

动态规划\LCS.cpp

```
1  /*
2  求最长公共子序列，并求出具体序列
3  */
4
5  /*
6  s串与p串组成一个二维矩阵，二维dp记录即可
7  如果相等则从左上角dp转移+1，否则取上侧和左侧最大的
8  最后输出dp[n][m]即可
9  具体序列的话直接从最末尾开始倒推就行
10 */
11 #include<bits/stdc++.h>
12 using namespace std;
13 using ll =long long;
14 const int N=1e3+10,P=1e9+7;
15
16 int dp[N][N],ans,n,m;
17 string a,b;
18
19 void solve1()//求具体序列（不唯一）
20 {
21     int i = n,j = m,op=dp[n][m];
22     string res;
23     while(op && i >= 1 && j >= 1)
24     {
25         if(a[i] == b[j])//相当于倒推，向左上走
26         {
27             res = a[i] + res;
28             --i,--j;
29         }else //否则走大的那条边
30             dp[i-1][j] >= dp[i][j-1] ? --i : --j;
31     }
32     cout << res;
33 }
34
35 void solve()//求LCS的长度
36 {
37     cin >> n >> m;
38     cin >> a >> b;
39     a = ' ' + a;
40     b = ' ' + b;
41     for(int i = 1;i <= n;++ i)
42         for(int j = 1;j <= m;++ j)
43             dp[i][j] = (a[i] == b[j] ?
44                 dp[i - 1][j - 1] + 1 : max(dp[i - 1][j],dp[i][j - 1]));
45     cout << dp[n][m] << endl;
46     solve1();
47 }
48
49 int main(){
50     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
51     int _ = 1;
```

```
52 //cin >> _;
53 while(_--)solve();
54 return 0;
55 }
```

动态规划\LIS蓝桥勇士.cpp

```
1 //2049 蓝桥勇士
2 /*
3 找到最长上升子序列的长度
4 */
5
6 /*
7 使用en数组记录当前最小的序列结尾
8 在en上二分找到第一个>=a[i]的位置x, 当x在len以内就用a[i]替换掉该位置, 否则加长en的长度
9 最后len就是en数组的长也是最后的答案
10 */
11 #include<bits/stdc++.h>
12 using namespace std;
13 using ll =long long;
14 const int N=1e6+10,P=1e9+7;
15
16 int a[N],en[N],len;
17
18 void solve()
19 {
20     int n;cin >> n;
21     for(int i = 1;i <= n;++ i)
22     {
23         cin >> a[i];
24         int x = lower_bound(en,en + len + 1,a[i]) - en;
25         if(x > len)en[++ len] = a[i];
26         else en[x] = a[i];
27     }
28     cout << len;
29 }
30
31 int main(){
32     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
33     int _ = 1;
34     //cin >> _;
35     while(_--)solve();
36     return 0;
37 }
```

基础算法\差分.txt

```
diff[i] = a[i] - a[i - 1];
```

对差分数组做前缀和便可还原数组

快速区间修改

```
diff[l] += x;
```

```
diff[r + 1] -= x;
```

//再做次前缀和便可使区间[l,r]都加上x

适用于多次修改后多次查询离线操作，边修改边查询要使用树状数组，线段树等

基础算法\二分.txt

二分前提--单调性

缩小到一个边界要那边依题意选择

1. 整数二分

```
ll Binary_Search(ll l,ll r,ll key)
{
    l = l-1,r = r+1;
    while(l + 1 != r)
    {
        ll mid = l + r >> 1;
        if(a[mid] >= key)r = mid;
        else l = mid;
    }
    if(key != a[r])return -1;
    return r;
}
```

2. 浮点二分

```
ll Binary_Search(double l,double r,double key)
{
    l = l-1,r = r+1,eps = 1e-6;
    while(r - l >= eps)
    {
        double mid = l + r >> 1;
        if(f(mid) >= key)r = mid;
        else l = mid;
    }
    return r;//返回l和r差别不大
}
```

3. 二分答案

```
ll check(ll x)
{
    return x;
}

ll Binary_Search(ll l,ll r,ll key)
{
    l = l-1,r = r+1;
    while(l + 1 != r)
    {
        ll mid = l + r >> 1;
        if(check(mid) >= key)r = mid;
        else l = mid;
    }
    if(key != check(r))return -1;
    return r;
}
```

基础算法\离散化.cpp

```
1  /*
2  将无序数据映射到可以快速通过下标访问
3  */
4
5  /*
6  先将数据加入容器，排序去重后，
7  利用getidx里在容器中二分可以访问到对应数据的下标
8  */
9 #include<iostream>
10 #include<vector>
11 #include<algorithm>
12 using namespace std;
13 vector<int>v;
14 int getidx(int x)
15 {
16     return lower_bound(v.begin(),v.end(),x)-v.begin();
17 }
18 int main(){
19     int n;cin >> n;
20     int a[n + 1];
21     for(int i = 1;i <= n;++ i)cin >> a[i],v.push_back(a[i]);
22     sort(v.begin(),v.end());
23     v.erase(unique(v.begin(),v.end()),v.end());
24     for(auto i : v)cout << i << " ";
25     cout << endl;
26     int x;cin >> x;
27     cout << getidx(x);
28 }
```

基础算法\排序.txt

```
sort(v.begin(),v.end(),compare());//algorithm  
默认升序排序，可以自定义比较函数，O(nlogn);
```

基础算法\前后缀和.txt

用于一些预处理

```
prefix[i] = prefix[i - 1] + a[i];
suffix[i] = suffix[i + 1] + a[i];
```

库函数\其他库函数.txt

```
memset(v,0,sizeof(v));//<cstring>
swap(T &a,T &b);//<algorithm>
reverse(v.begin(),v.end());//<algorithm>
unique(v.begin(),v.end());//<algorithm>
先排序再去重，返回的是有序段的末尾+1，左闭右开区间
erase(unique(v.begin(),v.end()),v.end())
```

库函数\全排列.txt

1.

```
next_permutation(v.begin(),v.end());
```

生成下一个排列，存在下一个排列返回true，否则返回false

```
prev_permutation(v.begin(),v.end());
```

生成上一个排列，存在上一个排列返回true，否则返回false

通常放在while里搭配使用

库函数\一些容器以及用法.txt

1.
pair<T,T>v;//<utility>
v.first,v.second;
//使用sort对其排序时first优先second其次升序排序
2.
vector<T>v;//<vector>
常用函数：
v.push_back();
v.pop_back();要保证v非空
排序去重：
sort(v.begin(),v.end());
v.erase(unique(v.begin(),v.end()),v.end());
3.
stack<T>v;//<stack>
常用函数：
v.push(x);
v.pop();
v.top();
v.empty();
v.size();
4.
queue<T>v;//<queue>
常用函数：
v.push(x);
v.pop();
v.front();
v.back();
v.empty();
v.size();
priority_queue<T>v;//<queue>
常用函数：
v.push(x);
v.pop();
v.top();
v.empty();
v.size();
5.
deque<T>v;//<deque>
常用函数：
v.push_back(x);
v.push_front(x);
v.pop_back();
v.pop_front();
v.front();
v.back();
v.empty();
v.size();
v.clear();
v.insert(pos,x);
v.erase(pos);
erase(first,last);
6.
map<T,T>v;//<map>
v.insert(k,v);
v.erase(k);
v.find(k);
v.conunt(k);

```

v.size();
v.lower_bound(x); // 返回第一个大于等于x的键的位置
v.upper_bound(x); // 返回第一个大于x的键的位置
7.
set//默认升序
greater int //降序,非正常成员时需要自定义比较函数
multiset<T>v;//多重集合可以有重复元素同样是有序的
v.insert(x);
v.count(x);
v.find(x); //找到x的迭代器, 没找到便返回v.end();

v.erase(x):
1.传入迭代器删除某个值v.erase(v.find(x));
2.传入某个值删除所有相同的值v.erase(x);
3.传入一个范围把这个范围的值删除v.erase(v.lower_bound(2),v.upper_bound(3));
//删除[2,3]的所有值

v.size();
v.empty();
v.begin();
v.end(); //末尾元素的下一个
v.rbegin(); //逆向迭代器的第一个也就是末尾元素
v.rend();
v.swap(a,b);
v.lower_bound(x); //返回第一个大于等于x的元素的迭代器
v.upper_bound(x); //返回第一个大于x的元素的迭代器

```

1. 去重与有序存储

需要快速去重并保持元素有序时:

```

vector<int> vec = {5, 2, 2, 3, 3, 1};
set<int>uniqueSorted(vec.begin(), vec.end()); // 自动去重并排序
// 结果: {1, 2, 3, 5}

```

2. 快速查找

利用 $O(\log n)$ 的查找效率检查元素是否存在:

```
if (mySet.count(42) > 0) // count返回0或1
```

3. 集合运算

求交集、并集、差集等:

```

#include <iostream>
#include <set>
#include <algorithm>
#include <iterator>
using namespace std;

// 通用打印函数
template <typename Container>
void print(const string& title, const Container& c) {
    cout << title << ": ";
    for (auto& x : c)
        cout << x << " ";
    cout << "\n";
}

int main() {
    set<int> a = {1, 2, 3, 4};
    set<int> b = {3, 4, 5, 6};
    set<int> res;           // 用来存放中间结果

    // 交集

```

```

    set_intersection(
        a.begin(), a.end(),
        b.begin(), b.end(),
        inserter(res, res.begin()))
);
print("Intersection", res);
res.clear();

// 并集
set_union(
    a.begin(), a.end(),
    b.begin(), b.end(),
    inserter(res, res.begin()))
);
print("Union", res);
res.clear();

// 差集 a - b
set_difference(
    a.begin(), a.end(),
    b.begin(), b.end(),
    inserter(res, res.begin()))
);
print("Difference (a - b)", res);
res.clear();

// 对称差集
set_symmetric_difference(
    a.begin(), a.end(),
    b.begin(), b.end(),
    inserter(res, res.begin()))
);
print("Symmetric Difference", res);

return 0;
}

```

4. 维护动态有序数据

如排行榜、实时数据监控等需要频繁插入且保持顺序的场景。

8.

`unordered_set<T>;`
无序集合复杂度更优秀

9. 双向链表

```

list<T>v;//<list>
v.push_back();
v.push_front();
v.pop_back();
v.pop_front();
v.size();
v.empty();
v.front();
v.back();
v.reverse(v.begin(),v.end());

```


库函数\最值函数.txt

```
1.//<algorithm>
min(a,b),max(a,b);
2.
min_element(v.begin(),v.end());
max_element(v.begin(),v.end());
返回的是容器中最值的地址,O(n);
3.
nth_element(v.begin(),v.begin()+k,v.end());
排序使k位置为正确的，且前面的都比他小，后面的都大，O(n);
```

其他\快读.txt

```
int ri()
{
    int x=0;
    char c=getchar(),f=1;
    while(c<'0'||c>'9'){
        if(c=='-')
            f=-f;
        c=getchar();
    }
    while(c<='9'&&c>='0')
        x=(x<<1)+(x<<3)+(c^48),c=getchar();
    return x*f;
}
//一次读入一个，直接调用，如int a = ri();
```

```

1 //P2294 [HNOI2005] 狡猾的商人
2 /*
3 对w组数据，n为节点数，m为信息条数，w为两节点之间的总和的关系，判断是否有信息是矛盾的
4 */
5
6 /*
7 直接带权并查集，对于每条信息先看是否可以在已有的信息里找到（两者根相同）
8 如果可以就判断一下dis[u] - dis[v] == w,
9 不相等即矛盾了，无法判断的话之间加入（两根不同）
10
11 细节：
12 对于路径压缩时先更新根的权值再更新节点的权
13 合并两根的时候dis[rx] = dis[r] - dis[l] + v (通过画图向量的合成或者举例来理解)
14 查询就是dis[l] - dis[r]
15 */
16 #include<bits/stdc++.h>
17 using namespace std;
18 using ll = long long;
19 const ll inf = 2e18;
20 const int N=1e5+10,P=1e9+7;
21
22 int pre[N];
23 ll dis[N];
24
25 int root(int x)
26 {
27     if(pre[x] == x) return x;
28     int t = pre[x];
29     pre[x] = root(pre[x]);
30     dis[x] += dis[t];
31     return pre[x];
32 }
33
34 void merge(int l,int r,ll v)
35 {
36     int lr = root(l),rr = root(r);
37     pre[lr] = rr;
38     dis[lr] = dis[r] + v - dis[l];
39 }
40
41 bool check(int l,int r,ll v)
42 {
43     int lr = root(l),rr = root(r);
44     if(lr == rr && dis[l] - dis[r] != v) return 1;
45     //只有在两者位于一个集合时才可以判断是否与之前的信息矛盾否则无法判断
46     return 0;
47 }
48
49 void solve()
50 {
51     int n,m;cin >> n >> m;
52     for(int i = 1;i <= n + 1;++ i)pre[i] = i,dis[i] = 0;
53 }
```

```
54 int flag = 0;
55 while(m --)
56 {
57     int l,r,ll,v;cin >> l >> r >> v;
58     flag |= check(l,r + 1,v);
59     if(flag)continue;
60     merge(l,r + 1,v);
61 }
62
63 if(flag)cout << "false" << endl;
64 else cout << "true" << endl;
65 }
66
67 int main(){
68     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
69     int _ = 1;
70     cin >> _;
71     while(_--)solve();
72     return 0;
73 }
```

数据结构\并查集\带权并查集\食物链.cpp

```
1 //P2024 [NOI2001] 食物链
2 /*
3 n个动物, k个信息, 1代表l和r是同类, 2代表l吃r, 求假话总数
4 假话: 1.与前面冲突 2.1, r不在n以内 3.当op == 2时 l和r相等
5 */
6
7 /*
8 dis含义: 0表示同类, 1表示吃根节点的动物, 2表示被根节点吃
9 合并两根的时候dis[rx] = (dis[r] - dis[l] + v + 3) % 3 (通过画图向量的合成或者举例来理解)
10 查询就是(dis[l] - dis[r] + 3) % 3
11 mod3是为了使权值落在0到2之间
12 */
13 #include<bits/stdc++.h>
14 using namespace std;
15 using ll = long long;
16 const ll inf = 2e18;
17 const int N=1e5+10,P=1e9+7;
18
19 int ans,pre[N],dis[N];
20
21 int root(int x)
22 {
23     if(pre[x] == x) return x;
24     else {
25         int t = pre[x];
26         pre[x] = root(pre[x]); //写并查集注意这里是pre[x]
27         dis[x] = (dis[x] + dis[t]) % 3;
28         return pre[x];
29     }
30 }
31
32 bool check(int op,int l,int r)
33 {
34     int lr = root(l),rr = root(r);
35     if(lr != rr || op == (dis[l] - dis[r] + 3) % 3) return 1;
36     //不在一个集合或者关系与信息符合时可以合并否则信息为假
37     return 0;
38 }
39
40 void merge(int l,int r,int v)
41 {
42     int lr = root(l),rr = root(r);
43     pre[lr] = rr;
44     dis[lr] = (dis[r] + v - dis[l] + 3) % 3; //画图
45 }
46
47 void solve()
48 {
49     int n,k;cin >> n >> k;
50     for(int i = 1;i <= n;++ i)pre[i] = i;
51 }
```

```
52 while(k --)
53 {
54     int op,l,r;cin >> op >> l >> r;
55
56     //非法状态处理
57     if(op == 2 && l == r || l > n || r > n || !check(op - 1,l,r))
58     {
59         ++ ans;
60         continue;
61     }
62
63     merge(l,r,op - 1);
64 }
65
66 cout << ans;
67 }
68
69 int main(){
70     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
71     int _ = 1;
72     //cin >> _;
73     while(_--)solve();
74     return 0;
75 }
```

