

Luiss

Libera Università Internazionale
degli Studi Sociali Guido Carli

Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica

Programmazione dinamica

Irene Finocchi

27 marzo 2023

LUISS






Reminder: Discord, OrientationLuiss

- <https://discord.gg/2TTJCgrX>
 - #domande-generalì (chat lezione)
 - #codice (share your code here!)
- <https://orientation.luiss.it/>
 - Forum Tematico (share your code offline here!)
 - Materiale didattico (slide ed esercizi)
- La chat di Webex non sarà monitorata!

Lezioni

Puoi accedere alle lezioni tramite YouTube o Webex:

 **YouTube** (preferibile se hai problemi di connessione)

 **webex** (limitata a 1000 utenti; però puoi diventare **Chad (Stacy) per un giorno** condividendo la tua soluzione ai problemi che consideriamo di volta in volta!)



Chi vuol essere PetaChad?

- Finora: Chad, GigaChad, TeraChad
- Oggi è il PetaChad day: potete essere 1000 volte TeraChad!

10^{15}	peta	1 000 000 000 000 000
10^{12}	tera	1 000 000 000 000
10^9	giga	1 000 000 000

Regole delle FantaOlimpiadi

Punti bonus:

- Rispondere a domande sulla chat: **+5**
- Chad/Stacy per un giorno: **+10**
- Chad/Stacy per un giorno ringrazia professori Luiss: **+10**
- Superamento territoriali: **+30**
- Fase finale: Oro **+100** , Argento **+70**, Bronzo **+50**

Punti minus:

- Stare nella chat sbagliata: **-10**
- Dire "Fantaolimpiadi" su chat/video: **-10**
- Scrivere "buon pomeriggio" in chat appena inizia la lezione: **-5**
- Chiedere se si può usare Python alle territoriali: **-10**

Programma di oggi (ambiziosissimo!)

1. Soluzione dei problemi della volta scorsa (almeno due...)
2. Ingredienti della programmazione dinamica: torniamo a Fibonacci
3. Confronto tra tecniche
4. Esempi più avanzati di programmazione dinamica
 - Il problema dello zaino
 - Un problema dalle Territoriali (2007): Lino il giornalista
 - Sottovettore di somma massima
 - Distanza tra stringhe
 - Un problema dalle Territoriali (2016): Discesa massima
5. Esercizi per casa

**Soluzione di (alcuni degli) esercizi della scorsa
settimana
Ricorsione e Divide et impera**



Domino massimale (ricorsione)

<https://training.olinfo.it/#/task/domino/statement>

Missioni segrete (ricorsione)

<https://training.olinfo.it/#/task/missioni/statement>

Controllo degli estintori (divide-et-impera)

https://training.olinfo.it/#/task/ois_estintori/statement

Allenamento su China Forces (divide-et-impera)

https://training.olinfo.it/#/task/preoii_allenamento/statement

Scrivete in chat
per esporre la
vostra soluzione

Programmazione Dinamica
Visita guidata molto semplificata...
ancora Fibonacci!

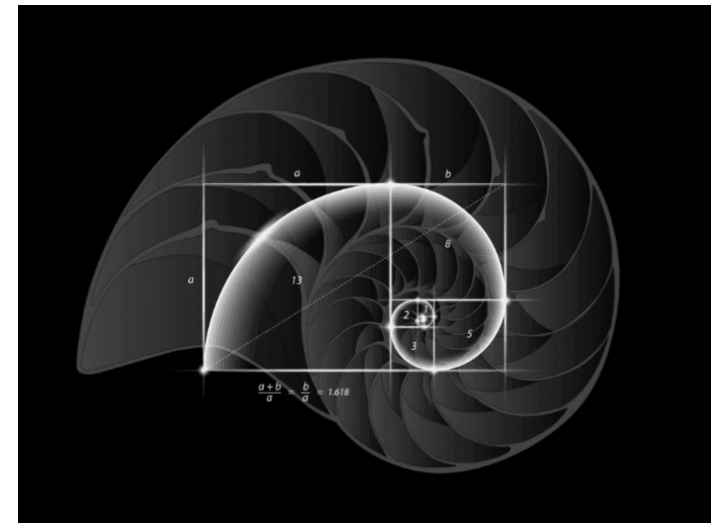


Numeri di Fibonacci

- Ormai sapete tutto, ma ricapitoliamo...
- Definiti come relazione di ricorrenza:

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{se } n \geq 3 \\ 1 & \text{se } n = 1, 2 \end{cases}$$

- Problema: come calcolare F_n ?



