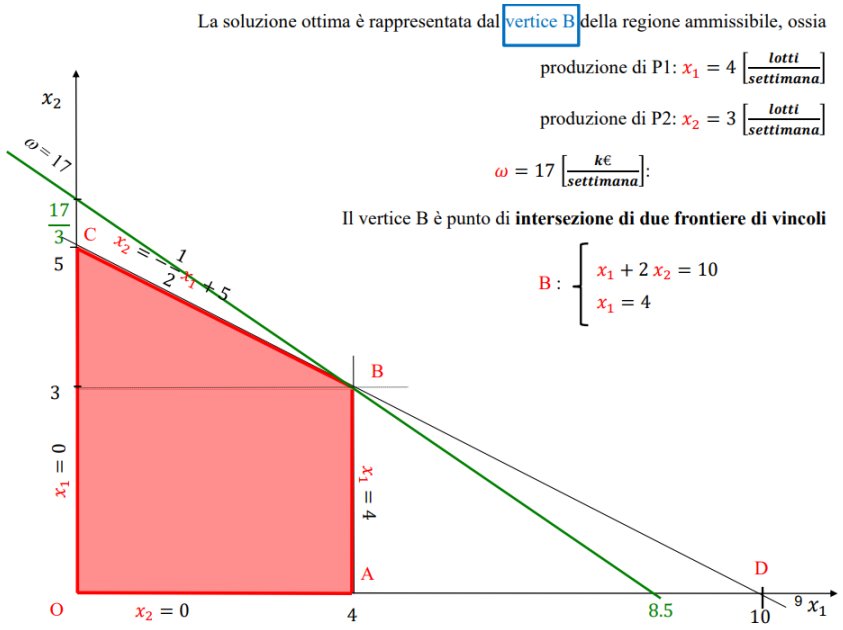
MAO: Modelli a algoritmi di ottimizzazione

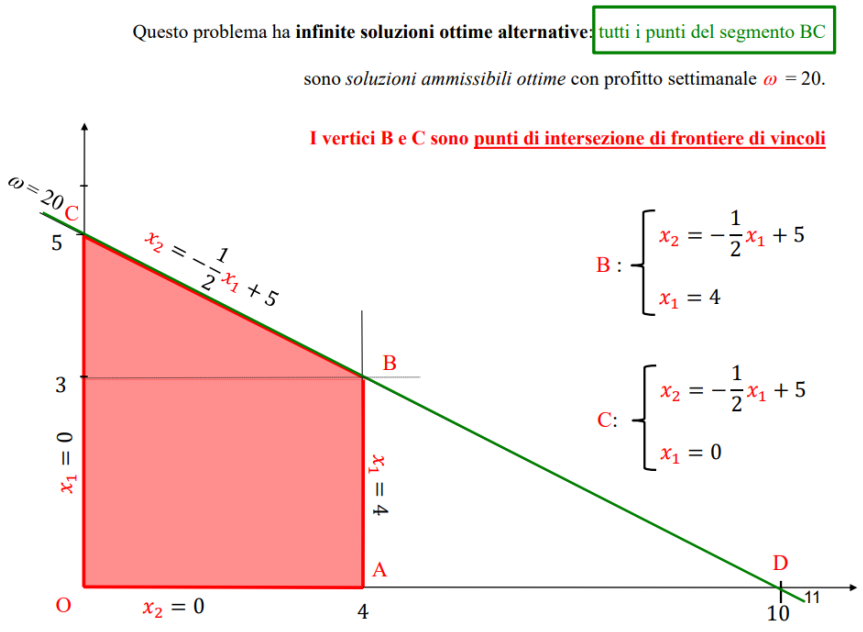
problemi di PL (programmazione lineare)

differenti casi di problemi di PL:

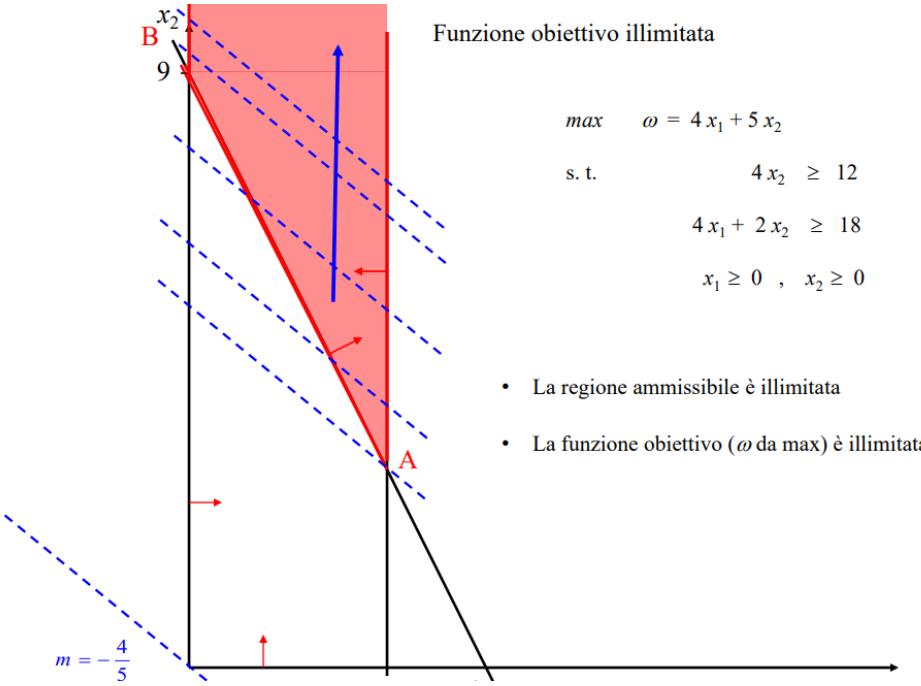
- soluzione ammissibile ottima unica
  - il vertice che è interseione di vincoli è la soluzione ottima



- soluzione ammissibili ottime alternative
  - son soluzione tutti i punti del segmento BC, soluzioni infinite e ottime.
  - questo perchè le due rette sono sovrapposte.



- soluzioni degeneri
  - nessun punto soddisfa i vincoli del modello, quindi la regione ammissibile è vuota.
  - soluzione non ammissibile
- problema illimitato
  - regione non limitata superiormente, problema infinito.
  - la funzione obiettivo ω è illimitata



Funzione obiettivo illimitata

$$max \quad \omega = 4x_1 + 5x_2$$

$$\text{s. t.} \quad 4x_2 \geq 12$$

$$4x_1 + 2x_2 \geq 18$$

$$x_1 \geq 0 \ , \quad x_2 \geq 0$$

- La regione ammissibile è illimitata
- La funzione obiettivo (ω da max) è illimitata

- problema con regione ammissibile vuota

$$\min \varphi = 0.4x_1 + 0.5x_2$$

$$0.3x_1 + 0.1x_2 \leq 2.7$$

$$0.6x_1 + 0.4x_2 \geq 6.0$$

$$0.5x_1 + 0.5x_2 = 6.0$$

$$x_1 \geq 0, \quad x_2 \geq 0$$

regione ammissibile non vuota

(non contiene $x_1 = x_2 = 0$)

$$\min \varphi = 0.4x_1 + 0.5x_2$$

$$0.3x_1 + 0.1x_2 \leq 2.7$$

$$0.6x_1 + 0.4x_2 \geq 6.0$$

$$0.5x_1 + 0.5x_2 = 5.0$$

$$x_1 \geq 0, \quad x_2 \geq 0$$

regione ammissibile vuota

derivazione dell algoritmo del simplesso

La derivazione dell'algoritmo del simplesso viene fatta in due passi:

- passo 1:
  - si considerano i problemi di PL nei quali i vincoli funzionali sono nella forma di disequazioni $\leq$ con termine noto non negativo per i quali è ammissibile la soluzione con tutte le variabili decisionali nulle
  - Per tali problemi la questione di ammissibilità è già risolta

Esempio di riferimento:

$$\max \omega = 2\,x_1 + 3\,x_2$$
$$x_1 + 2\,x_2 \leq 10$$
$$x_1 \leq 4$$
$$x_1 \geq 0,\ x_2 \geq 0$$

- passo 2:
  - problemi per i quali la soluzione con tutte le variabili decisionali nulle non è ammissibile,
  - si deve determinare se esistono soluzione ammissibili per il problema

Esempi di riferimento:

$$\min \varphi = 0.4\,x_1 + 0.5\,x_2$$
$$0.3\,x_1 + 0.1\,x_2 \leq 2.7$$
$$0.6\,x_1 + 0.4\,x_2 \geq 6.0$$
$$0.5\,x_1 + 0.5\,x_2 = 6.0$$
$$x_1 \geq 0,\quad x_2 \geq 0$$

regione ammissibile non vuota

(non contiene $x_1 = x_2 = 0$)

$$\min \varphi = 0.4\,x_1 + 0.5\,x_2$$
$$0.3\,x_1 + 0.1\,x_2 \leq 2.7$$
$$0.6\,x_1 + 0.4\,x_2 \geq 6.0$$
$$0.5\,x_1 + 0.5\,x_2 = 5.0$$
$$x_1 \geq 0,\quad x_2 \geq 0$$

regione ammissibile vuota

L'algoritmo del simplesso risolve il problema prendendo in considerazione solo soluzioni di base.

Teorema Fondamentale della Programmazione Lineare, parte I:

- Se per il problema di PL esiste una s.a., allora esiste anche una s.b.a.
- Se non esistono s.b. ammissibili (ossia tutte le s.b. sono non ammissibili), allora non esistono s.a. (ossia la regione ammissibile è vuota)

Teorema Fondamentale della Programmazione Lineare, parte II:

- Se esistono soluzioni ammissibili ottime del problema di PL, almeno una è soluzione di base.

Introduzione alla dualità nella programmazione lineare

1. Parto da una soluzione ammissibile (nessuna componente è negativa),da questa s.b.a. iniziamo la procedura di ottimizzazione
2. Porto in base xi, con valore di riga R0 positivo, la v.b. che esce di base è *si*
3. Divi per la colonna Wb per trovare la riga pivotale
4. Operazioni elementari di riga per calcolare la forma canonica associata alla base ***B***2
   a. R hat = 2/3 R
5. Cvontinuo a portare elementi in base finche soluzione non diventa ottima

Teorema della dualità debole

Per ogni coppia $\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{pmatrix}$ e $\begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix}$ dove

- $\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{s} \end{pmatrix}$ è *soluzione ammissibile* del problema in $\boldsymbol{x}$ (ossia $\boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$ e $\boldsymbol{s} \geq \boldsymbol{0}$)

- $\begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{y} \end{pmatrix}$ è *soluzione ammissibile* del problema in $\boldsymbol{y}$ (ossia $\boldsymbol{A}^T\boldsymbol{y} - \boldsymbol{z} = \boldsymbol{p}$, $\boldsymbol{y} \geq \boldsymbol{0}$ e $\boldsymbol{z} \geq \boldsymbol{0}$)

vale che

$$\omega(\boldsymbol{x}, \boldsymbol{s}) \leq \varphi_D(\boldsymbol{y}, \boldsymbol{z}).$$

Il teorema della dualità debole afferma che: in qualunque s.a. del problema in $\boldsymbol{y}$ il valore assunto da $\varphi_d = b^t y + \eta$ è maggiore o uguale del valore assunto da $\omega = p^t x + \eta$ in qualunque s.a. del problema in $\boldsymbol{x}$.

- il valore assunto da $\omega = p^t x + \eta$ in qualunque s.a. del problema in $\boldsymbol{x}$ è un minorante del valore assunto da $\varphi_d = b^t y + \eta$ in qualunque s.a. del problema in $\boldsymbol{y}$
- il valore assunto da $\varphi_d = b^t y + \eta$ in qualunque s.a. del problema in $\boldsymbol{y}$ è un maggiorante del valore assunto da $\omega = p^t x + \eta$ in qualunque s.a. del problema in x

Corollario 1 del Teorema di dualità debole

Se vale $\omega^* = \varphi_d{}^*$, allora

- ω^* è il massimo di $\omega = p^t x + \eta$ sulla regione ammissibile $\boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$ e $\boldsymbol{s} \geq \boldsymbol{0}$
- $\varphi_d{}^*$ è il minimo di $\varphi_d = b^t y + \eta$ sulla regione ammissibile $\boldsymbol{A}\;\boldsymbol{y} - \boldsymbol{z} = \boldsymbol{p}$, $\boldsymbol{y} \geq \boldsymbol{0}$ e $\boldsymbol{z} \geq 0$

Corollario 2 del Teorema di dualità debole

Nei problemi primale-duale, se uno dei due problemi è illimitato, l'altro ha regione ammissibile vuota

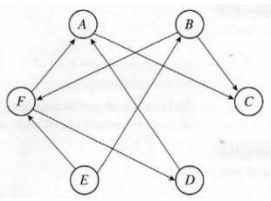
dualità

Primale	Duale
$\max \omega = p^T x + \eta$	$\min \varphi = b^T y + \eta$
s. a $\quad A\,x \leq b$	s. a $\quad A^T y \geq p$
$x \geq 0$	$y \geq 0$

1 PLI (Programmazione lineare intera)

Classe di problemi con variabili intere, funzione obietivo e vincoli lineari.
il metodo del simplesso non può aiutare in quanto trova solamente soluzioni nei vertici della soluzione ammissibile, i quali potrebbero non essere soluzioni intere.
Quindi utilizziamo il rilassamento continuo, ossia si lavora con il simplesso sul problema continuo,

1.1 Il problema della raggiungibilità:

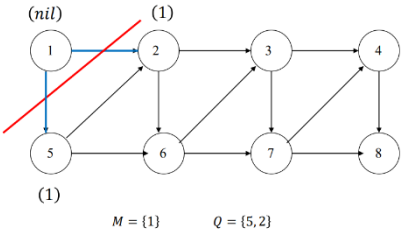


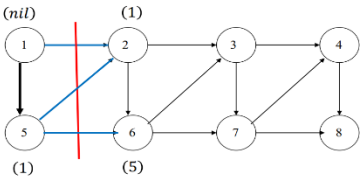
Dato un grafo orientato G= (N,A), il problema della raggiungibilità prevede che, avendo identificato un particolare **nodo sorgente s**∈N, si determini l'**insieme dei nodi raggiungibili da s** sfruittando gli archi (i,j)∈A

Il problema della raggiungibilità è affrontato e risolto dall'**algoritmo di visita.**
permette di: "determinare i nodi raggiungibili dalla radice per mezzo di cammini orientati."
e' un algoritmo iterativo che richiedere l'aggiornamento di due insiemi:

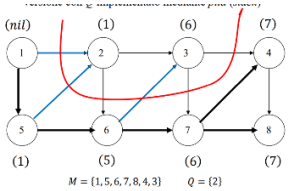
- **M** = insieme dei nodi **raggiunti**
- **Q** = insieme dei nodi **adiacenti** agli archi uscenti dai nodi di M.
- iterazione:
  - all'inizio di ogni iterazione si estrae il primo nodo dalla lista Q
  - questo nodo e' messo in M e in Q si aggiungono i nodi raggiungibili da questo nodo
  - continuo finche la lista Q non e' vuota.
  - Output: vettore di predecessori p

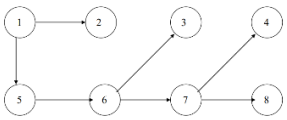
  - l'insieme **Q** e' implementato come pila/stack oppure come coda/queue
    - **pila/stack**: in questo caso i figli del nodo i vengono inseriti in Q in **cima** alla fila
      - strategia di esplorazione grafo **depth-first search (dfs)** o visita in profondità
      - tende a costruire cammini "lunghi"



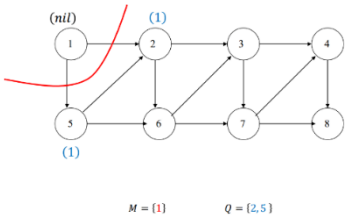


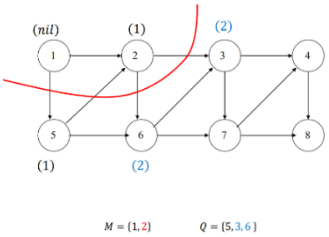
i	1	2	3	4	5	6	7	8
p_i	–	1	0	0	1	5	0	0

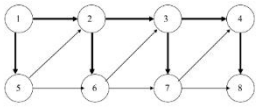




- **coda/queue**: i figli del nodo i-esimo sono inseriti in Q in **fondo** alla lista.
  - strategia di esplorazione del grafo:**breadth-first search (bfs)** o visita in larghezza
  - tende a costruire cammini "corti"







$M = \{1,2,5,3,6,4,7,8\}$ $Q = \emptyset$

i	1	2	3	4	5	6	7	8
p_i	–	1	2	3	1	2	3	4

Poiché tutte le componenti sono diverse da 0,
tutti i nodi sono raggiungibili dal nodo-radice $r = 1$

- da nodo 1 a nodo 2 : (1, 2)
- da nodo 1 a nodo 3 : (1, 2), (2, 3)
- da nodo 1 a nodo 4 : (1, 2), (2, 3), (3, 4)
- da nodo 1 a nodo 5 : (1, 5)
- da nodo 1 a nodo 6 : (1, 2), (2, 6)
- da nodo 1 a nodo 7 : (1, 2), (2, 3), (3, 7)
- da nodo 1 a nodo 4 : (1, 2), (2, 3), (3, 4), (4, 8)

in corrispondenza di ogni nodo s ideve specificare un **etichetta** numerica contenente il numero del nodo predecessore. Non e' un problema di ottimizzazione, bensi' lo si usa per rispondere a "esiste un cammino tra una certa coppia di nodi?"

2 Alberi di supporto di costo minimo

È un albero in cui **sommando i pesi degli archi si ottiene un valore minimo.**
un **albero di supporto** di un grafo non orientato G= (V,E) è un **sottografo** ad albero **di G contenente tutti** i **vertici** di **G**.
Il problema dell'**albero di supporto di costo minimo** si pone su un **grafo non orientato** nel quale **ogni arco** $e \in E$ ha un **costo** c_e.
obiettivo: è di **trovare un albero T di supporto** di G per cui **è minima la somma dei costi dei lati.**

$$\min c(T) = \sum_{e \in T} c_e$$

un grafo G = (N , A), con | N | = n, è detto **albero** se:

- è **connesso** (esiste un cammino tra ogni coppia di nodi)
- è **privo di cicli**

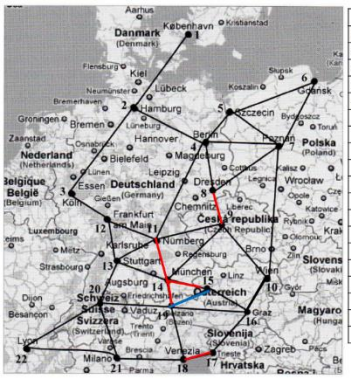
2.1 Metodo di Kruskal

è un algoritmo usato per la **costruzione dell albero di supporto di costo minimo**, si tratta di un algoritmo **greedy**, ossia un algoritmo iterativo che ad ogni passo effettua la **scelta migliore per l'immediato** piuttosto che adottare una strategia a lungo termine.

