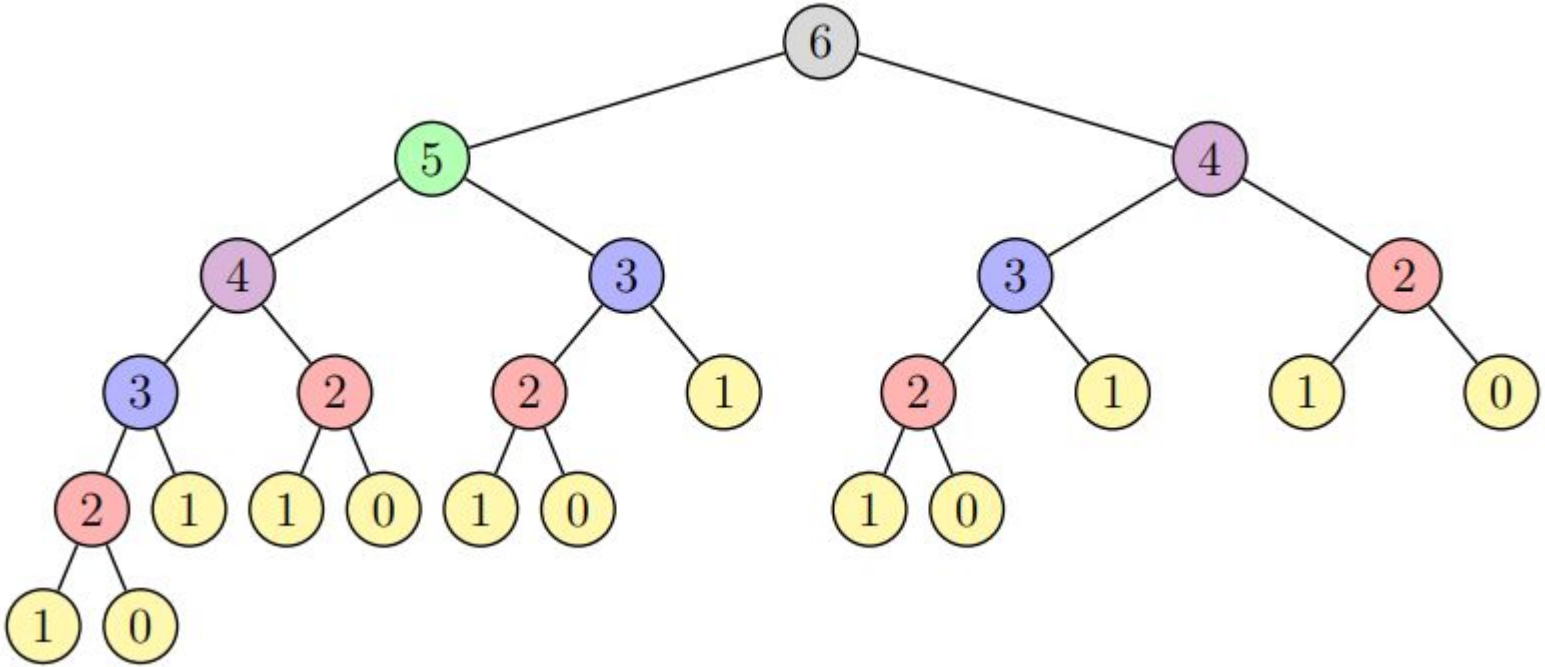
Palestra di algoritmi

05/12/2023

Fibonacci

```
int fibonacci(int n) {
        if(n==0) return 0;
        if(n==1) return 1;
        return fibonacci(n-1) + fibonacci(n-2);
}
```

Fibonacci(6)



Molti problemi ripetuti!!!

Possiamo fare di meglio?

Evitare di risolvere un sotto-problema più volte

- Memorizziamo il risultato di ogni sotto-problema in una tabella (in questo caso un vettore, ma potrebbe essere anche una matrice)

Metodo bottom-up:

- Iniziamo inserendo nella tabella i casi base
- Partendo dai casi base risolviamo i problemi più complessi, riempiendo la tabella

Questa tecnica è detta programmazione dinamica

Programmazione dinamica applicata a Fibonacci

Usiamo un vettore vet.

