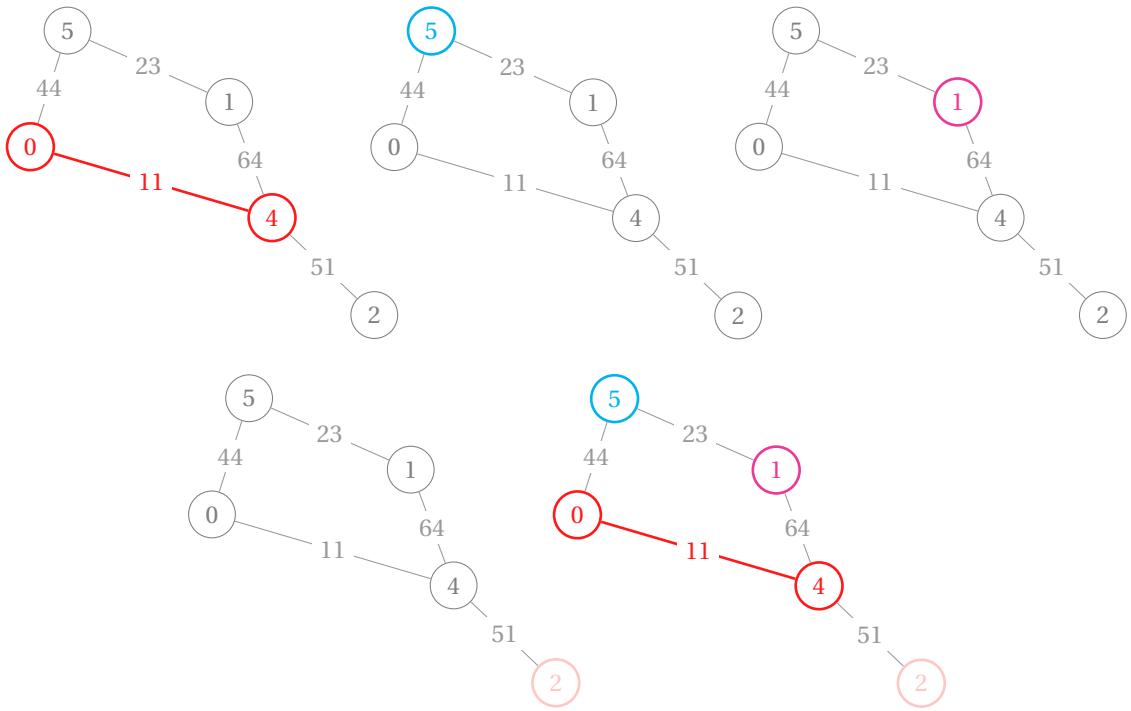


Figura 5.2: Iteration 0: Merging component 0 with component 1 via edge (0, 4)

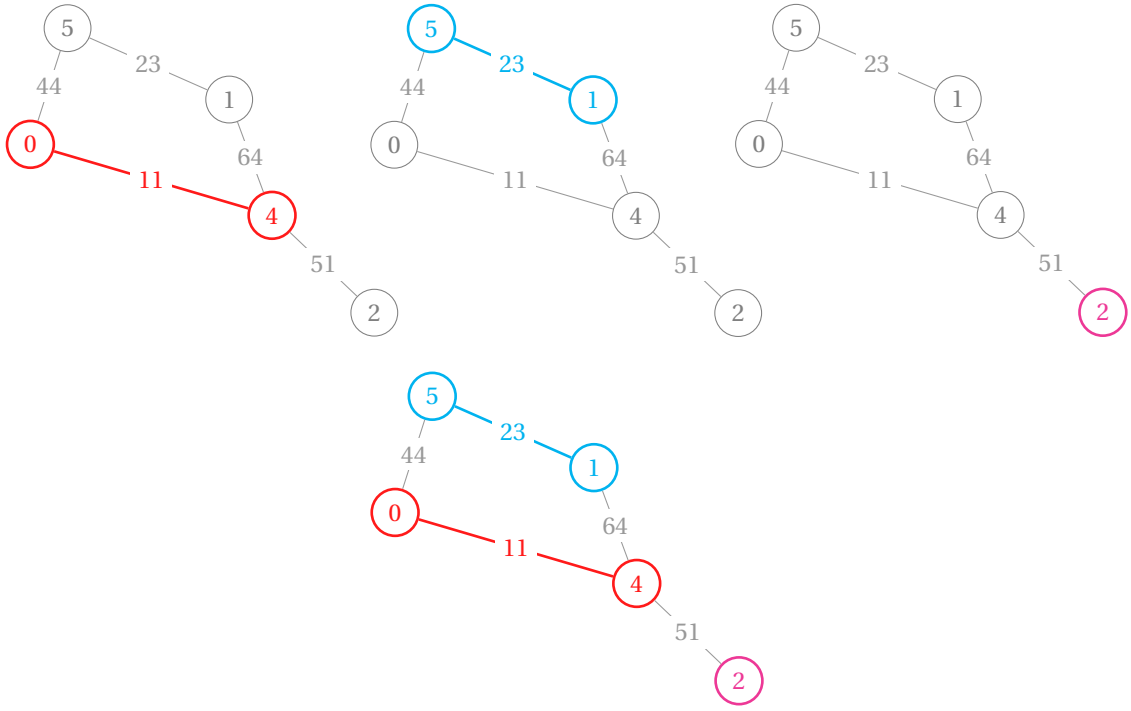


Figura 5.3: Iteration 1: Merging component 1 with component 2 via edge (5, 1)

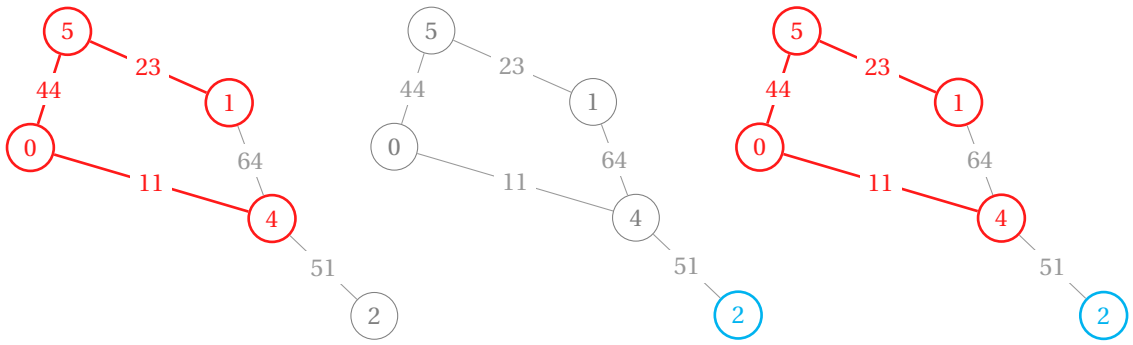


Figura 5.4: Iteration 2: Merging component 0 with component 1 via edge (0, 5)

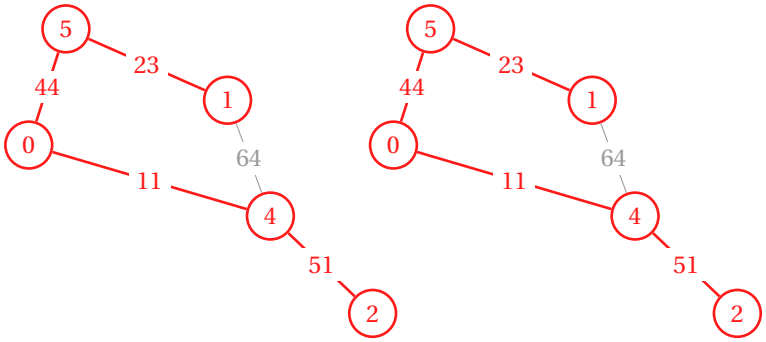


Figura 5.5: Iteration 3: Merging component 0 with component 1 via edge (4, 2)

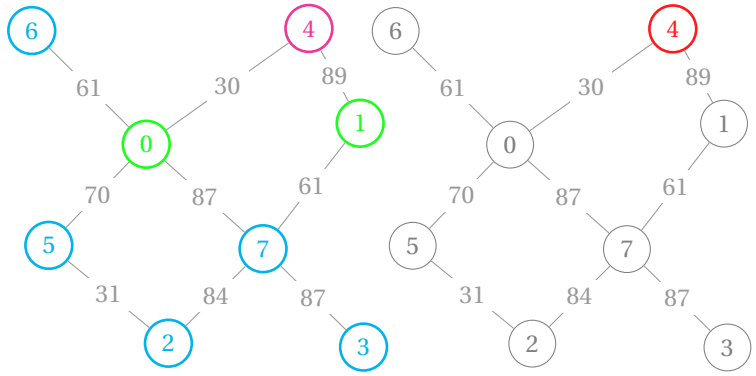
5.3.1 Algoritmo di Prim

Sia $H = (V(H), T)$ un albero di G = (V, E) con inizialmente $V(H) = \{r\}$ per qualche $r \in V$ e $T = \emptyset$.

A ogni passo viene aggiunto a T il lato a costo minimo $e \notin T$ tale che H rimanga un albero.

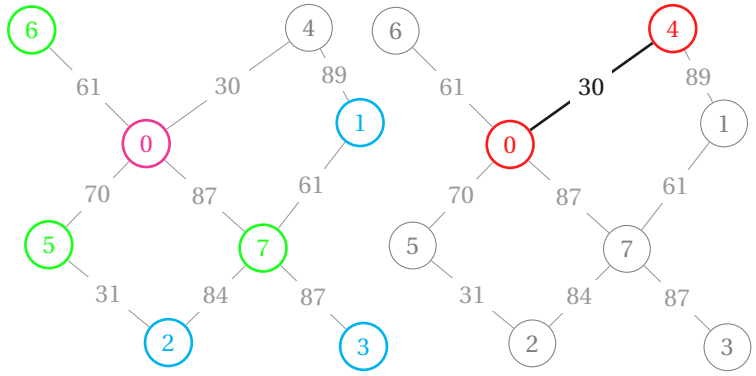
Il procedimento si interrompe quando H diviene un albero ricoprente.

