

Backtrack

Giuseppe Persiano

Università di Salerno

Ottobre, 2020

L'Algoritmo di BackTrack per il Maze

- 1 inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*

L'Algoritmo di BackTrack per il Maze

- 1 inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- 2 poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.

L'Algoritmo di BackTrack per il Maze

- 1 inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- 2 poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- 3 segna S come visitata

L'Algoritmo di BackTrack per il Maze

- 1 inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- 2 poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- 3 segna S come visitata
- 4 finché lo stack è non vuoto:

L'Algoritmo di BackTrack per il Maze

- 1 inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- 2 poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- 3 segna S come visitata
- 4 finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,

L'Algoritmo di BackTrack per il Maze

- 1 inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- 2 poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- 3 segna S come visitata
- 4 finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,
 - ▶ se non ci sono altre mosse ammissibili

L'Algoritmo di BackTrack per il Maze

- 1 inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- 2 poni nello stack ($S = (sr, sc), 0$) ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- 3 segna S come visitata
- 4 finché lo stack è non vuoto:
 - ▶ fai pop dallo stack ($r, c, lmove$),
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*

L'Algoritmo di BackTrack per il Maze

- ❶ inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- ❷ poni nello stack ($S = (sr, sc), 0$) ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- ❸ segna S come visitata
- ❹ finché lo stack è non vuoto:
 - ▶ fai pop dallo stack ($r, c, lmove$),
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*
 - ★ continue

L'Algoritmo di BackTrack per il Maze

- ❶ inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- ❷ poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- ❸ segna S come visitata
- ❹ finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*
 - ★ continue
 - ▶ sia $nmove$ prossima mossa ammissibile

L'Algoritmo di BackTrack per il Maze

- ❶ inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- ❷ poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- ❸ segna S come visitata
- ❹ finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*
 - ★ continue
 - ▶ sia $nmove$ prossima mossa ammissibile
 - ▶ marca (r, c) come *visitata*

L'Algoritmo di BackTrack per il Maze

- ❶ inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- ❷ poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- ❸ segna S come visitata
- ❹ finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*
 - ★ continue
 - ▶ sia $nmove$ prossima mossa ammissibile
 - ▶ marca (r, c) come *visitata*
 - ▶ fai push di $(r, c, nmove)$

L'Algoritmo di BackTrack per il Maze

- ❶ inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- ❷ poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- ❸ segna S come visitata
- ❹ finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*
 - ★ continue
 - ▶ sia $nmove$ prossima mossa ammissibile
 - ▶ marca (r, c) come *visitata*
 - ▶ fai push di $(r, c, nmove)$
 - ▶ fai mossa $nmove$ e calcola la nuova posizione $(newr, newc)$

L'Algoritmo di BackTrack per il Maze

- ❶ inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- ❷ poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- ❸ segna S come visitata
- ❹ finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*
 - ★ continue
 - ▶ sia $nmove$ prossima mossa ammissibile
 - ▶ marca (r, c) come *visitata*
 - ▶ fai push di $(r, c, nmove)$
 - ▶ fai mossa $nmove$ e calcola la nuova posizione $(newr, newc)$
 - ▶ if $(newr, newc) == E$, return TRUE

L'Algoritmo di BackTrack per il Maze

- ❶ inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- ❷ poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- ❸ segna S come visitata
- ❹ finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*
 - ★ continue
 - ▶ sia $nmove$ prossima mossa ammissibile
 - ▶ marca (r, c) come *visitata*
 - ▶ fai push di $(r, c, nmove)$
 - ▶ fai mossa $nmove$ e calcola la nuova posizione $(newr, newc)$
 - ▶ if $(newr, newc) == E$, return TRUE
 - ▶ fai push di $(newr, newc, 0)$

L'Algoritmo di BackTrack per il Maze

- ❶ inizia dallo stato in cui tutte le caselle sono *libere* o *bloccate*
- ❷ poni nello stack $(S = (sr, sc), 0)$ ad indicare che l'ultima casella visitata è S e che nessuna mossa è stata effettuata.
- ❸ segna S come visitata
- ❹ finché lo stack è non vuoto:
 - ▶ fai pop dallo stack $(r, c, lmove)$,
 - ▶ se non ci sono altre mosse ammissibili
 - ★ marca (r, c) come *esaurita*
 - ★ continue
 - ▶ sia $nmove$ prossima mossa ammissibile
 - ▶ marca (r, c) come *visitata*
 - ▶ fai push di $(r, c, nmove)$
 - ▶ fai mossa $nmove$ e calcola la nuova posizione $(newr, newc)$
 - ▶ if $(newr, newc) == E$, return TRUE
 - ▶ fai push di $(newr, newc, 0)$
- ❺ return FALSE

Backtrack

- 1 Si parte dallo *stato iniziale* **statIn**

Backtrack

- 1 Si parte dallo *stato iniziale* `statIn`
- 2 Segna `(statIn, None)` come **visitata**

Backtrack

- 1 Si parte dallo *stato iniziale* `statIn`
- 2 Segna `(statIn, None)` come **visitata**
- 3 `Stack.Push(statIn, None)`

Backtrack

- 1 Si parte dallo *stato iniziale* `statIn`
- 2 Segna `(statIn, None)` come **visitata**
- 3 `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata

Backtrack

- 1 Si parte dallo *stato iniziale* `statIn`
- 2 Segna `(statIn, None)` come **visitata**
- 3 `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`

Backtrack

- 1 Si parte dallo *stato iniziale* `statIn`
- 2 Segna `(statIn, None)` come **visitata**
- 3 `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- 4 Finché lo stack non è vuoto

Backtrack

- 1 Si parte dallo *stato iniziale* `statIn`
- 2 Segna `(statIn, None)` come **visitata**
- 3 `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- 4 Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`

Backtrack

- ① Si parte dallo *stato iniziale* `statIn`
- ② Segna `(statIn, None)` come **visitata**
- ③ `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- ④ Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`
 - ▶ Calcola `newMove`

Backtrack

- ① Si parte dallo *stato iniziale* `statIn`
- ② Segna `(statIn, None)` come **visitata**
- ③ `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- ④ Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`
 - ▶ Calcola `newMove`
 - ★ mossa dopo `move` ammissibile per `state`

Backtrack

- ① Si parte dallo *stato iniziale* `statIn`
- ② Segna `(statIn, None)` come **visitata**
- ③ `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- ④ Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`
 - ▶ Calcola `newMove`
 - ★ mossa dopo `move` ammissibile per `state`
 - ▶ if `newMove ≠ None`

