

I problemi informatici si possono suddividere in 3 categorie

- Decisione  $\xrightarrow{\text{Risposta}}$  Si/No // Il grafo è ciclico? // grafo è un DAG?
- Ricerca  $\xrightarrow{\quad}$  Soluzione // Albero di Ricerca
- Ottimizzazione  $\xrightarrow{\quad}$  Minimo/Massimo // Cammino Minimo, oltre a dare una soluzione, dà la migliore

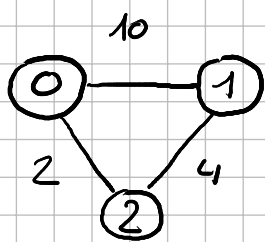
Ora vedremo il 1° algoritmo di ottimizzazione

## Algoritmo di Dijkstra =

L'algoritmo di Dijkstra è in grado di trovare il cammino minimo su un grafo con archi pesati.

Inoltre è un algoritmo Greedy, ovvero un algoritmo che sceglie in un dato momento la soluzione migliore, e non cambia mai più la scelta appena fatta.

Es:



Sorg 0

$0 \rightarrow 2 \rightarrow 1$

Dijkstra funziona solo con pesi non negativi

Pseudo codice:

algoritmo Dijkstra (sorg) {

$S = \{sorg\}$

$d[sorg] = 0$  // la distanza di un nodo a se stesso è zero

finché possibile scegliere un arco  $(u,v)$  con  $u \in S$  e  $v \notin S$  tale che

$$d[u] + \text{peso}(u,v) \text{ minimo}$$

//  $d$  sono le distanze dalla sorgente

$$S = S \cup \{sorg\}$$

$$d[v] = d[u] + \text{peso}(u,v)$$

Adesso che hai capito il funzionamento entriamo più nel dettaglio di questa operazione:

finché possibile scegliere un arco  $(u,v)$  con  $u \in S$  e  $v \notin S$  tale che  $d[u] + \text{peso}(u,v)$  minimo

Come si fa a scegliere un arco con il peso minimo?

Per farlo utilizziamo un Heap! Infatti nell'heap salviamo tutti

gli archi che attraversano la frontiera, poi sarà l'heap a restituirci il minimo.

Pseudo codice con Heap:

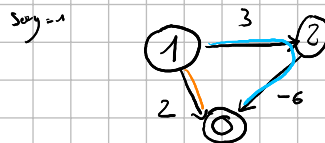
**INITIALIZATIONS CORRECT**

**Dijkstra (sorg)**  
 $S = \{sorg\}, d[sorg] = 0$   
 $A = \emptyset$   
 $MinHeap = mh$   
 $\forall \text{ arco } u \rightarrow v \text{ da sorg } (sorg, v)$   
 $mh \leftarrow ((u, v), d[sorg] + p(sorg, v))$   
**FINCHÉ**  $mh$  non è vuoto **PRIMA**  
 $(u, v)$  arco minimo da  $mh$   $\rightarrow$  estraggo l'arco con dist. minima min

**IF**  $v \notin S$  **{** // se  $v$  non scoperto  
 $S = S \cup \{v\}$  // lo aggiungo non scoperto  
 $A = A \cup \{(u, v)\}$  // aggiungo l'arco all'albero  
 $d[v] = d[u] + p(u, v)$ , per ogni  $z$  vicino di  $v$   
 $mh \leftarrow ((v, z), d[v] + p(v, z))$   
**}**

**Costo**

Perché Dijkstra non funziona con pesi negativi



- Scelto da Dijkstra
- // vero cammino minimo

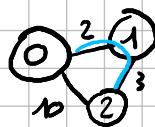
| sottocammini di un cammino

minimo sono a loro volta cammini minimi.

