

**实验报告**

**课程名称： 算法设计与分析**

**学 院： 应用技术学院**

**专业班级： 18计科一班**

**学 号： 201833050027**

**姓 名： 叶成宇**

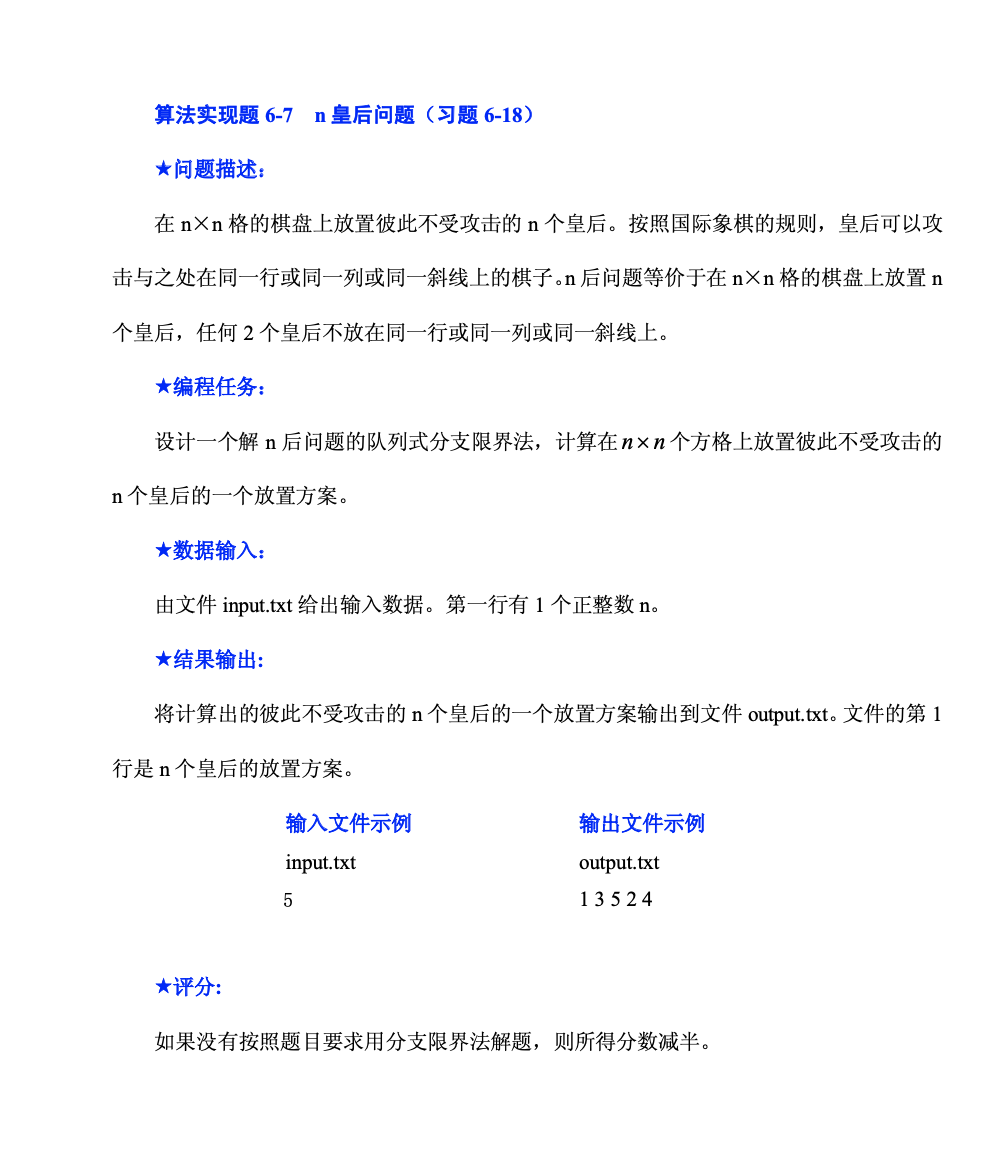
1．实验目的

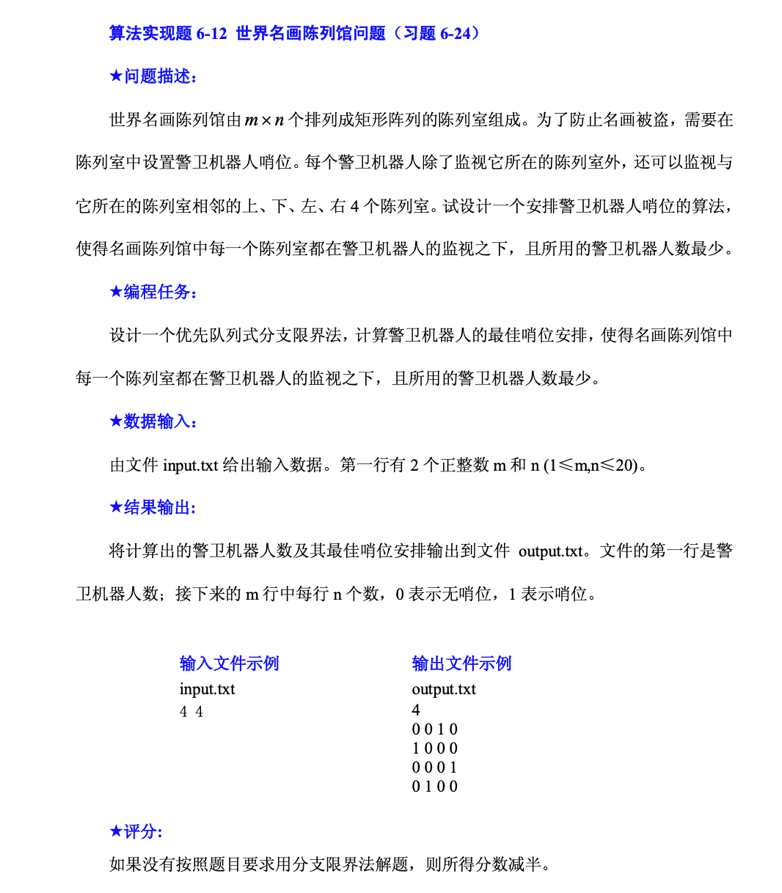
（1）在 n×n 格的棋盘上放置彼此不受攻击的 n 个皇后。按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。n 后问题等价于在 n×n 格的棋盘上放置 n 个皇后，任何 2 个皇后不放在同一行或同一列或同一斜线上。

（2）世界名画陈列馆由 m × n 个排列成矩形阵列的陈列室组成。为了防止名画被盗，需要在陈列室中设置警卫机器人哨位。每个警卫机器人除了监视它所在的陈列室外，还可以监视与 它所在的陈列室相邻的上、下、左、右 4 个陈列室。试设计一个安排警卫机器人哨位的算法， 使得名画陈列馆中每一个陈列室都在警卫机器人的监视之下，且所用的警卫机器人数最少。

2．实验内容

（1）

（2）



3．实验步骤和实验结果

（1）

#include<iostream>

#include<queue>

#include<cmath>

#include<vector>

#include<fstream>

#include<sstream>

using namespace std;

//定义一个节点类

struct Node{

int number;

vector<int>x;//保存当前解

};

//定义一个Queen的类

class Queen{

friend int nQueen(int);

public:

bool Place(Node q,int n);

void Research();

int n;//皇后个数

int \*bestx;//最优解

};

//判断是否能够放置的函数

bool Queen::Place(Node q,int n)

{

for(int j=1;j<n;j++)

if((abs(n-j)==abs(q.x[j]-q.x[n]))||(q.x[j]==q.x[n])) return false;

return true;

}

void Queen::Research()

{

queue<Node>Q;//活节点队列

Node sign;

sign.number=-1;

Q.push(sign);//同层节点尾部标志

int t=1;//当前节点所处的层

Node Ew;//当前扩展节点

Ew.number=0;

//搜索子集空间树

while(1){

//检查所有的孩子节点

for(int k=1;k<=n;k++){

//把当前扩展节点的值赋给下一个节点

Node q;

q.number=t;

q.x.push\_back(0);//第一个位置为0

for(int i=1;i<t;i++) q.x.push\_back(Ew.x[i]);

q.x.push\_back(k);

if(Place(q,t))

