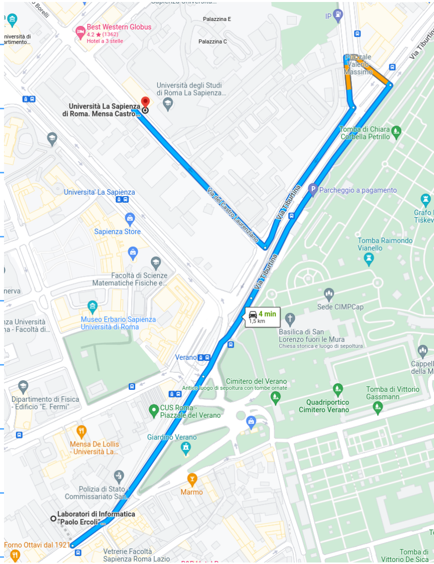
Cammini minimi in un grafo

Trovare il percorso piú breve tra due punti é un'applicazione estremamente comune



Questa ad esempio é la strada piú breve calcolata da un certo sito,

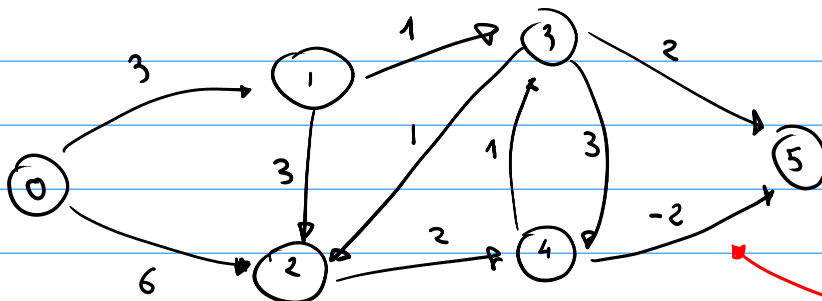
DA: Mensa del Castro Laurenziano

A: Laboratorio di via Tiburtina

Come si calcola un percorso del genere?

- Tipicamente un problema del genere si può modellare come un grafo <u>orientato</u> e <u>pesato</u>

- vertici sono i punti in cui scegliere la direzione (e.g. le intersezioni stradali)
- gli archi sono i percorsi (privi di scelte) tra due di questi punti
- il peso di ogni arco può essere: costo / durata / lunghezza



peso negativo!

Qual è il cammino minimo tra 0 e 5?

A occhio mi sembra $0 \xrightarrow{3} 1 \xrightarrow{1} 3 \xrightarrow{1} 2 \xrightarrow{2} 4 \xrightarrow{-2} 5$

Osservate che nonostante ci siano cammini con MENO ARCHI, non ci sono cammini con peso minore di 5.

Def Dato un grafo orientato e pesato con pesi $w: E \to \mathbb{R}$ e dato un cammino

$$p = v_0 e_1 v_1 e_2 v_2 \cdots e_\ell v_\ell \quad \text{con } e_i = (v_{i-1}, v_i) \text{ e } v_i \neq v_j \ \forall i \neq j$$

allora il COSTO del cammino p è $w(p) := \sum_{i=1}^{\ell} w(e_i)$

e la LUNGHEZZA del cammino p è ℓ.

Def Dati due vertici u e v in G orientato e pesato con pesi $w: E \to \mathbb{R}$

$$\delta(u,v) = \begin{cases} +\infty & \text{se } v \text{ non è raggiungibile da } u \\ \min\{w(p) : p \text{ cammino da } u \text{ a } v\} \end{cases}$$

ci sono un numero finito di cammini tra due vertici, quindi questa quantità è ben definita

OSSERVAZIONE

- Se un grafo non è pesato, convenzionalmente il peso di ogni arco è 1
- Quindi in un grafo non pesato $\delta(u,v)$ è la distanza di v da u

