Risolviamo il SSSP

$G \equiv (V, E)$ $s \in V$ $\forall v \in V$ vogliamo calcolare $\delta(s, v)$

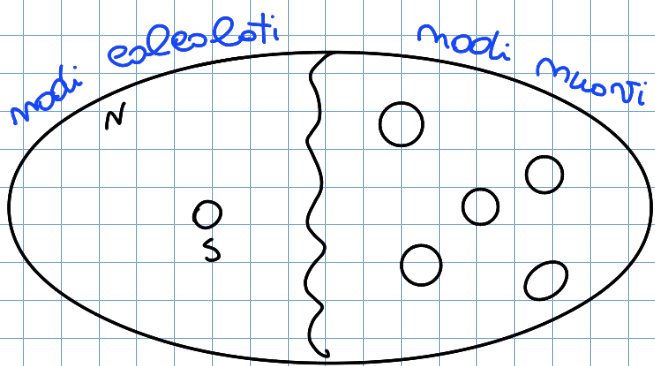
Per ordine fatto con $\forall v \in V$ $d[v]$ = stima
$\pi[v]$ = predecessore

$d[v] = +\infty$ all'inizio
$\downarrow$ relax(u, v)
$d[v] = \delta(s, v)$

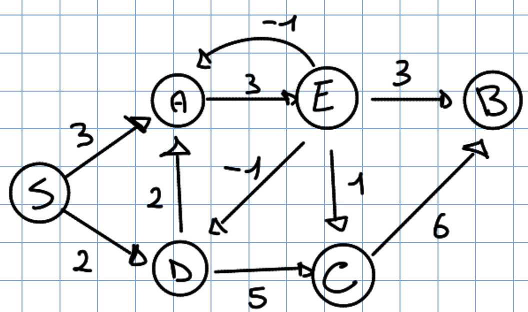
Relax(u, v, w)
if $(d[v] > d[u] + w(u, v))$
then $d[v] \leftarrow d[u] + w(u, v)$
$\pi[v] \leftarrow u$

Algoritmo Bellman-Ford (l'unico generalista tra quelli studiati)

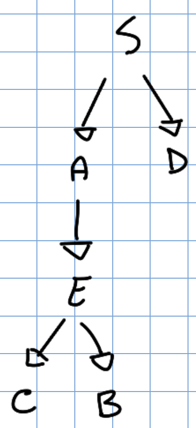
Complessità $= O(VE)$ con liste di adiacenza
$O(V^3)$ con matrice di adiacenza



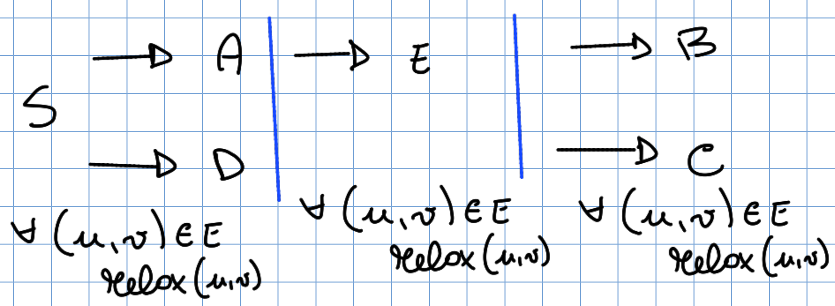
Questa è la situazione iniziale nella maggior parte dei casi tranne nei casi in cui ci sono archi non raggiungibili, saranno già nei calcolati ma noi non lo sapremo prima della fine dell'algoritmo



albero ottimale dei cammini minimi da s



Bellman-Ford



Non sapendo quali archi rilassare li rilasso tutti, ora il problema è capire quando fermarsi:

Deve essere fatto $\underbrace{n-1}$ volte

↳ lunghezza del cammino minimo più lungo possibile

così sicuramente rilasso tutti gli archi

```
Bellman-Ford (G, s, w)
  d = new Array( len(V))
  for each v ∈ V Do
      d[v] = +∞
      π[v] = NIL
  d[s] = 0
  for i ← 1 to V-1 Do
     for each (u,v) in E Do
         relax (u, v, w)
  return d
```

