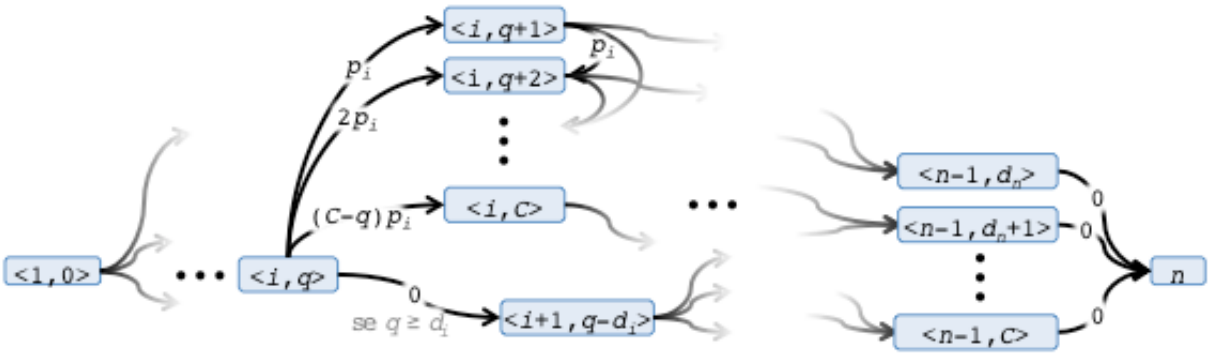
Progettazione di Algoritmi - lezione 19

Discussione dell'esercizio [viaggio]

Un viaggio in auto prevede n tappe. Gli interi d_i, $1 \le i < n$ rappresentano il numero di litri di benzina necessari per spostarsi dalla località i alla successiva. Il serbatoio dell'auto ha capacità C e un litro di benzina acquistato nella località i ha costo p_i, $1 \le i < n$. Dobbiamo trovare un algoritmo che calcola, in tempo $O(nC^2)$, il costo minimo per portare a termine il viaggio.

Un modo per risolvere il problema è di rappresentarlo tramite un grafo. L'idea è di rappresentare con un nodo una possibile situazione durante il viaggio e quindi anche uno stato da cui si possono prendere delle decisioni che porteranno ad altri stati. Durante il viaggio ci possiamo trovare in una delle località, diciamo la località i, e il serbatoio, in quel momento, può contenere q litri. Rappresentiamo tale stato con un nodo relativo alla coppia $\langle i, q\rangle$. Lo stato o nodo iniziale è $\langle 1, 0\rangle$ (località 1 e serbatoio vuoto). Per rappresentare lo stato destinazione introduciamo un nodo n (la coppia non occorre perché il contenuto del serbatoio quando arriviamo a destinazione è irrilevante). Quindi i nodi corrispondono a tutte le coppie $\langle i, q\rangle$ per $i = 1,\ldots,n\text{-}1$ e $q = 0,1,\ldots,C$ e il nodo n. In totale $(n - 1)(C + 1) + 1 = (n - 1)C + n$ nodi. Da uno stato $\langle i, q\rangle$ che decisioni possiamo prendere? Possiamo fare rifornimento e/o andare alla prossima località. Fare rifornimento significa acquistare un certo numero k di litri di benzina con $k = 1,\ldots,C - q$. Tali decisioni possono essere rappresentate da archi: per ogni $k = 1,\ldots,C - q$, c'è un arco da $\langle i, q\rangle$ a $\langle i, q+k\rangle$ di peso kp_i. Il peso dell'arco rappresenta così il costo della decisione. La decisione di andare alla prossima località può essere presa solamente se $q \ge d_i$ e in questo caso può essere rappresentata con un arco da $\langle i, q\rangle$ a $\langle i+1, q-d_i\rangle$ di peso 0. Se $i = n\text{-}1$, l'arco va al nodo destinazione n.



