ACM-ICPC

算法模板

2018.11

孙永飞

目录

1. 基础

1.1 二分查找

在有序表中高效查找元素的常用方法是二分查找，所谓二分即是折半，遵循分治的思想，每次将元序列划分成数量尽量相等的两个子序列，然后递归查找，最终定位到目标元素。

```
int binsearch(int array[], int low, int high, int target)
{
    if (low > high)
        return -1;
    int mid = (low+high)/2;
    if (array[mid] > target)
        return binsearch(array, low, mid-1, target);
    if (array[mid] < target)
        return binsearch(array, mid+1, high, target);
    return mid;
}
```

1.2 BFS

广度优先搜索的英文简写是 BFS(Breadth First Search)，属于图论中搜索算法中的一种，它所遵循的搜索策略是尽可能“广”地搜索图。

```
#include <bits/stdc++.h>
using namespace std;
struct node
{
    int x, y, step;
}s, e;
int dir[4][2] = {{1,0}, {-1,0}, {0,1}, {0,-1}}, m, n;
bool vis[100][100];
char mp[100][100];
bool check(node a)
{
    if(a.x<0 || a.y<0 || a.x>=n || a.y>=m || a.step>m*n || vis[a.x][a.y]
|| mp[a.x][a.y]=='#')
        return 0;
    return 1;
}
int bfs()
{
    queue<node> q;
    node now, next;
    now = s;
    q.push(now);
    vis[now.x][now.y] = 1;
    while(!q.empty())
    {
        now = q.front();
        q.pop();
        if(now.x==e.x && now.y==e.y)
            return now.step;
        for(int i = 0; i < 4; i++)
        {
```

```
            next.x = now.x + dir[i][0];
            next.y = now.y + dir[i][1];
            next.step = now.step+1;
            if(check(next))
            {
                q.push(next);
                vis[next.x][next.y] = 1;
            }
        }
    }
    return -1;
}
int main()
{
    int t;
    scanf("%d", &t);
    while(t--)
    {
        memset(vis, 0, sizeof(vis));
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i++)
            scanf("%s", mp[i]);
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
            {
                if(mp[i][j] == 'S')
                {
                    s.x = i;
                    s.y = j;
                    s.step = 0;
                }
                if(mp[i][j] == 'E')
                {
                    e.x = i;
                    e.y = j;
                    e.step = 0;
                }
            }
        printf("%d\n", bfs());
    }
    return 0;
}
```

1.3 DFS

深度优先搜索的英文简写是 DFS(Depth First Search)，属于图论中搜索算法中的一种，它所遵循的搜索策略是尽可能"深"地搜索图。

```
#include <bits/stdc++.h>
using namespace std;

char map_[105][105];
int m, n;
bool vis[105][105];

void dfs(int r, int c, int id)
{
```

```
    if(r >= m || r < 0 || c < 0 || c >= n)
        return;
    if(vis[r][c] || map_[r][c] != '@')
        return;
    vis[r][c] = 1;
    for(int dr = -1; dr <= 1; dr++)
        for(int dc = -1; dc <= 1; dc++)
            if(dr != 0 || dc != 0)
                dfs(r+dr, c+dc, id);
}

int main()
{
    while(scanf("%d%d", &m, &n) != EOF && m && n)
    {
        memset(vis, 0, sizeof(vis));
        for(int i = 0; i < m; i++)
            scanf("%s", map_[i]);
        int cnt = 0;
        for(int i = 0;i < m; i++)
            for(int j = 0; j < n; j++)
                if(vis[i][j] == 0 && map_[i][j] == '@')
                    dfs(i, j, ++cnt);
        printf("%d\n", cnt);
    }
    return 0;
}
```

2. 数据结构

2.1 并查集

并查集一般用于对动态连通性的判断，主要应用于判断两个元素是否在同一个集合，两个点是否连通，变量名等同性以及间接好友的判断。同时并查集经常作为其他模板的一部分实现某些功能。并查集常用于的题型为判断某两个元素是否属于同一个集合，判断图是否连通或是否有环，或配合其他算法如最小生成树 Kruskal，与 DP 共同使用等。

```
#include <bits/stdc++.h>
using namespace std;
int pre[1050];
bool t[1050]; //to mark whether t[i] is a root node
int find(int x)
{
    int r = x;
    while(pre[r] != r)
        r = pre[r];
    int i = x;
    while(i != r) //path conpression
    {
        int p = pre[i];
        pre[i] = r;
        i = p;
    }
    return r;
}
void mix(int x, int y)
```

```
{
    int fx = find(x), fy = find(y);
    if (fx != fy)
        pre[fy] = fx;
}
int main()
{
    int N, M, a, b, i, j, ans;
    while (scanf("%d%d", &N, &M) && N)
    {
        for (i = 1; i <= N; i++)
            pre[i] = i;
        for (i = 1; i <= M; i++)
        {
            scanf("%d%d", &a, &b);
            mix(a, b);
        }
        memset(t, 0, sizeof(t));
        for (i = 1; i <= N; i++)
            t[find(i)] = 1;
        for (ans = 0, i = 1; i <= N; i++)
            if (t[i])
                ans++;
        printf("%d\n", ans-1);
    }
    return 0;
}
```

2.2 线段树

线段树是一种二叉搜索树，与区间树相似，它将一个区间划分成一些单元区间，每个单元区间对应线段树中的一个叶结点。

对于线段树中的每一个非叶子节点[a,b]，它的左儿子表示的区间为[a,(a+b)/2]，右儿子表示的区间为[(a+b)/2+1,b]。因此线段树是平衡二叉树，最后的子节点数目为 N，即整个线段区间的长度。使用线段树可以快速的查找某一个节点在若干条线段中出现的次数，时间复杂度为 O(logN)。而未优化的空间复杂度为 2N，因此有时需要离散化让空间压缩。

```
#include <bits/stdc++.h>
#define MAXN 200009
using namespace std;

struct Node
{
    int l, r, max;
}tree[MAXN*4];
int a[MAXN];

void build(int root, int l, int r)
{
    int mid;
    tree[root].l = l;
    tree[root].r = r;
    if (l == r)
    {
```

```
        tree[root].max = a[l];
        return;
    }
    mid = (l + r) / 2;
    build(root * 2, l, mid);
    build(root * 2 + 1, mid + 1, r);
    tree[root].max = max(tree[root * 2].max, tree[root * 2 + 1].max);
}

int query(int r, int x, int y)
{
    int mid;
    if (y < tree[r].l || tree[r].r < x)
        return 0;
    if (x <= tree[r].l && tree[r].r <= y)
        return tree[r].max;
    return max(query(r * 2, x, y), query(r * 2 + 1, x, y));
}

void update(int r, int p)
{
    if (p < tree[r].l || tree[r].r < p)
        return;
    if (p == tree[r].l && p == tree[r].r)
    {
        tree[r].max = a[p];
        return;
    }
    update(2 * r, p);
    update(2 * r + 1, p);
    tree[r].max = max(tree[r * 2].max, tree[r * 2 + 1].max);
}

int main()
{
    // freopen("input.txt", "r", stdin);
    int n, m;
    while (scanf("%d%d", &n, &m) != EOF)
    {
        int x, y;
        char c;
        for (int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        build(1, 1, n);
        while (m--)
        {
            getchar();
            scanf("%c%d%d", &c, &x, &y);
            if (c == 'Q')
                printf("%d\n", query(1, x, y));
            else
            {
                a[x] = y;
                update(1, x);
            }
        }
    }
    return 0;
```

```
}
```

2.3 字典树

字典树，又称单词查找树，Trie 树，是一种树形结构，是一种哈希树的变种。典型应用是用于统计，排序和保存大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来节约存储空间，最大限度地减少无谓的字符串比较，查询效率比哈希表高。

字典树与字典很相似,当你要查一个单词是不是在字典树中,首先看单词的第一个字母是不是在字典的第一层,如果不在,说明字典树里没有该单词,如果在就在该字母的孩子节点里找是不是有单词的第二个字母,没有说明没有该单词,有的话用同样的方法继续查找.字典树不仅可以用来储存字母,也可以储存数字等其它数据。

```
#include <bits/stdc++.h>
using namespace std;

struct node
{
    int cnt;
    node *childs[26];
    node()
    {
        cnt = 0;
        for(int i = 0; i < 26; i++)
            childs[i] = NULL;
    }
};

node *root = new node;
node *current;

void insert(char *str)
{
    current = root;
    int len = strlen(str);
    for(int i = 0; i < len; i++)
    {
        int m = str[i] - 'a';
        if(current->childs[m] == NULL)
            current->childs[m] = new node;
        current = current->childs[m];
        (current->cnt)++;
    }
}

int search(char *str)
{
    current = root;
    int len = strlen(str);
    for(int i = 0; i < len; i++)
    {
        int m = str[i] - 'a';
        if(current->childs[m] == NULL)
            return 0;
```

```
        current = current->childs[m];
    }
    return current->cnt;
}

int main()
{
    // freopen("input.txt", "r", stdin);
    char str[20];
    while(gets(str) && strlen(str))
        insert(str);
    while(scanf("%s", str) != EOF)
        printf("%d\n", search(str));

    return 0;
}
```

2.4 __int128，64 位编译器大数

64 位编译器可以使用__int128 类型作为大数容器。

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5+9;
const int INF = 0x7fffffff;

unsigned __int128 read(char* x)
{
    int len = strlen(x);
    unsigned __int128 r = 0;
    for(int i = 0; i < len; i++)
    {
        r *= 10;
        r += x[i]-'0';
    }
    return r;
}

void print(unsigned __int128 x)
{
    if(!x)
        return;
    print(x/10);
    putchar(x % 10 + '0');
}

int main()
{
    // freopen("input.txt", "r", stdin);
    __int128 table[60];
    table[0] = read("4");
    table[1] = read("14");
    for(int i = 2; i <= 52; i++)
        table[i] = 4*table[i-1] - table[i-2];
    int t;
    scanf("%d\n", &t);
    while(t--)
```

```
    {
        char in[50];
        scanf("%s", in);
        __int128 a = read(in);
        __int128 ans = *lower_bound(table, table+52, a);
        print(ans);
        puts("");
    }
    return 0;
}
```

2.5 ST 表

ST 表是一种动态规划的方法，放在数据结构分类里是因为其解决的 RMQ（Range Minimum/Maximum Query）即区间最值问题，先前的线段树可以解决。**然而！ST 表的查询时间复杂度为 O(1)！**以最小值为例。a 为所寻找的数组，用一个二维数组 f(i,j)记录区间[i, i+2^j-1]区间中的最小值。其中 f[i,0] = a[i];所以，对于任意 的一组(i,j)，f(i,j) = min{f(i,j-1),f(i+2^(j-1),j-1)}来使用动态规划计算出来。

