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, ... a_k) è una soluzione) {stampala
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
   } ELSE {
     calcola S_{k+1}
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
    } ELSE {
      calcola S_{k+1}
      WHILE S_{k+1} ≠ ∅ {
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
   } ELSE {
     calcola S_{k+1}
     WHILE S_{k+1} ≠ ∅ {
       a_{k+1} ← un elemento di S_{k+1},
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
   } ELSE {
     calcola S_{k+1}
     WHILE S_{k+1} ≠ ∅ {
       a_{k+1} ← un elemento di S_{k+1}, forma (a_1, ... a_k, a_{k+1})
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
   } ELSE {
     calcola S_{k+1}
     WHILE S_{k+1} ≠ ∅ {
       a_{k+1} ← un elemento di S_{k+1}, forma (a_1, ... a_k, a_{k+1})
       S_{k+1} ← S_{k+1} − {a_{k+1}}
```

Ricordiamo lo schema generale del Backtracking

```
BACKTRACKING(SOL, k)
IF ((a_1, ... a_k) è una soluzione) {stampala
   } ELSE {
     calcola S_{k+1}
     WHILE S_{k+1} ≠ ∅ {
       a_{k+1} ← un elemento di S_{k+1}, forma (a_1, ... a_k, a_{k+1})
       S_{k+1} ← 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 $n \times$, in modo tale che nessuna regina ne "minacci" un'altra.

Problema delle 8 regine

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

Cosa vuol dire "non minaccia?"

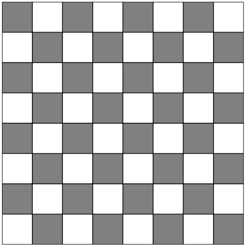
Problema delle 8 regine

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

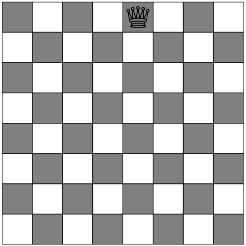
Cosa vuol dire "non minaccia?"

