哈尔滨理工大学

ACM-ICPC 集训队常用代码库

(2020 年 5 月 1 日发布版)

组织编写：唐远新

内容审核：刘明辉

内容编撰：计 16-1 刘明辉，计 18-9 胡小文，计 18-8 鞠永全

计 18-5 王东琛，计 18-8 杨睿，计 18-7 牛仔超

网络 18-2 吴国庆，网络 18-3 董文睿，计 18-5 王佳妮

网络 18-3 高云泽，网络 18-3 冯紫君，计 18-8 蒙晟维

前　言

哈尔滨理工大学 ACM-ICPC 集训队自 2011 年正式组建以来，培养和造就了一大批热爱算法和编程的同学。集训队也因为这些同学的热忱和坚守而保持旺盛生命力和战斗力。高年级同学不遗余力指导学弟学妹入门和提高，积极建言献策，将学习训练内容在实践中不断迭代优化，至今采用的是第四版集中训练内容。

尽管网上流传有很多学校和大牛的模板库，但是我校集训队员们一直期待有一份自己学校队伍的代码库，它应该更接地气，更适合我校同学学习与使用。几年来，不同年级同学做了一些尝试，包括在网上建立 WIKI 风格模板代码库、Word 版代码库，这些尝试因为各种原因没有有效坚持完成，未形成一个内容较完整的可发布版本。

2016 级刘明辉同学在阿里实习归来后，主动再次承担起了这项任务，组织 2018 级同学整理新生培训内容，并亲自撰写了部分内容，形成了一个内容较丰富、知识点较全面的代码库，并汇集了部分知识点不同算法的时空复杂度和练习题。

这是自 2012 年集训队员合力编写《ACM-ICPC 培训资料汇编》后的又一次集体创作，给予完成的时限短，内容也主要限于培训讲解的知识。作为代码库，还有很多需要补充、完善的地方。期待后继队员能够持续丰富和完善这个代码库，也期待现在这个版本能为后来者起到很好的入门指导作用。

感谢刘明辉和所有参与内容整理、撰写的同学！也感谢十年来集训队薪火相传、不断奋勇前进的每一位同学！相信他们在未来的学习、工作、生活中能继续秉持成就自己、提升他人的协作共赢理念，不断取得新的收获和进步。

本版本参与人员和完成工作如下：

内容审核：刘明辉

内容编撰：计 16-1 刘明辉，计 18-9 胡小文，计 18-8 鞠永全，计 18-5 王东琛，计 18-8 杨睿，计 18-7 牛仔超，网络 18-2 吴国庆，网络 18-3 董文睿，计 18-5 王佳妮，网络 18-3 高云泽，网络 18-3 冯紫君，计 18-8 蒙晟维

唐远新

2020 年 5 月 1 日

目 录

第 1 章 搜索

1.1 深度优先搜索

```
// hrbust 1743   题意: n*m 字符矩阵，q 次查询某个字符串是否是矩阵的一条路径
#include<stdio.h>
#include<string.h>
#define N 55
char a[N][N];
char str[N];
int len;                    // 存 str 字符串长度
bool vis[N][N];             //每个点只能走一次
int n,m;
class Dfs
{
public:
    int dx[10]= {0,0,-1,1}; // dx,dy 组成上下左右
    int dy[10]= {1,-1,0,0};
    bool dfs(int x,int y,int z)
    {
        if(z==len)
            return true;
        for(int i=0; i<4; i++)
        {
            int xx=x+dx[i];
            int yy=y+dy[i];
            if(xx>0&&yy>0&&xx<=n&&yy<=m&&a[xx][yy]==str[z]&&vis[xx][yy])
            {
                vis[xx][yy]=false;
                if(dfs(xx,yy,z+1))
                    return true;
                vis[xx][yy]=true;
            }
        }
        return false;
    }
} dfs;
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
```

```
            int q;
            scanf("%d %d %d",&n,&m,&q);
            for(int i=1; i<=n; i++)
                scanf("%s",a[i]+1);
            for(int qq=0; qq<q; qq++)
            {
                scanf("%s",str);
                len=strlen(str);
                memset(vis,true,sizeof(vis));
                int zt=0;
                for(int i=1; i<=n; i++)
                {
                    for(int j=1; j<=m; j++)
                    {
                        vis[i][j]=false;
                        if(a[i][j]==str[0]&&dfs.dfs(i,j,1))
                        {
                            zt=1;
                            break;
                        }
                        vis[i][j]=true;
                    }
                    if(zt)
                        break;
                }
                if(zt)
                    printf("Yes\n");
                else
                    printf("No\n");
            }
            printf("\n");
        }
}
```

整理人：计 16-1 刘明辉

1.2 广度优先搜索

```
// bfs 查找 从 s 点 能否到达 t 点
class Bfs
{
public:
    bool vis[N];
    queue<int>q;
    void bfs(int s,int t)
```

```
    {
        while(!q.emtpy())
            q.pop();
        memset(vis,true,sizeof(vis));
        vis[s]=false;
        q.push(s);
        while(!q.empty())
        {
            int z=q.front();
            q.pop();
            if(z==t)
                return true;
            // 搜索 z 点能够到达的所有点，入队，并且标记
        }
        return false;
    }
};
```

整理人：计 16-1 刘明辉

1.3 双向搜索

// 对于单向搜索有 2^n 可能的情况，若从起始点和终点同时搜，可以变成 2 个 2^(n/2)次幂中可能，然后进行合并。例如：每次操作可以选择*a 或者+b，问数字从 n 开始操作 40 次变成 m 的可能性有多少种。可以选择正向反向同时求出 2^20 个答案，然后进行合并。

整理人：计 16-1 刘明辉

1.4 记忆化搜索

// 一般来说搜索的结果数量有限，因此在搜索时，会把搜到的结果也记录到一个数组，下次可以直接使用。

整理人：计 16-1 刘明辉

1.5 极大极小搜索-alpha-beta 剪枝

// 暴力搜索，但对于某些特定情况可以通过 alphat-beta 值直接优化掉

```
#include<stdio.h>
#include<math.h>
#include<algorithm>
using namespace std;
char a[10][10];
```

```
class MinimaxSearch
{
public:
	int xx,yy;
	int Evaluate()//判断  1 是 x 胜利  -1 是 o 胜利  0 是没有胜利
	{
		int i,j,z,zz,zt,zzt;
		for(i=0; i<4; i++)
		{
			z=0,zz=0,zt=0,zzt=0;
			for(j=0; j<4; j++)
			{
				if(a[i][j]=='x')
					z++;
				else if(a[i][j]=='o')
					zz++;
				if(a[j][i]=='x')
					zt++;
				else if(a[j][i]=='o')
					zzt++;
			}
			if(z==4||zt==4)
				return 1;
			if(zz==4||zzt==4)
				return -1;
		}
		zt=0,zzt=0,z=0,zz=0;
		for(i=0; i<4; i++)
		{
			if(a[i][i]=='x')
				z++;
			if(a[i][i]=='o')
				zz++;
			if(a[i][3-i]=='x')
				zt++;
			if(a[i][3-i]=='o')
				zzt++;
		}
		if(z==4||zt==4)
			return 1;
		if(zz==4||zzt==4)
			return -1;
		return 0;
	}
	int Max(int depth, int upalpha, int upbeta)
	{
```

```
    int alpha=upalpha, beta=upbeta;
    int val;
    int flag=Evaluate();
    if(flag||depth==0)//已经有一方胜利，或者全下完
        return flag;
    int i,j;
    for(i=0; i<4; i++)
    {
        for(j=0; j<4; j++)
        {
            if(a[i][j]=='.')
            {
                a[i][j]='x';
                val = Min(depth - 1, alpha, beta);
                a[i][j]='.';
                alpha=max(alpha, val);
                if(alpha>=beta)
                {
                    xx=i,yy=j;
                    return alpha;
                }
            }
        }
    }
    return alpha;
}
int Min(int depth, int upalpha, int upbeta)
{
    int alpha=upalpha, beta=upbeta;
    int val;
    int flag=Evaluate();
    if(flag||depth==0)//已经有一方胜利，或者全下完
        return flag;
    int i,j;
    for(i=0; i<4; i++)
    {
        for(j=0; j<4; j++)
        {
            if(a[i][j]=='.')
            {
                a[i][j]='o';
                val = Max(depth - 1, alpha, beta);
                a[i][j]='.';
                beta=min(beta, val);
                if(beta<=alpha)
                    return beta;
```

```
                    }
                }
            }
            return beta;
        }
} miniMax;
int main()
{
    char c;
    while(scanf("%c",&c)!=EOF)
    {
        if(c=='$')
            break;
        int i,j;
        int sum=0;
        for(i=0; i<4; i++)
        {
            scanf("%s",a[i]);
            for(j=0; j<4; j++)
            {
                if(a[i][j]=='.')
                    sum++;
            }
        }
        if(sum>11)//刚下四个子时不可能有决胜点，没有这个一直 TLE
        {
            printf("#####\n");
            getchar();
            continue;
        }
        int z=miniMax.Max(sum,-1,1);
        if(z==1)
            printf("(%d,%d)\n",miniMax.xx,miniMax.yy);
        else
            printf("#####\n");
        getchar();
    }
}
```

整理人：计 16-1 刘明辉

1.6 启发式搜索

//poj 1077 A* 解决八数码问题（3*3 的棋盘，有 8 个标号是 1-8 的棋子，一个空格。每

次可以把一个与空格相邻的棋子移动到空格上，给出一个目标棋盘，问最少移动次数）

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <string>
#include <queue>
using namespace std;
const int maxn = 15, maxs = 362885;

class AStar
{
public:
	const int nxt[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
	const char dir[4] = {'u', 'd', 'l', 'r'};
	int bit[maxn], fact[maxn];
	AStar()
	{
		fact[0] = 1;
		for (int i = 1; i < 9; ++i)
			fact[i] = fact[i - 1] * i;
	}

	inline int ask(int x)
	{
		int ret = 0;
		while (x)
			ret += bit[x], x -= x & -x;
		return ret;
	}

	inline void add(int x, int d)
	{
		while (x <= 9)
			bit[x] += d, x += x & -x;
	}

	struct Node
	{
		int i, x, y;
		string s, ans;

		bool operator < (const Node& rhs) const
		{
			return x + y > rhs.x + rhs.y;
```

```
	}
} st;

int vis[maxs];
priority_queue<Node> q;

inline int cantor(string s)
{
	memset(bit, 0, sizeof(bit));
	for (int i = 0; i < 9; ++i)
		if (s[i] == 'x')
			s[i] = '0';
	int ans = 1;
	for (int i = 8; i >= 0; --i)
	{
		ans += ask(s[i] - '0') * fact[8 - i];
		add(s[i] - '0' + 1, 1);
	}
	return ans;
}

inline int diff(string s)
{
	int ans = 0;
	for (int i = 0; i < 9; ++i)
	{
		int x = i / 3, y = i % 3;
		if (s[i] == 'x')
			ans += abs(x - 2) + abs(y - 2);
		else
		{
			int j = s[i] - '1';
			ans += abs(x - j / 3) + abs(y - j % 3);
		}
	}
	return ans;
}

inline string bfs(string s)
{
	for (int i = 0; i < 9; ++i)
		if (s[i] == 'x')
			st.i = i;
	st.x = 0, st.y = diff(s);
	st.s = s, st.ans = "";
	vis[cantor(st.s)] = 1;
```

```
            q.push(st);
            while (!q.empty())
            {
                Node u = q.top(), v;
                q.pop();
                if (u.s == "12345678x")
                    return u.ans;
                int x = u.i / 3, y = u.i % 3;
                for (int i = 0; i < 4; ++i)
                {
                    int nx = x + nxt[i][0], ny = y + nxt[i][1];
                    if (nx < 0 || nx > 2 || ny < 0 || ny > 2)
                        continue;
                    v.i = 3 * nx + ny;
                    v.s = u.s;
                    swap(v.s[u.i], v.s[v.i]);
                    if (vis[cantor(v.s)])
                        continue;
                    vis[cantor(v.s)] = 1;
                    v.x = u.x + 1, v.y = diff(v.s);
                    v.ans = u.ans + dir[i];
                    q.push(v);
                }
            }
            return "unsolvable";
        }
} a_star;
int main()
{
        char in[2];
        string s = "";
        for (int i = 1; i <= 9; ++i)
        {
            scanf("%s", in);
            s += in[0];
        }
        int cnt = 0;
        for (int i = 8; i >= 0; --i)
            if (s[i] != 'x')
                cnt += a_star.ask(s[i] - '0'), a_star.add(s[i] - '0', 1);
        if (cnt & 1)
            printf("unsolvable");
        else
            cout << a_star.bfs(s);
        return 0;
}
```

```
// IDA* 解决八数码问题
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<algorithm>
#define INF 40000
using namespace std;
int f[10];
class IDAStar
{
public:
    int dx[10]= {0,0,1,-1};
    int dy[10]= {1,-1,0,0};
    char dxy[10]="rldu";
    char ans[1005];
    int hh()//估值函数=每一个都直接走到自己位置的步数
    {
        int i,j,sum=0;
        for(i=0; i<9; i++)
        {
            if(f[i])
                sum+=abs(i/3-(f[i]-1)/3)+abs(i%3-(f[i]-1)%3);
        }
        return sum;
    }
    int minn;
    bool dfs(int x,int g,int depth)//x 代表'x'的位置  g 代表现在走了多少  depth 代表本次最多走多少
    {
        int h=hh();
        minn=min(minn,h);//找到这次离终点最近的距离是多少，但实际距离一定是>=估计距离
        if(h==0)
        {
            ans[g]='\0';
            printf("%s\n",ans);
            return true;
        }
        if(g+h>depth)
            return false;
        int xx,yy;
        int i,j;
        for(i=0; i<4; i++)
        {
            xx=x/3+dx[i];
```

```
            yy=x%3+dy[i];
            if(xx>=0&&yy>=0&&xx<3&&yy<3)
            {
                if(g)
                {
                    if(i==0&&ans[g-1]=='l')
                        continue;//这次往上，上次往下肯定是不可以的
                    if(i==1&&ans[g-1]=='r')
                        continue;
                    if(i==2&&ans[g-1]=='u')
                        continue;
                    if(i==3&&ans[g-1]=='d')
                        continue;
                }
                ans[g]=dxy[i];
                f[x]=f[xx*3+yy];
                f[xx*3+yy]=0;
                if(dfs(xx*3+yy,g+1,depth))
                    return true;
                f[xx*3+yy]=f[x];
                f[x]=0;
            }
        }
        return false;
}
void IDA_star(int x)
{
        int depth=hh();
        while(1)
        {
            minn=INF;
            if(dfs(x,0,depth))
            {
                break;
            }
            depth+=minn;
        }
}
bool pan()
{
        int i,j,k=0;
        for(i=0; i<9; i++)
        {
            if(f[i]==0)
                continue;
            for(j=0; j<i; j++)
```

