Capitolo 11

Cammini minimi con sorgente singola

Definizione 11.1. Sia $G = (V, E, w)$ un grafo orientato e pesato; dato il cammino $p = \langle v_0, v_1, ..., v_k \rangle$ in G, il valore $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$ rappresenta il *peso* del cammino.

Dati $u, v \in V$, definiamo:

$$\mathcal{C}(u, v) = \{p : p \text{ è cammino da } u \text{ a } v \text{ in } G\}$$

L'insieme è infinito se il grafo è ciclico (infatti basta eseguire più volte un ciclo per ottenere cammini diversi).

Definizione 11.2. La *distanza* da u a v corrisponde al peso minimo di un cammino da u a v:

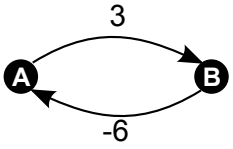
$$\delta(u, v) = \begin{cases} +\infty & \text{se } \mathcal{C}(u, v) = \emptyset \\ \min\limits_{p \in \mathcal{C}(u,v)} w(p) & \text{se } \mathcal{C}(u, v) \neq \emptyset \end{cases}$$

Definizione 11.3. Sia $p = \langle u, ..., v \rangle$: si dice che p è *cammino minimo* se $w(p) = \delta(u, v)$.

Varianti al problema dei cammini minimi

- cammino minimo tra due vertici;
- cammini minimi con sorgente singola;
- cammini minimi con destinazione singola;
- cammini minimi tra tutte le coppie di vertici.

Cicli di peso negativo



L'esistenza dei cicli di peso negativo è problematica nell'ambito della ricerca dei cammini minimi; nell'esempio in figura, *non esiste* un cammino minimo tra a e b: infatti, se supponiamo di aver trovato un cammino minimo, basta eseguire un'altra volta il ciclo e se ne ottiene uno di peso inferiore.

Rappresentazione dei cammini minimi

Ad ogni vertice, associamo un attributo π`[u]`, che rappresenta il precedente di u in un albero di cammini minimi; gli algoritmi aggiornano opportunamente i valori di π in modo che, alla fine, l'albero rappresentato sia un albero di cammini minimi. Definiamo:

Sottografo dei predecessori indotto da π
$G_\pi = (V_\pi, E_\pi)$
$V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$
$E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}\}$

Albero dei cammini minimi: dato $G = (V, E, w)$ orientato e pesato, l'albero dei cammini minimi è $G' = (V', E')$, con $V' \subseteq V$, $E' \subseteq E$ tale che:

- V' è l'insieme dei vertici raggiungibili da s in G;
- G' è albero radicato in s;
- $\forall u \in V'$, l'unico cammino da s a u in G' è un cammino minimo da s a u in G.

11.1 Cammini minimi e rilassamento

Lemma 11.1 (Sottocammini di cammini minimi sono cammini minimi)**.** *Siano $G = (V, E, w)$ un grafo orientato e pesato e $p = \langle v_0, v_1, .., v_k \rangle$ cammino minimo da v_0 a v_k; presi gli indici $0 \leq i < j \leq k$, il sottocammino $p_{ij} = \langle v_i, ..., v_j \rangle$ è cammino minimo da v_i a v_j.*

Dimostrazione. Procediamo per assurdo. Assumiamo che p_{ij} non sia minimo: esisterà dunque un cammino q da v_i a v_j tale che $w(q) < w(p_{ij})$; consideriamo ora il cammino p_{0i}, q, p_{jk}:

$$\begin{aligned} w(p_{0i}, q, p_{jk}) &= w(p_{0i}) + w(q) + w(p_{jk}) \\ &< w(p_{0i}) + w(p_{ij}) + w(p_{jk}) \\ &= w(p) \end{aligned}$$

Ma p è cammino minimo per ipotesi: *assurdo.* Quindi p_{ij} deve essere minimo. □

