

Lezione 32

Tecnica Backtracking

Ricordiamo lo schema generale del Backtracking

BACKTRACKING(*SOL*, *k*)

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)  
IF (( $a_1, \dots, a_k$ ) è una soluzione) {stampala
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)  
IF (( $a_1, \dots, a_k$ ) è una soluzione) {stampala  
  } ELSE {  
    calcola  $S_{k+1}$  % =insieme degli elementi con cui è possibile  
                   estendere la soluzione parziale corrente
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF (( $a_1, \dots, a_k$ ) è una soluzione) {stampala
} ELSE {
  calcola  $S_{k+1}$  % =insieme degli elementi con cui è possibile
                estendere la soluzione parziale corrente
  WHILE  $S_{k+1} \neq \emptyset$  {
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF (( $a_1, \dots, a_k$ ) è una soluzione) {stampala
} ELSE {
  calcola  $S_{k+1}$  % =insieme degli elementi con cui è possibile
                estendere la soluzione parziale corrente
  WHILE  $S_{k+1} \neq \emptyset$  {
     $a_{k+1} \leftarrow$  un elemento di  $S_{k+1}$ ,
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF (( $a_1, \dots, a_k$ ) è una soluzione) {stampala
} ELSE {
  calcola  $S_{k+1}$  % =insieme degli elementi con cui è possibile
                estendere la soluzione parziale corrente
  WHILE  $S_{k+1} \neq \emptyset$  {
     $a_{k+1} \leftarrow$  un elemento di  $S_{k+1}$ ,
    soluzione parziale= ( $a_1, \dots, a_k, a_{k+1}$ )
```


Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF (( $a_1, \dots, a_k$ ) è una soluzione) {stampala
} ELSE {
  calcola  $S_{k+1}$  % =insieme degli elementi con cui è possibile
                estendere la soluzione parziale corrente
  WHILE  $S_{k+1} \neq \emptyset$  {
     $a_{k+1} \leftarrow$  un elemento di  $S_{k+1}$ ,
    soluzione parziale= ( $a_1, \dots, a_k, a_{k+1}$ )
     $S_{k+1} \leftarrow S_{k+1} - \{a_{k+1}\}$ 
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF (( $a_1, \dots, a_k$ ) è una soluzione) {stampala
} ELSE {
  calcola  $S_{k+1}$  % =insieme degli elementi con cui è possibile
                    estendere la soluzione parziale corrente
  WHILE  $S_{k+1} \neq \emptyset$  {
     $a_{k+1} \leftarrow$  un elemento di  $S_{k+1}$ ,
    soluzione parziale= ( $a_1, \dots, a_k, a_{k+1}$ )
     $S_{k+1} \leftarrow S_{k+1} - \{a_{k+1}\}$ 
    BACKTRACKING(SOL,  $k + 1$ )
  }
}
```

Abbiamo anche visto che, se le soluzioni che andiamo cercando devono soddisfare dati “vincoli”, allora è possibile usare funzioni di taglio per evitare di visitare parti dell’albero delle soluzioni in cui sappiamo *non* esistono soluzioni ammissibili,

Abbiamo anche visto che, se le soluzioni che andiamo cercando devono soddisfare dati “vincoli”, allora è possibile usare funzioni di taglio per evitare di visitare parti dell’albero delle soluzioni in cui sappiamo *non* esistono soluzioni ammissibili, ovvero soluzioni che non rispettano i vincoli dati.

Problema delle 8 regine

Problema delle 8 regine

Problema: Posizionare n regine in una scacchiera $n \times n$, in modo tale che nessuna regina ne "minacci" un'altra.

Problema delle 8 regine

Problema: Posizionare n regine in una scacchiera $n \times n$, in modo tale che nessuna regina ne "minacci" un'altra.

Cosa vuol dire "non minaccia?"

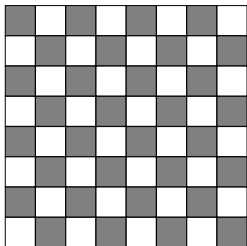
Problema delle 8 regine

Problema: Posizionare n regine in una scacchiera $n \times n$, in modo tale che nessuna regina ne "minacci" un'altra.

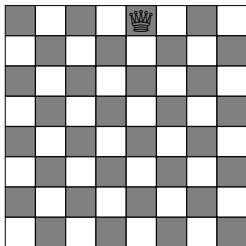
Cosa vuol dire "non minaccia?"

Che *nessuna* regina si trovi sulla stessa riga, colonna o diagonale di *ogni* altra regina.

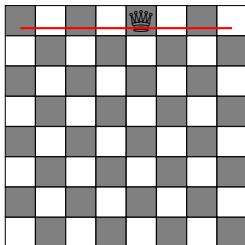
Esempio



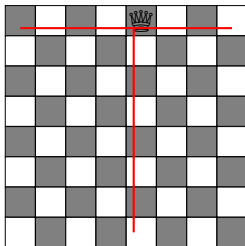
Esempio



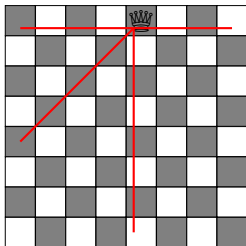
Esempio



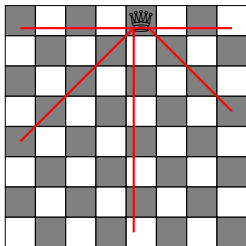
Esempio



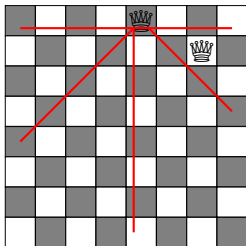
Esempio



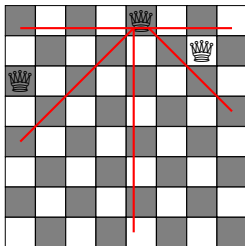
Esempio



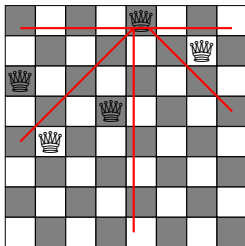
Esempio



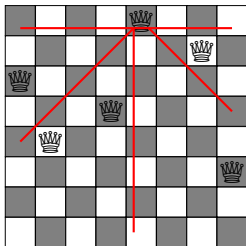
Esempio



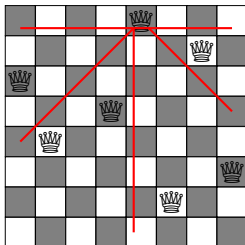
Esempio



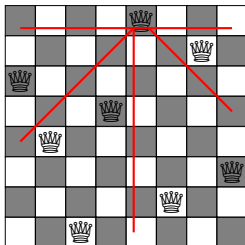
Esempio



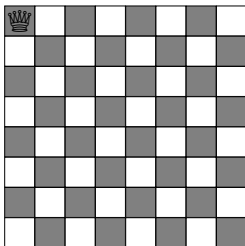
Esempio



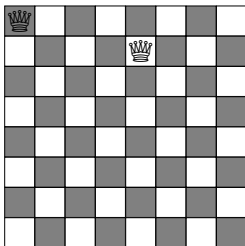
Esempio



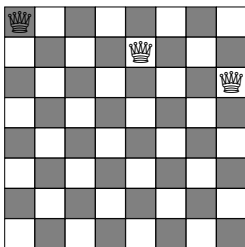
Un'altra possibile soluzione



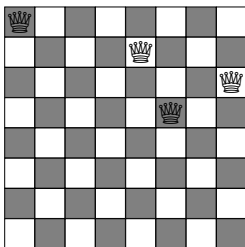
Un'altra possibile soluzione



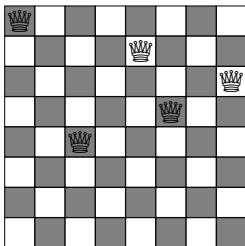
Un'altra possibile soluzione



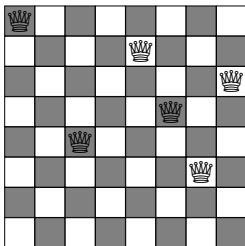
Un'altra possibile soluzione



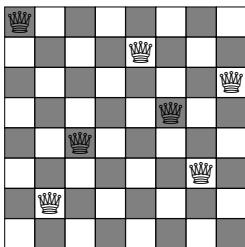
Un'altra possibile soluzione



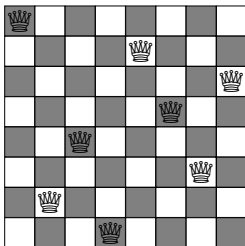
Un'altra possibile soluzione



Un'altra possibile soluzione



Un'altra possibile soluzione

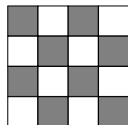


Rappresentiamo le posizioni delle regine usando un array $Q[1 \dots n]$, dove $Q[i]$ indica la colonna della riga i -esima che contiene una regina.

Rappresentiamo le posizioni delle regine usando un array $Q[1 \dots n]$, dove $Q[i]$ indica la colonna della riga i -esima che contiene una regina.

Quando chiamiamo l'algoritmo `PlaceQueens`, (che vediamo nelle prossime slide) il parametro input r è l'indice della prima riga vuota e il prefisso $Q[1 \dots r - 1]$ contiene le posizioni delle prime $r - 1$ regine.

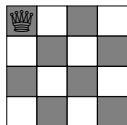
Esempio



L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

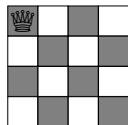
Esempio



L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Esempio

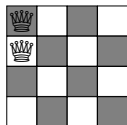


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima),

Esempio

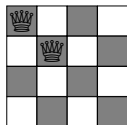


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

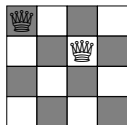


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

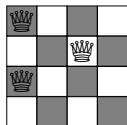


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

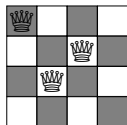


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

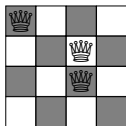


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

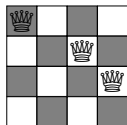


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

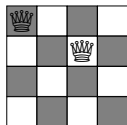


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

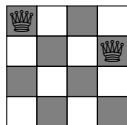


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

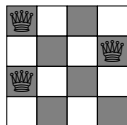


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

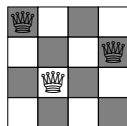


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

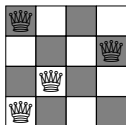


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

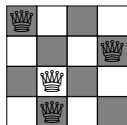


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

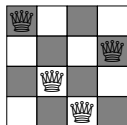


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

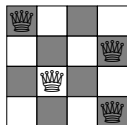


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

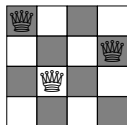


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

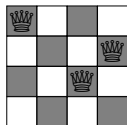


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

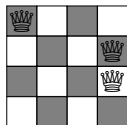


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

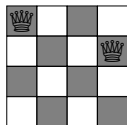


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

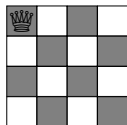


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

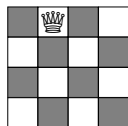


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

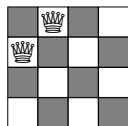


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

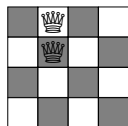


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

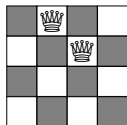


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

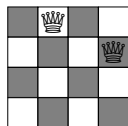


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

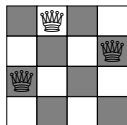


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

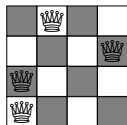


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

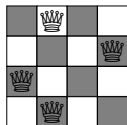


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio

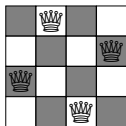


L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Esempio



L'algoritmo funziona nel modo seguente.

Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

PlaceQueens($Q[1 \dots n], r$)


```
PlaceQueens( $Q[1 \dots n], r$ )
```

```
1. IF ( $r = n + 1$ ) {
```

```
2.     print  $Q[1 \dots n]$ 
```

```
PlaceQueens( $Q[1 \dots n], r$ )
```

1. IF ($r = n + 1$){
2. print $Q[1 \dots n]$
3. }ELSE{
4. FOR($j=1, j < n+1, j=j+1$){
5. legal=True

PlaceQueens($Q[1 \dots n], r$)

1. IF ($r = n + 1$){
2. print $Q[1 \dots n]$
3. }ELSE{
4. FOR($j=1, j < n+1, j=j+1$){
5. legal=True
6. FOR($i = 1, i < r, i = i + 1$){
7. IF($((Q[i] = j) || (Q[i] = j + r - i) || (Q[i] = j - r + i))$){

PlaceQueens($Q[1 \dots n], r$)

1. IF ($r = n + 1$){
2. print $Q[1 \dots n]$
3. }ELSE{
4. FOR($j=1, j < n+1, j=j+1$){
5. legal=True
6. FOR($i = 1, i < r, i = i + 1$){
7. IF($(Q[i] = j) || (Q[i] = j + r - i) || (Q[i] = j - r + i)$){
8. legal=False

```
PlaceQueens(Q[1...n], r)
```

```
1. IF (r = n + 1){  
2.     print Q[1...n]  
3.     }ELSE{  
4.         FOR(j=1, j<n+1, j=j+1){  
5.             legal=True  
6.             FOR(i = 1, i < r, i = i + 1){  
7.                 IF((Q[i] = j)||((Q[i] = j + r - i)||((Q[i] = j - r + i))){  
8.                     legal=False  
9.                 }  
10.            }  
11.        }  
12.    }  
13. }  
14. IF(legal){
```

PlaceQueens($Q[1 \dots n], r$)

```
1. IF ( $r = n + 1$ ){
2.     print  $Q[1 \dots n]$ 
3.     }ELSE{
4.         FOR( $j=1, j < n+1, j=j+1$ ){
5.             legal=True
6.             FOR( $i = 1, i < r, i = i + 1$ ){
7.                 IF( $((Q[i] = j) || (Q[i] = j + r - i) || (Q[i] = j - r + i))$ ){
8.                     legal=False
9.                 }
10.            }
11.        }
12.    }
13.    IF(legal){
14.         $Q[r] = j$ 
```

```

PlaceQueens(Q[1...n], r)
1.  IF (r = n + 1){
2.      print Q[1...n]
3.      }ELSE{
4.          FOR(j=1, j<n+1, j=j+1){
5.              legal=True
6.              FOR(i = 1, i < r, i = i + 1){
7.                  IF((Q[i] = j)||((Q[i] = j + r - i)||((Q[i] = j - r + i))){
8.                      legal=False
9.                  }
10.             }
11.             IF(legal){
12.                 Q[r] = j
13.                 PlaceQueens(Q[1..n], r + 1)
14.             }
15.         }
16.     }
17. }

```

```

PlaceQueens(Q[1 . . . n], r)
1.  IF (r = n + 1){
2.      print Q[1 . . . n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)|| (Q[i] =
                    j+r-i)|| (Q[i] = j-r+i)){
8.                  legal=False
9.              }
10.             }
11.             IF(legal){
12.                 Q[r] = j
13.                 PlaceQueens(Q[1..n], r + 1)
14.             }
15.         }
16.     }
17. }

```

L'algoritmo funziona nel modo seguente.

Se $r = n + 1$ vuol dire che stiamo tentando di posizionare la $n+1$ -esima regine (che non esiste...) cioè abbiamo correttamente posizionato le precedenti n regine. Ritorniamo quindi la soluzione.


```

PlaceQueens(Q[1 . . . n], r)
1.  IF (r = n + 1){
2.      print Q[1 . . . n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)|| (Q[i] =
                    j+r-i)|| (Q[i] = j-r+i)){
8.                  legal=False
                    }
                }
9.          IF(legal){
10.             Q[r] = j
11.             PlaceQueens(Q[1..n], r + 1)
                }
            }
        }
    }
}

```

L'algoritmo funziona nel modo seguente.

Se $r = n + 1$ vuol dire che stiamo tentando di posizionare la $n+1$ -esima regine (che non esiste...) cioè abbiamo correttamente posizionato le precedenti n regine. Ritorniamo quindi la soluzione.

Altrimenti, nel FOR della linea 4. cerchiamo una colonna $j \in \{1, 2, \dots, n\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

```

PlaceQueens(Q[1...n], r)
1.  IF (r = n + 1){
2.      print Q[1...n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)|| (Q[i] =
                    j+r-i)|| (Q[i] = j-r+i)){
8.                  legal=False
                    }
                }
9.          IF(legal){
10.             Q[r] = j
11.             PlaceQueens(Q[1..n], r + 1)
                }
            }
        }
    }
}

```

L'algoritmo funziona nel modo seguente.

Se $r = n + 1$ vuol dire che stiamo tentando di posizionare la $n+1$ -esima regine (che non esiste...) cioè abbiamo correttamente posizionato le precedenti n regine. Ritorniamo quindi la soluzione.

Altrimenti, nel FOR della linea 4. cerchiamo una colonna $j \in \{1, 2, \dots, n\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

```

PlaceQueens(Q[1...n], r)
1.  IF (r = n + 1){
2.      print Q[1...n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)|| (Q[i] =
                    j+r-i)|| (Q[i] = j-r+i)){
8.                  legal=False
                }
            }
9.          IF(legal){
10.             Q[r] = j
11.             PlaceQueens(Q[1..n], r + 1)
                }
            }
        }
    }
}

```

L'algoritmo funziona nel modo seguente.

Se $r = n + 1$ vuol dire che stiamo tentando di posizionare la $n+1$ -esima regine (che non esiste...) cioè abbiamo correttamente posizionato le precedenti n regine. Ritorniamo quindi la soluzione.

Altrimenti, nel FOR della linea 4. cerchiamo una colonna $j \in \{1, 2, \dots, n\}$ dove piazzare la regina r -esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \dots, n\}$ (valore della colonna candidata ad ospitare la regina r -esima), andiamo a controllare che nessuna delle regine già piazzate minacci la regina che vorremmo piazzare nella colonna j della riga r .

Questa funzione di taglio, la realizziamo nel FOR della linea 6.

Ovvero, controlliamo (per tutte le precedenti r regine già piazzate) se una di esse stà nella stessa colonna j ,

```
PlaceQueens(Q[1..n], r)
1.  IF (r = n + 1){
2.      print Q[1..n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)||((Q[i] =
                j+r-i)||((Q[i] = j-r+i)){
8.                  legal=False
                }
            }
9.          IF(legal){
10.             Q[r] = j
11.             PlaceQueens(Q[1..n], r + 1)
            }
        }
    }
```

```

PlaceQueens(Q[1..n], r)
1.  IF (r = n + 1){
2.      print Q[1..n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)||((Q[i] =
                j+r-i)||((Q[i] = j-r+i)){
8.                  legal=False
                }
            }
9.          IF(legal){
10.             Q[r] = j
11.             PlaceQueens(Q[1..n], r + 1)
            }
        }
    }
}

```

Ovvero, controlliamo (per tutte le precedenti r regine già piazzate) se una di esse stà nella stessa colonna j , oppure in una delle due diagonali che passano per la casella della riga r e colonna j .

```

PlaceQueens(Q[1..n], r)
1.  IF (r = n + 1){
2.      print Q[1..n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)||((Q[i] =
                j+r-i)||((Q[i] = j-r+i)){
8.                  legal=False
                }
            }
9.          IF(legal){
10.             Q[r] = j
11.             PlaceQueens(Q[1..n], r + 1)
            }
        }
    }
}

```

Ovvero, controlliamo (per tutte le precedenti r regine già piazzate) se una di esse stà nella stessa colonna j , oppure in una delle due diagonali che passano per la casella della riga r e colonna j . Se una di queste tre condizioni accade, non possiamo mettere la r -esima regina nella colonna j della riga r e quindi assegniamo il valore false alla variabile legal.

```

PlaceQueens(Q[1..n], r)
1.  IF (r = n + 1){
2.      print Q[1..n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)||((Q[i] =
                j+r-i)||((Q[i] = j-r+i)){
8.                  legal=False
                }
            }
        }
9.      IF(legal){
10.         Q[r] = j
11.         PlaceQueens(Q[1..n], r + 1)
        }
    }
}

```

Ovvero, controlliamo (per tutte le precedenti r regine già piazzate) se una di esse stà nella stessa colonna j , oppure in una delle due diagonali che passano per la casella della riga r e colonna j . Se una di queste tre condizioni accade, non possiamo mettere la r -esima regina nella colonna j della riga r e quindi assegniamo il valore `false` alla variabile `legal`.

Se dopo aver eseguito il FOR della linea 6. il valore della variabile `legal` è `false`, *non* possiamo mettere la regina r -esima nella colonna j della riga r ,

```

PlaceQueens(Q[1..n], r)
1.  IF (r = n + 1){
2.      print Q[1..n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)||((Q[i] =
                j+r-i)||((Q[i] = j-r+i)){
8.                  legal=False
                }
            }
        }
9.      IF(legal){
10.         Q[r] = j
11.         PlaceQueens(Q[1..n], r + 1)
        }
    }
}

```

Ovvero, controlliamo (per tutte le precedenti r regine già piazzate) se una di esse stà nella stessa colonna j , oppure in una delle due diagonali che passano per la casella della riga r e colonna j . Se una di queste tre condizioni accade, non possiamo mettere la r -esima regina nella colonna j della riga r e quindi assegniamo il valore `false` alla variabile `legal`.