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

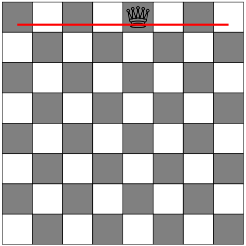
Esempio



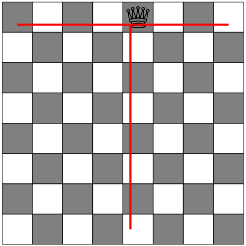
Esempio



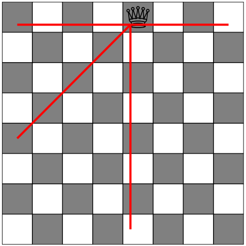
Esempio



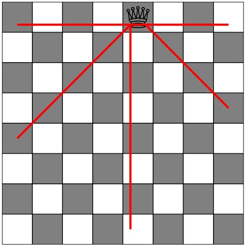
Esempio



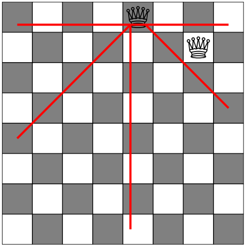
Esempio



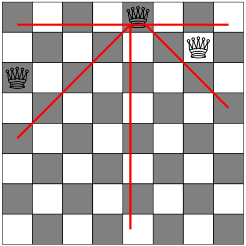
Esempio



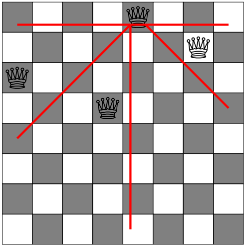
Esempio



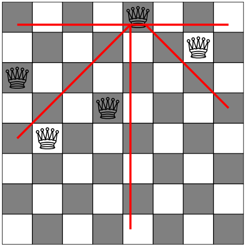
Esempio



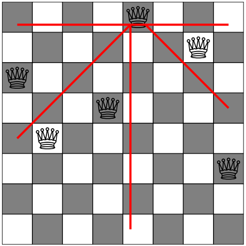
Esempio



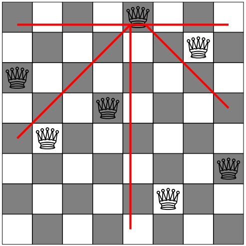
Esempio



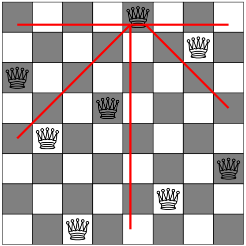
Esempio



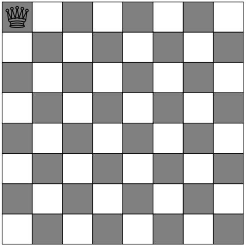
Esempio



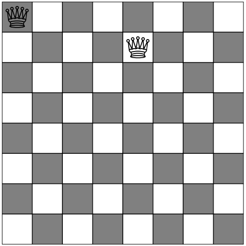
Esempio



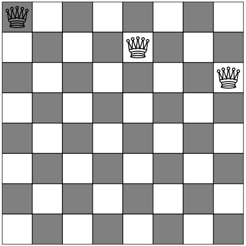
Un'altra possibile soluzione



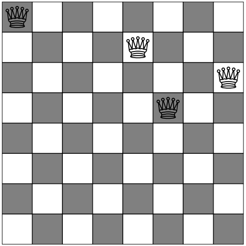
Un'altra possibile soluzione



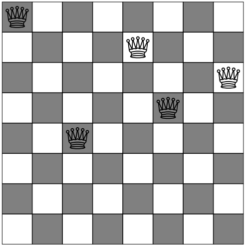
Un'altra possibile soluzione



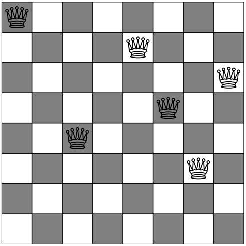
Un'altra possibile soluzione



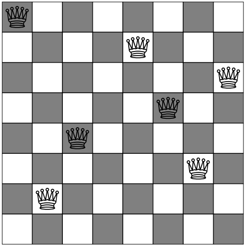
Un’altra possibile soluzione



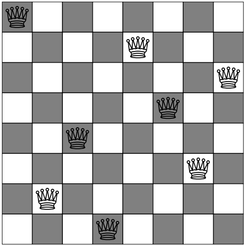
Un'altra possibile soluzione



Un'altra possibile soluzione



Un'altra possibile soluzione

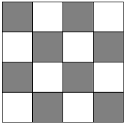


Rappresentiamo le posizioni delle regine usando un array $Q[1 \ldots n]$, dove $Q[i]$ indica la colonna della riga i-esima che contiene una regina.

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.

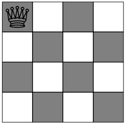
Quando chiamiamo l'algoritmo `PlaceQueens`, (che vediamo nelle prossime slide) il parametro input r è l'indice della prima riga vuota e il prefisso $Q[1 \dots r-1]$ contiene le posizioni delle prime $r-1$ regine.

Esempio



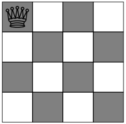
L'algoritmo funziona nel modo seguente.
Cerchiamo una colonna $j \in \{1, 2, 3, 4\}$ dove piazzare la regina r-esima (che ricordiamo, *mettiamo* nella riga r)

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)

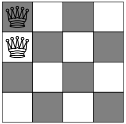
Esempio



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

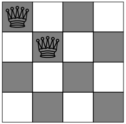
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

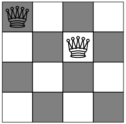
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

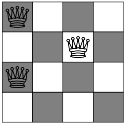
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

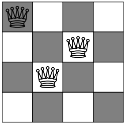
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

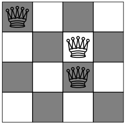
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

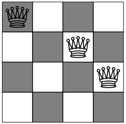
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

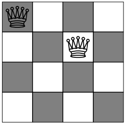
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

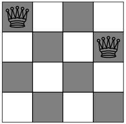
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

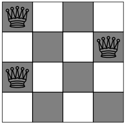
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

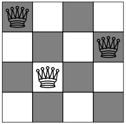
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

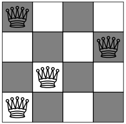
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

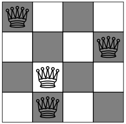
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

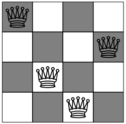
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

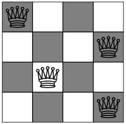
Esempio



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.

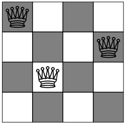
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

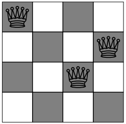
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

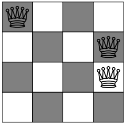
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

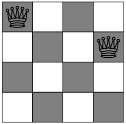
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

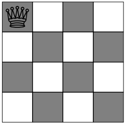
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

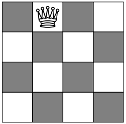
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

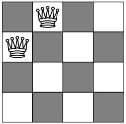
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

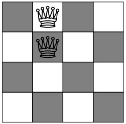
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

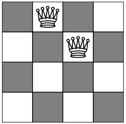
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

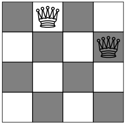
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

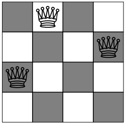
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

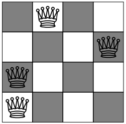
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

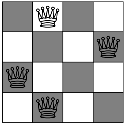
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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.