```
#include <cstdio>
#include <algorithm>
using namespace std;

const int MAXN = 50009;
const int MAXM = 20;
int dp_min[MAXN][MAXM];
int dp_max[MAXN][MAXM];
int a[MAXN];
int n;

void rmq_init()
{
    for(int i = 1; i <= n; i++)
    {
        dp_min[i][0] = a[i];
        dp_max[i][0] = a[i];
    }
    for(int j = 1; (1<<j) <= n; j++)
        for(int i = 1; i+j-1 <= n; i++)
        {
            dp_min[i][j] = min(dp_min[i][j-1], dp_min[i+(1<<(j-1))][j-1]);
            dp_max[i][j] = max(dp_max[i][j-1], dp_max[i+(1<<(j-1))][j-1]);
        }
}

int rmq_min(int l, int r)
{
    int k = 0;
    while((1<<(k+1)) <= r-l+1)
        k++;
    return min(dp_min[l][k], dp_min[r-(1<<k)+1][k]);
}

int rmq_max(int l, int r)
{
```

```
    int k = 0;
    while((1<<(k+1)) <= r-l+1)
        k++;
    return max(dp_max[l][k], dp_max[r-(1<<k)+1][k]);
}

int main()
{
    // freopen("input.txt", "r", stdin);
    int q;
    scanf("%d%d", &n, &q);
    for(int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    rmq_init();
    while(q--)
    {
        int l, r;
        scanf("%d%d", &l, &r);
        printf("%d\n", rmq_max(l, r) - rmq_min(l, r));
    }

    return 0;
}
```

3. 动态规划

3.1 背包模型

背包模型构成了动态规划中的一大类问题。比赛中的很多问题都是通过背包模型变形得到的。所以，背包模型在基础的 dp 里是非常重要的一部分。

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1009;

int dp[MAXN];
int c[MAXN], w[MAXN];

int main()
{
    // freopen("input.txt", "r", stdin);
    int t, n, v, i, j;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d", &n, &v);
        for (i = 0; i < n; i++)
            scanf("%d", &c[i]);
        for (i = 0; i < n; i++)
            scanf("%d", &w[i]);
        memset(dp, 0, sizeof(dp));
        for (i = 0; i < n; i++)
            for (j = v; j >= w[i]; j--)
                dp[j] = max(dp[j], dp[j-w[i]] + c[i]);
        printf("%d\n", dp[v]);
    }
```

```
    return 0;
}
```

3.1.1 01 背包

有 N 件物品和一个容量为 V 的背包。第 i 个物品的体积和价值分别是 Ci 和 Wi 。求解将哪些物品放进包里可以使价值最大。

```
void ZeroOnePack(int c, int w)
{
    for(int i = n; i>= c; i--)
        dp[i] = max(dp[i], dp[i-c]+w);
}
```

3.1.2 完全背包

有 N 件物品和一个容量为 V 的背包。第 i 个物品的体积和价值分别是 Ci 和 Wi 。每个物品能无限制的使用任意个。求解将哪些物品放进包里可以使价值最大。

```
void CompletePack(int c, int w)
{
    for(int i = c; i <= c; i++)
        dp[i] = max(dp[i], dp[i-c]+w);
}
```

3.1.3 多重背包

有 N 件物品和一个容量为 V 的背包。第 i 个物品的体积和价值分别是 Ci 和 Wi 。每个物品最多只有 Mi 个物品可用。求解将哪些物品放进包里可以使价值最大。

```
int MultiplePack(int c[], int w[], int num[])
{
    memset(dp, 0, sizeof(dp));
    for(int i = 1; i <= m; i++)
    {
        if(c[i]*num[i]>=n)
          CompletePack(c[i], w[i]);
        int k = 1;
        while(k<num[i])
        {
            ZeroOnePack(k*c[i], k*w[i]);
            num[i]-=k;
            k*=2;
        }
        ZeroOnePack(num[i]*c[i], num[i]*w[i]);
    }
    return dp[n];
}
```

3.2 最长公共子序列（LCS）

问题描述：字符序列的子序列是指从给定字符序列中随意地（不一定连续）去掉若干个字符（可能一个也不去掉）后所形成的字符序列。令给定的字符序列 X="x0，x1，…，xm-1"，序列 Y="y0，y1，…，yk-1"是 X 的子序列，存在 X 的一个严格递增下标序列<i0，i1，…，ik-1>，使得对所有的 j=0，1，…，k-1，有 xij=yj。例如，X="ABCBDAB"，Y="BCDB"是 X 的一个子序列。

考虑最长公共子序列问题如何分解成子问题，设 A="a0，a1，…，am-1"，B="b0，b1，…，bm-1"，并 Z="z0，z1，…，zk-1"为它们的最长公共子序列。不难证明有以下性质：

（1） 如果 am-1=bn-1，则 zk-1=am-1=bn-1，且"z0，z1，…，zk-2"是"a0，a1，…，am-2"和"b0，b1，…，bn-2"的一个最长公共子序列；

（2） 如果 am-1!=bn-1，则若 zk-1!=am-1，蕴涵"z0，z1，…，zk-1"是"a0，a1，…，am-2"和"b0，b1，…，bn-1"的一个最长公共子序列；

（3） 如果 am-1!=bn-1，则若 zk-1!=bn-1，蕴涵"z0，z1，…，zk-1"是"a0，a1，…，am-1"和"b0，b1，…，bn-2"的一个最长公共子序列。

这样，在找 A 和 B 的公共子序列时，如有 am-1=bn-1，则进一步解决一个子问题，找"a0，a1，…，am-2"和"b0，b1，…，bm-2"的一个最长公共子序列；如果 am-1!=bn-1，则要解决两个子问题，找出"a0，a1，…，am-2"和"b0，b1，…，bn-1"的一个最长公共子序列和找出"a0，a1，…，am-1"和"b0，b1，…，bn-2"的一个最长公共子序列，再取两者中较长者作为 A 和 B 的最长公共子序列。

求解：

引进一个二维数组 c[][]，用 c[i][j]记录 X[i]与 Y[j] 的 LCS 的长度，b[i][j]记录 c[i][j]是通过哪一个子问题的值求得的，以决定搜索的方向。

我们是自底向上进行递推计算，那么在计算 c[i,j]之前，c[i-1][j-1]，c[i-1][j]与 c[i][j-1]均已计算出来。此时我们根据 X[i] = Y[j]还是 X[i] != Y[j]，就可以计算出 c[i][j]。

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN=1e3+10;
char x[MAXN],y[MAXN];
int dp[MAXN][MAXN];

int main()
{
    while(~scanf("%s%s",x+1,y+1))
    {
        x[0]=y[0]='.';
        int len=strlen(x)>strlen(y)?strlen(x):strlen(y);
        for(int i=0;i<=len;++i)
            dp[i][0]=dp[0][i]=0;
        for(int j,i=1;i<strlen(x);++i)
            for(j=1;j<strlen(y);++j)
                if(x[i]==y[j])
                    dp[i][j]=dp[i-1][j-1]+1;
                else
                    dp[i][j]=dp[i-1][j]>dp[i][j-1]?dp[i-1][j]:dp[i][j-1];
        printf("%d\n",dp[strlen(x)-1][strlen(y)-1]);
    }
    return 0;
}
```

3.3 最长递增子序列（LIS）

现在给你一个序列，要你求最长上升子序列，那么把这个序列排一个序，然后和原有序列进行 LCS 是不是就解决问题了呢？
当然还有 dp 思路了、
dp[i]以序列中第 i 个元素结尾的最长上升子序列的长度
那么状态转移方程为：if (a[i] > a[j]) dp[i] = MAX (dp[i], dp[j] + 1)

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN=1005;
int dp[MAXN];
int num[MAXN];

int main()
{
    int n;
    while(~scanf("%d",&n))
    {
        memset(dp,0,sizeof(dp));
        for(int i=0;i<n;++i)
        scanf("%d",&num[i]);
        dp[0]=1;
        int x=0;
        for(int j,i=0;i<n;++i)
        {
            int maxn=0;
            for(j=0;j<i;++j) //这里利用了递推的原理
                if(num[i]>num[j]) //由前面的最长递增子序列推出后面的最长递增子
序列
                    maxn=maxn>dp[j]?maxn:dp[j];
            dp[i]=maxn+1;
            if(dp[i]>x)
                x=dp[i]; //x 记录的是当前的最大值
        }
        printf("%d\n",x);
    }
    return 0;
}
```

3.4 最短编辑距离

给定一个长度为 m 和 n 的两个字符串，设有以下几种操作：替换（R），插入（I）和删除（D）且都是相同的操作。寻找到转换一个字符串插入到另一个需要修改的最小（操作）数量。这个数量就可以被视为最小编辑距离。如：acd 与 ace 的 EditionDistance 距离为 1，abc 与 cab 的距离为 1。

```
int calcDistanceDP (char* A, char* B)
{
    int m = strlen(A), n = strlen(B);
    // 生成表
    int *T = (int *)malloc(m * n * sizeof(int));
    // 赋初始值
```

```
    for (int i = 0; i <= m; i++)
        for (int j = 0; j <= n; j++)
            *(T + i * n + j) = 0;
    for (int i = 0; i <= m; i++)
        *(T + i * n) = i;
    for (int i = 0; i <= n; i++)
        *(T + i) = i;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
        {
            int cost = (int)A[i] != B[j];
            int caseA = *(T + i * n + j + 1) + 1;
            int caseB = *(T + (i + 1) * n + j) + 1;
            int caseC = *(T + i * n + j) + cost;
            *(T + (i + 1) * n + j + 1) = getMin(caseA, caseB, caseC);
        }
    return *(T + m * n - 1);
}
```