L'algoritmo di kruskal prevede **due fasi:**

1. **Inizializzazione**. si ha un **ordinamento** dei **lati** in ordine di **costo non decrescente**
2. **Iterazione**. Tra i collegamenti non ancora considerati bisogna:
   a. **scegliere** il **lato di costo minimo**
      i. se l'insirimento di questo lato di costo minimo **non** determina la formazione di un **ciclo**, allora questo lato è **inserito** nel albero di supporto.
      ii. se l'inserimetno del lato comporta la formazione di un **ciclo**, allora lo **scarto** ed elimino il lato dall insieme dei lati da considerare
   b. **ripeto le iterazioni** finche non vengono **inseriti** $n - 1$ **lati** all interno dell albero.

la soluzione viene **costruita** per **iterazioni successive** che **non** garantiscono **la proprietà di connessione delle soluzioni intermedie**, si ha questa condizione sono quando si raggiunge l'ultima iterazione.
Ho la **garanzia** della **connessione** solo all **ultimo passaggio**.



	lato	Km
1	{14, 15}	86
2	{11, 14}	94
3	{14, 19}	95
4	{8,9}	101
5	{17, 18}	102
	{15, 19}	111
	{3,12}	115
	{11,13}	120
	{10,16}	122
	{4,5}	131
	{13, 14}	131
	{12, 13}	133
	{11, 12}	146
	{4,8}	149
	{20, 13}	152
	{18, 21}	155
	{15, 16}	170
	{9,11}	179
	{18, 19}	192

lato	Km
{20,22}	192
{19, 20}	194
{10,15}	195
{9,10}	196
{1,2}	204
{2,4}	209
{21,22}	211
{20,21}	217
{5,7}	226
{9,14}	227
{16, 17}	229
{4,7}	234
{2,3}	256
{6,7}	267
{5,6}	309
{16, 19}	318
{4,11}	338
{7,8}	368
{7, 10}	475

93

	lato	Km
1	{14, 15}	86
2	{11, 14}	94
3	{14, 19}	95
4	{8,9}	101
5	{17, 18}	102
	{15, 19}	111
6	{3,12}	115
7	{11,13}	120
8	{10,16}	122
9	{4,5}	131
	{13, 14}	131
	{12, 13}	133
	{11, 12}	146
	{4,8}	149
	{20, 13}	152
	{18, 21}	155
	{15, 16}	170
	{9,11}	179
	{18, 19}	192

lato	Km
{20,22}	192
{19, 20}	194
{10,15}	195
{9,10}	196
{1,2}	204
{2,4}	209
{21,22}	211
{20,21}	217
{5,7}	226
{9,14}	227
{16, 17}	229
{4,7}	234
{2,3}	256
{6,7}	267
{5,6}	309
{16, 19}	318
{4,11}	338
{7,8}	368
{7, 10}	475

94

	lato	Km
1	{14, 15}	86
2	{11, 14}	94
3	{14, 19}	95
4	{8,9}	101
5	{17, 18}	102
	{15, 19}	111
6	{3,12}	115
7	{11,13}	120
8	{10,16}	122
9	{4,5}	131
	{13, 14}	131
10	{12, 13}	133
	{11, 12}	146
11	{4,8}	149
12	{20, 13}	152
13	{18, 21}	155
14	{15, 16}	170
15	{9,11}	179
16	{18, 19}	192

	lato	Km
17	{20,22}	192
	{19, 20}	194
	{10,15}	195
	{9,10}	196
18	{1,2}	204
19	{2,4}	209
	{21,22}	211
	{20,21}	217
20	{5,7}	226
	{9,14}	227
	{16, 17}	229
	{4,7}	234
	{2,3}	256
21	{6,7}	267
	{5,6}	309
	{16, 19}	318
	{4,11}	338
	{7,8}	368
	{7, 10}	475

99

Ho 22 nodi connessi con archi, voglio collegare tutti i nodi in modo da creare una sottorete che consente di avere **almeno un collegamento per ogni città**, e questo **collegamento** deve essere **il più veloce**.
Per esempio da berlino – monaco cerco un unico percorso i cui tratti sono più veloci.
Collego i primi 4 nodi e vedo che non danno origine ad un ciclo, fino al 5° che crea ciclo (img a sinistra), questo collegamento viene scartato e viene considerato il successivo, ossia {3,12}. Poi a {13,14} ho ancora ciclo, lo scarto e vado avanti, il successivo non ha un ciclo, vado avanti.
Continuo così finche non arrivo a **n-1 rami**, ossia 21 dato che ho 22 città.
Così ho tutte le città collegate, con 1 solo percorso per ogni coppia di nodi e pure di costo minimo.

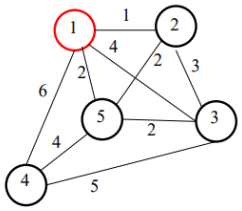
2.2 Algoritmo di Prim

algoritmo alternativo a Kruskal.
Pim garantisce che la **struttura** ad **ogni passo sia connessa**, mentre Kruskal garantisce all ultima iterazione.

Si compone di 2 fasi:

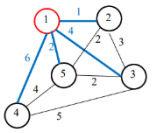
1. **Inizializzazione**. Coincide con la prima iterazione dell algoritmo:
   a. scelgo albitrariamente un **nodo di partenza**
   b. inserisco il nodo scelto nell insieme '**S**' (insieme dei **nodi connessi**)
   c. Considero la **stella** del nodo scelto (l'insieme di tutti i **lati incidenti sul nodo**)
      i. tra tutti i **lati** della **stella** scelgo quello di **costo minimo**
      ii. **inserisco** nell **insieme** '**S**' il Nodo coincidente con l'arco di costo minimo
   d. **inserisco** in **T*** (**insieme dei colelgamenti scelti**) il **lato di costo minimo** scelto.

Inizializzazione: $T^* = \emptyset$ e $S = \{1\}$



2. **Iterazione**. ad ogni iterazione bisogna:
   a. definire l'**insieme dei collegamenti del taglio $\delta(S)$**, ossia l'insieme contenente tutti i lati che hanno come nodi di incidenza un nodo di S e un nodo non appartenente ad S.
   b. considero il **lato di $\delta(S)$** che ha **costo minore**
   c. **aggiungo** il **nodo** ad '**S**' e il **lato** in '**T***'
   d. ripeto le iterazioni fino a quanto T* non contiene **n-1 elementi.**

Iterazione 1



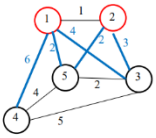
Essendo $S = \{1\}$, i collegamenti del taglio $\delta(S)$ sono $\{\{1,2\}, \{1,3\}, \{1,4\}, \{1,5\}\}$:

il lato di costo minimo è $\{1,2\}$ che ha costo 1.

Si pone $T^* = \{\{1,2\}\}$ e $S = \{1,2\}$.

Essendo $|T^*| = 1 < n - 1 = 4$, la procedura continua.

Iterazione 2



Essendo $S = \{1,2\}$, i collegamenti del taglio $\delta(S)$ sono

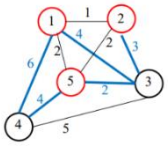
$\{\{1,3\}, \{1,4\}, \{1,5\}, \{2,3\}, \{2,5\}\}$:

il lato di costo minimo è $\{1,5\}$ che ha costo 2.

Si pone $T^* = \{\{1,2\}, \{1,5\}\}$ e $S = \{1,2,5\}$.

Essendo $|T^*| = 2 < n - 1 = 4$, la procedura continua.

Iterazione 3



Essendo $S = \{1,2,5\}$, i collegamenti del taglio $\delta(S)$ sono

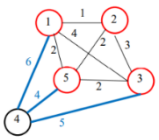
$\{\{1,3\}, \{1,4\}, \{2,3\}, \{5,3\}, \{5,4\}\}$:

il lato di costo minimo è $\{5,3\}$ che ha costo 2.

Si pone $T^* = \{\{1,2\}, \{1,5\}, \{5,3\}\}$ e $S = \{1,2,3,5\}$.

Essendo $|T^*| = 3 < n - 1 = 4$, la procedura continua.

Iterazione 4



Essendo $S = \{1,2,3,5\}$, i collegamenti del taglio $\delta(S)$ sono $\{\{1,4\},\{5,4\}, \{3,4\}\}$:

il lato di costo minimo è $\{5,4\}$ che ha costo 4.

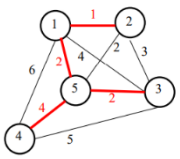
Si pone $T^* = \{\{1,2\}, \{1,5\}, \{5,3\}, \{5,4\}\}$ e $S = \{1,2,3,4,5\}$.

Essendo $|T^*| = 4$, la procedura termina.

Applicazione: si sceglie un nodo di partenza (1 in questo caso), considero la stella del nodo 1, ossia i collegamenti del grafo che incidono su quel nodo {(1,2), (1,3), (1,4), (1,5)}. Ogni arco ha un costo, si prende il costo minimo tra nodo {1,4,2,6}, ossia 1, quindi il collegamento (1,2) con costo 1, lo metto in T* quindi l'insieme dei collegamenti dell albero ottimo, in **s** metto i valori {1,2} perché ho raggiunto anche il nodo 2., continuo l'interazione finche **|T*| < n-1 .** specifico poi in **$\delta(S)$** i vari percorsi che terminano con un nodo che non fa parte di **s**, quindi $\delta(S)$ = {(1,3), (1,4), (1,5), (2,3), (2,5)}, scelgo quello con il **costo minimo** e ripeto l'iterazione.

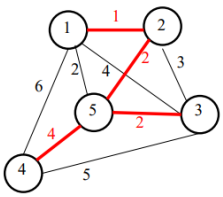
soluzioni ottime alternative: possono esistere più alberi di supporto di costo minimo.

Soluzioni ottime alternative



$T^* = \{\{1,2\}, \{\mathbf{1,5}\}, \{5,3\}, \{5,4\}\}$

$c(T^*) = 9$



$T^* = \{\{1,2\}, \{\mathbf{2,5}\}, \{5,3\}, \{5,4\}\}$

$c(T^*) = 9$

3 Cammini di costo minimo

Siano dati:

- Un grafo orientato G = (N,A) con **pesi degli archi** $c_{i,j}$ dove $(i,j) \in A$
  - $c_{i,j}$ **costo** di **attraversamento** dell'**arco**.
- Un nodo **sorgente 's'** ed un nodo **destinazione 't'.**

Il problema dei cammini di costo minimo consiste nel **trovare il cammino di costo minimo dal nodo sorgente 's' al nodo destinazione 't'**

Ci sono **3 algoritmi**:

- grafi con **presenza di cicli e** $c_{i,j} \geq 0$, $\forall\, i,j \in A$ → **dijkstra**
- grafi **aciclici** (numerazione topologica dei nodi) → **etichette**
- grafi con presenza di **cicli e con** $c_{i,j} < 0$ per qualche $i,j \in A$ → **floyd-warshall**

3.1 Algoritmo di dijkstra

L'algoritmo di dijsktra può essere applicato se e solo se $c_{i,j} \geq 0, \forall (i,j)$ ossia se il costo dell'arco dal nodo 'i' a 'j' è positivo.
Dijsktra si basa sull'assegnazione ad ogni nodo di **etichette** del tipo ***[Nodo predeces, Costo del cammino]***
Al noso sorgente 's' viene assegnata l'etichetta fittizia [s,0]

Ad ogni iterazione dell'algoritmo bisogna considerare il grafo come se fosse diviso in due insiemi:

- S = insieme dei **nodi raggiunti** dal cammino di costo minimo
- N\S = insieme dei **nodi non ancora raggiunti** dal cammino di costo minimo.

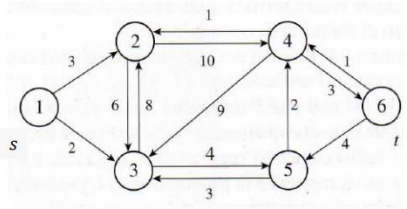
La compresenza degli insiemi S ed N\S determina l'esistenza di un **taglio**. Esistendo un taglio bisgona determinare gli **archi del taglio**, ovvero quegli archi **che collegano qualunque nodo di S con qualunque nodo di N\S.** Gli **archi** vengono **inclusi nell insieme di δ.**
Per **ogni arco di δ si calcola la somma tra:**

- **Costo** dell'**arco del taglio**
- **Valore** di **costo** dell'**etichetta** del nodo di S coinvolto nell'arco del taglio.

Avendo calcolato tale somma per ogni arco di **δ si sceglie l'arco che presenta il costo minore in δ** . Il nodo di N\S coinvolto nell'arco scelto viene eliminato da N\S e **aggiunto ad S.**
Le iterazioni si **concludono** nel momento in cui viene **aggiungo ad S anche il nodo destinazione t.**
La strategia è di tipo **greedy**.

Determinare il cammino minimo dal nodo $s = 1$ al nodo $t = 6$

Tutti gli archi hanno costo nonnegativo: $c_{i,j} \geq 0 \quad \forall\ (i,j) \in A$



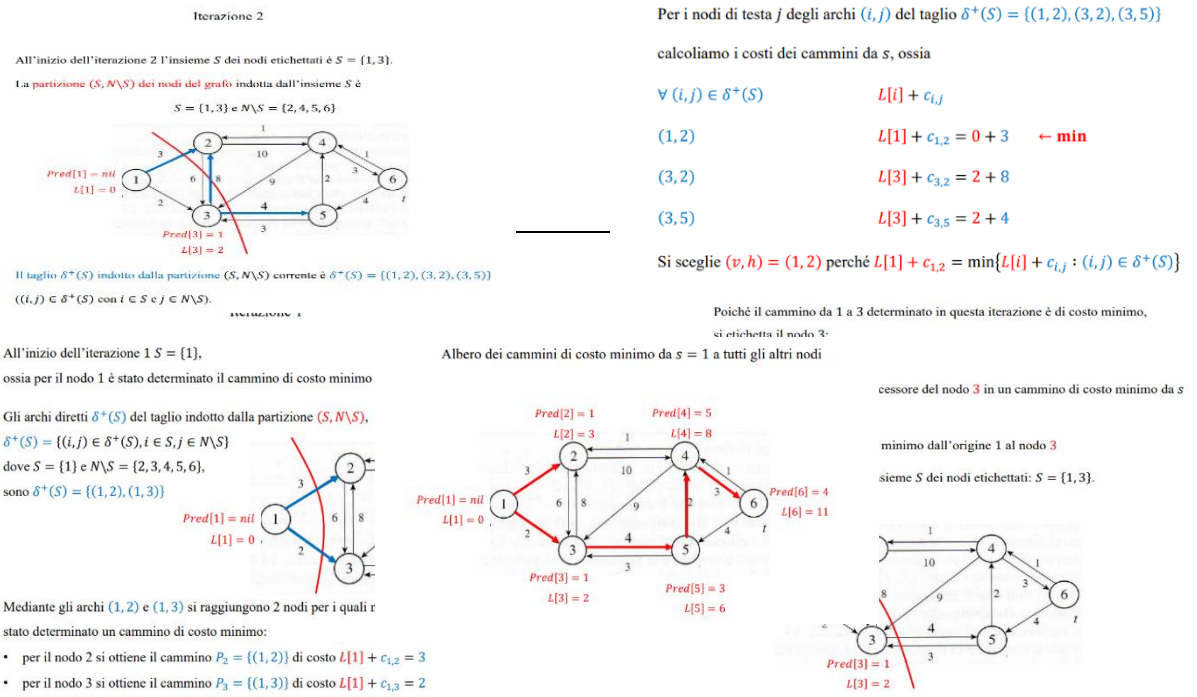
Inizializzazione

- $S = \{1\}$
- Etichette dei nodi $S = \{1\}$
  $Pred[1] = nil$ (il nodo 1 non ha predecessori)
  $L[1] = 0$ (costo del cammino di costo minimo dal nodo $s = 1$ al nodo 1)

Nota: S è l'insieme dei nodi per i quali è stato determinato il cammino di costo minimo a partire da s

---

Ho un taglio su 1, è l'unico nodo raggiunto s={1}, tutti gli altri fanno parte dei nodi non ancora raggiunti.
Così individuo gli archi del taglio, quelli che collegano un nodo di 's' ad un nodo non di 's'.
Per ogni nodo dell arco del taglio calcolo il costo per raggiungerlo, quindi calcolo i costi (1,2) e (1,3), scelgo il nodo con il percorso di costo minore, poi aggiungo l'etichetta [pred, costo].



