

课 程 实 验 报 告

**课程名称： Akari问题求解**

|  |  |
| --- | --- |
| 专业班级： | CS1607 |
| 学 号： | U201614702 |
| 姓 名： | 樊俊超 |
| 指导教师： | 陆枫 |
| 报告日期： | 2019/7/29 |

目录

[实验一 使用回溯法解决Akari问题 3](#_Toc15420907)

[1.1实验目的与要求 3](#_Toc15420908)

[1.2实验内容 3](#_Toc15420909)

[1.3实验设计 4](#_Toc15420910)

[实验二 Akari串行回溯算法的并行化 5](#_Toc15420911)

[2.1 实验目的与要求 5](#_Toc15420912)

[2.2 实验内容 5](#_Toc15420913)

[2.3 算法描述 6](#_Toc15420914)

[实验三 使用改进的并行回溯法求解问题 6](#_Toc15420915)

[3.1 实验目的与要求 6](#_Toc15420916)

[3.2 实验内容 6](#_Toc15420917)

[3.3 算法描述 7](#_Toc15420918)

[Akari问题体会 7](#_Toc15420919)

[附录 8](#_Toc15420920)

[实验一代码 8](#_Toc15420921)

[实验二代码 14](#_Toc15420922)

# 实验一 使用回溯法解决Akari问题

## 1.1实验目的与要求

1、了解Akari问题，探索其解决方法。

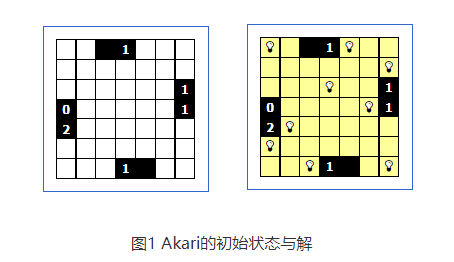
2、了解回溯法的概念，掌握回溯法的原理，并能够使用一种编程语言通过使用回溯法解决问题。

3、培养和锻炼分析问题与解决问题的能力。

## 1.2实验内容

Akari问题有时又被称为Light up或者Beleuchtung，源于日本逻辑解密游戏系列Nikoli，同属于Nikoli谜题的除Akari之外还有Sudoku(数独)和Kakuro(数谜)等。

游戏规则很简单。点灯游戏的棋盘是一张方形格网，其中的格子可能是黑色也可能是白色。游戏目标是在格网中放置灯泡，使之能照亮所有的白色方格。如果一个方格所在的同一行或同一列有一个灯泡，并且方格和灯泡之间没有黑色格子阻挡，那么这个方格将被灯泡照亮。同时，放置的每个灯泡不能被另一个灯泡照亮。某些黑色格子中标有数字。这些数字表示在该格子四周相邻的格子中共有多少个灯泡。Akari问题的示意如图1所示。



回溯法

回溯法是一种迭代算法。在回溯法中，首先将问题的解决分为若干步，其次通过枚举每一步的选择构造解空间树。在此基础上通过深度优先搜索遍历此解空间树，若当前节点的局部解不能构造出全局解，则向上回溯，否则向下扩展。重复此步骤直到找到全局解。

回溯法的关键点在于：

问题可分步并且可枚举每一步的选择。可以迅速的判断出当前局部解可否构造出全局解。若问题可分为N步，每一步有M种选择，易知其时间复杂度为O(m^n)O(mn )，虽然和穷举法有着相同的时间复杂度，但因为是逐步搜索并不断摒弃局部解，因此在实际应用中比穷举法效率高的多。

## 1.3实验设计

按照游戏规则，先根据黑色方格中数字大小不同枚举出所有的放置情况，

若黑色方格中的数字为4，则其相邻的周围格子的“灯泡”的放置方式为1种，即上下左右均放置一个“灯泡”。

若黑色方格中的数字为3，则其相邻的周围格子的“灯泡”的放置方式为4种，即在左右下、下右上、右上左、上左下的格子中放置“灯泡”。

若黑色方格中的数字为2，则其相邻的周围格子的“灯泡”的放置方式有6种，即在上左、上下、上右、左下、左右、下右的格子中放置“灯泡”。

若黑色方格中的数字为1，则其相邻的周围格子的“灯泡”的放置方式有4种，即在上、下、左、右的格子中放置“灯泡”。

若黑色方格中的数字为0，则其相邻的周围格子的“灯泡”的放置方式有1种，即所有周围格子均不放置。

为了防止规模增大后可能出现的堆栈溢出情况，在编程时选择采用尾递归的方法尽量减少递归层数太多引起的调用栈溢出。由于黑色数字格子的处理时间在整个程序中占的比例比白色格子的处理时间要少得多，所以只需要按照规则放置“灯泡”即可。

