(به نام خدا)

محبی نوشاد 9714305N

توضیح در مورد الگوریتم Simulated annealing

برای مسئله‌ی n وزیر

در این الگوریتم من یک حالت اولیه را به صورت تصادفی انتخاب می‌کنم و همین طور ادامه می‌دهم تا زمانی که دما صفر شود.

(من یک دمای اولیه دارم بعد یک قانونی دارم که آن قانون دمای من را کاهش می‌دهد طبق زمان بند) و من زمانی متوقف می‌شوم که دما برابر صفر هست.

نکته‌ی اول، دما را با T در خاطر می‌گیرد، آن این دما باید برابر صفر باشد همین حالتی که هست را می‌پذیرد.

اما اگر دما صفر نیست می‌آید:

یک حالت دیگر را اتفاقی به صورت random طبق تابع successor، حالت بعدی را انتخاب می‌کنم و می‌آیم یک ΔE حساب می‌کنم که من خواهم بیشتر باشد.

ΔE = حالت جاری − مقدار حدی

حالا اگر $\Delta E > 0$ هست یا نه؟ اگر هست می‌گوییم برو به حالت بعدی و اگر یک حرکت بد بود می‌آییم:

① من حالت را عوض نمی‌کنم (حالت قبلی هست) (دوباره یک دمای جدید تر در خاطر می‌گیرم.

② با احتمال $e^{\Delta E/T}$ می‌آیم حالت بعدی را انتخاب می‌کنم.

③ و بعد می‌رویم با این احتمال حرکت بعدی را انجام بده.

یک تابع successor برای n وزیر.

$$P(S_0, S', t) = \alpha \times \begin{cases} 1 & \text{if } E(S') > E(S) \quad (\Delta E > 0) \\ e^{(E(S') - E(S))/T(t)} & \text{غیره} \end{cases}$$

S_0: حالت فعلی
S': حالت بعدی
t: تعداد تکرارها

دما که بالا باشد احتمال حرکت های بد افزایش پیدا می‌کند.

توضیح در مورد کد در مسئله‌ی ۸ وزیر:

① تابع random_borad() : یک صفحه شطرنج تصادفی را ایجاد می‌کند و برمی‌گرداند.

② تابع num_of_conficts : یک حالت را گرفته و تعداد درگیری‌های (برخورد)های آن را برمی‌گرداند.

فعلاً پاسخ را برابر false می‌گذاریم.

③ تابع simulated_annealing : ابتدا یک حالت اولیه را به صورت random انتخاب می‌کند و و بعد با استفاده از تابع num_of_conficts تعداد برخوردهای آن را به دست می‌آورد و در متغیر curr_num_conflicts قرار می‌دهد.

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

```python
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
```

```
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
```

بعد یک حلقه while که تا زمانی که دما صفر نشود ادامه می‌یابد:

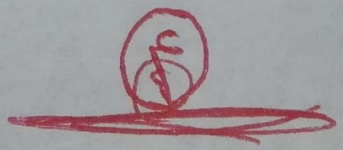
ابتدا یکی از successor داخل متغیر curr_stat قرار می‌گیرد بعد دو عدد تصادفی بین 1- N تا 1 را تولید کرده و داخل col و row قرار می‌دهد، تعداد درگیری‌ها (برخوردها) را با استفاده از تابع ② محاسبه کرده و در successor_conflicts قرار می‌دهد.

سپس مقدار ΔE را محاسبه می‌کنیم:

delta = successor_conflicts - curr_num_conflicts

سپس بررسی می‌کنیم که آیا $\Delta E > 0$ هست یا نه؟

اگر بهتر باشد به حالت بعدی حرکت می‌کند و بعد دما را افزایش و یکی از شمارنده کم می‌کنیم.



```
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
```

در آخر حلقه چک می‌کنیم که آیا این دما، برابر با صفر بود آن حالت curr_state
را خارج می‌کند. با استفاده از دستور if

زمانی که حلقه به پایان رسید با یک شرط چک می‌کنم آیا متغیر
solution_found == false بود یعنی راه حلی برای این
مسئله پیدا نکرده است پیغام "Failed" را خارج می‌کند.

```
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")
```

تابع Print-board می‌آید صفحه‌ی شطرنج را چاپ می‌کند.

۷

تابع main برای فراخوانی الگوریتم Simulated-annealing هست

۸

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

```
def main():
    simulated_annealing()


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