Osserviamo che il grafo appena definito è un DAG. Come possiamo formulare il nostro problema in termini di questo grafo? Ogni cammino dal nodo $\langle 1, 0\rangle$ al nodo n corrisponde a un modo di fare il viaggio, cioè alle decisioni circa dove e quanta benzina fare nelle diverse località. Il peso del cammino è esattamente il costo del viaggio, cioè il costo totale di tutti i litri di benzina acquistati. Quindi per risolvere il nostro problema dobbiamo trovare un cammino di peso minimo dal nodo $\langle 1, 0\rangle$ al nodo n. Possiamo allora usare l'algoritmo di Dijkstra. Per conoscere il tempo di calcolo dobbiamo sapere quanti archi ha il grafo. Il numero preciso è difficile da calcolare, però ogni nodo $\langle i, q\rangle$ con $q \le C/2$ ha almeno $C/2$ archi uscenti. Questo implica che almeno la metà dei nodi ha almeno $C/2$ archi uscenti. Perciò il numero di archi è almeno:

$$\frac{(n-1)C+n}{2} \cdot \frac{C}{2} = \frac{(n-1)C^2+nC}{4} = \Theta(nC^2)$$

Ne segue che l'algoritmo di Dijkstra richiederà tempo $O(nC^2\log(nC))$. Il fattore $\log(nC)$ può essere risparmiato? Possiamo sfruttare il fatto che il grafo è un DAG nello stesso modo che abbiamo già visto per il calcolo del longest path. Invero la stessa tecnica funziona anche per calcolare i cammini minimi sostituendo il max con il min:

$$M[i] = \begin{cases} \min_{i<j\le n}\{M[j] + p(i,j) \mid M[j] \ne -1 \wedge (i,j) \in E\} & \text{se esiste almeno un arco } (i,j) \text{ con } M[j] \ne -1 \\ -1 & \text{altrimenti} \end{cases}$$

dove $M[i]$ è il minimo peso di un cammino dal nodo i al nodo destinazione n (-1 se non ci sono cammini), assumendo che 1,2,…,i,…,n sia un ordinamento topologico dei nodi del DAG. Possiamo riscrivere sia la tabella che il calcolo direttamente per il nostro problema:

$$V[i,q] = \text{minimo costo di un viaggio che parte dalla località } i \text{ con } q \text{ litri e arriva a destinazione}$$

Il calcolo degli elementi della tabella V segue quello per i cammini minimi nel DAG: per ogni $i = 1,\ldots,n-1$ e per ogni $q = 0,\ldots,C$

$$V[i,q] = \begin{cases} \min\{V[i+1, q-d_i], \min\{V[i,q+k] + kp_i \mid 1 \leqslant k \leqslant C-q\}\} & \text{se } q \geqslant d_i \text{ e } q < C \\ \min\{V[i,q+k] + kp_i \mid 1 \leqslant k \leqslant C-q\} & \text{se } q < d_i \\ V[i+1, q-d_i] & \text{se } q = C \end{cases}$$

Caso base: $V[n, q] = 0$, per ogni $q = 0,\ldots,C$. L'algoritmo che calcola la tabella V è facile da scrivere:

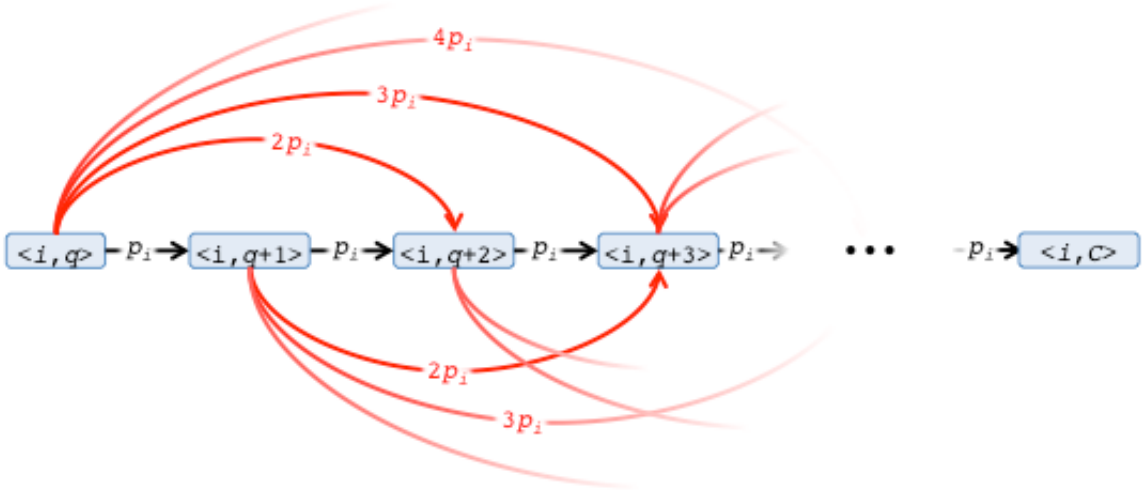
```
VIAGGIO(D: array distanze, P: array prezzi, n:numero località, C: capacità)
    V: tabella nx(C + 1)
    FOR q <- 0 TO C DO V[n, q] = 0
    FOR i <- n-1 DOWNTO 1 DO
        FOR q <- C DOWNTO 0 DO
            IF q < C THEN
                V[i, q] <- ∞
                FOR k <- 1 TO C - q DO
                    IF V[i, q] > V[i, q + k] + k*P[i] THEN
                        V[i, q] = V[i, q + k] + k*P[i]
                IF q ≥ D[i] AND V[i, q] > V[i + 1, q - D[i]] THEN
                    V[i, q] <- V[i + 1, q - D[i]]
            ELSE
                V[i, q] = V[i + 1, q - D[i]]
    RETURN V[1, 0]
```

