Lezione 14

Sommario della Lezione

Programmazione Dinamica

Input del problema: Un insieme $A = \{A_1, A_2, \dots, A_n\}$ di attività di valore $v(A_1), \dots, v(A_n)$.

Input del problema: Un insieme $A = \{A_1, A_2, \ldots, A_n\}$ di attività di valore $v(A_1), \ldots, v(A_n)$. Ogni attività A_i può essere svolta nell'intervallo temporale $[s_i, f_i]$, con $s_i < f_i$.

Input del problema: Un insieme $A = \{A_1, A_2, \ldots, A_n\}$ di attività di valore $v(A_1), \ldots, v(A_n)$. Ogni attività A_i può essere svolta nell'intervallo temporale $[s_i, f_i]$, con $s_i < f_i$.

Le attività in A devono essere eseguite da uno stesso **server**, sotto la condizione che A_i ed A_j possono essere entrambe eseguite se e solo se $[s_i, f_i] \cap [s_j, f_j] = \emptyset$

Input del problema: Un insieme $A = \{A_1, A_2, \dots, A_n\}$ di attività di valore $v(A_1), \dots, v(A_n)$. Ogni attività A_i può essere svolta nell'intervallo temporale $[s_i, f_i]$, con $s_i < f_i$.

Le attività in A devono essere eseguite da uno stesso **server**, sotto la condizione che A_i ed A_j possono essere entrambe eseguite se e solo se $[s_i, f_i] \cap [s_j, f_j] = \emptyset$ (in tal caso, diremo che l'attivitá A_i ed A_j sono *compatibili*).

Input del problema: Un insieme $A = \{A_1, A_2, \ldots, A_n\}$ di attività di valore $v(A_1), \ldots, v(A_n)$. Ogni attività A_i può essere svolta nell'intervallo temporale $[s_i, f_i]$, con $s_i < f_i$.

Le attività in A devono essere eseguite da uno stesso **server**, sotto la condizione che A_i ed A_j possono essere entrambe eseguite se e solo se $[s_i, f_i] \cap [s_j, f_j] = \emptyset$ (in tal caso, diremo che l'attivitá A_i ed A_j sono *compatibili*).

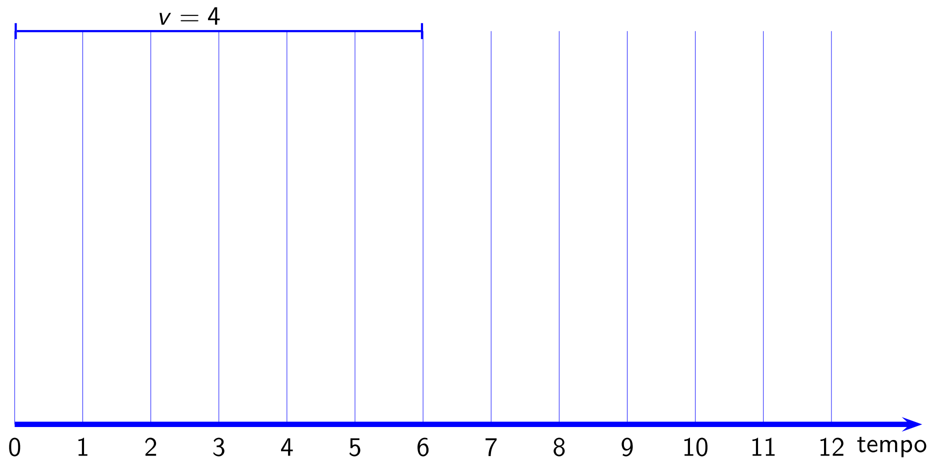
Output del problema: Un sottoinsieme di $S \subseteq A$ di attività a due a due compatibili, di valore totale $\sum_{A \in S} v(A)$ massimo.

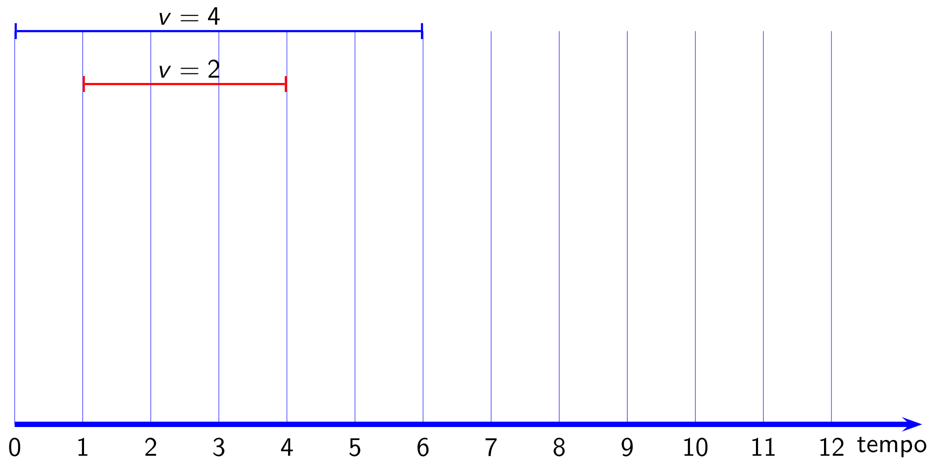
Input del problema: Un insieme $A = \{A_1, A_2, \dots, A_n\}$ di attività di valore $v(A_1), \dots, v(A_n)$. Ogni attività A_i può essere svolta nell'intervallo temporale $[s_i, f_i]$, con $s_i < f_i$.

Le attività in A devono essere eseguite da uno stesso **server**, sotto la condizione che A_i ed A_j possono essere entrambe eseguite se e solo se $[s_i, f_i] \cap [s_j, f_j] = \emptyset$ (in tal caso, diremo che l'attivitá A_i ed A_j sono *compatibili*).

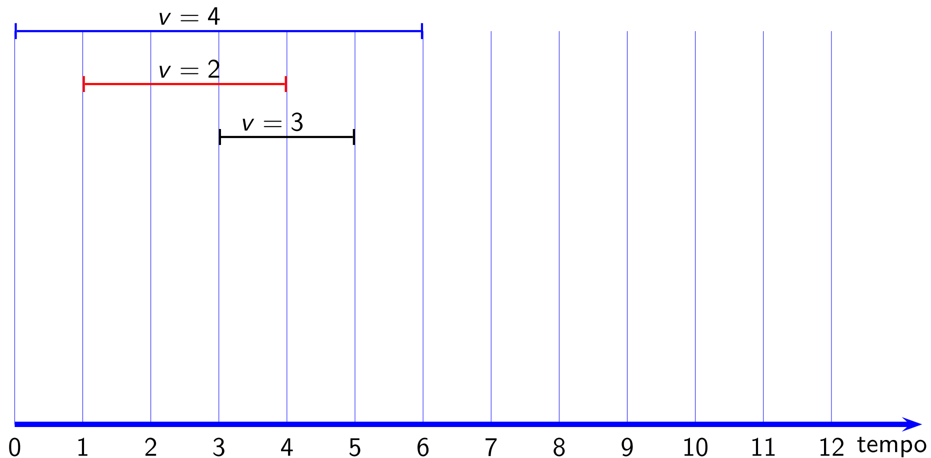
Output del problema: Un sottoinsieme di $S \subseteq A$ di attività a due a due compatibili, di valore totale $\sum_{A \in S} v(A)$ massimo. Ovvero, vogliamo calcolare