Così ottengo 5 archi che collegano 6 nodi, quindi i soliti n-1 archi. costruisco i cammini minimi su tutti nodi

3.2 Algoritmo delle etichette

L'algoritmo delle etichette può essere applicato solo a grafi orientati aciclici solo dopo aver ridenominato i nodi secondo l'**ordine topologico**, che prevede che partendo da un **grafo aciclico**, si ottenga un nuovo grafo con la stessa struttura di quello originario i cui archi $(i, j) \in A$ sono tali da connettere un **nodo di coda *i* ad un nodo di testa *j***, con $i < j$, il nodo di testa j è numerato con numero maggiore rispetto a quello presente nel nodo di coda i.
Per effettuare l'ordinamento topologico si sfrutta l'**algoritmo di ordinamento topologico**:

1. Considero il nodo **sorgente s**, nodo che ha solo archi uscenti, e gli assegno il numero '1'.
2. **Ignoro gli archi uscenti** dal nodo appena numerato, così "cancello" le entrate per i nodi successivi
3. Considero solo i **nodi che hanno solo archi uscenti** e assegno numeri crescenti
4. Continuo a numerare finche non arrivo al nodo di **destinazione t.**

L'obiettivo dell'algoritmo delle etichette: **determinare il cammino di costo minimo dal nodo sorgente s al nodo di destinazione t** in un grafo con **nodi numerati** secondo l'**ordinamento topologico**.
Un **etichetta** è del tipo **[pred, costo].**

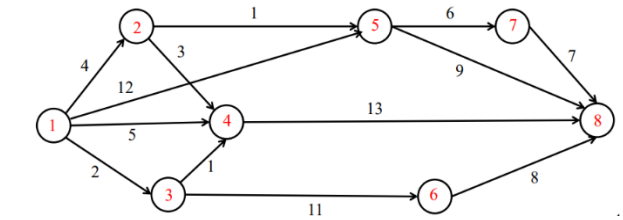
Passi algoritmo:

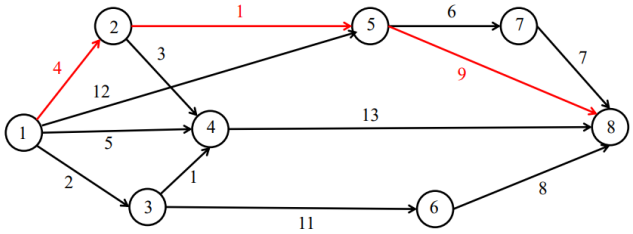
1. Parto dal **nodo sorgente s** e gli assegno l'etichetta fittizia [s,0]
2. Considero uno ad uno i nodi del grafo **procedendo** in base all'**ordinamento topologico**. In particolare, considero la BS(i) (Backward star, stella entrante nel nodo i-esimo)

3. Per ogni nodo ccoinvolto negli archi di BS(i) calcolo il **costo da spendere per raggiungere il nodo i-esimo sommando:**
    a. **Costo** dell **arco** che **connette i nodi** coinvolti negli archi di BS(i) al nodo i-esimo
    b. **Valore costo** dell **etichetta** del nodo connesso al nodo i-esimo
4. Per ogni nodo coinvolto negli archi di BS(i) considero solo quello che viene connesso al nodo i-esimo con **l'arco di costo minore**
5. Aggiungo al cammino minimo l'arco scelto
6. Le **iterazioni proseguono** fino a quando non viene **raggiunto il nodo di destinazione t**.

$N = \{1,2,3,4,5,6,7,8\}$

$A = \{(1,2),(1,3),(1,4),(2,4),(3,4),(1,5),(2,5),(3,6),(5\,7),(4,8),(5,8),(6,8),(7,8)\}$





$A = \{(1,2),(1,3),(1,4),(2,4),(3,4),(1,5),(2,5),(3,6),(5\,7),(4,8),(5,8),(6,8),(7,8)\}$

Inizializzazione			$Pred[1]=1$	$L[1]=0$
$t=2$	$(1,2)$	$L[1]+c_{1,2}=0+4=4$	$Pred[2]=1$	$L[2]=4$
$t=3$	$(1,3)$	$L[1]+c_{1,3}=0+2=2$	$Pred[3]=1$	$L[3]=2$
$t=4$	$(1,4)$	$L[1]+c_{1,4}=0+5=5$		
	$(2,4)$	$L[2]+c_{2,4}=4+3=7$		
	$(3,4)$	$L[3]+c_{3,4}=2+1=3$	$Pred[4]=3$	$L[4]=3$
$t=5$	$(1,5)$	$L[1]+c_{1,5}=0+12=12$		
	$(2,5)$	$L[2]+c_{2,5}=4+1=5$	$Pred[5]=2$	$L[5]=5$
$t=6$	$(3,6)$	$L[3]+c_{3,6}=2+11=13$	$Pred[6]=3$	$L[6]=13$
$t=7$	$(5\,7)$	$L[5]+c_{5,7}=5+6=11$	$Pred[7]=5$	$L[7]=11$
$t=8$	$(4,8)$	$L[4]+c_{4,8}=3+13=16$		
	$(5,8)$	$L[5]+c_{5,8}=5+9=14$	$Pred[8]=5$	$L[8]=14$
	$(6,8)$	$L[6]+c_{6,8}=13+8=21$		
	$(7,8)$	$L[7]+c_{7,8}=11+7=18$		

Analizzo i nodi in ordine numerico, dal 1 al 8, considero per ogni nodo tutti i modi per arrivare a quel nodo, dato che non ho cicli posso arrivarci solo dai precedenti e prendo il costo minimo.

3.3 Algoritmo di floyd-warshall

L'algoritmo di floyd-warshall calcola i cammini di costo minimo tra ogni coppia di nodi in un grafo qualunque. Perciò è applicabile anche a **grafi con archi di costo negativo** (caso in cui non è appliabile l'algoritmo di dijkstra) o a **grafi che contengono cicli** (caso in cui non è applicabile etichette).
Ha un **alta complessità computazionale** e viene applicato solo in quei casi in cui non posso applicare gli altri due algoritmi.

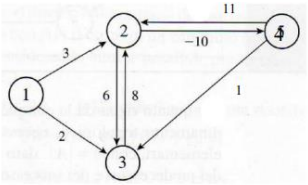
Considero solo grafici che hanno cicli e archi di costo negativo, in questo caso il problema della determinazione del cammino di costo minimo potrebbe non essere ben definita perché se il grafo ha un **ciclo di costo negativo** si arebbe un **problema inferiormente illimitato**, cioè in ottima di **minimizzazione dei costi** sarebbe convieniente **percorrerlo infinite volte** piuttosto che uscirne.

Input dell algoritmo

- Un **grafo orientato** G = (N,A)
- Una **matrice dei costi di raggiungimento** $C \in M(n)$ contenente i **costi di percorrenza degli archi**
    - Ha **0 sulle diagonali**, perché è il costo da (1,1), quindi nullo
    - L'elemento in posizione $c_{1,2}$ ossia '3' indica il costo da nodo 1 a nodo 2.
    - Se **l'arco non esiste il valore è** ∞**,** come nel caso di $c_{1,4}$, $c_{2,1}$, ..

Esempio:

$$C = \begin{bmatrix} 0 & 3 & 2 & \infty \\ \infty & 0 & 6 & -10 \\ \infty & 8 & 0 & \infty \\ \infty & 11 & 1 & 0 \end{bmatrix}$$



dove si è posto

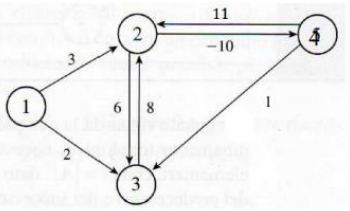
- $c_{i,i} = 0$ per ogni $i \in N$
- $c_{i,j} = \infty$ per ogni $(i,j) \notin A$

Output algoritmo:

- Matrice $\boldsymbol{D} \in M(n) \rightarrow$ il **costo di un cammino minimo da i a j.**
  - Inizialmente uguale alla matrice C
- Matrice $\boldsymbol{P} \in M(n) \rightarrow$ il **predecessore** di **j** nel **cammino minimo da i a j.**
  - Indica il predecessore del cammino minimo. $c_{1,1}$ è 1, ossia '1' è il precedessore da (1,1), mentre $c_{1,2}$ sarà il predecessore da (1,2) ossia ancora '1', e così via.

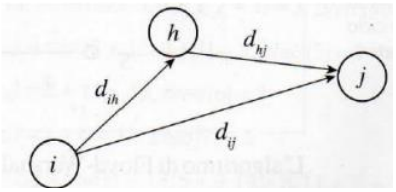
Fase di inizializzazione:

$$D = C = \begin{bmatrix} 0 & 3 & 2 & \infty \\ \infty & 0 & 6 & -10 \\ \infty & 8 & 0 & \infty \\ \infty & 11 & 1 & 0 \end{bmatrix}$$



$$P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

All'**iterazione h-esima** viene esegutia la seguente **operazione triangolare** relativa al nodo h: per ogni coppia di nodi i e j si controlla se l'unione del migliore cammino trovato sino a quel momento dal nodo i al nodo h con quello dal nodo h al nodo j fornisce un cammino:



- di **costo inferiore** a quello attuale, ossia se si verifica che $\boldsymbol{d_{i,h} + d_{h,j} < d_{i,j}}$.
  - ogni **terna di nodi controllo** se il costo attuale è minore dell'altro
  - se questa **condizione** viene **verificata** si pone:
    - $\boldsymbol{d_{i,j} = d_{i,h} + d_{h,j}}$ è il **nuovo cammino di costo inferiore**
    - $\boldsymbol{p_{i,j} = p_{h,j}}$ il **predecessore di *j* ora è *h* e non più *i*.**
  - Faccio questo controllo **per ogni elemento della matrice**
- se **non si trova alcun elemento** che soddisfa tale condizione:
  - si lasciano **inalterate le matrici D e P**
  - si **incrementa h** (controllo gli elementi successivi).

Dopo l'esecuzione di tutte le iterazioni, se **sulla diagonale della matrice D** si **riscontrano elementi negativi**, l'algoritmo **terminerebbe immediatamente**, dato che avremmo individuato un **ciclo di costo negativo**, e quindi il problema risulterebbe **inferiormente illimitato**.

4 Massimo flusso

Il problema del massimo flusso è definito per una **rete di flusso R**= (G, $u_{i,j}$) costituita da:

- un **grafo orientato** $G = (N, A)$
- **capacità** $u_{i,j}$ associate agli **archi** (i,j) $\in A$
- un **nodo** $s \in N$ (detto **sorgente**)
- un **nodo** $t \in N$ (detto **pozzo**)

il **problema del flusso massimo** consiste nel **trovare la massima quantità di flusso che può essere inviata da *s* a *t*** in modo che
• in ogni arco $(i,j) \in A$ il flusso $x_{i,j}$ **non superi la capacità** $u_{i,j}$ dell'arco
• in ogni **nodo intermedio** $i \in N\backslash\{s,t\}$ il **flusso totale entrante** sia **uguale** al **flusso totale uscente** (conservazione del flusso nei nodi intermedi)

4.1 Formulazione PL del problema di Massimo Flusso

Per il problema di massimo flusso è possibile scriere un **modello matematico** in cui:

- i singoli flussi $\boldsymbol{x_{i,j}}$ sull arco (i,j) $\in A$ rappresentano le **variabili decisionali**
- la quantità **v** è il **valore del flusso**, cioè la **quantità immessa nel nodo s** e **prelevata al nodo t.**

equazioni di conservazione del flusso:

- **nodo sorgente:**

$$\begin{gathered} quantità\ immessa\ al\ nodo (da\ massimizzare) \\ +\ flusso\ totale\ negli\ archi\ entranti\ che\ giunge\ al\ nodo\ s \\ =\ flusso\ totale\ negli\ archi\ uscenti \end{gathered}$$

$$v + \sum_{(j,s)\in BS(s)} x_{j,s} = \sum_{(s,j)\in FS(s)} x_{s,j}$$

- **nodo pozzo:**

$$\begin{gathered} flusso\ totale\ negli\ archi\ entranti\ nel\ nodo\ t \\ = quantità\ prelevata\ al\ nodo (da\ massimizzare) \\ + flusso\ totale\ negli\ archi\ uscenti\ da\ t \end{gathered}$$

$$\sum_{(j,t)\in BS(t)} x_{j,t} = v + \sum_{(t,j)\in FS(t)} x_{t,j}$$

- **nodi di trasferimento:**

$$flusso\ totale\ sugli\ archi\ entranti = flusso\ ttoale\ sugli\ archi\ uscenti$$

$$\sum_{(j,i)\in BS(i)} x_{j,i} = \sum_{(i,j)\in FS(i)} x_{i,j}$$

- **vincoli di conservazione del flusso**: associati ai nodi del grafo diversi da s e t : stabiliscono che in ciascun nodo intermedio il flusso entrante deve essere uguale al flusso uscente
- **Vincoli di nonnegatività** del flusso sull'arco e vincoli di capacità massima : $0 \le x_{i,j} \le u_{i,j}$
- **Funzione obiettivo** : $\omega = v$

- ω : variabile obiettivo da max

Riassumendo, definendo le variabili decisionali

$$x_{i,j}: \text{ flusso sull'arco } (i,j) \in A$$

la formulazione PL del problema è la seguente:

$$\max \quad \omega = v$$

$$s.a \quad \sum_{(j,s)\in BS(s)} x_{j,s} - \sum_{(s,j)\in FS(s)} x_{s,j} = -v$$

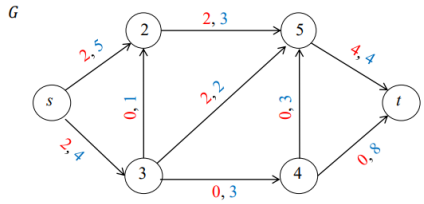
$$\sum_{(j,i)\in BS(i)} x_{j,i} - \sum_{(i,j)\in FS(i)} x_{i,j} = 0 \qquad i \in N\backslash\{s,t\}$$

$$\sum_{(j,t)\in BS(t)} x_{j,t} - \sum_{(t,j)\in FS(t)} x_{t,j} = v$$

$$0 \le x_{i,j} \le u_{i,j} \qquad (i,j) \in A$$

- Il problema consiste nel **determinare**
  - i **flussi $x_{i,j}$ sugli archi** $(i,j) \in A$
  - il **modulo del peso v** dei nodi di immissione e di prelievo (dobbiamo **determinare la massima comunicazione con l'esterno** consentita dalla struttura della rete)

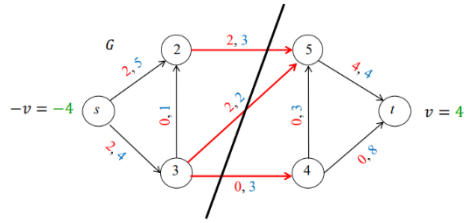
Esempio di rete di flusso $R = (G, u_{i,j})$ con flusso ammissibile $x_{i,j}$



I flussi sugli archi soddisfano

- bilanci nodali
- vincoli di nonnegatività e di capacità

Taglio $(S, N\backslash S)$ con $S = \{s, 2, 3\}$ e $N\backslash S = \{4, 5, t\}$



	$x_{s,2}$	$x_{s,3}$	$x_{2,5}$	$x_{3,2}$	$x_{3,4}$	$x_{3,5}$	$x_{4,5}$	$x_{4,t}$	$x_{5,t}$	
s	−1	−1	0	0	0	0	0	0	0	$-v$
2	1	0	−1	1	0	0	0	0	0	0
3	0	1	0	−1	−1	−1	0	0	0	0
4	0	0	0	0	1	0	−1	−1	0	0
5	0	0	1	0	0	1	1	0	−1	0
t	0	0	0	0	0	0	0	1	1	v

Flusso del taglio: $x_{2,5} + x_{3,4} + x_{3,5} = 2 + 0 + 2 = 4$

4.2 Algoritmo di Ford-Fulkerson

L'algoritmo risolve il **problema del massimo flusso** in modo **incrementale**: dato un certo flusso iniziale, l'algoritmo **verifica** se **tale flusso è migliorabile**, ossia se è possibile inviare **ulteriore flusso dalla sorgente s al pozzo t.**
Per stabilire se tale miglioramento è possibile si utilizza il **grafo residuale (o rete incrementale)** $\bar{G}(x) = (N, \bar{A})$, ovvero una struttura che rappresenta tutte le possibili **modifiche** che si possono apportare ai **flussi** correnti $x_{i,j}$ sugli archi $(i,j) \in A$ della rete originaria.
Le modifiche ammissibili sono quelle che non violano i **vincoli** di **nonnegatività** del flusso e di **capacità** dell **arco** $0 \le x_{i,j} \le u_{i,j}$.

Sul grafo residuale $\overline{G}$ si determina, se esiste, un qualunque **cammino dalla sorgente s al pozzo t lungo il quale inviare ulteriore flusso**, tale **cammino** è detto **aumentante**.

- se tale **cammino <u>non esiste</u>**, il **flusso** ammissibile **corrente** è **ottimo**
- se tale **cammino <u>esiste</u>**
  - si determina **l'incremento di flusso** in modo che i nuovi flussi sugli archi soddisfino i vincoli di bilancio ai nodi
  - **l'incremento di flusso** è **sommato al flusso iniziale** ottenendo il **flusso ammissibile** la cui ottimalità o meno sarà determinata nella successiva iterazione.

Nella prima iterazione il flusso ammissibile iniziale può essere il flusso nullo.

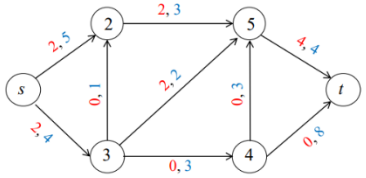
La rete incrementale tiene conto delle possibilità di

- **aumentare** il **flusso** sugli **archi non saturi**: tale possibilità è rappresentata con **archi diretti** con capacità residua non nulla
- **diminuire** il **flusso** sugli **archi non scarichi**: tale possibilità è rappresentata con **archi inversi** con capacità residua non nulla.

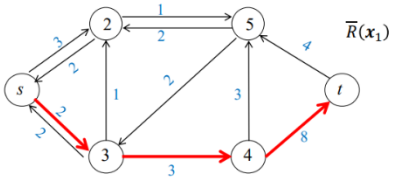
Il massimo aumento di flusso su un cammino aumentante è pari **al minimo delle capacità residue archi** che lo compongono. incremento di flusso è **sommato al flusso iniziale** ottenendo un **flusso ammissibile.**

L'**ottimalità** della soluzione ammissibile viene testata al passaggio successivo, nel momento in cui si cerca un nuovo cammino aumentante.
Se **non esiste più alcun cammino aumentante**, allora il flusso ammissibile corrente è **ottimo**.

Consideriamo la rete di flusso $R = (N, A, u_{i,j})$ con flusso ammissibile x_1

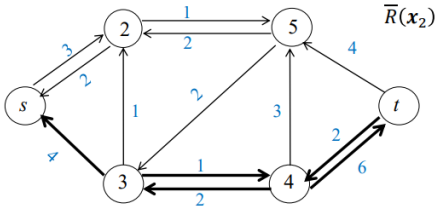


La rete incrementale $\overline{R}(x_1) = (N, \bar{A}, \bar{u}_{i,j})$ associata al flusso iniziale x_1 è

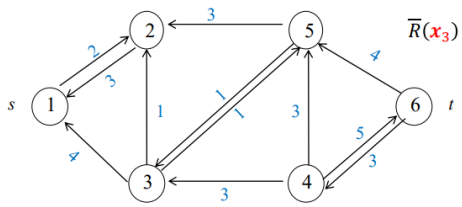


Su $\overline{R}(x_1)$ è possibile inviare flusso da s a t, per esempio lungo il cammino (aumentante) $s, 3, 4, t$ di $\theta = \min\{2, 3, 8\} = 2$.