数据结构\并查集\带权并查集\推导部分和.cpp

```
1 //P8779 [蓝桥杯 2022 省 A] 推导部分和
2 /*
3 给出区间和查找区间和，找不到输出未知
4 */
5
6 /*
7 经典带权并查集，维护一下关系即可
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll = long long;
12 const ll inf = 2e18;
13 const int N=1e5+10,P=1e9+7;
14
15 int pre[N];
16 ll dis[N];
17
18 int root(int x)
19 {
20     if(pre[x] == x) return x;
21     int t = pre[x];
22     pre[x] = root(pre[x]); //先找根再回退更新dis
23     dis[x] += dis[t]; //加上更新后的父亲到根节点的距离
24     return pre[x];
25 }
26
27 void merge(int l,int r,ll v)
28 {
29     int lr = root(l),rr = root(r);
30     pre[lr] = rr;
31     dis[lr] = dis[r] + v - dis[l]; //更新lr到rr的距离画个图就很任意理解
32 }
33
34 void query(int l,int r)
35 {
36     int lr = root(l),rr = root(r);
37     if(lr != rr)
38         cout << "UNKNOWN" << endl;
39     else cout << dis[l] - dis[r] << endl; //只有当端点在一个集合时才可以得出距离关系
40 }
41
42 void solve()
43 {
44     int n,m,q;cin >> n >> m >> q;
45     for(int i = 1;i <= n + 1;++ i)pre[i] = i;
46
47     while(m --)
48     {
49         int l,r,ll v;cin >> l >> r >> v;
50         merge(l,r + 1,v);
51     }
52 }
```

```
52
53     while(q --)
54     {
55         int l,r;cin >> l >> r;
56         query(l,r + 1);
57     }
58 }
59
60 int main(){
61     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
62     int _ = 1;
63     //cin >> _;
64     while(_--)solve();
65     return 0;
66 }
```

数据结构\并查集\带权并查集\维护异或关系.cpp

```
1 //UVA12232 Exclusive-OR
2 /*
3 每组给定0 ~ n-1节点，告知q条信息
4 对于 I p v 指 p节点值为 v
5 对于 I p q v 指 p节点与 q节点异或为 v
6 对于 Q k p1 ~ pk 指查询这 k个节点异或和
7 */
8
9 /*
10 对于多个节点的关系维护想到带权并查集，由于当知道一个集合里任意一个值可以推出根的值从而退出该集合所有的值
11 为了记录这些已知的节点，我们设置一个虚拟源点该点可设置为 n他的值设置成 0这样就可以判断一个值是否已知
12 细节：对于输入可以使用 getline得到信息字符串再使用 s.c_str()转成 c风格字符串用于sscanf读取得
13 合并的时候要把一直把源点作为根
14 查询的时候若除去已知的点之外存在有同一个根上的数量为奇数则该查询不可知
15 输出记得对上题目要求
16 */
17
18 //血的教训：t 和 root 里一开始的 t 重复了，调了一个小时才发现，改 temp 后就过了。
19 #include<bits/stdc++.h>
20 using namespace std;
21 using ll = long long;
22 const ll inf = 2e18;
23 const int N=2e4+10,P=1e9+7;
24
25 int pre[N],t,dis[N],out;//pre 记录父亲， dis 记录点与根节点值的异或值。
26 //out是当发现冲突时用于跳出除读入外操作的标记。
27
28 int root(int x)
29 {
30     if(pre[x] == x) return x;
31     int temp = pre[x];
32     pre[x] = root(temp);//先递归更新父亲的 dis 的值。
33     dis[x] ^= dis[temp];//再回溯更新 x 的 dis 值可以通过画图得出这个规律。
34     return pre[x];
35 }
36
37 int read(string &s,int &a,int &b,int &c)
38 {
39     getline(cin,s);
40     return sscanf(s.c_str(),"%d %d %d",&a,&b,&c); //通过 sscanf 的返回值来处理题目的输入
41 }
42
43 void init(int n)
44 {
45     out = 0;
46     cout << "Case " << ++t << ":\n";
47     for(int i = 0;i <= n;++ i)pre[i] = i,dis[i] = 0;
48 }
49
```

```

50 int merge(int x,int y,int v,int n)
51 {
52     int xr = root(x),yr = root(y);
53     if(xr == yr) return (v == (dis[x] ^ dis[y]));
54
55     if(xr == n)swap(xr,yr); //设n号节点为虚拟源点，意指已经推断出值的点的集合
56     pre[xr] = yr;
57     dis[xr] = dis[x] ^ dis[y] ^ v;
58     return 1;
59 }
60
61 void query(int k,int n)
62 {
63     int res = 0;
64     map<int,int>mp;
65     //使用map记录各个非n的根节点的数量当某个根的数为偶数时计算的时候就会被约去，此时这个根的值不会影响结果
66     mp.clear();
67     for(int i = 1;i <= k;++ i)
68     {
69         int x;cin >> x;
70         if(out)continue;
71         int xr = root(x);
72         if(xr != n)++ mp[xr];
73         res ^= dis[x];
74     }
75
76     if(out)return;
77
78     for(auto x : mp)
79     {
80         if(x.second & 1)//当某个根的数为奇数时那么它的值会影响结果且该根不在n节点上所以我们无从知晓res。
81         {
82             cout << "I don't know.\n";
83             return;
84         }
85     }
86     cout << res << "\n";
87 }
88
89 void solve()
90 {
91     int n,q;cin >> n >> q;
92     if(n == 0 && q == 0) return;
93     init(n);
94     int cnt = 0;//记录有几条声明。
95     while(q --)
96     {
97         char op;cin >> op;
98         if(op == 'I')
99         {
100             ++ cnt;
101             string s;

```

```
102     int a,b,c;
103     int _ = read(s,a,b,c); //通过read的返回值确定I的操作类型。
104     if(out)continue;
105     int fg;
106
107     if(_ == 2) fg = merge(a,n,b,n);
108     //因为当一个集合里任意知道一个元素的值可以通过dis推断出根节点的值再由根节点推断出集合
109     //内所有元素的值,
110     //所以应该将该点集合内所有元素全部合到 n 节点上。
111     else fg = merge(a,b,c,n);
112
113     if(!fg)out = cnt;
114     }else {
115         int k;cin >> k;
116         query(k,n);
117     }
118
119     if(out)cout << "The first " << out << " facts are conflicting.\n";
120     cout << "\n";
121
122     solve();
123 }
124
125 int main(){
126     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
127     int _ = 1;
128     //cin >> _;
129     while(_--)solve();
130     return 0;
131 }
132 }
```

数据结构\并查集\带权并查集\银河英雄传说.cpp

```
1 //P1196 [NOI2002] 银河英雄传说、
2 /*
3 t条信息，M i j 是指将 i接到 j的队尾，C i j 指查询 i到 j之间有多少战舰
4 */
5
6 /*
7 带权并查集维护节点之间关系
8 细节：合并两个集合的时候要记录 sz，因为是把一方放在另一方队尾，所以 dis[lr] = sz[rr]
9 */
10 #include<bits/stdc++.h>
11 using namespace std;
12 using ll = long long;
13 const ll inf = 2e18;
14 const int N=1e5+10,P=1e9+7;
15
16 int pre[N],dis[N],sz[N];
17
18 int root(int x)
19 {
20     if(pre[x] == x) return x;
21     else {
22         int t = pre[x];
23         pre[x] = root(pre[x]); //先更新父亲
24         dis[x] += dis[t]; //常规更新
25         return pre[x];
26     }
27 }
28
29 void merge(int l,int r)
30 {
31     int lr = root(l),rr = root(r);
32     if(lr == rr) return;
33     pre[lr] = rr;
34     dis[lr] = sz[rr]; //根作为队头 dis[rr] = 0, 所已合并完之后 dis[lr] = sz[rr]
35     sz[rr] += sz[lr]; //合并sz
36 }
37
38 void query(int l,int r)
39 {
40     int lr = root(l),rr = root(r);
41     if(lr != rr) cout << -1 << endl;
42     else cout << max(0,abs(dis[r] - dis[l]) - 1) << endl;
43 }
44
45 void solve()
46 {
47     for(int i = 1;i <= N;++ i)pre[i] = i,sz[i] = 1;
48     int t;cin >> t;
49     while(t --)
50     {
51         char op;int l,r;cin >> op >> l >> r;
```

```
52     if(op == 'M')merge(l,r);
53     else query(l,r);
54 }
55 }
56
57 int main(){
58     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
59     int _ = 1;
60     //cin >> _;
61     while(_--)solve();
62     return 0;
63 }
```

数据结构\并查集\普通并查集分类处理\关押罪犯.cpp

```
1 //P1525 [NOIP 2010 提高组] 关押罪犯
2 /*
3 给定1 ~ n节点，不同节点之间有个边权 c，求解分成两个集合使得两图中最大的边权值最小的值
4 */
5
6 /*
7 类似于二分图，这里采用贪心+并查集解决，使用 enemy记录每个人的一个代表性敌人
8 将l连到r的代表性敌人上，r连到l的代表性敌人上，最后只要不存在l和r在同一个集合里的信息
9 则可以不发生任何冲突，不然在第一条冲突的信息即为最小的最大边权
10 */
11 //普通并查集做分类处理
12 #include<bits/stdc++.h>
13 using namespace std;
14 using ll = long long;
15 const ll inf = 2e18;
16 const int N=1e5+10,P=1e9+7;
17
18 int pre[N],enemy[N];
19
20 struct people{
21     int a,b,c;
22
23     bool operator < (const people & u) const
24     {
25         return c < u.c;
26     }
27 }p[N];//按罪犯之间的仇恨值排序
28
29 int root(int x)
30 {
31     return pre[x] = pre[x] == x ? x : root(pre[x]);
32 }
33
34 void solve()
35 {
36     int n,m;cin >> n >> m;
37
38     for(int i = 1;i <= n;++ i)pre[i] = i;
39
40     for(int i = 0;i < m;++ i)
41     {
42         int l,r,v;cin >> l >> r >> v;
43         p[i] = {l,r,v};
44     }
45
46     sort(p,p + m);
47
48     for(int i = m - 1;i >= 0;-- i)//贪心的先选仇恨大的
49     {
50         int l = p[i].a,r = p[i].b;
51         int lr = root(l),rr = root(r);
```

```
52
53     if(lr == rr)//当l,r不可避免的在一个集合中该值为答案
54     {
55         cout << p[i].c;
56         return;
57     }
58
59 //记录l,r的敌人，将各组l,r的敌人合并到一个区间
60 if(enemy[l])
61 {
62     int t = enemy[l];
63     pre[rr] = root(t);
64 }else enemy[l] = r;
65
66 if(enemy[r])
67 {
68     int t = enemy[r];
69     pre[lr] = root(t);
70 }else enemy[r] = l;
71 }
72 cout << 0;
73 }
74
75 int main(){
76     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
77     int _ = 1;
78     //cin >> _;
79     while(_--)solve();
80     return 0;
81 }
```

数据结构\并查集\普通并查集分类处理\团伙.cpp

```
1 //P1892 [BalticOI 2003] 团伙
2 /*
3 n个节点, m条关系, E 表示 u和 v是敌对关系, F 则是朋友关系
4 具有如下传递关系: 朋友的朋友是朋友, 敌人的敌人也是朋友, 求有几个不交集合
5 */
6
7 /*
8 enemy记录代表性敌人, 对于操作 E, 将 l和 enemy[r]连接, r和 enemy[1]连接就行
9 细节: 注意若 lr == rr时上面的操作把 rr合并到 enemy[1]上 root(lr)被改变, 所以要更新一下 lr
10 */
11
12 //普通并查集分类处理
13 #include<bits/stdc++.h>
14 using namespace std;
15 using ll = long long;
16 const ll inf = 2e18;
17 const int N=1e3+10,P=1e9+7;
18
19 int pre[N],enemy[N];
20
21 int root(int x)
22 {
23     return pre[x] = pre[x] == x ? x : root(pre[x]);
24 }
25
26 void solve()
27 {
28     int n;cin >> n;
29     for(int i = 1;i <= n;++ i)pre[i] = i;
30
31     int _;cin >> _;
32     while(_ --)
33     {
34         char op;int p,q;cin >> op >> p >> q;
35         int lr = root(p),rr = root(q);
36         if(op == 'F')pre[lr] = rr;
37         else {
38             //分别将p,q与对方的敌人合并
39             if(enemy[p])pre[root(rr)] = root(enemy[p]);
40             else enemy[p] = q;
41             //注意这里如果lr==rr的话lr就被更新了, 不等于root(lr)了, 所以下面不能直接使用pre[lr] =
42             //root(enemy[q])
43
44             if(enemy[q])pre[root(lr)] = root(enemy[q]);
45             else enemy[q] = p;
46         }
47     }
48
49     int ans = 0;
50     for(int i = 1;i <= n;++ i)if(pre[i] == i)++ ans;//当节点指向自己那他便是根可以省一个vis
51     cout << ans;
52 }
```

```
52
53 int main(){
54     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
55     int _ = 1;
56     //cin >> _;
57     while(_--)solve();
58     return 0;
59 }
```

数据结构\单调数据结构\单调队列手搓买蛋糕.cpp

```
1 ///////////////////////////////////////////////////////////////////蓝桥 3707 买蛋糕
2 /*
3  给定n个数，求连续的区间最大值最小值差值小于等于x的概率
4 */
5
6 /*
7  给大小为k的区间按顺序标号，跑两遍单调队列得到各区间的最大最小值再统计符合要求的区间然后除以总区
8 间个数即可
9 */
10 //复杂度很优秀
11 //这类问题也可以使用multiset(多重集合搞定)
12 #include<bits/stdc++.h>
13 using namespace std;
14 using ll =long long;
15 const int N=1e5+10,P=998244353;
16
17 int mi[N],mx[N],a[N],dq[N],ans,hh,tt;
18
19 ll qmi(ll a,ll b)
20 {
21     ll res = 1;
22     while(b)
23     {
24         if(b & 1)res = res * a % P;
25         a = a * a % P;b >>= 1;
26     }
27     return res;
28 }
29
30 ll inv(ll x){return qmi(x,P - 2);}
31
32 void solve()
33 {
34     int n,k,x;cin >> n >> k >> x;
35     for(int i = 1;i <= n;++ i)cin >> a[i];
36     //单增
37     dq[tt] = 1;
38     for(int i = 2;i <= n;++ i)
39     {
40         if(i >= k && dq[hh] < i - k + 1)++ hh;
41         while(hh <= tt && a[i] <= a[dq[tt]])-- tt;
42         dq[++ tt] = i;
43         if(i >= k)mi[i - k + 1] = a[dq[hh]];
44     }
45     //单减
46     hh = tt = 0;
47     dq[tt] = 1;
48     for(int i = 2;i <= n;++ i)
49     {
50         if(i >= k && dq[hh] < i - k + 1)++ hh;
51         while(hh <= tt && a[i] >= a[dq[tt]])-- tt;
52         dq[++ tt] = i;
53     }
54 }
```

```
52     if(i >= k)mx[i - k + 1] = a[dq[hh]];
53 }
54 int en = n - k + 1;
55 for(int i = 1;i <= en;++ i)if(mx[i] - mi[i] <= x)++ ans;
56 cout << ans * inv(en) % P;
57 }
58
59 int main(){
60     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
61     int _ = 1;
62     //cin >> _;
63     while(_--)solve();
64     return 0;
65 }
```