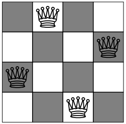
Esempio



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 $j \in \{1, 2, \ldots, n\}$ (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 $j \in \{1, 2, \ldots, n\}$ (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`$(Q[1 \ldots n], r)$

```
PlaceQueens(Q[1...n], r)
IF (r = n+1){
     print Q[1...n]
```

```
PlaceQueens(Q[1...n], r)
IF (r = n + 1){
    print Q[1...n]
     }ELSE{
       FOR(j=1, j<n+1, j=j+1){
          legal=True
```

```
PlaceQueens(Q[1...n], r)
IF (r = n + 1){
      print Q[1...n]
      }ELSE{
        FOR(j=1, j<n+1, j=j+1){
          legal=True
         FOR(i = 1, i < r, i = i + 1){
           IF((Q[i] = j)||(Q[i] = j + r - i)||(Q[i] = j - r + i)){
```

```
PlaceQueens(Q[1...n], r)
    IF (r = n+1){
        print Q[1...n]
         }ELSE{
           FOR(j=1, j<n+1, j=j+1){
             legal=True
            FOR(i = 1, i < r, i = i+1){
              IF((Q[i] = j)||(Q[i] = j + r - i)||(Q[i] = j - r + i)){
               legal=False
```

```
PlaceQueens(Q[1...n], r)
    IF (r = n + 1){
        print Q[1...n]
         }ELSE{
           FOR(j=1, j<n+1, j=j+1){
             legal=True
            FOR(i = 1, i < r, i = i + 1){
              IF((Q[i] = j)||(Q[i] = j + r − i)||(Q[i] = j − r + i)){
               legal=False
        }
      }
           IF(legal){
```

```
PlaceQueens(Q[1...n], r)
    IF (r = n + 1){
        print Q[1...n]
         }ELSE{
           FOR(j=1, j<n+1, j=j+1){
             legal=True
            FOR(i = 1, i < r, i = i + 1){
              IF((Q[i] = j)||(Q[i] = j + r - i)||(Q[i] = j - r + i)){
               legal=False
        }
      }
           IF(legal){
             Q[r] = j
```

```
PlaceQueens(Q[1...n], r)
    IF (r = n + 1){
        print Q[1...n]
         }ELSE{
           FOR(j=1, j<n+1, j=j+1){
             legal=True
            FOR(i = 1, i < r, i = i + 1){
              IF((Q[i] = j)||(Q[i] = j + r - i)||(Q[i] = j - r + i)){
               legal=False
        }
      }
           IF(legal){
             Q[r] = j
             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.

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 $j \in \{1, 2, \ldots, n\}$ dove piazzare la regina r-esima (che ricordiamo, *mettiamo* nella riga r)

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 $j \in \{1, 2, \ldots, n\}$ dove piazzare la regina r-esima (che ricordiamo, *mettiamo* nella riga r)

Per un dato valore $j \in \{1, 2, \ldots, n\}$ (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.

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, 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.

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`.

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,

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.

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`.

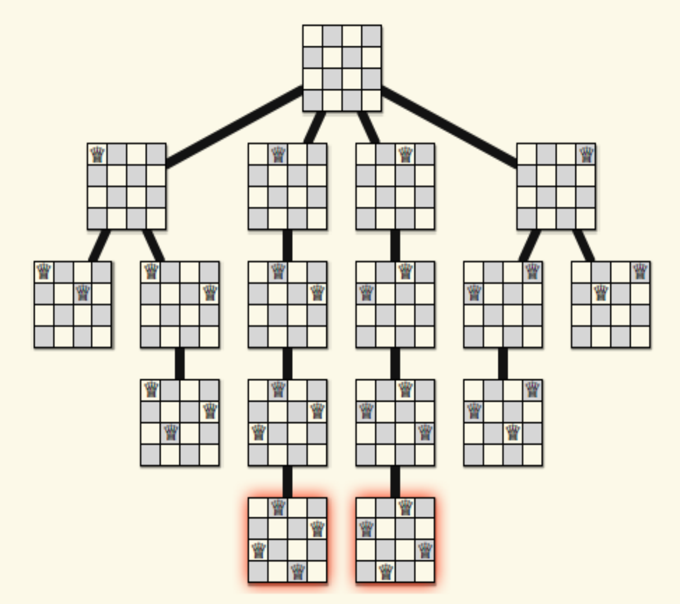
Se dopo aver eseguito il `FOR` della linea 6. il valore della variabile `legal` è `false`, *non* possiamo mettere la regina r-esima nella colonna j della riga r, incrementiamo il valore di j nel `FOR` 4 e ripetiamo il tutto. Se arriviamo a $j = n$ senza poter piazzare la regina r-esima, deduciamo che il piazzamento dell 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.

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

L'algoritmo `PlaceQueens`$(Q[1 \ldots n], n)$ teoricamente può generare tutte le permutazioni degli elementi $1, 2, \ldots, n$, 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`$(Q[1 \dots n], n)$ 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 d forza bruta che genererebbe tutte le n^n possibili posizionamenti delle n matrici nella scacchiera (cioè, n possibili posizione 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`$(Q[1 \dots n], n)$ 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 d forza bruta che genererebbe tutte le n^n possibili posizionamenti delle n matrici nella scacchiera (cioè, n possibili posizione per ognuna delle n righe).

Ad esempio, per $n = 8$ l'algoritmo `PlaceQueens`$(Q[1 \dots n], n)$ 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 $\texttt{PlaceQueens}(Q[1 \dots n], n)$ 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 d forza bruta che genererebbe tutte le n^n possibili posizionamenti delle n matrici nella scacchiera (cioè, n possibili posizione per ognuna delle n righe).