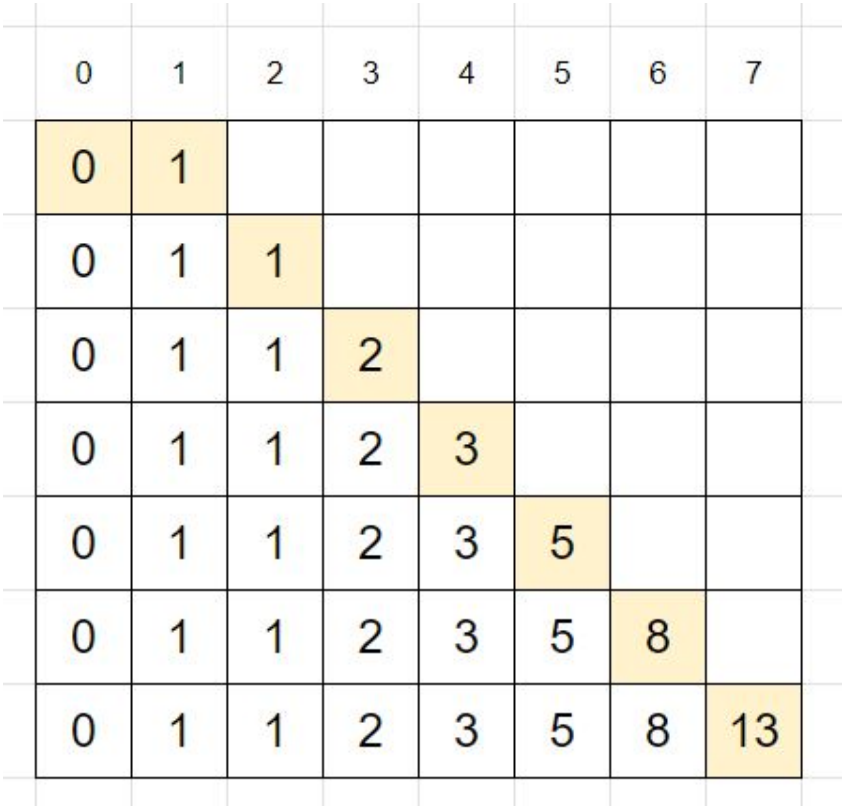
Casi base:

- vet[0] = 0
- vet[1] = 1

Possiamo via via riempire così la tabella: vet[n] = vet[n-1] + vet[n-2]

In codice

```
int fibonacci(int n) {
        int vet[MAX];
        vet[0] = 0;
        vet[1] = 1;
        for(int i=2; i<n; i++)
                vet[i] = vet[i-1] + vet[i-2];
        return vet[n]
}
```



0	1	2	3	4	5	6	7
0	1						
0	1	1					
0	1	1	2				
0	1	1	2	3			
0	1	1	2	3	5		
0	1	1	2	3	5	8	
0	1	1	2	3	5	8	13

Poldo

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.

Poldo si trova al ristorante e ha il menù, composto da una serie di panini (con il loro peso), che verranno serviti in quell'ordine esatto. Poldo, per non violare la regola della sua dieta, 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, di capire qual è il numero massimo di panini che può mangiare senza violare la regola della sua dieta.

