

Luiss
Libera Università Internazionale
degli Studi Sociali Guido Carli

Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica

Lezione 5. Programmazione Dinamica

Giuseppe F. Italiano

22 aprile 2021

LUISS



Reminder: Mattermost

1. Non avrai altra chat al di fuori di *mattermost*!
(<https://coding.luiss.ml/mattermost>)
2. Controlla i canali: spesso troverai qui le risposte alle tue domande!
3. Non inquinare / spammare i canali!
Contribuisci a mantenerli puliti e non postare contenuti impropri. Grazie!

▼ CHANNELS

🌐 Announcements

🌐 Chat

🌐 General


🌐 Materials



Reminder: Lezioni

Le lezioni si svolgono giovedì dalle 16 alle 19 (fino al 13 maggio compreso).



Puoi accedere alle lezioni tramite Webex o YouTube:

 (puoi provare l'ebrezza di diventare tu docente per qualche minuto condividendo la tua soluzione ai problemi che consideriamo!)

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

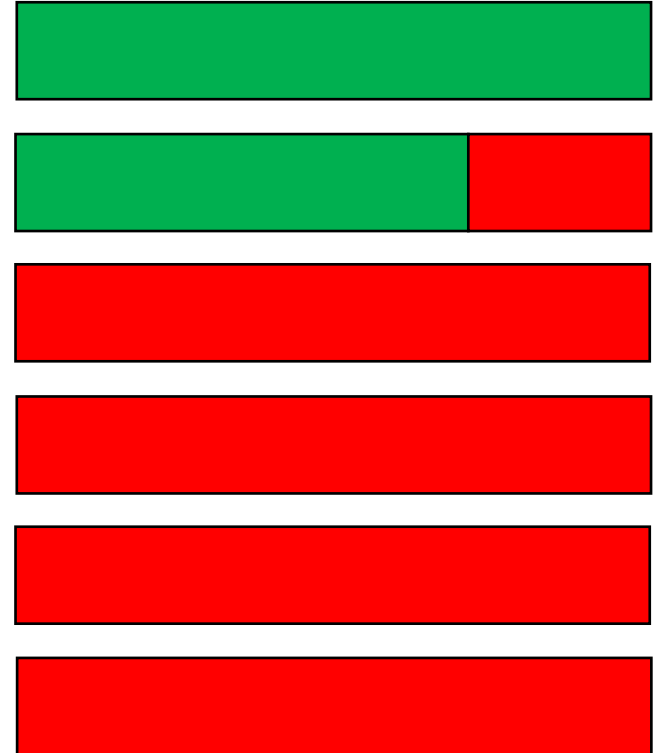
I link delle lezioni sono postati **SOLO** su mattermost (Announcements)

Avviso

- Oggi *NON* correggeremo insieme i compiti a casa.
- Siete bravissimi, e sapete farli da soli. 
- Potete sempre discuterne le soluzioni sulla Chat di mattermost (e anche su questo siete bravissimi!) 
- Meglio dedicare il tempo a disposizione per imparare cose nuove?

Programma di oggi (ambizioso?)

1. Programmazione Dinamica
2. Un problema dalle Territoriali (2004): *Poldo*
3. Un problema dalle Territoriali (2016): *discesa*
4. Un problema dalle Territoriali (2014): *sommelier*
5. Un problema dalle OIS (2020): *fibonaccibug*
6. Un problema delle OIS (2017): *rescaling*
7. Un problema dalle Territoriali (2008): *missioni*
8. Esercizi Olimpiadi (compiti da fare a casa per la prossima volta)



TENETEVI FORTE



TENETEVI FORTE

**ARRIVA LA
PROGRAMMAZIONE DINAMICA**

Programmazione Dinamica









Programmazione Dinamica (visita guidata molto semplificata)

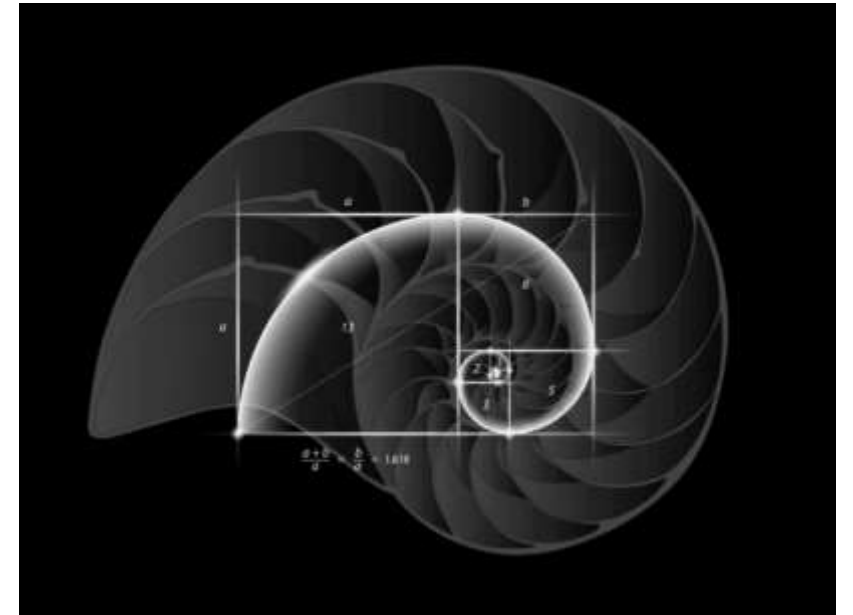


Numeri di Fibonacci

- 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 fibonacci1

Sembra naturale utilizzare direttamente la definizione (ricorsiva):

```
algoritmo fibonacci1(intero n)  $\rightarrow$  intero  
  if (n  $\leq$  2) then return 1  
  else return fibonacci1(n-1) + fibonacci1(n-2)
```

E' una soluzione accettabile?

 recursive-fibonacci.cpp

Possiamo fare di meglio?

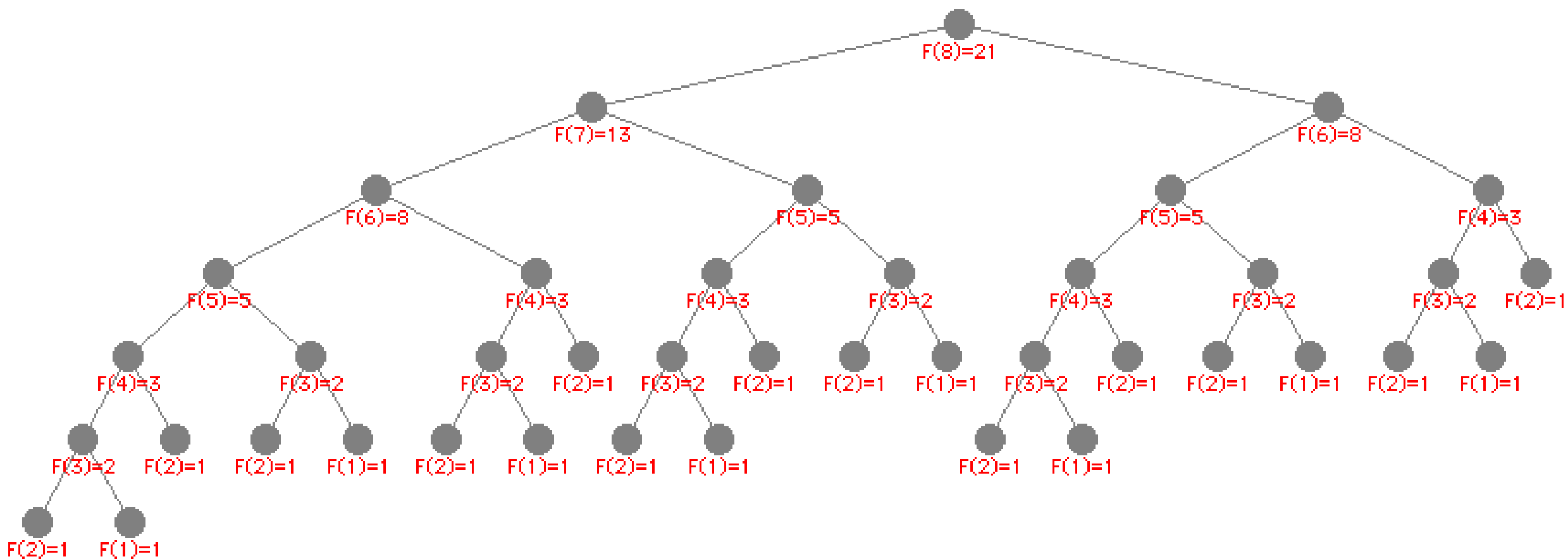
Approccio (doppiamente) ricorsivo produce un algoritmo molto lento:

$$T(n) \approx O(F_n) \approx O(2^n)$$

Perché è lento?

