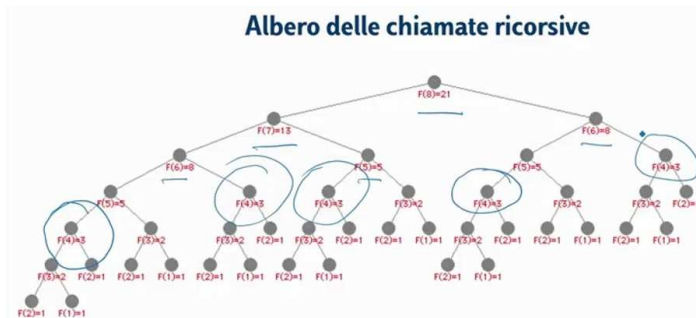


La **Programmazione Dinamica** è utile per problemi che non trovano soluzione con la tecnica Greedy o con le funz. Ricorsive, ma può risultare difficile impostare. Si può utilizzare la Progr. Dinamica (classificazione DP nel Portale Allenamenti) quando la soluzione ottima del problema di dimensione N può essere "ricostruita" a partire da sottoproblemi più piccoli di cui si conosce la soluzione; si utilizza **un vettore o una matrice per salvare i risultati intermedi**.

p.14-16 l'algoritmo **fibonacci1** doppiamente **ricorsivo** non ha buone prestazioni (già per $n > 40$ i tempi di risposta possono raggiungere il minuto!) all'aumentare di n ricalcola più volte i valori precedenti; per calcolare Fibonacci di 8 calcola 5 volte Fibonacci di 3 e 13 volte Fibonacci di 2 - **complessità esponenziale: $O(2^n)$**



p.17-19 I numeri di Fibonacci possono essere determinati con la Programmazione Dinamica: l'algoritmo **fibonacci2** utilizza un array per salvare i risultati precedenti

Fib.1	Fib.2	Fib.3	Fib.4	Fib.5	Fib.6	Fib.7	Fib.8	Fib.9	Fib.10	Fib.11	Fib.12	Fib.13	Fib.14	Fib.15	Fib.16
1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987

In questo caso i tempi di risposta **crescono linearmente al crescere di n** : la **complessità è $O(n)$**

Con questa soluzione anche l'occupazione di memoria (array) cresce linearmente

p.20 L'algoritmo **fibonacci3** utilizza solo 3 variabili: migliora l'occupazione di memoria ma mantiene una **complessità lineare**. (algoritmo di complessità logaritmica **fibonacci6** trovato su <http://www.cs.unibo.it/~donat/FBonacci.pdf>)

le **5 Leggi della Programmazione Dinamica** possono essere realizzate con programmi impostati con uno schema di questo tipo →

Schema del programma in Programmazione Dinamica:

- //--- Leggi input
- //--- Individua il sottoproblema da risolvere
- //--- Condizioni iniziali
- //--- Riempi la tabella di programmazione dinamica
- //--- Trova la soluzione
- //--- Scrivi output

p.38-42 esempio: problema **La dieta di Poldo (poldo)** <https://training.olinfo.it/#/task/poldo/statement>

```
#include <fstream>
using namespace std;
int main() {
    ifstream in ("input.txt");
    ofstream out ("output.txt");

    int nPanini, i, prec;
    //--- Leggi input
    in >> nPanini;
    int peso[nPanini];
    for (i=0; i < nPanini; i++)
        in >> peso[i];

    //--- Individua il sottoproblema da risolvere

    int quantiPanini[nPanini] = {0};

    //--- Condizioni iniziali

    quantiPanini[0] = 1;
```

```
//--- Riempi la tabella di programmazione dinamica
for (i=1; i < nPanini; i++) {
    for (prec = i-1; prec >= 0; prec--)
        if (peso[i] < peso[prec]
            && quantiPanini[i] <
                quantiPanini[prec])
            // considera solo i precedenti con quantiPanini maggiori del valore
            // salvato per i al momento, purchè prec sia di peso maggiore di i
            quantiPanini[i] =
                quantiPanini[prec];
        quantiPanini[i]++; // consdiera panino i-esimo
    }
    //--- Trova la soluzione
    int max = 1;
    for (i=1; i < nPanini; i++)
        if (quantiPanini[i] > max)
            max = quantiPanini[i];

    //--- Scrivi output
    out << max;
    return 0;
}
```

p.44 esempio con nPanini = 10

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

INPUT - array **pesi [nPanini]**



array **quantiPanini [nPanini]**

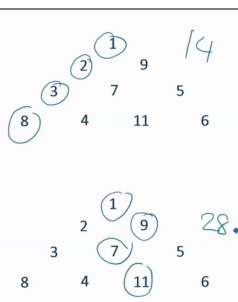
Il ciclo indicato in **//--- Riempi la tabella di programmazione dinamica** imposta progressivamente l'array `quantiPanini[]` determinando il numero di Panini che Poldo può mangiare **compreso il panino di posto i** :