Se dopo aver eseguito il FOR della linea 6. il valore della variabile `legal` è `false`, *non* possiamo mettere la regina r -esima nella colonna j della riga r , incrementiamo il valore di j nel FOR 4 e ripetiamo il tutto.


```

PlaceQueens(Q[1..n], r)
1.  IF (r = n + 1){
2.      print Q[1..n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)||((Q[i] =
                    j+r-i)||((Q[i] = j-r+i)){
8.                  legal=False
9.              }
10.             }
11.             PlaceQueens(Q[1..n], r + 1)
12.         }
13.     }
14. }

```

Ovvero, controlliamo (per tutte le precedenti r regine già piazzate) se una di esse stà nella stessa colonna j , oppure in una delle due diagonali che passano per la casella della riga r e colonna j . Se una di queste tre condizioni accade, non possiamo mettere la r -esima regina nella colonna j della riga r e quindi assegniamo il valore `false` alla variabile `legal`.

Se dopo aver eseguito il FOR della linea 6. il valore della variabile `legal` è `false`, *non* possiamo mettere la regina r -esima nella colonna j della riga r , incrementiamo il valore di j nel FOR 4 e ripetiamo il tutto. Se arriviamo a $j = n$ senza poter piazzare la regina r -esima, deduciamo che il piazzamento delle precedenti $r - 1$ regine non permette di piazzarne altre, terminiamo la ricorsione, facciamo backtrack (torniamo indietro) cercando un nuovo piazzamento (se c'è) della regina $(r - 1)$ -esima.

```

PlaceQueens(Q[1..n], r)
1.  IF (r = n + 1){
2.      print Q[1..n]
3.  }ELSE{
4.      FOR(j=1, j<n+1, j=j+1){
5.          legal=True
6.          FOR(i = 1, i < r, i = i + 1){
7.              IF((Q[i] = j)||((Q[i] =
                j+r-i)||((Q[i] = j-r+i))){
8.                  legal=False
                }
            }
9.          IF(legal){
10.             Q[r] = j
11.             PlaceQueens(Q[1..n], r + 1)
            }
        }
    }
}

```

Ovvero, controlliamo (per tutte le precedenti r regine già piazzate) se una di esse stà nella stessa colonna j , oppure in una delle due diagonali che passano per la casella della riga r e colonna j . Se una di queste tre condizioni accade, non possiamo mettere la r -esima regina nella colonna j della riga r e quindi assegniamo il valore `false` alla variabile `legal`.

Se dopo aver eseguito il FOR della linea 6. il valore della variabile `legal` è `false`, *non* possiamo mettere la regina r -esima nella colonna j della riga r , incrementiamo il valore di j nel FOR 4 e ripetiamo il tutto. Se arriviamo a $j = n$ senza poter piazzare la regina r -esima, deduciamo che il piazzamento delle precedenti $r - 1$ regine non permette di piazzarne altre, terminiamo la ricorsione, facciamo backtrack (torniamo indietro) cercando un nuovo piazzamento (se c'è) della regina $(r - 1)$ -esima.

Se invece, dopo aver eseguito il FOR della linea 6. il valore della variabile `legal` è `true`, piazziamo la regina r -esima nella colonna j della riga r e continuiamo con la ricorsione alla ricerca di un piazzamento della regina $(r + 1)$ -esima.

Generazione di tutte le soluzioni per 4 regine

Ricordiamo che il vettore $Q[1 \dots n]$ ha $Q[i] = j$, con $j \in \{1, \dots, n\}$, se e solo se abbiamo messo la regina i -esima nella colonna j -esima della riga i -esima della scacchiera.

Ricordiamo che il vettore $Q[1 \dots n]$ ha $Q[i] = j$, con $j \in \{1, \dots, n\}$, se e solo se abbiamo messo la regina i -esima nella colonna j -esima della riga i -esima della scacchiera. Quindi $Q[1 \dots n]$ è una permutazione degli interi $\{1, \dots, n\}$.

Ricordiamo che il vettore $Q[1 \dots n]$ ha $Q[i] = j$, con $j \in \{1, \dots, n\}$, se e solo se abbiamo messo la regina i -esima nella colonna j -esima della riga i -esima della scacchiera. Quindi $Q[1 \dots n]$ è una permutazione degli interi $\{1, \dots, n\}$.

L'algoritmo `PlaceQueens(Q[1 .. n], n)` teoricamente può generare tutte le permutazioni degli elementi $1, 2, \dots, n$, ovvero $n!$ elementi.

Ricordiamo che il vettore $Q[1 \dots n]$ ha $Q[i] = j$, con $j \in \{1, \dots, n\}$, se e solo se abbiamo messo la regina i -esima nella colonna j -esima della riga i -esima della scacchiera. Quindi $Q[1 \dots n]$ è una permutazione degli interi $\{1, \dots, n\}$.

L'algoritmo `PlaceQueens(Q[1 .. n], n)` teoricamente può generare tutte le permutazioni degli elementi $1, 2, \dots, n$, ovvero $n!$ elementi.

Anche se l'algoritmo è esponenziale, è molto migliore dell'algoritmo di forza bruta che genererebbe tutte le n^n possibili posizionamenti delle n matrici nella scacchiera (cioè, n possibili posizioni per ognuna delle n righe).

Ricordiamo che il vettore $Q[1 \dots n]$ ha $Q[i] = j$, con $j \in \{1, \dots, n\}$, se e solo se abbiamo messo la regina i -esima nella colonna j -esima della riga i -esima della scacchiera. Quindi $Q[1 \dots n]$ è una permutazione degli interi $\{1, \dots, n\}$.

L'algoritmo `PlaceQueens(Q[1 .. n], n)` teoricamente può generare tutte le permutazioni degli elementi $1, 2, \dots, n$, ovvero $n!$ elementi.

Anche se l'algoritmo è esponenziale, è molto migliore dell'algoritmo di forza bruta che genererebbe tutte le n^n possibili posizionamenti delle n matrici nella scacchiera (cioè, n possibili posizioni per ognuna delle n righe).

Ad esempio, per $n = 8$ l'algoritmo `PlaceQueens(Q[1 .. n], n)` potrebbe generare al più $8! = 40320$

Ricordiamo che il vettore $Q[1 \dots n]$ ha $Q[i] = j$, con $j \in \{1, \dots, n\}$, se e solo se abbiamo messo la regina i -esima nella colonna j -esima della riga i -esima della scacchiera. Quindi $Q[1 \dots n]$ è una permutazione degli interi $\{1, \dots, n\}$.

L'algoritmo `PlaceQueens(Q[1 .. n], n)` teoricamente può generare tutte le permutazioni degli elementi $1, 2, \dots, n$, ovvero $n!$ elementi.

Anche se l'algoritmo è esponenziale, è molto migliore dell'algoritmo di forza bruta che genererebbe tutte le n^n possibili posizionamenti delle n matrici nella scacchiera (cioè, n possibili posizioni per ognuna delle n righe).

Ad esempio, per $n = 8$ l'algoritmo `PlaceQueens(Q[1 .. n], n)` potrebbe generare al più $8! = 40320$ (ma per trovare la prima soluzione ne esamina 15720).

Ricordiamo che il vettore $Q[1 \dots n]$ ha $Q[i] = j$, con $j \in \{1, \dots, n\}$, se e solo se abbiamo messo la regina i -esima nella colonna j -esima della riga i -esima della scacchiera. Quindi $Q[1 \dots n]$ è una permutazione degli interi $\{1, \dots, n\}$.

L'algoritmo `PlaceQueens(Q[1 .. n], n)` teoricamente può generare tutte le permutazioni degli elementi $1, 2, \dots, n$, ovvero $n!$ elementi.

Anche se l'algoritmo è esponenziale, è molto migliore dell'algoritmo di forza bruta che genererebbe tutte le n^n possibili posizionamenti delle n matrici nella scacchiera (cioè, n possibili posizioni per ognuna delle n righe).

Ad esempio, per $n = 8$ l'algoritmo `PlaceQueens(Q[1 .. n], n)` potrebbe generare al più $8! = 40320$ (ma per trovare la prima soluzione ne esamina 15720). Mentre invece l'algoritmo di forza bruta tenterebbe $8^8 = 16777216$ possibili soluzioni.

Riconsideriamo il problema decisionale dello Zaino

Riconsideriamo il problema decisionale dello Zaino

Dato un insieme X di numeri interi positivi e un numero intero T , vogliamo stabilire se esiste un sottoinsieme di elementi in X che somma esattamente a T

Riconsideriamo il problema decisionale dello Zaino

Dato un insieme X di numeri interi positivi e un numero intero T , vogliamo stabilire se esiste un sottoinsieme di elementi in X che somma esattamente a T

Notiamo che può esistere più di un sottoinsieme di questo tipo. Ad esempio, se $X = \{8, 6, 7, 5, 3, 10, 9\}$ e $T = 15$, la risposta è True, perché i sottoinsiemi $\{8, 7\}$, $\{7, 5, 3\}$, $\{6, 9\}$ e $\{5, 10\}$ hanno tutti somma pari a 15.

Riconsideriamo il problema decisionale dello Zaino

Dato un insieme X di numeri interi positivi e un numero intero T , vogliamo stabilire se esiste un sottoinsieme di elementi in X che somma esattamente a T

Notiamo che può esistere più di un sottoinsieme di questo tipo. Ad esempio, se $X = \{8, 6, 7, 5, 3, 10, 9\}$ e $T = 15$, la risposta è True, perché i sottoinsiemi $\{8, 7\}$, $\{7, 5, 3\}$, $\{6, 9\}$ e $\{5, 10\}$ hanno tutti somma pari a 15.

D'altra parte, se fosse $X = \{11, 6, 5, 1, 7, 13, 12\}$ e $T = 15$, la risposta è False.

Riconsideriamo il problema decisionale dello Zaino

Dato un insieme X di numeri interi positivi e un numero intero T , vogliamo stabilire se esiste un sottoinsieme di elementi in X che somma esattamente a T

Notiamo che può esistere più di un sottoinsieme di questo tipo. Ad esempio, se $X = \{8, 6, 7, 5, 3, 10, 9\}$ e $T = 15$, la risposta è True, perché i sottoinsiemi $\{8, 7\}$, $\{7, 5, 3\}$, $\{6, 9\}$ e $\{5, 10\}$ hanno tutti somma pari a 15.