SOTTOSTRUTTURA OTTIMA DI UN CAMMINO MINIMO

- Dato un cammino di costo minimo, ovvero $w(p) = \delta(v_0, v_\ell)$

$$p = (v_0 e_1 v_1 e_2 v_2 \cdots e_\ell v_\ell)$$

ogni sottocammino $p_{ij} = (v_i e_{i+1} v_{i+1} \cdots e_j v_j)$ con $i < j$

ha costo minimo, ovvero $w(p_{ij}) = \delta(v_i, v_j)$

Dim per esercizio in classe

La sottostruttura ottima é la chiave per

- algoritmi greedy (e.g. selezione di attività, Kruskal, Prim, Dijkstra)
- programmazione dinamica (ne vedremo alcuni più avanti, ma uno per i cammini minimi è Floyd-Warshall)

e si basa sul fatto che la soluzione ottima deve contenere le soluzioni ottime di alcuni sottoproblemi.

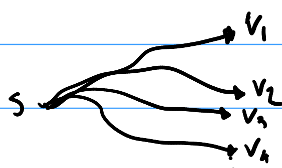
Varianti del problema dei cammini minimi su $G=(V,E,W)$ ← pesi

singola sorgente, singola destinazione

calcolo di $\delta(s,t)$

(A)

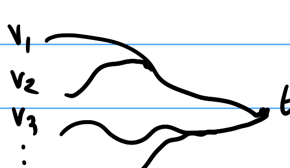
singola sorgente, multipla destinazione



calcolo di $\delta(s,v_i)$ $\forall v_i \in V(G)$

(B)

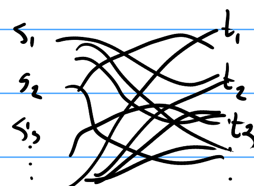
multipla sorgente, singola destinazione



calcolo di $\delta(v_i,t)$ $\forall v_i \in V(G)$

(C)

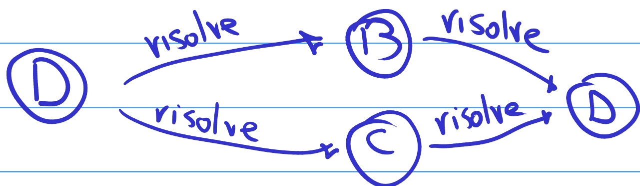
multipla sorgente, multipla destinazione



calcolo $\delta(u,v)$ $\forall u,v \in V(G)$

(D)

Considerazioni sulla complessità:



- Complessità di (B) = Complessità di (C) [Perché?]
- Complessità di (B) ≤ n · Complessità di (A)
- Complessità di (D) ≤ n · Complessità di (B)

Lo stato dell'arte

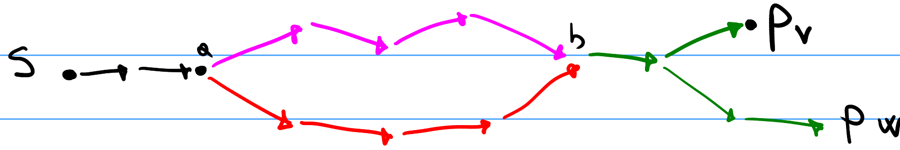
- Tutti gli algoritmi noti per (A) hanno la stessa complessità dei migliori per (B) e (C)

  Potete immaginare il perché?

- Esistono algoritmi per (D) migliori di n · Complessità di (B)

Soluzione di un problema singola sorgente/multipla destinazione

- Prendiamo le soluzioni dei cammini minimi: p_v il cammino dalla sorgente s a v, per ogni $v \in V(G)/\{s\}$



- È possibile che l'unione di due cammini p_v e p_w producano due cammini alternativi tra due vertici (in questo esempio a, b)

