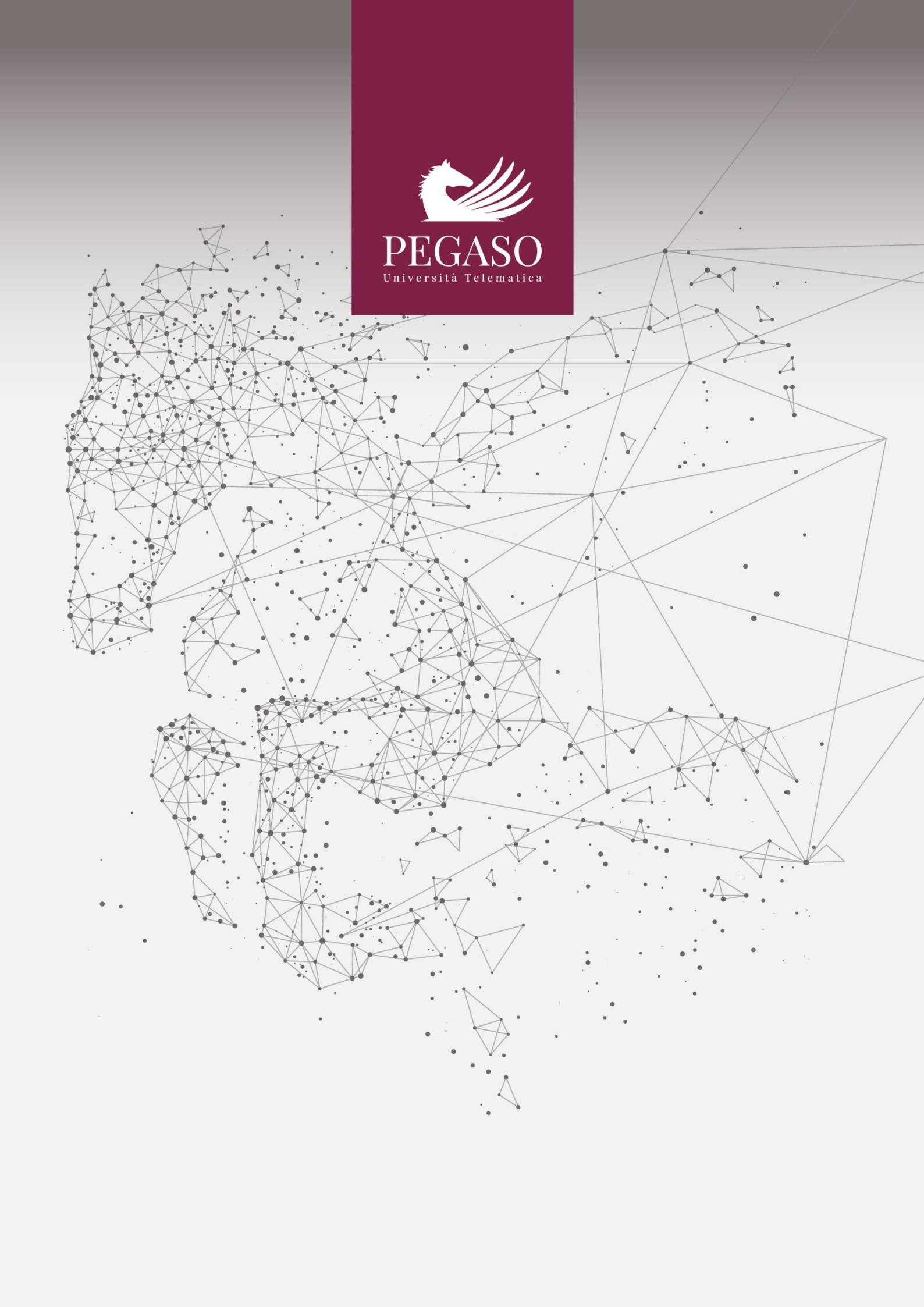
PEGASO
Università Telematica

Indice

1. Problema dei cammini minimi

Consideriamo un Grafo Orientato $G = (V, E)$ con un nodo sorgente s ed una funzione di peso $w : E \rightarrow R$

Dato un cammino $p = (v_1, v_2, \ldots, v_k)$ con $k > 1$, il costo del cammino è dato da:

$$w(p) = \sum_{i=2}^{k} w(v_{i-1}, v_i)$$

Il problema del **cammino minimo** è quello di trovare un cammino da s ad u, per ogni nodo $u \in V$, il cui costo sia minimo, ovvero più piccolo o uguale del costo di qualunque altro cammino da s ad u.