Sorg: 0

Cam. min.:  $0 \rightarrow 1 \rightarrow 2$

Anche  $0 \rightarrow 1$  è cam. minimo



Costo:

$O(m \log n)$

data dai costi di gestione dell'Heap (Riordinamento)

data dall'inserimento degli archi nell'heap

$m \rightarrow n^2$

Questo è sempre vero, ma se abbiamo un grafo denso otteniamo:

$$O(m \log n^2) \rightarrow O(m \log n)$$

asintoticamente parlando

// perché si abbassa se il grafo è denso? Ho comunque più archi da controllare

Dimostrazione:

Per dimostrare il grafo utilizziamo una dimostrazione per induzione.

distanza calcolata da Dijkstra:  $d$

distanza effettiva nel grafo:  $\delta$

Dobbiamo dimostrare che la distanza di  $d[v]$  sia  $= \delta(sorg, v)$

Pseudo codice:

algoritmo Dijkstra(sorg)

$S = \{sorg\}$   
 $d[sorg] = 0$

finché possibile scegliere un arco  $(u, v)$  con  $u \in S$  e  $v \notin S$  tale che

$$d[u] + \text{peso}(u, v) \text{ minimo}$$

$$S = S \cup \{sorg\}$$

$$d[v] = d[u] + \text{peso}(u, v)$$

Ad ogni iterazione si scopre un nodo nuovo.  
Quindi l'induzione è su  $|S|$ .

Caso BASE:

$$|S|=1 \quad d[sorg]=0$$

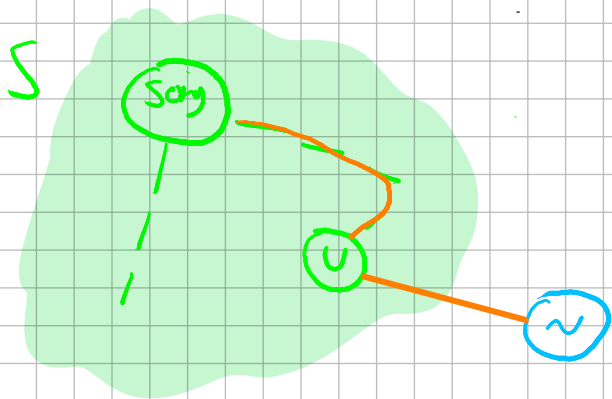
Passo Induttivo (Ipotesi  $\Rightarrow$  Tesi)  $\rightarrow$  implica

$$Ipotesi: |S|=k \Rightarrow d[v] = d(sorg, v) \text{ con } v \in S$$

$$Tesi: |S|=k+1 \Rightarrow d[v] = d(sorg, v) \text{ con } v \in S$$

Questo significa che al passo successivo la distanza di  $v$  è ancora corretta, ovvero uguale a quella reale.

Come dimostrare la tesi

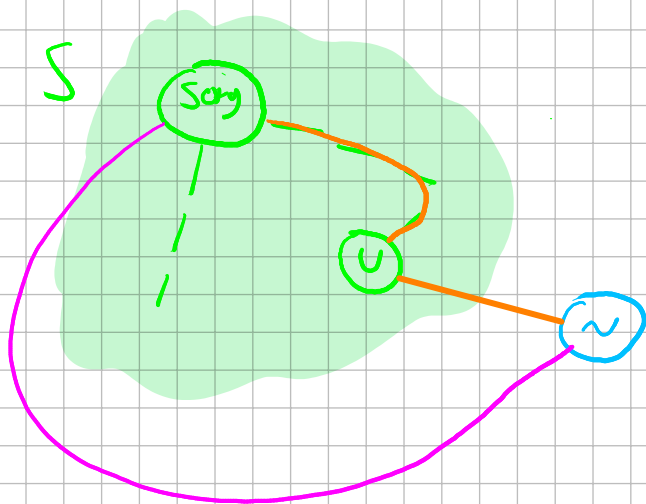


Si presuppone che

le distanze nella regione  $S$  sono tutte calcolate correttamente.

Dimostriamo che anche la distanza del nuovo nodo  $v$  sia calcolata correttamente.

