西安交通大学实验报告

课程名称： 算法设计与分析 **实 验 名 称：** 回溯/分支界限法

学　　院： 电子与信息学部 **实 验 日 期：** 2023 年 12 月 12 日

姓　　名： 林圣翔 **班级：** 计试 2201 **学号：** 2223312202

诚信承诺：我保证本实验报告中的程序和本实验报告是我自己编写，没有抄袭。

一、实验目的

1.深入了解回溯/分支界限法及相关经典问题

2.掌握用回溯/分支界限法解题的基本步骤

3.运用回溯/分支界限法的思想解决最大 0-1 互斥矩阵问题

二、实验环境

系统类型：64 位操作系统, 基于 x64 的处理器

处理器：12th Gen Intel(R) Core(TM) i7-1260P 2.10 GHz

软件：Dev-C++ 5.9.2 TDM-GCC 4.8.1 64-bit Debug

三、问题描述

给定 1 个 1000 行×20 列的 0-1 矩阵，对于该矩阵的任意 1 列，其中值为 1 的元素的数量不超过 10%。设有两个非空集合 A 和 B，每个集合由矩阵的若干列组成。集合 A 和 B 互斥是指对于矩阵的任意一行，同时满足下列 2 个条件：1）若 A 中有一个或多个元素在这一行上的值是 1，则 B 中的元素在这一行全部是 0；2）若 B 中有一个或多个元素在这一行上的值是 1，则 A 中的元素在这一行全部是 0。请你设计一个算法，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。为保证输出结果唯一，每个集合的输出按升序排列，如果存在并列的情况,则依次采用以下策略：

1. A 和 B 元素个数差的绝对值最小

2. A 的个数要大于 B 的个数

3. A 的元素和要小于 B 的元素和

四、问题分析

由题意可得，两个列向量互斥指的是两个列向量的内积为 0。集合 A,B 互斥当且仅当$\forall x \in A$，$\forall y \in B$，x 与 y 的内积为 0。这道题我认为主要有两个思考方向，一个是回溯法，思路简单且实现起来难度不大，但复杂度较高；二是非回溯法，这里可以用枚举的方式，在进行一些优化之后可以得到非常低的时间复杂度，但有一定的思维难度。由

于本实验主要考察的是回溯/分支界限法的运用，所以这里先对基于回溯的算法进行分析。由于这题可能产生多种答案，故题目对并列情况进行了一些处理，我们可以先不考虑，到具体的算法设计中再进行处理。我们可以对第 0-19 个列向量进行一一分析。每个列向量都有三种情况：将入 A 集合、加入 B 集合和两个集合都不加。那么，在最坏的情况下，时间复杂度为（3n）（在本题中，n=20）。但我们可以对其进行剪枝优化，主要的优化方式如下：①若当前列向量和 A、B 中每个已有的每个列向量都互斥，则其有三种情况，加入 A，B 和都不加；②若当前列向量和 A 中每个已有的每个列向量都互斥，B 中存在已有向量与它不互斥，则其有两种情况，加入 B 和都不加；③若当前列向量和 B 中每个已有的每个列向量都互斥，A 中存在已有向量与它不互斥，则其有两种情况，加入 A 和都不加；④若当前列向量和 A、B 中每个已有的列向量都存在不互斥的情况，则只有一种情况，就是两者都不加。我们可以看到，经过这种方式，算法一定程度上得到了优化，但是，在最坏的情况下，就是所有的列向量都满足①的情况，那就相当于没有优化，所以该算法的时间复杂度仍然较高。

这道题还可以用枚举法+二进制优化的角度进行思考。我们可以用一个 20 位的二进制数来表示 A 集合内列向量的情况：若第 k 位为 1，表示 A 集合中有第 k 个列向量；若第 k 为位 0，表示 A 集合中没有第 k 个列向量。之后，针对 A 集合的每一中情况，我们可以根据题意，生成一个符合题意的最大的 B 集合，然后将该次的 A 集合和 B 集合与当前最优解进行比较，根据题意进行取舍，从而得出最优解。这种方法的时间复杂度为 O（2n）（在本题中，n=20）故这种方法和回溯法相比，时间复杂度更优。