Ad esempio, per $n = 8$ l'algoritmo $\texttt{PlaceQueens}(Q[1 \dots n], n)$ 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`$(Q[1 \dots n], n)$ 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 d forza bruta che genererebbe tutte le n^n possibili posizionamenti delle n matrici nella scacchiera (cioè, n possibili posizione per ognuna delle n righe).

Ad esempio, per $n = 8$ l'algoritmo `PlaceQueens`$(Q[1 \dots n], n)$ 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.

Sudoku

Data una griglia 9×9, l'obiettivo è di riempire le sue caselle con cifre $n \in \{1, 2, \ldots, 9\}$ in maniera tale che *ogni* colonna,

Sudoku

Data una griglia 9×9, l'obiettivo è di riempire le sue caselle con cifre $n \in \{1, 2, \ldots, 9\}$ in maniera tale che *ogni* colonna, *ogni* riga

Sudoku

Data una griglia 9×9, l'obiettivo è di riempire le sue caselle con cifre $n \in \{1, 2, \dots, 9\}$ in maniera tale che *ogni* colonna, *ogni* riga e *ogni* 3×3 sottogriglia contenga *tutti* i numeri in $\{1, 2, \dots, 9\}$

Sudoku

Data una griglia 9×9, l'obiettivo è di riempire le sue caselle con cifre $n \in \{1, 2, \ldots, 9\}$ in maniera tale che *ogni* colonna, *ogni* riga e *ogni* 3×3 sottogriglia contenga *tutti* i numeri in $\{1, 2, \ldots, 9\}$

In generale, in input è data una griglia 9×9 con alcune caselle già riempite e vi è un *unico* modo per completare l'intera griglia.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Possibile algoritmo di Backtracking

- Riempiremo le celle vuote da sinistra a destra, per ogni riga,

Possibile algoritmo di Backtracking

- Riempiremo le celle vuote da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

Possibile algoritmo di Backtracking

- Riempiremo le celle vuote da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.
- Quando esaminiamo una cella ℓ vuota, l'insieme da cui possiamo scegliere l'elemento da inserire consta di $\{1, 2, \ldots, 9\} \setminus \{$gli elementi che già appaiono nella stessa riga, o stessa colonna, o stesso blocco 3×3 della cella $\ell\}$

Possibile algoritmo di Backtracking

- Riempiremo le celle vuote da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.
- Quando esaminiamo una cella ℓ vuota, l'insieme da cui possiamo scegliere l'elemento da inserire consta di $\{1, 2, \ldots, 9\} \setminus \{$gli elementi che già appaiono nella stessa riga, o stessa colonna, o stesso blocco 3×3 della cella $\ell\}$ Calcoleremo quindi
- $R_i = \{$gli elementi già inseriti nella riga i$\}$

Possibile algoritmo di Backtracking

- Riempiremo le celle vuote da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.
- Quando esaminiamo una cella ℓ vuota, l'insieme da cui possiamo scegliere l'elemento da inserire consta di $\{1, 2, \ldots, 9\} \setminus \{$gli elementi che già appaiono nella stessa riga, o stessa colonna, o stesso blocco 3×3 della cella $\ell\}$ Calcoleremo quindi
- $R_i = \{$gli elementi già inseriti nella riga i$\}$
- $C_j = \{$gli elementi già inseriti nella colonna j$\}$

Possibile algoritmo di Backtracking

- Riempiremo le celle vuote da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.
- Quando esaminiamo una cella ℓ vuota, l'insieme da cui possiamo scegliere l'elemento da inserire consta di $\{1, 2, \ldots, 9\} \setminus \{$gli elementi che già appaiono nella stessa riga, o stessa colonna, o stesso blocco 3×3 della cella $\ell\}$ Calcoleremo quindi
- $R_i = \{$gli elementi già inseriti nella riga i$\}$
- $C_j = \{$gli elementi già inseriti nella colonna j$\}$
- $B_k = \{$gli elementi già inseriti nel blocco k$\}$ per $1 \leq i, j, k \leq 9$

Numeriamo le 81 celle da sinistra a destra, per ogni riga,

Numeriamo le 81 celle da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

Numeriamo le 81 celle da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

La generica cella $\ell \in \{1, 2, \ldots, 81\}$, si troverà nella riga

$$r(\ell) = \left\lfloor \frac{\ell - 1}{9} \right\rfloor + 1$$

Numeriamo le 81 celle da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

La generica cella $\ell \in \{1, 2, \ldots, 81\}$, si troverà nella riga