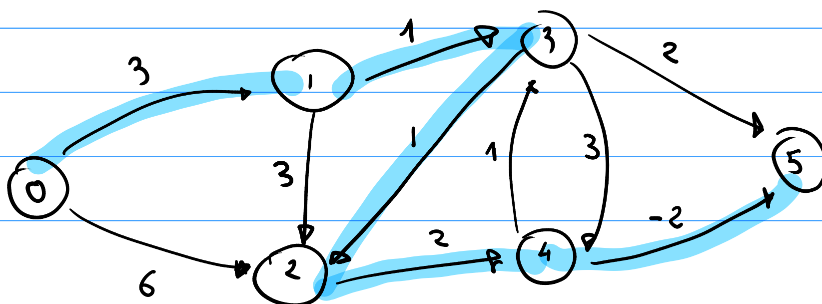
- Osservate che il costo del cammino rosso e di quello viola devono essere $\delta(a,b)$ per la proprietà della sotto struttura ottima

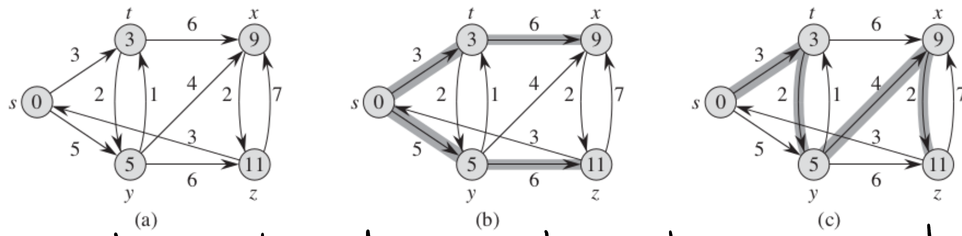
- Pertanto si può usare lo stesso segmento (rosso o viola è indifferente) per p_v e p_w

- Si può ripetere questo procedimento ogni volta che il grafo contenente l'unione di tutti i cammini ha un nodo con grado entrante > 1

- Alla fine del processo l'unione di tutti i cammini sarà un grafo nel quale
  - esiste un cammino da s a qualunque altro vertice
  - il grado entrante di ogni vertice $\neq s$ è 1

OVVERO UN ALBERO RADICATO IN S NEL QUALE TUTTI I VERTICI DEL GRAFO SONO SUOI DISCENDENTI





Qui potete vedere due diversi alberi che rappresentano ognuno tutti i cammini minimi dalla sorgente 0

Rappresentazione della soluzione: albero dei cammini minimi

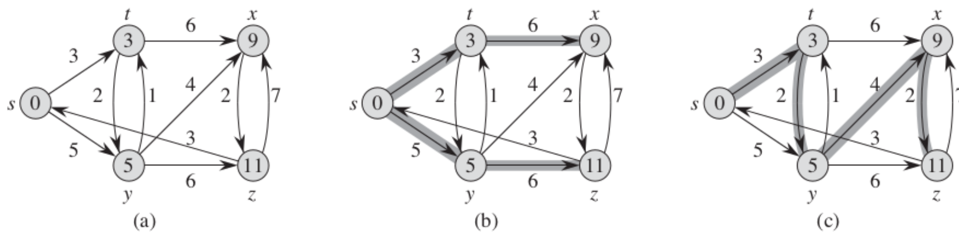
array dei predecessori $P = [\underbrace{\cdots\cdots}_{n}]$

$P[v]$ indica il vertice w che precede v nel cammino dalla sorgente s

array delle distanze $dist = [\underbrace{\quad\quad}_{n}]$

dove $dist[v] = \delta(s,v)$ se esiste un cammino tra s e v

$= +\infty$ altrimenti



s t x y z
$P = [-, s, t, s, y]$
$dist = [0, 3, 9, 5, 11]$

s t x y z
$P = [-, s, y, t, x]$
$dist = [0, 3, 9, 5, 11]$

DURANTE L'ESECUZIONE DEGLI ALGORITMI

P, dist mantengono una soluzione parziale

INIZ. • $P = [NIL, NIL, NIL, \cdots\cdots]$, $dist[s] = 0$ $dist[v] = +\infty$ per $v \neq s$