D'altra parte, se fosse $X = \{11, 6, 5, 1, 7, 13, 12\}$ e $T = 15$, la risposta è False.

Ci sono due casi banali. Se il valore T è zero, allora possiamo restituire immediatamente True, perché l'insieme vuoto è un sottoinsieme di ogni insieme X , e gli elementi dell'insieme vuoto sommano a zero.

Riconsideriamo il problema decisionale dello Zaino

Dato un insieme X di numeri interi positivi e un numero intero T , vogliamo stabilire se esiste un sottoinsieme di elementi in X che somma esattamente a T

Notiamo che può esistere più di un sottoinsieme di questo tipo. Ad esempio, se $X = \{8, 6, 7, 5, 3, 10, 9\}$ e $T = 15$, la risposta è True, perché i sottoinsiemi $\{8, 7\}$, $\{7, 5, 3\}$, $\{6, 9\}$ e $\{5, 10\}$ hanno tutti somma pari a 15.

D'altra parte, se fosse $X = \{11, 6, 5, 1, 7, 13, 12\}$ e $T = 15$, la risposta è False.

Ci sono due casi banali. Se il valore T è zero, allora possiamo restituire immediatamente True, perché l'insieme vuoto è un sottoinsieme di ogni insieme X , e gli elementi dell'insieme vuoto sommano a zero.

D'altra parte, se $T < 0$, o se $T \neq 0$ ma l'insieme X è vuoto, allora possiamo immediatamente restituire False.

Per il caso generale, consideriamo un elemento arbitrario $x \in X$.

Per il caso generale, consideriamo un elemento arbitrario $x \in X$. Esiste un sottoinsieme di X che somma a T se e solo se almeno una delle seguenti affermazioni è vera:

Per il caso generale, consideriamo un elemento arbitrario $x \in X$. Esiste un sottoinsieme di X che somma a T se e solo se almeno una delle seguenti affermazioni è vera:

- ▶ Esiste un sottoinsieme di X che include x e la cui somma è T .

Per il caso generale, consideriamo un elemento arbitrario $x \in X$. Esiste un sottoinsieme di X che somma a T se e solo se almeno una delle seguenti affermazioni è vera:

- ▶ Esiste un sottoinsieme di X che include x e la cui somma è T .
- ▶ Esiste un sottoinsieme di X che esclude x e la cui somma è T .

Per il caso generale, consideriamo un elemento arbitrario $x \in X$. Esiste un sottoinsieme di X che somma a T se e solo se almeno una delle seguenti affermazioni è vera:

- ▶ Esiste un sottoinsieme di X che include x e la cui somma è T .
- ▶ Esiste un sottoinsieme di X che esclude x e la cui somma è T .

Nel primo caso, deve anche esistere un sottoinsieme di $X \setminus \{x\}$ che somma a $T - x$;

Per il caso generale, consideriamo un elemento arbitrario $x \in X$. Esiste un sottoinsieme di X che somma a T se e solo se almeno una delle seguenti affermazioni è vera:

- ▶ Esiste un sottoinsieme di X che include x e la cui somma è T .
- ▶ Esiste un sottoinsieme di X che esclude x e la cui somma è T .

Nel primo caso, deve anche esistere un sottoinsieme di $X \setminus \{x\}$ che somma a $T - x$; nel secondo caso, ci deve essere un sottoinsieme di $X \setminus \{x\}$ che somma a T .

Per il caso generale, consideriamo un elemento arbitrario $x \in X$. Esiste un sottoinsieme di X che somma a T se e solo se almeno una delle seguenti affermazioni è vera:

- ▶ Esiste un sottoinsieme di X che include x e la cui somma è T .
- ▶ Esiste un sottoinsieme di X che esclude x e la cui somma è T .

Nel primo caso, deve anche esistere un sottoinsieme di $X \setminus \{x\}$ che somma a $T - x$; nel secondo caso, ci deve essere un sottoinsieme di $X \setminus \{x\}$ che somma a T .

In questo modo possiamo risolvere $\text{Zaino}(X, T)$ riducendolo a due casi più semplici: $\text{Zaino}(X \setminus \{x\}, T - x)$

Per il caso generale, consideriamo un elemento arbitrario $x \in X$. Esiste un sottoinsieme di X che somma a T se e solo se almeno una delle seguenti affermazioni è vera:

- ▶ Esiste un sottoinsieme di X che include x e la cui somma è T .
- ▶ Esiste un sottoinsieme di X che esclude x e la cui somma è T .

Nel primo caso, deve anche esistere un sottoinsieme di $X \setminus \{x\}$ che somma a $T - x$; nel secondo caso, ci deve essere un sottoinsieme di $X \setminus \{x\}$ che somma a T .

In questo modo possiamo risolvere $\text{Zaino}(X, T)$ riducendolo a due casi più semplici: $\text{Zaino}(X \setminus \{x\}, T - x)$ e $\text{Zaino}(X \setminus \{x\}, T)$.


```
Zaino( $X, T$ )  
  IF( $T==0$ ) {  
    return True
```

```
Zaino( $X, T$ )  
  IF( $T=0$ ){  
    return True  
  }ELSE{  
    IF( $(T < 0) || (X = \emptyset)$ ){
```

```
Zaino( $X$ ,  $T$ )  
  IF( $T=0$ ){  
    return True  
  }ELSE{  
    IF( $(T < 0) || (X = \emptyset)$ ){  
      return False
```

```
Zaino( $X, T$ )
  IF( $T==0$ ){
    return True
  }ELSE{
    IF( $(T < 0) || (X = \emptyset)$ ){
      return False
    }ELSE{
       $x =$  ultimo elemento di  $X$ 
```

```
Zaino( $X$ ,  $T$ )
  IF( $T=0$ ) {
    return True
  } ELSE {
    IF(( $T < 0$ ) || ( $X = \emptyset$ )) {
      return False
    } ELSE {
       $x$  = ultimo elemento di  $X$ 
      con=Zaino( $X \setminus \{x\}$ ,  $T - x$ )
    }
  }
}
```

```
Zaino( $X$ ,  $T$ )
  IF( $T=0$ ){
    return True
  }ELSE{
    IF(( $T < 0$ )||( $X = \emptyset$ )){
      return False
    }ELSE{
       $x$  = ultimo elemento di  $X$ 
      con=Zaino( $X \setminus \{x\}$ ,  $T - x$ )
      senza=Zaino( $X \setminus \{x\}$ ,  $T$ )
```

```
Zaino( $X$ ,  $T$ )
  IF( $T=0$ ) {
    return True
  } ELSE {
    IF(( $T < 0$ ) || ( $X = \emptyset$ )) {
      return False
    } ELSE {
       $x$  = ultimo elemento di  $X$ 
      con = Zaino( $X \setminus \{x\}$ ,  $T - x$ )
      senza = Zaino( $X \setminus \{x\}$ ,  $T$ )
      return (con  $\vee$  senza)
    }
  }
```

```

Zaino( $X$ ,  $T$ )
  IF( $T=0$ ) {
    return True
  } ELSE {
    IF( $(T < 0) || (X = \emptyset)$ ) {
      return False
    } ELSE {
       $x$  = ultimo elemento di  $X$ 
      con = Zaino( $X \setminus \{x\}$ ,  $T - x$ )
      senza = Zaino( $X \setminus \{x\}$ ,  $T$ )
      return (con  $\vee$  senza)
    }
  }

```

La prova che l'algoritmo è corretto è un semplice esercizio di induzione. Se $T = 0$, gli elementi del sottoinsieme vuoto sommano a T , quindi True è il corretto output.


```

Zaino( $X, T$ )
  IF( $T=0$ ){
    return True
  }ELSE{
    IF( $(T < 0) || (X = \emptyset)$ ){
      return False
    }ELSE{
       $x =$  ultimo elemento di  $X$ 
       $con =$  Zaino( $X \setminus \{x\}, T - x$ )
       $senza =$  Zaino( $X \setminus \{x\}, T$ )
      return ( $con \vee senza$ )
    }
  }

```

La prova che l'algoritmo è corretto è un semplice esercizio di induzione. Se $T = 0$, gli elementi del sottoinsieme vuoto sommano a T , quindi True è il corretto output.

Altrimenti, se T è negativo o l'insieme X è vuoto, allora nessun sottoinsieme di X somma a T , quindi False è l'output corretto.

```

Zaino( $X, T$ )
  IF( $T==0$ ){
    return True
  }ELSE{
    IF( $(T < 0) || (X = \emptyset)$ ){
      return False
    }ELSE{
       $x =$  ultimo elemento di  $X$ 
      con= $Zaino(X \setminus \{x\}, T - x)$ 
      senza= $Zaino(X \setminus \{x\}, T)$ 
      return (con $\vee$ senza)
    }
  }

```

La prova che l'algoritmo è corretto è un semplice esercizio di induzione. Se $T = 0$, gli elementi del sottoinsieme vuoto sommano a T , quindi True è il corretto output.

Altrimenti, se T è negativo o l'insieme X è vuoto, allora nessun sottoinsieme di X somma a T , quindi False è l'output corretto. Altrimenti, se c'è un sottoinsieme che somma a T , esso o contiene x o non contiene x , e le due ricorsioni controllano correttamente ciascuna di queste possibilità.

```

Zaino(X, T)
  IF(T==0){
    return True
  }ELSE{
    IF((T < 0)|| (X = ∅)){
      return False
    }ELSE{
      x = ultimo elemento di X
      con=Zaino(X \ {x}, T - x)
      senza=Zaino(X \ {x}, T)
      return (con∨senza)
    }
  }

```

L'algoritmo può essere implementato in modo tale che la sua complessità di tempo $T(n)$ soddisfi la ricorrenza $T(n) = 2T(n-1) + O(1)$, che ha soluzione $T(n) = \Theta(2^n)$.

```

Zaino(X, T)
  IF(T==0){
    return True
  }ELSE{
    IF((T < 0)|| (X = ∅)){
      return False
    }ELSE{
      x = ultimo elemento di X
      con=Zaino(X \ {x}, T - x)
      senza=Zaino(X \ {x}, T)
      return (con∨senza)
    }
  }

```

L'algoritmo può essere implementato in modo tale che la sua complessità di tempo $T(n)$ soddisfi la ricorrenza $T(n) = 2T(n-1) + O(1)$, che ha soluzione $T(n) = \Theta(2^n)$.

