

ACM模板

目录

一、STL

二、字符串

- 1、Trie
- 2、KMP
- 3、Manacher算法
- 4、AC自动机
- 5、最小表示法

三、数据结构

- 1、带权并查集
- 2、树状数组
- 3、线段树
- 4、平衡树-Treap
- 5、重链剖分
- 6、ST表
- 7、分块
- 8、树套树
- 9、可并堆
- 10、对顶堆

四、图论

- 1、二分图判断
- 2、二分图最大匹配
- 3、最短路-Dijkstra
- 4、欧拉路

- 5、负环
- 6、差分约束
- 7、缩点
- 8、2-SAT
- 9、割点
- 10、边双连通
- 11、最小生成树-Kruskal
- 12、Kruskal重构树
- 13、A*
- 14、IDA*
- 15、LCA

五、网络流

- 1、最大流
- 2、费用流
- 3、无源汇上下界可行流
- 4、有源汇上下界可行流
- 5、有源汇上下界最大流

六、数学

数论：

- 1、乘法逆元
- 2、裴蜀定理&扩展欧几里得
- 3、中国剩余定理
- 4、扩展欧拉定理
- 5、威尔逊定理
- 6、原根
- 7、离散对数
- 8、Miller-rabin
- 9、Pollard-Rho
- 10、积性函数

组合数学：

11、组合数学

12、范德蒙德卷积

数值算法：

13、数值积分

代数学：

14、线性代数

多项式与生成函数：

15、多项式

概率论：

16、常用分布

七、计算几何

八、杂项

1、生成树计数

2、点分治

输入输出优化

快读

```
template <typename T> inline void read(T& t) {
    int f = 0, c = getchar(); t = 0;
    while (!isdigit(c)) f |= c == '-' , c = getchar();
    while (isdigit(c)) t = t * 10 + c - 48, c = getchar();
    if (f) t = -t;
}

template <typename T> void print(T x) {
    if (x < 0) x = -x, putchar('-');
    if (x > 9) print(x / 10);
    putchar(x % 10 + 48);
}
```

cin同步

```
ios::sync_with_stdio(false) ;
cin.tie(0);
```

一、STL

随机数

```
mt19937_64 rng(random_device{}());
uniform_int_distribution<int>dist(0,LLONG_MAX);
```

vector

`size()` 返回元素个数
`empty()` 返回是否为空
`clear()` 清空
`front() / back()`
`push_back() / pop_back()`
`begin() / end()`
[]
支持比较运算，按字典序

pair

```
pair<int, int>
first, 第一个元素
second, 第二个元素
支持比较运算，以first为第一关键字，以second为第二关键字（字典序）
```

string

```
string
size() / length() 返回字符串长度
empty()
clear()
substr(起始下标, (子串长度)) 返回子串
c_str() 返回字符串所在字符数组的起始地址
```

queue

`size()`
`empty()`
`push()` 向队尾插入一个元素
`front()` 返回队头元素
`back()` 返回队尾元素
`pop()` 弹出队头元素

priority_queue

```
push() 插入一个元素  
top() 返回堆顶元素  
pop() 弹出堆顶元素  
定义成小根堆的方式: priority_queue<int, vector<int>, greater<int>> q;
```

stack

```
size()  
empty()  
push() 向栈顶插入一个元素  
top() 返回栈顶元素  
pop() 弹出栈顶元素
```

deque

```
size()  
empty()  
clear()  
front()/back()  
push_back()/pop_back()  
push_front()/pop_front()  
begin()/end()  
[]
```

set, map类

(multiset, multimap,unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表)

```
size()  
empty()  
clear()  
begin()/end()  
++, -- 返回前驱和后继, 时间复杂度 O(logn)  
  
set/multiset  
insert() 插入一个数  
find() 查找一个数  
count() 返回某一个数的个数  
erase()  
    (1) 输入是一个数x, 删除所有x O(k + logn)  
    (2) 输入一个迭代器, 删除这个迭代器  
lower_bound()/upper_bound()  
lower_bound(x) 返回大于等于x的最小的数的迭代器  
upper_bound(x) 返回大于x的最小的数的迭代器  
  
map/multimap  
insert() 插入的数是一个pair
```

```
erase() 输入的参数是pair或者迭代器  
find()  
[] 注意multimap不支持此操作。 时间复杂度是  $O(\log n)$   
lower_bound()/upper_bound()  
  
unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表  
和上面类似, 增删改查的时间复杂度是  $O(1)$   
不支持 lower_bound()/upper_bound(), 迭代器的++, --
```

bitset

```
bitset<10000> s;  
~, &, |, ^  
>>, <<  
==, !=  
[]  
  
count() 返回有多少个1  
  
any() 判断是否至少有一个1  
none() 判断是否全为0  
  
set() 把所有位置成1  
set(k, v) 将第k位变成v  
reset() 把所有位变成0  
flip() 等价于~  
flip(k) 把第k位取反
```

位运算

```
int __builtin_ffs(int x) 返回x的二进制末尾最后一个1的位置, 位置的编号从1开始(最低位编号为1)。当x为0时返回0。  
  
int __builtin_clz(unsigned int x) 返回x的二进制的前导0的个数。当x为0时, 结果未定义。  
  
int __builtin_ctz(unsigned int x) 返回x的二进制末尾连续0的个数。当x为0时, 结果未定义。  
  
int __builtin_clrsb(int x) 当x的符号位为0时返回x的二进制的前导0的个数减一, 否则返回x的二进制的前导1的个数减一。  
  
int __builtin_popcount(unsigned int x) 返回x的二进制中1的个数。  
  
int __builtin_parity(unsigned int x) 判断x的二进制中1的个数的奇偶性。
```

可并堆



push(): 向堆中压入一个元素，返回该元素位置的迭代器。

pop(): 将堆顶元素弹出。

top(): 返回堆顶元素。

size(): 返回元素个数。

empty(): 返回是否非空。

modify(point_iterator, const key): 把迭代器位置的 **key** 修改为传入的 **key**，并对底层储存结构进行排序。

erase(point_iterator): 把迭代器位置的键值从堆中擦除。

join(__gnu_pbds :: priority_queue &other): 把 **other** 合并到 ***this** 并把 **other** 清空。

```
#include <algorithm>
#include <cstdio>
#include <ext/pb_ds/priority_queue.hpp>
#include <iostream>
using namespace __gnu_pbds;
// 由于面向Oier，本文以常用堆 : pairing_heap_tag作为范例
// 为了更好的阅读体验，定义宏如下：
#define pair_heap __gnu_pbds ::priority_queue<int>
pair_heap q1; // 大根堆，配对堆
pair_heap q2;
pair_heap ::point_iterator id; // 一个迭代器

int main() {
    id = q1.push(1);
    // 堆中元素 : [1];
    for (int i = 2; i <= 5; i++) q1.push(i);
    // 堆中元素 : [1, 2, 3, 4, 5];
    std ::cout << q1.top() << std ::endl;
    // 输出结果 : 5;
    q1.pop();
    // 堆中元素 : [1, 2, 3, 4];
    id = q1.push(10);
    // 堆中元素 : [1, 2, 3, 4, 10];
    q1.modify(id, 1);
    // 堆中元素 : [1, 1, 2, 3, 4];
    std ::cout << q1.top() << std ::endl;
    // 输出结果 : 4;
    q1.pop();
    // 堆中元素 : [1, 1, 2, 3];
    id = q1.push(7);
    // 堆中元素 : [1, 1, 2, 3, 7];
    q1.erase(id);
    // 堆中元素 : [1, 1, 2, 3];
    q2.push(1), q2.push(3), q2.push(5);
    // q1中元素 : [1, 1, 2, 3], q2中元素 : [1, 3, 5];
    q2.join(q1);
    // q1中无元素，q2中元素 : [1, 1, 1, 2, 3, 3, 5];
}
```

平衡树

insert(x): 向树中插入一个元素 x , 返回 `std::pair<point_iterator, bool>`。
erase(x): 从树中删除一个元素/迭代器 x , 返回一个 `bool` 表明是否删除成功。
order_of_key(x): 返回 x 以 `Cmp_Fn` 比较的排名。
find_by_order(x): 返回 `Cmp_Fn` 比较的排名所对应元素的迭代器。
lower_bound(x): 以 `Cmp_Fn` 比较做 `lower_bound`, 返回迭代器。
upper_bound(x): 以 `Cmp_Fn` 比较做 `upper_bound`, 返回迭代器。
join(x): 将 x 树并入当前树, 前提是两棵树的类型一样, x 树被删除。
split(x,b): 以 `Cmp_Fn` 比较, 小于等于 x 的属于当前树, 其余的属于 b 树。
empty(): 返回是否为空。
size(): 返回大小。

```

// Common Header Simple over C++11
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
typedef pair<int, int> pii;
#define pb push_back
#define mp make_pair
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
__gnu_pbds ::tree<pair<int, int>, __gnu_pbds::null_type, less<pair<int, int> >,
    __gnu_pbds::rb_tree_tag,
    __gnu_pbds::tree_order_statistics_node_update>
trr;

int main() {
    int cnt = 0;
    trr.insert(mp(1, cnt++));
    trr.insert(mp(5, cnt++));
    trr.insert(mp(4, cnt++));
    trr.insert(mp(3, cnt++));
    trr.insert(mp(2, cnt++));
    // 树上元素 {{1,0},{2,4},{3,3},{4,2},{5,1}}
    auto it = trr.lower_bound(mp(2, 0));
    trr.erase(it);
    // 树上元素 {{1,0},{3,3},{4,2},{5,1}}
    auto it2 = trr.find_by_order(1);
    cout << (*it2).first << endl;
    // 输出排名 0 1 2 3 中的排名 1 的元素的 first:3
    int pos = trr.order_of_key(*it2);
    cout << pos << endl;
    // 输出排名
    decltype(trr) newtr;
    trr.split(*it2, newtr);
    for (auto i = newtr.begin(); i != newtr.end(); ++i) {
        cout << (*i).first << ' ';
    }
    cout << endl;
    // {4,2},{5,1} 被放入新树
    trr.join(newtr);
    for (auto i = trr.begin(); i != trr.end(); ++i) {
        cout << (*i).first << ' ';
    }
    cout << endl;
}

```

```
cout << newtr.size() << endl;
// 将 newtr 树并入 trr 树, newtr 树被删除。
return 0;
}
```

可持久化平衡树

rope

同样属于 C++ 拓展库，是一棵可持久化平衡树。

需要头文件 `#include<ext/rope>` 和名字空间 `__gnu_cxx`。

变量定义 `rope<int> *h[1000007];`。

可持久化 `h[i]=new rope<int>(*h[j]);`，其中

为当前版本，

为某个历史版本。时间复杂度

。

函数调用

- * `h[i]->push_back(val)` 在 `h[i]` 的末尾加入 `val`。
- * `h[i]->at(k)` 访问第 `k` 个元素。
- * `h[i]->replace(pos,val)` 将位置为 `pos` 的元素换成 `val`。
- * `h[i]->size()` 返回 `h[i]` 的元素个数。
- * `h[i]->insert(pos,val)` 在 `pos` 位置插入 `val`。
- * `h[i]->erase(pos,k)` 从 `pos` 位置向后删除 `k` 个元素。
- * `h[i]->substr(pos,k)` 从 `pos` 位置开始提取 `k` 个元素。
- * `h[i]->copy(pos,k,q)` 将从 `pos` 位置向后 `x` 个元素拷贝到 `q` 中。

和 `vector` 基本一致。

二、字符串

1、Trie

```
void insert(string s){
    int cur=0;
    for(auto x:s){
        if(!nxt[cur][x]){
            nxt[cur][x]=++idx;
        }
        cur=nxt[cur][x];
    }
}
```

2、KMP

```
int nxt[N];
string s,t;

void getnxt(string s){
    int n=s.size()-1;
    for(int i=2,j=0;i<=n;++i){
        while(j && s[i]!=s[j+1]) j=nxt[j];
        if(s[i]==s[j+1]) j++;
        nxt[i]=j;
    }
}

void kmp(string s,string t){
    int n=s.size()-1,m=t.size()-1;
    for(int i=1,j=0;i<=n;++i){
        while(j && s[i]!=t[j+1]) j=nxt[j];
        if(s[i]==t[j+1]) j++;
        if(j==m){
            cout<<i-m+1<<endl;
            j=nxt[j];
        }
    }
}
```

3、Manacher

```
struct Manacher{
    int sz;
    vector<int>p;
    string s,t;
    Manacher(string _s){
        s=_s;
        t="#";
        for(auto i:_s) t+="+",t+=i;
        t+="!";
    }
}
```

```

        p.assign(t.size()+1,111);
        for(int i=1,mid=0;t[i];++i){
            if(i<mid+p[mid]) p[i]=min(p[mid*2-i],mid+p[mid]-i);
            while(t[i-p[i]]==t[i+p[i]]) p[i]++;
            if(i+p[i]>mid+p[mid]) mid=i;
        }
    }
    bool ispa(int l,int r){
        l=2*l+2,r=2*r+2;
        int mid=l+r>>1;
        if(mid+p[mid]>r) return 1;
        else return 0;
    };
}

```

4、AC自动机

```

#include<bits/stdc++.h>
#define int long long

using namespace std;
const int N=2e6+8;
string s,t;
int n,cnt,vis[N],ans,in[N],m[N];

struct trie{
    int son[26],fail,flag,ans;
}trie[N];
queue<int>q;

void insert(int num){
    int p=1,len=s.size();
    for(int i=0;i<len;i++){
        int v=s[i]-'a';
        if(!trie[p].son[v]) trie[p].son[v]=++cnt;
        p=trie[p].son[v];
    }
    if(!trie[p].flag) trie[p].flag=num;
    m[num]=trie[p].flag;
}

void getFail(){
    for(int i=0;i<26;i++) trie[0].son[i]=1;
    q.push(1);
    while(!q.empty()){
        int u=q.front();q.pop();
        int Fail=trie[u].fail;
        for(int i=0;i<26;i++){
            int v=trie[u].son[i];
            if(!v){
                trie[u].son[i]=trie[Fail].son[i];
                continue;
            }
            trie[v].fail=trie[Fail].son[i];
            in[trie[v].fail]++;
            q.push(v);
        }
    }
}

```

```

        }
    }

void topu(){
    for(int i=1;i<=cnt;i++){
        if(in[i]==0) q.push(i);
    while(!q.empty()){
        int u=q.front();q.pop();
        vis[trie[u].flag]=trie[u].ans;
        int v=trie[u].fail;in[v]--;
        trie[v].ans+=trie[u].ans;
        if(in[v]==0)q.push(v);
    }
}

void query(){
    int u=1,len=t.size();
    for(int i=0;i<len;i++){
        u=trie[u].son[t[i]-'a'];
        trie[u].ans++;
    }
}

signed main(){
    cin>>n;
    cnt=1;
    for(int i=1;i<=n;i++){
        cin>>s;
        insert(i);
    }
    getFail();
    cin>>t;
    query();
    topu();
    for(int i=1;i<=n;i++) cout<<vis[m[i]]<<"\n";
}
}

```

5、最小表示法

```

string getmin(string &s) {
    int n=s.size();
    s+=s;
    s+="+"+s;
    int i=1,j=2;
    while(j<=n){
        int k=0;
        while(k<n && s[i+k]==s[j+k]) k++;
        if(s[i+k]>s[j+k]) i+=k+1;
        else j+=k+1;
        if(i==j) j++;
        if(i>j) swap(i,j);
    }
    string ans=s.substr(i,n);
    return ans;
}

```

6、双哈希

```
const int N=1e6+10;
const int mod1=3145739;//如有误差手动修改
const int mod2=6291469;
struct Hash
{
    ll has1[N],has2[N];
    ll bas1[N],bas2[N];
    ll p1,p2;
    void init(char s[]){
        p1=2333;//如有误差手动修改
        p2=17;
        has1[0]=has2[0]=0;
        bas1[0]=bas2[0]=1;
        int len=strlen(s+1);
        for(int i=1;i<len;++i){//多串hash可以这块预处理
            bas1[i]=(bas1[i-1]*p1)%mod1;
            bas2[i]=(bas2[i-1]*p2)%mod2;
        }
        for(int i=1;i<len;++i){
            has1[i]=(has1[i-1]*p1+s[i])%mod1;
            has2[i]=(has2[i-1]*p2+s[i])%mod2;
        }
    }
    ll gethash1(int r,int len){//返回以r结尾长度len的子串has1值
        return ((has1[r]-has1[r-len]*bas1[len])%mod1+mod1)%mod1;
    }
    ll gethash2(int r,int len){//返回以r结尾长度len的子串has2值
        return ((has2[r]-has2[r-len]*bas2[len])%mod2+mod2)%mod2;
    }
}
```

三、数据结构

1、并查集

普通

```
struct DSU{
    vector<int>fa;
    DSU(int n){
        fa.resize(n+1);
        iota(fa.begin(),fa.end(),0);
    }
    int find(int x){
        return fa[x]==x?x:fa[x]=find(fa[x]);
    }
    bool same(int x,int y){
        return find(x)==find(y);
    }
    void merge(int a,int b){
        int x=find(a),y=find(b);
        if(x!=y) fa[x]=y;
    }
};
```

带权并查集

```
struct DSU{
    vector<int>fa;
    vector<int>val;
    DSU(int n){
        fa.resize(n+1);
        val.assign(n+1,0);
        iota(fa.begin(),fa.end(),0);
    }
    int find(int x){
        if(fa[x]!=x){
            int tmp=fa[x];
            fa[x]=find(fa[x]);
            val[x]+=val[tmp];
        }
        return fa[x];
    }
    void merge(int a,int b,int s){
        int x=find(a),y=find(b);
        if(x!=y){
            fa[x]=y;
            val[x]=-val[a]+val[b]+s;
        }
    }
};
```

可撤销并查集

```
struct DSU{
    vector<int>fa,sz;
    vector<int>st;
    vector<long long>tag;
    DSU(int n){
        fa.resize(n+1);
        iota(fa.begin(),fa.end(),0);
        sz.assign(n+1,1);
        tag.assign(n+1,011);
        vector<int>().swap(st);
    }
    int find(int x){return fa[x]==x?x:find(fa[x]);}
    void merge(int a,int b){
        int x=find(a),y=find(b);
        if(x!=y){
            if(sz[x]>sz[y]) swap(x,y);
            tag[x]-=tag[y];
            sz[y]+=sz[x];
            fa[x]=y;
            st.push_back(x);
        }
    }
    void regress(int lst){
        while((int)st.size()>lst){
            int x=st.back(),y=fa[x];
            tag[x]+=tag[y];
            sz[y]-=sz[x];
            fa[x]=x;
            st.pop_back();
        }
    }
};
```

2、树状数组

```
struct FenwickTree{
    int sz;
    vector<int>c;
    FenwickTree(int n=0){sz=n,c.assign(sz+1,011);}
    int lowbit(int x){return -x&x;}
    void add(int u,int k){
        for(;u<=sz;u+=lowbit(u)) c[u]+=k;
    }
    int query(int u){
        int ans=0;
        for(;u;u-=lowbit(u)) ans+=c[u];
        return ans;
    }
};
```

3、线段树

通用

```
struct SegmentTree{
    vector<int>s;
    vector<int>tag;
    SegmentTree(int _n){s.assign(_n+1<<2,011),tag.assign(_n+1<<2,011);};
    void pushup(int u){                                //}
    }
    void pushtag(int u,int cl,int cr,int k){           //}
    }
    void pushdown(int u,int cl,int cr){
        int mid=cl+cr>>1;
        pushtag(u<<1,cl,mid,tag[u]);
        pushtag(u<<1|1,mid+1,cr,tag[u]);
        tag[u]=0;
    }
    void modify(int u,int cl,int cr,int l,int r,int k){
        if(l<=cl && cr<=r){
            pushtag(u,cl,cr,k);
            return;
        }
        pushdown(u,cl,cr);
        int mid=cl+cr>>1;
        if(l<=mid) modify(u<<1,cl,mid,l,r,k);
        if(mid<r) modify(u<<1|1,mid+1,cr,l,r,k);
        pushup(u);
    }
    int query(int u,int cl,int cr,int l,int r){          //}
        if(l<=cl && cr<=r) return s[u];
        pushdown(u,cl,cr);
        int mid=cl+cr>>1;
        return ;
    }
}seg;
```

修改：区间最小值；查询：区间最小值，查询区间和。

```
int a[N],n;
struct SegTree{
    int sum,ma,cma,sma;
    int tag;
}s[N<<2];

SegTree operator+(const SegTree x,const SegTree y){
    SegTree z;
    z.sum=x.sum+y.sum;
    if(x.ma==y.ma){
        z.ma=x.ma;
        z.cma=x.cma+y.cma;
        z.sma=max(x.sma,y.sma);
    }
    else{
        z.ma=max(x.ma,y.ma);
        if(x.ma>y.ma){
```

```

        z.cma=x.cma;
        z.sma=max(x.sma,y.ma);
    }
    else{
        z.cma=y.cma;
        z.sma=max(x.ma,y.sma);
    }
}
z.tag=inf;
return z;
}

void build(int u=1,int cl=1,int cr=n){
    if(cl==cr){
        s[u]={a[cl],a[cl],1,-1,inf};
        return;
    }
    int mid=cl+cr>>1;
    build(u<<1,cl,mid);
    build(u<<1|1,mid+1,cr);
    s[u]=s[u<<1]+s[u<<1|1];
}

void pushtag(int u,int k){
    if(s[u].ma<=k) return;
    s[u].sum+=(k-s[u].ma)*s[u].cma;
    s[u].ma=s[u].tag=k;
}

void pushdown(int u){
    if(s[u].tag==inf) return;
    pushtag(u<<1,s[u].tag);
    pushtag(u<<1|1,s[u].tag);
    s[u].tag=inf;
}

void modifymin(int l,int r,int k,int u=1,int cl=1,int cr=n){
    if(k>s[u].ma) return;
    if(l<=cl && cr<=r && s[u].sma<k){
        pushtag(u,k);
        return;
    }
    pushdown(u);
    int mid=cl+cr>>1;
    if(l<=mid) modifymin(l,r,k,u<<1,cl,mid);
    if(mid<r) modifymin(l,r,k,u<<1|1,mid+1,cr);
    s[u]=s[u<<1]+s[u<<1|1];
}

int querymax(int l,int r,int u=1,int cl=1,int cr=n){
    if(l<=cl && cr<=r) return s[u].ma;
    pushdown(u);
    int mid=cl+cr>>1;
    int ans=0;
    if(l<=mid) ans=max(ans,querymax(l,r,u<<1,cl,mid));
    if(mid<r) ans=max(ans,querymax(l,r,u<<1|1,mid+1,cr));
    return ans;
}

```

```

int querysum(int l,int r,int u=1,int cl=1,int cr=n){
    if(l<=cl && cr<=r) return s[u].sum;
    pushdown(u);
    int mid=cl+cr>>1;
    int ans=0;
    if(l<=mid) ans+=querysum(l,r,u<<1,cl,mid);
    if(mid<r) ans+=querysum(l,r,u<<1|1,mid+1,cr);
    return ans;
}

```

修改：区间加，区间最小值；查询：区间和，区间最小值，区间历史最小值。

```

int n,a[N];

struct SegTree{
    int sum,ma,cma,sma,ma_;
    int t1,t1_,t2,t2_;
}s[N<<2];

SegTree operator+(const SegTree x,const SegTree y){
    SegTree z;
    z.sum=x.sum+y.sum;
    z.ma=max(x.ma,y.ma);
    z.ma_=max(x.ma_,y.ma_);
    if(x.ma==y.ma){
        z.cma=x.cma+y.cma;
        z.sma=max(x.sma,y.sma);
    }
    else if(x.ma>y.ma){
        z.cma=x.cma;
        z.sma=max(x.sma,y.ma);
    }
    else{
        z.cma=y.cma;
        z.sma=max(x.ma,y.sma);
    }
    z.t1=z.t1_=z.t2=z.t2_=0;
    return z;
}

void build(int u=1,int cl=1,int cr=n){
    if(cl==cr){
        s[u]={a[cl],a[cl],1,-inf,a[cl]};
        return;
    }
    int mid=cl+cr>>1;
    build(u<<1,cl,mid);
    build(u<<1|1,mid+1,cr);
    s[u]=s[u<<1]+s[u<<1|1];
}

void update(int u,int cl,int cr,int k1,int k1_,int k2,int k2_){
    s[u].sum+=k1*s[u].cma+k2*(cr-cl+1-s[u].cma);
    s[u].ma_=max(s[u].ma_,s[u].ma+k1_);
    s[u].ma+=k1;
}

```

```

    if(s[u].sma!=inf) s[u].sma+=k2;
    s[u].t1_=max(s[u].t1_,s[u].t1+k1_);
    s[u].t2_=max(s[u].t2_,s[u].t2+k2_);
    s[u].t1+=k1;
    s[u].t2+=k2;
}

void pushdown(int u,int cl,int cr){
    int mid=cl+cr>>1,ma=max(s[u<<1].ma,s[u<<1|1].ma);
    if(s[u<<1].ma==ma) update(u<<1,cl,mid,s[u].t1,s[u].t1_,s[u].t2,s[u].t2_);
    else update(u<<1,cl,mid,s[u].t2,s[u].t2_,s[u].t2,s[u].t2_);
    if(s[u<<1|1].ma==ma) update(u<<1|1,mid+1,cr,s[u].t1,s[u].t1_,s[u].t2,s[u].t2_);
    else update(u<<1|1,mid+1,cr,s[u].t2,s[u].t2_,s[u].t2,s[u].t2_);
    s[u].t1=s[u].t1_=s[u].t2=s[u].t2_=0;
}

void modifyadd(int l,int r,int k,int u=1,int cl=1,int cr=n){
    if(r<cl || cr<l) return;
    if(l<=cl && cr<=r){
        update(u,cl,cr,k,k,k,k);
        return;
    }
    int mid=cl+cr>>1;
    pushdown(u,cl,cr);
    modifyadd(l,r,k,u<<1,cl,mid);modifyadd(l,r,k,u<<1|1,mid+1,cr);
    s[u]=s[u<<1]+s[u<<1|1];
}

void modifymin(int l,int r,int k,int u=1,int cl=1,int cr=n){
    if(r<cl || cr<l || s[u].ma<=k) return;
    if(l<=cl && cr<=r && s[u].sma<k){
        update(u,cl,cr,k-s[u].ma,k-s[u].ma,0,0);
        return;
    }
    int mid=cl+cr>>1;
    pushdown(u,cl,cr);
    modifymin(l,r,k,u<<1,cl,mid);modifymin(l,r,k,u<<1|1,mid+1,cr);
    s[u]=s[u<<1]+s[u<<1|1];
}

int querysum(int l,int r,int u=1,int cl=1,int cr=n){
    if(r<cl || cr<l) return 0ll;
    if(l<=cl && cr<=r) return s[u].sum;
    int mid=cl+cr>>1;
    pushdown(u,cl,cr);
    return querysum(l,r,u<<1,cl,mid)+querysum(l,r,u<<1|1,mid+1,cr);
}

int querymax(int l,int r,int u=1,int cl=1,int cr=n){
    if(r<cl || cr<l) return -inf;
    if(l<=cl && cr<=r) return s[u].ma;
    int mid=cl+cr>>1;
    pushdown(u,cl,cr);
    return max(querymax(l,r,u<<1,cl,mid),querymax(l,r,u<<1|1,mid+1,cr));
}

int querymax_(int l,int r,int u=1,int cl=1,int cr=n){
    if(r<cl || cr<l) return -inf;