4. 字符串处理

4.1 KMP

KMP 是一种在线性时间内能处理两个字符串的包含关系的算法，例如求一个字符串里有没有另一个字符串，一个字符串里有几个另一个字符串（可重叠和不可重叠两种）。

```
#include <bits/stdc++.h>
using namespace std;
const int INF = 0x3f3f3f3f;
const int MAXN = 1e6+9, MAXM = 1e4+9;
int str[MAXM], ptr[MAXN], nxt[MAXN];

void getNext(int slen)
{
    memset(nxt, 0, sizeof(str));
    nxt[0] = -1;
    int k = -1;
    for(int i = 1; i <= slen-1; i++)
    {
        while(k>-1 && str[k+1]!=str[i])
            k = nxt[k];
        if(str[k+1] == str[i])
            k++;
        nxt[i] = k;
    }
}

int kmp(int plen, int slen)
{
    int k = -1;
    for(int i = 0; i <= plen-1; i++)
    {
        while(k>-1 && str[k+1]!=ptr[i])
            k = nxt[k];
        if(str[k+1] == ptr[i])
```

```
            k++;
        if(k == slen-1)
            return i-slen+2;
    }
    return -1;
}

int main()
{
    // freopen("input.txt", "r", stdin);
    int t, n, m;
    scanf("%d", &t);
    while(t--)
    {
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i++)
            scanf("%d", &ptr[i]);
        for(int i = 0; i < m; i++)
            scanf("%d", &str[i]);
        getNext(m);
        printf("%d\n", kmp(n, m));
    }
    return 0;
}
```

4.2 字符串哈希

求一个字符串的哈希值

```
#include <bits/stdc++.h>
using namespace std;

unsigned int BKDRHash(char* str)
{
    unsigned int seed = 31;
    unsigned int hash = 0;
    while(*str)
    {
        printf("%d*31 + %d", hash, (int)(*str));
        hash = hash * seed + (*str++);
        printf("=%d\n", hash);
    }
}

int main()
{
    freopen("input.txt", "r", stdin);
    char a[10] = "hello";
    BKDRHash(a);

    return 0;
}
```

4.3 Trie 树

Trie 树，即字典树，又称单词查找树或键树，是一种树形结构，是一种哈希树的变种。典型应用是用于统计和排序大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：最大限度地减少无谓的字符串比较，查询效率比哈希表高。

```
#include <bits/stdc++.h>
using namespace std;

struct node
{
    int cnt;
    node *childs[26];
    node()
    {
        cnt = 0;
        for(int i = 0; i < 26; i++)
            childs[i] = NULL;
    }
};

node *root = new node;
node *current;

void insert(char *str)
{
    current = root;
    int len = strlen(str);
    for(int i = 0; i < len; i++)
    {
        int m = str[i] - 'a';
        if(current->childs[m] == NULL)
            current->childs[m] = new node;
        current = current->childs[m];
        (current->cnt)++;
    }
}

int search(char *str)
{
    current = root;
    int len = strlen(str);
    for(int i = 0; i < len; i++)
    {
        int m = str[i] - 'a';
        if(current->childs[m] == NULL)
            return 0;
        current = current->childs[m];
    }
    return current->cnt;
}

int main()
{
    // freopen("input.txt", "r", stdin);
    char str[20];
    while(gets(str) && strlen(str))
```

```
        insert(str);
    while(scanf("%s", str) != EOF)
        printf("%d\n", search(str));

    return 0;
}
```

4.4 AC 自动机

Aho-Corasick automation，该算法在 1975 年产生于贝尔实验室，是著名的多模匹配算法之一。一个常见的例子就是给出 n 个单词，再给出一段包含 m 个字符的文章，让你找出有多少个单词在文章里出现过。

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 500009;

struct Tire
{
    int nxt[MAXN][128], fail[MAXN], end[MAXN];
    int root, L;
    int newnode()
    {
        for(int i = 0; i < 128; i++)
            nxt[L][i] = -1;
        end[L++] = -1;
        return L-1;
    }
    void init()
    {
        L = 0;
        root = newnode();
    }
    void insert(char buf[], int id)
    {
        int len = strlen(buf);
        int now = root;
        for(int i = 0; i < len; i++)
        {
            if(nxt[now][buf[i]] == -1)
                nxt[now][buf[i]] = newnode();
            now = nxt[now][buf[i]];
        }
        end[now] = id;
    }
    void build()
    {
        queue<int> Q;
        fail[root] = root;
        for(int i = 0; i < 128; i++)
            if(nxt[root][i] == -1)
                nxt[root][i] = root;
            else
            {
                fail[nxt[root][i]] = root;
```

```
                Q.push(nxt[root][i]);
            }
        while(!Q.empty())
        {
            int now = Q.front();
            Q.pop();
            for(int i = 0; i < 128; i++)
                if(nxt[now][i] == -1)
                    nxt[now][i] = nxt[fail[now]][i];
                else
                {
                    fail[nxt[now][i]] = nxt[fail[now]][i];
                    Q.push(nxt[now][i]);
                }
        }
    }
    bool query(char buf[], int id, int n)
    {
        int len = strlen(buf);
        int now = root;
        // int res = 0;
        bool vis[MAXN];
        memset(vis, 0, sizeof(vis));
        bool f = 1;
        for(int i = 0; i < len; i++)
        {
            now = nxt[now][buf[i]];
            int tmp = now;
            while(tmp != root)
            {
                // printf("%d\n", end[tmp]);
                if(end[tmp] != -1)
                {
                    f = 0;
                    vis[end[tmp]] = 1;
                }
                // res += end[tmp];
                // end[tmp] = -1;
                tmp = fail[tmp];
            }
        }
        if(f)
            return false;
        else
        {
            printf("web %d:", id);
            for(int i = 1; i <= n; i++)
                if(vis[i])
                    printf(" %d", i);
            puts("");
            return true;
        }
        // return res;
    }
}ac;

char buf[MAXN << 1];
```

```
int main()
{
    // freopen("input.txt", "r", stdin);
    int m, n;
    scanf("%d", &n);
    ac.init();
    for(int i = 1; i <= n; i++)
    {
        scanf("%s", buf);
        ac.insert(buf, i);
    }
    ac.build();
    scanf("%d", &m);
    int cnt = 0;
    for(int i = 1; i <= m; i++)
    {
        scanf("%s", buf);
        if(ac.query(buf, i, n))
            cnt++;
    }
    printf("total: %d\n", cnt);

    return 0;
}
```

4.5 AC 自动机+矩阵快速幂

不包含敏感词的定长字符串总数

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <queue>
typedef long long ll;
typedef unsigned long long ull;
using namespace std;
struct Matrix
{
    ull mat[40][40];
    int n;
    Matrix(){}
    Matrix(int _n)
    {
        n=_n;
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                mat[i][j] = 0;
    }
    Matrix operator *(const Matrix &b)const
    {
        Matrix ret = Matrix(n);
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                for(int k=0;k<n;k++)
                    ret.mat[i][j]+=mat[i][k]*b.mat[k][j];
```

```
        return ret;
    }
};
ull pow_m(ull a,int n)
{
    ull ret=1;
    ull tmp = a;
    while(n)
    {
        if(n&1)ret*=tmp;
        tmp*=tmp;
        n>>=1;
    }
    return ret;
}
Matrix pow_M(Matrix a,int n)
{
    Matrix ret = Matrix(a.n);
    for(int i=0;i<a.n;i++)
        ret.mat[i][i] = 1;
    Matrix tmp = a;
    while(n)
    {
        if(n&1)ret=ret*tmp;
        tmp=tmp*tmp;
        n>>=1;
    }
    return ret;
}
struct Trie
{
    int next[40][26],fail[40];
    bool end[40];
    int root,L;
    int newnode()
    {
        for(int i = 0;i < 26;i++)
            next[L][i] = -1;
        end[L++] = false;
        return L-1;
    }
    void init()
    {
        L = 0;
        root = newnode();
    }
    void insert(char buf[])
    {
        int len = strlen(buf);
        int now = root;
        for(int i = 0;i < len;i++)
        {
            if(next[now][buf[i]-'a'] == -1)
                next[now][buf[i]-'a'] = newnode();
            now = next[now][buf[i]-'a'];
        }
        end[now] = true;
    }
```

```
    void build()
    {
        queue<int>Q;
        fail[root]=root;
        for(int i = 0;i < 26;i++)
            if(next[root][i] == -1)
                next[root][i] = root;
            else
            {
                fail[next[root][i]] = root;
                Q.push(next[root][i]);
            }
        while(!Q.empty())
        {
            int now = Q.front();
            Q.pop();
            if(end[fail[now]])end[now]=true;
            for(int i = 0;i < 26;i++)
                if(next[now][i] == -1)
                    next[now][i] = next[fail[now]][i];
                else
                {
                    fail[next[now][i]] = next[fail[now]][i];
                    Q.push(next[now][i]);
                }
        }
    }
    Matrix getMatrix()
    {
        Matrix ret = Matrix(L+1);
        for(int i = 0;i < L;i++)
            for(int j = 0;j < 26;j++)
                if(end[next[i][j]]==false)
                    ret.mat[i][next[i][j]] ++;
        for(int i = 0;i < L+1;i++)
            ret.mat[i][L] = 1;
        return ret;
    }
    void debug()
    {
        for(int i = 0;i < L;i++)
        {
            printf("id = %3d,fail = %3d,end = %3d,chi =
[",i,fail[i],end[i]);
            for(int j = 0;j < 26;j++)
                printf("%2d",next[i][j]);
            printf("]\n");
        }
    }
}ac;
char buf[10];
int main()
{
    // freopen("input.txt", "r", stdin);
    int n,L;
    while(scanf("%d%d",&n,&L)==2)
    {
        ac.init();
```

```
        for(int i = 0;i < n;i++)
        {
            scanf("%s",buf);
            ac.insert(buf);
        }
        ac.build();
        Matrix a = ac.getMatrix();
        a = pow_M(a,L);
        ull res = 0;
        for(int i = 0;i < a.n;i++)
            res += a.mat[0][i];
        res--;

        a = Matrix(2);
        a.mat[0][0]=26;
        a.mat[1][0] = a.mat[1][1] = 1;
        a=pow_M(a,L);
        ull ans=a.mat[1][0]+a.mat[0][0];
        ans--;
        ans-=res;
        cout<<ans<<endl;
    }
    return 0;
}
```

4.6 最长回文子串（Manacher 算法）

在有序表中高效查找元素的常用方法是二分查找，所谓二分即是折半，遵循分治的思想，每次将元序列划分成数量尽量相等的两个子序列，然后递归查找，最终定位到目标元素。

```
#include <bits/stdc++.h>
using namespace std;

int manacher (char* s, int len)
{
    int nlen = 2 * len + 3;
    char* str = new char[nlen];
    int i = 0;
    int max = 0;
    str[0] = '$';
    str[1] = '#';
    for (; i < len; i++)
    {
        str[i * 2 + 2] = s[i];
        str[i * 2 + 3] = '#';
    }
    str[nlen - 1] = 0;
    int* p = new int[nlen];
    for (int i = 1; i < nlen; i++)
    {
        p[i] = 0;
    }
    int id = 0;
    for (i = 1; i < nlen; i++)
    {
        if (max > i)
```

```
            p[i] = min(p[2*id-i], p[id]+id-i);
        else
            p[i] = 1;
        while (str[i+p[i]] == str[i-p[i]])
            p[i]++;
        if (p[i]+i > max)
        {
            max = p[i] + i;
            id = i;
        }
    }
    int mx = 0;
    for (i = 1; i < nlen; i++)
    {
        if (mx < p[i] - 1)
            mx = p[i] - 1;
    }
    return mx;
}

int main()
{
    char ch[100];
    while (cin >> ch)
        cout << manacher(ch, strlen(ch)) << endl;
    return 0;
}
```