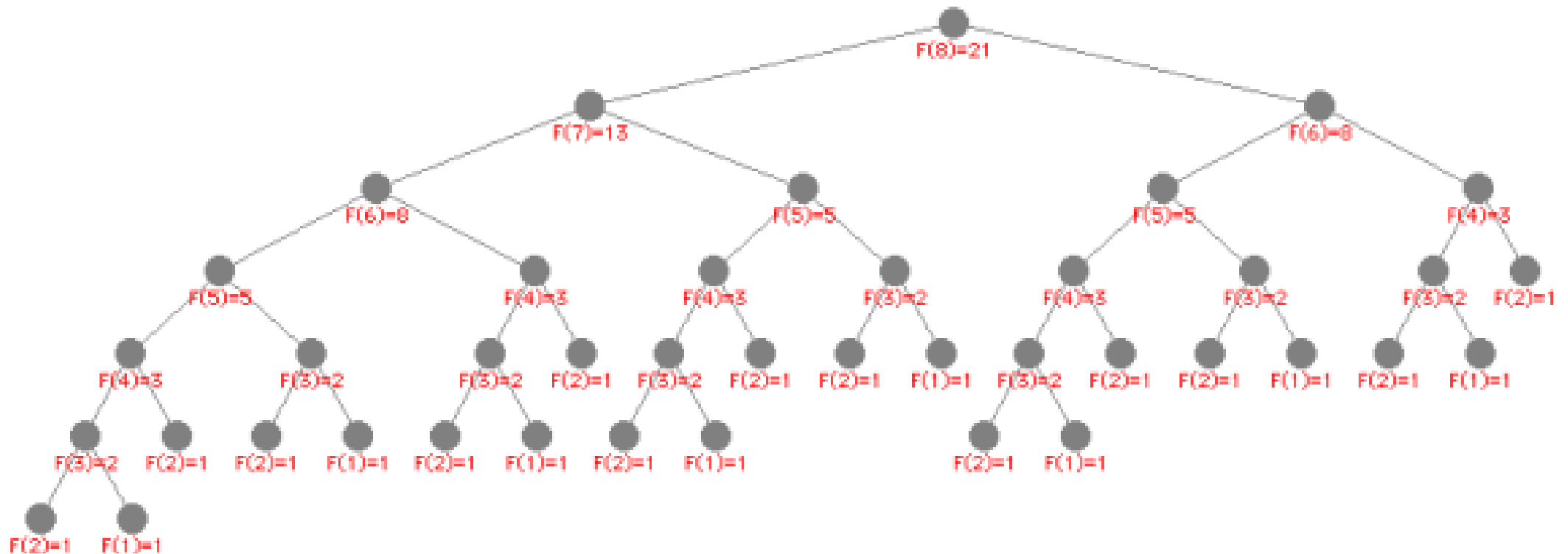
Algoritmo fibonacci ricorsivo

Sembra naturale utilizzare direttamente la definizione (ricorsiva):

```
algoritmo fibonacci_ric(intero  $n$ )  $\rightarrow$  intero  
  if ( $n \leq 2$ ) then return 1  
  else return fibonacci_ric( $n-1$ ) + fibonacci_ric( $n-2$ )
```

E' una soluzione accettabile?

No! Continuiamo a risolvere lo stesso sottoproblema!



Programmazione Dinamica

Per evitare di risolvere più volte lo stesso sottoproblema, sembra naturale **memorizzare la soluzione dei vari sottoproblemi in un array.**

Memorizza F_i in $\text{Fib}[i]$

Per calcolare F_i ($\text{Fib}[i]$) possiamo utilizzare le soluzioni memorizzate in $\text{Fib}[i-1]$ e $\text{Fib}[i-2]$

1	1	2	3	5	8	13	21	34	55	89	144
1	2	3	4	5	6	7	8	9	10	11	12

Algoritmo fibonacci_prog_dinamica

```
algoritmo fibonacci_prog_dinamica(intero n) → intero  
  sia Fib un array di n interi  
  Fib[1] ← 1  
  Fib[2] ← 1  
  for i = 3 to n do  
    Fib[i] ← Fib[i-1] + Fib[i-2]  
  return Fib[n]
```

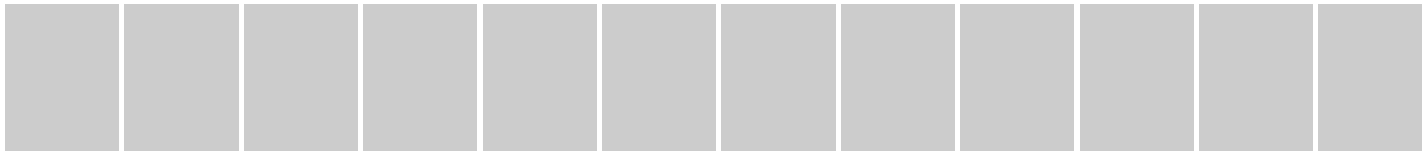
Possiamo semplificare ulteriormente

- Ci serve proprio una tabella così grande (array di dimensione n)?
- Non servono tutti i valori di F_n precedenti, ma solo gli ultimi due: bastano tre variabili!

```
algoritmo fibonacci_prog_din_ottimizzato(intero  $n$ )  $\rightarrow$  intero  
   $a \leftarrow b \leftarrow 1$   
  for  $i = 3$  to  $n$  do  
     $c \leftarrow a + b$   
     $a \leftarrow b$   
     $b \leftarrow c$   
  return  $b$ 
```