La rete incrementale $\overline{R}(x_2)$, associata al nuovo flusso x_2, differisce dalla rete incrementale $\overline{R}(x_1)$ solo per gli archi dell'ultimo cammino aumentante trovato:



Dopo un paio di passaggi ottengo una situazione in cui non ho più cammini aumentanti da s a t.



Su tale grafo non è possibile trovare un cammino aumentante da s a t, come si può verificare applicando l'algoritmo di visita al grafo $\overline{G}(x_3)$: infatti, gli unici nodi raggiungibili dalla sorgente sono $\{s, 2\}$.

Risulta così determinato il taglio s-t con $N_s = \{s, 2\}$ ed $N_t = N \backslash N_s$:

- gli <u>archi diretti del taglio</u> (ossia quelli con la coda in N_s e la testa in N_t) sono <u>saturi</u> perché, se vi fosse un arco (i, j) non saturo con $i \in N_s$ e $j \in N_t$, l'arco (i, j) comparirebbe in $\overline{R}(x_3)$ e quindi j sarebbe raggiungibile da s (sarebbe perciò in N_s anziché in N_t)
- gli <u>archi inversi del taglio</u> (ossia quelli con la coda in N_t e la testa in N_s) sono <u>scarichi</u> perché, se vi fosse un arco (i, j) non scarico con $i \in N_t$ e $j \in N_s$, l'arco (i, j) comparirebbe in $\overline{R}(x_3)$ e quindi i sarebbe raggiungibile da s.

Quindi il taglio s-t trovato ha la proprietà di avere flusso che lo attraversa uguale al valore della sua capacità, essendo

$$X(N_s, N_t) = \sum_{(i,j)\in\delta^+(N_s)} x_{i,j} - \sum_{(i,j)\in\delta^-(N_s)} x_{i,j} = \sum_{(i,j)\in\delta^+(N_s)} u_{i,j} - 0 = U(N_s, N_t)$$

Risulta così dimostrato il **Teorema del Massimo flusso-minimo taglio** (**max flow – min cut**).
Data **rete di flusso G** = (N, A) il **valore del flusso massimo coincide** con la **capacità minima dei tagli s-t.**

5 Flusso di Costo Minimo (min cost flow)

Il problema del flusso di costo minimo è definito per una **rete di flusso** costituita da:

- Un grafo orientato G = (N, A)
- per ogni arco (i, j) ∈ A dotati di **capacità** $u_{i,j}$ ed un **costo unitario** $c_{i,j}$ **per ogni unità di flusso** che vi transita
- ad ogni nodo i ∈ N è associato un valore b_i detto **bilancio** del nodo, che rappresenta la **differenza** tra il **flusso totale entrante** nel nodo e il **flusso totale uscente**:
  - $b_i < 0$: il nodo i è **origine** o **sorgente** o di **input**, perché **mette a disposizione del flusso**
  - $b_i > 0$: il nodo è **destinazione** o **pozzo** o di **output**, perché **richiede flusso**
  - $b_i = 0$: il nodo è di **transito**, perché **né richiede né offre flusso**

Il problema del flusso di costo minimo consiste nel determinare la **distribuzione del flusso sulla rete** in modo che siano **soddisfatti le seguenti condizioni**

- ogni **origine** **invia** tutta la quantità disponibile di flusso.
- ogni **destinazione** **riceve** tutta la quantità richiesta di flusso.
- le **capacità** degli archi **non devono essere superate**.
- il **costo complessivo** di utilizzo della rete deve essere **minimo**.

Denotiamo con $x_{i,j}$ **il flusso sull'arco** (i, j) ∈ A, il **modello di PL del problema del flusso di costo minimo.**

$$\min \quad \varphi = \sum_{(i,j)\in A} c_{i,j} x_{i,j}$$

← Minimizzazione del costo complessivo

$$s.a \quad \sum_{(j,i)\in BS(i)} x_{j,i} - \sum_{(i,j)\in FS(i)} x_{i,j} = b_i \qquad \forall i \in N$$

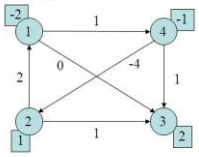
← La totalità dei flussi deve essere tale da rispettare il vincolo di bilancio ai nodi.

$$0 \le x_{i,j} \le u_{i,j} \qquad \forall (i,j) \in A$$

Il modello presenta una variabile decisionale $x_{i,j}$ per ogni arco $(i,j) \in A$.
Le variabili decisionali sono vincolate ad essere nonnegative e minori, o al più uguali, della capacità $u_{i,j}$ dell'arco $(i,j) \in A$.

Flusso di Costo Minimo
Applicazione di algoritmi: Cammini Minimi Successivi (SSP)

Esercizio 1 Sia data la seguente rete di flusso, in cui i valori riportati vicino agli archi sono i costi, e quelli riportati vicino ai nodi sono le richieste di flusso (ovvero i termini noti *b*)



L'arco (4,2) ha capacità superiore 3 e capacità inferiore 1; l'arco (2,3) ha capacità superiore 1 e capacità inferiore -1; gli altri archi hanno capacità superiore 2 e capacità inferiore 0.

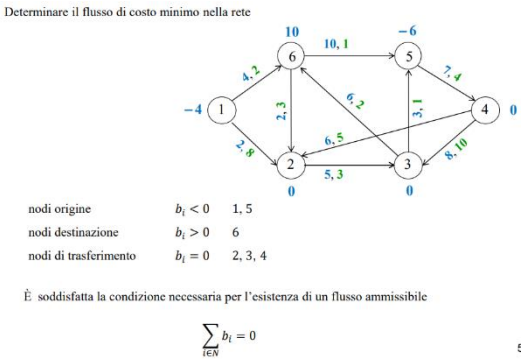
5.1 Algoritmo ad eliminazione dei cicli di costo negativo

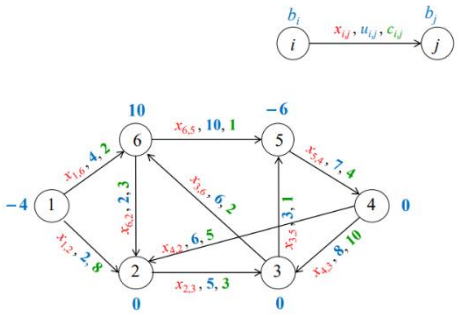
Il **problema del flusso di costo minimo** viene affrontato e **risolto** dall'**algoritmo ad eliminazione dei cicli** di **costo negativo.**

1: Dato un grafo orientato, la prima cosa da fare è verificare se il problema è in **equilibrio**:

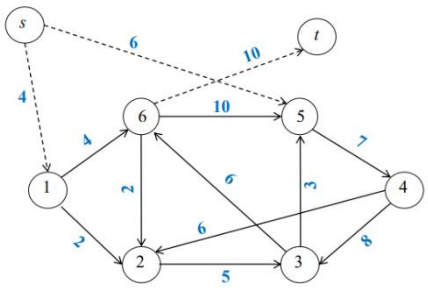
$$\sum_{i\in N} b_i = 0$$

Se il problema è in equilibrio è soddisfatta la **condizione necessaria** per **esistenza di un flusso ammissibile**.





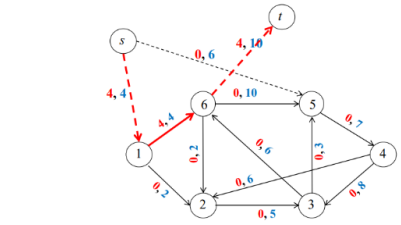
2: Fase di inizializzazione: determinare se esiste un **flusso ammissibile**. Per determinare l'esistenza di un flusso ammissibile da usare come soluzione iniziale del processo di ottimizzazione, si risolve un problema di massimo flusso su una **rete amplicata** così costruita.

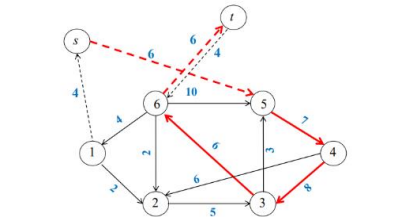


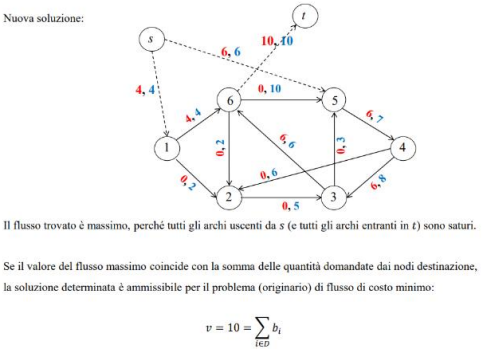
- Aggiungere al grafo G una **sorgente fittizia s** e un **pozzo fittizio t**
- Per ogni **nodo origine i** $(b_i < 0)$ definire **l'arco fittizio** (s,i) di **capacità** $u_{s,i} = -b_i$
- Per ogni **nodo destinazione i** $(b_i > 0)$ definire **l'arco fittizio** (i,t) di **capacità** $u_{i,t} = b_i$
- Per tutti gli **altri archi** la **capacità coincide con quella del problema originario**

3: Trattandosi di un **problema di massimo flusso**, per risolverlo si utilizza **l'algoritmo di Ford-Fulkerson**.

- Alla fine dell'algoritmo **otterrò un valore del flusso** v







4 controllo ammissibilità

- Il problema di flusso di costo minimo ha **soluzioni ammissibili (per il problema originario) se:**

$$v = flusso\ massimo = \sum_{i \in D} b_i = -\sum_{i \in O} b_i$$

L'aver trovato un **flusso massimo che satura gli archi** è la prova che la **soluzione ottima** per la rete ampliata è **ammissibile** per il problema originario, ciò è il punto di partenza per **processo di ottimizzazione.**

- Il problema di flusso di costo minimo **non ha soluzioni ammissibili se:**

$$v < \sum_{i \in D} b_i$$

5 Disponendo di una soluzione ammissibile bisogna **verificarne l'ottimalità**. Il processo di verifica dell'ottimalità viene svolto **costruendo** una opportuna **rete incrementale**. Uso la **rete incrementale per** cercare un **flusso di uguale valore** v e di **costo minore.**

- Per ogni arco (i,j) **non saturo** $(x_{i,j} < u_{i,j})$ si definisce **arco diretto** (i,j) con:
  - **capacità residua** $\bar{u}_{i,j} = u_{i,j} - x_{i,j}$ e **costo unitario** $\bar{c}_{i,j} = c_{i,j}$

- Per ogni arco (i,j) **non vuoto** ($x_{i,j} > 0$) si definisce **arco inverso** (j,i) con **capacità residua** $\bar{u}_{j,i} = x_{i,j}$ e **costo unitario** $\bar{c}_{j,i} = -c_{i,j}$
- Per ogni arco (i,j) **ne vuoto ne saturo** ($0 < x_{i,j} < u_{i,j}$) si definiscono **arco diretto che arco inverso**.

6 Il **valore del flusso non può essere modificato**, deve rimanere v, perciò le uniche modifiche che si possono apportare al flusso corrente sono le **redistribuzioni del flusso**, ossia variazioni su archi che compongono un ciclo C. Le **variazioni di flusso sugli archi** che compongono un ciclo devono:

- **Ridurre il costo totale del flusso**: il ciclo C sulla rete incrementale deve essere di costo negativo.
- **Soddisfare i vincoli di nonnegavitità** e **capacità sugli archi.**
- **Soddisfare** i **bilanci** ai nodi.

7 Le iterazioni **proseguono** fino a quando sono **presenti cicli** di costo **negativo** sulla rete incrementale G(x). La prova di **ottimalità del flusso** la si ha nel momento in cui **non sono più presenti cicli di costo negativo all'interno della rete incrementale. (quando non ho più cicli vuol dire che non posso più diminuire *il* costo e quindi termino l'algoritmo.)**

6 Il problema dello zaino (Knapsack Problem, *KP*)

Sono dati n elementi (o oggetti), 1, 2, … , n, ad ognuno dei quali è associata una **coppia di interi positivi**

- $\boldsymbol{p_j} > 0$: **peso** dell'elemento (caratteristica **negativa**)
- $\boldsymbol{v_j} > 0$: **valore** dell'elemento (caratteristica **positiva**)

Problema dello zaino in forma di massimo: **scegliere gli oggetti da inserire nello zaino in modo che**

- la somma dei loro pesi **non superi** la **capacità** $\bar{P}$ dello zaino
- la **somma** dei loro **valori** sia **massima**

$$\max V = \sum_{j=1}^{n} v_j \, x_j \;, \quad massimizzo\ il\ valore\ dell'oggetto.$$

problema $\overline{KP}$

$$\sum_{j=1}^{n} p_j \, x_j \leq \bar{P}\,, \quad la\ somma\ degli\ oggetti\ che\ ho\ preso \leq capacità\ max.$$

$$x_j \in \{0,1\} \quad 1 \leq j \leq n$$

$\boldsymbol{x_j}$ è una **variabile decisionale binaria**

- $x_j = 0$ indica che l'oggetto j-esimo **non** deve essere **inserito** nello zaino
- $x_j = 1$ indica che l'oggetto j-esimo deve essere **inserito** nello zaino.

Problema dello zaino in forma di minimo: **scegliere gli oggetti da inserire nello zaino in modo che**

- la somma dei **valori** degli **oggetti** scelti sia **non minore** di un **valore minimo** $\bar{V}$
- la **somma** dei **pesi** degli oggetti scelti sia **minima**

$$\min P = \sum_{j=1}^{n} p_j \, x_j \quad minimizzo\ il\ peso\ degli\ oggetti$$

$$\sum_{j=1}^{n} v_j \, x_j \geq \bar{V}\,, somma\ valore\ oggetti\ maggiore\ valore\ minimo$$

$$x_j \in \{0,1\} \quad 1 \leq j \leq n$$

Esempio

Insieme di 5 oggetti: $\{A, B, C, D, E\}$

	A	B	C	D	E
v_j	28	36	3	16	6
p_j	10	12	2	7	3

Capacità dello zaino: $\bar{P} = 21$

Ordiniamo gli oggetti in ordine non crescente di valore per unità di peso ($VUNC$)

j	B	A	D	E	C
$\frac{v_j}{p_j}$	$\frac{36}{12} = 3$	$\frac{28}{10} = 2.8$	$\frac{16}{7} = 2.28$	$\frac{6}{3} = 2$	$\frac{3}{2} = 1.5$
	x_1	x_2	x_3	x_4	x_5

$$\max V = 36\,x_1 + 28\,x_2 + 16\,x_3 + 6\,x_4 + 3\,x_5$$
$$12\,x_1 + 10\,x_2 + 7\,x_3 + 3\,x_4 + 2\,x_5 \leq 21$$
$$x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}$$

Per ogni oggetto calcolo il valore per unità di peso, valore/peso, poi li ordino in valore non crescente, ossia x2 x1 x4 x5 x3. Prendo **interamente** x2, ossia 12 di peso su 21, poi considero x1, con peso 10, dato che sono nel **rilassamento** ne prendo **9 unità** e **ne lascio 1**, ossai ne prendo 9/10, **saturo lo zaino** e ho resto nullo, poi pongo le **altre variabili = 0**

Per risolvere il problema dello zaino si utilizza una versione specializzata dell'algoritmo **Branch&Bound.** Tale algoritmo è particolarmente efficiente per questo tipo di problema, perché la soluzione del problema che deve essere risolto ad ogni nodo dell'albero di Branch&Bound è una **soluzione** che può essere **espressa** in **forma chiusa**. Il fatto che la soluzione ad ogni nodo possa essere espressa in forma chiusa comporta che questa possa essere calcolata con una formula ad-hoc.

1 Considero il **rilassamento continuo** **del problema dello zaino** (ossia non considero variabili che prendono solo valori di 0,1 ma **prendono** anche **frazioni** di **0,1**) :

$$\overline{KP}: \max V = \sum_{j=1}^{n} v_j\, x_j$$

$$\sum_{j=1}^{n} p_j\, x_j \leq \bar{P}$$

$$0 \leq x_j \leq 1 \quad 1 \leq j \leq n$$

2 La **soluzione ottima del problema $\overline{KP}$**, rilassamento continuo del problema KP, si ottiene:

- ordinando gli oggetti in **ordine non crescente** di valore per **unità di peso** (**VUNC**)
- **inserendo** nello zaino gli oggetti nell'ordine VUNC **fino** all'oggetto **k-esimo**, oggetto oltre il quale si **oltrepassa il peso dello zaino**

$x_j = 1$	$1 \leq j \leq k-1$	Oggetto inserito nello zaino
$x_k = \frac{\bar{P} - \sum_{j=1}^{k-1} p_j\, x_j}{p_k}$		La variabile x_k, $0 \leq x_k \leq 1$, esprime la **frazione dell'oggetto k** inserendo la quale la capacità dello zaino è **interamente utilizzata.** x_k è il rapporto tra • la capacità residua dello zaino dopo l'inserimetno dei primi k-1 oggetti • il peso p_k dell'oggetto k
$x_j = 0$	$k+1 \leq j \leq n$	Oggetti non inseriti nello zaino

- Fino all'oggetto **k-1** esimo gli oggetti vengono **introdotti interamente** nello zaino.
- L'oggetto **k-esimo** viene introdotto **parzialmente**.
- Tutti gli oggetti per **k>0 non** vengono **introdotti** per mancanza di spazio.

3 Per **dimostrare l'ottimalità** della soluzione bisogna **costruire il duale del problema rilassato $\overline{KP}$.**
È possibile definire problemi duali solo di problemi a **variabili continue**, non esiste una formulazione duale per problemi a variabili intere o binarie.
A tal fine si utilizza la **relazione fondamentale tra primale e duale**, così ottengo il **duale:**

$$\omega + \sum_{i=1}^{m} s_i \cdot y_i + \sum_{j=1}^{n} x_j \cdot z_j = \varphi_D$$

(valore obiettivo problm max + 2 sommatorie di variab complementari = valore obiettivo problm min)

Faccio varie sostituzioni sui simboli (ω, s_i, x_j), Il risultato è:

Problema $\overline{KP}$:

$$\max V = \sum_{j=1}^{n} v_j\, x_j$$

$$x_j \le 1 \qquad 1 \le j \le n \qquad [y_j \ge 0]$$

$$\sum_{j=1}^{n} p_j\, x_j \le \bar{P} \qquad [y_{n+1} \ge 0]$$

$$x_j \ge 0 \qquad 1 \le j \le n$$

Duale del problema $\overline{KP}$:

$$\min \sum_{j=1}^{n} y_j + \bar{P} \cdot y_{n+1}$$

$$y_j + p_j\, y_{n+1} \ge v_j \qquad 1 \le j \le n \qquad [x_j \ge 0]$$

$$y_j \ge 0 \qquad 1 \le j \le n$$

$$y_{n+1} \ge 0$$

Il primo vincolo del duale mi dice che ho **prodotti vincolati ad essere non negative** ($y_j + p_j\, y_{n+1} \ge v_j$), ho 'n' vincoli duali di questo tipo.