init grafo

lunghezza massima di un cammino minimo

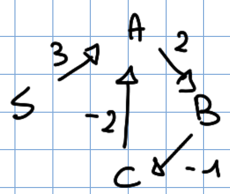
V · E volte

Se consideriamo i cicli negativi → modi mai in convergenza

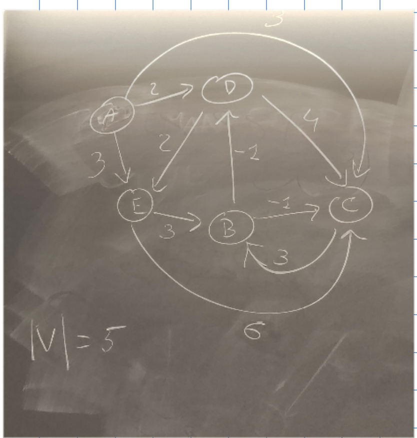
```
Bellman-Ford (G, s, w)
  d = new Array( len(V))
  for each v ∈ V Do
      d[v] = +∞
      π[v] = NIL
  d[s] = 0
for i ← to V-1 Do
    for each (u,v) in E Do
        relax (u,v,w)

for each (u,v) in E Do
    if (d[v] > d[u] + w(u,v)) then
        return false

return d
```

se alla V-esima iterazione trovo delle differenze allora ritorno FALSE

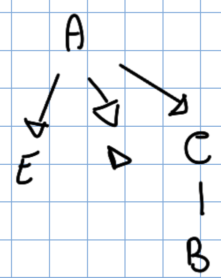
Bellman-Ford con



→ ritorna FALSE



	D^0	D^1	D^2	D^3
A	0	0	0	0
B	∞	6	6	6
C	∞	3	3	3
D	∞	2	2	2
E	∞	3	3	3



All'esame può chiedere queste cose fatte in questo modo

$W : E \to \mathbb{R}^+$ } peso solo positivo, niente archi negativi

$\exists\, u : d[u] = \delta(s,u)$ } se troviamo un nodo convergente

$\forall v \in ADJ(u)$

relax(u, v)

possiamo rilassare tutti gli archi uscenti

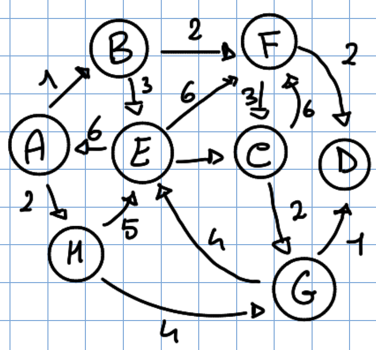
→ è il minore tra tutti i nodi che partono dalla sorgente e non sono in convergenza

all'inizio s poi cambia

visto che sono archi uscenti da v questi avremmo sicuramente peso maggiore questo ci conferma che v è in convergenza $v = s$ all'inizio

questo si basa sull'ipotesi che gli archi hanno solo peso positivo

Applicazione di DIJKSTRA



	d^0	d^1_A	d^2_B	d^3_H	d^4_F	d^5_E	d^6_C	d^7_C
A	0	0	0	0	0	0	0	0
B	∞	1	1	1	1	1	1	1
C	∞	∞	∞	∞	6	5	5	5
D	∞	∞	∞	∞	5	5	5	5
E	∞	∞	4	4	4	4	4	4
F	∞	∞	3	3	3	3	3	3
G	∞	∞	∞	6	6	6	6	6
H	∞	2	2	2	2	2	2	2

V passi

→ considerato già in convergenza

```
DijKstra(G, s, w)
    d = new Array(len(V))
    for each v ∈ V Do
        d[v] = +∞
        π[v] = NIL
    d[s] = 0
    S = ∅ // nodi in convergenze
    for i ← 1 to V-1 Do
        [Prendo il nodo v con stime più piccole in V-S]
        S ← S ∪ {v}
        ∀u ∈ AdJ Do
            relax(v, u, w
```

Con un min-heap

a buono livello:

```
DijKstra(G, s, w)
    d = new Array(len(V))
    for each v ∈ V Do
        d[v] = +∞
        π[v] = NIL
    d[s] = 0
    Q ← build-min-heap(v)
    while Q ≠ ∅ Do
        v ← extractMin(Q)
        Foreach u ADJ(v)
            if d[u] > d[v] + w(v, u) then
                Decrease-Key(Q, u, d[v] + w(v, u))
```

Costi: le righe da "for each" a "d[s] = 0" sono $O(V)$; build-min-heap: $O(V)$; while: $O(V)$; extractMin: $O(\log V)$; Decrease-Key: $O(\log V)$

$$O(V \log V + E \log V) \rightarrow O((V+E)\log V)$$