5

Programmazione dinamica

5.1 Stringhe Palindrome

Una stringa si dice palindroma se è uguale alla sua trasposta, ossia se è la stessa letta da sinistra a destra, o da destra a sinistra (es: anna, osso, alla, arara, ...). Scrivere un algoritmo di programmazione dinamica che prende in input una stringa $S = a_1 a_2 \cdots a_n$ su un certo alfabeto Σ e restituisce il numero minimo di caratteri da aggiungere ad S per renderla palindroma.

5.1.1 Spazio dei sottoproblemi e struttura ricorsiva del problema

Nel seguito, per ogni $1 \leq i \leq j \leq n$, denotiamo con $S(i,j)$ la sottostringa $S(i,j) = a_i \cdots a_j$; per ogni $i > j$, invece, $S(i,j) = \epsilon$ (la stringa vuota). Siano inoltre $P(i,j)$ una possibile soluzione ottima per la sottostringa $S(i,j)$ e $m(i,j)$ il valore di tale soluzione ottima (ossia, il numero minimo di caratteri da aggiungere a $S(i,j)$ per renderla palindroma).

Consideriamo la generica sottostringa $S(i,j)$ e cerchiamo di capire come è possibile costruire la soluzione ottima $P(i,j)$. Se $S(i,j) = \epsilon$ (ossia se $i > j$) oppure se $|S(i,j)| = 1$ (ossia se $i = j$ e $S(i,j) = a_i$) allora la sottostringa $S(i,j)$ è già palindroma e non dobbiamo aggiungere nessun carattere. In questo caso: $P(i,j) = a_i$ e $m(i,j) = 0$. Se, invece, $|S(i,j)| > 1$ (ossia se $i < j$) confrontiamo a_i e a_j (il primo e l'ultimo simbolo di $S(i,j)$). Distinguiamo i seguenti casi:

- $a_i = a_j$.
  Possiamo rendere palindroma $S(i,j)$ semplicemente rendendo palindroma la sottostringa $S(i+1, j-1) = a_{i+1} \cdots a_{j-1}$ (la sottostringa che otteniamo eliminando il primo e l'ultimo carattere da $S(i,j)$). In questo caso: $P(i,j) = a_i \; P(i+1, i-1) \; a_j$ e $m(i,j) = m(i+1, j-1)$.
- $a_i \neq a_j$. In questo caso, possiamo seguire due strade:

1. rendiamo palindroma $S(i+1,j) = a_{i+1} \cdots a_j$ e costruiamo una soluzione ammissibile per $S(i,j)$ come segue: $a_i\ P(i+1,j)\ a_i$. Al numero minimo di caratteri necessari per rendere $S(i+1,j)$ palindroma aggiungiamo un carattere (nello specifico il carattere a_i aggiunto in fondo).
2. rendiamo prima palindroma $S(i,j-1) = a_i \cdots a_{j-1}$ e costruiamo una soluzione ammissibile per $S(i,j)$ come segue: $a_j\ P(i,j-1)\ a_j$. Al numero minimo di caratteri necessari per rendere $S(i,j-1)$ palindroma aggiungiamo un carattere (il carattere a_j aggiunto in testa).

Tra le due alternative scegliamo quella che ci consente di minimizzare il numero di caratteri aggiunti. Ricapitolando, se $i < j$ e $a_i \neq a_j$, $P(i,j)$ è la stringa di lunghezza minima tra $a_i\ P(i+1,j)\ a_i$ e $a_j\ P(i,j-1)\ a_j$, e

$$\begin{aligned} m(i,j) &= \min\{m(i+1,j)+1, m(i,j-1)+1\} \\ &= \min\{m(i+1,j), m(i,j-1)\}+1 \end{aligned}$$