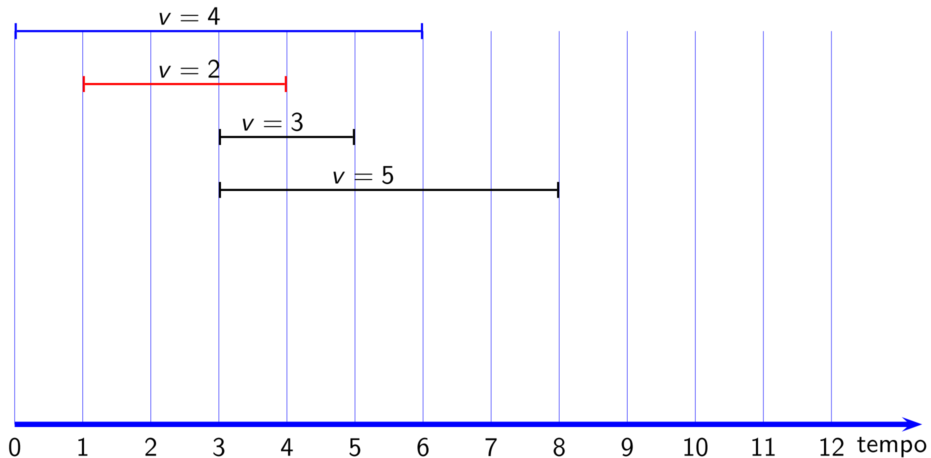
$$\max_{\substack{S \subseteq A : S \text{ è composto da attività} \\ \text{mutualmente compatibili}}} \sum_{A \in S} v(A).$$


$v = 4$
0
1
2
3
4
5
6
7
8
9
10
11
12
tempo

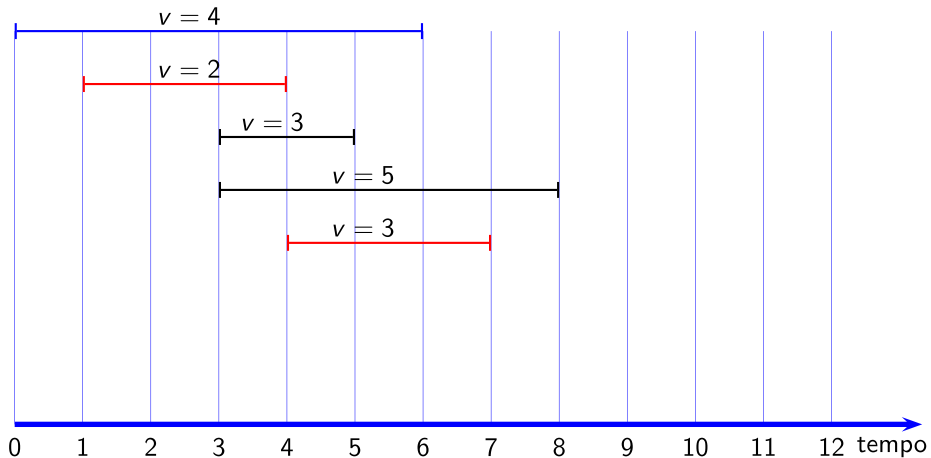


$v = 4$
$v = 2$
0
1
2
3
4
5
6
7
8
9
10
11
12
tempo

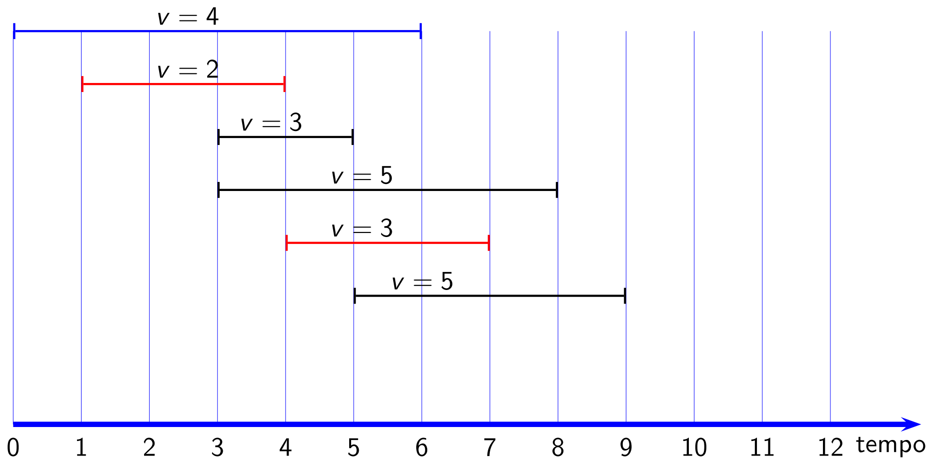

$v = 4$
$v = 2$
$v = 3$
0
1
2
3
4
5
6
7
8
9
10
11
12
tempo

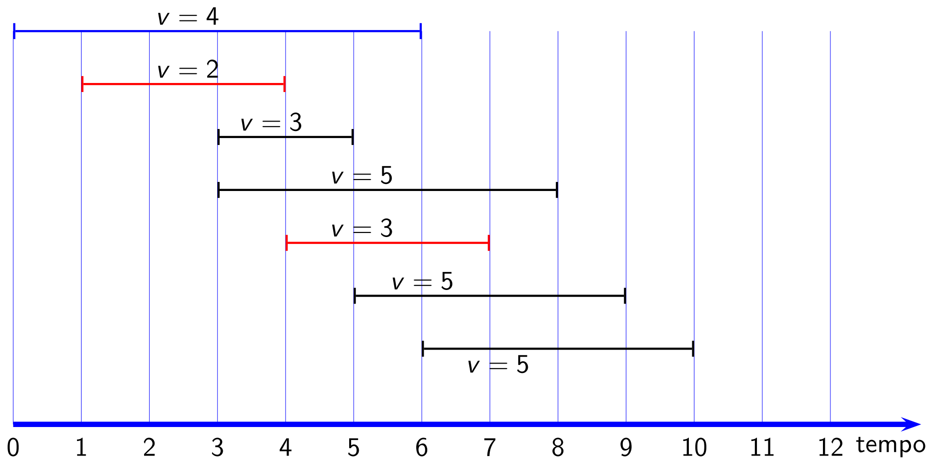


$v = 4$
$v = 2$
$v = 3$
$v = 5$
0
1
2
3
4
5
6
7
8
9
10
11
12
tempo

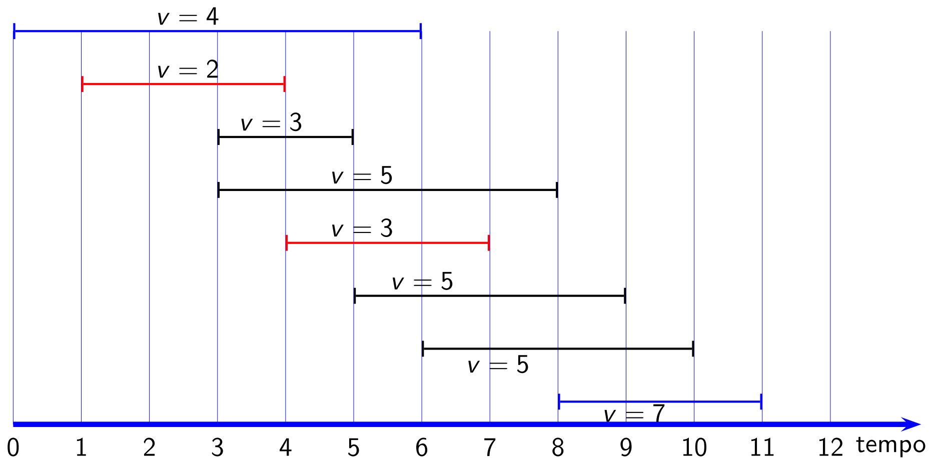


v = 4
v = 2
v = 3
v = 5
v = 3
0
1
2
3
4
5
6
7
8
9
10
11
12
tempo


$v = 4$
$v = 2$
$v = 3$
$v = 5$
$v = 3$
$v = 5$
0
1
2
3
4
5
6
7
8
9
10
11
12
tempo



