

Esercitazione di programmazione dinamica

- Concentratevi su PD – evitate per oggi approcci diversi!
- Stabilite chiaramente **cosa volete calcolare**, in italiano:
Sia $DP[i]$ il valore massimo che si ottiene con i primi $i \dots$
- Se non riuscite a scriverlo in italiano, **fermatevi!** Non l'avete capito
- Ragionate sulla **sottostruttura ottima**: una soluzione ottima per un sottoproblema può essere utilizzata per trovare la soluzione ottima di un problema più grande?
- Scrivete un'**equazione ricorsiva DP** che calcoli il **valore di una soluzione ottima** a partire dal valore della soluzione più piccola
- Trasformate l'equazione ricorsiva in codice utilizzando **programmazione dinamica** o **memoization**

Ottimizza la somma

Supponete di avere in input un vettore V contenente n interi positivi distinti e un valore intero W . Scrivere un algoritmo

```
int bestSum(int[]  $V$ , int  $n$ , int  $W$ )
```

che

- 1 restituisca il massimo valore $X = \sum_{i=0}^{n-1} x[i]V[i]$ tale che $X \leq W$ e ogni $x[i]$ è un intero non negativo;
- 2 stampi il vettore x .

Ad esempio, per $V = [18, 3, 21, 9, 12, 24]$ e $W = 17$, una possibile soluzione ottima è $x = [0, 2, 0, 1, 0, 0]$ da cui deriva $X = 2 \cdot 3 + 1 \cdot 9 = 15$.

Discutere correttezza e complessità.

Mosse su scacchiera

- Sia P una matrice di dimensione $n \times n$ che rappresenta una **scacchiera** in cui ogni casella (r, c) è associata a un **profitto** $P[r][c]$.
- Considerate un pedone che si può muovere (1) verso l'**alto**, oppure (2) in **diagonale alto-destra**, oppure (3) in **diagonale alto-sinistra**.
- Il pedone parte da una qualunque casella della **riga inferiore** e deve raggiungere una qualunque casella della **riga superiore**.
- Quando il pedone visita una casella (r, c) , **guadagna** il profitto $P[r][c]$.

	0	1	2	3	4
0	6	7	4	7	<u>8</u>
1	7	6	1	1	<u>4</u>
2	3	5	7	<u>8</u>	2
3	2	6	<u>7</u>	0	2
4	7	3	5	<u>6</u>	1

Mosse su scacchiera

Scrivere un algoritmo

```
int searchPath(int[][] P, int n)
```

che restituisca il **massimo profitto ottenibile** (calcolato come la somma dei profitti delle caselle visitate) facendo partire un pedone dalla riga inferiore e raggiungendo una qualunque casella della riga, rispettando le regole di movimento del pedone.

Alcuni suggerimenti:

- Nella matrice/scacchiera, dov'è l'alto? Dov'è il basso? Lo dovete decidere voi!
- La tabella DP è una matrice

I Promessi Sposi

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a restringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte; e il ponte, che ivi congiunge le due rive, par che renda ancor più sensibile all'occhio questa trasformazione, e segni il punto in cui il lago cessa, e l'Adda ricomincia, per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo, lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni."

Quante volte questo testo contiene la sottosequenza "lucia"?

I Promessi Sposi

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte; e il ponte, che ivi congiunge le due rive, par che renda ancor più sensibile all'occhio questa trasformazione, e segni il punto in cui il lago cessa, e l'Adda ricomincia, per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo, lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni."

Quante volte questo testo contiene la sottosequenza "lucia"?

Alcune considerazioni:

- Due sottosequenze sono **distinte** (e quindi vanno contate separatamente) se corrispondono a due **sequenze diverse di indici** del testo, anche se producono la stessa stringa di caratteri.
- Esempio: "**did you go**" contiene due volte la sottosequenza "**dog**"

I Promessi Sposi

Scrivere un algoritmo

```
int lucia(ITEM[] T, ITEM[] P, int n, int m)
```

che

- prenda in input
 - una stringa **testo** *T* lunga *n* caratteri;
 - una stringa **pattern** *P* lunga *m* caratteri;
- restituisca il **numero di volte** che la stringa *P* appare come **sottosequenza distinta** di *T*.