由于白色格子的处理时间比黑色数字格子的要长得多。则在实验过程中，白色格子应该这样放置，即在黑色带数字格子放置“灯泡”后，按照游戏规则遍历棋盘使得照亮的格子能显示。然后在仍未被照亮的格子中放置“灯泡”，进入递归，重复“照亮—寻找未被照亮白色格子—放置灯泡”步骤，直至无法再放置灯泡为止。若结果正确则返回，否则返回上一层，取消原本此层中对应格子中放置的“灯泡”，寻找另外的未被照亮的白色格子，放置“灯泡”。在白色格子放置“灯泡”的过程中，需要将上一层递归放置的“灯泡”的坐标传给当前递归，防止前面递归失效。代码见附录。

# 实验二 Akari串行回溯算法的并行化

## 2.1 实验目的与要求

1、掌握串行程序并行化的方法。

2、熟悉在Linux的C++语言环境下进行并行程序设计的方法。

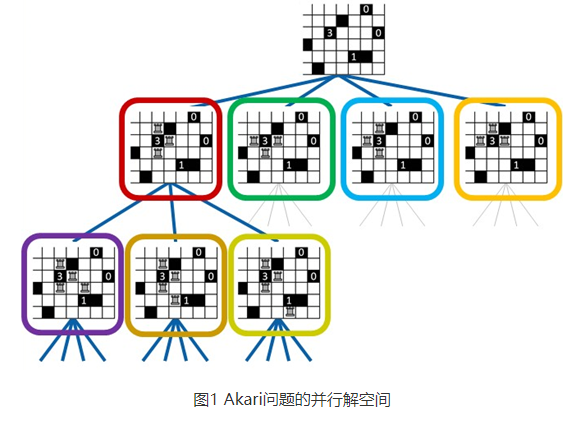
3、了解并行程序设计与并行算法设计的基本分析方法与途径。

## 2.2 实验内容

串行算法并行化的过程中的核心问题是问题分解和解除数据相关的问题。所谓的问题分解问题即将串行算法采取分治的理念分成可并行计算的子问题，解除数据相关的问题就是采取一定的冗余策略或锁策略使各子问题的私有数据之间不相互影响。

回溯法是易于并行化的典型算法之一，在Akari问题的回溯算法中，可以清楚的认识到回溯的具体路线的选择对产生解的正确性没有影响，因此该回溯法中每个节点的计算过程以及节点的分裂过程均是可以并行计算的。同时由算法的性质决定了每个解空间节点各自数据不会相互影响，没有任何数据相关。

并行的Akari回溯算法的示意图如图1所示。



图中彩色的矩形表示可以并行执行的单元，可以看出，解空间树中相同层的执行是可以并行化的，不同层之间的执行依然可以并行化。

## 2.3 算法描述

由于Akari的回溯解决方法非常适合使用并行编程技术，所以在上一次实验的基础上，利用pthread实现并行编程。在解空间树每一次分支选择的时候，选择pthread创建多个线程并行处理不同的分支，若其中一个分支（pthread）返回Akari问题的正确解，则结束线程并返回结果到上一层；若所有结果都不能返回正确结果，则返回上一层的选择分支。在实验中，每一次创建线程都要申请新的空间储存该分支选择的结果。所以使用多线程后空间复杂度是O()，在程序实践中内存直接溢出，无法继续运行。所以这里采用当遇到3号黑色方块时创建线程的方式，进行并行处理，详细代码见附录。