$v = 4$
$v = 2$
$v = 3$
$v = 5$
$v = 3$
$v = 5$
$v = 5$
0
1
2
3
4
5
6
7
8
9
10
11
12
tempo



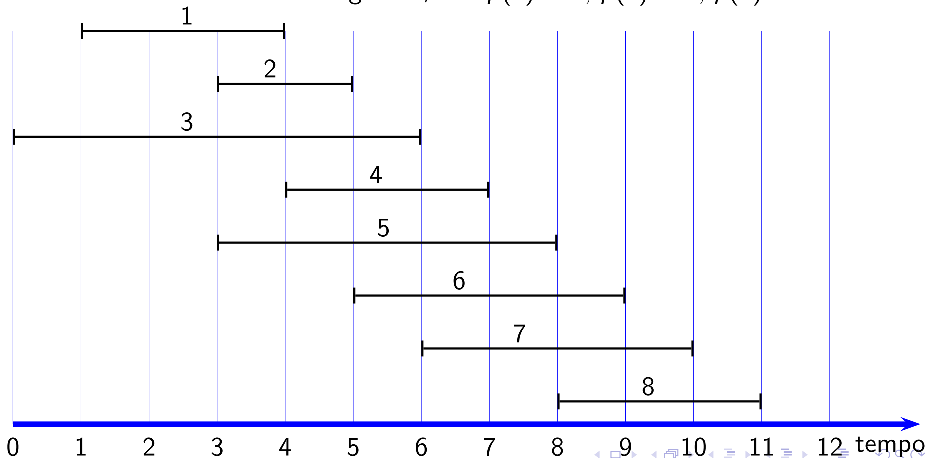
v = 4
v = 2
v = 3
v = 5
v = 3
v = 5
v = 5
v = 7
0
1
2
3
4
5
6
7
8
9
10
11
12
tempo

Ordiniamo le attivita (e rinominiamole) in base ai numeri f_i per cui varrà : $f_1 \leq f_2 \leq \ldots \leq f_n$.

Ordiniamo le attivita (e rinominiamole) in base ai numeri f_i per cui varrà : $f_1 \leq f_2 \leq \ldots \leq f_n$. Per ogni attività j, sia $p(j) =$ il più grande indice $i < j$ tale che attività i è compatibile con l'attività j

Ordiniamo le attivita (e rinominiamole) in base ai numeri f_i per cui varrà : $f_1 \leq f_2 \leq \ldots \leq f_n$. Per ogni attività j, sia $p(j) =$ il più grande indice $i < j$ tale che attività i è compatibile con l'attività j ($p(j) = 0$ se tale indice non esiste).

Ordiniamo le attivita (e rinominiamole) in base ai numeri f_i per cui varrà : $f_1 \leq f_2 \leq \ldots \leq f_n$. Per ogni attività j, sia $p(j) =$ il più grande indice $i < j$ tale che attività i è compatibile con l'attività j ($p(j) = 0$ se tale indice non esiste). Nell'esempio precedente, avremmo che dopo aver ordinato la situazione è la eguente, con $p(8) = 5$, $p(7) = 3$, $p(2) = 0$.



Sia $\mathcal{O}$ un sottoinsieme delle attività $A = \{A_1, \ldots, A_n\}$ composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile (ottima).

Sia $\mathcal{O}$ un sottoinsieme delle attività $A = \{A_1, \ldots, A_n\}$ composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile (ottima).

- Se $n \in \mathcal{O}$ allora **tutte** le attività $p(n)+1, p(n)+2, \ldots, n-1$ intersecano n, quindi esse **non** possono essere in $\mathcal{O}$.

Sia $\mathcal{O}$ un sottoinsieme delle attività $A = \{A_1, \ldots, A_n\}$ composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile (ottima).

- Se $n \in \mathcal{O}$ allora **tutte** le attività $p(n) + 1, p(n) + 2, \ldots, n - 1$ intersecano n, quindi esse **non** possono essere in $\mathcal{O}$. In più $\mathcal{O} - \{n\}$ è una soluzione ottima per le attività $\{1, 2, \ldots, p(n)\}$ (che non intersecano l'attività n).

Sia $\mathcal{O}$ un sottoinsieme delle attività $A = \{A_1, \ldots, A_n\}$ composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile (ottima).

- Se $n \in \mathcal{O}$ allora **tutte** le attività $p(n) + 1, p(n) + 2, \ldots, n - 1$ intersecano n, quindi esse **non** possono essere in $\mathcal{O}$. In più $\mathcal{O} - \{n\}$ è una soluzione ottima per le attività $\{1, 2, \ldots, p(n)\}$ (che non intersecano l'attività n). Perchè ?

Sia $\mathcal{O}$ un sottoinsieme delle attività $A = \{A_1, \ldots, A_n\}$ composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile (ottima).

- Se $n \in \mathcal{O}$ allora **tutte** le attività $p(n) + 1, p(n) + 2, \ldots, n - 1$ intersecano n, quindi esse **non** possono essere in $\mathcal{O}$. In più $\mathcal{O} - \{n\}$ è una soluzione ottima per le attività $\{1, 2, \ldots, p(n)\}$ (che non intersecano l'attività n). Perchè ? Perchè se $\mathcal{O} - \{n\}$ **non** fosse ottima relativamente all'insieme delle attività $\{1, 2, \ldots, p(n)\}$, allora innanzitutto potremmo trovare una soluzione in $\{1, 2, \ldots, p(n)\}$ *migliore* di $\mathcal{O} - \{n\}$.

Sia $\mathcal{O}$ un sottoinsieme delle attività $A = \{A_1, \ldots, A_n\}$ composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile (ottima).

- Se $n \in \mathcal{O}$ allora **tutte** le attività $p(n) + 1, p(n) + 2, \ldots, n - 1$ intersecano n, quindi esse **non** possono essere in $\mathcal{O}$. In più $\mathcal{O} - \{n\}$ è una soluzione ottima per le attività $\{1, 2, \ldots, p(n)\}$ (che non intersecano l'attività n). Perchè ? Perchè se $\mathcal{O} - \{n\}$ **non** fosse ottima relativamente all'insieme delle attività $\{1, 2, \ldots, p(n)\}$, allora innanzitutto potremmo trovare una soluzione in $\{1, 2, \ldots, p(n)\}$ *migliore* di $\mathcal{O} - \{n\}$. Tale soluzione, **unita** all'attività n sarebbe una soluzione relativamente all'insieme delle attività $\{1, 2, \ldots, n\}$ **globalmente migliore** di $\mathcal{O}$ stessa!

Sia $\mathcal{O}$ un sottoinsieme delle attività $A = \{A_1, \ldots, A_n\}$ composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile (ottima).

- Se $n \in \mathcal{O}$ allora **tutte** le attività $p(n) + 1, p(n) + 2, \ldots, n - 1$ intersecano n, quindi esse **non** possono essere in $\mathcal{O}$. In più $\mathcal{O} - \{n\}$ è una soluzione ottima per le attività $\{1, 2, \ldots, p(n)\}$ (che non intersecano l'attività n). Perchè ? Perchè se $\mathcal{O} - \{n\}$ **non** fosse ottima relativamente all'insieme delle attività $\{1, 2, \ldots, p(n)\}$, allora innanzitutto potremmo trovare una soluzione in $\{1, 2, \ldots, p(n)\}$ *migliore* di $\mathcal{O} - \{n\}$. Tale soluzione, **unita** all'attività n sarebbe una soluzione relativamente all'insieme delle attività $\{1, 2, \ldots, n\}$ **globalmente migliore** di $\mathcal{O}$ stessa! Ciò è contro l'ipotesi di partenza che $\mathcal{O}$ è un sottoinsieme di A composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile.