```
                {
                    if(f[j]>f[i])
                        k++;
                }
            }
            if(k%2)
                return true;
            return false;
        }
} ida_star;
char a[105];
int main()
{
        while(gets(a))
        {
            int i,j,a1=strlen(a),k=0,x,y;
            for(i=0; i<a1; i++)
            {
                if(a[i]!=' ')
                {
                    if(a[i]=='x')
                    {
                        x=k/3;
                        y=k%3;
                        f[k]=0;
                        k++;
                    }
                    else
                        f[k++]=a[i]-'0';
                }
            }
            if(ida_star.pan())
                printf("unsolvable\n");
            else
                ida_star.IDA_star(x*3+y);
        }
}
// 1 2 3 4 5 6 7 8 x
```

整理人：计 16-1 刘明辉

第 2 章 动态规划

2.1 状压 Dp

///hdu5418 题意：n 个城市，m 条无向边，问从 1 号城市经过所有城市在回到 1 号城市的最少花费

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=2e6+6;
const int N=20;///城市点数
const int inf=0x3f3f3f3f;
const int mod=1e9+7;

int n,m;///城市数量，边数量
int a[N][N];///邻接矩阵存图
class State
{
public :
    int dp[N][(1<<N)+10];///dp[i][S]  为当前在 i 走过的集合为 S 的花费,

    int solve(){
        for(int k=0;k<n;k++){
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
                }
            }
        }///floyd  求两点间最短距离

        memset(dp,inf,sizeof dp);

        dp[0][1]=0;
        int limit=1<<n;
        for(int S=0;S<limit;S++){
            for(int i=0;i<n;i++) if((1<<i)&S){///起点
                for(int j=0;j<n;j++)if(!((1<<j)&S)){///终点
                    dp[j][S|(1<<j)]=min(dp[j][S|(1<<j)],dp[i][S]+a[i][j]);
                }
            }
        }

        int ans=inf;
```

```
            for(int i=0;i<n;i++){
                ans=min(ans,dp[i][limit-1]+a[0][i]);
            }
            return ans;
        }
}zdp;
int main(){
        int t;
        scanf("%d",&t);
        while(t--){
            scanf("%d%d",&n,&m);
            for(int i=0;i<n;i++) {
                for(int j=0;j<n;j++) a[i][j]=(i==j?0:inf);
            }
            for(int i=1;i<=m;i++){
                int u,v,w;scanf("%d%d%d",&u,&v,&w);
                u--;v--;///边从 0 开始计数，方便转移
                a[u][v]=min(a[u][v],w);///避免有重边
                a[v][u]=a[u][v];
            }
            printf("%d\n",zdp.solve());
        }
}
```

整理人：计 18-8 鞠永全

2.2 树形 Dp

```
///求树的重心
class TreeRoot
{
public:
int sz[maxn],f[maxn],vis[maxn],root;
//sz[i] i 节点为根的子树大小
// f[i] i  节点为根的最大子树大小
//root 重心  n 树的节点数量
    void solve(int u){
        sz[u]=1;f[u]=0;
        vis[u]=1;
        for(int i=head[u];i!=-1;i=edge[i].next){
            int v=edge[i].to;
            if(vis[v]) continue;
            solve(v);
            sz[u]+=sz[v];
            f[u]=max(f[u],sz[v]);
```

```
            }
            f[u]=max(f[u],n-sz[u]);
            if(f[root]>f[u]||root==0) root=u;
        }
        int get_root(int x){//接口
            root=0;
            memset(vis,0,sizeof vis);
            solve(x);
            return root;
        }
};

//求树的最大独立集
vector<int>G[maxn];
class TreeNum
{
public:
        int sz[maxn],dp[maxn][2],vis[maxn];
        ///子树大小，dp[i][0/1]节点 i 选/不选的子树最优解，重复点不访问

        void dfs(int x){
            vis[x]=1;dp[x][0]=0;
            dp[x][1]=1;
            for(int i=0; i<G[x].size(); i++){
                int y=G[x][i];
                if(vis[y])continue;
                dfs(y);
                dp[x][0]+=max(dp[y][0],dp[y][1]);
                dp[x][1]+=dp[y][0];
            }
        }
        int get_num(int x){//接口
            memset(vis,0,sizeof vis);
            solve(x);
            return max(dp[x][0],dp[x][1]);
        }
};
```

整理人：计 18-8 鞠永全

2.3 区间 Dp

区间 DP

//n 堆石子排成一列，每堆石子有一个重量 w[i]

//每次合并可以合并相邻的两堆石子，一次合并的代价为两堆石子的重量和 w[i]+w[i+1]。

```
//问安排怎样的合并顺序，能够使得总合并代价达到最大
long long    w[maxn],n;
class IntervalDp{
public:
      long long dp[maxn][maxn],sum[maxn];
//dp[i][j]表示把第 i 堆到第 j 堆的石子合并在一起的最优值
//sum[i]为前 i 堆石子的和
      long long solve(LL *w,LL n){//接口
            for(int i=1;i<=n;i++){
                  sum[i]=sum[i-1]+w[i];
            }

            for(int len=2; len<=n; len++){
                  for(int i=1; i<=n-len+1; i++){
                        int j=i+len-1;
                        for(int k=i; k<=j-1; k++){
                              dp[i][j]=max(dp[i][j],dp[i][k]+dp[k+1][j]+sum[j]-sum[i-1]);
                        }
                  }
            }
            return dp[1][n];
      }
};
```

整理人：计 18-8 鞠永全

2.4 数位 Dp

(一定要把 dp 方程所表示的状态设好再转移)

```
///hdu2089  题意：区间内不包含 62 和 4 的个数
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 25;
int n, m;
class DigitDp {
public :
      int dp[N][2], a[N];///dp[i][j]代表枚举到第 i 位前一位是 j 的方案数
      int dfs(int now, int pre, int limit) {
            if(!now)
                  return  1;///枚举到最后一位后就要检查状态这里因为枚举的过程中就把状态检查完了直接返回 1
            if(!limit && ~dp[now][pre]) ///如果当前位的值没达到最高且次状态已经访问过
                  return dp[now][pre];
            int up = limit ? a[now] : 9, ans = 0; ///根据 limit 是否是 1 决定下一位最多
```

枚举到多少

```
        for(int i = 0; i <= up; i++) {
            if((pre && i == 2) || (i == 4)) ///如果前一位是 6 且当前位是 2 或者当前位是 4 直接跳过
                continue;
            ans += dfs(now - 1, i == 6, i == up && limit); ///统计当前状态的答案也就是 dp 的过程
        }
        if(!limit)///当前位值未达到最高才能记忆化
            dp[now][pre] = ans;
        return ans;
    }
    int solve(int num) { ///拆数字
        int cnt = 0;
        while(num) {
            a[++cnt] = num % 10;
            num /= 10;
        }
        memset(dp, -1, sizeof dp); ///初始化-1 的原因就是因为有些状态的方案数可能是 0
        return dfs(cnt, 0, 1);
    }
} dp;
int main() {
    while(scanf("%d%d", &n, &m) && n + m) {
        printf("%d\n", dp.solve(m) - dp.solve(n - 1));
    }
}
```

整理人：网络 18-2 吴国庆

2.5 概率 Dp

(一般是正推)

///CF148D

原来袋子里有 w 只白鼠和 b 只黑鼠 龙和王妃轮流从袋子里抓老鼠。谁先抓到白色老师谁就赢。 王妃每次抓一只老鼠，龙每次抓完一只老鼠之后会有一只老鼠跑出来。 每次抓老鼠和跑出来的老鼠都是随机的。 如果两个人都没有抓到白色老鼠则龙赢。王妃先抓。 问王妃赢的概率。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1005;
const int mod = 1e9 + 7;
int w, b;
class Pdp {
```

```
public:
    double  dp[maxn][maxn];///设 dp[i][j]表示现在轮到王妃抓时有 i 只白鼠，j 只黑鼠，王妃赢的概率
    void init() {///初始化


        for(int i = 1; i <= w; i++)///因为都是白色老鼠，抓一次肯定赢了。
            dp[i][0] = 1;
        for(int i = 0; i <= b; i++)///因为没有白色老鼠了
            dp[0][i] = 0;
    }
    void solve() {///推出递推公式
        for(int i = 1; i <= w; i++) {
            for(int j = 1; j <= b; j++) {
                if(i + j == 0)
                    continue;
                dp[i][j] += i * 1.0 / (i + j); ///白
                if(j >= 3)
                    dp[i][j] += (j * 1.0) / (i + j) * (j - 1) * 1.0 / (i + j - 1) * (j - 2) / (i
+ j - 2) * dp[i][j - 3]; ///黑黑黑
                if(j >= 2)
                    dp[i][j] += (j * 1.0) / (i + j) * (j - 1) * 1.0 / (i + j - 1) * (i) * 1.0 /
(i + j - 2) * dp[i - 1][j - 2]; ///黑黑白
            }
        }
    }
    void printans() {///输出答案
        printf("%.10f\n", dp[w][b]);
    }
} dp;

int main() {
    while(~scanf("%d%d", &w, &b)) {
        dp.init();
        dp.solve();
        dp.printans();
    }


}
```

整理人：网络 18-2 吴国庆

2.6 期望 Dp

(一般是逆推)

///POJ2096 一个软件有 s 个子系统，会产生 n 种 bug
某人一天发现一个 bug,这个 bug 属于一个子系统，属于一个分类
每个 bug 属于某个子系统的概率是 1/s,属于某种分类的概率是 1/n
问发现 n 种 bug,每个子系统都发现 bug 的天数的期望。

```
const int N = 1005;
const int mod = 1e9 + 7;
int n, m;
class Edp {
public :
    double dp[N][N];///dp[i][j]表示已经找到 i 种 bug,j 个系统的 bug，达到目标状态的天数的期望

    void init() {///初始化
        dp[n][m] = 0;///dp[n][m]=0;
    }
    void solve() {///推导出递推方程，每一种状态的概率不要算错(要用 double)
        for(int i = n; i >= 0; i--) {
            for(int j = m; j >= 0; j--) {
                if(i == n && j == m)
                    continue;
                double p1 = (i * j) * 1.0 / (n * m);
                double p2 = ((n - i) * j) * 1.0 / (n * m);
                double p3 = (i * (m - j)) * 1.0 / (n * m);
                double p4 = ((n - i) * (m - j)) * 1.0 / (n * m);
                dp[i][j] = (dp[i + 1][j] * p2 + dp[i][j + 1] * p3 + dp[i + 1][j + 1] * p4
+ 1.0) / (1 - p1);
            }
        }
    }
    void printans() {///输出
        printf("%.4f\n", dp[0][0]);///要求的答案
    }

} dp;
```

整理人：网络 18-2 吴国庆

第 3 章 数据结构

3.1 一维树状数组

```
//洛谷 p3368 区间更新，单点查询
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

#define LL long long
const int maxn =5e5 + 7;

LL C[maxn];
int n,m;

class BITree {
public :

	int lowbit(int x) {
		return x & (-x);
	}

	void update(int x,LL y) {
		for(int i = x ; i <= n ; i += lowbit(i)) {
			C[i] += y;
		}
	}

	LL query(int x) {
		LL ans = 0;
		for(int i = x ; i > 0 ; i -= lowbit(i)) {
			ans += C[i];
		}
		return ans;
	}
}bitree;

int main() {
	while(~scanf("%d%d",&n,&m)) {
		memset(C,0,sizeof(C));
```

```
            LL temp = 0;
            for(int i = 1 ; i <= n ; i++) {
                LL x;
                scanf("%lld",&x);
                bitree.update(i,x-temp);
                temp = x;
            }
            while(m--) {
                int opt;
                scanf("%d",&opt);
                if(opt == 1) {
                    int x,y;
                    LL k;
                    scanf("%d%d%lld",&x,&y,&k);
                    bitree.update(x,k);
                    bitree.update(y+1,-k);
                } else {
                    int x;
                    scanf("%d",&x);
                    printf("%lld\n",bitree.query(x));
                }
            }
        }
        return 0;
}
```

整理人：计 18-9 胡小文

3.2 二维树状数组

```
//hdu1892 单点更新，区间求和
#include <map>
#include <cmath>
#include <vector>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

#define LL long long

const int maxn = 1010;
const int mod = 99991;
```

```
const int INF = 0x3f3f3f3f;

int c[maxn][maxn];

class BITree_2 {
public:

	int lowbit(int x) {
		return x & (-x);
	}

	void update(int x,int y,int val) {
		for(int i = x ; i <= 1001 ; i += lowbit(i)) {
			for(int j = y ; j <= 1001 ; j += lowbit(j)) {
				c[i][j] += val;
			}
		}
	}

	int query(int x,int y) {
		int ans = 0;
		for(int i = x ; i > 0 ; i -= lowbit(i)) {
			for(int j = y ; j > 0 ; j -= lowbit(j)) {
				ans += c[i][j];
			}
		}
		return ans;
	}
}bitree_2;

int main() {
	int T;
	scanf("%d",&T);
	int cas = 1;
	while(T--) {
		int Q;
		scanf("%d",&Q);
		printf("Case %d:\n",cas++);
		memset(c,0,sizeof(c));
		while(Q--) {
			char c;
			scanf(" %c",&c);
			if(c == 'S') {
				int x1,y1,x2,y2;
				scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
				x1++ , y1++ , x2++ , y2++;
```

```
                    if(x1 > x2) swap(x1,x2);
                    if(y1 > y2) swap(y1,y2);
                    printf("%d\n",bitree_2.query(x2,y2) - bitree_2.query(x2,y1-1) - bitree_2.query(x1-1,y2) + bitree_2.query(x1-1,y1-1) + (x2 - x1 + 1) * (y2 - y1 + 1));
                } else if(c == 'A') {
                    int x,y,val;
                    scanf("%d%d%d",&x,&y,&val);
                    x++ , y++;
                    bitree_2.update(x,y,val);
                } else if(c == 'D') {
                    int x,y,val;
                    scanf("%d%d%d",&x,&y,&val);
                    x++ , y++;
                    int temp = bitree_2.query(x,y) - bitree_2.query(x,y-1) - bitree_2.query(x-1,y) + bitree_2.query(x-1,y-1);
                    temp++;
                    temp = min(temp,val);
                    bitree_2.update(x,y,-temp);
                } else {
                    int x1,y1,x2,y2,val;
                    scanf("%d%d%d%d%d",&x1,&y1,&x2,&y2,&val);
                    x1++ , y1++ , x2++ , y2++;
                    int temp = bitree_2.query(x1,y1) - bitree_2.query(x1,y1-1) - bitree_2.query(x1-1,y1) + bitree_2.query(x1-1,y1-1);
                    temp++;
                    temp = min(temp,val);
                    bitree_2.update(x1,y1,-temp);
                    bitree_2.update(x2,y2,temp);
                }
            }
        }
        return 0;
}
```

整理人：计 18-9 胡小文

3.3 ST 表