```

```

    if(l<=cl && cr>=r) return s[u].ma_;
    int mid=cl+cr>>1;
    pushdown(u,cl,cr);
    return max(querymax_(l,r,u<<1,cl,mid),querymax_(l,r,u<<1|1,mid+1,cr));
}

```

扫描线

```

#include<bits/stdc++.h>
#define int long long

using namespace std;
const int N=1e6+8;
int x[N<<1];

struct line{
    int l,r,h,f;
}l[N<<1];

struct SegTree{
    int len,tag;
}s[N<<2];

void pushup(int u,int cl,int cr){
    if(s[u].tag) s[u].len=x[cr+1]-x[cl];
    else s[u].len=s[u<<1].len+s[u<<1|1].len;
}

void modify(int u,int cl,int cr,int l,int r,int c){
    if(x[cr+1]<=l || r<=x[cl]) return;
    if(l<=x[cl] && x[cr+1]<=r){
        s[u].tag+=c;
        pushup(u,cl,cr);
        return;
    }
    int mid=cl+cr>>1;
    modify(u<<1,cl,mid,l,r,c);modify(u<<1|1,mid+1,cr,l,r,c);
    pushup(u,cl,cr);
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n;
    cin>>n;
    for(int i=1;i<=n;++i){
        int x1,y1,x2,y2;
        cin>>x1>>y1>>x2>>y2;
        x[2*i-1]=x1,x[2*i]=x2;
        l[2*i-1]={x1,x2,y1,1};l[2*i]={x1,x2,y2,-1};
    }
    n<<=1;
    sort(x+1,x+n+1);
    int cnt=unique(x+1,x+n+1)-x-1;
    sort(l+1,l+n+1,[&](line &x,line &y){
        return x.h<y.h;
    });
}

```

```

int res=0;
for(int i=1;i<n;++i){
    modify(1,1,cnt-1,l[i].l,l[i].r,l[i].f);
    res+=s[1].len*(l[i+1].h-l[i].h);
}
cout<<res<<endl;
}

```

可持久化线段树

```

struct SegTree{
    struct tree{
        int l,r,ma;
    }s[N<<6];
    int modify(int u,int k,int x,int cl=1,int cr=n){
        int v=++tot;
        s[v]=s[u];
        if(cl==cr){
            s[v].ma=max(s[v].ma,x);
            return v;
        }
        int mid=cl+cr>>1;
        if(k<=mid) s[v].l=modify(s[u].l,k,x,cl,mid);
        else s[v].r=modify(s[u].r,k,x,mid+1,cr);
        s[v].ma=max(s[s[v].l].ma,s[s[v].r].ma);
        return v;
    }
    int query(int u,int l,int r,int cl=1,int cr=n){
        if(!u || r<cl || cr<l) return 0;
        if(l<=cl && cr<=r) return s[u].ma;
        int mid=cl+cr>>1;
        return max(query(s[u].l,l,r,cl,mid),query(s[u].r,l,r,mid+1,cr));
    }
}seg;

```

权值线段树合并

```

int merge(int a,int b,int cl=1,int cr=N){
    if(!a || !b) return a+b;
    if(cl==cr){
        s[a].v+=s[b].v;
        return a;
    }
    int mid=cl+cr>>1;
    s[a].ls=merge(s[a].ls,s[b].ls,cl,mid);
    s[a].rs=merge(s[a].rs,s[b].rs,mid+1,cr);
    pushup(a);
    return a;
}

```

李超线段树

- 加入一个一次函数，定义域为 $[l, r]$ ；
- 给定 k ，求定义域包含 k 的所有一次函数中，在 $x = k$ 处取值最大的那个，如果有多个函数取值相同，选编号最小的。

```
int s[N<<4], cnt;
double k[N], b[N];

int cmp(double x) {
    return fabs(x)<=eps?0:(x<0?-1:1);
}

double f(int x,int p){
    return k[x]*p+b[x];
}

bool judge(int x,int y,int p){
    double fx=f(x,p),fy=f(y,p);
    return cmp(fx-fy)?fx<fy:x>y;
}

void update(int l,int r,int k,int u=1,int cl=1,int cr=40000){ \\\k为编号
    if(cr<l || r<cl) return;
    int mid=(cl+cr)>>1;
    if(l<=cl && cr<=r){
        if(judge(k,s[u],cl) && judge(k,s[u],cr)) return;
        if(judge(s[u],k,cl) && judge(s[u],k,cr)){
            s[u]=k;
            return;
        }
        if(judge(s[u],k,mid)) swap(s[u],k);
        if(judge(s[u],k,cl)) update(l,r,k,u<<1,cl,mid);
        else update(l,r,k,u<<1|1,mid+1,cr);
        return;
    }
    update(l,r,k,u<<1,cl,mid),update(l,r,k,u<<1|1,mid+1,cr);
}

int query(int x,int u=1,int cl=1,int cr=40000){
    if(x<cl || cr<x) return 0;
    if(cl==cr) return s[u];
    int mid=(cl+cr)>>1;
    int ret=(x<=mid?query(x,u<<1,cl,mid):query(x,u<<1|1,mid+1,cr));
    if(judge(ret,s[u],x)) ret=s[u];
    return ret;
}
```

4、平衡树

FHQTreap(split on val)

```
mt19937_64 rng(random_device{}());
uniform_int_distribution<int> dist(0,LLONG_MAX);

int tot,root;
struct FHQTreap{
    struct tree{
        int ls,rs,val,sz,key;
    }a[N];
    int New(int x){
        a[++tot]={0,0,x,1,(int)rng()};
        return tot;
    }
    void pushup(int k){a[k].sz=a[a[k].ls].sz+a[a[k].rs].sz+1;}
    void split(int k,int val,int &x,int &y){
        if(k==0){x=y=0;return;}
        if(a[k].val<=val){
            x=k;
            split(a[k].rs,val,a[k].rs,y);
        }
        else{
            y=k;
            split(a[k].ls,val,x,a[k].ls);
        }
        pushup(k);
    }
    int merge(int x,int y){
        if(x==0||y==0) return x+y;
        if(a[x].key>a[y].key){
            a[x].rs=merge(a[x].rs,y);
            pushup(x);
            return x;
        }
        else{
            a[y].ls=merge(x,a[y].ls);
            pushup(y);
            return y;
        }
    }
    void ins(int val){
        int x,y;
        split(root,val-1,x,y);
        root=merge(merge(x,New(val)),y);
    }
    void del(int val){
        int x,y,z;
        split(root,val,x,z);
        split(x,val-1,x,y);
        if(y) y=merge(a[y].ls,a[y].rs);
        root=merge(merge(x,y),z);
    }
    int rank(int val){
        int x,y,ans;
        split(root,val-1,x,y);
        ans=a[x].sz+1;
        root=merge(x,y);
        return ans;
    }
}
```

```

int kth(int x){
    int id=root;
    while(id){
        if(a[a[id].ls].sz+1==x) break;
        else if(a[a[id].ls].sz>=x) id=a[id].ls;
        else x-=a[a[id].ls].sz+1,id=a[id].rs;
    }
    return id==0?inf:a[id].val;
}
int pre(int val){
    int x,y,id,ans;
    split(root,val-1,x,y);
    id=x;
    while(a[id].rs) id=a[id].rs;
    ans=a[id].val;
    root=merge(x,y);
    return ans;
}
int nxt(int val){
    int x,y,id,ans;
    split(root,val,x,y);
    id=y;
    while(a[id].ls) id=a[id].ls;
    ans=a[id].val;
    root=merge(x,y);
    return ans;
}
}treap;

```

FHQTreap(split on size)

- +懒标记，文艺平衡树实现区间翻转

```

#include<bits/stdc++.h>
#define int long long

using namespace std;
const int N=1e5+8;

mt19937_64 rng(random_device{}());
uniform_int_distribution<int> dist(0,LLONG_MAX);

int tot,root;

struct FHQTreap{
    struct tree{
        int ls,rs,val,sz,key;
        bool fl;
    }a[N];
    int New(int x){
        a[++tot]={0,0,x,1,(int)rng(),0};
        return tot;
    }
    void pushup(int x){a[x].sz=a[a[x].ls].sz+a[a[x].rs].sz+1;}
    void pushdown(int x){
        swap(a[x].ls,a[x].rs);
        if(a[x].ls) a[a[x].ls].fl^=1;
        if(a[x].rs) a[a[x].rs].fl^=1;
        a[x].fl=0;
    }
}treap;

```

```

}

int merge(int x,int y){
    if(!x || !y) return x+y;
    if(a[x].key<a[y].key){
        if(a[x].fl) pushdown(x);
        a[x].rs=merge(a[x].rs,y);
        pushup(x);
        return x;
    }
    if(a[y].fl) pushdown(y);
    a[y].ls=merge(x,a[y].ls);
    pushup(y);
    return y;
}
void split(int i,int k,int &x,int &y){
    if(!i){
        x=y=0;
        return;
    }
    if(a[i].fl) pushdown(i);
    if(a[a[i].ls].sz<k){
        x=i;
        split(a[i].rs,k-a[a[i].ls].sz-1,a[i].rs,y);
    }
    else{
        y=i;
        split(a[i].ls,k,x,a[i].ls);
    }
    pushup(i);
}
void print(int i){
    if(!i) return;
    if(a[i].fl) pushdown(i);
    print(a[i].ls);
    cout<<a[i].val<<" ";
    print(a[i].rs);
}
void reverse(int l,int r){
    int x,y,z;
    split(root,l-1,x,y);
    split(y,r-l+1,y,z);
    a[y].fl^=1;
    root=merge(x,merge(y,z));
}
}treap;

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++) root=treap.merge(root,treap.New(i));
    for(int i=1;i<=m;i++){
        int l,r;
        cin>>l>>r;
        treap.reverse(l,r);
    }
    treap.print(root);
    return 0;
}

```

5、重链剖分

- `1 x y z`, 表示将树从 x 到 y 结点最短路径上所有节点的值都加上 z 。
- `2 x y`, 表示求树从 x 到 y 结点最短路径上所有节点的值之和。
- `3 x z`, 表示将以 x 为根节点的子树内所有节点值都加上 z 。
- `4 x` 表示求以 x 为根节点的子树内所有节点值之和。

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

const int N=2e5+8,inf=1e16;
int n,m,r,p,a[N];
int head[N],to[N],nxt[N],cnt;
int tot,c[N],id[N],dep[N],sz[N],top[N],son[N],f[N];
int w[N<<2],tag[N<<2];

void build(int u=1,int cl=1,int cr=n){
    if(cl==cr){
        w[u]=c[cl]%p;
        return;
    }
    int mid=(cl+cr)>>1;
    build(u<<1,cl,mid);
    build(u<<1|1,mid+1,cr);
    w[u]=(w[u<<1]+w[u<<1|1])%p;
}

void pushdown(int u,int len){
    if(len<=1) return;
    (tag[u<<1]+=tag[u])%=p;
    (tag[u<<1|1]+=tag[u])%=p;
    (w[u<<1]+=tag[u]*(len-len/2))%=p;
    (w[u<<1|1]+=tag[u]*(len/2))%=p;
    tag[u]=0;
}

void update(int l,int r,int k,int u=1,int cl=1,int cr=n){
    if(l<=cl && cr<=r){
        (w[u]+=k*(cr-cl+1))%=p;
        (tag[u]+=k)%=p;
        return;
    }
    pushdown(u,cr-cl+1);
    int mid=(cl+cr)>>1;
    if(l<=mid) update(l,r,k,u<<1,cl,mid);
    if(mid<r) update(l,r,k,u<<1|1,mid+1,cr);
    w[u]=(w[u<<1]+w[u<<1|1])%p;
}

int query(int l,int r,int u=1,int cl=1,int cr=n){
    if(l<=cl && cr<=r) return w[u];
```

```

pushdown(u,cr-cl+1);
int mid=(cl+cr)>>1,ans=0;
if(l<=mid) (ans+=query(l,r,u<<1,cl,mid))%=p;
if(mid<r) (ans+=query(l,r,u<<1|1,mid+1,cr))%=p;
return ans;
}

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

void dfs(int u,int fa,int d){
    dep[u]=d;
    sz[u]=1;
    f[u]=fa;
    int ans=-1;
    for(int i=head[u];i;i=nxt[i]){
        int v=to[i];
        if(v==fa) continue;
        dfs(v,u,d+1);
        sz[u]+=sz[v];
        if(sz[v]>ans) son[u]=v,ans=sz[v];
    }
}

void dfss(int u,int tp){
    id[u]=++tot;
    c[tot]=a[u];
    top[u]=tp;
    if(!son[u]) return;
    dfss(son[u],tp);
    for(int i=head[u];i;i=nxt[i]){
        int v=to[i];
        if(v==f[u] || v==son[u]) continue;
        dfss(v,v);
    }
}

void addrange(int x,int y,int z){
    z%=p;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        update(id[top[x]],id[x],z);
        x=f[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    update(id[x],id[y],z);
}

int queryrange(int x,int y){
    int ans=0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        ans=(ans+query(id[top[x]],id[x]))%p;
        x=f[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
}

```

```

        return (ans+query(id[x],id[y]))%p;
    }

void addpoint(int x,int z){
    z%=p;
    update(id[x],id[x]+sz[x]-1,z);
}

int querypoint(int x){
    return query(id[x],id[x]+sz[x]-1);
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin>>n>>m>>r>>p;
    for(int i=1;i<=n;++i) cin>>a[i];
    for(int i=1;i<n;++i){
        int u,v;
        cin>>u>>v;
        add(u,v);add(v,u);
    }
    dfs(r,0,1);
    dfss(r,r);
    build();
    for(int i=0;i<m;++i){
        int op,x,y,z;
        cin>>op;
        if(op==1){
            cin>>x>>y>>z;
            addrange(x,y,z);
        }
        else if(op==2){
            cin>>x>>y;
            cout<<queryrange(x,y)<<endl;
        }
        else if(op==3){
            cin>>x>>z;
            addpoint(x,z);
        }
        else{
            cin>>x;
            cout<<querypoint(x)<<endl;
        }
    }
}

```

6、ST表

```

struct ST{
    vector<vector<int>>r;
    void build(vector<int>a){
        int n=a.size()-1;
        r.assign(n+1,vector<int>(_lg(n)+1,0));
        for(int i=1;i<=n;++i) r[i][0]=a[i];
        for(int i=1;(1<<i)<=n;++i){

```

```

        for(int j=1;j+(1<<i)-1<=n;++j){
            r[j][i]=max(r[j][i-1],r[j+(1<<i-1)][i-1]);
        }
    }
    int querymax(int x,int y){
        int t=__lg(y-x+1);
        return max(r[x][t],r[y-(1<<t)+1][t]);
    }
}

```

7、分块

区间加，区间大于等于k个数

$$((n/B + 2 \log B) + (2B + B \log(n/B)))$$

```

int a[N*N],c[N*N],tag[N],bel[N*N],st[N],ed[N],sz;
void resort(int x){
    for(int i=st[x];i<=ed[x];++i) c[i]=a[i];
    sort(c+st[x],c+ed[x]+1);
}
void block(){
    sz=sqrt(n);
    for(int i=1;i<=n;++i) bel[i]=(i-1)/sz+1;
    for(int i=1;(i-1)*sz<=n;++i){
        st[i]=(i-1)*sz+1;
        ed[i]=min(i*sz,n);
        resort(i);
    }
}
void modify(int l,int r,int x){
    if(bel[l]==bel[r]){
        for(int i=l;i<=r;++i) a[i]+=x;
        resort(bel[l]);
        return;
    }
    for(int i=l;i<=ed[bel[l]];++i) a[i]+=x;resort(bel[l]);
    for(int i=bel[l]+1;i<=bel[r]-1;++i) tag[i]+=x;
    for(int i=st[bel[r]];i<=r;++i) a[i]+=x;resort(bel[r]);
}
int query(int l,int r,int v){
    int cnt=0;
    if(bel[l]==bel[r]){
        for(int i=l;i<=r;++i) if(a[i]+tag[bel[i]]>=v) ++cnt;
        return cnt;
    }
    for(int i=l;i<=ed[bel[l]];++i) if(a[i]+tag[bel[i]]>=v) ++cnt;
    for(int i=bel[l]+1;i<=bel[r]-1;++i){
        int t=lower_bound(c+st[i],c+ed[i]+1,v-tag[i])-c;
        cnt+=ed[i]-t+1;
    }
    for(int i=st[bel[r]];i<=r;++i) if(a[i]+tag[bel[i]]>=v) ++cnt;
    return cnt;
}

```

区间修改，区间第k大

```
N=1e5
int st[N], ed[N], sz, len, bel[N], a[N], c[N];
int tag[N];

void resort(int x){
    for(int i=st[x]; i<=ed[x]; ++i) c[i]=a[i];
    sort(c+st[x], c+ed[x]+1);
}

void block(){
    sz=160;
    for(int i=1; i<=n; ++i) bel[i]=(i-1)/sz+1;
    for(int i=1; (i-1)*sz<=n; i++){
        st[i]=(i-1)*sz+1;
        ed[i]=min(i*sz, n);
        resort(i);
    }
}

void modify(int l, int r, int x){
    if(bel[l]==bel[r]){
        for(int i=l; i<=r; ++i) a[i]+=x;
        resort(bel[l]);
        return;
    }
    for(int i=l; i<=ed[bel[l]]; ++i) a[i]+=x; resort(bel[l]);
    for(int i=bel[l]+1; i<=bel[r]-1; ++i) tag[i]+=x;
    for(int i=st[bel[r]]; i<=r; ++i) a[i]+=x; resort(bel[r]);
}

int check(int x, int y, int w){
    int cnt=0;
    if(bel[x]==bel[y]){
        for(int i=x; i<=y; i++) if(a[i]+tag[bel[x]]<=w) cnt++;
        return cnt;
    }
    for(int i=x; i<=ed[bel[x]]; i++) if(a[i]+tag[bel[x]]<=w) cnt++;
    for(int i=bel[x]+1; i<bel[y]; i++){
        if(c[st[i]]+tag[i]>w) continue;
        if(c[ed[i]]+tag[i]<=w){
            cnt+=ed[i]-st[i]+1;
            continue;
        }
        int t=upper_bound(c+st[i], c+ed[i]+1, w-tag[i])-c-1;
        cnt+=t-st[i]+1;
    }
    for(int i=st[bel[y]]; i<=y; i++) if(a[i]+tag[bel[y]]<=w) cnt++;
    return cnt;
}

int getmin(int x, int y){
    int ans=inf;
    if(bel[x]==bel[y]){
        for(int i=x; i<=y; i++) ans=min(ans, a[i]+tag[bel[i]]);
        return ans;
    }
    for(int i=x; i<=ed[bel[x]]; i++) ans=min(ans, a[i]+tag[bel[i]]);
}
```

```

        for(int i=bel[x]+1;i<bel[y];i++) ans=min(ans,c[st[i]]+tag[i]);
        for(int i=st[bel[y]];i<=y;i++) ans=min(ans,a[i]+tag[bel[i]]);
        return ans;
    }

    int getmax(int x,int y){
        int ans=-inf;
        if(bel[x]==bel[y]){
            for(int i=x;i<=y;i++) ans=max(ans,a[i]+tag[bel[i]]);
            return ans;
        }
        for(int i=x;i<=ed[bel[x]];i++) ans=max(ans,a[i]+tag[bel[i]]);
        for(int i=bel[x]+1;i<bel[y];i++) ans=max(ans,c[ed[i]]+tag[i]);
        for(int i=st[bel[y]];i<=y;i++) ans=max(ans,a[i]+tag[bel[i]]);
        return ans;
    }

    int query(int x,int y,int k){
        if(k<1 || k>y-x+1) return -1;
        int l=getmin(x,y),r=getmax(x,y);
        while(l<r){
            int mid=l+r>>1;
            if(check(x,y,mid)<k) l=mid+1;
            else r=mid;
        }
        return l;
    }
}

```

区间众数的个数

```

int a[N],b[N],bel[N],st[M],ed[M],sz;
int n,m,cnt,f[M][M];
vector<int>v[N];
int p[N];

void block(){
    sz=sqrt(n);
    for(int i=1;i<=n;++i) bel[i]=(i-1)/sz+1;
    for(int i=1;(i-1)*sz<=n;++i){
        st[i]=(i-1)*sz+1;
        ed[i]=min(i*sz,n);
    }
    for(int i=1;i<=n;++i){
        v[a[i]].push_back(i);
        p[i]=v[a[i]].size()-1;
    }
    for(int i=1;(i-1)*sz<=n;++i){
        vector<int>c(cnt+1);
        int ans=0;
        for(int j=i;(j-1)*sz<=n;++j){
            for(int k=st[j];k<=ed[j];++k){
                c[a[k]]++;
                ans=max(ans,c[a[k]]);
            }
            f[i][j]=ans;
        }
    }
}

```

```

int query(int l,int r){
    if(bel[l]==bel[r]){
        vector<int>c(cnt+1);
        int ans=0;
        for(int i=l;i<=r;++i) c[a[i]]++,ans=max(ans,c[a[i]]);
        return ans;
    }
    int ans=f[bel[l]+1][bel[r]-1];
    for(int i=l;i<=ed[bel[l]];++i){
        int t=a[i];
        while(p[i]+ans<v[t].size() && v[t][p[i]+ans]<=r) ans++;
    }
    for(int i=st[bel[r]];i<=r;++i){
        int t=a[i];
        while(p[i]-ans>=0 && v[t][p[i]-ans]>=l) ans++;
    }
    return ans;
}

```

区间众数的数

```

int a[N],b[N],bel[N],st[M],ed[M],sz,f[M][M],g[M][M],p[N];
int n,m,cnt;
vector<int>v[N];

void block(){
    sz=sqrt(n);
    for(int i=1;i<=n;++i) bel[i]=(i-1)/sz+1;
    for(int i=1;i<=bel[n];++i){
        st[i]=(i-1)*sz+1;
        ed[i]=min(i*sz,n);
    }
    for(int i=1;i<=n;++i){
        v[a[i]].emplace_back(i);
        p[i]=v[a[i]].size()-1;
    }
    for(int i=1;i<=bel[n];++i){
        vector<int>c(cnt+1);
        int ans=0,res=1e9;
        for(int j=i;j<=bel[n];++j){
            for(int k=st[j];k<=ed[j];++k){
                c[a[k]]++;
                if(c[a[k]]>ans){
                    ans=c[a[k]];
                    res=a[k];
                }
                else if(c[a[k]]==ans && a[k]<res){
                    res=a[k];
                }
            }
            f[i][j]=res,g[i][j]=ans;
        }
    }
}

int query(int l,int r){
    if(bel[l]+1>=bel[r]){

```

```

vector<int>t(cnt+2,011);
int ans=0,res=1e9;
for(int i=1;i<=r;++i){
    t[a[i]]++;
    if(t[a[i]]>ans){
        ans=t[a[i]];
        res=a[i];
    }
    else if(t[a[i]]==ans && res>a[i]){
        res=a[i];
    }
}
return b[res];
}
int res=f[bel[1]+1][bel[r]-1],ans=g[bel[1]+1][bel[r]-1];
for(int i=1;i<=ed[bel[1]];++i){
    while(p[i]+ans<v[a[i]].size() && v[a[i]][p[i]+ans]<=r) ans++,res=a[i];
    if(p[i]+ans-1<v[a[i]].size() && v[a[i]][p[i]+ans-1]<=r && a[i]<res)
res=a[i];
}
for(int i=st[bel[r]];i<=r;++i){
    while(0<=p[i]-ans && 1<=v[a[i]][p[i]-ans]) ans++,res=a[i];
    if(0<=p[i]-ans+1 && 1<=v[a[i]][p[i]-ans+1] && a[i]<res) res=a[i];
}
return b[res];
}

```

区间逆序对

```

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

const int N=1e5+8,M=400,sz=260;
int a[N],b[N],n,m,pre[N],suf[N],ed[N],st[N],s[M][N],w[N],wl[N],wr[N],pos[N],bel[N];
ll t[M][M];

struct FenwickTree{
    int c[N];
    int lowbit(int x){return -x&x;}
    void add(int u,int k){
        for(;u<=N-8;u+=lowbit(u)) c[u]+=k;
    }
    int query(int u){
        int ans=0;
        for(;u;u-=lowbit(u)) ans+=c[u];
        return ans;
    }
}ft;
void init(int id){
    for(int j=st[id];j<=ed[id];j++){
        pre[j]=(j==st[id]?0:pre[j-1])+(j-st[id])-ft.query(a[j]);
        ft.add(a[j],1);
    }
    for(int j=st[id];j<=ed[id];j++) ft.add(a[j],-1);
    for(int j=ed[id];j>=st[id];j--){
        suf[j]=(j==ed[id]?0:suf[j+1])+ft.query(a[j]);
    }
}

```

```

        ft.add(a[j],1);
    }
    for(int j=ed[id];j>=st[id];j--) ft.add(a[j],-1);
    for(int i=st[id];i<=ed[id];i++) w[i]=a[i];
    sort(w+st[id],w+ed[id]+1);
    for(int i=1,j=0;i<=n;i++){
        while(j+st[id]<=ed[id] && w[j+st[id]]<i) j++;
        s[id][b[i]]=j;
    }
    for(int i=1;i<=n;i++) s[id][i]+=s[id-1][i],t[bel[i]][id]+=s[id][i];
}

