中山大学计算机学院算法设计与分析本科生实验报告

课程名称：算法设计与分析

教学班级	专业（方向）	学号	姓名
2班	计算机科学与技术	21307174	刘俊杰

一、实验内容

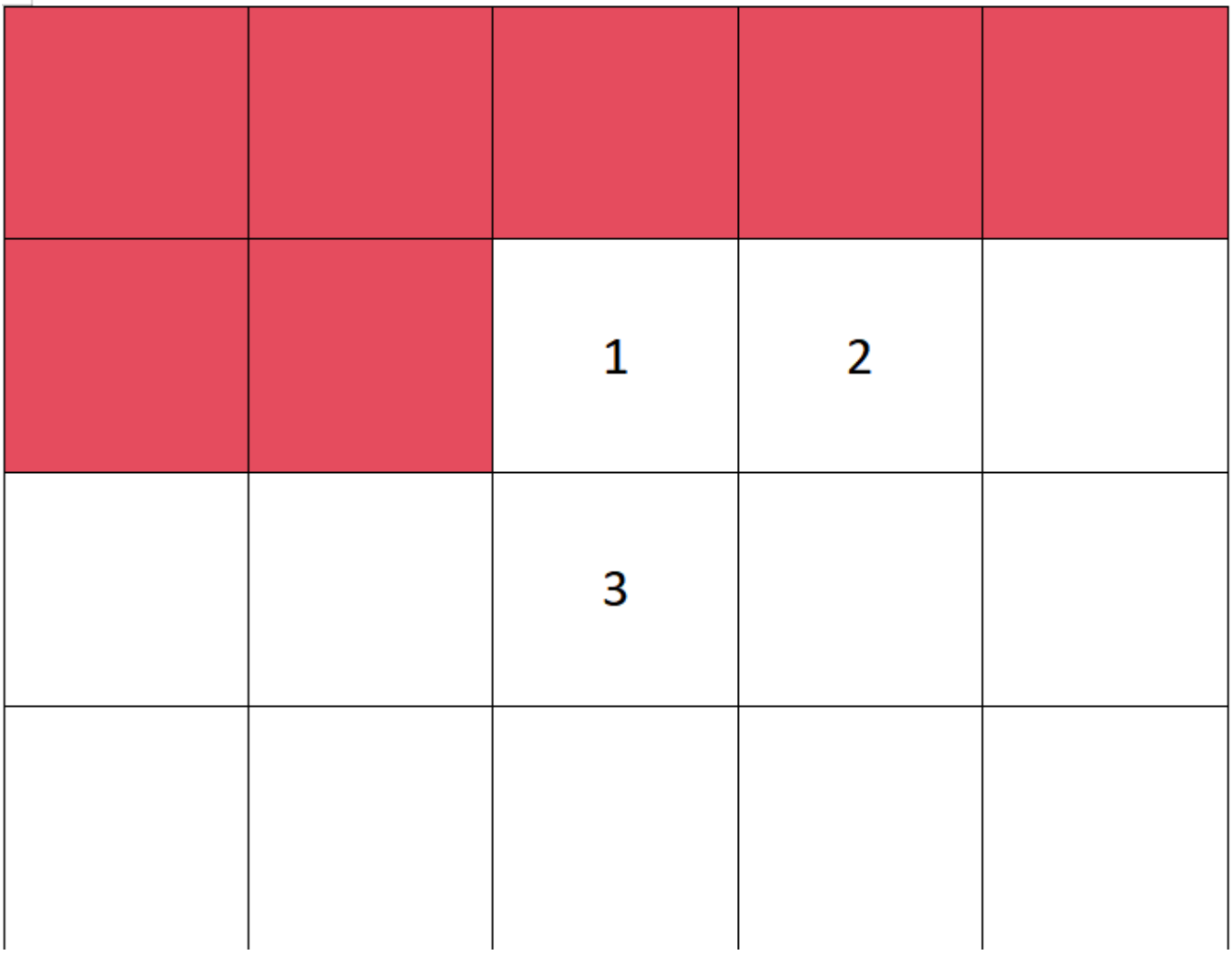
哨兵布置问题。一个博物馆由排成m*n个矩阵阵列的陈列室组成，需要在陈列室中设立哨位，每个哨位上的哨兵除了可以监视自己所在陈列室外，还可以监视他上、下、左、右四个陈列室，试基于分支限界法给出一个最佳哨位安排方法，使得所有陈列室都在监视之下，但使用的哨兵最少。

