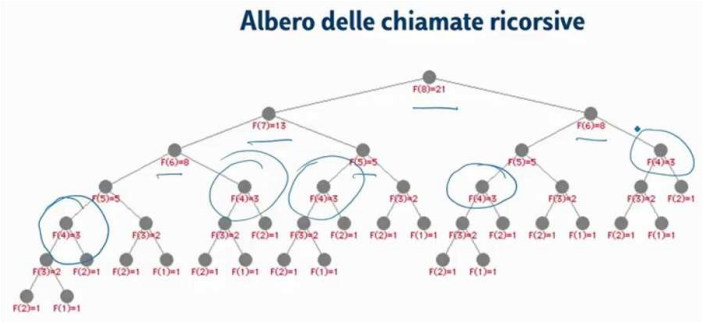
La **Programmazione Dinamica** è utile per problemi che non trovano soluzione con la tecnica Greedy o con le funz. Ricorsive, ma può risultare difficile impostare. Si può utilizzare la Progr. Dinamica (classificazione DP nel Portale Allenamenti) quando la soluzione ottima del problema di dimensione N 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

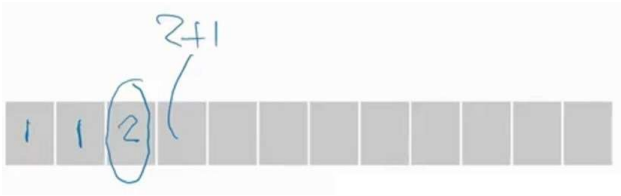
```
#include <fstream>
using namespace std;
int main() {
  ifstream in ("input.txt");
  ofstream out("output.txt");

  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]++;   // condsidera 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]**

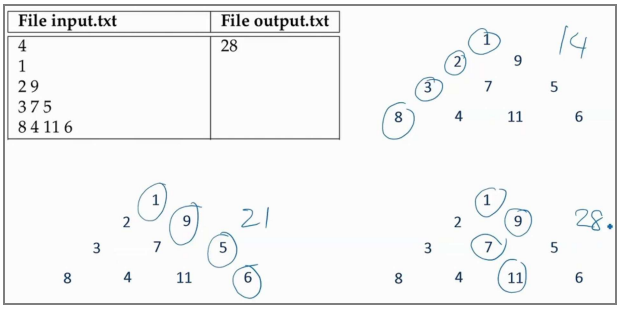
Il ciclo indicato in **//--- Riempi la tabella di programmazione dinamica** imposta progressivamente l'array `quantiPanini[]` determinando il numero di Panini che Poldo può mangiare **compreso il panino di posto i** :

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

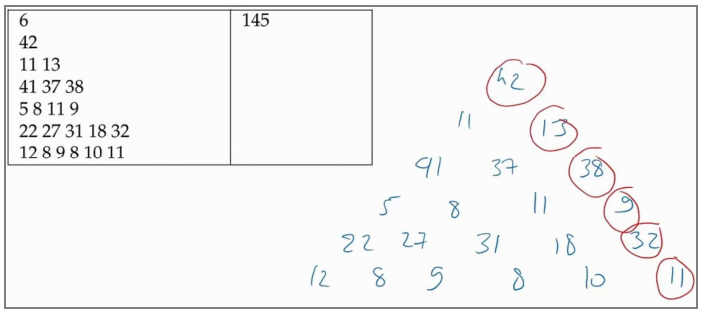
Quanti Panini può mangiare Poldo nell'intervallo
`[0,0]` per `i=0`? 1 (solo il panino di peso **8**)
`[0,1]` per `i=1`? 1 (solo il panino **38**)
`[0,2]` per `i=2`? 2 (panini 38 e **20**)
`[0,3]` per `i=3`? 3 (panini 38, 20 e **15**)
`[0,4]` per `i=4`? 2 (panini 38 e **30**) perchè le condizioni sono 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)

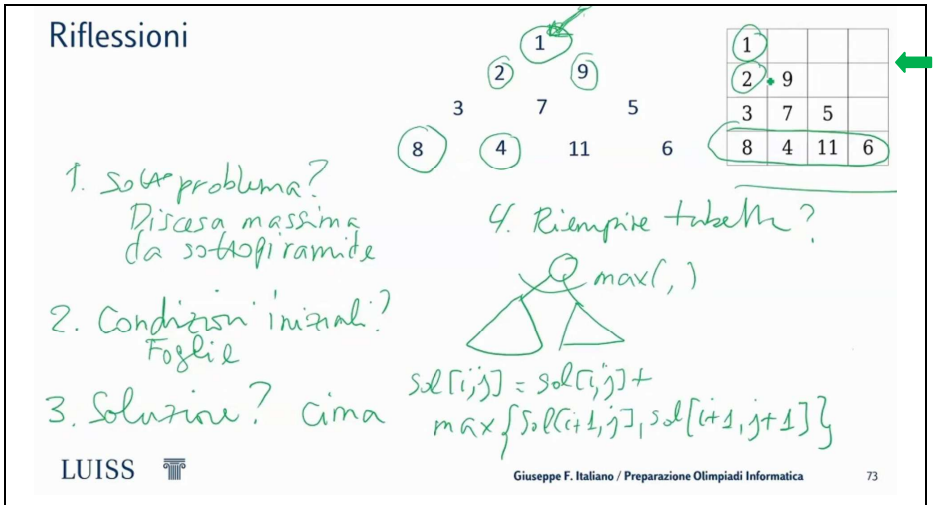
p.62-73 esercizio: problema **Discesa massima (discesa)** https://training.olinfo.it/#/task/discesa/statement

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

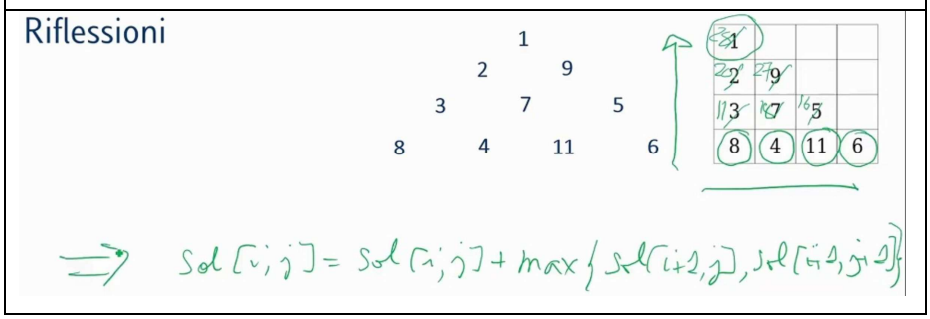


6 42 11 13 41 37 38 5 8 11 9 22 27 31 18 32 12 8 9 8 10 11	145





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

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

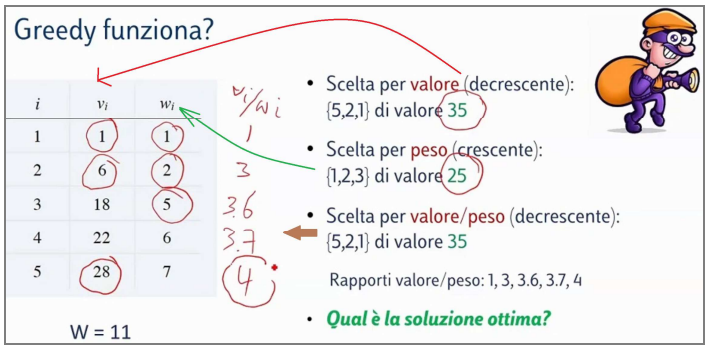
```
if (piramide[i+1][j]
  > piramide[i+1][j+1])