```
//洛谷 p3865 题意：查询静态区间最大值
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
```

```
using namespace std;

#define LL long long
const int maxn =1e5 + 7;

int dp[maxn][30];
int a[maxn];

class ST {
public :
	void init_ST(int n) {
		memset(dp,0,sizeof(dp));
		for(int i = 1 ; i <= n ; i++) {
			dp[i][0] = a[i];
		}
		for(int j = 1 ; (1 << j) <= n ; j++) {
			for(int i = 1 ; i + (1 << j) - 1 <= n ; i++) {
				dp[i][j] = max(dp[i][j-1],dp[i + (1 << (j-1))][j-1]);
			}
		}
	}

	int query(int l,int r) {
		int k = log(r - l + 1.0) / log(2.0);
		return max(dp[l][k],dp[r - (1 << k) + 1][k]);
	}
}st;
int main() {
	int n,m;
	while(~scanf("%d%d",&n,&m)) {
		for(int i = 1 ; i <= n ; i++) {
			scanf("%d",&a[i]);
		}
		st.init_ST(n);
		while(m--) {
			int l,r;
			scanf("%d%d",&l,&r);
			printf("%d\n",st.query(l,r));
		}
	}
	return 0;
}
```

整理人：计 18-9 胡小文

3.4 RMQ

```
//poj3264  静态区间最大值减最小值
#include <map>
#include <cmath>
#include <vector>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

#define LL long long

const int maxn = 1e5 + 7;
const int mod = 99991;
const int INF = 0x3f3f3f3f;

int MIN[maxn << 2],MAX[maxn << 2];

class RMQ {
public :

    #define lson l , m , rt<<1
    #define rson m+1 , r , rt<<1|1

    void push_up(int rt) {
        MIN[rt] = min(MIN[rt << 1] , MIN[rt << 1|1]);
        MAX[rt] = max(MAX[rt << 1] , MAX[rt << 1|1]);
    }

    void build(int l,int r,int rt) {
        if(l == r) {
            scanf("%d",&MIN[rt]);
            MAX[rt] = MIN[rt];
            return ;
        }
        int m = (l + r) >> 1;
        build(lson);
        build(rson);
        push_up(rt);
    }

    int query_min(int L,int R,int l,int r,int rt) {
```

```
        if(L <= l && r <= R) {
            return MIN[rt];
        }
        int IMIN = INF;
        int m = (l + r) >> 1;
        if(L <= m) IMIN = min(IMIN,query_min(L,R,lson));
        if(R > m) IMIN = min(IMIN,query_min(L,R,rson));
        return IMIN;
    }

    int query_max(int L,int R,int l,int r,int rt) {
        if(L <= l && r <= R) {
            return MAX[rt];
        }
        int IMAX = 0;
        int m = (l + r) >> 1;
        if(L <= m) IMAX = max(IMAX,query_max(L,R,lson));
        if(R > m) IMAX = max(IMAX,query_max(L,R,rson));
        return IMAX;
    }

}rmq;

int main() {
    int n,q;
    while(~scanf("%d%d",&n,&q)) {
        rmq.build(1,n,1);
        while(q--) {
            int l,r;
            scanf("%d%d",&l,&r);
            printf("%d\n",rmq.query_max(l,r,1,n,1) - rmq.query_min(l,r,1,n,1));
        }
    }
    return 0;
}
```

注：这是基于线段树实现 RMQ，它支持动态 RMQ 问题的求解，至于区间修改的代码，请详见线段树模板。

整理人：计 18-9 胡小文

3.5 线段树

3.5.1 线段树 lazy 标记

```
///poj3468 题意：n 个数，m 次操作，每次区间加上一个数，区间查询和
#include<bits/stdc++.h>
using namespace std;
#define rep(i,j,k) for(int i=j;i<=k;i++)
#define debug puts("*");
const int N=220;
int n,cnt,t=1;
struct node{
	int l,r,cnt;
	double len;
}tree[N<<2];
struct knode{
	double x1,y1,y2;
	int k;
	friend bool operator <(knode a,knode b){
		return a.x1<b.x1;
	}
}line[N];
double raw[N],b[N],val[N];
void discrete(){
	sort(raw+1,raw+2*n+1);
//	rep(i,1,2*n)cout<<raw[i]<<" ";puts("");
	rep(i,1,2*n)
		if(i==1||raw[i]!=raw[i-1])
			b[++cnt]=raw[i];
//	rep(i,1,cnt)cout<<b[i]<<" ";
}
int findx(double x){
	return lower_bound(b+1,b+cnt+1,x)-b;
}
void pushup(int rt,int l,int r){
	if(tree[rt].cnt){
		tree[rt].len=b[r+1]-b[l];
	}
	else if(l!=r){
		tree[rt].len=tree[rt<<1].len+tree[rt<<1|1].len;
	}else tree[rt].len=0;
	return ;
}
void build(int rt,int l,int r){
	tree[rt].l=l,tree[rt].r=r;
```

```
        tree[rt].len=0.0;tree[rt].cnt=0;
//      cout<<l<<" "<<r<<endl;
        if(l==r)return;
        int mid=(l+r)>>1;
        build(rt<<1,l,mid);
        build(rt<<1|1,mid+1,r);
}
void update(int rt,int l,int r,int x){

        if(l<=tree[rt].l&&tree[rt].r<=r){
                tree[rt].cnt+=x;
                pushup(rt,tree[rt].l,tree[rt].r);
                return;
        }
        int mid=(tree[rt].l+tree[rt].r)>>1;      ///2
        if(l<=mid)update(rt<<1,l,r,x);
        if(r>mid)update(rt<<1|1,l,r,x);
        pushup(rt,tree[rt].l,tree[rt].r);
}
int main(){
        while(~scanf("%d",&n)){
                if(!n)break;
                cnt = 0;          /// 1
                rep(i,1,n){
                        double x1,y1,x2,y2;
                        scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
                        raw[i*2-1]=y1,raw[i*2]=y2;
                        line[i*2-1].x1=x1,line[i*2-1].y1=y1,line[i*2-1].y2=y2;
                        line[i*2-1].k=1;
                        line[i*2].x1=x2,line[i*2].y1=y1,line[i*2].y2=y2;
                        line[i*2].k=-1;
                }
                discrete();
                sort(line+1,line+2*n+1);
                double ans=0;
                build(1,1,cnt);
                rep(i,1,2*n){
                        ans+=tree[1].len*(line[i].x1-line[i-1].x1);
                        update(1,findx(line[i].y1),findx(line[i].y2)-1,line[i].k);
                }
                printf("Test case #%d\nTotal explored area: %.2lf\n\n", t++, ans);
        }
}
```

整理人：计 18-8 杨睿

3.5.2 线段树区间合并

///HDU 3911 题意：反转操作：每次反转一个区间内 0 1 的值，查询操作：查询最长的 1
代码：

```
#include<bits/stdc++.h>
using namespace std;
#define lson rt<<1
#define rson rt<<1|1
const int N=1e5+9;
int a[N],n,m;
int tree0[N<<2],tree1[N<<2],lazy[N<<2],lmax0[N<<2];
int lmax1[N<<2],rmax0[N<<2],rmax1[N<<2];

void swp(int rt){
	swap(tree0[rt],tree1[rt]);
	swap(rmax0[rt],rmax1[rt]);
	swap(lmax0[rt],lmax1[rt]);
	return;
}

void pushup(int rt){
	if(!tree0[lson])lmax1[rt]=lmax1[lson]+lmax1[rson];
	else lmax1[rt]=lmax1[lson];
	if(!tree1[lson])lmax0[rt]=lmax0[lson]+lmax0[rson];
	else lmax0[rt]=lmax0[lson];
	if(!tree0[rson])rmax1[rt]=rmax1[rson]+rmax1[lson];
	else rmax1[rt]=rmax1[rson];
	if(!tree1[rson])rmax0[rt]=rmax0[rson]+rmax0[lson];
	else rmax0[rt]=rmax0[rson];
	tree0[rt]=max(rmax0[lson]+lmax0[rson],max(tree0[lson],tree0[rson]));
	tree0[rt]=max(tree0[rt],max(tree0[lson],tree0[rson]));
	tree0[rt]=max(tree0[rt],max(lmax0[rt],rmax0[rt]));

	tree1[rt]=max(rmax1[lson]+lmax1[rson],max(tree1[lson],tree1[rson]));
	tree1[rt]=max(tree1[rt],max(tree1[lson],tree1[rson]));
	tree1[rt]=max(tree1[rt],max(lmax1[rt],rmax1[rt]));
}
void pushdown(int rt){
	if(lazy[rt]){
		swp(lson);
		swp(rson);
		lazy[lson]^=1;
		lazy[rson]^=1;
		lazy[rt]=0;
	}
	return;
```

```
}
void build(int rt,int l,int r){
	lazy[rt]=0;
	if(l==r){
		scanf("%d",&a[l]);
		if(a[l]==1){
			tree0[rt]=0;tree1[rt]=1;
			lmax0[rt]=0;rmax0[rt]=0;
			lmax1[rt]=1;rmax1[rt]=1;
		}
		else{
			tree0[rt]=1;tree1[rt]=0;
			lmax0[rt]=1;rmax0[rt]=1;
			lmax1[rt]=0;rmax1[rt]=0;
		}
//		  cout<<l<<endl;
		return;
	}
	int mid=(l+r)>>1;
	build(lson,l,mid);
	build(rson,mid+1,r);
	pushup(rt);
}
void change(int rt,int l,int r,int x,int y){
	if(l>=x&&r<=y){
		lazy[rt]^=1;
		swp(rt);
		return;
	}
	int mid=(l+r)>>1;
	pushdown(rt);
	if(x<=mid)change(lson,l,mid,x,y);
	if(y>mid)change(rson,mid+1,r,x,y);
	pushup(rt);
}
int query(int rt,int l,int r,int x,int y){
	if(l>=x&&r<=y){
		return tree1[rt];
	}
	int ans=0;
	pushdown(rt);
	int mid=(l+r)>>1;
	if(x<=mid){ans=max(query(lson,l,mid,x,y),ans);}
	if(y>mid){ans=max(query(rson,mid+1,r,x,y),ans);}
	ans=max(ans,min(mid-x+1,rmax1[lson])+min(y-mid,lmax1[rson]));
	return ans;
```

```
}
int main(){
	while(~scanf("%d",&n)){
		build(1,1,n);

		scanf("%d",&m);
		while(m--){
			int op,l,r;
			scanf("%d%d%d",&op,&l,&r);
			if(op==0){
				printf("%d\n",query(1,1,n,l,r));
			}
			else{
				change(1,1,n,l,r);
			}
//			  for(int i=1;i<=4*n;i++){
//			  printf("%d ",tree1[i]);
//		  }puts("");
		}
	}
}
```

整理人：计 18-8 杨睿

3.5.3 扫描线算法

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i,j,k) for(int i=j;i<=k;i++)
#define debug puts("*");
const int N=220;
int n,cnt,t=1;
struct node{
	int l,r,cnt;
	double len;
}tree[N<<2];
struct knode{
	double x1,y1,y2;
	int k;
	friend bool operator <(knode a,knode b){
		return a.x1<b.x1;
	}
}line[N];
double raw[N],b[N],val[N];
void discrete(){
	sort(raw+1,raw+2*n+1);
```

```
//      rep(i,1,2*n)cout<<raw[i]<<" ";puts("");
        rep(i,1,2*n)
            if(i==1||raw[i]!=raw[i-1])
                b[++cnt]=raw[i];
//      rep(i,1,cnt)cout<<b[i]<<" ";
}
int findx(double x){
        return lower_bound(b+1,b+cnt+1,x)-b;
}
void pushup(int rt,int l,int r){
        if(tree[rt].cnt){
            tree[rt].len=b[r+1]-b[l];
        }
        else if(l!=r){
            tree[rt].len=tree[rt<<1].len+tree[rt<<1|1].len;
        }else tree[rt].len=0;
        return ;
}
void build(int rt,int l,int r){
        tree[rt].l=l,tree[rt].r=r;
        tree[rt].len=0.0;tree[rt].cnt=0;
//      cout<<l<<" "<<r<<endl;
        if(l==r)return;
        int mid=(l+r)>>1;
        build(rt<<1,l,mid);
        build(rt<<1|1,mid+1,r);
}
void update(int rt,int l,int r,int x){

        if(l<=tree[rt].l&&tree[rt].r<=r){
            tree[rt].cnt+=x;
            pushup(rt,tree[rt].l,tree[rt].r);
            return;
        }
        int mid=(tree[rt].l+tree[rt].r)>>1;     ///2
        if(l<=mid)update(rt<<1,l,r,x);
        if(r>mid)update(rt<<1|1,l,r,x);
        pushup(rt,tree[rt].l,tree[rt].r);
}
int main(){
        while(~scanf("%d",&n)){
            if(!n)break;
            cnt = 0;        /// 1
            rep(i,1,n){
                double x1,y1,x2,y2;
                scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
```

```
            raw[i*2-1]=y1,raw[i*2]=y2;
            line[i*2-1].x1=x1,line[i*2-1].y1=y1,line[i*2-1].y2=y2;
            line[i*2-1].k=1;
            line[i*2].x1=x2,line[i*2].y1=y1,line[i*2].y2=y2;
            line[i*2].k=-1;
        }
        discrete();
        sort(line+1,line+2*n+1);
        double ans=0;
        build(1,1,cnt);
        rep(i,1,2*n){
            ans+=tree[1].len*(line[i].x1-line[i-1].x1);
            update(1,findx(line[i].y1),findx(line[i].y2)-1,line[i].k);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n", t++, ans);
    }
}
```