Il calcolo di ogni elemento della tabella V richiede al più tempo $O(C)$. Siccome la tabella ha $O(nC)$ elementi, la complessità è $O(nC^2)$. Come al solito, se si vogliono anche conoscere i rifornimenti durante il viaggio, è sufficiente ripercorrere a ritroso la tabella dall'elemento $V[1, 0]$ seguendo le decisioni che hanno portato ai valori ottimi:

```
VIAGGIO_SOL(V: tabella, D: array distanze, P: array prezzi, n: numero località, C: capacità)
    R: array dei rifornimenti, inizializzato a 0
    i <- 1
    q <- 0
    WHILE i < n DO
        IF q < C THEN
            v <- V[i, q]
            k <- 1
            WHILE k ≤ C - q AND v ≠ V[i, q + k] + k*P[i] DO
                k <- k + 1
            IF k ≤ C - q THEN
                q <- q + k
                R[i] <- R[i] + k     /* rifornimento */
            ELSE
                q <- q - D[i]
                i <- i + 1
        ELSE
            q <- q - D[i]
            i <- i + 1
    RETURN R
```

La complessità è O(*nC*) perchè al più vengono attraversati tutti i possibili stati. Si poteva precalcolare insieme alla tabella *V* una tabella aggiuntiva che registra le decisioni prese per calcolare ogni elemento. Con tale tabella si possono ritrovare i rifornimenti ottimi in tempo O(*n*).

Perché non possiamo calcolare la tabella *V* in tempo O(*nC*)? Ciò che causa l'aggravio di un fattore *C* nella complessità è il calcolo del minimo relativamente alla decisione sui litri di benzina da acquistare, 1,2,3,… Questa decisione può essere decomposta in una serie di decisioni più semplici: acquistare o meno un litro di benzina. In termini del grafo si tratta di rendersi conto che molti archi sono inutili. Nella figura qui sotto, gli archi in rosso possono essere eliminati senza effetti sui cammini minimi. Perchè per ognuno di tali archi (*u*, *v*) c'è un cammino da *u* a *v* di peso uguale a quello dell'arco:



Eliminando questi archi, ogni nodo ha al più due archi uscenti e quindi il grafo ha O(*nC*) archi e il calcolo dei cammini minimi prende tempo O(*nC*). D'altronde, questo è simile all'approccio usato nel calcolo della tabella del problema del *resto* in cui invece di considerare come singola decisione il numero di banconote da usare di un certo taglio abbiamo considerato la decisione per ogni banconota. Seguendo questo approccio il calcolo della tabella *V* diventa:

$$V[i,q] = \begin{cases} \min\{V[i+1, q-d_i], V[i,q+1]+p_i\} & \text{se } q \geqslant d_i \text{ e } q < C \\ V[i,q+1]+p_i & \text{se } q < d_i \\ V[i+1,q-d_i] & \text{se } q = C \end{cases}$$