5.1.2 Sottostruttura ottima

Per quanto visto nella precedente sezione

$$P(i,j) = \begin{cases} S(i,j) & \text{se } i \geq j \\ a_i\, P(i+1,j-1)\, a_j & \text{se } i < j \text{ e } a_i = a_j \\ \min\{a_i\, P(i-1,j)\, a_i, a_j\, P(i,j-1)\, a_j\} & \text{se } i < j \text{ e } a_i \neq a_j \end{cases}$$

dove con $\min\{a_i\, P(i-1,j)\, a_i, a_j\, P(i,j-1)\, a_j\}$ abbiamo indicato la stringa di lunghezza minima nell'insieme $\{a_i\, P(i-1,j)\, a_i, a_j\, P(i,j-1)\, a_j\}$. Questa definizione di $P(i,i)$ dimostra che se una soluzione ottima di un sottoproblema non banale contiene al suo interno soluzioni ottime di sottoproblemi (vedi in particolare la definizione di $P(i,j)$ nel casi in cui $i < j$).

5.1.3 Funzione ricorsiva per il calcolo del valore della soluzione ottima

$$m(i,j) = \begin{cases} 0 & \text{se } i \geq j \\ m(i+1,j-1) & \text{se } i < j \text{ e } a_i = a_j \\ \min\{m(i-1,j), m(i,j-1)\}+1 & \text{se } i < j \text{ e } a_i \neq a_j \end{cases}$$

5.1.4 Schema Bottom-Up

È molto simile a quello visto per il problema della moltiplicazione di una sequenza di matrici (Matrix-Chain-Order).

Sia M una matrice di dimensione $n \times n$. Riempiamo (nell'ordine) tutte le seguenti posizioni della tabella

- tutte le osizioni $M[i,j]$ per $i \geq j$ (corrispondono alla stringa vuota e a sottostringhe lunghe 2). Queste posizioni corrispondono ai casi base e possono essere poste a zero.
- tutte le posizioni $M[i,i+1]$ per $i = 1, n-1$ (corrispondono a sottostringhe lunghe 2)
- tutte le posizioni $M[i,i+2]$ per $i = 1, n-2$ (corrispondono a sottostringhe lunghe 3)
- $\cdots$
- tutte le posizioni $M[i,i+h]$ per $i = 1, n-h$ (corrispondono a sottostringhe lunghe $h+1$)
- $\cdots$
- la posizione $M[1,n]$ (corrisponde all'unica sottostringa lunga n, ossia $S(1,n) = a_1 \cdots n = S$).

Il seguente algoritmo prende in input la stringa S da rendere palindroma:

```
MAKE-PALINDROME(S)
 1  n ← length[S]
 2  for j ← 1 to n
 3      do for i ← j to n
 4             do M[i,j] ← 0
 5  for h ← 1 to n − 1
 6      do for i ← 1 to n − h
 7             do j ← i + h
 8                 if S[i] = S[j]
 9                    then M[i,j] ← M[i + 1, j − 1]
10                    else if M[i + 1, j] < M[i, j − 1]
11                            then M[i,j] ← M[i + 1, j] + 1
12                            else M[i,j] ← M[i, j − 1] + 1
13  return M
```

5.1.5 Costruzione della soluzione ottima

Homework: Modificare l'algoritmo fornito nella sezione precedente in modo che consenta anche di ricostruire una soluzione ottima.

5.2 Piano di Studi

Sia $A = \{a_1, a_2 \cdots, a_n\}$ l'insieme degli n esami attivati nel Corso di Laurea in informatica. Ogni esame a_i vale c_i crediti formativi ed ha un grado di difficoltà rappresentato da un coefficiente d_i. Gli studenti possono redigere il loro piano di studi individuale scegliendo nella lista degli esami attivati un insieme di esami tale che la somma dei crediti corrispondenti sia almeno P.

Progettare un algoritmo che redige un piano di studi regolare (la somma dei crediti deve essere almeno P) con difficoltà minima in $O(nP)$.

Il problema può essere riformulato come segue:

$$\min \sum_{i=1}^{n} d_i \cdot x_i$$
$$\sum_{i=1}^{n} c_i \cdot x_i \geq P$$
$$x_i \in \{0, 1\}$$

Questa formulazione rende evidente come questo problema sia simile a quello dello zaino 0-1.