Forse ricorderete che abbiamo risolto lo stesso problema con PD ed il relativo algoritmo aveva complessità $\Theta(nT)$.

```

Zaino(X, T)
  IF(T==0){
    return True
  }ELSE{
    IF((T < 0)|| (X = ∅)){
      return False
    }ELSE{
      x = ultimo elemento di X
      con=Zaino(X \ {x}, T - x)
      senza=Zaino(X \ {x}, T)
      return (con∨senza)
    }
  }

```

L' algoritmo può essere implementato in modo tale che la sua complessità di tempo $T(n)$ soddisfi la ricorrenza $T(n) = 2T(n - 1) + O(1)$, che ha soluzione $T(n) = \Theta(2^n)$.

Forse ricorderete che abbiamo risolto lo stesso problema con PD ed il relativo algoritmo aveva complessità $\Theta(nT)$.

Chi dei due è migliore? Dipende da quanto è grande T

Usando lo stesso algoritmo con piccole modifiche, possiamo risolvere anche il problema di costruire un sottoinsieme X la cui somma degli elementi è pari a T , se esso esiste. Assumiamo che gli n elementi di X siano memorizzati in un vettore $X[1 \dots n]$.

Usando lo stesso algoritmo con piccole modifiche, possiamo risolvere anche il problema di costruire un sottoinsieme X la cui somma degli elementi è pari a T , se esso esiste. Assumiamo che gli n elementi di X siano memorizzati in un vettore $X[1 \dots n]$.

```
ConstructSubset( $X, i, T$ )  
  IF ( $T == 0$ )  
    return  $\emptyset$ 
```

Usando lo stesso algoritmo con piccole modifiche, possiamo risolvere anche il problema di costruire un sottoinsieme X la cui somma degli elementi è pari a T , se esso esiste. Assumiamo che gli n elementi di X siano memorizzati in un vettore $X[1 \dots n]$.

```
ConstructSubset( $X, i, T$ )  
  IF ( $T == 0$ )  
    return  $\emptyset$   
  IF ( $(T < 0) || (n == 0)$ )  
    return ‘‘non c’è’’
```


Usando lo stesso algoritmo con piccole modifiche, possiamo risolvere anche il problema di costruire un sottoinsieme X la cui somma degli elementi è pari a T , se esso esiste. Assumiamo che gli n elementi di X siano memorizzati in un vettore $X[1 \dots n]$.

```
ConstructSubset( $X, i, T$ )  
  IF ( $T == 0$ )  
    return  $\emptyset$   
  IF ( $(T < 0) || (n == 0)$ )  
    return 'non c'è'  
   $Y = \text{ConstructSubset}(X, i - 1, T)$ 
```

Usando lo stesso algoritmo con piccole modifiche, possiamo risolvere anche il problema di costruire un sottoinsieme X la cui somma degli elementi è pari a T , se esso esiste. Assumiamo che gli n elementi di X siano memorizzati in un vettore $X[1 \dots n]$.

```
ConstructSubset( $X, i, T$ )
  IF ( $T == 0$ )
    return  $\emptyset$ 
  IF ( $(T < 0) || (n == 0)$ )
    return ‘non c’è’
   $Y = \text{ConstructSubset}(X, i - 1, T)$ 
  IF  $Y \neq \text{‘non c’è’}$ 
    return  $Y$ 
```

Usando lo stesso algoritmo con piccole modifiche, possiamo risolvere anche il problema di costruire un sottoinsieme X la cui somma degli elementi è pari a T , se esso esiste. Assumiamo che gli n elementi di X siano memorizzati in un vettore $X[1 \dots n]$.

```
ConstructSubset( $X, i, T$ )
  IF ( $T == 0$ )
    return  $\emptyset$ 
  IF ( $(T < 0) || (n == 0)$ )
    return "non c'è"
   $Y = \text{ConstructSubset}(X, i - 1, T)$ 
  IF  $Y \neq \text{"non c'è"}$ 
    return  $Y$ 
   $Y = \text{ConstructSubset}(X, i - 1, T - X[i])$ 
  IF  $Y \neq \text{"non c'è"}$ 
```

Usando lo stesso algoritmo con piccole modifiche, possiamo risolvere anche il problema di costruire un sottoinsieme X la cui somma degli elementi è pari a T , se esso esiste. Assumiamo che gli n elementi di X siano memorizzati in un vettore $X[1 \dots n]$.

```
ConstructSubset( $X, i, T$ )
  IF ( $T == 0$ )
    return  $\emptyset$ 
  IF ( $(T < 0) || (n == 0)$ )
    return "non c'è"
   $Y = \text{ConstructSubset}(X, i - 1, T)$ 
  IF  $Y \neq \text{"non c'è"}$ 
    return  $Y$ 
   $Y = \text{ConstructSubset}(X, i - 1, T - X[i])$ 
  IF  $Y \neq \text{"non c'è"}$ 
    return  $Y \cup \{X[i]\}$ 
```

Usando lo stesso algoritmo con piccole modifiche, possiamo risolvere anche il problema di costruire un sottoinsieme X la cui somma degli elementi è pari a T , se esso esiste. Assumiamo che gli n elementi di X siano memorizzati in un vettore $X[1 \dots n]$.

```
ConstructSubset( $X, i, T$ )
  IF ( $T == 0$ )
    return  $\emptyset$ 
  IF ( $(T < 0) || (n == 0)$ )
    return "non c'è"
   $Y = \text{ConstructSubset}(X, i - 1, T)$ 
  IF  $Y \neq \text{"non c'è"}$ 
    return  $Y$ 
   $Y = \text{ConstructSubset}(X, i - 1, T - X[i])$ 
  IF  $Y \neq \text{"non c'è"}$ 
    return  $Y \cup \{X[i]\}$ 
  return "non c'è"
```

Generazione di tutte le clique di un grafo

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

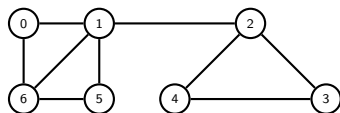
Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



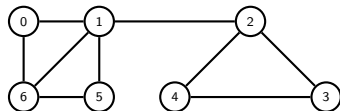
Clique nel grafo sono: $\{\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



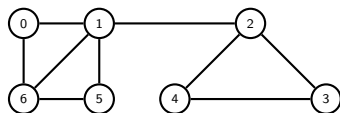
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



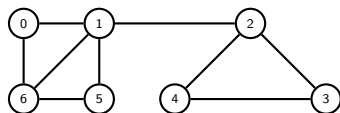
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



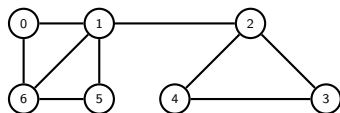
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



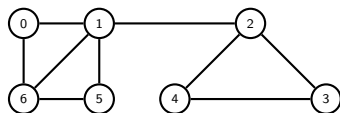
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



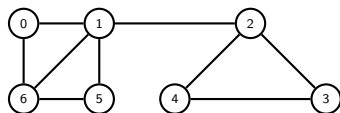
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



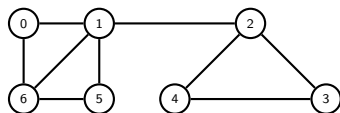
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



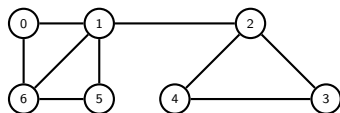
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$, $\{3, 4\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



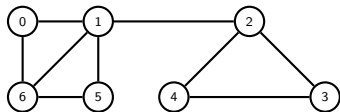
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$, $\{3, 4\}$, $\{0, 1, 6\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



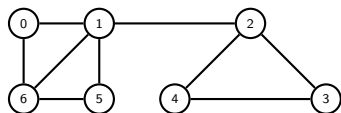
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$, $\{3, 4\}$, $\{0, 1, 6\}$, $\{1, 5, 6\}$

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



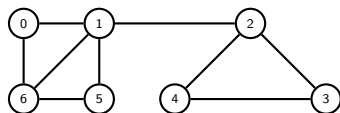
Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$, $\{3, 4\}$, $\{0, 1, 6\}$, $\{1, 5, 6\}$ e $\{2, 3, 4\}$.

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$, $\{3, 4\}$, $\{0, 1, 6\}$, $\{1, 5, 6\}$ e $\{2, 3, 4\}$.

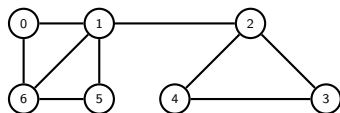
Le clique massimali sono: $\{0, 1, 6\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$, $\{3, 4\}$, $\{0, 1, 6\}$, $\{1, 5, 6\}$ e $\{2, 3, 4\}$.

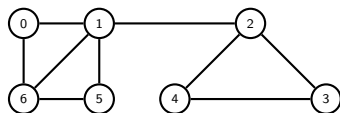
Le clique massimali sono: $\{0, 1, 6\}$, $\{1, 2\}$,

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$, $\{3, 4\}$, $\{0, 1, 6\}$, $\{1, 5, 6\}$ e $\{2, 3, 4\}$.

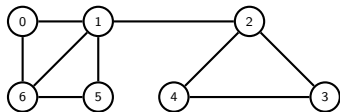
Le clique massimali sono: $\{0, 1, 6\}$, $\{1, 2\}$, $\{1, 5, 6\}$

Generazione di tutte le clique di un grafo

Dato un grafo $G = (V, E)$, un sottoinsieme $X \subseteq V$ è detto una “clique” se e solo se

$$\forall x, y \in X \text{ vale che } (x, y) \in E.$$

Una clique X è detta massimale se $\forall x \in V \setminus X$ vale che $X \cup \{x\}$ non è una clique.



Clique nel grafo sono: $\{\}$, $\{0, 1\}$, $\{0, 6\}$, $\{1, 2\}$, $\{1, 5\}$, $\{1, 6\}$, $\{2, 4\}$, $\{2, 3\}$, $\{3, 4\}$, $\{0, 1, 6\}$, $\{1, 5, 6\}$ e $\{2, 3, 4\}$.

Le clique massimali sono: $\{0, 1, 6\}$, $\{1, 2\}$, $\{1, 5, 6\}$ e $\{2, 3, 4\}$.

Per generare tutte le clique di un grafo $G = (V, E)$ mediante backtracking occorre definire cosa è una soluzione parziale e dare un metodo per estenderla.

Per generare tutte le clique di un grafo $G = (V, E)$ mediante backtracking occorre definire cosa è una soluzione parziale e dare un metodo per estenderla.