La correttezza di questo metodo di calcolo deriva anche dal fatto che in base al metodo precedente si vede facilmente che per ogni $i < n$, per ogni $q \geq d_i$ e per ogni $k \leq C - q$ risulta:

$$V[i,q+1]+p_i \leqslant V[i,q+2]+2p_i \leqslant \cdots \leqslant V[i,q+k]+kp_i$$

Perciò si ha che

$$\min\{V[i,q+k]+kp_i \mid 1 \leqslant k \leqslant C-q\} = V[i,q+1]+p_i$$

Quindi con questo nuovo metodo possiamo calcolare la tabella in tempo O(*nC*).

Cammini minimi in presenza di pesi anche negativi

Sia *G* un grafo diretto e pesato con pesi anche negativi. Se *G* ha un ciclo di peso negativo il concetto di cammino di peso minimo tra due nodi può risultare malposto perchè, ad esempio, per due qualsiasi nodi del ciclo esistono cammini di peso arbitrariamente basso e lunghezza arbitrariamente grande. Lo stesso vale per due nodi *u* e *v* tali che esiste un cammino da *u* a *v* che passa per un qualche nodo del ciclo di peso negativo. Se però il grafo non contiene cicli di peso negativo o non sono raggiungibili da un nodo *s* allora i cammini minimi da *s* esistono ed hanno una lunghezza limitata.

Sia G un grafo diretto pesato con pesi anche negativi. Se da un nodo s non sono raggiungibili cicli di peso negativo, allora ogni nodo v raggiungibile da s ha un cammino minimo da s a v con un numero di archi inferiore a n.

Dimostrazione Supponiamo per assurdo che ci sia un cammino P da s a v di lunghezza almeno n di peso strettamente minore a quello di qualsiasi cammino di minore lunghezza. Avendo lunghezza almeno n, P attraversa almeno $n+1$ nodi e quindi ha un nodo u che è attraversato due volte. Decomponiamo P in tre parti: $P_{s,u}$ la parte che va da s alla prima occorrenza di u, $P_{u,u}$ la parte che va dalla prima occorrenza alla seconda occorrenza di u e la parte rimanente $P_{u,v}$. Concatenando $P_{s,u}$ con $P_{u,v}$ si ottiene un cammino P' di lunghezza inferiore a quella di P e affinché P abbia peso minore a quello di P' la parte di cammino $P_{u,u}$ deve avere peso negativo. Quindi $P_{u,u}$ è un ciclo di peso negativo, in contraddizione con l'ipotesi che da s non sono raggungibili cicli di peso negativo.

Questa proprietà ci dice che, in un grafo senza cicli di peso negativo, i cammini minimi da un nodo s possono essere cercati tra quelli di lunghezza inferiore a n. Per risolvere il problema dei cammini minimi tramite la programmazione dinamica, la proprietà suggerisce di considerare i sotto-problemi che si ottengono limitando la lunghezza dei cammini. Così definiamo la seguente tabella:

$$M[k, v] = \begin{cases} \text{peso di un cammino minimo di lunghezza al più } k \text{ da } s \text{ a } v & \text{se esiste almeno un cammino di lunghezza al più } k \text{ da } s \text{ a } v \\ \infty & \text{altrimenti} \end{cases}$$

I casi base sono $M[0, s] = 0$ e $M[0, v] = \infty$ per $v \neq s$. Per calcolare $M[k, v]$ osserviamo che o $M[k, v] = M[k - 1, v]$ oppure c'è un cammino P di peso inferiore a $M[k - 1, v]$ di lunghezza esattamente k. Il cammino P è composto da un ultimo arco (u, v) e da un cammino P_u di lunghezza $k - 1$ che va da s a u. Chiaramente, il peso di P_u è uguale a $M[k - 1, u]$. Quindi per ogni $k = 1,\ldots,n - 1$, abbiamo che

$$M[k, v] = \min\left\{M[k-1, v],\ \min\left\{M[k-1, u] + p(u, v) \mid (u, v) \in E\right\}\right\}$$

Si osservi che questa regola di calcolo è valida per qualsiasi lunghezza k anche superiore a n. Se facciamo un passo in più, cioè calcoliamo anche la riga n della tabella M, possiamo determinare se ci sono cicli di peso negativo raggiungibili da s.