I Promessi Sposi

Scrivere un algoritmo

```
int lucia(ITEM[] T, ITEM[] P, int n, int m)
```

che

- prenda in input
 - una stringa **testo** T lunga n caratteri;
 - una stringa **pattern** P lunga m caratteri;
- restituisca il **numero di volte** che la stringa P appare come **sottosequenza distinta** di T .

Suggerimenti:

- Fate attenzione al ruolo non-simmetrico di i e j : cercare "dog" in una stringa vuota è diverso da cercare una stringa vuota in "dog"

Spoiler alert!

Ottimizza la somma

Questo problema è un caso particolare dello **Zaino senza limiti**:

- gli interi V rappresentano i pesi w : se prendo un intero $V[i]$, riduco la capacità di $V[i]$;
- gli interi V rappresentano i profitti w : se prendo un intero $V[i]$, guadagno $V[i]$;
- la capacità è data da W .

È quindi possibile invocare il codice che abbiamo discusso a lezione, passando il vettore V sia come peso che come profitto, e passando W come capacità.

```
int bestSum(int[] V, int n, int W)
```

```
return knapsack(V, V, n, W)
```

La complessità è $O(nW)$.

Ottimizza la somma

Risolviamo invece il problema "da zero".

Sia $DP[i][w]$ il massimo valore che posso ottenere utilizzando i primi i interi del vettore V (selezionabili più di una volta) e avendo come valore limite massimo w .

$$DP[i][w] = \begin{cases} -\infty & i \geq 0 \wedge w < 0 \\ 0 & i = 0 \vee w = 0 \\ \max\{DP[i-1][w], DP[i][w-V[i-1]] + V[i-1]\} & \text{altrimenti} \end{cases}$$

La complessità (spaziale e temporale) della soluzione è $O(nW)$. Si può adattare un approccio simile a quello visto per lo zaino senza limiti per ridurre la complessità spaziale a $O(W)$.

Ottimizza la somma

```
int bestSum(int[] V, int n, int W)
```

```
% Crea un vettore DP inizializzato a -1  
int[][] DP = new int[0...n][0...W] = {-1}  
return bsRec(V, n, W, DP)
```

```
int bsRec(int[] V, int i, int w, int[][] DP)
```

```
if w < 0 then  
    | return  $-\infty$   
else if i == 0 or w == 0 then  
    | return 0  
else  
    | if DP[i][w] < 0 then  
        | DP[i][w] =  
            |  $\max(\text{bsRec}(V, i - 1, w, DP), \text{bsRec}(V, i, w - V[i - 1], DP) + V[i - 1])$   
return DP[i][w]
```

Mosse su scacchiera (Soluzione 2.7 in 13-pd.pdf)

Sia $DP[r][c]$ il massimo guadagno che si può ottenere partendo da una cella (r, c) e arrivando ad una cella della riga superiore ($r = 0$), dove r e c rappresentano la riga e la colonna, rispettivamente. Il valore di ritorno corrisponderà al valore massimo dell'ultima riga ($r = n - 1$).

Per esercizio, scrivere un algoritmo che restituisca anche il percorso, e non solo il valore massimo.

$$DP[r][c] = \begin{cases} -\infty & c < 0 \text{ or } c > n - 1 \\ P[0][c] & r = 0 \text{ and } 0 \leq c < n \\ \max \begin{pmatrix} DP[r - 1][c - 1], \\ DP[r - 1][c], \\ DP[r - 1][c + 1] \end{pmatrix} + P[r][c] & 1 \leq r < n - 1 \text{ and } 0 \leq c < n \end{cases}$$

La complessità dell'algoritmo risultante, che deve riempire ogni cella della tabella, é $\Theta(n^2)$.