La prima parte è semplice: un vettore $X = [x_0, x_1, \dots, x_{\ell-1}]$ composto di vertici di G è una soluzione parziale se e solo se $\{x_0, x_1, \dots, x_{\ell-1}\}$ è una clique.

Per generare tutte le clique di un grafo $G = (V, E)$ mediante backtracking occorre definire cosa è una soluzione parziale e dare un metodo per estenderla.

La prima parte è semplice: un vettore $X = [x_0, x_1, \dots, x_{\ell-1}]$ composto di vertici di G è una soluzione parziale se e solo se $\{x_0, x_1, \dots, x_{\ell-1}\}$ è una clique.

Per capire come estendere una generica soluzione parziale $X = [x_0, x_1, \dots, x_{\ell-1}]$, denotiamo con $S_{\ell-1} = \{x_0, x_1, \dots, x_{\ell-1}\}$,

Per generare tutte le clique di un grafo $G = (V, E)$ mediante backtracking occorre definire cosa è una soluzione parziale e dare un metodo per estenderla.

La prima parte è semplice: un vettore $X = [x_0, x_1, \dots, x_{\ell-1}]$ composto di vertici di G è una soluzione parziale se e solo se $\{x_0, x_1, \dots, x_{\ell-1}\}$ è una clique.

Per capire come estendere una generica soluzione parziale $X = [x_0, x_1, \dots, x_{\ell-1}]$, denotiamo con $S_{\ell-1} = \{x_0, x_1, \dots, x_{\ell-1}\}$, e con

$$C_0 = V,$$

Per generare tutte le clique di un grafo $G = (V, E)$ mediante backtracking occorre definire cosa è una soluzione parziale e dare un metodo per estenderla.

La prima parte è semplice: un vettore $X = [x_0, x_1, \dots, x_{\ell-1}]$ composto di vertici di G è una soluzione parziale se e solo se $\{x_0, x_1, \dots, x_{\ell-1}\}$ è una clique.

Per capire come estendere una generica soluzione parziale

$X = [x_0, x_1, \dots, x_{\ell-1}]$, denotiamo con $S_{\ell-1} = \{x_0, x_1, \dots, x_{\ell-1}\}$, e con

$$C_0 = V, \text{ e } C_\ell = \{v \in V \setminus S_{\ell-1} : (v, x) \in E, \forall x \in S_{\ell-1}\}.$$

Per generare tutte le clique di un grafo $G = (V, E)$ mediante backtraking occorre definire cosa è una soluzione parziale e dare un metodo per estenderla.

La prima parte è semplice: un vettore $X = [x_0, x_1, \dots, x_{\ell-1}]$ composto di vertici di G è una soluzione parziale se e solo se $\{x_0, x_1, \dots, x_{\ell-1}\}$ è una clique.

Per capire come estendere una generica soluzione parziale

$X = [x_0, x_1, \dots, x_{\ell-1}]$, denotiamo con $S_{\ell-1} = \{x_0, x_1, \dots, x_{\ell-1}\}$, e con

$$C_0 = V, \text{ e } C_\ell = \{v \in V \setminus S_{\ell-1} : (v, x) \in E, \forall x \in S_{\ell-1}\}.$$

É chiaro che $\forall v \in C_\ell$ la soluzione parziale $X = [x_0, x_1, \dots, x_{\ell-1}]$ può essere estesa ad una nuova soluzione $[x_0, x_1, \dots, x_{\ell-1}, v]$.

Per generare tutte le clique di un grafo $G = (V, E)$ mediante backtraking occorre definire cosa è una soluzione parziale e dare un metodo per estenderla.

La prima parte è semplice: un vettore $X = [x_0, x_1, \dots, x_{\ell-1}]$ composto di vertici di G è una soluzione parziale se e solo se $\{x_0, x_1, \dots, x_{\ell-1}\}$ è una clique.

Per capire come estendere una generica soluzione parziale

$X = [x_0, x_1, \dots, x_{\ell-1}]$, denotiamo con $S_{\ell-1} = \{x_0, x_1, \dots, x_{\ell-1}\}$, e con

$$C_0 = V, \text{ e } C_\ell = \{v \in V \setminus S_{\ell-1} : (v, x) \in E, \forall x \in S_{\ell-1}\}.$$

É chiaro che $\forall v \in C_\ell$ la soluzione parziale $X = [x_0, x_1, \dots, x_{\ell-1}]$ può essere estesa ad una nuova soluzione $[x_0, x_1, \dots, x_{\ell-1}, v]$. Notiamo che

$$C_\ell = \{v \in C_{\ell-1} \setminus \{x_{\ell-1}\} : (v, x_{\ell-1}) \in E\}.$$

Questo modo di procedere (ovvero estendere soluzioni parziali

$X = [x_0, x_1, \dots, x_{\ell-1}]$, con arbitrari elementi di

$C_\ell = \{v \in C_{\ell-1} \setminus \{x_{\ell-1}\} : (v, x_{\ell-1}) \in E\}$ porta a generare più volte la stessa clique.

Questo modo di procedere (ovvero estendere soluzioni parziali

$X = [x_0, x_1, \dots, x_{\ell-1}]$, con arbitrari elementi di

$C_\ell = \{v \in C_{\ell-1} \setminus \{x_{\ell-1}\} : (v, x_{\ell-1}) \in E\}$ porta a generare più volte la stessa clique. Ad esempio, la clique $\{0, 1, 6\}$ nel grafo esempio verrebbe generata come $\{0, 1, 6\}$, $\{1, 0, 6\}$, $\{1, 6, 0\}$, $\{0, 6, 1\}$, $\{6, 1, 0\}$, $\{6, 0, 1\}$.

Questo modo di procedere (ovvero estendere soluzioni parziali $X = [x_0, x_1, \dots, x_{\ell-1}]$, con arbitrari elementi di $C_\ell = \{v \in C_{\ell-1} \setminus \{x_{\ell-1}\} : (v, x_{\ell-1}) \in E\}$ porta a generare più volte la stessa clique. Ad esempio, la clique $\{0, 1, 6\}$ nel grafo esempio verrebbe generata come $\{0, 1, 6\}$, $\{1, 0, 6\}$, $\{1, 6, 0\}$, $\{0, 6, 1\}$, $\{6, 1, 0\}$, $\{6, 0, 1\}$. Per evitare questo problema, supponiamo di aver ordinato i vertici in V in maniera tale che $v_0 < v_1 < \dots < v_{n-1}$, e li considereremo in quest'ordine.

Questo modo di procedere (ovvero estendere soluzioni parziali $X = [x_0, x_1, \dots, x_{\ell-1}]$, con arbitrari elementi di $C_\ell = \{v \in C_{\ell-1} \setminus \{x_{\ell-1}\} : (v, x_{\ell-1}) \in E\}$ porta a generare più volte la stessa clique. Ad esempio, la clique $\{0, 1, 6\}$ nel grafo esempio verrebbe generata come $\{0, 1, 6\}$, $\{1, 0, 6\}$, $\{1, 6, 0\}$, $\{0, 6, 1\}$, $\{6, 1, 0\}$, $\{6, 0, 1\}$. Per evitare questo problema, supponiamo di aver ordinato i vertici in V in maniera tale che $v_0 < v_1 < \dots < v_{n-1}$, e li considereremo in quest'ordine. Quindi, ridefiniamo C_ℓ come

$$C_\ell = \{v \in C_{\ell-1} : (v, x_{\ell-1}) \in E \text{ e } v > x_{\ell-1}\}.$$

Questo modo di procedere (ovvero estendere soluzioni parziali $X = [x_0, x_1, \dots, x_{\ell-1}]$, con arbitrari elementi di $C_\ell = \{v \in C_{\ell-1} \setminus \{x_{\ell-1}\} : (v, x_{\ell-1}) \in E\}$ porta a generare più volte la stessa clique. Ad esempio, la clique $\{0, 1, 6\}$ nel grafo esempio verrebbe generata come $\{0, 1, 6\}$, $\{1, 0, 6\}$, $\{1, 6, 0\}$, $\{0, 6, 1\}$, $\{6, 1, 0\}$, $\{6, 0, 1\}$. Per evitare questo problema, supponiamo di aver ordinato i vertici in V in maniera tale che $v_0 < v_1 < \dots < v_{n-1}$, e li considereremo in quest'ordine. Quindi, ridefiniamo C_ℓ come

$$C_\ell = \{v \in C_{\ell-1} : (v, x_{\ell-1}) \in E \text{ e } v > x_{\ell-1}\}.$$

Per calcolare in modo efficiente l'elemento da scegliere al passo ℓ in C_ℓ con cui estendere la soluzione parziale $X = [x_0, x_1, \dots, x_{\ell-1}]$, introduciamo due nuovi insiemi.

Gli insiemi sono:

$$A_v = \{u \in V : (u, v) \in E\}, \text{ e } B_v = \{u \in V : u > v\}.$$

Gli insiemi sono:

$$A_v = \{u \in V : (u, v) \in E\}, \text{ e } B_v = \{u \in V : u > v\}.$$

Questi insiemi possono ovviamente essere precomputati prima che il backtracking abbia inizio.

Gli insiemi sono:

$$A_v = \{u \in V : (u, v) \in E\}, \text{ e } B_v = \{u \in V : u > v\}.$$

Questi insiemi possono ovviamente essere precomputati prima che il backtracking abbia inizio.

Dalla definizione, si ha:

$$C_\ell = A_{x_{\ell-1}} \cap B_{x_{\ell-1}} \cap C_{\ell-1}.$$

Gli insiemi sono:

$$A_v = \{u \in V : (u, v) \in E\}, \text{ e } B_v = \{u \in V : u > v\}.$$

Questi insiemi possono ovviamente essere precomputati prima che il backtracking abbia inizio.

Dalla definizione, si ha:

$$C_\ell = A_{x_{\ell-1}} \cap B_{x_{\ell-1}} \cap C_{\ell-1}.$$

Definiamo anche

$$N_\ell = N_{\ell-1} \cap A_{x_{\ell-1}}, \text{ con } N_0 = V.$$

Gli insiemi sono:

$$A_v = \{u \in V : (u, v) \in E\}, \text{ e } B_v = \{u \in V : u > v\}.$$

Questi insiemi possono ovviamente essere precomputati prima che il backtracking abbia inizio.

Dalla definizione, si ha:

$$C_\ell = A_{x_{\ell-1}} \cap B_{x_{\ell-1}} \cap C_{\ell-1}.$$

Definiamo anche

$$N_\ell = N_{\ell-1} \cap A_{x_{\ell-1}}, \text{ con } N_0 = V.$$

Poichè $X = [x_0, x_1, \dots, x_{\ell-1}]$ è una clique massimale se e solo se essa non può essere estesa con nessun elemento, ne segue che X è una clique massimale se e solo se $N_\ell = \emptyset$.

