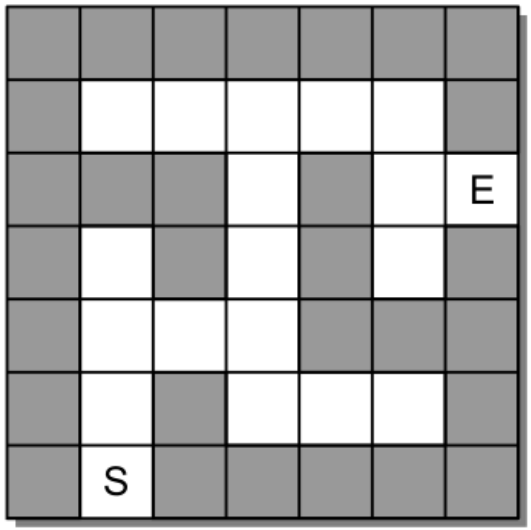
Solving a Maze

Giuseppe Persiano

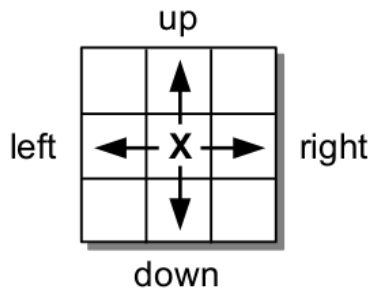
Università di Salerno

Ottobre, 2020

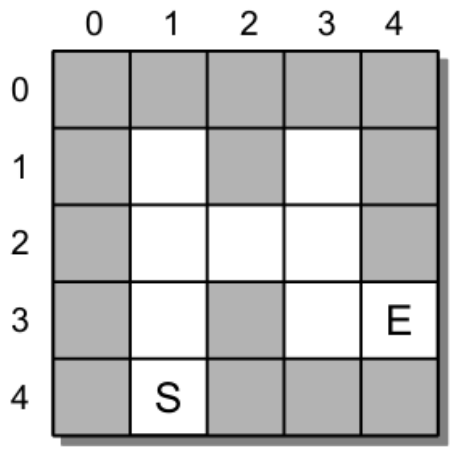
Capitolo 7.4 di
Data Structures and Algorithms Using Python di Rance D. Necaise



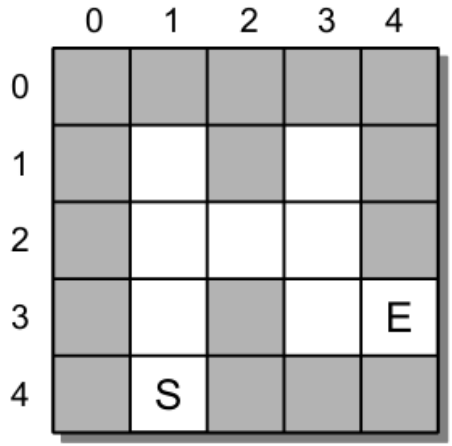
1. Determinare se esiste un cammino da *start* S a *exit* E
2. Descrivere il cammino



Mosse consentite dalla posizione x



Numerazione delle righe e delle colonne



Numerazione delle righe e delle colonne

Trovare un cammino da $S = (4, 1)$ a $E = (3, 4)$

Backtrack

1. si parte dallo *stato* iniziale

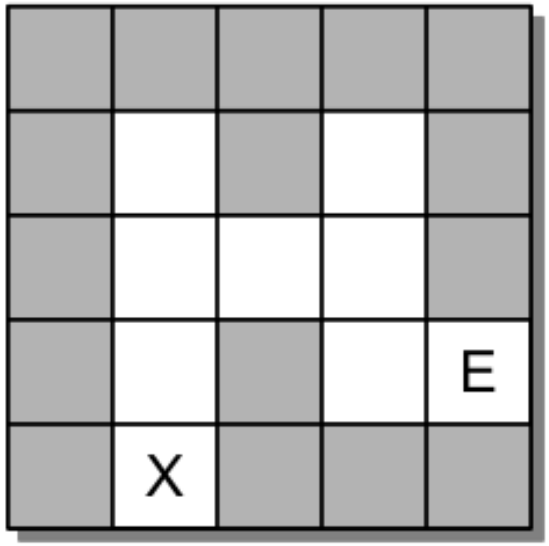
Backtrack

1. si parte dallo *stato* iniziale
2. in ogni stato effettuare le mosse *ammissibili* per lo stato una per volta in un ordine fissato

Backtrack

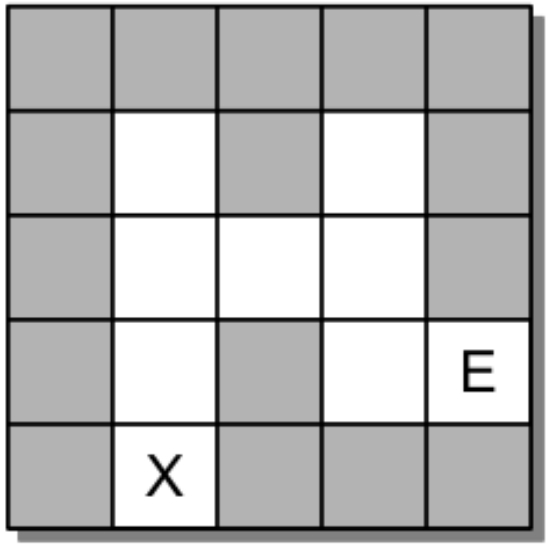
1. si parte dallo *stato* iniziale
2. in ogni stato effettuare le mosse *ammissibili* per lo stato una per volta in un ordine fissato
3. se si arriva in uno *stato* in cui non ci sono più mosse ammissibili, si torna ad uno stato precedente e si prova una mossa non ancora provata

mosse in senso orario: N, E, S, O



si parte dallo stato iniziale

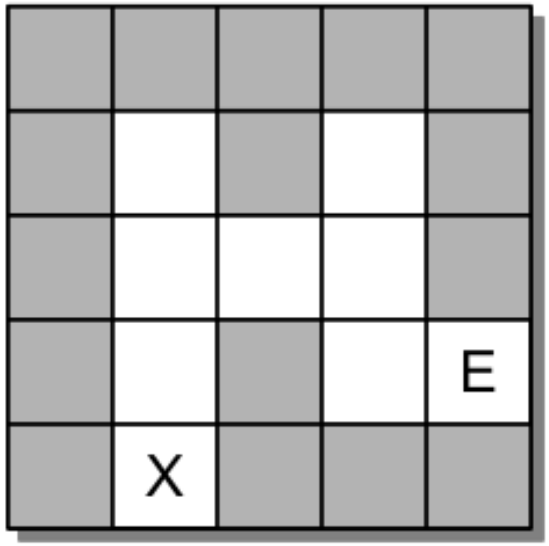
mosse in senso orario: N, E, S, O



si parte dallo stato iniziale

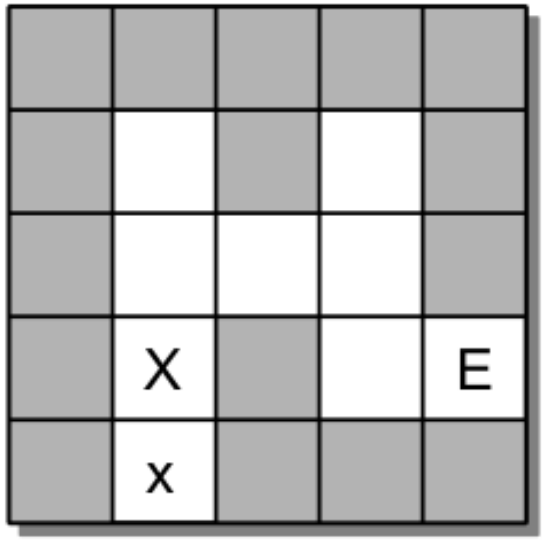
- ogni casella è *percorribile* o *bloccata*