Continua a ricalcolare più volte la soluzione dello stesso sottoproblema!

Albero delle chiamate ricorsive



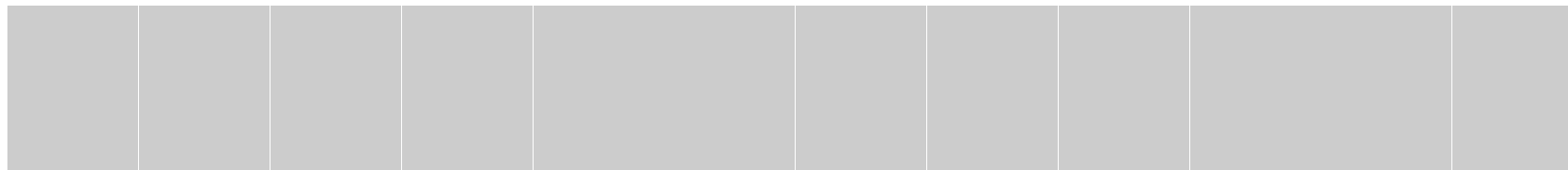
Programmazione Dinamica

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 fibonacci2

Algoritmo fibonacci2

```
algoritmo fibonacci2(intero n) → intero  
  sia Fib un array di n interi  
  Fib[1] ← Fib[2] ← 1  
  for i = 3 to n do  
    Fib[i] ← Fib[i-1] + Fib[i-2]  
  return Fib[n]
```