Posizione	0	1	2	3	4	5	6	7	8	9
Valori quando										
i=1	1	1								
i=2	1	1	2							
i=3	1	1	2	3						
i=4	1	1	2	3	2					
i=5	1	1	2	3	2	3				
i=6	1	1	2	3	2	3	4			
i=7	1	1	2	3	2	3	4	5		
i=8	1	1	2	3	2	3	4	5	6	
i=9	1	1	2	3	2	3	4	5	6	6

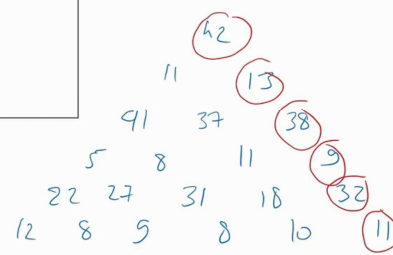
Quanti Panini può mangiare Poldo nell'intervallo `[0,0]` per `i=0?` **1 (solo il panino di peso 8)**
`[0,1]` per `i=1?` **1 (solo il panino 38)**
`[0,2]` per `i=2?` **2 (panini 38 e 20)**
`[0,3]` per `i=3?` **3 (panini 38, 20 e 15)**
`[0,4]` per `i=4?` **2 (panini 38 e 30)** perchè le condizioni sono verificate quando `prec=1` quindi `quantiPanini[i]` viene posto a `quantiPanini[1] = 1` poi per `prec = 0` le condizioni 1 e 2 sono FALSE
`i=5?` **3 (38, 30, 29)** `i=6?` **4 (38, 30, 29, 18)**
`i=7?` **5 (38, 30, 29, 18, 16)**
`i=8?` **6 (38, 30, 29, 18, 16, 9)** `i=9?` **6 (10 invece di 9)**

p.62-73 esercizio: problema **Discesa massima (discesa)** <https://training.olinfo.it/#/task/discesa/statement>

File input.txt	File output.txt
4 1 29 375 84 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
--	-----

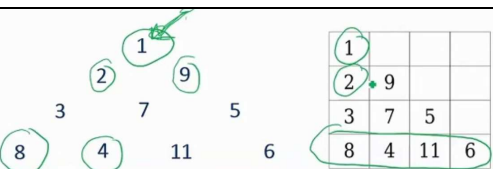


Riflessioni

1. Solto problema?
Discesa massima da sottopiramide

2. Condizioni iniziali?
Foglie

3. Soluzione? cima

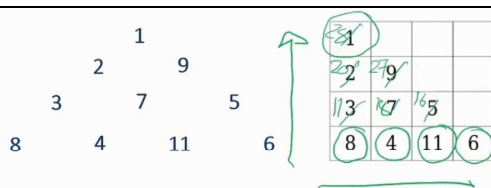


4. Riempire tabella?
 $sol[i,j] = sol[i,j] + \max\{sol[i+1,j], sol[i+1,j+1]\}$

Memorizzata la piramide in una matrice triangolare, gli elementi sottostanti l'elemento (0,0) hanno coordinate (0+1, 0) e (0+1, 0+1) elemento a sin: riga+1 e stessa colonna elemento a dx : riga+1 e colonna+1
 >> considerare come condizioni iniziali le foglie (8, 4, 11 e 6);
 >> posizionare la soluzione in cima alla piramide (0,0)
 >> riempire la tabella in posizione (i,j) considerando le 2 sottopiramidi sottostanti dx e sin e sommando il massimo alla posizione (i,j)

Partendo dal basso
 a 3 sommo il massimo tra 8 e 4 ottenendo 11= 3+8
 >> 7 + massimo tra 4 e 11 → 18= 7+11
 >> 5 + massimo tra 11 e 6 → 16
 >> 2 + massimo tra 11 e 18 → 20
 >> 9 + massimo tra 18 e 16 → 27
 >> 1 + massimo tra 20 e 27 → 28

Riflessioni



⇒ $sol[i,j] = sol[i,j] + \max\{sol[i+1,j], sol[i+1,j+1]\}$

Per il problema *discesa massima* la tecnica GREEDY non funziona e la ricorsione è lenta per piramidi grandi. Meglio la programmazione dinamica (è come una ricorsione con caching)