Gli insiemi sono:

$$A_v = \{u \in V : (u, v) \in E\}, \text{ e } B_v = \{u \in V : u > v\}.$$

Questi insiemi possono ovviamente essere precomputati prima che il backtracking abbia inizio.

Dalla definizione, si ha:

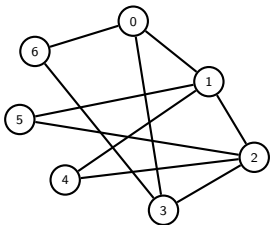
$$C_\ell = A_{x_{\ell-1}} \cap B_{x_{\ell-1}} \cap C_{\ell-1}.$$

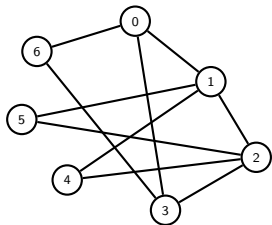
Definiamo anche

$$N_\ell = N_{\ell-1} \cap A_{x_{\ell-1}}, \text{ con } N_0 = V.$$

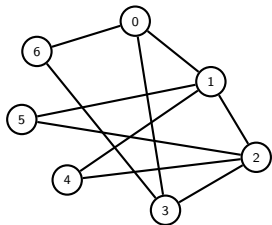
Poichè $X = [x_0, x_1, \dots, x_{\ell-1}]$ è una clique massimale se e solo se essa non può essere estesa con nessun elemento, ne segue che X è una clique massimale se e solo se $N_\ell = \emptyset$.

Sulla base di ciò che abbiamo definito, vediamo come opera l'algoritmo che estende soluzioni parziali $X = [x_0, x_1, \dots, x_{\ell-1}]$ prendendo i successivi elementi da C_ℓ .



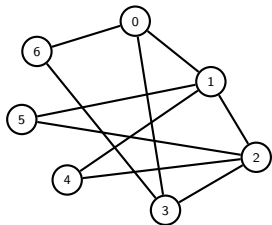


v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



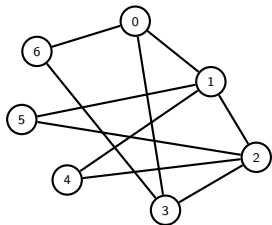
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset

○

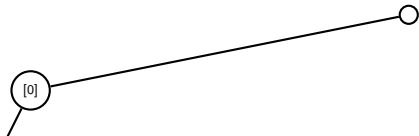


v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset

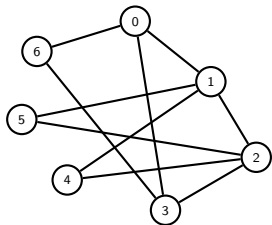




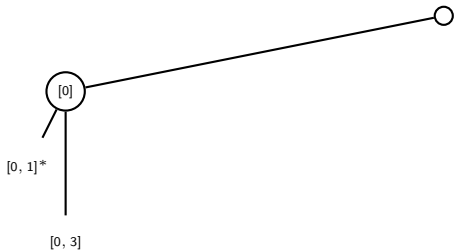
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset

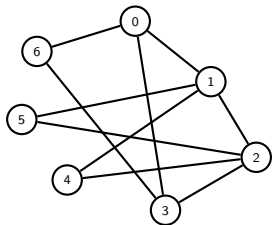


$[0, 1]^*$

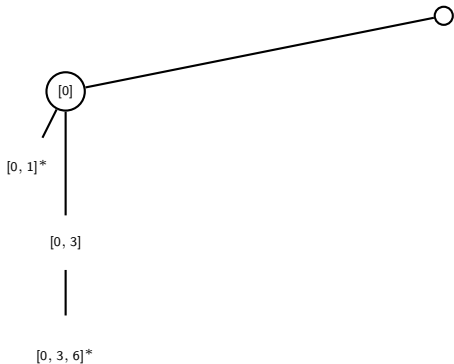


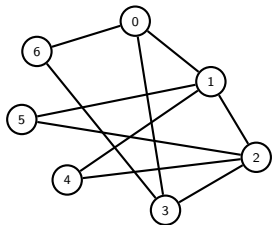
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



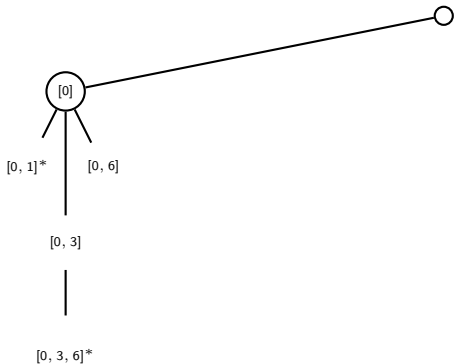


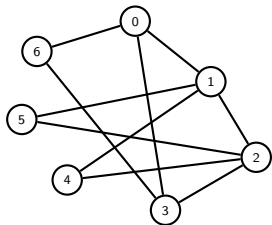
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



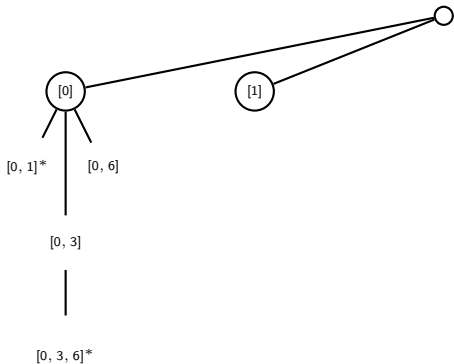


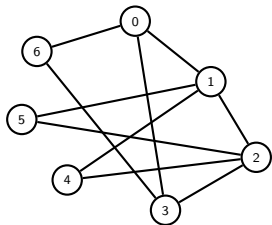
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



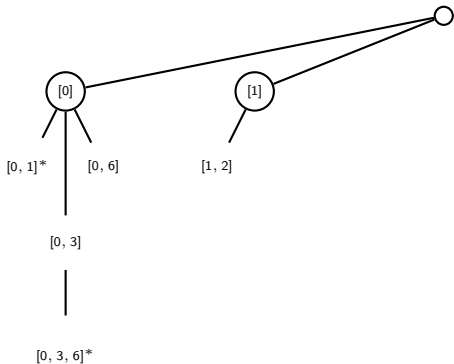


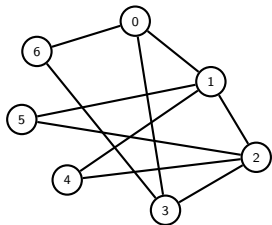
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



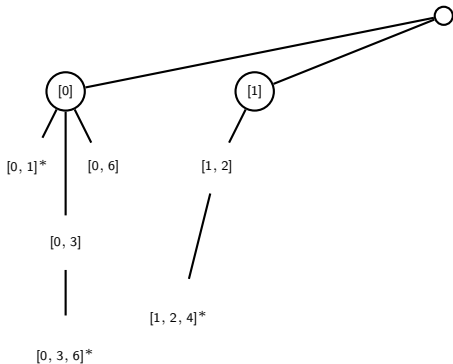


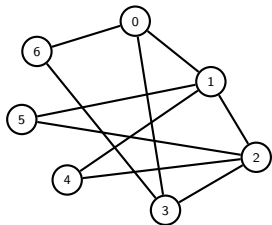
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



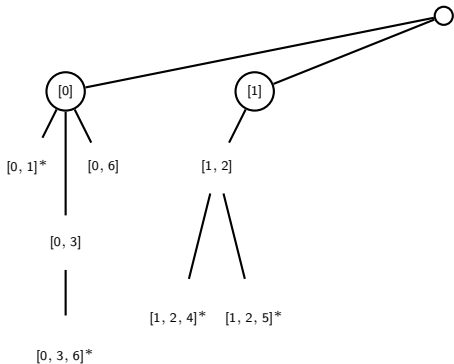


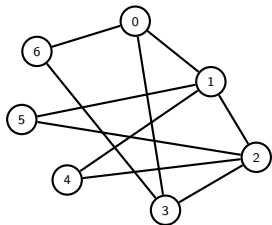
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



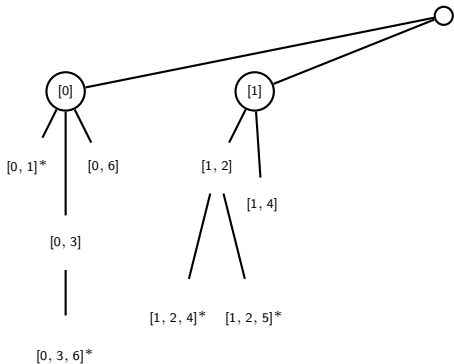


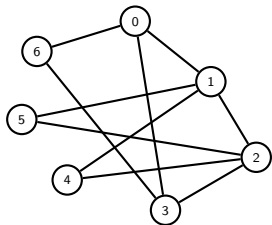
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



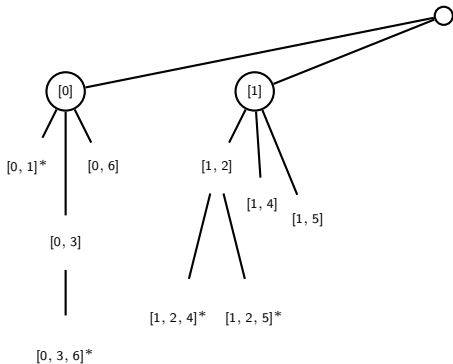


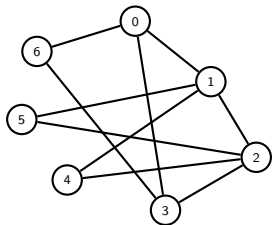
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



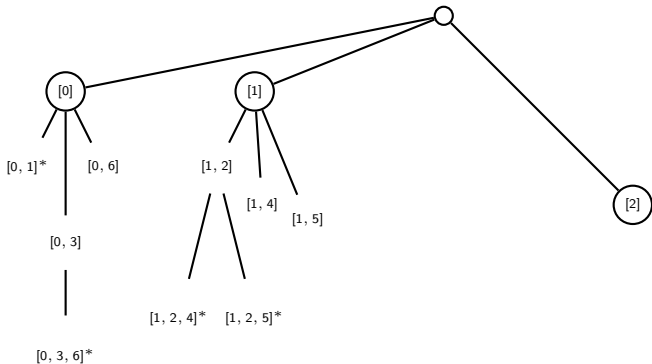


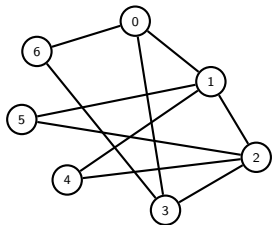
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



