N-Queen-problem-using-simulated annealing-algorithm

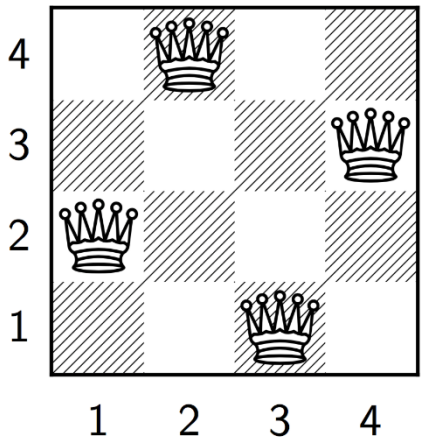
Professor : **Dr.AliShakiba**

By : **Maryam Mohammadabadi**

Simulated annealing in N-queens

مسئله N-queens این است که N ملکه ها را روی صفحه شطرنج N x N قرار دهید تا هیچ کدام در یک ردیف ، ستون یکسان یا مورب یکسان نباشند.

به عنوان مثال ، اگر N = 4 باشد :



قصد داریم این مسئله را با استفاده از الگوریتم Simulated annealing حل کنیم.

الگوریتم Simulated annealing به این صورت است: .

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE(current) – VALUE(next)
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(−ΔE/T)
```

1.در ابتدا کتابخوانه های مورد نظر راimport میکند.

```
import random
import math
```

2.سپس تعداد ملکه ها را از کاربر دریافت میکند.

```
N = int(input("Enter Number of Queens: "))
```

```
Enter Number of Queens:
```

3.TEMPERATURE را مساوی با 200000 قرار میدهیم.

```
TEMPERATURE = 200000
```

4. تابع random_board() یک صفحه شطرنج تصادفی ایجاد و برمیگرداند.

```
def random_board():
    board = list([random.randint(0, N-1) for x in range(N)])
    return board
```

5. تابع num_of_conflicts یک حالت را گرفته و تعداد درگیری های ان را برمیگرداند.

```
                                    """number of conflicts"""
def num_of_conflicts(state):
    conflicts = 0
    for curr_queen in range(N):
        for other_queen in range(curr_queen + 1, N)
            if state[curr_queen] == state[other_queen]:
                conflicts += 1
            if abs(state[curr_queen] - state[other_queen]) == (other_queen - curr_queen):
                conflicts += 1
    return conflicts
```

6. تابع simulated_annealing :

ابتدا یک حالت اولیه را انتخاب کرده(curr_state)

```
curr_state = random_board()
```

6.1 بااستفاده از تابعnum_of_conflicts تعداد درگیری های حالتcurr_state را به دست می اورد.

```
curr_num_conflicts = num_of_conflicts(curr_state)
```

6.2 T را دما در نظر گرفته.

```
t = TEMPERATURE
```

6.3 یک حلقه ایجاد کرده که شرط پایان t > 0 و iterations > 0 است.

(iterations تعداد تکرار است ک قبل از حلقه مقدار ان را مساوی با 100000 قرار داده است.)

```
while t > 0 and iterations > 0:
```

ابتدا یک کپی از curr_state داخل متغیری به اسم successor ریخته سپس دو عدد تصادفی بین یک و N – 1 را تولید کرده و داخل col و row میریزد.

داخل successor[col] مقدار row را قرار داده.

تعداد درگیری های successor را با استفاده از تابعnum_of_conflicts محاسبه کرده و داخل successor_conflicts میریزد.

سپس مقدار ΔE را محاسبه میکند

delta = successor_conflicts - curr_num_conflicts

ΔE = ((successor)تعداد conflicts (درگیری) ساخته شده) – ((تعداد conflicts (درگیری) اولیه)

سپس بررسی میکند ایا $\Delta E < 0$ است یا خیر؟یعنی کاندیدای جدید بهتر از قبلی هست یا نه؟

```
    # check if the neighbor is better or the propability is bigger then random prop
    if delta < 0 or random.uniform(0, 1) < math.exp(-delta / t):
```

اگر بهتر باشد به حالت بعدی حرکت میکند

```
    curr_state = successor.copy()

    curr_num_conflicts = num_of_conflicts(curr_state)
```

سپس دما را افزایش داد و یکی از شمارنده کم میکند.

```
    t *= sch
    iterations -= 1
```

و در اخر حلقه چک میکند اگر curr_num_conflicts == 0 بود

solution_found را True میکند وبااستفاده از تابعprint_board حالت curr_state را چاپ میکند.

```
        if curr_num_conflicts == 0:
            solution_found = True
            print_board(curr_state)
            break
```

7. زمانی که حلقه به پایان میرسد با یک شرط چک میکند اگر متغیر بولین solution_found, False بود یعنی راه حلی برای حل این حالت پیدا نشده پس پیغام " Failed"را چاپ میکند.

تابع simulated_annealing :

```
def simulated_annealing():
    solution_found = False
    curr_state = random_board()
    curr_num_conflicts = num_of_conflicts(curr_state)
    t = TEMPERATURE
    # cooling rate
    sch = 0.99
    iterations = 100000
    while t > 0 and iterations > 0:
        successor = curr_state.copy()
        col = random.randint(0, N - 1)
        row = random.randint(0, N - 1)
        successor[col] = row
        successor_conflicts = num_of_conflicts(successor)
        delta = successor_conflicts - curr_num_conflicts
        # check if the neighbor is better or the propability is bigger then random prop
        if delta < 0 or random.uniform(0, 1) < math.exp(-delta / t):
            curr_state = successor.copy()
            curr_num_conflicts = num_of_conflicts(curr_state)
            t *= sch
            iterations -= 1
        if curr_num_conflicts == 0:
            solution_found = True
            print_board(curr_state)
            break
    if solution_found is False:
        print("Failed")
```

8. تابعprint_board صفحه شطرنج را چاپ میکند.

```
                                    """Print the board"""
def print_board(board):
    for col in range(N):
        for row in range(N):
            if board[col] == row:
                print('Q', end=" ")
            else:
                print('x', end=" ")
        print()
    print()
```

9. و در پایان در تابع main تابع simulated_annealing فراخوانی کرده.

```
def main():
    simulated_annealing()

if __name__ == "__main__":
        main()
```

ورودی:

```
N = int(input("Enter Number of Queens: "))
```

```
Enter Number of Queens: 5
```

خروجی:

```
In [31]: def main():
             simulated_annealing()

         if __name__ == "__main__":
                 main()
```

```
Q x x x x
x x x Q x
x Q x x x
x x x x Q
x x Q x x
```