Laboratorio di programmazione

Python

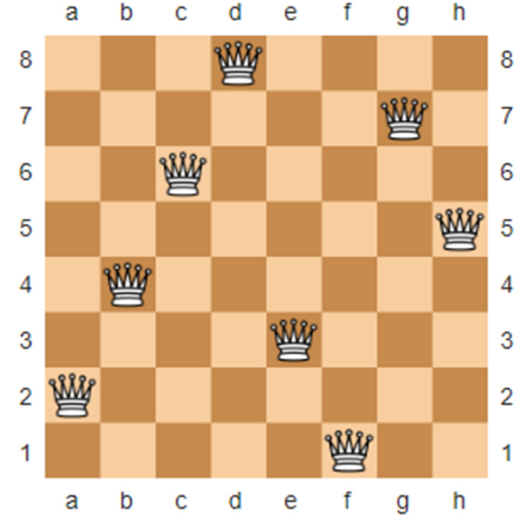
A.A. 2020-2021

Lezione 5

Strutture dati e Programmi

Problema

Esaminiamo il problema delle 8 regine:
"come posizionare su una scacchiera 8 regine in modo che non possano attaccarsi"



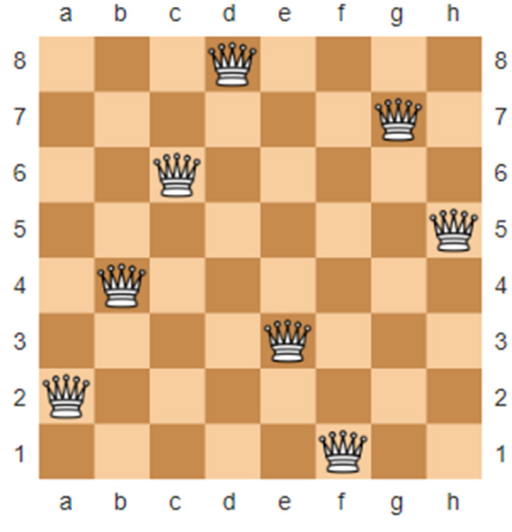
cioè non devono stare sulla stessa riga, colonna o diagonale.

Abbiamo 2 metodi semplici per risolverlo:

- *brute force*: si dispone regine in tutte le posizioni possibili e si controlla le condizioni
- *ricorsivo*: regina 1, si controlla se posizione è valida, si ripete con regina2, ...

Struttura dati

Come rappresentare il problema?



L'idea più immediata è sfruttare le matrici:

Scacchiera

Matrice righe x colonne

Valori

`0` : casella vuota

`1` : casella con Regina

La mia scacchiera diventa una cosa del genere:

```
board = [[0,0,0,1,0,0,0,0],
         [0,0,0,0,0,0,1,0],
         [0,0,1,0,0,0,0,0],
         [0,0,0,0,0,0,0,1],
         [0,1,0,0,0,0,0,0],
         [0,0,0,0,1,0,0,0],
         [1,0,0,0,0,0,0,0],
         [0,0,0,0,0,1,0,0]]
```

Test condizioni

Controlliamo se in ogni posizione (riga, colonna) della *board* è possibile posizionare una regina

```
def posizione_valida(board, riga, colonna):
    '''
    controlla se (riga,colonna) è un punto valido
    '''
    # tutta la riga e la colonna sonno vuote?
    for i in range(0, 8):
        if board[riga][i] == 1:    # se colonna ha regina, stop
            return False
        if board[i][colonna] == 1: # se riga ha regina, stop
            return False

    # tutte le diagonali sono vuote?
    for i in range(0, 8):
        for j in range(0, 8):
            # diagonale è retta con inclinazione +1 o -1
            # quindi controllo dX == dY
            if i+j == riga+colonna and board[i][j] == 1: # diagonale ha regina, stop
                return False

            if i-j == riga-colonna and board[i][j] == 1: # diagonale ha regina, stop
                return False

    # il posto è valido
    return True
```

