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*

Backtrack generale

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.

Backtrack generale

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

Backtrack generale

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:

Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,

Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili

Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*

▸ Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*
     - ★ continue

Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*
     - ★ continue
   - sia `nmove` prossima mossa ammissibile

Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*
     - ★ continue
   - sia `nmove` prossima mossa ammissibile
   - marca $(\texttt{r}, \texttt{c})$ come *visitata*

Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*
     - ★ continue
   - sia `nmove` prossima mossa ammissibile
   - marca $(\texttt{r}, \texttt{c})$ come *visitata*
   - fai push di $(\texttt{r}, \texttt{c}, \texttt{nmove})$

Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*
     - ★ continue
   - sia `nmove` prossima mossa ammissibile
   - marca $(\texttt{r}, \texttt{c})$ come *visitata*
   - fai push di $(\texttt{r}, \texttt{c}, \texttt{nmove})$
   - fai mossa `nmove` e calcola la nuova posizione $(\texttt{newr}, \texttt{newc})$

▸ Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*
     - ★ continue
   - sia `nmove` prossima mossa ammissibile
   - marca $(\texttt{r}, \texttt{c})$ come *visitata*
   - fai push di $(\texttt{r}, \texttt{c}, \texttt{nmove})$
   - fai mossa `nmove` e calcola la nuova posizione $(\texttt{newr}, \texttt{newc})$
   - if $(\texttt{newr}, \texttt{newc}) == E$, return TRUE

Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*
     - ★ continue
   - sia `nmove` prossima mossa ammissibile
   - marca $(\texttt{r}, \texttt{c})$ come *visitata*
   - fai push di $(\texttt{r}, \texttt{c}, \texttt{nmove})$
   - fai mossa `nmove` e calcola la nuova posizione $(\texttt{newr}, \texttt{newc})$
   - if $(\texttt{newr}, \texttt{newc}) == E$, return TRUE
   - fai push di $(\texttt{newr}, \texttt{newc}, 0)$

▸ Backtrack generale

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 $(\texttt{r}, \texttt{c}, \texttt{lmove})$,
   - se non ci sono altre mosse ammissibili
     - ★ marca $(\texttt{r}, \texttt{c})$ come *esaurita*
     - ★ continue
   - sia `nmove` prossima mossa ammissibile
   - marca $(\texttt{r}, \texttt{c})$ come *visitata*
   - fai push di $(\texttt{r}, \texttt{c}, \texttt{nmove})$
   - fai mossa `nmove` e calcola la nuova posizione $(\texttt{newr}, \texttt{newc})$
   - if $(\texttt{newr}, \texttt{newc}) == E$, return TRUE
   - fai push di $(\texttt{newr}, \texttt{newc}, 0)$
5. return FALSE

▸ Backtrack generale

Backtrack

1. Si parte dallo *stato iniziale* `statIn`

Backtrack per Maze

Backtrack

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

▸ Backtrack per Maze

Backtrack

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

▸ Backtrack per Maze

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 per Maze

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 per Maze

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 per Maze

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)` $\leftarrow$ `Stack.Pop`

▸ Backtrack per Maze

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)` $\leftarrow$ `Stack.Pop`
   - Calcola `newMove`

Backtrack per Maze

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)` $\leftarrow$ `Stack.Pop`
   - Calcola `newMove`
     - ★ mossa dopo `move` ammissibile per `state`

▸ Backtrack per Maze

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)` $\leftarrow$ `Stack.Pop`
   - Calcola `newMove`
     - ⋆ mossa dopo `move` ammissibile per `state`
   - if `newMove` $\neq$ `None`

▸ Backtrack per Maze

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)` $\leftarrow$ `Stack.Pop`
   - Calcola `newMove`
     - ★ mossa dopo `move` ammissibile per `state`
   - if `newMove` $\neq$ `None`
     - ★ `Stack.Push(state, newMove)`

▸ Backtrack per Maze

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)` $\leftarrow$ `Stack.Pop`
   - Calcola `newMove`
     - ★ mossa dopo `move` ammissibile per `state`
   - if `newMove` $\neq$ `None`
     - ★ `Stack.Push(state, newMove)`
     - ★ Calcola lo stato `newState` ottenuta dall'eseguire `newMove` in `state`