4 **L'ottimalità** viene **dimostrata** verificando che la **soluzione di base duale complementare è ammissibile.** (Quando è ottima devo solo applicare l'espressione e non ho bisogno di fare un simplesso)

6.1 Algoritmo di Branch-and-Bound per *KP*

Nell'algoritmo di **Branch-and-Bound** per KP è usato il **rilassamento lineare** KP

$$\max V = \sum_{j=1}^{n} v_j\, x_j$$

$$x_j \le 1 \qquad 1 \le j \le n$$

$$\sum_{j=1}^{n} p_j\, x_j \le \bar{P}$$

$$x_j \ge 0 \qquad 1 \le j \le n$$

in cui gli **oggetti** sono preventivamente ordinati in **ordine non crescente di valore per unità di peso**.

Inizializzazione: valore della migliore **soluzione ammissibile finora trovata: $-\infty$**

Iterazione:

Si risolve P_0:

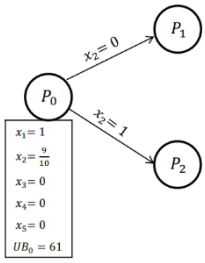
	x_1	x_2	x_3	x_4	x_5
v_j	36	28	16	6	3
p_j	12	10	7	3	2
x_j	1	$\frac{21-12}{10}$	0	0	0

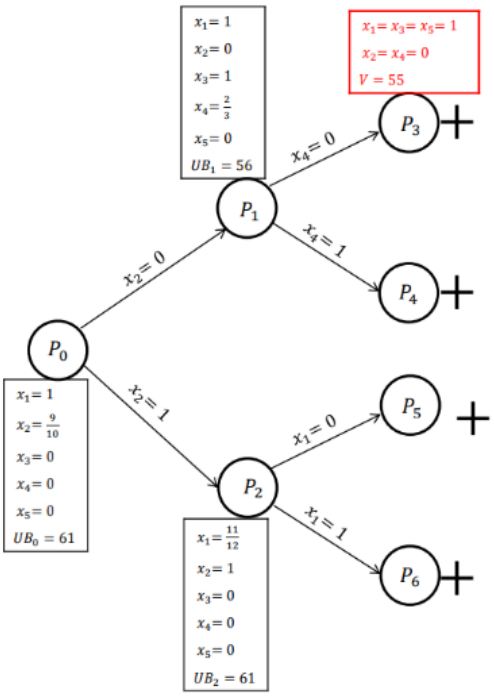
$$P = 12 + 10 \cdot \frac{9}{10} = 21$$

$$V = 36 + 28 \cdot \frac{9}{10} = 61.2$$

$$UB_0 = \lfloor V \rfloor = 61$$

- (noto che prendo x1 per intero e già x2 frazionario)
- La soluzione ottima di $P0$ **non è ammissibile per KP.**
- Essendo $UB0 = 61 > -\infty$ (valore della migliore soluzione ammissibile finora trovata), Si esegue il **branching sulla variabile con valore frazionario xj** nella soluzione ottima, ossia $x2$**:**
- Si impostano i **problemi figli:**
  - problema figlio $P1$: si **esclude** la presenza dell'elemento 2 ($x2 = 0$)
  - problema figlio $P2$: si **impone** la presenza dell'elemento 2 ($x2 = 1$)
- continuo ad iterare finchè non trovo **soluzioni ammissibili al problema originario**, ossia con **soluzioni xj intere**.
  - Se lo trovo **chiudo** il nodo.
- Se trovo **soluzioni** con **upper bound inferiore** a **soluzioni ammissibili**, le **chiudo**.





$$\max V = \sum_{j=1}^{n} v_j x_j$$

$$x_j \le 1 \qquad 1 \le j \le n \qquad [y_j \ge 0]$$

$$\sum_{j=1}^{n} p_j x_j \le \bar{P} \qquad [y_{n+1} \ge 0]$$

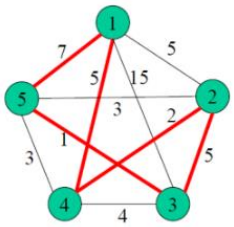
$$x_j \ge 0 \qquad 1 \le j \le n$$

$$\begin{aligned}
\max V = \quad & 36\,x_2 + 28\,x_1 + 16\,x_4 + 6\,x_5 + 3\,x_3 & & & \\
\text{s.a} \quad & x_2 & & \le 1 & y_1 \\
& & x_1 & \le 1 & y_2 \\
& & x_4 & \le 1 & y_3 \\
& & x_5 & \le 1 & y_4 \\
& & x_3 & \le 1 & y_5 \\
& 12\,x_2 + 10\,x_1 + 7\,x_4 + 3\,x_5 + 2\,x_3 & & \le 21 & y_6 \\
& x_2 \ge 0 \quad x_1 \ge 0 \quad x_4 \ge 0 \quad x_5 \ge 0 \quad x_3 \ge 0 & & &
\end{aligned}$$

7 TSP: Il problema del commesso viaggiatore (Traveling Salesman Problem, *TSP*)

Dato un grafo, **orientato o non orientato** $G = (N, A)$ con **pesi** $c_{i,j}$ associati agli archi $(i, j) \in A$, determinare una sequenza di **visita dei nodi** (detta circuito o **ciclo Hamiltoniano**) che:

- parte da un nodo qualsiasi, **passa una sola volta per tutti gli altri nodi e torna al nodo iniziale**.
- **minimizza** la **somma** dei **pesi** degli **archi** utilizzati per **passare da un nodo a un altro.**



Il ***TSP*** **si dice:**

- **simmetrico** se $c_{i,j} = c_{j,i} \ \forall (i,j)$ → **grafi non orientati**
  - dal nodo 4 al nodo 5 e viceversa, hanno stesso costo, perché non orientati.
  - La matrice dei costi sarà simmetrica rispetto la diagonale.
- **asimmetrico** se $c_{i,j} \neq c_{j,i} \ \forall (i,j)$ → **grafi orientati**

7.1 formulazione PLI del *TSP* Asimmetrico

$$\min \sum_{(i,j)\in A} c_{i,j}\, x_{i,j}$$

$$\sum_{(j,i)\in BS(i)} x_{j,i} = 1 \qquad i \in N$$

$$\sum_{(i,j)\in FS(i)} x_{i,j} = 1 \qquad i \in N$$

$$\sum_{i\in Q} \sum_{j\in N\setminus Q} x_{i,j} \geq 1 \qquad \forall\, Q \subset N,\ |Q| \geq 1$$

$$x_{i,j} \in \{0,1\}$$

- *Variabile decisionale* $x_{i,j}$ ***binaria***: **inserisco l'arco nel cammino oppure no**. Quindi per ogni arco dico se viene messo nel ciclo hamiltoniano o meno.
- **1° vincolo:** Ad ogni nodo i si arriva da un unico nodo j, ossia dalla sua stella entrante, c'è un ***j*** **unico predecessore di *i***. ($\sum x_{j,i} = 1$).
- **2° vincolo**: impongo anche che ogni nodo ***i*** **abbia un unico successore *j*** ($\sum x_{i,j} = 1$).
- **3° vincolo** Per ogni **partizione** Q, N\Q (si dice N meno Q) esiste **almeno un arco** che **collega** un **nodo di Q** con un **nodo non di Q, ossia N\Q.**
  - **le partizioni** sono i possibili **percorsi alternativi** che posso **usare** per **creare il cammino.**
  - **Devo avere almeno 1 possibile cammino**
- **Funzione obiettivo**: **minimizzazione del costo del cammino hamiltoniano**.

7.2 formulazione PLI del *TSP* Simmetrico

$$\min \sum_{(i,j)\in A} c_{i,j}\, x_{i,j}$$

$$\sum_{(j,i)\in S(i)} x_{i,j} = 2 \qquad i \in N$$

$$\sum_{i\in Q}\sum_{j\in N\setminus Q} x_{i,j} \geq 1 \qquad \forall\, Q \subset N,\ |Q| \geq 1$$

$$x_{i,j} \in \{0,1\}$$

1° vincolo: per ogni nodo i, data la stella del nodo, la **somma dei valori delle variabili decisionali** associate agli **archi** della **stella** deve essere **pari a 2** (ovvero: per ogni nodo del grafo non orientato ci devono essere **solo due lati incidenti all'interno del cammino scelto**).

- **S** è l'**insieme degli archi** archi **appartenenti al circuito**

2° vincolo: considerando qualsiasi sottoinsieme Q in N con almeno un elemento al suo interno, devo considerare solo gli **archi che collegano un nodo** dell'insieme **Q ad** uno dell insieme **Q\N.**

Il 2° vincolo è presente perché il 1° vincolo non assicura l'esistenza di un unico ciclo Hamiltoniano: assicura che esiste almeno un ciclo Hamiltoniano, ma non assicura che questo sia unico.
L'obiettivo è quello di **avere un unico ciclo Hamiltoniano**.

Il 2° vincolo serve per impedire che si accetti una soluzione dalla quale si possano generare dei sottocicli disgiunti. Per evitare questa situazione bisogna impedire che gli archi del taglio indotto dalla partizione dei nodi abbiano asssociato valore 0 alla loro $x_{i,j}$. Ecco perché il vincolo ha come condizione ≥ 1.

7.3 Algoritmi *greedy* per TSP

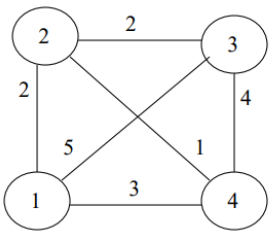
Dal punto di vista algoritmico, il **TSP viene risolto da algoritmi greedy**.
Un algoritmo greedy è un tipo di algoritmo di ottimizzazione che, ad ogni passo, **seleziona la scelta che al momento sembra la migliore senza considerare le conseguenze delle scelte future**. Per far ciò, l'algoritmo utilizza una **funzione di valutazione** per **determinare** la **scelta migliore ad ogni passo**.

Tutti gli algiritmi prevedono due fasi:

1. **Inizializzazione**:
   a. $S = \emptyset$, l'insieme ***S*** degli **archi appartenenti** al **circuito** è vuoto.
   b. Pongo il nodo corrente come **nodo iniziale.**
2. **Iterazioni**:
   a. si individuano **tutti** gli **archi** che **uniscono** il **nodo corrente *i*** a nodi **non ancora toccati** dagli **archi contenuti in S.**
   b. si **seleziona** tra essi l'arco (i,j) "**più promettente**" secondo un dato **criterio euristico** e lo **aggiunge ad *S*.**
   c. si definisce ***j*** come **nuovo nodo corrente.**
3. **Terminazione**: **L'algoritmo termina** quando **tutti i nodi sono toccati dagli archi contenuti in S.**
   a. A questo punto si inserisce **l'arco di ritorno dall'ultimo nodo al nodo iniziale.**

7.3.1 greedy per *TSP*: algoritmo "Nearest Neighbour"

Il **criterio euristico** per selezionare l'arco **più promettente** è scegliere l'**arco di lunghezza minima** tra il **nodo corrente** e i nodi non ancora raggiunti
(la prossima tappa è verso la località più vicina a quella in cui il commesso si trova attualmente)



	nodi visitati
• passo 1: (1, 2): 2, (1, 3):5, (1, 4):3	1, 2
• passo 2: (2, 1), (2, 3):2, (2, 4):1	1, 2, 4
• passo 3: (4, 1), (4, 2), (4, 3):4	1, 2, 4, 3

Tutti i nodi sono stati visitati: si aggiunge l'arco (3,1):5

Costo del circuito Hamiltoniano determinato: 12

Il circuito ottimo è 1 – 4 – 2 – 3 di costo 11.

- Nodo corrente: 1, posso andare verso 2,3,4.
- Scelgo 2, qui posso scegliere se 3,4
- Scelgo 4, poi scelto 3 per forza.
- Concludo aggiungendo l'arco (3,1) che è **l'arco di ritorno.**

Ottengo un costo di 12, questa è una soluzione greedy dato che la soluzione ottima è 11.

- se il grafo G è **completo**:
  - l'algoritmo **greedy** costruisce **sicuramente un circuito Hamiltoniano**
- se il grafo G **non è completo**:
  - può accadere che tutti gli archi uscenti dal nodo corrente i **portino a nodi già visitati** dagli archi di S: l'algoritmo greedy può non produrre una soluzione ammissibile, anche qualora una soluzione ammissibile esista.

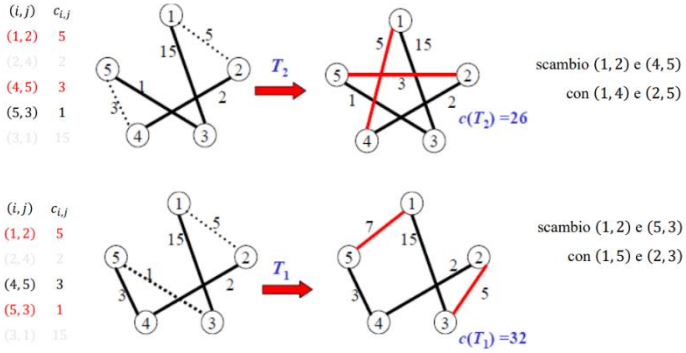
7.3.2 Algoritmo greedy per *TSP*: con ordinamento degli archi in ordine non decrescente di costo

è un'**euristica costruttiva**, nel senso che determina una soluzione senza bisogno di una precedente stima, viene quindi determinata partendo da zero.
Ordina gli archi in ordine non decrescente di costo e crea una struttura che mi consente di arrivare ad una **soluzione ammissibile** e a queste modalità di **ricerca greedy associa** una **seconda fase** chiamata fase di ricerca locale o **Local search.**

Local search:

- è un **euristica migliorativa**: **parte da una soluzione ottenuta sfruttando un algoritmo greedy e cerca il modo di migliorarla.**
- Ottenuta una soluzione ammissibile si cerca di considerare le **soluzioni appartenente all intorno della soluzione corrente**. In questo caso l'intorno è chiamato "**2-scambio**".

Il **2-scambio** **scambia 2 archi che fanno parte del cammino** con altri **2 archi che non fanno parte** del cammino. Questa operazione viene valutata **per tutti i possibili scambi** da fare in questo modo.



Iterazioni:
1. Prendo l'elenco dei costi scritti in **ordine decrescente**, **escludo gli archi adiacenti** (2,4), (3,1) perché sono **adiacenti** a (1,2).
2. Gli **archi non adiacenti** sono (4,5) e (5,3) **considero 2 casistiche in cui li accoppio ad (1,2).**
   a. Quindi ottengo coppia di archi (1,2)(4,5) nella figura sopra e (1,2)(5,3) nella figura sotto.
3. Ora ad entrambi gli archi **scambio gli elementi intermedi**, quindi da (1,2)(4,5) ottengo (1,4)(2,5)

Devo **cambiare per forza almeno 2 archi** perché 1 solo cambiamento non mi porta a nessuna conclusione.

7.3.3 TPS metrico

È un **euristica** che cerca di tenere traccia del **possibile errore di approssimazione compiuto**.
Dato un grafo $G = (N, A)$ **non orientato**, completo con costi $c_{i,j}$ associati agli archi.

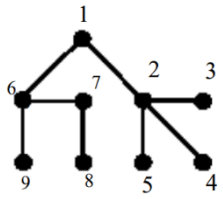
Definisco un TPS metrico se:

- $c_{i,j} \geq 0$ per ogni arco $(i,j) \in A$. **Costi non negativi**
- $c_{i,j} = 0$ se e solo se $i = j$. **Costo nullo** su **autoanelli**
- $c_{i,j} \leq ci,k + ck,j$ (**disuguaglianza triangolare**) per ogni terna di nodi distinti i, j, k.
  - Il costo per andare da i-j è sempre minore di quello da i-j passando per un k intermedio
  - Passaggio **diretto** sempre **minore** a **intermedio**

Per **affrontare questo tipo di problema** si usa **l'algoritmo taMST** (twice around Minimum spanning tree).
Tale algoritmo è applicabile quando:

- **grafo G è completo** (c'è un arco per ogni coppia di nodi, tutti i nodi connessi a tutti)
- e il **TSP è metrico**.

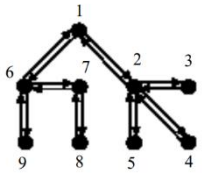
7.3.3.1 taMST (twice around Minimum spanning tree).



Immagino di avere un arco con *n* nodi, e **completo**, ossia con **un arco ogni coppia di nodi**, questa struttura è **l'albero di supporto di costo minimo** (ottenuto con prim o kruskal)

L'algoritmo taMST consta di **3 passi:**

1. si determina un **albero di copertura di costo minimo T***, ovvero una struttura per cui, considerando una qualunque coppia di nodi, **esiste un cammino che li connette.**
2. da T * si ottiene un **ciclo orientato** (detto **ciclo non semplice**, dato che i nodi si ripetono):
   a. per fare ciò si **duplicano gli archi di T*** (in archi discordi), e si attribuiscono agli **archi orientati** lo **stesso costo** degli **archi non orientati** originali:
   b. così **costituisco un ciclo**, o un cammino chiuso, come si vede nell'esempio sotto, 1-2-3-2…,
      i. **non è però Hamiltoniano** perché **passo più volte sullo stesso nodo**, lo ottengo al passaggio 3.



ciclo orientato non semplice:

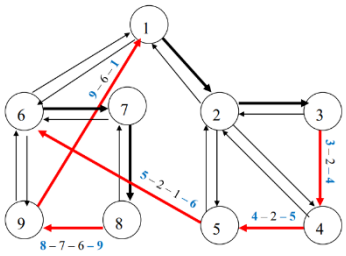
1 – 2 – 3 – 2 – 4 – 2 – 5 – 2 – 1 – 6 – 7 – 8 – 7 – 6 – 9 – 6 – 1

3. dal ciclo orientato (non semplice, chiamato così perché passo più volte sullo stesso nodo) si ottiene il **ciclo Hamiltoniano *H*** **eliminando** la **ripetizione** dei **nodi** mediante **l'operazione di "scorciatoia"**
   a. si **parte da un qualsiasi nodo**, detto nodo 1.

b. il **successore** del nodo corrente i nel ciclo Hamiltoniano viene determinato esaminando i **nodi successori** di i **sul ciclo Hamiltoniano fino** ad incontrare **un nodo h non ancora raggiunto dal ciclo Hamiltoniano**: il **cammino da i ad h** sul ciclo orientato non semplice è **sostituito nel ciclo Hamiltoniano dalla "scorciatoia" (i, h),** che **esiste sempre perché il grafo è completo**. Il **procedimento viene ripetuto a partire dal nodo h.**

c. Ossia, sto facendo 1-2-3, ora vorrei andare a 4, per andarci prima facevo 1-2-3-2-4, così però passo 2 volte sul 2, ora invece con la disuguaglianza triangolare spendo uguale o meno facendo direttamente 3-4, questa operazione è la scorciatoia, quindi da 3 faccio il salto a 4, poi da 4 salto a 5, **salto i nodi già visitati e vado a quelli non visitati.**

d. Quando **viene visitato l'ultimo nodo**, il **ciclo** viene **chiuso** con l'aggiunta **dell'arco che conduce** al **nodo iniziale**, ossia operando una scorciatoia dall'ultimo nodo visitato al nodo 1.