Soluzione presentata dal professore

```

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 }
    
```

con operatore ternario '?' <http://www.programmiamo.altervista.org/C/If/if4.html>

if equivalente

```

if (piramide[i+1][j] > piramide[i+1][j+1])
    piramide[i][j] += piramide[i+1][j];
else
    piramide[i][j] += piramide[i+1][j+1];
    
```

contenuto di piramide[][] PRIMA di discesa_dinamica					contenuto di piramide[][] DOPO discesa_dinamica						
Caso 1 A = 4											
1					28						
2	9				20	27					
3	7	5			11	18	16				
8	4	11	6		8	4	11	6			
Caso 2 A = 6											
42					145						
11	13				100	103					
41	37	38			89	88	90				
5	8	11	9		41	48	51	52			
22	27	31	18	32	34	36	40	28	43		
12	8	9	8	10	11	12	8	9	8	10	11

p.74 **esercizio: Corso per Sommelier (sommelier)**
 è una variazione di Poldo (non può prendere 2 vini consecutivi) – pag 4 una soluzione

p.78-85 **esempio: Il problema dello zaino (knapsack)**

Greedy funziona?


i	v _i	w _i	v _i /w _i
1	1	1	1
2	6	2	3
3	18	5	3.6
4	22	6	3.7
5	28	7	4

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

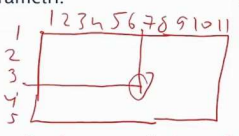
Qual è la soluzione ottima?



pag.80

- 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

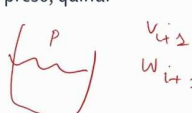


pag.81

Come passiamo da i ad (i+1) oggetti per un certo peso residuo p?

- Se $w_{i+1} > p$, sicuramente l'oggetto (i+1) non può essere preso, quindi $OPT(i+1, p) = OPT(i, p)$
- 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!



pag.82

Esempio

i	v _i	w _i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

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

$1, 2, 3, \dots, i-1$ i

NON PRENDO i $OPT(i-1, w)$

PRENDO i $v_i + OPT(i-1, w-w_i)$

pag.83

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

pag.84

KNAPSACK (n, W, w₁, ..., w_n, v₁, ..., v_n)

```

FOR w = 0 TO W
    M[0, w] ← 0.

FOR i = 1 TO n
    FOR w = 0 TO W
        IF (wi > w) M[i, w] ← M[i-1, w].
        ELSE M[i, w] ← max { M[i-1, w], vi + M[i-1, w - wi] }.

RETURN M[n, W].
    
```

pag.85

p.91 **esercizio: Rescaling Sequence (rescaling)**

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

- 8 è multiplo di 4 OK
- 7 è minore di 8 OK
- 21 è multiplo di 7 OK
- 19 è minore di 21 OK

- eliminare 20 (non è multiplo di 7 ed è maggiore di 7)
- eliminare 19 (non è multiplo di 7 ed è maggiore di 7)

Soluzioni proposte dagli alunni:

Rescaling Sequence (rescaling)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     //freopen("input.txt", "r", stdin);
6     //freopen("output.txt", "w", stdout);
7     int n, maximum=1;
8     cin>>n;
9     vector<int> v(n), sub(n);
10    for(int i=0; i<n; i++)
11        cin>>v[i];
12
13    for(int i=0; i<n; i++){
14        sub[i]=1;
15        for(int j=0; j<i; j++){
16            if(v[j]>v[i] || v[i]%v[j]==0)
17            {
18                sub[i] = max(sub[i], sub[j]+1);
19                if(sub[i]>maximum)
20                    maximum=sub[i];
21            }
22        }
23    }
24    cout<<n-maximum;
25    return 0;
26 }
    
```

Corso per Sommelier (sommelier)

```

1 #include <fstream>
2
3 int main(){
4     freopen("input.txt", "r", stdin);
5     freopen("output.txt", "w", stdout);
6
7     int n, Tmax = 0;
8
9     std::cin >> n;
10
11    std::vector<short> vet (n), soluzione (n, 0);
12
13    for(int i = 0; i < n; i++)
14        std::cin >> vet[i];
15
16    for(int i = 0; i < n; i++){
17        int max = 0;
18        for(int j = 0; j <= i - 2; j++){
19            if(vet[j] <= vet[i] && soluzione[j] > soluzione[i]){
20                soluzione[i] = soluzione[j];
21            }
22        }
23        soluzione[i]++;
24        if(soluzione[i] > Tmax)
25            Tmax = soluzione[i];
26    }
27
28    std::cout << Tmax;
29 }
    
```