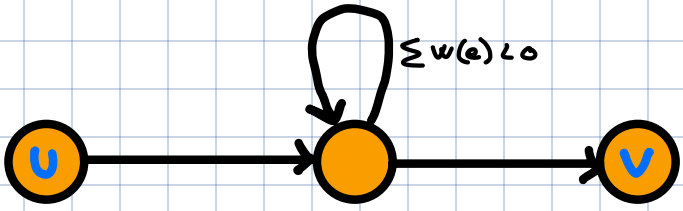
CAMMINI MINIMI

CAMMINO MINIMO TRA UNA COPPIA DI VERTICI → CAMMINO CON COSTO MINORE O UGUALE A QUELLO DI OGNI ALTRO CAMMINO TRA GLI STESSI VERTICI.
PUÓ NON ESSERE UNICO

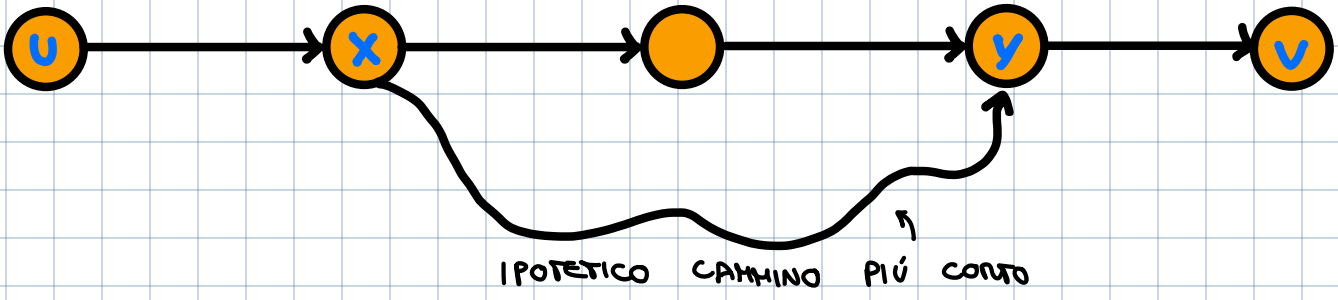
DATI DUE NODI u e v INDICHIAMO CON $d_G(u,v)$ IL CAMMINO MINIMO. (DISTANZA)

- SE NON ESISTE $d_G(u,v) = +\infty$
- SE C'É UN CAMMINO CON UN CICLO CON COSTO NEGATIVO ALLORA $d_G(u,v) = -\infty$



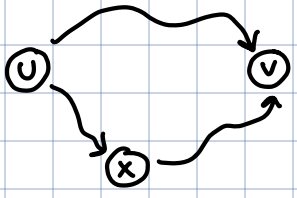
SOTTOSTRUTTURA OTTIMA

OGNI SOTTOCAMMINO DI UN CAMMINO MINIMO É UN CAMMINO MINIMO.

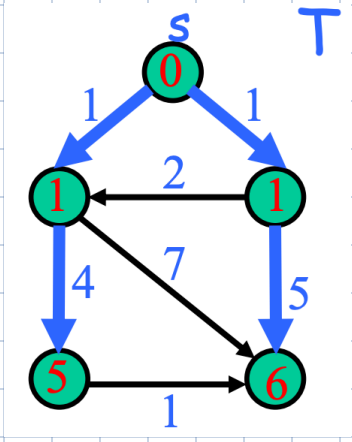


SE COSÍ FOSSE ALLORA IL CAMMINO MINIMO TRA u e v DOVREBBE ESSERE u-x, x-y, y-v

$\forall u,v,x \in V: \; d(u,v) \leq d(u,x) + d(x,v)$ → DISUGUAGLIANZA TRIANGOLARE



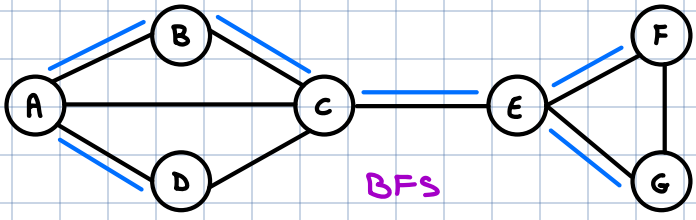
ALBERO DEI CAMMINI MINIMI

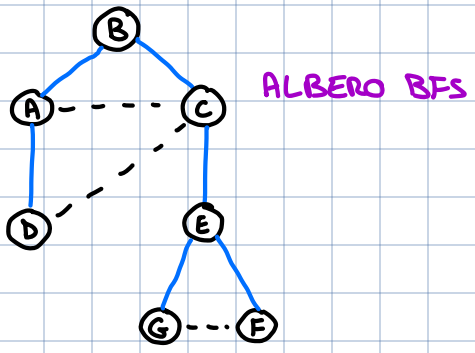


⟶ T È UN ALBERO DEI CAMMINI MINIMI CON SORGENTE S.
È RELATIVO AD UN GRAFO $G = (V, E, w)$.