整理人：计 18-8 杨睿

3.6 树链剖分

3.6.1 点剖

///洛谷 P3384
题意：对于一颗树有如下操作
操作 1: 1 x y z 表示将树从 x 到 y 结点最短路径上所有节点的值都加上 z。
操作 2: 2 x y 表示求树从 x 到 y 结点最短路径上所有节点的值之和。
操作 3: 3 x z 表示将以 x 为根节点的子树内所有节点值都加上 z。
操作 4: 4 x 表示求以 x 为根节点的子树内所有节点值之和。

```
#include <bits/stdc++.h>
#define ls rt<<1
#define rs rt<<1|1
using namespace std;
typedef long long ll;
const int N=1e5+5;
int n,m,mod;
struct Edge{int next,to;}edge[N<<1];///边
int etot,head[N],Root;
int a[N];

inline void add(int u,int v){
    edge[etot]={head[u],v};
```

```
    head[u]=etot++;
}

inline void init(){
    memset(head,-1,sizeof(head));dfscnt=etot=0;
}

class treeChainSubdivision{
public:
    int son[N],tid[N],fa[N],dep[N],siz[N],top[N],rnk[N],dfscnt;
///son 重儿子编号,tid 是 dfs 序,fa 父亲节点,dep 深度从 1 开始的
///siz 子树大小,top 当前链顶端节点,dfscnt 是 dfs 序,rnk 是 dfs 序为 i 的节点
///多组输入时 son 要初始化！！！
    inline void dfs1(int x,int f,int deep){
        dep[x]=deep;
        fa[x]=f;
        siz[x]=1;
        int maxson=-1;
        for(int i=head[x];i!=-1;i=edge[i].next){
            int to=edge[i].to;
            if(to==f)continue;
            dfs1(to,x,deep+1);
            siz[x]+=siz[to];
            if(siz[to]>maxson)son[x]=to,maxson=siz[to];
        }
    }
    inline void dfs2(int x,int topf){
        tid[x]=++dfscnt;
        rnk[dfscnt]=x;
        top[x]=topf;
        if(!son[x])return;
        dfs2(son[x],topf);
        for(int i=head[x];i!=-1;i=edge[i].next){
            int to=edge[i].to;
            if(to==fa[x]||to==son[x])continue;
            dfs2(to,to);
        }
    }
    inline void updpath(int x,int y,int val){
        val%=mod;
        while(top[x]!=top[y]){
            if(dep[top[x]]<dep[top[y]])swap(x,y);
            Tree.update(1,1,n,tid[top[x]],tid[x],val);
            x=fa[top[x]];
        }
        if(dep[x]>dep[y])swap(x,y);
```

```
        Tree.update(1,1,n,tid[x],tid[y],val);
    }
    inline int Qpath(int x,int y){
        int ans=0;
        while(top[x]!=top[y]){
            if(dep[top[x]]<dep[top[y]])swap(x,y);
            ans=(ans+Tree.query(1,1,n,tid[top[x]],tid[x]))%mod;
            x=fa[top[x]];
        }
        if(dep[x]>dep[y])swap(x,y);
        ans=(ans+Tree.query(1,1,n,tid[x],tid[y]))%mod;
        return ans%mod;
    }
    inline void updson(int x,int val){
        Tree.update(1,1,n,tid[x],tid[x]+siz[x]-1,val);
    }
    inline int Qson(int x){
        return Tree.query(1,1,n,tid[x],tid[x]+siz[x]-1);
    }
}TCS;

class segmentTree{
public:
    int tree[N<<2],laz[N<<2];
    inline void build(int rt,int l,int r){
        if(l==r){
            tree[rt]=a[TCS.rnk[l]]%mod;
            return;
        }
        int mid=(l+r)>>1;
        build(ls,l,mid);
        build(rs,mid+1,r);
        tree[rt]=(tree[ls]+tree[rs])%mod;
    }
    inline void pushdown(int rt,int l,int r){
        if(laz[rt]){
            int mid=(l+r)>>1;
            laz[ls]=(laz[ls]+laz[rt])%mod;
            laz[rs]=(laz[rs]+laz[rt])%mod;
            tree[ls]=(tree[ls]+laz[rt]*(mid-l+1)%mod)%mod;
            tree[rs]=(tree[rs]+laz[rt]*(r-mid)%mod)%mod;
            laz[rt]=0;
        }
    }
    inline void update(int rt,int l,int r,int ql,int qr,int val){
        if(ql<=l&&r<=qr){
```

```
            laz[rt]=(laz[rt]+val)%mod;
            tree[rt]=(tree[rt]+(r-l+1)*val%mod)%mod;
            return ;
        }
        pushdown(rt,l,r);
        int mid=(l+r)>>1;
        if(ql<=mid)update(ls,l,mid,ql,qr,val);
        if(qr>mid)update(rs,mid+1,r,ql,qr,val);
        tree[rt]=(tree[ls]+tree[rs])%mod;
    }
    inline int query(int rt,int l,int r,int ql,int qr){
        if(ql<=l&&r<=qr)return tree[rt];
        pushdown(rt,l,r);
        int mid=(l+r)>>1;
        int res=0;
        if(ql<=mid)res+=query(ls,l,mid,ql,qr);
        if(qr>mid)res+=query(rs,mid+1,r,ql,qr);
        return res%mod;
    }
}Tree;

int main(){
    scanf("%d%d%d%d",&n,&m,&Root,&mod);
    for(int i=1;i<=n;i++)scanf("%d",&a[i]);
    for(int i=1;i<n;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        add(u,v);
        add(v,u);
    }
    TCS.dfs1(Root,0,1);
    TCS.dfs2(Root,Root);
    Tree.build(1,1,n);
    while(m--){
        int op,x,y,val;
        scanf("%d",&op);
        if(op==1){
            scanf("%d%d%d",&x,&y,&val);
            Tree.updpath(x,y,val);
        }
        else if(op==2){
            scanf("%d%d",&x,&y);
            printf("%d\n",TCS.Qpath(x,y));
        }
        else if(op==3){
            scanf("%d%d",&x,&y);
```

```
                TCS.updson(x,y);
            }
            else{
                scanf("%d",&x);
                printf("%d\n",TCS.Qson(x));
            }
        }
        return 0;
}
```

整理人：计 18-7 牛仔超

3.6.2 边剖

```
///洛谷 P4315
```
题意：对于一颗树有如下操作
操作 1: Change k w：将第 k 条树枝上毛毛果的个数改变为 w 个。
操作 2: Cover u v w：将节点 u 与节点 v 之间的树枝上毛毛果的个数都改变为 w 个。
操作 3: Add u v w：将节点 u 与节点 v 之间的树枝上毛毛果的个数都增加 w 个。
操作 4: Max u v：询问节点 u 与节点 v 之间树枝上毛毛果个数最多有多少个。

```
#include <bits/stdc++.h>
#define ls rt<<1
#define rs rt<<1|1
using namespace std;
typedef long long ll;
const int N=1e5+5;
int n,m,t,Root;
int a[N],rnk[N];
///a 用每条边的两个节点中深度较大的那个节点代表这条边的权值，rnk 是 dfs 序为 i 的节点

struct Edge{int next,to,val;}edge[N<<1];///边
int head[N],etot;

void add(int u,int v,int val){
    edge[etot]={head[u],v,val};
    head[u]=etot++;
}

void init(){///初始化
    memset(head,-1,sizeof(head));etot=0;
}

class segmentTree{
public:
    int tree[N<<2],laz[N<<2],laz2[N<<2];///laz 是赋值 laz，laz2 是加法 laz
```

```
void build(int rt,int l,int r){
	laz[rt]=-1;
	laz2[rt]=0;
	if(l==r){
		tree[rt]=a[rnk[l]];
		return ;
	}
	int mid=(l+r)>>1;
	build(ls,l,mid);
	build(rs,mid+1,r);
	tree[rt]=max(tree[ls],tree[rs]);
}
void pushdown(int rt,int l,int r){
	if(laz[rt]!=-1){
		laz2[ls]=laz2[rs]=0;
		tree[ls]=laz[ls]=laz[rt];
		tree[rs]=laz[rs]=laz[rt];
		laz[rt]=-1;
	}
	if(laz2[rt]){
		tree[ls]+=laz2[rt];
		tree[rs]+=laz2[rt];
		laz2[ls]+=laz2[rt];
		laz2[rs]+=laz2[rt];
		laz2[rt]=0;
	}
}
void updateadd(int rt,int l,int r,int ql,int qr,int val){
	if(ql>qr)return ;
	if(ql<=l&&r<=qr){
		tree[rt]+=val;
		laz2[rt]+=val;
		return ;
	}
	pushdown(rt,l,r);
	int mid=(l+r)>>1;
	if(ql<=mid)updateadd(ls,l,mid,ql,qr,val);
	if(qr>mid)updateadd(rs,mid+1,r,ql,qr,val);
	tree[rt]=max(tree[ls],tree[rs]);
}
void updatefu(int rt,int l,int r,int ql,int qr,int val){
	if(ql>qr)return ;
	if(ql<=l&&r<=qr){
		tree[rt]=val;
		laz[rt]=val;
		laz2[rt]=0;
```

```
            return ;
        }
        pushdown(rt,l,r);
        int mid=(l+r)>>1;
        if(ql<=mid)updatefu(ls,l,mid,ql,qr,val);
        if(qr>mid)updatefu(rs,mid+1,r,ql,qr,val);
        tree[rt]=max(tree[ls],tree[rs]);
    }
    inline int query(int rt,int l,int r,int ql,int qr){
        if(ql>qr)return 0;
        if(ql<=l&&r<=qr)return tree[rt];
        pushdown(rt,l,r);
        int mid=(l+r)>>1;
        int res=0;
        if(ql<=mid)res=max(res,query(ls,l,mid,ql,qr));
        if(qr>mid)res=max(res,query(rs,mid+1,r,ql,qr));
        return res;
    }
}Tree;

class TreeChainSubdivision{
public:
    int son[N],tid[N],fa[N],dep[N],siz[N],top[N],dfscnt=0;
///son 重儿子编号,tid 是 dfs 序,fa 父亲节点,dep 深度从 1 开始的
///siz 子树大小,top 当前链顶端节点,dfscnt 是 dfs 序,
///多组输入时 son 要初始化！！！
    void dfs1(int x,int f,int deep){
        dep[x]=deep;
        siz[x]=1;
        fa[x]=f;
        int maxson=-1;
        for(int i=head[x];i!=-1;i=edge[i].next){
            int to=edge[i].to;
            if(to==f)continue;
            a[to]=edge[i].val;
            dfs1(to,x,deep+1);
            siz[x]+=siz[to];
            if(siz[to]>maxson)son[x]=to,maxson=siz[to];
        }
    }
    void dfs2(int x,int topf){
        tid[x]=++dfscnt;
        rnk[dfscnt]=x;
        top[x]=topf;
        if(!son[x])return ;
        dfs2(son[x],topf);
```

```
        for(int i=head[x];i!=-1;i=edge[i].next){
            int to=edge[i].to;
            if(to==fa[x]||to==son[x])continue;
            dfs2(to,to);
        }
    }
    void updpathadd(int x,int y,int val){
        while(top[x]!=top[y]){
            if(dep[top[x]]<dep[top[y]])swap(x,y);
            Tree.updateadd(1,1,n,tid[top[x]],tid[x],val);
            x=fa[top[x]];
        }
        if(dep[x]>dep[y])swap(x,y);
        Tree.updateadd(1,1,n,tid[x]+1,tid[y],val);
    }
    void updpathfu(int x,int y,int val){
        while(top[x]!=top[y]){
            if(dep[top[x]]<dep[top[y]])swap(x,y);
            Tree.updatefu(1,1,n,tid[top[x]],tid[x],val);
            x=fa[top[x]];
        }
        if(dep[x]>dep[y])swap(x,y);
        Tree.updatefu(1,1,n,tid[x]+1,tid[y],val);
    }
    int Qpath(int x,int y){
        int ans=0;
        while(top[x]!=top[y]){
            if(dep[top[x]]<dep[top[y]])swap(x,y);
            ans=max(ans,Tree.query(1,1,n,tid[top[x]],tid[x]));
            x=fa[top[x]];
        }
        if(dep[x]>dep[y])swap(x,y);
        ans=max(ans,Tree.query(1,1,n,tid[x]+1,tid[y]));
        return ans;
    }
}TCS;

int main(){
    init();
    scanf("%d",&n);
    int u,v,k,val;
    for(int i=1;i<=n-1;i++){
        scanf("%d%d%d",&u,&v,&val);
        add(u,v,val);
        add(v,u,val);
    }
```

```
    Root=1;
    TCS.dfs1(Root,0,1);
    TCS.dfs2(Root,Root);
    Tree.build(1,1,n);
    char op[10];
    while(~scanf("%s",op)){
        if(op[1]=='h'){
            scanf("%d%d",&k,&val);
            k--;
            u=edge[k<<1].to;
            v=edge[k<<1|1].to;
            int now=TCS.dep[u]>TCS.dep[v]?u:v;
            Tree.updatefu(1,1,n,TCS.tid[now],TCS.tid[now],val);
        }
        else if(op[1]=='o'){
            scanf("%d%d%d",&u,&v,&val);
            TCS.updpathfu(u,v,val);
        }
        else if(op[1]=='d'){
            scanf("%d%d%d",&u,&v,&val);
            TCS.updpathadd(u,v,val);
        }
        else if(op[1]=='a'){
            scanf("%d%d",&u,&v);
            printf("%d\n",TCS.Qpath(u,v));
        }
        else break;
    }
    return 0;
}
```

整理人：计 18-7 牛仔超

3.7 分块算法

HDU3468 题意：n 个数，q 次询问吗，每次询问给出一个字母 s，若 s = ‘C’，则给出一组 x，y，z 将[x, y]中所有数加上 z。若 s = ‘Q’，则给出一组 x，y，问[x, y]的区间和是多少。

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
using namespace std;
const int N = 1e5 + 7;
typedef long long ll;
```

```
class Block
{
public:
    int pos[N], L[N], R[N], block, num, n, m;
    ll add[N], a[N], sum[N];
    void build()              //初始化
    {
        block = sqrt(n);
        num = block;
        if(n % block)
            num++;
        //num 表示块的个数，block 表示块的大小。
        for(int i = 1; i <= num; i++)
        {
            L[i] = (i - 1) * block + 1;        //该块左端点位置
            R[i] = i * block;                  //该块右端点位置
        }
        R[num] = n;
        for(int i = 1; i <= num; i++)
        {
            for(int j = L[i]; j <= R[i]; j++)
            {
                pos[j] = i;               //第几个块
                sum[i] += a[j];           //块的区间和
                add[j] = 0;               //增量标记
            }
        }
    }
    void change(int l, int r, ll d)        //修改操作
    {
        int p = pos[l];
        int q = pos[r];
        if(p == q)
        {
            for(int i = l; i <= r; i++)
                a[i] += d;
            sum[p] += (r - l + 1) * d;
        }
        else
        {
            for(int i = p + 1; i <= q - 1; i++)
                add[i] += d;
            for(int i = l; i <= R[p]; i++)
                a[i] += d;
            sum[p] += (R[p] - l + 1) * d;
            for(int i = L[q]; i <= r; i++)
```