11 query(int l,int r){
    11 ans=0;
    if(bel[l]==bel[r]){
        for(int i=r;i>=l;--i){
            ans+=ft.query(a[i]);
            ft.add(a[i],1);
        }
        for(int i=l;i<=r;i++) ft.add(a[i],-1);
        return ans;
    }
    int t1=0,t2=0;
    for(int i=st[bel[l]];i<=ed[bel[l]];i++) if(l<=pos[w[i]]) wl[t1++]=w[i];
    for(int i=st[bel[r]];i<=ed[bel[r]];i++) if(pos[w[i]]<=r) wr[t2++]=w[i];
    for(int i=0,j=0;i<t1;i++){
        while(j<t2 && wl[i]>wr[j]) j++;
        ans+=j;
    }
    for(int i=1;i<=ed[bel[l]];i++) ans+=s[bel[r]-1][i]-s[bel[l]][i];
    for(int i=bel[l]+1;i<bel[r];i++) ans+=t[i][bel[r]-1]-t[i][i]+pre[ed[i]];
    for(int i=st[bel[r]];i<=r;i++) ans+=(st[bel[r]]-ed[bel[l])-1)-(s[bel[r]-1][i]-s[bel[l]][i]);
    return ans+pre[r]+suf[l];
}

void block(){
    for(int i=1;i<=n;++i) bel[i]=(i-1)/sz+1;
    for(int i=1;i<=bel[n];++i){
        st[i]=(i-1)*sz+1;
        ed[i]=min(i*sz,n);
    }
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>a[i],b[i]=i,pos[a[i]]=i;
    sort(b+1,b+n+1,[&](int x,int y){return a[x]<a[y];});
    block();
    for(int i=1;i<=bel[n];i++) init(i);
    11 lsj=0,l,r;
    for(int i=1;i<=m;i++){
        cin>>l>>r;
        l^=lsj,r^=lsj;
        lsj=query(l,r);
        cout<<lsj<<"\n";
    }
}

```

}

8、树套树

树状数组套线段树

```

struct SegTree{
    struct tree{
        int l,r,cnt;
    }s[N<<8];
    void modify(int u,int &v,int cl,int cr,int k,int x){
        if(!v) v=++tot;
        if(cl==cr){
            s[v].cnt=s[u].cnt+x;
            return;
        }
        int mid=cl+cr>>1;
        if(k<=mid) s[v].r=s[u].r,modify(s[u].l,s[v].l,cl,mid,k,x);
        else s[v].l=s[u].l,modify(s[u].r,s[v].r,mid+1,cr,k,x);
        s[v].cnt=s[s[v].l].cnt+s[s[v].r].cnt;
    }
    int query(int u,int cl,int cr,int l,int r){
        if(!u || r<cl || cr<l) return 0;
        if(l<=cl && cr<=r) return s[u].cnt;
        int mid=cl+cr>>1;
        return query(s[u].l,cl,mid,l,r)+query(s[u].r,mid+1,cr,l,r);
    }
}seg;

struct FenwickOnSegTree{
    int lowbit(int x){return -x&x;}
    void add(int u,int k,int x){
        for(;u<=n;u+=lowbit(u)) seg.modify(R[u],R[u],1,n,k,x);
    }
    int query(int u,int l,int r){
        if(l>r) return 0;
        int ans=0;
        for(;u;u-=lowbit(u)) ans+=seg.query(R[u],1,n,l,r);
        return ans;
    }
}ft;

```

9、可并堆

一开始有 n 个小根堆，每个堆包含且仅包含一个数。接下来需要支持两种操作：

- 1 $x \ y$ ：将第 x 个数和第 y 个数所在的小根堆合并（若第 x 或第 y 个数已经被删除或第 x 和第 y 个数在同一个堆内，则无视此操作）。
- 2 x ：输出第 x 个数所在的堆最小数，并将这个最小数删除（若有多个最小数，优先删除先输入的；若第 x 个数已经被删除，则输出 -1 并无视删除操作）。

```

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

const int N=1e5+8;
int n,t;

```

```

struct Heap{
    int dis, val, ls, rs, rt;
}s[N];
int merge(int x, int y){
    if(!x || !y) return x+y;
    if(s[x].val>s[y].val || (s[x].val==s[y].val && x>y)) swap(x,y);
    s[x].rs=merge(s[x].rs,y);
    if(s[s[x].ls].dis<s[s[x].rs].dis) swap(s[x].ls,s[x].rs);
    s[s[x].ls].rt=s[s[x].rs].rt=s[x].rt=x;
    s[x].dis=s[s[x].rs].dis+1;
    return x;
}
int find(int x){
    return s[x].rt==x?x:find(s[x].rt);
}
void pop(int x){
    s[x].val=-1;
    s[s[x].ls].rt=s[x].ls;
    s[s[x].rs].rt=s[x].rs;
    s[x].rt=merge(s[x].ls,s[x].rs);
}
int main(){
    cin>>n>>t;
    s[0].dis=-1;
    for(int i=1;i<=n;++i) s[i].rt=i, cin>>s[i].val;
    for(int i=1;i<=t;++i){
        int o,x,y;
        cin>>o>>x;
        if(o==1){
            cin>>y;
            if(s[x].val== -1 || s[y].val== -1) continue;
            int f1=find(x),f2=find(y);
            if(f1!=f2) s[f1].rt=s[f2].rt=merge(f1, f2);
        }
        else {
            if(s[x].val== -1) cout<<"-1\n";
            else{
                cout<<s[find(x)].val<<"\n";
                pop(find(x));
            }
        }
    }
}
}

```

10、对顶堆(未验证)

```

struct DDD{
    multiset<int>s1,s2;
    int ss1=0,ss2=0;
    void change(){
        if(s1.size()+s2.size()<=1) return;
        if(s1.size()==1 && s2.size()==1){
            if(*s1.begin()>*s2.begin()){
                swap(s1,s2);
                return;
            }
        }
}

```

```

    if(s1.size() + 1 < s2.size()){
        int x = *s2.begin();
        s1.insert(x);
        s2.erase(s2.find(x));
    }
    else if(s1.size() > s2.size() + 1){
        int x = *s1.rbegin();
        s2.insert(x);
        s1.erase(s1.find(x));
    }
}
void update(int x){
    if(!s1.size()) s1.insert(x), ss1 += x;
    else if(!s2.size()) s2.insert(x), ss2 += x;
    else{
        auto l = *(--s1.end()), r = *(s2.begin());
        if(x < r) s1.insert(x), ss1 += x, ss2 -= x;
        else s2.insert(x), ss1 -= x, ss2 += x;
    }
    change();
}
void del(int x){
    if(!s1.size() || s2.find(x) != s2.end()) s2.erase(s2.find(x)), ss2 -= x;
    else s1.erase(s1.find(x)), ss1 -= x;
    change();
}
int query(){
    int ss1 = s1.size(), ss2 = s2.size();
    if(ss1 + 1 == ss2){
        return *s2.begin();
    }
    else if(ss1 == ss2){
        return (*(--s1.end()) + *(s2.begin())) / 2;
    }
    else return *(--s1.end());
}
}ddd;

```

四、图论

1、二分图判断

```
vector<int>e[N];
int c[N],n,m;

bool dfs(int x,int u){
    c[x]=u;
    for(auto y:e[x]){
        if(c[y]==u) return false;
        if(!c[y] && !dfs(y,-u)) return false;
    }
    return true;
}

bool bi_graph(){
    for(int i=1;i<=n;++i){
        if(c[i]==0 && !dfs(i,1))
            return false;
    }
    return true;
}
```

2、二分图最大匹配

```
int match[N],t[N];

bool dfs(int x,int y){
    for(int j=head[x];j;j=e[j].next){
        int i=e[j].to;
        if(t[i]!=y){
            t[i]=y;
            if(match[i]==0||dfs(match[i],y)){
                match[i]=x;
                o   return true;
            }
        }
    }
    return false;
}
```

3、最短路-Dijkstra

```
vector<int>dis(n+1,inf);
priority_queue<pair<int,int>>q;
auto dijkstra=[&](int s)->void{
    dis[s]=0;
    q.push({0,s});
    while(!q.empty()){
        auto [x,u]=q.top();
        q.pop();
        for(auto y:e[x]){
            if(dis[y]>dis[x]+1){
                dis[y]=dis[x]+1;
                q.push({dis[y],y});
            }
        }
    }
}
```

```

q.pop();
if(dis[u] != -x) continue;
for(auto [v,w]:e[u]){
    if(dis[v]>dis[u]+w){
        dis[v]=dis[u]+w;
        q.push({-dis[v],v});
    }
}
}
};


```

4、欧拉路（一笔画）

定义

- 欧拉回路：通过图中每条边恰好一次的回路
- 欧拉通路：通过图中每条边恰好一次的通路
- 欧拉图：具有欧拉回路的图
- 半欧拉图：具有欧拉通路但不具有欧拉回路的图

性质

- 欧拉图中所有顶点的度数都是偶数。
- 若 G 是欧拉图，则它为若干个环的并，且每条边被包含在奇数个环内。

判别法

无向图是欧拉图当且仅当：

- 非零度顶点是连通的
- 顶点的度数都是偶数

无向图是半欧拉图当且仅当：

- 非零度顶点是连通的
- 恰有 2 个奇度顶点

有向图是欧拉图当且仅当：

- 非零度顶点是强连通的
- 每个顶点的入度和出度相等

有向图是半欧拉图当且仅当：

- 非零度顶点是弱连通的
- 至多一个顶点的出度与入度之差为 1

- 至多一个顶点的入度与出度之差为 1

- 其他顶点的入度和出度相等

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N=1e5+8,inf=1e16;
int n,m,x,y,fa[N],t[N],in[N],out[N],cnt[N];
vector<int>v[N];
stack<int>s;

void dfs(int now){
    for(int i=cnt[now];i<v[now].size();i=cnt[now]){
        cnt[now]=i+1;
        dfs(v[now][i]);
    }
    s.push(now);
}

int find(int x){
    return fa[x]==x?x:fa[x]=find(fa[x]);
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    cin>>n>>m;
    for(int i=1;i<=n;++i) fa[i]=i,t[i]=1;
    for(int i=0;i<m;++i){
        cin>>x>>y;
        int u=find(x),z=find(y);
        if(u!=z) fa[u]=z,t[z]+=t[u];
        v[x].push_back(y);           //邻接表实现
        in[y]++,out[x]++;
    }                                //预处理，带权并查集

    for(int i=1;i<=n;++i) sort(v[i].begin(),v[i].end());
    int r=1;
    if(t[find(r)]!=n){
        cout<<"No"<<endl;
        return 0;
    }                                //连通性

    int tmp1=0,tmp2=0;
    for(int i=1;i<=n;++i){
        if(in[i]!=out[i]){
            if(out[i]-in[i]==1) tmp1++,r=i;
            else if(in[i]-out[i]==1) tmp2++;
            else{
                cout<<"No"<<endl;
                return 0;
            }
        }
    }
}                                //出入度关系（起点与终点出入度差一）
```

```

    if(!((tmp1==tmp2 && tmp1==1) || (!tmp1 && !tmp2))){
        cout<<"No"<<endl;
        return 0;
    }
    //起点(出度-入度==1) 终点(入度-出度==1) || (所有点出入度相同)

    dfs(r);
    while(!s.empty()) cout<<s.top()<<" ",s.pop();
}

```

5、负环

```

int dis[N],tmp[N],vis[N];
queue<int>q;

bool spfa(int s){
    memset(dis,0x3f,sizeof dis);
    dis[s]=0;
    vis[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        vis[u]=0;
        for(int i=head[u];i;i=e[i].next){
            int v=e[i].to,w=e[i].w;
            if(dis[v]>dis[u]+w){
                dis[v]=dis[u]+w;
                tmp[v]=tmp[u]+1;
                if(tmp[v]>=n) return 1;
                if(!vis[v]) vis[v]=1,q.push(v);
            }
        }
    }
    return 0;
}

```

6、差分约束

$$x_u - x_v \leq w$$

```

//在spfa基础上
for(int i=1;i<=n;++i) add(0,i,0);           //建立超级源点

add(u,v,w);                                //所有x最大解

add(u,v,-w);                               //所有x最小解

for(int i=1;i<=n;++i) cout<<dis[i]<<" "; //最短路为一组可行解

```

7、缩点

有向图

```
vector<int>dfn(n+1), low(n+1), vis(n+1), id(n+1);
int tot, tmp;
stack<int>s;
auto tarjan=[&](auto tarjan, int u)->void{
    dfn[u]=low[u]=++tot;
    s.push(u);
    vis[u]=1;
    for(auto v:e[u]){
        if(!dfn[v]){
            tarjan(tarjan, v);
            low[u]=min(low[u], low[v]);
        }
        else if(vis[v]) low[u]=min(low[u], dfn[v]);
    }
    if(low[u]==dfn[u]){
        ++tmp;
        while(1){
            int x=s.top(); s.pop();
            id[x]=tmp;
            vis[x]=0;
            if(x==u) break;
        }
    }
};
for(int i=1;i<=n;++i) if(!dfn[i]) tarjan(tarjan, i);
```

8、2-SAT

```
//tarjan同上

add(a+(b&1)*n, c+(d&1)*n);

add(c+(d&1)*n, a+(b&1)*n); //a为b或b为c

for(int i=1;i<=n;++i){
    if(r[i]==r[i+n]){
        cout<<"IMPOSSIBLE";
        return 0;
    }
}
cout<<"POSSIBLE"<<endl;
for(int i=1;i<=n;++i) cout<<(r[i]<r[i+n])<< ";
```

9、割点

```
int dfn[N], low[N], tot

void tarjan(int u, int root){
    dfn[u]=low[u]=++tot;
    int flag=0;
```

```

for(int i=head[u];i;i=e[i].next){
    int v=e[i].to;
    if(!dfn[v]){
        tarjan(v,root);
        low[u]=min(low[u],low[v]);
        if(low[v]>=dfn[u]){
            flag++;
            if(u!=root || flag>1) cut[u]=1;
        }
    }
    else low[u]=min(low[u],dfn[v]);
}
}

```

10、边双连通

```

int dfn[N],low[N],tot,tmp;
vector<int>res[N];
stack<int>st;

void tarjan(int u,int id){
    dfn[u]=low[u]=++tot;
    st.push(u);
    for(int i=head[u];i;i=e[i].next){
        int v=e[i].to;
        if(!dfn[v]){
            tarjan(v,i);
            low[u]=min(low[u],low[v]);
        }
        else if(i!=(id^1)) low[u]=min(low[u],dfn[v]); //cnt=1;
    }
    if(low[u]==dfn[u]){
        int x=st.top();st.pop();
        res[tmp].push_back(x);
        while(x!=u){
            x=st.top();st.pop();
            res[tmp].push_back(x);
        }
        tmp++;
    }
}

```

11、最小生成树

Kruskal $O(m \log m)$

```

int find(int x){
    return x==fa[x]?x:fa[x]=find(fa[x]);
}

int kruskal(int n,vector<array<int,3>>a){
    int ans=1;
    for(int i=1;i<=n;++i) fa[i]=i;
}

```

```

sort(a.begin(), a.end());
for(auto [w,u,v]:a){
    if(ans==n) break;
    x=find(u),y=find(v);
    if(x!=y){
        fa[x]=y;
        ans++;
        res+=w;
    }
}
return res;
}

```

Prim $O((n + m) \log n)$

```

void prim(){
    memset(dis,0x3f,sizeof dis);
    memset(vis,0,sizeof vis);
    dis[1]=cnt=res=0;
    priority_queue<pair<int,int>>q;
    q.push({0,1});
    while(!q.empty()){
        if(cnt==n) break;
        int u=q.top().second,d=q.top().first;
        q.pop();
        if(vis[u]) continue;
        vis[u]=1;
        ++cnt;
        res+=d;
        for(auto [v,w]:e[u]){
            if(dis[v]>w){
                dis[v]=e[u][v];
                q.push({-dis[v],v});
            }
        }
    }
}

```

12、Kruskal重构树

13、A*

```

q.push(node({x,y,z}))

struct node{
    int x,y,f,g;
    node(int xx,int yy,int gg){
        x=xx,y=yy,g=gg;
        f(x)=g(x)+h(x);           //构造函数f(x)=g(x)已走过路程+h(x)预估路程
    }
    bool operator<(const node &y) const{
        return f>y.f;
    }
};

```

14、IDA*

```

int eval(){                         //同A*的h(x)
}

void dfs(int g,int u,int fa){
    if(g==dep) return;
    ...
    if(g+eval()<=dep) dfs(g+1,v,u);
}

```

15、LCA

倍增

```

int fa[N][((lgN)],dep[N],lg[N];

void init(){
    for(int i=2;i<=n;++i)
        lg[i]=lg[i>>1]+1;
}

void dfs(int now,int fath){
    fa[now][0]=fath;
    dep[now]=dep[fath]+1;
    for(int i=1;i<=lg[dep[now]];++i)
        fa[now][i]=fa[fa[now][i-1]][i-1];
    for(int i=head[now];i;i=e[i].next)
        if(fath!=e[i].to)
            dfs(e[i].to,now);
}

int LCA(int x,int y){
    if(dep[x]>dep[y]) swap(x,y);
    while(dep[x]<dep[y])
        y=fa[y][lg[dep[y]-dep[x]]];
    if(x==y) return x;
    for(int i=lg[dep[x]];i>=0;i--)
        if(fa[x][i]!=fa[y][i]){

```

```

        x=fa[x][i];
        y=fa[y][i];
    }
    return fa[x][0];
}

```

dfs序

```

vector<int>dfn(n+1);
int idx=0;
vector f(__lg(n)+1,vector(n+1,0));
auto dfs=[&](auto dfs,int u,int pa)->void{
    f[0][dfn[u]=++idx]=pa;
    for(auto v:e[u]){
        if(v==pa) continue;
        dfs(dfs,v,u);
    }
};
auto buildst=[&]()>>void{
    for(int i=1;i<=__lg(idx);++i){
        for(int j=1;j+(1<<i)-1<=idx;++j){
            int f1=f[i-1][j],f2=f[i-1][j+(1<<i-1)];
            f[i][j]=dfn[f1]<dfn[f2]?f1:f2;
        }
    }
};
auto LCA=[&](int u,int v)>>int{
    if(u==v) return u;
    u=dfn[u],v=dfn[v];
    if(u>v) swap(u,v);
    int d=__lg(v-u++),f1=f[d][u],f2=f[d][v-(1<<d)+1];
    return dfn[f1]<dfn[f2]?f1:f2;
};
dfs(dfs,1,0);
buildst();

```

五、网络流

最大流

Edmond-Karp ($O(v e^2)$)

```
vector<int> lst;
int n, m, s, t, flow[N], dis[N][N];
int bfs(){
    lst.assign(n+1, 0);
    queue<int> q;
    q.push(s);
    flow[s] = inf;
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for(auto v : e[u]){
            if(dis[u][v] <= 0 || lst[v]) continue;
            lst[v] = u;
            flow[v] = min(flow[u], dis[u][v]);
            q.push(v);
            if(v == t) return 1;
        }
    }
    return 0;
}
int EK(){
    int maxflow = 0;
    while(bfs()){
        maxflow += flow[t];
        for(int u = t; u != s; u = lst[u]){
            int v = lst[u];
            dis[u][v] += flow[t];
            dis[v][u] -= flow[t];
        }
    }
    return maxflow;
}
```

dinic($O(v \sqrt{e})$)

```
int s, t, tot, cnt = 1;
vector<int> head, cur, lst;
int to[M<<1], nxt[M<<1], val[M<<1];

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

bool bfs(){
    lst.assign(tot+1, -1);
```

```

lst[s]=0;
cur=head;
queue<int>q;
q.push(s);
while(!q.empty()){
    int u=q.front();
    q.pop();
    for(int i=head[u];i;i=nxt[i]){
        int v=to[i],w=val[i];
        if(w>0 && lst[v]==-1) lst[v]=lst[u]+1,q.push(v);
    }
}
return lst[t]!=-1;
}

int dfs(int u=s,int flow=inf){
    if(u==t) return flow;
    int rem=flow;
    for(int i=cur[u];i && rem;i=nxt[i]){
        cur[u]=i;
        int v=to[i],w=val[i];
        if(w>0 && lst[v]==lst[u]+1){
            int c=dfs(v,min(w,rem));
            rem-=c;
            val[i]-=c;
            val[i^1]+=c;
        }
    }
    return flow-rem;
}

int dinic(){
    int ans=0;
    while(bfs()) ans+=dfs();
    return ans;
}

```

ISAP(O($v\sqrt{e}$))

```

int s,t,tot,cnt=1;
vector<int>head,lst,cur,dep,gap;
int to[M<<1],nxt[M<<1],val[M<<1];

void add(int u,int v,int w){
    to[++cnt]=v;nxt[cnt]=head[u];val[cnt]=w;head[u]=cnt;
    to[++cnt]=u;nxt[cnt]=head[v];val[cnt]=0;head[v]=cnt;
}

void bfs(){
    queue<int>q;
    q.push(t);
    dep.assign(tot+1,-1);
    gap.assign(tot*10+1,0);
    dep[t]=0;gap[0]=1;
    while(!q.empty()){
        int u=q.front();
        q.pop();

```

```

        for (int i=head[u];i;i=nxt[i]){
            int w=val[i],v=to[i];
            if(w==0 && dep[v]==-1){
                dep[v]=dep[u]+1;
                ++gap[dep[v]];
                q.push(v);
            }
        }
    }

int dfs(int u=s,int flow=inf){
    if(u==t) return flow;
    int used=0;
    for(int i=cur[u];i;i=nxt[i]){
        cur[u]=i;
        int w=val[i],v=to[i];
        if (dep[v]+1==dep[u] && w>0){
            int f=dfs(v,min(w,flow-used));
            used+=f;
            val[i]-=f;
            val[i^1]+=f;
            if(used==flow) return flow;
        }
    }
    if(--gap[dep[u]]==0) dep[s]=tot;
    ++dep[u];
    ++gap[dep[u]];
    return used;
}

int ISAP(){
    bfs();
    int maxflow=0;
    while(dep[s]<tot){
        cur=head;
        maxflow+=dfs();
    }
    return maxflow;
}

```

费用流

SPFA

```

int tot,s,t,cnt=1;
int head[N],incf[N],lst[N];
vector<int>dis;
vector<bool>vis;

struct edge{
    int next,to,flow,dis;
}e[M<<1];

void add(int u,int v,int w,int x){
    e[++cnt]={head[u],v,w,x};
}

```

```

        head[u]=cnt;
        e[++cnt]={head[v],u,0,-x};
        head[v]=cnt;
    }

bool spfa(){
    queue<int>q;
    dis.assign(tot+1,inf);
    vis.assign(tot+1,0);
    q.push(s);
    dis[s]=0;
    vis[s]=1;
    incf[s]=inf;
    while(!q.empty()){
        int u=q.front();
        vis[u]=0;
        q.pop();
        for(int i=head[u];i;i=e[i].next){
            if(!e[i].flow) continue;
            int v=e[i].to;
            if(dis[v]>dis[u]+e[i].dis){
                dis[v]=dis[u]+e[i].dis;
                incf[v]=min(incf[u],e[i].flow);
                lst[v]=i;
                if(!vis[v]) vis[v]=1,q.push(v);
            }
        }
    }
    return dis[t]!=inf;
}

pair<int,int> MCMF(){
    int maxflow=0,mincost=0;
    while(spfa()){
        maxflow+=incf[t];
        mincost+=dis[t]*incf[t];
        for(int u=t,i;u!=s;u=e[i^1].to){
            i=lst[u];
            e[i].flow-=incf[t];
            e[i^1].flow+=incf[t];
        }
    }
    return {mincost,maxflow};
}

```

Dijkstra

```

int cnt=1,head[N],nxt[M<<1],to[M<<1],flow[M<<1],cost[M<<1];
vector<int>dis,h;
int pre[N],lst[N];
void add(int u, int v, int w, int x) {
    nxt[++cnt]=head[u];
    head[u]=cnt;
    to[cnt]=v;
    flow[cnt]=w;
    cost[cnt]=x;
}