DURANTE L'ESECUZIONE • P = mantiene un albero che collega solo alcuni dei vertici

$dist[v]$ contiene una stima per eccesso di $\delta(s,v)$ dovuta a P che viene via via migliorata

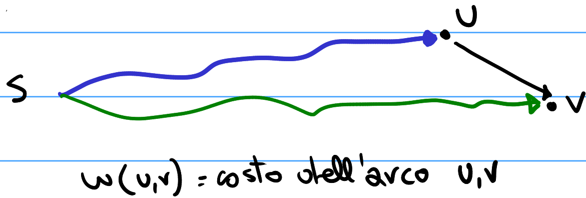
$dist[v] = +\infty$ . . . . $dist[v] \geq \delta(s,v)$ . . . $dist[v] = \delta(s,v)$

COME VIENE MIGLIORATA LA STIMA?

- Gli algoritmi singola sorgente - destinazione multipla che vedremo
  - Dijkstra
  - Bellman-Ford

si basano sulla stessa idea

"scegli un arco (u, v) e vedi se questo migliora la stima dist[v]"




```
if dist[v] > dist[u] + w(u,v):
    dist[v] = dist[u] + w(u,v)
    P[v] = u
```

nel libro questa operazione viene chiamata "rilassamento" di (u,v)

- Entrambi gli algoritmi sono in effetti procedure che producono una lista di archi $e_1, e_2, \ldots\ldots$ (anche con ripetizioni) e che effettuano l'operazione di rilassamento su questi archi

Vedi esempio precalcolato

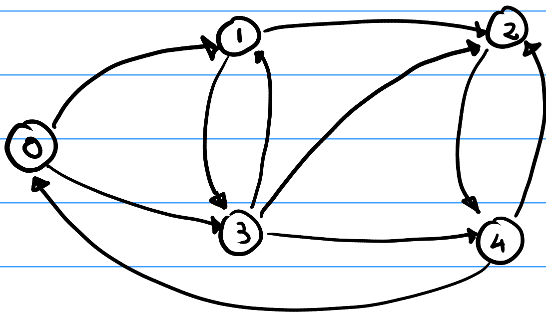
- Osserviamo che l'arco $(1, 2)$ nell'esempio viene rilassato due volte. La prima volta il vertice 1 non è ancora alla sua distanza minima, quindi il rilassamento di $(1,2)$ porta comunque ad un valore dist[2] troppo alto.

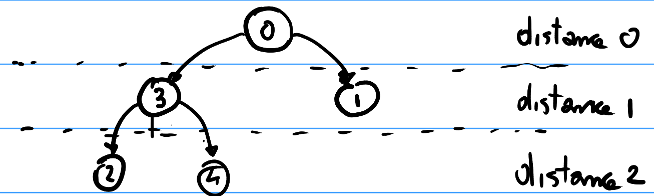
ALGORITMO di Dijkstra

- Dato un grafo pesato, orientato, dove tutti i pesi sono NON NEGATIVI (ovvero ≥ 0)
- e una sorgente $s \in V(G)$
- calcola P, dist che rappresentano l'albero dei cammini minimi e i costi dei cammini minimi

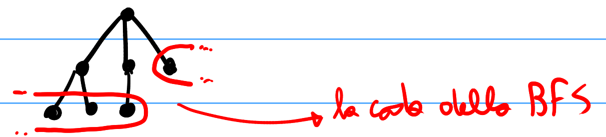
OSSERVAZIONE: se il peso di tutti gli archi del grafo è 1, il cammino minimo è quello con meno archi, e il costo $\delta(s,v)$ è in effetti la distanza in termini di numero di archi tra s e v.