# 实验三 使用改进的并行回溯法求解问题

## 3.1 实验目的与要求

1、掌握优化并行程序的方法；

2、了解并行粒度与性能之间的关系，掌握通过并行粒度分析程序性能的方法；

3、了解并行程序设计与并行算法设计的基本分析方法与途径。

## 3.2 实验内容

**并行计算中的粒度与性能**

多线程应用的并行任务工作量大小（粒度）会对其并行性能产生很大影响。在分解一项应用使之适用于多线程处理时，编程时通常采用的方法是从逻辑上将问题分割成尽量多的并行任务，或者在并行任务内根据共享数据与执行顺序决定进行哪些必要的通信。由于分割任务、将任务分配给线程以及在任务之间进行数据通信（共享）涉及到一定的成本，编程者通常需要聚合或整合分割的任务，用于避免随之产生的开销，尽量实现应用高效运行。通过聚合分割的任务可确定并行任务的最佳粒度。

粒度通常与工作负载在线程之间的均衡程度有关。尽管平衡大量小型任务的工作负载更容易，但这样做却可能导致通信和同步等方面的并行开销过高。此时，编程者可以通过将小型任务整合成一项任务，增加每项任务的粒度（工作量）来减少并行开销。

**并行回溯法求解Akari问题时的并行粒度划分方法**

并行粒度的划分实质上是一个将工作负载（任务）与工作单元（线程或进程）对应的过程，在本实验过程中，对于Akari问题的粒度划分方法有以下几种：

每个解空间的节点分裂均产生新任务

当且仅当解空间树节点第一次分裂的时候产生新任务

当且仅当解空间树节点在数字“3”的时候分裂时产生新任务

## 3.3 算法描述

这里采用“当且仅当解空间树节点第一次分裂的时候产生新任务”的方式进行并行粒度划分。引入计数量cnt，当为第一次遇到黑色数字块时，分裂解空间。

具体代码不再贴出，与前两次相似。

# Akari问题体会

Akari问题的并行化我选择采用了pthread进行多线程编程，在实验过程中，暴露了很多问题，编写代码生疏，对指针的理解与掌握有所欠缺，导致花了很长时间来解决问题。

但这次实验能让我在恢复编程技巧的同时对并行相比于串行的优势有个清楚的认识。输入的数据规模越大，优势体现的也越明显，在编程的过程中，我也学会了换个角度思考问题。

# 附录

## 实验一代码

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cmath>

#define maxn 10

using namespace std;

int n,m,b,ans;

int mp[maxn][maxn];

int vis[maxn][maxn];

int jmp[maxn][maxn];

int light[maxn][maxn];

int dx[]= {-1,1,0,0};

int dy[]= {0,0,-1,1};

bool getanswer = false;

void showvis() // 中间输出 方便debug

{

int i,j;

printf("\n");

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

{

for(j=1; j<=m; j++)

{

printf("%d",vis[i][j]);

}

printf("\n");

}

}

bool isok(int tx,int ty) // 判断能否放灯

{

int i,j;

for(i=tx-1; i>=1; i--)

{

if(vis[i][ty]) return false;

else if(mp[i][ty]) break ;

}

for(i=tx+1; i<=n; i++)

{

if(vis[i][ty]) return false;

else if(mp[i][ty]) break ;

}

for(j=ty-1; j>=1; j--)

{

if(vis[tx][j]) return false;

else if(mp[tx][j]) break ;

}

for(j=ty+1; j<=m; j++)

{

if(vis[tx][j]) return false;

else if(mp[tx][j]) break ;

}

return true ;

}

void expand(int tx,int ty) // 扩展 将灯能照亮的地方都点亮

{

int i,j;

for(i=tx; i>=1; i--)

{

if(mp[i][ty]) break ;

else jmp[i][ty]=1;

}

for(i=tx; i<=n; i++)

{

if(mp[i][ty]) break ;

else jmp[i][ty]=1;

}

for(j=ty-1; j>=1; j--)

{

if(mp[tx][j]) break ;

else jmp[tx][j]=1;

}

for(j=ty+1; j<=m; j++)

{

if(mp[tx][j]) break ;

else jmp[tx][j]=1;

}

}

bool judge() // 判断搜出来的情况是否合理

{

int i,j,k,cnt;

memset(jmp,0,sizeof(jmp));

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

{

for(j=1; j<=m; j++)

{

if(vis[i][j]) expand(i,j); // 将标记了的都扩展

}

}

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

{

for(j=1; j<=m; j++)