5. 图论

5.1 最小生成树

定义简单，不多 BB。Prim 算法

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e2+10;
const int INF = 0x7fffffff;

int g[MAXN][MAXN];
bool vis[MAXN];
int n;

int prim()
{
    int sum = 0;
    int mn, t = n-1, k;
    while(t--)
    {
        mn = INF;
        for(int i = 2; i <= n; i++)
            if(!vis[i] && g[1][i]<mn)
            {
                mn = g[1][i];
                k = i;
```

```
            }
        sum += mn;
        vis[k] = 1;
        for(int i = 2; i <= n; i++)
            if(!vis[i] && g[k][i]<g[1][i])
                g[1][i] = g[k][i];
    }
    return sum;
}

int main()
{
    // freopen("input.txt", "r", stdin);
    while(scanf("%d", &n) && n)
    {
        int m = n*(n-1)/2;
        memset(g, 0, sizeof(g));
        memset(vis, 0, sizeof(vis));
        for(int i = 1; i <= m; i++)
        {
            int a, b, c;
            scanf("%d%d%d", &a, &b, &c);
            g[a][b] = g[b][a] = c;
        }
        printf("%d\n", prim());
    }
    return 0;
}
```

5.2 单源最短路

Dijkstra 和 SPFA 算法，2 in 1

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 20010;
const int INF = 0x3f3f3f3f;
struct edge
{
    int to;
    int w;
    int next;
}e[MAXN];
struct point
{
    int val, id;
    point(int id, int val): id(id), val(val) {}
    bool operator <(const point &x)const
    {
        return val > x.val;
    }
};
int head[MAXN], dist[MAXN];
bool vis[MAXN];
int n, m, cnt;
void add(int i, int j, int w)
```

```
{
    e[cnt].to = j;
    e[cnt].w = w;
    e[cnt].next = head[i];
    head[i] = cnt++;
}
void spfa(int s)
{
    queue<int> q;
    memset(dist, 0x3f, sizeof(dist));
    memset(vis, 0, sizeof(vis));
    vis[s] = 1;
    q.push(s);
    dist[s] = 0;
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for(int i = head[u]; i != -1; i = e[i].next)
        {
            int v = e[i].to;
            if(dist[v] > dist[u] + e[i].w)
            {
                dist[v] = dist[u] + e[i].w;
                if(!vis[v])
                {
                    vis[v] = 1;
                    q.push(v);
                }
            }
        }
    }
}
void dijkstra(int s)
{
    memset(vis, 0, sizeof(vis));
    memset(dist, 0x3f, sizeof(dist));
    priority_queue<point> q;
    q.push(point(s, 0));
    dist[s] = 0;
    while(!q.empty())
    {
        int u = q.top().id;
        q.pop();
        if(vis[u])
            continue;
        vis[u] = true;
        for (int i = head[u]; i != -1; i = e[i].next)
        {
            int id = e[i].to;
            if (!vis[id] && dist[u] + e[i].w < dist[id])
            {
                dist[id] = dist[u] + e[i].w;
                q.push(point(id, dist[id]));
            }
        }
    }
```

```
}
int main()
{
    freopen("input.txt", "r", stdin);
    int a, b, c, s, e;
    while(scanf("%d%d", &n, &m) == 2)
    {
        cnt = 0;
        memset(head, -1, sizeof(head));
        for(int i = 0; i < m; i++)
        {
            scanf("%d%d%d", &a, &b, &c);
            add(a, b, c);
            add(b, a, c);
        }
        scanf("%d%d", &s, &e);
        spfa(s);
        // dijkstra(s);
        printf("%d\n", dist[e]==INF ? -1 : dist[e]);
    }
    return 0;
}
```

5.3 多源最短路

Floyd 算法，暴力美学

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const ll INF = 1e18;
const int VN = 105;

int n;
int m;
ll d[VN][VN];

ll X[VN];
ll L1,L2,L3,L4;
ll C1,C2,C3,C4;

ll myabs(ll x) {
    return x<0?-x:x;
}
void init() {
    for(int i=1; i<=n; ++i) {
        d[i][i] = 0;
        for(int j=i+1; j<=n; ++j)
            d[i][j]=d[j][i]=INF;
    }
}

ll getCost(ll dist) {
    if(0<dist && dist<=L1) return C1;
    if(L1<dist && dist<=L2) return C2;
```

```
    if(L2<dist && dist<=L3) return C3;
    if(L3<dist && dist<=L4) return C4;
    return INF;
}

void Floyd() {
    for(int k=1; k<=n; ++k)
        for(int i=1; i<=n; ++i)if(d[i][k]!=INF)
                for(int j=1; j<=n; ++j)if(d[k][j]!=INF)
                        d[i][j]=min(d[i][j], d[i][k]+d[k][j]);
}

int main() {
    // freopen("input.txt", "r", stdin);
    int T,cas=1;
    scanf("%d",&T);
    while(T--) {
        cin >> L1 >> L2 >> L3 >> L4;
        cin >> C1 >> C2 >> C3 >> C4;
        scanf("%d%d",&n,&m);
        for(int i=1; i<=n; ++i)
            cin >> X[i];
        init();
        for(int i=1; i<=n; ++i) {
            for(int j=i+1; j<=n; ++j) {
                d[i][j]=d[j][i]=getCost(myabs(X[i]-X[j]));
            }
        }
        Floyd();
        int u,v;

        printf("Case %d:\n",cas++);
        for(int i=0; i<m; ++i) {
            scanf("%d%d",&u,&v);
            if(d[u][v]!=INF) {
                printf("The minimum cost between station %d and station %d
is ",u,v);
                cout << d[u][v] << ".\n";
            }
            else
                printf("Station %d and station %d are not
attainable.\n",u,v);
        }
    }
    return 0;
}
```

5.4 匈牙利算法

匈牙利算法用来求二分图的最大匹配

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e3+9;
const int INF = 0x3f3f3f3f;
```

```
bool mp[MAXN][MAXN];
bool used[MAXN];
int link[MAXN];
int n, m;

bool find(int x)
{
    for(int i = 1; i <= n; i++)
        if(mp[x][i] && !used[i])
        {
            used[i] = 1;
            if(!link[i] || find(link[i]))
            {
                link[i] = x;
                return 1;
            }
        }
    return 0;
}

int main()
{
    // freopen("input.txt", "r", stdin);
    int t;
    scanf("%d", &t);
    while(t--)
    {
        scanf("%d%d", &n, &m);
        memset(mp, 0, sizeof(mp));
        memset(link, 0, sizeof(link));
        for(int i = 0; i < m; i++)
        {
            int a, b;
            scanf("%d%d", &a, &b);
            mp[a][b] = 1;
        }
        int cnt = 0;
        for(int i = 1; i <= n; i++)
        {
            memset(used, 0, sizeof(used));
            if(find(i))
                cnt++;
        }
        printf("%d\n", n - cnt);
    }

    return 0;
}
```

5.5 最近公共祖先（LCA）

LCA,Lowest Common Ancestor,最近的公共祖先。在一棵树中对于两个节点 u , v 找出节点 T，使得 T 同时为 u，v 的祖先。显然这样的 T 点肯定存在且有可能有多个，其中深度最大的那个点肯定为即为 u，v 两点的 LCA。关于 LCA 的解法有很多种，暴力枚举，事先需要知道所有询问的离

线的 tarjan 算法和基于 RMQ 的在线算法。本模板使用 Tarjan 实现。

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 4e4+9;
const int MAXQ = 2e4+9;
const int INF = 0x3f3f3f3f;

struct edge
{
    int to;
    int nxt;
    int lca;
}e[MAXN*2], qe[MAXQ];

int head[MAXN], qhead[MAXN], pre[MAXN], dist[MAXN];
bool vis[MAXN];
int n, m, cnt, qcnt;

int find(int x)
{
    if(x != pre[x])
        pre[x] = find(pre[x]);
    return pre[x];
}

void lca(int u)
{
    pre[u] = u;
    vis[u] = true;
    for(int k = head[u]; k != -1; k = e[k].nxt)
    {
        if(!vis[e[k].to])
        {
            dist[e[k].to] = dist[u] + e[k].lca;
            lca(e[k].to);
            pre[e[k].to] = u;
        }
    }
    for(int k = qhead[u]; k != -1; k = qe[k].nxt)
    {
        if(vis[qe[k].to])
        {
            qe[k].lca = dist[u] + dist[qe[k].to] - 2*dist[find(qe[k].to)];
            qe[k^1].lca = qe[k].lca;
        }
    }
}

void init()
{
    cnt = 0;
    qcnt = 0;
    memset(pre, 0, sizeof(pre));
    memset(head, -1, sizeof(head));
    memset(qhead, -1, sizeof(qhead));
    memset(vis, 0, sizeof(vis));
    memset(e, 0, sizeof(e));
```

```
    memset(qe, 0, sizeof(qe));
    memset(dist, 0, sizeof(dist));
}

void add(int u, int v, int w)
{
    e[cnt].to = v;
    e[cnt].lca = w;
    e[cnt].nxt = head[u];
    head[u] = cnt++;
}

void qadd(int u, int v)
{
    qe[qcnt].to = v;
    qe[qcnt].nxt = qhead[u];
    qhead[u] = qcnt++;
}

int main()
{
    // freopen("input.txt", "r", stdin);
    int n, m, u, v, w;
    char tmp;
    scanf("%d%d", &n, &m);
    init();
    for(int i = 0; i < m; i++)
    {
        scanf("%d%d%d %c", &u, &v, &w, &tmp);
        add(u, v, w);
        add(v, u, w);
    }
    int k;
    scanf("%d", &k);
    for(int i = 0; i < k; i++)
    {
        scanf("%d%d", &u, &v);
        qadd(u, v);
        qadd(v, u);
    }
    lca(1);
    for(int i = 0; i < qcnt; i += 2)
        printf("%d\n", qe[i].lca);

    return 0;
}
```

5.6 强连通分量

Tarjan 求一个图有多少连通分量

```
#include <bits/stdc++.h>
using namespace std;
#define maxn 1000000
vector<int >mp[maxn];
int ans[maxn];
```