Proposizione 11.1. *Siano $G = (V, E, w)$ grafo orientato e pesato e $p = \langle v_0, ..., v_k \rangle$ cammino minimo da $u = v_0$ a $v = v_k$; allora $\forall i = 0..k, \delta(u, v) = \delta(u, v_i) + \delta(v_i, v)$.*

Dimostrazione. Segue direttamente dal lemma precedente; sia $p = p_{0i}, p_{ik}$:

$$\begin{aligned} w(p) &= w(p_{0i}) + w(p_{ik}) \\ \delta(u, v) &= \delta(u, v_i) + \delta(v_i, v) \end{aligned}$$

come si voleva dimostrare. □

Corollario 11.1. *Siano $G = (V, E, w)$ un grafo orientato e pesato e p un cammino da s a v scomponibile in un cammino da s a $u \in V$, di nome p' e un arco da u a v; allora il peso di p è pari a $\delta(s, v) = \delta(s, u) + w(u, v)$.*

Lemma 11.2. *Siano $G = (V, E, w)$ un grafo orientato e pesato e $s \in V$ un nodo sorgente; allora, $\forall (u, v) \in E$ si ha $\delta(s, v) \leq \delta(s, u) + w(u, v)$.*

Dimostrazione. Dobbiamo distinguere due casi:

- u non è raggiungibile da s:

$$\delta(s, u) = +\infty \rightarrow \delta(s, v) = \delta(s, u) + w(u, v)$$

- u è raggiungibile da s, dunque esiste un cammino p da s a u; sia q il cammino formato da p e (u, v):

$$\begin{aligned} \delta(s, v) &\leq w(q) \\ &= w(p) + w(u, v) \\ &= \delta(s, u) + w(u, v) \end{aligned}$$

In entrambi i casi la proprietà è valida, dunque il lemma è dimostrato. □

Algoritmi

Gli algoritmi che analizzeremo (Dijkstra, Bellman-Ford) aggiornano due attributi assegnati ai nodi:

`d[u]` : stima del peso del cammino minimo dalla sorgente a u;

π`[u]` : predecessore di u nell'albero dei predecessori.

Al termine vogliamo avere:

- $d[u] = \delta(s, u)$;
- G_π è l'albero dei cammini minimi.

Inizializzazione

funzione `initSingleSource` (*grafo pesato G, sorgente S*) $\rightarrow$ *void*

```
1 for each u ∈ V do
2   d[u] = +∞
3   π[u] = NIL
4 d[s] = 0
```

Rilassamento

funzione `relax` (*nodo u, nodo v, funzione peso w*) $\rightarrow$ *void*

```
1 /* Precondizione: (u, v) in E */
2 if(d[v] > d[u] + w(u, v)) then
3   d[v] = d[u] + w(u, v)
4   π[v] = u
```

Lemma 11.3. *Dopo la chiamata a* `relax`(u, v, w) *si ha che* $d[v] \leq d[u] + w(u, v)$*, qualunque sia l'ordine delle chiamate alla funzione.*

Lemma 11.4. *Siano* $G = (V, E, w)$ *un grafo orientato e pesato e* $s \in V$ *il nodo sorgente; supponiamo che all'inizio sia stata invocata* `initSingleSource`(G, s)*. Allora:*

1. $\forall u \in V$ *si ha* $d[u] \geq \delta(s, u)$*;*
2. *la proprietà al punto 1 rimarrà invariata per qualunque sequenza di* `relax`*;*
3. *nel momento in cui si dovesse verificare che* $d[u] = \delta(s, u)$ *per qualche* $u \in V$*, allora* $d[u]$ *non verrà più modificato da alcuna chiamata di* `relax`*.*

Dimostrazione.