Backtrack

- ① Si parte dallo *stato iniziale* `statIn`
- ② Segna `(statIn, None)` come **visitata**
- ③ `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- ④ Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`
 - ▶ Calcola `newMove`
 - ★ mossa dopo `move` ammissibile per `state`
 - ▶ if `newMove ≠ None`
 - ★ `Stack.Push(state, newMove)`

Backtrack

- ① Si parte dallo *stato iniziale* `statIn`
- ② Segna `(statIn, None)` come **visitata**
- ③ `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- ④ Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`
 - ▶ Calcola `newMove`
 - ★ mossa dopo `move` ammissibile per `state`
 - ▶ if `newMove ≠ None`
 - ★ `Stack.Push(state, newMove)`
 - ★ Calcola lo stato `newState` ottenuta dall'eseguire `newMove` in `state`

Backtrack

- ① Si parte dallo *stato iniziale* `statIn`
- ② Segna `(statIn, None)` come **visitata**
- ③ `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- ④ Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`
 - ▶ Calcola `newMove`
 - ★ mossa dopo `move` ammissibile per `state`
 - ▶ if `newMove ≠ None`
 - ★ `Stack.Push(state, newMove)`
 - ★ Calcola lo stato `newState` ottenuta dall'eseguire `newMove` in `state`
 - ★ If `newState` è uno stato **finale**, return `True`

Backtrack

- ① Si parte dallo *stato iniziale* `statIn`
- ② Segna `(statIn, None)` come **visitata**
- ③ `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- ④ Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`
 - ▶ Calcola `newMove`
 - ★ mossa dopo `move` ammissibile per `state`
 - ▶ if `newMove ≠ None`
 - ★ `Stack.Push(state, newMove)`
 - ★ Calcola lo stato `newState` ottenuta dall'eseguire `newMove` in `state`
 - ★ If `newState` è uno stato **finale**, return `True`
 - ★ `Stack.Push(newState, None)`

Backtrack

- ① Si parte dallo *stato iniziale* `statIn`
- ② Segna `(statIn, None)` come **visitata**
- ③ `Stack.Push(statIn, None)`
 - ▶ `None` indica che nessuna mossa è stata effettuata
 - ▶ lo stack contiene coppie `(state, move)`
- ④ Finché lo stack non è vuoto
 - ▶ `(state, move) ← Stack.Pop`
 - ▶ Calcola `newMove`
 - ★ mossa dopo `move` ammissibile per `state`
 - ▶ if `newMove ≠ None`
 - ★ `Stack.Push(state, newMove)`
 - ★ Calcola lo stato `newState` ottenuta dall'eseguire `newMove` in `state`
 - ★ If `newState` è uno stato **finale**, return `True`
 - ★ `Stack.Push(newState, None)`
- ⑤ return `False`

Backtrack

- 1 Definire lo stato
- 2 Come calcolare lo stato iniziale
- 3 Come marcare uno stato $(M, (r, c))$ come già visitato
- 4 Generare la prossima mossa `nmove` dopo `lmove` quando siamo in `state`
- 5 Eseguire una mossa `nmove` per andare da `state` a `newState`
- 6 Controllare se uno stato $(M, (r, c))$ è finale

Backtrack

- ① Definire lo stato
 - ▶ Griglia M in cui ogni casella è
 - ★ libera
 - ★ bloccata
 - ★ visitata
 - ★ finale
 - ▶ Casella (r, c) attualmente occupata
- ② Come calcolare lo stato iniziale
- ③ Come marcare uno stato $(M, (r, c))$ come già visitato
- ④ Generare la prossima mossa $nmove$ dopo $lmove$ quando siamo in $state$
- ⑤ Eseguire una mossa $nmove$ per andare da $state$ a $newState$
- ⑥ Controllare se uno stato $(M, (r, c))$ è finale

Backtrack

- ① Definire lo stato
- ② Come calcolare lo stato iniziale
 - ▶ Nessuna casella visitata (quindi solo **libere**, **bloccate**, o **finale**)
 - ▶ Posizione iniziale S
- ③ Come marcare uno stato $(M, (r, c))$ come già visitato
- ④ Generare la prossima mossa $nmov$ dopo $lmov$ quando siamo in $state$
- ⑤ Eseguire una mossa $nmov$ per andare da $state$ a $newState$
- ⑥ Controllare se uno stato $(M, (r, c))$ è finale

Backtrack

- 1 Definire lo stato
- 2 Come calcolare lo stato iniziale
- 3 Come marcare uno stato $(M, (r, c))$ come già visitato
 - ▶ poni $M[r][c] = \textit{visitata}$
- 4 Generare la prossima mossa \textit{nmove} dopo \textit{lmove} quando siamo in \textit{state}
- 5 Eseguire una mossa \textit{nmove} per andare da \textit{state} a $\textit{newState}$
- 6 Controllare se uno stato $(M, (r, c))$ è finale

Backtrack

- 1 Definire lo stato
- 2 Come calcolare lo stato iniziale
- 3 Come marcare uno stato $(M, (r, c))$ come già visitato
- 4 Generare la prossima mossa `nmove` dopo `lmove` quando siamo in `state`
 - ▶ $N \rightarrow E \rightarrow S \rightarrow O$
- 5 Eseguire una mossa `nmove` per andare da `state` a `newState`
- 6 Controllare se uno stato $(M, (r, c))$ è finale

Backtrack

- 1 Definire lo stato
- 2 Come calcolare lo stato iniziale
- 3 Come marcare uno stato $(M, (r, c))$ come già visitato
- 4 Generare la prossima mossa `nmove` dopo `lmove` quando siamo in `state`
- 5 Eseguire una mossa `nmove` per andare da `state` a `newState`
 - ▶ calcola nuovi valori di r e c
- 6 Controllare se uno stato $(M, (r, c))$ è finale

Backtrack

- 1 Definire lo stato
- 2 Come calcolare lo stato iniziale
- 3 Come marcare uno stato $(M, (r, c))$ come già visitato
- 4 Generare la prossima mossa `nmove` dopo `lmove` quando siamo in `state`
- 5 Eseguire una mossa `nmove` per andare da `state` a `newState`
- 6 Controllare se uno stato $(M, (r, c))$ è finale
 - ▶ controlla se $M[r][c] = \text{finale}$

Backtrack

- Definiamo una classe **Backtrack** che ha solo

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ **Costruttore**

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ Costruttore
 - ▶ Metodo Solve

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ Costruttore
 - ▶ Metodo `Solve`