```
int vis[maxn];
int dfn[maxn];
int low[maxn];
int n,m,tt,cnt,sig;
void init()
{
    memset(low,0,sizeof(low));
    memset(dfn,0,sizeof(dfn));
    memset(vis,0,sizeof(vis));
    for(int i=1;i<=n;i++)mp[i].clear();
}
void Tarjan(int u)
{
    vis[u]=1;
    low[u]=dfn[u]=cnt++;
    for(int i=0;i<mp[u].size();i++)
    {
        int v=mp[u][i];
        if(vis[v]==0)Tarjan(v);
        if(vis[v]==1)low[u]=min(low[u],low[v]);
    }
    if(dfn[u]==low[u])
        sig++;
}
void Slove()
{
    tt=-1;cnt=1;sig=0;
    for(int i=1;i<=n;i++)
    {
        if(vis[i]==0)
        {
            Tarjan(i);
        }
    }
    printf("%d\n",sig);
}
int main()
{
    freopen("input.txt", "r", stdin);
    while(~scanf("%d",&n))
    {
        if(n==0)break;
        scanf("%d",&m);
        init();
        for(int i=0;i<m;i++)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            mp[x].push_back(y);
        }
        Slove();
    }
}
```

5.7 Tarjan 求割点/桥

在一个无向图中,如果删去一个点,这个图的连通度增加了,删去的这个点就是原图的一个割点;如果删掉一条边，这个图的连通度增加了，删掉的这条边就是原图的一个桥。

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 209
vector<int>G[MAXN];
int n,m,low[MAXN],dfn[MAXN];
bool is_cut[MAXN];
int father[MAXN];
int tim=0;

void input()
{
    scanf("%d%d",&n,&m);
    int a,b;
    for(int i=1;i<=m;++i)
    {
        scanf("%d%d",&a,&b);
        G[a].push_back(b);/*邻接表储存无向边*/
        G[b].push_back(a);
    }
}

void Tarjan(int i,int Father)
{
    father[i]=Father;/*记录每一个点的父亲*/
    dfn[i]=low[i]=tim++;
    for(int j=0;j<G[i].size();++j)
    {
        int k=G[i][j];
        if(dfn[k]==-1)
        {
            Tarjan(k,i);
            low[i]=min(low[i],low[k]);
        }
        else if(Father!=k)/*假如 k 是 i 的父亲的话，那么这就是无向边中的重边，有重边那么一定不是桥*/
            low[i]=min(low[i],dfn[k]);//dfn[k]可能! =low[k]，所以不能用low[k]代替 dfn[k],否则会上翻过头了。
    }
}

void count()
{
    int rootson=0;
    Tarjan(1,0);
    for(int i=2;i<=n;++i)
    {
        int v=father[i];
        if(v==1)
            rootson++;/*统计根节点子树的个数，根节点的子树个数>=2,就是割点*/
        else if(low[i]>=dfn[v])/*割点的条件*/
            is_cut[v]=true;
    }
    if(rootson>1)
        is_cut[1]=true;
```

```
    for(int i=1;i<=n;++i)
        if(is_cut[i])
            printf("%d\n",i);
    for(int i=1;i<=n;++i)
    {
        int v=father[i];
        if(v>0&&low[i]>dfn[v])/*桥的条件*/
            printf("%d,%d\n",v,i);
    }

}

int main()
{
    input();
    memset(dfn,-1,sizeof(dfn));
    memset(father,0,sizeof(father));
    memset(low,-1,sizeof(low));
    memset(is_cut,false,sizeof(is_cut));
    count();
    return 0;
}
```

5.8 拓扑排序

给出一些大小关系，然后求出一个总的大小关系。比如：甲>丙， 丙>乙，可以的到一个大小关系：甲 > 丙 > 乙

```
#include <bits/stdc++.h>
using namespace std;

int map[555][555];
int indegree[555];
bool vis[555];
int n,m;

void topSort()
{
    int cnt = 0;
    while(cnt != n)
    {
        for(int i=1;i<=n;i++)
        {
            if(indegree[i] == 0 && !vis[i])
            {
                if(cnt)
                    printf(" ");
                printf("%d",i);
                for(int j=1;j<=n;j++)
                    if(map[i][j])
                        indegree[j]--;
                cnt++;
                vis[i] = true;
                break;
            }
        }
```

```
    }
    printf("\n");
}
int main()
{
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        int a,b;
        memset(map,0,sizeof(map));
        memset(vis,0,sizeof(vis));
        memset(indegree,0,sizeof(indegree));
        for(int i=0;i<m;i++)
        {
            scanf("%d%d",&a,&b);
            if(!map[a][b])
            {
                map[a][b] = 1;
                indegree[b]++; //入度+1
            }
        }
        topSort();
    }
    return 0;
}
```

5.9 最大流

Dinic

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;

const int MAXN = 2009;
const int MAXM = 1200009;
const int INF = 0x3f3f3f3f;

struct Edge
{
    int to, next, cap, flow;
}edge[MAXM];
int tol;
int head[MAXN];
void init()
{
    tol = 2;
    memset(head, -1, sizeof(head));
}

void addedge(int u, int v, int w, int rw = 0)
{
    edge[tol].to = v; edge[tol].cap = w; edge[tol].flow = 0;
    edge[tol].next = head[u]; head[u] = tol++;
    edge[tol].to = u; edge[tol].cap = rw; edge[tol].flow = 0;
    edge[tol].next = head[v]; head[v] = tol++;
```

```
}

int Q[MAXN];
int dep[MAXN], cur[MAXN], sta[MAXN];

bool bfs(int s, int t)
{
    int front = 0, tail = 0;
    memset(dep, -1, sizeof(dep));
    dep[s] = 0;
    Q[tail++] = s;
    while(front < tail)
    {
        int u = Q[front++];
        for(int i = head[u]; i != -1; i = edge[i].next)
        {
            int v = edge[i].to;
            if(edge[i].cap > edge[i].flow && dep[v] == -1)
            {
                dep[v] = dep[u]+1;
                if(v == t)
                    return true;
                Q[tail++] = v;
            }
        }
    }
    return false;
}

int dinic(int s, int t)
{
    int maxflow = 0;
    while(bfs(s, t))
    {
        for(int i = 0; i < t; i++)
            cur[i] = head[i];
        int u = s, tail = 0;
        while(cur[s] != -1)
        {
            if(u == t)
            {
                int tp = INF;
                for(int i = tail-1; i >= 0; i--)
                    tp = min(tp, edge[sta[i]].cap - edge[sta[i]].flow);
                maxflow += tp;
                for(int i = tail-1; i >= 0; i--)
                {
                    edge[sta[i]].flow += tp;
                    edge[sta[i]^1].flow -= tp;
                    if(edge[sta[i]].cap - edge[sta[i]].flow == 0)
                        tail = i;
                }
                u = edge[sta[tail]^1].to;
            }
            else if(cur[u] != -1 && edge[cur[u]].cap > edge[cur[u]].flow
&& dep[u] + 1 == dep[edge[cur[u]].to])
            {
                sta[tail++] = cur[u];
```

```
                u = edge[cur[u]].to;
            }
            else
            {
                while(u != s && cur[u] == -1)
                    u = edge[sta[--tail]^1].to;
                cur[u] = edge[cur[u]].next;
            }
        }
    }
    return maxflow;
}

int main()
{
//  freopen("input.txt", "r", stdin);
    int n, m, k, f, d, t;
    while(scanf("%d%d%d", &n, &m, &k) == 3)
    {
        init();
        for(int i = 1; i <= m; i++)
            addedge(0, i, 1);
        for(int i = 1; i <= k; i++)
            addedge(i + m + 2*n, k + m + 2*n + 1, 1);
        for(int i = 1; i <= n; i++)
        {
            addedge(i + m, i + m + n, 1);
            scanf("%d%d", &f, &d);
            for(int j = 1; j <= f; j++)
            {
                scanf("%d", &t);
                addedge(t, i+m, 1);
            }
            for(int j = 1; j <= d; j++)
            {
                scanf("%d", &t);
                addedge(i + m + n, t + m + 2*n, 1);
            }
        }
        printf("%d\n", dinic(0, 2*n + m + k + 1));
    }

    return 0;
}
```

SAP

```
//1002
/*
HDU 4289
G++  62ms  1888K
最大流
SAP
*/
#include<stdio.h>
#include<iostream>
#include<map>
#include<set>
```

```
#include<algorithm>
#include<string.h>
#include<stdlib.h>
using namespace std;

const int MAXN=5000;//点数的最大值
const int MAXM=2500000;//边数的最大值
const int INF=0x3f3f3f3f;

struct Node
{
    int from,to,next;
    int cap;
}edge[MAXM];
int tol;
int head[MAXN];
int dep[MAXN];
int gap[MAXN];//gap[x]=y:说明残留网络中 dep[i]==x 的个数为 y

int n;//点的实际个数，一定是总的点的个数，包括源点和汇点
void init()
{
    tol=0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v,int w)
{
    edge[tol].from=u;
    edge[tol].to=v;
    edge[tol].cap=w;
    edge[tol].next=head[u];
    head[u]=tol++;
    edge[tol].from=v;
    edge[tol].to=u;
    edge[tol].cap=0;
    edge[tol].next=head[v];
    head[v]=tol++;
}
void BFS(int start,int end)
{
    memset(dep,-1,sizeof(dep));
    memset(gap,0,sizeof(gap));
    gap[0]=1;
    int que[MAXN];
    int front,rear;
    front=rear=0;
    dep[end]=0;
    que[rear++]=end;
    while(front!=rear)
    {
        int u=que[front++];
        if(front==MAXN)front=0;
        for(int i=head[u];i!=-1;i=edge[i].next)
        {
            int v=edge[i].to;
            if(edge[i].cap!=0||dep[v]!=-1)continue;
            que[rear++]=v;
            if(rear==MAXN)rear=0;
```