mosse in senso orario: N, E, S, O

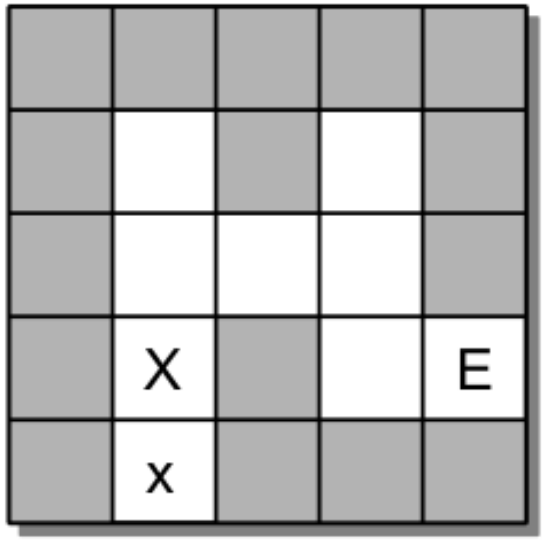


si parte dallo stato iniziale

- ogni casella è *percorribile* o *bloccata*
- la casella di start è *occupata* (X)

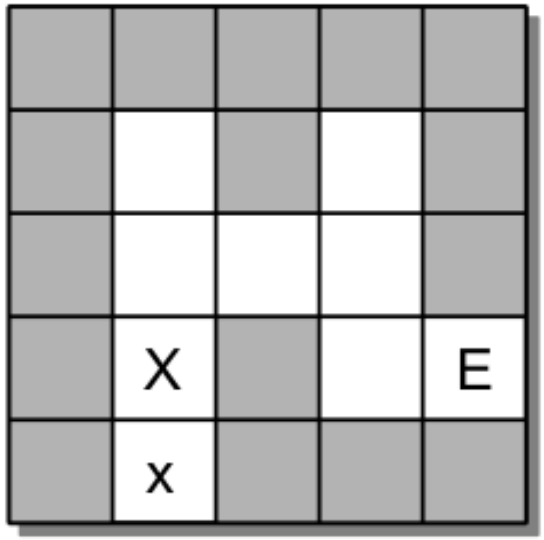


Mossa N – ammissibile



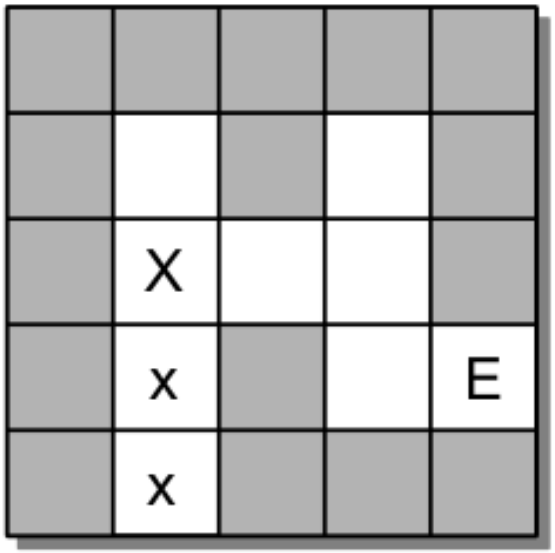
Mossa N – ammissibile

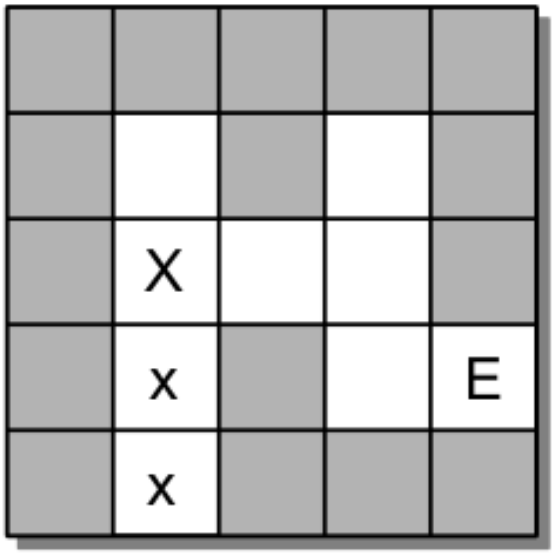
- la casella di start diventa *visitata* (x)



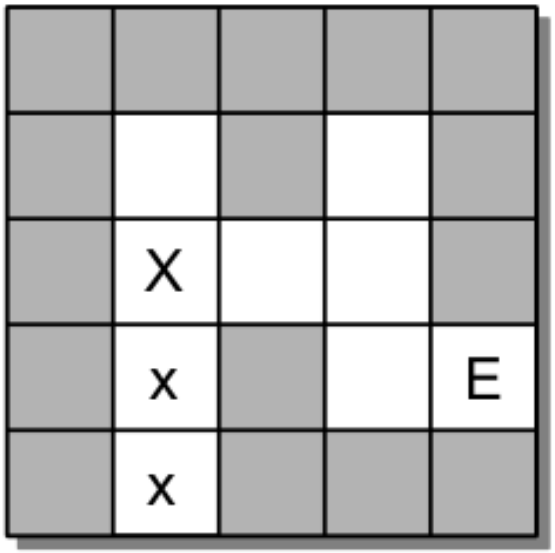
Mossa N – ammissibile

- la casella di start diventa *visitata* (x)
- casella (3,1) diventa *occupata* (X)