Sia $\mathcal{O}$ un sottoinsieme delle attività $A = \{A_1, \ldots, A_n\}$ composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile (ottima).

- Se $n \in \mathcal{O}$ allora **tutte** le attività $p(n) + 1, p(n) + 2, \ldots, n - 1$ intersecano n, quindi esse **non** possono essere in $\mathcal{O}$. In più $\mathcal{O} - \{n\}$ è una soluzione ottima per le attività $\{1, 2, \ldots, p(n)\}$ (che non intersecano l'attività n). Perchè ? Perchè se $\mathcal{O} - \{n\}$ **non** fosse ottima relativamente all'insieme delle attività $\{1, 2, \ldots, p(n)\}$, allora innanzitutto potremmo trovare una soluzione in $\{1, 2, \ldots, p(n)\}$ *migliore* di $\mathcal{O} - \{n\}$. Tale soluzione, **unita** all'attività n sarebbe una soluzione relativamente all'insieme delle attività $\{1, 2, \ldots, n\}$ **globalmente migliore** di $\mathcal{O}$ stessa! Ciò è contro l'ipotesi di partenza che $\mathcal{O}$ è un sottoinsieme di A composta da attività a due a due compatibili, di valore totale $\sum_{A \in \mathcal{O}} v(A)$ **massimo** possibile.
- Se invece $n \notin \mathcal{O}$ allora $\mathcal{O}$ è chiaramente anche una soluzione ottima per l'insieme delle attività $\{1, 2, \ldots, n - 1\}$

Formuliamo la soluzione del prooblema in termini ricorsivi.

Formuliamo la soluzione del proobblema in termini ricorsivi.

$\forall 1 \leq j \leq n$, sia $\mathcal{O}_j$ una soluzione ottima per il sottoproblema costituito dalle attività $\{1, \ldots, j\}$, e sia $\texttt{OPT}(j)$ il valore di $\mathcal{O}_j$

Formuliamo la soluzione del prooblema in termini ricorsivi.

$\forall 1 \leq j \leq n$, sia $\mathcal{O}_j$ una soluzione ottima per il sottoproblema costituito dalle attività $\{1, \ldots, j\}$, e sia $\texttt{OPT}(j)$ il valore di $\mathcal{O}_j$ (noi cerchiamo $\texttt{OPT}(n)$).

Formuliamo la soluzione del proobblema in termini ricorsivi.

$\forall 1 \leq j \leq n$, sia $\mathcal{O}_j$ una soluzione ottima per il sottoproblema costituito dalle attività $\{1, \ldots, j\}$, e sia $\texttt{OPT}(j)$ il valore di $\mathcal{O}_j$ (noi cerchiamo $\texttt{OPT}(n)$).

• O vale che $j \in \mathcal{O}_j$ (ed in tal caso $\mathcal{O}_j$ **non** può contenere le attività $p(j) + 1, \ldots, j - 1$).

Formuliamo la soluzione del prooblema in termini ricorsivi.

$\forall 1 \leq j \leq n$, sia $\mathcal{O}_j$ una soluzione ottima per il sottoproblema costituito dalle attività $\{1, \ldots, j\}$, e sia $\texttt{OPT}(j)$ il valore di $\mathcal{O}_j$ (noi cerchiamo $\texttt{OPT}(n)$).

• O vale che $j \in \mathcal{O}_j$ (ed in tal caso $\mathcal{O}_j$ **non** può contenere le attività $p(j) + 1, \ldots, j - 1$). Inoltre $\mathcal{O}_j - \{j\}$ è una soluzione ottima (ovvero di valore $\texttt{OPT}(j)$) per le attività $\{1, 2, \ldots, p(j)\}$.

Formuliamo la soluzione del proublema in termini ricorsivi.

$\forall 1 \leq j \leq n$, sia $\mathcal{O}_j$ una soluzione ottima per il sottoproblema costituito dalle attività $\{1, \ldots, j\}$, e sia $\texttt{OPT}(j)$ il valore di $\mathcal{O}_j$ (noi cerchiamo $\texttt{OPT}(n)$).

• O vale che $j \in \mathcal{O}_j$ (ed in tal caso $\mathcal{O}_j$ **non** può contenere le attività $p(j)+1, \ldots, j-1$). Inoltre $\mathcal{O}_j - \{j\}$ è una soluzione ottima (ovvero di valore $\texttt{OPT}(j)$) per le attività $\{1, 2, \ldots, p(j)\}$. In simboli

$$\texttt{OPT}(j) = v_j + \texttt{OPT}(p(j)).$$

Formuliamo la soluzione del proobblema in termini ricorsivi.

$\forall 1 \leq j \leq n$, sia $\mathcal{O}_j$ una soluzione ottima per il sottoproblema costituito dalle attività $\{1, \ldots, j\}$, e sia $\mathtt{OPT}(j)$ il valore di $\mathcal{O}_j$ (noi cerchiamo $\mathtt{OPT}(n)$).

• O vale che $j \in \mathcal{O}_j$ (ed in tal caso $\mathcal{O}_j$ **non** può contenere le attività $p(j)+1, \ldots, j-1$). Inoltre $\mathcal{O}_j - \{j\}$ è una soluzione ottima (ovvero di valore $\mathtt{OPT}(j)$) per le attività $\{1, 2, \ldots, p(j)\}$. In simboli

$$\mathtt{OPT}(j) = v_j + \mathtt{OPT}(p(j)).$$

• Oppure vale che $j \notin \mathcal{O}_j$, ed in tal caso sappiamo che

$$\mathtt{OPT}(j) = \mathtt{OPT}(j-1).$$

Formuliamo la soluzione del proobblema in termini ricorsivi.

$\forall 1 \leq j \leq n$, sia $\mathcal{O}_j$ una soluzione ottima per il sottoproblema costituito dalle attività $\{1, \ldots, j\}$, e sia $\texttt{OPT}(j)$ il valore di $\mathcal{O}_j$ (noi cerchiamo $\texttt{OPT}(n)$).

• O vale che $j \in \mathcal{O}_j$ (ed in tal caso $\mathcal{O}_j$ **non** può contenere le attività $p(j) + 1, \ldots, j - 1$). Inoltre $\mathcal{O}_j - \{j\}$ è una soluzione ottima (ovvero di valore $\texttt{OPT}(j)$) per le attività $\{1, 2, \ldots, p(j)\}$. In simboli

$$\texttt{OPT}(j) = v_j + \texttt{OPT}(p(j)).$$

• Oppure vale che $j \notin \mathcal{O}_j$, ed in tal caso sappiamo che

$$\texttt{OPT}(j) = \texttt{OPT}(j - 1).$$

In sintesi

$$\texttt{OPT}(j) = \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j - 1)\}.$$

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

Il che ci potrebbe suggerire il seguente algoritmo

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

$$\mathtt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \mathtt{OPT}(p(j)), \mathtt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

Il che ci potrebbe suggerire il seguente algoritmo

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

Il che ci potrebbe suggerire il seguente algoritmo

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

Il che ci potrebbe suggerire il seguente algoritmo

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$
3. `Calcola-OPT`(j)
4. `IF`$(j == 0)$ {
5. `RETURN 0`

