Progettazione di Algoritmi **Anno Accademico 2021–2022**

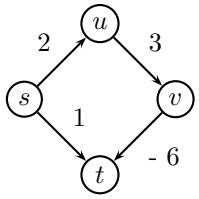
Note per la Lezione 29

Ugo Vaccaro

In questa lezione studieremo ancora problemi su cammini minimi, ma in grafi in cui vi possono essere archi di "costo negativo". Quindi, il problema che studieremo é il seguente:

Input: Grafo diretto $G = (V, E)$, costi $c(u, v)$ per ogni arco $(u, v) \in E$, nodi $s, t \in V$
Output: Cammino di costo totale minimo da s a t

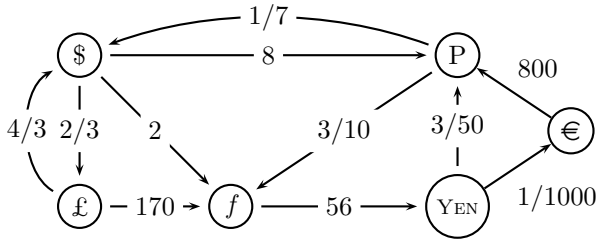
Il problema è ovviamente già risolto dall'algoritmo di Dijkstra nel caso in cui $c(u, v) \geq 0$, per ogni arco $(u, v) \in E$. Che succede se usiamo l'algoritmo di Dijkstra anche nel caso in cui $c(u, v) < 0$ per qualche arco $(u, v) \in E$? Che l'algoritmo sbaglia. Vediamo il seguente esempio:



L'algoritmo di Dijkstra sceglierebbe il cammino fatto dal solo arco (s, t) di costo 1, ma il cammino da s a t che passa per u e v ha costo totale inferiore, pari -1.

E perchè vale la pena di considerare grafi con archi di costo negativo? Perchè il linguaggio dei grafi e i relativi cammini minimi ad esso associati possono rappresentare anche questioni in cui ha molto senso considerare grafi con archi di costo negativo.

Esempio: Consideriamo un grafo i cui nodi rappresentano le valute ufficiali di nazioni (Euro, dollari, sterline, etc.). Per ogni coppia di valute i, j denotiamo con $t(i, j)$ il tasso di cambio della valuta i nella valuta j. Ad esempio, i tassi di cambio per alcune valute potrebbero essere rappresentati dal grafo di sotto, con costi $t(i, j)$ sugli archi



Nello scenario sopra descritto, si potrebbe cambiare un'unità di f (fiorini) in 56 YEN, da cui ottenere $56 \times (3/50)$ P (Peso messicano) ed infine $(3/10) \times 56 \times (3/50) = 1.008f$. Detto in altri termini, partendo da *un* fiorino, e cambiandolo prima in YEN, poi in Peso messicano, e poi di nuovo in fiorini, otterremmo 1.008 fiorini Se ripetiamo i cambi un'altra volta, otterremmo $(1.008)^2$ fiorini, e se ripetiamo n volte otterremmo $(1.008)^n$. Osservando che $(1.008)^n \to \infty$, per $n \to \infty$, avremmo trovato una sicura fonte di reddito. Situazioni del genere possono accadere in pratica per *brevi* periodi di tempo ed in finanza prendono il nome di *arbitraggio*. Gli speculatori tendono a farne ampio uso (perciò è critico scoprire una tale situazione velocemente e prima degli altri...).

Interessante, diranno gli speculatori che leggono queste note, ma gli archi con costo negativo che c'entrano? Lo vediamo... In generale, in un grafo come prima descritto (con costi associati agli archi pari ai tassi di cambio