X
x
x
E

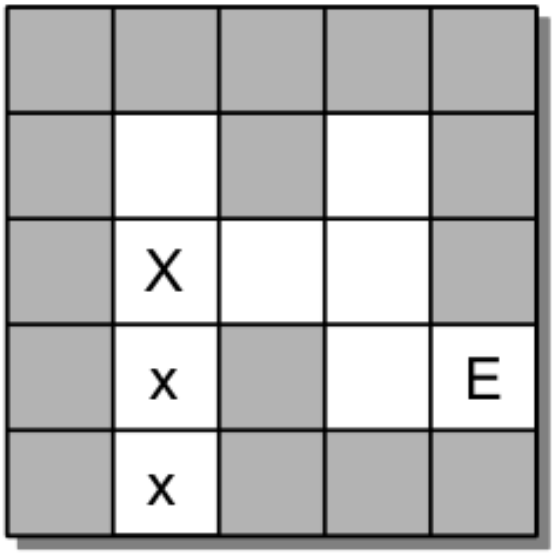


Mossa N – ammissibile



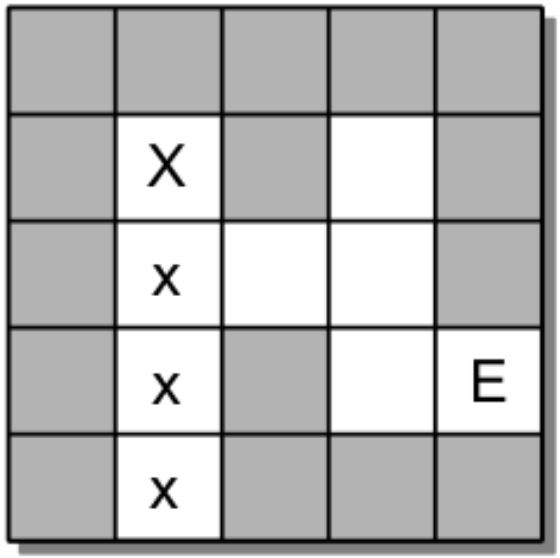
Mossa N – ammissibile

- la casella (3,1) diventa *visitata* (x)

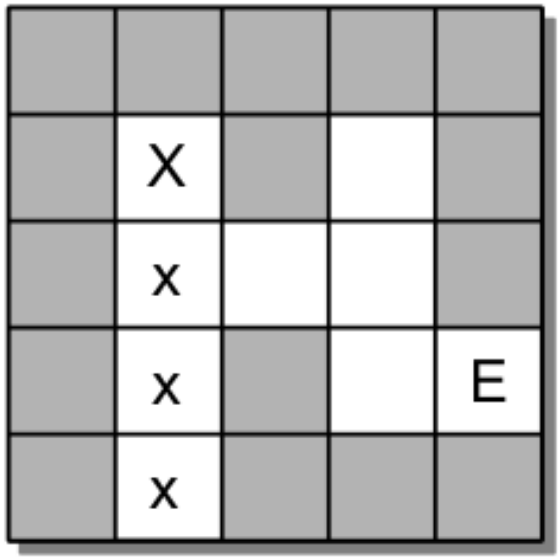


Mossa N – ammissibile

- la casella (3,1) diventa *visitata* (x)
- casella (2,1) diventa *occupata* (X)

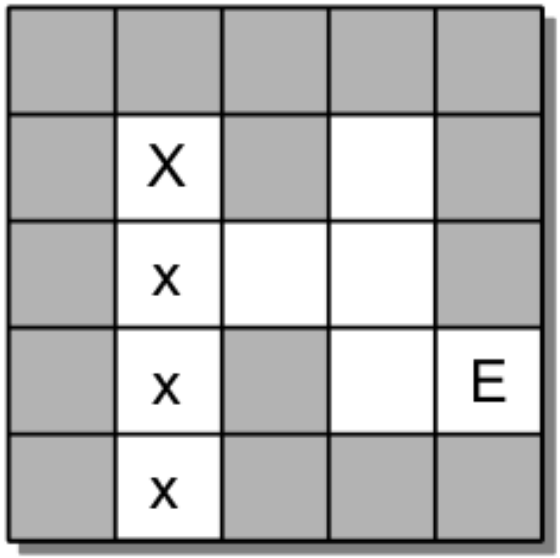


Mossa N – ammissibile



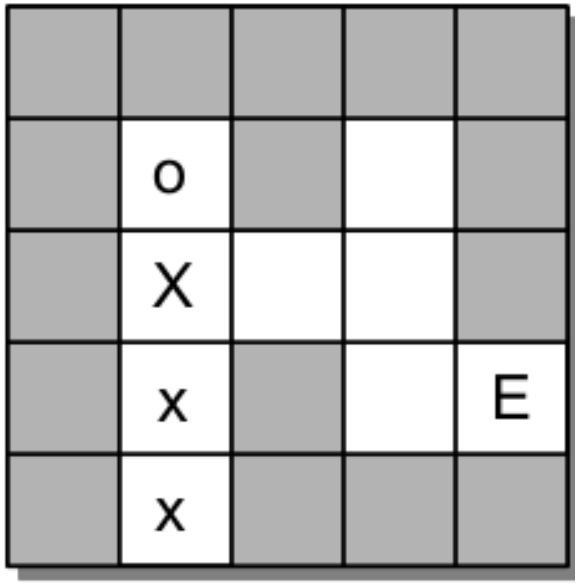
Mossa N – ammissibile

- la casella (2,1) diventa *visitata* (x)

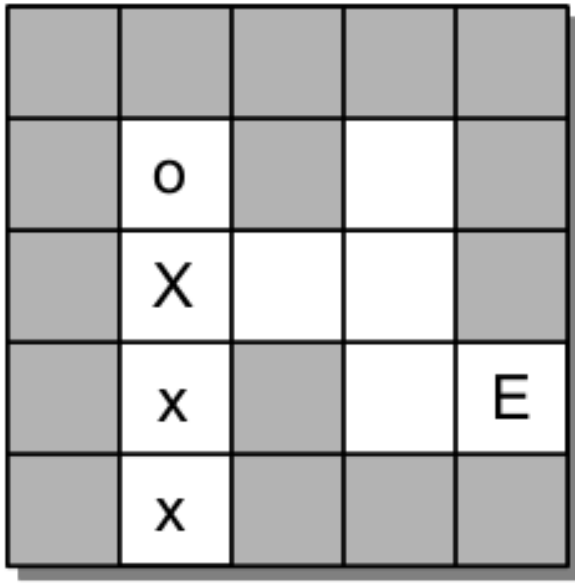


Mossa N – ammissibile

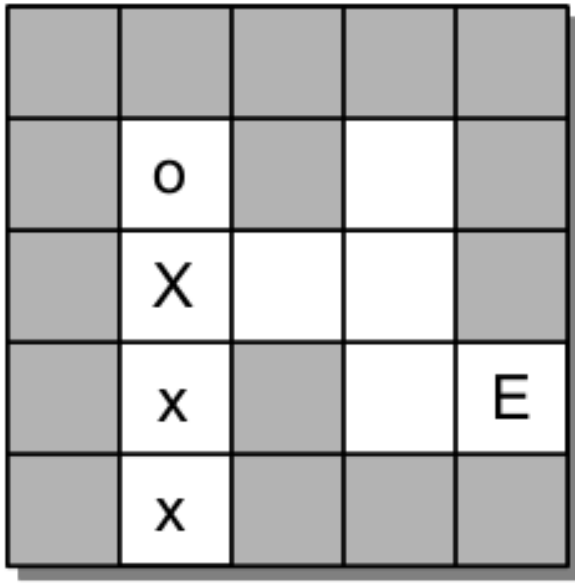
- la casella (2,1) diventa *visitata* (x)
- casella (1,1) diventa *occupata* (X)