```
                a[i] += d;
            sum[q] += (r - L[q] + 1) * d;
        }
    }
    ll ask(int l, int r)            //查询操作
    {
        int p = pos[l];
        int q = pos[r];
        ll ans = 0;
        if(p == q)
        {
            for(int i = l; i <= r; i++)
                ans += a[i];
            ans += (r - l + 1) * add[p];
        }
        else
        {
            for(int i = p + 1; i <= q - 1; i++)
                ans += sum[i] + add[i] * (R[i] - L[i] + 1);
            for(int i = l; i <= R[p]; i++)
                ans += a[i];
            ans += add[p] * (R[p] - l + 1);
            for(int i = L[q]; i <= r; i++)
                ans += a[i];
            ans += add[q] * (r - L[q] + 1);
        }
        return ans;
    }

} q;
int main()
{
    scanf("%d %d", &q.n, &q.m);
    for(int i = 1; i <= q.n; i++)
        scanf("%lld", &q.a[i]);
    q.build();
    while(q.m--)
    {
        char c;
        int x, y;
        ll z;
        scanf(" %c", &c);
        if(c == 'Q')
        {
            scanf("%d %d", &x, &y);
            printf("%lld\n", q.ask(x, y));
```

```
            }
            else
            {
                scanf("%d %d %lld", &x, &y, &z);
                q.change(x, y, z);
            }
        }
        return 0;
}
```

整理人：网络 18-3 董文睿

3.8 莫队算法

BZOJ 2038 题意：给出两个整数 n 和 m，之后给出 n 个数代表 n 个袜子的颜色，m 次询问，每次给出一个[l, r]，问区间内的随机抽取两个袜子颜色相同的概率为多少，输出最简分数 A/B 的形式，若概率为零则输出 0/1。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 7;
const int M = 1e6 + 7;
const int INF = 1e9 + 8;
int block;
struct node
{
    int l, r, id;
};
ll gcd(ll x, ll y)
{
    return y == 0 ? x : gcd(y, x % y);
}
bool cmp(node xx, node yy)          //奇偶排序优化
{
    if(xx.l / block == yy.l / block)
    {
        if(xx.l / block & 1)
            return xx.r < yy.r;
        else
            return xx.r > yy.r;
    }
    return xx.l / block < yy.l / block;
}
class Moteam
{
```

```
public:
    int a[N], sum[N], n, m;
    ll ansx[N],ansy[N], ans;
    node q[N];
    void Add(int i)         //添加操作
    {
        ans -= sum[a[i]] * sum[a[i]];
        sum[a[i]]++;
        ans += sum[a[i]] * sum[a[i]];
    }
    void Sub(int i)
    {
        ans -= sum[a[i]] * sum[a[i]];
        sum[a[i]]--;
        ans += sum[a[i]] * sum[a[i]];
    }
    void solve()            //修改操作
    {
        sort(q + 1, q + m + 1, cmp);
        int l = 1;
        int r = 1;
        sum[a[1]]++;
        ans = 1;
        for(int i = 1; i <= m; i++)
        {
            if(q[i].l == q[i].r)                    //特判一下
                ansx[q[i].id] = 0, ansy[q[i].id] = 1;
            while(l < q[i].l)
                Sub(l), l++;
            while(l > q[i].l)
                l--, Add(l);
            while(r < q[i].r)
                r++, Add(r);
            while(r > q[i].r)
                Sub(r), r--;
            ll x = (ans - (q[i].r - q[i].l + 1));
            ll y = (ll)(q[i].r - q[i].l + 1) * (ll)(q[i].r - q[i].l);
            ll d = gcd(x, y);
            ansx[q[i].id] = x / d;
            ansy[q[i].id] = y / d;
        }
        for(int i = 1; i <= m; i++)
            printf("%lld/%lld\n", ansx[i], ansy[i]);
    }
} mo;
int main()
```

```
{
    memset(mo.sum, 0, sizeof mo.sum);
    scanf("%d %d", &mo.n, &mo.m);
    for(int i = 1; i <= mo.n; i++)
        scanf("%d", &mo.a[i]);
    block = sqrt(mo.n);
    for(int i = 1; i <= mo.m; i++)
    {
        scanf("%d %d", &mo.q[i].l, &mo.q[i].r);
        mo.q[i].id = i;
    }
    mo.solve();
    return 0;
}
```

整理人：网络 18-3 董文睿

3.9 点分治

例一、树上距离为 k 的点对是否存在.

///洛谷 P3806

给定一颗 n 个结点的无根树,有 m 次询问,每次询问树上距离为 k 的点对是否存在.

思想:用桶记录路径,判断是否存在距离为 k 的点对

```
/**
```

使用方法:

调用 DFZ.sovle()函数后答案存入 ans[]数组

模块说明:

1.求树的重心函数 Root.getroot()

输入:

使用 Root.getroot(u,fa,sum),u 为当前结点,fa 为 u 结点的父亲结点,sum 是当前连通块的大小

输出:

返回重心结点编号 rt

2.计算所有结点到根节点的距离函数 CalDis.caldis()

输入:

使用 CalDis.caldis(int u,int fa),u 为当前结点,fa 为 u 结点的父亲结点

输出:

得到 di[]数组,di[]数组大小为 tp,存有所有基本路径的长度,还有 dis[]数组,存的是结点 u 到当前根节点的长度

3.计算合法路径函数 SovleDis.sovle()
输入:
使用 SovleDis.sovle(int u),u 为当前结点
输出:
得到当前 ans[]数组,,存有每次更新的答案信息

4.点分治函数 DFZ.dfz()
输入:
使用 DFZ.dfz(int u),u 为当前结点
输出:
递归各结点

5.点分治调用 DFZ.solve()
输入:
无
输出:
得到最终 ans[]数组,存有答案信息
*/

```
#include<bits/stdc++.h>
using namespace std;
const int amn=1e5+5,inf=1e9;
int n,m,K[amn];

///链式前向星存图
int head[amn],etot;
struct edge{
    int nxt,v,w;
}eg[amn];
void add(int u,int v,int w){
    eg[++etot]={head[u],v,w};
    head[u]=etot;
}

int vis[amn];

///求树的重心
class Root{
public:
    int siz[amn],maxt[amn],rt;
    void calsiz(int u,int fa,int sum){
        siz[u]=1;
        maxt[u]=0;
        for(int i=head[u];i;i=eg[i].nxt){
            int v=eg[i].v;
            if(vis[v]||v==fa)continue;
```

```
            calsiz(v,u,sum);
            siz[u]+=siz[v];
            maxt[u]=max(maxt[u],siz[v]);
        }
        maxt[u]=max(maxt[u],sum-siz[u]);
        if(maxt[u]<maxt[rt])rt=u;
    }
    void getroot(int u,int fa,int sum){
        rt=0;
        maxt[rt]=inf;
        calsiz(u,fa,sum);
    }
};

///求基本路径 dis
class CalDis{
    public:
    int dis[amn],di[amn],tp;
    void caldis(int u,int fa){
        if(dis[u]>(int)1e7)return;
        di[++tp]=dis[u];
        for(int i=head[u];i;i=eg[i].nxt){
            int v=eg[i].v,w=eg[i].w;
            if(vis[v]||v==fa)continue;
            dis[v]=dis[u]+w;
            caldis(v,u);
        }
    }
    void init(int v,int w){
        tp=0;
        dis[v]=w;
    }
};

///判断路径
bool jg[(int)1e7+1];
int ans[amn];
class SovleDis{
    public:
    CalDis cd;
    queue<int> bk;
    void sovle(int u){
        jg[0]=1;
        bk.push(0);
        for(int i=head[u];i;i=eg[i].nxt){
            int v=eg[i].v,w=eg[i].w;
```

```
            if(vis[v])continue;
            cd.init(v,w);
            cd.caldis(v,u);
            for(int j=1;j<=cd.tp;j++){
                for(int k=1;k<=m;k++){
                    if(K[k]>=cd.di[j])ans[k]+=jg[K[k]-cd.di[j]];
                }
            }
            for(int j=1;j<=cd.tp;j++){
                jg[cd.di[j]]=1;
                bk.push(cd.di[j]);
            }
        }
        while(bk.size()){
            jg[bk.front()]=0;
            bk.pop();
        }
    }
};

///点分治
class DFZ{
public:
    Root rt;
    SovleDis s;
    void dfz(int u){
        vis[u]=1;
        s.sovle(u);
        for(int i=head[u];i;i=eg[i].nxt){
            int v=eg[i].v;
            if(vis[v])continue;
            rt.getroot(v,u,rt.siz[v]);
            dfz(rt.rt);
        }
    }
    void sovle(){
        rt.getroot(1,-1,n);
        dfz(rt.rt);
    }
};


int main(){
    DFZ df;
    int a,b,c;
    scanf("%d%d",&n,&m);
```

```
    for(int i=1;i<=n-1;i++){
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);
        add(b,a,c);
    }
    for(int i=1;i<=m;i++){
        scanf("%d",&K[i]);
    }
    df.sovle();
    for(int i=1;i<=m;i++){
        if(ans[i])printf("AYE\n");
        else printf("NAY\n");
    }
}
```

例二、树上距离小于等于 K 的点对数量

///洛谷 P4178

给定一棵 n 个节点的树，每条边有边权，求出树上两点距离小于等于 k 的点对数量。

思想:用容斥和双指针记录路径,计算距离小于等于 k 的点对有多少个.

```
/**
```

使用方法:

调用 DFZ.sovle()函数后答案存入 ans

模块说明:

1.求树的重心函数 Root.getroot()

输入:

使用 Root.getroot(u,fa,sum),u 为当前结点,fa 为 u 结点的父亲结点,sum 是当前连通块的大小

输出:

返回重心结点编号 rt

2.计算所有结点到根节点的距离函数 CalDis.caldis()

输入:

使用 CalDis.caldis(int u,int fa),u 为当前结点,fa 为 u 结点的父亲结点

输出:

得到 di[]数组,di[]数组大小为 tp,存有所有基本路径的长度,还有 dis[]数组,存的是结点 u 到当前根节点的长度

3.计算合法路径函数 SovleDis.sovle()

输入:

使用 SovleDis.sovle(int u,int fa,int w),u 为当前结点,fa 为 u 结点的父亲结点,w 为 fa 到 u 的边权

输出:

得到当前 ans,存有每次更新的答案信息

4.点分治函数 DFZ.dfz()
输入:
使用 DFZ.dfz(int u),u 为当前结点
输出:
递归各结点

5.点分治调用 DFZ.solve()
输入:
无
输出:
得到最终 ans,存有答案信息
*/

```
#include<bits/stdc++.h>
using namespace std;
const int amn=1e5+5,inf=1e9;
int n,m,K;

///链式前向星存图
int head[amn],etot;
struct edge{
    int nxt,v,w;
}eg[amn];
void add(int u,int v,int w){
    eg[++etot]={head[u],v,w};
    head[u]=etot;
}

int vis[amn];

///求树的重心
class Root{
public:
    int siz[amn],maxt[amn],rt;
    void calsiz(int u,int fa,int sum){
        siz[u]=1;
        maxt[u]=0;
        for(int i=head[u];i;i=eg[i].nxt){
            int v=eg[i].v;
            if(vis[v]||v==fa)continue;
            calsiz(v,u,sum);
            siz[u]+=siz[v];
            maxt[u]=max(maxt[u],siz[v]);
        }
```

```
            maxt[u]=max(maxt[u],sum-siz[u]);
            if(maxt[u]<maxt[rt])rt=u;
        }
        void getroot(int u,int fa,int sum){
            rt=0;
            maxt[rt]=inf;
            calsiz(u,fa,sum);
        }
};

///求基本路径 dis
class CalDis{
        public:
        int dis[amn],di[amn],tp;
        void caldis(int u,int fa){
            if(dis[u]>K)return;
            di[++tp]=dis[u];
            for(int i=head[u];i;i=eg[i].nxt){
                int v=eg[i].v,w=eg[i].w;
                if(vis[v]||v==fa)continue;
                dis[v]=dis[u]+w;
                caldis(v,u);
            }
        }
        void init(int v,int w){
            tp=0;
            dis[v]=w;
        }
};

///判断路径
int ans;
class SovleDis{
        public:
        CalDis cd;
        int sovle(int u,int fa,int w){
            cd.init(u,w);
            cd.caldis(u,fa);
            sort(cd.di+1,cd.di+1+cd.tp);
            int l=1,r=cd.tp,ans=0;
            while(l<r){
                if(cd.di[l]+cd.di[r]<=K){
                    ans+=r-l;
                    l++;
                }
                else r--;
```

```
            }
            return ans;
        }
};

///点分治
class DFZ{
public:
        Root rt;
        SovleDis s;
        void dfz(int u){
                vis[u]=1;
                ans+=s.sovle(u,-1,0);
                for(int i=head[u];i;i=eg[i].nxt){
                        int v=eg[i].v,w=eg[i].w;
                        if(vis[v])continue;
                        ans-=s.sovle(v,u,w);
                        rt.getroot(v,u,rt.siz[v]);
                        dfz(rt.rt);
                }
        }
        void sovle(){
                rt.getroot(1,-1,n);
                dfz(rt.rt);
        }
};


int main(){
        DFZ df;
        int a,b,c;
        scanf("%d",&n);
        for(int i=1;i<=n-1;i++){
                scanf("%d%d%d",&a,&b,&c);
                add(a,b,c);
                add(b,a,c);
        }
        ans=0;
        scanf("%d",&K);
        df.sovle();
        printf("%d\n",ans);
}
```