➡ array-fibonacci.cpp

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 fibonacci3(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  $b$ 
```

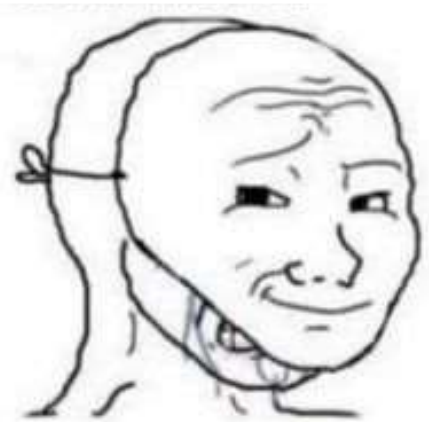
E' tutto qui o possiamo fare di meglio per Fibonacci?



fibonacci6

fibonacci3

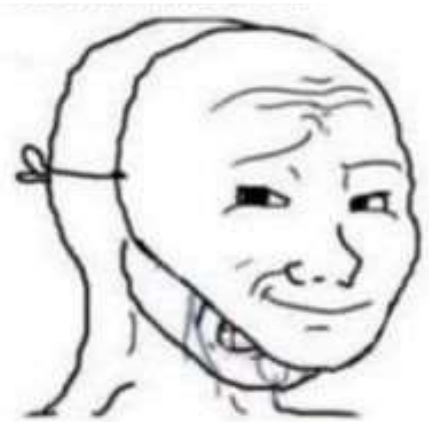
fibonacci2



Fibonacci più veloce?



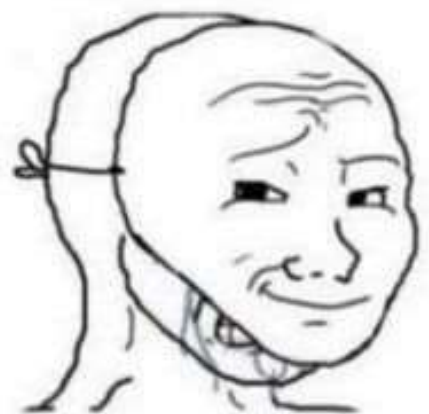
Non c'è tempo! Dobbiamo fare
programmazione dinamica!



Fibonacci più veloce?



Non c'è tempo! Dobbiamo fare
programmazione dinamica!



Faster Fibonacci?

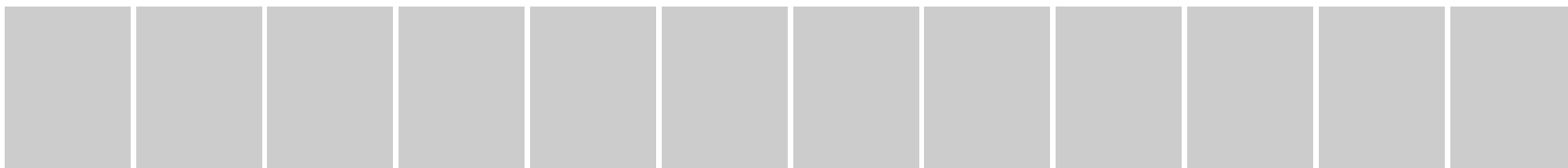
Indian man on YouTube



I will teach you my
son

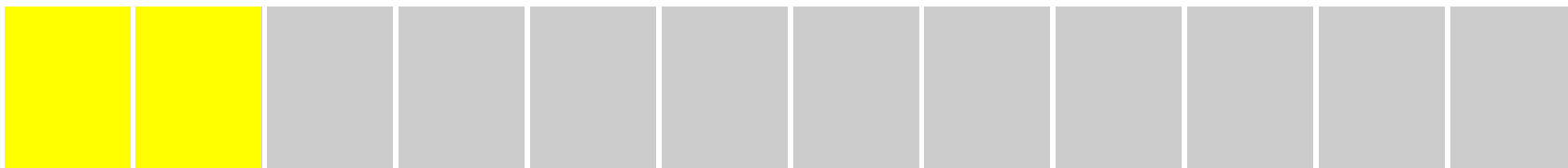
Programmazione Dinamica

1. Definire opportunamente sottoproblemi (tabella)



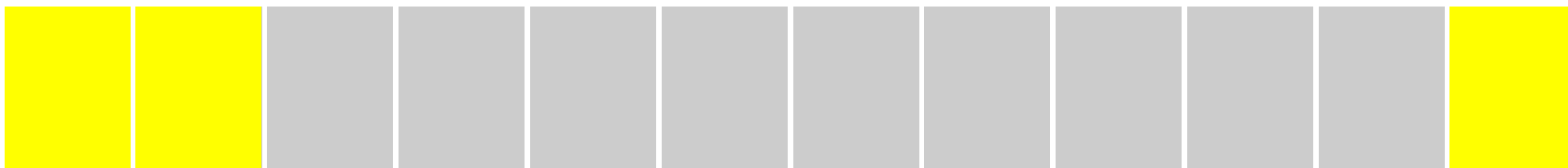
Programmazione Dinamica

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



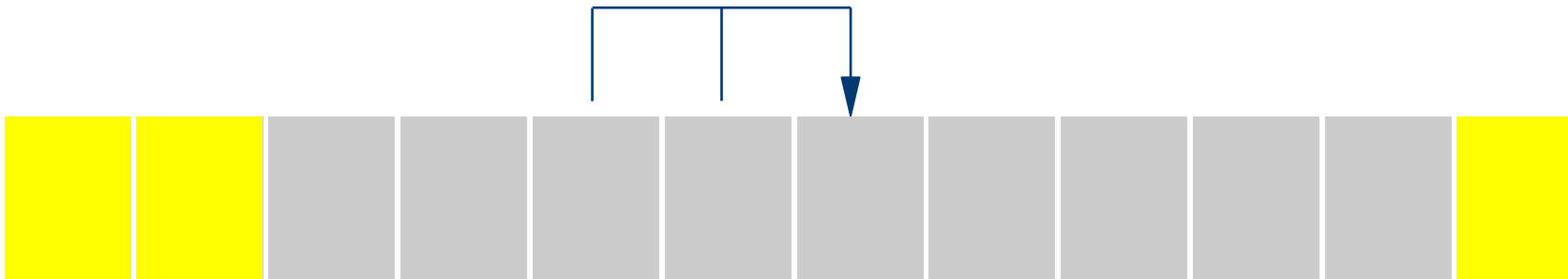
Programmazione Dinamica

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



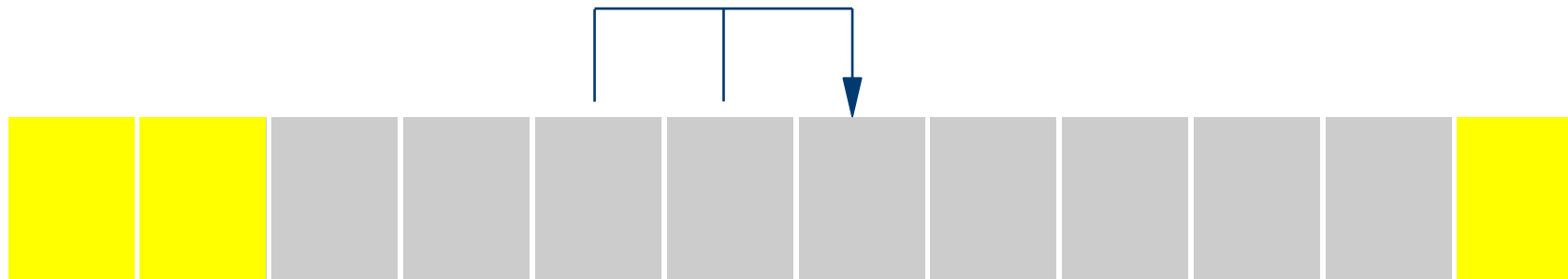
Programmazione Dinamica

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



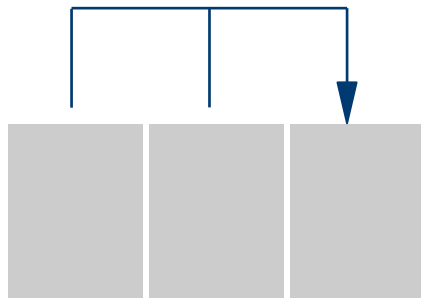
Programmazione Dinamica

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 (ridurre 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?)

Definizione dei sottoproblemi è il passo più critico

Fatta questa scelta, tutto il resto viene di conseguenza



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?)

Definizione dei sottoproblemi è il passo più critico

Fatta questa scelta, tutto il resto viene di conseguenza

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?)

Definizione dei sottoproblemi è il passo più critico

Fatta questa scelta, tutto il resto viene di conseguenza

Come facciamo questa scelta così cruciale?

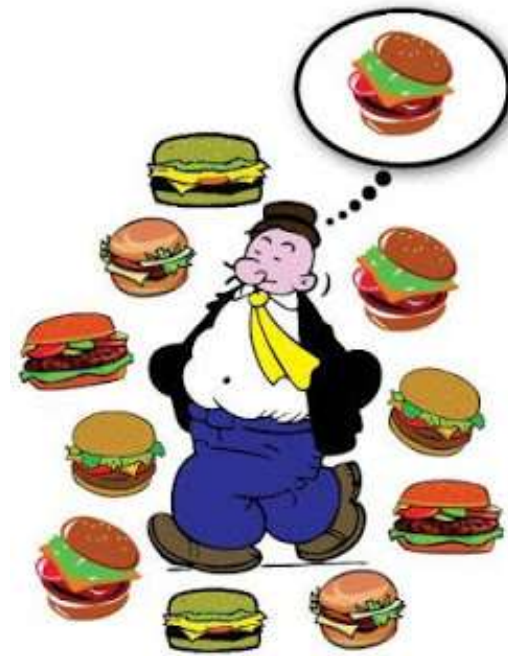


Domande?



Un problema dalle Territoriali (2004): Poldo

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



La dieta di Poldo (Poldo)

Descrizione del problema

Il dottore ordina a Poldo di seguire una dieta. Ad ogni pasto non può mai mangiare un panino che abbia un peso maggiore o uguale a quello appena mangiato. Quando Poldo passeggia per la via del suo paese, da ogni ristorante esce un cameriere proponendo il menù del giorno. Ciascun menù è composto da una serie di panini, che verranno serviti in un ordine ben definito, e dal peso di ciascun panino. Poldo, per non violare la regola della sua dieta, una volta scelto un menù, può decidere di mangiare o rifiutare un panino; se lo rifiuta il cameriere gli servirà il successivo e quello rifiutato non gli sarà più servito.

Si deve scrivere un programma che permetta a Poldo, leggendo un menù, di capire qual è il numero massimo di panini che può mangiare per quel menù senza violare la regola della sua dieta.

Riassumendo, Poldo può mangiare un panino se e solo se soddisfa una delle due condizioni:

- il panino è il primo che mangia in un determinato pasto;
- il panino non ha un peso maggiore o uguale all'ultimo panino che ha mangiato in un determinato pasto.

Dati di input

La prima linea del file `input.txt` contiene il numero m di panini proposti nel menù. Le successive m linee contengono un numero intero non negativo che rappresenta il peso del panino che verrà servito. I panini verranno serviti nell'ordine in cui compaiono nell'input.

Dati di output

Il file `output.txt` contiene il massimo numero di panini che Poldo può mangiare rispettando la dieta.

Assunzioni

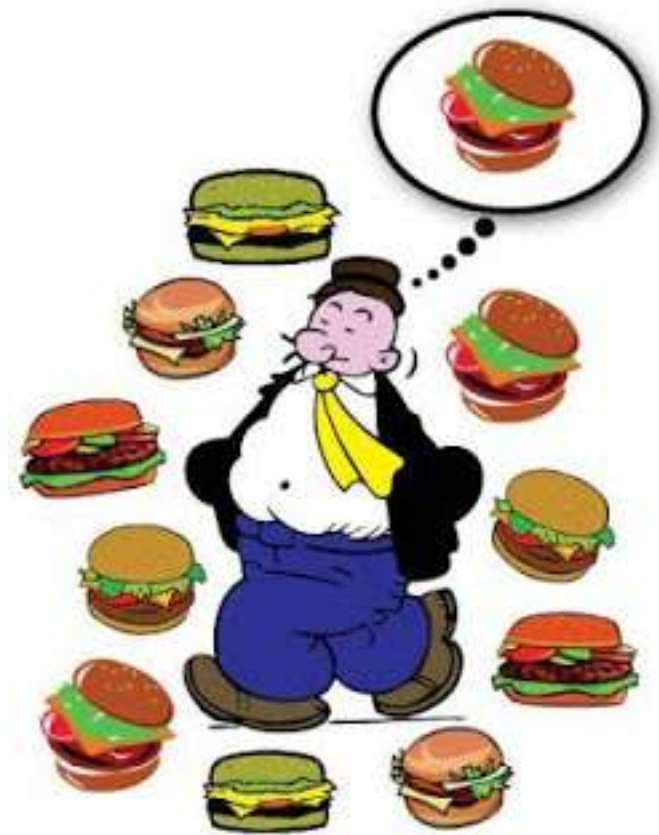
- I pesi dei panini sono espressi in grammi, e un panino pesa al massimo 10.000 grammi.