5.2.1 Spazio dei sottoproblemi e struttura ricorsiva del problema

Per risolvere tale problema dobbiamo risolvere tutti i sottoproblemi della forma (A_i, p) dove per $1 \leq i \leq n$, $A_i = \{a_1, a_2 \cdots, a_i\}$ è il sottoinsieme dei primi i esami attivati, $A_0 = \emptyset$, e $0 \leq p \leq P$ è il numero di crediti residui da acquisire. In generale, (A_i, p) rappresenta il sottoproblema che consiste nel redigere un piano di studi con almeno p crediti potendo scelgliere tra gli esami attivati elencati in A_i.

Problema originale: (A_n, P).

Casi base: (A_0, p) (per $0 < p \leq P$) e $(A_i, 0)$ (per $0 \leq i \leq n$).
Con (A_0, p) indichiamo il sottoproblema che consiste nel redigere un piano di studi con almeno p crediti potendo scelgliere tra un insieme vuoto di esami. Poichè $p > 0$, questo è chiaramente impossibile e (A_0, p) è un sottoproblema non ammissibile (originato da scelte sbagliate). Vedremo più avanti che il valore della soluzione ottima per tali sottoproblemi è ∞. $(A_i, 0)$ ha, invece, valore ottimo zero (non avendo dei crediti residui da acquisire, una soluzione ottima per questo problema consiste nel non scegliere nessun altro esame ed ha chiaramente valore 0).

Casi non banali: (A_i, p) con $i, p > 0$

Come possiamo risolvere un generico sottoproblema non banale (A_i, p)? In maniera del tutto analoga a quanto fatto nel caso dello zaino 0-1, possiamo:

- decidere di sostenere l'esame a_i e poi risolvere il sottoproblema che consiste nel realizzare un piano di studi con almeno $p - c_i$ crediti scegliendo esami nell'insieme A_{i-1}. Per tener conto del fatto che $p - c_i \leq 0$ (nel caso in cui $p \leq c_i$) e visto che i crediti residui non possono assumere un valore negativo, in realtà scegliendo l'esame a_i non ci resta che risolvere il sottoproblema $(A_{i-1}, \max\{0, p - c_i\})$.
- decidere di *non* sostenere l'esame a_i e poi risolvere il sottoproblema (A_{i-1}, p).

Tra le due soluzione ottenute in questo modo scegliamo quella con valore (somma dei gradi di difficoltà) è minima. Dovendo scegliere tra due possibili soluzioni quella la cui somma dei gradi di difficoltà è minima, possiamo segnalare che una dei due soluzioni è non ammissibile (e quindi non può essere scelte) ponendo il suo valore ottimo a ∞.

5.2.2 Sottostruttura ottima

Sia $S(i,p)$ una soluzione ottima per il sottoproblema non banale (A_i, p). Per quanto appena detto, possiamo distiguere due casi (dove, per brevità, abbiamo posto $q = \max\{0, p - c_i\}$):

(1) $S(i,p) = \{a_i\} \cup S(i-1,q)$, ossia $S(i,p)$ contiene l'esame a_i ed una soluzione $S(i-1,q)$ per il sottoproblema (A_{i-1}, q); oppure
(2) $a_i \notin S(i,p)$ e $S(i,p)$ è in realtà una soluzione per il sottoproblema (A_{i-1}, p).

Nel caso (1), dimostrare lo sottostruttura ottima significa dimostrare che se $S(i,p)$ è ottima allora lo è anche $S(i-1,q)$.

Assumiamo (per assurdo) che non lo sia. Deve quindi esistere una soluzione alternativa $S'(i-1,q)$ il cui valore (somma dei gradi di difficoltà degli esami selezionati) è minore del valore di $S(i-1,q)$. Allora, la soluzione ammissibile $S'(i,p) = \{a_i\} \cup S'(i-1,q)$ ha un valore (pari alla somma dei gradi di difficoltà degli esami selezionati in $S'(i-1,q)$ più il grado di difficoltà d_i dell'esame a_i) minore del valore di $S(i,p)$ (pari alla somma dei gradi di difficoltà degli esami selezionati in $S(i-1,q)$ più d_i). Impossibile perchè $S(i,p)$ è una soluzione ottima.