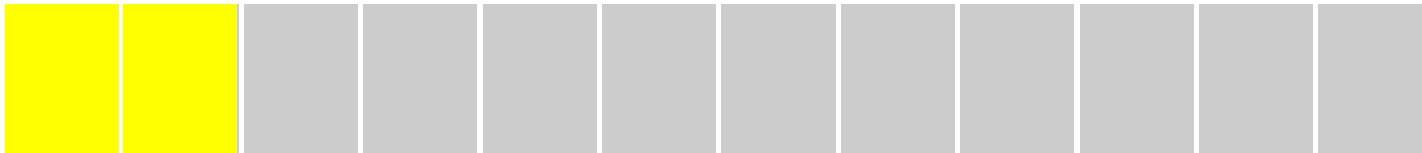

Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)



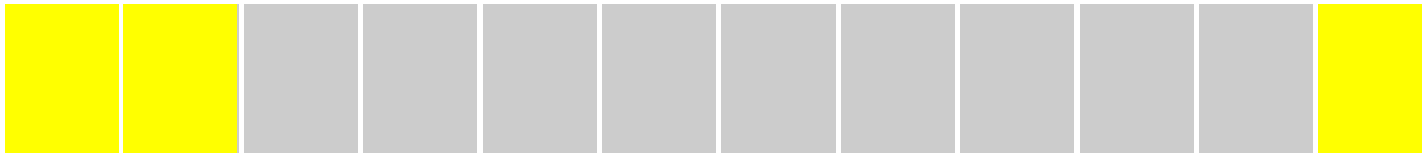
Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella



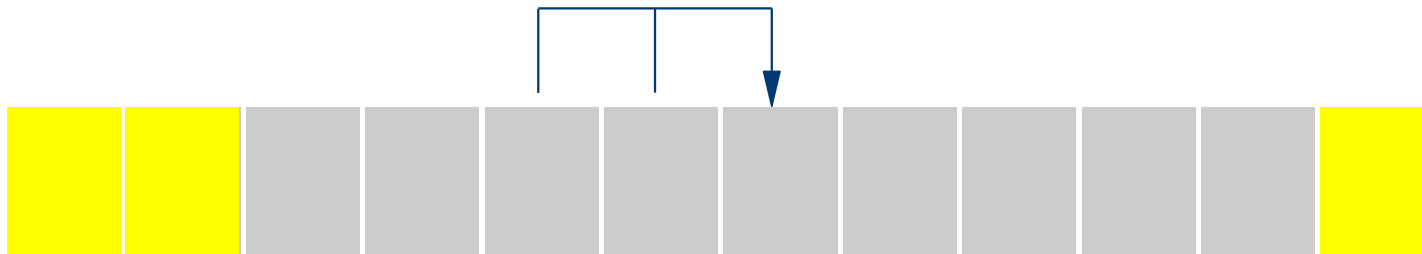
Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella
3. Dove sta la soluzione?



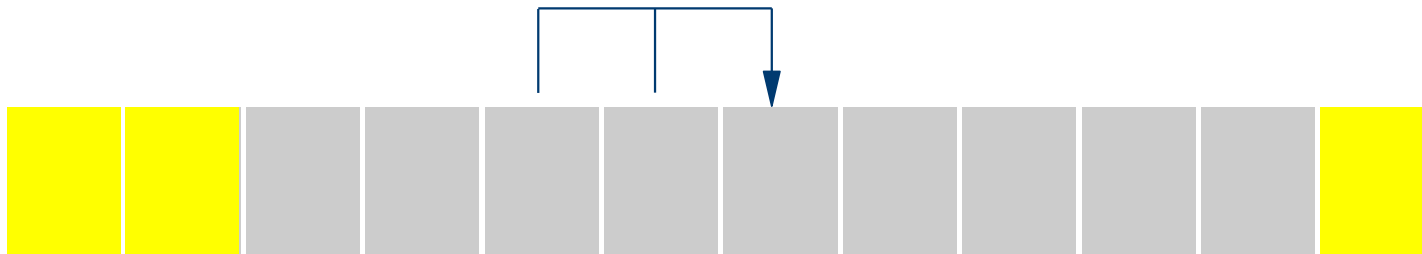
Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella
3. Dove sta la soluzione?
4. Riempire la tabella (da condizioni iniziali a soluzione)



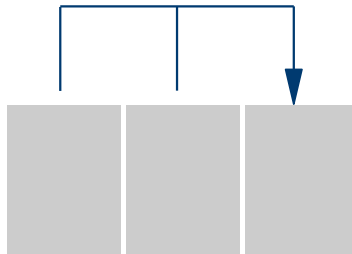
Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella
3. Dove sta la soluzione?
4. Riempire la tabella (da condizioni iniziali a soluzione)
5. Verificare cosa ci serve effettivamente della tabella (ridurne le dimensioni?)



Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella
3. Dove sta la soluzione?
4. Riempire la tabella (da condizioni iniziali a soluzione)
5. Verificare cosa ci serve effettivamente della tabella (ridurne le dimensioni?)



Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella
3. Dove sta la soluzione?
4. Riempire la tabella (da condizioni iniziali a soluzione)
5. Verificare cosa ci serve effettivamente della tabella (ridurne le dimensioni?)

Le 5 Leggi della Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella
3. Dove sta la soluzione?
4. Riempire la tabella (da condizioni iniziali a soluzione)
5. Verificare cosa ci serve effettivamente della tabella (ridurne le dimensioni?)



Punti critici

1. Definire opportunamente sottoproblemi (tabella)
2. Condizioni iniziali della tabella
3. Dove sta la soluzione?
4. Riempire la tabella (da condizioni iniziali a soluzione)
5. Verificare cosa ci serve effettivamente della tabella (ridurne le dimensioni?)

La definizione dei sottoproblemi è il passo più critico

Fatta questa scelta, tutto il resto viene di conseguenza



dynamic programming



All

Books

Images

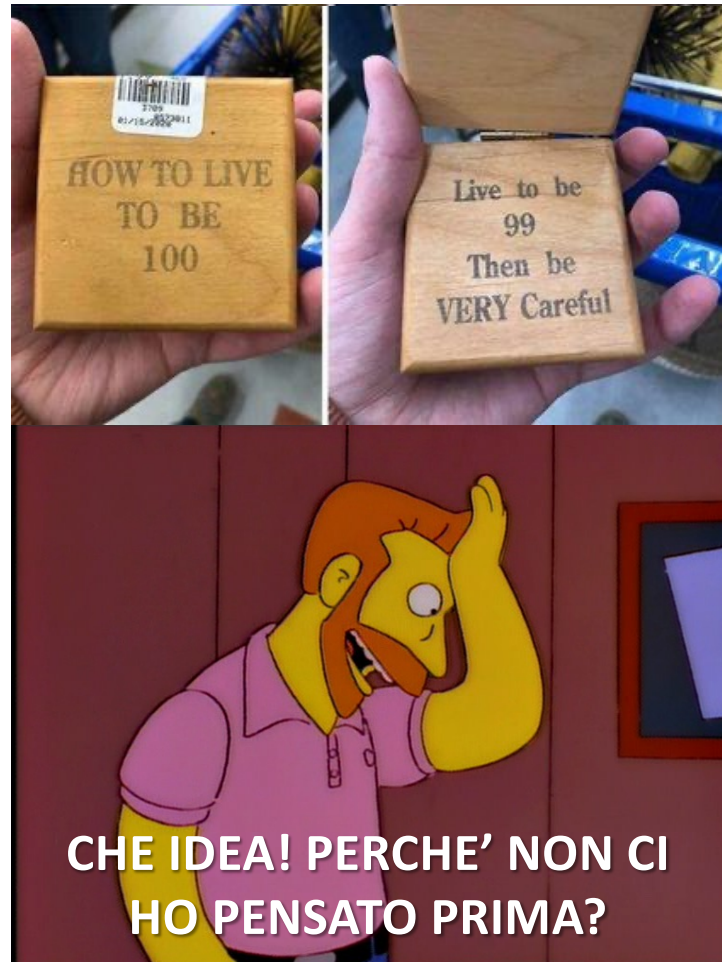
News

Videos

Maps

Shop

Did you mean: ***your worst nightmare***





It is so much easier to write dp
if you think this way.