5.4 Prim step by step



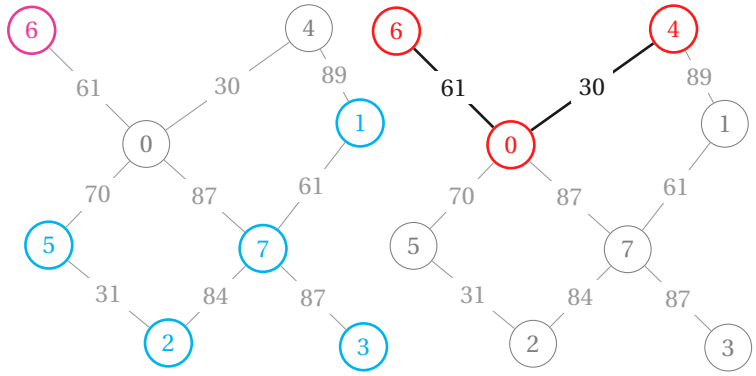
(a) (b)

Figura 5.6: Iteration 0: adding node 4.



(a) (b)

Figura 5.7: Iteration 1: adding node 0.



(a) (b)

Figura 5.8: Iteration 2: adding node 6.

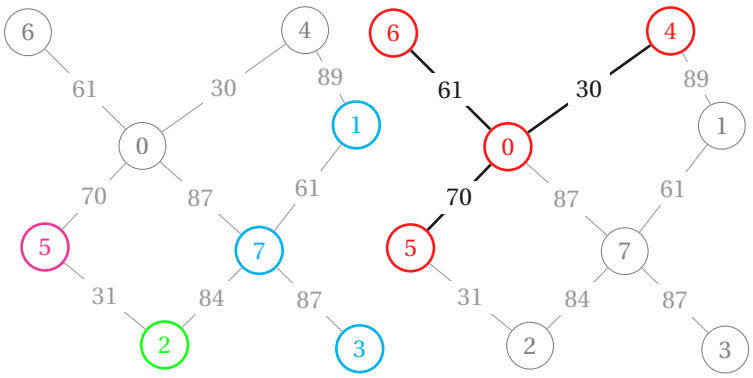


Figura 5.9: Iteration 3: adding node 5.

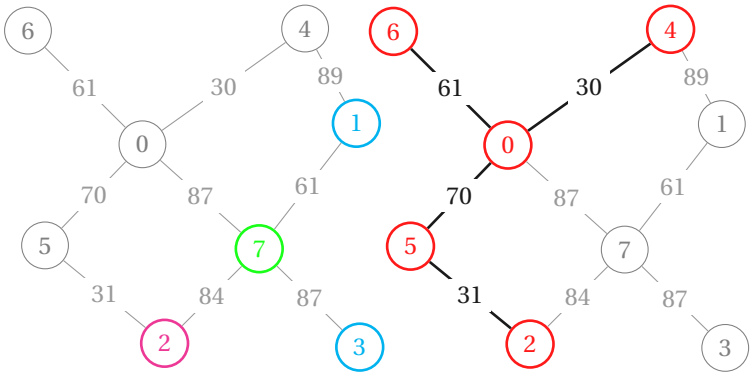


Figura 5.10: Iteration 4: adding node 2.

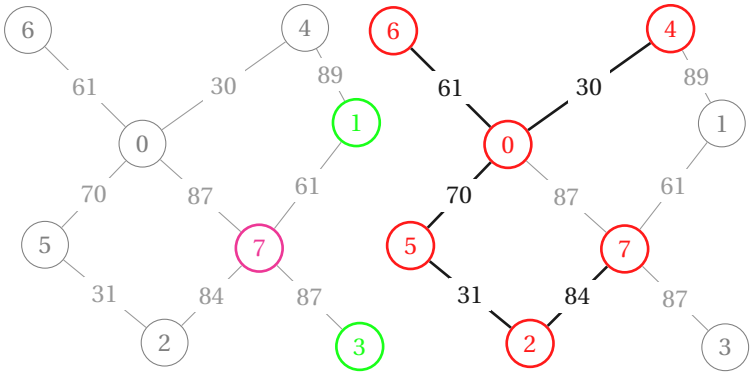
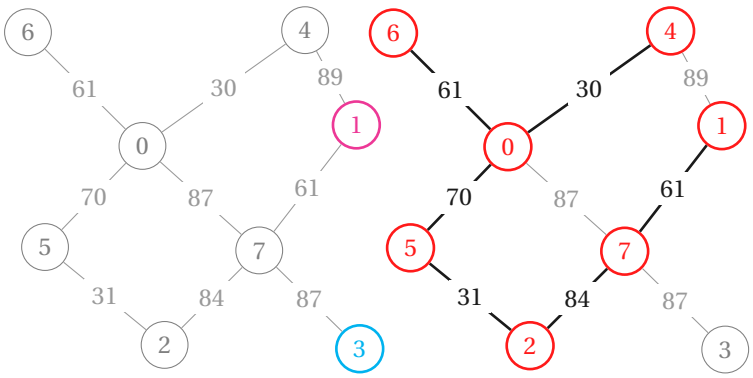
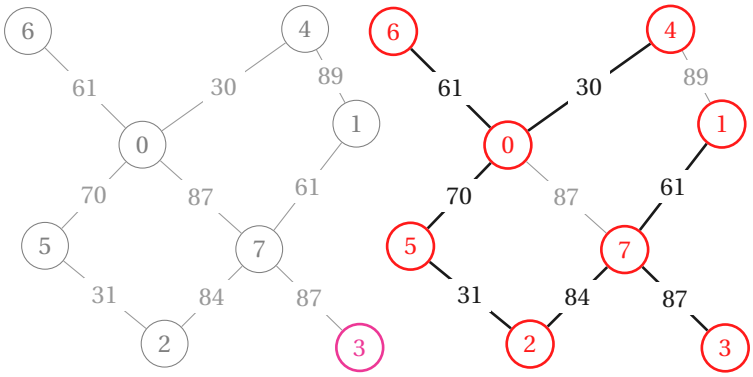


Figura 5.11: Iteration 5: adding node 7.



(a) (b)

Figura 5.12: Iteration 6: adding node 1.



(a) (b)

Figura 5.13: Iteration 7: adding node 3.

5.4.1 Validità degli algoritmi su MST

Definizione 5.4.1 (Taglio). Dato un grafo G = (V, E), chiamiamo $\delta(A)$ con $A \subseteq V$ l'insieme:

$$\delta(A) = \{e \in E : \text{entrambi i vertici di } e \text{ sono in } A\}$$

Questo insieme per qualche A è detto **taglio** di G.

Teorema 5.4.2 (Tagli di grafi connessi). Un grafo G = (V, E) è connesso se e solo se non esiste un insieme non vuoto $A \subset V$ tale che:

$$\delta(A) = \emptyset$$

Dimostrazione 5.4.3 (Tagli di grafi connessi). È chiaro che se $\delta(A) = \emptyset$, $u \in A$ e $v \notin A$, allora non ci può essere un cammino da u a v e quindi se $A \neq \emptyset$ e $A \subset V$ allora G non è connesso.

Dobbiamo dimostrare che, se G non è connesso, allora non esiste un tale set A: scegliamo quindi due vertici $u, v \in V$ tali che non esiste un cammino tra u a v.

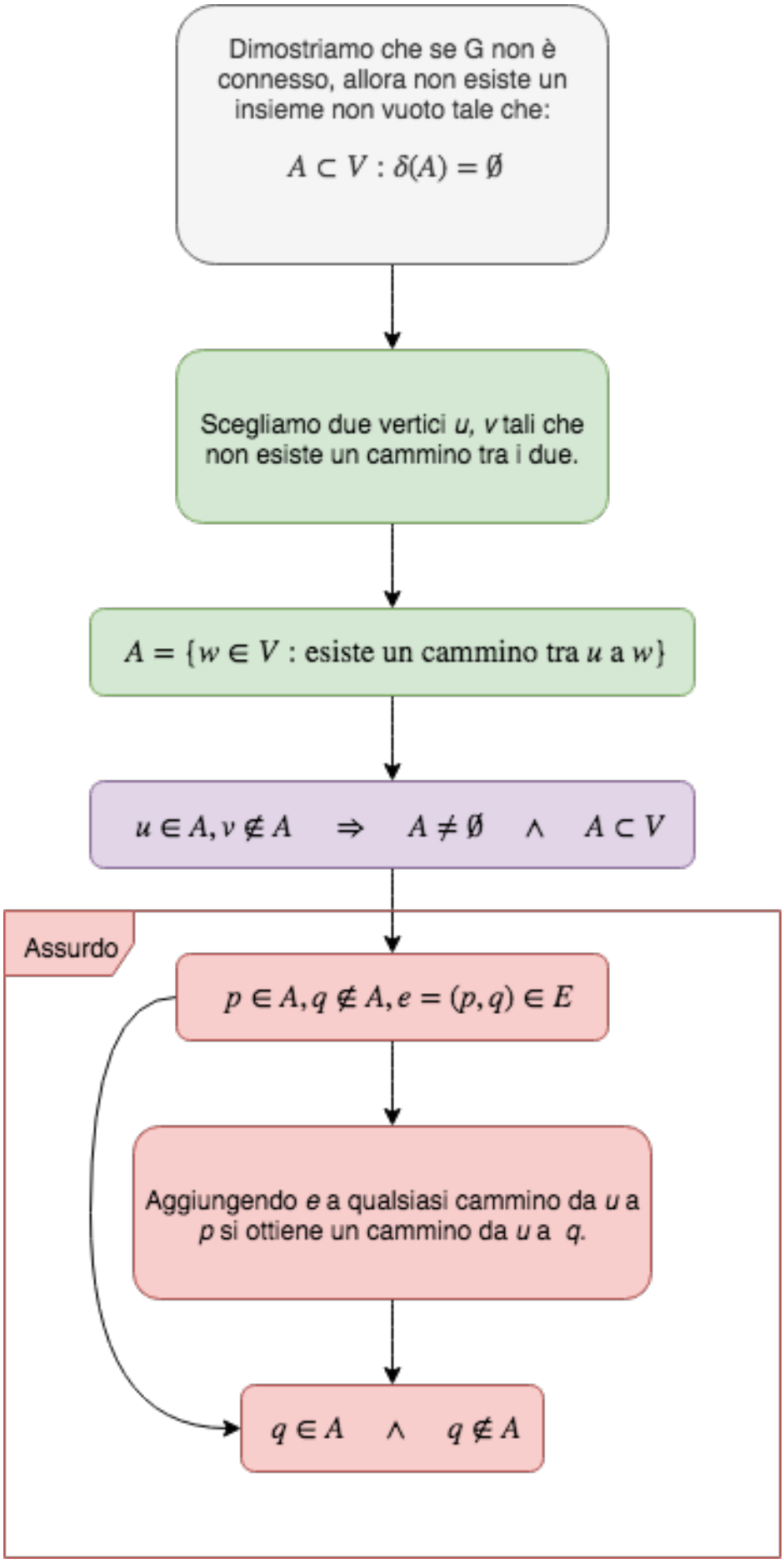
Definiamo A come:

$$A = \{w \in V : \text{esiste un cammino tra } u \text{ a } w\}$$

Quindi $u \in A$ e $v \notin A$, pertanto $A \neq \emptyset$ e $A \subset V$.