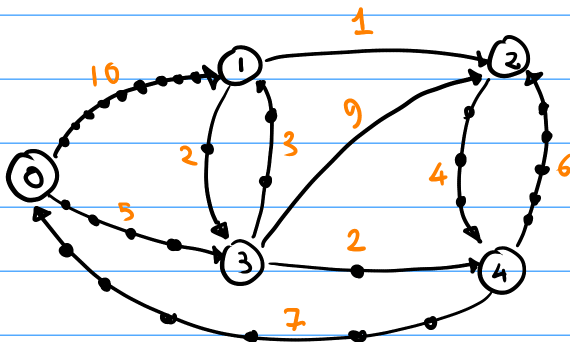
ABBIAMO GIÁ UN ALGORITMO CHE CALCOLA QUESTE DISTANZE: BFS





In BFS un vertice entra nella coda quando viene scoperto (e la sua distanza determinata) e viene analizzato quando esce dalla coda





- Assumiamo per un attimo che i pesi siano tutti interi ≥ 1 e $\leq W$

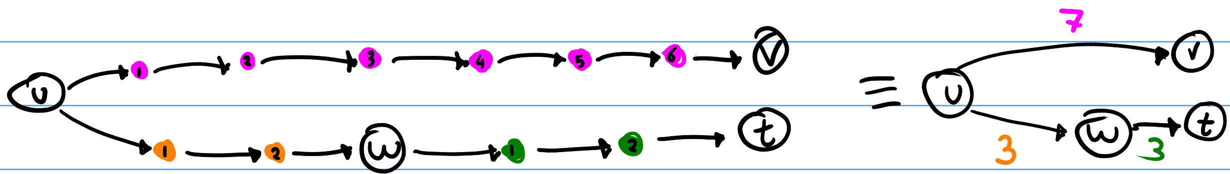
$u \xrightarrow{3} v$

può essere rappresentato con un numero di archi pari al peso

$G' = (V', E')$ con $|V'| = |V| + |E|(W-1)$ $\quad |E'| = |E| \cdot W$

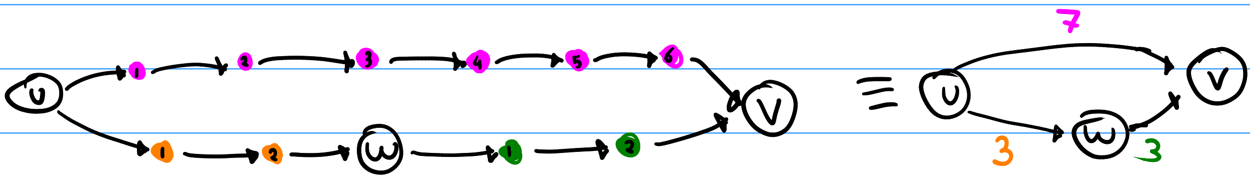
complessità di BFS su G' è $O(|V| + |E| \cdot W)$

OSSERVIAMO MEGLIO CHE SUCCEDE ALLA CODA DI QUESTA BFS



- In BFS, senza pesi, visitare (u) porterebbe (w) e (v) insieme in coda e il vertice (t) verrebbe inserito dopo l'uscita di (w) dalla coda e quindi successivamente a (v).
- La BFS, sul grafo con gli archi spezzati, inserirà 1 e 1, poi quando questi escono inserirà 2 e 2, e poi quando questi escono, inserirà (w) e 3.

- Prima che (v) entri nella coda, avremo che 1, 2 e (t) saranno entrati e usciti dalla coda
- In un certo senso questi vertici speciali "rallentano" l'ingresso della destinazione di un arco nella coda
  con $u \to v$, (v) entra immediatamente, con $u \xrightarrow{7} v$ l'ingresso di (v) è subordinato al fatto che 6 vertici speciali facciano 6 tour nella coda, mentre per $u \xrightarrow{3} w$ ne bastano 2.
- I vicini di (u) vengono inseriti nella coda con PRIORITÁ differenti



- In questo caso la visita di (u) attiva i 6 tour della coda da parte dei vertici speciali ●. Tuttavia nel frattempo completano i tour della coda tutti i vertici 1, 2, (w) il che attiva i tour della coda da parte di 1 e 2
- L'uscita di 2 dalla coda fa scoprire (v), "anticipando" la scoperta che si sarebbe ottenuta da 6.