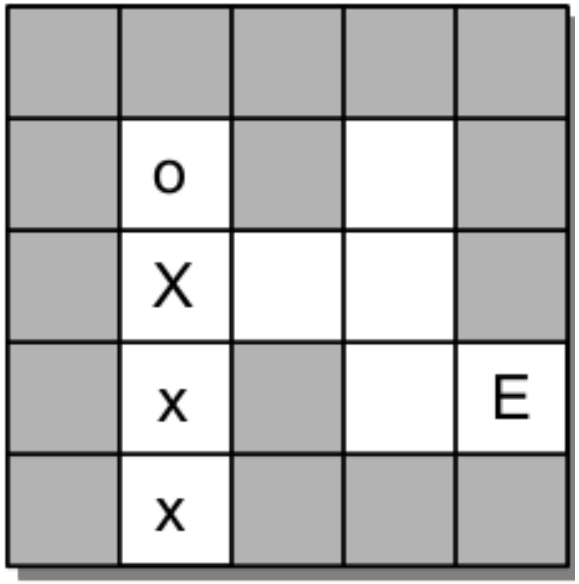
- mossa N – non ammissibile perché bloccata



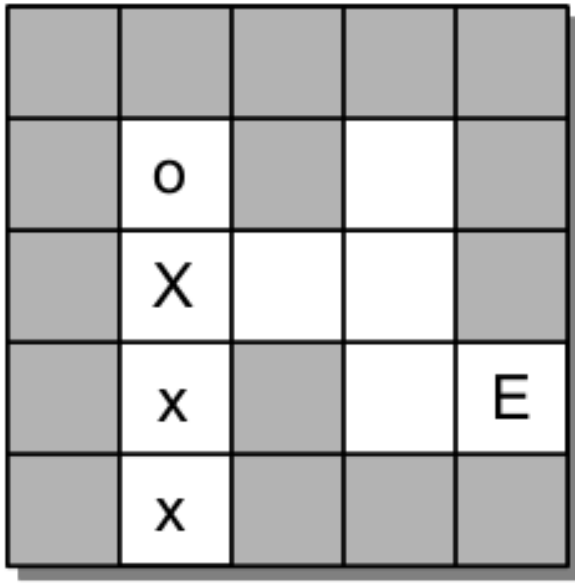
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata



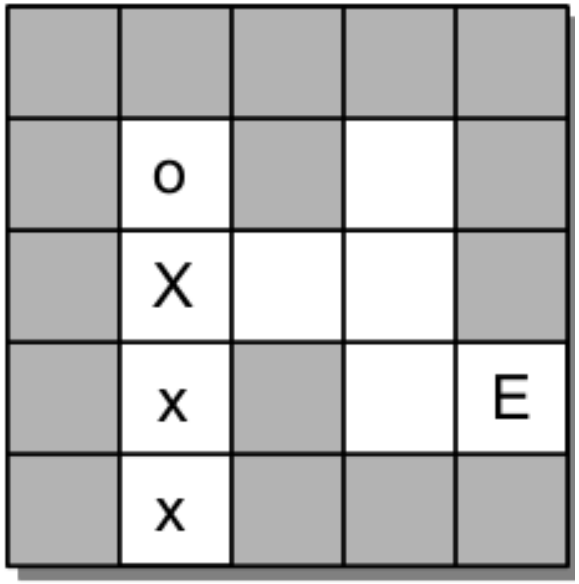
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata
- mossa S – non ammissibile perché già visitata



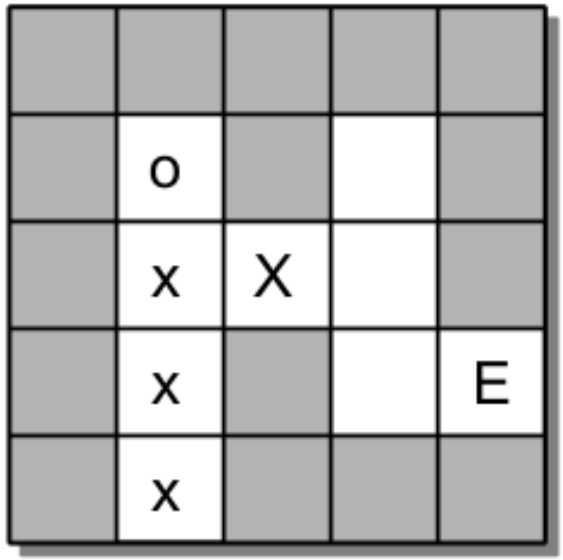
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata
- mossa S – non ammissibile perché già visitata
- mossa O – non ammissibile perché bloccata



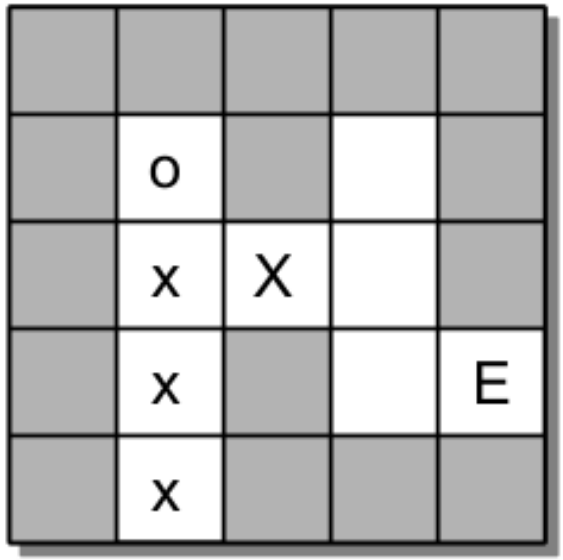
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata
- mossa S – non ammissibile perché già visitata
- mossa O – non ammissibile perché bloccata
- casella (1,1) diventa *esaurita* (o)



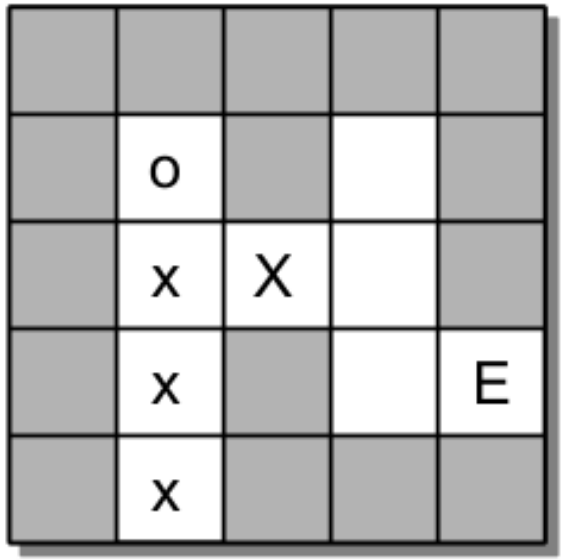
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata
- mossa S – non ammissibile perché già visitata
- mossa O – non ammissibile perché bloccata
- casella (1,1) diventa *esaurita* (o)
- casella (2,1) diventa *occupata* (X)