数据结构\单调数据结构\单调队列stl买蛋糕.cpp

```
1 //蓝桥 3707 买蛋糕
2 #include<bits/stdc++.h>
3 using namespace std;
4 using ll =long long;
5 const int N=1e5+10,P=998244353;
6
7 int mi[N],mx[N],ans,a[N];
8 deque<int>dq;
9
10 ll qmi(ll a,ll b)
11 {
12     ll res = 1;
13     while(b)
14     {
15         if(b & 1)res = res * a % P;
16         a = a * a % P;b >>= 1;
17     }
18     return res;
19 }
20
21 ll inv(ll x){return qmi(x,P - 2);}
22
23 void solve()
24 {
25     int n,k,x;cin >> n >> k >> x;
26     for(int i = 1;i <= n;++ i)cin >> a[i];
27     //单增
28     dq.push_back(1);
29     for(int i = 2;i <= n;++ i)
30     {
31         if(i >= k && dq.front() < i -k + 1)dq.pop_front();
32         while(dq.size() && a[i] <= a[dq.back()])dq.pop_back();
33         dq.push_back(i);
34         if(i >= k)mi[i - k + 1] = a[dq.front()];
35     }
36     dq.clear();
37     dq.push_back(1);
38     for(int i = 2;i <= n;++ i)
39     {
40         if(i >= k && dq.front() < i -k + 1)dq.pop_front();
41         while(dq.size() && a[i] >= a[dq.back()])dq.pop_back();
42         dq.push_back(i);
43         if(i >= k)mx[i - k + 1] = a[dq.front()];
44     }
45     int en = n - k + 1;
46     for(int i = 1;i <= en;++ i)if(mx[i] - mi[i] <= x)++ ans;
47     cout << ans * inv(en) % P;
48 }
49
50 int main(){
51     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
```

```
52 |     int _ = 1;
53 |     //cin >> _;
54 |     while(_--)solve();
55 |     return 0;
56 | }
```

数据结构\单调数据结构\单调栈百亿富翁.cpp

```
1 //蓝桥 1142 百亿富翁
2 /*
3 给定n个数，求每个数的左侧第一个比它大的，右侧第一个比它大的编号，不存在则为-1
4 */
5
6 /*
7 分从两侧跑两次单调栈，每次记录要压栈的左侧的编号，栈底放-1
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll =long long;
12 const int N=7e5+10,P=1e9+7;
13
14 int a[N],l[N],R[N],stk[N] = {-1},top;//栈底存入-1
15
16 void solve()
17 {
18     int n;cin >> n;
19     for(int i = 1;i <= n;++ i)cin >> a[i];
20     for(int i = 1;i <= n;++ i)
21     {
22         while(a[stk[top]] <= a[i] && top)-- top;//寻找更优直到栈底
23         l[i] = stk[top];//记录最优
24         stk[++ top] = i;//压栈
25     }
26     for(int j = 1;j <= n;++ j)cout << l[j] << " ";
27     cout << endl;
28     top = 0;//清空栈
29     for(int i = n;i >= 1;-- i)
30     {
31         while(a[stk[top]] <= a[i] && top)-- top;
32         R[i] = stk[top];
33         stk[++ top] = i;
34     }
35     for(int j = 1;j <= n;++ j)cout << R[j] << " ";
36 }
37
38 int main(){
39     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
40     int _ = 1;
41     //cin >> _;
42     while(_--)solve();
43     return 0;
44 }
```

数据结构\基础树上问题\树上差分\倍增差分砍树.cpp

```
1 //蓝桥 3517 砍树
2 /*
3 给定n个节点，m条边，选择一个编号使得m条边都不连通，输出编号最大的
4 */
5
6 /*
7 考虑树上边差分，（对于u到v以v作为该边的编号），对m条边的两端节点差分，再做前缀和得到权值，找到
8 权值为m的再比一下编号即可
9 细节：对差分复原是自下而上跑，普通树上前缀和是自上而下跑
10 */
11 #include<bits/stdc++.h>
12 using namespace std;
13 using ll =long long;
14 const int N=1e5+10,P=1e9+7;
15
16 vector<int>vec[N];
17 map<int,int>num[N];
18
19 int dep[N],st[N][17],diff[N],prefix[N];
20
21 void dfs(int x,int fa)
22 {
23     prefix[x] = diff[x];
24     for(auto y : vec[x])
25     {
26         if(y == fa)continue;
27         dfs(y,x);
28         prefix[x] += prefix[y];
29     }
30 }
31
32 void init(int x,int fa)
33 {
34     dep[x] = dep[fa] + 1;
35     st[x][0] = fa;
36     for(int i = 1;i <= 16;++ i)
37         st[x][i] = st[st[x][i - 1]][i - 1];
38
39     for(auto y : vec[x])
40     {
41         if(y == fa)continue;
42         init(y,x);
43     }
44 }
45
46 int lca(int x,int y)
47 {
48     if(dep[x] < dep[y])swap(x,y);
49
50     for(int i = 16;i >= 0;-- i)if(dep[st[x][i]] <= dep[y]])x = st[x][i];
51     if(x == y)return x;
```

```

52
53     for(int i = 16;i >= 0;-- i)if(st[x][i] != st[y][i])x = st[x][i],y = st[y][i];
54     return st[x][0];
55 }
56
57 void solve()
58 {
59     int n,m;cin >> n >> m;
60
61     for(int i = 1;i < n;++ i)
62     {
63         int u,v;cin >> u >> v;
64         vec[u].push_back(v);
65         vec[v].push_back(u);
66         num[u][v] = num[v][u] = i;
67     }
68
69     init(1,0);
70
71     for(int i = 1;i <= m;++ i)
72     {
73         int u,v;cin >> u >> v;
74         ++ diff[u],++ diff[v],diff[lca(u,v)] -= 2;
75     }
76
77     dfs(1,0);
78
79     int ans = -1;
80     for(int i = 1;i <= n;++ i)
81     {
82         if(prefix[i] == m)
83             ans = max(ans,num[st[i][0]][i]);
84
85         cout << ans;
86     }
87
88     int main(){
89         ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
90         int _ = 1;
91         //cin >> _;
92         while(_--)solve();
93         return 0;
94     }

```

数据结构\基础树上问题\树上差分\倍增差分 Max_Flow_P.cpp

```
1 //P3128 [USACO15DEC] Max Flow P
2 /*
3 n个节点n-1条边，表示u到v路径上都加一，求节点最大权值
4 */
5
6 /*
7 树上点差分，思路同上
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll = long long;
12 const int N=1e5+10,P=1e9+7;
13
14 vector<int>tree[N];
15
16 int diff[N],st[N][16],dep[N],prefix[N];
17
18 void dfs(int x,int fa)
19 {
20     prefix[x] = diff[x];
21     for(auto y : tree[x])
22     {
23         if(y == fa)continue;
24         dfs(y,x);
25         prefix[x] += prefix[y];
26     }
27 }
28
29 void init(int x,int fa)
30 {
31     dep[x] = dep[fa] + 1;
32
33     st[x][0] = fa;
34     for(int i = 1;i <= 15;++ i)
35         st[x][i] = st[st[x][i - 1]][i - 1];
36
37     for(auto y : tree[x])
38     {
39         if(y == fa)continue;
40         init(y,x);
41     }
42 }
43
44 int lca(int x,int y)
45 {
46     if(dep[x] < dep[y])swap(x,y);
47
48     for(int i = 15;i >= 0;-- i)if(dep[st[x][i]] >= dep[y])x = st[x][i];
49     if(x == y)return x;
50
51     for(int i = 15;i >= 0;-- i)if(st[x][i] != st[y][i])x = st[x][i],y = st[y][i];
```

```

52     return st[x][0];
53 }
54
55 void solve()
56 {
57     int n,k;cin >> n >> k;
58     for(int i = 1;i < n;++ i)
59     {
60         int x,y;cin >> x >> y;
61         tree[x].push_back(y);
62         tree[y].push_back(x);
63     }
64     init(1,0);
65     for(int i = 1;i <= k;++ i)
66     {
67         int u,v;cin >> u >> v;
68         //cout << lca(u,v) << endl;
69         int c = lca(u,v);
70         ++ diff[u],++ diff[v],-- diff[c],-- diff[st[c][0]];
71         //cout << c << " " << st[c][0] << endl;
72     }
73     dfs(1,0);
74     int ans = 0;
75     for(int i = 1;i <= n;++ i)ans = max(ans,prefix[i]);
76     cout << ans;
77 }
78
79 int main(){
80     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
81     int _ = 1;
82     //cin >> _;
83     while(_--)solve();
84     return 0;
85 }
```

数据结构\基础树上问题\树上差分\Tarjan差分 Max_Flow_P.cpp

```
1 ///////////////////////////////////////////////////////////////////P3128 [USACO15DEC] Max Flow P
2 /*
3 离线求法
4 */
5 #include<bits/stdc++.h>
6 using namespace std;
7 using ll = long long;
8 const int N=1e5+10,P=1e9+7;
9
10 map<int,int>lca[N];
11 vector<pair<int,int>>ques;
12 vector<int>vec[N];
13 vector<int>tree[N];
14 bitset<N>vis;
15
16 int diff[N],pre[N],prefix[N],f[N];
17
18 int root(int x){return pre[x] == pre[x] == x ? x : root(pre[x]);}
19
20 void dfs(int x,int fa)
21 {
22     prefix[x] = diff[x];
23     for(auto y : tree[x])
24     {
25         if(y == fa)continue;
26         dfs(y,x);
27         prefix[x] += prefix[y];//自下向上dp
28     }
29 }
30
31 void Tarjan(int x,int fa)
32 {
33     f[x] = fa;
34     vis[x] = 1;
35     for(auto y : tree[x])
36     {
37         if(y == fa)continue;
38         Tarjan(y,x);
39         pre[y] = x;
40     }
41     for(auto y : vec[x])if(vis[y])lca[x][y] = lca[y][x] = root(y);//注意这里是当y已经走过时取y的根为lca
42 }
43
44 void solve()
45 {
46     int n,k;cin >> n >> k;
47
48     for(int i = 1;i < n;++ i)
49     {
50         int x,y;cin >> x >> y;
51         tree[x].push_back(y);
```

```
52     tree[y].push_back(x);
53 }
54
55 for(int i = 1;i <= k;++ i)
56 {
57     int u,v;cin >> u >> v;
58     vec[u].push_back(v);
59     vec[v].push_back(u);//存入问题
60     ques.push_back({u,v});//存入需处理的边点
61 }
62
63 for(int i = 1;i <= n;++ i)pre[i] = i;
64 Tarjan(1,0);
65
66 for(auto x : ques)
67 {
68     int u = x.first,v = x.second;
69     int z = lca[u][v];
70     ++ diff[u],++ diff[v],-- diff[z],-- diff[f[z]];
71     //这里处理的是点权，处理边权的话：++ diff[u],++ diff[v],diff[z] -= 2，画个图便很好理解
72 }
73
74 dfs(1,0);//前缀和复原
75
76 int ans = 0;
77 for(int i = 1;i <= n;++ i)ans = max(ans,prefix[i]);
78 cout << ans;
79 }
80
81 int main(){
82     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
83     int _ = 1;
84     //cin >> _;
85     while(_--)solve();
86     return 0;
87 }
```

数据结构\基础树上问题\树上前缀和\Tarjan前缀和.cpp

```
1 //蓝桥 19707 零食采购
2 /*
3 n个节点n-1条边，每个点有一个属性，q次查询u到v路径上有多少种不同的属性
4 */
5
6 /*
7 注意到属性比较少，考虑对不同属性自上而下跑前缀和，再利用lca实现路径查询
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll =long long;
12 const int N=1e5+10,P=1e9+7;
13
14 vector<int>vec[N];
15 map<int,int>lca[N];
16 vector<pair<int,int>>Q;
17 vector<int>q[N];
18 bitset<N>vis;
19
20 int f[N],pre[N],a[N],prefix[N][21];
21
22 int root(int x){return pre[x] == pre[x] ? x : root(pre[x]);}
23
24 void Tarjan(int x,int fa)
25 {
26     f[x] = fa;
27     for(auto y : vec[x])
28     {
29         if(fa == y)continue;
30         Tarjan(y,x);
31         pre[y] = x;
32     }
33
34     vis[x] = 1;//放这里是为了防止方案中x == y的情况
35     for(auto y : q[x])if(vis[y])lca[x][y] = lca[y][x] = root(y);
36 }
37
38 void dfs(int x)
39 {
40     for(int i = 1;i <= 20;++ i)prefix[x][i] = prefix[f[x]][i];
41     ++ prefix[x][a[x]];
42     for(auto y : vec[x])
43     {
44         if(y == f[x])continue;
45         dfs(y);
46     }
47 }
48
49 void solve()
50 {
51     int n,_;cin >> n >> _;
```

```
52  for(int i = 1;i <= n;++ i)cin >> a[i],pre[i] = i;
53
54  for(int i = 1;i < n;++ i)
55  {
56      int u,v;cin >> u >> v;
57      vec[u].push_back(v);
58      vec[v].push_back(u);
59  }
60
61  for(int i = 1;i <= _;++ i)
62  {
63      int u,v;cin >> u >> v;
64      Q.push_back({u,v});
65      q[u].push_back(v);
66      q[v].push_back(u);
67  }
68
69  Tarjan(1,0);
70  dfs(1);
71
72  for(auto y : Q)
73  {
74      int ans = 0;
75      int u = y.first,v = y.second,z = lca[u][v];
76      for(int i = 1;i <= 20;++ i)
77          if(prefix[u][i] + prefix[v][i] - prefix[z][i] - prefix[f[z]][i])++ ans;
78      cout << ans << endl;
79  }
80 }
81
82 int main(){
83     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
84     int _ = 1;
85     //cin >> _;
86     while(_--)solve();
87     return 0;
88 }
```

数据结构\基础树上问题\lca\树上倍增求lca.cpp

```
1 //蓝桥 4385 最近公共祖先LCA查询
2 /*
3 n个节点n-1条边，q次查询，a和b的lca
4 */
5
6 /*
7 使用dfs求出dep深度以及st表（某点向上走多少步是谁），再利用st表找到lca
8 */
9 //在数据较大时可以考虑把dfs改成非递归的形式使用栈的方法实现
10 #include<bits/stdc++.h>
11 using namespace std;
12 using ll =long long;
13 const int N=1e5+10,P=1e9+7;
14
15 vector<int>vr[N];
16 int st[N][21],dep[N];//记录st表以及各个点的深度
17
18 void dfs(int x,int fa)
19 {
20     dep[x] = dep[fa] + 1;
21
22     st[x][0] = fa;//自己向上走一步到自己的父节点
23     for(int i = 1;i <= 20;++ i)
24         st[x][i] = st[st[x][i - 1]][i - 1];//内层是刚更新出的信息外层是自上而下的已经更新出了的信息（树形dp）
25
26     for(auto y : vr[x])
27     {
28         if(y == fa)continue;
29         dfs(y,x);
30     }
31 }
32
33 int lca(int x,int y)
34 {
35     if(dep[x] < dep[y])swap(x,y);//使x为较深的节点方便使两者到达较高者的同一线
36
37     for(int i = 20;i >= 0;-- i)if(dep[st[x][i]] >= dep[y])x = st[x][i];//跳至同一高度
38     if(x == y)return x;//对应x在y的子树上的情况
39
40     for(int i = 20;i >= 0;-- i)if(st[x][i] != st[y][i])x = st[x][i],y = st[y][i];
41     //当st相等时说明该位置在lca之上，不相等则在lca之下，因为st表对应二进制的组合则跳跃总步数可以
42     //为任意步，最后到达lca的子节点上
43     return st[x][0];//x为lca的子节点，返回此时的父节点即可
44 }
45
46 void solve()
47 {
48     int n;cin >> n;
49     for(int i = 1;i < n;++ i)
50     {
51         int u,v;cin >> u >> v;
```

```
51     vr[u].push_back(v);
52     vr[v].push_back(u);
53 }
54 dfs(1,0);
55 int q;cin >> q;
56 while(q--)
57 {
58     int a,b;cin >> a >> b;
59     cout << lca(a,b) << endl;
60 }
61 }
62
63 int main(){
64     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
65     int _ = 1;
66     //cin >> _;
67     while(_--)solve();
68     return 0;
69 }
```