Per dimostrare che $\delta(A) = \emptyset$ procediamo per assurdo: supponiamo che $p \in A$, $q \notin A$ e che $e = (p, q) \in E$. Aggiungendo e a qualsiasi cammino da u a p si ottiene un cammino da u a q, contraddicendo il fatto che $q \notin A$. □

5.4.2 Diagramma della dimostrazione - Tagli di grafi connessi



Definizione 5.4.4 (Sottografo estensibile a MST). Un sottoinsieme di lati di G = (V, E) $A \subseteq E$ è estensibile a un MST se A è contenuto in un insieme di lati di qualche MST di G.

Teorema 5.4.5 (Estensione a MST). Dato un grafo G = (V, E), sia $B \subseteq E$ un sottoinsieme estendibile a MST e e sia un lato a costo minimo di qualche taglio D tale che $D \cap B = \emptyset$. Allora $B \cup \{e\}$ è estendibile ad un MST.

Lemma 5.4.6 (Alberi ricoprenti). Sia $H = (V, T)$ un albero ricoprente di G, $e = (v, w)$ un lato di G ma non di H e sia f un lato di un cammino semplice in T da v a w. Allora il sottografo:

$$H' = \big(V, (T \cup \{e\}) \setminus \{f\}\big)$$

è un albero ricoprente di G.

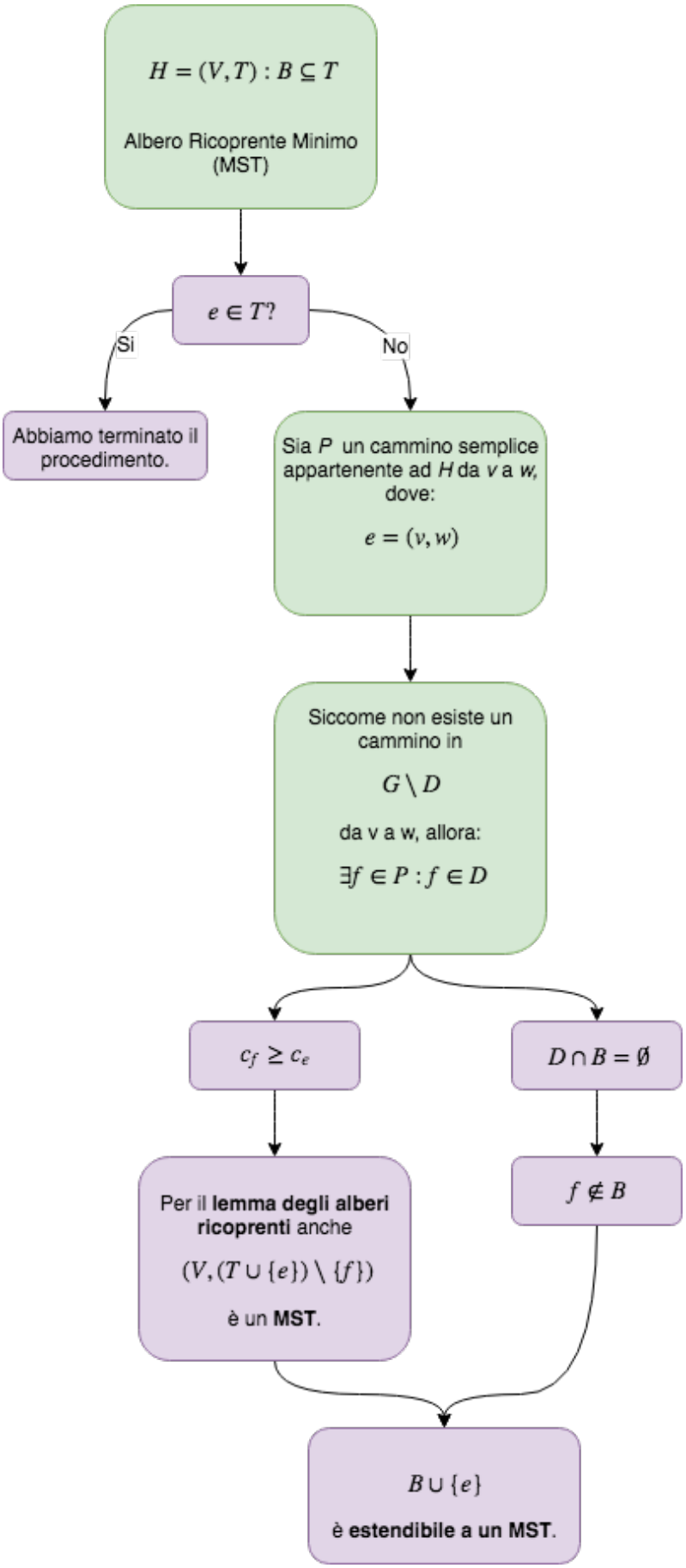
Dimostrazione 5.4.7 (Estensione a MST). Sia $H = (V, T)$ un MST tale che $B \subseteq T$. Se $e \in T$, allora abbiamo terminato il procedimento.

Altrimenti, sia P un cammino semplice appartenente a H da v a w, dove $e = (v, w)$. Siccome non esiste nessun cammino in $G \setminus D$ da v a w, esiste un lato $f \in P$ tale che $f \in D$.

Quindi $c_f \geqslant c_e$, e per il lemma sugli alberi ricoprenti anche $\big(V, (T \cup \{e\}) \setminus \{f\}\big)$ è un MST.

Siccome $D \cap B = \emptyset$, ne segue che $f \notin B$, pertanto $B \cup \{e\}$ è estendibile a un MST. □

5.4.3 Diagramma della dimostrazione - Estensione a MST



5.5 MST e programmazione lineare

Teorema 5.5.1 (Teorema di Edmonds). Sia $\underline{x}^*$ il vettore caratteristico di un MST in rispetto ai costi $\underline{c}$. ALlora $\underline{x}^*$ è una soluzione ottima del problema di programmazione lineare:

$$\begin{aligned} &\min \underline{c}^T \underline{x} \\ &x\left(\gamma(S)\right) \leqslant |S| - 1 \quad \forall S \neq \emptyset, S \subset V \\ &x(E) = |V| - 1 \\ &x_e \geqslant 0 \quad \forall e \in E \end{aligned}$$

Dimostrazione 5.5.2 (Teorema di Edmonds). Per un sottoinsieme di lati A, sia $k(A)$ il numero di componenti del sottografo (V, A) di G. Prendiamo in considerazione una versione equivalente del problema proposto nel teorema:

$$\begin{aligned} &\min \underline{c}^T \underline{x} \\ &x(A) \leqslant |V| - k(A) \quad \forall A \subset E \\ &x(E) = |V| - 1 \\ &x_e \geqslant 0 \quad \forall e \in E \end{aligned}$$

Sia $A \subseteq E$ e siano $S_1, \ldots, S_k$ gli insiemi di vertici delle componenti del sottografo (V, A). Allora:

$$x(A) \leqslant \sum_{i=1}^{k} x\left(\gamma(S_i)\right) \leqslant \sum_{i=1}^{k} (|S_i| - 1) = |V| - k$$

Procediamo ora a mostrare che $\underline{x}^*$ è la soluzione ottima del problema semplificato proposto e che è sufficiente perché ciò sia vero che essa è il vettore caratteristico di un albero ricoprente T generato dall'algoritmo di Kruskal.

Mostriamo che $\underline{x}^*$ è ottimale mostrando che l'algoritmo di Kruskal può essere utilizzato per calcolare una soluzione ammissibile al problema PL duale che soddisfa gli scarti complementari con $\underline{x}^*$. Riscrivendo la funzione obbiettivo del problema primale come $\max -\underline{c}^T \underline{x}$, il duale risulta:

$$\begin{aligned} \min \sum_{A \subseteq W} (|V| - k(A))\, \underline{y}_A \\ \sum \left(\underline{y}_a : e \in A\right) \geqslant -c_e, \quad \forall e \in E \\ \underline{y}_A \geqslant 0, \quad \forall A \subset E \end{aligned}$$

Sia $e_1, \ldots, e_m$ l'ordine con cui l'algoritmo di Kruskal considera i lati. Sia $R_i = \{e_1, \ldots, e_i\}, \forall i \in \mathbb{N} \cap [1, m]$.

Sia $\underline{y}^o_A = 0$ a meno che $\exists i : A = R_i$, e assegniamo $\underline{y}^o_{R_i} = c_{e_{i+1}} - c_{e_i} \, \forall i \in \mathbb{N} \cap [1, m-1]$. Infine assegniamo $\underline{y}^o_{R_m} = -c_{e_m}$.