- $1 \leq m \leq 10.000$

Esempi di input/output

File input.txt	File output.txt
8 389 207 155 300 299 170 158 65	6
File input.txt	File output.txt
3 22 23 27	1

File input.txt	File output.txt
22 15 14 15 389 201 405 204 130 12 50 13 26 190 305 25 409 3011 43 909 987 1002 900	6

Poldo: Soluzione



Risoluzione: Programmazione Dinamica

0	1	2	3	4	5	6	7	8	9
8	38	20	15	30	29	18	16	9	10

1. Qual è il sottoproblema da risolvere?

Come succede di solito (anche con la programmazione dinamica), ci sono varie possibilità...

Risoluzione: Programmazione Dinamica

0	1	2	3	4	5	6	7	8	9
8	38	20	15	30	29	18	16	9	10

Approccio 1

Più intuitivo, approccio più o meno standard

Produrrà soluzione $O(n^2)$

Tutto sommato, molto semplice

Non è la soluzione più veloce. Ce la farà a passare tutti i test?
(OK per le territoriali)



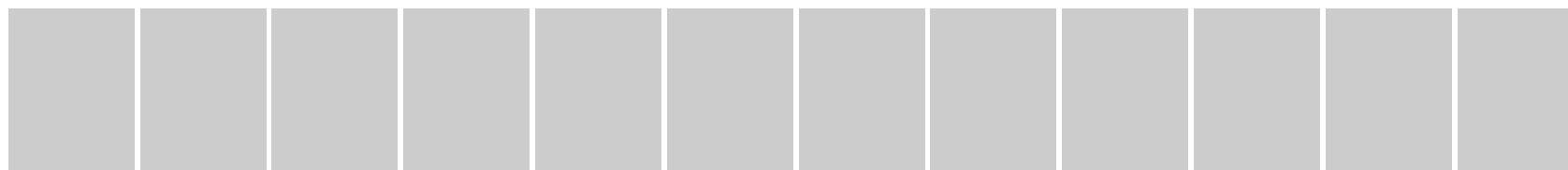
Risoluzione: Programmazione Dinamica

0	1	2	3	4	5	6	7	8	9
8	38	20	15	30	29	18	16	9	10




1. Qual è il sottoproblema da risolvere?

Approccio 1: Quanti panini può mangiare al massimo Poldo nell'intervallo $[1, i]$.



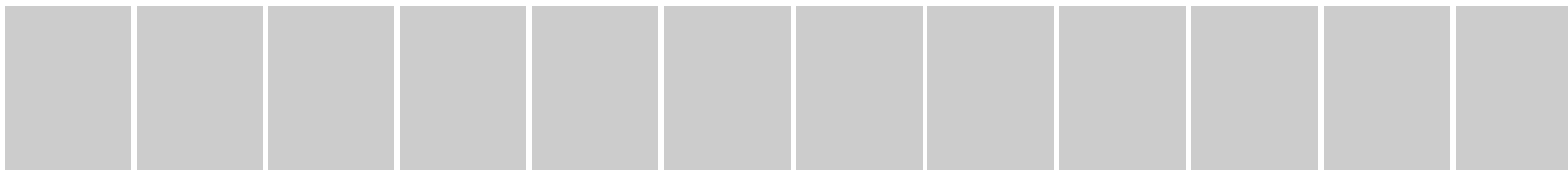
Risoluzione: Programmazione Dinamica

0	1	2	3	4	5	6	7	8	9
8	38	20	15	30	29	18	16	9	10




1. Qual è il sottoproblema da risolvere?

Approccio 1: Quanti panini può mangiare al massimo Poldo nell'intervallo $[1, i]$ se decide di mangiare il panino i .



Risoluzione: Programmazione Dinamica

0	1	2	3	4	5	6	7	8	9
8	38	20	15	30	29	18	16	9	10

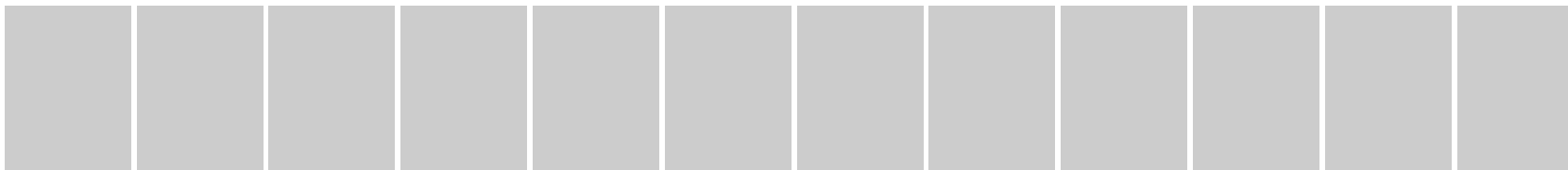


1. Qual è il sottoproblema da risolvere?

Approccio 1: Quanti panini può mangiare al massimo Poldo nell'intervallo $[1, i]$ se decide di mangiare il panino i .

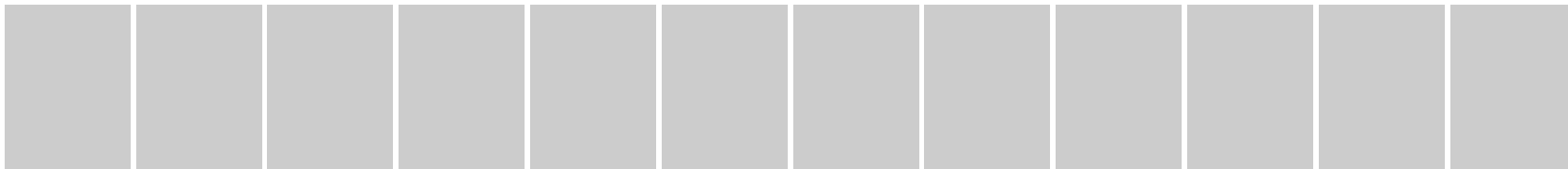
La tabella quindi sarà semplicemente un array *quantiPanini[]*.

- Darà luogo a una soluzione quadratica standard ($O(n^2)$)
- Scriveremo codice che si basa su questo approccio
- E' possibile trovare una soluzione $O(n \log n)$. Riuscite a trovarla?



Risoluzione: Programmazione Dinamica

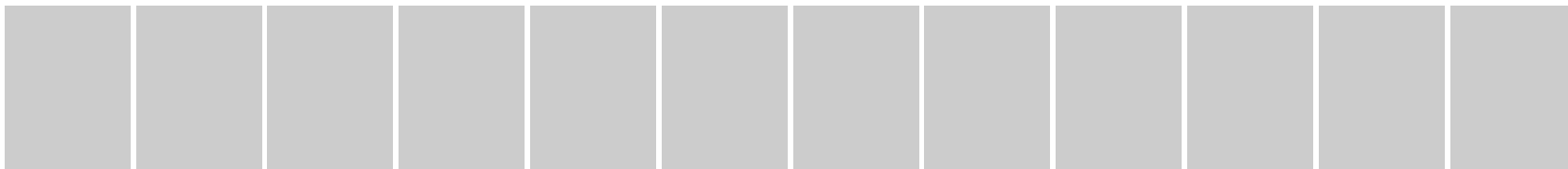
2. Condizioni iniziali?



Risoluzione: Programmazione Dinamica

2. Condizioni iniziali?

Sottoproblema per la prima posizione, la cui soluzione è 1: Se ho solo un panino a disposizione, lo mangio. *quantiPanini[0]=1*

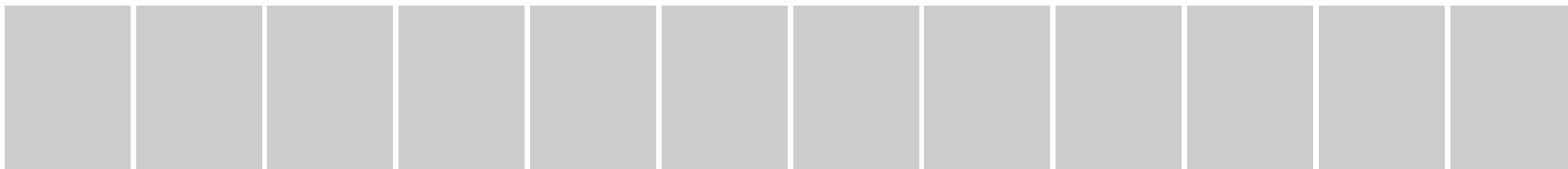


Risoluzione: Programmazione Dinamica

2. Condizioni iniziali?

Sottoproblema per la prima posizione, la cui soluzione è 1: Se ho solo un panino a disposizione, lo mangio. *quantiPanini[0]=1*

3. Dove si trova la soluzione?



Risoluzione: Programmazione Dinamica

2. Condizioni iniziali?

Sottoproblema per la prima posizione, la cui soluzione è 1: Se ho solo un panino a disposizione, lo mangio. $quantiPanini[0]=1$

3. Dove si trova la soluzione?

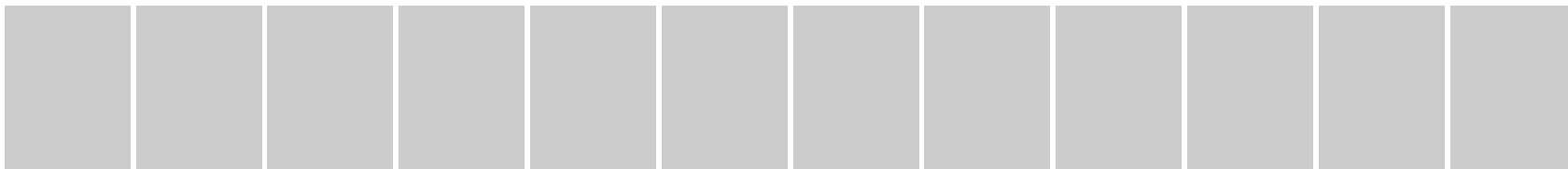
L'output del problema sarà il massimo tra tutti i sottoproblemi:

$$\max_i \{ quantiPanini[i] \}$$

Nota che la soluzione non è memorizzata in un punto specifico della tabella, ma la devo andare a cercare!!!

Risoluzione: Programmazione Dinamica

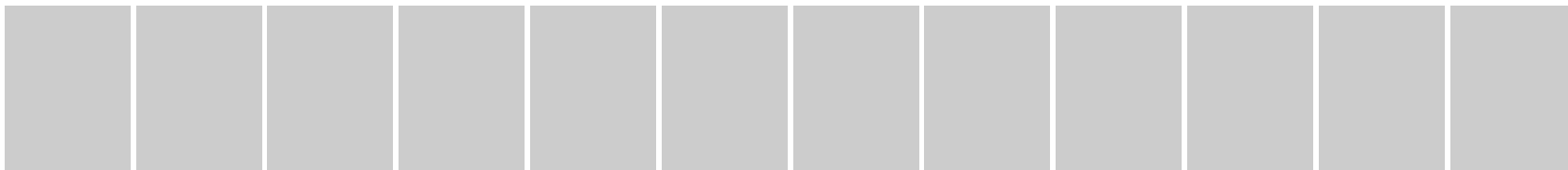
4. Come riempiamo la tabella?



Risoluzione: Programmazione Dinamica

4. Come riempiamo la tabella?

Per trovare il valore nella posizione i (quanti panini posso mangiare al massimo se mangio il panino i), esamino qualsiasi $j < i$ e calcolo quanti panini posso mangiare se mangio il panino j e subito dopo il panino i (ammesso che ciò sia possibile).



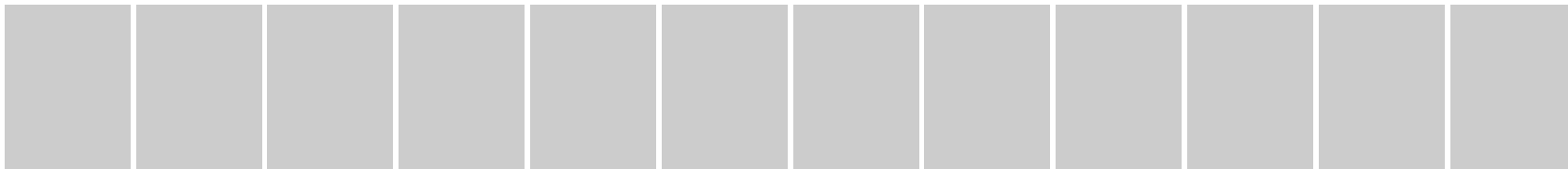
Risoluzione: Programmazione Dinamica

0	1	2	3	4	5	6	7	8	9
8	38	20	15	30	29	18	16	9	10

quantiPanini[5] può essere *quantiPanini[1] + 1* oppure *quantiPanini[4] + 1*

