

Programmazione dinamica (III parte)

Progettazione di Algoritmi a.a. 2020-21

Matricole congrue a 1

Docente: Annalisa De Bonis

45

45

Problema dello zaino

- **Input**
 - n oggetti ed uno zaino
 - L'oggetto i pesa $w_i > 0$ chili e ha valore $v_i > 0$.
 - Lo zaino puo` trasportare fino a W chili.
- **Obiettivo:** riempire lo zaino in modo da massimizzare il valore totale degli oggetti inseriti senza eccedere il limite W .
- **Esempio:** { 3, 4 } ha valore 40.

W = 11

Oggetto	Valore	Peso
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Greedy: seleziona ad ogni passo l'oggetto con il rapporto v_i/w_i piu` grande in modo che il peso totale dei pesi selezionati non superi w

Esempio: soluzione greedy { 5, 2, 1 } ha valore = 35 \Rightarrow greedy non e` ottimo

Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

46

46

Problema dello zaino

Input

- n oggetti: l'oggetto i pesa $w_i > 0$ chili e ha valore $v_i > 0$
- limite W

Obiettivo: selezionare un sottoinsieme S degli n oggetti in modo

- da rispettare il vincolo $\sum_{i \in S} w_i \leq W$, cioè che la somma dei pesi degli oggetti selezionati sia minore di W
- e da massimizzare $\sum_{i \in S} v_i$

Corrisponde al problema subset sums quanto $v_i = w_i$ per ogni i.

Problema dello zaino: estensione approccio usato per Subset Sums

Def. $OPT(i, w)$ = valore della soluzione ottima per gli oggetti 1, ..., i con limite di peso totale w.

- Caso 1: La soluzione ottima per i primi i oggetti, con limite di utilizzo w non include l'oggetto i.
 - La soluzione ottima è in questo caso la soluzione ottima per $\{1, 2, \dots, i-1\}$ con limite di utilizzo w \rightarrow in questo caso $OPT(i, w) = OPT(i-1, w)$
- Caso 2: La soluzione ottima per i primi i oggetti, con limite di utilizzo w include l'oggetto i.
 - La soluzione ottima include la soluzione ottima per $\{1, 2, \dots, i-1\}$ con limite di utilizzo $w - w_i \rightarrow$ in questo caso $OPT(i, w) = v_i + OPT(i-1, w - w_i)$

La soluzione ottima per i primi i oggetti con limite di utilizzo w va ricercata tra le soluzioni ottime per i due casi. Questo però se $i > 0$ e $w_i < w$.

Se $i=0$, banalmente si ha $OPT(i, w)=0$. Se $w_i > w$, è possibile solo il caso 1 perché i non può far parte della soluzione in quanto ha peso maggiore del peso trasportabile.

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{OPT(i-1, w), v_i + OPT(i-1, w - w_i)\} & \text{otherwise} \end{cases}$$

Problema dello zaino: algoritmo

```

Knapsack (n,w1,...,wn, v1,...,vn,W)
for w = 0 to W
    M[0, w] = 0

for i = 1 to n
    for w = 0 to W
        if (wi > w)
            M[i, w] = M[i-1, w]
        else
            M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi]}

return M[n, W]
    
```

Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

49

Algoritmo per il problema della zaino: esempio

		W →											
		0	1	2	3	4	5	6	7	8	9	10	11
n ↓	ϕ	0	0	0	0	0	0	0	0	0	0	0	0
	{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
	{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
	{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
	{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
	{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

$OPT(5,11) = OPT(4,11) = v_4 + OPT(3,11-w_4) = v_4 + OPT(3,5) =$
 $v_4 + v_3 + OPT(2,0) = v_4 + v_3 + OPT(1,0) =$
 $v_4 + v_3 + OPT(0,0) = 22 + 18 + 0 = 40$

Soluzione ottima : { 4, 3 }
 Valore soluzione ottima = 22 + 18 = 40

Oggetto	Valore	Peso
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

50

Problema dello zaino: tempo di esecuzione algoritmo

- Tempo di esecuzione. $\Theta(n W)$.
 - Non è polinomiale nella dimensione dell'input!
 - "Pseudo-polinomiale": L'algoritmo è efficiente quando W ha un valore ragionevolmente piccolo.
 - Se volessimo produrre la soluzione ottima, potremmo scrivere un algoritmo simile a quelli visti prima in cui la soluzione ottima si ricostruisce andando a ritroso nella matrice M . Tempo $O(n)$.
 - **Esercizio: Scrivere lo pseudocodice dell'algoritmo che produce la soluzione ottima per un'istanza del problema dello zaino.**
 - **Esercizio: Scrivere la versione ricorsiva dell'algoritmo di programmazione dinamica per il problema dello zaino.**

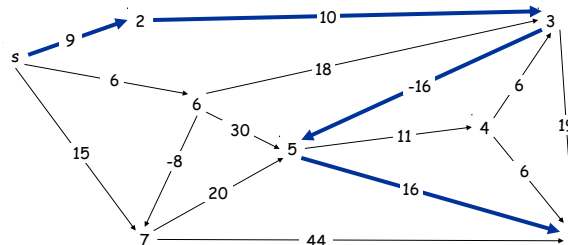
Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

51

51

Cammini minimi

- **Problema del percorso più corto.** Dato un grafo direzionato $G = (V, E)$, con pesi degli archi c_{vw} , trovare il percorso più corto da s a t .
- **Esempio.** I nodi rappresentano agenti finanziari e c_{vw} è il costo (eventualmente <0) di una transazione che consiste nel comprare dall'agente v e vendere immediatamente a w .