$t(i,j)$), un cammino dal vertice i al vertice j che passa attraverso i vertici $a_1, \ldots, a_k$ rappresenterebbe un possibile modo per cambiare la valuta i nella valuta j, effettuando i cambi intermedi $i \to a_1$, $a_1 \to a_2, \ldots, a_k \to j$. Uno speculatore vorrebbe evidentemente il cammino nel grafo che *massimizza il prodotto* dei tassi di cambio $t(i,a_1) \times t(a_1,a_2) \times \cdots \times t(a_k,j)$, ovvero che massimizza il prodotto dei costi associati agli archi del cammino. Se poi nel grafo ci fosse addirittura un ciclo che, partendo dalla valuta i ed attraverso una serie di cambi di valuta permettesse di riavere la valuta i con la condizione che $t(i,a_1) \times t(a_1,a_2) \times \cdots \times t(a_k,i) > 1$, uno speculatore vorrebbe CERTAMENTE saperlo, trovarlo, e farne buon uso... Ma come si fà? Noi sappiamo trovare in un grafo i cammini che *minimizzano la somma* dei costi associati agli archi del cammino! Ci arriviamo...

• Massimizzare $t(i,a_1) \times t(a_1,a_2) \times \cdots \times t(a_k,j)$ è perfettamente equivalente a massimizzare $\log[t(i,a_1) \times t(a_1,a_2) \times \cdots \times t(a_k,j)]$

• Massimizzare $\log[t(i,a_1) \times t(a_1,a_2) \times \cdots \times t(a_k,j)]$ è perfettamente equivalente a *minimizzare* $-\log[t(i,a_1) \times t(a_1,a_2) \times \cdots \times t(a_k,j)] = -\log t(i,a_1) - \log t(a_1,a_2) - \cdots - \log t(a_k,j)$

Ne segue che il cammino del grafo G con costi sugli archi $t(i,j)$, che massimizza $t(i,a_1) \times t(a_1,a_2) \times \cdots \times t(a_k,j)$ è lo *stesso cammino* che nel grafo G' con costi sugli archi $c(i,j) = -\log t(i,j)$ ha costo totale (=somma dei costi degli archi che lo costituiscono) *minimo*.

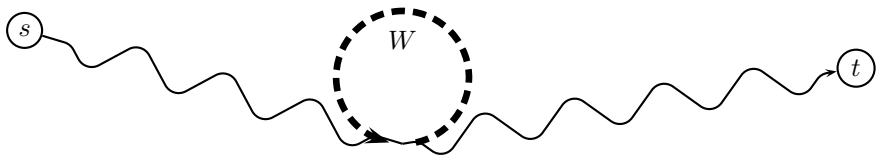
Ma questo nuovo grafo può adesso avere costi $c(i,j) = -\log t(i,j)$ sugli archi anche di valore *negativo*, quindi se vogliamo speculare sui cambi tra le valute dobbiamo saper trovare cammini di costo minimo in grafi con costi sugli archi arbitrari (ovvero sia positivi che negativi).

In generale vi sono molti problemi importanti la cui soluzione è equivalente a trovare cammini minimi in grafi con costi sia negativi che positivi sugli archi. Restando in ambito finanziario (ma si può andare ben oltre), potremmo avere delle transazioni in cui compriamo da un agente i e vendiamo ad un agente j ad un costo $c(i,j)$ che, se negativo, costituirebbe di fatto per noi un profitto. Di nuovo potremmo in tale ambito avere il problema di determinare la sequenza di compravendite con punto di inizio e fine fissati che minimizza i costi (ovvero massimizza il nostro guadagno).

La struttura delle soluzioni al problema di determinare cammini minimi in grafi con costi sia negativi che positivi sugli archi tende ad essere molto diversa dalle soluzioni relative allo stesso problema in grafi con costi sugli archi solo positivi. Nel primo caso le soluzioni tendono ad essere "cammini corti", nel secondo tendono ad essere "cammini lunghi" (il che fà anche capire perchè occorrono tecniche nuove) Una prima considerazione sui cicli di costo negativo:

Se nel grafo $G = (V,E)$ *esiste* un cammino dal nodo s al nodo t che contiene un ciclo di costo totale negativo, allora *NON esiste* alcun cammino di costo totale minimo da s a t. Viceversa, se ogni cammino da s a t *NON contiene* alcun ciclo di costo totale negativo, allora *esiste* un cammino di costo totale minimo da s a t ed esso contiene al più $n-1$ archi ($n = |V|$)