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

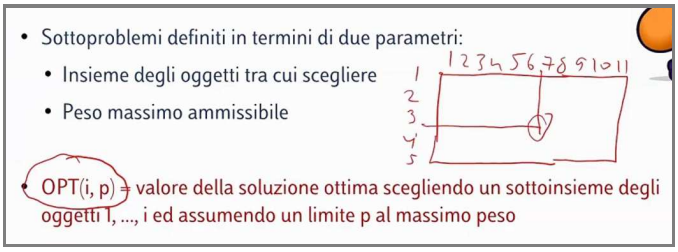
p.74 esercizio: **Corso per Sommelier (sommelier)**

è una variazione di Poldo (non può prendere 2 vini consecutivi) – pag 4 una soluzione

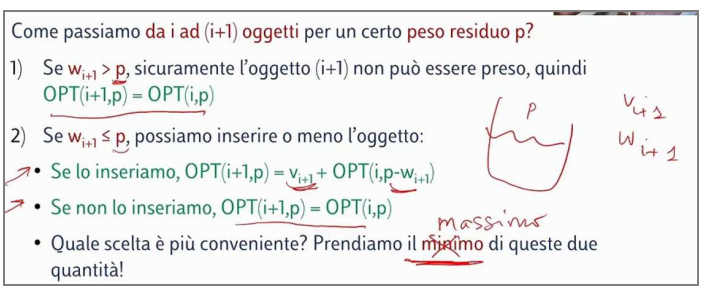
p.78-85 esempio: **Il problema dello zaino (knapsack)**