Tecniche di ottimizzazione a confronto

Greedy vs Divide et impera vs Programmazione dinamica

Greedy	Divide et impera	Programmazione dinamica
Ottimizza facendo la scelta migliore al momento	Ottimizza decomponendo il problema in sottoproblemi più semplici , dello stesso tipo, che vengono risolti ricorsivamente e le cui soluzioni sono ricombinate	Come divide et impera, ma è utile se lo stesso sottoproblema deve essere risolto più e più volte : fa caching delle soluzioni. Implementabile anche bottom up
Correttezza cruciale: non sempre trova l'ottimo, ma molto molto veloce	Trova sempre l'ottimo , ma più lento di greedy	Trova sempre l'ottimo , ma più lento di greedy
Richiede di solito pochissima memoria	Memoria per gestire le chiamate ricorsive (stack della ricorsione)	Richiede molta memoria (vettori, matrici)



Il problema dello zaino

Molte varianti...
Versione non frazionaria



Il problema dello zaino



- Un ladro ha uno zaino che può contenere W Kg di peso
- Ci sono n oggetti tra cui scegliere un sottoinsieme da rubare
- L'oggetto i pesa w_i Kg e vale v_i Euro
- Il ladro deve massimizzare il valore complessivo della refurtiva senza superare il massimo peso W sopportabile dallo zaino
- Gli oggetti vanno presi per intero: non è possibile inserire nello zaino porzioni di oggetti! (Versione non frazionaria)

Greedy funziona?

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$$W = 11$$

- Scelta per **valore decrescente**: {5,2,1} di valore **35**
- Scelta per **peso crescente**: {1,2,3} di valore **25**
- Scelta per **rapporto valore/peso decrescente**: {5,2,1} di valore **35**
 - I rapporti sono 1, 3, 3.6, 3.7, 4
- *Qual è l'ottimo?*



Programmazione dinamica



- Sottoproblemi definiti in termini di due parametri:
 - Insieme degli oggetti tra cui scegliere
 - Peso massimo ammissibile
- $OPT(i, p)$ = valore della soluzione ottima scegliendo un sottoinsieme degli oggetti 1, ..., i ed assumendo un limite p al massimo peso
- Come passiamo da i ad i+1 oggetti per un certo peso ammissibile p?

Da i a $i+1$ oggetti...

Come passiamo da i ad $i+1$ oggetti per un certo peso ammissibile p ?

- 1) Se $w_{i+1} > p$, sicuramente l'oggetto $i+1$ non può essere preso, quindi $OPT(i+1, p) = OPT(i, p)$
- 2) Se $w_{i+1} \leq p$, possiamo inserire o meno l'oggetto:
 - Se lo inseriamo, $OPT(i+1, p) = v_{i+1} + OPT(i, p - w_{i+1})$
 - Se non lo inseriamo, $OPT(i+1, p) = OPT(i, p)$
 - Quale scelta è più conveniente? Prendiamo il **massimo** di queste due quantità!

Esempio

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$W = 11$

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

Esempio

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

$W = 11$

Come si
trovano gli
oggetti
scelti?

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

	0	1	2	3	4	5	6	7	8	9	10	11
{ }	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

Implementazione (bottom up iterativa)

KNAPSACK ($n, W, w_1, \dots, w_n, v_1, \dots, v_n$)

FOR $w = 0$ TO W

$M[0, w] \leftarrow 0.$

FOR $i = 1$ TO n

FOR $w = 0$ TO W

IF ($w_i > w$) $M[i, w] \leftarrow M[i-1, w].$

ELSE $M[i, w] \leftarrow \max \{ M[i-1, w], v_i + M[i-1, w - w_i] \}.$

RETURN $M[n, W].$

Lino il giornalista

Selezioni territoriali 2007



Lino il giornalaio (lino)

Difficoltà $D = 2$

Descrizione del problema

Il giornalaio Lino è un appassionato di matematica e, prima di consegnare il resto ai propri clienti, si diverte a calcolare mentalmente quante differenti possibilità esistono per consegnare tale resto. Ad esempio, considerando l'Euro come valuta, per consegnare 6 centesimi di resto esistono le seguenti 5 possibilità:

- 6 monete da un centesimo,
- 4 monete da un centesimo e 1 da due centesimi,
- 2 monete da un centesimo e 2 da due centesimi,
- 1 moneta da un centesimo e 1 da cinque centesimi,
- 3 monete da due centesimi.

Lino si sta però accorgendo che a causa della lentezza nella consegna del resto sta perdendo molti clienti. Pertanto, aiuta Lino a calcolare il numero di possibili combinazioni.

Dati di input

Il file input.txt contiene nella prima riga un intero positivo N che rappresenta il numero di monete diverse disponibili. La seconda riga contiene un intero positivo R che rappresenta il resto da consegnare al cliente. Ciascuna delle successive N righe contiene un intero positivo che indica il valore di ogni singolo tipo di moneta.

Dati di output

Il file output.txt è composto da una riga contenente un solo intero, che rappresenta il numero di tutte le possibili combinazioni di monete per la consegna del resto R (notare che possono essere usate più copie dello stesso tipo di moneta, per esempio 6 monete da cinque centesimi).

Assunzioni

$1 < N < 100$ e $1 < R < 1000$. I valori dei vari tipi di N monete sono tutti diversi.