五、算法设计

我们对输入的数据进行处理:用 a[i][j]表示第 j 个向量的第 i 个元素，b[i][j]表示第 i 个向量和第 j 个向量之间的关系（b[i][j]=1 说明第 i 个向量和第 j 个向量是互斥关系，b[i][j]=0 说明第 i 个向量和第 j 个向量不是互斥关系），sum[i]表示第 i 个列向量所有元素的和（1 的个数）。

在回溯法中，我们用 anum 表示当前情况下 A 集合中的列向量个数，aa[i]表示当前情况下 A 集合中已加入的第 i 个元素，bnum 表示当前情况下集合中的列向量个数，bb[i]表示当前情况下 B 集合中已加入的第 i 个元素，numa 表示当前最优解中 A 集合中的列向量个数，numb 表示当前最优解中 B 集合中的列向量的个数，ansa[i]表示当前最优解中 A 集合的第 i 个列向量，ansb[i]表示当前最优解中 B 集合的第 i 个列向量。我们用 dfs(step)函数作为我们的深度有限搜索实现的主要函数，step 表示当前正在处理的是第 step 个列向量。我们从 0 开始搜，对每一个列向量进行判断，并确定是否需要进行下一步搜索。我们在搜索中可以发现一种情况，就是 A 集合和 B 集合是可以互换的，也就是说，若存在一组 A 集合个数小于 B 集合的个数，则必存在一组 A 集合的个数大于 B 集合的个数，

且两个集合的个数和和个数差的绝对值都相等。根据题目要求，在这种情况下，A 集合的个数要大于 B 集合的个数，故在我们的程序中，我们可以让 A 集合的个数≥B 集合的个数。这样，我们可以得出一条优化方式，就是当 anum+20-step<bnum 时，程序没有继续向下搜索的意义了，因为这种情况下得出最后 A 集合的个数不可能大于等于 B 集合的个数。当 step==20 时，说明我们已经对所有的列向量都判断过一遍了，得出了当前的 A 集合和 B 集合，我们将其与之前的最优解进行对比，这里我们主要用一个 check()函数实现，待确定当前的为更优解之后，使用 update()函数更新。最后，我们根据 numa 是否为 0 来判断是否有解，若 numa==0，违背题目中 A 为非空的情况，故无解；若 numa！=0，此时 numb 必不等于 0，因为 anum 和 bnum 是我们对 numa 和 numb 赋值的必备条件。

在枚举法中，我们主要运用了二进制优化的思路。为此，我们除了对数据做先前的初始化处理之后，还要有更多的准备工作。首先，我们定义 lowbit(x) 表示((x)&-(x))，方便之后对二进制数的处理计算。用 get_cc()函数得到 c[]数组(c[]数组用于存储针对第 i 个列向量，所能选择的方案数)用 get_sum(x)函数实现计算针对 x 数表示集合的情况求出集合的元素和，用 get_ans(x)将 x 数表示集合的情况转化为集合中所选择列向量的编号的形式输出。之后，我们可以用枚举的方式，由于该过程实现方式比较简单，且与上述回溯法有相似之处，故不再赘述，详见代码。

五、算法实现

这里根据算法设计的两种不同的思路给出两种实现方式。

①回溯法

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const ll N=1010;
ll a[N][22],b[22][22],he,cha=1e9,sum[22],anum,bnum,numa,numb,ansa[22],ansb[22],aa[22],bb[22];
void init()
{
    for(ll i=0;i<=19;i++)
    {
        for(ll j=i+1;j<=19;j++)
        {
            b[i][j]=b[j][i]=1;
            for(ll k=0;k<=999;k++)
            {
                if(a[k][i]&&a[k][j])
                {
                    b[i][j]=b[j][i]=0;
                    break;
                }
            }
        }
    }
}
void get_cc()
{
    for(ll i=0;i<=19;i++)
    {
        for(ll j=0;j<=999;j++) sum[i]=sum[i]+a[j][i];
```

