# 实验四：回溯算法

1、问题描述

给定一个 1000 行 20列 的 0-1 矩阵，找出一对互斥集合 A 和 B，使得 A 和 B 包含的列的总数最大，其中互斥是指对于矩阵的任意一行，A 和 B 中至少有一个集合在该行上全为 0 或全为1。

输入格式：

一个 1000 行 20 列的矩阵 matrix 作为测试用例，每行输入 20 列的元素，对矩阵找出满足题意的互斥集合。

输出格式：

输出两行，使得 A 的元素+B 的元素个数最大

第一行输出 A 集合中的所有元素（下标从 0 开始），以空格分开

第二行输出 B 集合中的所有元素，格式同上

如果，没有找到非空集合 A 和 B，则输出两行空行

2、问题分析

首先，要确定互斥集合的概念。若两个集合 A 和 B 没有交集，即 A\ B=空，则称两个集合 A 和 B 是互斥的。

我们可以用深度优先搜索（DFS）的思路解决这个问题。首先从第 1 列开始，枚举选不选这一列，若选择了该列，则在矩阵的后续每一行中，只能选另一个集合中的列；若不选这一列，则在矩阵的后续每一行中，可以任意选。

假设当前已经枚举到第 i 列， A 集合已选了 p 列，B 集合已选了 q 列，我们要做以下判断：

如果矩阵的后 20-i 列都选择 B 中的列，那么剩余的 A 集合的选择的最大数量为 20-p，所以，如果 20-i+q+p<bestcolumn（bestcolumn 表示当前最大的互斥集合中包含的列数），则没有必要继续搜索。

如果当前已选择的互斥集合中， A 和 B 集合的元素个数之和小于 bestcolumn，则也没有必要继续搜索。

如果 B 集合或 A 集合已经为空，或矩阵的所有行都包含了 A 和 B 中的元素，则记录下当前的 A 和 B 集合，更新 bestcolumn。

如果当前的 A 和 B 集合的列数之和等于 bestcolumn，则要根据一定的规则选择更新 A 和 B 集合。这里的规则有三个：（1）AB列数差绝对值最小；（2）A的列数大于B的列数；（3）使A B的首元素在列数和相同的情况下最小。

3、算法设计

回溯算法的基本思路是，在搜索树中对每个节点进行搜索，如果当前搜索的节点不符合要求，回溯到上一个节点，在搜索到的过程中，使用一些技巧来避免不必要的搜索。在本题中，可以设计一棵搜索树，以列为节点，每个节点有两个状态，即将该列添加到集合 A 或集合 B 中。对于每一个节点，通过判断当前的选择是否符合要求，以及当前搜索到的节点是否可能超过目前找到的最优解，从而决定是否继续进行搜索。