```
            dep[v]=dep[u]+1;
            ++gap[dep[v]];
        }
    }
}
int SAP(int start,int end)
{
    int res=0;
    BFS(start,end);
    int cur[MAXN];
    int S[MAXN];
    int top=0;
    memcpy(cur,head,sizeof(head));
    int u=start;
    int i;
    while(dep[start]<n)
    {
        if(u==end)
        {
            int temp=INF;
            int inser;
            for(i=0;i<top;i++)
               if(temp>edge[S[i]].cap)
               {
                   temp=edge[S[i]].cap;
                   inser=i;
               }
            for(i=0;i<top;i++)
            {
                edge[S[i]].cap-=temp;
                edge[S[i]^1].cap+=temp;
            }
            res+=temp;
            top=inser;
            u=edge[S[top]].from;
        }
        if(u!=end&&gap[dep[u]-1]==0)//出现断层，无增广路
          break;
        for(i=cur[u];i!=-1;i=edge[i].next)
           if(edge[i].cap!=0&&dep[u]==dep[edge[i].to]+1)
            break;
        if(i!=-1)
        {
            cur[u]=i;
            S[top++]=i;
            u=edge[i].to;
        }
        else
        {
            int min=n;
            for(i=head[u];i!=-1;i=edge[i].next)
            {
                if(edge[i].cap==0)continue;
                if(min>dep[edge[i].to])
                {
                    min=dep[edge[i].to];
                    cur[u]=i;
                }
```

```
            }
            --gap[dep[u]];
            dep[u]=min+1;
            ++gap[dep[u]];
            if(u!=start)
              u=edge[S[--top]].from;
        }

    }
    return res;
}

int main()
{
    int N,M;
    int u,v;
    int start;
    int end;
    while(scanf("%d%d",&N,&M)!=EOF)
    {
        init();
        scanf("%d%d",&start,&end);
        start=2*start-1;
        end=2*end;
        n=2*N;
        for(int i=1;i<=N;i++)
        {
            scanf("%d",&u);
            addedge(2*i-1,2*i,u);
            addedge(2*i,2*i-1,u);
        }
        while(M--)
        {
            scanf("%d%d",&u,&v);
            addedge(2*u,2*v-1,INF);
            addedge(2*v,2*u-1,INF);//这里一定要注意
        }
        printf("%d\n",SAP(start,end));
    }
    return 0;
}
```

5.10 最小费用最大流

增广函数使用 SPFA，求最大费用加负边；求累乘最小费用：取 log2，指数相加再用 pow(2, ans) 复原答案。

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 10000;
const int MAXM = 100000;
const int INF = 0x3f3f3f3f;
struct Edge
{
    int to, next, cap, flow;
    double cost;
```

```
    int x, y;
} edge[MAXM], HH[MAXN], MM[MAXN];
int head[MAXN], tol;
int pre[MAXN], dis[MAXN];
bool vis[MAXN];
int N, M;
char map[MAXN][MAXN];
void init()
{
    N = MAXN;
    tol = 0;
    memset(head, -1, sizeof(head));
}
void addedge(int u, int v, int cap, double cost) //左端点，右端点，容量，花费
{
    // printf("%d %d %d %lf\n", u, v, cap, cost);
    edge[tol].to = v;
    edge[tol].cap = cap;
    edge[tol].cost = cost;
    edge[tol].flow = 0;
    edge[tol].next = head[u];
    head[u] = tol++;
    edge[tol].to = u;
    edge[tol].cap = 0;
    edge[tol].cost = -cost;
    edge[tol].flow = 0;
    edge[tol].next = head[v];
    head[v] = tol++;
}
bool spfa(int s, int t)
{
    queue<int> q;
    for (int i = 0; i < N; i++)
    {
        dis[i] = INF;
        vis[i] = false;
        pre[i] = -1;
    }
    dis[s] = 0;
    vis[s] = true;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        vis[u] = false;
        for (int i = head[u]; i != -1; i = edge[i].next)
        {
            int v = edge[i].to;
            if (edge[i].cap > edge[i].flow &&
                dis[v] > dis[u] + edge[i].cost)
            {
                dis[v] = dis[u] + edge[i].cost;
                pre[v] = i;
                if (!vis[v])
                {
                    vis[v] = true;
```

```
                    q.push(v);
                }
            }
        }
    }
    if (pre[t] == -1)
        return false;
    else
        return true;
}
//返回的是最大流， cost 存的是最小费用
int minCostMaxflow(int s, int t, double &cost)
{
    int flow = 0;
    cost = 0;
    while (spfa(s, t))
    {
        int Min = INF;
        for (int i = pre[t]; i != -1; i = pre[edge[i ^ 1].to])
        {
            if (Min > edge[i].cap - edge[i].flow)
                Min = edge[i].cap - edge[i].flow;
        }
        for (int i = pre[t]; i != -1; i = pre[edge[i ^ 1].to])
        {
            edge[i].flow += Min;
            edge[i ^ 1].flow -= Min;
            cost += edge[i].cost * Min;
            // cout << cost << endl;
        }
        flow += Min;
    }
    return flow;
}

int main()
{
    freopen("input.txt", "r", stdin);
    int t;
    scanf("%d", &t);
    while(t--)
    {
        init();
        int n, m;
        scanf("%d%d", &n, &m);
        int s = 0, e = 300;
        for(int i = 1; i <= n; i++)
        {
            int a, b;
            scanf("%d%d", &a, &b);
            a -= b;
            if(a > 0)
                addedge(s, i, a, 0);
            if(a < 0)
                addedge(i, e, -a, 0);
        }
        for(int i = 0; i < m; i++)
        {
```

```
            int u, v, c;
            double p;
            scanf("%d%d%d%lf", &u, &v, &c, &p);
            p = -log2(1.0-p);
            // printf("%lf\n", p);
            if(c > 0)
            {
                addedge(u, v, 1, 0);
                c--;
            }
            // cout << c << endl;
            addedge(u, v, c, p);
        }
        double ans = 0;
        minCostMaxflow(s, e, ans);
        // cout << ans << endl;
        ans = pow(2, -ans);
        printf("%.2lf\n", 1.0-ans);
    }
    return 0;
}
```

5.11 欧拉回路

算法思想：

判断一个图中是否存在欧拉回路（每条边恰好只走一次，并能回到出发点的路径），在以下三种情况中有三种不同的算法：

一、无向图
每个顶点的度数都是偶数，则存在欧拉回路。

二、有向图（所有边都是单向的）
每个节顶点的入度都等于出度，则存在欧拉回路。
以上两种情况都很好理解。其原理就是每个顶点都要能进去多少次就能出来多少次。

三、混合图（有的边是单向的，有的边是无向的。常被用于比喻城市里的交通网络，有的路是单行道，有的路是双行道。）
找到一个给每条无向的边定向的策略，使得每个顶点的入度等于出度，这样就能转换成上面第二种情况。这就可以转化成一个二部图最大匹配问题。网络模型如下：
1．新建一个图。
2．对于原图中每一条无向边 i，在新图中建一个顶点 e（i）；
3．对于原图中每一个顶点 j，在新图中建一个顶点 v（j）。
4．如果在原图中，顶点 j 和 k 之间有一条无向边 i，那么在新图中从 e（i）出发，添加两条边，分别连向 v（j）和 v（k），容量都是 1。
5．在新图中，从源点向所有 e（i）都连一条容量为 1 的边。
6．对于原图中每一个顶点 j，它原本都有一个入度 in、出度 out 和无向度 un。显然我们的目的是要把所有无向度都变成入度或出度，从而使它的入度等于总度数的一半，也就是(in + out + un) / 2（显然与此同时出度也是总度数的一半，如果总度数是偶数的话）。当然，如果 in 已经大于总度数的一半，或者总度数是奇数，那么欧拉回路肯定不存大。如果 in 小于总度数的一半，并且总度数是偶数，那么我们在新图中从 v（j）到汇点连一条边，容量就是(in + out + un) / 2 – in，也就是原图中顶点 j 还需要多少入度。
按照这个网络模型算出一个最大流，如果每条从 v（j）到汇点的边都达到满流量的话，那么欧拉回路成立。

5.12 哈密顿回路

欧拉回路全点经过一次，哈密顿回路全边经过一次。

```
#include <string.h>
#include <iostream>
#include <stdio.h>
using namespace std;
#define V 200
int n,m;
bool c[V][V];
int x[V];
bool flag[V];
void hamilton()
{
    int i, k;
    bool s[V];
    for(i = 0; i < n; i++)
    {
        x[i] = -1;
        s[i] = false;
    }
    k = 1;
    s[0] = true;
    x[0] = 0;
    while(k >= 0)
    {
        x[k]++;
        while(x[k] < n)
            if(!s[x[k]] && c[x[k - 1]][x[k]])
                break;
            else
                x[k]++;
        if((x[k] < n) && (k != n - 1))
        {
            s[x[k]] = true;
            k++;
        }
        else if((x[k] < n) && k == n - 1 && c[x[k]][x[0]])
        {
            break;
        }
        else
        {
            x[k] = -1;
            k--;
            s[x[k]] = false;
        }
    }
}
int main()
{
    // freopen("input.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    int a,b;
```

```
    while(cin >> n >> m){
        memset(c,0,sizeof(c));
        memset(flag,0,sizeof(flag));
        memset(x,0,sizeof(x));
        for(int i=0;i<m;i++){
            cin >> a >> b;
            c[a-1][b-1]=c[b-1][a-1]=true;
        }

        hamilton();
        bool f=0;
        for(int i=0;i<n;i++){
            flag[x[i]]=1;
        }
        for(int i=0;i<n;i++){
            if(!flag[i]) f=1;
        }
        if(f) cout <<"no solution" <<endl;
        else{
            for(int i=0;i<n;i++){
                if(i==n-1) cout << x[i]+1 <<endl;
                else cout << x[i]+1 <<" ";
            }
        }
    }
    return 0;
}
```

5.13 2-SAT

有 n 对夫妻被邀请参加一个聚会，因为场地的问题，每对夫妻中只有 1 人可以列席。在 2n 个人中，某些人之间有着很大的矛盾（当然夫妻之间是没有矛盾的），有矛盾的 2 个人是不会同时出现在聚会上的。有没有可能会有 n 个人同时列席?

```
//#include <bits/stdc++.h>
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;
const int MAXN = 1e6+7;
struct node
{
    int v,next;
}edge[MAXN];
int head[MAXN],idx;
void add_edge(int u,int v)
{
    edge[idx].v=v;
    edge[idx].next=head[u];
    head[u]=idx++;
}
int DFN[MAXN],low[MAXN],stack_[MAXN],in_stack[MAXN],belong[MAXN];
int cir,top,temp;
```

```
void Tarjan(int u)
{
    int p;
    DFN[u]=low[u]=++temp;
    in_stack[u]=1;
    stack_[top++]=u;
    for(int i=head[u]; i+1; i=edge[i].next)
    {
        int v=edge[i].v;
        if(!DFN[v])
        {
            Tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(in_stack[v])
        {
            low[u]=min(low[u],DFN[v]);
        }
    }
    if(low[u]==DFN[u])
    {
        ++cir;
        do{
            p=stack_[--top];
            in_stack[p]=0;
            belong[p]=cir;
        }while(p!=u);
    }
}
int main()
{
    // freopen("input.txt", "r", stdin);
    int n,m;
    while(scanf("%d",&n)!=EOF)
    {
        int a,b,c,d;
        temp=cir=top=idx=0;
        memset(DFN,0,sizeof(DFN));
        memset(head,-1,sizeof(head));
        memset(in_stack,0,sizeof(in_stack));
        scanf("%d",&m);
        for(int i=0; i<m; ++i)
        {
            scanf("%d%d%d%d",&a,&b,&c,&d);
            a=(a<<1)+c;
            b=(b<<1)+d;
            add_edge(a,b^1);
            add_edge(b,a^1);
        }
        for(int i=0; i<n<<1; ++i)
            if(!DFN[i])
            Tarjan(i);
        int flag=0;
        for(int i=0; i<n; ++i)
        {
            if(belong[i<<1]==belong[(i<<1)^1])
                flag=1;
        }
```

```
        if(flag)printf("NO\n");
        else printf("YES\n");
    }
    return 0;
}
```