Segue dall'ordine dei lati considerati dall'algoritmo che $\underline{y}^o_A \geqslant 0$ se $A \neq E$. Considerando ora il vincolo a sommatoria del problema, dove $e = e_i$ abbiamo:

$$\sum_{\underline{y}^o_A : e \in A} = \sum_{j=i}^{m} \underline{y}^o_{R_j} = \sum_{j=i}^{m-1} \left(c_{e_{i+1}} - c_{e_i}\right) - c_{e_m} = -c_{e_i} = -c_e$$

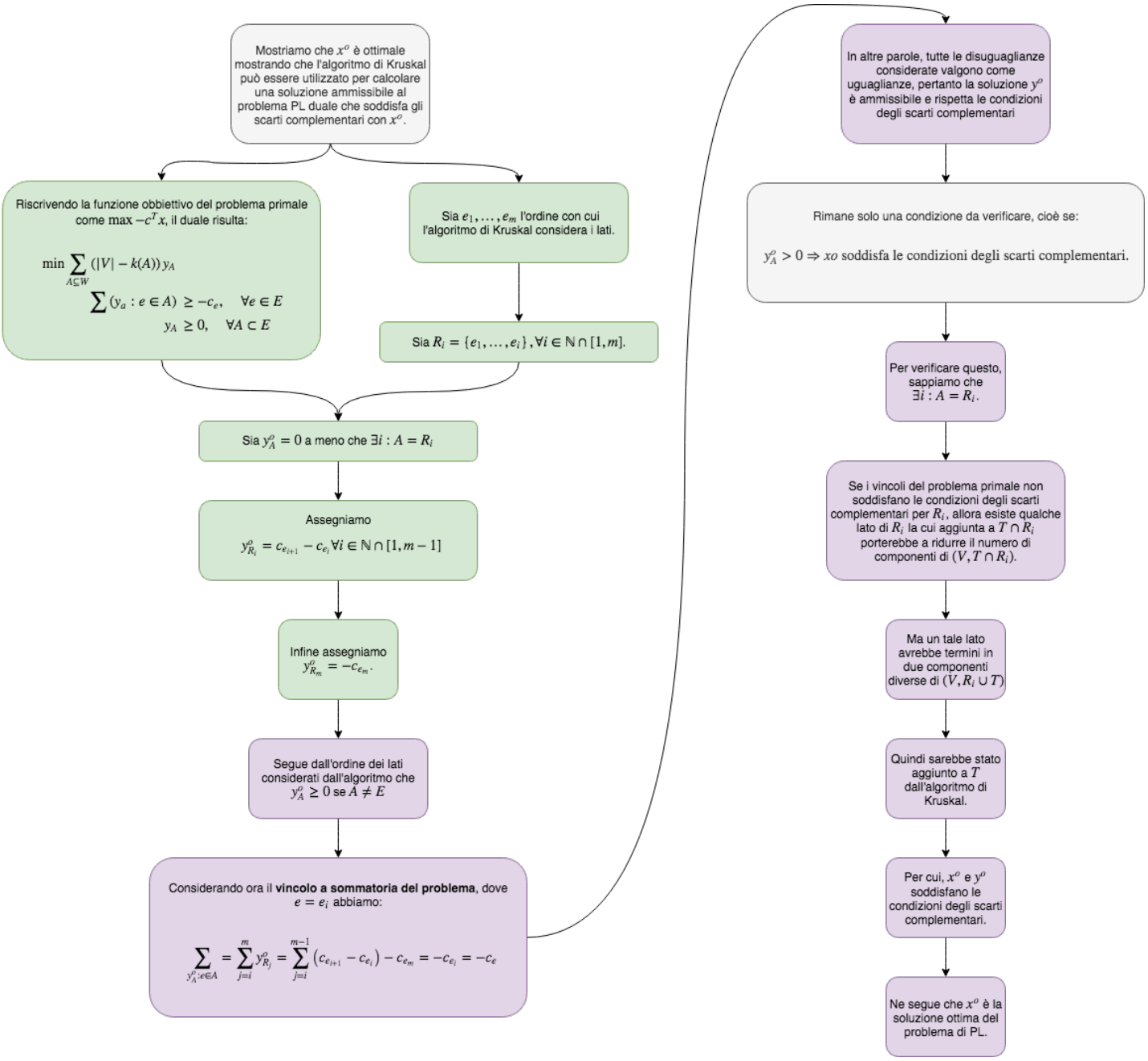
In altre parole, tutte le disuguaglianze considerate valgono come uguaglianze, pertanto la soluzione $\underline{y}^o$ è ammissibile e rispetta le condizioni degli scarti complementari: rimane solo una condizione da verificare, cioè se:

$$\underline{y}^o_A > 0 \Rightarrow \underline{\underline{x}^*} \text{ soddisfa le condizioni degli scarti complementari.}$$

Per verificare questo, sappiamo che $\exists i : A = R_i$. Se i vincoli del problema primale non soddisfano le condizioni degli scarti complementari per R_i, allora esiste qualche lato di R_i la cui aggiunta a $T \cap R_i$ porterebbe a ridurre il numero di componenti di $(V, T \cap R_i)$. Ma un tale lato avrebbe termini in due componenti diverse di $(V, R_i \cup T)$, e quindi sarebbe stato aggiunto a T dall'algoritmo di Kruskal.

Per cui, $\underline{\underline{x}^*}$ e $\underline{y}^o$ soddisfano le condizioni degli scarti complementari. Ne segue che $\underline{\underline{x}^*}$ è la soluzione ottima del problema di PL. □

5.5.1 Diagramma della dimostrazione - Teorema di Edmonds



6

Matroidi e Algoritmo Greedy

6.1 Sistema di Indipendenza

Definizione 6.1.1 (Sistema di Indipendenza (SI)). Sia $E = \{e_1, \ldots, e_n\}$ e $F \subseteq 2^E$ tale che per tutti $I, J \subseteq E$ valga la proprietà:

$$J \subset I \in F \rightarrow J \in F$$

Allora la coppia (E, F) è detta **Sistema di Indipendenza** ed i membri di F vengono detti **insiemi indipendenti**.

6.2 GREEDSUM

L'algoritmo riceve un insieme di elementi E ed una funzione di costo c ed inizia con l'insieme soluzione $I = \emptyset$.

Successivamente itera $|E|$ volte, scegliendo ogni volta l'elemento di E che massimizza la funzione c.

Questo elemento $e^* = \max_e c_E$ viene quindi rimosso dal set degli oggetti: $E = E - \{e^*\}$.

Se l'insieme $I \cup \{e^*\}$ risulta essere indipendente, allora il nuovo elemento viene aggiunto all'insieme di soluzione: $I = I \cup \{e^*\}$.

Definizione 6.2.1 (Matroide). Un SI che gode della proprietà del primo teorema di Rado 6.3.1 si dice **matroide**.

6.3 I teoremi di Rado

I teoremi di Rado consentono di fare un'importante affermazione su GREEDSUM: "GREEDSUM fornisce l'ottimo se e solo se il SI è un **matroide**".

Teorema 6.3.1 (Primo teorema di Rado). Sia (E,F) un Sistema di Indipendenza (SI). Se per due insiemi indipendenti I e J qualunque tali che $|I| = |J|+1$ vale la proprietà:

$$\exists e \in I : J \cup \{e\} \in F$$

allora **GREEDSUM** fornisce un membro di F di valore massimo per qualunque funzione di valutazione: $e : E \to \mathbb{R}^+$.

Dimostrazione 6.3.2 (Primo teorema di Rado). Iniziamo dimostrando che gli insiemi sono indipendenti massimali di pari cardinalità: la proprietà del teorema implica che se $A \subseteq E$ e $I, I' \subseteq A$ sono insiemi indipendenti massimali, allora $|I| = \left|I'\right|$.

Infatti se **per assurdo** $|I| < \left|I'\right|$, si potrebbe trovare un $I'' \subseteq I'$ tale che $\left|I''\right| = |I|+1$ eliminando da I' un numero di elementi pari a $\left|I'\right| - |I| - 1$, ma per la proprietà del teorema esiste $e \in I'' \setminus I$ tale che $I \cup \{e\} \in F$ e I non sarebbe massimale.

Quindi dimostriamo che GREEDSUM dia il risultato ottimo: supponiamo adesso ancora per assurdo che **GREEDSUM** non dia un risultato ottimo, cioè che per una certa funzione di valutazione $c : E \to \mathbb{R}^+$ l'algoritmo fornisce un insieme indipendente $I = \{e_1, \ldots, e_i\}$ mentre ne esiste un altro $J = \left\{e'_1, \ldots, e'_j\right\}$ tale che $c_J > c_I$.

Supponiamo senza perdita di generalità di aver ordinato gli elementi di I e J in ordine di valore non crescente.

Per costruzione I è massimale e quindi anche J per quanto visto sopra, per cui $i = j$. Vedremo che per $m = 1, 2, \ldots, i$ vale che $c_{e_m} \geqslant c_{e'_m}$, contraddicendo l'ipotesi per cui $c_J > c_I$.

Procediamo quindi per **induzione**:

Caso base: quando $m = 1$, la proprietà è vera per come **GREEDSUM** sceglie il primo elemento.