Q.push(q);

}

//取下一扩展节点,取出，赋值给Ew

Ew=Q.front();

Q.pop();

if(Ew.number==-1){

//同层节点尾部标记

t++;//进入下一层

Q.push(sign);//增加标记

//继续往下去下一个节点

Ew=Q.front();

Q.pop();

}

if(Ew.number==n){//找到最后一层的节点

for(int i=0;i<=n;i++) bestx[i]=Ew.x[i];

break;

}

}

}

int nQueen(int n,ofstream &outfile)

{

Queen X;

X.n=n;

X.bestx=new int[n+1];

for(int i=0;i<=n;i++) X.bestx[i]=0;

X.Research();

for(int i=1;i<=n;i++){

outfile<<X.bestx[i]<<" ";

}

}

int main(){

int N;

ifstream cinfile;

cinfile.open("/Users/ycy/Desktop/input.txt",ios::in);

cinfile>>N;

cinfile.close();

ofstream outfile;

outfile.open("/Users/ycy/Desktop/output.txt",ios::out);

nQueen(N,outfile);

outfile.close();

return 0;

}

（2）

#include<iostream>

#include<fstream>

using namespace std;

class Solution

{

int m,n;//矩阵行数、列数

int bestGuard,currentGuard,remainingRoom;//最佳警卫数，当前警卫数，余下房间数

int \*\*guardPlace,\*\*displayRoom,\*\*currentGuardPlace;//最佳警卫布置，受监视展览室情况，当前警卫布置

public:

void Backtrack(int i,int j,int\*\* displayRoom)

{

//继续向下检索，直到找到没有被监视的陈列室

while(displayRoom[i][j] >= 1){

j++;

if(j>=n){

i++;

j=0;

}

if(i==m)

break;

}

//查找到一组解，判断是否为最优，是则保留替换

if(i>=m){

if(currentGuard<bestGuard){

bestGuard=currentGuard;

for(int k=0;k<m;k++){

for(int l=0;l<n;l++){

guardPlace[k][l]=currentGuardPlace[k][l];

}

}

}

return ;

}

//当前警卫数+(还剩房间数+4)/5 >= bestGuard，则认为不会更佳，剪枝

if(currentGuard+(remainingRoom+4)/5 >= bestGuard){

return ;

}

if(i<m-1){//放置在(i+1,j)，保证i不是最后一行，此种布置法为最佳情况，无需限制

change(i+1,j);

Backtrack(i,j,displayRoom);

restore(i+1,j);

}

if(j<n-1){//放置在(i,j+1),保证j不是最后一行

if(i+1<=m-1){

if(displayRoom[i+1][j]>0){//(i,j+1)位置已经有了

change(i,j+1);

Backtrack(i,j,displayRoom);

restore(i,j+1);

}

}

else{

change(i,j+1);

Backtrack(i,j,displayRoom);

restore(i,j+1);

}

}

if(1){//此种布置法一定不会比放置在(i+1,j)(i,j+1)更好，共计四种情况要考虑

if(i+1<=m-1){//i+1未越界

if(displayRoom[i+1][j]>0){//判断(i+1,j)位置已经有了

if(j+1<=n-1){//j+1未越界

if(displayRoom[i][j+1]>0){//判断(i,j+1)位置已经有了

change(i,j);//(i+1,j)和(i,j+1)已经有了

Backtrack(i,j,displayRoom);

restore(i,j);

}

}

}

else{

change(i,j);//(i+1,j)位置已经有了，(i,j+1)位置不存在

Backtrack(i,j,displayRoom);

restore(i,j);

}

}

else{//i+1越界

if(j+1<=n-1){//判断j+1未越界

if(displayRoom[i][j+1]>0){//判断(i,j+1)位置已经有了

change(i,j);//(i+1,j)位置不存在，(i,j+1)位置已经有了

Backtrack(i,j,displayRoom);

restore(i,j);

}

}

else{//j+1越界

change(i,j);//(i+1,j)和(i,j+1)都不存在

Backtrack(i,j,displayRoom);

restore(i,j);

}

}

}

}

//初始化各变量

void init(void)