1. Dopo `initSingleSource`(G, s):
   - se $u \neq s, d[u] = +\infty \geq \delta(s, u)$
   - se $u = s, d[u] = 0$
     - se s non sta in un ciclo negativo:
       $$\delta(s, s) = 0$$
       $$d[s] = \delta(s, s)$$
     - se s sta in un ciclo negativo:
       $$\delta(s, s) = -\infty$$
       $$d[s] = 0 \geq \delta(s, s)$$

2. Procediamo per assurdo. Assumiamo che, dopo un certo numero di `relax`, ci sia un vertice v tale che $d[v] < \delta(s,v)$. Supponiamo, inoltre, che v sia il *primo* vertice per cui valga questa proprietà, immediatamente dopo la chiamata `relax`(u,v,w); abbiamo:

$$d[v] = d[u] + w(u,v) < \delta(s,v) \leq \delta(s,u) + w(u,v)$$

e, per la proprietà transitiva:

$$\begin{aligned} d[u] + w(u,v) &< \delta(s,u) + w(u,v) \\ d[u] &< \delta(s,u) \end{aligned}$$

Visto che $d[u]$ non può essere stato modificato da `relax`(u,v,w), dev'essere stato cambiato prima da un'altra chiamata; ma v è stato definito come il primo vertice tale per cui $d[v] < \delta(s,v)$: *assurdo.*

3. Osserviamo che:

   - nessuna `relax`, per definizione, incrementa $d[u]$, ma può solo decrementarlo;
   - $d[u] \geq \delta(s,u)$ dopo ogni `relax`.

   Segue che, dal momento in cui $d[u] = \delta(s,u)$, tale valore non può più cambiare. □

Corollario 11.2. *Se G è inizializzato con `initSingleSource`(G,s) e v **non** è raggiungibile da s, allora $d[v] = +\infty$ non verrà mai aggiornato da alcuna `relax`.*

Lemma 11.5. *Siano dati $G = (V,E,w)$ grafo orientato e pesato e p il cammino minimo da s a v che include (u,v) come arco finale. Supponiamo di inizializzare con `initSingleSource`(G,s) e chiamiamo `relax` più volte; se, ad un certo punto, si ha $d[u] = \delta(s,u)$, allora dopo la chiamata `relax`(u,v,w) si ha $d[v] = \delta(s,v)$.*

Dimostrazione. Prima di `relax`(u,v,w) vale $\delta(s,v) \leq d[v]$ per il lemma 11.4 e che $d[u] = \delta(s,u)$ per ipotesi. Dopo `relax`(u,v,w) si ha che:

$$\begin{aligned} d[v] &\leq d[u] + w(u,v) \\ &= \underbrace{\delta(s,u)}_{w(p')} + w(u,v) \\ &= w(p') + w(u,v) \\ &= w(p) \\ &= \delta(s,v) \end{aligned}$$

dove p' è cammino minimo da s a u essendo sottocammino di p. Dal lemma 11.4 si ha che $\delta(s,v) \leq d[v]$, da cui segue $d[v] = \delta(s,v)$. □

11.2 Algoritmo di Dijkstra

algoritmo `dijkstra` (*grafo pesato G, sorgente s*) → *albero dei cammini minimi*

```
1  /* Precondizione: pesi strettamente non negativi */
2  initSingleSource(G, s)
3  S = ∅
4  Q = V
5  while(Q != ∅) do
6    u = extractMin(Q)
7    S = S ∪ {u}
8    for each v ∈ Adj[u] do
9      relax(u, v, w)
10 return (d, G_π)
```

Complessità

Sia Q implementato come heap binario:

2. $O(n)$
3. $O(1)$
4. $O(n)$

5-9. il ciclo viene eseguito n volte:

   6. ogni estrazione costa $O(\log n)$, per un totale di $O(n \cdot \log n)$
   7. $O(1)$, per un totale di $O(n)$
   8-9. chiama `relax` una volta per ogni arco del grafo, quindi si ha un costo totale di $O(m \cdot \log n)$; ogni chiamata, infatti, può modificare la chiave di priorità, operazione dal costo $O(\log n)$

Totale: $3 \cdot O(n) + O(1) + O(n \cdot \log n) + O(m \cdot \log n) = O((n + m) \cdot \log n)$.