数据结构\基础树上问题\lca\树上Tarjan离线求lca.cpp

```
1 //蓝桥 4385 最近公共祖先LCA查询
2 /*
3 离线查询，记录要查询的组利用dfs+并查集求出lca
4 */
5
6 //在数据较大时可以考虑把dfs改成非递归的形式使用栈的方法实现
7 #include<bits/stdc++.h>
8 using namespace std;
9 using ll =long long;
10 const int N=1e5+10,P=1e9+7;
11
12 struct Q{int z,id;};
13
14 vector<int>vr[N];
15 vector<Q>q[N];
16 bitset<N>vis;
17
18 int pre[N],ans[N];
19
20 int root(int x){return pre[x] == x ? x : root(pre[x]);}
21
22 void dfs(int x,int fa)
23 {
24     vis[x] = 1;//也可以放在后面再标记
25     for(auto y : vr[x])
26     {
27         if(y == fa)continue;
28         dfs(y,x);
29         pre[y] = x;//搜完子树后合并子节点到父节点上
30     }
31     for(auto i : q[x])if(vis[i.z])ans[i.id] = root(i.z);//当z访问过直接等于其的根否则交给z来处理
32 }
33
34 void solve()
35 {
36     int n;cin >> n;
37     for(int i = 1;i <= n;++ i)pre[i] = i;
38     for(int i = 1;i < n;++ i)
39     {
40         int u,v;cin >> u >> v;
41         vr[u].push_back(v);
42         vr[v].push_back(u);
43     }
44
45     int _;cin >> _;
46     for(int i = 1;i <= _;++ i)
47     {
48         int x,y;cin >> x >> y;
49         q[x].push_back({y,i});
50         q[y].push_back({x,i});//记录问题离线求解
51     }
```

```
52
53 dfs(1,0);
54
55 for(int i = 1;i <= _;++ i)cout << ans[i] << endl;
56 }
57
58 int main(){
59     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
60     int _ = 1;
61     //cin >> _;
62     while(_--)solve();
63     return 0;
64 }
```

数据结构\基础树上问题\树的四种遍历 .cpp

```
1 //其实只有dfs以及bfs，在二叉树中dfs又分为前中后序遍历，bfs也叫层序遍历
2 #include<bits/stdc++.h>
3 using namespace std;
4 using ll =long long;
5 const int N=1e6+10,P=1e9+7;
6
7 int ls[N],rs[N];
8 //前序遍历，根->左儿子->右儿子
9 void dfs1(int x)
10 {
11     cout << x << " ";
12     if(ls[x])dfs1(ls[x]);
13     if(rs[x])dfs1(rs[x]);
14 }
15 //中序遍历，左儿子->根->右儿子(相当于画条竖线从左到右扫描)
16 void dfs2(int x)
17 {
18     if(ls[x])dfs2(ls[x]);
19     cout << x << " ";
20     if(rs[x])dfs2(rs[x]);
21 }
22 //后序遍历，左儿子->右儿子->根
23 void dfs3(int x)
24 {
25     if(ls[x])dfs3(ls[x]);
26     if(rs[x])dfs3(rs[x]);
27     cout << x << " ";
28 }
29 //层序遍历，根开始从左到右遍历
30 void bfs()
31 {
32     queue<int>q;
33     q.push(1);
34     while(!q.empty())
35     {
36         int x = q.front();q.pop();
37         cout << x << " ";
38         if(ls[x])q.push(ls[x]);
39         if(rs[x])q.push(rs[x]);
40     }
41 }
42 void solve()
43 {
44     int n;cin >> n;
45     for(int i = 1;i <= n;++ i)cin >> ls[i] >> rs[i];
46     dfs1(1);
47     cout << endl;
48     dfs2(1);
49     cout << endl;
50     dfs3(1);
51     cout << endl;
```

```
52     bfs();
53     cout << endl;
54 }
55
56 int main(){
57     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
58     int _ = 1;
59     //cin >> _;
60     while(_--)solve();
61     return 0;
62 }
63 /*
64 //test
65 10
66 2 3
67 4 5
68 6 7
69 8 0
70 0 9
71 0 0
72 10 0
73 0 0
74 0 0
75 0 0
76 */
```

数据结构\基础树上问题\树的直径卖树 .cpp

```
1 //蓝桥 3029 卖树
2 /*
3 给定n个节点的树，一号节点为初始根，边权为k，根可以花费c代价移动到相邻位，
4 求由根出发的最长的路径权值之和减掉花费的最大值
5 */
6
7 /*
8 由初始根更新各点到初始根的距离，找到最深的点，再由该点出发再找到离该点最远的点，  

9 方法同上，再更新各点到该点的距离，则  $ans = (dep_v, dep_u) * k - c * dep_1$  的最大值
10 */
11 #include<bits/stdc++.h>
12 using namespace std;
13 using ll =long long;
14 const int N=1e5+10,P=1e9+7;
15
16 vector<int>vr[N];
17
18 int depU[N],depV[N],dep1[N];//记录各点到U, V, 1的距离
19
20 void dfs(int x,int fa,int dep[])
21 {
22     dep[x] = dep[fa] + 1;//子树长等于父节点长加1
23     for(auto y : vr[x])//遍历子树
24     {
25         if(y == fa)continue;//当子节点等于父节点时跳过防止重复跑也就是沿根一直向下跑
26         dfs(y,x,dep);
27     }
28 }
29
30 void solve()
31 {
32     ll n,k,c;cin >> n >> k >> c;
33     for(int i = 1;i < n;++ i)
34     {
35         int u,v;cin >> u >> v;
36         vr[u].push_back(v);
37         vr[v].push_back(u);
38     }
39     dep1[0] = depU[0] = depV[0] = -1;//这样的话这三个点到自己的距离会被更新成0
40     dfs(1,0,dep1);
41     int U = 1;
42     for(int i = 1;i <= n;++ i)if(dep1[i] > dep1[U])U = i;
43     //找到最深的节点作为U到V的直径的一个端点
44     dfs(U,0,depU);//从U去找到最长的作为直径
45     int V = 1;
46     for(int i = 1;i <= n;++ i)if(depU[i] > depU[V])V = i;
47     dfs(V,0,depV);
48     ll ans = 0;
49     for(int i = 1;i <= n;++ i)
50     {
51         ans = max(ans,max(depV[i],depU[i]) * k - c * dep1[i]);//从i出发最长的路径为i到U或者到V可  

证明到其它点一定没这长
```

```
52     vr[i].clear(); //多组测试需重构图
53 }
54 cout << ans << endl;
55 }
56 int main(){
57     ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
58     int _ = 1;
59     cin >> _;
60     while(_--) solve();
61     return 0;
62 }
```

数据结构\基础树上问题\树的重心.cpp

```
1  /*
2  树的重心求法
3  */
4
5  /*
6  树重心的一些性质
7  定义：去除任意节点后使得两个子树大小中的最大值的最小对应的节点为重心
8  1.重心的子树中的最大值小于等于n/2;
9  2.除重心外的节点都必然存在一颗大于n/2的子树；
10 3.重心个数至多为2，若为2则两重心必然相邻，将链接两重心的边删去后一定划分为两颗大小相等的树
11 4. 树所有点到某点的距离和到重心最小，若存在两个重心则距离和相同，反之距离和最小点便为重心
12 5.把两颗树通过一条边相连新重心在较大树的链接点与其原重心的简单路径上，若两树大小相等则重心就是
两链接点
13 */
14 #include<bits/stdc++.h>
15 using namespace std;
16 using ll =long long;
17 const int N=1e6+10,P=1e9+7;
18
19 vector<int>vr[N];
20 vector<int>v;//记录重心
21
22 int mss[N],sz[N],n;
23
24 void dfs(int x,int fa)
25 {
26     sz[x] = 1;
27     for(auto y : vr[x])
28     {
29         if(y == fa)continue;
30         dfs(y,x);
31         sz[x] += sz[y];//合并子树到根上
32         mss[x] = max(mss[x],sz[y]);//更新子树中的最大值
33     }
34     mss[x] = max(mss[x],n - sz[x]);//继续更新
35     if(mss[x] <= n / 2)v.push_back(x);
36 }
37
38 void solve()
39 {
40     cin >> n;
41     for(int i = 1;i < n;++ i)
42     {
43         int u,v;cin >> u >> v;
44         vr[u].push_back(v);
45         vr[v].push_back(u);
46     }
47     dfs(1,0);
48     for(auto x : v)cout << x << " ";
49 }
50
51 int main(){
```

```
52     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
53     int _ = 1;
54     //cin >> _;
55     while(_--)solve();
56
57 }
```

数据结构\树形数据结构\树状数组\单点修改范围查询 .cpp

```
1 //P3374 【模板】树状数组 1
2 /*
3 单点修改范围查询
4 */
5
6 /*
7 维护前缀和即可
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll =long long;
12 const int N=1e6+10,P=1e9+7;
13
14 ll a[N],t[N],n;
15
16 ll lowbit(int x){return -x & x;}
17
18 void update(int k,int x)
19 {
20     a[k] += x;//更新a数组
21     for(int i = k;i <= n;i += lowbit(i))t[i] += x;//更新树状数组中与该点有关的区间和
22 }
23
24 ll getprefix(int x)
25 {
26     ll res = 0;
27     for(int i = x;i >= 1;i -= lowbit(i))res += t[i];//返回prefix[x]
28     return res;
29 }
30
31 void solve()
32 {
33     int m;cin >> n >> m;
34     for(int i = 1;i <= n;++ i)
35     {
36         int x;cin >> x;
37         update(i,x);//更新tree
38     }
39     while(m --)
40     {
41         int op,x,y;cin >> op >> x >> y;
42         if(op == 1)update(x,y);
43         else cout << getprefix(y) - getprefix(x - 1) << endl;
44     }
45 }
46
47 int main(){
48     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
49     int _ = 1;
50     //cin >> _;
51     while(_--)solve();
```

```
52 |     return 0;  
53 | }
```

数据结构\树形数据结构\树状数组\范围修改单点查询 .cpp

```
1 //P3368 【模板】树状数组 2
2 /*
3 区间修改单点查询
4 */
5
6 /*
7 想到可以使用树状数组维护差分数组然后还原得到原数组来实现要求
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll =long long;
12 const int N=1e6+10,P=1e9+7;
13
14 ll a[N],t[N],n;
15
16 ll lowbit(int x){return -x & x;}
17
18 void update(int k,int x)
19 {
20     for(int i = k;i <= n;i += lowbit(i))t[i] += x;
21 }
22
23 ll getprefix(int x)
24 {
25     ll res = 0;
26     for(int i = x;i >= 1;i -= lowbit(i))res += t[i];
27     return res;
28 }
29
30 void solve()
31 {
32     int m;cin >> n >> m;
33     for(int i = 1;i <= n;++ i)
34     {
35         cin >> a[i];
36         update(i,a[i] - a[i - 1]);//插入差分数组到树状数组
37     }
38     while(m --)
39     {
40         int op;cin >> op;
41         if(op == 1)
42         {
43             int x,y,k;cin >> x >> y >> k;
44             update(x,k);
45             update(y + 1,-k);//修改差分数组以及和该位置有关的前缀和
46         }else{
47             int x;cin >> x;
48             cout << getprefix(x) << endl;//返回所维护的差分数组的前缀和
49         }
50     }
51 }
```

```
52
53 int main(){
54     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
55     int _ = 1;
56     //cin >> _;
57     while(_--)solve();
58     return 0;
59 }
```

数据结构\树形数据结构\树状数组\范围修改范围查询 .cpp

```
1 //P3372 【模板】线段树 1
2 /*
3 操作一给x到y区间内所有数加上k, 二查询区间的和
4 */
5 //用树状数组实现范围修改范围查询
6 //对差分数组求和再求和便为原数组的求和可以推出 $sum = k * (d_1 + d_2 + \dots + d_k) - ((1 - 1) * d_1 + \dots + (k - 1) * d_k)$ 
7 #include<bits/stdc++.h>
8 using namespace std;
9 using ll = long long;
10 const int N=1e6+10,P=1e9+7;
11
12 ll a[N],t1[N],t2[N],n;//t1记录差分数组, t2记录 $(i - 1) * di$ 数组
13
14 ll lowbit(int x){return -x & x;}
15
16 void update(int x,ll k,ll t[])
17 {
18     for(int i = x;i <= n;i += lowbit(i))t[i] += k;
19 }
20
21 ll getprefix(int x,ll t[])
22 {
23     ll res = 0;
24     for(int i = x;i >= 1;i -= lowbit(i))res += t[i];
25     return res;
26 }
27
28 void add(int x,int y,ll k)
29 {
30     update(x,k,t1);
31     update(y + 1,-k,t1);
32     update(x,(x - 1) * k,t2);
33     update(y + 1,- y * k,t2); //维护两个差分数组
34 }
35
36 void solve()
37 {
38     int m;cin >> n >> m;
39     for(int i = 1;i <= n;++ i)
40     {
41         cin >> a[i];
42         add(i,i,a[i]);
43     }
44     while(m --)
45     {
46         int op;cin >> op;
47         if(op == 1)
48         {
49             ll x,y,k;cin >> x >> y >> k;
50             add(x,y,k);
51         }else{}
```

```
52     ll x,y;cin >> x >> y;
53     cout << getprefix(y,t1) * y - getprefix(y,t2) - getprefix(x - 1,t1) * (x - 1) +
54     getprefix(x - 1,t2) << endl;
55 }
56 }
57
58 int main(){
59     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
60     int _ = 1;
61     //cin >> _;
62     while(_--)solve();
63     return 0;
64 }
```

数据结构\树形数据结构\树状数组\殷老师排队.cpp

```
1 //蓝桥 3620 殷老师排队
2 /*
3 给定n个点，每个点有对应的权值，操作一更改某点的权值，二查询某点的一个有关区间和的式子值
4 */
5
6 /*
7 涉及在线查询区间和，考虑使用树状数组维护，单点修改区间查询操作
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll =long long;
12 const int N=1e5+10,P=1e9+7;
13
14 int a[N],t[N],n,m;
15
16 int lowbit(int x){return x & -x;}//lowbit操作
17
18 void update(int k,int x)
19 {
20     a[k] += x;
21     for(int i = k;i <= n;i += lowbit(i))t[i] += x;//更新所有和k位置有关的区间和
22 }
23
24 int getprefix(int x)
25 {
26     int res = 0;
27     for(int i = x;i >= 1;i -= lowbit(i))res += t[i];//由不同区间组成prefix[x]
28     return res;
29 }
30
31 void solve()
32 {
33     cin >> n >> m;
34     for(int i = 1;i <= n;++ i)
35     {
36         int x;cin >> x;
37         update(i,x);
38     }
39
40     while(m --)
41     {
42         int op;cin >> op;
43         if(op == 1)
44         {
45             int x,y;cin >> x >> y;
46             update(x,y - a[x]);
47         }else {
48             int x;cin >> x;
49             cout << (2 * x - n) * a[x] + getprefix(n) - 2 * getprefix(x) << endl;
50         }
51     }
52 }
```

```
52 }
53
54 int main(){
55     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
56     int _ = 1;
57     //cin >> _;
58     while(_--)solve();
59     return 0;
60 }
```