- T È RADICATO IN S
- $\forall v \in V$ VALE CHE $d_T(s,v) = d_G(s,v)$

NOTA: PER GRAFI NON PESATI L'ALBERO È UGUALE ALL'ALBERO BFS.





ALGORITMO DIJKSTRA

ASSUNZIONE ⟶ TUTTI GLI ARCHI HANNO PESO $w \geq 0$.

STEPS:

① INIZIALIZZAZIONE ⟶ ASSEGNA LA DISTANZA 0 AL NODO SORGENTE E LA DISTANZA $+\infty$ A TUTTI GLI ALTRI NODI.

USIAMO UNA STRUTTURA DATI (CODA DI PRIORITÀ / MINHEAP) PER GESTIRE I NODI DA ESPLORARE, IN BASE ALLA LORO DISTANZA MINIMA CORRENTE.

TIENE TRACCIA DEI NODI VISITATI IN UN'ALTRA STRUTTURA DATI.

② SCELTA DEL NODO ⟶ SELEZIONA IL NODO U CON DISTANZA MINIMA TRA QUELLI NON ANCORA VISITATI, VIENE ESTRATTO DALLA CODA DI PRIORITÀ.

MARCA IL NODO U COME VISITATO.

③ AGGIORNAMENTO DISTANZE ⟶ SIAMO SUL NODO U, PER OGNI VICINO CALCOLIAMO LA DISTANZA PROVVISORIA PASSANDO PER U:

$$\text{NUOVADIST}(V) = \text{DIST}(U) + \text{PESO}(U,V)$$

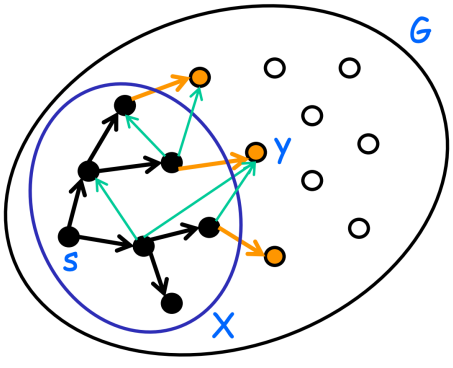
DIST. DALLA SORG. A U — PESO ARCO (U,V)

SE È MINORE DELLA DISTANZA ATTUALE DI V ALLORA LA AGGIORNA:

$$\text{DIST}(V) = \text{NUOVADISTANZA}(V)$$

AGGIUNGIAMO / AGGIORNIAMO V NELLA CODA DI PRIORITÀ.

④ RIPETI ⟶ RIPETI FINCHÈ LA CODA DI PRIORITÀ NON È VUOTA O NON HAI VISITATO TUTTI I NODI.



X = STRUTTURA DATI DEI NODI VISITATI

● = NODI SCOPERTI, HANNO COSTO $< +\infty$, SONO MANTENUTI IN UNA CODA DI PRIORITÀ, VERRÀ SCELTO QUELLO MENO COSTOSO.

algoritmo `Dijkstra`$(grafo\ G, vertice\ s) \rightarrow albero$

for each (vertice u in G) **do** $D_{su} \leftarrow +\infty$ → ASSEGNO VALORE $+\infty$ A TUTTI I NODI

$\widehat{T} \leftarrow$ albero formato dal solo nodo s; X←∅ → CREO $\widehat{T}$ E CI METTO LA RADICE, X È VUOTO

`CodaPriorita S` → CREO LA CODA DI PRIORITÀ S

$D_{ss} \leftarrow 0$ → PONGO LA DISTANZA DA S A S = 0

`S.insert`$(s, 0)$ → INSERISCO LA SORGENTE NELLA CODA DI PRIORITÀ

while (**not** `S.isEmpty()`) **do**

  $u \leftarrow$`S.deleteMin()`; X←X$\cup${u} → STACCO DALLA CODA IL NODO MIN, LO AGGIUNGO AI VISITATI

  for each (arco (u, v) in G) **do** → PER OGNI VICINO DEL NODO ESTRATTO:

    if $(D_{sv} = +\infty)$ **then** → SE IL COSTO DEL VICINO È $+\infty$

      `S.insert`$(v, D_{su} + w(u, v))$ → INSERISCI NELLA CODA $D_{su} + w(u,v)$, IL COSTO DA S A U + PESO ARCO (U,V)