$$r(\ell) = \left\lfloor \frac{\ell - 1}{9} \right\rfloor + 1$$

nella colonna

$$c(\ell) = (\ell - 1) \pmod 9 + 1$$

Numeriamo le 81 celle da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

La generica cella $\ell \in \{1, 2, \ldots, 81\}$, si troverà nella riga

$$r(\ell) = \left\lfloor \frac{\ell - 1}{9} \right\rfloor + 1$$

nella colonna

$$c(\ell) = (\ell - 1) \pmod 9 + 1$$

e nel blocco

$$b(\ell) = 3 \times \left\lfloor \frac{r(\ell) - 1}{3} \right\rfloor + \left\lfloor \frac{c(\ell) - 1}{3} \right\rfloor + 1$$

`RigaColonnaBlocco`(ℓ)

Numeriamo le 81 celle da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

La generica cella $\ell \in \{1, 2, \ldots, 81\}$, si troverà nella riga

$$r(\ell) = \left\lfloor \frac{\ell - 1}{9} \right\rfloor + 1$$

nella colonna

$$c(\ell) = (\ell - 1) \pmod 9 + 1$$

e nel blocco

$$b(\ell) = 3 \times \left\lfloor \frac{r(\ell) - 1}{3} \right\rfloor + \left\lfloor \frac{c(\ell) - 1}{3} \right\rfloor + 1$$

RigaColonnaBlocco(ℓ)
1. $i = \lfloor \frac{\ell-1}{9} \rfloor + 1$

Numeriamo le 81 celle da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

La generica cella $\ell \in \{1, 2, \ldots, 81\}$, si troverà nella riga

$$r(\ell) = \left\lfloor \frac{\ell - 1}{9} \right\rfloor + 1$$

nella colonna

$$c(\ell) = (\ell - 1) \pmod 9 + 1$$

e nel blocco

$$b(\ell) = 3 \times \left\lfloor \frac{r(\ell) - 1}{3} \right\rfloor + \left\lfloor \frac{c(\ell) - 1}{3} \right\rfloor + 1$$

RigaColonnaBlocco(ℓ)

1. $i = \lfloor \frac{\ell-1}{9} \rfloor + 1$
2. $j = (\ell - 1) \pmod 9 + 1$

Numeriamo le 81 celle da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

La generica cella $\ell \in \{1, 2, \ldots, 81\}$, si troverà nella riga

$$r(\ell) = \left\lfloor \frac{\ell - 1}{9} \right\rfloor + 1$$

nella colonna

$$c(\ell) = (\ell - 1) \pmod 9 + 1$$

e nel blocco

$$b(\ell) = 3 \times \left\lfloor \frac{r(\ell) - 1}{3} \right\rfloor + \left\lfloor \frac{c(\ell) - 1}{3} \right\rfloor + 1$$

RigaColonnaBlocco(ℓ)
1. $i = \lfloor \frac{\ell-1}{9} \rfloor + 1$
2. $j = (\ell - 1) \pmod 9 + 1$
3. $k = 3 \times \lfloor \frac{i-1}{3} \rfloor + \lfloor \frac{j-1}{3} \rfloor + 1$

Numeriamo le 81 celle da sinistra a destra, per ogni riga, considerando la prima riga, poi la seconda, etc.

La generica cella $\ell \in \{1, 2, \ldots, 81\}$, si troverà nella riga

$$r(\ell) = \left\lfloor \frac{\ell - 1}{9} \right\rfloor + 1$$

nella colonna

$$c(\ell) = (\ell - 1) \pmod 9 + 1$$

e nel blocco

$$b(\ell) = 3 \times \left\lfloor \frac{r(\ell) - 1}{3} \right\rfloor + \left\lfloor \frac{c(\ell) - 1}{3} \right\rfloor + 1$$

`RigaColonnaBlocco`(ℓ)

1. $i = \lfloor \frac{\ell - 1}{9} \rfloor + 1$
2. $j = (\ell - 1) \pmod 9 + 1$
3. $k = 3 \times \lfloor \frac{i-1}{3} \rfloor + \lfloor \frac{j-1}{3} \rfloor + 1$
4. `return`(i, j, k)

Occorre anche passare all'algoritmo i valori delle celle che sono state già fissate.

Occorre anche passare all'algoritmo i valori delle celle che sono state già fissate. Lo faremo mediante un vettore $a = (a_1, a_2, \ldots, a_{81})$ dove $a_\ell = 0$ se la cella ℓ-esima è vuota,

Occorre anche passare all'algoritmo i valori delle celle che sono state già fissate. Lo faremo mediante un vettore $a = (a_1, a_2, \ldots, a_{81})$ dove $a_\ell = 0$ se la cella ℓ-esima è vuota, altrimenti a_ℓ conterrà il valore pre-assegnato alla cella ℓ-esima.

Occorre anche passare all'algoritmo i valori delle celle che sono state già fissate. Lo faremo mediante un vettore $a = (a_1, a_2, \ldots, a_{81})$ dove $a_\ell = 0$ se la cella ℓ-esima è vuota, altrimenti a_ℓ conterrà il valore pre-assegnato alla cella ℓ-esima.

Occorrerà tenere a mente i valori già inseriti (ad ogni passo dell'algoritmo) nella riga i-esima, colonna j-esima e blocco k-esimo

Occorre anche passare all'algoritmo i valori delle celle che sono state già fissate. Lo faremo mediante un vettore $a = (a_1, a_2, \ldots, a_{81})$ dove $a_\ell = 0$ se la cella ℓ-esima è vuota, altrimenti a_ℓ conterrà il valore pre-assegnato alla cella ℓ-esima.

Occorrerà tenere a mente i valori già inseriti (ad ogni passo dell'algoritmo) nella riga i-esima, colonna j-esima e blocco k-esimo . Lo faremo mediante gli insiemi $R[i], C[j], B[k]$, per $1 \leq i, j, k \leq 9$, da aggiornare ad ogni passo.

Inizializzazione