Sia Q implementato come array lineare:

2. $O(n)$
3. $O(1)$
4. $O(n)$

5-9. il ciclo viene eseguito n volte:

   6. bisogna scorrere ogni volta l'intero array, per un totale di $O(n^2)$
   7. $O(1)$, per un totale di $O(n)$
   8-9. chiama `relax` una volta per ogni arco del grafo, che ha costo costante; in totale, dunque, il costo è $O(m)$

Totale: $3 \cdot O(n) + O(1) + O(n^2) + O(m) = O(n^2 + m)$.

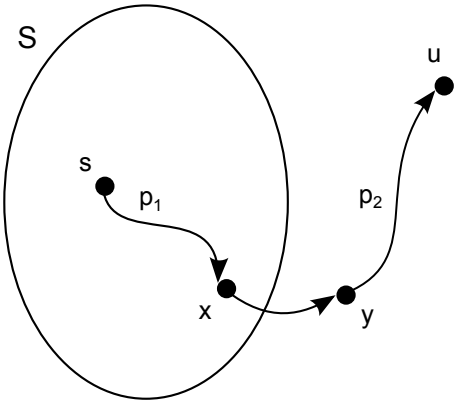
La prima organizzazione risulta migliore nel caso il grafo sia *sparso* ($m = O(n)$), mentre la seconda è conveniente se il grafo è denso ($m = O(n^2)$).

Correttezza

Teorema 11.1. *Sia $G = (V, E, w)$ un grafo orientato e pesato con $w(u, v) \geq 0, \forall (u, v) \in E$. Allora, al termine dell'esecuzione dell'algoritmo, si ha:*

1. *$d[u] = \delta(s, u), \forall u \in V$;*
2. *G_π è l'albero dei cammini minimi.*

Dimostrazione. Dimostriamo solo il primo punto, in quanto il secondo segue da `relax`.
Proviamo, comunque, una proprietà più forte: $\forall u \in V$, nel momento in cui u è inserito in S, vale $d[u] = \delta(s, u)$ (e per il lemma 11.4, non diminuirà ulteriormente).
Procediamo per assurdo: assumiamo che u sia il primo vertice tale che $d[u] \neq \delta(s, u)$ nel momento in cui viene inserito in S. Dopo `initSingleSource(`G, s`)` si ha $d[s] = 0$; nel grafo non ci sono cicli negativi, poiché i pesi sono tutti non negativi, dunque si ha $\delta(s, s) = 0$: abbiamo dunque $d[s] = \delta(s, s)$, quindi il vertice u non può essere s.
Visto che s viene estratto per primo, $S \neq \emptyset$ quando u viene inserito; può essere $\delta(s, u) = +\infty$? Dopo `initSingleSource`, $d[u] = +\infty$, quindi ancora prima di iniziare le iterazioni si ha $d[u] = \delta(s, u)$; nelle iterazioni si eseguono delle chiamate a `relax` che, però, non modificano $d[u]$: segue, dunque, che u dev'essere raggiungibile da s, ossia $\delta(s, u) < +\infty$. Sia p il cammino minimo da s a u; la situazione che si ha quando viene estratto u da Q è:



$$Q = V - S$$

Sia (x, y) arco di p tale che $x \in S$ e $y \in Q$. Si ha:

$d[u] \leq d[y]$, perché u estratto da Q che contiene anche y;

$$d[u] = \min_{v \in Q} d[v]$$

$d[x] = \delta(s, x)$ perché u è il primo vertice per cui non vale $d[u] = \delta(s, u)$ e x è inserito in S prima di u. Inoltre, quando è stato inserito x, è stata chiamata `relax`(x, y, w); dopo questa chiamata, $d[y] = \delta(s, y)$ per il lemma 11.5.

$\delta(s, y) \leq \delta(s, u)$ poiché i pesi sono non negativi.