数据结构\树形数据结构\线段树\维护区间和范围修改范围查询.cpp

```
1 //P3372 【模板】线段树 1
2 //经典模型：范围修改，单点修改，范围重置（修改和重置可以放在一起，关键在于重置优先级高于修改）
3 // + 维护区间和，区间最值
4 #include<bits/stdc++.h>
5 using namespace std;
6 using ll =long long;
7 const int N=1e5+10,P=1e9+7;
8
9 ll a[N],t[N << 2],lz[N << 2];//t树维护区间和， lz记录懒标记，最多扩展节点为总n的4倍
10
11 void up(int o){t[o] = t[o << 1] + t[o << 1 | 1];}//合并子节点信息
12
13 void lazy(int o,ll v,int n)//懒标记并更新该节点值
14 {
15     t[o] += v * n;
16     lz[o] += v;
17 }
18
19 void down(int o,int ln,int rn)//下放懒标记
20 {
21     if(lz[o])
22     {
23         lazy(o << 1,lz[o],ln);
24         lazy(o << 1 | 1,lz[o],rn);//更新左右儿子
25         lz[o] = 0;//懒标记下放完毕，清除懒标记
26     }
27 }
28
29 void bulid(int l,int r,int o)//建树
30 {
31     if(l == r)//basecase，递归出口，然后会自下向上合并信息
32     {
33         t[o] = a[l];
34         return;
35     }
36     int mid = (l + r) >> 1;
37     bulid(l,mid,o << 1);
38     bulid(mid + 1,r,o << 1 | 1);
39     up(o);
40 }
41
42 void add(int jobl,int jobr,ll v,int l,int r,int o)
43 {
44     if(jobl <= l && r <= jobr)
45     {
46         lazy(o,v,r - l + 1);
47         return;
48     }
49
50     int mid = (l + r) >> 1;
51     down(o,mid - 1,r - mid);
```

```

52     if(jobl <= mid)add(jobl,jobr,v,l,mid,o << 1);
53     if(jobr > mid)add(jobl,jobr,v,mid + 1,r,o << 1 | 1);
54     up(o);
55 }
56
57 ll query(int jobl,int jobr,int l,int r,int o)
58 {
59     if(jobl <= l && r <= jobr) return t[o];//该节点管辖区间被目标区间完全包含直接返回，该值一定是
正确的
60
61     int mid = (l + r) >> 1;
62     down(o,mid - 1 + 1,r - mid);//必定会向下走所有下放懒标记并更新子节点，以确保所返回的值都为正
确的值
63     ll ans = 0;
64     if(jobl <= mid)ans += query(jobl,jobr,l,mid,o << 1);
65     if(jobr > mid)ans += query(jobl,jobr,mid + 1,r,o << 1 | 1);
66     return ans;
67 }
68
69 void solve()
70 {
71     int n,m;cin >> n >> m;
72     for(int i = 1;i <= n;++ i)cin >> a[i];
73
74     bulid(1,n,1);
75
76     while(m --)
77     {
78         int op;cin >> op;
79         if(op == 1)
80         {
81             int l,r,ll v;cin >> l >> r >> v;
82             add(l,r,v,1,n,1);
83         }else {
84             int l,r;cin >> l >> r;
85             cout << query(l,r,1,n,1) << endl;
86         }
87     }
88 }
89
90 int main(){
91     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
92     int _ = 1;
93     //cin >> _;
94     while(_--)solve();
95     return 0;
96 }
```

数论\大素数判断.cpp

```
1 //P4718 【模板】Pollard-Rho
2 /*
3 判断一个大数是否是素数，不是则输入最大的质因子
4 */
5
6 /*
7 通过Miller-Rabin测试是否为素数，不是则通过Pollard-Rho去找质因子
8 记录后找到最大的输出即可
9 */
10 #include <bits/stdc++.h>
11 using namespace std;
12 using ll = long long;
13 using i128 = __int128_t;
14
15 // 读取一个 64 位整数，支持负数
16 ll read_ll() {
17     ll x = 0;
18     int f = 1;
19     char c = getchar();
20     // 跳过非数字字符，处理负号
21     while (c < '0' || c > '9') {
22         if (c == '-') f = -1;
23         c = getchar();
24     }
25     // 解析数字字符
26     while (c >= '0' && c <= '9') {
27         x = x * 10 + (c - '0');
28         c = getchar();
29     }
30     return x * f;
31 }
32
33 // 快速幂取模：计算 a^b % mod，时间 O(log b)
34 ll qpow(ll a, ll b, ll mod) {
35     ll res = 1;
36     a %= mod;
37     while (b) {
38         if (b & 1)
39             res = (i128)res * a % mod;
40         a = (i128)a * a % mod;
41         b >>= 1;
42     }
43     return res;
44 }
45
46 // Miller-Rabin 素性测试：判断 n 是否为素数
47 bool isPrime(ll n) {
48     // 小于 2 或偶数直接判断
49     if (n < 2) return false;
50     for (ll p : {2,3,5,7,11,13,17,19,23,29,31,37}) {
51         if (n % p == 0)
```

```

52         return n == p;
53     }
54     // 写 n-1 = d * 2^s
55     ll d = n - 1, s = 0;
56     while (!(d & 1)) {
57         d >>= 1;
58         ++s;
59     }
60     // 选定一组确定性底数，提高测试可靠性
61     for (ll a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
62         if (a % n == 0) continue;
63         ll x = qpow(a, d, n);
64         if (x == 1 || x == n - 1) continue;
65         bool composite = true;
66         // 检查 x^(2^r) 是否能变为 n-1
67         for (int r = 1; r < s; ++r) {
68             x = (i128)x * x % n;
69             if (x == n - 1) {
70                 composite = false;
71                 break;
72             }
73         }
74         // 若所有循环都没触达 n-1，则为合数
75         if (composite) return false;
76     }
77     return true;
78 }
79
80 // 全局随机数引擎，用于 Pollard Rho 随机化
81 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
82
83 // Pollard Rho 算法：找到 n 的一个非平凡因子
84 ll pollard(ll n) {
85     if (n % 2 == 0) return 2;
86     // 随机常数 c
87     ll c = uniform_int_distribution<ll>(1, n - 1)(rng);
88     // 初始值 x, y
89     ll x = uniform_int_distribution<ll>(0, n - 1)(rng);
90     ll y = x;
91     ll d = 1;
92     // f(v) = v^2 + c mod n
93     auto f = [&](ll v) { return ((i128)v * v + c) % n; };
94     // 循环直到找到非平凡因子
95     while (d == 1) {
96         x = f(x);
97         y = f(f(y));
98         d = __gcd(__llabs(x - y), n);
99     }
100    // 若失败 (d == n)，重试
101    return d == n ? pollard(n) : d;
102 }
103
104 // 分解 n 得到所有素因子，存入 res 向量
105 void factor(ll n, vector<ll>& res) {

```

```
106     if (n == 1) return;
107     // 若 n 是素数，直接加入
108     if (isPrime(n)) {
109         res.push_back(n);
110     } else {
111         // 否则用 Pollard Rho 找到因子 d
112         ll d = pollard(n);
113         factor(d, res);
114         factor(n / d, res);
115     }
116 }
117
118 // 主求解函数，处理多组测试
119 void solve() {
120     int T = (int)read_ll(); // 读入测试组数
121     while (T--) {
122         ll n = read_ll(); // 读入待测数
123         if (isPrime(n)) {
124             puts("Prime"); // 若是素数则输出 Prime
125         } else {
126             vector<ll> fac;
127             factor(n, fac); // 分解质因子
128             ll mx = 0;
129             // 找到最大的质因子
130             for (ll v : fac)
131                 mx = max(mx, v);
132             printf("%lld\n", mx);
133         }
134     }
135 }
136
137 int main() {
138     ios::sync_with_stdio(false);
139     cin.tie(nullptr);
140     cout.tie(nullptr);
141     solve(); // 调用主函数
142     return 0;
143 }
```

数论\快速幂+费马求逆元.txt

1. 快速幂

```
ll qmi(ll a,ll b,ll p)
{
    ll res = 1;
    while(b)
    {
        if(b & 1)res = res * a % p;
        a = a * a % p;b >= 1;
    }
    return res;
}
```

2. 费马小定理求逆元

引入

在 $\text{mod}(p)$ 下 a 的 $p-1$ 次方恒等于1

两次同除 a , 由于对除法取模不具有封闭性所有此时 a 的逆元便等价于 a 的 $p-2$ 次方
也就是 $\text{qmi}(x,p - 2,p);$

数论\扩展欧几里得.txt

1.辗转相除法

```
ll gcd(ll a,ll b)
{
    if(!b) return a;
    return gcd(b,a % b);
}
```

2.

```
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(!b)
    {
        x = 1,y = 0;
        return a;
    }
    int d = exgcd(b,a % b,y,x);
    y -= (a / b) * x;
    return d;
}
```

//保证a,b为正，为负变形一下就行，所求的便是-x

//传回的x, y为一组特解

3.二元一次不定方程的解的规律

```
x = x0 + b / a * k;
```

```
y = y0 + a / b * k;
```

//x0,y0通过exgcd求出一组特解

//求最小非负解， $x = (x0 \% (b / a) + b / a) \% (b / a)$;

// $ax + by = c$ 可变形为 $a1 * x + b1 * y = \text{gcd}(a,b)$ 的形式

数论\欧拉函数及欧拉降幂.txt

1. 欧拉函数

```
ll phi = n;
for(int i = 2;i <= n / i;++ i)
{
    if(n % i)continue;
    phi = phi / i * (i - 1);
    while(!(n % i))n /= i;
}
```

if(n > 1)phi = phi / n * (n - 1);
//phi 即为 $\leq n$ 的数中与 n 互质的数的个数

2. 欧拉降幂

引入

设欧拉函数为 $f(x)$ ；

欧拉定理：在模 c 下 a 的 $f(c)$ 次方恒等于 1

//特殊的：当 c 为质数时 $f(c)=c - 1$ ，不难看出这就是费马小定理

//这个也可以用于降幂当 p 为质数时 $b = b \% (p - 1)$

降幂

当 a 的指数 b 为大数时考虑使用欧拉降幂

当 a 与 c 互质时， a 的 b 次方 = a 的 $b(\text{mod}(f(c)))$ 次方 $(\text{mod}(f(c)))$ ；

一般的 a 与 c 不互质时， a 的 b 次方 = a 的 $b(\text{mod}(f(c))) + f(c)$ 次方 $(\text{mod}(f(c)))$ ；

数论\排列组合.txt

1.

加法原理，乘法原理

2.组合数求法

一.DP法(适用于p不是质数的情况，且n * m)

由 $C(n,m) = C(n - 1, m - 1) + C(n - 1, m)$;

//加法原理，分类选择第一个和不选择第一个

方程满足DP条件//其实也是杨辉三角

二维dp:

```
vector<vector<ll>> dp;
dp.resize(n + 1);
for(int i = 0; i <= n; ++ i)
{
    int en = min(i, m);
    dp[i].resize(en + 1);
    dp[i][0] = 1;
    for(int j = 1; j <= en; ++ j)
        dp[i][j] = (dp[i - 1][j - 1] + dp[i - 1][j]) % p;
}
```

//dp[n][m]即为所求

一维dp优化:

```
vector<ll> dp = {1};
for(int i = 0; i <= n; ++ i)
{
    int en = min(m, i);
    dp.resize(en + 2);
    for(int j = en; j >= 1; -- j)
        dp[j] = (dp[j - 1] + dp[j]) % p;
}
```

//dp[m]即为所求，复杂度很优秀

二.预处理阶乘数组以及阶乘逆元数组(适用于模数为质数时(此时才存在逆元))

```
for(int i = 1; i <= max; ++ i) fact[i] = fact[i - 1] * i % p;
invfact[max] = inv(fact[max]);
for(int i = max - 1; i >= 0; -- i) invfact[i] = invfact[i + 1] * (i + 1) % p;
//预处理的话复杂度会比较稳定注意0也是有逆元的对应n==m的条件
```

组合数函数

```
1.11 C(int a, int b){return fact[a] * invfact[b] % p * invfact[a - b] % p;}
```

2.11 C(int a, int b)

```
{
    ll res = 1;
    int mi = min(b, a - b), mx = max(b, a - b);
    for(int i = a; i >= mx; -- i) res = res * i % p;
    //O(mx)如果下面没有预处理逆元可以让上面多算一点
    for(int i = 1; i <= mi; ++ i) res = res * inv(i) % p;
    //这里也最好是预处理特别是要重复调用时
}
```

数论\裴蜀定理.txt

1.

$ax + by = c; d = \gcd(a, b);$

当且仅当 $d \mid c$ 时有整数解

//从另一个角度理解就是 $d =$ 对任意 x, y 拼凑出的最小正整数

2.

该结论可以推广到多元

3.

当 a 与 b 互质时，也就是 $d = 1$ ；此时 c 可以表示任意整数

4.

当 a 与 b 互质时，且规定 x, y 为非负，则 c 不能表示的最大非负数为 $ab - a - b$ ；

//和完全剩余系有关

数论\唯一分解定理.txt

```
1.  
for(int i = 2;i <= n / i;++ i)  
{  
    if(!(n % i))  
    {  
        v.push_back(i);  
        while(!n % i)n /= i;  
    }  
}  
if(n > 1)v.push_back(n);  
//v便含所有n的质因子，O(根号n)，处理过程有点像phi欧拉函数
```

2.
pollard_rho启发式方法分解质因数
有兴趣可以了解一下

3. 约数个数
由于n分解成了v的k次方，每个v的贡献为k+1个，最后ans=(k+1)的累乘

4. 约数之和
加起来再提公因子发现ans=v的i次方的累加的累乘

数论\质数判断.txt

单点查询

1.较小数

```
for(int i = 2;i <= n / i;++ i)if(!(n % i))return 0;  
return 1;
```

//O(根号n)

2.较大的数

Miller-Rabin测试法

有兴趣可以了解一下

//O(s * (logn)的三次方)

数论\质数筛.txt

1. 埃氏筛

版本一(适用于需要输出具体质数) $O(n \log(\log n))$

```
for(int i = 2; i <= n / i; ++ i)
{
    if(!vis[i])
    {
        for(int j = i * i; j <= n; j += i)
        {
            vis[j] = 1; // 打标记为合数
        }
    }
}
```

} // vis 中未标记的为质数

版本二(适用于只需要计数, 时间复杂度很优秀)

```
int cnt = n + 1 >> 1;
for(int i = 3; i <= n / i; i += 2)
{
    if(!vis[i])
    {
        for(int j = i * i; j <= n; j += 2 * i)
        {
            if(!vis[j])
            {
                vis[j] = 1;
                -- cnt;
            }
        }
    }
}
```

} // cnt 便为质数个数也可以叫它奇数筛

2. 欧拉筛 $O(n)$

```
int cnt = 0;
for(int i = 2; i <= n; ++ i)
{
    if(!vis[i]) prime[++ cnt] = i;
    for(int j = 1; j <= cnt; ++ j)
    {
        if(prime[j] * i > n) break;
        vis[prime[j] * i] = 1;
        if(!(i % prime[j])) break;
    }
}
```

} // cnt 为质数个数, prime 存放了质数

搜索\bfs\八数码难题.cpp

```
1 //P1379 八数码难题
2 /*
3 * 3的棋盘给定初始状态0可以和任意相邻位置交换，求到目标状态的最小操作次数
4 */
5
6 /*
7 遇到最小操作数问题可以考虑一下bfs，使用map将string映射一下记录vis， bfs里之间模拟移动的过程即可
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll = long long;
12 const ll inf = 2e18;
13 const int N=1e6+10,P=1e9+7;
14
15 struct op_0{string s;int o;}op;//记录0的位置以免反复遍历找0
16
17 int dx[4] = {-3,1,3,-1};
18 string goal = "123804765";
19 map<string,int>vis;
20
21 void solve()
22 {
23     string s;cin >> s;
24     if(s == goal)
25     {
26         cout << 0;
27         return;
28     }
29     queue<op_0>q;
30     int o = 0;
31     while(s[o] != '0')++ o;
32
33     int ans = 0;
34     q.push({s,o});
35     while(!q.empty())
36     {
37         ++ ans;
38         int en = q.size();
39         while(en --)
40         {
41             string t = q.front().s;
42             int x = q.front().o;
43             q.pop();
44             if(vis[t])continue;
45             vis[t] = 1;
46             for(int i = 0;i < 4;++ i)
47             {
48                 if(i == 3 && (x == 3 || x == 6) || i == 1 && (x == 2 || x == 5))continue;
49                 //左边界不能向左走，右边界不能向右走
50                 //这里可以将x映射到x = (x - 1) / 3 + 1, y = (x - 1) % 3 + 1,然后判边界
51                 int xx = x + dx[i];
```

```
52     if(xx < 0 || xx > 8)continue;//越界
53
54     swap(t[x],t[xx]);//交换0和dx位置
55     q.push({t,xx});
56     if(t == goal)//到达目标序列
57     {
58         cout << ans;
59         return;
60     }
61     swap(t[x],t[xx]);//复原现场
62 }
63 }
64 }
65 }
66
67 int main(){
68     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
69     int _ = 1;
70     //cin >> _;
71     while(_--)solve();
72     return 0;
73 }
```