In **nero**: archi cammino orientato non semplice
In **rosso**: operazioni di scorciatoia.

Ho così ottenuto un cammino hamiltoniano

Gli archi rossi rappresentano le scorciatoie operate sul ciclo orientato non semplice:

1 – 2 – 3 – 2 – 4 – 2 – 5 – 2 – 1 – 6 – 7 – 8 – 7 – 6 – 9 – 6 – 1

(vicino agli archi rossi è riportato il cammino evitato dalla scorciatoia)

mediante le quali si determina il ciclo Hamiltoniano

H = 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 1

Poiché i costi soddisfano la disuguaglianza triangolare, ogni operazione di scorciatoia non incrementa la lunghezza del ciclo: la lascia inalterata o la riduce.

Ora ho determinato **un ciclo Hamiltoniano H** con taMST, faccio delle **osservazioni sul costo c(H):**

- Il costo è **limitato superiormente**:

  $$c(H) \leq 2c(T^*) \quad \textbf{(maggiorante)}$$

  - c(H) **costo ciclo Hamiltoniano**
  - $c(T^*)$ **costo albero di supporto minimo** da cui sono partito. la T^* per indicare che è **ottimo.**
    - Lo moltiplico per "**2**" perché l'ho **sdoppiato** creando gli **archi discordi**

- Il costo è **limitato inferiormente**

  $$c(H^*) \leq c(H) \quad \textbf{(minorante)}$$

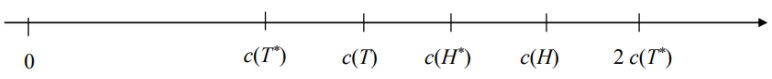
  - $c(H^*)$ **costo del ciclo Hamiltoniano di costo minimo**, è **incognito**.
  - Posso però ottenere un **minorante** di $c(H^*)$ così da dare senso alla disequazione sopra.
  - $c(H^*)$ è **limitato inferiormente** dal costo dell'albero di copertura T di G
    - Lo ottengo eliminando un qualsiasi arco da H*, così ho n-1 archi

      $$c(T) \leq c(H^*)$$

  - Fin dall'inizio avevo già calcolato il **costo albero di supporto minimo** (T^*)

    $$c(T^*) \leq c(T)$$

- Risulta quindi definita la catena di disuguaglianze $c(T^*) \leq c(T) \leq c(H^*) \leq c(H) \leq 2c(T^*)$



L'errore relativo compiuto dall'euristica taMST è definito da:

$$ER_{taMST} = \frac{c(H) - c(H^*)}{c(H^*)}$$

L'equazione per il calcolo dell'errore relativo è espressa in funzione di $c(H^*)$, incognito.
Per riscrivere l'equazione mi servo delle diseguaglianze della catena:

sostituisco $c(H) \leq 2c(T^*)$ e ottengo $c(H) - c(H^*) \leq 2c(T^*) - c(T^*)$,

ossia $c(H) - c(H^*) \leq c(T^*)$

ne segue che:

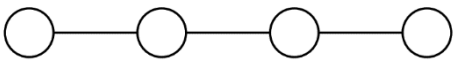
$$ER_{taMST} = \frac{c(H) - c(H^*)}{c(H^*)} \leq \frac{c(T^*)}{c(T^*)} = 1$$

L'**errore relativo** ha come **upper bound 1**, quindi **non supera mai 1.**

7.2.1 Algoritmo di Branch-and-Bound per *TSP* simmetrico

Ad **ogni nodo** viene **determinato** un **1-albero di costo minimo**, ossia un **sottografo di G**, costituito **da n collegamenti** (lati), che **differisce da ciclo Hamiltoniano per un solo elemento**,
è ottenuto con la seguente procedura:

1) i **primi $n-1$ lati** sono **determinati costruendo** **un** **albero di supporto di costo minimo (T^*)** con nodi di **grado al più pari a 2** (numero di collegamenti di ogni nodo max 2)



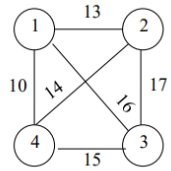
2) **l'n-esimo lato** è il **lato di costo minimo** tra quelli non appartenenti al T^* del punto 1)
   a. se il **lato di costo minimo** **chiude** la struttura, allora ho un **ciclo Hamiltoniano**
   b. se il **lato di costo minimo** **non chiude**, la struttura è chiamata **1-albero di costo minimo.**

Il costo di un **1**-albero di costo minimo è un lower bound del costo del ciclo Hamiltoniano ottimo

Il problema della **determinazione di un 1-albero è quindi un rilassamento del TSP** dato che fornisce:

- o una soluzione **non ammissibile** **per *TSP***, il cui costo è un lower bound del costo del cammino Hamiltoniano ottimo
- o una **soluzione ammissibile** **per *TSP*.**

Dato il TSP simmetrico definito sul seguente grafo $G = (N, A)$ pesato

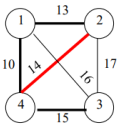


$\{1,4\}$ 10
$\{1,2\}$ 13
$\{2,4\}$ 14
$\{3,4\}$ 15
$\{1,3\}$ 16
$\{2,3\}$ 17

determinare la soluzione con il metodo di Branch-and-Bound

P_0: determinare un 1-albero sul grafo $G = (N, A)$

{1, 4} 10
{1, 2} 13
{2, 4} 14 : non lo aggiungo perché crea un sottociclo
{3, 4} 15 : stop, perché ho il cammino 2 - 1 - 4 - 3
{1, 3} 16
{2, 3} 17

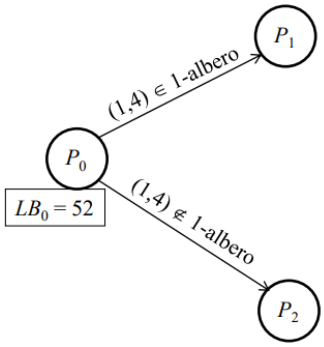


A questo cammino unisco il **lato {2, 4}**,
che è quello **di costo minimo tra i lati non appartenenti al cammino**.

Questa soluzione non è ammissibile per il TSP : si ottiene il lower bound

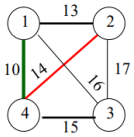
$LB_0 = 10 + 13 + 15 + 14 = 52$

ordino i collegamenti in ordine crescente, prendo ogni collegamento e inserisco nella struttura solo quelli validi. (2,4) non lo prendo perché crea sottociclo. **(1,3)(2,3) non li considero perché ho gia il cammino.** Fatto ciò unisco alla struttura il **lato rimanente di costo minimo, quindi (2,4)**, esso **non chiude il cammino ed ha grado 3**, quando il **max è 2**, infatti questa **soluzione non è ammissibile per il TSP.**



P_1: $\{1,4\} \in$ 1-albero

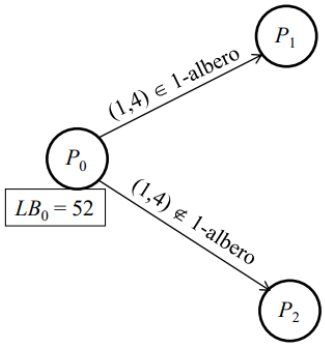
{1, 4} 10
{1, 2} **13**
{2, 4} 14 : non lo aggiungo perché crea un sottociclo
{3, 4} **15** : stop, perché ho il cammino 2 - 1 - 4 - 3
{1, 3} 16
{2, 3} 17



A questo cammino unisco il lato **{2, 4}**,
che è quello di costo minimo tra i lati non appartenenti al cammino.

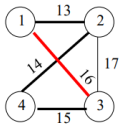
Si ottiene la stessa soluzione di P_0 e quindi $LB_1 = LB_0 = 52$

Ora ho 2 problemi in cui ogni problema ha 1 vincolo in più rispetto a quello di partenza.
Ho il lato (1,4) imposto, poi ho il solito procedimento. Prendo (2,4) come lato che costa di meno.
Ho lo stesso problema di prima.



P_2: $\{1,4\} \notin$ 1-albero

Il lato $\{1,4\}$ viene rimosso dall'elenco perché non deve essere contenuto nell'1-albero:
{1, 2} **13**
{2, 4} **14**
{3, 4} **15** : stop, perché ho il cammino 1 - 2 - 4 - 3
{1, 3} **16**
{2, 3} 17

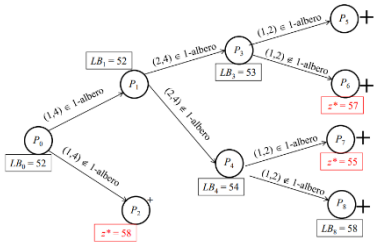


A questo cammino unisco il lato **{1, 3}**,
che è quello di costo minimo tra i lati non appartenenti al cammino.

È una soluzione ammissibile per TSP, di valore $z^* = 58$

Il nodo P_2 viene chiuso perché ha generato una soluzione ammissibile per TSP.

Esclude (1,4) in partenza. Il lato che costa meno da aggiungere è (1,3).
Ottengo un **ciclo hamiltoniano** di valore z* = 58.
Chiudo il nodo perché genero una soluzione ammissibile.
Procedo con gli altri nodi cercando una z* minore di 58 e ammissibile.



8.1 MCMS - Minimal Cardinality Machine Scheduling

Obiettivo: **minimizzare il numero di macchine che utilizzo**, ossia il minimal cardinality.
Dati:

- n: lavori/task da compiere.
- m: macchine su cui eseguire i tasak
- t_i: tempo di inizio del task i-esimo
- d_i: durata del task i-esimo (tempo di esecuzione)

L'obiettivo è quello di **assegnare i task alle macchine in modo tale che:**

- Su ogni macchina i **periodi di esecuzione dei task** ad essa assegnati **non si sovrappongono** (ogni task inizia non priam dell'istante di fine esecuzione dei task precedenti)
- Il **numero delle macchine da utilizzare** deve essere **minimo**.

Modello matematico:

- i: **numero lavori**
- j: **numero macchine a disposizione**

$\min \sum_{j=1}^{m} y_j$		obiettivo
$\sum_{j=1}^{m} x_{i,j} = 1$	$i = 1,\ldots,n$	vinc semiassegnamento
$x_{i,j} + x_{h,j} \le 1$	$i = 1,\ldots,n-1,\ h \in S(i),\ j = 1,\ldots,m$	vinc compatibilità
$x_{i,j} \le y_j$	$i = 1,\ldots,n,\ j = 1,\ldots,m$	macchine usate
$x_{i,j} \in \{0,1\}$	$i = 1,\ldots,n,\ j = 1,\ldots,m$	lavoro i mach j
$y_j \in \{0,1\}$	$j = 1,\ldots,m$	num macchine j

- **Variabili decisionali:**
  - Variabili di tipo **binario** a doppio indice i,j. $x_{i,j}$ mi indica che il **lavoro i è compiuto sulla macchina j**
  - Variabili di tipo **binario** a singolo indice j, per far si che la **macchina sia usata o meno.**
- **vincolo di semiassegnamento:**
  - sommatoria su j, così da indicare che il **lavoro $x_{i,j}$ sia svolto solo da una macchina j**, voglio usare il minor numero possibile di macchine j.

$$\sum_{j=1}^{m} x_{i,j} = 1 \qquad i = 1,\ldots,n$$

- **Vincoli di compatibilità:**
  - **coppie** di **lavori incompatibili** **non** possono essere **assegnati** alla **stessa macchina.**
  - due lavori **i e h,** con intervalli di lavorazione $[t_i, t_i + d_i]\ e\ [t_h, t_h + d_h]$ possono essere **assegnati** alla **stessa macchina j** solo se gli **intervalli** di **lavorazione non si sovrappongono**

$t_i + d_i \le t_h$, se il lavoro i termina prima dell'inizio del lavoro h

$t_h + d_h \le t_i$, viceversa

- avendo i dati di lavorazione t_i e d_i devo fare una **prelavorazione** nella quale **confronto**, per ogni coppia, la **sovrapposizione dei tempi di lavorazione**, e quindi **l'incompatibilità**. Se c'è il **" • " vuol dire che quella coppia di lavorazioni è incompatibile**. (figura sotto a sinistra)

$S(i) = \{\ h \in \{\ i+1,\ldots,n\ \} : [t_i\ ,\ t_i + d_i] \cap [t_h\ ,\ t_h + d_h] \neq \varnothing\ \}$

	1-5	3-8	4-6	5-9	6-7	8-11	9-13	10-12	11-15	13-16
1-5		•	•							
3-8			•	•	•					
4-6				•						
5-9					•	•				
6-7										
8-11							•	•		
9-13								•	•	
10-12									•	
11-15										•
13-16										

$S(1) = \{\ 2, 3\ \}$
$S(2) = \{\ 3, 4, 5\ \}$
$S(3) = \{\ 4\ \}$
$S(4) = \{\ 5, 6\ \}$
$S(5) = \varnothing$
$S(6) = \{\ 7, 8\ \}$
$S(7) = \{\ 8, 9\ \}$
$S(8) = \{\ 9\ \}$
$S(9) = \{\ 10\ \}$

riga 1 mi diche che il lavoro 2 e 3 è incompatibile con il lavoro 1

- per ogni lavoro *i* **l'insieme di incompatibilità S(i)** è **formato dagli indici dei lavori** cui è **associato il simbolo "•" nella riga del lavoro i** (figura sopra a destra)
- I **vincoli di compatibilità: impediscono** che le **variabili** relative a **due lavori incompatibili assumano entrambe valore 1**

$$x_{i,j} + x_{h,j} \leq 1 \qquad i = 1,\ldots,n-1,\ h \in S(i),\ \ j = 1,\ldots,m$$

  - ***h*: elemento dell'insieme S(i)**
  - ***S(i)*: insieme indici lavori incompatibili con il lavoro i.**
  - Il lavoro i sulla macchina j + il lavoro h sulla macchina j devono essere minori di 1, quindi se i,h incompatibili non possono essere assegnati alla stessa macchina j.
    - Così ho casistiche di 0+0 , oppure 0+1, oppure 1+0, non 1+1

- Avrò un **numero di vincoli** di **compatibilità non superiori** a $\frac{m \cdot n \cdot (n-1)}{2}$
  - (Num macchine * num max coppie che posso formare) / 2

- **macchine utilizzate**
  - Introduco una **variabile** $y_j \in \{0,1\}$ per ogni j, che mi **conta l'utilizzo delle macchine j.**

$$x_{i,j} \leq y_j\ , \quad i = 1,\ldots,n\ , \quad j = 1,\ldots,m \quad (m \cdot n \text{ vincoli})$$

  - $x_{i,j} \leq y_j$: $x_{i,j}$ ha valore 1, ossia **se il lavoro i è eseguito sulla macchina j**, il **vincolo forza** la variabile y_j ad **assumere valore $y_j = 1$**

8.1.1 Algoritmi greedy per MCMS

Inizializzazione:
- **nessun task è assegnato**
- **nessuna macchina è utilizzata**: **N(j) = ∅** ∀j

Iterazione:
- **selezionare**, secondo un **criterio euristico**, il **task i-esimo** tra quelli **non assegnati**
- tra le **macchine già "in uso" selezionare**, secondo un **criterio euristico**, la **macchina j-esima cui assegnare il lavoro i-esimo**
- se il **lavoro i-esimo non** può essere **eseguito** su alcuna delle **macchine "in uso",** lo si assegna ad una **nuova macchina**, che si **aggiunge all'insieme di macchine utilizzate**.

8.1.2 Particolare algoritmo greedy per MCMS che determina una soluzione ottima

il **lavoro da assegnare** viene **selezionato** in **ordine di tempo di inizio t_i non crescente** (vengono **assegnati** per **primi i lavori che iniziano prima**)

- le **macchine** vengono **ordinate in un ordine arbitrario** e, per l'inserzione del lavoro corrente, vengono esaminate sempre in tale ordinamento
- se il **lavoro non può essere inserito** nelle **macchine** attualmente **utilizzate**, sarà **attivata la successiva macchina nell'ordinamento**

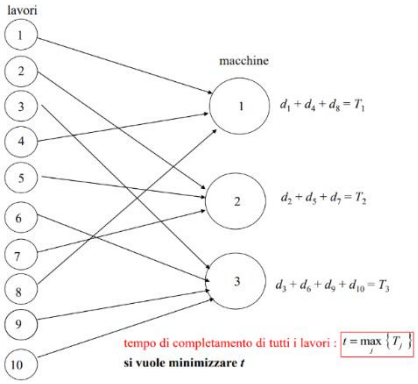
Questo algoritmo costruisce un assegnamento che **utilizza sempre il minor numero possibile di macchine.**

8.2 MMMS - Minimal Makespan Machine Scheduling

Obiettivo: decido come **suddividere i task tra le macchine** per **minimizzare il tempo di completamento di tutti i task,** ossia il **minimal makespan**. Quindi voglio che **l'ultimo task inizi ad essere lavorato prima della fine della lavorazione del primo.**
Il **makespan** è la **differenza la fine dell'ultimo task e l'inizio del primo task.**
Dati :

- *m:* macchine su cui eseguire i task.
- *n:* task.
- d_i: durata del task i-esimo.