```
    }
}
void update(ll i,ll j)
{
    numa=i,numb=j;
    he=anum+bnum;
    cha=abs(anum-bnum);
    for(ll i=1;i<=numa;i++) ansa[i]=aa[i];
    for(ll i=1;i<=numb;i++) ansb[i]=bb[i];
}
void check(ll anum,ll bnum)
{
    if(anum+bnum>he)  update(anum,bnum);
    else if(anum+bnum==he)
    {
        if(anum-bnum<cha)  update(anum,bnum);
        else if(anum-bnum==cha)
        {
            ll sum1=0,sum2=0,flag1=1,flag2=1;
            for(ll i=1;i<=anum;i++)
            {
                if(ansa[i]>aa[i])
                {
                    flag1=0;
                    break;
                }
                else if(ansa[i]<aa[i])
                {
                    flag1=2;
                    break;
                }
            }
            for(ll i=1;i<=bnum;i++)
            {
                if(ansb[i]>bb[i])
                {
                    flag2=0;
                    break;
                }
                else if(ansb[i]<bb[i])
                {
                    flag2=2;
                    break;
                }
            }
            if(flag1==0||(flag1==1&&flag2==0))
            {
                update(anum,bnum);
                return;
            }
            for(ll i=1;i<=anum;i++) sum1=sum1+sum[aa[i]];
            for(ll i=1;i<=bnum;i++) sum2=sum2+sum[bb[i]];
            if(sum1<sum2) update(anum,bnum);
        }
    }
}
void dfs(ll step)
{
    ll flaga=1,flagb=1;
    if(anum+20-step<bnum) return;
    if(step>=20)
    {
        if(anum&&bnum&&anum>=bnum) check(anum,bnum);
        return;
    }
    for(ll i=1;i<=anum;i++)
    {
        if(!b[step][aa[i]])
        {
            flaga=0;
```

```
            break;
        }
    }
    for(ll i=1;i<=bnum;i++)
    {
        if(!b[step][bb[i]])
        {
            flagb=0;
            break;
        }
    }
    dfs(step+1);
    if(flaga)
    {
        bnum++;
        bb[bnum]=step;
        dfs(step+1);
        bnum--;
    }
    if(flagb)
    {
        anum++;
        aa[anum]=step;
        dfs(step+1);
        anum--;
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    for(ll i=0;i<=999;i++)
    {
        for(ll j=0;j<=19;j++) cin>>a[i][j];
    }
    init();
    get_cc();
    dfs(0);
    if(numa)
    {
        for(ll i=1;i<=numa;i++) cout<<ansa[i]<<" ";
        cout<<"\n";
        for(ll i=1;i<=numb;i++) cout<<ansb[i]<<" ";
    }
    else cout<<"\n\n";
    return 0;
}
```

②枚举+二进制优化

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define lowbit(x) ((x)&-(x))
const ll N=1010;
ll a[N][22],b[22][22],he,cha=1e9,c[22],sum[22],ansa=-1,ansb=-1,anum,bnum;
void init()
{
    for(ll i=0;i<=19;i++)
    {
        for(ll j=i+1;j<=19;j++)
        {
            b[i][j]=b[j][i]=1;
            for(ll k=0;k<=999;k++)
            {
                if(a[k][i]&&a[k][j])
                {
```

```
                    b[i][j]=b[j][i]=0;
                    break;
                }
            }
        }
    }
}
void get_c()
{
    for(ll i=0;i<=19;i++)
    {
        for(ll j=0;j<=19;j++)
        {
            if(b[i][j]) c[i]=c[i]+(1<<j);
        }
    }
}
void get_cc()
{
    for(ll i=0;i<=19;i++)
    {
        for(ll j=0;j<=999;j++) sum[i]=sum[i]+a[j][i];
    }
}
ll get_sum(ll x)
{
    ll cnt=0,ans=0;
    while(x)
    {
        if(x&1) ans=ans+sum[cnt];
        cnt++;
        x>>=1;
    }
    return ans;
}
void get_answer(ll x)
{
    ll k=0,len=20;
    while(len--)
    {
        if(x&1) cout<<k<<" ";
        k++;
        x>>=1;
    }
    cout<<"\n";
}
ll get_num(ll x)
{
    ll ans=0;
    while(x) x=x-lowbit(x),ans++;
    return ans;
}
bool compare(ll x,ll y)
{
    while(x&&y)
    {
        if(lowbit(x)==lowbit(y))
        {
            x=x-lowbit(x);
            y=y-lowbit(y);
            continue;
        }
        else
        {
            if(lowbit(x)>lowbit(y))  return true;
            return false;
        }
    }
    return false;
}
void update(int i, int j)
{
```

```
    ansa=i,ansb=j;
    he=anum+bnum;
    cha=anum-bnum;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    for(ll i=0;i<=999;i++)
    {
        for(ll j=0;j<=19;j++) cin>>a[i][j];
    }
    init();
    get_c();
    get_cc();
    for(ll i=1;i<(1<<20);i++)
    {
        ll x=i,k=0,j=-1;
        while(x)
        {
            if(x&1)  j=(j&c[k]);
            x>>=1;
            k++;
        }
        anum=get_num(i);
        bnum=get_num(j);
        if(anum>=bnum&&bnum)
        {
            if(anum+bnum>he)  update(i,j);
            else if(anum+bnum==he)
            {
                if (anum-bnum<cha)  update(i,j);
                else if(anum-bnum==cha)
                {
                    if(compare(ansa,i))  update(i,j);
                    else if(get_sum(ansa)>=get_sum(ansb)&&get_sum(i)<get_sum(j)) update(i,j);
                }
            }
        }
    }
    if(~ansa)   get_answer(ansa),get_answer(ansb);
    else cout<<"\n\n";
    return 0;
}
```