整理人：计 18-8 蒙晟维

第 4 章 图论

4.1 二分图

4.1.1 匈牙利算法

///POJ1274 求最大匹配 题意：有 N 条牛和 M 个牛棚，每条牛都有自己喜欢的几个牛棚，问最多可以让多少条牛呆在自己喜欢的牛棚（牛棚和牛一对一）。

```
#include<cstdio>
#include<cstring>
using namespace std;
typedef long long ll;
const int N=1e5+10;
const int mod=1e9+7;
int d[205][205];            //d 数组标记有哪些边
int lin[205];          //lin 数组标记牛棚匹配到了哪头牛
int vi[205];        //vi 数组标记牛棚是否被访问过
int n,m;
int fin(int x)
{
    for(int i=1; i<=m; i++)
    {
        if(d[i][x]&&vi[i]==0)
        {
            vi[i]=1;
            if(lin[i]==0||fin(lin[i])==1)  //i 号牛棚还没有匹配牛或可以为它所匹配的牛找一个新的满意的牛棚
            {
                lin[i]=x;
                return 1;
            }
        }
    }
    return 0;
}
int main()
{
    while(~scanf("%d%d",&n,&m))
    {
        memset(d,0,sizeof(d));
        memset(lin,0,sizeof(lin));
        int c,u;
```

```
        for(int i=1; i<=n; i++)
        {
            scanf("%d",&c);
            while(c--)
            {
                scanf("%d",&u);
                d[i][u]=1;          //连边
            }
        }
        int cnt=0;
        for(int i=1; i<=n; i++)
        {
            memset(vi,0,sizeof(vi));    //vi 数组每次都要清
            if(fin(i)==1)               //i 号牛找到可以与它匹配的牛棚
                cnt++;
        }
        printf("%d\n",cnt);
    }
}
```

整理人：计 18-5 王佳妮

4.1.2 KM 算法

///HDU2255 求最优匹配 题意：有 N 家老百姓和 N 间房子，要给每家分配一间房，每个村民对不同的房子出价不同，现在村长要让利益最大化，问应怎么分配才能让村民出的钱数总和最大。

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
int mp[303][303]; //记录村民对各个房子出价情况
int lin[303];   //标记房子所匹配到的村民
int viy[303];   //标记房子是否被访问过
int vix[303];   //标记村民是否被访问过
int x[303];   //记录村民的点权
int y[303];   //记录房子的点权
int n;
int lack,t;   //lack 记录单次要改变的点权大小
int dfs(int u)
{
    vix[u]=1;
    for(int i=1; i<=n; i++)
    {
        if(!viy[i])
        {
```

```
                t=x[u]+y[i]-mp[u][i];
                if(t==0)       //t 为零代表 u 村民可以和 i 房子匹配
                {
                    viy[i]=1;
                    if(dfs(lin[i])||!lin[i])    //i 房子没匹配过或 i 房子匹配到的村民可以与其它房
子匹配
                    {
                        lin[i]=u;
                        return 1;
                    }
                }
                else if(lack>t)
                    lack=t;
            }
        }
        return 0;
}
void KM()
{
        memset(y,0,sizeof(y));
        memset(x,0,sizeof(x));
        memset(lin,0,sizeof(lin));
        for(int i=1; i<=n; i++)
            for(int j=1; j<=n; j++)
                x[i]=max(x[i],mp[i][j]);    //x[i]记录 i 村民所有出价中最大的值
        for(int i=1; i<=n; i++)
        {
            while(1)
            {
                memset(vix,0,sizeof(vix));
                memset(viy,0,sizeof(viy));
                lack=1e9+7;
                if(dfs(i))     //i 村民匹配成功
                    break;
                for(int i=1; i<=n; i++)
                {
                    if(vix[i])
                        x[i]-=lack;    //村民点权减小
                    if(viy[i])
                        y[i]+=lack;    //房子点权增大
                }
            }
        }
}
int main()
{
```

```
    while(~scanf("%d",&n))
    {
        memset(mp,0,sizeof(mp));
        for(int i=1; i<=n; i++)
            for(int j=1; j<=n; j++)
            {
                scanf("%d",&mp[i][j]);
            }
        KM();
        int ans=0;
        for(int i=1; i<=n; i++)
        {
            ans+=mp[lin[i]][i];   //求和
        }
        printf("%d\n",ans);
    }
}
```

整理人：计 18-5 王佳妮

4.2 网络流

4.2.1 EK 算法

```
///洛谷 P3376
typedef long long ll;
using namespace std;
const int INF = 2e9 + 7;
const int N = 2e4 + 7;
const int M = 2e5 + 7;

struct edge
{
    int to, nex, cap;
    edge(int to = 0, int nex = 0, int cap = 0) : to(to), nex(nex), cap(cap){}
};

class EK
{
public:
    int S, T, n, m, cnt;
    int head[N], vis[N];
    int pre[N], flow[N];
    edge e[M];

    void addedge(int u, int v, int cap)
```

```
{
    e[++cnt] = edge(v, head[u], cap);
    head[u] = cnt;
}

// 建图
void buildGraph(int _n, int _m, int _S, int _T)
{
    // 初始化部分
    n = _n; m = _m;
    S = _S; T = _T;
    mem(head); mem(vis);
    cnt = 1;

    // 构图部分
    for(int i = 1, u, v, cap; i <= m; i++)
    {
        read(u); read(v); read(cap);
        addedge(u, v, cap);
        addedge(v, u, 0);
    }
}

// 找增广路
bool bfs()
{
    mem(vis);
    queue<int> q;
    q.push(S);    vis[S] = 1; flow[S] = INF;

    int u, v;
    while(q.size())
    {
        u = q.front(); q.pop();
        for(int i = head[u]; i; i = e[i].nex)
        {
            v = e[i].to;
            if(vis[v] || e[i].cap <= 0) continue;
            flow[v] = min(flow[u], e[i].cap);
            pre[v] = i;
            q.push(v);    vis[v] = 1;

            if(v == T) return 1;
        }
    }
    return 0;
```

```
    }

    int update()
    {
        int u = T;
        while(u != S)
        {
            int i = pre[u];
            e[i].cap -= flow[T];
            e[i^1].cap += flow[T];
            u = e[i^1].to;
        }

        return flow[T];
    }

    int get_maxFlow()
    {
        int maxFlow = 0;

        while(bfs())        maxFlow += update();
        return maxFlow;
    }
};

EK ways;
signed main()
{
    int n, m, S, T;

    while(~scanf("%d%d%d%d", &n, &m, &S, &T))
    {
        ways.buildGraph(n, m, S, T);
        printf("%d\n", ways.get_maxFlow());
    }

    return 0;
}
```

整理人：网络 18-3 高云泽

4.2.2 Dinic+当前弧优化

```
///洛谷 P3376

typedef long long ll;
using namespace std;
const int INF = 2e9 + 7;
```

```
const int N = 2e4 + 7;
const int M = 2e5 + 7;
const int base = 100;

struct edge
{
	int to, nex, cap;
	edge(int to = 0, int nex = 0, int cap = 0) : to(to), nex(nex), cap(cap){}
};

class Dinic
{
public:
	int S, T, n, m, cnt;
	int head[N], d[N], cur[N];
	edge e[M];

	void addedge(int u, int v, int cap)
	{
		e[++cnt] = edge(v, head[u], cap);
		head[u] = cnt;
	}

	// 建图
	void buildGraph(int _n, int _m, int _S, int _T)
	{
		// 初始化部分
		n = _n; m = _m;
		S = _S; T = _T;
		mem(head);
		cnt = 1;

		// 构图部分
		for(int i = 1, u, v, cap; i <= m; i++)
		{
			read(u); read(v); read(cap);
			addedge(u, v, cap);
			addedge(v, u, 0);
		}
	}

	// 找增广路
	bool bfs()
	{
		mem(d);
		queue<int> q;
```

```
    q.push(S);    d[S] = 1;

    int u, v;
    while(q.size())
    {
        u = q.front(); q.pop();
        for(int i = head[u]; i; i = e[i].nex)
        {
            v = e[i].to;
            if(d[v] || e[i].cap <= 0) continue;
            d[v] = d[u] + 1;
            q.push(v);
        }
    }
    for(int i = 0; i <= n; i++) cur[i] = head[i];
    return d[T];
}

int dfs(int u, int flow)
{
    if(u == T) return flow;

    for(int& i = cur[u], v; i ; i = e[i].nex)
    {
        v = e[i].to;

        if(d[v] != d[u] + 1 || e[i].cap <= 0) continue;

        int delta = dfs(v, min(flow, e[i].cap));

        if(delta <= 0) continue;
        e[i].cap -= delta;
        e[i^1].cap += delta;

        return delta;
    }
    return 0;
}

int get_maxFlow()
{
    int maxFlow = 0, tmp;
    while(bfs())
        while(tmp = dfs(S, INF))
            maxFlow += tmp;
    return maxFlow;
```

```
    }

};

Dinic ways;
signed main()
{
    int n, m, S, T;

    while(~scanf("%d%d%d%d", &n, &m, &S, &T))
    {
        ways.buildGraph(n, m, S, T);
        printf("%d\n", ways.get_maxFlow());
    }

    return 0;
}
```

整理人：网络 18-3 高云泽

4.2.3 ISAP 算法：

```
//洛谷 P3376
typedef long long ll;
using namespace std;
const int INF = 2e9 + 7;
const int N = 2e4 + 7;
const int M = 2e5 + 7;
const int base = 100;
int n, m, S, T;

struct edge
{
    int to, nex, cap;
    edge(int to = 0, int nex = 0, int cap = 0) : to(to), nex(nex), cap(cap){}
};

class ISAP
{
public:
    int S, T, n, m, cnt;
    int head[N], d[N], cur[N], gap[N];
    edge e[M];

    void addedge(int u, int v, int cap)
    {
        e[++cnt] = edge(v, head[u], cap);
```

```
    head[u] = cnt;
}

// 建图
void buildGraph(int _n, int _m, int _S, int _T)
{
    // 初始化部分
    n = _n; m = _m;
    S = _S; T = _T;
    mem(head);
    cnt = 1;

    // 构图部分
    for(int i = 1, u, v, cap; i <= m; i++)
    {
        read(u); read(v); read(cap);
        addedge(u, v, cap);
        addedge(v, u, 0);
    }
}

// 找增广路
void bfs()
{
    queue<int> q;
    for(int i = 1; i <= n; i++)
    {
        cur[i] = head[i];
        d[i] = gap[i] = 0;
    }

    q.push(T);
    d[T] = gap[1] = 1;

    int u, v;
    while(q.size())
    {
        u = q.front(); q.pop();
        for(int i = head[u]; i; i = e[i].nex)
        {
            v = e[i].to;
            if(d[v]) continue;
            d[v] = d[u] + 1;
            ++gap[d[v]];
            q.push(v);
        }
```

```
        }
    }

    int dfs(int u, int flow)
    {
        if(u == T) return flow;
        int delta = 0, v, temp;
        for(int &i = cur[u]; i; i = e[i].nex)
        {
            v = e[i].to;
            if( d[v] + 1 != d[u]) continue;
            temp = dfs(v, min(flow-delta, e[i].cap));
            if(temp)
            {
                e[i].cap -= temp;
                e[i ^ 1].cap += temp;
                delta += temp;
            }
            if(flow == delta) return flow;
        }
        if(!(--gap[d[u]])) d[S] = n + 1;
        ++gap[++d[u]];
        cur[u] = head[u];
        return delta;
    }

    int get_maxFlow()
    {
        bfs();

        int maxFlow = 0;

        while(d[S] <= n)    maxFlow += dfs(S, INF);

        return maxFlow;
    }
};

ISAP ways;
signed main()
{
    int n, m, S, T;

    while(~scanf("%d%d%d%d", &n, &m, &S, &T))
    {
        ways.buildGraph(n, m, S, T);
```

```
        printf("%d\n", ways.get_maxFlow());
    }

    return 0;
}
```

例题：网络流 24 题 星际转移 Loj 6015

题意：现有 n 个太空站位于地球与月球之间，且有 m 艘公共交通太空船在其间来回穿梭。每个太空站可容纳无限多的人，而每艘太空船 i 只可容纳 Hi 个人。每艘太空船将周期性地停靠一系列的太空站，例如：1,3,4 表示该太空船将周期性地停靠太空站 134134134… 每一艘太空船从一个太空站驶往任一太空站耗时均为 1。人们只能在太空船停靠太空站（或月球、地球）时上、下船。 初始时所有人全在地球上，太空船全在初始站。试设计一个算法，找出让所有人尽快地全部转移到月球上的运输方案。

分析：这个题其实和“魔术球”这个题类似。题意就相当于一些人要从 A ---> B ，空间站为中转点，通过飞船转移，这些飞船每一天出现在固定空间站，也即是说天数不一定的时候，整个运输图就会发生变化，而每一天之间又是有关系的。就好像，每一天的空间站是不一样的（因为飞船飞行线路不一样了），所以考虑到 地球（0 号点）、n 个空间站和月亮（n + 1 号点）共 n + 2 个点，每一天分裂一次（即拥有一个新的状态），有因为共经历 t 天，完成运输，故 将要分裂出 t * (n + 2) 个点【即是说将每一天的这 n + 2 个点看成一个新的点】，枚举每一天 day，令 yesterday = (n + 2) * (day － 1)，today = (n + 2) * day；则每一个点 i’ = yesterday + i（代表昨天对应的点）, i = today + i（代表今天的点）；故 i’--->i，容量为 INF， (n + 1) + today ---> T，容量为 INF;

在从每一个飞船前一天所在的太空站连向后一天的太空站，流量为飞船可容纳人数；

在最开始的时候 链接一条， S ----> 0，容量为 INF 的边

跑最大流直到不小于总人数即可。（每一天的最大流应累加起来）

代码：

```
#pragma GCC optimize(2)
#include <bits/stdc++.h>
#define read(a) scanf("%d", &a)
#define readl(a) scanf("%lld", &a)
#define reads(a) scanf("%s", a)
#define readc(a) scanf("%c", &a)
#define pb push_back
#define mem(a) memset(a, 0, sizeof(a))
#define Buff ios::sync_with_stdio(false)
typedef long long ll;
using namespace std;
const int INF = 2e9 + 7;
const int N = 1e5 + 7;
const int M = 1e6 + 7;
const int _N = 102;
const int base = 100;
int n, m, S, T;

struct edge
```

```
{
    int to, nex, cap;
    edge(int to = 0, int nex = 0, int cap = 0) : to(to), nex(nex), cap(cap) {}
};
int pre[_N], station[_N][_N], r[_N], H[_N];
int head[N], d[N], cur[N], gap[N];
edge e[M];

class ISAP
{
public:
    int S, T, n, m, cnt, K;

    void addedge(int u, int v, int cap)
    {
        e[++cnt] = edge(v, head[u], cap);
        head[u] = cnt;

        e[++cnt] = edge(u, head[v], 0);
        head[v] = cnt;
        //          printf("%d --- %d, %d\n", u, v, cap);
    }

    // 建图
    void buildGraph()
    {
        // 初始化部分
        read(n);
        read(m);
        read(K);
        mem(head);
        cnt = 1;
        S = N - 2;
        T = N - 1;
        for (int i = 1; i <= m; i++)
        {
            read(H[i]);
            read(r[i]);
            for (int j = 1; j <= r[i]; j++)
                read(station[i][j]);
            pre[i] = 1;
        }
        n += 2;
        addedge(S, 0, INF);
        int ans = 0;
        for (int day = 1; day <= 30; day++)
```