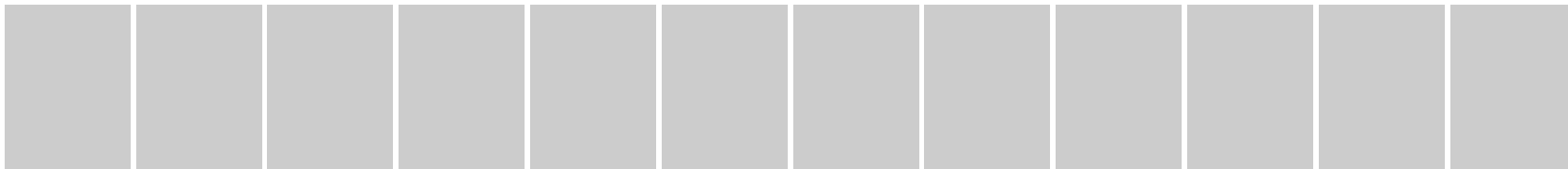
Risoluzione: Programmazione Dinamica

5. Possiamo eliminare qualcosa dalla tabella?



Risoluzione: Programmazione Dinamica

5. Possiamo eliminare qualcosa dalla tabella?
Non sembra molto semplice.



Implementazione

TENETEVI FORTE



TENETEVI FORTE

**ANCORA PIÙ
PROGRAMMAZIONE DINAMICA**

Un problema dalle Territoriali (2016): Discesa massima

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

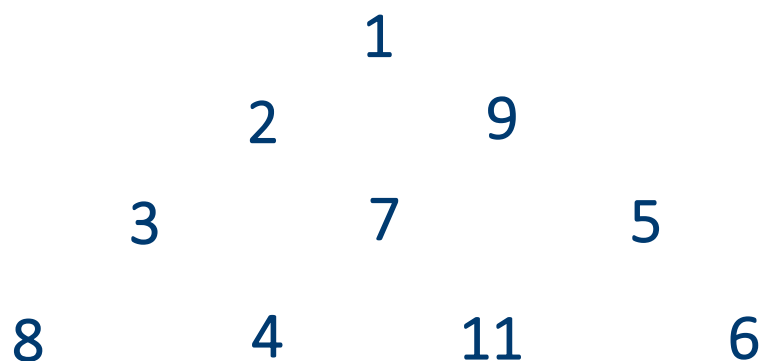


Discesa massima (discesa)

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.



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

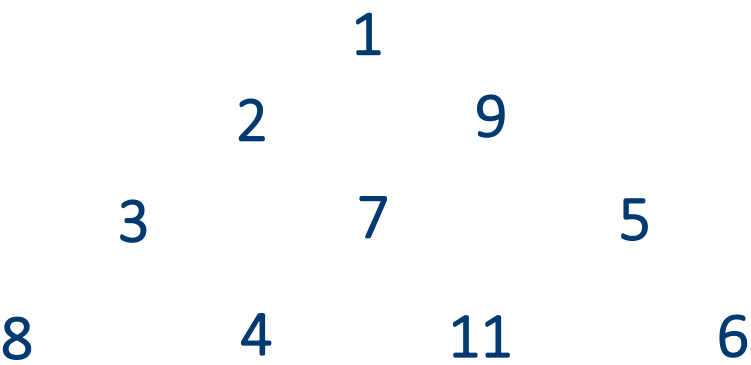
- $1 \leq A \leq 10$.
- Il valore di ciascun numero nella piramide è un intero positivo non superiore a 100.

Input e output: esempi

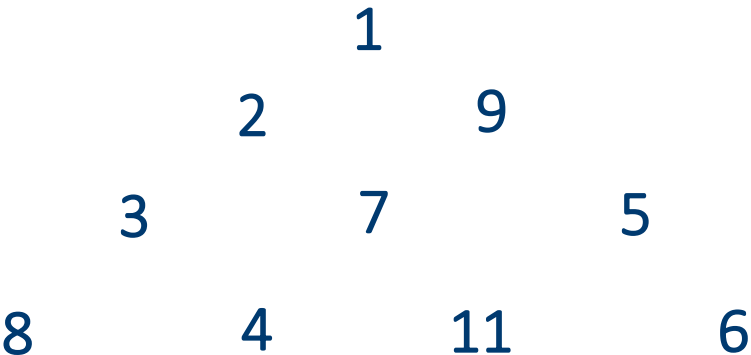
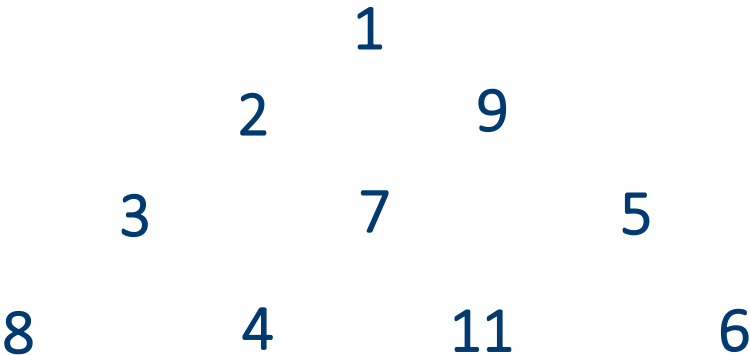
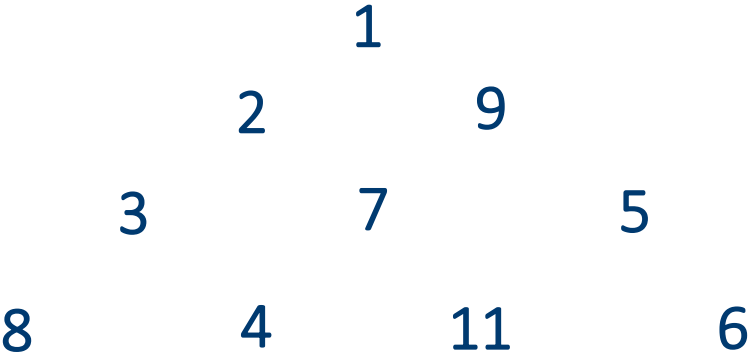
File input.txt	File output.txt
4 1 2 9 3 7 5 8 4 11 6	28
6 42 11 13 41 37 38 5 8 11 9 22 27 31 18 32 12 8 9 8 10 11	145

File input.txt	File output.txt
4 1 2 9 3 7 5 8 4 11 6	28

File input.txt	File output.txt
4 1 2 9 3 7 5 8 4 11 6	28



File input.txt	File output.txt
4 1 2 9 3 7 5 8 4 11 6	28



6	145
42	
11 13	
41 37 38	
5 8 11 9	
22 27 31 18 32	
12 8 9 8 10 11	

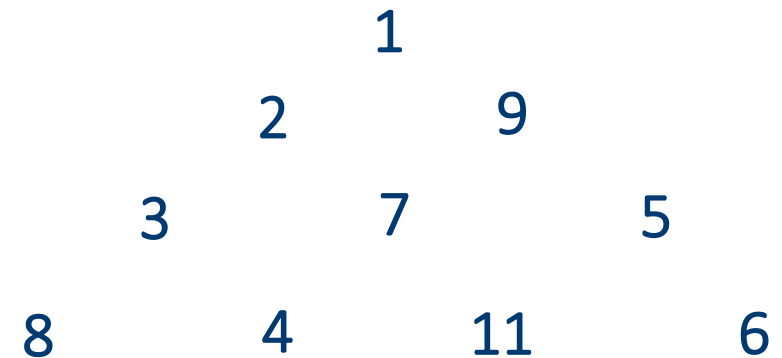
Riflessioni

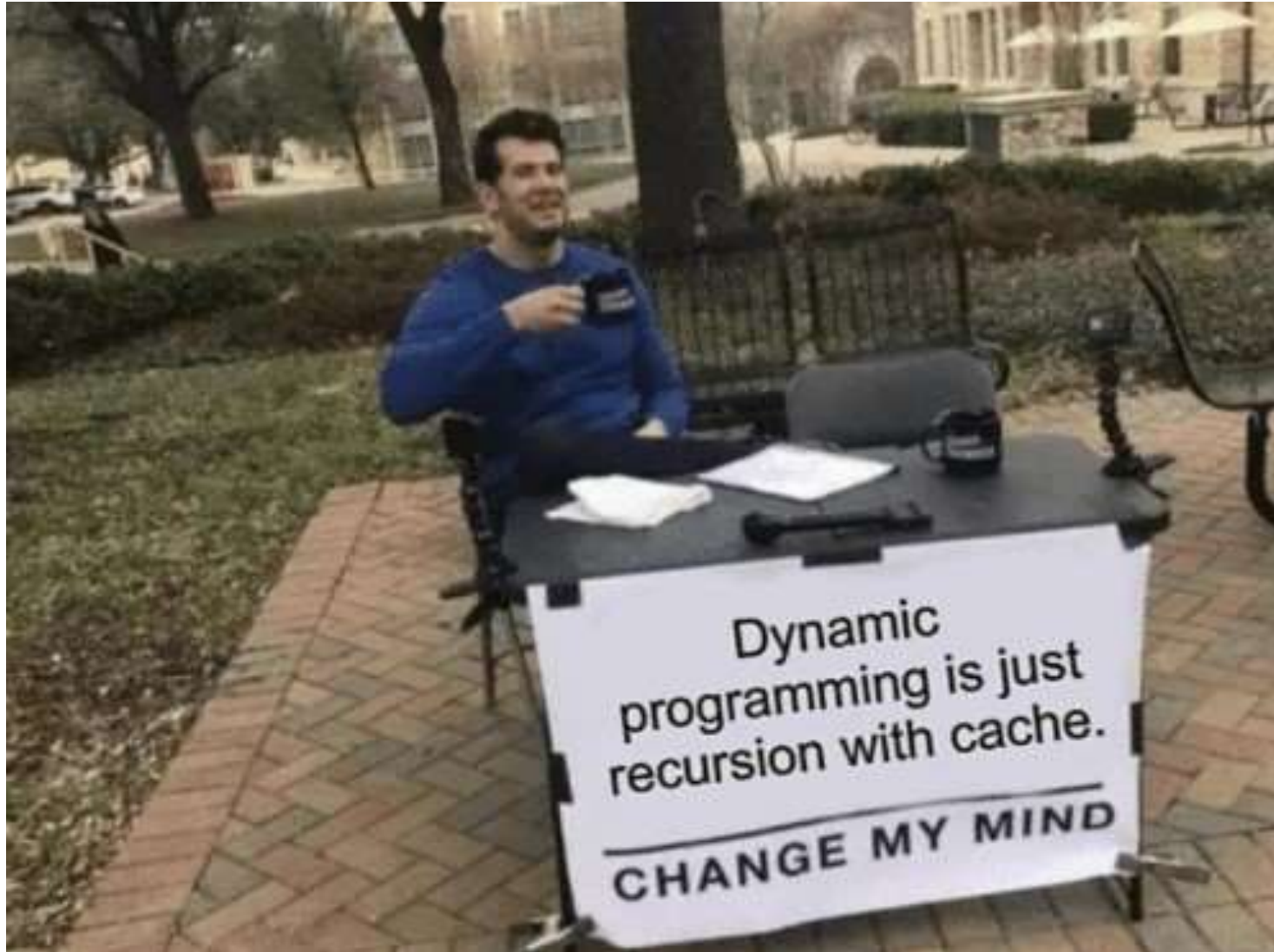
- Approccio naturale: scendo verso il figlio più grande (greedy). Funziona?
- Soluzione ricorsiva "pura": prendo il massimo tra il valore della discesa massima a sinistra e il valore della discesa massima a destra.
 - È corretto?
 - È veloce?
 - Si può migliorare?



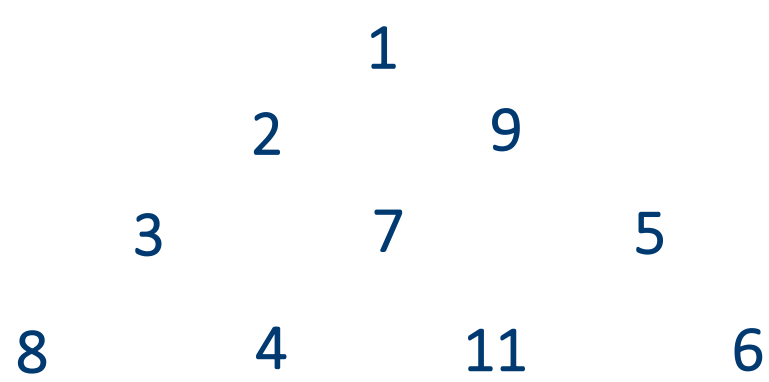
Riflessioni

- Approccio naturale: scendo verso il figlio più grande (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?
- Programmazione dinamica!



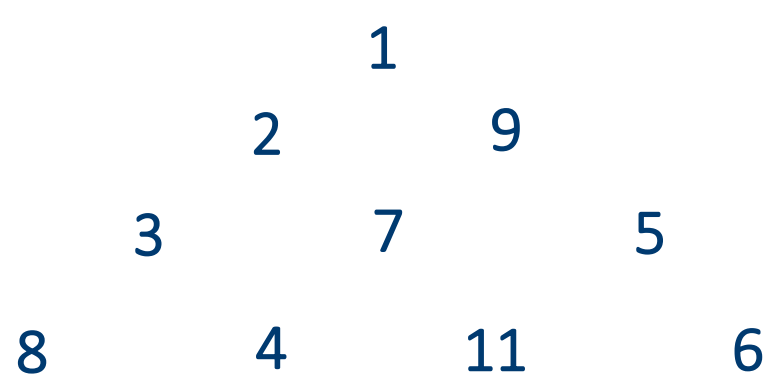


Riflessioni



1			
2	9		
3	7	5	
8	4	11	6

Riflessioni



1			
2	9		
3	7	5	
8	4	11	6



Un problema dalle Territoriali (2014): Sommelier

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



Corso per Sommelier (sommelier)

DIFFICOLTÀ D=2

Descrizione del problema

Paolo, per festeggiare il suo quarantesimo compleanno, si è iscritto a un corso per sommelier, dove impara a distinguere ed apprezzare le diverse tipologie di vini. Si è accorto però che, nonostante prenda solo un assaggio di ogni tipo di vino, per lui vale la regola fondamentale delle bevande alcoliche: quando le bevi, mai scendere di gradazione. Infatti, se per esempio Paolo assaggia un vino da 9 gradi e poi uno da 7, il giorno dopo si sveglierà con un grosso mal di testa indipendentemente dalle quantità. Per fortuna, in ogni serata del corso è disponibile l'elenco dei vini che verranno portati uno dopo l'altro, e di ogni vino viene riportata la gradazione alcolica. Non è ammesso mettere da parte un vino per berlo in seguito: ogni volta che gli viene passato un vino Paolo può decidere se assaggiarlo o meno, versandone un poco nel suo Tastevin. Inoltre, dal momento che dopo aver assaggiato un vino Paolo deve pulire accuratamente il suo Tastevin con un panno, questa operazione in pratica gli impedisce di assaggiare due vini consecutivi (nell'immagine qui a fianco potete vedere il Tastevin). Paolo desidera assaggiare il maggior numero di vini possibile.



1	2	3	4	5	6	7	8	9
Cilento	Barolo	Lambrusco	Picolit	Verdicchio	Cannonau	Chianti	Pigato	Donzelle
11	13	10	16	12	12	13	11	13