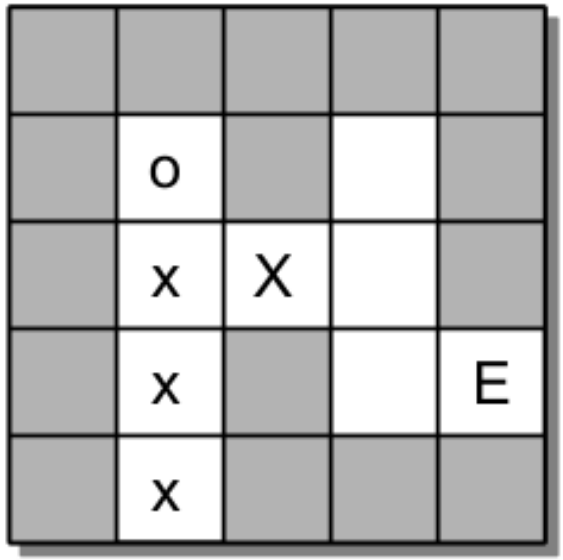
- si torna alla casella precedente (2,1)



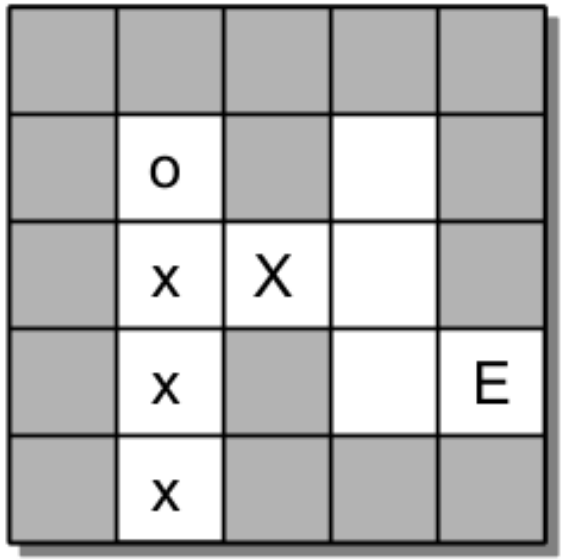
- si torna alla casella precedente (2,1)
- si prova la prossima mossa: E – ammissibile



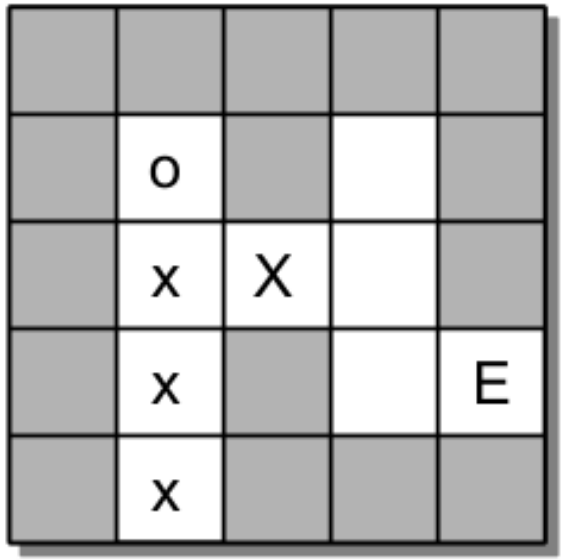
- si torna alla casella precedente (2,1)
- si prova la prossima mossa: E – ammissibile
- Dobbiamo ricordarci



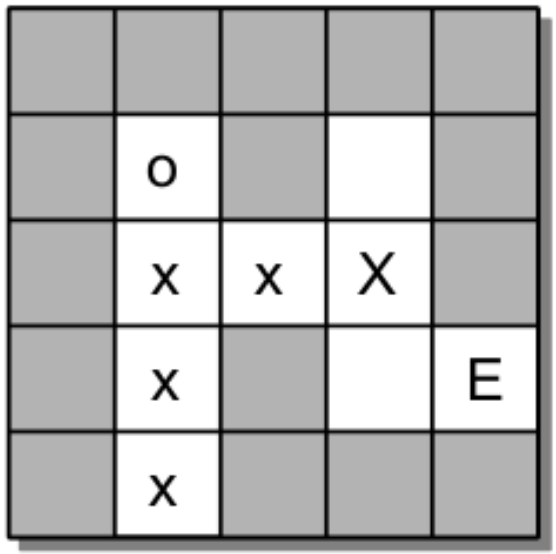
- si torna alla casella precedente (2,1)
- si prova la prossima mossa: E – ammissibile
- Dobbiamo ricordarci
  - ultima casella visitata



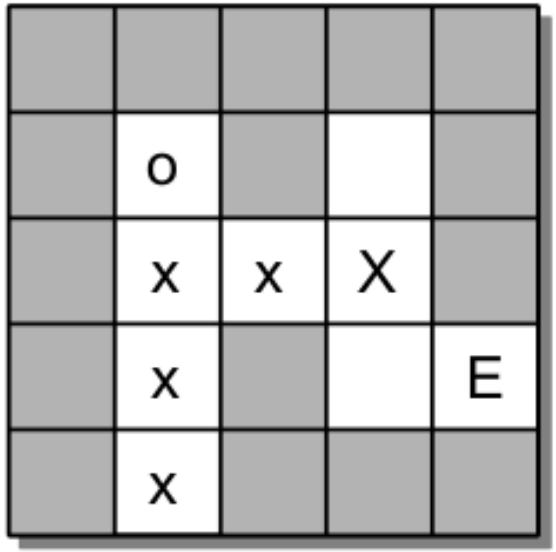
- si torna alla casella precedente (2,1)
- si prova la prossima mossa: E – ammissibile
- Dobbiamo ricordarci
  - ultima casella visitata
  - ultima mossa effettuata



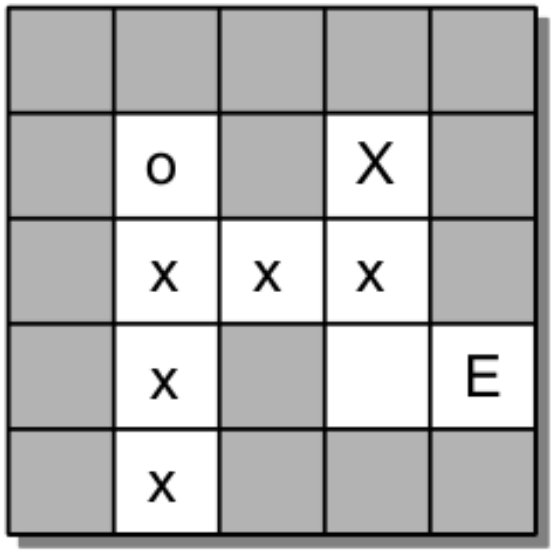
- si torna alla casella precedente (2,1)
- si prova la prossima mossa: E – ammissibile
- Dobbiamo ricordarci
  - ultima casella visitata
  - ultima mossa effettuata
- usiamo uno stack