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

Il che ci potrebbe suggerire il seguente algoritmo

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$
3. `Calcola-OPT`(j)
4. `IF`$(j == 0)$ $\{$
5. `RETURN 0`
6. $\}$ `ELSE` $\{$
7. `RETURN` $\max\{v_j + \texttt{Calcola-OPT}(p(j)), \texttt{Calcola-OPT}(j-1)\}$
8. $\}$

$$\mathtt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \mathtt{OPT}(p(j)), \mathtt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

Il che ci potrebbe suggerire il seguente algoritmo

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

```
1. Ordina le attività in modo che f_1 ≤ ... ≤ f_n
2. Calcola p(1), ..., p(n)
3. Calcola-OPT(j)
4.       IF(j == 0) {
5.          RETURN 0
6.          } ELSE {
7.             RETURN max{v_j + Calcola-OPT(p(j)), Calcola-OPT(j − 1)}
8.          }
```

Se per sfortuna fosse $p(j) = j - 2$ (e ciò può accadere), avremmo che Calcola-OPT(n) effettua due chiamate: Calcola-OPT($n-2$) e Calcola-OPT($n-1$)

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

Il che ci potrebbe suggerire il seguente algoritmo

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$
3. `Calcola-OPT`(j)
4. `IF`$(j == 0)$ $\{$
5. `RETURN 0`
6. $\}$ `ELSE` $\{$
7. `RETURN` $\max\{v_j +$ `Calcola-OPT`$(p(j))$, `Calcola-OPT`$(j-1)\}$
8. $\}$

Se per sfortuna fosse $p(j) = j - 2$ (e ciò può accadere), avremmo che `Calcola-OPT`(n) effettua due chiamate: `Calcola-OPT`$(n-2)$ e `Calcola-OPT`$(n-1)$ $\Rightarrow$ la complessità $T(n)$ di `Calcola-OPT`(n) è tale che $T(n) = T(n-2) + T(n-1) + d$

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

Il che ci potrebbe suggerire il seguente algoritmo

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$
3. `Calcola-OPT`(j)
4. IF$(j == 0)$ {
5. RETURN 0
6. } ELSE {
7. RETURN $\max\{v_j + \texttt{Calcola-OPT}(p(j)), \texttt{Calcola-OPT}(j-1)\}$
8. }

Se per sfortuna fosse $p(j) = j - 2$ (e ciò può accadere), avremmo che `Calcola-OPT`(n) effettua due chiamate: `Calcola-OPT`$(n-2)$ e `Calcola-OPT`$(n-1)$ $\Rightarrow$ la complessità $T(n)$ di `Calcola-OPT`(n) è tale che $T(n) = T(n-2) + T(n-1) + d \Rightarrow T(n) = \Theta(2^n)$

Applichiamo PD con Memoization.

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$ % fà uso di un'array M

Applichiamo PD con Memoization.

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$ % fà uso di un'array M

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$

Applichiamo PD con Memoization.

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$ % fà uso di un'array M

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$

Applichiamo PD con Memoization.

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$ % fà uso di un'array M

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$
3. `M_Calcola_OPT`(j)

Applichiamo PD con Memoization.

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$ % fà uso di un'array M
1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$
3. `M_Calcola_OPT`(j)
4. `IF`$(j == 0)$ {
6. `RETURN 0`

Applichiamo PD con Memoization.

```
Input: n, s_1, ..., s_n, f_1, ..., f_n, v_1, ..., v_n        % fà uso di un'array M
1. Ordina le attività in modo che f_1 ≤ ... ≤ f_n
2. Calcola p(1), ..., p(n)
3. M_Calcola_OPT(j)
4.    IF(j == 0) {
6.       RETURN 0
7.       } ELSE {
8.       IF(M[j] non è definito) {
9.          M[j] = max{v_j + M_Calcola_OPT(p(j)), M_Calcola_OPT(j − 1)}
         }
```

Applichiamo PD con Memoization.

```
Input: n, s_1, ..., s_n, f_1, ..., f_n, v_1, ..., v_n       % fà uso di un'array M
1. Ordina le attività in modo che f_1 ≤ ... ≤ f_n
2. Calcola p(1), ..., p(n)
3. M_Calcola_OPT(j)
4.    IF(j == 0) {
6.       RETURN 0
7.       } ELSE {
8.       IF(M[j] non è definito) {
9.          M[j] = max{v_j + M_Calcola_OPT(p(j)), M_Calcola_OPT(j − 1)}
        }
10.   RETURN M[j]
```

Applichiamo PD con Memoization.

```
Input: n, s_1, ..., s_n, f_1, ..., f_n, v_1, ..., v_n        % fà uso di un'array M
1. Ordina le attività in modo che f_1 ≤ ... ≤ f_n
2. Calcola p(1), ..., p(n)
3. M_Calcola_OPT(j)
4.    IF(j == 0) {
6.       RETURN 0
7.       } ELSE {
8.       IF(M[j] non è definito) {
9.          M[j] = max{v_j + M_Calcola_OPT(p(j)), M_Calcola_OPT(j − 1)}
         }
10.   RETURN M[j]
```

La 1. richiede tempo $O(n \log n)$,

Applichiamo PD con Memoization.

```
Input: n, s_1, ..., s_n, f_1, ..., f_n, v_1, ..., v_n        % fà uso di un'array M
1. Ordina le attività in modo che f_1 ≤ ... ≤ f_n
2. Calcola p(1), ..., p(n)
3. M_Calcola_OPT(j)
4.    IF(j == 0) {
6.       RETURN 0
7.       } ELSE {
8.       IF(M[j] non è definito) {
9.          M[j] = max{v_j + M_Calcola_OPT(p(j)), M_Calcola_OPT(j − 1)}
        }
10.   RETURN M[j]
```

La 1. richiede tempo $O(n \log n)$, la 2. tempo $O(n \log n)$ (una volta aver ordinato).

Applichiamo PD con Memoization.

```
Input: n, s_1, ..., s_n, f_1, ..., f_n, v_1, ..., v_n        % fà uso di un'array M
1. Ordina le attività in modo che f_1 ≤ ... ≤ f_n
2. Calcola p(1), ..., p(n)
3. M_Calcola_OPT(j)
4.     IF(j == 0) {
6.        RETURN 0
7.        } ELSE {
8.        IF(M[j] non è definito) {
9.           M[j] = max{v_j + M_Calcola_OPT(p(j)), M_Calcola_OPT(j − 1)}
          }
10.    RETURN M[j]
```

La 1. richiede tempo $O(n \log n)$, la 2. tempo $O(n \log n)$ (una volta aver ordinato). M_Calcola_OPT(n) chiama M_Calcola_OPT(j), $\forall j < n$.

Applichiamo PD con Memoization.

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$ % fà uso di un'array M

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$
3. `M_Calcola_OPT`(j)
4. `IF`$(j == 0)$ {
6. `RETURN 0`
7. } `ELSE` {
8. `IF`$(M[j]$ non è definito$)$ {
9. $M[j] = \max\{v_j + \texttt{M_Calcola_OPT}(p(j)), \texttt{M_Calcola_OPT}(j-1)\}$

   }

10. `RETURN` $M[j]$

La 1. richiede tempo $O(n \log n)$, la 2. tempo $O(n \log n)$ (una volta aver ordinato). `M_Calcola_OPT`(n) chiama `M_Calcola_OPT`(j), $\forall j < n$. Ogni chiamata richiede tempo $O(1)$ e o ritorna un valore $M[j]$ già calcolato, oppure calcola un nuovo valore $M[j]$ facendo due chiamate a valori già calcolati.

Applichiamo PD con Memoization.