```

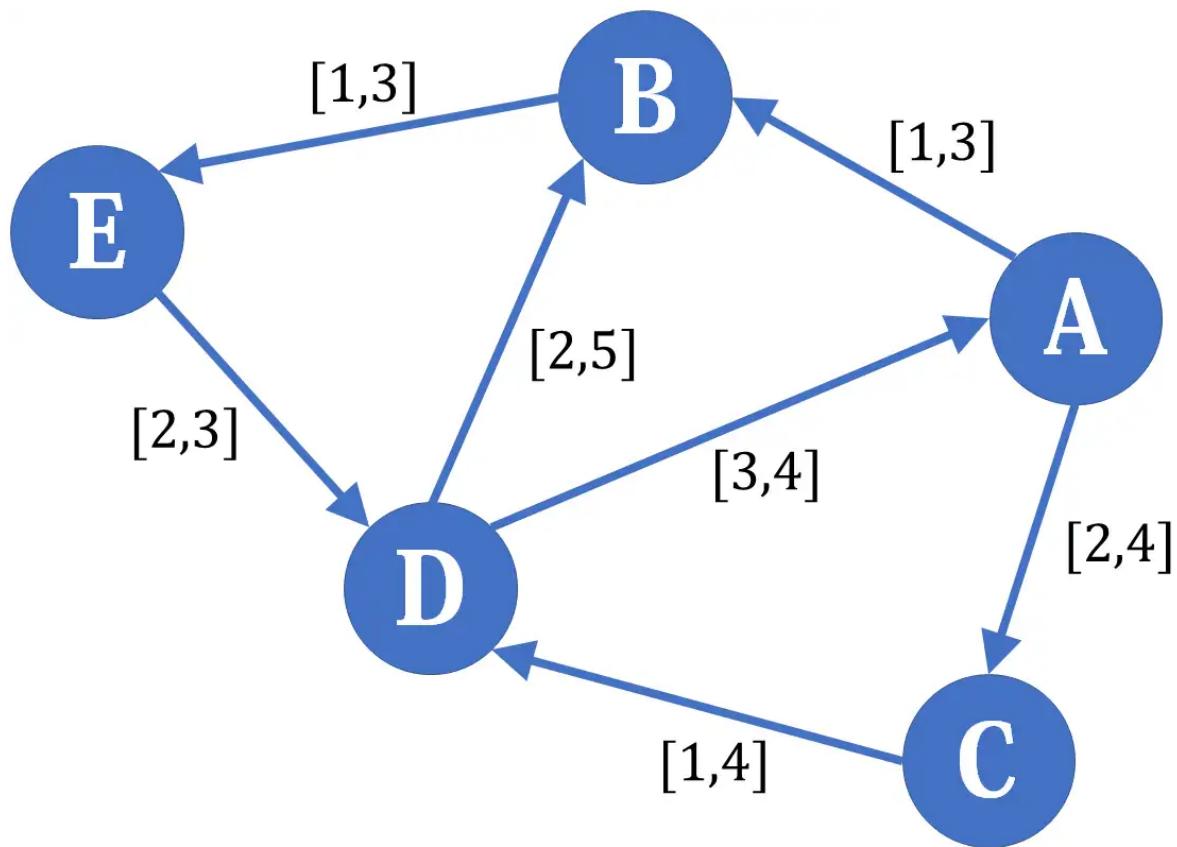
```

void dijkstra(){
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>q;
    dis.assign(n+1,-1);
    dis[s]=0;
    q.push({0,s});
    while(!q.empty()){
        auto [w,u]=q.top();
        q.pop();
        if(dis[u]<0) continue;
        for(int i=head[u];i;i=nxt[i]){
            int v=to[i];
            if(!flow[i]) continue;
            if(dis[v]<0 || dis[v]>dis[u]+cost[i]+h[u]-h[v]){
                dis[v]=dis[u]+cost[i]+h[u]-h[v];
                lst[v]=u,pre[v]=i;
                q.push(pair<int,int>(dis[v],v));
            }
        }
    }
}

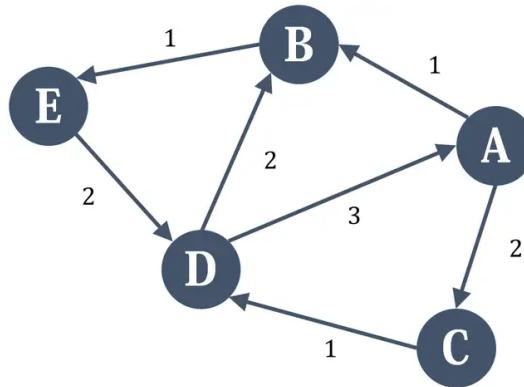
pair<int,int> MCMF(){
    int mincost=0,maxflow=0;
    h.assign(n+1,0);
    for(int f=inf;f>0;){
        dijkstra();
        if(dis[t]<0) break;
        for(int i=1;i<=n;++i) h[i]+=(dis[i]!=-1)?dis[i]:0;
        int d=f;
        for(int u=t;u!=s;u=lst[u]) if(d>flow[pre[u]]) d=flow[pre[u]];
        f-=d;
        maxflow+=d;
        mincost+=h[t]*d;
        for(int u=t;u!=s;u=lst[u]){
            flow[pre[u]]-=d;
            flow[pre[u]^1]+=d;
        }
    }
    return {mincost,maxflow};
}

```

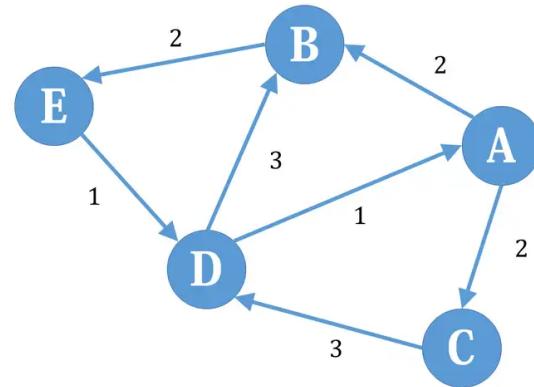
无源汇上下界可行流



我们把它拆成两个结构与原图相同的普通网络，一个每条边的容量为原网络对应边的流量下界，另一个为对应边的流量上界与下界之差。



下界网络

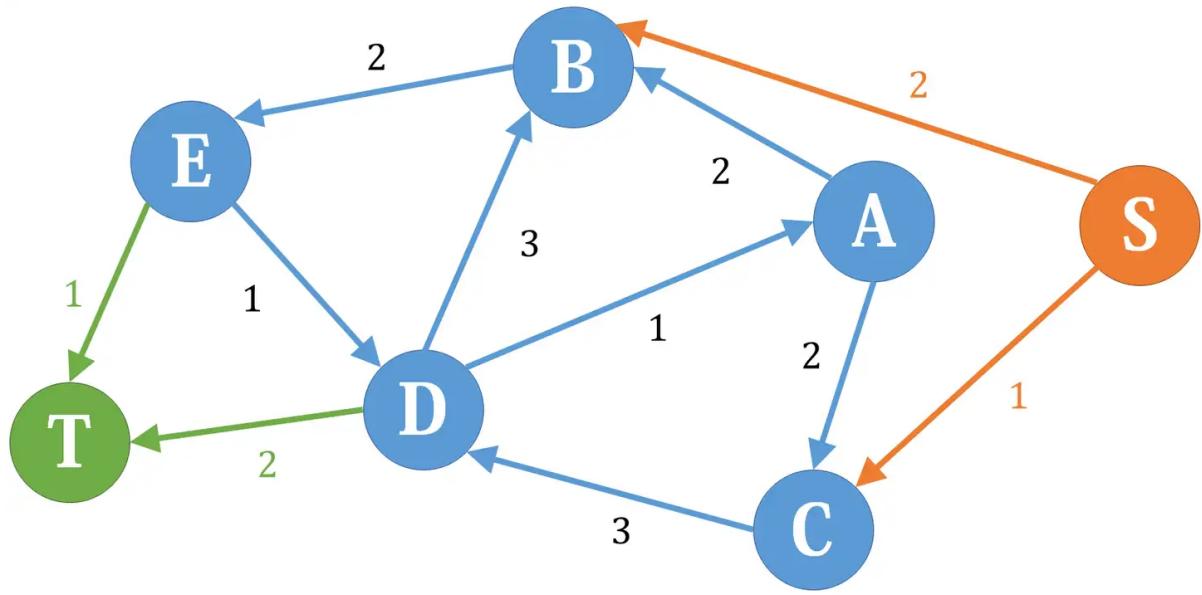


差网络

我们希望下界网络和差网络的流相加后恰好是原图的一个可行流，这首先要求下界网络是满流的（可行流必须达到每条边的下界）。但是下界网络满流后不一定流量平衡，所以我们要对差网络进行一定的修改以弥补这种不平衡。

我们分别考虑下界网络的每个点。A点，流入量为3，流出量也为3，所以是平衡的，那么在差网络中，也应该是平衡的，所以不做修改。B点，流入量为3，流出量为1，流入比流出多2，所以我们希望在差网络中，B的流出应该比流入多2，于是我们在差网络中新设一个源点，然后加入一条容量为2的附加边从源点连向B，这样在差网络平衡时，除去附加边，B点的流出恰好比流入多2。C点与B点类似。D点则相反，因为我们希望在差网络中D点流入比流出多2，所以我们新设一个汇点，然后从D点连一条容量为2的附加边到汇点，E点又和D类似。

也就是说，如果下界网络中某个点有x的净流入，在差网络中我们就从源点向它连一条容量为x的附加边；相反，如果下界网络中某个点有x的净流出，在差网络中我们就从它向汇点连一条容量为x的附加边。这样，我们把差网络修改如下：



在差网络上跑一遍最大流，把每条非附加边的流，加上下界网络的满流，就是一个可行流。但是，如果跑完最大流发现，存在附加边未满流，那说明平衡条件没有得到满足，于是原图不存在可行流。

在实际中，是不需要建立下界网络的，只需要对差网络进行操作即可。另外最后判断的时候并无必要遍历所有附加边，而只需要判断所有从源点出发的边，或者判断所有连向汇点的边即可，因为根据网络流的性质，两者容量和应该相等，于是它们要么都满流，要么都不满流。

```
#include<iostream>
#include<cstring>
#include<queue>
using namespace std;
const int N = 210, M = (10200+210)*2, INF = 0x3f3f3f3f;

int n, m, s, t, dis[N], cur[M], A[N]; //dis是存层数的数组，cur是当前弧优化，A数组是存每个点少了多少流量（用来控制流量守恒）
int e[M], w[M], l[M], ne[M], h[M], idx; //链式前向星建图

void add(int a, int b, int c, int d) //a点到b点，下界是c，上界是d
{
    e[idx] = b;
    w[idx] = d - c; //ab边的容量是上界减下界
    l[idx] = c; //ab边下界存在l数组里
    ne[idx] = h[a];
    h[a] = idx++;
}

bool bfs() //dinic板子
{
    queue<int> q;
    memset(dis, -1, sizeof dis);
    q.push(s);
    dis[s] = 0;
    cur[s] = h[s];
    while(q.size())
    {
        int u = q.front();
        q.pop();
        for(int i = h[u]; ~i; i = ne[i])
        {
            int v = e[i];
            if(dis[v] == -1 && w[i])

```

```

    {
        dis[v] = dis[u] + 1;
        cur[v] = h[v];
        if(v == t)
            return 1;
        q.push(v);
    }
}
return 0;
}

int dfs(int u,int limit)
{
    if(u == t)
        return limit;
    int flow = 0;
    for(int i = cur[u];~i;i = ne[i])
    {
        cur[u] = i;
        int v = e[i];
        if(dis[v] == dis[u]+1 && w[i])
        {
            int minf = dfs(v,min(w[i],limit-flow));
            w[i] -= minf;
            w[i^1] += minf;
            flow += minf;
            if(flow == limit)
                return flow;
        }
    }
    return flow;
}

int dinic()
{
    int ans = 0;
    while(bfs())
        ans += dfs(s,INF);
    return ans;
}

int main()
{
    cin >> n >> m;
    memset(h,-1,sizeof h);
    s = 0,t = n+1;      //自己建s和t
    for(int i = 1;i <= m;i++)
    {
        int a,b,c,d;
        cin >> a >> b >> c >> d;
        add(a,b,c,d);          //正向边的流量初始是上界减下界 (d-c)
        add(b,a,0,0);          //反向边的流量初始是0 (建残留网络)
        A[a] -= c;              //为使流量守恒，A数组计算每个点入边总共减了多少流量，出边总共减了多少流量，来判断和s连还是和t连，这里A是f出-f入的值。如果没导出来可以自己推一下
        A[b] += c;
    }
    int tot = 0;           //tot累加s出边的总容量，用来判断和最大流是否相等，从而判断原图有没有可行流
}

```

```

for(int i = 1;i <= n;i++)
{
    if(A[i] > 0)      //如果A[i]大于0说明f出-f入为正，说明流入的小于流出的，就要将i点和s建
边来补充流量

    {
        add(s,i,0,A[i]);    //从s到i建一条容量为A[i]的边
        add(i,s,0,0);       //反向边
        tot += A[i];
    }
    else if(A[i] < 0)      //如果A[i]小于0说明流入的大于流出的就要将i点和t点建边来
分担流量
    {
        add(i,t,0,-A[i]);   //正向边的容量就是差的流量的绝对值，即A的相反数
        add(t,i,0,0);
    }
}
if(dinic() != tot) //如果最大流和s出边的容量不相等
    cout << "NO" << endl; //说明原图不存在可行流
else
{
    cout << "YES" << endl;
    for(int i = 0;i < m*2;i += 2) //循环前m条边，因为是正反建两条，所以只循环i是偶数的
情况
        cout << w[i^1] + l[i] << endl; //可行流的流量是残留网络的反向边的大小加上减去
的下界l，注意这是残留网络，反向边的流量大小是原图的流量
}

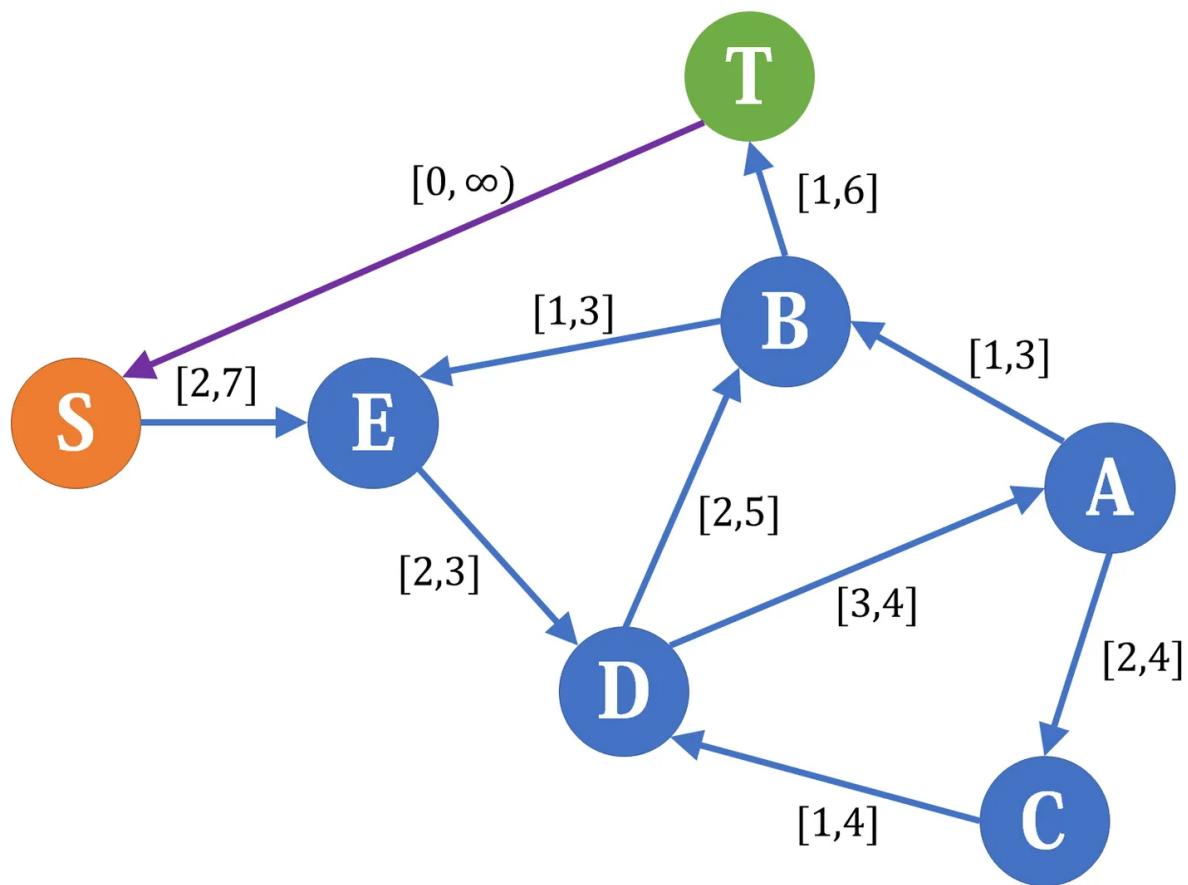
return 0;
}

```

有源汇上下界可行流

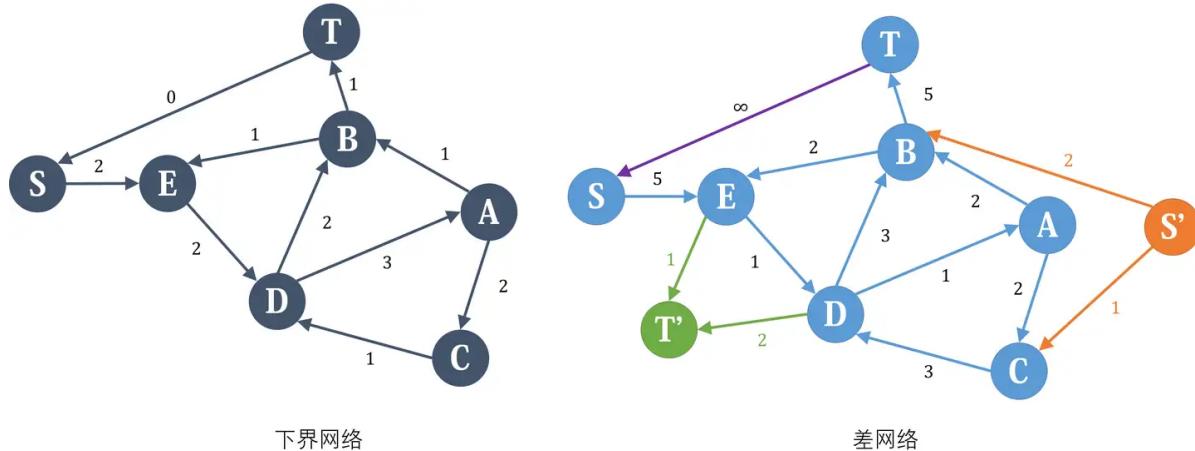
从汇点到源点连一条下界为 0， 上界为 ∞ 的附加边，得到一张和原图等价的无源汇流量网络，于是转化成了无源汇有上下界可行流问题。此时从源点到汇点的可行流流量，即为从汇点到源点的那条附加边的流量（注意下界网络中对应边流量为 0）。

注意，这时候原来的源点和汇点已被处理成普通点，和之后差网络需要额外建立的源、汇点是不同的点，之后如果这两者同时出现，我们记前者为 S 、 T ，后者为 S' 、 T' 。

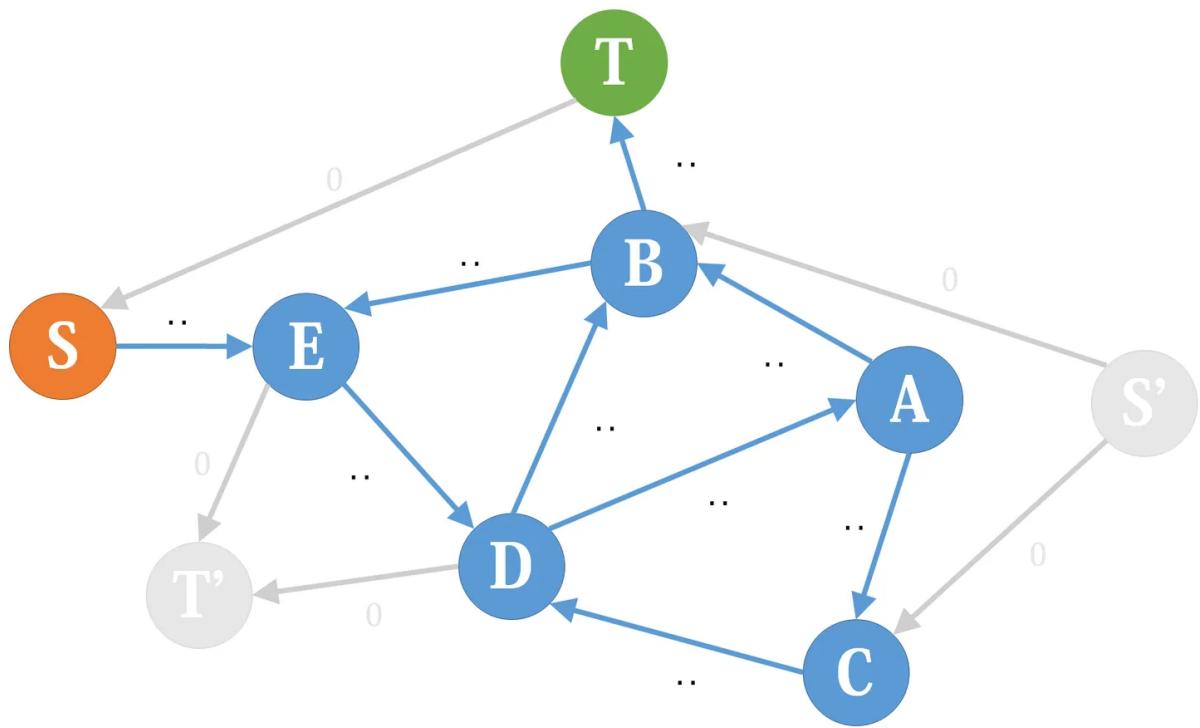


有源汇上下界最大流

按照上一节的方法，我们已经得到了一个可行流，且知道它的流量就是T到S的附加边的流量。



要求从S到T的最大流，我们可以在差网络中把所有附加边删除，然后以S与T为源点与汇点，再求残余网络的最大流，加上可行流的流量即为原网络的最大流。这是因为，可行流已经保证了流量平衡，那么删去附加边后，我们再跑一次最大流把残余网络“榨干”，最后得到的流仍保证是平衡的。



当然实际上 S' 与 T' 连接的附加边不需要删，这种出度或入度为0、又非源汇点的点是不影响最大流的，何况如果存在可行流，它们的残余容量已经为0了。

```

int in[MAXN], out[MAXN];
int main()
{
    int s1, t1;
    cin >> n >> m >> s1 >> t1; // 先把s,t当作普通点，用s1,t1存储
    for (int i = 1; i <= m; ++i)
    {
        int from, to, lb, ub;
        cin >> from >> to >> lb >> ub;
        add(from, to, ub - lb);
        add(to, from, 0);
        in[to] += lb, out[from] += lb;
    }
    add(t1, s1, INF);
    add(s1, t1, 0);
    s = 300, t = 301;
    for (int i = 1; i <= n; ++i)
    {
        if (in[i] > out[i])
        {
            add(s, i, in[i] - out[i]);
            add(i, s, 0);
        }
        else if (in[i] < out[i])
        {
            add(i, t, out[i] - in[i]);
            add(t, i, 0);
        }
    }
    dinic();
    for (int e = head[s]; e; e = edges[e].next)
        if (edges[e].w != 0)
        {
            cout << "please go home to sleep" << endl;
        }
}

```

```
    return 0;
}
s = s1, t = t1; // 切换源汇点
int flow = 0;
for (int e = head[t]; e; e = edges[e].next)
    if (edges[e].to == s)
    {
        flow = edges[e ^ 1].w;
        edges[e].w = edges[e ^ 1].w = 0; // 删除源汇点间的附加边
    }
cout << dinic() + flow << endl;
return 0;
}
```

六、数学

数论：

1、乘法逆元

快速幂

```
int qpow(int a,int b,int p){  
    int ans=1;  
    for(;b;b>>=1){  
        if(b&1) ans=(__int128)ans*a%p;  
        a=__int128)a*a%p;  
    }  
    return ans;  
}
```

线性求逆元

```
const int mod;  
int inv[N];  
  
void init(){  
    inv[1]=1;  
    for(int i=2;i<=N-8;++i) inv[i]=(mod-mod/i)*inv[mod%i]%mod;  
}
```

2、裴蜀定理&扩展欧几里得

对 $\forall a, b \in \mathbb{Z}$, 且 a, b 不全为0, 存在 x, y 使得 $\gcd(a, b) = ax + by$

```
//解方程a*x+b*y=gcd(a,b)  
void exgcd(int a,int b,int &x,int &y){  
    if(!b){  
        x=1,y=0;  
        return;  
    }  
    exgcd(b,a%b,y,x);  
    y-=(a/b)*x;  
}  
  
//int c=exgcd(a,b,x,y);
```

求同余方程

求解 $ax \equiv c \pmod{p}$

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

3、中国剩余定理

$$\begin{cases} x \equiv a_1 \pmod{p_1} \\ x \equiv a_2 \pmod{p_2} \\ \vdots \\ x \equiv a_n \pmod{p_n} \end{cases}$$

```
#include<bits/stdc++.h>  
#define int long long  
using namespace std;  
  
const int N=1e5+8;  
int n,a[N],p[N],mul=1,x,y,res,mod;  
  
int exgcd(int c,int p,int &x,int &y){  
    int tmp=c;  
    if(!p){  
        x=1;y=0;  
        return tmp;  
    }  
    tmp=exgcd(p,c%p,y,x);  
    y-=(c/p)*x;  
    return tmp;  
}  
  
int inv(int t){  
    int tmp=exgcd(t,mod,x,y);  
    return (x+mod)%mod;  
}  
  
signed main(){  
    ios::sync_with_stdio(false);  
    cin.tie(0);  
    cin>>n;  
    for(int i=1;i<=n;++i) cin>>p[i]>>a[i],mul*=p[i];  
    for(int i=1;i<=n;++i){  
        int t=mul/p[i];  
        mod=p[i];  
        res+=a[i]*t*inv(t);  
    }
```

```

    }
    cout<<res%mul<<endl;
}

```

扩展中国剩余定理

```

int ex crt(vector<int>a, vector<int>b){
    for(int i=1;i<len;++i){
        int x,y,g=__gcd(a[0],a[i]),p=a[0]/g;           \\0下标,a为取模数组
        if((b[0]-b[i])%g) return -1;
        exgcd(a[i]/g,p,x,y);
        x=(x%p*((b[0]-b[i])/g)%p+p)%p;
        a[0]=a[0]/g*a[i];
        b[0]=((x*a[i]+b[i])%a[0]+a[0])%a[0];
    }
    return b[0];
}

```

4、扩展欧拉定理

$$a^b \bmod m = \begin{cases} a^{b \bmod \varphi(m)} \bmod m, & \gcd(a, m) = 1, \\ a^b \bmod m, & \gcd(a, m) \neq 1, b < \varphi(m), \\ a^{b \bmod \varphi(m)+\varphi(m)} \bmod m, & \gcd(a, m) \neq 1, \varphi(m) \leq b, \end{cases}$$

5、威尔逊定理

对素数 p 有 $(p-1)! \equiv -1 \pmod{p}$

6、原根

最小原根 $\$ g_p = O(p^{0.25+\epsilon})$, $g_p = \Omega(\log p)$ \$

```

int ans=phi[n],g=-1;
vector<int>a;
for(int i=1;i*i<=ans;++i){
    if(ans%i==0){
        a.push_back(i);
        if(i*i!=ans) a.push_back(ans/i);
    }
}
sort(a.begin(),a.end());
for(int i=1;;++i){
    if(__gcd(i,n)!=1) continue;
    bool b=1;
    for(auto j:a){
        if(j!=ans && qpow(i,j)==1){
            b=0;
            break;
        }
    }
}