{

if(!mp[i][j]&&!jmp[i][j]) return false ; // 如果还有一个位置没亮的话就不行

}

}

// showvis();

for(i=1; i<=n; i++) // 检测墙的周围灯的个数是否满足条件

{

for(j=1; j<=m; j++)

{

if(light[i][j]==-1) continue ;

cnt=0;

for(k=0; k<4; k++)

{

if(vis[i+dx[k]][j+dy[k]]) cnt++;

}

if(cnt!=light[i][j]) return false ;

}

}

return true ;

}

bool midcut(int tx,int ty) // 中间剪枝 如果有墙的周围不满足条件了 就 return false

{

int i,j,k,cnt;

for(i=1; i<=tx-2; i++)

{

for(j=1; j<=m; j++)

{

if(light[i][j]==-1) continue ;

cnt=0;

for(k=0; k<4; k++)

{

if(vis[i+dx[k]][j+dy[k]]) cnt++;

}

if(cnt!=light[i][j]) return false ;

}

}

i=tx-1;

for(j=1; j<=ty-1; j++)

{

if(light[i][j]==-1) continue ;

cnt=0;

for(k=0; k<4; k++)

{

if(vis[i+dx[k]][j+dy[k]]) cnt++;

}

if(cnt!=light[i][j]) return false ;

}

return true ;

}

bool goodcut(int tx,int ty) // 强剪枝 如果所搜点为墙 且其上方为黑暗的，则剪枝！

{

int i,j,k,cnt;

memset(jmp,0,sizeof(jmp));

for(i=1; i<=tx-1; i++)

{

for(j=1; j<=m; j++)

{

if(vis[i][j]) expand(i,j);

}

}

if(!jmp[tx-1][ty]) return false;

return true ;

}

void dfs(int nx,int ny,int cxx)

{

if(getanswer) return ;

if(!midcut(nx,ny)) return ;

if(nx>=n&&ny>m||nx>n)

{

if(judge()) // 全部搜完后判断情况是否合理

{

getanswer=true;

}

return ;

}

if(ny>m) dfs(nx+1,1,cxx);

else if(mp[nx][ny])

{

if(nx>=2&&!mp[nx-1][ny]&&!goodcut(nx,ny)) return ; // 注意前面的条件

dfs(nx,ny+1,cxx);

}

else

{

dfs(nx,ny+1,cxx);

if(!vis[nx][ny]&&isok(nx,ny))

{

vis[nx][ny]=1;

dfs(nx,ny+1,cxx+1);

vis[nx][ny]=0;

}

}

}

int main()

{

int i,j,r,c,k;

while(scanf("%d%d",&n,&m),n||m)

{

scanf("%d",&b);

memset(mp,0,sizeof(mp));

memset(light,-1,sizeof(light));

for(i=1; i<=b; i++)

{

scanf("%d%d%d",&r,&c,&k);

mp[r][c]=1;

light[r][c]=k;

}

memset(vis,0,sizeof(vis));

ans=1000000;

dfs(1,1,0);

if(ans<1000000) printf("%d\n",ans);

else printf("No solution\n");

}

return 0;

}

## 实验二代码

# include <bits/stdc++.h>

# include "akari.h"

using namespace std;

namespace aka {

struct Arg{

int x, y, tag, i, j;

vector<vector<int>> \*next;

};

void \*thread(void \*arg);

bool Akari\_Res(vector<vector<int>>& res, int x, int y);

bool is\_puzzlecanbelighted(vector<vector<int>> res, int x, int y);

bool is\_SquareNotFull(vector<vector<int>>& res, int x, int y)

{

int num = res[x][y];

if (num < 0)

return true;

else if (num == 0)

return false;

else {

int cnt = 0;

if ((x - 1) >= 0 && res[x - 1][y] == 5)

++cnt;

if ((x + 1) <= res.size() - 1 && res[x + 1][y] == 5)

++cnt;

if ((y - 1) >= 0 && res[x][y - 1] == 5)

++cnt;

if ((y + 1) <= res[x].size() - 1 && res[x][y + 1] == 5)

++cnt;

if (cnt <= num)

return true;

else

return false;

}

}

bool is\_SquareFull(vector<vector<int>>& res, int x, int y)

{

int num = res[x][y];

if (num < 0)

return true;

else {

int cnt = 0;

if ((x - 1) >= 0 && res[x - 1][y] == 5)