Ad esempio, se in una serata serviranno i vini mostrati nella tabella qui sopra, nell'ordine in cui compaiono nella tabella, il numero massimo di vini che Paolo può riuscire ad assaggiare, rispettando la regola, è quattro: può iniziare, indifferentemente, con il Cilento o con il Lambrusco, e poi assaggiare Verdicchio, Chianti e Donzelle. In questa maniera, la sequenza delle gradazioni alcoliche non scende mai: 11 (oppure 10), 12, 13, 13. Ovviamente, come si vede nell'esempio, è possibile bere due o più vini con la stessa gradazione alcolica.

Dati di input

Il file `input.txt` è composto da 2 righe. La prima riga contiene N , un intero positivo: il numero di vini che saranno serviti nella serata. La seconda riga contiene N interi positivi: le gradazioni alcoliche dei vini che saranno serviti, nell'ordine in cui saranno serviti.

Dati di output

Il file `output.txt` è composto da una sola riga contenente un solo intero positivo: il numero massimo di vini che Paolo può assaggiare nella serata, rispettando la regola di non diminuire la gradazione alcolica nella sequenza e, contemporaneamente, il vincolo di dover pulire il Tastevin, che gli impedisce di assaggiare due vini consecutivi.

Assunzioni

- $2 \leq N \leq 99$.
- I vini hanno una gradazione alcolica compresa tra 1 e 99.

Esempio di input/output

File input.txt	File output.txt
9 11 13 10 16 12 12 13 11 13	4
File input.txt	File output.txt
12 11 13 11 10 11 12 16 12 12 11 10 14	5

Il problema dello zaino (knapsack)

Molte varianti...
(versione non frazionaria)



(Slide rubate alla Prof. Irene Finocchi)

Il problema dello zaino (knapsack)

- Un ladro ha uno zaino che può contenere al più W Kg di peso
- Può rubare n oggetti (senza superare il limite di peso)
- Oggetto i pesa w_i Kg e ha un valore di v_i Euro
- Cosa gli conviene rubare? Massimizzare 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 **valore/peso** (decrescente):
 $\{5,2,1\}$ di valore 35

Rapporti valore/peso: 1, 3, 3.6, 3.7, 4

- *Qual è la soluzione ottima?*



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 $1, \dots, i$ ed assumendo un limite p al massimo peso
- Come passiamo da i ad $(i+1)$ oggetti per un certo peso residuo p ?

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



Come passiamo da i ad $(i+1)$ oggetti per un certo peso residuo 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 minimo 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 $M[n, W].$

Un problema dalle OIS (2020): Fibonacci colonies

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



Fibonacci Colonies (fibonaccibug)

Bug colonies have been the center of attention of scientists for a long time. Through some technological advancements, we are now able to describe a bug colony using a number known as the *degree* of the colony. A colony of degree 0 or 1 represents a colony with one bug. A colony of degree $i > 1$ is obtained by merging a colony of degree $i - 1$ together with a colony of degree $i - 2$. As such, a colony of degree 2 has two bugs, a colony of degree 3 has three bugs, a colony of degree 4 has five bugs and so on.



Marco owns the biggest bug farm in the world, having at his disposal a virtually infinite amount of colonies of any degree. Every day he receives N offers, each described by two numbers A_i and B_i , meaning that he can sell as many colonies of degree A_i as he wants and get B_i money for each colony of that degree. Unfortunately, the antitrust laws on the bug trading market forbid him to sell more than K bugs in a single day overall (selling a colony is equivalent to selling all the bugs in that colony). Given the description of T days, if he optimally chooses which offers to accept, what is the maximum amount of money Marco can obtain in each day?