▸ Backtrack per Maze

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)` $\leftarrow$ `Stack.Pop`
   - Calcola `newMove`
     - ⋆ mossa dopo `move` ammissibile per `state`
   - if `newMove` $\neq$ `None`
     - ⋆ `Stack.Push(state, newMove)`
     - ⋆ Calcola lo stato `newState` ottenuta dall'eseguire `newMove` in `state`
     - ⋆ If `newState` è uno stato **finale**, return `True`

▸ Backtrack per Maze

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)` $\leftarrow$ `Stack.Pop`
   - Calcola `newMove`
     - ★ mossa dopo `move` ammissibile per `state`
   - if `newMove` $\neq$ `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 per Maze

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)` $\leftarrow$ `Stack.Pop`
   - Calcola `newMove`
     - ⋆ mossa dopo `move` ammissibile per `state`
   - if `newMove` $\neq$ `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 per Maze

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 `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
3. Come marcare uno stato $(M, (r, c))$ come già visitato
   - poni $M[r][c] = visitata$
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
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 $M[r][c] = \mathit{finale}$

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

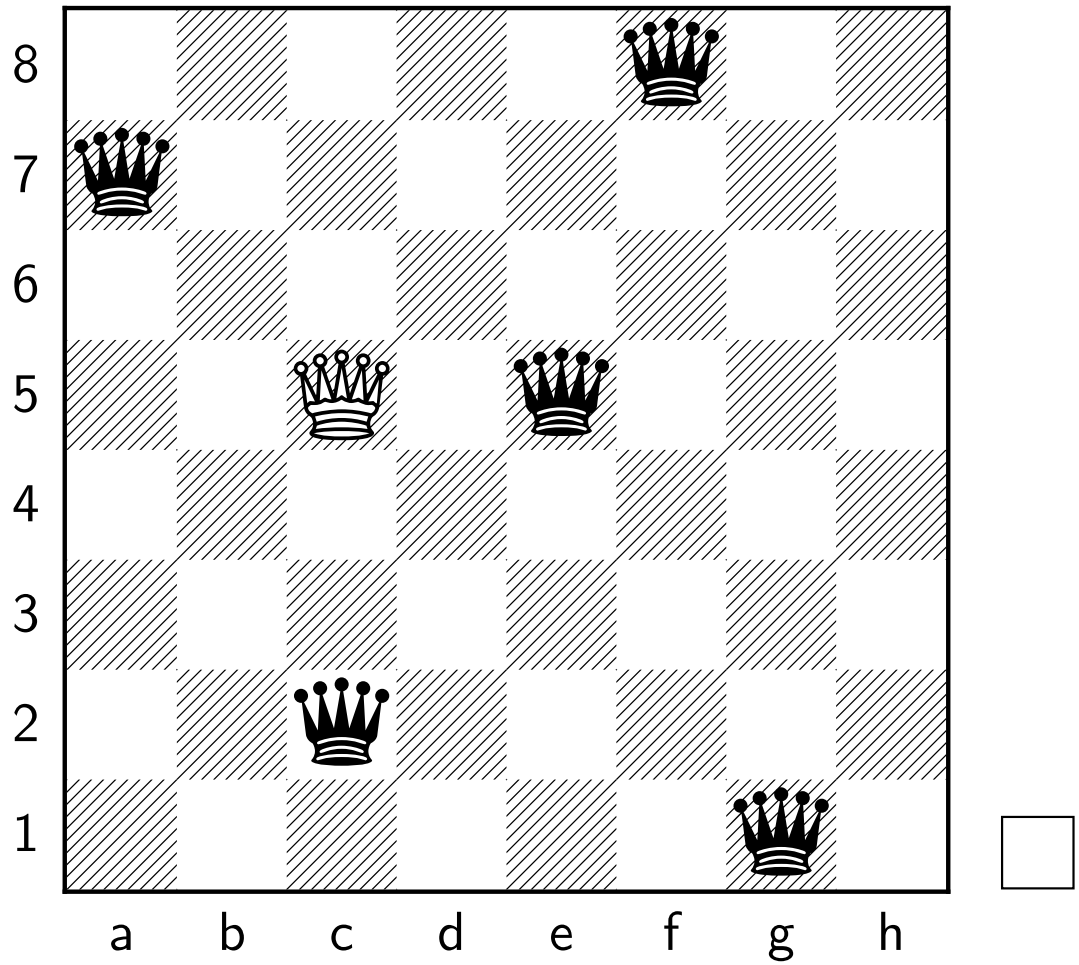
Cosa ci guadagno?