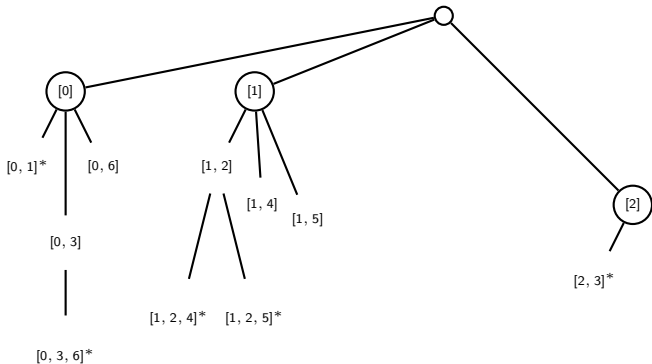


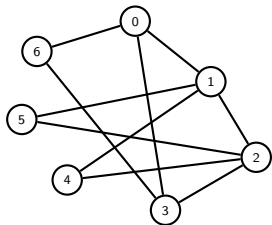
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



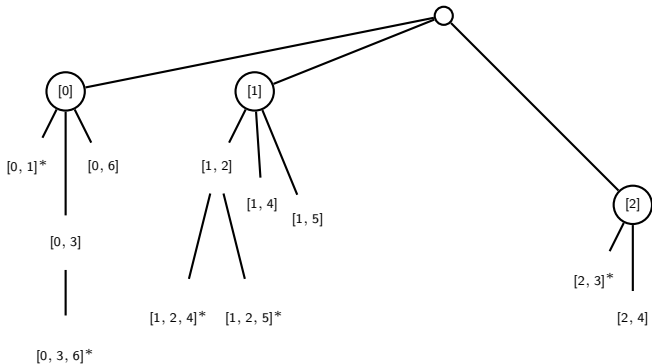


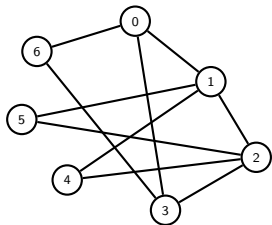
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



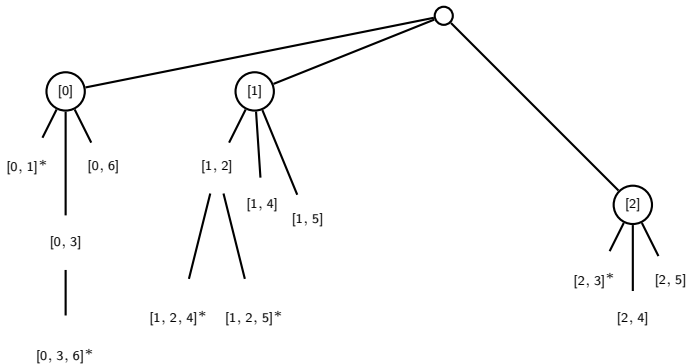


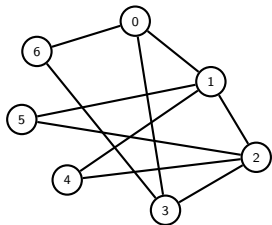
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



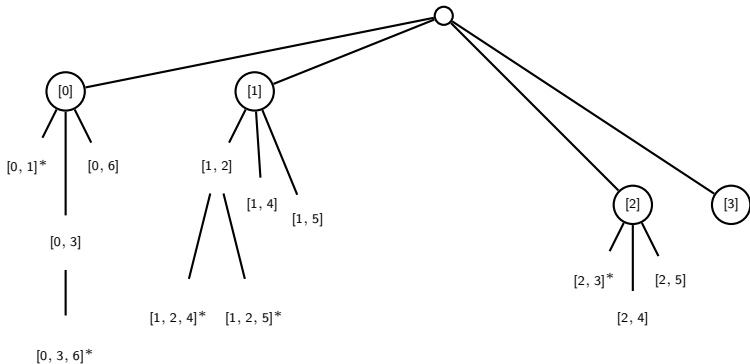


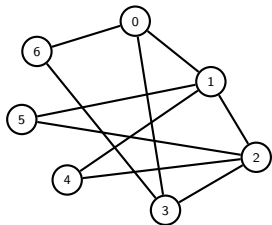
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



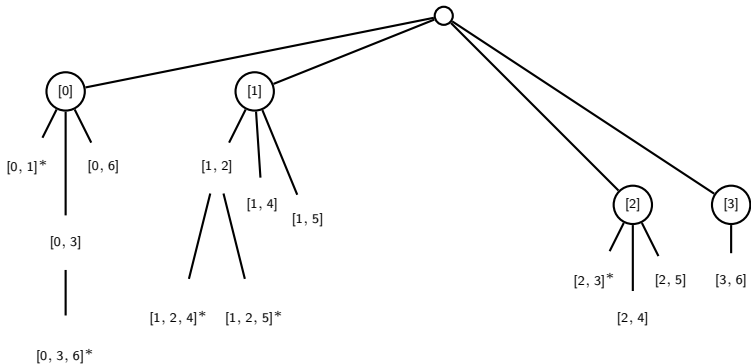


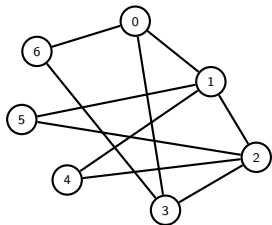
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



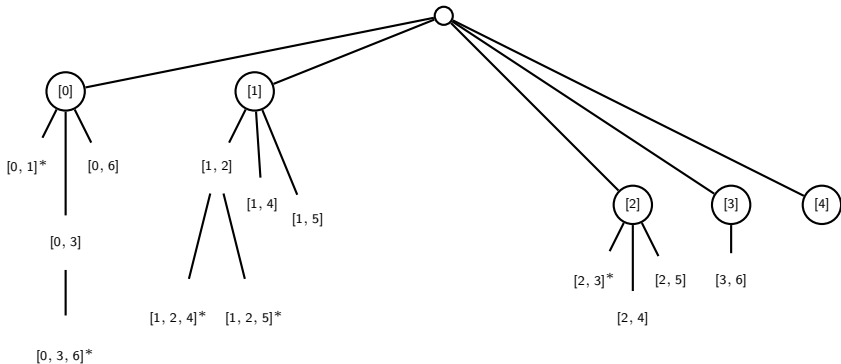


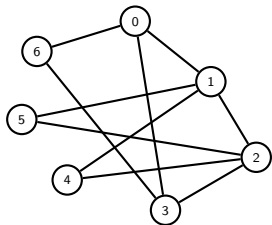
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



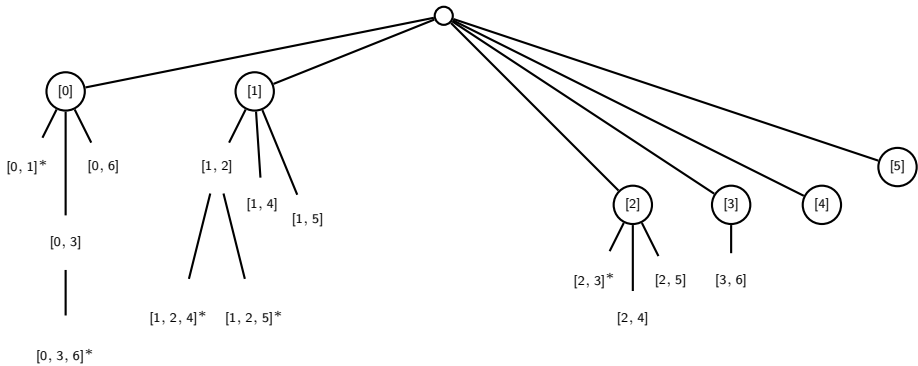


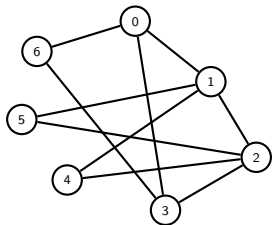
v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



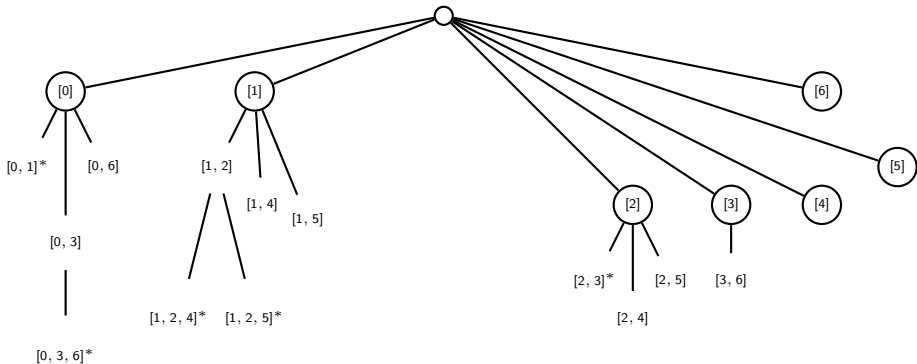


v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset





v	A_v	B_v	C
0	1,3,6	1,2,3,4,5,6	1,3,6
1	0,2,4,5	2,3,4,5,6	2,4,5
2	1,3,4,5	3,4,5,6	3,4,5
3	0,2,6	4,5,6	6
4	1,2	5,6	\emptyset
5	1,2	6	\emptyset
6	0,3	\emptyset	\emptyset



```

AllCliques( $\ell$ ) % Ogni  $X$  generato dall'algoritmo è una
                    clique
IF( $\ell == 0$ ){
    return []
}ELSE{return( $[x_0, \dots, x_{\ell-1}]$ )
}
IF( $\ell == 0$ ){
     $N_\ell = V$ 
}ELSE{ $N_\ell = A_{x_{\ell-1}} \cap N_{\ell-1}$ 
} IF( $N_\ell = \emptyset$ ){
     $\{x_0, \dots, x_{\ell-1}\}$  è massimale
}
IF( $\ell == 0$ ){
     $C_\ell = V$ 
}ELSE{ $C_\ell = A_{x_{\ell-1}} \cap B_{x_{\ell-1}} \cap C_{\ell-1}$ 
}
FOR( $x \in C_\ell$ ){
     $x_\ell = x$ 
    AllCliques( $\ell$ )
}

```