👉 Among the attachments of this task you may find a template file `fibonaccibug.*` with a sample incomplete implementation.

Input

The first line contains one integer T , the number of days. The following lines contain the description of each day. For each day, the first line contains two integers N and K , the number of offers and the maximum number of bugs you can sell that day. The following N lines contain two integers A_i and B_i , the colony of the offer and the price per colony.

Output






You need to write T lines, each with an integer: the maximum profit you can make for each day.

Constraints

- $1 \leq T, N, K \leq 100\,000$.
- $0 \leq A_i \leq 100\,000$.
- $1 \leq B_i \leq 10^9$.
- The sum of all N and all K across the days of a single input does not exceed $201\,000$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- | | |
|---|--|
| – Subtask 1 (0 points) | Examples. |
|  | |
| – Subtask 2 (10 points) | $T = 1, N \leq 6, K \leq 6, A_i \leq 10$. |
|  | |
| – Subtask 3 (10 points) | $K = 1$. |
|  | |
| – Subtask 4 (35 points) | $N, K \leq 5500$ and $A_i \leq 11\,000$. |
|  | |
| – Subtask 5 (45 points) | No additional limitations. |
|  | |

Examples

input	output
1 5 11 1 2 2 2 3 5 4 9 5 50	56
2 3 10 1 10 4 60 3 40 2 10 1 30 2 40	130 300

Explanation

In the **first sample case** it is optimal to choose the fifth offer once and the first one three times.

In the **second sample case**, for the first day it is optimal to choose the first offer once and the third offer three times; for the second day it is optimal to choose ten times the first offer.

Un problema dalle OIS (2017): Rescaling sequence

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



Rescaling Sequence (rescaling)

Giorgio is working on a new research paper, this time about integer sequences. Today, he's looking for a specific kind of sequence: the *rescaling sequence*.

A rescaling sequence is a sequence of integers such that, for each pair of adjacent elements, one of the following statements is true:

- The second element is smaller than the first element.
- The second element is a multiple of the first element.

So, this is a rescaling sequence: 4, 8, 7, 21, 19. This however is *not*: 4, 8, 7, 20, 19 because 20 is not a multiple of 7.

You are given a sequence of N integers. Giorgio can delete some elements from the sequence, but he wants to delete as few as possible of them (possibly zero!). Help Giorgio obtain a rescaling sequence by computing the minimum number of elements to be deleted.

👉 Among the attachments of this task you may find a template file `rescaling.*` with a sample incomplete implementation.

Input

The first line contains the only integer N . The second line contains N integers S_i .

Output

You need to write a single line with an integer: the minimum number of elements to delete.

Constraints

- $1 \leq N \leq 5000$.
- $1 \leq S_i \leq 1\,000\,000$ for each $i = 0 \dots N - 1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**5 points**]: Examples.
- **Subtask 2** [**25 points**]: $N \leq 10$.
- **Subtask 3** [**30 points**]: $N \leq 100$.
- **Subtask 4** [**20 points**]: All the elements of the sequence are *prime numbers*.
- **Subtask 5** [**20 points**]: No additional limitations.

Examples

input.txt	output.txt
5 4 8 7 21 19	0
5 4 8 7 20 19	2

Explanation

The **first sample case** is the one described above.

In the **second sample case**, it is sufficient to erase either 8, 7 or 20, 19.



Un problema dalle Territoriali (2008): Missioni

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

Missioni segrete (missioni)

Difficoltà $D = 2$.

Descrizione del problema