      $D_{sv} \leftarrow D_{su} + w(u, v)$ → PONI IL COSTO DI V = $D_{su} + w(u,v)$

      rendi u padre di v in $\widehat{T}$ → INSERISCI V COME FIGLIO DI U NELL'ALBERO

    else if $(D_{su} + w(u, v) < D_{sv})$ **then** → SE V HA GIÀ UN COSTO E $D_{su} + w(u,v) < D_{sv}$ (MIGLIORE):

    * `S.decreaseKey`$(v, D_{sv} - D_{su} - w(u, v))$ → DECREMENTO LA CHIAVE DI V NELLA CODA DI PRIORITÀ

      $D_{sv} \leftarrow D_{su} + w(u, v)$ → AGGIORNO IL COSTO

      rendi u nuovo padre di v in $\widehat{T}$ → AGGIORNO IL PADRE DI V NELL'ALBERO.

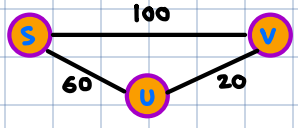
return $\widehat{T}$

* DECREMENTO LA CHIAVE DI $D_{sv} - D_{su} - w(u,v)$

CON:

- $D_{sv} = 100$
- $D_{su} = 60$
- $w(u,v) = 20$

S [V = 100]



DECREMENTO $S(v) - (\underbrace{100}_{D_{sv}} - \underbrace{60}_{D_{su}} - \underbrace{20}_{w(u,v)}) \rightarrow S(v) = 100 - 20 \rightarrow S(v) = 80$

(20)

NUOVO COSTO!!!

DIM CORRETTEZZA

LEMMA

QUANDO IL NODO V VIENE ESTRATTO DALLA CODA CON PRIORITÀ VALE:

- $D_{sv} = d(s,v)$
- IL CAMMINO DA S A V NELL'ALBERO HA COSTO $d(s,v)$ CIOÉ IL CAMMINO MINIMO IN G.

ANALISI DELLA COMPLESSITÀ

- OGNI NODO VIENE INSERITO UNA VOLTA, QUINDI $m \cdot$ INSERT
- OGNI NODO VIENE ESTRATTO UNA VOLTA QUANDO LA SUA DISTANZA È DEFINITIVA, QUINDI $m \cdot$ DELETEMIN.
- AGGIORNIAMO IL COSTO DI UN NODO AL PIÚ UNA VOLTA PER ARCO, QUINDI m DECREASEKEY.

L'arco (u, v) viene rilassato ogni volta che il nodo u viene visitato (cioè estratto dalla coda di priorità) e i suoi vicini vengono esaminati.

L'algoritmo esamina tutti gli archi **in uscita** dal nodo u al momento della sua estrazione dalla coda. Quando questo accade:

1. Verifica la possibilità di aggiornare $d[v]$ tramite u.
2. Se $d[v]$ viene migliorata, esegue un'operazione `decrease-key` per aggiornare la distanza di v nella coda.

QUANTE VOLTE VIENE FATTO DECREASEKEY

	Insert	DelMin	DecKey
Array non ord.	O(1)	O(n)	O(1)
Array ordinato	O(n)	O(1)	O(n)
Lista non ord.	O(1)	O (n)	O(1)
Lista ordinata	O(n)	O(1)	O(n)

	Insert	DelMin	DecKey
Heap binario	O(log n)	O(log n)	O(log n)
Heap Binom.	O(log n)	O(log n)	O(log n)
Heap Fibon.	O(1)	O(log n)* (ammortizzata)	O(1)* (ammortizzata)

UTILIZZANDO GLI HEAP DI FIBONACCI LA COMPLESSITÀ È:

- INSERT $\rightarrow O(1)$ FATTO m VOLTE
- DELMIN $\rightarrow O(\log m)$ FATTO m VOLTE
- DECREASEKEY $\rightarrow O(1)$ FATTO m VOLTE

$$m \cdot O(1) + m \cdot (\log m) + m \cdot O(1)$$

$$\downarrow$$

$$O(m + m \cdot \log m)$$

DIJKSTRA

↳ TROVA IL CAMMINO MINIMO TRA 2 NODI $O(m + m \log m)$