++cnt;

if ((x + 1) <= res.size() - 1 && res[x + 1][y] == 5)

++cnt;

if ((y - 1) >= 0 && res[x][y - 1] == 5)

++cnt;

if ((y + 1) <= res[x].size() - 1 && res[x][y + 1] == 5)

++cnt;

if (cnt == num)

return true;

else

return false;

}

}

bool is\_Square(vector<vector<int>>& res, int x, int y)

{

if ((x - 1) >= 0 && res[x - 1][y] >= 0 && res[x - 1][y] <= 5)

return false;

if ((x + 1) <= res.size() - 1 && res[x + 1][y] >= 0 && res[x + 1][y] <= 5)

return false;

if ((y - 1) >= 0 && res[x][y - 1] >= 0 && res[x][y - 1] <= 5)

return false;

if ((y + 1) <= res[x].size() - 1 && res[x][y + 1] >= 0 && res[x][y + 1] <= 5)

return false;

return true;

}

void lighted(vector<vector<int>>& tmp, int x, int y)

{

for (int i = x - 1; i >= 0; --i) {

if (tmp[i][y] == -2)

tmp[i][y] = 10; // lighted

else if (tmp[i][y] >= -1 && tmp[i][y] < 5)

break;

else

continue;

}

for (int i = x + 1; i < tmp.size(); ++i) {

if (tmp[i][y] == -2)

tmp[i][y] = 10; // lighted

else if (tmp[i][y] >= -1 && tmp[i][y] < 5)

break;

else

continue;

}

for (int j = y - 1; j >= 0; --j) {

if (tmp[x][j] == -2)

tmp[x][j] = 10; // lighted

else if (tmp[x][j] >= -1 && tmp[x][j] < 5)

break;

else

continue;

}

for (int j = y + 1; j < tmp[x].size(); ++j) {

if (tmp[x][j] == -2)

tmp[x][j] = 10; // lighted

else if (tmp[x][j] >= -1 && tmp[x][j] < 5)

break;

else

continue;

}

}

bool notDeadEnd(vector<vector<int>>& res, int row, int col)

{

int lighted = 0;

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

if (res[i][col] == 5)

lighted = 1;

else if (res[i][col] >= -1)

lighted = 0;

else

continue;

}

if (lighted == 1)

return false;

for (int i = res.size() - 1; i > row; --i) {

if (res[i][col] == 5)

lighted = 1;

else if (res[i][col] >= -1)

lighted = 0;

else

continue;

}

if (lighted == 1)

return false;

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

if (res[row][j] == 5)

lighted = 1;

else if (res[row][j] >= -1)

lighted = 0;

else

continue;

}

if (lighted == 1)

return false;

for (int j = res[row].size() - 1; j > col; --j) {

if (res[row][j] == 5)

lighted = 1;

else if (res[row][j] >= -1)

lighted = 0;

else

continue;

}

if (lighted == 1)

return false;

if (row == 0) {

if (col == 0) {

if (!is\_SquareNotFull(res, row + 1, col))

return false;

if (!is\_SquareNotFull(res, row, col + 1))

return false;

}

else if (col == res[row].size() - 1) {

if (!is\_SquareNotFull(res, row + 1, col))

return false;

if (!is\_SquareNotFull(res, row, col - 1))

return false;

}

else {

if (!is\_SquareNotFull(res, row + 1, col))

return false;

else if (!is\_SquareNotFull(res, row, col - 1))

return false;

else if (!is\_SquareNotFull(res, row, col + 1))

return false;

}

}

else if (row == res.size() - 1) {

if (col == 0) {

if (!is\_SquareNotFull(res, row - 1, col))

return false;

if (!is\_SquareNotFull(res, row, col + 1))

return false;

}

else if (col == res[row].size() - 1) {

if (!is\_SquareNotFull(res, row - 1, col))

return false;

if (!is\_SquareNotFull(res, row, col - 1))

return false;

}

else {

if (!is\_SquareNotFull(res, row - 1, col))

return false;

else if (!is\_SquareNotFull(res, row, col - 1))

return false;

else if (!is\_SquareNotFull(res, row, col + 1))

return false;