```
    {
        int today = n * day, yesterday = n * (day - 1), now;
        for (int i = 0; i < n; i++)
            addedge(yesterday + i, today + i, INF);
        addedge(today + n - 1, T, INF);
        for (int i = 1; i <= m; i++)
        {
            now = pre[i] + 1;
            if (now > r[i])
                now = 1;
            addedge(station[i][pre[i]] + yesterday, station[i][now] + today, H[i]);
            pre[i] = now;
        }
        ans += get_maxFlow();
        if (ans >= K)
        {
            printf("%d\n", day);
            return;
        }
    }
    printf("%d\n", 0);
}

// 找增广路
void bfs()
{
    queue<int> q;
    for (int i = 0; i <= T; i++)
    {
        cur[i] = head[i];
        d[i] = gap[i] = 0;
    }

    q.push(T);
    d[T] = gap[1] = 1;

    int u, v;
    while (q.size())
    {
        u = q.front();
        q.pop();
        for (int i = head[u]; i; i = e[i].nex)
        {
            v = e[i].to;
            if (d[v])
                continue;
```

```
                d[v] = d[u] + 1;
                ++gap[d[v]];
                q.push(v);
            }
        }
    }

    int dfs(int u, int flow)
    {
        if (u == T)
            return flow;
        int delta = 0, v, temp;
        for (int &i = cur[u]; i; i = e[i].nex)
        {
            v = e[i].to;
            if (d[v] + 1 != d[u])
                continue;
            temp = dfs(v, min(flow - delta, e[i].cap));
            if (temp)
            {
                e[i].cap -= temp;
                e[i ^ 1].cap += temp;
                delta += temp;
            }
            if (flow == delta)
                return flow;
        }
        if (!(--gap[d[u]]))
            d[S] = T + 1;
        ++gap[++d[u]];
        cur[u] = head[u];
        return delta;
    }

    int get_maxFlow()
    {
        bfs();

        int maxFlow = 0;

        while (d[S] <= T)
            maxFlow += dfs(S, INF);

        return maxFlow;
    }
};
```

```
ISAP ways;
signed main()
{
	ways.buildGraph();
	return 0;
}
```

整理人：网络 18-3 高云泽

4.3 费用流

4.3.1 SPFA 费用流

```
//洛谷 P3381
typedef long long ll;
using namespace std;
const int INF = 1e18 + 7;
const int N = 2e4 + 7;
const int M = 2e5 + 7;
const int base = 100;
ll dis[N];
struct edge
{
	int to, nex;
	ll cap, cost;
	edge(int to = 0, int nex = 0, ll cap = 0, ll cost = 0) : to(to), nex(nex), cap(cap), cost(cost){}
};
struct cmp
{
	bool operator()(int a, int b)
	{
		return dis[a] > dis[b];
	}
};
class MCMF	// MinCostMaxFlow
{
public:
	int S, T, n, m, cnt;
	int head[N], Inq[N], pre[N];
	ll flow[N];
	edge e[M];

	void addedge(int u, int v, ll cap, ll cost)
	{
```

```
        e[++cnt] = edge(v, head[u], cap, cost);
        head[u] = cnt;
    }

    // 建图
    void buildGraph(int _n, int _m, int _S, int _T)
    {
        // 初始化部分
        n = _n; m = _m;
        S = _S; T = _T;
        mem(head);
        cnt = 1;

        // 构图部分
        for(int i = 1; i <= m; i++)
        {
            int u, v;
            ll cap, cost;
            read(u); read(v);
            readl(cap); readl(cost);

            addedge(u, v, cap, cost);
            addedge(v, u, 0ll, -cost);
        }
    }

    // 找增广路
    bool SPFA()
    {
        priority_queue<int, vector<int>, cmp> q;
        for(int i = 0; i <= n; i++)
        {
            dis[i] = INF;
            Inq[i] = 0;
        }
        dis[S] = 0; Inq[S] = 1; flow[S] = INF;
//          q.push(make_pair(-dis[S], S));
        q.push(S);

        int u, v; ll cost;
//          pair<ll, int> tmp;
        while(q.size())
        {
            u = q.top(); q.pop(); Inq[u] = 0;
//              u = tmp.second; cost = -tmp.first;
            for(int i = head[u]; i; i = e[i].nex)
```

```
            {
                v = e[i].to;
//                cout << "u = " << u << " , v = " << v << "\n";
                if(!e[i].cap) continue;
                cost = dis[u] + e[i].cost;
                if(dis[v] > cost)
                {
                    dis[v] = cost;
                    flow[v] = min(flow[u], e[i].cap);
                    pre[v] = i;
                    if(!inq[v])
                    {
                        q.push(v);
                        inq[v] = 1;
                    }
                }
            }
        }
        return dis[T] != INF;
    }

    void    update()
    {
        int u = T;
        while(u != S)
        {
            int i = pre[u];
            e[i].cap -= flow[T];
            e[i^1].cap += flow[T];
            u = e[i^1].to;
        }
    }

    void get_MCMF()
    {
        ll maxFlow = 0;
        ll minCost = 0ll;

        while(SPFA())
        {
            update();
            maxFlow += flow[T];
            minCost += flow[T] * dis[T];
        }
        printf("%lld %lld\n", maxFlow, minCost);
    }
```

```
};

MCMF ways;
signed main()
{
	int n, m, S, T;

	while(~scanf("%d%d%d%d", &n, &m, &S, &T))
	{
		ways.buildGraph(n, m, S, T);
		ways.get_MCMF();

//		printf("%lld %lld\n", tmp.second, tmp.first);
	}

	return 0;
}
```

例题： 网络流 24 题 餐巾计划 LOJ 6008

题意：一个餐厅在相继的 N 天里,每天需用的餐巾数不尽相同。假设第 i 天需要 ri 块餐巾(i=1,2,...,N)。餐厅可以购买新的餐巾,每块餐巾的费用为 p 分;或者把旧餐巾送到快洗部,洗一块需 m 天,其费用为 f 分;或者送到慢洗部,洗一块需 n(n>m),其费用为 s 分(s<f)。每天结束时,餐厅必须决定将多少块脏的餐巾送到快洗部,多少块餐巾送到慢洗部,以及多少块保存起来延期送洗。但是每天洗好的餐巾和购买的新餐巾数之和,要满足当天的需求量。试设计一个算法为餐厅合理地安排好 N 天中餐巾使用计划,使总的花费最小。编程找出一个最佳餐巾使用计划。

分析：我们不妨先让每天开始时得到的 r[i]条干净的餐巾（左边一列的节点）流向 t，然后再在每天结束时从 s 补回 r[i]条脏的餐巾（从 s 向右边一列的节点连 r[i],0 的边），得到新图的链接方式：

1. s -> i (r,p) 每天早晨可以买最多 r 条新餐巾 一条 p 分
2. s -> i' (r,0) 每天用剩下 r 条脏餐巾 没有代价
3. i -> t (r,0) 每天要用 r 条干净餐巾 没有代价
4. i' -> i+m (inf,f)脏毛巾送到快洗店 洗干净送回来是第 i+m 天每条花费代价 f 分
5. i' ->i+n (inf,s)脏毛巾送到慢洗店 洗干净送回来是第 i+n 天 每条花费代价 s 分
6. i' -> (i+1)' (inf,s) 每条脏毛巾留到第二天再处理 没有代价

最后跑一次最小费用流

```
#pragma GCC optimize(2)
#include<bits/stdc++.h>
#define read(a) scanf("%d", &a)
#define readl(a) scanf("%lld", &a)
#define reads(a) scanf("%s", a)
#define readc(a) scanf("%c", &a)
#define pb push_back
#define mem(a) memset(a, 0, sizeof(a))
#define Buff ios::sync_with_stdio(false)
```

```
typedef long long ll;
using namespace std;
const ll INF = 1e16 + 7;
const int N = 2e5 + 7;
const int M = 1e6 + 7;
ll dis[N], flow[N];
int head[N], cnt, Inq[N], pre[N];
int S, T, n, d1, d2;
ll p, f, s;
struct edge
{
	int to, nex;
	ll w, cost;
	edge(int _to = 0, ll _w = 0, ll _cost = 0, int _nex = 0)
	{
		to = _to; nex = _nex;
		w = _w; cost = _cost;
	}
}e[M];

void addedge(int u, int v, ll w, ll cost)
{
	e[++cnt] = edge(v, w, cost, head[u]); head[u] = cnt;
	e[++cnt] = edge(u, 0ll, -cost, head[v]); head[v] = cnt;
}

queue<int> q;
bool SPAF()
{
	while(q.size()) q.pop();
	for(int i = 0; i < N; i++)
	{
		dis[i] = INF;
		Inq[i] = 0;
	}
	q.push(S); dis[S] = 0; Inq[S] = 1; flow[S] = INF;
	int u, v;
	ll cost;
	while(q.size())
	{
		u = q.front(); q.pop(); Inq[u] = 0;
		for(int i = head[u]; i; i = e[i].nex)
		{
			if(!e[i].w) continue;
			v = e[i].to; cost = dis[u] + e[i].cost;
			if(dis[v] > cost)
```

```
                {
                    dis[v] = cost;
                    flow[v] = min(flow[u], e[i].w);
                    pre[v] = i;
                    if(!Inq[v])
                    {
                        q.push(v);
                        Inq[v] = 1;
                    }
                }
            }
        }
        return dis[T] != INF;
}

ll maxFlow, minCost;
void update()
{
        int u = T, i;
        while(u != S)
        {
            i = pre[u];
            e[i].w -= flow[T];
            e[i ^ 1].w += flow[T];
            u = e[i ^ 1].to;
        }
        maxFlow += flow[T];
        minCost += flow[T] * dis[T];
}

void EK()
{
        maxFlow = minCost = 0ll;
        while(SPAF())
            update();
        printf("%lld\n", minCost);
}
void buildGraph()
{
        read(n); readl(p);
        read(d1); readl(f);
        read(d2); readl(s);
        mem(head); cnt = 1;
        S = 0; T = n << 1 | 1;

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

```
        {
            ll r; readl(r);
            addedge(S, i, r, p);
            addedge(S, i + n, r, 0ll);
            addedge(i, T, r, 0ll);
            if(i + 1 <= n)
                addedge(i + n, i + 1 + n, INF, 0ll);
            if(i + d1 <= n)
                addedge(i + n, i + d1, INF, f);
            if(i + d2 <= n)
                addedge(i + n, i + d2, INF, s);
        }
}
signed main()
{
    buildGraph();
    EK();
    return 0;
}
```

整理人：网络 18-3 高云泽

第 5 章 字符串

5.1 字典树

```
/* hdu1251
题意：给出一些模式串，然后有一些提问，对于每个提问，给出以该字符串为前缀的模式串的数量.
*/
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1e6+7;        //请开到模式串长度*模式串数量
const int ch_size = 26;        //请开成字符集大小
class Trie{
    public:
        int trie[maxn][ch_size];
        int vis[maxn],tot;
    void init(){
        memset(trie,0,sizeof trie);
        memset(vis,0,sizeof vis);
        tot=0;
    }
    void insert(char *str){
        int len = strlen(str);
        int pos = 0;
        for(int i=0;i<len;i++){
            int c = str[i]-'a';
            if(!trie[pos][c])trie[pos][c] = ++tot;
            pos = trie[pos][c];
            vis[pos]++;
        }
        //vis[pos]++;
    }
    int query(char *str){
        int len = strlen(str);
        int pos = 0;
        for(int i=0;i<len;i++){
            int c = str[i]-'a';
            if(!trie[pos][c])return 0;
            pos = trie[pos][c];
        }
        return vis[pos];
    }
};
```

```
Trie tr;
int main(){
	char str[maxn];
	while(cin.getline(str,maxn)&&strlen(str)){
		tr.insert(str);
	}
	while(~scanf("%s",str)){
		cout<<tr.query(str)<<endl;
	}
}
```

整理人：计 18-5 王东琛

5.2 AC 自动机

```
/*   hdu2222
题意：每组数据有 n 个模式串，询问在文本串中出现了多少模式串
*/
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1e6;            // 请开成 模式串长度*模式串数量
const int ch_size = 26;         // 请开成字符集大小
class Ac_automaton{
	public:
		int trie[maxn][ch_size];
		int vis[maxn],fail[maxn];
		int tot;
	void init(){
		memset(vis,0,sizeof vis);
		memset(trie,0,sizeof trie);
		tot = 0;
	}
	void insert(char *str){
		int len = strlen(str);
		int pos = 0;
		for(int i=0;i<len;i++){
			int c = str[i]-'a';
			if(!trie[pos][c])trie[pos][c] = ++tot;
			pos = trie[pos][c];
		}
		vis[pos]++;
	}
	void build(){
		queue<int> q;
		for(int i=0;i<ch_size;i++){
```

```
            if(trie[0][i]){
                fail[trie[0][i]]=0;
                q.push(trie[0][i]);
            }
        }

        while(!q.empty()){
            int pos = q.front();
            q.pop();
            for(int i=0;i<ch_size;i++){
                if(trie[pos][i]){
                    fail[trie[pos][i]] = trie[fail[pos]][i];
                    q.push(trie[pos][i]);
                }
                else{
                    trie[pos][i] = trie[fail[pos]][i];
                }
            }
        }
    }
    int query(char *str){
        int len = strlen(str);
        int pos = 0,ans = 0;
        for(int i=0;i<len;i++){
            int c = str[i]-'a';
            pos = trie[pos][c];
            for(int j=pos;j&&vis[j]!=-1;j=fail[j]){
                ans+=vis[j];
                vis[j]=-1;
            }
        }
        return ans;
    }
};

Ac_automaton ac;
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        ac.init();
        char str[maxn];
        int n;
        scanf("%d",&n);
        for(int i=0;i<n;i++){
            scanf("%s",str);
```

```
            ac.insert(str);
        }
        ac.build();
        scanf("%s",str);
        printf("%d\n",ac.query(str));
    }
}
```