Ricorsione

Caso finale: tutte le righe controllate e posizione OK

Ricorsione: se posizionamento OK, controllo nuova riga con nuova board

```
def posiziona_regina(board, riga):
    #
    if riga > 7:
        # condizione finale
        print('Soluzione trovata')
        visualizza_soluzione(board)
    else :
        # caso ricorsivo
        for col in range(8):
            # per ogni colonna controllo se poszione va bene
            if posizione_valida(board, riga, col) :
                # se la posizione va bene, metto la regina
                board[riga][col] = 1
                # continuo con la riga sotto
                posiziona_regina(board, riga + 1)
                board[riga][col] = 0
```

Brute-force

Disponiamo le 8 regine sulla scacchiera e controlliamo se le posizioni vanno bene Se non vanno bene proviamo con una nuova disposizione

Valori

`0` : casella vuota

`1` : casella con Regina

Scacchiera

```
board = [[0,0,0,1,0,0,0,0],
         [0,0,0,0,0,0,1,0],
         [0,0,1,0,0,0,0,0],
         [0,0,0,0,0,0,0,1],
         [0,1,0,0,0,0,0,0],
         [0,0,0,0,1,0,0,0],
         [1,0,0,0,0,0,0,0],
         [0,0,0,0,0,1,0,0]]
```

Brute-force

Disponiamo le 8 regine sulla scacchiera e controlliamo se le posizioni vanno bene Se non vanno bene proviamo con una nuova disposizione

Valori

`0` : casella vuota

`1` : casella con Regina

Scacchiera

```
board = [[0,0,0,1,0,0,0,0],
         [0,0,0,0,0,0,1,0],
         [0,0,1,0,0,0,0,0],
         [0,0,0,0,0,0,0,1],
         [0,1,0,0,0,0,0,0],
         [0,0,0,0,1,0,0,0],
         [1,0,0,0,0,0,0,0],
         [0,0,0,0,0,1,0,0]]
```

Ci sono **4426165368** modi per posizionare 8 regine in 64 caselle.

Ragionare sulla struttura dati

Ragioniamo su com'è fatta la nostra scacchiera:

```
board = [[0,0,0,1,0,0,0,0],
         [0,0,0,0,0,0,1,0],
         [0,0,1,0,0,0,0,0],
         [0,0,0,0,0,0,0,1],
         [0,1,0,0,0,0,0,0],
         [0,0,0,0,1,0,0,0],
         [1,0,0,0,0,0,0,0],
         [0,0,0,0,0,1,0,0]]
```

Gran parte della struttura dati è vuota: contiene solo zeri
Se usassimo le coordinate (riga, colonna) degli 8 punti?

```
posizioni = [(0,3) , (1,6) , (2,2) , (3,7) , (4,1) , (5,4) , (6,0) , (7,5)]
```

avremmo rappresentato la stessa informazione in modo molto più compatto.

Astrazione del problema

La scacchiera è diventata una *lista* di punti in cui si trovano le regine

```
posizioni = [(0,3) , (1,6) , (2,2) , (3,7) , (4,1) , (5,4) , (6,0) , (7,5)]
```

Le coordinate *riga* vanno da 0 a 7 poichè le regine non possono stare sulla stessa riga quindi sono univoche e le possiamo usare come indici della lista

```
posizioni = [3, 6, 2, 7, 1, 4, 0]
```

Le regine non possono condividere la colonna, quindi trovare le soluzioni del problema equivale a trovare le permutazioni della lista

```
[0,1,2,3,4,5,6,7]
```

che soddisfano le condizioni:

1. no elementi nella stessa riga: **ok per costruzione** perché l'indice lista è univoco
2. no elementi nella stessa colonna: **ok per costruzione** perché valori vanno da 0 a 7
3. **no elementi sulle stesse diagonali: da controllare** (es. [0,1,2,3,4,5,6,7] sono tutti sulla stessa diagonale)

Astrazione del problema

La scacchiera è diventata una *lista* di punti in cui si trovano le regine

```
posizioni = [(0,3) , (1,6) , (2,2) , (3,7) , (4,1) , (5,4) , (6,0) , (7,5)]
```

Le coordinate *riga* vanno da 0 a 7 poichè le regine non possono stare sulla stessa riga quindi sono univoche e le possiamo usare come indici della lista

```
posizioni = [3, 6, 2, 7, 1, 4, 0]
```

Le regine non possono condividere la colonna, quindi trovare le soluzioni del problema equivale a trovare le permutazioni della lista

```
[0,1,2,3,4,5,6,7]
```

che soddisfano le condizioni:

1. no elementi nella stessa riga: **ok per costruzione** perché l'indice lista è univoco
2. no elementi nella stessa colonna: **ok per costruzione** perché valori vanno da 0 a 7
3. **no elementi sulle stesse diagonali: da controllare** (es. [0,1,2,3,4,5,6,7] sono tutti sulla stessa diagonale)

Ci sono **40320** modi di posizionare gli 8 numeri (8!)
...eravamo partiti da **4426165368** modi...