L'arco $w \xrightarrow{3} v$ ha aumentato la PRIORITÁ di (v)

- Di fatto l'algoritmo di Dijkstra è come una BFS con coda di priorità invece di una normale coda.
- Il vertice V con maggiore priorità è quello con valore dist[v] più piccolo
- Quando un vertice (v) esce dalla coda, e ha un arco (v) → (w), anche se (w) è già in coda questo arco non è ignorato come nella BFS, ma è RILASSATO secondo lo schema visto a pag. 6.

- Altra differenza con BFS canonico:
  in BFS i vertici sono (1) nell'albero di visita (NERI) (2) nella coda (GRIGI) (3) non scoperti (BIANCHI)

  poiché usiamo una coda di priorità, possiamo considerare i vertici BIANCHI come vertici nella coda con valore di chiave $+\infty$.

ABBIAMO VISTO CHE BFS su grafi con archi spezzettati può essere simulato più efficientemente utilizzando una CODA DI PRIORITÀ

```
1
2  def Dijkstra(s,G,W):
3      n  = len(G)
4
5
6      Q  =...  # Coda di priorità
7
8      # Inizializzazione di P/dist
9      P        = [None]*n
10     dist     = [inf]*n
11     dist[s] = 0
12     for v in range(n):
13         Q.insert(v,dist[v])
14
15     while len(Q)>0:
16         v = Q.extract_min()
17
18         for w in G[v]:
19             # Relax W[v,w]
20             if w in Q and dist[w] > dist[v] + W[v,w]:
21                 P[w]=v
22                 dist[w] = dist[v] + W[v,w]
23                 Q.decrease_key(w,dist[w])
24
25     return P,dist
26
27
```

Annotazioni:
- righe 8–13 → Stato iniziale della coda: $s \mapsto 0$, $v \mapsto +\infty \ \forall v \neq s$
- riga 16 → estrazione del minimo
- righe 18–23 → rilassamento degli archi

Confrontate col ciclo interno di Prim

```
while len(Q)>0:
    v = Q.extract_min()

    u = closest[v]
    T.append(u,v)

    # Aggiorna l'arco migliore
    for w in G[v]:
        if w in Q and dist[w]>W[v,w]:
            closest[w]=v
            dist[w]=W[v,w]
            Q.decrease_key(w,dist[w])
```

Vedere esempi precalcolati

Complessità di Dijkstra

- Ogni arco viene rilassato al massimo una volta
- L'estrazione del minimo viene effettuata $|V|$ volte

come per Prim:

- La complessità dipende da come è implementata la coda di priorità

- Un semplice array : rilassamento $O(1)$, estrazione del minimo $O(|V|)$
  in totale $O(|V|^2 + |E|) = O(|V|^2)$

- Heap : rilassamento $O(\log|V|)$, estrazione del minimo $O(\log|V|)$

  in totale $O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$

se $|E| = o(|V|^2/\log|V|)$ allora la coda di priorità implementata con Heap è più efficiente

OSSERVAZIONE Implementando la coda di priorità con Heap di Fibonacci si arriva anche ad abbassare la complessità a $O(|V|\log|V| + |E|)$

Thm Dato un grafo $G = (V, E, w)$ pesato e orientato

l'algoritmo di Dijkstra calcola correttamente $\delta(s,v)$ $\forall v \in V$

dove s è la sorgente

dim

La dimostrazione è basata sull'invariante, dimostrato per induzione, che

$$dist[v] = \delta(s,v)$$ per ogni vertice che esce dalla coda.

Esercizio Dimostrare che P rappresenta l'albero dei cammini minimi ovvero che il cammino minimo da $s \to v$ di costo $\delta(s,v)$ si ricava da P