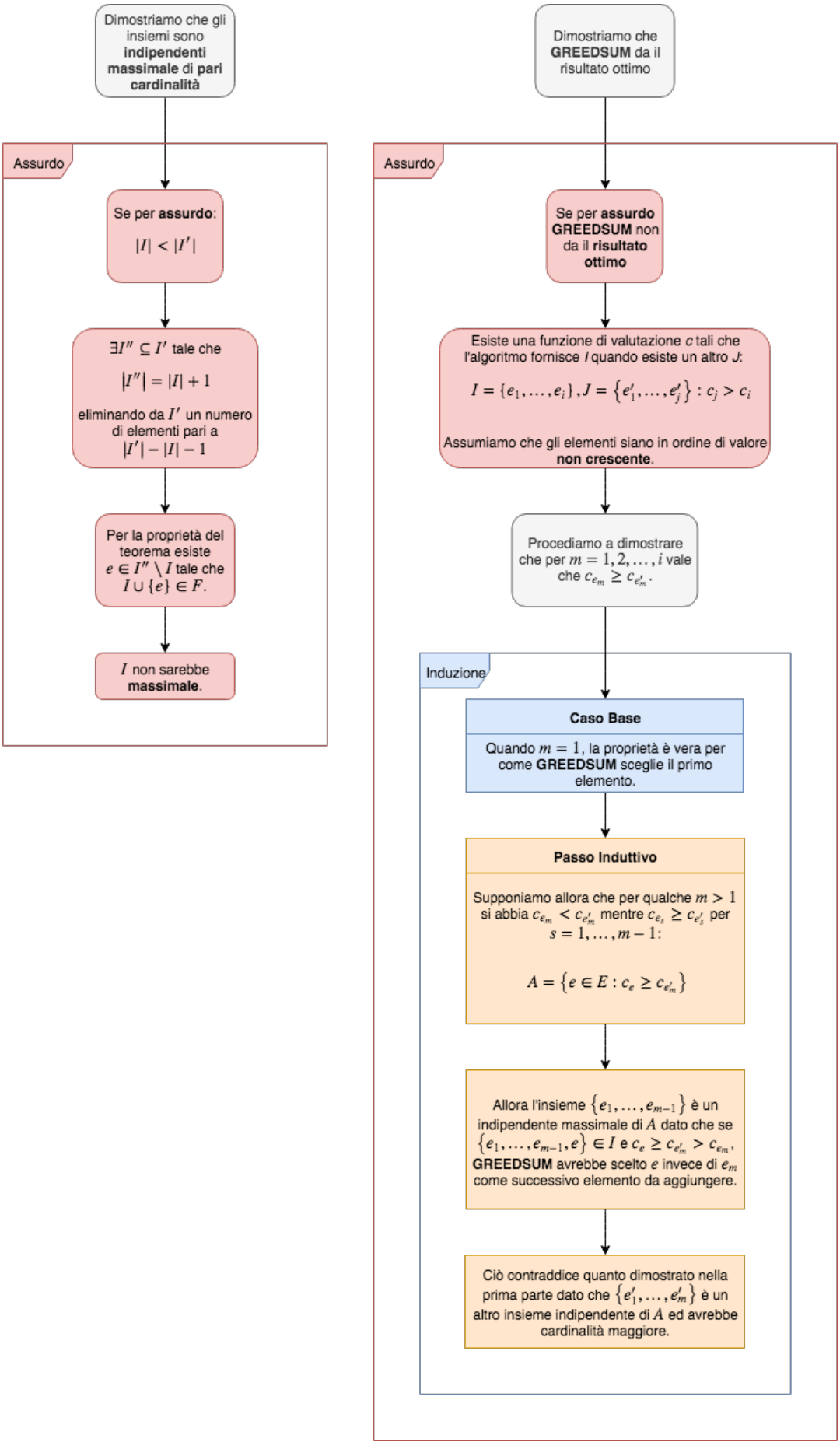
Passo induttivo: Supponiamo allora che per qualche $m > 1$ si abbia $c_{e_m} < c_{e'_m}$ mentre $c_{e_s} \geqslant c_{e'_s}$ per $s = 1, \ldots, m-1$:

$$A = \left\{e \in E : c_e \geqslant c_{e'_m}\right\}$$

Allora l'insieme $\{e_1, \ldots, e_{m-1}\}$ è un indipendente massimale di A dato che se $\{e_1, \ldots, e_{m-1}, e\} \in I$ e $c_e \geqslant c_{e'_m} > c_{e_m}$, **GREEDSUM** avrebbe scelto e invece di e_m come successivo elemento da aggiungere.

Ciò contraddice quanto dimostrato nella prima parte dato che $\left\{e'_1, \ldots, e'_m\right\}$ è un altro insieme indipendente di A ed avrebbe cardinalità maggiore. □

6.3.1 Diagramma della dimostrazione - Primo Teorema di Rado



Teorema 6.3.3 (Secondo teorema di Rado). Sia (E, F) un Sistema di Indipendenza (SI). Se per qualunque funzione di valutazione non negativa $c : E \to \mathbb{R}^+$ degli elementi di E, **GREEDSUM** fornisce sempre un membro di F di valore massimo, allora il SI gode della proprietà del primo teorema di Rado 6.3.1.

Dimostrazione 6.3.4 (Secondo teorema di Rado). Procediamo **per assurdo** e supponiamo che la proprietà del primo teorema di Rado 6.3.1 non valga.

Ciò significa che esistono due **insiemi indipendenti** I e J con $|I| = p$ e $|J| = p+1$ tali che per nessun elemento $e \in J \setminus I$ accade che $I \cup \{e\} \in I$.

Consideriamo allora la seguente maniera di valutare gli elementi di E:

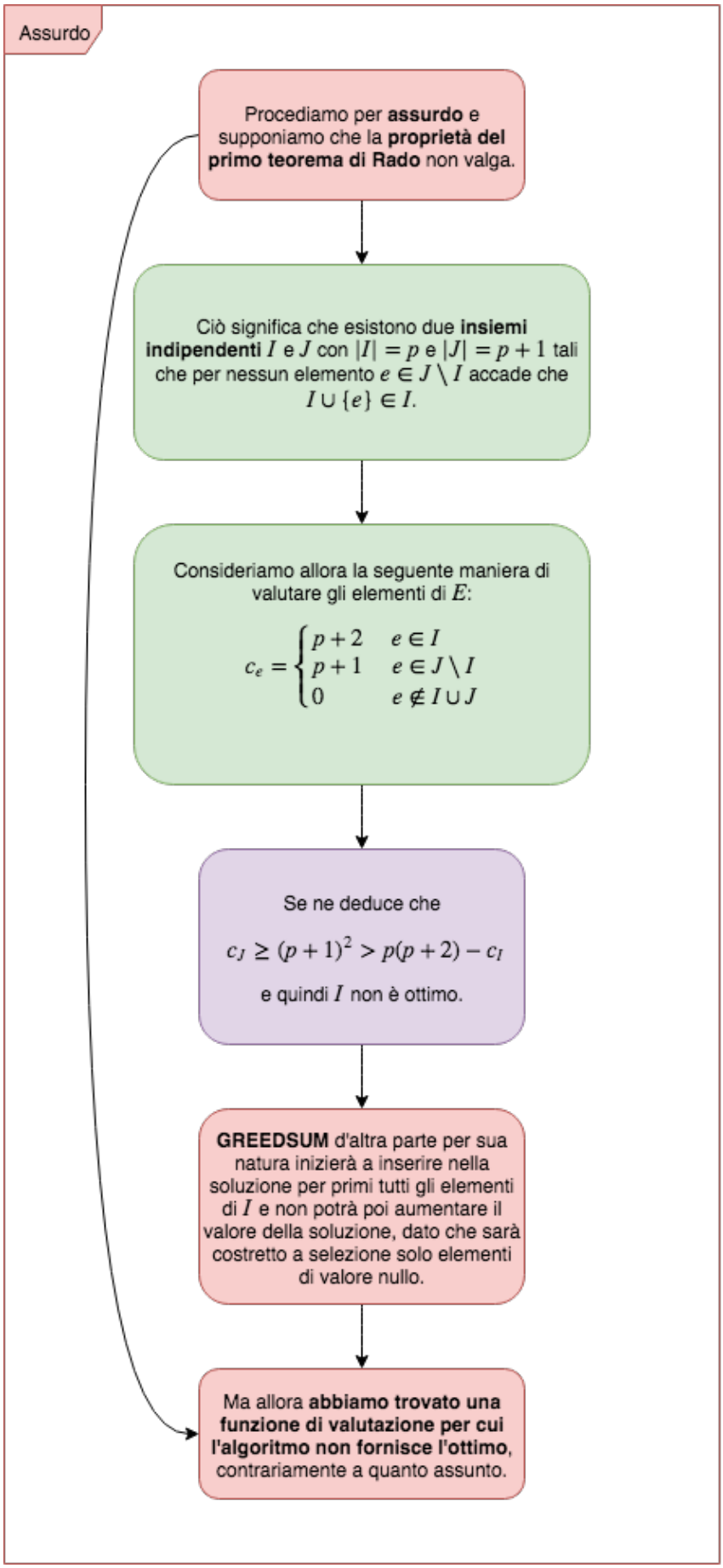
$$c_e = \begin{cases} p+2 & e \in I \\ p+1 & e \in J \setminus I \\ 0 & e \notin I \cup J \end{cases}$$

Se ne deduce che $c_J \geqslant (p+1)^2 > p(p+2) - c_I$ e quindi I non è ottimo.

GREEDSUM d'altra parte per sua natura inizierà a inserire nella soluzione per primi tutti gli elementi di I e non potrà poi aumentare il valore della soluzione, dato che sarà costretto a selezione solo elementi di valore nullo.

Ma allora abbiamo trovato una funzione di valutazione per cui l'algoritmo non fornisce l'ottimo, contrariamente a quanto assunto. □

6.3.2 Diagramma della dimostrazione - Secondo Teorema di Rado



7 Algoritmo di Dijkstra

7.1 Utilità

L'algoritmo di Dijkstra viene utilizzato per trovare il cammino più breve tra due nodi di un grafo.

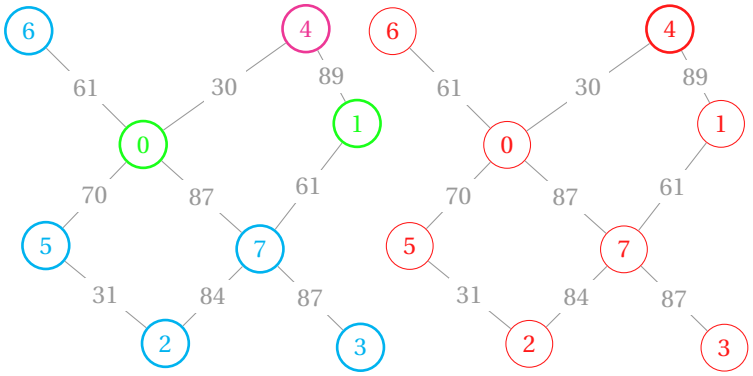
7.2 Complessità

Complessità computazionale 7.2.1 (Algoritmo di Dijkstra). L'implementazione dell'algoritmo di Dijkstra iniziale ha complessità $\mathcal{O}\left(|V|^2\right)$.