4、算法实现

设定变量，记录当前找到的最优解和集合 A 和 B 的选择情况。

从根节点开始，遍历搜索树，对于每个节点进行如下操作：

判断当前选择是否合法，即是否与已选中的列互斥。

根据上一次选择的结果，判断当前选择是否有必要进行。

根据已选中的列的数量和当前剩余未选择的列的数量，判断是否有可能超过目前找到的最优解。

如果选择当前列加入 A 集合，继续向下遍历搜索树。

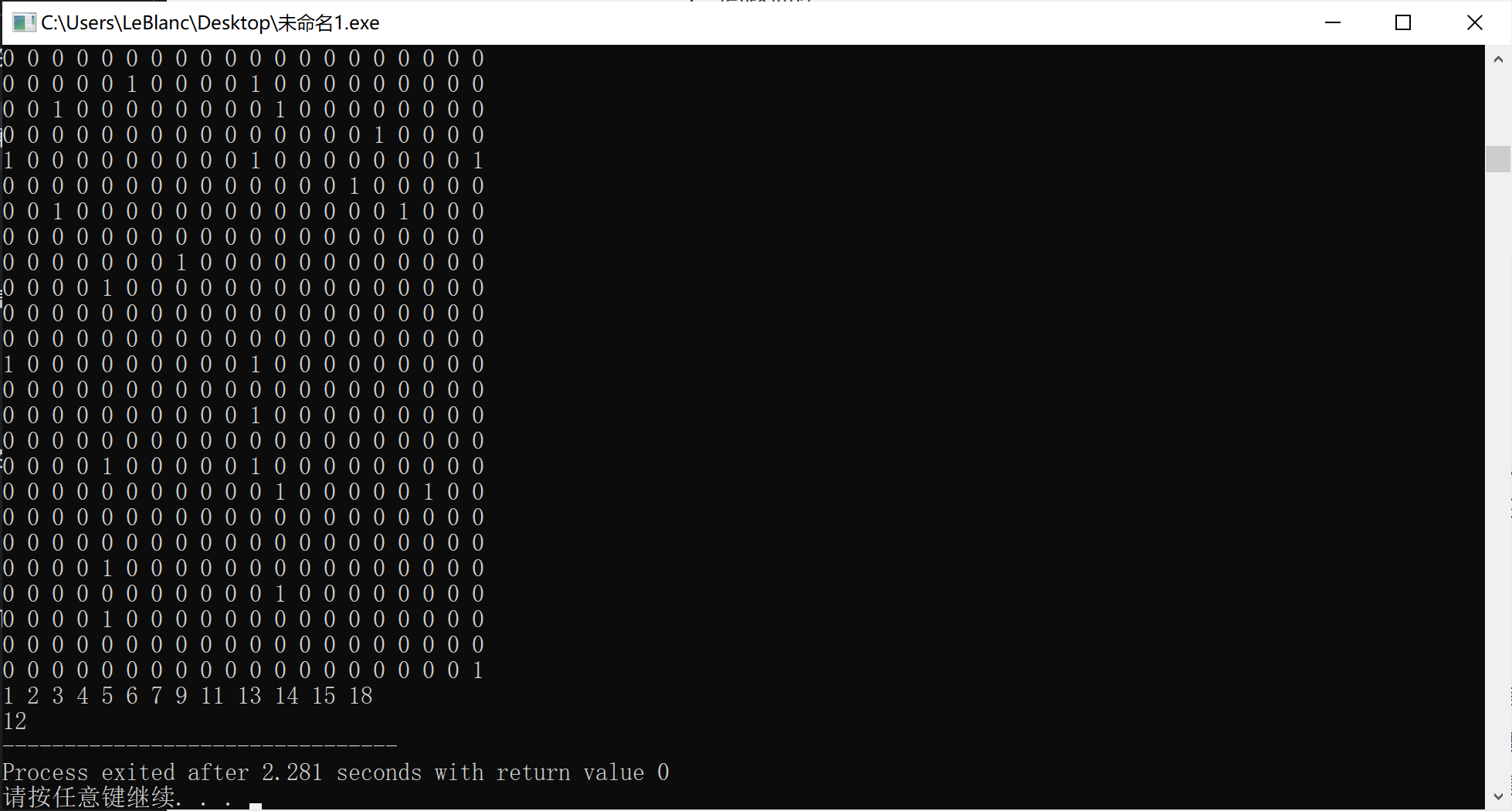
如果选择当前列加入 B 集合，继续向下遍历搜索树。

如果遍历完所有节点，找到了更优的解，则更新最优解和集合 A 和 B 的选择情况。

返回集合 A 和 B 的选择情况，输出结果

5、运行结果

输入用例：平台提供的用例



6.、实验总结

回溯算法是一种很有用的算法，在很多场合都能够发挥出作用。本题中，我们使用回溯算法寻找满足条件的 A 和 B 集合，将复杂的问题简化成了多个简单的子问题，通过不断地回溯，最终得到了最优解。