$\delta(s, u) \neq d[u]$ per ipotesi e $d[u] \geq \delta(s, u)$ per il lemma 11.4; segue che $d[u] > \delta(s, u)$.

Da tutte queste proprietà, segue:

$$\begin{aligned} d[u] &\leq d[y] \\ &= \delta(s, y) \\ &\leq \delta(s, u) \\ &< d[u] \end{aligned}$$

Si ottiene dunque un *assurdo*: il vertice u che cercavamo non esiste e la correttezza è dimostrata.

Esempio di esecuzione

Per ciascun nodo, è indicata la coppia (*chiave, predecessore*) associata ad ogni iterazione del ciclo; i nodi marcati in nero sono quelli ancora presenti in Q, il nodo rosso è quello estratto nell'iterazione considerata e quelli blu sono i nodi già estratti da Q ed esaminati. In ogni figura, sono evidenziati gli archi che compongono il sottografo indotto da π.

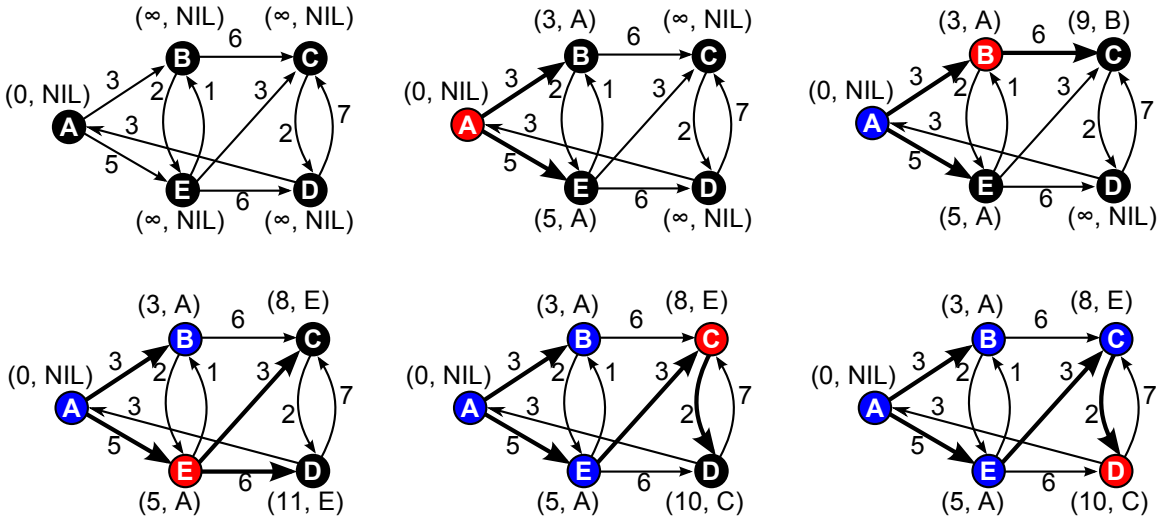


Figura 11.1: Simulazione dell'esecuzione dell'algoritmo di Dijkstra (nodo sorgente a)

11.3 Algoritmo di Bellman-Ford

L'algoritmo di Dijkstra presenta dei problemi nel caso in cui siano presenti dei cicli di peso negativo: per risolverli, è stato proposto l'algoritmo di *Bellman-Ford*.

algoritmo `bellmanFord` (*grafo pesato G, sorgente s*) → *booleano*

```
1 initSingleSource(G, s)
2 for i = 1 to |V| - 1 do
3   for each (u, v) ∈ E do
4     relax(u, v, w)
5 for each (u, v) ∈ E do
6   if(d[v] > d[u] + w(u, v)) then
7     return false
8 return true
```

L'algoritmo restituisce:

- *false*, se è stato trovato un ciclo negativo;
- *true* altrimenti.

Osservazione: l'algoritmo ritorna *true* se e solo se $d[v] \leq d[u] + w(u,v), \forall (u,v) \in E$.