Esiste un v tale che $M[n, v] \neq M[n - 1, v]$ se e solo se esiste un ciclo di peso negativo raggiungibile da s.

Dimostrazione Se esiste un v tale che $M[n, v] \neq M[n - 1, v]$, allora esiste un cammino di lunghezza n con peso inferiore a quello di qualsiasi cammino di lunghezza inferiore a n. Questo, in base alla proprietà che abbiamo già dimostrato, implica che c'è un ciclo di peso negativo raggiungibile da s. Viceversa, supponiamo che ci sia un ciclo di peso negativo raggiungibile da s e sia u un nodo di tale ciclo. Allora ci sono cammini da s a u che sfruttando tale ciclo avranno peso arbitrariamente basso. Quindi per una certa lunghezza $h \geq n$ si avrà che $M[h, u] < M[n - 1, u]$. Ma allora non può essere $M[n, x] = M[n - 1, x]$ per ogni nodo x, perché se fosse così dalla regola di calcolo di M si avrebbe che, per qualsiasi $t \geq n$, $M[t, x] = M[n - 1, x]$.

Quello che abbiamo descritto è conosciuto con il nome di algoritmo di *Bellman-Ford*. Passiamo ora a scrivere una versione in pseudo-codice. L'algoritmo ritorna o i pesi dei cammini minimi da s o segnala che c'è un ciclo di peso

negativo.

```
BELLMAN_FORD(G: grafo diretto e pesato, s: nodo)
    M: tabella (n+1)xn
    FOR ogni nodo u DO M[0, u] <- ∞
    M[0, s] <- 0
    FOR k <- 1 TO n DO
        FOR ogni nodo v DO
            M[k, v] <- M[k-1, v]
            FOR ogni arco (u, v) entrante in v DO
                IF M[k-1, u] + p(u, v) < M[k, v] THEN
                    M[k, v] <- M[k-1, u] + p(u, v)
            IF k = n AND M[k, v] < M[k-1, v] THEN
                RETURN "Ci sono cicli negativi"
    RETURN riga n di M
```

Per quanto riguarda la complessità osserviamo che l'inizializzazione della tabella prende tempo $O(n^2)$. Il calcolo di ogni riga di M esamina al più una volta ogni arco del grafo, quindi impiega tempo $O(n + m)$. In totale la complessità è $O(n(n + m)) = O(nm)$.

Per ritrovare anche i cammini minimi, oltre al loro peso, conviene calcolarsi insieme alla tabella anche l'albero dei cammini minimi. Questo si può fare facilmente mantenendo per ogni nodo v il suo predecessore, cioè il nodo u che lo precede nel cammino minimo. È quindi una facile modifica dell'algoritmo descritto sopra.

```
BELLMAN_FORD_SOL(G: grafo diretto e pesato, s: nodo)
    M: tabella (n+1)xn
    P: vettore dei padri inizializzato a -1
    FOR ogni nodo v DO M[0, u] <- ∞
    M[0, s] <- 0
    P[s] <- s               /* Radice */
    FOR k <- 1 TO n DO
        FOR ogni nodo v DO
            M[k, v] <- M[k-1, v]
            FOR ogni arco (u, v) entrante in v DO
                IF M[k-1, u] + p(u, v) < M[k, v] THEN
                    M[k, v] <- M[k-1, u] + p(u, v)
                    P[v] <- u
            IF k = n AND M[k, v] < M[k-1, v] THEN
                RETURN "Ci sono cicli negativi"
    RETURN riga n di M, P
```

Si osservi che non è necessario mantenere l'intera tabella ma bastano solamente le ultime due righe. Perciò l'algoritmo può essere facilmente modificato cosicché usi memoria $O(n)$ anzichè $O(n^2)$.

Esercizio [cammini]

Sia G un grafo diretto e pesato con pesi anche negativi, ma senza cicli di peso negativo. Dati due nodi u e v vogliamo sapere quanti cammini minimi ci sono da u a v in G. Descrivere un algoritmo efficiente per risolvere questo problema e valutarne la complessità.