}

}

else if (col == 0) {

if (!is\_SquareNotFull(res, row - 1, col))

return false;

else if (!is\_SquareNotFull(res, row + 1, col))

return false;

else if (!is\_SquareNotFull(res, row, col + 1))

return false;

}

else if (col == res[row].size() - 1) {

if (!is\_SquareNotFull(res, row - 1, col))

return false;

else if (!is\_SquareNotFull(res, row + 1, col))

return false;

else if (!is\_SquareNotFull(res, row, col - 1))

return false;

}

else {

if (!is\_SquareNotFull(res, row - 1, col))

return false;

else if (!is\_SquareNotFull(res, row + 1, col))

return false;

else if (!is\_SquareNotFull(res, row, col - 1))

return false;

else if (!is\_SquareNotFull(res, row, col + 1))

return false;

}

return true;

}

bool is\_DeadEnd(vector<vector<int>>& res)

{

for (int i = 0; i < res.size(); ++i) {

for (int j = 0; j < res[i].size(); ++j) {

if (res[i][j] == 5) {

if (!notDeadEnd(res, i, j))

return false;

else

continue;

}

else

continue;

}

}

return true;

}

bool is\_Solution(vector<vector<int>>& res)

{

vector<vector<int>> tmp(res);

for (int i = 0; i < tmp.size(); ++i) {

for (int j = 0; j < tmp[i].size(); ++j) {

if (tmp[i][j] == 5)

lighted(tmp, i, j);

else if (tmp[i][j] >= -1 && tmp[i][j] < 5) {

if (!is\_SquareFull(tmp, i, j))

return false;

else

continue;

}

else

continue;

}

}

for (int i = 0; i < tmp.size(); ++i) {

for (int j = 0; j < tmp[i].size(); ++j) {

if (tmp[i][j] == -2)

return false;

else

continue;

}

}

vector<vector<int>>().swap(tmp);

return true;

}

void full\_Square(vector<vector<int>>& res, int x, int y, int tag)

{

if (res[x][y] == -2)

res[x][y] = 5;

else {

switch (res[x][y]) {

case 4: res[x - 1][y] = res[x + 1][y] = res[x][y - 1] = res[x][y + 1] = 5; break;

case 3: {

switch (tag) {

case 0: res[x][y - 1] = res[x - 1][y] = res[x][y + 1] = 5; break;

case 1: res[x - 1][y] = res[x][y + 1] = res[x + 1][y] = 5; break;

case 2: res[x][y + 1] = res[x + 1][y] = res[x][y - 1] = 5; break;

case 3: res[x + 1][y] = res[x][y - 1] = res[x - 1][y] = 5; break;

default: break;

}

}

break;

case 2: {

switch (tag) {

case 0: res[x - 1][y] = res[x][y - 1] = 5; break;

case 1: res[x - 1][y] = res[x + 1][y] = 5; break;

case 2: res[x - 1][y] = res[x][y + 1] = 5; break;

case 3: res[x][y - 1] = res[x + 1][y] = 5; break;

case 4: res[x][y - 1] = res[x][y + 1] = 5; break;

case 5: res[x + 1][y] = res[x][y + 1] = 5; break;

default: break;

}

}

break;

case 1: {

switch (tag) {

case 0: res[x][y - 1] = 5; break;

case 1: res[x - 1][y] = 5; break;

case 2: res[x][y + 1] = 5; break;

case 3: res[x + 1][y] = 5; break;

default: break;

}

}

break;

case 0: break;

default: break;

}

}

}

bool is\_puzzlecanbelighted(vector<vector<int>> view, int x, int y)

{

for (int i = x - 1; i >= 0; --i) {

if (view[i][y] == -2) {

if (is\_Square(view, i, y))

return true;

else

continue;

}

else if (view[i][y] >= -1 && view[i][y] < 5)

break;

else

continue;

}

for (int i = x + 1; i < view.size(); ++i) {

if (view[i][y] == -2) {

if (is\_Square(view, i, y))

return true;

else

continue;

}

else if (view[i][y] >= -1 && view[i][y] < 5)

break;

else

continue;

}

for (int j = y - 1; j >= 0; --j) {

if (view[x][j] == -2) {

if (is\_Square(view, x, j))

return true;

else

continue;

}

else if (view[x][j] >= -1 && view[x][j] < 5)

break;

else

continue;