Quando parliamo di "pesi" possiamo far riferimento a:

- positivi / positivi + negativi
- reali / interi

ATTENZIONE: non tutti i tipi di algoritmi funzionano per tutte le categorie di pesi.

Sebbene possa sembrare strano, vi sono casi in cui ha senso considerare pesi negativi, tuttavia occorre fare attenzione poiché lo scenario in cui si ha un ciclo in cui sono presenti dei pesi negativi finirebbe per portare il peso a $-\infty$ e quindi avrebbe ben poco senso.

Consideriamo due cammini minimi possiedono un tratto in comune:

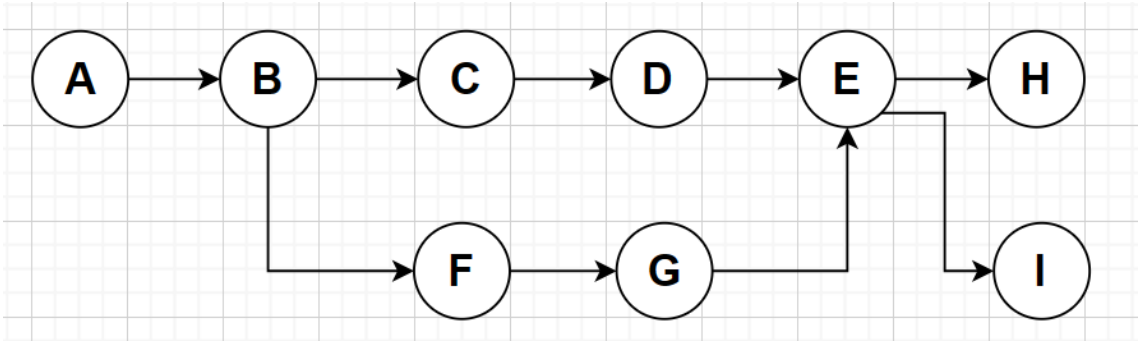

Figura 1 - Cammini minimi - tratto in comune

Abbiamo il cammino minimo (A, B, C, D, E, H) e l'altro (A, B, C, D, E, I);

I due cammini minimi possono avere un tratto in comune (quello da A ad E) ma non possono convergere in un nodo comune E dopo aver percorso un tratto distinto, cioè non è possibile che un cammino minimo passi per (B, C, D, E) ed un altro per (B, F, G, E).

L'**albero dei cammini minimi** è un albero di copertura radicato in s avente un cammino da s a tutti i nodi raggiungibili da s.

Dato un grafo $G = (V, E)$ non orientato e connesso, definiamo come **Albero di Copertura** (**Spanning Tree**) un albero di copertura di G è un sottografo $T = (V, E_T)$ tale che:

- T è un albero
- $E_T \subseteq E$
- T contiene tutti i vertici di G

Cioè in sostanza, T è un albero i cui vertici sono i vertiti di G ma gli archi sono un sottoinsieme degli archi del grafo di partenza.

Consideriamo il seguente grafo:

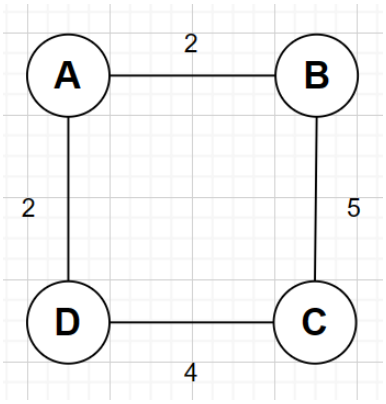

Figura 2 - Grafo

Costruiamo un vettore di distanza d i cui valori $d[x]$ rappresentano la distanza del nodo x dalla sorgente s.