Homework: Usare lo stesso schema per dimostrare la sottostruttura ottima anche nel caso (2).

5.2.3 Funzione ricorsiva per il calcolo del valore della soluzione ottima

Sia $v(i,p)$ il valore della soluzione ottima per il sottoproblema (A_i, p)

$$v(i,p) = \begin{cases} \infty & \text{se } i = 0 \text{ e } p > 0 \\ 0 & \text{se } i \geq 0 \text{ e } p = 0 \\ \min\{v(i-1, \max\{0, p-c_i\}) + d_i, v(i-1,p)\} & \text{se } i,p > 0 \end{cases}$$

5.2.4 Schema Bottom-Up

Sia V una matrice $(n+1) \times (P+1)$. Riempiamo (nell'ordine) le seguenti posizioni:

- $V[0,p] = \infty$ per $p = 1, \cdots, P$
- $V[i,0] = \infty$ per $i = 0, \cdots, n$ (questo copre i casi base)
- per riempire la generica posizione $V[i,p]$ (con $i,p > 0$) abbiamo bisogno dei valori $V[i-1,p]$ e $V[i-1, \max\{0, p-c_i\}]$, valori si trovano nella riga in alto.

Quindi una volta riempite le posizioni che corrispondono ai casi base, ci basta riempire la matrice riga per riga (vedi il problema della longest common subsequence).
Il seguente algoritmo prende in input il vettore $C = [c_1, \cdots, c_n]$ dei crediti formativi, il vettore $D = [d_1, \cdots, d_n]$ dei gardi di difficoltà, ed il naturale P.

```
CURRICULUM(C, D, P)
 1  n ← length[C]
 2  for p ← 1 to P
 3      do V[0,p] ← ∞
 4  for i ← 0 to n
 5      do V[i,0] ← 0
 6  for i ← 1 to n
 7      do for p ← 1 to P
 8              do q ← max{0, p − C[i]}
 9                 if V[i−1,q] + D[i] < V[i−1,p]
10                    then V[i,p] ← V[i−1,q] + D[i]
11                    else  V[i,p] ← V[i−1,p]
12  return V
```

5.2.5 Costruzione della soluzione ottima

Homework: Modificare l'algoritmo fornito nella sezione precedente in modo che consenta anche di ricostruire una soluzione ottima.

5.3 Somma di sottoinsiemi

Progettare un algoritmo che, preso un insieme $A = \{a_1, a_2, \cdots, a_n\}$ di interi ed un intero K, trovi (se esiste) un sottoinsieme di A la somma dei cui elementi sia pari a K. La complessità dell'algoritmo deve essere $O(nK)$.
Suggerimento la soluzione è molto simile a quella fornita per il problema del Piano di Studi (Sezione 5.2).

5.4 Scacchiera

Una pedina è posizionata sulla casella $(1,1)$ in alto a sinistra di una scacchiera $n \times n$ e deve raggiungere la casella (n,n) in basso a destra. Quando posizionata sulla generica casella (i,j) per la pedina sono possibili al più due mosse:

spostarsi verso il basso nella casella $(i+1, j)$ (possibile solo se $i < n$), o spostarsi verso destra nella casella $(i, j+1)$ (posto che $j < n$). Stando così le cose la pedina raggiungerà la casella (n, n) in basso a destra con un cammino che l'ha portata a toccare $2n1$ caselle. Ad ogni casella della scacchiera è associato un valore $c(i, j)$ ed il valore del cammino è dato dalla somma dei valori delle caselle toccate dalle pedina. Progettare un algoritmo che trova il cammino di valore massimo in $O(n^2)$.