- Definiamo una classe **Maze** derivata da `Backtrack` che fornisce

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ Costruttore
 - ▶ Metodo Solve

- Definiamo una classe **Maze** derivata da Backtrack che fornisce
 - ▶ Costruttore

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ Costruttore
 - ▶ Metodo Solve
- Definiamo una classe **Maze** derivata da Backtrack che fornisce
 - ▶ Costruttore
 - ▶ initState

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ Costruttore
 - ▶ Metodo Solve
- Definiamo una classe **Maze** derivata da Backtrack che fornisce
 - ▶ Costruttore
 - ▶ initState
 - ▶ setVisited

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ Costruttore
 - ▶ Metodo Solve
- Definiamo una classe **Maze** derivata da Backtrack che fornisce
 - ▶ Costruttore
 - ▶ initState
 - ▶ setVisited
 - ▶ nextAdmMove

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ Costruttore
 - ▶ Metodo Solve
- Definiamo una classe **Maze** derivata da Backtrack che fornisce
 - ▶ Costruttore
 - ▶ initState
 - ▶ setVisited
 - ▶ nextAdmMove
 - ▶ makeMove

Backtrack

- Definiamo una classe **Backtrack** che ha solo
 - ▶ Costruttore
 - ▶ Metodo Solve
- Definiamo una classe **Maze** derivata da Backtrack che fornisce
 - ▶ Costruttore
 - ▶ initState
 - ▶ setVisited
 - ▶ nextAdmMove
 - ▶ makeMove
 - ▶ isFinal

Cosa ci guadagno?

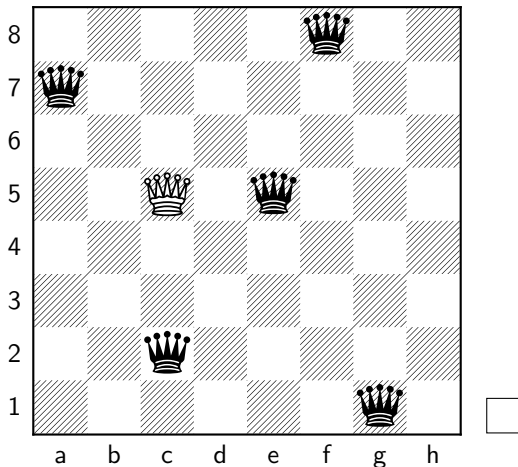
- Non devo riscrivere il metodo `Solve`
 - ▶ Posso usare la classe `Backtrack`

Cosa ci guadagno?

- Non devo riscrivere il metodo `Solve`
 - ▶ Posso usare la classe `Backtrack`
- Devo definire solo
 - ▶ `Come è fatto lo stato`
 - ▶ `Costruttore`
 - ▶ `initState`
 - ▶ `setVisited`
 - ▶ `nextAdmMove`
 - ▶ `makeMove`
 - ▶ `isFinal`

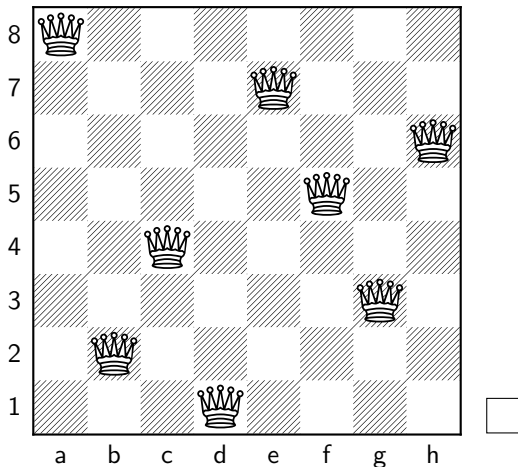
N regine

Piazzare 8 regine su una scacchiera in modo tale che nessuna regina attacchi un'altra regina.



N regine

Piazzare 8 regine su una scacchiera in modo tale che nessuna regina attacchi un'altra regina.



Cosa ci guadagno?

- Non devo riscrivere il metodo `Solve`
 - ▶ Posso usare la classe `Backtrack`
- Devo definire solo
 - ▶ `Come è fatto lo stato`
 - ▶ `Costruttore`
 - ▶ `initState`
 - ▶ `setVisited`
 - ▶ `nextAdmMove`
 - ▶ `makeMove`
 - ▶ `isFinal`

N regine

Lo stato

Osservazione: una riga può contenere una sola regina

- uno stato consiste di (q, nr)
 - ▶ lista q di N interi $q[0], \dots, q[N-1]$
 - ★ $q[i]$ è la colonna in cui si trova la regina della riga i
 - ▶ indice nr della prossima riga senza regina

N regine – I Metodi

- Stato iniziale

$$q = [\text{None}, \dots, \text{None}], \text{nr} = 0$$

N regine – I Metodi

- Stato iniziale

$$q = [\text{None}, \dots, \text{None}], \text{nr} = 0$$

- $\text{makeMove}(c) : q[\text{nr}] = c; \text{nr}++$

N regine – I Metodi

- Stato iniziale

$$q = [\text{None}, \dots, \text{None}], \text{nr} = 0$$

- $\text{makeMove}(c) : q[\text{nr}] = c; \text{nr}++$

- $\text{isFinal} : \text{nr} == N$

nextAdmMove

Siamo in stato (q, nr)

- prova tutte le colonne in riga nr :

$$q[nr] + 1, \dots, N - 1$$

e restituisce la prima di queste colonne che permette di piazzare una regina senza attaccare quelle che abbiamo messo nelle righe $0, \dots, nr - 1$

nextAdmMove

Siamo in stato (q, nr)

- prova tutte le colonne in riga nr :

$$q[nr] + 1, \dots, N - 1$$

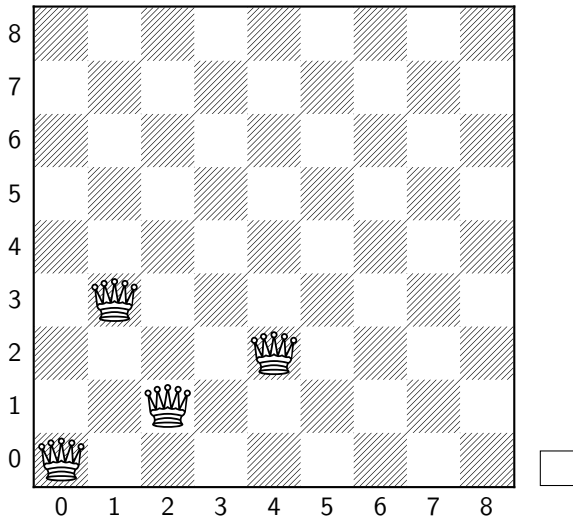
e restituisce la prima di queste colonne che permette di piazzare una regina senza attaccare quelle che abbiamo messo nelle righe