Correttezza

Teorema 11.2. *Siano $G = (V, E, w)$ un grafo orientato e pesato e $s \in V$ il nodo sorgente:*

1. *se il grafo non contiene cicli negativi raggiungibili da s allora, alla fine di `bellmanFord(G, s)` si ha:*
   (a) *per ogni $u \in V$ risulta $d[u] = \delta(s,u)$;*
   (b) *G_π è un albero di cammini minimi;*
   (c) *l'algoritmo ritorna true;*
2. *se il grafo G contiene cicli negativi raggiungibili da s, l'algoritmo ritorna false.*

Dimostrazione.

Punto 1a. Per ipotesi, G non ha cicli negativi raggiungibili da s. Sia $u \in V$; dobbiamo provare che vale $d[u] = \delta(s,u)$ al termine dell'esecuzione dell'algoritmo:

1. se u non è raggiungibile da s, si ha $\delta(s,u) = +\infty$. Dopo `initSingleSource(G, s)` vale $d[u] = +\infty = \delta(s,u)$ e, poiché `relax` non modifica ulteriormente $d[u]$ per il lemma 11.4, al termine vale ancora $d[u] = \delta(s,u)$;
2. se u è raggiungibile da s, si ha $\delta(s,u) < +\infty$. Sia $p = \langle x_0, ..., x_k \rangle$ cammino minimo da s a u, con $x_0 = s$ e $x_k = u$; p non ha cicli e, inoltre, il numero di nodi di p, ossia $k+1$, è inferiore o uguale alla cardinalità di V. Consideriamo il predecessore di u nel cammino e chiamiamolo v: se p è cammino minimo e $d[v] = \delta(s,v)$, dopo `relax(v, u, w)` vale $d[u] = \delta(s,u)$.

   Proprietà 11.1. Se esiste un cammino minimo da s a u con k archi, allora dopo il k-esimo ciclo dell'algoritmo (righe 2-4) vale $d[u] = \delta(s,u)$ (*invariante del ciclo*).

   Dimostrazione. Procediamo per induzione su k.

   Caso base: abbiamo $k = 0$, dunque $u = s$; si ha $d[s] = 0$ per `initSingleSource` e $\delta(s,s) = 0$ per l'ipotesi dell'assenza di cicli negativi, dunque la proprietà è dimostrata per il caso base.

   Ipotesi induttiva: supponiamo che la proprietà valga per $i < k$.

   Passo induttivo: sia $k > 0$; consideriamo il cammino $p = \langle x_0, ..., x_k \rangle$, con $x_0 = s$ e $x_k = u$. Per ipotesi induttiva, alla $k-1$-esima iterazione, $d[x_{k-1}] = \delta(s, x_{k-1})$; all'iterazione k-esima, tra le varie chiamate, viene eseguita `relax(`x_{k-1}, x_k, w`)` e, per il lemma 11.5, si ha $d[x_k] = \delta(s, x_k)$, ossia $d[u] = \delta(s,u)$. □

   Dunque, al termine del ciclo, tutti i cammini sono stati sistemati (se non ci sono cicli negativi) e hanno al più $n-1$ archi.

Punto 1b. Segue direttamente da `relax`.

Punto 1c. Dobbiamo dimostrare che $d[v] \leq d[u] + w(u,v), \forall (u,v) \in E$. Per il lemma 11.2, si ha $\delta(s,v) \leq \delta(s,u) + w(u,v)$; consideriamo $(u,v) \in E$:

$$\begin{aligned} d[v] &= \delta(s,v) \\ &\leq \delta(s,u) + w(u,v) \\ &= d[u] + w(u,v) \end{aligned}$$

L'ultima uguaglianza segue dal punto 1a.

