



# 算法设计与分析

作业（九）

|      |                |
|------|----------------|
| 姓 名  | 熊恪峥            |
| 学 号  | 22920202204622 |
| 日 期  | 2022年5月8日      |
| 学 院  | 信息学院           |
| 课程名称 | 算法设计与分析        |

# 作业（九）

## 目录

|   |        |   |
|---|--------|---|
| 1 | 题12.2  | 1 |
| 2 | 题12.3  | 1 |
| 3 | 题12.4  | 1 |
| 4 | 题12.5  | 1 |
| 5 | 题12.8  | 3 |
| 6 | 题13.2  | 3 |
| 7 | 题13.4  | 4 |
| 8 | 题13.10 | 5 |

## 1 题12.2

对棋盘建立一个坐标系，则两个皇后在一个对角线上当且仅当 $k_{ij} = 1$ 或 $k_{ij} = -1$ 。即

$$k_{ij} = \frac{x_i - x_j}{i - j} = 1$$

$$\Rightarrow x_i - x_j = i - j$$

同理可得 $x_i - x_j = j - i$

## 2 题12.3

递归地对第1, 2, ..., 8行枚举放的位置，放棋子时检查同一行、同一列、对角线上是否有棋子。对角线按第1节的结论进行检查得到算法 1。

---

### 算法 1 8皇后问题

---

```

1: procedure PUTQUENE( $k$ )
2:   if  $k = 8$  then
3:     PRINT( $pos$ )
4:   for  $i = 1 \rightarrow 8$  do
5:     if CHECK( $k, i$ ) then
6:        $pos[k] \leftarrow i$ 
7:        $t = \text{PUTQUENE}(k + 1)$ 
8:        $pos[k] \leftarrow 0$ 
9:   return  $t$ 
10: procedure CHECK( $row, col$ )
11:    $r = pos[row] == 0$ 
12:    $c = True$ 
13:   for  $i = 1 \rightarrow 8$  do
14:     if  $pos[i] == col$  then
15:        $c = False$ 
16:       break
17:    $d = True$ 
18:   for  $i = 1 \rightarrow 8$  do
19:     for  $j = 1 \rightarrow 8$  do
20:       if  $row - i == col - j$  or  $row - i = j - col$  then
21:         if  $pos[i] == j$  then
22:            $d = False$ 
23:           break
24:   return  $c$  and  $r$  and  $d$ 

```

---

## 3 题12.4

递归地检查所有棋子，然后标记所有能攻击的位置。最后检查是否所有的位置都被标记过了。如算法 2

## 4 题12.5

枚举每个位置的放置方法，如算法 3。

---

**算法 2** 检查能否攻击

---

```

1: procedure CHECK( $pos, n$ )
2:    $board[] = \emptyset$ 
3:   MARK( $board, pos, 0, n$ )
4:   for  $i = 1 \rightarrow n$  do
5:     for  $j = 1 \rightarrow n$  do
6:       if  $board[i][j] == 0$  then
7:         return False
8:   return True
9: procedure MARK( $board, pos, k, n$ )
10:   $r = pos[row] == 0$ 
11:   $c = True$ 
12:  for  $i = 1 \rightarrow 8$  do
13:    if  $pos[i] == col$  then
14:       $c = False$ 
15:      break
16:   $d = True$ 
17:  for  $i = 1 \rightarrow 8$  do
18:    for  $j = 1 \rightarrow 8$  do
19:      if  $row - i == col - j$  or  $row - i = j - col$  then
20:        if  $pos[i] == j$  then
21:           $d = False$ 
22:          break
23:  return  $c$  and  $r$  and  $d$ 

```

---



---

**算法 3** 检查能否攻击

---

```

1: procedure PERMU( $p, k, n$ )
2:   if  $k == n$  then
3:     PRINT( $p$ )
4:   for  $i = 1 \rightarrow n$  do
5:     if  $vis[i] = True$  then
6:        $vis[i] = True$ 
7:        $p[k] = i$ 
8:       PERMU( $p, k + 1, n$ )
9:        $vis[i] = False$ 

```

---

## 5 题12.8

可以在枚举每种顶点序列，并且每一步以 $w[x[i-1], x[j]] \neq \infty$ 为约束选择是否下一步走顶点 $j$ 。如算法4

算法 4 哈密顿回路

---

```

1: procedure HAMITON( $w, x, k$ )
2:   if  $k == n$  then
3:     if  $w[x[n], x[1]] \neq \infty$  and  $w[x[n-1], x[n]] \neq \infty$  then
4:       PRINT( $x$ )
5:   for  $i = k \rightarrow n-1$  do
6:     if  $w[x[k-1], x[j]] \neq \infty$  then
7:        $x[k] \leftrightarrow x[j]$ 
8:       HAMITON( $w, x, k+1$ )
9:        $x[k] \leftrightarrow x[j]$ 