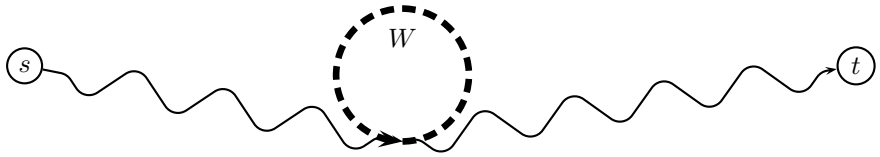
Infatti, se in un cammino da s da t esistesse un ciclo W di costo totale $c(W) < 0$, del tipo rappresentato nella figura di seguito riportata



allora ripassando *sempre più volte* sul ciclo W potremmo abbassare sempre di più il costo del percorso da s a t,

senza mai fermarci ad un minimo.

Viceversa, (e ora sotto l'ipotesi che ogni cammino da s a t *NON contiene* alcun ciclo di costo totale negativo) se per assurdo un cammino minimo da s a t contenesse *più* di $n-1 = |V|-1$ archi, allora in tale cammino almeno un vertice compare due volte (perchè? Perchè se il cammino contiene più di $n-1$ archi allora contiene *più* di n vertici, e quindi almeno un vertice compare due volte). Quindi nel cammino è presente un ciclo W



Per ipotesi il costo $c(W)$ di tale cammino è > 0; se lo eliminiamo possiamo andare da s a t con un costo inferiore, contro l'ipotesi che il cammino era di costo minimo.

Cammini minimi via Programmazione Dinamica.

Sia $OPT(i,v)$ =minimo costo di un cammino da v a t che usa al più i archi.

Possiamo avere vari casi:

Caso 1: il cammino minimo P da v a t usa al più $i-1$ archi $\Rightarrow OPT(i,v) = OPT(i-1,v)$

Caso 2: il cammino minimo P da v a t usa esattamente i archi.
Se P usa l'arco (v,w) come primo arco, allora il costo di P è pari a $c(v,w)$+costo del cammino minimo da w a t che usa al più $i-1$ archi, per cui

$$OPT(i,v) = c(v,w) + OPT(i-1,w).$$

Poiché in generale, non sappiamo chi é il primo arco del cammino minimo, dobbiamo calcolarci

$$\min_{w \in V}\{c(v,w) + OPT(i-1,w)\}.$$

Riassumendo, e tenedo conto delle condizioni iniziali, avremo:

$$OPT(i,v) = \begin{cases} 0 & \text{se } v = t \\ \min\left\{OPT(i-1,v), \min_{w\in V}\{c(v,w) + OPT(i-1,w)\}\right\} & \text{se } i > 0 \; v \neq t \\ \infty & \text{se } i = 0, v \neq t \end{cases} \quad (1)$$

In virtù della seconda parte della considerazione sui cicli di costo negativo, a noi interessa $OPT(n-1,s)$.

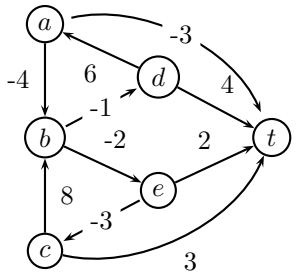
Vediamo un algoritmo per il calcolo della (**??**)

```
Shortest-Path(G, s, t)
1.  For ogni nodo v ∈ V
2.      M[0, v] ← ∞
```

```
3.  M[0,t] ← 0
4.  For i = 1 to n − 1
5.     For ogni nodo v ∈ V
6.        M[i,v] ← M[i − 1,v]
7.        For ogni arco (v,w) ∈ E
8.           If M[i,v] > M[i − 1,w] + c(v,w)
9.              M[i,v] ← M[i − 1,w] + c(v,w)
10. return M[n − 1,s]
```

Analisi: L'inizializzazione 1.-3. prende tempo $O(n)$, $n = |V|$. Nel `For` 5. si analizza ogni vertice ed ogni arco ad esso adiacente (ovvero tutti gli archi del grafo), per un tempo totale $O(m)$, con $m = |E|$. Quindi le istruzioni 4.-9. prendono tempo $O(nm)$ e spazio $O(n^2)$ (per la matrice $M[\cdot,\cdot]$.)

