2015 级算法第五次上机解题报告（助教版）

目 录

2015 级算法第五次上机解题报告（助教版）

马国瑞

一、引言

本次上机题目平均分 3.0863 分，为历次上机的最高值。本次上机大多数题目网上都能找到原题题解，但题目本身是有一定难度的，部分题目需要一定的思考量才能解出。这里希望大家通过解题能学到网上相关题解的解题思想。

二、解题报告

A 大家一起来排队

思路分析

二分法

对于一个已经排好序的数组，使用线性查找会浪费很多不必要的时间，使用二分查找是一个不错的选择。对于本题，只要求数组中那个数是否在数组中即可。C++ STL 的 binary_search 函数也可过。

二分法有非递归算法和递归算法两种实现方式，分别参见参考代码一和参考代码二。

算法分析

二分法也是一种分治算法。其递归式如下：

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

使用主方法求解得到：

$$T(n) = O(\lg n)$$

故时间复杂度为 O(lgn)

参考代码一

```
//Source: Magry
#include <stdio.h>
int dat[4000010];
int main()
{
    int n,m;
```

```
    int cnt=0;
    while(~scanf("%d%d",&n,&m)){
        int i,j;
        for(i=0;i<1;i++){
            for(j=0;j<m;j++)
                scanf("%d",&dat[j]);
        }
        cnt++;
        printf("Case %d:\n",cnt);
        for(i=0;i<n;i++){
            int b;
            scanf("%d",&b);
            //二分非递归算法
            int l=0,r=m-1;
            int mid=(l+r)/2;
            while(dat[mid]!=b){
                if(r-l==1){
                    if(dat[r]==b) mid=r;
                    else if(dat[l]==b) mid=l;
                    else mid=-1;
                    break;
                }
                else if(r==l){
                    if(dat[r]==b) mid=r;
                    else mid=-1;
                    break;
                }
                else{
                    if(dat[mid]<b) l=mid;
                    else if(dat[mid]>b) r=mid;
                    mid=(l+r)/2;
                }
            }
            if(mid==-1)
                printf("No\n");
            else
                printf("Yes\n");
        }
    }
}
```

参考代码二

```
//Source: Magry
#include<cstdio>
int a[4000010];
//二分法递归算法
int bsearch(int low, int high, int target)
{
    if (low > high) return -1;

    int mid = (low + high)/2;
    if (a[mid]> target)
        return   bsearch(low, mid -1, target);
    if (a[mid]< target)
        return   bsearch(mid+1, high, target);

    return mid;
}
int main()
{
    int n,m,cnt=0;
    while(~scanf("%d%d",&n,&m)){
        printf("Case %d: \n",++cnt);
        for(int i=0;i<m;i++)
            scanf("%d",&a[i]);
        for(int i=0;i<n;i++){
            int x;
            scanf("%d",&x);
            int ans=bsearch(0,m-1,x);
            if(ans==-1||a[ans]!=x)
                printf("No\n");
            else printf("Yes\n");
        }
    }
}
```

B 简单的贪心

思路分析

贪心经典问题之过河问题，对于样例如何得出需要一定的思考量。

我们可以这么认为：站在河对岸的人也可以把船划回来。因此对于样例可通过如下方法实现：(岸①->岸②)

```
T1=1,T2=2,T3=5,T4=10
Step 1: ①->② 运送 1、2——T=2
Step 2: ①<-② 运送 1——T=1
Step 3: ①->② 运送 3、4——T=10
Step 4: ①<-② 运送 2——T=2
Step 5: ①->② 运送 1、2——T=2，结束，总时间 17
```

拓展到 n 个人的情况如下：

考虑 n<=3 时，情况均能够很明显得出；那么当人数 n>=4 时有如下两种策略：

策略一 最快的人两次运最慢的，然后把船划回来

策略二 最快的人和次快的人一起过去，最快的回来，两个最慢的过去，已经在对岸的次快的人回来

上述策略 1、2 优先选时间较短的，直到剩下小于 4 个人为止。

算法分析

本算法排序时间 O(lgn)，贪心选择时间为 O(n).

参考代码

```
//Source: Magry
#include <iostream>
#include <algorithm>
using namespace std;
int main()
{
    int n;
    while(cin>>n){
        long long ans=0;
        long long a[1007];
        long long x,minx=0;
        for(int i=0;i<n;i++){
            cin>>a[i];
```

```
        }
        sort(a,a+n);
        int i;
        //贪心选择
        for(i=n-1;i>2;i-=2){
            long long buf1=a[0]*2+a[i]+a[i-1];
            long long buf2=a[0]+2*a[1]+a[i];
            if(buf1>buf2) ans+=buf2;
            else ans+=buf1;
        }
        if(i==2)
            ans+=(a[0]+a[1]+a[2]);
        else if(i==1)
            ans+=a[1];
        else ans+=a[0];
        cout<<ans<<endl;
    }
}
```