}

for (int j = y + 1; j < view[x].size(); ++j) {

if (view[x][j] == -2) {

if (is\_Square(view, x, j))

return true;

else

continue;

}

else if (view[x][j] >= -1 && view[x][j] < 5)

break;

else

continue;

}

return false;

}

bool nextmove(vector<vector<int>>& res, int x, int y)

{

int tag = 0;

vector<vector<int>> tmp(res);

for (int i = 0; i < tmp.size(); ++i) {

for (int j = 0; j < tmp.size(); ++j) {

switch (tmp[i][j]) {

case 4: {

if (!is\_SquareFull(tmp, i, j)) {

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

break;

}

}

case 3: {

if (!is\_SquareFull(tmp, i, j)) {

if ((i == 0) || (tmp[i - 1][j] >= -1 && tmp[i - 1][j] < 5)) {

tag = 2;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if ((i == tmp.size() - 1) || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

tag = 0;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if ((j == 0) || (tmp[i][j - 1] >= -1 && tmp[i][j - 1] < 5)) {

tag = 1;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if ((j == tmp[i].size() - 1) || (tmp[i][j + 1] >= -1 && tmp[i][j + 1] < 5)) {

tag = 3;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

if (i - 2 >= 0 && !tmp[i - 2][j]) {

tag = 2;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if (i + 2 <= res.size() - 1 && !tmp[i + 2][j]) {

tag = 0;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if (j - 2 >= 0 && !tmp[i][j - 2]) {

tag = 1;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if (j + 2 <= tmp[i].size() && !tmp[i][j + 2]) {

tag = 3;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

pthread\_t pid[4];

Arg \*arg = new Arg [4];

vector<vector<int>> \*g\_tmp = new vector<vector<int>> [4];

for (int k = 0; k < 4; ++k) {

g\_tmp[k] = res;

arg[k].x = x;

arg[k].y = y;

arg[k].i = i;

arg[k].j = j;

arg[k].tag = k;

arg[k].next = &g\_tmp[k];

if (pthread\_create(&pid[k], NULL, thread, &arg[k])) {

delete [] arg;

delete [] g\_tmp;

cout << "pthread create error" << endl;

exit(-1);

}

}

for (int k = 0; k < 4; ++k) {

pthread\_join(pid[k], NULL);

if (is\_Solution(g\_tmp[k])) {

res = g\_tmp[k];

delete [] arg;

delete [] g\_tmp;

return true;

}

}

delete [] arg;

delete [] g\_tmp;

return false;

}

}

}

else {

break;

}

}

case 2: {

if (!is\_SquareFull(tmp, i, j)) {

if (j == 0 || (tmp[i][j - 1] >= -1 && tmp[i][j - 1] < 5)) {

if (i == 0 || (tmp[i - 1][j] >= -1 && tmp[i - 1][j] < 5)) {

tag = 5;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if (j == tmp[i].size() - 1 || (tmp[i][j + 1] >= -1 && tmp[i][j + 1] < 5)) {

tag = 1;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

tag = 2;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

while (1) {

switch (tag) {

case 0: tag = 1; full\_Square(tmp, i, j, tag); break;

case 3:

case 4: tag = 5; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 5)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (i == 0 || (tmp[i - 1][j] >= -1 && tmp[i - 1][j] < 5)) {

if (j == tmp[i].size() - 1 || (tmp[i][j + 1] >= -1 && tmp[i][j + 1] < 5)) {

tag = 3;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

tag = 4;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

while (1) {

switch (tag) {

case 0:

case 1:

case 2: tag = 3; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 5)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (j == tmp[i].size() - 1 || (tmp[i][j + 1] >= -1 && tmp[i][j + 1] < 5)) {

if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

tag = 0;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

while (1) {

switch (tag) {

case 2: tag = 3; full\_Square(tmp, i, j, tag); break;

case 4:

case 5: tag = 0; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 3)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

while (1) {

switch (tag) {

case 1: tag = 2; full\_Square(tmp, i, j, tag); break;

case 3: tag = 4; full\_Square(tmp, i, j, tag); break;

case 5: tag = 0; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 4)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

else {

while (1) {

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 5)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else {

break;