Il tempo di inizio di ciascun task non è dato, può essere liberamente fissato.



Il **tempo di completamento T_j di ogni macchina m** è **la somma dei completamenti** dei **tempi** dei vari **task d_i,** chiamo questa somma T_j .
Voglio minimizzare il tempo di completamento di tutti i task, voglio minimizzare *t.*

- **vincolo di semiassegnamento:**
  - sommatoria su j, così da indicare che il **lavoro $x_{i,j}$** sia **svolto solo da una macchina j**, voglio usare il **minor numero possibile di macchine j.**

$$\sum_{j=1}^{m} x_{i,j} = 1 \qquad i = 1,\ldots,n$$

- **Tempo di completamento della macchina T_j** con $1 \leq j \leq m$, necessario per terminare i task assegnati

$$T_j = \sum_{i=1}^{n} d_i x_{i,j}$$

  - Sommatoria sull'insieme dei task j. d_i, tempo lavorazione moltiplicato a $x_{i,j}$ lavorazioni.

- **Funzione obiettivo**: $max_j \{T_j\}$ **da minimizzare**.
  - introduco una nuova variabile ***t*** che rappresenta una **approssimazione per eccesso del tempo di completamento**, ossia si **impone** che la **durata** della **lavorazione** su ogni **macchina non superi la quantità t incognita** (t è un maggiorante, incognito, dei tempi di lavorazione delle macchine)

- per ogni **macchina j** deve **valere** $T_j \leq t$
- **L'obiettivo** è espresso chiedendo di **minimizzare il maggiorante *t*.** (funzione obiettivo)

$$\min \quad t$$

$$\sum_{j=1}^{m} x_{i,j} = 1 \qquad i = 1,\ldots,n$$

$$\sum_{i=1}^{n} d_i x_{i,j} \leq t \qquad j = 1,\ldots,m$$

$$x_{i,j} \in \{0,1\} \qquad i = 1,\ldots,n, \quad j = 1,\ldots,m$$

Per esempio se ho 3 disequazioni di $\sum d_i\, x_{i,j}$ di valori 9,11,13, quindi il t piccolo essendo un maggiorante può assumere un qualsiasi numero maggiore o pari a 13, ma prenderò 13 perché voglio minimizzare.

8.2.1 Algoritmi greedy di list scheduling

Inizializzazione:

- tutte le **macchine sono scariche** : **N(j)** = ∅ $\quad 1 \leq j \leq m$, ossia non ho task assegnati alle macchine.
- **nessun task è assegnato**
- i **task** vengono **ordinati** in ordine **non decrescente** (**SPT**) o **non crescente** (**LPT**) dei **tempi di esecuzione**

Iterazione:

- **selezionare** il **lavoro** i tra quelli **non ancora assegnati**
- selezionare la **macchina "più scarica"**, ossia quella che, nella soluzione corrente, ha il **tempo di lavoro più basso**: tale macchina è indicata con ***j***
- si assegna il **lavoro i-esimo alla macchina j-esima** ($x_{i,j}$**= 1**), ossia si inserisce ***"i"*** **in** N_j , e si **incrementa** il **tempo** di **utilizzo** della **macchina j-esima**: $T_j = \sum_{i=1}^{n} d_i\, x_{i,j}$

Terminazione : **tutti i lavori sono stati assegnati**.

Criteri per **l'ordinamento dei lavori :**

- **SPT (Shortest Processing Time)** : **min** d_i tra i lavori non ancora assegnati
  - i task sono assegnati in **ordine non decrescente dei tempi di esecuzione** (prima più "corti")
- **LPT (Longest Processing Time)** : **max** d_i tra i lavori non ancora assegnati
  - i task sono assegnati in **ordine non crescente dei tempi di esecuzione** (prima più "lunghi")
  - In generale LPT produce soluzioni di migliore qualità.

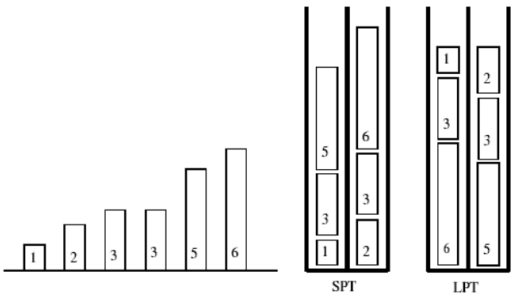
N° macchine : $m = 2$

Durate dei lavori : 1 2 3 3 5 6

SPT : durata su macchina 1 : 1 + 3 + 5 = 9
durata su macchina 2 : 2 + 3 + 6 = 11

LPT : durata su macchina 1 : 6 + 3 + 1 = 10
durata su macchina 2 : 5 + 3 + 2 = 10

SOLUZIONE OTTIMA



↑
durate dei lavori

LPT fornisce una soluzione ottima

In SPT la funz obiettivo =11, in LPT=10. La soluzione **in LPT è ottima** perché le **macchine** hanno lo **stesso tempo di esecuzione.**

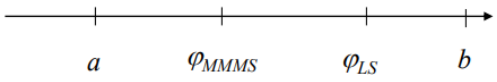
8.2.1.1 Valutazione dell'errore relativo delle euristiche List Scheduling

$$ErrRel_{LS} = \frac{\varphi_{LS} - \varphi_{MMMS}}{\varphi_{MMMS}}$$

- $\boldsymbol{\varphi_{LS}}$ = **errore euristica** di list scheduling
- $\boldsymbol{\varphi_{MMMS}}$ = valore dell'**ottimo**
- Vale sempre che: $\boldsymbol{\varphi_{MMMS} \leq \varphi_{LS}}$

Devo trovare un **minorante del denominatore** $\boldsymbol{\varphi_{MMMS}}$ e un **maggiorante del numeratore** $\boldsymbol{\varphi_{LS} - \varphi_{MMMS}}$, così da ottenere un **maggiorante della frazione** $\boldsymbol{ErrRel_{LS}}$. Chiamo **a,b** i **valori da determinare**, tale che:

$$\boldsymbol{a < \varphi_{MMMS} \ \wedge \ b \geq \varphi_{LS}}$$



a: è un valore che **trovo sotto l'ottimo**, quindi solo **quando rilasso un vincolo** e ottengo una **soluzione inamissibile** per il **problema originario**. Il vincolo che rilasso è il **vincolo indivisibilità dei lavori**.

$$\boldsymbol{a = \frac{\sum_{i=1}^{n} d_i}{m}}$$

è il valore della soluzione non ammissibile per MMMS che si ottiene rilassando il vincolo. Così do ad **ogni macchina** una **stessa frazione** di **tempo da lavorare.**

b: cerco **soluzione** con **tempo di completamento** $\boldsymbol{b \geq \varphi_{LS}}$**.** È il più alto (ossia il **peggiore**) valore che si può ottenere con l'euristica.
Valore peggiore: Lo si ottiene quando nel momento dell'assegnamento dell ultimo lavoro (di durata d_h) le **macchine** sono **perfettamente bilanciate**, ossia tutte utilizzate per il tempo:

$$\boldsymbol{da\ cui:\ \ b = \frac{\sum_{i=1}^{n} d_i - d_h}{m} + d_h}$$

ossia la casistica di **macchine perfettamente bilanciate + ultimo lavoro(d_h)**

Ora ho tutti i membri **a,b noti** , posso **riscrivere l'espressione dell'errore:**

$$ErrRel_{LS} = \frac{\varphi_{LS} - \varphi_{MMMS}}{\varphi_{MMMS}} \leq \frac{b-a}{a} = \frac{\cancel{\frac{\sum_{i=1}^{n} d_i}{m}} + \frac{m-1}{\cancel{m}} d_h - \cancel{\frac{\sum_{i=1}^{n} d_i}{m}}}{\frac{\sum_{i=1}^{n} d_i}{\cancel{m}}} = \frac{m-1}{\sum_{i=1}^{n} d_i} d_h$$

È una funzione in **che dipende solo dai dati in ingresso**, ossia **solo da *m* e d_h**.

9 WVC: Weighted Vertex Cover (problema di copertura per nodi pesati)

Rete di telecomunicazione rappresentata da un **grafo non orientato**(per questo weighted) $G = (N, A)$, in cui ad **ogni nodo è associato un peso**
Gli **unici dati del problema sono i pesi dei nodi.**

- **collegamenti** (insieme A): **linee di trasmissione** (flussi di informazioni)
- **nodi** (insieme N): **punti di interconnessione delle linee**

Il **monitoraggio** delle informazioni avviene mediante **dispositivi installati nei nodi**: il dispositivo installato nel nodo i permette di controllare le comunicazioni trasmesse sulle linee incidenti nel nodo i.
Il **costo di installazione** del dispositivo **nel nodo i è positivo ($c_i > 0$).**

Determinare in quali nodi installare il dispositivo di controllo affinché

- tutte le **linee siano sotto controllo**
- sia **minimizzato** il **costo complessivo** dei **dispositivi installati**

Formulazione matematica del problema di Weighted Vertex Cover

$$\min \quad \sum_{i \in N} c_i \, x_i$$

$$\text{s.a} \quad x_i + x_j \geq 1 \qquad (i,j) \in A$$

$$x_i \in \{0,1\} \qquad i \in N$$

variabili decisionali binarie:

- $x_i = 1$, se il dispositivo di controllo **è installato** nel nodo i
- $x_i = 0$, se **non è installato**

Funzione obiettivo:

- **Minimizzare** la **somma** dei **costi** dei **installazione dispositivi.**

Vincolo di copertura degli archi:

- le informazioni che transitano sul **collegamento $(i,j) \in A$** sono **sotto controllo se** il **dispositivo** è **installato in almeno uno dei nodi i e j.**

Il rilassamento continuo di questa formulazione richiede i **vincoli di nonnegatività** $x_j \geq 0$ e i **vincoli di upper bound** $x_j \leq 1$ (il **duale è definito per problemi con variabili continue**)

9.1 Algoritmo approssimato per WVC (formulazione a variabili intere)

Formulazione equivalente: **da variabili binarie a variabili intere nonnegative**

$$\min \quad \sum_{i \in N} c_i \, x_i$$

$$\text{s.a} \quad x_i + x_j \geq 1 \qquad (i,j) \in A \qquad \text{modello } \boldsymbol{WVC}$$

$$x_i \geq 0 \text{ e intero} \qquad i \in N$$

Questa formulazione ha valori interi di x_i maggiori di 1 ammessi dai vincoli funzionali di disuguaglianza $\geq$. Possiamo interpretare x_i come il **numero di dispositivi di controllo installati nel nodo i**: osserviamo che nella **soluzioni ottima** gli **unici valori che x_i** può **assumere** sono

- $x_i = 0$, per indicare che il dispositivo di controllo **non è installato** nel nodo i
- $x_i = 1$, per indicare che il dispositivo di controllo **è installato** nel nodo i (infatti installare nel nodo i un numero di dispositivi maggiore di 1 costa di più, ma per il controllo viene esercitato mediante un dispositivo)

La **soluzione ottima** sarà la stessa perché voglio **minimizzarli**, quindi **x_i sarà sempre comunque 0 o 1**

Quando faccio il **rilassamento continuo** ho solo i **vincolo di nonnegatività** e **non** più quelli di **upper bound**, perché x_i è una **variabile positiva** e intera **non superiormente limitata**.

Quindi ottengo solo il vincolo: $x_i \geq 0$ senza dover specificare il "intero". Quindi il **modello** appena introdotto è **più efficiente per passare al duale** rispetto al precedente.

Il **rilassamento continuo** ***WVC*** di ***WVC*** è

$$\min \quad \varphi = \sum_{i \in N} c_i \, x_i$$

$$\text{s.a} \quad x_i + x_j \geq 1 \qquad (i,j) \in A \qquad y_{i,j}$$

$$x_i \geq 0 \qquad i \in N$$

Lo ottengo:

1) **cancellando il requisiti di integralità**, ossia il "**e intere**" **del x_i.**
2) Definisco **tante variabili duali quanti** sono i **vincoli di copertura degli archi**, le chiamo $y_{i,j}$

Ottengo il **duale di *WVC*** considerando la **formulazione matriciale del rilassamento di base**

$$\min \varphi = \boldsymbol{c}^T \boldsymbol{x}$$

$$\text{s.a} \quad \boldsymbol{E}^T \boldsymbol{x} \geq \boldsymbol{1} \qquad \text{vettore delle variabili duali: } \boldsymbol{y} \in \mathbb{R}^{|A|}$$

$$\boldsymbol{x} \geq \boldsymbol{0}$$

La matrice $\boldsymbol{E}^T$ **dei coefficienti** dei **vincoli funzionali** è la **trasposta** della **matrice di incidenza** $\boldsymbol{E}$ del grafo non orientato G. Matrice che ha **sugli assi nodi e archi, ogni cella ha 1 o -1 se** c'è un **arco che passa.**
Su di essa avrò tutti **elementi nulli sulla riga** (perché trasposto, altrimenti colonna) , **tranne due elementi** che **sono gli elementi sui nodi in cui l'arco incide e leggerò**:

- (-1, 1) se il grafo è **orientato.** Nel nostro caso il grado è orientato.
- (1, 1) se il grafo è **non orientato.**

In **forma scalare** ottengo il **modello duale**:

$$\max \omega_D = \sum_{(i,j) \in A} y_{i,j}$$

$$\text{s.a} \sum_{(i,j) \in S_j} y_{i,j} \leq c_i \qquad i \in N$$

$$y_{i,j} \geq 0 \qquad (i,j) \in A$$

S_j: **stella del nodo j**, ossia insieme degli archi incidenti nel nodo i

9.1.1 Costruzione di un algoritmo approssimato con determinazione dell' upper bound dell'errore relativo

Processo iterativo che **costruisce**:

- una **soluzione ammissibile di *WVC***
  - per ogni ***i*** vale $x_i = 0$ *o* $x_i = 1$
  - Per ogni **arco** sono soddisfatti i **vincoli di copertura degli archi** $x_i + x_j \geq 1 \quad (i,j) \in A$
- una **soluzione ammissibile del duale del rilassamento continuo** di *WVC*

$$y_{i,j} \geq 0, \qquad (i,j) \in A, \qquad c_i \geq \sum_{(i,j) \in S_j} y_{i,j} \qquad i \in N$$

(questa soluzione viene costruita perché il valore obiettivo della duale è un minorante del valore della funzione obiettivo del problema di base)

  - dove le **variabili duali** $y_{i,j}$ sono **determinate** mediante **equazioni** derivanti dalle **condizioni di complementarietà ai nodi** della coppia primale-duale ***WVC*** e ***DualeWVC***, ossia se x_j **= 1** allora **il valore** dell'**obiettivo duale**:

$$c_i - \sum_{(i,j) \in S_j} y_{i,j} = 0$$

Il **valore dell'obiettivo** associato ad una **soluzione duale ammissibile** è **lower bound** del **costo minimo** di **copertura degli archi**

Inizialmente:

- x_j **= 1** e $y_{i,j}$**=0, variabili primali** x_j **, duali** $y_{i,j}$ **nulle**
- l'insieme degli **archi non sottocontrollo (Q)** contiene tutti gli archi
- l'insieme dei **nodi scelti (S) vuoto.**

iterazione:

- seleziona un **qualsiasi arco** (i,j) **non ancora coperto**
- si assegna alla corrispondente variabile $y_{i,j}$ **il massimo valore possibile** che **soddisfa** le **condizioni** degli **scarti complementari** ai nodi i e j: per tale valore di $y_{i,j}$ **soddisfatta almeno una delle due condizioni**

$$c_i = \sum_{(i,h) \in S(i)} y_{i,h} \geq 0\,, \qquad c_j = \sum_{(j,h) \in S(i)} y_{j,h} \geq 0$$

  - Se **vale la prima condizione**, ***i*** viene **aggiunto ad *S*** e tutti gli **archi incidenti in *i*** sono **eliminati da *Q***
  - se **vale la seconda condizione**, ***j*** viene **aggiunto ad *S*** e tutti gli **archi incidenti in *j* sono eliminati da *Q*.**

La **soluzione** $y_{i,j}$, $(i,j) \in A$, **determinata dall'algoritmo è duale-ammissibile** :

- lo è **sicuramente all'inizio** (dato che i costi sono positivi) ;
- quando il **vincolo duale corrispondente** al **nodo** i **diviene "attivo"** tutti gli archi incidenti in i vengono eliminati da Q, "congelando" i loro valori yi,j fino al termine dell'algoritmo e quindi assicurando che il vincolo resti verificato.

Terminazione:

- A terminazione, la **soluzione primale** S **copre tutti gli archi** ($Q = \emptyset$) ed è **quindi primale-ammissibile**

Inizializzazione

$x_1 = 0$	$z_1 = 1 - y_{1,2} - y_{1,5} - y_{1,6} = 1$
$x_2 = 0$	$z_2 = 3 - y_{1,2} - y_{2,3} - y_{2,6} = 3$
$x_3 = 0$	$z_3 = 5 - y_{2,3} - y_{3,4} - y_{3,6} = 5$
$x_4 = 0$	$z_4 = 4 - y_{3,4} - y_{4,5} - y_{4,6} = 4$
$x_5 = 0$	$z_5 = 2 - y_{4,5} - y_{1,5} - y_{5,6} = 2$
$x_6 = 0$	$z_6 = 2 - y_{1,6} - y_{2,6} - y_{3,6} - y_{4,6} - y_{5,6} = 2$
$s_{1,2} = x_1 + x_2 - 1 = -1$	$y_{1,2} = 0$
$s_{2,3} = x_2 + x_3 - 1 = -1$	$y_{2,3} = 0$
$s_{3,4} = x_3 + x_4 - 1 = -1$	$y_{3,4} = 0$
$s_{4,5} = x_4 + x_5 - 1 = -1$	$y_{4,5} = 0$
$s_{5,1} = x_5 + x_1 - 1 = -1$	$y_{5,1} = 0$
$s_{1,6} = x_1 + x_6 - 1 = -1$	$y_{1,6} = 0$
$s_{2,6} = x_2 + x_6 - 1 = -1$	$y_{2,6} = 0$
$s_{3,6} = x_3 + x_6 - 1 = -1$	$y_{3,6} = 0$
$s_{4,6} = x_4 + x_6 - 1 = -1$	$y_{4,6} = 0$
$s_{5,6} = x_5 + x_6 - 1 = -1$	$y_{5,6} = 0$

ogni riga ho la primale e la duale, ottenuta come variabile scarto(z). si vede che inizialmente le primali e duali sono nulle. Ottengo valori positivi per le scarto primali e negativi per le duali. Così ho inamissibilità primale e ammissibilità duale.