C 无法停止的计数

思路分析

根据题意我们发现这是《算法导论》及算法课程提到的二进制计数器递增问题。

使用势能法做的话，我们需要假设势为当前状态下二进制位为 1 的个数。那么对于每一步+1 的操作，有：

摊还代价= 操作数 + 势差

其中，势差 = 操作后的势 - 操作前的势

操作数的计算可以通过位运算或模拟二进制计数器翻转操作的次数得出。

需要注意的是，本题为 31 位的二进制计数器，清零时的摊还代价与其他递增操作的摊还代价并不相同。

参考代码

```
//Source: Magry
#include<cstdio>
int a[32];
int main()
{
    int n;
    long long mo=1<<31;
    long long x;
    scanf("%d%lld",&n,&x);
    long long px=x;
    for(int i=0;px>0;i++){
        a[i]=px%2;
        px>>=1;
    }
    long long ansx=x;
    for(int cnt=0;cnt<n;cnt++){
        printf("Case %d:\n",cnt+1);
        int k;
        scanf("%d",&k);
        ansx=(ansx+k)%mo;
        while(k--){
            int ansk=0;
            int ans=0;
            int j=0;
```

```
            //二进制位翻转为 0
            while(j<31&&a[j]==1){
                a[j]=0;
                j++;
                ans+=0;
                ansk++;
            }
            //有且最多仅有一次操作二进制位翻转为 1
            if(j<31&&a[j]==0){
                ans+=2;    //当出现一次由 0 变为 1 的操作时摊还代价+2，思考一下为什么
                ansk++;
                a[j]=1;
            }
            printf("%d %d\n",ansk,ans);
        }
        printf("Sum: %lld\n",ansx);
    }
}
```

D 赢了这局我就睡觉

思路分析

Source: 15211095 吴星哲

对于本题的求解，第一步，求出某一天会删游戏的概率；第二步，根据 p 求出游戏保留期望天数 x。得到方程如下：

```
Ex=1*p+2*(1-p)*p+3*(1-p)2*p+.....+n*p*(1-p)(n-1),n->正无穷
Ex-(1-p)Ex=p+(1-p)*p+.......+p*(1-p)(n-1)
所以 Ex=1/p
```

难题就在于第一步，可以考虑用一个和 01 背包很类似的动态规划。对于第 i 次玩，有二种情况，赢或不赢。设 res[i][j]表示玩到第 i 局时，已经赢了 j 局且胜率一直没有超过 p 的概率。Res[i][j]=0 表示这种情况不可能。res 推理过程如下：

```
for( int i=1;i<=n;i++ )
          for( int j=0;j*b<=i*a;j++ )
          {
              if(j==0)
                  res[i][j]=res[i-1][j]*(1-p);
              else
                  res[i][j]=res[i-1][j]*(1-p)+res[i-1][j-1]*p;//输赢二种情况
          }
```

所以时间复杂度为 O(n*n)。空间复杂度为 O(n*n)级别，其实可以优化为 O(n)级别，res 开一个一维数组，优化方法与 01 背包相同。

参考代码

```
//Source: 15211095 吴星哲
#include<cstdio>
#include<cstring>
#define N 105
using namespace std;

double res[N][N];//每种情况的概率

int main()
{
    int T;
    int a,b,n;
    double p;
```

```
    scanf( "%d",&T);
    while(T--)
    {
        scanf( "%d/%d %d",&a,&b,&n );
        p=(double)a/b;
        for( int i=0;i<N;i++ )
            for( int j=0;j<N;j++ )
                res[i][j]=0.0;

        res[0][0]=1.0;
        for( int i=1;i<=n;i++ )
            for( int j=0;j*b<=i*a;j++ )
            {
                if(j==0)
                    res[i][j]=res[i-1][j]*(1-p);
                else
                    res[i][j]=res[i-1][j]*(1-p)+res[i-1][j-1]*p;//输赢二种情况
            }

        double ans=0.0;
        for(int i=0;i<=n;i++ )
            ans+=res[n][i];//某一天打 n 次还达不到胜率的概率
        printf("%d\n",(int)(1/ans));//第二步

    }

    return 0;
}
```

E Magry 遇上了初雪第三弹

思路分析

Source: 15211077 王媛媛

这道题主要考察**最大流问题**。

首先是网络流的构建，整个网络是一个带权的有向图，采用邻接表的方式存储；

接下来是从起点对整个图进行深度优先搜索，找到残余网络的一条增广路径，求得新的最大流。

需要注意的是，在修改每条边上的权值的时候，不要忘了增加一条反向边，以便在流过大的时候能够将流推回去，从而得到正确的最大流结果；

每次深搜的过程中都会形成新的残余网络，当形成的参与网络中不再能找到新的增广路径的时候，即得到网络的最大流。

PS：需要注意的是，本题需要多次查询不同起点和终点的最大流。然而，每次查询过程中原网络已经遭到破坏。因此，在每一次新的查询过程中，都应该重新构建新的图并重新进行上述操作。

