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

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

$$\begin{aligned} k_{ij} &= \frac{x_i - x_j}{i-j} = 1 \\ &\Rightarrow x_i - x_j = i - j \end{aligned}$$

同理可得$x_i - x_j = j - i$

2 题12.3

递归地对第$1,2,\ldots,8$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

```
procedure PutQuene(k)
    if k = 8 then
        Print(pos)
    for i = 1 → 8 do
        if Check(k, i) then
            pos[k] ← i
            t = PutQuene(k + 1)
            pos[k] ← 0
    return t
procedure Check(row, col)
    r = pos[row] == 0
    c = True
    for i = 1 → 8 do
        if pos[i] == col then
            c = False
            break
    d = True
    for i = 1 → 8 do
        for j = 1 → 8 do
            if row − i == col − j or row − i = j − col then
                if pos[i] == j then
                    d = False
                    break
    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

```
procedure CHECK(pos, n)
    board[][] = ∅
    MARK(board, pos, 0, n)
    for i = 1 → n do
        for j = 1 → n do
            if board[i][j] == 0 then
                return False
    return True
procedure MARK(board, pos, k, n)
    r = pos[row] == 0
    c = True
    for i = 1 → 8 do
        if pos[i] == col then
            c = False
            break
    d = True
    for i = 1 → 8 do
        for j = 1 → 8 do
            if row − i == col − j or row − i = j − col then
                if pos[i] == j then
                    d = False
                    break
    return c and r and d
```

算法 3 检查能否攻击

```
procedure PERMU(p, k, n)
    if k == n then
        PRINT(p)
    for i = 1 → n do
        if vis[i]! = True then
            vis[i] = True
            p[k] = i
            PERMU(p, k + 1, n)
            vis[i] = False
```

5 题12.6

算法 4 象棋覆盖问题

```
procedure SAT(i, m)
    for k = 1 → 8 do
        if x + dx[k] ≥ 0 and y + dy[k] ≥ 0 and x + dx[k] ≤ 8 and y + dy[k] ≤ 8 then
            step[i] ← k
        x ← x + dx[k]
        y ← y + dy[k]
        if x = x_0 and y = y_0 then
            OUTPUT
            return
        else
            if flag[x, y] = 0 then
                flag[x, y] ← 1
```

6 题12.7

按照3CNF-SAT问题的定义，可以简单地写出回溯算法 5。

算法 5 3SAT问题

```
procedure SAT(i, m)
    if i > m then
        OUTPUT
    for k ∈ {False, True} do
        x[i] = k
        for j = 1 → m do
            if c[j] = 1 then
                t ← True
                return
        if t ← False then
            SAT(i + 1, m)
```

7 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6

算法 6 哈密顿回路

```
procedure HAMITON(w, x, k)
    if k == n then
        if w[x[n], x[1]] ≠ ∞ and w[x[n−1], x[n]] ≠ ∞ then
            PRINT(x)
    for i = k → n − 1 do
        if w[x[k−1], x[j]] ≠ ∞ then
            x[k] ↔ x[j]
            HAMITON(w, x, k + 1)
            x[k] ↔ x[j]
```

8 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)

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

可以实现如代码 1。

代码 1: 工人分配问题

```python
from queue import PriorityQueue

c = [[3, 5, 2, 4], [6, 7, 5, 3], [3, 7, 4, 5], [8, 5, 4, 6]]


def get_min():
    mins = []
    for i in  range(0, 4):
        minval = 0x7fffffff
        for j in  range(0, 4):
            minval =  min([minval, c[j][i]])
        mins.append(minval)
    return mins


def assign():
    mins = get_min()

    def b():
        ret = cur
        for i in  range(step, 4):
            if not vis[i]:
                ret += mins[i]
        return ret

    ans = 0x7fffffff
    cur, step, vis = 0, 0, [False for _ in  range(0, 4)]
    q = PriorityQueue()
    while step < 4:
        for i in  range(0, 4):
            new_cost = cur + c[i][step]

            if vis[i] or new_cost > ans:
                continue

            if step >= 3:
                ans =  min([ans, new_cost])

            vis[i] = True
            q.put([b(), (new_cost, step + 1, vis)])
            vis[i] = False

```

```
        _, (cur, step, vis) = q.get()
    return ans
```

运行结果是13。

9 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

```python
from queue import Queue


def check(pos, k, i):
    if pos[k] >= 0:
        return False

    for j in  range(0, k):
        if pos[j] == i:
            return False

        if i - pos[j] == j - k:
            return False

        if i - pos[j] == k - j:
            return False

    return True


def n_queen(n):
    k, pos = 0, [-1 for i in  range(0, n)]
    ans = 0
    q = Queue()
    q.put((k, pos))
    while not q.empty():
        k, pos = q.get()
        if k == n:
            print(pos)
            ans += 1
            continue

        for i in  range(0, n):
            if check(pos, k, i):
                q.put((k + 1, [i if r == k else pos[r] for r in  range(0, n)]))
    return ans
```

对于$N = 8$的情形，该算法输出92。

10 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

```python

from queue import Queue

dx = [-1, -1, -2, -2, 1, 1, 2, 2]
dy = [2, -2, 1, -1, 2, -2, 1, -1]


def horse(n, sx, sy, ex, ey):
    vis = [[False for i in  range(0, n)] for j in  range(0, n)]
    vis[sx][sy] = True

    def check(x, y, dx, dy):
        if vis[x + dx][y + dy]:
            return False

        if x + dx < 0 or x + dx >= n:
            return False

        if y + dy < 0 or y + dy >= n:
            return False

        return True

    q = Queue()
    t = 0
    q.put((t, sx, sy))
    while not q.empty():
        t, x, y = q.get()
        if x == ex and y == ey:
            return t

        for nx, ny in  zip(dx, dy):
            if check(x, y, nx, ny):
                q.put((t + 1, x + nx, y + ny))
                vis[x + nx][y + ny] = True

    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。