Guardiamo un qualunque altro cammino che va dalla sorgente al nodo  $v$



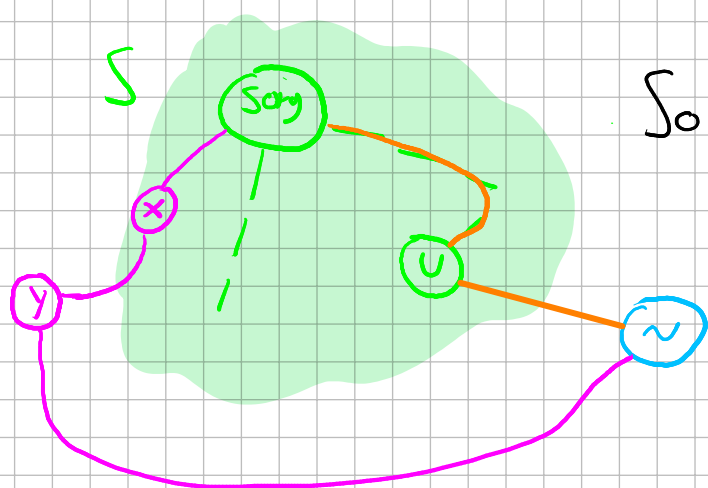
- Calcolato da Dijkstra
- Possibile altro cammino

Vogliamo che la lunghezza del cammino  $l(sorg \rightarrow v)$  sia  $l(v)$   $\rightarrow$   
 $\leq$  di  $l(sorg \rightarrow v)$   
 $l(v)$

Se dimostriamo questa proprietà allora possiamo dire che l'algoritmo sceglie sempre correttamente il cammino minimo.

$$l(c) \leq l(c')$$

Noi sappiamo che qualsiasi sia il cammino  $c'$  prima o poi dovrà attraversare la frontiera. Per esempio passiamo la frontiera con l'arco  $(x, y)$ .



So che questo arco deve esistere perché parte da un nodo facente parte di  $S$  e deve finire in un nodo che non ne fa parte, ovvero  $v$ .

Possiamo dire che  $l(c') = l(\text{Sorg} \rightsquigarrow y) + l(y \rightsquigarrow v)$ ,

questo ci serve perché così possiamo dire che  $l(y \rightsquigarrow v) \geq 0$ .

Inoltre possiamo dire che  $l(y \rightsquigarrow v) \geq l(\text{Sorg} \rightsquigarrow y)$ .

Se simulassimo di togliere il pezzo di cammino  $y, v$  avremmo:

$$l(\text{Sorg} \rightsquigarrow x) + \text{peso}(x, y)$$

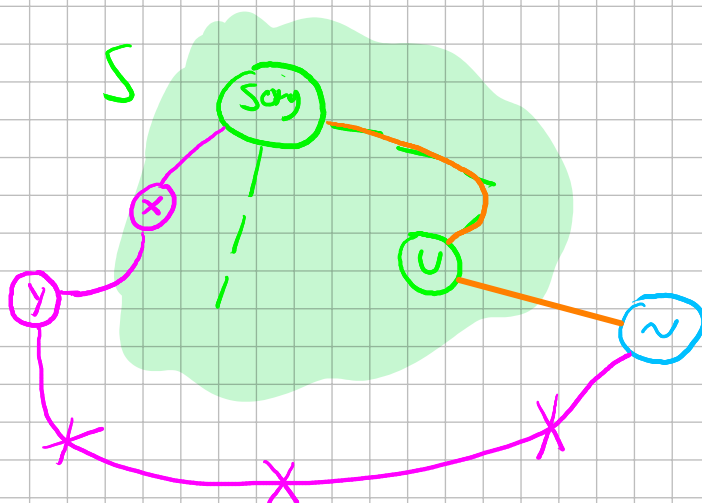
Così possiamo dire sul cammino

$$l(\text{Sorg} \rightsquigarrow x)?$$

È un cammino qualunque da

Sorg ad  $x$ , quindi la sua lunghezza è

$$\geq d(\text{Sorg}, x) + \text{peso}(x, y)$$



Ma  $x$  fa parte di  $S$ , quindi è stato esplorato dall'algoritmo.  
Dato che è stato esplorato dall'algoritmo possiamo usare  
l'ipotesi induttiva di partenza, ovvero  $d[v] = d(s_{orig}, v)$ .

Quindi abbiamo  $d[x] + peso(x, y)$ .

Ma l'algoritmo per sua natura sceglie l'arco meno distante, quindi  
sicuramente abbiamo che

$$d[x] + peso(x, y) \geq d[v] + peso(v, y)$$

→ ma questo è quello che aveva scelto

Dijkstra dalla partenza, quindi abbiamo  
verificato la tesi, ovvero

$$l(c) \leq l(c')$$