5.14 不定根最小树形图

HDU 2121 无根节点最小树形图

题意：n，m 分别代表有 n 个城市，m 条边，然后是 s,t,c 代表一条从 s 出发到 t 的边权值是 c。问你求一个城市，这个城市到其他所有城市的总和最小

思路：因为是有向图，很容易想到最小树形图，不过无根要怎么处理呢？ 实际上我们只要假设有一个超级源点 0 号点，在原有边的基础上，再从 0 号点到各个点拉一条边，0 号点到其他所有点的距离是所有边的总和+1（为什么要+1 后面解释） 然后以超级源点为起点对新图求一个最小树形图。

又因为从最小树形图里超级源点到其他所有点里有且仅能有一条边，我们可以有反证法，如果存在两条从超级源点出发的边，那么由于从超级源点出发的边的长度比所有边都要长，而在已经有一条从超级源点出发的边的基础上，第二个点仍然要选择超级源点，那么我们可以确定，第二个点是单独出来的点，不跟其他任何边有连接或者图里存在两个入度为 0 的点（这两种情况都是不可能存在最小树形图的情况）

懂了这个我们就可以求出有超级源点的情况下最小树形图的最大长度（上界）是多少，假设题目给的边都在最小树形图里，那么此时达到最大长度，是 sum+sum-1

因此如果最小树形图的长度 > =2*sum，那么就不满足情况了。

注意 sum 不能不加一，我们可以假设存在一种情况有两个点，0 条边，那么此时会有两条从超级源点出发的长度为 0 的边，那么此时无法判断是不是满足条件(0>2*0 不满足，会认为是满足条件的，实际上 0 条边是不满足条件的)

所以排除掉不成立情况，最后结果就是有超级源点的情况下最小树形图的长度减去 sum

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
#define N 1010
#define INF 0x7f7f7f7f
struct Edge
{
    int u,v,w;
} e[N*N];
int cnt;
int in[N];
int vis[N],pre[N],id[N];
int minroot;
void addedge(int u,int v,int w)
{
    e[cnt].u=u;
    e[cnt].v=v;
    e[cnt++].w=w;
}
int Directed_MST(int root,int NV,int NE)
{
    int ret = 0;
```

```
    while(true)
    {
        ///步骤1：找到最小边
        for(int i = 0; i < NV; i ++)
            in[i] = INF;
            memset(pre,-1,sizeof(pre));
        for(int i = 0; i < NE; i ++)
        {
            int u = e[i].u , v = e[i].v;
            if(e[i].w < in[v] && u != v)
            {
                pre[v] = u;
                in[v] = e[i].w;
                if(u==root) minroot=i;
            }
        }
        for(int i = 0; i < NV; i ++)
        {
            if(i == root) continue;
            if(in[i] == INF) return -1;///除了根节点以外有点没有入边，则根无
法到达他
        }
        int cntnode = 0;
        memset(id,-1,sizeof(id));
        memset(vis,-1,sizeof(vis));
        ///找环
        in[root] = 0;
        for(int i = 0; i < NV; i ++) ///标记每个环，编号
        {
            ret += in[i];
            int v = i;
            while(vis[v] != i && id[v] == -1 && v != root)
            {
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && id[v] == -1)
            {
                for(int u = pre[v]; u != v; u = pre[u])
                {
                    id[u] = cntnode;
                }
                id[v] = cntnode ++;
            }
        }
        if(cntnode == 0) break;//无环
        for(int i = 0; i < NV; i ++)
            if(id[i] == -1)
                id[i] = cntnode ++;
        ///步骤3：缩点，重新标记
        for(int i = 0; i < NE; i ++)
        {
            int u=e[i].u;
            int v = e[i].v;
            e[i].u = id[u];
            e[i].v = id[v];
            if(e[i].u != e[i].v) e[i].w -= in[v];
        }
```

```
        NV = cntnode;
        root = id[root];
    }
    return ret;///最小树形图的长度
}

int main()
{
    // freopen("input.txt", "r", stdin);
    int n,m,sum;
    int u,v,w;
    while(scanf("%d %d",&n,&m)!=EOF)
    {
        cnt=0;sum=0;
        for(int i=0; i<m; i++)
        {
            scanf("%d %d %d",&u,&v,&w);
            addedge(u+1,v+1,w);
            sum+=w;
        }
        sum++;
        for(int i=1; i<=n; i++)
            addedge(0,i,sum);
        int ans=Directed_MST(0,n+1,cnt);
        if(ans==-1||ans>=2*sum)
            printf("impossible\n\n");
        else
            printf("%d %d\n\n",ans-sum,minroot-m);
    }
    return 0;
}
```

5.15 TSP 旅行商

Problem:

有 n 个城市，John 从第一个城市出发，最终回到第一个城市，每个城市只经过一次，求最小代价?

Solution:

先利用 floyd 求出两两城市之间的最短路，然后以 1 为起点跑一下 TSP 算法，min(dp[1<<(n-1)][k]+dist[k][1]）就是最小代价。

TSP 算法解释:

定义 dp[i][k]为集合 i 中最后一个到达 k 节点所付出的代价。i 的每一个二进制位都代表对应的元素是否存在。状态转换为:

dp[i|(1<<(j-1))][j] = min(dp[i|(1<<(j-1))][j], dp[i][k]+G[k][j]);

可以理解为不包含 j 元素的集合 i 中最后一个元素为 k，加上 k 到 j 的距离就是包含 j 元素的集合中最后一个元素是 j 的值，遍历所有的 k 找到最小值即可。dp[1][1] = 0 开始遍历则说明了起点是 1，如果 dp[1<<(k-1)][k] = 0，则说明 k 可以使起点，表示集合中只有 k 这个元素且重点为 k。

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
```

```
typedef unsigned long long ull;

const double PI = 3.141592653589;
const int INF = 0x3fffffff;

int dp[1 << 17][17];
int G[17][17];
int n;

int main()
{
    // freopen("input.txt", "r", stdin);
    ios::sync_with_stdio(false);

    int t, m, u, v, w;
    cin >> t;
    while (t--)
    {
        cin >> n >> m;
        //init
        for (int i = 0; i <= n; i++)
            for (int j = 0; j <= n; j++)
                G[i][j] = (i == j) ? 0 : INF;
        for (int i = 0; i <= (1 << n); i++)
            for (int j = 0; j <= n; j++)
                dp[i][j] = INF;
        //input
        for (int i = 0; i < m; i++)
        {
            cin >> u >> v >> w;
            G[u][v] = G[v][u] = min(G[u][v], w);
        }
        //Floyd
        for (int k = 1; k <= n; k++)
            for (int i = 1; i <= n; i++)
                for (int j = 1; j <= n; j++)
                    G[i][j] = G[j][i] = min(G[i][j], G[i][k] + G[k][j]);
        //dp
        dp[1][1] = 0;
        for (int i = 1; i < (1 << n); i++)
            for (int j = 1; j <= n; j++)
                if ((i & (1 << (j - 1))) == 0)
                    for (int k = 1; k <= n; k++)
                        if (i & (1 << (k - 1)))
                            dp[i | (1 << (j - 1))][j] = min(dp[i | (1 << (j
- 1))][j], dp[i][k] + G[k][j]);
        //output
        int ans = dp[(1 << n) - 1][n] + G[1][n];
        for (int i = 1; i <= n; i++)
            ans = min(ans, dp[(1 << n) - 1][i] + G[1][i]);
        cout << ans << endl;
    }

    return 0;
}
```

5.16 最小权点基

题意：有 n 颗炸弹，位置为(Xi,Yi)，爆炸影响的半径为 Ri，引爆需要花费 Ci，如果一个炸弹引爆，在它影响范围内的炸弹都会被引爆。问：引爆所有炸弹的最小花费
题解：强连通图缩点
将形成环的点缩为 1 个点，将它的花费设为环中的最小花费，缩点完后就是一个有向无环图，答案为入度为 0 的点的花费之和

求图 G 的强连通分量，入度为 0 的强连通分量个数即为最小点基，从每个入度为 0 的强连通分量中取出权值最小的点，构成的集合即最小权点基。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int MX = 1e3 + 5;
stack<int> st;       // 存储已遍历的结点
int dfn[MX];         // 深度优先搜索访问次序
int low[MX];         // 能追溯到的最早的次序
int mark[MX];        // 检查是否在栈中(2 为在栈中，1 为已访问，且不在栈中，0 为不在)
vector<int> ver[MX]; // 获得强连通分量结果
int id[MX];          // 记录每个点在第几号强连通分量里
int idx, sz;       // DFS 序，强连通分量个数

struct Edge
{
    int v, nxt;
} E[MX * MX];
int head[MX], tot, IN[MX], c[MX], cost[MX];
double r[MX], x[MX], y[MX];

void add(int u, int v)
{
    E[tot].v = v;
    E[tot].nxt = head[u];
    head[u] = tot++;
}
void init(int n)
{
    while (!st.empty())
        st.pop();
    memset(dfn, 0, sizeof(dfn));
    memset(low, 0, sizeof(low));
    for (int i = 1; i <= n; ++i)
        ver[i].clear();
    idx = sz = 0;
    memset(head, -1, sizeof(head));
    tot = 0;
}

void tarjan(int u)
{
    mark[u] = 2;
    low[u] = dfn[u] = ++idx;
    st.push(u);
    for (int i = head[u]; ~i; i = E[i].nxt)
```