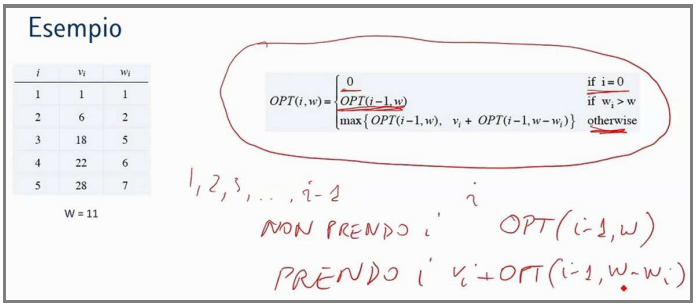
pag.80



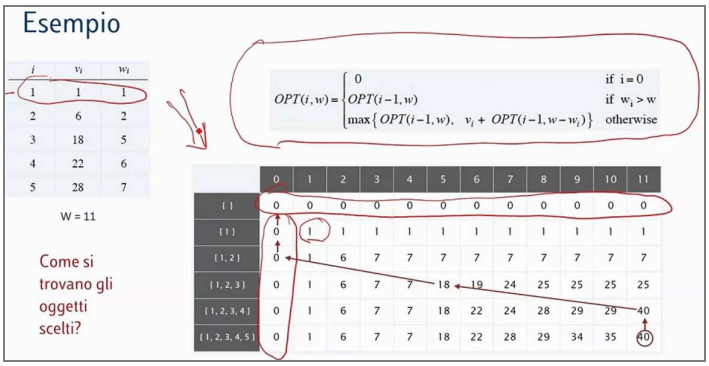
pag.81



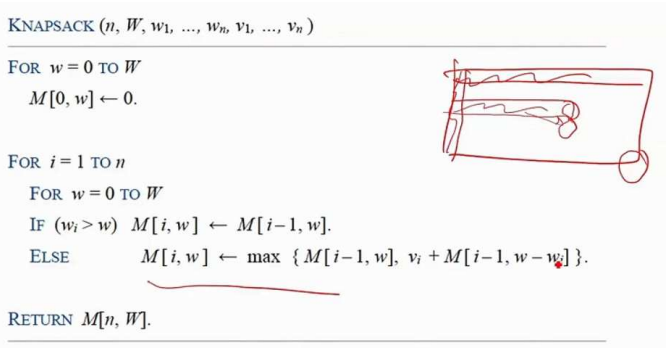
pag.82



pag.83

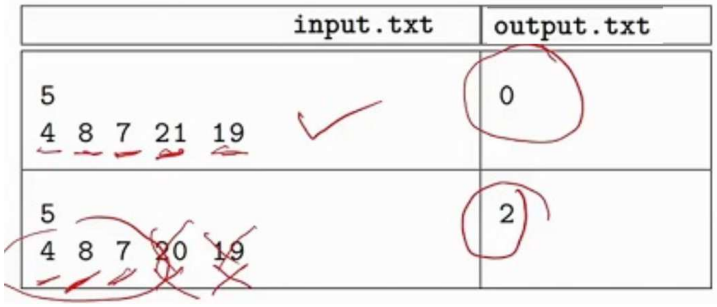


pag.84



pag.85

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



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)

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    //freopen("input.txt","r",stdin);
    //freopen("output.txt","w",stdout);
    int n,maximum=1;
    cin>>n;
    vector<int> v(n),sub(n);
    for(int i=0;i<n;i++)
        cin>>v[i];

    for(int i=0;i<n;i++){
        sub[i]=1;
        for(int j=0;j<i;j++){
            if(v[j]>v[i] || v[i]%v[j]==0)
            {
                sub[i] = max(sub[i],sub[j]+1);
                if(sub[i]>maximum)
                    maximum=sub[i];
            }
        }
    }
    cout<<n-maximum;
    return 0;
}
```

Corso per Sommelier (sommelier)

```cpp
#include <fstream>

int main(){
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);

    int n, Tmax = 0;

    std::cin >> n;

    std::vector <short> vet (n), soluzione (n, 0);

    for(int i = 0; i < n; i++)
        std::cin >> vet[i];

    for(int i = 0; i < n; i++){
        int max = 0;
        for(int j = 0; j <= i - 2; j++){
            if(vet[j] <= vet[i] && soluzione[j] > soluzione[i]){
                soluzione[i] = soluzione[j];
            }
        }
        soluzione[i]++;
        if(soluzione[i] > Tmax)
            Tmax = soluzione[i];
    }

    std::cout << Tmax;
}
```