{

currentGuard=0;//当前警卫数

bestGuard=((m+2)/3)\*n;//初始设定的警卫数 假定每个警卫只覆盖三个

remainingRoom=m\*n;//剩余未监视房间数

//创建警卫位置二维数组并初始设定警卫位置

guardPlace = new int\*[m];

for(int i=0;i<m;i++){

guardPlace[i]=new int[n];

}

for(int i=0;i<m;i++){

for(int j=0;j<n;j++){

if( (i%3==1) || (i==m-1 && m%3==1)){//在第1、4、7、10……行全设为1 (从0开始),并考虑余下的行

guardPlace[i][j]=1;

}

else{

guardPlace[i][j]=0;

}

}

}

//创建展览室位置二维数组并将元素均初始化为0

displayRoom = new int\*[m];

for(int i=0;i<m;i++){

displayRoom[i]=new int[n];

}

for(int i=0;i<m;i++){

for(int j=0;j<n;j++){

displayRoom[i][j]=0;

}

}

//创建存放当前警卫布置情况的数组

currentGuardPlace = new int\*[m];

for(int i=0;i<m;i++){

currentGuardPlace[i]=new int[n];

}

for(int i=0;i<m;i++){

for(int j=0;j<n;j++){

currentGuardPlace[i][j]=0;

}

}

}

//输入函数，读入文件变量

void input(void)

{

freopen("/users/ycy/Desktop/input.txt","r",stdin);

scanf("%d %d",&m,&n);

fclose(stdin);

}

//输出函数，输出最佳结果并释放数组空间

void output(void)

{

FILE \*fp;

fp=fopen("/users/ycy/Desktop/output.txt","w");

fprintf(fp,"%d\n",bestGuard);

for(int i=0;i<m;i++){

for(int j=0;j<n;j++){

fprintf(fp,"%d ",guardPlace[i][j]);

}

fprintf(fp,"\n");

}

for(int i=0;i<m;i++){

delete [] guardPlace[i];

delete [] displayRoom[i];

delete [] currentGuardPlace[i];

}

}

int \*\*returnPointer(void)

{

return displayRoom;

}

private:

//撤销(i,j)处的机器人，并改变周围展览室状态及剩余未监视房间数

void restore(int i,int j)

{

currentGuardPlace[i][j]--;

currentGuard--;

displayRoom[i][j]--;

remainingRoom+=(!displayRoom[i][j]);

if(i+1<=m-1){

displayRoom[i+1][j]--;

remainingRoom+=(!displayRoom[i+1][j]);//cout<<"加了房子了！！！"<<endl;

}

if(j+1<=n-1){

displayRoom[i][j+1]--;

remainingRoom+=(!displayRoom[i][j+1]);//cout<<"加了房子了！！！"<<endl;

}

if(j-1>=0){

displayRoom[i][j-1]--;

remainingRoom+=(!displayRoom[i][j-1]);//cout<<"加了房子了！！！"<<endl;

}

if(i-1>=0){

displayRoom[i-1][j]--;

remainingRoom+=(!displayRoom[i-1][j]);//cout<<"加了房子了！！！"<<endl;

}

}

//在(i,j)放置一个机器人，并改变周围展览室状态及剩余未监视房间数

void change(int i,int j)

{

currentGuardPlace[i][j]++;

currentGuard++;

remainingRoom-=(!displayRoom[i][j]);

displayRoom[i][j]++;

if(j+1<=n-1){

remainingRoom-=(!displayRoom[i][j+1]);

displayRoom[i][j+1]++;

}

if(i+1<=m-1){

remainingRoom-=(!displayRoom[i+1][j]);

displayRoom[i+1][j]++;

}

if(i-1>=0){

remainingRoom-=(!displayRoom[i-1][j]);

displayRoom[i-1][j]++;

}

if(j-1>=0){

remainingRoom-=(!displayRoom[i][j-1]);

displayRoom[i][j-1]++;

}

}

};

int main(void)

{

Solution solution;

solution.input();

solution.init();

solution.Backtrack(0,0,solution.returnPointer());

solution.output();

}

4．分析与讨论

通过对n后问题以及世界名画陈列馆问题的算法的学习，使用了分支界限法进行求解。

分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树。

在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。

此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。