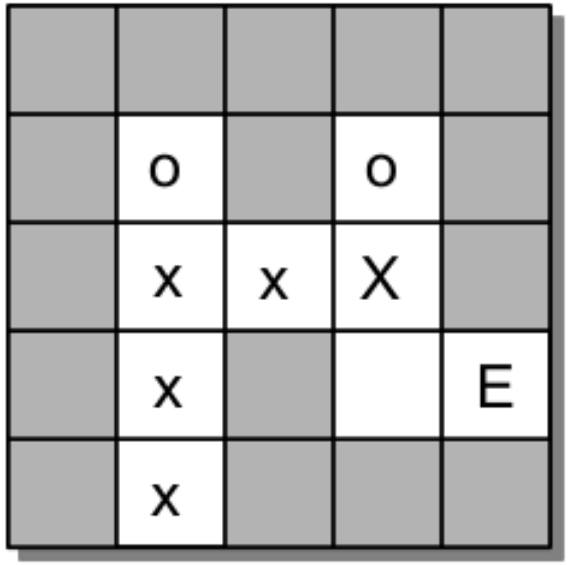
- mossa N – non ammissibile perché bloccata



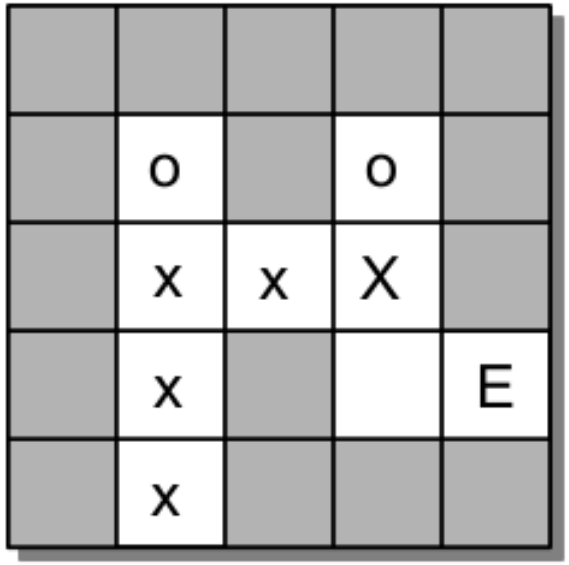
- mossa N – non ammissibile perché bloccata
- mossa E – ammissibile



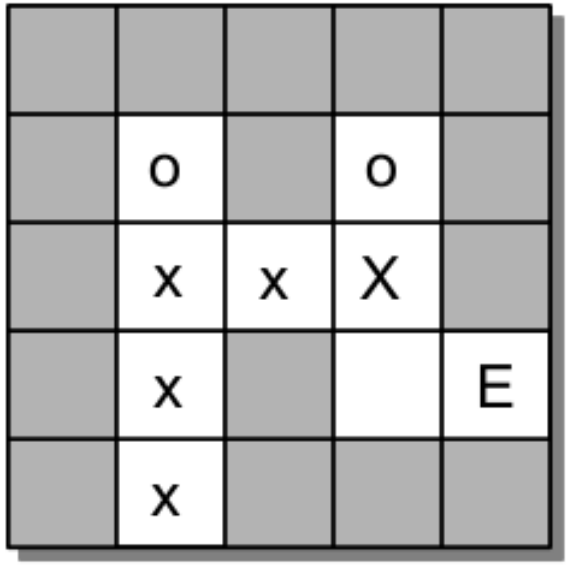
- mossa N – ammissibile



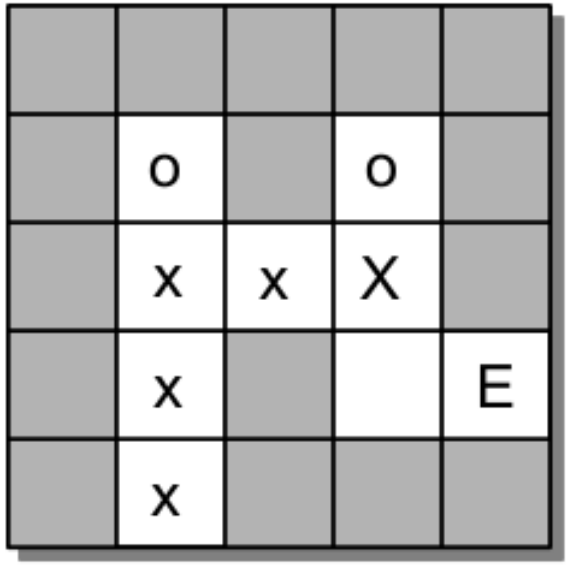
- mossa N – non ammissibile perché bloccata



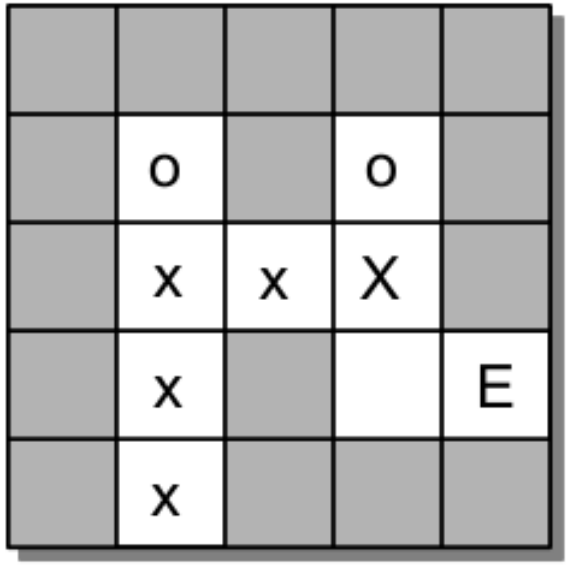
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata



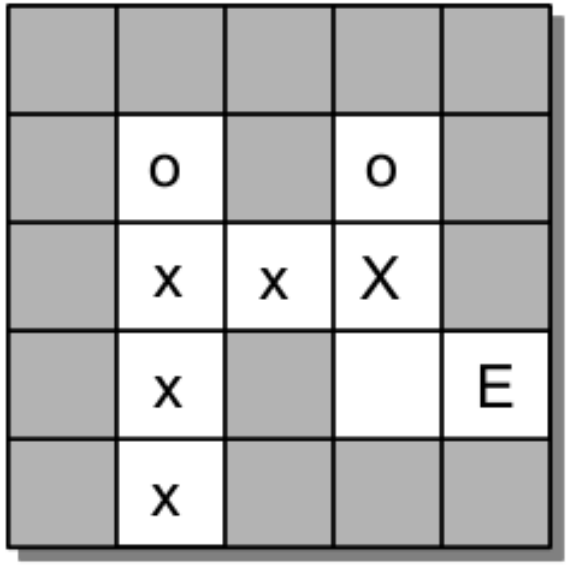
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata
- mossa S – non ammissibile perché visitata