}

}

case 1: {

if (!is\_SquareFull(tmp, i, j)) {

if (j == 0 || (tmp[i][j - 1] >= -1 && tmp[i][j - 1] < 5)) {

if (i == 0 || (tmp[i - 1][j] >= -1 && tmp[i - 1][j] < 5)) {

if (j == tmp[i].size() - 1 || (tmp[i][j + 1] >= -1 && tmp[i][j + 1] < 5)) {

tag = 3;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

tag = 2;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

while (1) {

switch (tag) {

case 0:

case 1: tag = 2; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 3)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (j == tmp[i].size() - 1 || (tmp[i][j + 1] >= -1 && tmp[i][j + 1] < 5)) {

if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

tag = 1;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

while (1) {

switch (tag) {

case 0: tag = 1; full\_Square(tmp, i, j, tag); break;

case 2: tag = 3; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 3)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

while (1) {

switch (tag) {

case 0:

case 3: tag = 1; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 2)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

else {

while (1) {

switch (tag) {

case 0: tag = 1; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 3)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (i == 0 || (tmp[i - 1][j] >= -1 && tmp[i - 1][j] < 5)) {

if (j == tmp[i].size() - 1 || (tmp[i][j + 1] >= -1 && tmp[i][j + 1] < 5)) {

if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

tag = 0;

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else {

return false;

}

}

else {

while (1) {

switch (tag) {

case 1:

case 2: tag = 3; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 3)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

while (1) {

switch (tag) {

case 1: tag = 2; full\_Square(tmp, i, j, tag); break;

case 3: tag = 0; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 2)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

else {

while (1) {

switch (tag) {

case 1: tag = 2; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 3)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (j == tmp[i].size() - 1 || (tmp[i][j + 1] >= -1 && tmp[i][j + 1] < 5)) {

if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

while (1) {

switch (tag) {

case 2:

case 3: tag = 0; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 1)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

else {

while (1) {

switch (tag) {

case 2: tag = 3; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 3)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else if (i == tmp.size() - 1 || (tmp[i + 1][j] >= -1 && tmp[i + 1][j] < 5)) {

while (1) {

switch (tag) {

case 3: tag = 0; full\_Square(tmp, i, j, tag); break;

default: full\_Square(tmp, i, j, tag); break;

}

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 2)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

else {

while (1) {

full\_Square(tmp, i, j, tag);

if (Akari\_Res(tmp, x, y)) {

res = tmp;

return true;

}

else if (tag == 3)

return false;

tmp.clear();

tmp = res;

++tag;

}

}

}

else {

break;

}

}

default:break;

}

}

}

int cnt = 0;

int time = 0;

int xs = x, ys = y;

vector<vector<int>> view(res);

for (int i = 0; i < view.size(); ++i) {

for (int j = 0; j < view[i].size(); ++j) {

if (view[i][j] == 5) {

lighted(view, i, j);

}

else {

continue;

}

}

}

while (1) {

for (int i = x; i < tmp.size(); ++i) {

int j;

if (x == i)

j = y;

else

j = 0;

for (; j < tmp[i].size(); ++j) {

if (view[i][j] == -2) {

if (cnt == tag && !time) {

if (is\_Square(tmp, i, j)) {

tmp[i][j] = 5;

++time;

xs = i;

ys = j;

}

else {

if (is\_puzzlecanbelighted(view, i, j)) {

++tag;

++cnt;

}

else

return false;

}

}

else

++cnt;

}

else

continue;

}

}

if (!time)

return false;

else {

if (Akari\_Res(tmp, xs, ys)) {

res = tmp;

return true;

}

else if (tag == cnt)

return false;

tmp.clear();

tmp = res;

time = 0;

cnt = 0;

++tag;

}

}

}

bool Akari\_Res(vector<vector<int>>& res, int x, int y)

{

if (!is\_DeadEnd(res))

return false;

if (is\_Solution(res))

return true;

return nextmove(res, x, y);

}

vector<vector<int>> solveAkari(vector<vector<int>> &g)

{

Akari\_Res(g, 0, 0);

return g;

}

void \*thread(void \*arg)

{

Arg \*par\_arg = (Arg \*)arg;

full\_Square(\*(par\_arg->next), par\_arg->i, par\_arg->j, par\_arg->tag);

Akari\_Res(\*(par\_arg->next), par\_arg->x, par\_arg->y);

}

}