搜索\bfs\魔板.cpp

```
1 //P2730 [USACO3.2] 魔板 Magic Squares
2 /*
3 给定末状态，求初始状态通过a, b, c三种操作到达该状态的最小操作数
4 */
5
6 /*
7 和八数码本质一样的最小操作数问题，通过坐标变换之间模拟三种操作进行状态转移即可
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll = long long;
12 const ll inf = 2e18;
13 const int N=1e6+10,P=1e9+7;
14
15 string st = "12345678";
16 map<string,string>mp;
17
18 void solve()
19 {
20     char x;
21     string en = "";
22     for(int i = 0;i < 8;++ i)cin >> x,en += x;
23
24
25     queue<string>q;
26     q.push(st);
27     while(!q.empty())
28     {
29         string t = q.front();
30         q.pop();
31
32         //A
33         string y = t;
34         for(int i = 0;i < 4;++ i)swap(y[i],y[8 - i - 1]);
35         if(!mp.count(y))q.push(y),mp[y] = mp[t] + 'A';
36         //这个位置使用了mp[st]，所以自动创建了一个st的键值
37
38         //B
39         for(int i = 1;i < 4;++ i)y[i] = t[i - 1];
40         for(int i = 4;i < 7;++ i)y[i] = t[i + 1];
41         y[0] = t[3],y[7] = t[4];
42         if(!mp.count(y))q.push(y),mp[y] = mp[t] + 'B';
43
44         //C
45         y = t;
46         y[1] = t[6],y[2] = t[1],y[5] = t[2],y[6] = t[5];
47         if(!mp.count(y))q.push(y),mp[y] = mp[t] + 'C';
48
49         if(mp.count(en))
50         {
51             int n = mp[en].length();
```

```
52     cout << n << endl;
53     for(int i = 0;i < n;++ i)
54     {
55         if(i && i % 60 == 0)cout << endl;
56         cout << mp[en][i];
57     }
58     return;
59 }
60 }
61 }
62
63 int main(){
64     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
65     int _ = 1;
66     //cin >> _;
67     while(_--)solve();
68     return 0;
69 }
```

搜索\ bfs\ 奇怪的电梯.cpp

```
1 //P1135 奇怪的电梯
2 /*
3 每个点一个权值电梯在一条直线上移动，在某点可以向左移动w步或者向右移动w步，求最小操作次数
4 */
5
6 /*
7 直接按题意扩展状态即可
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll =long long;
12 const int N=1e6+10,P=1e9+7;
13
14 int a[210],vis[210],n,sx,ex;
15 queue<int>q;
16
17 void bfs()
18 {
19     q.push(sx);
20     while(!q.empty())
21     {
22         int x = q.front();
23         q.pop();
24         int xx = x + a[x];
25         if(xx <= n && vis[xx] == -1)
26         {
27             q.push(xx);
28             vis[xx] = vis[x] + 1;
29         }
30         xx = x - a[x];
31         if(xx > 0 && vis[xx] == -1)
32         {
33             q.push(xx);
34             vis[xx] = vis[x] + 1;
35         }
36     }
37 }
38
39 void solve()
40 {
41     cin >> n >> sx >> ex;
42     for(int i = 1;i <= n;++ i)cin >> a[i];
43     memset(vis,-1,sizeof(vis));
44     vis[sx] = 0;
45     bfs();
46     //for(int i = 1;i <= n;++ i)cout << vis[i] << " ";
47     cout << vis[ex];
48 }
49
50 int main(){
51     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
```

```
52 |     int _ = 1;
53 |     //cin >> _;
54 |     while(_--)solve();
55 |     return 0;
56 | }
```

搜索\ bfs\ 双端队列 **bfs.cpp**

```
1 //P4667 [BalticOI 2011] Switch the Lamp On 电路维修 (Day1)
2 /*
3 方格的对角线含电线有两个方向，转动电线使得线路由左上连至右下，求最小代价
4 */
5
6 /*
7 最小操作代价考虑bfs，又由于每个格子里电线只有两种状态，设0为联通1为不连通，
8 把格点看作点，优先扩展连通的边，这样可以更快逼近答案，类似dj算法，A*思想
9 */
10 //类似于dj算法，将可能更优的路径先拓展，且弹出复杂度O(1)，也可以叫01bfs（边权为0或1），大大减少了的要扩展的点
11 #include<bits/stdc++.h>
12 using namespace std;
13 using ll = long long;
14 const ll inf = 2e18;
15 const int N=3e5,P=1e9+7;
16
17 int vr[N][4],n,m,dx[4],dis[N];
18 bitset<N>vis;
19
20 void solve()
21 {
22     cin >> n >> m;
23     int en = ++n * ++m;
24     int dx[4] = {- m - 1,- m + 1,m + 1,m - 1}; //向四个角走的坐标变换（0代表左上角，1代表右上）
25     for(int i = 1;i < n;++ i)
26         for(int j = 1;j < m;++ j)
27         {
28             char op;cin >> op;
29             int x = (i - 1) * m + j;
30             if(op == '/')vr[x][2] = vr[x + m + 1][0] = 1;//1代表不连通操作代价为1
31             else vr[x + 1][3] = vr[x + m][1] = 1;
32             //该边影响的四条边权
33         }
34
35 deque<int>q;
36 q.push_back(1);
37 vis[1] = 1;
38 while(!q.empty())
39 {
40     int t = q.front();
41     q.pop_front();
42     for(int i = 0;i < 4;++ i)//向四个对角扩展
43     {
44         int xx = t + dx[i];//也可以映射到x, y后再判不合法
45         if(!(1 <= xx && xx <= en) || t % m == 0 && (i == 1 || i == 2) || t % m == 1 && (i ==
46 0 || i == 3))continue;//不越界且移动合法
47         if(!vis[xx] || dis[t] + vr[t][i] < dis[xx])
48         {
49             if(vr[t][i])q.push_back(xx);
50             else q.push_front(xx);//将与t连通的放在队列前面优先拓展
51             dis[xx] = dis[t] + vr[t][i],vis[xx] = 1;
```

```
51     }
52 }
53 }
54 if(vis[en])cout << dis[en];
55 else cout << "NO SOLUTION";
56 }
57 }
58
59 int main(){
60     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
61     int _ = 1;
62     //cin >> _;
63     while(_--)solve();
64     return 0;
65 }
```

搜索\dfs\八皇后.cpp

```
1 //P1219 [USACO1.5] 八皇后 Checker Challenge
2 /*
3 填下一个格子， 标记行， 列， 斜线后dfs回溯取消标记找下一种可能
4 */
5 #include<bits/stdc++.h>
6 using namespace std;
7 using ll =long long;
8 const int N=1e6+10,P=1e9+7;
9 int ans,n,b1[20],b2[30],b3[30],a[20];
10
11 void dfs(int x)
12 {
13     if(x > n)
14     {
15         ++ ans;
16         if(ans <= 3)
17         {
18             for(int i = 1;i <= n;++ i)cout << a[i] << " ";
19             cout << endl;
20         }
21         return;
22     }
23     for(int y = 1;y <= n;++ y)
24     {
25         if(!b1[y] && !b2[n - x + y] && !b3[x + y - 1])
26         {
27             b1[y] = b2[n - x + y] = b3[x + y - 1] = 1;
28             a[x] = y;
29             dfs(x + 1);
30             b1[y] = b2[n - x + y] = b3[x + y - 1] = 0;
31         }
32     }
33 }
34
35 void solve()
36 {
37     cin >> n;
38     dfs(1);
39     cout << ans;
40 }
41
42 int main(){
43     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
44     int _ = 1;
45     //cin >> _;
46     while(_--)solve();
47     return 0;
48 }
```

搜索\dfs\取数游戏.cpp

```
1 //P1123 取数游戏
2 /*
3 给定n*m矩阵取出一些两两不相邻的数求最大总和
4 */
5
6 /*
7 同理标记回溯，更新出最大总和即可
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll =long long;
12 const int N=1e6+10,P=1e9+7;
13 int n,m,a[10][10],vis[10][10],ans;
14 int xx[9] = {-1,-1,-1,0,0,0,1,1,1},yy[9] = {-1,0,1,-1,0,1,-1,0,1};
15
16 int judge(int x,int y)
17 {
18     for(int o = 0;o <= 8;++ o)if(vis[x + xx[o]][y + yy[o]])return 0;
19     return 1;
20 }
21
22 void dfs(int x,int y,int sum)
23 {
24     for(int i = x;i <= n;++ i)
25         for(int j = (i == x ? y : 1);j <= m;++ j)
26         {
27             if(!judge(i,j))continue;
28             vis[i][j] = 1;
29             dfs(i,j,sum + a[i][j]);
30             vis[i][j] = 0;
31         }
32         ans = max(ans , sum);
33     return;
34 }
35
36 void solve()
37 {
38     cin >> n >> m;
39     for(int i = 1;i <= n;++ i)
40         for(int j = 1;j <= m;++ j)cin >> a[i][j];
41     dfs(1,1,0);
42     memset(vis,0,sizeof(vis));
43     cout << ans << endl;
44     ans = 0;
45 }
46
47 int main(){
48     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
49     int _ = 1;
50     cin >> _;
51     while(_--)solve();
```

```
52 |     return 0;  
53 | }
```

搜索\dfs\智乃的凑数题.cpp

```
1 //NC288979 智乃的“凑数”题
2 /*
3 对n个数进行操作，进行m次查询输出行和列相乘为x的方案
4 */
5
6 /*
7 dfs先把该数加入找下一层，再弹出向下转移，用元组记录标记当前同一状态防止反复转移
8 */
9 //有时候可以把表打出来来找寻合法的剪枝方法
10 #include<bits/stdc++.h>
11 using namespace std;
12 using ll = long long;
13 const ll inf = 2e18;
14 const int N=1e6+10,P=1e9+7;
15
16 struct {int o,p,q;}back;
17
18 map<tuple<int,int,int>,bool>vin;
19 vector<int>v1,v2;
20 int a[N],flag = 0,n,x;
21 bitset<N>vis;
22
23 void dfs(int u,int v,int op)
24 {
25     if(vin[make_tuple(u,v,op)])return;//记录重复状态
26     vin[make_tuple(u,v,op)] = 1;
27     if(u * v > x || flag || u > x || v > x)return;//去除不合法状态
28     if(u * v == x)
29     {
30         flag = 1;//标记找到了，方便其他函数退出
31         cout << "Yes" << endl;
32         cout << v1.size() << " " << v2.size() << endl;
33         for(auto i : v1)cout << i << " ";
34         cout << endl;
35         for(auto i : v2)cout << i << " ";
36         cout << endl;
37         return;
38     }
39     if(op == n + 1)return;//越界
40     //三种操作
41     if((u + a[op]) * v <= x)
42     {
43         v1.push_back(a[op]);
44         dfs(u + a[op],v,op + 1);//回溯
45         v1.pop_back();
46     }
47     if((v + a[op]) * u <= x)
48     {
49         v2.push_back(a[op]);
50         dfs(u,v + a[op],op + 1);
```

```
52     v2.pop_back();
53 }
54
55 dfs(u,v,op + 1);
56 return;
57 }
58
59 void solve()
60 {
61     int m;cin >> n >> m;
62     for(int i = 1;i <= n;++ i)cin >> a[i];
63     while(m --)
64     {
65         cin >> x;
66         if(vis[x])
67         {
68             cout << "No" << endl;
69             continue;
70         }
71         vin.clear();
72         v1.clear();
73         v2.clear();
74         flag = 0;
75         dfs(0,0,1);//行和, 列和, 步数坐标
76         if(!flag)cout << "No" << endl,vis[x] = 1;
77     }
78 }
79
80 int main(){
81     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
82     int _ = 1;
83     //cin >> _;
84     while(_--)solve();
85     return 0;
86 }
```

图论\拓扑排序\bfs实现.cpp

```
1 //B3644 【模板】拓扑排序 / 家谱树
2 //bfs实现也叫卡恩算法
3 #include<bits/stdc++.h>
4 using namespace std;
5 using ll = long long;
6 const ll inf = 2e18;
7 const int N=1e6+10,P=1e9+7;
8
9 vector<int>vr[110],tp;//tp内便是拓扑序列
10 int din[110],n;
11
12 void topo()
13 {
14     queue<int>q;
15     for(int i = 1;i <= n;++ i)
16         if(!din[i])q.push(i);//从入度为零的点开始扩展
17
18     while(!q.empty())
19     {
20         int x = q.front();q.pop();
21         tp.push_back(x);
22         for(auto y : vr[x])
23             if(-- din[y] == 0)q.push(y);
24     }
25 }
26
27 void solve()
28 {
29     cin >> n;
30     for(int i = 1;i <= n;++ i)
31     {
32         int x;
33         while(cin >> x,x)vr[i].push_back(x),++ din[x];
34     }
35
36     topo();
37
38     for(auto x : tp)cout << x << " ";
39     //当tp.size() != n时说明该有向图中有环
40 }
41
42 int main(){
43     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
44     int _ = 1;
45     //cin >> _;
46     while(_--)solve();
47     return 0;
48 }
```

图论\拓扑排序\dfs实现 .cpp

```
1 //B3644 【模板】拓扑排序 / 家谱树
2 //dfs实现拓扑排序，为匈牙利算法垫基
3 #include<bits/stdc++.h>
4 using namespace std;
5 using ll = long long;
6 const ll inf = 2e18;
7 const int N=1e6+10,P=1e9+7;
8
9 vector<int>vr[110],tp;
10 int c[110],n;
11
12 bool dfs(int x)
13 {
14     c[x] = -1;
15     for(auto y : vr[x])
16         if(x == -1 || !c[y] && !dfs(y))return 0;
17     c[x] = 1;
18     tp.push_back(x);
19     return 1;
20 }
21
22 bool topo()
23 {
24     for(int i = 1;i <= n;++ i)
25         if(!c[i] && !dfs(i))return 0;
26     reverse(tp.begin(),tp.end());
27 }
28
29 void solve()
30 {
31     cin >> n;
32     for(int i = 1;i <= n;++ i)
33     {
34         int x;
35         while(cin >> x,x)vr[i].push_back(x);
36     }
37
38     topo();
39
40     for(auto x : tp)cout << x << " ";
41     //当tp.size() != n或者topo返回0时说明该有向图中有环
42 }
43
44 int main(){
45     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
46     int _ = 1;
47     //cin >> _;
48     while(_--)solve();
49     return 0;
50 }
```