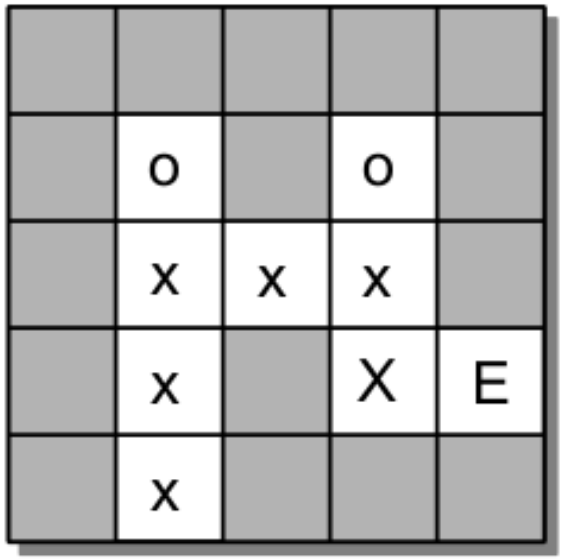
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata
- mossa S – non ammissibile perché visitata
- mossa O – non ammissibile perché bloccata



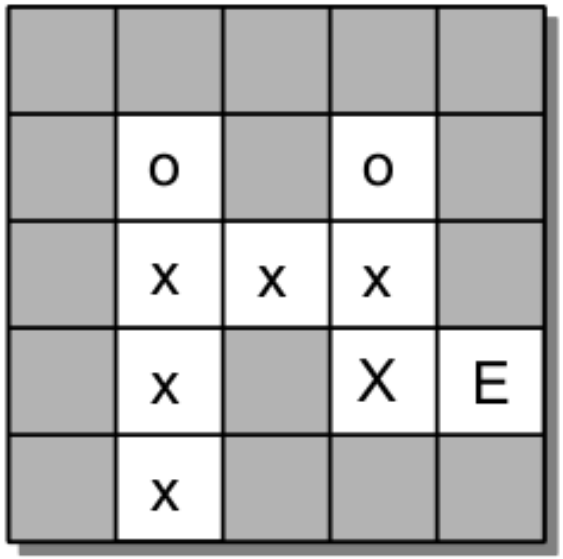
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata
- mossa S – non ammissibile perché visitata
- mossa O – non ammissibile perché bloccata
- (1,3) diventa esaurita (o)



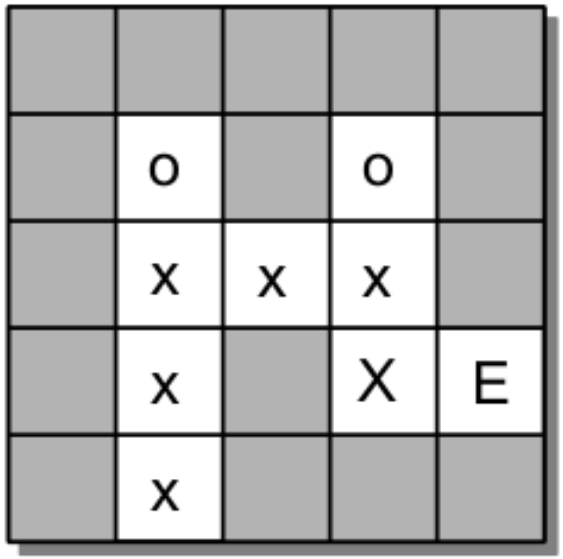
- mossa N – non ammissibile perché bloccata
- mossa E – non ammissibile perché bloccata
- mossa S – non ammissibile perché visitata
- mossa O – non ammissibile perché bloccata
- (1,3) diventa esaurita (o)
- (2,3) diventa occupata (X)



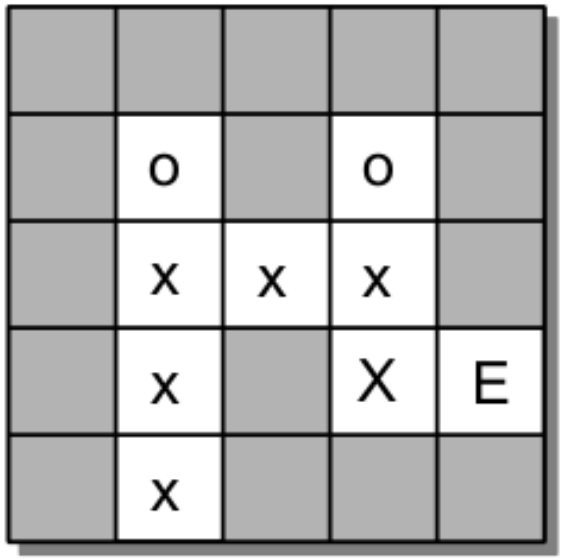
- Dallo stack prendiamo ultima casella visitata e ultima mossa effettuata



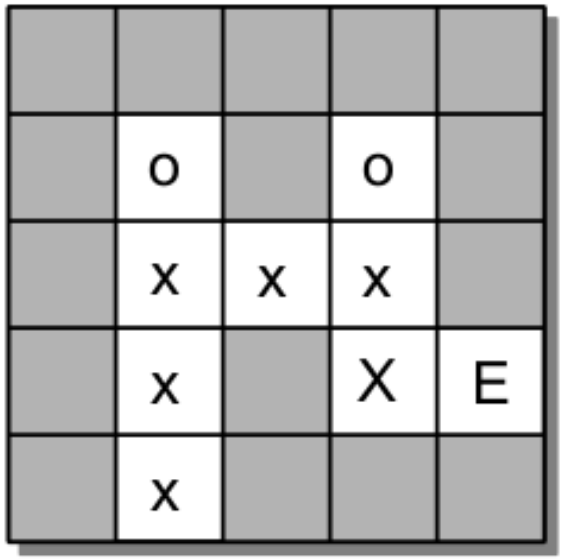
- Dallo stack prendiamo ultima casella visitata e ultima mossa effettuata
  - casella: (2,3)



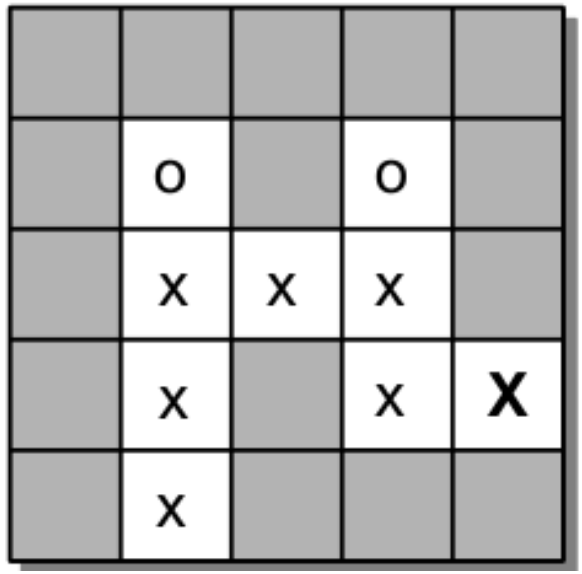
- Dallo stack prendiamo ultima casella visitata e ultima mossa effettuata
  - casella: (2,3)
  - mossa: *N*