$0, \dots, nr - 1$

- ▶ le regine sono a

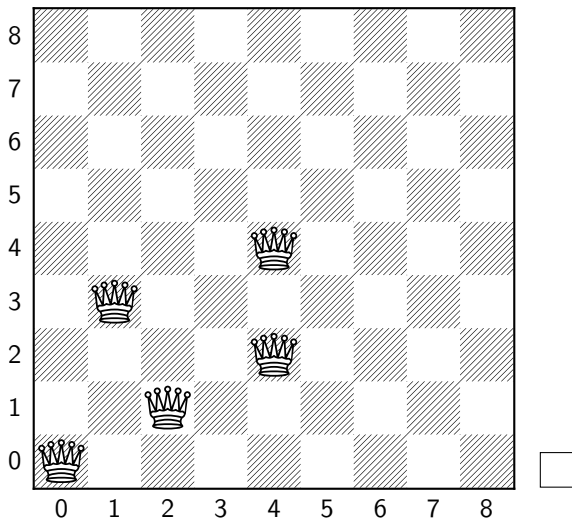
$$(0, q[0]), (1, q[1]), \dots, (nr - 1, q[nr - 1])$$

Stato: $q = [0, 2, 4, 1, 3, \text{None}, \text{None}, \text{None}, \text{None}]$, $\text{nr} = 4$



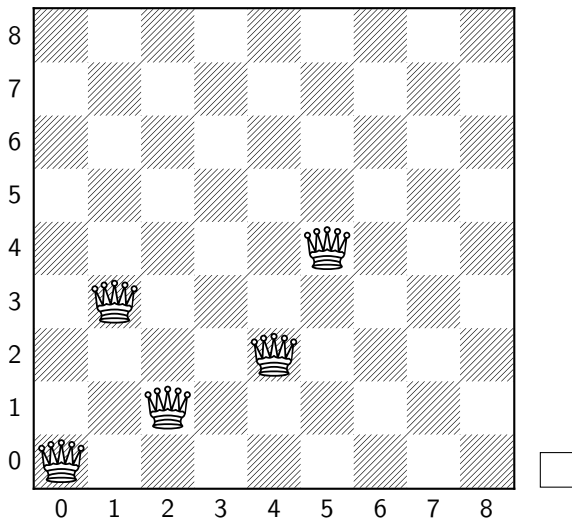
Stato: $q = [0, 2, 4, 1, 3, \text{None}, \text{None}, \text{None}, \text{None}]$, $\text{nr} = 4$

Prova colonna 4 per riga 4 – Attacca regina in riga 2



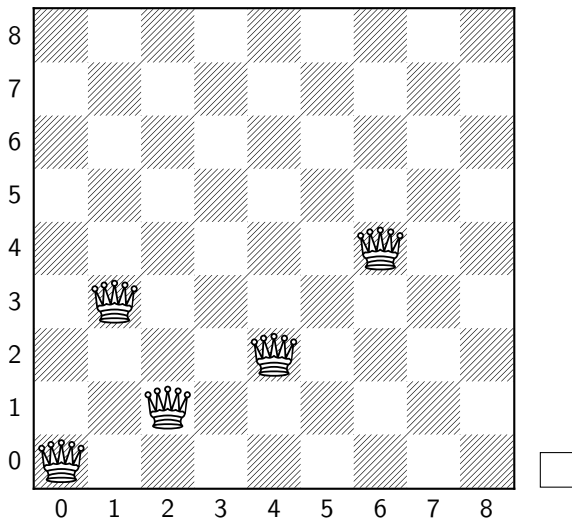
Stato: $q = [0, 2, 4, 1, 3, \text{None}, \text{None}, \text{None}, \text{None}]$, $nr = 4$

Prova colonna 5 per riga 4 – Attacca regina in riga 1



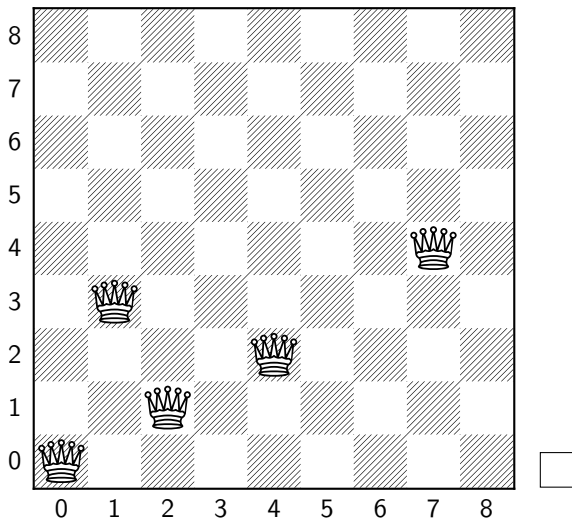
Stato: $q = [0, 2, 4, 1, 3, \text{None}, \text{None}, \text{None}, \text{None}]$, $\text{nr} = 4$

Prova colonna 6 per riga 4 – Attacca regina in riga 2



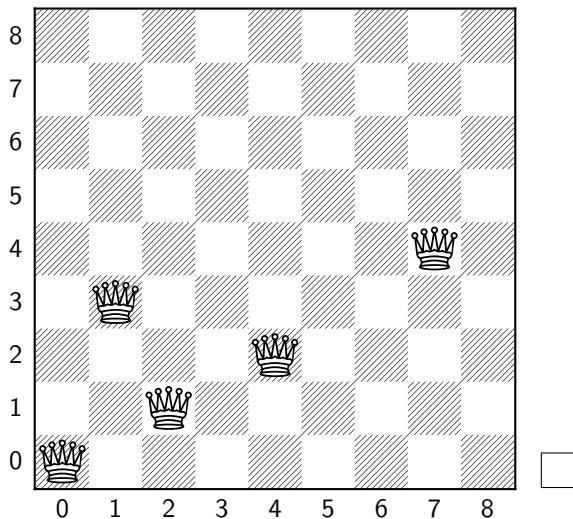
Stato: $q = [0, 2, 4, 1, 3, \text{None}, \text{None}, \text{None}, \text{None}]$, $\text{nr} = 4$