Esempio (tenendo a mente la (**??**):



$$
\begin{aligned}
OPT(1,a) &= \min\{\infty,-3\} = -3\\
OPT(1,b) &= \min\{\infty,\infty\} = \infty\\
OPT(1,c) &= \min\{\infty,3\} = 3\\
OPT(1,d) &= \min\{\infty,4\} = 4\\
OPT(1,e) &= \min\{\infty,2\} = 2\\
OPT(2,a) &= \min\{-3,-4+\infty\} = -3\\
OPT(2,b) &= \min\{\infty,\min\{-1+4,-2+2\}\} = 0\\
OPT(2,c) &= \min\{3,\min\{8+\infty\}\} = 3\\
OPT(2,d) &= \min\{4,\min\{6-3\}\} = 3\\
OPT(2,e) &= \min\{2,\min\{3-3\}\} = 0\\
OPT(3,a) &= \min\{-3,\min\{-4+0\}\} = -4\\
&\vdots
\end{aligned}
$$

Miglioramenti dell'implementazione. Non è necessario ricordarsi del valore $M[i,v]$ per ogni i, ma solo dell'ultimo valore, ed aggiornarlo se esso diminuisce. Per memorizzare i valori $M[v]$ avremo ora bisogno solo di spazio $O(n)$.

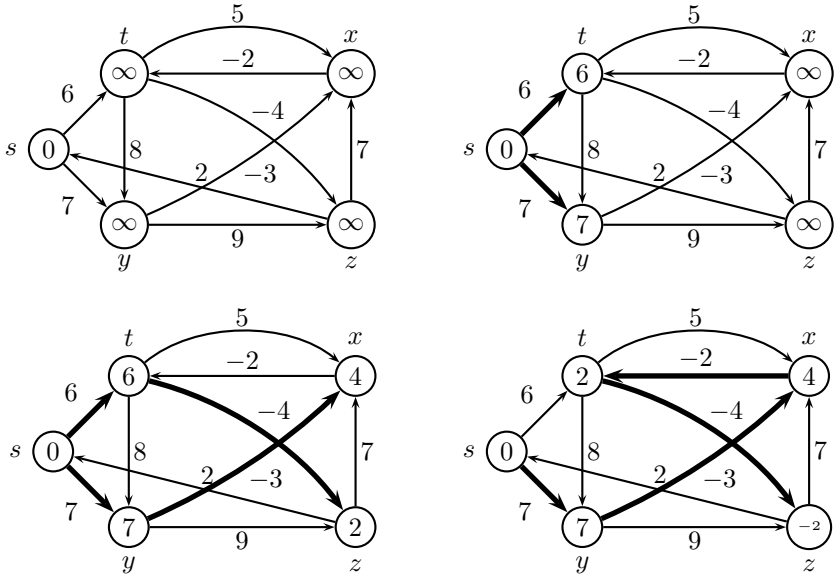
```
Shortest-Path(G,s,t)
1.  For ogni nodo v ∈ V
2.     ~~M[0,v] ← ∞~~     M[v] ← ∞
3.  ~~M[0,t] ← 0~~     M[t] ← 0
4.  For i = 1 to n − 1
5.     For ogni nodo v ∈ V
6.        ~~M[i,v] ← M[i − 1,v]~~
7.        For ogni arco (v,w) ∈ E
8.           If ~~M[i,v] > M[i − 1,w] + c(v,w)~~
                 M[v] > M[w] + c(v,w)
9.              ~~M[i,v] ← M[i − 1,w] + c(v,w)~~
                 M[v] ← M[w] + c(v,w)
10. return ~~M[n − 1,s]~~     M[s]
```

Nell'iterazione generica i del `For 4.`, ha senso fare il controllo $M[v] > M[w] + c(v,w)$ solo se il valore di $M[w]$ è diminuito nella precedente iterazione $i-1$, altrimenti sprechiamo solo tempo. Se in una iterazione completa del `For 4.` *nessun* valore $M[w]$ cambia, allora non cambierà neanche nella iterazione successiva e tanto vale allora terminare l'algoritmo.