Scegliamo come sorgente s il nodo A; immagino di spostarmi da A a B, poi da B a C e poi da C a D; in questo modo ho "coperto" tutti i nodi dell'albero:

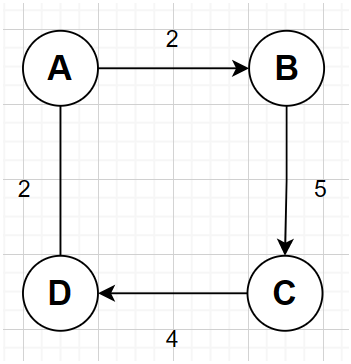

Figura 3 - Primo albero di copertura

In questo primo albero di copertura abbiamo:

- $d[A] = 0$
- $d[B] = d[A] + w(A, B) = 2$
- $d[C] = d[B] + w(B, C) = 7$
- $d[D] = d[C] + w(C, D) = 11$

Essendo $w(x, y)$ il peso dell'arco (x, y)

Tuttavia, potremmo anche avere il seguente scenario in cui mi sposto da A a D, da A a B e da B a C; in questo modo ho "coperto" tutti i nodi dell'albero:

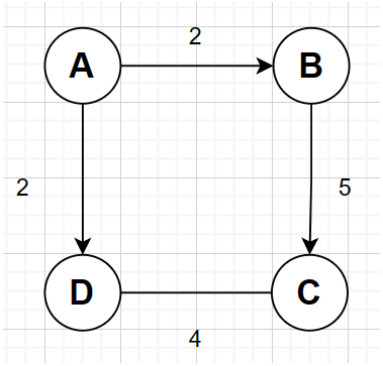

Figura 4 - Secondo albero di copertura

In questo secondo albero di copertura abbiamo:

- $d[A] = 0$
- $d[B] = d[A] + w(A, B) = 2$
- $d[C] = d[B] + w(B, C) = 7$
- $d[D] = w(A, D) = 2$

Come individuare una soluzione ottima?

2. Teorema di Bellman

Una soluzione ammissibile T è ottima se e solo se:

- $d[v] = d[u] + w(u,v)$ per ogni arco $(u,v) \in T$
- $d[v] \leq d[u] + w(u,v)$ per ogni arco $(u,v) \in E$

Se consideriamo il primo albero di copertura:

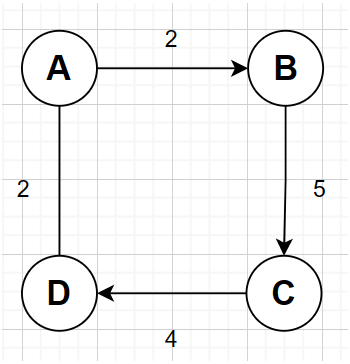

Figura 5 - Primo albero di copertura

In questo primo albero di copertura abbiamo:

- $d[A] = 0$
- $d[B] = d[A] + w(A,B) = 2$
- $d[C] = d[B] + w(B,C) = 7$
- $d[D] = d[C] + w(C,D) = 11$

Tuttavia, esiste anche un arco (A, D) e se calcoliamo $d[D] = w(A,D) = 2$

Quindi in sostanza abbiamo che:

$$d[C] + w(C,D) \geq w(A,D)$$

e quindi non viene rispettato il teorema di Bellman e quindi la soluzione non è ottima.

Consideriamo il secondo scenario:

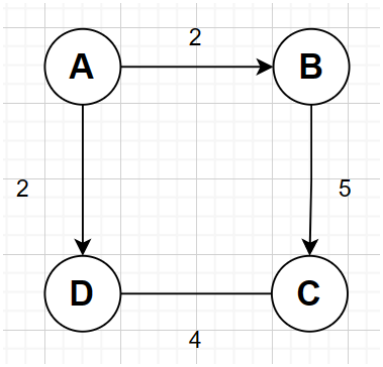

Figura 6 - Secondo albero di copertura

In questo secondo albero di copertura abbiamo:

- $d[A] = 0$
- $d[B] = d[A] + w(A, B) = 2$
- $d[C] = d[B] + w(B, C) = 7$
- $d[D] = w(A, D) = 2$

Se provassimo a raggiungere D passando per A, B e C avremmo:

$$d[C] + w(C, D) \geq w(A, D)$$

Viene dunque rispettata la condizione di Bellman e quindi la soluzione è ottima.

Costruiamo ora l'algoritmo per il calcolo dei cammini minimi. Costruiamo innanzitutto un "prototipo" di algoritmo, molto generico, che poi andiamo ad adattare alle varie circostanze.