Prova colonna 7 per riga 4 – Nessun attacco

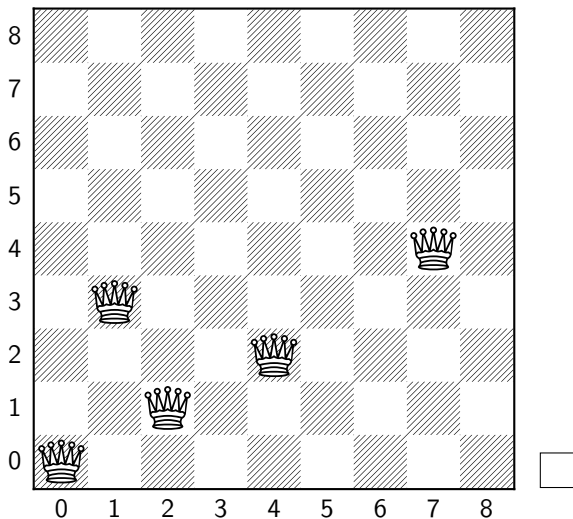


Nuovo Stato= $q = [0, 2, 4, 1, 7, \text{None}, \text{None}, \text{None}, \text{None}]$,

$nr = 5$



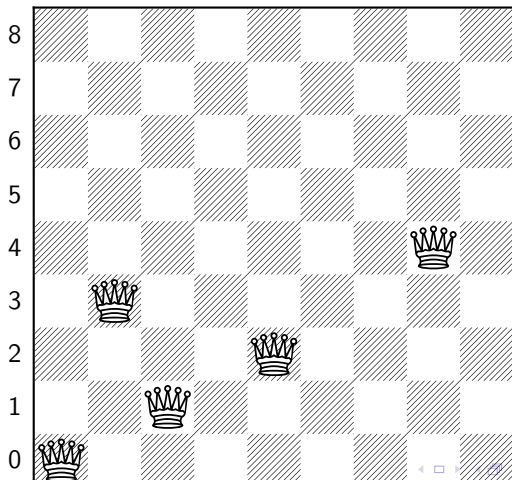
nextAdmMove($q = [0, 2, 4, 1, 3, \text{None}, \text{None}, \text{None}, \text{None}]$, $\text{nr} = 4$) = 7



makeMove($(q = [0, 2, 4, 1, 3, \text{None}, \text{None}, \text{None}, \text{None}], \text{nr} = 4), 7)$)



$(q = [0, 2, 4, 1, 7, \text{None}, \text{None}, \text{None}, \text{None}], \text{nr} = 5)$



Controllare se due regine si attaccano

Regine a (r_1, c_1) e (r_2, c_2)

- controlla se sono sulla stessa riga:

Controllare se due regine si attaccano

Regine a (r_1, c_1) e (r_2, c_2)

- controlla se sono sulla stessa riga:
 - ▶ $r_1 == r_2$

Controllare se due regine si attaccano

Regine a (r_1, c_1) e (r_2, c_2)

- controlla se sono sulla stessa riga:
 - ▶ $r_1 == r_2$
- controlla se sono sulla stessa colonna:

Controllare se due regine si attaccano

Regine a (r_1, c_1) e (r_2, c_2)

- controlla se sono sulla stessa riga:
 - ▶ $r_1 == r_2$
- controlla se sono sulla stessa colonna:
 - ▶ $c_1 == c_2$

Controllare se due regine si attaccano

Regine a (r_1, c_1) e (r_2, c_2)

- controlla se sono sulla stessa riga:
 - ▶ $r_1 == r_2$
- controlla se sono sulla stessa colonna:
 - ▶ $c_1 == c_2$
- controlla se sono sulla stessa diagonale maggiore:

Controllare se due regine si attaccano

Regine a (r_1, c_1) e (r_2, c_2)

- controlla se sono sulla stessa riga:
 - ▶ $r_1 == r_2$
- controlla se sono sulla stessa colonna:
 - ▶ $c_1 == c_2$
- controlla se sono sulla stessa diagonale maggiore:
 - ▶ $c_1 - r_1 == c_2 - r_2$

Controllare se due regine si attaccano

Regine a (r_1, c_1) e (r_2, c_2)

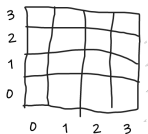
- controlla se sono sulla stessa riga:
 - ▶ $r_1 == r_2$
- controlla se sono sulla stessa colonna:
 - ▶ $c_1 == c_2$
- controlla se sono sulla stessa diagonale maggiore:
 - ▶ $c_1 - r_1 == c_2 - r_2$
- controlla se sono sulla stessa diagonale minore:

Controllare se due regine si attaccano

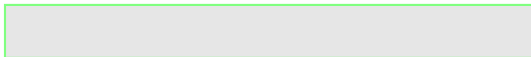
Regine a (r_1, c_1) e (r_2, c_2)

- controlla se sono sulla stessa riga:
 - ▶ $r_1 == r_2$
- controlla se sono sulla stessa colonna:
 - ▶ $c_1 == c_2$
- controlla se sono sulla stessa diagonale maggiore:
 - ▶ $c_1 - r_1 == c_2 - r_2$
- controlla se sono sulla stessa diagonale minore:
 - ▶ $c_1 + r_1 == c_2 + r_2$

Esecuzione per $N = 4$

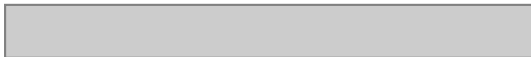


Stato corrente

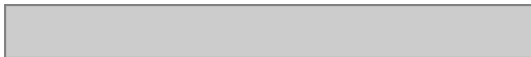


Prossima mossa:

Stack[3]



Stack[2]



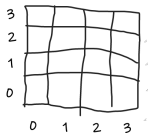
Stack[1]



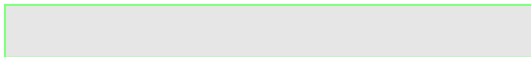
Stack[0]



Esecuzione per $N = 4$

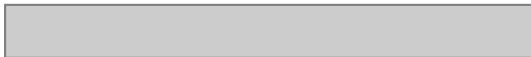


Stato corrente

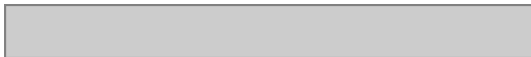


Prossima mossa:

Stack[3]



Stack[2]



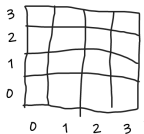
Stack[1]



Stack[0]

[[[None, None, None, None], 0], None]