Il Commissario Basettoni ha presentato a Topolino le missioni che egli dovrà svolgere segretamente nel corso dell'anno. Per ogni missione, oltre al luogo da raggiungere, Basettoni ne indica la durata in giorni e la data massima entro cui deve essere completata. In altri termini, la missione può iniziare in qualunque giorno dell'anno ma deve durare esattamente il numero di giorni indicato e terminare non oltre la data di scadenza.

Topolino, presa la lista delle missioni ricevuta da Basettoni, ordina tali missioni in base alla loro data di scadenza. Quindi, numera i giorni dell'anno da 1 a 365 (non esistono anni bisestili a Topolinia) e trasforma le date di scadenza in numeri secondo tale numerazione. Per esempio, se una missione dura 15 giorni e deve essere svolta entro il 18 febbraio, Topolino la vede semplicemente come una coppia di interi 15 49 (in quanto il 18 febbraio è il quarantanovesimo giorno dell'anno).

Poiché può svolgere una sola missione alla volta, Topolino sa che potrebbe svolgerne solo alcune pur iniziando una missione il giorno immediatamente successivo a quello in cui termina la precedente missione. Vuole perciò sapere il numero massimo di missioni che è in grado di svolgere rispettando i vincoli sulla loro durata e scadenza. Supponendo che Topolino già fornisca le coppie di interi ordinate per scadenza (il secondo membro delle coppie), aiutatelo a calcolare il massimo numero di missioni che può svolgere.

Per esempio, se ci sono quattro missioni, una di tre giorni da terminare entro il 5 gennaio, una di quattro giorni entro l'8 gennaio, una di tre giorni entro il 9 gennaio e una di 6 giorni entro il 12 gennaio, Topolino vi fornisce la lista di quattro coppie 3 5, 4 8, 3 9 e 6 12. Il numero massimo di missioni che può svolgere è pari a tre, ossia le missioni corrispondenti alle coppie 3 5, 3 9 e 6 12: la prima missione inizia il primo di gennaio e termina il 3 gennaio; la seconda inizia il 4 gennaio e termina il 6 gennaio; la terza inizia il 7 gennaio e termina il 12 gennaio. (Notare che, scegliendo la missione corrispondente alla coppia 4 8, Topolino può svolgere al più due missioni.)

Dati di input

Il file `input.txt` è composto da $N+1$ righe. La prima riga contiene un intero positivo che rappresenta il numero N di missioni presentate da Basettoni a Topolino.

Le successive N righe rappresentano durata e scadenza delle missioni: ciascuna riga è composta da due interi d e s separati da uno spazio, a rappresentare che la corrispondente missione dura d giorni e deve essere completata entro l' s -esimo giorno dell'anno.

Dati di output

Il file `output.txt` è composto da una sola riga contenente un intero che rappresenta il massimo numero di missioni che Topolino può svolgere rispettando i vincoli su durata e scadenza.

Assunzioni

- $1 \leq N \leq 100$.
- $1 \leq d, s \leq 365$.

Esempi di input/output

File input.txt	File output.txt
4 3 5 4 8 3 9 6 12	3

Programmazione dinamica in poche parole...





**QUANDO FINALMENTE CAPISCI
LA PROGRAMMAZIONE DINAMICA**



Programmazione



Programmazione
dinamica

Compiti a casa (Esercizi Olimpiadi)



Compiti a casa

Prova a risolvere i seguenti problemi:

- Sushi variegato (pre OII 2017) https://training.olinfo.it/#/task/preoii_yutaka/statement
- Esame di maturità (ABC 2016) https://training.olinfo.it/#/task/abc_esame/statement
- Numeri di Figonacci (OIS 2014) <https://training.olinfo.it/#/task/figonacci/statement>
- Petali di margherita (Nazionali 2007) <https://training.olinfo.it/#/task/petali/statement>
- Board game (OIS 2020) https://training.olinfo.it/#/task/ois_marcel/statement
- Canottaggio (Gator 2015) <https://training.olinfo.it/#/task/canoa/statement>