7.3 Il funzionamento di Dijkstra

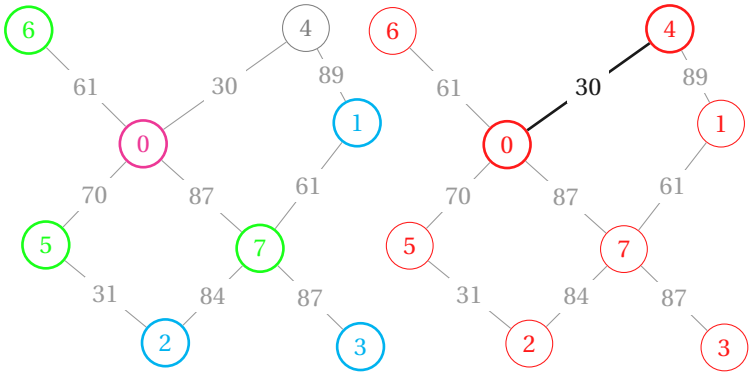
1. Inizialmente tutti i nodi sono marcati come **non visitati** e viene assegnata ad ognuno di essi un valore di distanza: 0 per il nodo iniziale ed ∞ per tutti gli altri.
2. Per il nodo corrente, prendiamo in considerazione tutti i nodi vicini non visitati: per ognuno calcoliamo la distanza dalla radice passando attraverso il nodo corrente e confrontiamo il valore di distanza preesistente con quello nuovo e manteniamo il valore minore.
3. Una volta determinata la distanza per tutti i vicini del nodo corrente marchiamo il nodo corrente come **visitato**.
4. Se il **nodo destinazione** è stato marcato visitato o se la lunghezza minore tra i nodi non visitati è infinita (come quando si vuole raggiungere un nodo non connesso) l'algoritmo si interrompe.
5. Altrimenti si sceglie un nodo non visitato a distanza minima: esso diviene il nuovo nodo corrente e l'operazione si ripete dal punto 2.

7.4 Step by step



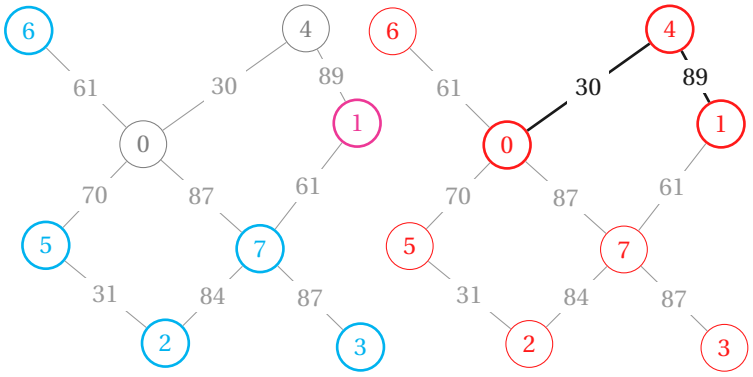
(a) Current step (b) Current Tree

Figura 7.1: Iteration 0



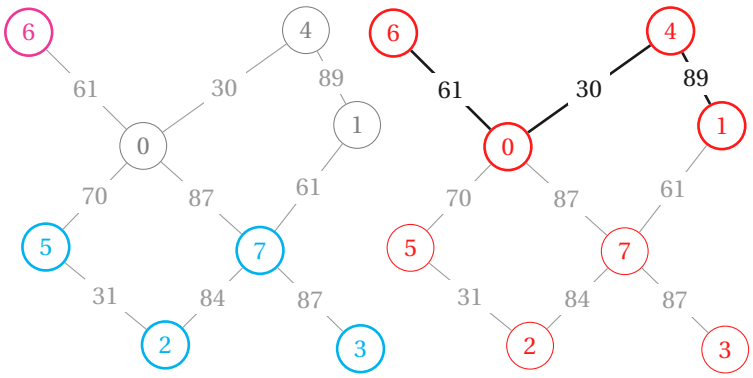
(a) Current step (b) Current Tree

Figura 7.2: Iteration 1



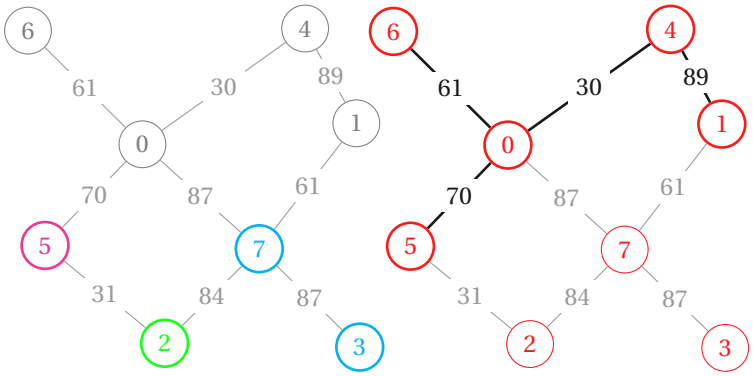
(a) Current step (b) Current Tree

Figura 7.3: Iteration 2



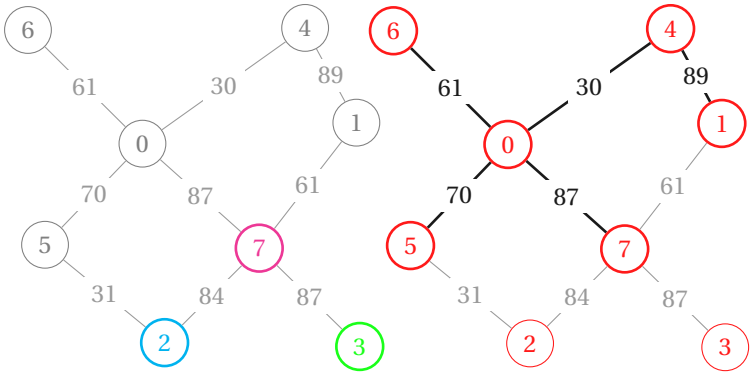
(a) Current step (b) Current Tree

Figura 7.4: Iteration 3



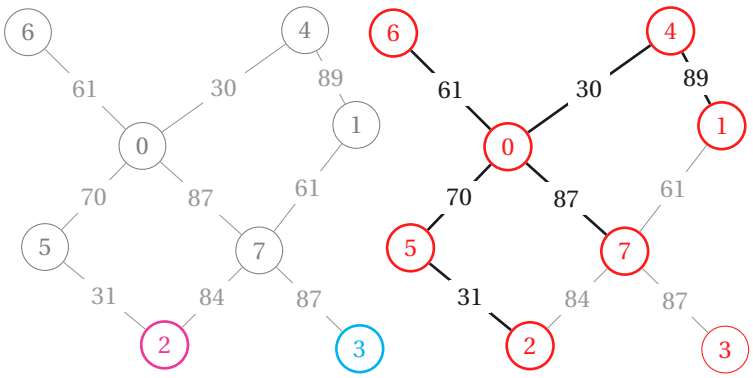
(a) Current step (b) Current Tree

Figura 7.5: Iteration 4



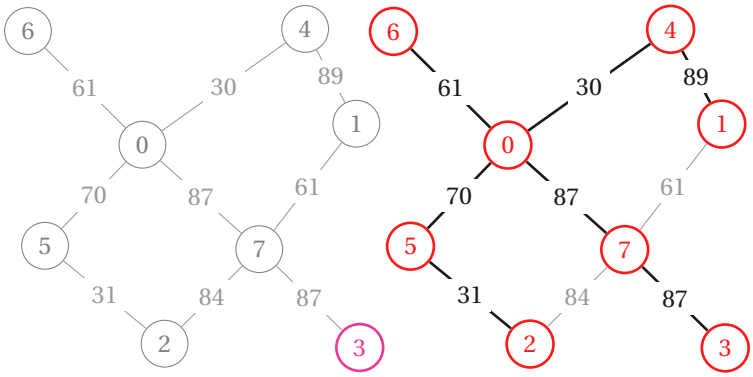
(a) Current step (b) Current Tree

Figura 7.6: Iteration 5



(a) Current step (b) Current Tree

Figura 7.7: Iteration 6



(a) Current step (b) Current Tree

Figura 7.8: Iteration 7

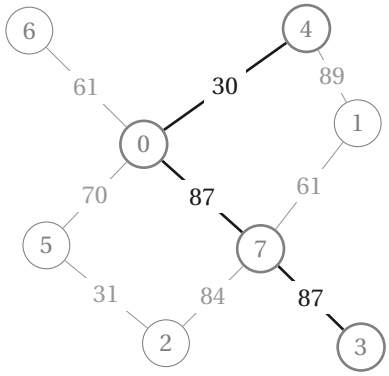


Figura 7.9: Completed Algorithm

7.5 Correttezza di Dijkstra

Dimostrazione 7.5.1 (Correttezza di Dijkstra). Procediamo a dimostrare la correttezza dell'algoritmo di Dijkstra **per induzione** sul numero di nodi visitati.

Per ogni nodo visitato v, la distanza $d(v)$ è la distanza minore dal nodo sorgente al nodo v. Per ogni non non visitato u, $d(u)$ è assunta essere la distanza più breve attraverso solo nodi visitati, dalla radice a u.

Questa assunzione vale solo se un cammino esiste, altrimenti la distanza è inizializzata a ∞.

Il caso base è quando esiste solo un nodo visitato, cioè il nodo sorgente, in tal caso l'ipotesi è banale.