E se vogliamo i cammini minimi e non solo i loro costi? Basterà memorizzare, per ogni nodo, un puntatore `first`$[v]$ che contiene il primo nodo (dopo v) del cammino di costo minimo da v a t. Tale puntatore andrà aggiornato ogni qualvolta il costo $M[v]$ cambia (ovvero dminuisce) in quanto abbiamo trovato un cammino migliore. Facendolo, otteniamo:

```
Shortest-Path(G, s, t)
1.  For ogni nodo v ∈ V
2.      M[v] ← ∞,    first[v] ← NIL
3.  M[t] ← 0
4.  For i = 1 to n − 1
5.     For ogni nodo w ∈ V
6.      If M[w] è cambiato nella precedente iterazione
7.       For ogni v ∈ V tale che (v, w) ∈ E
8.           If M[v] > M[w] + c(v, w)
9.             M[v] ← M[w] + c(v, w)    first[v] ← w
10.      If nessun M[w] è cambiato nell'iterazione i Stop
11.  return M[s]
```

Esempio:



Qualche considerazione sull'algoritmo (detto di Bellman e Ford)

Sia P il grafo con nodi V ed archi $(v, \texttt{first}[v])$, per $v \in V$. Se P contenesse un ciclo C allora tale ciclo avrebbe costo totale negativo.

Notiamo innanzitutto che durante tutti gli istanti dell'esecuzione dell'algoritmo in cui vale vale che `first`$[v] = w$,

vale anche $M[v] \geq c(u,w) + M[w]$.

Infatti, quando viene effettuata l'assegnazione $\texttt{first}[v] \leftarrow w$ viene anche posto $M[v] \leftarrow c(u,w) + M[w]$. In tempi successivi il valore $M[w]$ potrebbe cambiare (solo diminuendo, ricordate!) e quindi in generale $M[v] \geq c(u,w) + M[w]$. Siano $v_1, v_2, \ldots, v_k$ i nodi del ciclo C in P e sia (v_k, v_1) l'ultimo arco aggiunto.

Consideriamo i valori delle variabili immediatamente prima di tale aggiunta. Per quanto prima detto vale che $M[v_i] \geq c(v_i, v_{i+1}) + M[v_{i+1}]$, per $i = 1, \ldots k-1$. Poichè stiamo per aggiungere l'arco (v_k, v_1), ovvero stiamo per assegnare $\texttt{first}[v_k] \leftarrow v_1$, ciò vuol dire che vale $M[v_k] > c(v_k, v_1) + M[v_1]$. Addizionando tutti i termini otteniamo

$$\sum_{i=1}^{k} M[v_i] > \sum_{i=1}^{k-1} (c(v_i, v_{i+1}) + M[v_{i+1}]) + c(v_k, v_1) + M[v_1]$$

da cui $0 > \sum_{i=1}^{k-1} c(v_i, v_{i+1}) + c(v_k, v_1)$, ovvero la somma del costo degli archi di C è < 0.

Noi siamo nell'ipotesi che G non contiene cicli di costo negativo. Pertanto, dalla considerazione precedente, il grafo P non contiene cicli. E che contiene? Cammini di costo totale minimo da ogni vertice a t.

Inanzitutto osserviamo che P contiene cammini da ogni vertice $v \in V$ a t. Infatti, seguendo la successione dei vertici $\texttt{first}[v] = v_1$, $\texttt{first}[v_1] = v_2$, e così via, visto che non ci sono ripetizioni di vertici (ovvero non ci sono cicli in P), prima o poi ci dobbiamo fermare e l'unico vertice in cui ci possiamo fermare è quello con $\texttt{first}[\cdot] = NIL$, ovvero il nodo t in quanto, avendo esso $M[t] = 0$ fin dall'inizio, mai vedrà tale valore cambiare e mai assegneremo $\texttt{first}[t]$ ad un valore diverso da NIL.