Punto 2. Dal punto 1 sappiamo che, se non ci sono cicli negativi raggiungibili da s, allora l'algoritmo ritorna *true*. Proviamo ora il viceversa, ossia che, se l'algoritmo ritorna *true*, non ci sono cicli negativi raggiungibili da s.
Sia $C = \langle x_0, ..., x_k \rangle$ un ciclo raggiungibile da s; poiché, per ipotesi, l'algoritmo ritorna true, si ha:

$$d[v] \leq d[u] + w(u,v)$$

In particolare, considerando gli archi (x_i, x_{i+1}) con $i = 0..k-1$:

$$\begin{aligned} & d[x_{i+1}] \leq d[x_i] + w(x_i, x_{i+1}) \\ & \sum_{i=0}^{k-1} d[x_{i+1}] \leq \sum_{i=0}^{k-1} d[x_i] + \sum_{i=0}^{k-1} w(x_i, x_{i+1}) \\ & d[x_1] + d[x_2] + ... + d[x_k] \leq d[x_0] + d[x_1] + ... + d[x_{k-1}] + \sum_{i=0}^{k-1} w(x_i, x_{i+1}) \\ & d[x_k] \leq d[x_0] + \sum_{i=0}^{k-1} w(x_i, x_{i+1}) \end{aligned}$$

Poiché C è un ciclo, si ha $x_0 = x_k$, ossia $d[x_0] = d[x_k]$; segue:

$$0 \leq \sum_{i=0}^{k-1} w(x_i, x_{i+1}) = w(C)$$

Abbiamo così dimostrato che, se l'algoritmo ritorna *true*, non esistono cicli negativi raggiungibili da s; segue dunque che, se ci sono cicli negativi raggiungibili da s, l'algoritmo restituisce *false*. □

Complessità

1. $O(n)$

2-4. vengono eseguite $m \cdot n$ chiamate a `relax`, il cui costo è $O(1)$; in totale si ha $O(m \cdot n)$

5-7. il ciclo viene eseguito m volte ed ogni iterazione ha costo $O(1)$; in totale si paga è $O(m)$

8. $O(1)$

Totale: $O(n) + O(m \cdot n) + O(m) + O(1) = O(m \cdot n)$.

L'algoritmo risulta essere computazionalmente più costoso rispetto a quello di Dijkstra, ma può essere applicato anche se gli archi hanno pesi negativi: la scelta di quale algoritmo utilizzare, dunque, dipende da come è definita la funzione peso.

Esempio di esecuzione

L'ordine di analisi degli archi segue l'ordine alfabetico con cui sono identificati i vertici: prima tutti gli archi che partono da a, poi quelli che partono da b, etc. Le figure si riferiscono a:

1. situazione dopo `initSingleSource`;
2. situazione dopo la prima passata di tutti gli archi;
3. situazione dopo la seconda passata di tutti gli archi (le successive non producono variazioni).

Chiaramente l'algoritmo restituisce *true*.

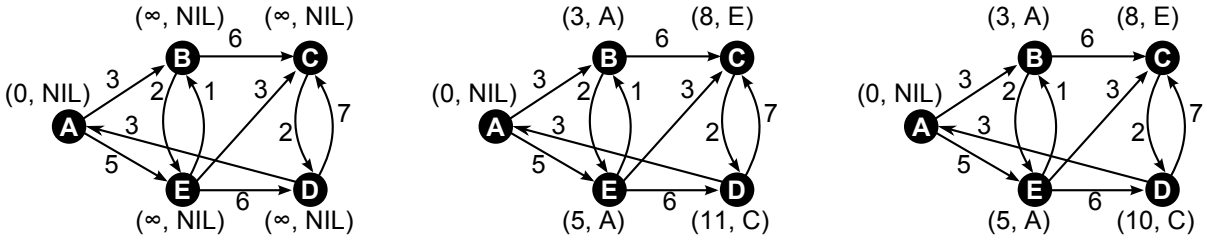


Figura 11.2: Esempio di esecuzione dell'algoritmo di Bellman - Ford (nodo sorgente a)