Altrimenti, assumiamo l'ipotesi per $n-1$ nodi visitati. In tal caso, scegliamo un lato (v,u) dove u possiede la distanza minima di ogni nodo non visitato ed il lato (v,u) è tale che:

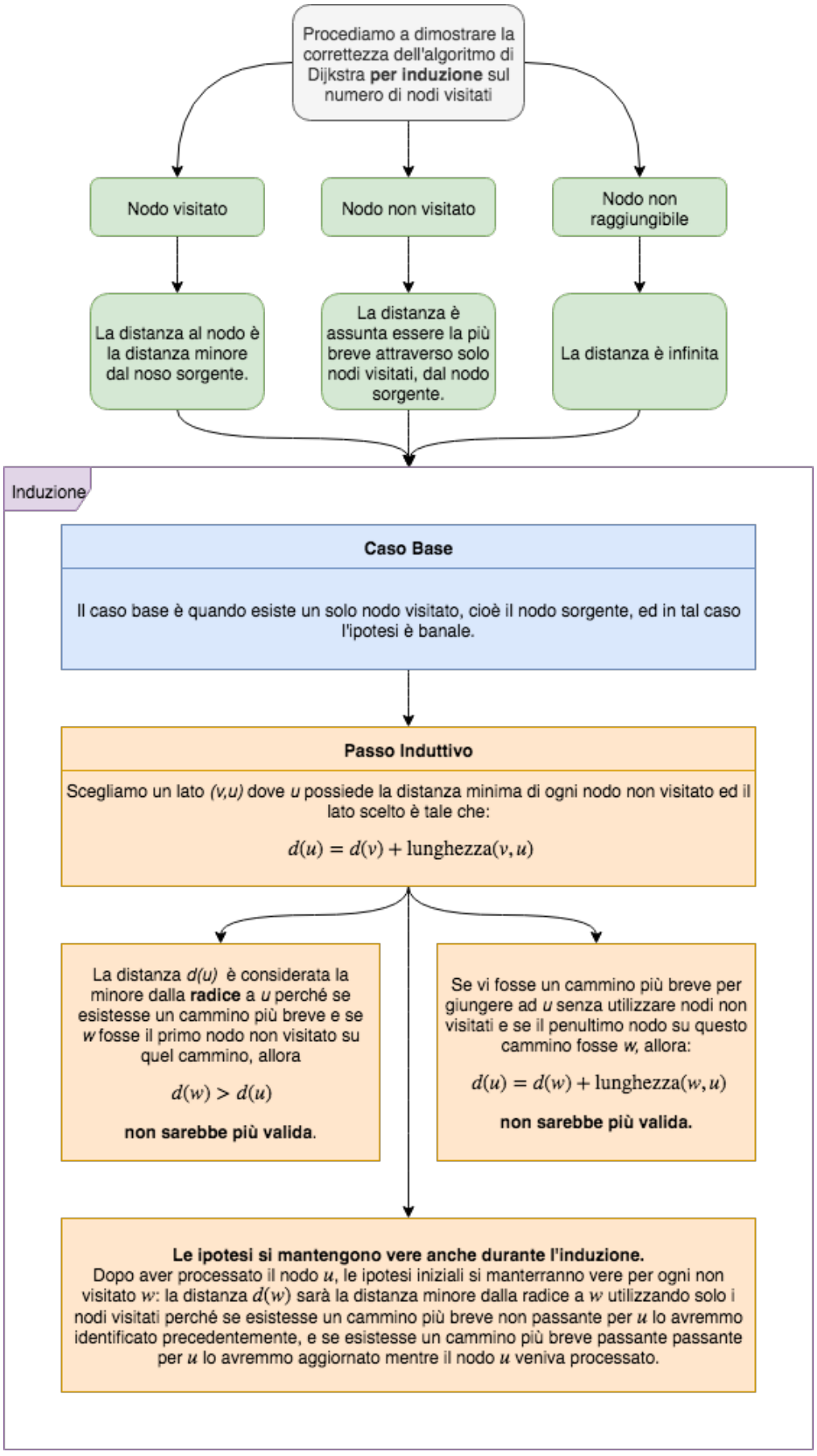
$$d(u) = d(v) + \text{lunghezza}(v,u)$$

La distanza $d(u)$ è considerata la distanza minore dalla **radice** a u perché se esistesse un cammino più breve e se w fosse il primo nodo non visitato su quel cammino allora l'ipotesi originale che $d(w) > d(u)$ **non sarebbe più valida**.

Similarmente se vi fosse un cammino più breve per giungere ad u senza utilizzare nodi non visitati e se il penultimo nodo su questo cammino fosse w, allora si avrebbe che $d(u) = d(w) + \text{lunghezza}(w,u)$ **che è nuovamente una contraddizione**.

Dopo aver processato il nodo u, le ipotesi iniziali si manterranno vere per ogni non visitato w: la distanza $d(w)$ sarà la distanza minore dalla radice a w utilizzando solo i nodi visitati perché se esistesse un cammino più breve non passante per u lo avremmo identificato precedentemente, e se esistesse un cammino più breve passante passante per u lo avremmo aggiornato mentre il nodo u veniva processato. □

7.5.1 Diagramma della dimostrazione - Correttezza di Dijkstra



8

Algoritmo di Ford-Fulkerson

8.1 Utilità

Si tratta di un **algoritmo greedy** che calcola il **flusso massimo** un un grafo. Spesso è chiamato *metodo* invece di *algoritmo* perché l'approccio per trovare i cammini aumentanti nel grafo residuo non è completamente specificato. Spesso viene implementato completamente nell'algoritmo di **Edmonds-Karp**, in cui si procede a scegliere ad ogni iterazione il **cammino aumentante più corto**.

Definizione 8.1.1 (Cammino aumentante). Un cammino P da s a t tale che ogni arco $(i,j) \in P$ orientato verso t (in avanti o forward) ha capacità $x_{ij} < u_{ij}$ ed ogni arco orientato verso s (all'indietro o reverse) ha capacità $x_{ji} > 0$ si dice **cammino aumentante**.

Definizione 8.1.2 (Cammino incrementante). Un cammino P da s a v tale che $v \neq t$ e per ogni arco $(i,j) \in P$ forward ha capacità $x_{ij} < u_{ij}$ e per ogni arco $(i,j) \in P$ reverse ha capacità $x_{ji} > 0$ si dice **cammino incrementante**.

8.2 Algoritmo di Edmonds-Karp

Lemma 8.2.1 (Primo lemma di Edmonds-Karp). Sia $d_x(v,w)$ la lunghezza del cammino da v a w avente il minor numero di archi e siano x e x' il flusso alla iterazione i ed $i+1$.

$$\forall v \in \mathbb{N}, \quad d_{x'}(s,v) \geqslant d_x(s,v) \quad \wedge \quad d_{x'}(v,t) \geqslant d_x(v,t)$$

La lunghezza dei cammini passanti dalla sorgente ad ogni nodo e dal ogni nodo alla destinazione da una iterazione alla successiva, aumenta o rimane uguale.

Dimostrazione 8.2.2 (Primo lemma di Edmonds-Karp). Iniziamo dividendo il lemma in due parti:

$$\forall v \in \mathbb{N}, \quad \underbrace{d_{x'}(s,v) \geqslant d_x(s,v)}_{\text{Prima parte}} \quad \wedge \quad \underbrace{d_{x'}(v,t) \geqslant d_x(v,t)}_{\text{Seconda parte}}$$

Procediamo ora a dimostrare la prima parte del lemma: procediamo per assurdo e supponiamo che esista un nodo v tale che $d_{x'}(s,v) < d_x(s,v)$ e scegliamo v in modo che $d_{x'}(s,v)$ sia la distanza più piccola il possibile.

Essendo $v \neq s$, $d_{x'}(s,v) > 0$.

Sia P' un cammino (s,v) in $G_{x'}$ e w il penultimo nodo di P'. Si ha che:

$$d_x(s,v) > d_{x'}(s,v) = d_{x'}(s,w) + 1 \geqslant d_x(s,w) + 1$$

Se $d_x(s,v) > d_x(s,w) + 1$, allora l'arco (w,v) non appartiene a G_x, perché altrimenti $d_x(s,v) = d_x(s,w) + 1$.

Se l'arco (w,v), che appartiene a $G_{x'}$ non appartiene a G_x allora esiste un indice $i : w = v_i$ e $v = v_{i-1}$, pertanto:

$$d_x(s,v) = i - 1 > d_x(s,w) + 1 = i + 1$$

e si ha una **contraddizione**.

Analogamente si procede a dimostrare la seconda parte del lemma. □

Lemma 8.2.3 (Secondo lemma di Edmonds-Karp). Durante l'esecuzione dell'algoritmo di Edmonds e Karp, un arco (i,j) scompare (e compare) in G_x al più $\frac{n}{2}$ volte.

Dimostrazione 8.2.4 (Secondo lemma di Edmonds-Karp). Se un arco (i,j) "scompare" dalla rete ausiliaria, significa che esso è su un cammino aumentate e che il corrispondente arco in G si satura oppure si svuota. Pertanto, nella rete ausiliaria successiva compare l'arco (j,i). Sia x_f il flusso al momento della "scomparsa dell'arco".

Supponiamo che ad una successiva iterazione l'arco (i,j) ricompaia in G_{x_h}. Ciò significa che il cammino aumentante che ha generato x_h contiene l'arco (j,i).