```

```

if(b){
    g=i;
    break;
}

```

7、离散对数

BSGS

求解 $a^x \equiv b \pmod{m}$, $\gcd(a, m) = 1$, $O(\sqrt{m})$

$$\begin{aligned} \text{令 } x &= A\lceil\sqrt{m}\rceil - B, 0 \leq A, B, \leq \lceil\sqrt{m}\rceil \\ \text{则有 } a^{A\lceil\sqrt{m}\rceil} &\equiv ba^B \pmod{m} \end{aligned}$$

求最小解

```

int BSGS(int a,int b,int p){
    map<int,int>m;
    int cur=1,t=sqrt(p)+1;
    for(int i=1;i<=t;++i){
        cur=cur*a%p;
        m[b*cur%p]=i;
    }
    int tmp=cur;
    for(int i=1;i<=t;++i){
        if(m.count(cur)) return i*t-m[cur];
        cur=cur*tmp%p;
    }
    return -1;
}

```

求所有解

```

\\注意m!=1
vector<int> BSGS(int a,int b,int m){
    int sz=ceil(__builtin_sqrtl(m));
    map<int,vector<int>>mp;
    int cur=b,tmp=a;
    for(int i=0;i<=sz;++i){
        mp[cur].push_back(i);
        cur=cur*tmp%m;
    }
    set<int>st;
    cur=1,tmp=qpow(a,sz,m);
    for(int A=0;A<=sz;++A){
        if(mp.count(cur)){
            for(auto x:mp[cur]){
                int tmp=A*sz-x;
                if(0<=tmp && tmp<m) st.insert(tmp);
            }
        }
        cur=cur*tmp%m;
    }
}

```

```

    vector<int>ans;
    ans.assign(st.begin(), st.end());
    return ans;
}

```

利用原根转化，求解 $x^a \equiv b \pmod{m}$, $O(\sqrt{m})$

所有解 $\forall i \in \mathbb{Z}, x \equiv g^{c + \frac{\varphi(p)}{\gcd(\varphi(p), a)} \cdot i}$

exBSGS

$$a^x \equiv b \pmod{m}, \gcd(a, m) \neq 1$$

令 $d_1 = (a, m)$, 若 $d_1 \nmid b$, 原方程无解

$$\text{得 } \frac{a}{d_1} \cdot a^{x-1} \equiv \frac{b}{d_1} \pmod{\frac{m}{d_1}}$$

令 $d_2 = (a, \frac{m}{d_1})$, 若 $d_2 \nmid \frac{b}{d_1}$, 原方程无解

$$\text{得 } \frac{a^2}{d_1 d_2} \cdot a^{x-2} \equiv \frac{b}{d_1 d_2} \pmod{\frac{m}{d_1 d_2}}$$

递归, 使得 $(a, \frac{m}{d_1 d_2 \cdots d_n}) = 1$

$$\text{令 } D = \prod_{i=1}^k d_i, \text{ 则 } \frac{a^k}{D} \cdot a^{x-k} \equiv \frac{b}{D} \pmod{\frac{m}{D}}$$

可通过逆元得到普通BSGS问题, 所求 $x - k$ 加上 k 即为解

```
int exgcd(int a, int b, int &x, int &y){
```

```
    int tmp=a;
```

```
    if(!b){
```

```
        x=1, y=0;
```

```
        return tmp;
```

```
}
```

```
    tmp=exgcd(b, a%b, y, x);
```

```
    y-=(a/b)*x;
```

```
    return tmp;
```

```
}
```

```
int inv(int a, int p){
```

```
    int x, y;
```

```
    exgcd(a, p, x, y);
```

```
    return (x%p+p)%p;
```

```
}
```

```
int BSGS(int a, int b, int p){
```

```
    map<int, int>m;
```

```
    int cur=1, t=sqrt(p)+1;
```

```
    for(int i=1; i<=t; ++i){
```

```
        cur=cur*a%p;
```

```
        m[b*cur%p]=i;
```

```
}
```

```
    int tmp=cur;
```

```
    for(int i=1; i<=t; ++i){
```

```

        if(m.count(cur)) return i*t-m[cur];
        cur=cur*tmp%p;
    }
    return -1;
}

int exBSGS(int a,int b,int p){
    a%=p,b%=p;
    if(b==1 || p==1) return 0;
    int tmp=__gcd(a,p),ans=0,na=1;
    while(tmp>1){
        if(b%tmp!=0) return -1;
        b/=tmp;p/=tmp;
        ans++;na=na*(a/tmp)%p;
        if(na==b) return ans;
        tmp=__gcd(a,p);
    }
    int cnt=BSGS(a,b*inv(na,p)%p,p);
    return cnt==-1?-1:cnt+ans;
}

```

***pohlig-hellman** $\mathcal{O}(\sum_{i=1}^t e_i \sqrt{q_i} + \log N)$

```

#include <bits/stdc++.h>
using namespace std;
#define int long long

std::mt19937 rng(std::random_device{}());
int A[]={2,325,9375,28178,450775,9780504,1795265022};

int qpow(int a,int b,int p){
    int ans=1;
    for(;b;b>>=1){
        if(b&1) ans=(__int128)ans*a%p;
        a=(__int128)a*a%p;
    }
    return ans;
}

bool isprime(int x){
    if(x<3) return x==2;
    if(x%2==0) return false;
    int d=x-1,r=__builtin_ctz(d);
    d>>r;
    for(auto a:A){
        int v=qpow(a,d,x);
        if(v<=1 || v==x-1) continue;
        for(int i=0;i<r;++i){
            v=(__int128)v*v%x;
            if(v==x-1 && i!=r-1){
                v=1;
                break;
            }
            if(v==1) return false;
        }
        if(v!=1) return false;
    }
}
```

```

    }
    return true;
}

int Pollard_Rho(int x) {
    int s=0,t=0,val=1;
    int c=rng()% (x-1)+1;
    for(int i=1;;i*=2,s=t,val=1){
        for(int j=1;j<=i;++j){
            t=((__int128)t*t+c)%x;
            val=(__int128)val*abs(t-s)%x;
            if(j%127==0){
                int d=__gcd(val,x);
                if(d>1) return d;
            }
        }
        int d=__gcd(val,x);
        if(d>1) return d;
    }
}

int getfac(int x,int ma){
    if(x<=ma || x<2) return ma;
    if(isprime(x)){
        ma=max(ma,x);
        return ma;
    }
    int p=x;
    while(p>=x) p=Pollard_Rho(x);
    while(x%p==0) x/=p;
    return max(getfac(x,ma),getfac(p,ma));
}

map<int,int> getallfacs(int x){
    map<int,int>fac;
    while(x!=1){
        int ma=getfac(x,0);
        while(x%ma==0) fac[ma]++;
        x/=ma;
    }
    return fac;
}

namespace DLP{
    int fastpow(int a,int n){ //快速幂不解释
        int res=1;
        while (n>0){
            if (n&1) res=res*a;
            a=a*a;
            n>>=1;
        }
        return res;
    }
    int fastpow(int a,int n,int p){ //快速幂*2不解释
        int res=1;
        a%=p;
        while (n>0){
            if (n&1) res=(__int128)res*a%p;
            a=(__int128)a*a%p;
            n>>=1;
        }
        return res;
    }
}

```

```

    }
    return res;
}
int getorg(int p,int phi,vector <array<int,2>> &v){ //获取ord
    for (int k=2;;k++){
        int flag=1;
        for (int i=0;i<(int)v.size();++i){
            if (fastpow(k,phi/v[i][0],p)==1){
                flag=0;
                break;
            }
        }
        if (flag) return k;
    }
}
int BSGS(int a,int b,int p,int mod){ //BSGS模板，具体可以见其它题解的一般做法
    a%=mod;b%=mod;
    if (b==1) return 0;
    if (a==0){
        if (b==0) return 1;
        else return -1;
    }
    int t=1;
    int m=__builtin_sqrtl(p)+1;
    int base=b;
    unordered_map <int,int> vis;
    for (int i=0;i<m;++i){
        vis[base]=i;
        base=(__int128)base*a%mod;
    }
    base=fastpow(a,m,mod);
    int now=t;
    for (int i=1;i<=m+1;++i){
        now=(__int128)now*base%mod;
        if (vis.count(now)) return i*m-vis[now];
    }
    return -1;
}
int getksi(int g,int h,int p,int c,int n,int mod){ //得到合并后的解集，然后上BSGS
    vector <int> pi;
    int tp=1;
    for (int i=0;i<=c;++i){
        pi.push_back(tp);
        tp*=p;
    }
    int gq=fastpow(g,pi[c-1],mod);
    int inv=0;
    tp=1;
    for (int i=c-1;i>=0;--i){
        int tx=tp*BSGS(gq,fastpow((__int128)h*fastpow(g,pi[c]-inv,mod)%mod,pi[i],mod),p,mod);
        inv+=tx;tp*=p;
    }
    return inv;
}
int exgcd(int a,int b,int &x,int &y){ //exgcd模板不解释
    if (b==0){
        x=1;y=0;
        return a;
    }
}

```

```

    }
    int d=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return d;
}
int getinv(int a,int p){ //逆元不解释
    if(a==0) return 1;
    int x,y;
    exgcd(a,p,x,y);
    return (x%p+p)%p;
}
int gcd(int x,int y){ //gcd不解释
    if (x%y==0) return y;
    return gcd(y,x%y);
}
int ExgcdSolve(int a,int b,int c,int &x,int &y){ //求解exgcd
    int d;
    if (c%(d=gcd(a,b))) return -1;
    a/=d;b/=d;c/=d;
    exgcd(a,b,x,y);
    x=(_int128)x*c%b;
    while (x<=0) x+=b;
    return x;
}
int crt(vector <int> ksi,vector <array<int,2>> v){ //crt
    int sz=v.size();
    int M=1,ans=0;
    vector <int> m;
    for (int i=0;i<sz;++i){
        m.push_back(fastpow(v[i][0],v[i][1]));
        M*=m[i];
    }
    for (int i=0;i<sz;++i){
        int Mi=M/m[i];
        ans=((__int128)ans+(__int128)Mi*getinv(Mi,m[i])*ksi[i])%M;
    }
    if (ans<0) ans+=M;
    return ans;
}
int getx(int h,int g,int N,int mod,vector <array<int,2>> &v){ //获取解集，然后用
crt合并
    vector<int>ksi;
    for (array<int,2> tp:v){
        int tg=fastpow(g,N/fastpow(tp[0],tp[1]),mod);
        int th=fastpow(h,N/fastpow(tp[0],tp[1]),mod);
        ksi.push_back(getksi(tg,th,tp[0],tp[1],N,mod));
    }
    return crt(ksi,v);
}
int solve(int g,int h,int p){ //求解
    if (h==1) return 0;
    int phiP=p-1;
    vector <array<int,2>> v;
    auto mp=getallfac(phiP);
    for(auto [x,y]:mp) v.push_back({x,y});
    int rt=getorg(p,phiP,v);
    int x=getx(g,rt,phiP,p,v);
    int y=getx(h,rt,phiP,p,v);
    int a=0,b=0;

```

```

    if(x==0){
        if(y==0) return 1;
        else if(y==1) return 0;
        else return -1;
    }else return ExgcdSolve(x,phiP,y,a,b);
}
};

signed main(){
    int t;
    cin>>t;
    while(t--){
        int p,a,b;
        cin>>p>>a>>b;
        int ans=DLP::solve(a,b,p);
        if(ans==-1) puts("-1");
        else cout<<ans<<"\n";
    }
}

```

*二次剩余Cipolla

```

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

ll read() {
    ll x=0,f=1; char ch=getchar();
    while(ch<'0'||ch>'9') {if(ch=='-') f=-1; ch=getchar();}
    while(ch>='0'&&ch<='9') {x=(x<<1)+(x<<3)+ch-'0'; ch=getchar();}
    return x*f;
}

struct num {
    ll x;// 实部
    ll y;// 虚部(即虚数单位w的系数)
};

ll t,w,n,p;

num mul(num a,num b,ll p) {// 复数乘法
    num res;
    res.x=( (a.x*b.x%p+a.y*b.y%p*w%p) %p+p)%p;// x = a.x*b.x + a.y*b.y*w
    res.y=( (a.x*b.y%p+a.y*b.x%p) %p+p)%p;// y = a.x*b.y + a.y*b.x
    return res;
}
ll qpow_r(ll a,ll b,ll p) {// 实数快速幂
    ll res=1;
    while(b) {
        if(b&1) res=res*a%p;
        a=a*a%p;
        b>>=1;
    }
    return res;
}
ll qpow_i(num a,ll b,ll p) {// 复数快速幂
    num res={1,0};

```

```

    while(b) {
        if(b&1) res=mul(res,a,p);
        a=mul(a,a,p);
        b>>=1;
    }
    return res.x%p;// 只用返回实数部分，因为虚数部分没了
}
11 cipolla(11 n,11 p) {
    n%=p;
    if(qpow_r(n,(p-1)/2,p)==-1+p) return -1;// 据欧拉准则判定是否有解

    11 a;
    while(1) {// 找出一个符合条件的a
        a=rand()%p;
        w=((a*a)%p-n)%p+p;// w = a^2 - n, 虚数单位的平方
        if(qpow_r(w,(p-1)/2,p)==-1+p) break;
    }

    num x={a,1};
    return qpow_i(x,(p+1)/2,p);
}
int main() {
    srand(time(0));
    t=read();
    while(t--) {
        n=read(); p=read();
        if(!n) {
            printf("0\n");
            continue;
        }

        11 ans1=cipolla(n,p),ans2=-ans1+p;// 另一个解就是其相反数
        if(ans1==1) printf("Hola!\n");
        else {
            if(ans1>ans2) swap(ans1,ans2);
            if(ans1==ans2) printf("%lld\n",ans1);
            else printf("%lld %lld\n",ans1,ans2);
        }
    }

    return 0;
}

```

8、Miller-Rabin

```

int A[]={2,325,9375,28178,450775,9780504,1795265022};

int qpow(int a,int b,int p){
    int ans=1;
    for(;b;b>>=1){
        if(b&1) ans=(__int128)ans*a%p;
        a=(__int128)a*a%p;
    }
    return ans;
}

```

```

bool isprime(int x){
    if(x<3) return x==2;
    if(x%2==0) return false;
    int d=x-1,r=__builtin_ctz(d);
    d>>r;
    for(auto a:A){
        int v=qpow(a,d,x);
        if(v<=1 || v==x-1) continue;
        for(int i=0;i<r;++i){
            v=(__int128)v*v%x;
            if(v==x-1 && i!=r-1){
                v=1;
                break;
            }
            if(v==1) return false;
        }
        if(v!=1) return false;
    }
    return true;
}

```

9、Polar-Rho

```

std::mt19937 rng(std::random_device{}());
int A[]={2,325,9375,28178,450775,9780504,1795265022};

int qpow(int a,int b,int p){
    int ans=1;
    for(;b;b>>=1){
        if(b&1) ans=(__int128)ans*a%p;
        a=(__int128)a*a%p;
    }
    return ans;
}

bool isprime(int x){
    if(x<3) return x==2;
    if(x%2==0) return false;
    int d=x-1,r=__builtin_ctz(d);
    d>>r;
    for(auto a:A){
        int v=qpow(a,d,x);
        if(v<=1 || v==x-1) continue;
        for(int i=0;i<r;++i){
            v=(__int128)v*v%x;
            if(v==x-1 && i!=r-1){
                v=1;
                break;
            }
            if(v==1) return false;
        }
        if(v!=1) return false;
    }
    return true;
}

```

```

int Pollard_Rho(int x) {
    int s=0,t=0,val=1;
    int c=rng()% (x-1)+1;
    for(int i=1;;i*=2,s=t,val=1){
        for(int j=1;j<=i;++j){
            t=((__int128)t*t+c)%x;
            val=(__int128)val*abs(t-s)%x;
            if(j%127==0){
                int d=__gcd(val,x);
                if(d>1) return d;
            }
        }
        int d=__gcd(val,x);
        if(d>1) return d;
    }
}

int getfac(int x,int ma){
    if(x<=ma || x<2) return ma;
    if(isprime(x)){
        ma=max(ma,x);
        return ma;
    }
    int p=x;
    while(p>=x) p=Pollard_Rho(x);
    while(x%p==0) x/=p;
    return max(getfac(x,ma),getfac(p,ma));
}

map<int,int> getallfacs(int x){
    map<int,int> fac;
    while(x!=1){
        int ma=getfac(x,0);
        while(x%ma==0) fac[ma]++;
        x/=ma;
    }
    return fac;
}

```

10、积性函数

存在交换律，结合律，加法交换律

$$(a, b) = 1, f(ab) = f(a)(b)$$

$$\sum_{d|n} \mu(d) = [n=1]$$

$$\mu * I = \epsilon$$

$$\sum_{d|n} \varphi(d) = n$$

$$\varphi * I = n$$

$$\sum_{d|n} \mu(d) \frac{n}{d} = \varphi(n)$$

$$\mu * Id = \varphi$$

$$d(ij) = \sum_{x|i} \sum_{y|j} [gcd(x, y) = 1]$$

$$\sum_{i=1}^n \sum_{j=1}^n [gcd(i, j) = 1] = 2 \sum_{i=1}^n \varphi(i) - 1$$

莫比乌斯函数

$$\mu(n) = \begin{cases} 1, & n = 1 \\ (-1)^k, & n = p_1 p_2 \cdots p_k \\ 0, & \text{otherwise} \end{cases}$$

```
int p[N], cnt, mu[N];
bool st[N];

void init(){
    mu[1]=1;
    for(int i=2;i<=N-8;++i){
        if(!st[i]) p[cnt++]=i, mu[i]=-1;
        for(int j=0;j<cnt && i*p[j]<=N-8;++j){
            st[i*p[j]]=1;
            if(i%p[j]) mu[i*p[j]]=-mu[i];
            else break;
        }
    }
}
```

欧拉函数

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

$$\varphi(p^n) = p^{n-1}(p-1)$$

$$\varphi(ax) = a\varphi(x)$$

$$\varphi(a)\varphi(b) = \varphi(ab)$$

欧拉函数+埃氏筛

```
int phi[N];
void init(int n){
    for(int i=1;i<=n;i++) phi[i]=i;
    for(int i=2;i<=n;i++){
        if(phi[i]!=i) continue;
        for(int j=i;j<=n;j+=i){
            phi[j]=phi[j]/i*(i-1);
        }
    }
}
```

欧拉函数+欧拉筛

```
int p[N], cnt, phi[N];
bool st[N];

void init(){
    phi[1]=1;
    for(int i=2;i<=N-8;i++){
        if(!st[i]) p[cnt++]=i, phi[i]=i-1; // 性质一， 指数为1的情形
        for(int j=0;j<cnt && i*p[j]<=N-8;++j){
            st[p[j]*i]=1;
            if(i%p[j]==0){
                phi[p[j]*i]=phi[i]*p[j]; // 性质二
            }
        }
    }
}
```

```
        break;
    }
    else phi[p[j]*i]=phi[p[j]]*phi[i]; // 这时肯定互质, 用性质三
}
}
}
```

约数个数筛

```

int p[N],cnt,d[N],g[N];
bool st[N];

void init(){
    d[1]=1;
    for(int i=2;i<=N-8;i++){
        if(!st[i]) p[cnt++]=i,d[i]=2,g[i]=1;
        for(int j=0;j<cnt && i*p[j]<=N-8;j++){
            st[i*p[j]]=1;
            if(i%p[j]){
                g[i*p[j]]=1;
                d[i*p[j]]=d[i]*d[g[i*p[j]]+1];
            }
            else{
                g[i*p[j]]=g[i]+1;
                d[i*p[j]]=d[i]/(g[i]+1)*(g[i*p[j]]+1);
                break;
            }
        }
    }
}

```

约数和筛

```
int p[N],cnt,d[N],g[N];
bool st[N];

void init(){
    d[1]=1;
    for(int i=2;i<=N-8;i++){
        if(!st[i]) p[cnt++]=i,d[i]=g[i]=i+1;
        for(int j=0;j<cnt && i*p[j]<=N-8;j++){
            st[i*p[j]]=1;
            if(i%p[j]){
                g[i*p[j]]=p[j]+1;
                d[i*p[j]]=d[i]*d[p[j]];
            }
            else{
                g[i*p[j]]=g[i]*p[j]+1;
                d[i*p[j]]=d[i]/g[i]*g[i*p[j]];
                break;
            }
        }
    }
}
```

杜教筛

令 $S(n) = \sum_{i=1}^n f(i)$, 构造 g 使得 $f * g$ 易求

$$g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{y=2}^n g(y)S(\lfloor n/y \rfloor)$$

```
const int N=5e6+8;
\\sphi==phi smu==mu

int getphi(int x){
    if(x<=N-8) return phi[x];
    if(sphi.count(x)) return sphi[x];
    int ans=(x+1)*x/2;
    for(int l=2,r;l<=x;l=r+1){
        r=x/(x/l);
        r=min(r,x);
        ans-=(r-l+1)*getphi(x/l);
    }
    sphi[x]=ans;
    return ans;
}

int getmu(int x){
    if(x<=N-8) return mu[x];
    if(smu.count(x)) return smu[x];
    int ans=1;
    for(int l=2,r;l<=x;l=r+1){
        r=x/(x/l);
        r=min(r,x);
        ans-=(r-l+1)*getmu(x/l);
    }
    smu[x]=ans;
    return ans;
}
```

Min25筛

先求 $g(n) = \sum_{p \leq n} f(p)$, f 可分拆完全积性多项式,

$$g(n, j) = g(n, j-1) - f(p_j)(g(\lfloor \frac{n}{p_j} \rfloor, j-1) - g(p_{j-1}, j-1)), \text{ 预处理 } \lfloor \frac{n}{p_j} \rfloor$$

分为质数和合数

$$S(n, j) = g(n) - g(p_j) + \sum_{j < k, p_k \leq n, 1 \leq e, p_k^e \leq n} f(p_k^e)(S(\lfloor \frac{n}{p_k^e} \rfloor, k) + [e \neq 1])$$

```
//f(p^k)=p^k(p^k-1)
#include<bits/stdc++.h>
#define int long long

using namespace std;
const int N=1e6+8,mod=1e9+7,inv2=mod+1>>1,inv6=(mod+1)/6;
```

```

int p[N],cnt,g1[N],g2[N],v[N];
bool st[N];
unordered_map<int,int>mp;

void init(int N){
    for(int i=2;i<=N;++i){
        if(!st[i]) p[cnt++]=i;
        for(int j=0;j<cnt && i*p[j]<=N;++j){
            st[i*p[j]]=1;
            if(i*p[j]==0) break;
        }
    }
}

int s1(int x){
    x%=mod;
    return x*(x+1)%mod*inv2%mod;
}

int s2(int x){
    x%=mod;
    return x*(x+1)%mod*(2*x+1)%mod*inv6%mod;
}

int F(int x){
    x%=mod;
    return (x*x%mod-x+mod)%mod;
}

int S(int x,int y){
    if(y!=-1 && p[y]>=x) return 0;
    int res=(g2[mp[x]]-g1[mp[x]]+mod)%mod;
    if(y!=-1) res=(res-g2[mp[p[y]]]+g1[mp[p[y]]]+mod)%mod;
    for(int i=y+1;i<cnt && p[i]<=x/p[i];i++){
        for(int w=p[i];w<=x/p[i];w=w*p[i]){
            res=(res+F(w)*S(x/w,i)%mod+F(w*p[i]))%mod;
        }
    }
    return res;
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n;
    cin>>n;
    int sq=sqrtl(n)+1;
    init(sq);
    int idx=0;
    for(int l=1,r;l<=n;l=r+1){
        r=n/(n/l);
        int x=n/l;
        mp[x]=++idx;
        v[idx]=x;
        g1[idx]=(s1(x)-1+mod)%mod;
        g2[idx]=(s2(x)-1+mod)%mod;
    }
    for(int j=0;j<cnt;j++){
        for(int i=1;i<=idx && p[j]<=v[i]/p[j];i++){

```

```

        g1[i]=(g1[i]-p[j]*(g1[mp[v[i]/p[j]]]-g1[mp[p[j-1]]]))%mod+mod)%mod;
        g2[i]=(g2[i]-p[j]*p[j]%mod*(g2[mp[v[i]/p[j]]]-g2[mp[p[j-
1]]]))%mod+mod)%mod;
    }
}
cout<<(S(n,-1)+1)%mod<<"\n";
}

```

组合数学：

11、组合数学

(1)组合数

递归法 (N<3000)

```

// c[a][b] 表示从a个苹果中选b个的方案数
void init(){
    for(int i=0;i<N;i++){
        for (int j=0;j<=i;j++)
            if(!j) c[i][j]=1;
            else c[i][j]=(c[i-1][j]+c[i-1][j-1])%mod;
    }
}

```

逆元法

```

const int mod=998244353,N=2e5+8;
int fac[N],invfac[N];

int qpow(int a,int b){
    int ans=1;
    for(;b;b>>=1){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
    }
    return ans;
}

void init(){
    fac[0]=invfac[0]=1;
    for(int i=1;i<=N-8;++i) fac[i]=i*fac[i-1]%mod;
    invfac[N-8]=qpow(fac[N-8],mod-2);
    for(int i=N-8;i-->0) invfac[i]=invfac[i]*i%mod;
}

int C(int a,int b){
    return fac[a]*invfac[b]%mod*invfac[a-b]%mod;
}

```

Lucas定理

$$C_n^m \bmod p = C_{n/p}^{m/p} C_{n \bmod p}^{m \bmod p} \bmod p$$

```
const int mod;
int fac[N];

void init(){
    fac[0]=1;
    for(int i=1;i<=mod;++i) fac[i]=i*fac[i-1]%mod;
}

int qpow(int a,int b){
    int ans=1;
    for(;b;b>>=1){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
    }
    return ans;
}

int C(int a,int b){
    return a<b?0:fac[a]*qpow(fac[b],mod-2)%mod*qpow(fac[a-b],mod-2)%mod;
}

int lucas(int a,int b){
    return b?lucas(a/mod,b/mod)*C(a%mod,b%mod)%mod:1;
}
```

(2)错排数

容斥计算

$$D_n = n! \sum_{k=0}^n \frac{(-1)^k}{k!}$$

递推计算

$$\begin{aligned} D_n &= (n-1)(D_{n-1} + D_{n-2}) \\ D_n &= nD_{n-1} + (-1)^n \end{aligned}$$