Input e output: un esempio

File input.txt	File output.txt
8 6 1 2 5 10 20 50 100 200	5

Programmazione dinamica

- **Sottoproblemi**: il primo contiene solo la prima moneta, il secondo contiene le prime due e così di seguito
- Per ogni problema troviamo la soluzione per ogni possibile valore del resto r , con $1 \leq r \leq R$
- Come è legato un sottoproblema al suo precedente?
- Simile al problema dello zaino!
- Data la soluzione ottima con un insieme di monete $M_i = \{m_1, \dots, m_i\}$ e un resto di dimensione r , vediamo cosa succede aggiungendo una nuova moneta m_{i+1} di taglio t :

tutti i possibili modi di dare resto r con le monete in M_i ci permetteranno ora di dare resto $r+t$ aggiungendo la moneta m_{i+1}

Implementazione

```
1  int monete[100];
2  int soluzioni[1001];
3  int N,R;
4
5  int main()
6  {
7      ifstream in("input.txt");
8      ofstream out("output.txt");
9      in >> N >> R;
10     for (int i=0;i<N;i++)
11         in >> monete[i];
12     for (int i=0; i<=R; i++)
13         soluzioni[i] = 0;
14     soluzioni[0]=1;
15     for (int i = 0; i < N; i++)
16         for (int j = 0; j <= R - monete[i]; j++)
17             soluzioni[j + monete[i]] =
18                 soluzioni[j + monete[i]] + soluzioni[j];
19     out << soluzioni[R];
20     return 0;
21 }
```

Sottovettore di somma massima



Il problema

Dato un vettore V di n interi (positivi o negativi) individuare il sottovettore (posizioni consecutive) la somma dei cui elementi è massima

3	-5	10	2	-3	1	4	-8	7	-6	-1
---	----	----	---	----	---	---	----	---	----	----

Esistono esattamente $n(n-1)/2$ sottovettori di un vettore di lunghezza n , quindi una soluzione che compie circa n^3 passi è banale

Con un minimo di sforzo, possiamo trasformarla in una soluzione che compie circa n^2 passi

Con la programmazione dinamica possiamo ottenere una soluzione lineare

Idea: aggiungiamo un vincolo alla soluzione!

- Problema $P(i)$ che risolveremo, per ogni i tale che $1 \leq i \leq n$:

qual è il valore $s(i)$ del sottovettore con somma massima ***che termina nella posizione i ?***

- $P(1)$ ammette un'unica soluzione, che è banalmente ottima

$$S(1) = V[1]$$

- La soluzione ottima del nostro problema può essere ottenuta calcolando

$$\max_{1 \leq i \leq n} s(i)$$

Tabella e avanzamento

- Memorizziamo i valori $s(i)$ in un **array di dimensione n** , la tabella della programmazione dinamica
- Regola di avanzamento:

$$s(i) = V[i] + \max\{0, s(i - 1)\}$$

- Il sottovettore con somma massima che termina nella posizione i può essere ottenuto **concatenando a $V[i]$ il sottovettore vuoto oppure un sottovettore che termina nella posizione $(i - 1)$**
- Tra tutti i sottovettori terminanti in $(i - 1)$, la scelta migliore è proprio data da quello di somma massima, il cui valore è $s(i - 1)$

Esempio

3	-5	10	2	-3	1	4	-8	7	-6	-1
---	----	----	---	----	---	---	----	---	----	----

Array di input

3	-2	10	12	9	10	14	6	13	7	6
---	----	----	----	---	----	----	---	----	---	---

Tabella di
programmazione
dinamica

Implementazione

algoritmo sottovettoreMax(*array* V , *intero* n) \rightarrow *intero*

1. $S = \text{array di } n \text{ interi}$
2. $S[1] \leftarrow V[1]$
3. $max \leftarrow 1$
4. **for** $i = 2$ **to** n **do**
5. **if** ($S[i - 1] \geq 0$) **then** $S[i] \leftarrow V[i] + S[i - 1]$
6. **else** $S[i] \leftarrow V[i]$
7. **if** ($S[max] < S[i]$) **then** $max \leftarrow i$
8. **return** $S[max]$

Tempo di esecuzione ed occupazione di memoria: lineare $O(n)$

Distanza tra stringhe
(edit distance)



Il problema

Siano X e Y due stringhe di lunghezza m ed n :

$$X = x_1 \cdot x_2 \cdot \dots \cdot x_m \qquad Y = y_1 \cdot y_2 \cdot \dots \cdot y_n$$

Vogliamo calcolare la “distanza” tra X e Y , ovvero il minimo numero delle seguenti operazioni elementari che permetta di trasformare X in Y

- `inserisci(a):` Inserisci il carattere a nella posizione corrente della stringa.
- `cancella(a):` Cancella il carattere a dalla posizione corrente della stringa.
- `sostituisci(a, b):` Sostituisci il carattere a con il carattere b nella posizione corrente della stringa.