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$ % fà uso di un'array M

1. `Ordina le attività in modo che` $f_1 \leq \ldots \leq f_n$
2. `Calcola` $p(1), \ldots, p(n)$
3. `M_Calcola_OPT`(j)
4. `IF`$(j == 0)$ {
6. `RETURN 0`
7. } `ELSE` {
8. `IF`$(M[j]$ non è definito$)$ {
9. $M[j] = \max\{v_j + \texttt{M_Calcola_OPT}(p(j)), \texttt{M_Calcola_OPT}(j-1)\}$

   }
10. `RETURN` $M[j]$

La 1. richiede tempo $O(n \log n)$, la 2. tempo $O(n \log n)$ (una volta aver ordinato). `M_Calcola_OPT`(n) chiama `M_Calcola_OPT`(j), $\forall j < n$. Ogni chiamata richiede tempo $O(1)$ e o ritorna un valore $M[j]$ già calcolato, oppure calcola un nuovo valore $M[j]$ facendo due chiamate a valori già calcolati. Il numero totale di chiamate sarà al più pari a $2n$, da cui segue che il tempo impiegato da `M_Calcola_OPT`(n) è $O(n)$

Applichiamo PD con Memoization.

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$ % fà uso di un'array M

1. Ordina le attività in modo che $f_1 \leq \ldots \leq f_n$
2. Calcola $p(1), \ldots, p(n)$
3. M_Calcola_OPT(j)
4. IF($j == 0$) {
6. RETURN 0
7. } ELSE {
8. IF($M[j]$ non è definito) {
9. $M[j] = \max\{v_j + \texttt{M_Calcola_OPT}(p(j)), \texttt{M_Calcola_OPT}(j-1)\}$

   }

10. RETURN $M[j]$

La 1. richiede tempo $O(n \log n)$, la 2. tempo $O(n \log n)$ (una volta aver ordinato). M_Calcola_OPT(n) chiama M_Calcola_OPT(j), $\forall j < n$. Ogni chiamata richiede tempo $O(1)$ e o ritorna un valore $M[j]$ già calcolato, oppure calcola un nuovo valore $M[j]$ facendo due chiamate a valori già calcolati. Il numero totale di chiamate sarà al più pari a $2n$, da cui segue che il tempo impiegato da M_Calcola_OPT(n) è $O(n) \Rightarrow$ Complessità $= O(n \log n)$.

E se vogliamo trovare la soluzione ottima? (e non solo il suo valore).

E se vogliamo trovare la soluzione ottima? (e non solo il suo valore).

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

1. M_Calcola_OPT(n)
2. Trova_Soluzione(n)

E se vogliamo trovare la soluzione ottima? (e non solo il suo valore).

$$\mathtt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \mathtt{OPT}(p(j)), \mathtt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

```
1. M_Calcola_OPT(n)
2. Trova_Soluzione(n)
```

```
Trova_Soluzione(j)
1. IF(j==0) {
2.    RETURN nulla
```

E se vogliamo trovare la soluzione ottima? (e non solo il suo valore).

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

1. M_Calcola_OPT(n)
2. Trova_Soluzione(n)

```
Trova_Soluzione(j)
1. IF(j==0) {
2.    RETURN nulla
3.    } ELSE {
4.       IF(v_j + M[p(j)] > M[j − 1]) {
```

E se vogliamo trovare la soluzione ottima? (e non solo il suo valore).

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

```
1. M_Calcola_OPT(n)
2. Trova_Soluzione(n)
```

```
Trova_Soluzione(j)
1. IF(j==0) {
2.    RETURN nulla
3.    } ELSE {
4.       IF(v_j + M[p(j)] > M[j − 1]) {
5.          stampa j
```

E se vogliamo trovare la soluzione ottima? (e non solo il suo valore).

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

1. M_Calcola_OPT(n)
2. Trova_Soluzione(n)

```
Trova_Soluzione(j)
1. IF(j==0) {
2.    RETURN nulla
3.    } ELSE {
4.       IF(v_j + M[p(j)] > M[j − 1]) {
5.          stampa j
6.       Trova_Soluzione(p(j))
```

E se vogliamo trovare la soluzione ottima? (e non solo il suo valore).

$$\mathtt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \mathtt{OPT}(p(j)), \mathtt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

```
1. M_Calcola_OPT(n)
2. Trova_Soluzione(n)
```

```
Trova_Soluzione(j)
1. IF(j==0) {
2.   RETURN nulla
3.   } ELSE {
4.     IF(v_j + M[p(j)] > M[j − 1]) {
5.       stampa j
6.     Trova_Soluzione(p(j))
7.   } ELSE {Trova_Soluzione(j − 1)
       }
   }
```

E se vogliamo trovare la soluzione ottima? (e non solo il suo valore).

$$\texttt{OPT}(j) = \begin{cases} 0 & \text{se } j = 0, \\ \max\{v_j + \texttt{OPT}(p(j)), \texttt{OPT}(j-1)\} & \text{se } j \geq 1 \end{cases}$$

```
1. M_Calcola_OPT(n)
2. Trova_Soluzione(n)
```

```
Trova_Soluzione(j)
1. IF(j==0) {
2.    RETURN nulla
3.    } ELSE {
4.       IF(v_j + M[p(j)] > M[j-1]) {
5.          stampa j
6.       Trova_Soluzione(p(j))
7.    } ELSE {Trova_Soluzione(j-1)
        }
   }
```

Complessità : $O(n)$

Esercizio:
Consideriamo un gioco in cui vi sono n monete di valori $v_1, \ldots, v_n$, rispettivamente, disposte su di una riga da sinistra a destra con n pari, e due giocatori: Alice e Bob.

Esercizio:
Consideriamo un gioco in cui vi sono n monete di valori $v_1, \ldots, v_n$, rispettivamente, disposte su di una riga da sinistra a destra con n pari, e due giocatori: Alice e Bob.

All'istante 1 Alice sceglie una delle due monete all'estremità della riga (quindi, all'inizio del gioco Alice può scegliere una tra le monete di valore v_1 e v_n), **e se la prende**.

Esercizio:
Consideriamo un gioco in cui vi sono n monete di valori $v_1, \ldots, v_n$, rispettivamente, disposte su di una riga da sinistra a destra con n pari, e due giocatori: Alice e Bob.

All'istante 1 Alice sceglie una delle due monete all'estremità della riga (quindi, all'inizio del gioco Alice può scegliere una tra le monete di valore v_1 e v_n), **e se la prende**.

All'istante 2 Bob sceglierà una delle due monete all'estremità della riga **rimanente**

Esercizio:
Consideriamo un gioco in cui vi sono n monete di valori $v_1, \ldots, v_n$, rispettivamente, disposte su di una riga da sinistra a destra con n pari, e due giocatori: Alice e Bob.

All'istante 1 Alice sceglie una delle due monete all'estremità della riga (quindi, all'inizio del gioco Alice può scegliere una tra le monete di valore v_1 e v_n), **e se la prende**.

All'istante 2 Bob sceglierà una delle due monete all'estremità della riga **rimanente** (quindi, Bob può scegliere una tra le monete di valore v_2 e v_n, **se** Alice ha scelto nel turno precedente la moneta di valore v_1,

Esercizio:
Consideriamo un gioco in cui vi sono n monete di valori $v_1, \ldots, v_n$, rispettivamente, disposte su di una riga da sinistra a destra con n pari, e due giocatori: Alice e Bob.

All'istante 1 Alice sceglie una delle due monete all'estremità della riga (quindi, all'inizio del gioco Alice può scegliere una tra le monete di valore v_1 e v_n), **e se la prende**.