Test condizioni

In questo caso l'unico controllo è "Non devono essere sulla stessa diagonale"

```
def stessa_diagonale(x0, y0, x1, y1):
    '''Ritorna Vero se posizioni (x0, y0) e (x1, y1)
       sono sulla stessa "diagonale"
    '''
    dy = abs(y1 - y0)   # distanza lungo y
    dx = abs(x1 - x0)   # distanza lungo x
    return dx == dy     # se dx == dy , dx/dy == 1 e sono sulla stessa diagonale
```

E va verificato per ogni elemento (colonna) della lista `[3, 6, 2, 7, 1, 4, 0]` precedente

```
def incrocia_colonne(scacchiera_posizioni, col):
    '''Ritorna Vero se la colonna 'col' incrocia qualcuna
       delle posizioni precedenti nella diagonale
    '''
    # controllo tutte le precedenti fino a questa 'col'
    for c in range(col):
        # coordinata X è indice (c) , coordinata Y è valore lista(c)
        if stessa_diagonale(c, scacchiera_posizioni[c], col, scacchiera_posizioni[col]):
            return True  # stop se trovo problemi

    return False    # nessun incrocio, la posizione va bene
```

Random

Siamo **partiti** da posizionare 8 regine in 64 caselle: **4.426.165.368 tentativi** possibili

Abbiamo **ridotto** il problema a "trovare le permutazioni di [0,1,2,3,4,5,6,7]": **40320 tentativi**

1) Provare tutte le permutazioni in ordine: approccio *brute force*

2) Provare permutazioni casuali e verificare che siano corrette

> *Il problema con 8 regine ha 92 soluzioni: ci vorranno in media **438** tentativi (40320/92)*

```python
import random
import time

def board_ok(scacchiera_posizioni):
    '''Controlla tutte le posizioni per cercare
       se ogni colonna incrocia la diagonale
    '''
    for col in range(1,len(scacchiera_posizioni)):
        if incrocia_colonne(scacchiera_posizioni, col):
            return False # stop se trovo problemi

    return True # Nessun incrocio, posizione valide

def main():
    '''Program'''
    random_generator = random.Random() # inizializzo generatore
    scacchiera = list(range(8))        # preparo posizoni da testare
    solutions  = 0                     # conto le soluzioni trovate
    start_time = time.time()           # tempo di partenza

    # loop finchè non trovo una soluzione
    while solutions < 1:

        random_generator.shuffle(scacchiera) # 'mescolo' posizioni

        if board_ok(scacchiera): # controllo se va bene
            print(f'Found solution {scacchiera} in {time.time() - start_time} s.')
            solutions += 1      # conto la soluzione "trovata"
            start_time = time.time()   # reset timer
```

Esercizi 3

1) Trovate 15 soluzioni per il gioco delle regine con il metodo delle permutazioni: quanto è il tempo medio?

2) Contate quanti tentativi fa il programma per trovare ogni soluzione

3) Alcune soluzioni possono essere ripetute: fate in modo che le soluzioni siano "uniche"

4) Se ci sono soluzioni ripetute, contate quante volte ogni soluzione è ripetuta

4) Generalizzate il programma per risolvere una scacchiera di qualunque dimensione NxN

5) Trovate qual'è la scacchiera piu' grande di cui si riesce a trovare 1 soluzione in meno di 30s

Esercizi 4

7) Ogni soluzione è 'simmetrica' per rotazioni della scacchiera 8x8 di 90, 180 e 270 gradi. Trovata una soluzione, costruite le 4 simmetriche per rotazione prima di cercarne un'altra