整理人：计 18-5 王东琛

第 6 章 计算几何

6.1 点积叉积的运用

//poj2318　　题意: 在一个矩形盒子内用 n 块隔板将矩形分成若个个区域(隔板不相交), 给定 m 个点，问（n + 1）个区域内分别有多少个点；
思路：对于每一个点二分求出它所在的区域

```
#include<iostream>
#include<cmath>
#include<cstdio>
#include<vector>
#include<algorithm>
#include<cstring>
#define mem(a) memset(a, 0, sizeof a)
using namespace std;
const int N = 1e5+7;
const double eps = 1e-6;//控制精度
int dcmp(double x)
{
    if(fabs(x) < eps)   return 0;
    else                return x < 0 ? -1 : 1;
}
struct Point
{
    double x;
    double y;
    Point(double x=0, double y=0):x(x), y(y) {}
};
typedef Point Vector;
double operator * (const Vector & v, const Vector & w)  {return v.x * w.x + v.y * w.y;}//点积
double operator ^ (const Vector & v, const Vector & w)  {return v.x * w.y - v.y * w.x;}//叉积
double sqr(Vector v)                                    {return v * v;}//向量模的平方
double length(Vector v)                                 {return sqrt(v * v);}向量的模长
Vector operator - (const Vector & v, const Vector & w)  {return Vector(v.x - w.x, v.y - w.y);}//向量减法
Vector operator + (const Vector & v, const Vector & w)  {return Vector(v.x + w.x, v.y + w.y);}//向量加法
Vector operator * (const Vector & v, double k)          {return Vector(v.x * k, v.y * k);}//向量的数乘
```

```
Vector rotate(Vector v, double rad)                    {return
Vector(v.x * cos(rad) - v.y * sin(rad), v.x * sin(rad) + v.y *
cos(rad));}// 将向量逆时针旋转 rad
Vector normal(Vector v)                                {return
Vector(-v.y/length(v), v.x/length(v));}//向量的单位法向量
bool OnSeg(Point O, Point A, Point B)                  {return !dcmp((A
- O) ^ (B - O)) && dcmp((A - O) * (B - O)) <= 0;}//判定点是否在线段上
bool operator == (const Vector & A, const Vector & B)
{return !dcmp(A.x-B.x) && !dcmp(A.y-B.y);}//相量相等
bool operator < (const Vector & v, const Vector & w)    {return v.x ==
w.x ? v.y < w.y : v.x < w.x;}
Point segs[N][2];
int n, m, res[N];
int find(Point P)
{
    int l = 1, r = n + 1;
    while(l < r)
    {
        int mid = l + r >> 1;
        Vector v = segs[mid][0] - segs[mid][1], w = P - segs[mid][1];
        if((v ^ w) > 0) r = mid;
        else            l = mid + 1;
    }
    return l - 1;
}
int main()
{
    bool flag = 0;
    while(scanf("%d", &n), n)
    {
        int x1, x2, y1, y2;
        scanf("%d%d%d%d%d", &m, &x1, &y1, &x2, &y2);
        segs[0][0] = Point(x1, y1), segs[0][1] = Point(x1, y2);
        segs[n+1][0] = Point(x2, y1), segs[n+1][1] = Point(x2, y2);
        for(int i = 1; i <= n; i ++)
        {
            int a, b;
            scanf("%d%d", &a, &b);
            segs[i][0] = Point(a, y1);
            segs[i][1] = Point(b, y2);
        }
        mem(res);
        for(int i = 0; i < m; i ++)
        {
            int a, b;
            scanf("%d%d", &a, &b);
```

```
            res[find(Point(a, b))] ++;
        }
        if(!flag)   flag = 1;
        else        puts("");
        for(int i = 0; i <= n; i ++)
            printf("%d: %d\n", i, res[i]);
    }
    return 0;
}
```

整理人：网络 18－3 冯紫君

6.2 凸包的求取

//poj 1912;　题意：给定平面上 n 个点，之后给定若干直线，询问所有点时候在直线的一侧；
//思路：首先求取凸包，之后将凸包上的每一条边的倾斜角求出，对倾斜角二分，找出倾斜角最接近给定直线倾斜角的边所对应的点，将直线旋转 180° 后在求出对应点，判定所求两点是否在直线两侧

```
struct Line
{
    Point p;
    Vector v;
    double ang;
    Line()   {}
    Line(Point p, Vector v) : p(p), v(v) {ang = atan2(v.y, v.x);}
    Point point(double t)                {return p + v * t;}//求取直线上的某一个点
    bool operator < (Line & l)          {return ang < l.ang;}//直线默认以倾斜角排序
    int Onleft(Point P)                  {return dcmp(v ^ (P - p));}//判断一个点是否在直线的左侧
    Point operator & (Line & l)//求取两直线的交点
    {
        Vector u = p - l.p;
        double t = (l.v ^ u) / (v ^ l.v);
        return p + v * t;
    }
};
Point p[N], sta[N];
int n, top;
double af[N];//用于记录凸包每一条边的倾斜角
Line l;
bool cmp(Point a, Point b) //极角排序
{
    Vector v = a - p[0], w = b - p[0];//以某一点为基准求出其他所有点相对于它的极角
```

```
    if(dcmp(v ^ w) < 0)                                    return false;
    else if(!dcmp(v ^ w) && dcmp(sqr(v) - sqr(w)) > 0)     return false;//极角相同以向量模长排序
    return true;
}
void graham()//凸包
{
    sort(p + 1, p + n, cmp);
    n = unique(p, p + n) - p;
    int t = n - 1;
    while(t && !((p[n-1] - p[0]) ^ (p[t] - p[0]))) t --;
    if(!t)//当凸包退化为一个点或者一条直线时的情况
    {
        sta[0] = p[0];    top = 1;
        if(n > 1)
            sta[1] = p[n - 1], top = 2;
        return ;
    }
    reverse(p + t + 1, p + n); //为求取所有在凸包边界上的点
    sta[top ++] = p[0]; sta[top ++] = p[1]; p[n ++] = p[0];
    for(int i = 2; i < n; i ++)
    {
        while(top > 1 && ((sta[top-1] - sta[top-2]) ^ (p[i] - sta[top-2])) <= 0)      top --;确定每一个角都不为优角
        sta[top ++] = p[i];
    }
    top --;
    for(int i = 0; i < top; i ++)
        af[i] = atan2(sta[i+1].y - sta[i].y, sta[i+1].x - sta[i].x);//求取所有边的倾斜角
}
int main()
{

    scanf("%d", &n);
    for(int i = 0; i < n; i ++)
    {
        scanf("%lf %lf", &p[i].x, &p[i].y);
        if(p[i] < p[0]) swap(p[i], p[0]);
    }
    if(n > 1)     graham();
    double a, b, c, d;
    while(~scanf("%lf %lf %lf %lf", &a, &b, &c, &d))
    {
        if(n < 2)
        {
            puts("GOOD");
```

```
                continue;
            }
            l = Line(Point(a, b), Vector(c - a, d - b));
            double l1 = atan2(l.v.y, l.v.x), l2 = atan2(-l.v.y, -l.v.x);
            int p1 = lower_bound(af, af + top, l1) - af;
            int p2 = lower_bound(af, af + top, l2) - af;
            if(l.Onleft(sta[p1]) * l.Onleft(sta[p2]) < 0)        puts("BAD");
            else                                                 puts("GOOD");
        }
}
```

整理人：网络 18 – 3 冯紫君

6.3 半平面交

//poj3525； 题意给出一个凸多边形，求取在该凸多边形内最大的圆的半径为多少
思路：每次将凸多边的每一条边向内部移动一定距离，直到凸多边面积恰好为零时该距离边是所求圆的半径

```
int HAI(Line *L, int n)
{
    sort(L, L+n); //直线向按照倾斜角排序
    int first, last;
    Point *p = new Point[n];//利用双端队列存取半平面交的交点，以及半平面
    Line *l = new Line[n];
    l[first = last = 0] = L[0];
    for(int i = 1; i < n; i++)
    {
        while(first < last && !L[i].Onleft(p[last-1]))   last--;
        while(first < last && !L[i].Onleft(p[first]))    first++;
        l[++last] = L[i];
        if(!dcmp(l[last].v^l[last-1].v))      //若存在两个倾斜角相等的半平面，选择距离相交区域更近的一个
        {
            last--;
            if(l[last].Onleft(L[i].p))   l[last] = L[i];
        }
        if(first < last)       p[last-1] = l[last-1]&l[last];
    }
    while(first < last && !l[first].Onleft(p[last-1]))   last--;
    if(last - first <= 1)     return 0;
    p[last] =   l[last]&l[first];
    return last - first + 1;
}
```

```
int n;
Point p[N];
Vector dir[N], nor[N];
Line line[N];

int main()
{
    while(scanf("%d", &n), n)
    {
        for(int i = 0; i < n; i ++)
            scanf("%lf %lf", &p[i].x, &p[i].y);
        p[n] = p[0];
        for(int i = 0; i < n; i ++)
            dir[i] = p[i+1] - p[i], nor[i] = normal(dir[i]);
        double l = 0.0, r = INF;
        while(dcmp(r - l) > 0)
        {
            double mid = (l + r) / 2;
            for(int i = 0; i < n; i ++) line[i] = Line(p[i] + nor[i] * mid, dir[i]);
            if(HAI(line, n))    l = mid;
            else                r = mid;
        }
        printf("%.6f\n", l);
    }
    return 0;
}
```

整理人：网络 18 – 3 冯紫君

6.4 旋转卡壳

```
//Aizu CGL_4_B;    题意：给定平面 n 个点，求取距离最大的点对；
double diame()
{
    int i, j, k;
    double res, t;
    graham();
    sta[top] = sta[0], res = 0.0, j = 1;
    for(i = 0; i < top; i ++)
    {
        while(dcmp(fabs((sta[i] - sta[j]) ^ (sta[i+1] - sta[j])) - fabs((sta[i] - sta[j+1]) ^ (sta[i+1] -
sta[j+1]))) < 0) //寻找距离固定直线距离最大的点
            j = (j + 1) % top;
        t = sqr(sta[i] - sta[j]);
```

```
            if(dcmp(t - res) > 0)      res = t;
        }
        return sqrt(res);
}
```
整理人：网络 18 – 3 冯紫君

附录 A　时间/空间复杂度

一、搜索

n 个点/状态，m 个边	深度优先搜索	广度优先搜索	双向搜索	记忆化搜索	极大极小搜索	启发式搜索
时间复杂度	O(n)	O(n)	O(n^1/2~n/2)	O(n)	<<O(n)	<<O(n)
空间复杂度	O(n)	O(n)	O(n^1/2~n/2)	O(n)	<<O(n)	<<O(n)
备注					不稳定	与启发函数有关

二、动态规划

	状压 dp	树形 dp	区间 dp	数位 dp	概率 dp
时间复杂度	O(n^2*2^n)	O(n)	O(n^3)	O(size[dp])	O(size[dp])
空间复杂度	O(n*2^n)	O(n)	O(n^2)	O(size[dp])	O(size[dp])
备注				都是数组大小	都是数组大小

三、数据结构

n 个点/状态	一维树状数组	二维树状数组	ST 表	RMQ	线段树
时间复杂度	O(log(n))	O(log(n)*log(m))	O(1)	O(logn)	O(logn)
空间复杂度	O(n)	O(n*m)	O(nlogn)	O(n)	O(n)
备注		n*m 的矩阵	预处理 O(nlogn)	预处理 O(nlogn)	空间需要开 4n，预处理为空间大小

	树链剖分-点剖	树链剖分-边剖	分块算法	莫队算法
时间复杂度	O(n*logn*logn)	O(n*logn*logn)	O(n * sqrt(n))	O(n * sqrt(n))
空间复杂度	O(10*n)	O(10*n)	O(n)	O(n)
备注	由于数组过多给了一个 10 的常数	由于数组过多给了一个 10 的常数		

	点分治	caldis()	求重心
时间复杂度	O(nlogn)	O(n)	O(nlogn)
空间复杂度	O(nlogn)	O(n)	O(nlogn)
备注		点分治内部：求基本路径	

四、图论

	匈牙利算法（邻接矩阵）	匈牙利算法（邻接表）	KM 算法
时间复杂度	O(n³)	O(mn)	O(n³)
空间复杂度	O(n²)	O(m+n)	O(n²)
备注			

	网络流-EK	网络流-Dinic	网络流-ISAP	费用流-SPFA
时间复杂度	O(n*m*m)	O(n*n*m)	O(n*n*m)	$O(\|V\|*\|E\|*log_2\|V\|)$
空间复杂度	O(n+m)	O(n+m)	O(n+m)	O(n+m)
备注	N 为节点数，M 为边数	最大流跑最大匹配 时间复杂度为 O(n*sqrt(m))		所有的时间复杂度都是最大上界，平常都达不到

五、字符串

	字典树	AC 自动机
时间复杂度	O(n)	O(m*n)
空间复杂度	O(n*k)	O(n*k)
备注	K 为模式串数量	M 为文本串长度

六、计算几何

	Graham	半平面交
时间复杂度	O(nlogn)	O(nlogn)
空间复杂度	O(n)	O(n)
备注		瓶颈在求取凸包上，旋转卡壳的求取是线性的

附录 B　习题

一、搜索

1.1 深度优先搜素：hrbust 1743
1.2 广度优先搜素：hdu 2612
1.3 双向搜索：
1.4 记忆化搜索：hdoj1078
1.5 极大极小搜索：poj1568
1.6 启发式搜索：poj1077

二、动态规划

2.1 状压 Dp：hdu 5418
2.2 树形 Dp：P1352
2.3 区间 Dp：noi1995
2.4 数位 Dp：https://loj.ac/problems/tag/104
2.5 概率&期望 Dp：https://vjudge.net/contest/76505

三、数据结构

3.1 一维树状数组：poj3468
3.2 二维树状数组：poj2155
3.3 ST 表：洛谷 p3865
3.4 Rmq：poj3264
3.5 线段树：Poj3468, HDU3911, HDU1542
3.6.1 树链剖分-点剖：洛谷 P3384，HYSBZ 4196，HYSBZ 3531，HYSBZ 2243
3.6.2 树链剖分-边剖：洛谷 P4315，HYSBZ 2157
3.7 分块算法：POJ　3468
3.8 莫队算法：ZOJ　2038
3.9 点分治：POJ 1655，POJ 2114，POJ 1741，HDU 4812，HYSBZ 2152，HDU 5977，POJ 1987

四、图论

4.1.1 匈牙利算法：POJ1274、POJ1469、POJ3041、POJ1325、POJ1466、POJ2226、HDU1281

4.1.2 KM 算法：HDU2255
4.2 最大流：洛谷 P3376、LOJ 6015
4.3 费用流：洛谷 P3381、LOJ 6008

五、字符串

5.1 字典树：hdu1251
5.2 AC 自动机：hdu2222

六、计算几何

6.1 点积叉积的应用：poj2318
6.2 Graham：poj 1912
6.3 半平面交：poj3525
6.4 旋转卡壳：Aizu CGL_4_B