simulated annealing algorithm

این الگوریتم را می توان در ۴ مرحله تجزیه کرد:

1. در یک نقطه xتصادفي شروع کن .
2. یک نقطه جدید x_j را در یک مکان $N(x)$انتخاب کنید.
3. تصمیم بگیرید که به نقطه جدید x_jحرکت کنید یا نه. این تصمیم بر اساس تابع احتمال گرفته خواهد شد.
4. کاهش T.

تابع $P(x, x_j, T)$ ما را در مورد اینکه به نقطه ي جدید حرکت کنیم یا نه راهنمايي ميكنه:

$$\mathbb{P}(x, x_j, T) = \begin{cases} 1 & si \quad F(x_j) \geq F(x) \\ e^{\frac{F(x_j)-F(x)}{T}} & si \quad F(x_j) < F(x) \end{cases}$$

تابع $F(x)$ عینی است (تابعی که برای آن می خواهیم نقطه بهینه x را پیدا کنیم). اگر نقطه جدید x_j نتیجه تابع عینی را بهبود بخشد، با احتمال ۱ به نقطه جدید حرکت خواهیم کرد.

هنگامی که نقطه جدید تابع هدف را بهبود نمی بخشد، بسته به تفاوت $F(x_j) - F(x)$ و متغیر T به نقطه جدید حرکت خواهیم کرد. هنگامی که T بالا است امکان حرکت به نقطه جدید نیز بالا خواهد بود و زمانی که کم است امکان حرکت به نقطه جدید نیز کم است. به همین دلیل است که در ابتدا با Tبالا شروع خواهیم کرد تا اکتشافات بیشتری انجام دهیم و به تدریج ارزش رسیدن به نقطه مطلوب را پایین بیاوریم.

N-Queen

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

N-Queen problem using simulated annealing algorithm

تابع random_board() :

یک صفحه شطرنجی تصادفی ایجاد وبرمیگرداند

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

تابعnum_of_conflicts :

یک حالت را گرفته و تعداد درگیری های ان را برمیگرداند

```
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
```

تابع simulated_ annealing :

ابتدا یک حالت اولیه را بصورت رندوم انتخاب کنید

```
curr_state = random_board()
```

با استفاده از تابع num_of_conflicts تعداد درگیری های حالت curr_state را بدست اورید

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

T را دما درنظر گرفته

```
t = TEMPERATURE
```

این مرحله را باید تا جایی که دما صفر شود ادامه دهید

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

ابتدا یک کپی ازcurr_state داخل متغیر successor ریخته

دو عدد تصادفی بین 1 و N-1 را تولید کرده و داخل col و row بریزید

داخلsuccessor[col] مقدار row را قرار داده

تعداد درگیری های successor را با استفاده از تابعnum_of_conflictsمحاسبه کرده و داخل successor_conflictsبریزید.

سپس مقدار ΔE را محاسبه میکند

delta = successor_conflicts - curr_num_conflicts

سپس بررسی کنید ایا ΔE< 0است یا خیر؟یعنی کاندیدای جدید بهتر از قبلی هست یا نه؟

```
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 ویکی از شمارنده کم میکند

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

در اخر حلقه چک میکند اگرcurr_num_conflicts==0 بود حالت curr_stateرا چاپ میکند.

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

زمانی که حلقه به پایان میرسد با یک شرط چک میکند اگر متغیرsolution_found , false بود یعنی راه حلی برای حل این مسیله پیدا نشده پیغام"Failed " را چاپ میکند.

```
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

    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")
```

تابعprint_board:

برای چاپ صفحه شطرنج است

```
def print_board(board):
    for col in range(N):
        for row in range(N):
            if board[col] == row:
                print(1, end=" ")
            else:
                print(0, end=" ")
        print()
    print()
```

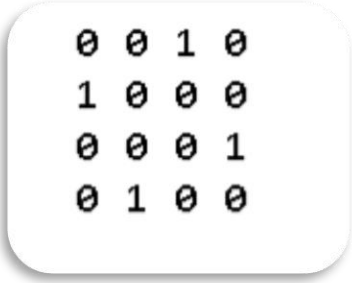
تابع main:

برای فراخوانی تابع simulated_annealing

```
def main():
    simulated_annealing()


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

خروجی کد برای ورودی 4:



فاطمه خدادادی_97143013