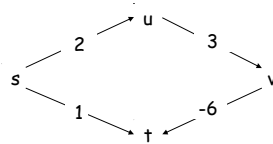
Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

52

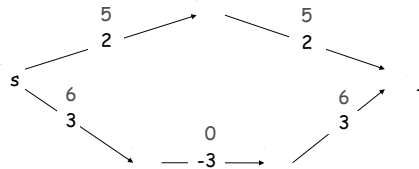
52

Cammini minimi in presenza di archi con costo negativo

- Dijkstra. Può fallire se ci sono archi di costo negativo



- Re-weighting. Aggiungere una costante positiva ai pesi degli archi potrebbe non funzionare.



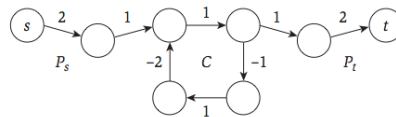
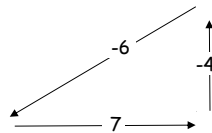
Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

53

53

Cammini minimi in presenza di archi con costo negativo

- Ciclo di costo negativo.



Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

54

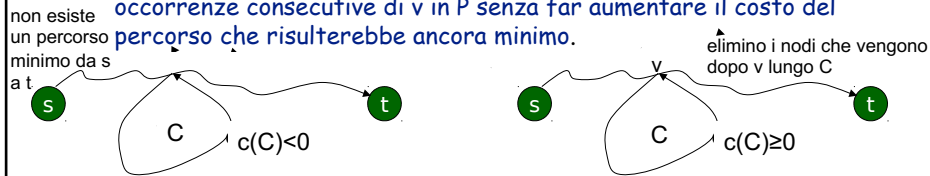
54

Cammini minimi in presenza di archi con costo negativo

Osservazione. Se qualche percorso da s a t contiene un ciclo di costo negativo allora non esiste un percorso minimo da s a t . In caso contrario esiste un percorso minimo da s a t che è semplice (nessun nodo compare due volte sul percorso).

- **Dim.** Se esiste un percorso P da s a t con un ciclo C di costo negativo $-c$ allora ogni volta che attraversiamo il ciclo riduciamo il costo del percorso di un valore pari a c . Ciò rende impossibile definire il costo del percorso minimo perché dato un percorso riusciamo sempre a trovarne uno di costo minore attraversando il ciclo C (osservazione questa che avevamo già fatto in precedenti lezioni).

Supponiamo ora che nessun percorso da s a t contenga cicli negativi e sia P un percorso minimo da s a t (ovviamente P è privo di cicli di costo negativo). Supponiamo che un certo vertice v appaia almeno due volte in P . C'è quindi in P un ciclo che contiene v e che per ipotesi deve avere costo **non negativo**. In questo caso potremmo rimuovere le porzioni di P tra due occorrenze consecutive di v in P senza far aumentare il costo del percorso che risulterebbe ancora minimo.



55

Cammini minimi: Programmazione dinamica

Def. $OPT(i, v)$ = lunghezza del cammino più corto P per andare da v a t che consiste di al più i archi

Per computare $OPT(i, v)$ quando $i > 0$ e $v \neq t$, osserviamo che

- il percorso ottimo P deve contenere almeno un arco (che ha come origine v).
- se (v, w) è il primo arco di P allora P è formato da (v, w) e dal percorso più corto da w a t di al più $i-1$ archi

$$OPT(i, v) = \min_{(v,w) \in E} \{ OPT(i-1, w) + c_{vw} \}$$

$$OPT(i, v) = \begin{cases} 0 & \text{se } v=t \\ \infty & \text{se } i=0 \text{ e } v \neq t \\ \min_{(v,w) \in E} \{ OPT(i-1, w) + c_{vw} \} & \text{altrimenti} \end{cases}$$

56

56

Cammini minimi: Programmazione dinamica

$$OPT(i,v) = \begin{cases} 0 & \text{se } v=t \\ \infty & \text{se } i=0 \text{ e } v \neq t \\ \min_{(v,w) \in E} \{OPT(i-1, w) + c_{vw}\} & \text{altrimenti} \end{cases}$$

Dove si usa l'osservazione di prima sul fatto che in assenza di cicli negativi il percorso minimo è semplice?

Ecco dove....

Affermazione. Se non ci sono cicli di costo negativo allora $OPT(n-1, v)$ = lunghezza del percorso più corto da v a t .

Dim. Dall'osservazione precedente se non ci sono cicli negativi allora esiste un percorso di costo minimo da v a t che è semplice e di conseguenza contiene al più $n-1$ archi

Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

57

57

Algoritmo di Bellman-Ford per i cammini minimi

```
Shortest-Path(G, t) {
  foreach node v ∈ V
    M[0, v] ← ∞
    S[0, v] ← ∅ // ∅ indica che non ci sono percorsi
                //da v a t di al più 0 archi
  for i = 0 to n-1
    M[i, t] ← 0
    S[i, t] ← t //t indica che non ci sono successori
                //lungo il percorso ottimo da t a t
  for i = 1 to n-1
    foreach node v ∈ V
      M[i, v] ← ∞, S[i, v] ← ∅
      foreach edge (v, w) ∈ E
        if M[i-1, w] + cvw < M[i, v]
          M[i, v] ← M[i-1, w] + cvw
          S[i, v] ← w //serve per ricostruire i
                    //percorsi minimi verso t
}
```

- Assumiamo che per ogni v esista un percorso da v a $t \rightarrow n=O(m)$
- Analisi. Tempo $\Theta(mn)$, spazio $\Theta(n^2)$.
- $S[i,v]$: Memorizza il successore di v lungo il percorso minimo per andare da v a t attraversando al più i archi

Progettazione di Algoritmi A.A. 2020-21
A. De Bonis

58

58