5.5 Selezione di attività pesate

Sia $S = \{a_1, a_2, \cdots, a_n\}$ un insieme di n attività. Ogni attività a_i si svolve nell' intervallo temporale $[s_i, f_i)$ (con $s_i < f_i$) ed ha associato il guadagno v_i che si ottiene dal suo svolgimento. Due attività a_i e a_j sono compatibili se gli intervalli $[s_i, f_i)$ e $[s_j, f_j)$ non si sovrappongono (se $s_i \geq f_j$ o $s_j \geq f_i$). Inoltre, il valore di un sottoinsieme di attività è dato dalla somma valori delle attività compongono. Scrivere un algoritmo che selezioni un sottoinsieme di attività mutuamente compatibili in S con valore massimo. Descrivere un algoritmo che risolve il problema in $O(n^2)$.

N.B in questo caso non cerchiamo il più grande insieme di attività mutuamente compatibili, ma il sottoinsieme di attività mutuamente compatibili che ci assicura il maggior guadagno.

Suggerimento: Assumete (al solito, per questa classe di problemi) che le attività siano memorizzate in S in ordine crescente rispetto ai tempi di fine e considerate la classe di sottoproblemi $S_{ij} = \{a_k \in S \,|\, f_i \leq s_k < f_k \leq s_j\}$. Alcune osservazioni:

- Se $S_{ij} \neq \emptyset$ (il che si verfica solo se $i < j$), e una soluzione ottima A_{ij} per il sottoproblema S_{ij} contiene una qualche attività $a_m \in S_{ij}$, allora $S_{ij} = S_{im} \cup \{a_m\} \cup S_{mj}$ + un insieme di altre attività non compatibili con a_m (e che quindi non possono appartenere ad A_{ij}).
- Quindi una volta scelta a_m, non ci rimane che risolvere i sottoproblemi S_{im} e S_{mj} (le cui attività sono tutte compatibili con a_m).

  Cose da fare:

1. Ridimostrare la sottostruttura ottima, seguendo lo stesso schema visto a lezione, ma tenendo conto del fatto che in questo caso massimizziamo il guadagno (e non il numero di attività). Questo significa che A_{ij} è una soluzione ottima per S_{ij} se contiene solo attività mutuamente compatibili e la somma dei guadagni dovuti alle sue attività è maggiore o uguale ai guadagni relativi ad un qualsiasi altro sottoinsieme di attività compatibili in S_{ij}. Vicersa una soluzione per un generico sottoproblema (ad esempio S_{im}, ma anche S_{mj}) non è ottima se esiste un'altro sottoinsieme di attività ch ci consente un guadagno maggiore.

2. Definire una funzione ricorsiva $v(i, j)$ che calcoli il valore di una soluzione ottima per un generico sottoproblema S_{ij}. La funzione molto simile a quella vista per il problema della moltiplicazione di una sequenza di matrici. Infatti:
   - Se $S_{ij} = \emptyset$ chiaramente $v(i, j) = 0$ (nessuna attività significa nessun guadagno)
   - Se una soluzione ottima per $S_{i,j} \neq \emptyset$ contiene una qualche attività a_m allora il suo valore $v(i, j)$ è pari a

     $$v(i, j) = v(i, m) + v(m, j) + v_m$$

     dove $v(i, m)$ e $v(m, j)$ rappresentano il valori delle soluzioni ottime per i sottoproblemi S_{im} e $_{mj}$, mentre v_i è il guadagno dovuto all'attività a_m. Poichè non sappiamo quale attività in S_{ij} verrà scelta dalla soluzione ottima, (come nel caso della moltiplicazione di matrici) dobbiamo far variare a_m in S_{ij} e prendere il massimo dei valori del tipo $v(i, m) + v(m, j) + v_m$ così ottenuti. Tenete conto del fatto che $a_m \in S_{ij}$ implica $i < m < j$. (Perchè??). Quindi:

     $$v(i, p) = \begin{cases} 0 & \text{se } i \geq j \\ \max\limits_{i<m<j} \{v(i, m) + v(m, p) + v_m\} & \text{se } i < j \end{cases}$$

3. Trovare uno schema bottom-up (homework).