图论\最短路\Dijkstra\带等待时间的最短路.cpp

```
1 //https://codeforces.com/gym/102875/problem/I
2 /*
3 给定n*m的矩阵，起点 (sx,sy) ，终点 (ex,ey) ，格子属性矩阵a, b，边权矩阵c（横着走），矩阵w（竖着走）
4 当t % (a + b) < a可以横着走，否则可以竖着走，可以在原地等待，求st到ed的最短时间
5 */
6
7 /*
8 这是一个带等待时间的最短路，考虑贪心+dj，在松弛的时候，贪心的等待，判断wait + w + dis[u] <
9 dis[v]，若满足则进行松弛
10 细节：在压缩二维到一维后回转的时候应该是x = (u - 1) / m + 1, y = (u - 1) % m + 1
11 */
12 #include<bits/stdc++.h>
13 using namespace std;
14 using ll = long long;
15 const ll inf = 1e18;
16 const int N=25e4 + 10,P=1e9+7;
17
18 int a[N],b[N],c[N][2],w[N][2],n,m,sx,sy,ex,ey,st,ed;//c[][0]代表左，w[][0]代表上
19 ll dis[N];
20
21 void dj()
22 {
23     bitset<N>vis;
24     priority_queue<pair<ll,int>>pq;
25
26     pq.push({0,st});
27     while(!pq.empty())
28     {
29         auto _ = pq.top();pq.pop();
30         int u = _.second;
31         if(vis[u])continue;
32         if(u == ed) return;
33         vis[u] = 1;
34         int op = dis[u] % (a[u] + b[u]);
35         if(op >= a[u])
36         {
37             int x = (u - 1) / m + 1,y = (u - 1) % m + 1;
38             if(y - 1 >= 1 && dis[u] + c[u][0] < dis[u - 1])
39                 dis[u - 1] = dis[u] + c[u][0],pq.push({- dis[u - 1],u - 1});
40             if(y + 1 <= m && dis[u] + c[u][1] < dis[u + 1])
41                 dis[u + 1] = dis[u] + c[u][1],pq.push({- dis[u + 1],u + 1});
42
43             ll wait = a[u] - op + b[u];
44             if(x - 1 >= 1 && dis[u] + w[u][0] + wait < dis[u - m])
45                 dis[u - m] = dis[u] + w[u][0] + wait,pq.push({- dis[u - m],u - m});
46             if(x + 1 <= n && dis[u] + w[u][1] + wait < dis[u + m])
47                 dis[u + m] = dis[u] + w[u][1] + wait,pq.push({- dis[u + m],u + m});
48         }else{
49             int x = (u - 1) / m + 1,y = (u - 1) % m + 1;
50             if(x - 1 >= 1 && dis[u] + w[u][0] < dis[u - m])
```

```

51     dis[u - m] = dis[u] + w[u][0],pq.push({- dis[u - m],u - m});
52     if(x + 1 <= n && dis[u] + w[u][1] < dis[u + m])
53         dis[u + m] = dis[u] + w[u][1],pq.push({- dis[u + m],u + m});
54
55     ll wait = a[u] - op;
56     if(y - 1 >= 1 && dis[u] + c[u][0] + wait < dis[u - 1])
57         dis[u - 1] = dis[u] + c[u][0] + wait,pq.push({- dis[u - 1],u - 1});
58     if(y + 1 <= m && dis[u] + c[u][1] + wait < dis[u + 1])
59         dis[u + 1] = dis[u] + c[u][1] + wait,pq.push({- dis[u + 1],u + 1});
60     }
61 }
62 }
63
64 void solve()
65 {
66     cin >> n >> m >> sx >> sy >> ex >> ey;
67
68     st = (sx - 1) * m + sy;
69     ed = (ex - 1) * m + ey;
70
71     for(int i = 1;i <= n;++ i)
72         for(int j = 1;j <= m;++ j)cin >> a[(i - 1) * m + j];
73
74     for(int i = 1;i <= n;++ i)
75         for(int j = 1;j <= m;++ j)cin >> b[(i - 1) * m + j];
76
77     for(int i = 1;i <= n;++ i)
78         for(int j = 1;j < m;++ j)
79         {
80             int u = (i - 1) * m + j,v = u + 1;
81             int x;cin >> x;
82             c[u][1] = c[v][0] = x;
83         }
84
85     for(int i = 1;i < n;++ i)
86         for(int j = 1;j <= m;++ j)
87         {
88             int u = (i - 1) * m + j,v = u + m;
89             int x;cin >> x;
90             w[u][1] = w[v][0] = x;
91         }
92
93     for(int i = 1;i <= n * m;++ i)dis[i] = inf;
94     dis[st] = 0;
95     dj();
96
97     /*
98     for(int i = 1;i <= n;++ i)
99     {
100         for(int j = 1;j <= m;++ j)
101         {
102             if(dis[(i - 1) * m + j] == inf)cout << "no ";
103             else cout << dis[(i - 1) * m + j] << " ";
104         }

```

```
105     cout << endl;
106 }
107 */
108 cout << dis[ed];
109 }
110
111 int main(){
112     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
113     int _ = 1;
114     //cin >> _;
115     while(_--)solve();
116     return 0;
117 }
```

图论\最短路\Dijkstra\堆实现.cpp

```
1 //P4779 【模板】单源最短路径（标准版）
2 //不能正确处理负边权
3 //O(mlogm)
4 /*
5 也可以通过重载结构体内运算符来排
6 本代码使用的是pair大根堆（加入堆时dis加个负号就可以是距离小的点在上面）
7 bool operator < (const load &other) const
8 {
9     return w > other.w;
10 }
11 */
12 #include<bits/stdc++.h>
13 using namespace std;
14 using ll = long long;
15 const int inf = INT_MAX;
16 const int N=1e6+10,P=1e9+7;
17
18 struct edge{int v,w;};
19 int n,dis[N];
20 bitset<N>vis;
21 vector<edge>vr[N];
22
23 void dj(int st)
24 {
25     priority_queue<pair<int,int>>pq;
26     for(int i = 1;i <= n;++ i)dis[i] = inf;
27     dis[st] = 0;
28     pq.push({0,st});
29     while(!pq.empty())
30     {
31         auto t = pq.top();pq.pop();
32
33         int x = t.second;
34         if(vis[x])continue;
35         vis[x] = 1;
36
37         //松弛操作
38         for(auto ed : vr[x])
39         {
40             ll y = ed.v,w = ed.w;
41
42             //通过x到y更近就更新并加入堆中
43             if(dis[x] + w < dis[y])
44             {
45                 dis[y] = dis[x] + w;
46                 pq.push({- dis[y],y});
47             }
48         }
49     }
50 }
```

```
52 | void solve()
53 | {
54 |     int m,s;cin >> n >> m >> s;
55 |
56 |     while(m --)
57 |     {
58 |         int u,v,w;cin >> u >> v >> w;
59 |         vr[u].push_back({v,w});
60 |     }
61 |
62 |     dj(s);
63 |
64 |     for(int i = 1;i <= n;++ i)cout << dis[i] << " ";
65 |
66 |
67 | int main(){
68 |     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
69 |     int _ = 1;
70 |     //cin >> _;
71 |     while(_--)solve();
72 |     return 0;
73 | }
```

图论\最短路\Dijkstra\堆实现带路径.cpp

```
1 //路径输出
2 #include<bits/stdc++.h>
3 using namespace std;
4 using ll = long long;
5 const int inf = INT_MAX;
6 const int N=1e6+10,P=1e9+7;
7
8 struct edge{int v,w;};
9 int n,dis[N],pre[N];//使用pre前驱数组记录
10 bitset<N>vis;
11 vector<edge>vr[N];
12
13 void dj(int st)
14 {
15     priority_queue<pair<int,int>>pq;
16     for(int i = 1;i <= n;++ i)dis[i] = inf;
17     dis[st] = 0;
18     pq.push({0,st});
19     while(!pq.empty())
20     {
21         auto t = pq.top();pq.pop();
22
23         int x = t.second;
24         if(vis[x])continue;
25         vis[x] = 1;
26
27         //松弛操作
28         for(auto ed : vr[x])
29         {
30             ll y = ed.v,w = ed.w;
31
32             if(dis[x] + w < dis[y])
33             {
34                 dis[y] = dis[x] + w;
35                 pre[y] = x;//记录前驱点
36                 pq.push({- dis[y],y});
37             }
38         }
39     }
40 }
41
42 void dfs_path(int x,int st)
43 {
44     if(x == st)//一直通过dfs找到起点后递归输出
45     {
46         cout << x;
47         return;
48     }
49     dfs_path(pre[x],st);//递归到结束后再往下执行输出路径
50     cout << x;
51 }
```

```
52
53 void solve()
54 {
55     int m,s;cin >> n >> m >> s;
56
57     while(m --)
58     {
59         int u,v,w;cin >> u >> v >> w;
60         vr[u].push_back({v,w});
61     }
62
63     dj(s);
64
65     for(int i = 1;i <= n;++ i)cout << dis[i] << " ";
66 }
67
68 int main(){
69     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
70     int _ = 1;
71     //cin >> _;
72     while(_--)solve();
73     return 0;
74 }
```

图论\最短路\Dijkstra\分层图最短路.cpp

```
1 //P4568 [JLOI2011] 飞行路线
2 /*
3 给定 $0 \sim n-1$ 节点,  $m$ 条边,  $k$ 层,  $st$ ,  $ed$ 作为起点终点,  $u$ 到 $v$ 存在双向边权为 $w$ 
4 */
5
6 /*
7 考虑分层图加dj解决, 将图分成 $k + 1$ 层, 只能从小往大或者同层松弛操作, 一旦弹出即为最小
8 细节: 点是 $0 \sim n-1$ , 所以初始化的时候小心
9 无向图记得建立双向边
10 */
11 #include<bits/stdc++.h>
12 using namespace std;
13 using ll = long long;
14 const int inf = INT_MAX;
15 const int N=1e4+10,P=1e9+7;
16
17 struct edge{int v,w;};
18
19 vector<edge>vr[N];
20
21 int n,m,k,st,ed,dis[N][11],vis[N][11];
22
23 int dj()
24 {
25     for(int i = 0;i <= k;++ i)
26         for(int j = 0;j <= n;++ j)dis[j][i] = inf;
27
28     for(int i = 0;i <= k;++ i)dis[st][i] = 0;
29
30     priority_queue<pair<ll,pair<int,int>>>pq;
31     pq.push({0,{0,st}});
32     while(!pq.empty())
33     {
34         auto _ = pq.top();pq.pop();
35         int cnt = -_.second.first,u = _.second.second;
36         if(vis[u][cnt])continue;
37         if(u == ed)return dis[u][cnt];
38         vis[u][cnt] = 1;
39         for(auto ed : vr[u])
40         {
41             int v = ed.v,w = ed.w;
42             if(cnt + 1 <= k && dis[u][cnt] < dis[v][cnt + 1])dis[v][cnt + 1] = dis[u]
43 [cnt],pq.push({- dis[v][cnt + 1],{- cnt - 1,v}});
44             if(dis[u][cnt] < dis[v][cnt] - w)dis[v][cnt] = dis[u][cnt] + w,pq.push({- dis[v]
45 [cnt],{- cnt,v}});
46         }
47     }
48     return -1;
49 }
50
51 void solve()
52 {
```

```
51     cin >> n >> m >> k;
52     cin >> st >> ed;
53
54     while(m --)
55     {
56         int u,v,w;cin >> u >> v >> w;
57         vr[u].push_back({v,w});
58         vr[v].push_back({u,w});
59     }
60
61     cout << dj();
62 }
63
64 int main(){
65     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
66     int _ = 1;
67     //cin >> _;
68     while(_--)solve();
69     return 0;
70 }
```

图论\最短路\Floyd\Floyd.cpp

```
1 //1121 蓝桥公园
2 //O(n的三次方)
3 //用邻接矩阵储存
4 #include<bits/stdc++.h>
5 using namespace std;
6 using ll = long long;
7 const int N = 1e5 + 10, P = 1e9 + 7;
8
9 int n;
10 ll dis[405][405], inf = 2e18; //dis[i][j]表示i到j的距离
11
12 void floyd()
13 {
14     //枚举以k为中转点, i到j会不会变短
15     for(int k = 1;k <= n;++ k)
16         for(int i = 1;i <= n;++ i)
17             for(int j = 1;j <= n;++ j)
18                 dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
19 }
20
21 void solve()
22 {
23     int m,q;cin >> n >> m >> q;
24
25     //初始化一下邻接矩阵, 自己到自己为0, 其他为无穷
26     for(int i = 1;i <= n;++ i)
27         for(int j = 1;j <= n;++ j)
28             dis[i][j] = i == j ? 0 : inf;
29
30     //可能有重边所以取min, 无向图取双向
31     while(m --)
32     {
33         ll u,v,w;cin >> u >> v >> w;
34         dis[u][v] = min(w,dis[u][v]);
35         dis[v][u] = min(w,dis[v][u]);
36     }
37
38     floyd();
39
40     while(q --)
41     {
42         int u,v;cin >> u >> v;
43         cout << (dis[u][v] == inf ? -1 : dis[u][v]) << endl;
44     }
45 }
46
47 int main(){
48     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
49     int _ = 1;
50     //cin >> _;
51     while(_ --)solve();
```

```
52 |     return 0;  
53 | }  
54 |
```

图论\最短路\Floyd\Floyd带路径.cpp

```
1 //1121 蓝桥公园
2 //O(n的三次方)
3 //用邻接矩阵储存
4 #include<bits/stdc++.h>
5 using namespace std;
6 using ll = long long;
7 const int N = 1e5 + 10, P = 1e9 + 7;
8
9 int n, pre[405][405];
10 ll dis[405][405], inf = 2e18; //dis[i][j]表示i到j的距离
11
12 void floyd()
13 {
14     //枚举以k为中转点, i到j会不会变短
15     for(int k = 1; k <= n; ++ k)
16         for(int i = 1; i <= n; ++ i)
17             for(int j = 1; j <= n; ++ j)
18                 if(dis[i][k] + dis[k][j] < dis[i][j])
19                 {
20                     dis[i][j] = dis[i][k] + dis[k][j];
21                     pre[i][j] = k; //记录一下插点
22                 }
23 }
24
25 void path(int i, int j)
26 {
27     if(pre[i][j] == 0) return;
28     int k = pre[i][j];
29     path(i, k);
30     cout << k; //先输出左侧插点以保证路径由a到b
31     path(k, j);
32 }
33
34 void out_path()
35 {
36     int a, b; cin >> a >> b;
37     cout << a;
38     path(a, b);
39     cout << b;
40 }
41
42 void solve()
43 {
44     int m, q; cin >> n >> m >> q;
45
46     //初始化一下邻接矩阵, 自己到自己为0, 其他为无穷
47     for(int i = 1; i <= n; ++ i)
48         for(int j = 1; j <= n; ++ j)
49             dis[i][j] = i == j ? 0 : inf;
50
51     //可能有重边所以取min, 无向图取双向
```

```
52 while(m --)
53 {
54     ll u,v,w;cin >> u >> v >> w;
55     dis[u][v] = min(w,dis[u][v]);
56     dis[v][u] = min(w,dis[v][u]);
57 }
58
59 floyd();
60
61 while(q --)
62 {
63     int u,v;cin >> u >> v;
64     cout << (dis[u][v] == inf ? -1 : dis[u][v]) << endl;
65 }
66
67
68 int main(){
69     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
70     int _ = 1;
71     //cin >> _;
72     while(_ --)solve();
73     return 0;
74 }
75
```

图论\最短路\Johnson\Johnson.cpp

```
1 //  
2 #include<bits/stdc++.h>  
3 using namespace std;  
4 using ll = long long;  
5 const int inf = 1e9;  
6 const int N=3e3+10,P=1e9+7;  
7  
8 struct edge{int v,w;};  
9  
10 vector<edge>vr[N];  
11  
12 int cnt[N],h[N],n,dis[N];  
13  
14 bool spfa(int x)  
15 {  
16     bitset<N>vis;  
17     for(int i = 1;i <= n;++ i)h[i] = inf;  
18  
19     queue<int>q;  
20     q.push(x),vis[0] = 1,h[0] = 0;  
21     while(!q.empty())  
22     {  
23         int u = q.front();q.pop();  
24         vis[u] = 0;  
25         for(auto ed : vr[u])  
26         {  
27             int v = ed.v,w = ed.w;  
28             if(h[u] + w < h[v])  
29             {  
30                 h[v] = h[u] + w;  
31                 ++ cnt[v];  
32                 if(cnt[v] >= n) return 0;  
33                 if(!vis[v])q.push(v),vis[v] = 1;  
34             }  
35         }  
36     }  
37     return 1;  
38 }  
39  
40 void dj(int x)  
41 {  
42     for(int i = 1;i <= n;++ i)dis[i] = inf;  
43     bitset<N>vis;  
44     dis[x] = 0;  
45  
46     priority_queue<pair<int,int>>pq;  
47     pq.push({0,x});  
48     while(!pq.empty())  
49     {  
50         int u = pq.top().second;pq.pop();  
51         if(vis[u])continue;
```

```

52     vis[u] = 1;
53     for(auto ed : vr[u])
54     {
55         int v = ed.v,w = ed.w;
56         if((ll)dis[u] + w < dis[v])
57         {
58             dis[v] = dis[u] + w;
59             pq.push({- dis[v],v});
60         }
61     }
62 }
63 }
64
65 void solve()
66 {
67     int m;cin >> n >> m;
68     while(m --)
69     {
70         int u,v,w;cin >> u >> v >> w;
71         vr[u].push_back({v,w});
72     }
73
74     for(int i = 1;i <= n;++ i)vr[0].push_back({i,0});
75
76     if(!spfa(0))
77     {
78         cout << -1 << "\n";
79         return;
80     }
81
82 //for(int i = 1;i <= n;++ i)cout << h[i] << " ";
83
84     for(int i = 1;i <= n;++ i)
85     {
86         for(auto &ed : vr[i])
87             ed.w += h[i] - h[ed.v];
88
89         for(int i = 1;i <= n;++ i)
90         {
91             ll ans = 0;
92             dj(i);
93             for(int j = 1;j <= n;++ j)
94             {
95                 if(dis[j] == inf)ans += (ll)j * inf;
96                 else ans += (ll)j * (dis[j] + h[j] - h[i]);
97             }
98             cout << ans << "\n";
99         }
100    }
101
102 int main(){
103     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
104     int _ = 1;
105     //cin >> _;
106     while(_--)solve();