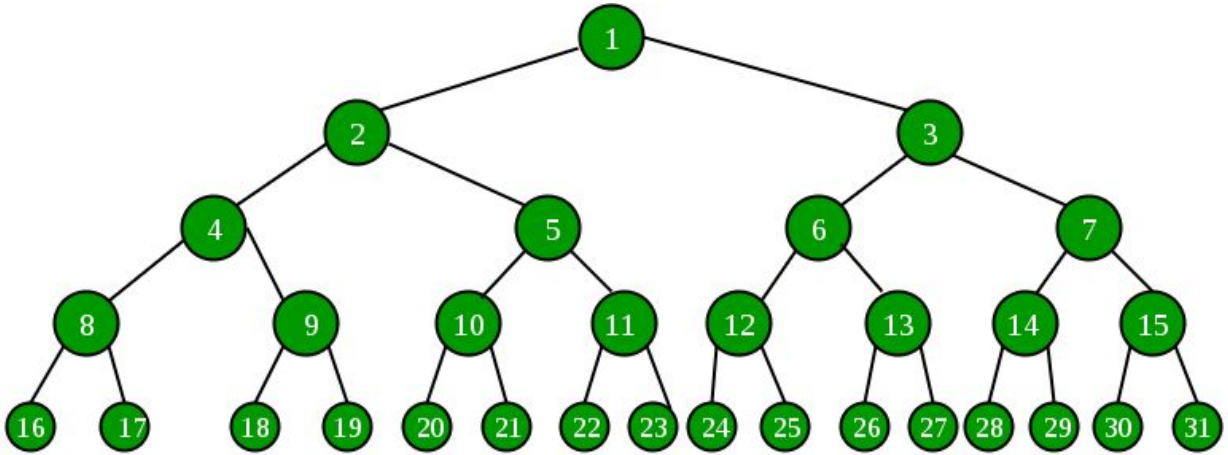
Esempi

Input:	Output:	
5	3	Poldo può mangiare i panini 6, 5, 3
3 6 7 5 3		

Input:	Output:	
8	4	Poldo può mangiare i panini 9, 8, 5, 4
0 9 8 5 1 8 4 7		

Soluzione ricorsiva???

A ogni panino possiamo scegliere se prenderlo o no e poi risolvere sugli n-1 panini rimasti.



Soluzione greedy???

Mangio il primo panino e da quel momento in poi mangio ogni panino che posso.

Cosa succede se ho la seguente sequenza?

100 200 150

Se mangio il primo, non posso mangiare gli altri due, mentre se partissi dal secondo potrei mangiare anche il terzo.

Soluzione in programmazione dinamica

Mi tengo un vettore dove segno la soluzione ottimale a seconda del numero di panini.

vet[1] conterrà la soluzione migliore se avessi un solo panino sul menù

vet[2] conterrà la soluzione migliore se avessi due panini sul menù

…

vet[n] conterrà la soluzione migliore totale (con tutti i panini)

Caso base: 1 solo panino -> soluzione ottimale è di prenderlo, quindi 1

	0	1	2	3	4	5	6	7	8
Pesi	inf	389	207	155	300	299	112	158	65
Soluzioni		1							

Panino numero 2:

- prendo solo il panino 2 -> soluzione = 1
- posso aumentare la sequenza dal primo panino? SI -> soluzione = 2

Soluzione migliore = 2.

	0	1	2	3	4	5	6	7	8
Pesi	inf	389	207	155	300	299	112	158	65
Soluzioni		1							
		1	2						

Panino numero 3:

- se prendo solo il panino numero 3 -> soluzione = 1
- posso aumentare la sequenza dal primo panino? SI -> soluzione = 2
- posso aumentare la sequenza dal secondo panino? SI -> soluzione = 3

Soluzione migliore = 3

	0	1	2	3	4	5	6	7	8
Pesi	inf	389	207	155	300	299	112	158	65
Soluzioni		1							
		1	2						
		1	2	3					

Panino numero 4:

- se prendo solo il 4 -> soluzione = 1
- posso aumentare dal primo panino? SI -> soluzione = 2
- posso aumentare dal secondo panino? NO -> soluzione = 1
- posso aumentare dal terzo panino? NO -> soluzione = 1

Soluzione migliore = 2

	0	1	2	3	4	5	6	7	8
Pesi	inf	389	207	155	300	299	112	158	65
Soluzioni		1							
		1	2						
		1	2	3					
		1	2	3	2				

Così via…

La soluzione finale è 5.

Alcune possibili sequenze:

389 207 155 112 65

389 300 299 158 65

	0	1	2	3	4	5	6	7	8
Pesi	inf	389	207	155	300	299	112	158	65
Soluzioni		1							
		1	2						
		1	2	3					
		1	2	3	2				
		1	2	3	2	3			
		1	2	3	2	3	4		
		1	2	3	2	3	4	4	
		1	2	3	2	3	4	4	5

In codice

```
soluz[1] = 1;        //caso base

for(int i=2;i<=n;i++) {            //per ogni panino che aggiungo
        int migliore = 1;
        for(int j=1; j<i; j++) {            //controllo le sequenze massime precedenti
                if(pesi[i]<pesi[j]) {        //se posso aumentare la sequenza
                        if((soluz[j]+1)>migliore)        //e la soluzione migliora
                                migliore = soluz[j]+1;
                        }
                }
                soluz[i] = migliore;
}

//trovo la soluzione ottima totale
int soluzione_ottima = 1;

for(int i=1;i<=n;i++) {
        if(soluz[i]>soluzione_ottima)
                soluzione_ottima = soluz[i];
}
out << soluzione_ottima;
```

Sommelier(https://training.olinfo.it/#/task/sommelier/statement)

Paolo si è iscritto a un corso per sommelier, ma si è accorto di sentirsi male se beve un vino di gradazione alcolica minore rispetto al precedente.