Esecuzione per $N = 4$



Stato corrente

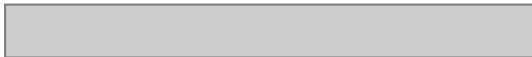
`[None, None, None, None], nr = 0, move = None`

Prossima mossa:

Stack[3]



Stack[2]



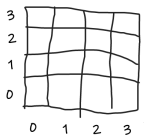
Stack[1]



Stack[0]



Esecuzione per $N = 4$

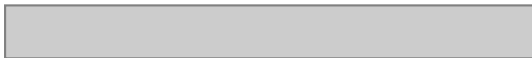


Stato corrente

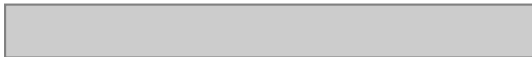
`[None, None, None, None], nr = 0, move = None`

Prossima mossa: 0

Stack[3]



Stack[2]



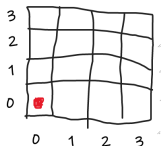
Stack[1]



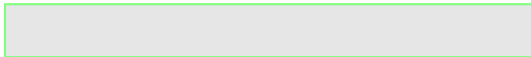
Stack[0]



Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]



Stack[2]



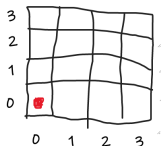
Stack[1]



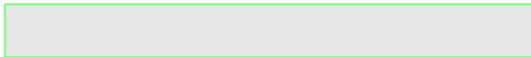
Stack[0]

[[[0, None, None, None], 0], 0]

Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]



Stack[2]



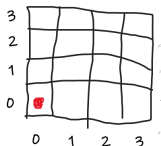
Stack[1]

[[[0, None, None, None], 1], None]

Stack[0]

[[[0, None, None, None], 0], 0]

Esecuzione per $N = 4$



Stato corrente

`[0, None, None, None], nr = 1, move = None`

Prossima mossa:

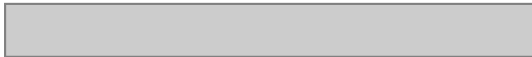
Stack[3]



Stack[2]



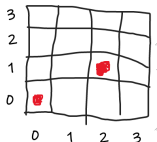
Stack[1]



Stack[0]

`[[[0, None, None, None], 0], 0]`

Esecuzione per $N = 4$

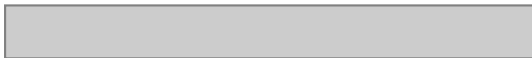


Stato corrente

`[0, None, None, None], nr = 1, move = None`

Prossima mossa: 2

Stack[3]



Stack[2]



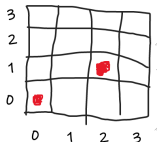
Stack[1]



Stack[0]

`[[[0, None, None, None], 0], 0]`

Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]



Stack[2]

[[[0, 2, None, None], 2], None]

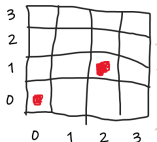
Stack[1]

[[[0, 2, None, None], 1], 2]

Stack[0]

[[[0, None, None, None], 0], 0]

Esecuzione per $N = 4$



Stato corrente

`[0, 2, None, None], nr = 2, move = None`

Prossima mossa:

Stack[3]



Stack[2]



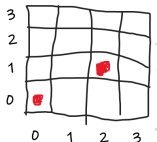
Stack[1]

`[[[0, 2, None, None], 1], 2]`

Stack[0]

`[[[0, None, None, None], 0], 0]`

Esecuzione per $N = 4$



Stato corrente

`[0, 2, None, None], nr = 2, move = None`

Prossima mossa: None

Stack[3]



Stack[2]



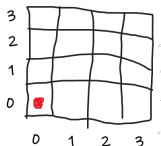
Stack[1]

`[[[0, 2, None, None], 1], 2]`

Stack[0]

`[[[0, None, None, None], 0], 0]`

Esecuzione per $N = 4$



Stato corrente

$[0, 2, \text{None}, \text{None}], \text{nr} = 1, \text{move} = 2$

Prossima mossa:

Stack[3]



Stack[2]



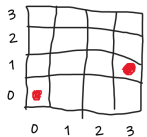
Stack[1]



Stack[0]

$[[[0, \text{None}, \text{None}, \text{None}], 0], 0]$

Esecuzione per $N = 4$



Stato corrente

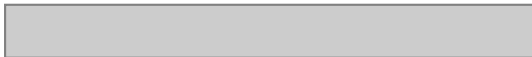
$[0, 2, \text{None}, \text{None}], \text{nr} = 1, \text{move} = 2$

Prossima mossa: 3

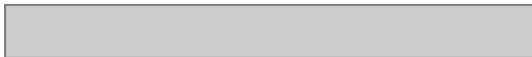
Stack[3]



Stack[2]



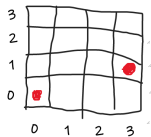
Stack[1]



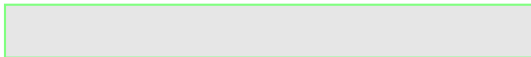
Stack[0]

$[[[0, \text{None}, \text{None}, \text{None}], 0], 0]$

Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]



Stack[2]

[[[0, 3, None, None], 2], None]

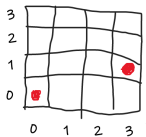
Stack[1]

[[[0, 3, None, None], 1], 3]

Stack[0]

[[[0, None, None, None], 0], 0]

Esecuzione per $N = 4$



Stato corrente

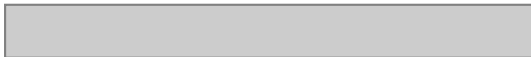
`[0, 3, None, None], nr = 2, move = None`

Prossima mossa:

Stack[3]



Stack[2]



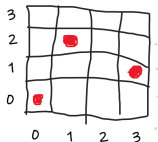
Stack[1]

`[[[0, 3, None, None], 1], 3]`

Stack[0]

`[[[0, None, None, None], 0], 0]`

Esecuzione per $N = 4$



Stato corrente

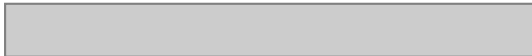
`[0, 3, None, None], nr = 2, move = None`

Prossima mossa: 1

Stack[3]



Stack[2]



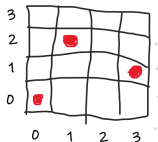
Stack[1]

`[[[0, 2, None, None], 1], 2]`

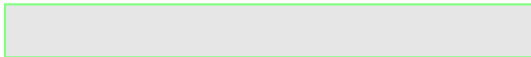
Stack[0]