Sia allora $v = v_1, v_2, \ldots, v_k = t$ il cammino in P da un nodo v a t (ovvero $\texttt{first}[v_1] = v_2$, $\texttt{first}[v_2] = v_3$, ... , $\texttt{first}[v_{k-1}] = v_k$) e calcoliamone il costo $\sum_{i=1}^{k} c(v_i, v_{i+1})$.

Calcolo del costo $\sum_{i=1}^{k} c(v_i, v_{i+1})$.

Alla fine dell'algoritmo, se abbiamo che $\texttt{first}[v_i] = v_{i+1}$ vuol dire che $M[v_i] = c(v_i, v_{i+1}) + M[v_{i+1}]$, ovvero $c(v_i, v_{i+1}) = M[v_i] - M[v_{i+1}]$. Sommando, otteniamo

$$\sum_{i=1}^{k} c(v_i, v_{i+1}) = (M[v_1] - M[v_2]) + (M[v_2] - M[v_3]) + \ldots + (M[v_{k-1}] - M[v_k])$$

ovvero

$$\sum_{i=1}^{k} c(v_i, v_{i+1}) = M[v_1] - M[v_k] = M[v] - M[t] = M[v]$$

da cui, ricordando che $M[v]$ era proprio il costo del cammino minimo da v a t, otteniamo il risultato.

Cicli di costo negativo in grafi.

Vi sono due naturali problemi che sorgono al riguardo:

1. Come scoprire se un grafo contiene cicli di costo negativo?
2. Come trovare un ciclo di costo negativo in un grafo che ne contiene?

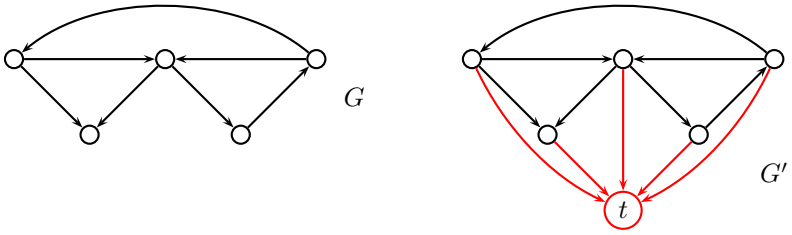
Il problema 1. è motivato dal fatto che se un grafo contiene cicli di costo negativo lungo il cammino ad un vertice destinazione t, allora come abbiamo visto in precedenza non ha senso parlare di cammini di costo minimo che portano a t.

Il problema 2. è motivato dal fatto che in certe situazioni (ricordate l'arbitraggio?) si può trarre profitto dall'esistenza di cicli di costo negativo in grafi.

Cicli di costo negativo in grafi

Osserviamo innanzitutto che se sappiamo decidere, per ogni grafo G:
a) In G esiste o meno un ciclo di costo totale negativo da cui è possibile raggiungere un vertice t?
allora possiamo anche decidere in generale:
b) In G esiste o meno un ciclo di costo totale negativo?

Supponiamo infatti di sapere risolvere **a)** e, a partire dal grafo $G = (V, E)$ in cui vorremmo invece risolvere **b)**, ci costruiamo un nuovo grafo $G' = (V', E')$ con $V' = V \cup \{t\}$, $E' = E \cup \{(v, t) : v \in V\}$, con costo$(v, t) = 0$, $\forall v \in V$.



$G' = (V', E')$ $V' = V \cup \{t\}$, $E' = E \cup \{(v, t) : v \in V\}$, costo$(v, t) = 0$, $\forall v \in V$.

- Se G contiene un ciclo negativo, allora G' ne contiene sicuramente uno da cui è possibile raggiungere t (visto che in G' tutti i vertici possono raggiungere t) e noi quest'ultimo fatto abbiamo assunto lo sappiamo scoprire.
- Se G' ha un ciclo negativo con un cammino a t, tale ciclo non può contenere t (visto che da t non si và da nessuna parte). Ciò vuol dire che se scopriamo che G' ha un ciclo negativo, allora vuol dire in realtà che tale ciclo lo ha anche G.