In ogni serata del corso è disponibile l'elenco dei vini che verranno portati uno dopo l'altro, con la sua 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, dopo aver assaggiato un vino Paolo deve pulire accuratamente il suo Tastevin con un panno, quindi non può assaggiare due vini di fila.

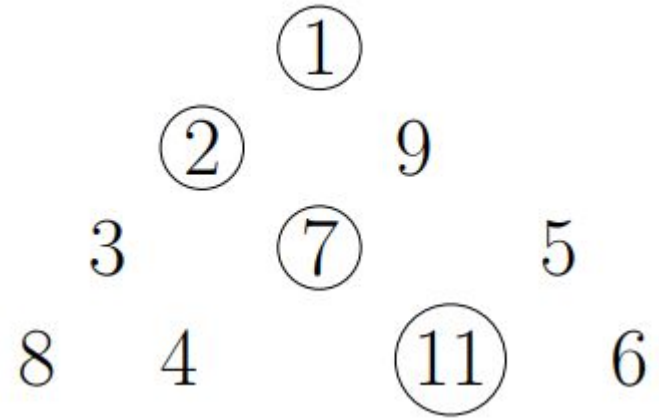
Calcola il numero massimo di vini che può assaggiare Paolo.

Soluzione sommelier

Idea: come Poldo, ma non si può considerare il vino immediatamente precedente.

```
int ric(int n) {
	if(dp[n]>0) { //se è già stato calcolato
		return dp[n];
	}
	//altrimenti lo calcoliamo
}
```

Discesa massima



Discesa = sequenza di numeri ottenuti partendo dalla cima del triangolo e passando per uno dei due numeri sottostanti, fino a giungere alla base

Valore di una discesa = somma dei numeri della discesa.

Discesa massima = quella che ha il massimo valore tra tutte le discese del triangolo

In questo caso il valore della discesa massima è 1+9+7+11 = 28

Soluzione discesa

Parto dal vertice. Sicuramente la discesa massima con solo il vertice è il vertice stesso.

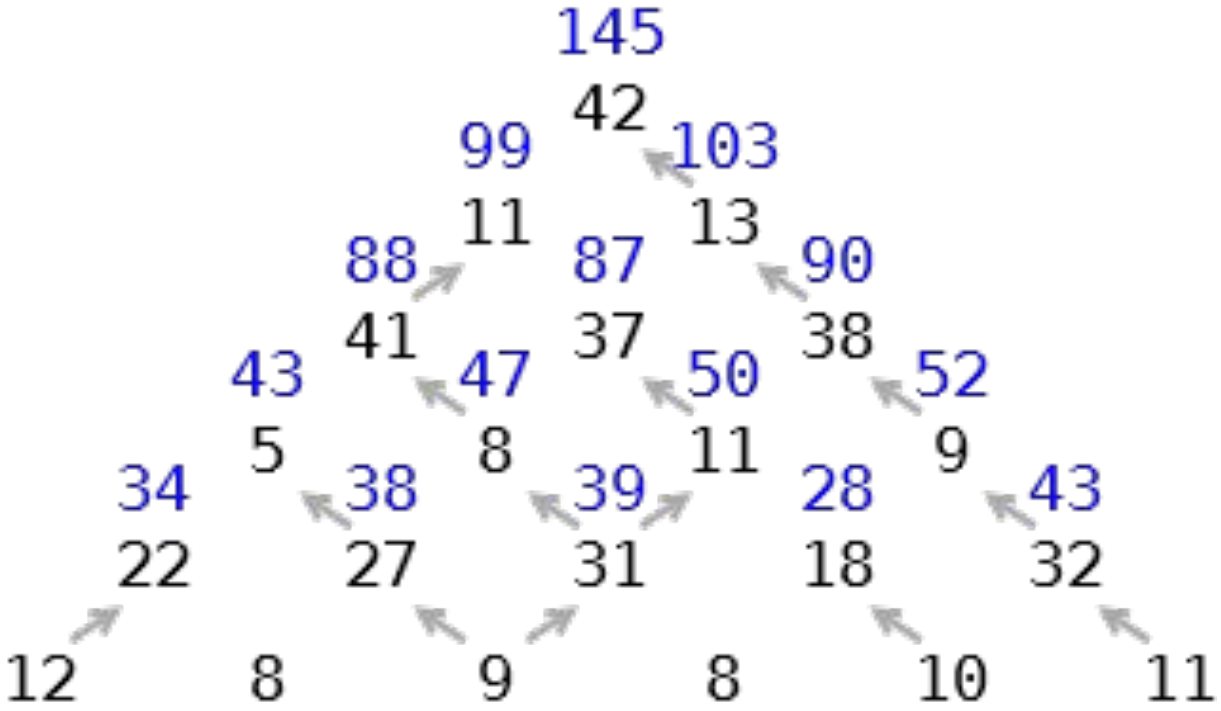
Passo al livello appena sotto. La discesa massima è data dal numero + vertice.