(3)卡特兰数

递归定义

$$f_n = f_0 * f_{n-1} + f_1 * f_{n-2} + \dots f_{n-1} * f_0 \quad n \geq 2$$

递推关系

$$f_n = \frac{4n-2}{n+1} f_{n-1}$$

通项公式

$$\begin{aligned} f_n &= \frac{1}{n+1} C_{2n}^n \\ f_n &= C_{2n}^n - C_{2n}^{n-1} \end{aligned}$$

推广

默慈金数

移动规则 $(1, 0), (1, -1), (1, 1)$

$$M_1 = 1$$

$$M_2 = 2$$

施罗德数

移动规则 $(2, 0), (1, -1), (1, 1)$

$$\begin{aligned} S_0 &= S_1 = 1 \\ S_n &= \frac{(6n-3)S_{n-1} - (n-2)S_{n-2}}{n+1} \end{aligned}$$

(4)第一类斯特林数

定义

将 n 个元素，划分为 k 个圆排列的方案数，记作 $s(n, k)$ 。

$$\begin{aligned} s(0, 0) &= 1 \\ s(n, 0) &= 1 \\ s(n, n) &= 1 \\ s(n, 1) &= (n-1)! \\ s(n, n-1) &= C_n^2 \\ s(n, 2) &= (n-1)! \cdot \sum_{i=1}^{n-1} \frac{1}{i} \\ s(n, n-2) &= 2 \cdot C_n^3 + 3 \cdot C_n^4 \\ \sum_{k=1}^n s(n, k) &= n! \end{aligned}$$

递推式

$$s(n, k) = s(n-1, k-1) + (n-1) \times s(n-1, k)$$

(5)第二类斯特林数

定义

将 n 个不同的元素拆分成 k 个集合间有序，记作 $S(n, k)$ 。

$$\begin{aligned} S(0, 0) &= 1 \\ S(n, 0) &= 0 \\ S(n, n) &= 1 \\ S(n, 1) &= 1 \\ S(n, n-1) &= C_n^2 \\ S(n, 2) &= 2^{n-1} - 1 \\ S(n, n-2) &= C_n^3 + 3 \cdot C_n^4 \\ \sum_{k=1}^n S(n, k) &= B_n \\ \text{容斥得 } S(n, m) &= \sum_{k=0}^m (-1)^k C_m^k (m-k)^n \end{aligned}$$

递推式

$$S(n, k) = S(n-1, k-1) + k \times S(n-1, k)$$

(6)贝尔数

定义

基数为 n 的集合不相交划分方法的数目。

递推公式

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

(7)划分数

$$D(n, k) = \begin{cases} D(n, k-1) + D(n-k, k) & k \leq n \\ D(n, k-1) & k > n \end{cases}$$

(8)球盒问题归纳

n 个球放入 k 个盒子

情况	球是否相同	盒子是否相同	盒子的限制	答案
1	F	F	无	k^n
2	F	F	单放	A_k^n
3	F	F	非空	$k! \cdot S(n, k)$
4	T	F	无	C_{n+k-1}^{k-1}
5	T	F	单放	C_k^n
6	T	F	非空	C_{n-1}^{k-1}
7	F	T	无	$\sum_{i=1}^k S(n, i)$
8	F	T	单放	1
9	F	T	非空	$S(n, k)$
10	T	T	无	$D(n, k)$
11	T	T	单放	1
12	T	T	非空	$D(n - k, k)$

$S(n, k)$: 第二类斯特林数

$D(n, k)$: 划分数

12、范德蒙德卷积

$$\sum_{i=0}^k C_n^i C_m^{k-i} = C_{n+m}^k$$

数值算法：

13、数值积分

自适应辛普森

```

double f(double x){
}

double simpson(double l,double r){
    double mid=(l+r)/2;
    return (f(l)+4*f(mid)+f(r))*(r-l)/6;
}

double asr(double l,double r,double eps,double ans){
    double mid=(l+r)/2;
    double l_=simpson(l,mid),r_=simpson(mid,r);
    if(fabs(l_+r_-ans)<=15*eps) return l_+r_+(l_+r_-ans)/15;
    return asr(l,mid,eps/2,l_)+asr(mid,r,eps/2,r_);
}

```

```

double asr(double l,double r,double eps){
    return asr(l,r,eps,simpson(l,r));
}

```

龙贝格

```

double f(double x){
    return (c*x+d)/(a*x+b);
}
double Romberg(double x1, double x2, int n=10){
    vector R(n+1,vector(n+1,(double)0));
    vector h(n+1,(double)0);
    R[1][1]=(x2-x1)*(f(x1)+f(x2))/2.0;
    for(int j=2;j<=n;++j){
        h[j]=(x2-x1)/pow(2.0,j-1);
        double sum=0;
        for(int i=1;i<=(int)(pow(2,j-2));++i) sum+=f(x1+(2.0*i-1.0)*h[j]);
        R[j][1]=R[j-1][1]/2.0+h[j]*sum;
        for(int k=2;k<=j;++k) R[j][k]=(pow(4.0,k-1)*R[j][k-1]-R[j-1][k-1])/pow(4.0,k-1)-1.0;
    }
    return R[n][n];
}

```

代数学:

14、线性代数

矩阵

```

struct Matrix{
    int n,x[N][N];
    Matrix(int _n=0){n=_n,memset(x,0,sizeof(x));}
    void idn(){for(int i=0;i<n;++i) x[i][i]=1;}
    Matrix operator*(const Matrix &y) const{
        Matrix res(n);
        for(int i=0;i<n;++i)
            for(int j=0;j<n;++j)
                for(int k=0;k<n;++k)
                    (res.x[i][k]+=x[i][j]*y.x[j][k])%mod)%=mod;
        return res;
    }
    Matrix operator^(int b){
        Matrix res(n),a=*this;
        res.idn();
        while(b){
            if(b&1) res=res*a;
            a=a*a;
            b>>=1;
        }
        return res;
    }
};

```

高斯消元求0-1矩阵逆

```
int Gauss_rev(int n){
    for(int i=1;i<=n;i++){
        for(int j=i;j<=n;j++){//找第i列非零的行换上来
            if(a[j][i]){
                swap(a[i],a[j]);
                break;
            }
        }
        if(!a[i][i])return 0;//无解
        for(int j=1;j<=n;j++){//其他行
            if(a[j][i]&&j!=i){
                a[j]^=a[i];
            }
        }
    }
    return 1;
}
```

高斯消元求 R^n 矩阵逆

```
#include<bits/stdc++.h>
using namespace std;
const int N=505;
const int mod=1e9+7;
int a[N][N<<1];
int n;
int ppow(int a,int b,int mod){
    int ans=1%mod;a%=mod;
    while(b){
        if(b&1)ans=111*ans*a%mod;
        a=111*a*a%mod;
        b>>=1;
    }
    return ans;
}
int Gauss_rev(int n){
    for(int i=1;i<=n;i++){
        for(int j=i;j<=n;j++){//找第i列非零的行换上来
            if(a[j][i]){
                swap(a[i],a[j]);
                break;
            }
        }
        if(!a[i][i])return 0;//无解
        int kk=ppow(a[i][i],mod-2,mod);//逆元
        for(int j=i;j<=n*2;j++){//当前行每一列都除以a[i][i]
            a[i][j]=111*a[i][j]*kk%mod;
        }
        for(int j=1;j<=n;j++){//其他行
            if(j!=i){
                kk=a[j][i];
                for(int k=i;k<=n*2;k++){
                    a[j][k]=(a[j][k]-111*kk*a[i][k]%mod+mod)%mod;
                }
            }
        }
    }
}
```

```

    }
    return 1;
}
signed main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            scanf("%d",&a[i][j]);
        }
        a[i][i+n]=1;
    }
    if(!Gauss_rev(n)){
        puts("No Solution");
    }else{
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                printf("%d ",a[i][j+n]);
            }
            puts("");
        }
    }
    return 0;
}

```

高斯约旦消元求线性方程组唯一解

```

int gauss(int n){
    for(int i=1;i<=n;i++){
        int x=i;
        for(int j=i;j<=n;j++) if(a[j][i]>a[x][i]) x=i;
        if(a[x][i]==0) {cout<<"No Solution";return 0;} //无解或有自由元
        for(int j=1;j<=n+1;j++) swap(a[i][j],a[x][j]);
        double k=a[i][i];
        for(int j=1;j<=n+1;j++) a[i][j]=a[i][j]/k;
        for(int j=1;j<=n;j++){
            if(i!=j){
                double ki=a[j][i];
                for(int m=1;m<=n+1;m++)
                    a[j][m]=a[j][m]-ki*a[i][m];
            }
        }
    }
    return 1;
}

```

线性基

```

#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cmath>
#include <map>
#include <queue>
using namespace std;
typedef long long ll;

```

```

typedef int itn;
typedef pair<int, int> PII;
const int N = 200007, mod = 1e9 + 7, INF = 2.1e9;
const double eps = 1e-8;
const int Base = 60 * 2;//线性基数组大小，开大一点，嘻嘻嘻

int base_cnt;
bool zero;

struct LinearBase
{
    int dimension = 60; // dimension 维数，就是线性基的维数 = logs, (s为元素最大值)
    //dimension一定要是logs，错一点就会WA呜呜呜
    //线性基数组
    ll a[Base + 7];//注意这里要加一点，因为下面的循环也是这个数
    vector<ll> mmap;
    LinearBase() {/*构造方法*/
        //for(int i = 0; i <= Base; ++ i)
        //    a[i] = 0;
        fill(a, a + Base + 7, 0);
    }

    LinearBase(ll *x, int n) {build(x, n);}

    void insert(ll t)/*插入一个数*/
    {
        // 逆序枚举二进制位
        for(int i = dimension; i >= 0; -- i) {
            //if(t == 0) return ;
            // 如果 t 的第 i 位为 0，则跳过
            if(!(t >> i & 1)) continue;
            // 如果 a[i] != 0，则用 a[i] 消去 t 的第 i 位上的 1
            if(a[i]) t ^= a[i];
            else {
                // 插入到 a[i] 的位置上

                // 找到可以插入 a[i] 的位置
                // 用 a[0...i - 1] 消去 t 的第 [0, i) 位上的 1
                // 如果某一个 a[k] = 0 也无须担心，因为这时候第 k 位
                // 不存在于线性基中，不需要保证 t 的第 k 位为 0
                for(int k = 0; k < i; ++ k)
                    if(a[k] && (t >> k & 1)) t ^= a[k];
                // 用 t 消去 a[i + 1...L] 的第 i 位上的 1
                for(int k = i + 1; k <= dimension; ++ k)
                    if(a[k] >> i & 1) a[k] ^= t;
                a[i] = t;
                base_cnt++;
                break;
            }
            // 此时 t 的第 i 位为 0，继续寻找其最高位上的 1
        }
        // 如果没有插入到任何一个位置上，则表明 t 可以由 a 中若干个元素的异或和表示出，即 t 在
        // span(a) 中
    }

    // 数组 x 表示集合 S，下标范围 [1...n]
    void build(ll *x, int n)/*求整个数组（向量组）的线性基*/
    {
        base_cnt = 0;
        zero = 0;
    }
};

```

```

        fill(a, a + Base + 7, 0);
        for(int i = 1; i <= n; ++ i)
            insert(x[i]);
        zero = base_cnt != n; /*是否有零行, 表示是否能够线性表出0*/
        mmap.clear();
        for(int i = 0; i <= dimension; ++ i)
            if(a[i])
                mmap.push_back(a[i]); //逆序从第0位存到第t位, 这样下标可以直接对应, 比较方便
    }

11 query_max()/*求最大异或和*/
{
    ll res = 0;
    for(int i = 0; i <= dimension; ++ i)
        res ^= a[i];
    return res;
}

11 query_min(int n)/*求最小异或和, 至少选择一个数*/
{
    zero = base_cnt != n;
    if(zero) return 0;
    for(int i = 0; i <= dimension; ++ i)
        if(mmap[i])
            return mmap[i];
}

bool check_in(ll x)/*check一个数是否在线性基中*/
{
    for(int i = dimension; i >= 0; -- i)
        if(x >> i & 1)
            x ^= a[i];
    return x == 0;
}

void mergefrom(const LinearBase &other) {/*一个线性基与另一个线性基合并*/
    for(int i = 0; i <= dimension; ++ i)
        insert(other.a[i]);
}

static LinearBase merge(const LinearBase &a, const LinearBase &b) {/*合并两个线性
    基, 多用于线段树维护线性基*/
    LinearBase res = a;
    for(int i = 0; i <= 60; ++ i)
        res.insert(b.a[i]);
    return res;
}

11 query_kth(ll k, int n) {/*查询线性空间中第k小的值//细节颇多, 注意zero的特判*/
    /*这里别忘了在主函数里初始化base_cnt = 0以及zero = 0, 已在build函数中初始化, 所以数组尽量
    直接用build*/
    ll ans = 0;
    if(zero) k--;
    if(k >= (1ll << (int)mmap.size()))
        return -1;
    for(int i = 0; i < (int)mmap.size(); ++ i)
        if((k >> i) & 1)
            ans ^= mmap[i];
    return ans;
}

```

```

    }

    ll query_rank(ll x) {
        ll ans = 0;
        for(int i = 0; i <= (int)mmap.size(); ++ i)
            if(x >= mmap[i])
                ans += (ll) << i, x ^= mmap[i];
        return ans + zero;
    }
};//上述操作均通过简单测试
int n;
ll x;
ll cnt, t, q, k;
ll a[N];

int main()
{
    scanf("%d", &t);
    while(t -- ) {
        scanf("%d", &n);
        for(ll i = 1; i <= n; ++ i)
            scanf("%lld", &a[i]);

        LinearBase Line(a, n);

        printf("Case #%d:\n", ++ cnt);
        scanf("%lld", &q);

        while(q -- ) {
            scanf("%lld", &k);
            printf("%lld\n", Line.query_kth(k, n));
        }
    }
    return 0;
}

```

多项式与生成函数：

15、多项式

$$(1 + x + x^2 + x^3 + \dots)^k = \sum_{i=0}^{\infty} C_{k+i-1}^i \cdot x_i$$

快速傅里叶变换

```

void fft(vector<complex<double>>&a) {
    int n=a.size(), L=6311-__builtin_clzll(n);
    vector<complex<long double>>R(2, 1);
    vector<complex<double>>rt(2, 1);
    for(int k=2; k<n; k*=2){
        R.resize(n);
        rt.resize(n);
        auto x=polar(1.0L, acos(-1.0L)/k);
        for(int i=k; i<2*k; ++i) rt[i]=R[i]=i&1?R[i/2]*x:R[i/2];
    }
}

```

```

    }
    vector<int>rev(n);
    for(int i=0;i<n;++i) rev[i]=(rev[i]>>1 | (i&1)<<L)/2;
    for(int i=0;i<n;++i) if(i<rev[i]) swap(a[i],a[rev[i]]);
    for(int k=1;k<n;k*=2)
        for (int i=0;i<n;i+=2*k)
            for(int j=0;j<k;++j){
                complex<double>z=rt[j+k]*a[i+j+k];
                a[i+j+k]=a[i+j]-z;
                a[i+j]+=z;
            }
    }

vector<double>mul(const vector<double>&a, const vector<double>&b){
    if(a.empty() || b.empty()) return {};
    vector<double>res(a.size()+b.size()-1);
    int L=32-__builtin_clz(res.size()),n=1<<L;
    vector<complex<double>>in(n),out(n);
    copy(a.begin(),a.end(),in.begin());
    for(int i=0;i<b.size();++i) in[i].imag(b[i]);
    fft(in);
    for(auto &x:in) x*=x;
    for(int i=0;i<n;++i) out[i]=in[-i&(n-1)]-conj(in[i]);
    fft(out);
    for(int i=0;i<res.size();++i) res[i]=imag(out[i])/(4 * n);
    return res;
}

```

快速数论变换

```

int mod=998244353;

int qpow(int a,int b){
    int ans=1;
    for(;b;b>>=1){
        if(b&1) ans=ans*a%mod;
        a=a*a%mod;
    }
    return ans;
}

vector<int>roots{0,1};
vector<int>rev;

void dft(vector<int>&a){
    int n=a.size();
    if(rev.size()!=n){
        rev.resize(n);
        int k=__builtin_ctzll(n)-1;
        for(int i=0;i<n;++i) rev[i]=rev[i>>1]>>1 | (i&1)<<k;
    }
    for(int i=0;i<n;++i) if(i<rev[i]) swap(a[i],a[rev[i]]);
    if(roots.size()

```

```

        for(int i=(1<<(k-1));i<(1<<k);++i){
            roots[2*i]=roots[i];
            roots[2*i+1]=roots[i]*e%mod;
        }
        k++;
    }
}
for(int k=1;k<n;k<<=1){
    for(int i=0;i<n;i+=2*k){
        for(int j=0;j<k;++j){
            int u=a[i+j],v=a[i+j+k]*roots[k+j]%mod;
            a[i+j]=(u+v)%mod;
            a[i+j+k]=(u-v+mod)%mod;
        }
    }
}
void idft(vector<int>&a){
    reverse(a.begin()+1,a.end());
    dft(a);
    int n=a.size(),inv=(1-mod)/n+mod;
    for(int i=0;i<n;++i) a[i]=a[i]*inv%mod;
}

struct Poly{
    vector<int>a;
    Poly(int _n=0){a.assign(_n+1,011);}
    friend Poly operator*(Poly a,Poly b){
        int sz=1,tot=a.a.size()+b.a.size()-1;
        while(sz<tot) sz<<=1;
        a.a.resize(sz);b.a.resize(sz);
        dft(a.a);dft(b.a);
        for(int i=0;i<sz;++i) a.a[i]=a.a[i]*b.a[i]%mod;
        idft(a.a);
        a.a.resize(tot);
        return a;
    }
};

```

任意模数NTT

```

/*
   _/_/_/      _/_/_/_/      _/_/_/_/_/      _/_/_/_/_/_/      _/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/
   _/_/_/_/      _/_/_/_/_/      _/_/_/_/_/_/      _/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/
   _/_/_/_/_/      _/_/_/_/_/_/      _/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/
   _/_/_/_/_/_/      _/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/
   _/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/
   _/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/_/
   _/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/_/_/
   _/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/_/_/      _/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
*/
#include<bits/stdc++.h>
#define int long long

```

```

using namespace std;
const int inf=1e18;

const array<int,3>P={469762049,998244353,1004535809};
const int M=P[0]*P[1];

int mod=469762049;

int qpow(int a,int b,int p=mod){
    int ans=1;
    for(;b;b>>=1){
        if(b&1) ans=ans*a%p;
        a=a*a%p;
    }
    return ans;
}

vector<int>roots{0,1};
vector<int>rev;

void dft(vector<int>&a){
    int n=a.size();
    if(rev.size()!=n){
        rev.resize(n);
        int k=__builtin_ctzll(n)-1;
        for(int i=0;i<n;++i) rev[i]=rev[i>>1]>>1|(i&1)<<k;
    }
    for(int i=0;i<n;++i) if(i<rev[i]) swap(a[i],a[rev[i]]);
    if(roots.size()

```

```

Poly(int _n=0){a.assign(_n+1,0ll);}
friend Poly operator*(Poly a,Poly b){
    int sz=1,tot=a.a.size()+b.a.size()-1;
    while(sz<tot) sz<<=1;
    a.a.resize(sz);b.a.resize(sz);
    dft(a.a);dft(b.a);
    for(int i=0;i<sz;++i) a.a[i]=a.a[i]*b.a[i]%mod;
    idft(a.a);
    a.a.resize(tot);
    return a;
}
};

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n,m,p;
    cin>>n>>m>>p;
    vector<Poly>a(3),b(3),c(3);
    for(int i=0;i<3;++i){
        a[i].a.resize(n+1);
        b[i].a.resize(m+1);
    }
    for(int i=0;i<=n;++i){
        cin>>a[0].a[i];
        for(int j=1;j<3;++j) a[j].a[i]=a[0].a[i];
    }
    for(int i=0;i<=m;++i){
        cin>>b[0].a[i];
        for(int j=1;j<3;++j) b[j].a[i]=b[0].a[i];
    }
    for(int i=0;i<3;++i){
        mod=P[i];
        roots.resize(2);
        c[i]=a[i]*b[i];
    }
    Poly res(n+m);
    for (int i=0;i<=n+m;++i) {
        int A=(__int128)c[0].a[i]*P[1]%M*qpow(P[1]%P[0],P[0]-2,P[0])%M;
        A=(A+(__int128)c[1].a[i]*P[0]%M*qpow(P[0]%P[1],P[1]-2,P[1])%M)%M;
        int k=((c[2].a[i]-A)%P[2]+P[2])%P[2]*qpow(M%P[2],P[2]-2,P[2])%P[2];
        res.a[i]=(k%p)*(M%p)%p+A%p)%p;
        cout<<res.a[i]<<"\n"[i==n+m];
    }
}
}

```

快速莫比乌斯变换

```

vector<int> mul(vector<int>a,vector<int>b){
    for(int i=0;i<a.size();++i) a[i]=a[i]*b[i]%mod;
    return a;
}

void fwtr(vector<int>&a,int x){
    int n=a.size();
    for(int o=2,k=1;o<=n;o<<=1,k<<=1){
        for(int i=0;i<n;i+=o){

```

```

        for(int j=0;j<k;++j){
            (a[i+j+k]+=a[i+j]*x%mod)%=mod;
        }
    }
}

vector<int> OR(vector<int>a,vector<int>b){
    fwtoR(a,1);fwtoR(b,1);
    a=mul(a,b);
    fwtoR(a,mod-1);
    return a;
}

void fwtaND(vector<int>&a,int x){
    int n=a.size();
    for(int o=2,k=1;o<=n;o<<=1,k<<=1){
        for(int i=0;i<n;i+=o){
            for(int j=0;j<k;++j){
                (a[i+j]+=a[i+j+k]*x%mod)%=mod;
            }
        }
    }
}

vector<int> AND(vector<int>a,vector<int>b){
    fwtaND(a,1);fwtaND(b,1);
    a=mul(a,b);
    fwtaND(a,mod-1);
    return a;
}

void fwtxOR(vector<int>&a,int x){
    int n=a.size();
    for(int o=2,k=1;o<=n;o<<=1,k<<=1){
        for(int i=0;i<n;i+=o){
            for(int j=0;j<k;++j){
                (a[i+j]+=a[i+j+k])%=mod;
                a[i+j+k]=(a[i+j]-2*a[i+j+k]%mod+mod)%mod;
                (a[i+j]*=x)%=mod;
                (a[i+j+k]*=x)%=mod;
            }
        }
    }
}

vector<int> XOR(vector<int>a,vector<int>b){
    fwtxOR(a,1);fwtxOR(b,1);
    a=mul(a,b);
    fwtxOR(a,(mod+1)/2);
    return a;
}

```

多项式求逆

$$\begin{aligned} & \text{若 } F \cdot G \equiv 1 \pmod{x^n} \\ & \text{令 } F \cdot G_1 \equiv 1 \pmod{x^{\lceil \frac{n}{2} \rceil}} \\ & F \cdot G \equiv 1 \pmod{x^{\lceil \frac{n}{2} \rceil}} \\ & \therefore G - G_1 \equiv 0 \pmod{x^{\lceil \frac{n}{2} \rceil}} \\ & (G - G_1)^2 \equiv 0 \pmod{x^n} \\ & G^2 - 2GG_1 + G_1^2 \equiv 0 \pmod{x^n} \\ & \text{左乘 } F, \text{ 得 } G \equiv 2G_1 - FG_1^2 \pmod{x^n}, \text{ 递归求解} \end{aligned}$$

```
Poly trunc(int k) const{ //截取原多项式mod x^k
    if(k<=size()) return Poly(a.begin(), a.begin() + k);
    return Poly{a}.resize(k);
}
Poly inv(int n) const{ //多项式F(x)的mod x^n意义下逆元
    Poly res{qpow(a[0], P-2)};
    int k=1;
    while(k<n){
        k*=2;
        res = res * (Poly{2} - trunc(k) * res).resize(k);
    }
    return res.resize(n);
}
```

多项式求ln

$$\begin{aligned} & \text{当且仅当 } [x^0]F(x) = 1 \text{ 时, 存在 } \ln F \\ & G(x) \equiv \ln F(x) \pmod{x^n} \\ & G'(x) \equiv \frac{F'(x)}{F(x)} \pmod{x^n} \\ & \text{对 } G'(x) \text{ 求积分即可} \end{aligned}$$

```
Poly deriv() const{ //多项式求导
    if(a.empty()) return Poly();
    Poly res(size() - 1);
    for(int i=0; i<size() - 1; i++){
        res[i] = (i+1) * a[i+1] % P;
    }
    return res;
}
Poly integr() const{ //多项式积分
    Poly res(size() + 1);
    for(int i=1; i<size(); i++){
        res[i] = a[i-1] * qpow(i, P-2) % P;
    }
    return res;
}
Poly log(int n) const{ //多项式求ln
    return (deriv() * inv(n)).integr().resize(n);
}
```

牛顿迭代法求多项式零值

首先 $n = 1$ 的时候， $G(F(z)) \equiv 0 \pmod{z}$ ，这是要单独求出来的

现在假设已经求出了

$$G(F_0(z)) \equiv 0 \pmod{z^{\lceil \frac{n}{2} \rceil}}$$

考虑如何扩展到 $\pmod{z^n}$ 下，可以把 $G(F(z))$ 在 $F_0(z)$ 这里进行泰勒展开

$$\begin{aligned} G(F(z)) &= G(F_0(z)) \\ &+ \frac{G'(F_0(z))}{1!}(F(z) - F_0(z)) \\ &+ \frac{G''(F_0(z))}{2!}(F(z) - F_0(z))^2 \\ &+ \dots \end{aligned}$$