All'istante 2 Bob sceglierà una delle due monete all'estremità della riga **rimanente** (quindi, Bob può scegliere una tra le monete di valore v_2 e v_n, **se** Alice ha scelto nel turno precedente la moneta di valore v_1, oppure Bob potrà scegliere una tra le monete di valore v_1 e v_{n-1}, se Alice ha scelto nel suo turno la moneta di valore v_n), e la rimuove dalla riga.

Esercizio:
Consideriamo un gioco in cui vi sono n monete di valori $v_1, \ldots, v_n$, rispettivamente, disposte su di una riga da sinistra a destra con n pari, e due giocatori: Alice e Bob.

All'istante 1 Alice sceglie una delle due monete all'estremità della riga (quindi, all'inizio del gioco Alice può scegliere una tra le monete di valore v_1 e v_n), **e se la prende**.

All'istante 2 Bob sceglierà una delle due monete all'estremità della riga **rimanente** (quindi, Bob può scegliere una tra le monete di valore v_2 e v_n, **se** Alice ha scelto nel turno precedente la moneta di valore v_1, oppure Bob potrà scegliere una tra le monete di valore v_1 e v_{n-1}, se Alice ha scelto nel suo turno la moneta di valore v_n), e la rimuove dalla riga.

Il gioco procede in questo modo, a turni alterni, fino a quando le monete sono esaurite.

Esercizio:
Consideriamo un gioco in cui vi sono n monete di valori $v_1, \dots, v_n$, rispettivamente, disposte su di una riga da sinistra a destra con n pari, e due giocatori: Alice e Bob.

All'istante 1 Alice sceglie una delle due monete all'estremità della riga (quindi, all'inizio del gioco Alice può scegliere una tra le monete di valore v_1 e v_n), **e se la prende**.

All'istante 2 Bob sceglierà una delle due monete all'estremità della riga **rimanente** (quindi, Bob può scegliere una tra le monete di valore v_2 e v_n, **se** Alice ha scelto nel turno precedente la moneta di valore v_1, oppure Bob potrà scegliere una tra le monete di valore v_1 e v_{n-1}, se Alice ha scelto nel suo turno la moneta di valore v_n), e la rimuove dalla riga.

Il gioco procede in questo modo, a turni alterni, fino a quando le monete sono esaurite. Vince il giocatore che, alla fine del gioco, ha scelto le $n/2$ monete di valore totale *massimo*.

l problema algoritmico che vogliamo risolvere è di determinare il valore **massimo** che il primo giocatore (Alice) può ottenere, sotto l'ipotesi che anche il suo opponente Bob giochi in maniera razionale.

l problema algoritmico che vogliamo risolvere è di determinare il valore **massimo** che il primo giocatore (Alice) può ottenere, sotto l'ipotesi che anche il suo opponente Bob giochi in maniera razionale.

Esempio: abbiamo 4 monete, di valore $2, 4, 8, 10$ e disposte nell'ordine.

l problema algoritmico che vogliamo risolvere è di determinare il valore **massimo** che il primo giocatore (Alice) può ottenere, sotto l'ipotesi che anche il suo opponente Bob giochi in maniera razionale.

Esempio: abbiamo 4 monete, di valore $2, 4, 8, 10$ e disposte nell'ordine. Il massimo valore che il primo giocatore può ottenere è pari a 14 (scegliendo le monete di valore 10 e 4).

l problema algoritmico che vogliamo risolvere è di determinare il valore **massimo** che il primo giocatore (Alice) può ottenere, sotto l'ipotesi che anche il suo opponente Bob giochi in maniera razionale.

Esempio: abbiamo 4 monete, di valore $2, 4, 8, 10$ e disposte nell'ordine. Il massimo valore che il primo giocatore può ottenere è pari a 14 (scegliendo le monete di valore 10 e 4).

Se le monete fossero invece $8, 20, 3, 2$ allora il massimo valore che il primo giocatore può ottenere è pari a 22 (scegliendo le monete di valore 2 e 20).

l problema algoritmico che vogliamo risolvere è di determinare il valore **massimo** che il primo giocatore (Alice) può ottenere, sotto l'ipotesi che anche il suo opponente Bob giochi in maniera razionale.

Esempio: abbiamo 4 monete, di valore $2, 4, 8, 10$ e disposte nell'ordine. Il massimo valore che il primo giocatore può ottenere è pari a 14 (scegliendo le monete di valore 10 e 4).

Se le monete fossero invece $8, 20, 3, 2$ allora il massimo valore che il primo giocatore può ottenere è pari a 22 (scegliendo le monete di valore 2 e 20).

La semplice strategia di scegliere ad ogni passo la moneta disponibile di valore maggiore **non** ci porta alla soluzione migliore.

l problema algoritmico che vogliamo risolvere è di determinare il valore **massimo** che il primo giocatore (Alice) può ottenere, sotto l'ipotesi che anche il suo opponente Bob giochi in maniera razionale.

Esempio: abbiamo 4 monete, di valore $2, 4, 8, 10$ e disposte nell'ordine. Il massimo valore che il primo giocatore può ottenere è pari a 14 (scegliendo le monete di valore 10 e 4).

Se le monete fossero invece $8, 20, 3, 2$ allora il massimo valore che il primo giocatore può ottenere è pari a 22 (scegliendo le monete di valore 2 e 20).

La semplice strategia di scegliere ad ogni passo la moneta disponibile di valore maggiore **non** ci porta alla soluzione migliore. Infatti, se il primo giocatore seguisse questa strategia, sceglierebbe al primo passo la moneta di valore 8, l'avversario sceglierà la moneta di valore 20, ed il primo giocatore sceglierebbe la moneta di valore 3, per un valore totale $=11 < 22$.

Studiamo ora il caso generale. Sia $n > 0$ e sia data una sequenza di valori di monete $v_1, \ldots, v_n$. Assumiamo che Alice giochi per prima.

Studiamo ora il caso generale. Sia $n > 0$ e sia data una sequenza di valori di monete $v_1, \ldots, v_n$. Assumiamo che Alice giochi per prima.
Indichiamo con $f(i,j)$ il **massimo** valore che Alice può ottenere sulla sottosequenza di valore $v_i, \ldots, v_j$

Studiamo ora il caso generale. Sia $n > 0$ e sia data una sequenza di valori di monete $v_1, \ldots, v_n$. Assumiamo che Alice giochi per prima. Indichiamo con $f(i, j)$ il **massimo** valore che Alice può ottenere sulla sottosequenza di valore $v_i, \ldots, v_j$ (assumendo che l'avversario Bob si comporti in maniera razionale, ovvero che cerchi di **massimizzare** il suo profitto, e quindi di **minimizzare** il profitto di Alice).

Studiamo ora il caso generale. Sia $n > 0$ e sia data una sequenza di valori di monete $v_1, \ldots, v_n$. Assumiamo che Alice giochi per prima.
Indichiamo con $f(i, j)$ il **massimo** valore che Alice può ottenere sulla sottosequenza di valore $v_i, \ldots, v_j$ (assumendo che l'avversario Bob si comporti in maniera razionale, ovvero che cerchi di **massimizzare** il suo profitto, e quindi di **minimizzare** il profitto di Alice).
Chiaramente, se $i = j$ varrà $f(i, j) = v_i$.

Studiamo ora il caso generale. Sia $n > 0$ e sia data una sequenza di valori di monete $v_1, \ldots, v_n$. Assumiamo che Alice giochi per prima.
Indichiamo con $f(i, j)$ il **massimo** valore che Alice può ottenere sulla sottosequenza di valore $v_i, \ldots, v_j$ (assumendo che l'avversario Bob si comporti in maniera razionale, ovvero che cerchi di **massimizzare** il suo profitto, e quindi di **minimizzare** il profitto di Alice).
Chiaramente, se $i = j$ varrà $f(i, j) = v_i$. Un'altro caso semplice è il calcolo di $f(i, i + 1)$ che ovviamente è pari a $\max\{v_i, v_{i+1}\}$.