Ai livelli successivi, la soluzione migliore è data dal numero + la soluzione migliore che posso ottenere al livello precedente.

1			
2	9		
3	7	5	
8	4	11	6

1			
3	10		
13	17	15	
25	21	28	21

Potremmo anche partire dal fondo



```
int ric(int i, int j) {
      if (j > i || i > n - 1)
            return 0;
      int ric1 = ric(i + 1, j);
      int ric2 = ric(i + 1, j + 1);
    if (ric1 > ric2)
       return ric1 + mat[i][j];
    else
       return ric2 + mat[i][j];
}
```

Zaino

Un ladro entra in una casa, con uno zaino che può sopportare fino a un peso C.

Nella casa ci sono diversi oggetti, ognuno con un peso e un valore.

Si vuole trovare il valore massimo che il ladro può rubare, senza superare la capacità dello zaino.

Input: vettore dei pesi p[], vettore dei valori v[], capacità dello zaino C

Tabella DP

Creiamo una tabella DP.

DP[i][c] = massimo profitto che può essere ottenuto dai primi i oggetti, usando uno zaino di capacità c.

La nostra soluzione si troverà in DP[n][C]

Parte ricorsiva

Cosa succede se non prendo un oggetto?

DP[i][c] = DP[i − 1][c] //la soluzione è uguale a quella con l'oggetto precedente

E se lo prendo?

DP[i][c] = DP[i−1][c−w[i]] +p[i] //aumento il profitto e diminuisco la capacità

Soluzione migliore = massimo tra prendere e non prendere l'oggetto

Casi base

Non ho più oggetti

DP[0][c] = 0

Non ho più capacità

DP[i][0] = 0

Capacità negativa

DP[i][c<0] = -infinito

Formula completa

$$DP[i][c] = \begin{cases} 0 & i = 0 \textbf{ or } c = 0 \\ -\infty & c < 0 \\ \max(DP[i-1][c - w[i]] + p[i], DP[i-1][c]) & \text{otherwise} \end{cases}$$

Algoritmo

int knapsack(**int**[] w, **int**[] p, **int** n, **int** C)

$DP = \textbf{new int}[0 \ldots n][0 \ldots C]$
for $i = 0$ **to** n **do**
  $DP[i][0] = 0$
for $c = 0$ **to** C **do**
  $DP[0][c] = 0$
for $i = 1$ **to** n **do**
  for $c = 1$ **to** C **do**
    if $w[i] \leq c$ **then**
      $DP[i][c] = \max(\overbrace{DP[i-1][c-w[i]] + p[i]}^{\text{Preso}}, \overbrace{DP[i-1][c]}^{\text{Non preso}})$
    else
      $DP[i][c] = \overbrace{DP[i-1][c]}^{\text{Non preso}}$
return $DP[n][C]$

Esempio

```
w = [ 4, 2, 3, 4]
p = [10, 7, 8, 6]
C = 9
```

	c									
i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	10	10	10	10	10	10
2	0	0	7	7	10	10	17	17	17	17
3	0	0	7	8	10	15	17	18	18	25
4	0	0	7	8	10	15	17	18	18	25

Esercizi

- trampolino (https://training.olinfo.it/#/task/ois_trampolino/statement)
- torri (https://training.olinfo.it/#/task/torri/statement)