```
Inizializza(a)
```

Inizializzazione

```
Inizializza(a)
FOR(i=1, i<10, i=i+1){
    R[i] = ∅
```

Inizializzazione

```
Inizializza(a)
FOR(i=1, i<10, i=i+1){
      R[i] = ∅
      C[i] = ∅
```

Inizializzazione

Inizializza(a)
FOR(i=1, i<10, i=i+1){
    $R[i] = \emptyset$
    $C[i] = \emptyset$
    $B[i] = \emptyset$

Inizializzazione

```
Inizializza(a)
1.  FOR(i=1, i<10, i=i+1){
2.        R[i] = ∅
3.        C[i] = ∅
4.        B[i] = ∅
     }
5.  FOR(ℓ=1, i<82, ℓ = ℓ + 1){
6.        (i, j, k)= RigaColonnaBlocco(ℓ)
```

Inizializzazione

Inizializza(a)
FOR(i=1, i<10, i=i+1)$\{$
    $R[i] = \emptyset$
    $C[i] = \emptyset$
    $B[i] = \emptyset$
$\}$
FOR(ℓ=1, i<82, $\ell = \ell + 1$)$\{$
    (i, j, k)= RigaColonnaBlocco(ℓ)
    $R[i] = R[i] \cup \{a_\ell\}$

Inizializzazione

Inizializza(a)
FOR(i=1, i<10, i=i+1){
    $R[i] = \emptyset$
    $C[i] = \emptyset$
    $B[i] = \emptyset$
}
FOR(ℓ=1, i<82, $\ell = \ell + 1$){
    (i, j, k)= RigaColonnaBlocco(ℓ)
    $R[i] = R[i] \cup \{a_\ell\}$
    $C[j] = C[j] \cup \{a_\ell\}$

Inizializzazione

Inizializza(a)
FOR(i=1, i<10, i=i+1)$\{$
    $R[i] = \emptyset$
    $C[i] = \emptyset$
    $B[i] = \emptyset$
  $\}$
FOR(ℓ=1, i<82, $\ell = \ell + 1$)$\{$
    (i, j, k)= RigaColonnaBlocco(ℓ)
    $R[i] = R[i] \cup \{a_\ell\}$
    $C[j] = C[j] \cup \{a_\ell\}$
    $B[k] = B[k] \cup \{a_\ell\}$
  $\}$

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
1.  IF(ℓ = 81){
2.      return x
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
IF(ℓ = 81){
     return x
     }ELSE{
     IF(a_{ℓ+1} ≠ 0) {
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

Sudoku(ℓ)
IF($\ell = 81$){
    return x
    }ELSE{
    IF($a_{\ell+1} \neq 0$) {
        $x_{\ell+1} = a_{\ell+1}$
        Sudoku($\ell + 1$)

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
IF(ℓ = 81){
     return x
     }ELSE{
     IF(a_{ℓ+1} ≠ 0) {
        x_{ℓ+1} = a_{ℓ+1}
        Sudoku(ℓ + 1)
       }ELSE{
        (i, j, k)=RigaColonnaBlocco(ℓ + 1)
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

Sudoku(ℓ)
IF($\ell = 81$){
    return x
    }ELSE{
    IF($a_{\ell+1} \neq 0$) {
        $x_{\ell+1} = a_{\ell+1}$
        Sudoku($\ell + 1$)
      }ELSE{
        (i, j, k)=RigaColonnaBlocco($\ell + 1$)
        scelta=$\{1, \ldots, 9\} \setminus (R[i] \cup C[j] \cup B[k])$

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
IF(ℓ = 81){
    return x
    }ELSE{
    IF(a_{ℓ+1} ≠ 0) {
        x_{ℓ+1} = a_{ℓ+1}
        Sudoku(ℓ + 1)
      }ELSE{
        (i, j, k)=RigaColonnaBlocco(ℓ + 1)
        scelta={1, . . . , 9} \ (R[i] ∪ C[j] ∪ B[k])
         FOR (y ∈ scelta){
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
  IF(ℓ = 81){
      return x
      }ELSE{
      IF(a_{ℓ+1} ≠ 0) {
          x_{ℓ+1} = a_{ℓ+1}
          Sudoku(ℓ + 1)
        }ELSE{
          (i, j, k)=RigaColonnaBlocco(ℓ + 1)
          scelta={1, ..., 9} \ (R[i] ∪ C[j] ∪ B[k])
           FOR (y ∈ scelta){
           R[i] = R[i] ∪ {y}; C[j] = C[j] ∪ {y}; B[k] = B[k] ∪ {y};
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
IF(ℓ = 81){
      return x
      }ELSE{
      IF(a_{ℓ+1} ≠ 0) {
          x_{ℓ+1} = a_{ℓ+1}
          Sudoku(ℓ + 1)
        }ELSE{
          (i, j, k)=RigaColonnaBlocco(ℓ + 1)
          scelta={1, . . . , 9} \ (R[i] ∪ C[j] ∪ B[k])
           FOR (y ∈ scelta){
           R[i] = R[i] ∪ {y}; C[j] = C[j] ∪ {y}; B[k] = B[k] ∪ {y};
           x_{ℓ+1} = y
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
  IF(ℓ = 81){
       return x
       }ELSE{
       IF(a_{ℓ+1} ≠ 0) {
          x_{ℓ+1} = a_{ℓ+1}
          Sudoku(ℓ + 1)
        }ELSE{
          (i, j, k)=RigaColonnaBlocco(ℓ + 1)
          scelta={1, . . . , 9} \ (R[i] ∪ C[j] ∪ B[k])
           FOR (y ∈ scelta){
           R[i] = R[i] ∪ {y}; C[j] = C[j] ∪ {y}; B[k] = B[k] ∪ {y};
           x_{ℓ+1} = y
           Sudoku(ℓ + 1)
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
IF(ℓ = 81){
      return x
      }ELSE{
      IF(a_{ℓ+1} ≠ 0) {
         x_{ℓ+1} = a_{ℓ+1}
         Sudoku(ℓ + 1)
        }ELSE{
         (i, j, k)=RigaColonnaBlocco(ℓ + 1)
         scelta={1, . . . , 9} \ (R[i] ∪ C[j] ∪ B[k])
          FOR (y ∈ scelta){
          R[i] = R[i] ∪ {y}; C[j] = C[j] ∪ {y}; B[k] = B[k] ∪ {y};
          x_{ℓ+1} = y
          Sudoku(ℓ + 1)
          R[i] = R[i] \ {y}; C[j] = C[j] \ {y}; B[k] = B[k] \ {y};
```