Allora se x_g è il flusso a partire dal quale si è generato x_h, si ha per il primo lemma 8.2.1 che:

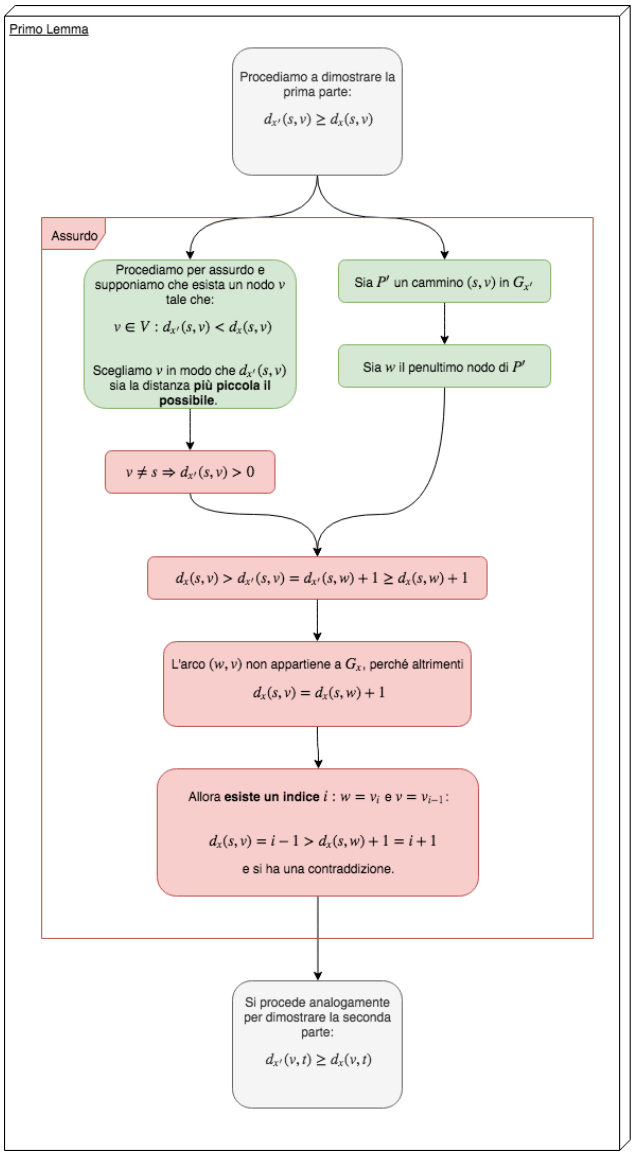
$$d_g(s,i) = d_g(s,j) + 1 \geqslant d_f(s,j) + 1 = d_f(s,i) + 2$$

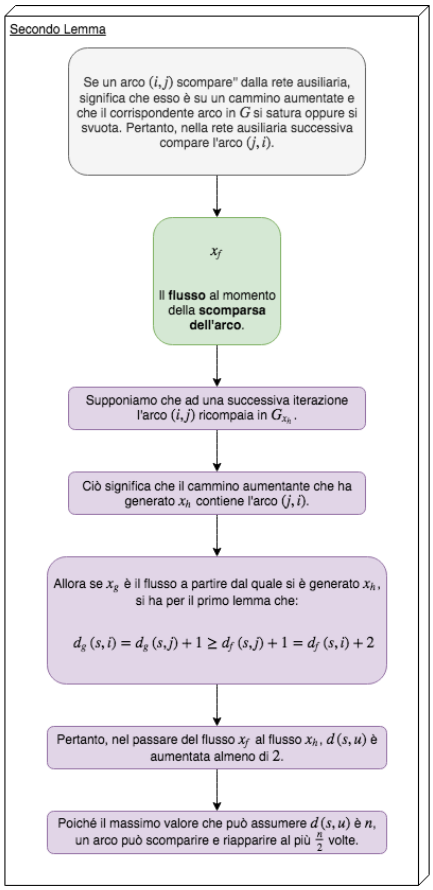
Pertanto, nel passare del flusso x_f al flusso x_h, $d(s,u)$ è aumentata almeno di 2. Poiché il massimo valore che può assumere $d(s,u)$ è n, un arco può scomparire e riapparire al più $\frac{n}{2}$ volte. □

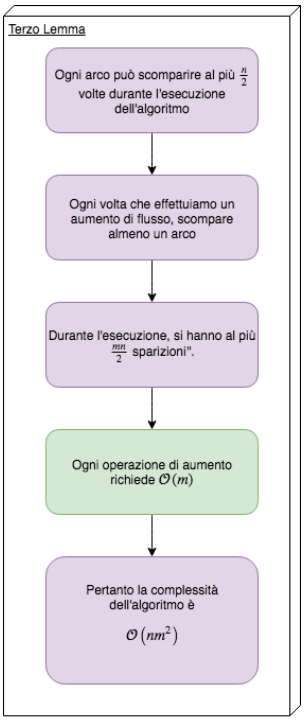
Lemma 8.2.5 (Terzo lemma di Edmonds-Karp). L'algoritmo di Edmonds-Karp ha complessità $\mathcal{O}(nm^2)$.

Dimostrazione 8.2.6 (Terzo lemma di Edmonds-Karp). Ogni arco può scomparire al più $\frac{n}{2}$ volte durante l'esecuzione dell'algoritmo per il secondo lemma 8.2.3. Ogni volta che effettuiamo un aumento di flusso, scompare almeno un arco. Pertanto, durante l'esecuzione, si hanno al più $\frac{mn}{2}$ "sparizioni". Ogni operazione di aumento richiede $\mathcal{O}(m)$ e, quindi, la complessità dell'algoritmo è $\mathcal{O}(nm^2)$. □

8.2.1 Diagramma della dimostrazione - Lemmi di Edmonds-Karp







9 Algoritmo Push-Relabel

L'idea generale è che si risolve un problema primale usando una soluzione inammissibile: il rapporto tra taglio e flusso si mantiene, mentre i vincoli di bilancio dei flussi non sono mantenuti.

Vengono quindi aggiornati nella fase di relabel i vincoli del flusso e quando il quasi-flusso diviene ammissibile questo risulta un flusso massimo, una soluzione ottima.

L'algoritmo **push-relabel** utilizza una funzione di etichettatura valida non negativa che utilizza **etichette di distanza** (dette anche altezze), su nodi per determinare quali archi dovrà andare a selezionare per l'operazione di push. Questa funzione di etichettatura viene denotata come $d : V \to \mathbb{N}$.

L'algoritmo inizia creando un grafo residuale, inizializzando i valori a zero e saturando tramite operazioni di push gli archi residuali all'uscita della sorgente. Quindi, le etichette sono inizializzate in modo tale che la sorgente coincide con il numero dei nodi nel grafo: $d(s) = |V|$ e tutti gli altri nodi ricevono un'etichettatura pari a 0. Una volta che l'inizializzazione è completa, l'algoritmo ripete l'operazione di push o relabel (da cui il nome dell'algoritmo) su i **nodi attivi**, cioè quelli con **eccesso di flusso** fintanto che nessuna operazione applicabile rimane.

Push L'operazione di push si applica su un lato uscente ammissibile (i, j) di un nodo attivo i. Va a spostare $\min\{e_x i, u_{ij}\}$ unità di flusso tra i a j.

Un'operazione di push che porta al massimo la capacità disponibile nell'arco viene detta **push di saturazione**, altrimenti tutto l'eccesso rimanente al nodo è mosso ulteriormente sfruttando i lati residuali: quest'ultima è detta **non-saturating push**.

Relabel L'operazione di relabel viene applicata su un nodo attivo i privo di nodi uscenti ammissibili. Va a modificare l'etichettatura del nodo in modo tale che coincida con il minimo valore tale per cui un nodo uscente ammissibile viene creato: questo porta sempre il valore dell'etichettatura ad aumentare.

9.1 Quasi-Flusso

Definizione 9.1.1 (Eccesso di flusso). Quando un flusso non è ammissibile perché non sono soddisfatti i vincoli di bilanciamento ai nodi (cioè vi è più flusso in entrata che in uscita) per almeno un nodo vi è un **eccesso di flusso**:

$$e_x(i) = \text{“eccesso”, l'eccesso di flusso nel nodo } i$$

Definizione 9.1.2 (Quasi-Flusso). Un vettore $\underline{\boldsymbol{x}} \in \mathbb{Z}_+$ tale che:

1. $e_x(n) \geqslant 0$ per ogni nodo $n \in \mathbb{N} \setminus \{s, t\}$
2. $0 \leqslant x_{ij} \leqslant u_{ij}$ per ogni arco $(i, j) \in A$

si dice **quasi-flusso.**

Definizione 9.1.3 (Rete residuale associata a un quasi-flusso). Una rete residuale $G(\underline{\boldsymbol{x}})$ associata ad un quasi flusso $\underline{\boldsymbol{x}}$ ha le seguenti caratteristiche:

1. Esiste un arco $(i,j) \in G(\underline{\boldsymbol{x}}) \Leftrightarrow x_{ji} > 0 \quad \vee \quad x_{ij} < u_{ij}$.
2. L'etichettatura dell'arco $(i,j) \in G(\underline{\boldsymbol{x}})$ è pari a $u'_{ij} = u_{ij} - x_{ij} + x_{ji}$

Definizione 9.1.4 (Nodo attivo). Un nodo i si dice attivo se

$$e_x(i) > 0$$

Definizione 9.1.5 (Arco ammissibile). Un arco $(i,j) \in G(\underline{\boldsymbol{x}}) : d(i) = d(j) + 1$ si dice **ammissibile**.

Definizione 9.1.6 (Etichettatura valida rispetto a quasi-flusso). Un vettore $\underline{\boldsymbol{d}} \in (\mathbb{Z}_+ \cup \{+\infty\})$ è una etichettatura valida rispetto ad un quasi flusso $\underline{\boldsymbol{x}}$ se:

Source condition $d(s) = n$

Sink conservation $d(t) = 0$

Etichettatura valida $\forall (i,j) \in G(\underline{\boldsymbol{x}}), d(i) \leqslant d(j) + 1$

9.2 Massimo flusso tramite quasi-flussi

Dato un grafo $G = (N, A)$ è sempre possibile individuare un quasi-flusso ammissibile e una etichettatura valida, ponendo:

1. $x_{si} = u_{si}$, per ogni arco (s,i) uscente da s
2. $x_{ij} = 0$ per tutti gli altri archi di A
3. $d(s) = n$, $d(i) = 0$ per tutti gli altri nodi di N

Teorema 9.2.1 (Massimo flusso tramite quasi-flussi). Se $\underline{\boldsymbol{x}}$ è un quasi-flusso ammissibile e $\underline{\boldsymbol{d}}$ è un'etichettatura valida per $\underline{\boldsymbol{x}}$, allora esiste un (s,t)-taglio $\delta(R)$ tale che $x_{ij} = u_{ij}$ per ogni $(i,j) \in \delta(R)$ e $x_{ij} = 0$ per ogni $(i,j) \in \delta(R)$.

Ne segue che se $\underline{\boldsymbol{x}}$ è un **flusso ammissibile** e **ammette un'etichettatura valida**, allora $\underline{\boldsymbol{x}}$ è **un massimo flusso**.