1° iterazione: scelgo un arco da controllare, prendo (1,2):
L'arco 1, 2 è controllato se almeno una delle variabili x_1 o x_2 assume valore 1:

- x_1=1 allora $z_1 = 1 - y_{1,2} - y_{1,5} - y_{1,6} = 0$ da cui: $y_{1,2} = 1$, inoltre $z_1 \geq 0$. Ciò vuol dire che ho messo il dispositivo su '1', cosa devo fare perché continui a valere la **condizione di complementarietà**, dato che se $x_1 \neq 0$ *allora* $z_1 = 0$, perché sono moltiplicati. infatti in questo caso $x_1 \neq 0$, quindi $z_1 = 0$, per farlo diventare 0 trovo i valori corretti delle varie $y_{i,j}$.
- x_2=1 mi da $y_{1,2} = 3$ ma $z_1 < 0$

Aggiorno i valori: $y_{1,2} = 1$, $x_1 = 1$
da cui si ottengono i valori aggiornati delle variabili scarto $z1$, $z2$, $s1,2$, $s5,1$, $s1,6$: che mostrano che le condizioni di complementarietà sono tutte soddisfatte

10 Selezione di sottoinsiemi

Suppongo di avere:

- Un **insieme** finito di **elementi** $E = \{1, 2, \dots, m\}$
- Una famiglia di n=2^m **sottoinsiemi** di E detto $F = \{F1, F2, \dots, Fn\}$, con $F_j \subseteq E$, $j = 1, \dots, n$

a ciascun sottoinsieme F_j è associato un **costo** c_j , $1 \leq j \leq n$

Obiettivo: **determinare una sottofamiglia** D, $D \subseteq F$ che **soddisfi** i **vincoli** che sia di **costo minimo**

per individuare una sottofamiglia D definisco n **variabili binarie** $x_1, \dots, x_n$ tali che:

- $x_j = 1$ se si sceglie di **inserire** F_j **in** D
- $x_j = 0$ se **non** si sceglie di **inserire** F_j **in** D

Il **costo** della **scelta della sottofamiglia** D è:

$$\sum_{j=1}^{n} c_j \, x_j$$

Si possono individuare **3 diversi problemi di selezione dei sottoinsiemi**:

- **Problema di copertura**
- **Problema di partizione**
- **Problema di riempimento**

Rappresentazione della famiglia F di sottoinsiemi di E

La famiglia F viene rappresentata mediante una **matrice**

$$A = [a_{i,j}] \qquad 1 \leq i \leq m, \quad 1 \leq j \leq n$$

con

- $a_{i,j} = 1$ se $i \in Fj$
- $a_{i,j} = 0$ altrimenti

10.1 Problema di copertura (PC)

La **sottofamiglia** D deve essere **scelta** in modo che **ogni elemento di** E **appaia in almeno** un $F_j \in D$, ossia **ogni elemento di** E deve essere **selezionato almeno una volta.**
Si dice che una sottofamiglia D con tale proprietà "**copre**" l'insieme degli elementi di E.

Obiettivo: trovo **copertura** D di E mediante sottoinsiemi scelti da F che **minimizza costo totale copertura**

Formulazione PLI del problema di copertura (PC)

$$\min \sum_{j=1}^{n} c_j x_j$$

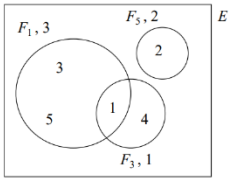
$$\sum_{j=1}^{n} a_{i,j} x_j \geq 1 \qquad i = 1, \dots, m$$

$$x_j \in \{0, 1\} \qquad j = 1, \dots, n$$

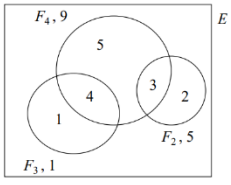
esempio:

$$F = \{\underbrace{\{1,3,5\}}_{F_1}, \underbrace{\{2,3\}}_{F_2}, \underbrace{\{1,4\}}_{F_3}, \underbrace{\{3,4,5\}}_{F_4}, \underbrace{\{2\}}_{F_5}, \underbrace{\{5\}}_{F_6}, \underbrace{\{1,5\}}_{F_7}\}$$

$$c(F) = [3 \;\; 5 \;\; 1 \;\; 9 \;\; 2 \;\; 4 \;\; 1]^T$$



costo della copertura F_1, F_3, F_5 di E : 6

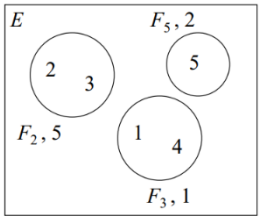


costo della copertura F_2, F_3, F_4 di E : 15

10.2 Problema di Partizione (PP)

La **sottofamiglia D** deve essere scelta in modo che ogni elemento di **E** appaia in **uno ed un solo** $F_j \in D$,
ogni elemento di E deve essere selezionato **esattamente una volta**
Si dice che una sottofamiglia D con tale proprietà "**partiziona**" l'insieme degli elementi di E.

Esempio:



$$F = \{\underbrace{\{1,3,5\}}_{F_1}, \underbrace{\{2,3\}}_{F_2}, \underbrace{\{1,4\}}_{F_3}, \underbrace{\{3,4,5\}}_{F_4}, \underbrace{\{2\}}_{F_5}, \underbrace{\{5\}}_{F_6}, \underbrace{\{1,5\}}_{F_7}\}$$

$$\underline{c}(F) = [3 \quad 5 \quad 1 \quad 9 \quad 2 \quad 4 \quad 1]^T$$

costo della partizione F_2, F_3, F_5 di E : 8

43

Obiettivo: trovo **partizione** D di E mediante sottoinsiemi scelti da F che **minimizza costo totale partizione**

il **vincolo di partizione** è:

$$\sum_{j=1}^{n} a_{i,j} x_j = 1 \qquad i = 1, \ldots, m$$

ossia, per ogni $i \in$ E esiste un solo indice j per il quale è contemporaneamente $\boldsymbol{a_{i,j}} = 1$ e $\boldsymbol{x_j} = 1$
ossia in D esiste **esattamente un** sottoinsieme F_j che contiene l'elemento *i-esimo*.

Formulazione PLI del problema di partizione (PP)

$$\min \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_{i,j} x_j = 1 \qquad i = 1, \ldots, m$$

$$x_j \in \{0, 1\} \qquad j = 1, \ldots, n$$

10.3 Problema di riempimento (PR)

La sottofamiglia D deve essere scelta in modo che ogni elemento di E appaia in **al più un** $F_j \in D$,
ogni elemento di E deve essere selezionato **non più di una volta.**

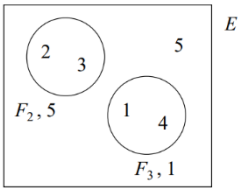
- qualche elemento di E può non essere presente
- nessun elemento di E può essere ripetuto

Si dice che una sottofamiglia D con tale proprietà "**riempie**" l'insieme degli elementi di E.

Nei problemi di riempimento il coefficiente c_j associato al sottoinsieme F_j rappresenta un valore (anziché un costo):

obiettivo: è determinare il **riempimento** D di E mediante sottoinsiemi scelti da F che **massimizza** il **valore totale del riempimento**

esempio:



valore del riempimento F_2, F_3 di E : 8

il **vincolo di riempimento** è:

$$\sum_{j=1}^{n} a_{i,j} x_j \leq 1 \qquad i = 1, \ldots, m$$

ossia, per ogni i ∈ E esiste al più un indice j per il quale è contemporaneamente $a_{i,j} = 1$ e $x_j = 1$ ossia in D esiste **al più** un sottoinsieme Fj che contiene l'elemento i-esimo.

Il problema di riempimento è quindi

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_{i,j} x_j \leq 1 \qquad i = 1, \ldots, m$$

$$x_j \in \{0, 1\} \qquad j = 1, \ldots, n$$

11 Lettura output GAMS

L'esecuzione del file .gms genera automaticamente due file:

- file di **log**
  - contiene al suo interno tutte le **informazioni** di natura **computazionale** legate alla **risoluzione** del **modello** di ottimizzazione.
  - ***Status: normal completion*** = definire lo **stato del risolutore**, se ha restituito soluzione **senza** imbattersi in **errori computazionali**
    - Non mi basta come garanzia che il codice sia corretto, mi serve anche ***optimal***.
  - ***LP status (1) : optimal*** = riporta lo **stato del modello**, il modello è stato risolto fino a raggiungere **l'ottimo** e inoltre **non** ci sono **errori modellistici**

- file di **listing** (detto anche **report della soluzione**)
  - contiene le **info** relative alla **risoluzione del problema**
  - è suddiviso **in 3 sezioni**:
    - ***input***: da Compilation fino a Model Statistics

      - dentro ***Equations*** sono riportate le **equazioni** che **compongono** il **modello**, sono scritte nel seguente modo:
        - a **sx** dell **uguale** si ha: **variabili decisionali** con i relativi coefficienti moltiplicativi
        - a **dx** dell **uguale** si ha: specificati i **termini noti.**

      - Dentro ***Column*** è contenuto un listing delle variabili decisionali introdotte tramite la keyword ***Variables***

      - Dentro ***Model statistics*** sono contenute le info di: **numero di variabili** e **vincoli** contenuti nel modello

    - ***Output***: da solution report fino a SolVAR

      - All interno della voce **Solution Report** è presente una **sintesi del modello**:
        - **Nome** modello
        - **tipologia** (LP, MIP)
        - Risolutore(CPLEX)

- quale è la **funzione obiettivo** (specificata da *z*)
- direazione di ottimizzazione (minimize o maximize)
- **stato** del risolutore (normal completions)
- **stato** del modello (optimal)
- **valore della fuznione obiettivo**
- **tempo richiesto** per trovare soluzione ottima
- quante **iterazione** richieste

- All interno di **SolVAR** è presente **valore ottimo** delle **variabili decisionali**
  - Il pallino "°" indica lo 0
  - Colonna **LOWER** contiene il valore del **lower bound** per la variabile decisionale
  - Colonna **LEVEL** contiene il valore **ottimo** che GAMS attribuisce alla variabile decisionale
  - Colonna **UPPER** contiene il valore dell **upper bound** per la var dec
  - Colonna **MARGINAL** contiene il valore del **coefficiente di costo relativo** della variabile decisionale.
    - Per tutte le **VB vale 0**
    - Per le **VNB** è **diverso da zero**

    - Inoltre per le **variabili decisionali** dice quanto si è **distanti** dalla **soglia** di **profitto** che permette l'**attivazione della variabile decisionale**, ossia quanto il costo unitario dovrebbe aumentare/diminuire per **rendere** la **VNB** una **VB**
      - se il **valore** di **MARGINAL** per una variabile decisionale **è 0**, indica che la variabile decisionale è **strettamente positiva**, il suo profitto unitario è già abbastanza elevato. **Conviene produrre quel prodotto.**
      - se il **valore** di **MARGINAL** per una variabile decisionale è **diverso da 0**, indica di quanto **dovrebbe aumentare** il **profitto unitario** per renderla economicamente **conveniente** la **produzione**.

- all interno della voce **SolEQU** è presente il **valore dei vincoli.**
  - Ha tante sottosezioni quanti vincoli che compongono il problema
  - Ha la stessa struttura di SolVAR
  - La colonna **MARGINAL** contiene i **coefficienti di costo relativo** delle **variabili slack** associate al vincolo.
  - Il suo significato è quello di rappresentare il **prezzo ombra** della risorsa, ossia mi dice di **quanto aumenterebbe la funzione obietivo se potessi aumentare di una unità il valore del termine noto del vincolo:**
    - **MARGINAL=0:** la **risorsa è in eccesso**, se ne avessi un unità in più, la soluzione ottima non cambia
    - **MARGINAL≠0**: la **risorsa** è **scarsa**, con una risorsa extra posso aumentare la produzione

- ***Display***: da Execution a display
  - Sono presenti tutte le **info** richieste tramite la **keyword *Display***

11.1 Fase di risoluzione

La **fase di costruzione** prevede:

- Identificazione degli **insiemi**: uso le keyword ***sets***
- Identificazione dei **dati di riferimento**: i dati in input possono essere:
  - **Scalari**, uso ***Scalars***
  - **Vettoriali**, uso ***parameters***
  - **Matriciali**, uso ***table***

La **fase di modellazione** prevede:

- Costruzione delle **variabili decisionali**, uso ***variables***
- Definizione dei **vincoli**, uso ***equations***
- Costruzione del **modello**, uso ***model***

La **fase di risoluzione** prevede:

- Obiettivo di **ottimizzazione** (max/min) variabile obietivo, Uso ***solve***.

ALCUNE DOMANDE TIPICHE ESAME ORALE MAO

Parla del problema di ordinamento di lavori su macchine con riferimento al problema di minimizzazione del numero di macchine utilizzate. Parla sia del modello sia dell'algoritmo risolutivo.

Con riferimento al problema di flusso di costo minimo parla del rispettivo algoritmo risolutivo; non mi interessa la costruzione del modello, ma solo conoscere le fasi dell'algoritmo risolutivo.

Parla del problema dello zaino a livello di modello e poi di algoritmo risolutivo.

Parla dell'algoritmo risolutivo del Min Cost Flow.

Formula, descrivi e scrivi il modello matematico del problema del commesso viaggiatore senza entrare nei dettagli dell'algoritmo risolutivo.

Descrivi il problema e gli algoritmi risolutivi per il problema dell'identificazione dell'albero di supporto di costo minimo.

GAMS: spiega come è strutturato l'output fornito dalla computazione.

Relativamente alla parte di modellazione, formula il problema decisionale (solo il modello matematico senza entrare nei dettagli dell'algoritmo risolutivo) per il problema di massimo flusso.

Relativamente alla parte algoritmica, parla del problema dello zaino.

Descrivi il problema di raggiungibilità nei grafi e il relativo algoritmo risolutivo.

Relativamente alla parte algoritmica, descrivi gli algoritmi risolutivi per il problema di cammini di costo minimo.

Relativamente alla parte di modellazione, descrivi il modello matematico relativo al problema di flusso di costo minimo.

Presenta le varie tipologie di modelli che possono essere utilizzati per affrontare il problema di selezione di sottoinsiemi.

SCHEMINO

- **problema della raggiungibilità**: G or, s
  - **algoritmo di visita**
    - m,q, pila, coda, etichetta
- **Alberi di supporto di costo minimo**: G non or, **T***, e, ce
  - **Metodo di Kruskal**
    - Greedy, ordino e scelgo, connessione
  - **Metodo di Prim**
    - Sempre connesso, stella nodo, S, T* , $\delta(S)$, soluz alternative
- **Cammini di costo minimo:** G or, cij costo, s,t
  - **Disktra**:
    - Etichetta, S, N\S, taglio + etich, greedy
  - **Etichette**
    - Ord topologico, etichette + arco
  - **Floyd-warshal**
    - Costo neg, inf lim, C con casistiche, D, P, h-esimo
- **Massimo flusso:** R, G or, uij, s,t
  - PL: sorg, pozzo, interm
  - **Ford-fulk**: rete incrementale, cammino aumentante, archi non saturi, scarichi
- **Flusso di Costo Minimo (min cost flow):** R, G or, uij, cij, bi
  - PL: bilancio sui nodi
  - **Algoritmo ad eliminazione dei cicli di costo negativo**. 7 passi:
    - Equil, rete aumentata, ford-fulk, flux max, ottimo con rete increm, redistrib, eliminaz cicli costo neg.
- **Il problema dello zaino:** pj, vj
  - **Max, min**
  - **B&B**, rilassamento, unità di peso, duale x ottimalità
- **Algoritmo di Branch-and-Bound per *K*P**
  - Varie casistiche frazionarie, prendo xj=0 e xj=1
- **TSP: Il problema del commesso viaggiatore**: G or e non or, cij pesi, ciclo hamiltoniano
  - **TSP simm**
  - **TSP asimm**
  - **Algoritmi greedy per TSP**

        - algoritmo "**Nearest Neighbour**"
        - con **ordinamento degli archi in ordine non decrescente di costo**
            - **local serach, 2 scambio**
- **TPS metrico:** G non or, cij costi, completo, casistiche dgli cij ><=
    - **taMST: 3 fasi**
        - T*, ciclo orientato, ciclo hamiltoniano
        - Osservazioni su costo c(h)
            - **Err tamst**
- **Algoritmo di Branch-and-Bound per *TSP* simmetrico**
    - **1**-albero di costo minimo
- **MCMS - Minimal Cardinality Machine Scheduling:** n,m,ti,di
    - Matematico: i,j, xij,
        - Matrice incompatbilità
    - **Algoritmi greedy per MCMS**
        - Soluzione ottima con ordinamento ti non crescente
- **MMMS - Minimal Makespan Machine Scheduling**: m,n, di
    - Tj, minimozzo t
    - **Algoritmo greedy di list scheduling**, a macchina "più scarica",
        - SPT, LPT
        - errore relativo delle euristiche List Scheduling
- **WVC: Weighted Vertex Cover**: G non or, peso
    - Linee sottocontrollo + minimo peso installazione
    - **Algoritmo approssimato per WVC**
        - Normale, rilassato, ancora più rilassato
        - Forma matriciale poi duale
    - **upper bound dell'errore relativo**
        - Q per archi, S per nodi,
        - $\boldsymbol{yi,j}$ il massimo valore possibile degli scarti complementari soddifstti
        - Duale ammiss, primale ammiss
- **Selezione di sottoinsiemi:** E, F, D, xi, matrice A
    - **Problema di copertura (PC):** D scelta tc ogni elemento di E appaia in almeno un Fj
        - ogni elemento di E deve essere selezionato almeno una volta
    - **Problema di Partizione (PP):**
        - ogni elemento di E deve essere selezionato esattamente una volta
    - **Problema di riempimento (PR)**
        - cj rappresenta un valore
        - Massimizza
        - ogni elemento di E deve essere selezionato non più di una volta.
- **Lettura output GAMS**
    - **Log**
        - Normal completion, LP status (1) : optimal
    - **Listiing**
        - Input
            - Equation, columns, model statistics
        - Output
            - Solution report
            - **solvar**: lower, level, upper, marginal (coefficiente di costo relativo)
            - **solequ**: lower, level, upper, marginal (coefficiente di costo relativo)
                - tante sottosezioni quanti vincoli
                - MARGINAL=0: la risorsa è in eccesso
        - display