二、实验思想

1.搜索方法

首先思考最直接可以解决此问题的思想，就是考虑每一个房间放与不放，假设一共有n个房间，则对应不同的哨兵放置方式一共有2^n个，就需要判断这2^n个方法是否能满足题目要求，但这种方法的时间耗费太大。

我们每次搜索可以考虑每个房间是被哪个房间放置的哨兵所监督的:



如图所示，假设1号房间前所以的房间都已经被监督了而1号房间还未被监督，现在考虑如何放置哨兵能使得1号房间被监督:

能监督到1号房间的可以在其原位、左右或上下的位置放置哨兵。但由于在其上面只能多监督1号房间，放置在左边只能多监督1号房间和最多其余一个房间(这样不如放在3号房间的效果好)

所以我们可以选择在1或2或3号放置哨兵，于是我们便可以构建出搜索树，每一层的状态继续向下扩展时，可以向1或2或3号放置哨兵搜索。

2.回溯

在每次当前状态搜索结束，需要返回上一个状态时我们需要回溯操作，在这里即是取消本次搜索所放置的哨兵，同时使得在放置这个哨兵后才被监督的房间回溯到未被监督状态。

3.分支限界

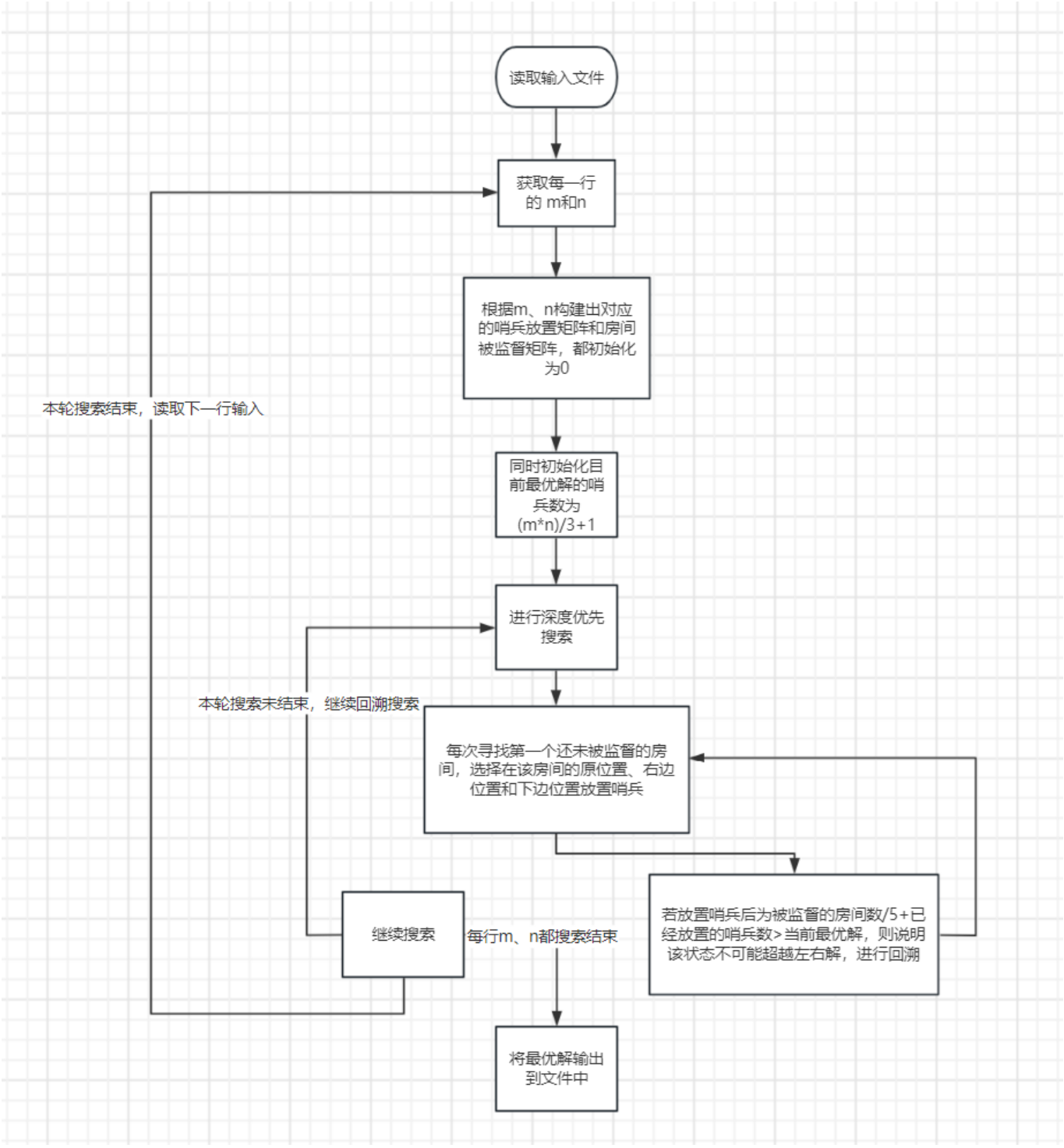
为了进一步加快搜索的速度，记录当前最优解的哨兵数为**B**。

为了提高搜索效率，使用分支限界，考虑状态函数**C=已放置的哨兵数+未监督的房间数/5**(状态下还需放置的哨兵最少的情况下，是还需放置的每个哨兵监督的房间不重叠)。

若**C**>**B**,则说明考虑当前状态在最好的情况下放置哨兵后总的哨兵数仍大于当前的最优解，说明此状态下不可能超过最优解，所以不需要继续在此状态下搜索，可以进行回溯。

为了进一步加速搜索的速度，我们可以在未搜索前初始化最优解为**m*n/3+1** (此初始最优解是在考虑每个哨兵只能监督包括自己在内的三个房间的最优解，由于本题目哨兵能监督五个位置，故本题的最优解不会大于**m*n/3+1**)

三、流程图



四、源代码

1.寻找第一个没有被监督的房间的位置

```
def where_not_supervise(supervise):#寻找第一个没有被监督的房间的位置
    for i in range(0,len(supervise)):
        for j in range(0,len(supervise[0])):
            if supervise[i][j]==0:
                return i,j
    return -1,-1
```

2、在row,col位置放置一个哨兵(录在这一步放置哨兵新增的被监督房间的位置,方便回溯)

```
def place(board,supervise,row,col):#在row,col位置放置一个哨兵
    board[row][col]=1
    # 同时需要记录在这一步放置哨兵新增的被监督房间的位置，方便后面的回溯操作
    m,n = len(board),len(board[0])
    dirs = [(0,0),(1,0),(-1,0),(0,1),(0,-1)]