```

```
106 |     return 0;  
107 | }
```

图论\最短路\SPFA\spfa判负环.cpp

```
1 //P3385 【模板】负环
2 //可以处理负环
3 //O(nm)
4 #include<bits/stdc++.h>
5 using namespace std;
6 using ll = long long;
7 const int inf = INT_MAX;
8 const int N=2e3+10,P=1e9+7;
9
10 struct edge{int v,w;};
11
12 vector<edge>vr[N];
13 int dis[N],cnt[N],n;
14 queue<int>q;
15 bitset<N>vis;
16
17 void clear_()
18 {
19     for(int i = 1;i <= n;++ i)
20         vr[i].clear(),vis[i] = 0,cnt[i] = 0;
21     while(!q.empty())q.pop();
22 }
23
24 int spfa(int st)
25 {
26     for(int i = 1;i <= n;++ i)dis[i] = inf;
27     q.push(st);vis[st] = 1;dis[st] = 0;
28     while(!q.empty())
29     {
30         int u = q.front();q.pop();vis[u] = 0;
31         for(auto ed : vr[u])
32         {
33             int v = ed.v,w = ed.w;
34             if(dis[u] + w < dis[v])
35             {
36                 dis[v] = dis[u] + w;
37                 cnt[v] = cnt[u] + 1;
38                 if(cnt[v] >= n) return 1;//当源点到该点的步数超过n了那便是有负环了
39                 if(!vis[v])q.push(v),vis[v] = 1;
40             }
41         }
42     }
43     return 0;
44 }
45
46 void solve()
47 {
48     int m;cin >> n >> m;
49
50     while(m --)
51     {
```

```
52     int u,v,w;cin >> u >> v >> w;
53     vr[u].push_back({v,w});
54     if(w >= 0)vr[v].push_back({u,w});
55 }
56
57 if(spfa(1))cout << "YES\n";
58 else cout << "NO\n";
59 clear_();
60 }
61
62 int main(){
63     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
64     int _ = 1;
65     cin >> _;
66     while(_--)solve();
67     return 0;
68 }
```

图论\最短路\SPFA\spfa判负环带路径.cpp

```
1 //P3385 【模板】负环
2 //可以处理负环
3 #include<bits/stdc++.h>
4 using namespace std;
5 using ll = long long;
6 const int inf = INT_MAX;
7 const int N=2e3+10,P=1e9+7;
8
9 struct edge{int v,w;};
10
11 vector<edge>vr[N];
12 int dis[N],cnt[N],n,pre[N];
13 queue<int>q;
14 bitset<N>vis;
15
16 void clear_()
17 {
18     for(int i = 1;i <= n;++ i)
19         vr[i].clear(),vis[i] = 0,cnt[i] = 0;
20     while(!q.empty())q.pop();
21 }
22
23 int spfa(int st)
24 {
25     for(int i = 1;i <= n;++ i)dis[i] = inf;
26     q.push(st);vis[st] = 1;dis[st] = 0;
27     while(!q.empty())
28     {
29         int u = q.front();q.pop();vis[u] = 0;
30         for(auto ed : vr[u])
31         {
32             int v = ed.v,w = ed.w;
33             if(dis[u] + w < dis[v])
34             {
35                 dis[v] = dis[u] + w;
36                 pre[v] = u;//同理记录前驱点
37                 cnt[v] = cnt[u] + 1;
38                 if(cnt[v] >= n) return 1;//当源点到该点的步数超过n了那便是有负环了
39                 if(!vis[v])q.push(v),vis[v] = 1;
40             }
41         }
42     }
43     return 0;
44 }
45
46 void dfs_path(int x,int st)
47 {
48     if(x == st)//一直通过dfs找到起点后递归输出
49     {
50         cout << x;
51         return;
```

```
52 }
53 dfs_path(pre[x],st);//递归到结束后再往下执行输出路径
54 cout << x;
55 }
56
57 void solve()
58 {
59     int m;cin >> n >> m;
60
61     while(m --)
62     {
63         int u,v,w;cin >> u >> v >> w;
64         vr[u].push_back({v,w});
65         if(w >= 0)vr[v].push_back({u,w});
66     }
67
68     if(spfa(1))cout << "YES\n";
69     else cout << "NO\n";
70     clear_();
71 }
72
73 int main(){
74     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
75     int _ = 1;
76     cin >> _;
77     while(_--)solve();
78     return 0;
79 }
```

图论\图上 **bfs**可行路径的方案数.cpp

```
1 //蓝桥 3352 可行路径的方案数
2 /*
3 n个节点m条边求1到n的最短路径方案数
4 */
5
6 /*
7 树上dp，松弛操作即可，当更新成功时，直接重置v的dp值
8 当两者相等时v的dp值加上u的即可，最后输出dp[n]即为所求
9 */
10 #include<bits/stdc++.h>
11 using namespace std;
12 using ll =long long;
13 const int N=2e5+10,P=1e9+7;
14
15 vector<int>v[N];
16 ll dp[N],d[N];
17
18 void bfs()
19 {
20     bitset<N>vis;
21     queue<int>q;
22     q.push(1);
23     memset(d,0x3f,sizeof(d));
24     d[1] = 0;
25     dp[1] = 1;
26     while(!q.empty())
27     {
28         int x = q.front();q.pop();
29         if(vis[x])continue;
30         vis[x] = 1;
31
32         for(auto y : v[x])
33         {
34             if(vis[y])continue;
35             if(d[y] < d[x] + 1)continue;
36             else if(d[y] > d[x] + 1)
37             {
38                 d[y] = d[x] + 1;
39                 dp[y] = dp[x];
40             }
41             else
42                 dp[y] = (dp[x] + dp[y]) % P;
43             q.push(y);
44         }
45     }
46
47 void solve()
48 {
49     int n,m;cin >> n >> m;
50     while(m--)
51     {
```

```
52     int a,b;cin >> a >> b;
53     v[a].push_back(b);
54     v[b].push_back(a);
55 }
56 bfs();
57 //for(int i = 1;i <= n;++ i)cout << dp[i] << " ";
58 cout << dp[n];
59 }
60
61 int main(){
62     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
63     int _ = 1;
64     //cin >> _;
65     while(_--)solve();
66     return 0;
67 }
```

图论\图上dfs帮派弟位.cpp

```
1 //蓝桥 3891 帮派弟位
2 /*
3 按子树大小为第一关键词序号为第二关键词输出序号m的排名
4 */
5
6 /*
7 先dfs到底，初始siz为1，然后自下而上合并子树大小，再排个序即可
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll =long long;
12 const int N=1e5+10,P=1e9+7;
13
14 int siz[N];
15 vector<int>v[N];
16 pair<int,int>v1[N];
17
18 void dfs(int x)
19 {
20     for(auto y : v[x])
21     {
22         dfs(y); //先找子树的大小
23         siz[x] += siz[y]; //合并到父亲上
24     }
25 }
26
27 void solve()
28 {
29     int n,m;cin >> n >> m;
30     for(int i = 1;i < n;++ i)
31     {
32         int a,b;cin >> a >> b;
33         v[b].push_back(a); //将b的儿子压入，v用来记录儿子
34     }
35     for(int i = 1;i <= n;++ i)siz[i] = 1;
36     dfs(1);
37     for(int i = 1;i <= n;++ i)v1[i] = {-siz[i] + 1,i};
38     sort(v1 + 1,v1 + n + 1);
39     for(int i = 1;i <= n;++ i)
40     {
41         if(v1[i].second == m)
42         {
43             cout << i;
44             return;
45         }
46     }
47
48 int main(){
49     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
50     int _ = 1;
51     //cin >> _;
52     while(_--)solve();
53 }
```

```
52 |     return 0;  
53 | }
```

字符串\01tire.cpp

```
1 //19721 最大异或结点
2 /*
3 n个点找到两个不直接相连的节点异或的最大值
4 */
5
6 /*
7 在对某个点找最大时先移除与他相连的点，查询完再把这些点加入进行下一步查询
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll = long long;
12 const int M=30e5+10,N=1e5+10;
13
14 int son[M][2],tot=1,ans,e[M],a[N];
15 vector<int>g[N];
16
17 //删去树中x节点的影响
18 void move(int x){
19     int o = 1;
20     e[o]--;
21     for(int i = 30;i >= 0;-- i)
22     {
23         int y = x >> i & 1;
24         o=son[o][y];
25         e[o]--;
26     }
27 }
28
29 void insert(int x){
30     int o = 1;
31     e[o]++;
32     for(int i = 30;i >= 0;-- i)
33     {
34         int y = x >> i & 1;
35         if(!son[o][y])son[o][y] = ++ tot;//动态开点
36         o = son[o][y];
37         e[o]++;
38     }
39     //++ en[o];有需要可以加上这个记录结尾
40 }
41
42 int query(int x){
43     int res = 0,o = 1;
44     for(int i = 30;i >= 0;-- i)
45     {
46         int y = x >> i & 1;
47         if(son[o][!y] && e[o])o=son[o][!y],res |= (1 << i);//从高位往低位贪心的走相反值
48         else o = son[o][y];
49     }
50     return res;
51 }
```

```
52
53 int main(){
54     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
55     int n;cin >> n;
56     for(int i = 1;i <= n;++ i)
57     {
58         cin >> a[i];
59         insert(a[i]);
60     }
61
62     //记录相连的点方便从树中剔除
63     for(int i = 1;i <= n;++ i)
64     {
65         int x;cin >> x;
66         if(x== -1)continue;
67         g[i].push_back(a[x]);
68         g[x].push_back(a[i]);
69     }
70
71     //剔除查询最大值后再插入进行下一次查询
72     for(int i = 1;i <= n;++ i)
73     {
74         for(auto j : g[i])move(j);
75         ans=max(ans,query(a[i]));
76         for(auto j : g[i])insert(j);
77     }
78     cout << ans;
79     return 0;
80 }
```

字符串\哈希匹配.cpp

```
1 //2047.斤斤计较的小Z
2 #include<bits/stdc++.h>
3 using namespace std;
4 using ull = unsigned long long;
5 const int N=1e6+10,P=1e9+7;
6
7 ull h[N],h_ans,ans,base = 7,op = 1;
8
9 void solve()
10 {
11     string s,p;cin >> s >> p;
12     s = ' ' + s;
13     p = ' ' + p;
14     int n = s.length() - 1,m = p.length() - 1;
15     for(int i = 1;i <= n;++ i)h[i] = h[i - 1] * base + s[i];
16     for(int i = 1;i <= m;++ i)h_ans = h_ans * base + p[i],op *= base;//p长度不固定的话预处理一下b[i]=b的i次方
17
18     for(int i = m;i <= n;++ i)if(h[i] - h[i-m]*op == h_ans)++ ans;
19     cout << ans;
20 }
21
22 int main(){
23     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
24     int _ = 1;
25     //cin >> _;
26     while(_--)solve();
27     return 0;
28 }
```

字符串\字典树.cpp

```
1 //P8306 【模板】字典树
2 /*
3  查询有多少以s为前缀
4 */
5
6 /*
7 建树然后记录pass，查询的时候让s走到底再返回pass即可
8 */
9 #include<bits/stdc++.h>
10 using namespace std;
11 using ll = long long;
12 const ll inf = 2e18;
13 const int N=3e6+10,P=1e9+7;
14
15 int son[N][63],tot = 1,pass[N];//动态开点，还可以加上一个en数字记录结尾个数
16
17 //映射一下字母到数字尽量减少空间的浪费
18 int f(char x)
19 {
20     if('A' <= x && x <= 'Z')return (x - 'A' + 26);
21     if('a' <= x && x <= 'z')return (x - 'a');
22     return (x - '0' + 52);
23 }
24
25 void insert(string s)
26 {
27     int o = 1;
28     for(int i = 0;s[i];++ i)
29     {
30         int x = f(s[i]);
31         if(!son[o][x])son[o][x] = ++ tot;
32         o = son[o][x];
33         ++ pass[o];
34     }
35     //++en[o];en数组加在这里记录结尾
36 }
37
38 int query(string s)
39 {
40     int o = 1;
41     for(int i = 0;s[i];++ i)
42     {
43         int x = f(s[i]);
44         if(!son[o][x])return 0;
45         o = son[o][x];
46     }
47     return pass[o];
48 }
49
50 //清空树
51 void clear()
```

```
52 {
53     for(int i = 1;i <= tot;++ i)
54     {
55         pass[i] = 0;
56         for(int j = 0;j <= 61;++ j)son[i][j] = 0;
57     }
58     tot = 1;
59     return;
60 }
61
62 void solve()
63 {
64     int n,q;cin >> n >> q;
65     while(n --)
66     {
67         string s;cin >> s;
68         insert(s);
69     }
70     while(q --)
71     {
72         string s;cin >> s;
73         cout << query(s) << endl;
74     }
75     clear();
76 }
77
78 int main(){
79     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
80     int _ = 1;
81     cin >> _;
82     while(_--)solve();
83     return 0;
84 }
```

字符串\KMP.cpp

```
1 //P3375 【模板】KMP
2 #include<bits/stdc++.h>
3 using namespace std;
4 using ll =long long;
5 const int N=1e6+10,P=1e9+7;
6
7 int nx[N];//next数组也叫失配数组
8
9 void solve()
10 {
11     string s,p;cin >> s >> p;
12     s = ' ' + s;
13     p = ' ' + p;
14     int n = s.length() - 1,m = p.length() - 1;
15
16     //自我匹配得到失配数组
17     for(int i = 2,j = 0;i <= m;++ i)
18     {
19         while(j && p[i] != p[j+1])j = nx[j];
20         if(p[i] == p[j+1])++ j;
21         nx[i] = j;
22     }
23
24     //利用nx数组加速匹配
25     for(int i = 1,j = 0;i <= n;++ i)
26     {
27         while(j && s[i] != p[j+1])j = nx[j];
28         if(s[i] == p[j+1])++ j;
29         if(j == m)cout << i - m + 1 << endl;//完全匹配
30     }
31
32     for(int i = 1;i <= m;++ i)cout << nx[i] << " ";
33 }
34
35 int main(){
36     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
37     int _ = 1;
38     //cin >> _;
39     while(_--)solve();
40     return 0;
41 }
```

字符串\Manacher.cpp

```
1 //1225.最长回文子串
2 #include<bits/stdc++.h>
3 using namespace std;
4 using ll =long long;
5 const int N=1e6+10,P=1e9+7;
6
7 int p[N];//p数组里记录的以i为中心的最长回文串的长度
8 char s[N];
9
10 void solve()
11 {
12     cin >> s + 1;
13     int n = strlen(s + 1);
14     for(int i = ((n << 1) | 1);i >= 1;-- i)s[i] = (i & 1) ? '#' : s[i >> 1];//预处理s应对偶数
字符串
15     int C = 0,R = 0;
16     for(int i = 1;i <= ((n << 1) | 1);++ i)
17     {
18         p[i] = i < R ? min(R-i,p[2*C-i]) : 1;
19         while(s[i+p[i]] == s[i-p[i]])++p[i];
20         if(i + p[i] > R)C = i,R = i + p[i];
21     }
22     cout << *max_element(p+1,p+2*n+1)-1;
23 }
24
25 int main(){
26     ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
27     int _ = 1;
28     //cin >> _;
29     while(_--)solve();
30     return 0;
31 }
```