```
    {
        int v = E[i].v;
        if (dfn[v] == 0)
        {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if (mark[v] == 2)
            low[u] = min(low[u], dfn[v]);
    }
    if (low[u] == dfn[u])
    {
        ++sz;
        while (!st.empty())
        {
            int v = st.top();
            st.pop();
            mark[v] = 1;
            ver[sz].push_back(v);
            id[v] = sz;
            if (v == u)
                break;
        }
    }
}
double dis(int i, int j)
{
    return (x[i] - x[j]) * (x[i] - x[j]) + (y[i] - y[j]) * (y[i] - y[j]);
}
int main()
{
    // freopen("input.txt", "r", stdin);
    int T, n;
    scanf("%d", &T);
    for (int cas = 1; cas <= T; cas++)
    {
        scanf("%d", &n);
        init(n);
        for (int i = 1; i <= n; i++)
            scanf("%lf%lf%lf%d", &x[i], &y[i], &r[i], &c[i]);
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                if (i == j)
                    continue;
                if (dis(i, j) <= r[i] * r[i])
                    add(i, j);
            }
        }
        for (int i = 1; i <= n; i++)
            if (!dfn[i])
                tarjan(i);
        memset(IN, 0, sizeof(IN));
        for (int i = 1; i <= sz; i++)
        {
            cost[i] = 0x3f3f3f3f;
            for (int j = 0; j < ver[i].size(); j++)
```

```
            {
                int u = ver[i][j];
                cost[i] = min(cost[i], c[u]);
                for (int k = head[u]; ~k; k = E[k].nxt)
                {
                    int v = E[k].v;
                    if (id[v] == id[u])
                        continue;
                    IN[id[v]]++;
                }
            }
        }
        int ans = 0;
        for (int i = 1; i <= sz; i++)
            if (IN[i] == 0)
                ans += cost[i];
        printf("Case #%d: %d\n", cas, ans);
    }
    return 0;
}
```

5.17 判负环 (bellman_ford & spfa)

BF

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;
/*
 * 单源最短路 bellman_ford 算法，复杂度 O(VE)
 * 可以处理负边权图。
 * 可以判断是否存在负环回路。返回 true,当且仅当图中不包含从源点可达的负权回路
 * vector<Edge>E;先 E.clear()初始化，然后加入所有边
 * 点的编号从 1 开始(从 0 开始简单修改就可以了)
 */
const int INF=0x3f3f3f3f;
const int MAXN=550;
int dist[MAXN];
struct Edge
{
    int u,v;
    int cost;
    Edge(int _u=0,int _v=0,int _cost=0):u(_u),v(_v),cost(_cost){}
};
vector<Edge>E;
bool bellman_ford(int start,int n)//点的编号从 1 开始
{
    for(int i=1;i<=n;i++)dist[i]=INF;
    dist[start]=0;
    for(int i=1;i<n;i++)//最多做 n-1 次
    {
        bool flag=false;
```

```
        for(int j=0;j<E.size();j++)
        {
            int u=E[j].u;
            int v=E[j].v;
            int cost=E[j].cost;
            if(dist[v]>dist[u]+cost)
            {
                dist[v]=dist[u]+cost;
                flag=true;
            }
        }
        if(!flag)return true;//没有负环回路
    }
    for(int j=0;j<E.size();j++)
        if(dist[E[j].v]>dist[E[j].u]+E[j].cost)
            return false;//有负环回路
    return true;//没有负环回路
}

int main()
{
//    freopen("in.txt","r",stdin);
//    freopen("out.txt","w",stdout);
    int T;
    int N,M,W;
    int a,b,c;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d%d",&N,&M,&W);
        E.clear();
        while(M--)
        {
            scanf("%d%d%d",&a,&b,&c);
            E.push_back(Edge(a,b,c));
            E.push_back(Edge(b,a,c));
        }
        while(W--)
        {
            scanf("%d%d%d",&a,&b,&c);
            E.push_back(Edge(a,b,-c));
        }
        for(int i=1;i<=N;i++)
            E.push_back(Edge(N+1,i,0));
        if(!bellman_ford(N+1,N+1))printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

SPFA

```
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <algorithm>
#include <vector>
#include <queue>
```

```
using namespace std;
/*
 * 单源最短路 SPFA
 * 时间复杂度 0(kE)
 * 这个是队列实现，有时候改成栈实现会更加快，很容易修改
 * 这个复杂度是不定的
 */
const int MAXN=1010;
const int INF=0x3f3f3f3f;
struct Edge
{
    int v;
    int cost;
    Edge(int _v=0,int _cost=0):v(_v),cost(_cost){}
};
vector<Edge>E[MAXN];
void addedge(int u,int v,int w)
{
    E[u].push_back(Edge(v,w));
}
bool vis[MAXN];
int cnt[MAXN];
int dist[MAXN];
bool SPFA(int start,int n)
{
    memset(vis,false,sizeof(vis));
    for(int i=1;i<=n;i++)dist[i]=INF;
    dist[start]=0;
    vis[start]=true;
    queue<int>que;
    while(!que.empty())que.pop();
    que.push(start);
    memset(cnt,0,sizeof(cnt));
    cnt[start]=1;
    while(!que.empty())
    {
        int u=que.front();
        que.pop();
        vis[u]=false;
        for(int i=0;i<E[u].size();i++)
        {
            int v=E[u][i].v;
            if(dist[v]>dist[u]+E[u][i].cost)
            {
                dist[v]=dist[u]+E[u][i].cost;
                if(!vis[v])
                {
                    vis[v]=true;
                    que.push(v);
                    if(++cnt[v]>n)return false;
                    //有负环回路
                }
            }
        }
    }
    return true;
}
int main()
```

```
{
//    freopen("in.txt","r",stdin);
//    freopen("out.txt","w",stdout);
    int T;
    int N,M,W;
    int a,b,c;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d%d",&N,&M,&W);
        for(int i=1;i<=N+1;i++)E[i].clear();
        while(M--)
        {
            scanf("%d%d%d",&a,&b,&c);
            addedge(a,b,c);
            addedge(b,a,c);
        }
        while(W--)
        {
            scanf("%d%d%d",&a,&b,&c);
            addedge(a,b,-c);
        }
        for(int i=1;i<=N;i++)
            addedge(N+1,i,0);
        if(!SPFA(N+1,N+1))printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

6. 数论

6.1 GCD&LCM

最大公约数 GCD，最小公倍数 LCM

```
long long gcd(long long a, long long b)
{
    return b?gcd(b,a%b):a;
}

inline long long lcm(long long a, long long b)
{
    return a/gcd(a,b)*b;
}
```

6.2 快速幂

a ^ b mod n

```
ll ksm(ll a, ll b, ll n)
{
    a %= n;
    ll ans = 1;
```

```
    while (b)
    {
        if (b % 2 == 1)
            ans = ans * a % n;
        b /= 2;
        a = a * a % n;
    }
    return ans;
}
```

6.3 欧拉素数筛法

素数打表必备

```
void init_prime(int n)
{
    for (int i = 2; i <= n; i++)
    {
        if (!prime[i])
            prime[++prime[0]] = i;
        for (int j = 1; j <= prime[0] && prime[j] <= n / i; j++)
        {
            prime[prime[j] * i] = 1;
            if (i % prime[j] == 0)
                break;
        }
    }
}
```

6.4 中国剩余定理

今有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二。问物几何?

求同余方程。

```
#include <bits/stdc++.h>
using namespace std;
typedef ll long long

void gcd(ll a,ll b,ll &d,ll &x,ll &y)
{
    if(b==0)
    {
        d=a;
        x=1,y=0;
    }
    else
    {
        gcd(b,a%b,d,y,x);
        y-=(a/b)*x;
    }
}

ll China(int n,ll *m,ll *a)
{
```

```
    ll M=1,d,y,x=0;
    for(int i=0;i<n;i++) M*=m[i];
    for(int i=0;i<n;i++)
    {
        ll w=M/m[i];
        gcd(m[i],w,d,d,y);
        x=(x+y*w*a[i])%M;
    }
    return (x+M)%M;
}
ll m[15],a[15];

int main()
{
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
        scanf("%lld%lld",&m[i],&a[i]);
    printf("%lld",China(n,m,a));
}
```

6.5 矩阵快速幂

a ^ b mod n

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int num = 3;
ll mod;
struct matrix
{
    ll a[num][num];
};

matrix multiply(matrix x, matrix y)
{
    matrix temp;
    for (int i = 0; i < num; i++)
        for (int j = 0; j < num; j++)
        {
            int ans = 0;
            for (int k = 0; k < num; k++)
            {
                ans += (x.a[i][k] * y.a[k][j])%mod;
                ans = (ans+mod)%mod;
            }
            if(ans < 0)
                puts("a");
            temp.a[i][j] = ans % mod;
        }
    return temp;
}

matrix calc(matrix origin, matrix answ, int n)
```

```
{
    while (n)
    {
        if (n % 2 == 1)
            answ = multiply(origin, answ);
        origin = multiply(origin, origin);
        n /= 2;
    }
    return answ;
}

int main()
{
    // freopen("input.txt", "r", stdin);
    // freopen("2.txt", "w", stdout);
    int n, m;
    while(scanf("%d%d", &n, &m) == 2)
    {
        mod = m;
        matrix origin = {1, 2, 1,
                         1, 0, 0,
                         0, 0, 1};
        matrix answ = {0, 0, 0,
                       0, 0, 0,
                       1, 0, 0};
        answ = calc(origin, answ, n);
        printf("%lld\n", answ.a[0][0]);
    }
    return 0;
}
```

6.6 卢卡斯定理

组合数取模

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

ll quick_mod(ll a, ll b, ll m)
{
    ll ans = 1;
    a %= m;
    while (b)
    {
        if (b & 1)
            ans = ans * a % m;
        b >>= 1;
        a = a * a % m;
    }
    return ans;
}

ll getC(ll n, ll m, ll mod)
{
```

```
    if (m > n)
        return 0;
    if (m > n - m)
        m = n - m;
    ll a = 1, b = 1;
    while (m)
    {
        a = (a * n) % mod;
        b = (b * m) % mod;
        m--;
        n--;
    }
    return a * quick_mod(b, mod - 2, mod) % mod;
}

ll Lucas(ll n, ll k, ll mod)
{
    if (k == 0)
        return 1;
    return getC(n % mod, k % mod, mod) * Lucas(n / mod, k / mod, mod) %
mod;
}

int main()
{
    // freopen("input.txt", "r", stdin);
    int T;
    scanf("%d", &T);
    while (T--)
    {
        ll n, m, mod;
        scanf("%lld%lld%lld", &n, &m, &mod);
        printf("%lld\n", Lucas(n + m, m, mod));
    }
    return 0;
}
```

6.7 威尔逊定理

当 p 为素数时，那么 p 能整除（p-1）!+1。