    new_supervised = []#记录在这一步放置哨兵新增的被监督房间的位置
    for dir in dirs:
        new_row,new_col = row+dir[0],col+dir[1]
        if is_valid(m,n,new_row,new_col) and supervise[new_row][new_col]==0:
            supervise[new_row][new_col]=1
            new_supervised.append((new_row,new_col))
    return new_supervised
```

3.深度优先搜索(回溯+分支限界)

```
def dfs(board,supervise,best_cost,best_plan,cost,supervise_count):
    m,n = len(board),len(board[0])
    # 分支限界
    if cost+(m*n-supervise_count)//5>best_cost:
        return best_cost,best_plan
    #所有房间都已被监督
    if supervise_count == m*n:
        if cost < best_cost: # 与之前的最优解比较
            best_cost = cost
            best_plan = copy.deepcopy(board)
        return best_cost,best_plan

    # 寻找第一个未被监督的房间位置
    row,col = where_not_supervise(supervise)
    dirs = [(1,0),(0,0),(0,1)]
    # 在第一个未被监督的房间的原位置、右位置、下位置放置哨兵搜索
    for dir in dirs:
        new_row,new_col = row+dir[0],col+dir[1]
        if is_valid(m,n,new_row,new_col)==False:
            continue
        be_supervise = place(board,supervise,new_row,new_col)
        best_cost,best_plan =
dfs(board,supervise,best_cost,best_plan,cost+1,supervise_count+len(be_supervise))
        # 回溯操作，将此步放置哨兵的效果回退
        board[new_row][new_col] = 0
        for ele in be_supervise:
            supervise[ele[0]][ele[1]] = 0
    return best_cost,best_plan
```