`[[[0, None, None, None], 0], 0]`

Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]

[[[0, 3, 1, None], 3], None]

Stack[2]

[[[0, 3, 1, None], 2], 1]

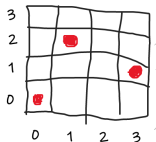
Stack[1]

[[[0, 3, None, None], 1], 3]

Stack[0]

[[[0, None, None, None], 0], 0]

Esecuzione per $N = 4$



Stato corrente

$[0, 3, 1, \text{None}]$, $\text{nr} = 3$, $\text{move} = \text{None}$

Prossima mossa:

Stack[3]

Stack[2]

Stack[1]

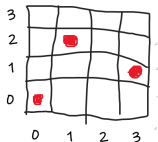
Stack[0]

$[[[0, 3, 1, \text{None}], 2], 1]$

$[[[0, 3, \text{None}, \text{None}], 1], 3]$

$[[[0, \text{None}, \text{None}, \text{None}], 0], 0]$

Esecuzione per $N = 4$



Stato corrente

$[0, 3, 1, \text{None}]$, $\text{nr} = 3$, $\text{move} = \text{None}$

Prossima mossa: None

Stack[3]

Stack[2]

Stack[1]

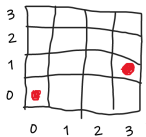
Stack[0]

$[[[0, 3, 1, \text{None}], 2], 1]$

$[[[0, 3, \text{None}, \text{None}], 1], 3]$

$[[[0, \text{None}, \text{None}, \text{None}], 0], 0]$

Esecuzione per $N = 4$



Stato corrente

$[0, 3, 1, \text{None}], \text{nr} = 2, \text{move} = 1$

Prossima mossa:

Stack[3]



Stack[2]



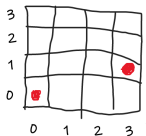
Stack[1]

$[[[0, 3, \text{None}, \text{None}], 1], 3]$

Stack[0]

$[[[0, \text{None}, \text{None}, \text{None}], 0], 0]$

Esecuzione per $N = 4$



Stato corrente

$[0, 3, 1, \text{None}], \text{nr} = 2, \text{move} = 3$

Prossima mossa: None

Stack[3]



Stack[2]



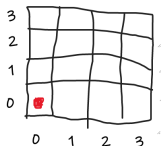
Stack[1]

$[[[0, 3, \text{None}, \text{None}], 1], 3]$

Stack[0]

$[[[0, \text{None}, \text{None}, \text{None}], 0], 0]$

Esecuzione per $N = 4$



Stato corrente

$[0, 3, \text{None}, \text{None}], \text{nr} = 1, \text{move} = 3$

Prossima mossa:

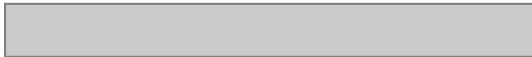
Stack[3]



Stack[2]



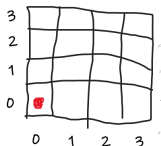
Stack[1]



Stack[0]

$[[[0, \text{None}, \text{None}, \text{None}], 0], 0]$

Esecuzione per $N = 4$



Stato corrente

$[0, 3, \text{None}, \text{None}], \text{nr} = 1, \text{move} = 3$

Prossima mossa: None

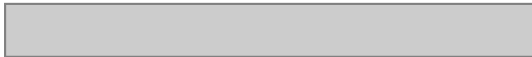
Stack[3]



Stack[2]



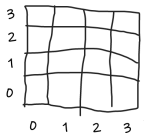
Stack[1]



Stack[0]

$[[[0, \text{None}, \text{None}, \text{None}], 0], 0]$

Esecuzione per $N = 4$

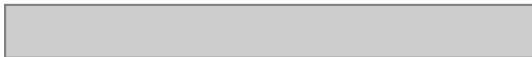


Stato corrente

`[0, None, None, None], nr = 0, move = 0`

Prossima mossa:

Stack[3]



Stack[2]



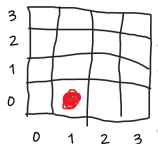
Stack[1]



Stack[0]



Esecuzione per $N = 4$



Stato corrente

`[0, None, None, None], nr = 0, move = 0`

Prossima mossa: 1

Stack[3]



Stack[2]



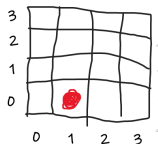
Stack[1]



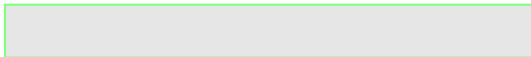
Stack[0]



Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]



Stack[2]



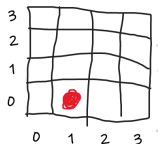
Stack[1]

[[[1, None, None, None], 1], None]

Stack[0]

[[[1, None, None, None], 0], 1]

Esecuzione per $N = 4$



Stato corrente

`[1, None, None, None], nr = 1, move = None`

Prossima mossa:

Stack[3]

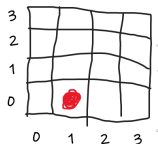
Stack[2]

Stack[1]

Stack[0]

`[[[1, None, None, None], 0], 1]`

Esecuzione per $N = 4$



Stato corrente

`[1, None, None, None], nr = 1, move = None`

Prossima mossa: 3

Stack[3]



Stack[2]



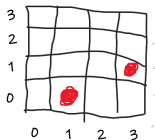
Stack[1]



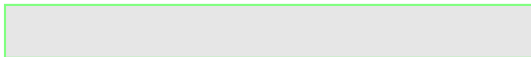
Stack[0]

`[[[1, None, None, None], 0], 1]`

Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]



Stack[2]

[[[1, 3, None, None], 2], None]

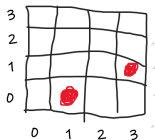
Stack[1]

[[[1, 3, None, None], 1], 3]

Stack[0]

[[[1, None, None, None], 0], 1]

Esecuzione per $N = 4$



Stato corrente

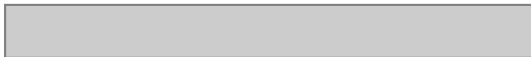
`[1, 3, None, None], nr = 2, move = None`

Prossima mossa:

Stack[3]



Stack[2]



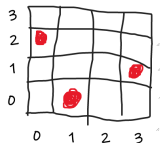
Stack[1]

`[[[1, 3, None, None], 1], 3]`

Stack[0]