七、运行结果

首先我们可以制作一个简单的随机数据生成器，随机生成符合题意的输入，输出到1.in 文件中，然后将该输入以文件的形式输入到前面给出的两个不同思路的代码中，将其结果进行对比。

随机输入生成代码如下：

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
ll x,num,a[1010][22],maxnum;
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    freopen("1.in","w",stdout);
    srand(time(0));
    for(ll i=0;i<=19;i++)
    {
        num=0;
        maxnum=rand()%100;
        for(ll j=0;j<=999;j++)
        {
```

```
            x=rand()%2;
            if(x==1) num++;
            if(num>=maxnum) x=0;
            a[j][i]=x;
        }
    }
    for(ll i=0;i<=999;i++)
    {
        for(ll j=0;j<=19;j++) cout<<a[i][j]<<" ";
        cout<<"\n";
    }
    return 0;
}
```

由于是以文件形式读入，故我们需要在先前代码的输入语句前加如下语句：

```
freopen("1.in","r",stdin);
```

为了能体现算法的时间复杂度，我们可以用如下方式记录程序运行时间：

数据读入之前加入如下该语句：

```
clock_t start_time, end_time;
start_time = clock();
```

在程序输出之后加入该语句：

```
end_time=clock();
cout<< "Total time: "<<(double)(end_time-start_time)/CLOCKS_PER_SEC<<"s"<< std::endl;
```

下面是针对部分样例的输出情况（由于输入数据太多，此处省略）

第一组

回溯法程序输出：

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 18 19
14
Total time: 0.055s
```

枚举法+二进制优化程序输出：

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 18 19
14
Total time: 0.199s
```

经检验，输出正确

第二组

回溯法程序输出：

```


Total time: 0.011s
```

枚举法+二进制优化程序输出：

```


Total time: 0.362s
```

经检验，这是一种无解的情况，输出正确

第三组

回溯法程序输出：

```
0 1 2 3 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
4
Total time: 0.036s
```

枚举法+二进制优化程序输出：

```
0 1 2 3 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
4
Total time: 0.233s
```

经检验，输出正确

第四组

回溯法程序输出：

```
1 2 3 6 7 8 12 13 14 15 16
0 4 5 9 10 11 17 18 19
Total time: 2.992s
```

枚举法+二进制优化程序输出：

```
1 2 3 6 7 8 12 13 14 15 16
0 4 5 9 10 11 17 18 19
Total time: 0.285s
```

经检验，输出正确

第五组

回溯法程序输出：

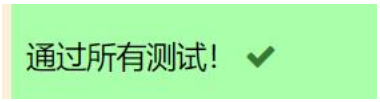
```
0 1 2 5 9 11 16 17 18 19
3 4 6 7 8 10 12 13 14 15
Total time: 12.759s
```

枚举法+二进制优化程序输出：

```
0 1 2 5 9 11 16 17 18 19
3 4 6 7 8 10 12 13 14 15
Total time: 0.17s
```

经检验，输出正确

……



通过大量输入和输出的对比，我们可以看出：当输入中 1 的个数比较大时，两者的运行时间差不多，甚至回溯法占优；当输入中 1 的个数偏少时，两者的运行时间相差变大，枚举法+二进制优化的程序明显占优。回溯法的程序受输入中 1 个数的多少影响大，符合之前算法理论中剪枝以及最坏时间复杂度，而枚举法+二进制优化的程序运行时间受输入影响不大，符合其算法设计理论。