Mosse su scacchiera

```
int searchPath(int[][] P, int n)

---

int DP = new int[1...n][1...n]  
for c = 0 to n - 1 do                                % Copia la prima riga nel vettore DP  
     $DP[0][c] = P[0][c]$   
for r = 1 to n - 1 do  
    for c = 0 to n - 1 do  
         $DP[r][c] = -\infty$   
        foreach  $d \in \{-1, 0, +1\}$  do  
            int newc = c + d  
            if  $0 \leq \textit{newc} < n$  then  
                 $DP[r][c] = \max(DP[r][c], DP[r-1][\textit{newc}] + P[r][c])$   
return  $\max(DP[n-1])$                                 % Restituisce il massimo dell'ultima riga
```

I Promessi Sposi

Sia $DP[i][j]$ il numero di occorrenze del prefisso j -esimo del pattern $P(j)$ come sottosequenza del prefisso i -esimo del testo $T(i)$.

$$DP[i][j] = \begin{cases} 0 & i = 0 \text{ and } j > 0 \\ 1 & j = 0 \\ DP[i-1][j] + DP[i-1][j-1] & i > 0 \text{ and } j > 0 \text{ and } T[i] = P[j] \\ DP[i-1][j] & i > 0 \text{ and } j > 0 \text{ and } T[i] \neq P[j] \end{cases}$$

- Se il testo è finito ($i = 0$) e il pattern non è vuoto ($j > 0$), non siamo riusciti a trovare il pattern
- Se il pattern è vuoto ($j = 0$), significa che siamo riusciti a trovare il pattern, e lo contiamo per uno
- Se l'ultimo carattere del testo e del pattern sono uguali, possiamo sfruttare quest'uguaglianza oppure no; i due casi vanno sommati
- Se l'ultimo carattere del testo e del pattern sono diversi, ignoriamo l'ultimo carattere del testo.

I Promessi Sposi

Utilizziamo un vettore $DP[0 \dots m]$ invece che una matrice, in quanto il valore si ottiene a partire dalla sola riga precedente, che viene memorizzata in DP' .

Complessità in tempo: $\Theta(mn)$, complessità in spazio $\Theta(m)$

```
int lucia(ITEM[] T, ITEM[] P, int n, int m)
```

```
int[] DP = new int[0...m]
```

```
int[] DP' = new int[0...m]
```

```
DP[0] = 1
```

```
for j = 1 to m do DP[j] = 0
```

```
for i = 1 to n do
```

```
    for j = 0 to m do DP'[j] = DP[j]
```

```
    for j = 1 to m do
```

```
        if T[i] == P[j] then
```

```
            | DP[j] = DP'[j] + DP'[j - 1]
```

```
        else
```

```
            | DP[j] = DP'[j]
```

```
return DP[m]
```

5 Maggio e Promessi Sposi

Subject: Sottosequenza Promessi sposi

Date: Fri, 11 Dec 2015 00:25:41 +0100

To: Alberto Montresor <alberto.montresor@unitn.it>

Il 5 maggio nei promessi sposi senza considerare spazi, punteggiatura e numeri, ma considerando gli accenti ci sta:

```
21975465301516630979573617593825769513857583563025262379789778337947615817191757
43398428321975621542396623347442197637158184228160098596758725678010178001365659
73566045203119340799723612777962220975263078675519750712637479432237655210391918
48601874737423942438531018213728179566210700422537584195776715536664949343794694
74341304486367721199869205484517178136400988317581077715393614892844560303556628
57753722957658724970416074137670807561854959314707635812057348445333068267511860
81613043221797286605904378408112853795888506693820006728695515750235630153301285
93082577269020619288952011873970086359263850877345042074027243309950696549344467
34109698508036913355586284550592994928187284736568396263368466837671143105426910
99608601301418040383501823489133955578653269527834272234364741431604038516826654
82045722458282856692545688317000656065623166181081105288235575975572352074726528
75237693750708795898738364470324968401496146509587976160485483512050002468507459
5216118769351618590697862390987987264814086004487458833092023307
```

Allego lo script in python che ha generato il risultato (con testo completo dei promessi sposi e del 5 maggio). Ovviamente ha tempi assurdi per svariati motivi.