`[[[1, None, None, None], 0], 1]`

Esecuzione per $N = 4$

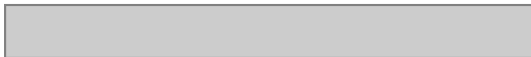


Stato corrente

`[1, 3, None, None], nr = 2, move = None`

Prossima mossa: 0

Stack[3]



Stack[2]



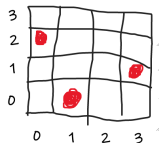
Stack[1]

`[[[1, 3, None, None], 1], 3]`

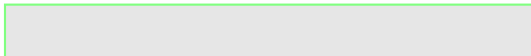
Stack[0]

`[[[1, None, None, None], 0], 1]`

Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]

[[[1, 3, 0, None], 3], None]

Stack[2]

[[[1, 3, 0, None], 2], 0]

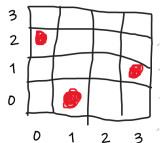
Stack[1]

[[[1, 3, None, None], 1], 3]

Stack[0]

[[[1, None, None, None], 0], 1]

Esecuzione per $N = 4$



Stato corrente

$[1, 3, 0, \text{None}]$, $\text{nr} = 3$, $\text{move} = \text{None}$

Prossima mossa:

Stack[3]

Stack[2]

Stack[1]

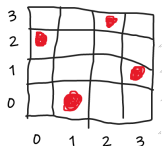
Stack[0]

$[[[1, 3, 0, \text{None}], 2], 0]$

$[[[1, 3, \text{None}, \text{None}], 1], 3]$

$[[[1, \text{None}, \text{None}, \text{None}], 0], 1]$

Esecuzione per $N = 4$



Stato corrente

$[1, 3, 0, \text{None}], \text{nr} = 3, \text{move} = \text{None}$

Prossima mossa: 2

Stack[3]

Stack[2]

Stack[1]

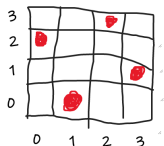
Stack[0]

$[[[1, 3, 0, \text{None}], 2], 0]$

$[[[1, 3, \text{None}, \text{None}], 1], 3]$

$[[[1, \text{None}, \text{None}, \text{None}], 0], 1]$

Esecuzione per $N = 4$



Stato corrente



Prossima mossa:

Stack[3]

$[[[1, 3, 0, 2], 3], 2]$

Stack[2]

$[[[1, 3, 0, \text{None}], 2], 0]$

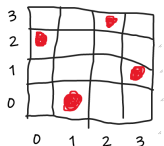
Stack[1]

$[[[1, 3, \text{None}, \text{None}], 1], 3]$

Stack[0]

$[[[1, \text{None}, \text{None}, \text{None}], 0], 1]$

Esecuzione per $N = 4$



Stato corrente

`[[1, 3, 0, 2], 4]`

Prossima mossa:

Stack[3]

`[[[1, 3, 0, 2], 3], 2]`

Stack[2]

`[[[1, 3, 0, None], 2], 0]`

Stack[1]

`[[[1, 3, None, None], 1], 3]`

Stack[0]

`[[[1, None, None, None], 0], 1]`

Subset Sum

Subset Sum

Input:

- Una lista L di N interi positivi
- Un intero positivo target T

Output:

- Un sottoinsieme S degli interi di L la cui somma è uguale a T

Subset Sum

Subset Sum

Input:

- Una lista L di N interi positivi
- Un intero positivo target T

Output:

- Un sottoinsieme S degli interi di L la cui somma è uguale a T

Example

Subset Sum Input:

- $L = [3, 2, 11, 4, 17]$
- $T = 22$

Output:

- $S = \{3, 2, 17\}$

Subset Sum

Subset Sum

Input:

- Una lista L di N interi positivi
- Un intero positivo target T

Output:

- Un sottoinsieme S degli interi di L la cui somma è uguale a T

Example

Subset Sum Input:

- $L = [3, 2, 11, 4, 17]$
- $T = 36$

Output:

- Nessuna soluzione
- $36 = 2 + 2 + 11 + 4 + 17$ non è una soluzione perché 2 è usato due volte

Usiamo Backtrack

Definiamo gli stati

- consideriamo gli interi $L[i]$ della lista L uno per volta, $i = 0, \dots, N - 1$
- i è il prossimo intero da considerare
- abbiamo due mosse possibili per ogni intero $L[i]$
 - ▶ Mossa 0: non aggiungere $L[i]$ a S
 - ▶ Mossa 1: aggiungere $L[i]$ a S

Usiamo Backtrack

Definiamo gli stati

- consideriamo gli interi $L[i]$ della lista L uno per volta, $i = 0, \dots, N - 1$
- i è il prossimo intero da considerare
- abbiamo due mosse possibili per ogni intero $L[i]$
 - ▶ Mossa 0: non aggiungere $L[i]$ a S
 - ▶ Mossa 1: aggiungere $L[i]$ a S

Quindi uno stato consiste di:

- 1 i , indice del prossimo intero della lista L da considerare
- 2 S , che ci dice per ogni intero $L[j]$ della lista L :
 - ▶ $S[j] = \text{None}$: l'intero $L[j]$ non ancora considerato
 - ▶ $S[j] = 0$: l'intero $L[j]$ considerato e scartato
 - ▶ $S[j] = 1$: l'intero $L[j]$ considerato e aggiunto

Example

$(2, [0, 1, \text{None}, \text{None}, \text{None}])$

- abbiamo scartato $L[0] = 3$ ed aggiunto $L[1] = 2$
- dobbiamo ancora decidere su $L[2] = 11$, $L[3] = 4$ e $L[4] = 17$
- prossima mossa:
 - ▶ scartare $L[2]$
- prossimo stato

$(3, [0, 1, 0, \text{None}, \text{None}])$

Example

$(2, [0, 1, 0, \text{None}, \text{None}])$

- abbiamo scartato $L[0] = 3$, aggiunto $L[1] = 2$ e scartato $L[2] = 11$
- dobbiamo ancora decidere su $L[3] = 4$ e $L[4] = 17$
- prossima mossa:
 - ▶ aggiungere $L[2]$
- prossimo stato

$(3, [0, 1, 1, \text{None}, \text{None}])$

Example

(2, [0, 1, 1, None, None])

- abbiamo scartato $L[0] = 3$, aggiunto $L[1] = 2$ e $L[2] = 11$
- dobbiamo ancora decidere su $L[3] = 4$ e $L[4] = 17$
- Non esiste una prossima mossa

Esercizio

Definire una classe **SubsetSum** derivata da **Backtrack** che fornisce

- Costruttore
- initState
- setVisited
- nextAdmMove
- makeMove
- isFinal