回溯算法是一种思路清晰、代码简洁的算法，但在解决某些问题时，可能会因为状态数太多而导致运行时间过长或者内存不足，需要进一步优化算法，或者采用其他更为高效的算法来解决问题。

7、附录

#include<iostream>

using namespace std;

int judge[20][20];

int arr[1000][20];

int bestcolumn = 0; // record the best ans

int a[20], b[20]; // record the chosen column of A or B

int ansA[20], ansB[20];

void dfs(int column, int nowcoA, int nowcoB)

{

if(20-column + nowcoB + nowcoA < bestcolumn)

return; // last column is less than bestcolumn

if(column == 20 && nowcoA + nowcoB >= bestcolumn && nowcoB != 0 && nowcoA != 0)

{ int i,nowfirstA=0,nowfirstB=0,bestfirstA=0,bestfirstB=0;

int a1=0,a2=0,b1=0,b2=0;

for(i=0;i<20;i++)

{

if(ansA[i]==1)a1++;

if(a[i]==1)a2++;

if(ansB[i]==1)b1++;

if(b[i]==1)b2++;

}

for(i=0;i<20;i++)

{

if(nowfirstA&&nowfirstB&&bestfirstA&&bestfirstB)break;

if(a[i]==1&&nowfirstA==0)nowfirstA=i;

if(b[i]==1&&nowfirstB==0)nowfirstB=i;

if(ansA[i]==1&&bestfirstA==0)bestfirstA=i;

if(ansB[i]==1&&bestfirstB==0)bestfirstB=i;

}

if((nowcoA+nowcoB>bestcolumn&&a2>b2)||(nowcoA+nowcoB==bestcolumn&&a2>b2&&a1-b1>=a2-b2&&nowfirstA<=bestfirstA&&nowfirstB<=bestfirstB))

{

int j, ok = 1;

for(i = 0; i < 1000; i++)

{

int A = 0; int B = 0;

for(j = 0; j < 20; j++)

{

if(a[j] == 1 && arr[i][j] == 1)

A = 1;

if(b[j] == 1 && arr[i][j] == 1)

B = 1;

}

//cout<<A<<" "<<B<<endl;

if(A == 1 && B == 1) // only when A = B = 1, break

{ok = 0; break;}

}

if(ok)

{

bestcolumn = nowcoB + nowcoA;

for(int m = 0; m < 20; m++)

{

ansA[m] = a[m];

ansB[m] = b[m];

}

}

return;

}

}

if(column == 20) return;

int okB = 1, okA = 1;

if(nowcoB != 0)

{

for(int i = 0; i < 20; i++)

{

if(b[i] == 1 && judge[column][i] == 0)

{

okA = 0; break;

}

}

}

if(okA)

{

a[column] = 1; // only add this column to A

dfs(column+1, nowcoA+1, nowcoB);

a[column] = 0;

}

if(nowcoA != 0)

{

for(int i = 0; i < 20; i++)

{

if(a[i] == 1 && judge[column][i] == 0)

{

okB = 0; break;

}

}

}

if(okB)

{

b[column] = 1; // only add this column to B

dfs(column+1, nowcoA, nowcoB+1);

b[column] = 0;

}

dfs(column+1, nowcoA, nowcoB); // neither

}

int main()

{

int i, j, k;

for(i = 0; i < 1000; i++)

for(j = 0; j < 20; j++)

cin>>arr[i][j];

for(i = 0; i < 20; i++)

for(j = i+1; j < 20; j++)

{

int ok = 0;

for(k = 0; k < 1000; k++)

{

if(arr[k][i] == 1 && arr[k][j] == 1)

{

ok = 1; break;

}

}

if(!ok)

{

judge[i][j] = 1;

judge[j][i] = 1;

}

}

dfs(0, 0, 0);

for(i = 0; i < 20; i++)

{

if(ansA[i]) cout<<i<<" ";

}

cout<<endl;

for(i = 0; i < 20; i++)

{

if(ansB[i]) cout<<i<<' ';

}

return 0;

}