完整代码

```
import copy
import time
def is_valid(m,n,row,col):#判断该位置是否合法
    if row<0 or col<0 or row>=m or col>=n:
        return False
    return True

def where_not_supervise(supervise):#寻找第一个没有被监督的房间的位置
    for i in range(0,len(supervise)):
        for j in range(0,len(supervise[0])):
            if supervise[i][j]==0:
                return i,j
    return -1,-1

def place(board,supervise,row,col):#在row,col位置放置一个哨兵
    board[row][col]=1
    # 同时需要记录在这一步放置哨兵新增的被监督房间的位置，方便后面的回溯操作
    m,n = len(board),len(board[0])
    dirs = [(0,0),(1,0),(-1,0),(0,1),(0,-1)]
    new_supervised = []#记录在这一步放置哨兵新增的被监督房间的位置
    for dir in dirs:
        new_row,new_col = row+dir[0],col+dir[1]
        if is_valid(m,n,new_row,new_col) and supervise[new_row][new_col]==0:
            supervise[new_row][new_col]=1
            new_supervised.append((new_row,new_col))
    return new_supervised

def dfs(board,supervise,best_cost,best_plan,cost,supervise_count):
    m,n = len(board),len(board[0])
    # 分支限界
    if cost+(m*n-supervise_count)//5>best_cost:
        return best_cost,best_plan
    #所有房间都已被监督
    if supervise_count == m*n:
        if cost < best_cost: # 与之前的最优解比较
            best_cost = cost
            best_plan = copy.deepcopy(board)
        return best_cost,best_plan

    # 寻找第一个未被监督的房间位置
    row,col = where_not_supervise(supervise)
    dirs = [(1,0),(0,0),(0,1)]
    # 在第一个未被监督的房间的原位置、右位置、下位置放置哨兵搜索
    for dir in dirs:
        new_row,new_col = row+dir[0],col+dir[1]
        if is_valid(m,n,new_row,new_col)==False:
            continue
        be_supervise = place(board,supervise,new_row,new_col)
        best_cost,best_plan =
dfs(board,supervise,best_cost,best_plan,cost+1,supervise_count+len(be_supervise))
        # 回溯操作，将此步放置哨兵的效果回退
        board[new_row][new_col] = 0
        for ele in be_supervise:
```

```
                supervise[ele[0]][ele[1]] = 0
    return best_cost,best_plan

def read_input(input):#读文件获取矩阵阵列的大小
    MN = []
    with open(input, 'r', encoding='utf-8') as f:
        for line in f:
            tmp = (line.strip()).split()
            MN.append((int(tmp[0]),int(tmp[1])))
    return MN

def write_output(best_cost,best_plan,output_path):#将结果输出到文件中
    with open(output_path, 'a') as f:
        f.write(str(best_cost)+'\n')
        for line in best_plan:
            f.write(str(line)+'\n')

def solve(MN,output_path):
    for mn in MN:
        time_start = time.time()
        m,n = mn[0],mn[1]
        best_cost = (m*n)//3+1 # 搜索过程中记录的最优放置哨兵数
        best_plan = None # 搜索过程中记录的最优放置方法
        board = [[0]*n for i in range(m)] # 哨兵放置矩阵
        supervised = [[0]*n for i in range(m)] # 记录房间是否被监督的状态
        best_cost,best_plan = dfs(board,supervised,best_cost,best_plan,0,0)
        time_end = time.time()
        time_c= time_end - time_start
        print('time cost', time_c, 's')
        write_output(best_cost,best_plan,output_path)


if __name__ == "__main__":
    input_path = "D:\d_code\\algorithm\Lab3\input.txt"
    output_path = "D:\d_code\\algorithm\Lab3\output.txt"
    MN = read_input(input_path)
    solve(MN,output_path)
```

五、输出结果

输入文件内容:

```
4 4
5 5
6 6
7 7
8 8
9 9
10 10
```

输出文件内容:

```
4
[0, 0, 1, 0]
[1, 0, 0, 0]
[0, 0, 0, 1]
[0, 1, 0, 0]
7
[0, 0, 0, 1, 0]
[1, 1, 0, 0, 0]
[0, 0, 0, 0, 1]
[0, 0, 1, 0, 0]
[1, 0, 0, 1, 0]
10
[0, 0, 0, 0, 1, 0]
[1, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0]
[1, 1, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0]
12
[0, 0, 1, 0, 1, 0, 0]
[1, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 1, 0, 0]
16
[0, 0, 0, 1, 0, 0, 1, 0]
[1, 1, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 1, 1]
[0, 1, 0, 0, 1, 0, 0, 0]
```

每一轮搜索时间:

```
time cost 0.0 s
time cost 0.06374382972717285 s
time cost 1.182725429534912 s
time cost 3.8839266300201416 s
time cost 63.34631085395813 s
```