Esempio

Azione	Costo	Stringa ottenuta
Inserisco P	1	P RISOTTO
Mantengo R	0	PR ISOTTO
Sostituisco I con E	1	PRE SOTTO
Mantengo S	0	PRES OTTO
Cancello O	1	PRES TTO
Mantengo T	0	PREST TO
Cancello T	1	PREST O
Mantengo O	0	PRESTO

Approccio

- Denotiamo con $\delta(X,Y)$ la distanza tra X e Y
- Definiamo X_i il prefisso di X fino all' i -esimo carattere, per $0 \leq i \leq m$ (X_0 = stringa vuota):

$$X_i = x_1 \cdot x_2 \cdot \dots \cdot x_i \text{ se } i \geq 1$$

- Risolveremo il problema di calcolare $\delta(X,Y)$ calcolando $\delta(X_i,Y_j)$ per ogni i, j tali che $0 \leq i \leq m$ e $0 \leq j \leq n$
- Manterremo le informazioni in una tabella D di dimensione $m \times n$

Inizializzazione della tabella

- Alcuni sottoproblemi sono molto semplici
- $\delta(X_0, Y_j)=j$ partendo dalla stringa vuota X_0 , basta **inserire uno ad uno** i j caratteri di Y_j
- $\delta(X_i, Y_0)=i$ partendo da X_i , basta **rimuovere uno ad uno** gli i caratteri per ottenere Y_0
- Queste soluzioni sono **memorizzate** rispettivamente nella **prima riga** e nella **prima colonna** della tabella D

Esempio

		P	R	E	S	T	O
	0	1	2	3	4	5	6
R	1	1	1	2	3	4	5
I	2	2	2	2	2	1	5
S	3						
O	4						
T	5						
T	6						
O	7						

La tabella D costruita
dall'algoritmo


Inizializzazione

Avanzamento nella tabella (1/3)

- Se $x_i = y_j$, il minimo costo per trasformare X_i in Y_j è uguale al minimo costo per trasformare X_{i-1} in Y_{j-1}

$$D[i,j] = D[i-1, j-1]$$

1	2
1	1



- Se $x_i \neq y_j$, distinguiamo in base all'ultima operazione usata per trasformare X_i in Y_j in una sequenza ottima di operazioni

Avanzamento nella tabella (2/3)

`inserisci(y_j):`

2	3
2	3

il minimo costo per trasformare X_i in Y_j è uguale al minimo costo per trasformare X_i in Y_{j-1} più 1 per inserire il carattere y_j

➡ $D[i,j] = 1 + D[i,j-1]$

`cancella(x_i):`

4	3
5	4

il minimo costo per trasformare X_i in Y_j è uguale al minimo costo per trasformare X_{i-1} in Y_j più 1 per la cancellazione del carattere x_i

➡ $D[i,j] = 1 + D[i-1,j]$

Avanzamento nella tabella (3/3)

sostituisci(x_i, y_j):

1	2
2	2



il minimo costo per trasformare X_i in Y_j è uguale
al **minimo costo per trasformare X_{i-1} in Y_{j-1} più 1**
per sostituire il carattere x_i per y_j

$$D[i,j] = 1 + D[i-1,j-1]$$

In conclusione:

$$D[i,j] = \begin{cases} D[i-1,j-1] & \text{se } x_i = y_j \\ 1 + \min\{D[i,j-1], D[i-1,j], D[i-1,j-1]\} & \text{se } x_i \neq y_j \end{cases}$$

Esempio

		P	R	E	S	T	O
	0	1	2	3	4	5	6
R	1	1	1	2	3	4	5
I	2	2	2	2	3	4	5
S	3	3	3	3	2	3	4
O	4	4	4	4	3	3	3
T	5	5	5	5	4	3	4
T	6	6	6	6	5	4	4
O	7	7	7	7	6	5	4

La tabella D costruita dall'algoritmo

In **grassetto** sono indicate due **sequenze di operazioni** che **permettono di ottenere la distanza** tra le stringhe

Implementazione

```
algoritmo distanzaStringhe(stringa  $X$ , stringa  $Y$ )  $\rightarrow$  intero  
  matrice  $D$  di  $(m + 1) \times (n + 1)$  interi  
  for  $i = 0$  to  $m$  do  $D[i, 0] \leftarrow i$   
  for  $j = 1$  to  $n$  do  $D[0, j] \leftarrow j$   
  for  $i = 1$  to  $m$  do  
    for  $j = 1$  to  $n$  do  
      if  $(x_i \neq y_j)$  then  
         $D[i, j] \leftarrow 1 + \min\{D[i, j - 1], D[i - 1, j], D[i - 1, j - 1]\}$   
      else  $D[i, j] \leftarrow D[i - 1, j - 1]$   
  return  $D[m, n]$ 
```