```

---

## 6 题13.2

设计限界函数 $B(i)$ 是当前耗时的值加上之后步骤中空闲工人中耗时最小的耗时的和。设 $v[i] = \begin{cases} \infty & \text{工人已被分配} \\ 1 & \text{工人未被分配} \end{cases}$   
则 $B(i)$ 如(1)

$$B(i) = C(i) + \sum_{j=i}^n \min_k \{c[k, j] \cdot v[k]\} \quad (1)$$

可以实现如代码 1。

代码 1: 工人分配问题

```

1 from queue import PriorityQueue
2
3 c = [[3, 5, 2, 4], [6, 7, 5, 3], [3, 7, 4, 5], [8, 5, 4, 6]]
4
5
6 def get_min():
7     mins = []
8     for i in range(0, 4):
9         minval = 0x7fffffff
10        for j in range(0, 4):
11            minval = min([minval, c[j][i]])
12        mins.append(minval)
13    return mins
14
15
16 def assign():
17     mins = get_min()
18
19     def b():
20         ret = cur
21         for i in range(step, 4):
22             if not vis[i]:
23                 ret += mins[i]
24         return ret
25

```

```
26     ans = 0x7fffffff
27     cur, step, vis = 0, 0, [False for _ in range(0, 4)]
28     q = PriorityQueue()
29     while step < 4:
30         for i in range(0, 4):
31             new_cost = cur + c[i][step]
32
33             if vis[i] or new_cost > ans:
34                 continue
35
36             if step >= 3:
37                 ans = min([ans, new_cost])
38
39             vis[i] = True
40             q.put([b(), (new_cost, step + 1, vis)])
41             vis[i] = False
42
43     _, (cur, step, vis) = q.get()
44     return ans
```

运行结果是13。

## 7 题13.4

N皇后问题的限制函数与回溯法相同，实现如代码 2。

代码 2: N皇后问题

```
1  from queue import Queue
2
3
4  def check(pos, k, i):
5      if pos[k] >= 0:
6          return False
7
8      for j in range(0, k):
9          if pos[j] == i:
10             return False
11
12         if i - pos[j] == j - k:
13             return False
14
15         if i - pos[j] == k - j:
16             return False
17
18     return True
19
20
21 def n_queen(n):
22     k, pos = 0, [-1 for i in range(0, n)]
23     ans = 0
24     q = Queue()
25     q.put((k, pos))
26     while not q.empty():
```

```
27         k, pos = q.get()
28         if k == n:
29             print(pos)
30             ans += 1
31             continue
32
33         for i in range(0, n):
34             if check(pos, k, i):
35                 q.put((k + 1, [i if r == k else pos[r] for r in range(0, n)]))
36     return ans
```

对于 $N = 8$ 的情形，该算法输出92。

## 8 题13.10

约束函数是马只能走到8个位置之一，并且该位置不越界。实现如代码3。

代码 3: 跳马问题

```
1
2 from queue import Queue
3
4 dx = [-1, -1, -2, -2, 1, 1, 2, 2]
5 dy = [2, -2, 1, -1, 2, -2, 1, -1]
6
7
8 def horse(n, sx, sy, ex, ey):
9     vis = [[False for i in range(0, n)] for j in range(0, n)]
10    vis[sx][sy] = True
11
12    def check(x, y, dx, dy):
13        if vis[x + dx][y + dy]:
14            return False
15
16        if x + dx < 0 or x + dx >= n:
17            return False
18
19        if y + dy < 0 or y + dy >= n:
20            return False
21
22        return True
23
24    q = Queue()
25    t = 0
26    q.put((t, sx, sy))
27    while not q.empty():
28        t, x, y = q.get()
29        if x == ex and y == ey:
30            return t
31
32        for nx, ny in zip(dx, dy):
33            if check(x, y, nx, ny):
34                q.put((t + 1, x + nx, y + ny))
35                vis[x + nx][y + ny] = True
```

```
36  
37     return -1
```

对从(1,1)到(2,1)的情形，该算法输出4；从(1,5)到(5,1)的情形，该算法输出5。