因为 $F(z)$ 和 $F_0(z)$ 的最后 $\lceil \frac{n}{2} \rceil$ 项相同，所以 $(F(z) - F_0(z))^2$ 的最低的非 0 项次数大于 $2\lceil \frac{n}{2} \rceil$ ，所以可以得到

$$G(F(z)) \equiv G(F_0(z)) + G'(F_0(z))(F(z) - F_0(z)) \pmod{z^n}$$

然后因为 $G(F(z)) \equiv 0 \pmod{z^n}$ ，可以得到

$$F(z) \equiv F_0(z) - \frac{G(F_0(z))}{G'(F_0(z))} \pmod{z^n}$$

多项式求exp

当且仅当 $[x^0]F(x) = 0$ 时，存在 $\exp F$

通过牛顿迭代法： $B(x) \equiv e^{A(x)} \pmod{x^n}$

$$\ln B(x) - A(x) \equiv 0 \pmod{x^n}$$

令 $F(B(x)) = \ln B(x) - A(x)$, $A(x)$ 为常数

$$\therefore F'(B(x)) = \frac{1}{B(x)}$$

$$\therefore B(x) \equiv B_0(x) - \frac{F(B_0(x))}{F'(B_0(x))} \equiv B_0(x)(1 - \ln B_0(x) + A(x)) \pmod{x^n}$$

```
Poly trunc(int k) const{                                     // 截取原多项式 mod x^k
    if(k<=size()) return Poly(a.begin(), a.begin()+k);
    return Poly{a}.resize(k);
}
Poly exp(int n) const{                                     // 多项式求exp
    Poly res{1};
    int k=1;
    while(k<n){
        k*=2;
        res=res*(Poly{1}-res.log(k)+trunc(k)).resize(k);
    }
    return res.resize(n);
}
```

多项式开根

$$\begin{aligned} \text{令 } G^2(x) &\equiv F(x) (\text{mod } x^n), G(x) \equiv G_0(x) (\text{mod } x^{\lceil \frac{n}{2} \rceil}) \\ (G(x) - G_0(x))^2 &\equiv 0 (\text{mod } x^{\lceil \frac{n}{2} \rceil}) \\ G^2(x) - 2G(x)G_0(x) + G_0^2(x) &\equiv 0 (\text{mod } x^n) \\ G(x) &= \frac{F(x) + G_0^2(x)}{2G_0(x)} \end{aligned}$$

```
Poly sqrt(int n) const {  
    Poly res{1};  
    int k=1;  
    while(k<n){  
        k*=2;  
        res=(trunc(k)*res.inv(k)+res).resize(k)*qpow(2,P-2);  
    }  
    return res.resize(n);  
}
```

多项式平移

$$\begin{aligned} f(x+c) &= \sum_{i=0}^n f_i (x+c)^i \\ &= \sum_{i=0}^n f_i \cdot \left(\sum_{j=0}^i C_i^j x^j c^{i-j} \right) \\ &= \sum_{i=0}^n f_i \cdot i! \cdot \left(\sum_{j=0}^i \frac{x^j}{j!} \frac{c^{i-j}}{(i-j)!} \right) \\ &= \sum_{i=0}^n \frac{x^i}{i!} \cdot \left(\sum_{j=i}^n \frac{k^{j-i}}{(j-i)!} j! f_j \right) \end{aligned}$$

$$\text{令 } g(x) = \sum_{i=0}^n \frac{k^i}{i!} x^{n-i}, h(x) = \sum_{i=0}^n i! f_i x^i$$

$$\text{令 } F(x) = \frac{g * h}{x^n}$$

$$f(x+c) = \sum_{i=0}^n \frac{F_i}{i!} x^i$$

分治FFT

$$f_i = \sum_{j=1}^i f_{i-j} g_j, f_0 = 1$$

$$\text{构造生成函数 } F(x) = \sum_{i=0}^{\infty} f_i x^i, G(x) = \sum_{i=0}^{\infty} g_i x^i$$

$$\text{令 } g_0 = 0, F(x)G(x) = \sum_{i=0}^{\infty} x^i \sum_{j+k=i} f_j \cdot g_k = F(x) - f_0$$

$$\therefore F(x) = (1 - G(x))^{-1} (\text{mod } x^n)$$

多项式多点快速求值

```
vector<Poly>Q;

Poly MULT(Poly a,Poly b){
    int n=a.size(),m=b.size();
    reverse(b.a.begin(),b.a.end());
    b=a*b;
    for(int i=0;i<n;++i) a[i]=b[i+m-1];
    return a;
}

void MPinit(Poly &a,int u,int cl,int cr){
    if(cl==cr){
        Q[u].resize(2);
        Q[u][0]=1,Q[u][1]=mod-a[cl];
        return;
    }
    int mid=cl+cr>>1;
    MPinit(a,u<<1,cl,mid);MPinit(a,u<<1|1,mid+1,cr);
    Q[u]=Q[u<<1]*Q[u<<1|1];
}

void MPCal(int u,int cl,int cr,Poly f,Poly &g){
    f.resize(cr-cl+1);
    if(cl==cr){
        g[cl]=f[0];
        return;
    }
    int mid=cl+cr>>1;
    MPCal(u<<1,cl,mid,MULT(f,Q[u<<1|1]),g);
    MPCal(u<<1|1,mid+1,cr,MULT(f,Q[u<<1]),g);
}

Poly Multipoints(Poly f,Poly a,int n){          //n为f和a的最大长度
    f.resize(n+1),a.resize(n);
    Poly v(n);
    Q.resize(n<<2);
    MPinit(a,1,0,n-1);
    MPCal(1,0,n-1,MULT(f,Q[1].inv(n+1)),v);
    return v;
}
```

全家桶

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

int P=998244353;

int qpow(int a,int b){
    int res=1;
    while(b){
        if(b&1) res=a*res%P;
        a=a*a%P;
```

```

        b/=2;
    }
    return res;
}
vector<int>roots{0,1};
vector<int>rev;

void dft(vector<int>&a){
    int n=a.size();
    if(rev.size()!=n){
        rev.resize(n);
        int k=__builtin_ctzll(n)-1;
        for(int i=0;i<n;i++) rev[i]=rev[i>>1]>>1|(i&1)<<k;
    }
    for(int i=0;i<n;i++) if(i<rev[i]) swap(a[i],a[rev[i]]);
    if(roots.size()<n){
        int k=__builtin_ctzll(roots.size());
        roots.resize(n);
        while((1<<k)<n){
            int e=qpow(3,(P-1)>>(k+1));
            for(int i=1<<(k-1);i<1<<k;i++){
                roots[2*i]=roots[i];
                roots[2*i+1]=1LL*roots[i]*e%P;
            }
            k++;
        }
    }
    for(int k=1;k<n;k*=2){
        for(int i=0;i<n;i+=2*k){
            for(int j=0;j<k;j++){
                int u=a[i+j],v=a[i+j+k]*roots[k+j]%P;
                a[i+j]=(u+v)%P;
                a[i+j+k]=(u-v+P)%P;
            }
        }
    }
}
void idft(vector<int>&a){
    reverse(a.begin()+1,a.end());
    dft(a);
    int n=a.size(),inv=(1-P)/n+P;
    for(int i=0;i<n;i++) a[i]=a[i]*inv%P;
}

struct Poly{
    vector<int>a;
    Poly():a(){}
    explicit Poly(int n,int val=0):a(n,val){}
    Poly(const vector<int>&a):a(a){}
    Poly(const vector<int>&&a):a(a){}
    Poly(const initializer_list<int>&t):a(t){}
    template<class It>
    explicit Poly(It first,It last):a(first,last){}
    Poly shift(int k) const{
        if(k>=0){
            auto b=a;
            b.insert(b.begin(),k,0);
            return b;
        }else if(size()<=-k){
            return Poly();
        }
    }
}

```

```

    }else{
        return Poly(a.begin()-k,a.end());
    }
}
int size() const{return a.size();}
Poly& resize(int k){
    this->a.resize(k);
    return *this;
}
Poly trunc(int k) const{ //截取原多项式mod x^k
    if(k<=size())return Poly(a.begin(),a.begin()+k);
    return Poly{a}.resize(k);
}
bool empty() const{
    return a.empty();
}
int& operator[](int idx){
    return a[idx];
}
int operator[](int idx) const{
    if(idx<size())return a[idx];
    return 0;
}
void push_back(int v){a.push_back(v);}
Poly deriv() const{ //多项式求导
    if(a.empty())return Poly();
    Poly res(size()-1);
    for(int i=0;i<size()-1;i++){
        res[i]=(i+1)*a[i+1]%P;
    }
    return res;
}
Poly integr() const{ //多项式积分
    Poly res(size()+1);
    for(int i=1;i<size();i++){
        res[i]=a[i-1]*qpow(i,P-2)%P;
    }
    return res;
}
Poly inv(int n) const{ //多项式F(x)的mod x^n意义下逆元
    Poly res{qpow(a[0],P-2)};
    int k=1;
    while(k<n){
        k*=2;
        res=res*(Poly{2}-trunc(k)*res).resize(k);
    }
    return res.resize(n);
}
Poly sqrt(int n) const{ //多项式F(x)开根, 默认a0=1
    Poly res{1};
    int k=1;
    while(k<n){
        k*=2;
        res=(trunc(k)*res.inv(k)+res).resize(k)*qpow(2,P-2);
    }
    return res.resize(n);
}
Poly log(int n) const{ //多项式求ln
    return (deriv()*inv(n)).integr().resize(n);
}

```

```

}

Poly exp(int n) const{ //多项式求exp
    Poly res{1};
    int k=1;
    while(k<n){
        k*=2;
        res=res*(Poly{1}-res.log(k)+trunc(k)).resize(k);
    }
    return res.resize(n);
}

Poly pow(int k,int n) const{ //多项式k次方mod x^n
    int i=0;
    while(i<size() && a[i]==0){
        i++;
    }
    if(i==size() || i*k>=n){
        return Poly(n);
    }
    int v=a[i];
    auto f=shift(-i)*qpow(v,P-2);
    return (f.log(n-i*k)*k).exp(n-i).shift(i*k)*qpow(v,k);
}

friend Poly operator*(Poly a,Poly b){
    if(a.empty() || b.empty())return Poly();
    if(a.size()>b.size())swap(a, b);
    if(a.size()<64){
        Poly c(a.size()+b.size()-1);
        for(int i=0;i<a.size();i++){
            for(int j=0;j<b.size();j++){
                c[i+j]=(c[i+j]+a[i]*b[j])%P;
            }
        }
        return c;
    }

    int sz=1,tot=a.size()+b.size()-1;
    while(sz<tot) sz*=2;
    a.resize(sz);b.resize(sz);
    dft(a.a);dft(b.a);
    for(int i=0;i<sz;i++) a[i]=a[i]*b[i]%P;
    idft(a.a);
    a.resize(tot);
    return a;
}

Poly& operator/= (int k){
    return (*this)=(*this)/k;
}

Poly& operator*=(int k){
    return (*this)=(*this)*k;
}

Poly& operator*=(const Poly &b){
    return (*this)=(*this)*b;
}

Poly& operator+=(const Poly &b){
    return (*this)=(*this)+b;
}

Poly& operator-=(const Poly &b){
    return (*this)=(*this)-b;
}

```

```

friend Poly operator/ (Poly a,int k){
    int inv=qpow(k,P-2);
    for(int i=0;i<a.size();i++){
        a[i]=a[i]*inv%P;
    }
    return a;
}
friend Poly operator* (Poly a,int k){
    for(int i=0;i<a.size();i++){
        a[i]=a[i]*k%P;
    }
    return a;
}
friend Poly operator* (int k,Poly a){
    for(int i=0;i<a.size();i++){
        a[i]=a[i]*k%P;
    }
    return a;
}
friend Poly operator+ (const Poly &a,const Poly &b){
    Poly res(max(a.size(),b.size()));
    for(int i=0;i<res.size();i++){
        res[i]=(a[i]+b[i])%P;
    }
    return res;
}
friend Poly operator- (const Poly &a,const Poly &b){
    Poly res(max(a.size(),b.size()));
    for(int i=0;i<res.size();i++){
        res[i]=(a[i]-b[i]+P)%P;
    }
    return res;
}
auto begin()const{
    return a.begin();
}
auto end()const{
    return a.end();
}
};

void DAC(Poly &f,Poly &g,int l,int r){
    if(l==r-1) return;
    int mid=(l+r)/2;
    DAC(f,g,l,mid);
    auto t=Poly(f.begin()+l,f.begin()+mid)*g.trunc(r-l);
    for(int i=mid;i<r;i++){
        f[i]=(f[i]+t[i-1])%P;
    }
    DAC(f,g,mid,r);
};

vector<Poly>Q;

Poly MULT(Poly a,Poly b){
    int n=a.size(),m=b.size();
    reverse(b.a.begin(),b.a.end());
    b=a*b;
    for(int i=0;i<n;++i) a[i]=b[i+m-1];
    return a;
}

```

```

void MPinit(Poly &a,int u,int cl,int cr){
    if(cl==cr){
        Q[u].resize(2);
        Q[u][0]=1,Q[u][1]=mod-a[cl];
        return;
    }
    int mid=cl+cr>>1;
    MPinit(a,u<<1,cl,mid);MPinit(a,u<<1|1,mid+1,cr);
    Q[u]=Q[u<<1]*Q[u<<1|1];
}

void MPCal(int u,int cl,int cr,Poly f,Poly &g){
    f.resize(cr-cl+1);
    if(cl==cr){
        g[cl]=f[0];
        return;
    }
    int mid=cl+cr>>1;
    MPCal(u<<1,cl,mid,MulT(f,Q[u<<1|1]),g);
    MPCal(u<<1|1,mid+1,cr,MulT(f,Q[u<<1]),g);
}

Poly Multipoints(Poly f,Poly a,int n){
    f.resize(n+1),a.resize(n);
    Poly v(n);
    Q.resize(n<<2);
    MPinit(a,1,0,n-1);
    MPCal(1,0,n-1,MulT(f,Q[1].inv(n+1)),v);
    return v;
}

```

Fast

```

#include <bits/stdc++.h>
using namespace std;
using i64 = long long;

constexpr int P = 998244353;
//1004535809, 469762049

template<int P = P>
constexpr int power(int a, i64 b) {
    int res = 1;
    while (b) {
        if (b & 1) res = 1LL * a * res % P;
        a = 1LL * a * a % P;
        b /= 2;
    }
    return res;
}

template<int V, int P>
constexpr int CInv = power<P>(V, P - 2);

template<int P>
vector<int> roots{0, 1};

```

```

vector<int> rev;
template<int P>
void dft(vector<int> &a) {
    int n = a.size();
    if (rev.size() != n) {
        rev.resize(n);
        int k = __builtin_ctz(n) - 1;
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0 ; i < n; i++) {
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    }

    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);

        while ((1 << k) < n) {
            int e = power<P>(3, (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < 1 << k; i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = 1LL * roots<P>[i] * e % P;
            }
            k++;
        }
    }
}

for (int k = 1; k < n; k *= 2) {
    for (int i = 0; i < n; i += 2 * k) {
        for (int j = 0; j < k; j++) {
            int u = a[i + j], v = 1LL * a[i + j + k] * roots<P>[k + j] % P;
            a[i + j] = (u + v) % P;
            a[i + j + k] = (u - v + P) % P;
        }
    }
}
}

template<int P>
void idft(vector<int> &a) {
    reverse(a.begin() + 1, a.end());
    dft<P>(a);
    int n = a.size(), inv = (1 - P) / n + P;
    for (int i = 0; i < n; i++) {
        a[i] = 1LL * a[i] * inv % P;
    }
}

template<int P = 998244353>
struct Poly {
    vector<int> a;
    Poly() {}
    explicit Poly(int n, int val = 0) : a(n, val) {}
    Poly(const vector<int> &a) : a(a) {}
    Poly(const vector<int> &&a) : a(a) {}
    Poly(const initializer_list<int> &t) : a(t) {}
}

```

```

template<class It>
explicit constexpr Poly(It first, It last) : a(first, last) {}
template<class F>
explicit constexpr Poly(int n, const F &f) : a(n) {
    for (int i = 0; i < n; i++) {
        a[i] = f(i);
    }
}

Poly shift(int k) const {
    if (k >= 0) {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return b;
    } else if (size() <= -k) {
        return Poly();
    } else {
        return Poly(a.begin() - k, a.end());
    }
}
//ok
int size() const {
    return a.size();
}

constexpr Poly& resize(int k) {
    this->a.resize(k);
    return *this;
}

//ok
Poly trunc(int k) const {
    if (k <= size()) return Poly(a.begin(), a.begin() + k);
    return Poly{a}.resize(k);
}

bool empty() const {
    return a.empty();
}
//ok
int& operator[](int idx) {
    return a[idx];
}
//ok
int operator[] (int idx) const {
    if (idx < size()) return a[idx];
    return 0;
}

void push_back(int v) {
    a.push_back(v);
}

Poly deriv() const {
    if (a.empty()) return Poly();
    Poly res(size() - 1);
    for (int i = 0; i < size() - 1; i++) {
        res[i] = 1LL * (i + 1) * a[i + 1] % P;
    }
}

```

```

        return res;
    }
Poly integr() const {
    Poly res(size() + 1);
    for (int i = 1; i <= size(); i++) {
        res[i] = 1LL * a[i - 1] * power<P>(i, P - 2) % P;
    }
    return res;
}
//ok
Poly inv(int n) const {
    Poly res{power<P>(a[0], P - 2)};
    int k = 1;
    while (k < n) {
        k *= 2;
        res = res * (Poly{2} - trunc(k) * res).resize(k);
    }
    return res.resize(n);
}
Poly sqrt(int n) const {
    Poly res{1};
    int k = 1;
    while (k < n) {
        k *= 2;
        res = (trunc(k) * res.inv(k) + res).resize(k) * CInv<2, P>;
    }
    return res.resize(n);
}
Poly log(int n) const {
    return (deriv() * inv(n)).integr().resize(n);
}
Poly exp(int n) const {
    Poly res{1};
    int k = 1;
    while (k < n) {
        k *= 2;
        res = res * (Poly{1} - res.log(k) + trunc(k)).resize(k);
    }
    return res.resize(n);
}
Poly pow(int k, int n) const {
    int i = 0;
    while (i < size() && a[i] == 0) {
        i++;
    }
    if (i == size() || 1LL * i * k >= n) {
        return Poly(n);
    }
    int v = a[i];
    auto f = shift(-i) * power(v, P - 2);
    return (f.log(n - i * k) * k).exp(n - i).shift(i * k) * power(v, k);
}
Poly pow(const string &s, int n) {
    int k = 0, k1 = 0;
    for (auto x : s) {
        k = (10LL * k + x - '0') % P;
        k1 = (10LL * k1 + x - '0') % (P - 1);
    }
}

```

```

    }
    int i = 0;
    while (i < size() && a[i] == 0) {
        i++;
    }
    if ((i && (s.size() > 10 || stoll(s) > n)) || i == size() || 1LL * i * k >=
n) {
        return Poly(n);
    }
    int v = a[i];
    auto f = shift(-i) * power(v, P - 2);
    return (f.log(n - i * k) * k).exp(n - i).shift(i * k) * power(v, k1);
}

//ok
friend Poly operator* (Poly a, Poly b) {
    if (a.empty() || b.empty()) return Poly();
    if (a.size() > b.size()) swap(a, b);
    if (a.size() < 64) {
        Poly c(a.size() + b.size() - 1);
        for (int i = 0; i < a.size(); i++) {
            for (int j = 0; j < b.size(); j++) {
                c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
            }
        }
        return c;
    }
    int sz = 1, tot = a.size() + b.size() - 1;
    while (sz < tot) sz *= 2;
    a.resize(sz);
    b.resize(sz);

    dft<P>(a.a);
    dft<P>(b.a);
    for (int i = 0; i < sz; i++) {
        a[i] = 1LL * a[i] * b[i] % P;
    }

    idft<P>(a.a);
    a.resize(tot);
    return a;
}

Poly& operator/= (int k) {
    return (*this) = (*this) / k;
}
Poly& operator*= (int k) {
    return (*this) = (*this) * k;
}
Poly& operator*= (const Poly &b) {
    return (*this) = (*this) * b;
}
Poly& operator+= (const Poly &b) {
    return (*this) = (*this) + b;
}
Poly& operator-= (const Poly &b) {
    return (*this) = (*this) - b;
}

```

```

friend Poly operator/ (Poly a, int k) {
    int inv = power<P>(k, P - 2);
    for (int i = 0; i < a.size(); i++) {
        a[i] = 1LL * a[i] * inv % P;
    }
    return a;
}
friend Poly operator* (Poly a, int k) {
    for (int i = 0; i < a.size(); i++) {
        a[i] = 1LL * a[i] * k % P;
    }
    return a;
}
friend Poly operator* (int k, Poly a) {
    for (int i = 0; i < a.size(); i++) {
        a[i] = 1LL * a[i] * k % P;
    }
    return a;
}
friend Poly operator+ (const Poly &a, const Poly &b) {
    Poly res(max(a.size(), b.size()));
    for (int i = 0; i < res.size(); i++) {
        res[i] = 1LL * (a[i] + b[i]) % P;
    }
    return res;
}
friend Poly operator- (const Poly &a, const Poly &b) {
    Poly res(max(a.size(), b.size()));
    for (int i = 0; i < res.size(); i++) {
        res[i] = 1LL * (a[i] - b[i] + P) % P;
    }
    return res;
}

//ok
auto begin() const {
    return a.begin();
}
auto end() const {
    return a.end();
}
;

template<int P>
void DAC(Poly<P> &f, Poly<P> &g, int l, int r) {
    if (l == r - 1) return;
    int mid = (l + r) / 2;
    DAC(f, g, l, mid);
    auto t = Poly(f.begin() + 1, f.begin() + mid) * g.trunc(r - 1);

    for (int i = mid; i < r; i++) {
        f[i] = (f[i] + t[i - 1]) % P;
    }
    DAC(f, g, mid, r);
};

int main(){
    int n;
    cin>>n;

```

```

Poly x(n);
for(int i=0;i<n;++i) cin>>x[i];
Poly y=x.exp(n);
for(int i=0;i<n;++i) cout<<y[i]<<" \n"[i==n];
}

```

连续拉格朗日插值法

```

int Lagrange(int *f,int k,int n) {
    if(k<=n) return f[k];
    pre[0]=suf[n]=1;
    for(int i=1;i<=n;i++) pre[i]=pre[i-1]*(k-i+1)%mod;
    for(int i=n;i>=1;i--) suf[i-1]=suf[i]*(k-i)%mod;
    int ans=0;
    for(int i=0;i<=n;i++){
        int opt=(n-i)&1?-1:1;
        ans=(ans+opt*pre[i]%mod*suf[i]%mod*inv[i]%mod*inv[n-i]%mod*f[i]%mod+mod)%mod;
    }
    ans=(ans%mod+mod)%mod;
    return ans;
}

```

非连续拉格朗日插值法

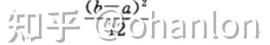
```

int Lagrange(vector<int>x,vector<int>y,int n,int k){
    for(int i=0;i<n;++i) if(x[i]==k) return y[i];
    vector<int>inv(n,1);
    for(int i=0;i<n;++i){
        for(int j=i+1;j<n;++j){
            inv[i]=inv[i]*(x[i]-x[j]+mod)%mod;
            inv[j]=inv[j]*(x[j]-x[i]+mod)%mod;
        }
    }
    int sum=1,ans=0;
    for(int i=0;i<n;++i) sum=sum*(k-x[i]+mod)%mod;
    for(int i=0;i<n;++i){
        int tmp=inv[i]*(k-x[i]+mod)%mod;
        ans=(ans+y[i]*sum%mod*qpow(tmp,mod-2)%mod)%mod;
    }
    return ans;
}

```

概率论：

16、常用分布

分布	参数	分布律或概率密度	数学期望	方差
(0-1)分布	$0 < p < 1$	$P\{X=k\} = p^k(1-p)^{1-k}, k=0,1$	p	$p(1-p)$
二项分布	$n \geq 1$ $0 < p < 1$	$P\{X=k\} = \binom{n}{k} p^k (1-p)^{n-k}$ $k=0,1,\dots,n$	np	$np(1-p)$
负二项分布 (巴斯卡分布)	$r \geq 1$ $0 < p < 1$	$P\{X=k\} = \binom{k-1}{r-1} p^r (1-p)^{k-r}$ $k=r,r+1,\dots$	$\frac{r}{p}$	$\frac{r(1-p)}{p^2}$
几何分布	$0 < p < 1$	$P\{X=k\} = (1-p)^{k-1} p$ $k=1,2,\dots$	$\frac{1}{p}$	$\frac{1-p}{p^2}$
超几何分布	N, M, n $(M \leq N)$ $(n \leq N)$	$P\{X=k\} = \frac{\binom{M}{k} \binom{N-M}{n-k}}{\binom{N}{n}}$ k 为整数, $\max\{0, n-N+M\} \leq k \leq \min\{n, M\}$	$\frac{nM}{N}$	$\frac{nM}{N} \left(1 - \frac{M}{N}\right) \left(\frac{N-n}{N-1}\right)$
泊松分布	$\lambda > 0$	$P\{X=k\} = \frac{\lambda^k e^{-\lambda}}{k!}$ $k=0,1,2,\dots$	λ	λ
均匀分布	$a < b$	$f(x) = \begin{cases} \frac{1}{b-a}, & a < x < b \\ 0, & \text{其他} \end{cases}$	$\frac{a+b}{2}$	知乎  hanlon

七、计算几何

1、几何公式

1.1 三角形

1.半周长 $P = (a + b + c)/2$

2.面积 $S = aHa/2 = ab\sin(C)/2 = \sqrt{P(P - a)(P - b)(P - c)}$

3.中线 $Ma = \sqrt{2(b^2 + c^2) - a^2}/2 = \sqrt{b^2 + c^2 + 2bccos(A)}/2$

4.角平分线 $Ta = \sqrt{bc((b + c)^2 - a^2)}/(b + c) = 2bccos(A/2)/(b + c)$

5.高线 $Ha = b\sin(C) = c\sin(B) = \sqrt{b^2 - ((a^2 + b^2 - c^2)/(2a))^2}$

6.内切圆半径 $r = S/P = \sin(B/2)\sin(C/2)/\sin((B + C)/2)$

$$= 4R\sin(A/2)\sin(B/2)\sin(C/2) = \sqrt{(P - a)(P - b)(P - c)/P}$$

$$= Ptan(A/2)\tan(B/2)\tan(C/2)$$

7.外接圆半径 $R = abc/(4S) = a/(2\sin(A)) = b/(2\sin(B)) = c/(2\sin(C))$

2、直线与线段

3、圆

4、凸包

Andrew求凸包