Tempo di esecuzione ed occupazione di memoria: $O(m \cdot n)$

Discesa massima

Selezioni territoriali 2016

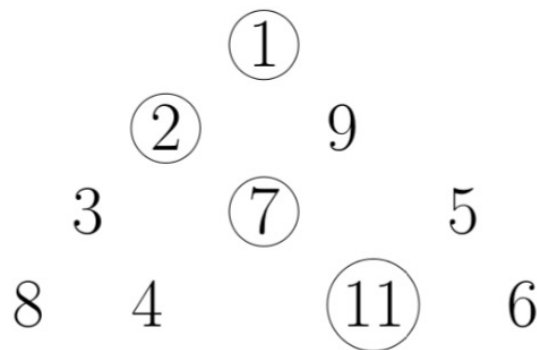


Discesa massima (discesa)

Difficoltà D = 2

Descrizione del problema

Come ben sanno gli studenti che hanno passato le selezioni scolastiche delle Olimpiadi di Informatica di quest'anno, data una piramide di numeri, definiamo una **discesa** come *una sequenza di numeri ottenuti partendo dalla cima della piramide e passando per uno dei due numeri sottostanti, fino a giungere alla base della piramide*. Inoltre, il **valore** di una discesa è definito come la somma dei numeri della discesa. La **discesa massima** di una piramide è quella che ha il massimo valore tra tutte le discese della piramide.



Valore della
discesa
evidenziata:
 $1+2+7+11=21$

Valore della
discesa
massima:
 $1+9+7+11=28$

Dati di input

Il file input.txt è composto da $1 + A$ righe di testo. La prima riga contiene A , un intero positivo rappresentante l'altezza della piramide. Le seguenti A righe descrivono effettivamente la piramide: l' i -esima riga (con i compreso tra 1 e A) contiene i interi positivi rappresentanti l' i -esimo "livello" della piramide.

Dati di output

Il file output.txt è composto da una sola riga contenente un intero positivo: il valore della discesa massima.

Assunzioni

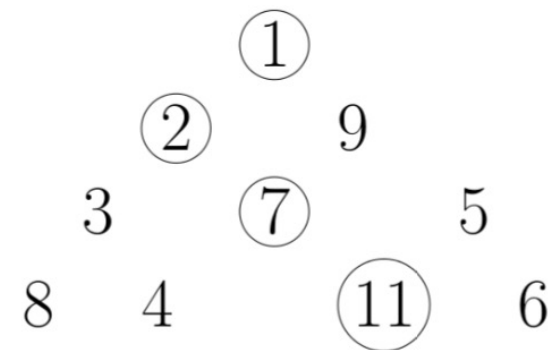
- $1 \leq A \leq 10$.
- Il valore di ciascun numero nella piramide è un intero positivo non superiore a 100.

Input e output: esempi

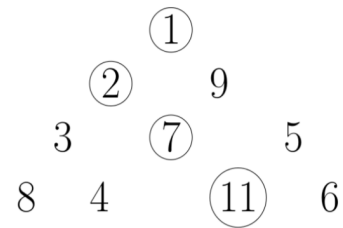
File input.txt	File output.txt
4 1 2 9 3 7 5 8 4 11 6	28
6 42 11 13 41 37 38 5 8 11 9 22 27 31 18 32 12 8 9 8 10 11	145

Riflessioni

- Approccio naturale: scendo verso il figlio massimo (greedy). Funziona?
- Soluzione ricorsiva "pura": prendo il massimo tra il valore della discesa massima a sinistra e il valore della discesa massima a destra.
 - È corretto?
 - È veloce?
 - Si può migliorare?
- Come memorizzo la piramide?



Implementazione



1			
2	9		
3	7	5	
8	4	11	6

$$V(n_{i,j}) = n_{i,j} + \text{Max}(V(n_{i+1,j}), V(n_{i+1,j+1}))$$

```
1 | int discesa_dinamica()
2 | {
3 |     for (int i = A - 2; i >= 0; i--)
4 |         for (int j = 0 ; j< i+1; j++)
5 |             piramide[i][j] += piramide[i+1][j] > piramide[i+1][j+1] ?
6 |                 piramide[i+1][j] : piramide[i+1][j+1];
7 |     return piramide[0][0];
8 | }
```

Esercizi per casa



Esercizi per casa

- Sui tetti di Pisa (Pre-EGOI 2022)
<https://training.olinfo.it/-/task/pre-egoi-parkour/statement>
- Truffa al sushi (Practice round OII 2020)
https://training.olinfo.it/#/task/preoii_sushi/statement
- La coda per il buffet (OII 2022)
https://training.olinfo.it/#/task/oii_coda/statement
- Pub Encounter (OIS 2022)
https://training.olinfo.it/#/task/ois_pickup/statement