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

Sudoku(ℓ)
IF($\ell = 81$){
    return x
    }ELSE{
    IF($a_{\ell+1} \neq 0$) {
        $x_{\ell+1} = a_{\ell+1}$
        Sudoku($\ell + 1$)
      }ELSE{
        (i, j, k)=RigaColonnaBlocco($\ell + 1$)
        scelta=$\{1, \ldots, 9\} \setminus (R[i] \cup C[j] \cup B[k])$
         FOR ($y \in$ scelta){
         $R[i] = R[i] \cup \{y\}; C[j] = C[j] \cup \{y\}; B[k] = B[k] \cup \{y\};$
         $x_{\ell+1} = y$
         Sudoku($\ell + 1$)
         $R[i] = R[i] \setminus \{y\}; C[j] = C[j] \setminus \{y\}; B[k] = B[k] \setminus \{y\};$
    }
  }

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \dots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
IF(ℓ = 81){
    return x
    }ELSE{
    IF(a_{ℓ+1} ≠ 0) {
       x_{ℓ+1} = a_{ℓ+1}
       Sudoku(ℓ + 1)
      }ELSE{
       (i, j, k)=RigaColonnaBlocco(ℓ + 1)
       scelta={1, ..., 9} \ (R[i] ∪ C[j] ∪ B[k])
        FOR (y ∈ scelta){
        R[i] = R[i] ∪ {y}; C[j] = C[j] ∪ {y}; B[k] = B[k] ∪ {y};
        x_{ℓ+1} = y
        Sudoku(ℓ + 1)
        R[i] = R[i] \ {y}; C[j] = C[j] \ {y}; B[k] = B[k] \ {y};
    }
  }
```

L'Algoritmo completo:
Inizializza(a)

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
    IF(ℓ = 81){
        return x
        }ELSE{
        IF(a_{ℓ+1} ≠ 0) {
            x_{ℓ+1} = a_{ℓ+1}
            Sudoku(ℓ + 1)
          }ELSE{
            (i, j, k)=RigaColonnaBlocco(ℓ + 1)
            scelta={1, ..., 9} \ (R[i] ∪ C[j] ∪ B[k])
             FOR (y ∈ scelta){
             R[i] = R[i] ∪ {y}; C[j] = C[j] ∪ {y}; B[k] = B[k] ∪ {y};
             x_{ℓ+1} = y
             Sudoku(ℓ + 1)
             R[i] = R[i] \ {y}; C[j] = C[j] \ {y}; B[k] = B[k] \ {y};
        }
     }
```