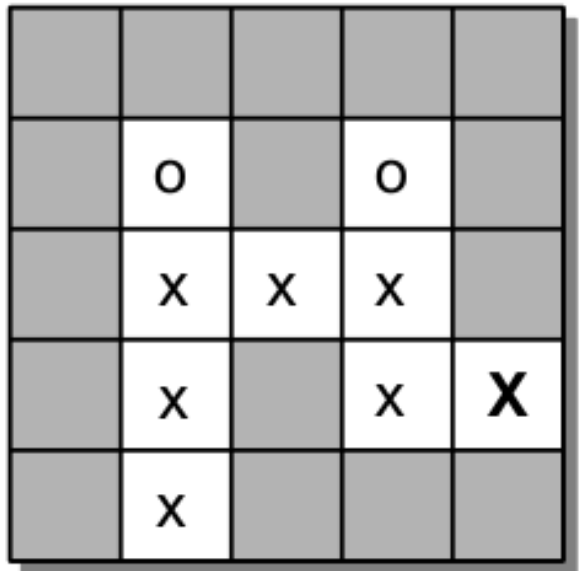
- Dallo stack prendiamo ultima casella visitata e ultima mossa effettuata
  - casella: (2,3)
  - mossa: *N*
- Mossa *E* – non ammissibile perché bloccata



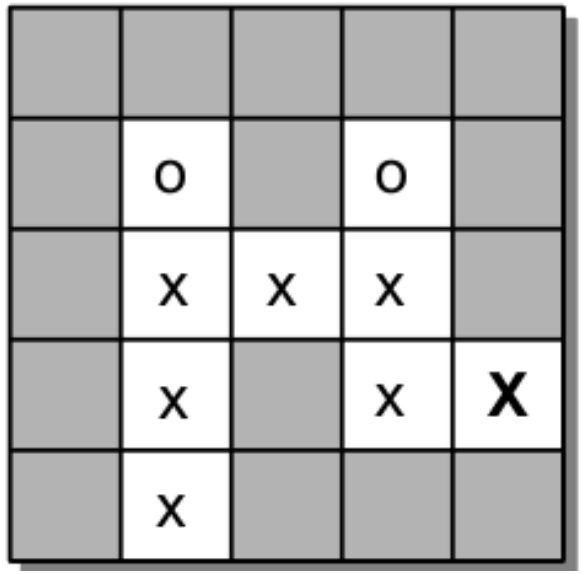
- Dallo stack prendiamo ultima casella visitata e ultima mossa effettuata
  - casella: (2,3)
  - mossa: *N*
- Mossa *E* – non ammissibile perché bloccata
- Mossa *S* – ammissibile



- Mossa N – non ammissibile perché visitata



- Mossa N – non ammissibile perché visitata
- Mossa E – ammissibile



- Mossa N – non ammissibile perché visitata
- Mossa E – ammissibile
- Fine!!

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

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

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

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

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*

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

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 $(\texttt{newr}, \texttt{newc})$

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

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 $(\texttt{newr}, \texttt{newc}, 0)$

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

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

`MOVES[i][0]` spiazzamento di riga per la i-esima mossa
`MOVES[i][1]` spiazzamento di riga per la i-esima mossa

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

`MOVES[i][0]` spiazzamento di riga per la i-esima mossa
`MOVES[i][1]` spiazzamento di riga per la i-esima mossa

1. if $\mathtt{lmove} = 4$ return NONE

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

`MOVES[i][0]` spiazzamento di riga per la i-esima mossa
`MOVES[i][1]` spiazzamento di riga per la i-esima mossa

1. if $\mathtt{lmove} = 4$ return NONE
2. for `m` in range($\mathtt{lmove} + 1$,5)

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

`MOVES[i][0]` spiazzamento di riga per la i-esima mossa
`MOVES[i][1]` spiazzamento di riga per la i-esima mossa

1. if $\mathtt{lmove} = 4$ return NONE
2. for `m` in range($\mathtt{lmove} + 1$,5)
   - `newr=r+MOVES[m][0]`

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

`MOVES[i][0]` spiazzamento di riga per la i-esima mossa
`MOVES[i][1]` spiazzamento di riga per la i-esima mossa

1. if $\mathtt{lmove} = 4$ return NONE
2. for `m` in range($\mathtt{lmove} + 1$,5)
   - `newr=r+MOVES[m][0]`
   - `newc=c+MOVES[m][1]`

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

`MOVES[i][0]` spiazzamento di riga per la i-esima mossa
`MOVES[i][1]` spiazzamento di riga per la i-esima mossa

1. if $\mathtt{lmove} = 4$ return NONE
2. for `m` in range($\mathtt{lmove} + 1$,5)
   - `newr=r+MOVES[m][0]`
   - `newc=c+MOVES[m][1]`
   - Controllare:

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

`MOVES[i][0]` spiazzamento di riga per la i-esima mossa
`MOVES[i][1]` spiazzamento di riga per la i-esima mossa

1. if $\mathtt{lmove} = 4$ return NONE
2. for `m` in range($\mathtt{lmove} + 1$,5)
   - `newr=r+MOVES[m][0]`
   - `newc=c+MOVES[m][1]`
   - Controllare:
     - ★ (newr,newc) non esce dalla griglia

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

MOVES[i][0] spiazzamento di riga per la i-esima mossa
MOVES[i][1] spiazzamento di riga per la i-esima mossa

1. if lmove $= 4$ return NONE
2. for m in range(lmove $+ 1$,5)
   - newr=r+MOVES[m][0]
   - newc=c+MOVES[m][1]
   - Controllare:
     - ★ (newr,newc) non esce dalla griglia
     - ★ (newr,newc) non è bloccata

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

MOVES[i][0] spiazzamento di riga per la i-esima mossa
MOVES[i][1] spiazzamento di riga per la i-esima mossa

1. if lmove $= 4$ return NONE
2. for m in range(lmove $+ 1$,5)
   - newr=r+MOVES[m][0]
   - newc=c+MOVES[m][1]
   - Controllare:
     - ★ (newr,newc) non esce dalla griglia
     - ★ (newr,newc) non è bloccata
     - ★ (newr,newc) non è visitata

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

MOVES[i][0] spiazzamento di riga per la i-esima mossa
MOVES[i][1] spiazzamento di riga per la i-esima mossa

1. if lmove $= 4$ return NONE
2. for m in range(lmove $+ 1$,5)
   - newr=r+MOVES[m][0]
   - newc=c+MOVES[m][1]
   - Controllare:
     - (newr,newc) non esce dalla griglia
     - (newr,newc) non è bloccata
     - (newr,newc) non è visitata
     - (newr,newc) non è esaurita

La prossima mossa ammissibile (se esiste)

Abbiamo

- la posizione corrente $(\mathtt{r}, \mathtt{c})$
- l'ultima mossa $\mathtt{lmove} \in \{0, 1, 2, 3, 4\}$

$$\mathtt{MOVES} = [[0, 0], [-1, 0], [0, 1], [1, 0], [0, -1]]$$

MOVES[i][0] spiazzamento di riga per la i-esima mossa
MOVES[i][1] spiazzamento di riga per la i-esima mossa

1. if lmove $= 4$ return NONE
2. for m in range(lmove $+ 1$,5)
   - newr=r+MOVES[m][0]
   - newc=c+MOVES[m][1]
   - Controllare:
     - ★ (newr,newc) non esce dalla griglia
     - ★ (newr,newc) non è bloccata
     - ★ (newr,newc) non è visitata
     - ★ (newr,newc) non è esaurita
3. return NONE