参考代码

```
//Source: 15211077 王媛媛
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<algorithm>
#include<vector>
#define INF 2147483647
using namespace std;
const int MaxSize=10010;//10000000+5;
int cap[MaxSize],x[MaxSize],y[MaxSize];//起始点及容量的记录数组
struct edge{
   int to;
   int cap;
   int rev;
};//边结构体
vector<edge> G[MaxSize];//邻接表法表示图
bool used[MaxSize];//dfs 过程中是否已遍历过的记录数组

void add_edge(int from,int to,int cap)//加边及反向边
{
```

```
    G[from].push_back((edge){to,cap,G[to].size()});
    G[to].push_back((edge){from,0,G[from].size()-1});
}
int dfs(int v,int t,int f)//深搜
{
    if(v==t) return f;
    used[v]=true;
    for(int i=0;i<G[v].size();i++)
    {
        edge &e=G[v][i];
       if(!used[e.to]&&e.cap>0)
       {
           int d=dfs(e.to,t,min(f,e.cap));//寻找增广路径求得新的流
           if(d>0)
           {
                e.cap-=d;//修改权值
                G[e.to][e.rev].cap+=d;//增加反向边
                return d;
           }
       }
    }
    return 0;
}
int max_flow(int s,int t)//从 s 到 t 的最大流
{
    int flow=0;
    for(;;)
    {
       memset(used,0,sizeof(used));
       int f=dfs(s,t,INF);
       if(f==0) return flow;
       flow+=f;
    }
}
int main()
{
    int V,N;//代表顶点数
    while(scanf("%d %d",&V,&N)!=EOF)
    {
        memset(x,0,sizeof(x));
       memset(y,0,sizeof(y));
```

```
        memset(cap,0,sizeof(cap));
        for(int i=0;i<N;i++)
        {
            scanf("%d %d",&x[i],&y[i]);
            scanf("%d",&cap[i]);
         add_edge(x[i],y[i],cap[i]);
        }
         int ans=max_flow(1,V);
         if(ans==0)
             printf("404 Not Found\n");
         else
            printf("%d\n",ans);
         memset(G,0,sizeof(G));
    }
}
```

F 二营长，你他娘的意大利炮呢

思路分析

Source: 15151165 马宇航

本题的实质是求二分图的最大匹配。采用匈牙利算法（最大流 EK 算法会 TLE）。二分图简而言之，就是顶点集 V 可分割为两个互不相交的子集，并且图中每条边依附的两个顶点都分属于这两个互不相交的子集，两个子集内的顶点不相邻。本题中摆在南面和西面的意大利炮可以看成这样两组点集，若在西面和南面的意大利炮的射程交点上有一个敌人，可以看成这两个点之间是联通的。则该战场则可以转化为一个二分图，题目所求即二分图的最小覆盖，而最小覆盖等于最大匹配。

匈牙利算法的核心思想也是寻找增广路径。

增广路径的定义(也称增广轨或交错轨)：

若 P 是图 G 中一条连通两个未匹配顶点的路径，并且属 M 的边和不属 M 的边(即已匹配和待匹配的边)在 P 上交替出现，则称 P 为相对于 M 的一条增广路径。

增广路径必须满足的性质

1.有奇数条边。

2.起点在二分图的左半边，终点在右半边。

3.路径上的点一定是一个在左半边，一个在右半边，交替出现。（由二分图的性质决定）

4.整条路径上没有重复的点。

5.起点和终点都是目前还没有配对的点，而其它所有点都是已经配好对的。

6.路径上的所有第奇数条边都不在原匹配中，所有第偶数条边都出现在原匹配中。

7.最后，也是最重要的一条，把增广路径上的所有第奇数条边加入到原匹配中去，并把增广路径中的所有第偶数条边从原匹配中删除（这个操作称为增广路径的取反），则新的匹配数就比原匹配数增加了 1 个（奇数=偶数+1）。

采用 dfs 方式寻找增广路径

然后依据增广路径更新最大匹配。详见代码注释。

参考代码一

```
//Source: 15151165 马宇航
#include<cstdio>
#include<cstring>
```

```
#include<cstdlib>
#define MAX 1000
#define MAXN 1001
bool map[MAXN][MAXN], vis[MAXN];//用邻接矩阵记录图
int match[MAXN];//记录匹配结果
int n, k;

bool dfs( int u)//寻找增广路径
{
    int v;
    for( v = 1; v <= MAX; v ++)
        if( map[u][v] && !vis[v])//u v 间联通且 v 没被检查过
        {
            vis[v] = true;
            if( match[v] == -1 || dfs( match[v])) v 没被匹配或者 v 的匹配存在其他合理
匹配
            {
                match[v] = u; //将 u v 匹配
                return true;
            }
        }
        return false;//u 无法匹配，返回 false
}

int MaxMatch()
{
    int u, ret = 0;
    memset( match, -1, sizeof match);//初始化
    for( u = 1; u <= MAX; u ++)
    {
        memset( vis, false, sizeof vis); //每次匹配前，初始化已看过的点
        if( dfs(u)) ret ++;
    }
    return ret;
}

int main()
{
        int x, y;
        while(~scanf("%d",&k)){
        memset( map, false, sizeof map);//初始化
```

```
        for( int i = 1; i <= k; i ++)
        {
            scanf( "%d%d", &x, &y);//输入
            map[x][y] = true;
        }
        int ans = MaxMatch();
        printf( "%d\n", ans);//输出结果
    }

    return 0;
}
```