L'Algoritmo completo:
Inizializza(a)
$x = ()$

L'algoritmo: restituisce la soluzione in un vettore $x = (x_1, x_2, \ldots, x_{81})$

Le variabili a, R, C, B, B, x sono globali

```
Sudoku(ℓ)
IF(ℓ = 81){
    return x
    }ELSE{
    IF(a_{ℓ+1} ≠ 0) {
        x_{ℓ+1} = a_{ℓ+1}
        Sudoku(ℓ + 1)
      }ELSE{
        (i, j, k)=RigaColonnaBlocco(ℓ + 1)
        scelta={1, ..., 9} \ (R[i] ∪ C[j] ∪ B[k])
         FOR (y ∈ scelta){
         R[i] = R[i] ∪ {y}; C[j] = C[j] ∪ {y}; B[k] = B[k] ∪ {y};
         x_{ℓ+1} = y
         Sudoku(ℓ + 1)
         R[i] = R[i] \ {y}; C[j] = C[j] \ {y}; B[k] = B[k] \ {y};
      }
  }
```

L'Algoritmo completo:

```
Inizializza(a)
x = ()
Sudoku(0)
```

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 ci può essere più di un sottoinsieme di questo tipo. A 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 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 ci può essere più di un sottoinsieme di questo tipo. A 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 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 ci può essere più di un sottoinsieme di questo tipo. A 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 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 restituiscono immediatamente `True`, perché il l'insieme vuoto è un sottoinsieme di ogni insieme X, e gli elementi del set 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 ci può essere più di un sottoinsieme di questo tipo. A 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 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 restituiscono immediatamente `True`, perché il l'insieme vuoto è un sottoinsieme di ogni insieme X, e gli elementi del set 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 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 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 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 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 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 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 $\texttt{Zaino}(X, T)$ riducendolo a due casi più semplici: $\texttt{Zaino}(X \setminus \{x\}, T - x)$

Per il caso generale, consideriamo un elemento arbitrario $x \in X$. Esiste un sottoinsieme di X che somma a T se e solo se 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 $\texttt{Zaino}(X, T)$ riducendolo a due casi più semplici: $\texttt{Zaino}(X \setminus \{x\}, T - x)$ e $\texttt{Zaino}(X \setminus \{x\}, T)$.

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

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

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

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

```
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)
```

```
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)
```

```
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)
     }
```

```
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)
       }
```

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 = ∅)){
         return False
        }ELSE{
         x = ultimo elemento di X
         con=Zaino(X \ {x}, T − x)
         senza=Zaino(X \ {x}, T)
         return (con∨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 = ∅)){
       return False
      }ELSE{
       x = ultimo elemento di X
       con=Zaino(X \ {x}, T − x)
       senza=Zaino(X \ {x}, T)
       return (con∨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 ∅
```

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 ∅
  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 $X[1 \ldots n]$.

```
ConstructSubset(X, i, T)
  IF(T == 0)
      return ∅
  IF((T < 0)||(n == 0))
      return ‘‘non c’è’’
  Y =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 ∅
  IF((T < 0)||(n == 0))
      return ``non c'è''
  Y =ConstructSubset(X, i − 1, T)
  IF Y ≠ ``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 ∅
   IF((T < 0)||(n == 0))
      return ``non c'è''
   Y =ConstructSubset(X, i − 1, T)
   IF Y ≠ ``non c'è''
      return Y
   Y =ConstructSubset(X, i − 1, T − X[i])
   IF Y ≠ ``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 $X[1 \ldots n]$.

```
ConstructSubset(X, i, T)
  IF(T == 0)
     return ∅
  IF((T < 0)||(n == 0))
     return ‘‘non c’è’’
  Y =ConstructSubset(X, i − 1, T)
  IF Y ≠ ‘‘non c’è’’
     return Y
  Y =ConstructSubset(X, i − 1, T − X[i])
  IF Y ≠ ‘‘non c’è’’
     return Y ∪ {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 $X[1 \ldots n]$.

```
ConstructSubset(X, i, T)
  IF(T == 0)
      return ∅
  IF((T < 0)||(n == 0))
      return ``non c'è''
  Y =ConstructSubset(X, i − 1, T)
  IF Y ≠ ``non c'è''
      return Y
  Y =ConstructSubset(X, i − 1, T − X[i])
  IF Y ≠ ``non c'è''
      return Y ∪ {X[i]}
  return ``non c'è''
```