Studiamo ora il caso generale. Sia $n > 0$ e sia data una sequenza di valori di monete $v_1, \dots, v_n$. Assumiamo che Alice giochi per prima.
Indichiamo con $f(i,j)$ il **massimo** valore che Alice può ottenere sulla sottosequenza di valore $v_i, \dots, v_j$ (assumendo che l'avversario Bob si comporti in maniera razionale, ovvero che cerchi di **massimizzare** il suo profitto, e quindi di **minimizzare** il profitto di Alice).
Chiaramente, se $i = j$ varrà $f(i,j) = v_i$. Un'altro caso semplice è il calcolo di $f(i, i+1)$ che ovviamente è pari a $\max\{v_i, v_{i+1}\}$. Consideriamo quindi il caso generale in cui $i < j - 1$. Varrà

$$
\begin{aligned}
f(i,j) = \max\Big(& v_i + \min\{f(i+2,j), f(i+1,j-1)\}, \\
& v_j + \min\{f(i+1,j-1), f(i,j-2)\}\Big).
\end{aligned}
$$

Studiamo ora il caso generale. Sia $n > 0$ e sia data una sequenza di valori di monete $v_1, \ldots, v_n$. Assumiamo che Alice giochi per prima.
Indichiamo con $f(i,j)$ il **massimo** valore che Alice può ottenere sulla sottosequenza di valore $v_i, \ldots, v_j$ (assumendo che l'avversario Bob si comporti in maniera razionale, ovvero che cerchi di **massimizzare** il suo profitto, e quindi di **minimizzare** il profitto di Alice).
Chiaramente, se $i = j$ varrà $f(i,j) = v_i$. Un'altro caso semplice è il calcolo di $f(i, i+1)$ che ovviamente è pari a $\max\{v_i, v_{i+1}\}$. Consideriamo quindi il caso generale in cui $i < j - 1$. Varrà

$$
\begin{aligned}
f(i,j) = \max\Big(& v_i + \min\{f(i+2,j), f(i+1,j-1)\}, \\
& v_j + \min\{f(i+1,j-1), f(i,j-2)\}\Big).
\end{aligned}
$$

Perchè ?

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i.

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1}

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1} (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+2} \dots v_j$, ovvero risolvere $f(i+2, j)$)

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1} (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+2} \dots v_j$, ovvero risolvere $f(i+2, j)$) **oppure** la moneta di valore v_j

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1} (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+2} \dots v_j$, ovvero risolvere $f(i+2, j)$) **oppure** la moneta di valore v_j (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+1} \dots v_{j-1}$, ovvero risolvere $f(i+1, j-1)$),

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1} (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+2} \dots v_j$, ovvero risolvere $f(i+2, j)$) **oppure** la moneta di valore v_j (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+1} \dots v_{j-1}$, ovvero risolvere $f(i+1, j-1)$), a seconda di quale massimizza il suo profitto (ovvero **minimizza** il profitto di Alice).

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1} (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+2} \dots v_j$, ovvero risolvere $f(i+2, j)$) **oppure** la moneta di valore v_j (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+1} \dots v_{j-1}$, ovvero risolvere $f(i+1, j-1)$), a seconda di quale massimizza il suo profitto (ovvero **minimizza** il profitto di Alice). Detto in altri termini, Bob per massimizzare il suo profitto effettuerà una scelta che porterà al sottoproblema che ottiene il $\min\{f(i+2, j), f(i+1, j-1)\}$.

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1} (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+2} \dots v_j$, ovvero risolvere $f(i+2, j)$) **oppure** la moneta di valore v_j (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+1} \dots v_{j-1}$, ovvero risolvere $f(i+1, j-1)$), a seconda di quale massimizza il suo profitto (ovvero **minimizza** il profitto di Alice). Detto in altri termini, Bob per massimizzare il suo profitto effettuerà una scelta che porterà al sottoproblema che ottiene il $\min\{f(i+2, j), f(i+1, j-1)\}$. Questo sottoproblema è quello che poi dovrà risolvere Alice nel suo successivo turno.

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1} (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+2} \dots v_j$, ovvero risolvere $f(i+2, j)$) **oppure** la moneta di valore v_j (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+1} \dots v_{j-1}$, ovvero risolvere $f(i+1, j-1)$), a seconda di quale massimizza il suo profitto (ovvero **minimizza** il profitto di Alice). Detto in altri termini, Bob per massimizzare il suo profitto effettuerà una scelta che porterà al sottoproblema che ottiene il $\min\{f(i+2, j), f(i+1, j-1)\}$. Questo sottoproblema è quello che poi dovrà risolvere Alice nel suo successivo turno.
- Alice sceglie la moneta di valore v_j.

Per giustificare la formula ricorsiva, consideriamo distintamente i due casi possibili, ovvero che Alice scelga la moneta di valore v_i o quella di valore v_j.

- Alice sceglie la moneta di valore v_i. In questo caso, Bob sceglierà **o** la moneta di valore v_{i+1} (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+2} \dots v_j$, ovvero risolvere $f(i+2, j)$) **oppure** la moneta di valore v_j (e quindi al passo successivo Alice dovrà poi scegliere tra le monete $v_{i+1} \dots v_{j-1}$, ovvero risolvere $f(i+1, j-1)$), a seconda di quale massimizza il suo profitto (ovvero **minimizza** il profitto di Alice). Detto in altri termini, Bob per massimizzare il suo profitto effettuerà una scelta che porterà al sottoproblema che ottiene il $\min\{f(i+2, j), f(i+1, j-1)\}$. Questo sottoproblema è quello che poi dovrà risolvere Alice nel suo successivo turno.
- Alice sceglie la moneta di valore v_j. La discussione è analoga.

Di conseguenza, Alice sa che se sceglierà la moneta di valore v_i *allora* Bob effettuerà la scelta che ottiene il

$$\min\{f(i+2,j), f(i+1,j-1)\}$$

Di conseguenza, Alice sa che se sceglierà la moneta di valore v_i *allora* Bob effettuerà la scelta che ottiene il

$$\min\{f(i+2,j), f(i+1,j-1)\}$$

mentre se sceglierà la moneta di valore v_j *allora* Bob effettuerà la scelta che ottiene il

$$\min\{f(i+1,j-1), f(i,j-2)\}$$

Di conseguenza, Alice sa che se sceglierà la moneta di valore v_i *allora* Bob effettuerà la scelta che ottiene il

$$\min\{f(i+2,j), f(i+1,j-1)\}$$

mentre se sceglierà la moneta di valore v_j *allora* Bob effettuerà la scelta che ottiene il

$$\min\{f(i+1,j-1), f(i,j-2)\}$$

Di conseguenza, per massimizzare il suo profitto, tra le monete di valore v_i e v_j Alice sceglierà quella che ottiene il max indicato nella

$$f(i,j) = \max\Big(v_i + \min\{f(i+2,j), f(i+1,j-1)\}, \\ v_j + \min\{f(i+1,j-1), f(i,j-2)\}\Big).$$

Per esercizio, si derivi l'algoritmo di Programmazione Dinamica che calcola $f(1,n)$ usando l'equazione di ricorrenza ottenuta e le condizioni iniziali $f(i,i) = v_i$ e $f(i,i+1) = \max\{v_i, v_{i+1}\}$. Inoltre, valutatene la complessità .