L'idea è la seguente:

Inizializza T ad una foresta di copertura composta da nodi isolati
Inizializza d con sovrastima della distanza ($d[s] = 0, d[x] = +\infty$)
while ∃(u, v) : d[u] + G.w(u, v) < d[v]
d[v] = d[u] + G.w(u, v)
% Sostituisci il padre di v in T con u
return (T, d)

In questo schema generale:

- T è dunque l'albero di copertura minimo che inizializzo come una foresta di copertura composta da nodi isolati e dunque anche la distanza di ogni nodo è posta ad infinito per tutti i nodi tranne che per la sorgente che ha distanza 0
- l'idea è quella di considerare solo gli archi che potrebbero aver avuto un miglioramento dalle modifiche precedenti; se dunque esiste un arco (u, v) per cui la condizione di Bellman non è rispettata (esattamente il duale di quanto indicato nell'algoritmo: d[u] + G.w(u, v) < d[v]) allora questo vuol dire che la soluzione non è quella ottima e quindi posso prima aggiornare la distanza (d[v] = d[u] + G.w(u, v)) ed aggiorno poi il padre del nodo v che sto raggiungendo con il nodo u in questione
- se al termine dell'esecuzione qualche nodo mantiene una distanza infinita, esso non è raggiungibile
- occorre domandarci "come" implementare la condizione di esistenza dell'arco

3. Algoritmo generico

Di seguito l'algoritmo "generico" per il calcolo del cammin minimo:

```
(int[], int[]) shortestPath(Graph G, Node s)
int[] d = new int[1… G.n] // vettore delle distanze
int[] T = new int[1… G.n] // vettore dei padri
DataStructure S = DataStructure()
boolean[] b = new boolean[1… G.n] // b[u]=true se u ∈ S
foreach u ∈ G.V() − {s} do
T[u] = nil
d[u] = +∞
b[u] = false
T[s] = nil
d[s] = 0
b[s] = true
S.add(s)
while !S.isEmpty()
int u = S.extract()
b[u] = false
foreach v ∈ G.adj(u)
if d[u] + G.w(u, v) < d[v]
                    if not b[v]
                                S.add(v)
                                b[v] = true
                    else
                                // DO SOMETHING
          T[v] = u
d[v] = d[u] + G.w(u, v)
return (T, d)
```

Alcuni dettagli sull'algoritmo:

- Il vettore delle distanze $d[u]$ rappresenta la distanza del nodo u dalla sorgente ed inizialmente tale valore è sovrastimato a $+\infty$
- La distanza della sorgente $d[s] = 0$
- T rappresenta il vettore dei padri e $T[u]$ identifica il padre del nodo u nell'albero di copertura che si sta costruendo; inizialmente tale albero contiene tutti valori nil
- Il vettore di boolean contiene l'informazione se un oggetto è presente nella struttura dati: se $b[u] = true$ significa che tale oggetto è nella struttura dati
- La struttura dati S è una struttura dati che potrò "personalizzare" a seconda del problema e che per il momento considero come una struttura dati "generica" su cui non faccio particolari ipotesi ma su cui vado da subito ad inserire la sorgente
- Finchè vi sono elementi nella struttura dati procedo con l'estrazione (secondo qualche criterio che andrò a definire) e non appena lo estraggo, aggiorno il mio vettore di boolean per gestire il fatto che questo elemento non appartiene più alla struttura dati
- Verifico se tra i vicini del nodo in esame, la distanza per andare a questo vicino è migliorata (if d[u] + G.w(u, v) < d[v]) e se così è allora procedo con l'aggiornamento del padre di v, cioè sto dicendo che per raggiungere v devo passare da u (essendo questa strada la migliore fino a questo punto), dopodichè posso aggiornare la distanza (d[v] = d[u] + G.w(u, v)). Occorre inoltre gestire il fatto se il nodo da raggiungere v sia già o meno stato scoperto / analizzato (cioè sia presente nella struttura dati): se non è presente lo vado ad inserire ed aggiorno il vettore dei booleani, altrimenti proseguo (ed eventualmente in base all'implementazione dell'algoritmo deciso cosa fare)

Abbiamo 4 punti da "personalizzare":

1. La scelta della struttura dati
2. La modalità di estrazione dell'elemento dalla struttura dati
3. L'inserimento del nodo nella struttura dati
4. L'azione da svolgere nel caso il nodo sia presente nella struttura dati

Bibliografia e sitografia

- Alan Bertossi, Alberto Motresor: Algoritmi e strutture di dati, Città Studi Edizioni, terza edizione
- C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione
- Crescenzi, Gambosi, Grossi: Strutture di Dati e Algoritmi, Pearson/Addison-Wesley
- Sedgewick: Algoritmi in C, Pearson, 2015
- Cormen Leiserson Rivest Stein-Introduzione Agli Algoritmi E Strutture Dati-Prima Edizione