- 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

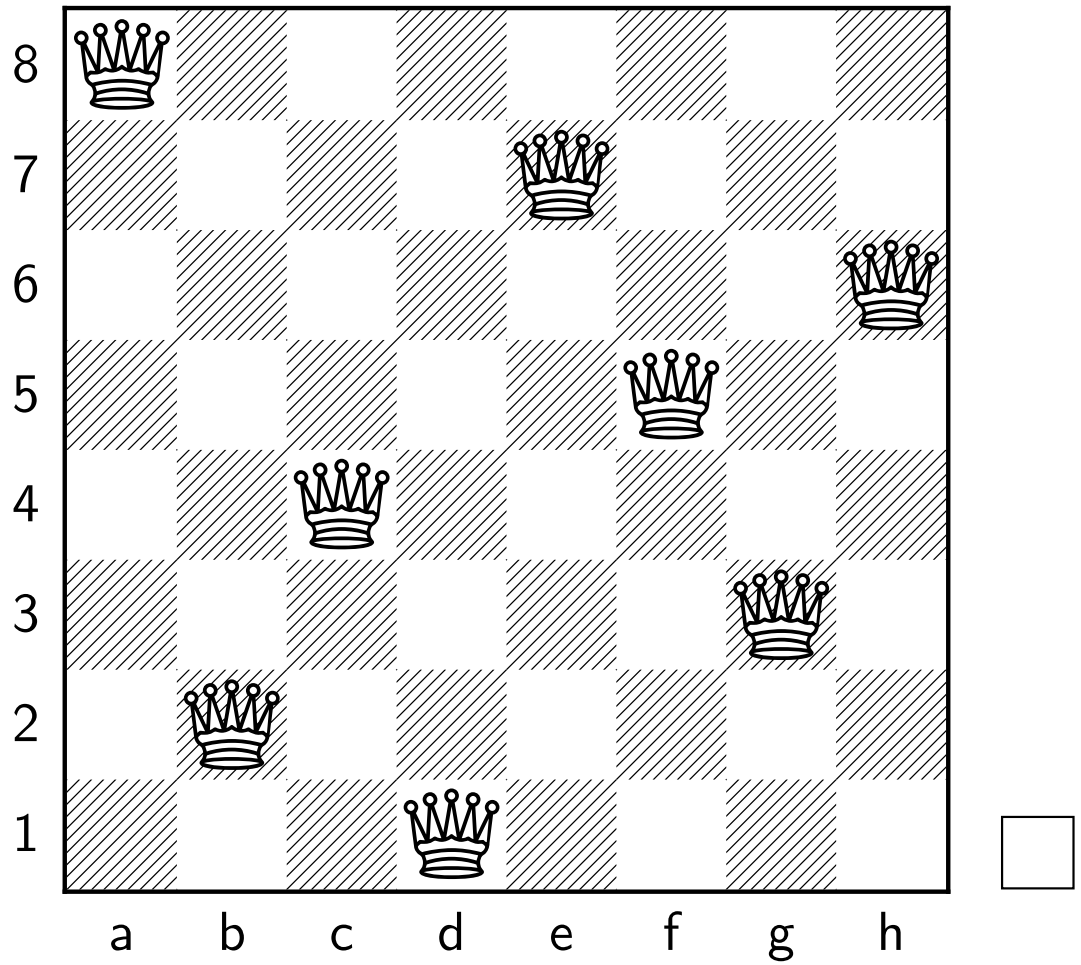
N regine

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 $(q, \texttt{nr})$
  - lista q di N interi $q[0], \ldots, q[N-1]$
    - $q[i]$ è la colonna in cui si trova la regina della riga i
  - indice `nr` della prossima riga senza regina