Come scoprire se un grafo contiene cicli di costo negativo?

In G non esistono cicli di costo negativo da cui si può raggiungere t se e solo se vale che $OPT(n, v) = OPT(n - 1, v)$ per ogni nodo v.

Ricordiamoci cosa abbiamo provato qualche minuto fà:

- Se ogni cammino da s a t NON contiene alcun ciclo di costo totale minimo, allora esiste un cammino di costo totale minimo da s a t e contiene al più $n - 1$ archi ($n = |V|$)

Ciò immediatamente implica che $OPT(n, v) = OPT(n - 1, v)$ in quanto l'aggiunta di più archi non può evidentemente portare a cammini di costo totale inferiore al costo del cammino di minimo costo.

Viceversa, assumiamo che $OPT(n, v) = OPT(n - 1, v)$ per ogni nodo v.

I valori $OPT(n+1, v)$ vengono calcolati da $OPT(n, v)$ attraverso la formula

$$OPT(n+1, v) = \min\left\{OPT(n, v), \min_{w \in V}\{c(v, w) + OPT(n, w)\}\right\}$$

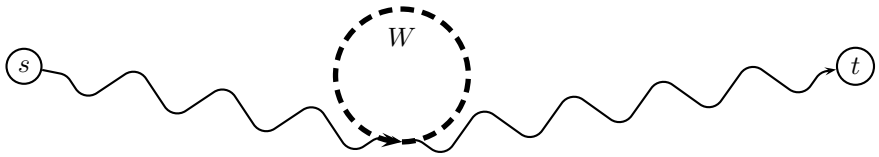
ma dall'assunzione abbiamo che

$$OPT(n+1, v) = \min\left\{OPT(n-1, v), \min_{w \in V}\{c(v, w) + OPT(n-1, w)\}\right\} = OPT(n, v) = OPT(n-1, v)$$

e così via per provare che $OPT(i, v) = OPT(n-1, v), \forall i \geq n$.

Quindi $OPT(n, v) = OPT(n-1, v) \Rightarrow OPT(i, v) = OPT(n-1, v), \forall i \geq n$.

Se per assurdo in un cammino da v da t esistesse un ciclo W di costo totale $c(W) < 0$



allora ripassando *sempre più volte* sul ciclo W potremmo abbassare sempre di più il costo del percorso da s a t, ovvero *per qualche* $i > n$ varrebbe $OPT(i, v) < OPT(n-1, v)$, raggiungendo una contraddizione.

Algoritmo per scoprire se G ha un ciclo di costo < 0:

- Esegui l'algoritmo di Bellman e Ford (nella prima versione di complessità $O(mn)$) per calcolare i valori $OPT(i, v)$, $\forall v \in V$ e per $i = 1, \ldots, n$.
- Dichiara che non esistono cicli di costo negativo in G se e solo se esiste qualche valore $i \leq n$ per cui $OPT(i, v) = OPT(i-1, v)$ $\forall v \in V$.

Algoritmo per trovare in G un ciclo di costo < 0 (se G lo ha)

- Esegui l'algoritmo di Bellman e Ford (nella prima versione di complessità $O(mn)$) per calcolare i valori $OPT(i, v)$, $\forall v \in V$ e per $i = 1, \ldots, n$.
- Sia v un nodo per cui $OPT(n, v) < OPT(n-1, v)$ (ci deve essere, altrimenti abbiamo visto prima che G non avrebbe cicli negativi).
- Il cammino da v a t di costo $OPT(n, v)$ ha esattamente n archi (se ne avesse di meno avrebbe costo $\geq OPT(n-1, v) > OPT(n, v)$
- $\Rightarrow$ tale cammino ha $n+1$ nodi e quindi un ciclo (almeno due nodi si ripetono, $n = |V|$)
- tale ciclo ha costo < 0, altrimenti lo potremmo eliminare, ottenere un cammino di $\leq n-1$ archi di costo $< OPT(n, v) < OPT(n-1, v)$ (assurdo!).