```
#include<algorithm>
#include<cmath>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
#define pi acos(-1.0)
using namespace std;

const int maxn=1e5+5;
int n,top,tmp;
struct Point{
    double x,y;
    Point(double xx=0,double yy=0):x(xx),y(yy){}
}pt[maxn],st[maxn];

double cross(Point p0,Point p1,Point p2){
    return(p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

bool cmp(Point a,Point b){
    if(a.x!=b.x) return a.x<b.x;
    return a.y<b.y;
}

double dis(Point a,Point b){
```

```
    return sqrt((b.x-a.x)*(b.x-a.x)+(b.y-a.y)*(b.y-a.y));
}

void andrew(){
    top=1;
    st[1]=pt[1];
    for(int i=2;i<=n;++i){ //求下凸包
        while(top>1 && cross(st[top-1],st[top],pt[i])<=0) --top;
        st[++top]=pt[i];
    }
    tmp=top;
    for(int i=n;i-->0){ //求上凸包
        while(top>tmp+1 && cross(st[top-1],st[top],pt[i])<=0) --top;
        st[++top]=pt[i];
    }
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin>>n;
    for(int i=1;i<=n;++i) cin>>pt[i].x>>pt[i].y;
    sort(pt+1,pt+n+1,cmp);
    andrew();
    for(int i=1;i<=top;++i) cout<<st[i].x<<" "<<st[i].y<<endl;
    return 0;
}
```

八、杂项

1、生成树计数

Prüfer 序列

每次选择一个编号最小的叶结点并删掉它，然后在序列中记录下它连接到的那个结点。重复 $n - 2$ 次后就只剩下两个结点，算法结束。

Prüfer 序列的性质

1. 在构造完 Prüfer 序列后原树中会剩下两个结点，其中一个一定是编号最大的点 n 。
2. 每个结点在序列中出现的次数是其度数减 1。（没有出现的就是叶结点）

n 个点的标记无根树方案数

$$n^{n-2}$$

n 个点的标记有根树方案数

$$n^{n-1}$$

n 个点的标记无根树方案数，要求第 i 个点度为 d_i

$$\frac{(n-2)!}{\prod(d_i-1)!}$$

矩阵树定理

矩阵树定理解决了一张图的生成树个数计数问题。

本篇记号声明

本篇中的图，无论无向还是有向，都允许重边，但是默认没有自环。

自环并不影响生成树的个数，也不影响下文中 Laplace 矩阵的计算，故而矩阵树定理对有自环的情形依然成立。计算时不必删去自环。如果删去自环，会影响根据 BEST 定理应用矩阵树定理统计有向图的欧拉回路个数。

无向图情况

设 G 是一个有 n 个顶点的无向图。定义度数矩阵 $D(G)$ 为

$$D_{ii}(G) = \deg(i), D_{ij} = 0, i \neq j.$$

设 $\#e(i, j)$ 为点 i 与点 j 相连的边数，并定义邻接矩阵 A 为

$$A_{ij}(G) = A_{ji}(G) = \#e(i, j), i \neq j.$$

定义 Laplace 矩阵（亦称 Kirchhoff 矩阵） L 为

$$L(G) = D(G) - A(G).$$

记图 G 的所有生成树个数为 $t(G)$ 。

有向图情况

设 G 是一个有 n 个顶点的有向图。定义出度矩阵 $D^{\text{out}}(G)$ 为

$$D_{ii}^{\text{out}}(G) = \deg^{\text{out}}(i), D_{ij}^{\text{out}} = 0, i \neq j.$$

类似地定义入度矩阵 $D^{\text{in}}(G)$ 。

设 $\#e(i, j)$ 为点 i 指向点 j 的有向边数，并定义邻接矩阵 A 为

$$A_{ij}(G) = \#e(i, j), i \neq j.$$

定义出度 Laplace 矩阵 L^{out} 为

$$L^{\text{out}}(G) = D^{\text{out}}(G) - A(G).$$

定义入度 Laplace 矩阵 L^{in} 为

$$L^{\text{in}}(G) = D^{\text{in}}(G) - A(G).$$

记图 G 的以 k 为根的所有根向树形图个数为 $t^{\text{root}}(G, k)$ 。所谓根向树形图，是说这张图的基图是一棵树，所有的边全部指向父亲。

记图 G 的以 k 为根的所有叶向树形图个数为 $t^{\text{leaf}}(G, k)$ 。所谓叶向树形图，是说这张图的基图是一棵树，所有的边全部指向儿子。

定理叙述

矩阵树定理具有多种形式。

定义 $[n] = \{1, 2, \dots, n\}$ ，矩阵 A 的子矩阵 $A_{S,T}$ 为选取 $A_{i,j}$ ($i \in S, j \in T$) 的元素得到的子矩阵。

定理 1 (矩阵树定理, 无向图, 行列式形式)

对于无向图 G 和任意的 k ，都有

$$t(G) = \det L(G)_{[n] \setminus \{k\}, [n] \setminus \{k\}}.$$

也就是说，无向图的 Laplace 矩阵所有 $n - 1$ 阶主子式都相等，且都等于图的生成树的个数。

推论 1 (矩阵树定理, 无向图, 特征值形式)

设 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{n-1} \geq \lambda_n = 0$ 为 $L(G)$ 的 n 个特征值，那么有

$$t(G) = \frac{1}{n} \lambda_1 \lambda_2 \dots \lambda_{n-1}.$$

定理 2 (矩阵树定理, 有向图根向树, 行列式形式)

对于有向图 G 和任意的 k ，都有

$$t^{\text{root}}(G, k) = \det L^{\text{out}}(G)_{[n] \setminus \{k\}, [n] \setminus \{k\}}.$$

也就是说，有向图的出度 Laplace 矩阵删去第 k 行第 k 列得到的主子式等于以 k 为根的叶向树形图的个数。

因此如果要统计一张图所有的根向树形图，只要枚举所有的根 k 并对 $t^{\text{root}}(G, k)$ 求和即可。

定理 3 (矩阵树定理, 有向图叶向树, 行列式形式)

对于有向图 G 和任意的 k ，都有

$$t^{\text{leaf}}(G, k) = \det L^{\text{in}}(G)_{[n] \setminus \{k\}, [n] \setminus \{k\}}.$$

也就是说，有向图的入度 Laplace 矩阵删去第 k 行第 k 列得到的主子式等于以 k 为根的根向树形图的个数。

因此如果要统计一张图所有的叶向树形图，只要枚举所有的根 k 并对 $t^{\text{leaf}}(G, k)$ 求和即可。

根向树形图也被称为内向树形图，但因为计算内向树形图用的是出度，为了不引起 in 和 out 的混淆，所以采用了根向这一说法。

BEST 定理

这一定理将有向欧拉图中欧拉回路的数目和该图的根向树形图的数目联系起来，从而解决了有向图中的欧拉回路的计数问题。注意，任意无向图中的欧拉回路的计数问题是 NP 完全的。

在实现该算法时，应当首先判定给定图是否是欧拉图，移除所有零度顶点，然后建图计算根向树形图的个数，并由 BEST 定理得到欧拉回路的计数。注意，如果所求欧拉回路个数要求以给定点作为起点，需要将答案再乘上该点出度，相当于枚举回路中首条边。

在证明 BEST 定理之前，需要知道如下结论。

性质（有向图具有欧拉回路的判定）

一个有向图具有欧拉回路，当且仅当非零度顶点是强连通的，且所有顶点的出度和入度相等。

对于欧拉图，因为出度和入度相等，可以将它们略去上标，记作 $\deg(v)$ 。BEST 定理可以叙述如下。

定理 6 (BEST 定理)

设 G 是有向欧拉图， k 为任意顶点，那么 G 的不同欧拉回路总数 $\text{ec}(G)$ 是

$$\text{ec}(G) = t^{\text{root}}(G, k) \prod_{v \in V} (\deg(v) - 1)!$$

这也说明，对欧拉图 G 的任意两个节点 k, k' ，都有 $t^{\text{root}}(G, k) = t^{\text{root}}(G, k')$ 。

证明

证明的大致思路是建立以 k 为起点的欧拉回路和以 k 为根的根向树形图以及各个顶点处出边的排列的对应关系。在指定欧拉回路的顶点后，需要证明的计数应当等于

$$\deg(k)\text{ec}(G) = t^{\text{root}}(G, k) \deg(k)! \prod_{v \neq k} (\deg(v) - 1)!$$

这一计数的组合含义对应的构造如下。对于起点为 k 的欧拉回路，根据回路中每条边的出现顺序，可以构造出

- 一个以 k 为根的根向树形图，由所有非根顶点处的最后一条出边组成，即 $t^{\text{root}}(G, k)$ ，
- 根 k 处所有出边的排列顺序，即 $\deg(k)!$ ，和
- 非根顶点 $v \neq k$ 处除去最后一条出边之外的其他所有出边的排列顺序，即 $(\deg(v) - 1)!$ 。

下面说明，这样的构造得到的映射是双射。

一方面，给定欧拉回路，要证明所有非根顶点处的最后一条出边组成了一个根向树形图。根据构造，树中每个非根顶点的确只有一条出边，所以只需要证明这些出边不会成环。注意到，如果将所有顶点根据它在欧拉回路最后一次出现的顺序排序，那么非根顶点的最后一次出边必然指向顺序严格更靠后的顶点。如果存在环，那么环中就有一个顺序最靠后的顶点，因为它在环中，所以它指向了一个顺序并不靠后的点，这与上文矛盾。所以，非根顶点的最后一次出边必然构成根向树形图。

另一方面，给定任意根向树形图和其余出边的排列顺序，可以复原出一条欧拉回路，使得该欧拉回路经上述构造后可以得到给定的根向树形图和其余出边的排列顺序。对此，只需要从根 k 出发，每当到达一个顶点时，都根据给定的该顶点的出边排列顺序，选择顺序最靠前的、尚未经过的出边作为欧拉回路中本次的出边；如果该顶点处的排列中所有出边都已经经过了，就选择根向树形图中该顶点的出边作为欧拉回路中本次的出边。因为图是欧拉图，每个顶点的入度都等于出度，所以，这一过程不会在非根顶点处终止，即所得路径的确是回路。要证明所得路径是合法的欧拉回路，只需要证明这一过程能够遍历所有边就可以。

如果不能，则必然有某个顶点 v 的某个出边没有遍历到。考察顶点 v 。顶点 v 不能是根，因为最后会终止在根，如果根仍有出边剩余，这与过程终止矛盾。所以， v 必然不是根。根据前文描述的过程，只要非根顶点 v 有任何出边剩余，那么非根顶点在树中的出边 e 必然剩余。记 $e = (v, u)$ 。因为 u 的某个入边没有遍历到，根据 u 的出度等于入度，必然有 u 的某条出边没有遍历到。然后，可以类似地考察顶点 u 。这些推理将考察的顶点从 v 移动到了 u ，即沿着根向树形图向树的根移动了一步。可以归纳地证明，此时必有根 k 的某个出边没有遍历到。前文已经说明这不可能，故得到矛盾。这说明，上一段所得路径的确是合法的欧拉回路。

可以验证这些映射都是单射，则必然同为双射。原命题得证。

实现

根据图写出 Laplace 矩阵，删去一行一列，求所得矩阵的行列式即可。求行列式可以使用 Gauss–Jordan 消元法。

例如，一个正方形图的生成树个数

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix}$$

$$\begin{vmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{vmatrix} = 4$$

可以用 Gauss–Jordan 消元解决，时间复杂度为 $O(n^3)$ 。

```
#include <algorithm>
#include <cassert>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
constexpr int MOD = 100000007;
constexpr double eps = 1e-7;
struct matrix {
    static constexpr int MAXN = 20;
    int n, m;
    double mat[MAXN][MAXN];

    matrix() { memset(mat, 0, sizeof(mat)); }

    void print() {
        cout << "MATRIX " << n << " " << m << endl;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                cout << mat[i][j] << "\t";
            }
            cout << endl;
        }
    }

    void random(int n) {
        this->n = n;
        this->m = n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) mat[i][j] = rand() % 100;
    }
};
```

```

}

void initSquare() {
    this->n = 4;
    this->m = 4;
    memset(mat, 0, sizeof(mat));
    mat[0][1] = mat[0][3] = 1;
    mat[1][0] = mat[1][2] = 1;
    mat[2][1] = mat[2][3] = 1;
    mat[3][0] = mat[3][2] = 1;
    mat[0][0] = mat[1][1] = mat[2][2] = mat[3][3] = -2;
    this->n--; // 去一行
    this->m--; // 去一列
}

double gauss() {
    double ans = 1;
    for (int i = 0; i < n; i++) {
        int sid = -1;
        for (int j = i; j < n; j++)
            if (abs(mat[j][i]) > eps) {
                sid = j;
                break;
            }
        if (sid == -1) continue;
        if (sid != i) {
            for (int j = 0; j < n; j++) {
                swap(mat[sid][j], mat[i][j]);
                ans = -ans;
            }
        }
        for (int j = i + 1; j < n; j++) {
            double ratio = mat[j][i] / mat[i][i];
            for (int k = 0; k < n; k++) {
                mat[j][k] -= mat[i][k] * ratio;
            }
        }
    }
    for (int i = 0; i < n; i++) ans *= mat[i][i];
    return abs(ans);
}
};

int main() {
    srand(1);
    matrix T;
    // T.random(2);
    T.initSquare();
    T.print();
    double ans = T.gauss();
    T.print();
    cout << ans << endl;
}

```

2、点分治

```

struct CenDec{
    int cen,n;

```

```

vector<int>sz;
vector<bool>del;
vector<vector<int>>e;
CenDec(int _n){
    n=_n;
    sz.assign(n+1,0);
    del.assign(n+1,false);
    e.assign(n+1,vector<int>());
}
void dfs(int u,int fa){
    sz[u]=1;
    int ans=0;
    for(auto v:e[u]){
        if(del[v] || v==fa) continue;
        dfs(v,u);
        if(cen!=-1) return;
        ans=max(ans,sz[v]);
        sz[u]+=sz[v];
    }
    ans=max(ans,n-sz[u]);
    if(ans<=n/2){
        cen=u;
        sz[fa]=n-sz[u];
    }
}
int res=0;
.....
void run(int u){

.....
del[u]=1;
for(auto v:e[u]){
    if(del[v]) continue;
    n=sz[v];
    cen=-1;
    dfs(v,0);
    run(cen);
}
}
int count(){
.....
cen=-1;
dfs(1,0);
run(cen);
return res;
}
};


```

3、有向无环图上不相交路径计数

简介

Lindström–Gessel–Viennot lemma, 即 LGV 引理, 可以用来处理有向无环图上不相交路径计数等问题。

LGV 引理仅适用于 **有向无环图**。

定义

$\omega(P)$ 表示 P 这条路径上所有边的边权之积。 (路径计数时, 可以将边权都设为 1) (事实上, 边权可以为生成函数)

$e(u, v)$ 表示 u 到 v 的 **每一条** 路径 P 的 $\omega(P)$ 之和, 即 $e(u, v) = \sum_{P: u \rightarrow v} \omega(P)$ 。

起点集合 A , 是有向无环图点集的一个子集, 大小为 n 。

终点集合 B , 也是有向无环图点集的一个子集, 大小也为 n 。

一组 $A \rightarrow B$ 的不相交路径 S : S_i 是一条从 A_i 到 $B_{\sigma(S)_i}$ 的路径 ($\sigma(S)$ 是一个排列), 对于任何 $i \neq j$, S_i 和 S_j 没有公共顶点。

$t(\sigma)$ 表示排列 σ 的逆序对个数。

引理

$$M = \begin{bmatrix} e(A_1, B_1) & e(A_1, B_2) & \cdots & e(A_1, B_n) \\ e(A_2, B_1) & e(A_2, B_2) & \cdots & e(A_2, B_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(A_n, B_1) & e(A_n, B_2) & \cdots & e(A_n, B_n) \end{bmatrix}$$
$$\det(M) = \sum_{S: A \rightarrow B} (-1)^{t(\sigma(S))} \prod_{i=1}^n \omega(S_i)$$

其中 $\sum_{S: A \rightarrow B}$ 表示满足上文要求的 $A \rightarrow B$ 的每一组不相交路径 S 。

证明

由行列式定义可得

$$\begin{aligned} \det(M) &= \sum_{\sigma} (-1)^{t(\sigma)} \prod_{i=1}^n e(a_i, b_{\sigma(i)}) \\ &= \sum_{\sigma} (-1)^{t(\sigma)} \prod_{i=1}^n \sum_{P: a_i \rightarrow b_{\sigma(i)}} \omega(P) \end{aligned}$$

观察到 $\prod_{i=1}^n \sum_{P: a_i \rightarrow b_{\sigma(i)}} \omega(P)$, 实际上是所有从 A 到 B 排列为 σ 的路径组 P 的 $\omega(P)$ 之和。

$$\begin{aligned} &\sum_{\sigma} (-1)^{t(\sigma)} \prod_{i=1}^n \sum_{P: a_i \rightarrow b_{\sigma(i)}} \omega(P) \\ &= \sum_{\sigma} (-1)^{t(\sigma)} \sum_{P=\sigma} \omega(P) \\ &= \sum_{P: A \rightarrow B} (-1)^{t(\sigma)} \prod_{i=1}^n \omega(P_i) \end{aligned}$$

此处 P 为任意路径组。

设 U 为不相交路径组, V 为相交路径组,

$$\begin{aligned} & \sum_{P:A \rightarrow B} (-1)^{t(\sigma)} \prod_{i=1}^n \omega(P_i) \\ &= \sum_{U:A \rightarrow B} (-1)^{t(U)} \prod_{i=1}^n \omega(U_i) + \sum_{V:A \rightarrow B} (-1)^{t(V)} \prod_{i=1}^n \omega(V_i) \end{aligned}$$

设 P 中存在一个相交路径组 $P_i : a_1 \rightarrow u \rightarrow b_1, P_j : a_2 \rightarrow u \rightarrow b_2$, 则必然存在和它相对的一个相交路径组 $P'_i = a_1 \rightarrow u \rightarrow b_2, P'_j = a_2 \rightarrow u \rightarrow b_1$, P' 的其他路径与 P 相同。可得 $\omega(P) = \omega(P')$, $t(P) = t(P') \pm 1$.

因此我们有 $\sum_{V:A \rightarrow B} (-1)^{t(\sigma)} \prod_{i=1}^n \omega(V_i) = 0$.

则 $\det(M) = \sum_{U:A \rightarrow B} (-1)^{t(U)} \prod_{i=1}^n \omega(U_i) = 0$.

证毕。

例题

例 1 CF348D Turtles

题意：有一个 $n \times m$ 的格点棋盘，其中某些格子可走，某些格子不可走。有一只海龟从 (x, y) 只能走到 $(x+1, y)$ 和 $(x, y+1)$ 的位置，求海龟从 $(1, 1)$ 到 (n, m) 的不相交路径数对 $10^9 + 7$ 取模之后的结果。 $2 \leq n, m \leq 3000$ 。

比较直接的 LGV 引理的应用。考虑所有合法路径，发现从 $(1, 1)$ 出发一定要经过 $A = \{(1, 2), (2, 1)\}$ ，而到达终点一定要经过 $B = \{(n-1, m), (n, m-1)\}$ ，则 A, B 可立即选定。应用 LGV 引理可得答案为：

$$\begin{vmatrix} f(a_1, b_1) & f(a_1, b_2) \\ f(a_2, b_1) & f(a_2, b_2) \end{vmatrix} = f(a_1, b_1) \times f(a_2, b_2) - f(a_1, b_2) \times f(a_2, b_1)$$

其中 $f(a, b)$ 为图上 $a \rightarrow b$ 的路径数，带有障碍格点的路径计数问题可以直接做一个 $O(nm)$ 的 dp，则 f 易求。最终复杂度 $O(nm)$ 。

```
#include <cstring>
#include <iostream>
#include <string>

using namespace std;

using ll = long long;
constexpr int MOD = 1e9 + 7;
constexpr int SIZE = 3010;

string board[SIZE];
int dp[SIZE][SIZE];

int f(int x1, int y1, int x2, int y2) {
    memset(dp, 0, sizeof dp);

    dp[x1][y1] = board[x1][y1] == '.';
    for (int i = 1; i <= x2; i++) {
        for (int j = 1; j <= y2; j++) {
            if (board[i][j] == '#') {
                continue;
            }
            dp[i][j] = (dp[i][j] + dp[i - 1][j]) % MOD;
            dp[i][j] = (dp[i][j] + dp[i][j - 1]) % MOD;
        }
    }
}
```

```

        }
    }

    return dp[x2][y2] % MOD;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;

    for (int i = 1; i <= n; i++) {
        cin >> board[i];
        board[i] = " " + board[i];
    }

    ll f11 = f(1, 2, n - 1, m);
    ll f12 = f(1, 2, n, m - 1);
    ll f21 = f(2, 1, n - 1, m);
    ll f22 = f(2, 1, n, m - 1);

    ll ans = ((f11 * f22) % MOD - (f12 * f21) % MOD + MOD) % MOD;
    cout << ans << '\n';

    return 0;
}

```

例 2 HDU 5852 Intersection is not allowed

题意：有一个 $n \times n$ 的棋盘，一个棋子从 (x, y) 只能走到 $(x, y+1)$ 或 $(x+1, y)$ ，有 k 个棋子，一开始第 i 个棋子放在 $(1, a_i)$ ，最终要到 (n, b_i) ，路径要两两不相交，求方案数对 $10^9 + 7$ 取模。

$1 \leq n \leq 10^5$, $1 \leq k \leq 100$, 保证 $1 \leq a_1 < a_2 < \dots < a_n \leq n$, $1 \leq b_1 < b_2 < \dots < b_n \leq n$ 。

观察到如果路径不相交就一定是 a_i 到 b_i ，因此 LGV 引理中一定有 $\sigma(S)_i = i$ ，不需要考虑符号问题。边权设为 1，直接套用引理即可。

从 $(1, a_i)$ 到 (n, b_j) 的路径条数相当于从 $n - 1 + b_j - a_i$ 步中选 $n - 1$ 步向下走，所以 $e(A_i, B_j) = \binom{n-1+b_j-a_i}{n-1}$ 。

行列式可以使用高斯消元求。

复杂度为 $O(n + k(k^2 + \log p))$ ，其中 $\log p$ 是求逆元复杂度。

```

#include <algorithm>
#include <iostream>

using ll = long long;

constexpr int K = 105;
constexpr int N = 100005;
constexpr int mod = 1e9 + 7;

int T, n, k, a[K], b[K], fact[N << 1], m[K][K];

int qpow(int x, int y) {
    int out = 1;
    while (y) {
        if (y & 1) out = (ll)out * x % mod;
        x = (ll)x * x % mod;
        y >>= 1;
    }
    return out;
}

int solve() {
    fact[0] = 1;
    for (int i = 1; i < N; i++) fact[i] = fact[i - 1] * i % mod;
    for (int i = 0; i < K; i++) {
        for (int j = 0; j < K; j++) {
            if (i == j) m[i][j] = 1;
            else m[i][j] = 0;
        }
    }
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < K; j++) {
            if (j <= a[i]) m[j][i] = 0;
        }
    }
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < K; j++) {
            if (j >= b[i]) m[i][j] = 0;
        }
    }
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < K; j++) {
            if (m[i][j] == 0) continue;
            for (int l = 0; l < K; l++) {
                if (l != i) m[l][j] = (ll)m[l][j] * qpow(m[i][j], N - 1) % mod;
            }
        }
    }
    ll ans = 1;
    for (int i = 0; i < k; i++) {
        ans *= fact[b[i] - a[i]];
        ans %= mod;
    }
    return ans;
}

int main() {
    cin >> T;
    for (int i = 0; i < T; i++) {
        cin >> n >> k;
        for (int j = 0; j < k; j++) {
            cin >> a[j];
        }
        for (int j = 0; j < k; j++) {
            cin >> b[j];
        }
        cout << solve() << endl;
    }
}

```

```

        x = (ll)x * x % mod;
        y >>= 1;
    }
    return out;
}

int c(int x, int y) {
    return (ll)fact[x] * qpow(fact[y], mod - 2) % mod *
           qpow(fact[x - y], mod - 2) % mod;
}

using std::cin;
using std::cout;

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    fact[0] = 1;
    for (int i = 1; i < N * 2; ++i) fact[i] = (ll)fact[i - 1] * i % mod;

    cin >> T;

    while (T--) {
        cin >> n >> k;

        for (int i = 1; i <= k; ++i) cin >> a[i];
        for (int i = 1; i <= k; ++i) cin >> b[i];

        for (int i = 1; i <= k; ++i) {
            for (int j = 1; j <= k; ++j) {
                if (a[i] <= b[j])
                    m[i][j] = c(b[j] - a[i] + n - 1, n - 1);
                else
                    m[i][j] = 0;
            }
        }

        for (int i = 1; i < k; ++i) {
            if (!m[i][i]) {
                for (int j = i + 1; j <= k; ++j) {
                    if (m[j][i]) {
                        std::swap(m[i], m[j]);
                        break;
                    }
                }
            }
            if (!m[i][i]) continue;
            int inv = qpow(m[i][i], mod - 2);
            for (int j = i + 1; j <= k; ++j) {
                if (!m[j][i]) continue;
                int mul = (ll)m[j][i] * inv % mod;
                for (int p = i; p <= k; ++p) {
                    m[j][p] = (m[j][p] - (ll)m[i][p] * mul % mod + mod) % mod;
                }
            }
        }

        int ans = 1;
        for (int i = 1; i <= k; ++i) ans = (ll)ans * m[i][i] % mod;
    }
}

```

```

    cout << ans << '\n';
}

return 0;
}

```

4、随机游走

树上随机游走

给定一棵有根树，树的某个结点上有一个硬币，在某一时刻硬币会等概率地移动到邻接结点上，问硬币移动到邻接结点上的期望距离。

需要用到的定义

- $T = (V, E)$: 所讨论的树
- $d(u)$: 结点 u 的度数
- $w(u, v)$: 结点 u 与结点 v 之间的边的边权
- p_u : 结点 u 的父结点
- $root$: 树的根结点
- son_u : 结点 u 的子结点集合
- $sibling_u$: 结点 u 的兄弟结点集合

向父结点走的期望距离

设 $f(u)$ 代表 u 结点走到其父结点 p_u 的期望距离，则有：

$$f(u) = \frac{w(u, p_u) + \sum_{v \in son_u} (w(u, v) + f(v) + f(u))}{d(u)}$$

分子中的前半部分代表直接走向了父结点，后半部分代表先走向了子结点再由子结点走回来然后再向父结点走；分母 $d(u)$ 代表从 u 结点走向其任何邻接点的概率相同。

化简如下：

$$\begin{aligned} f(u) &= \frac{w(u, p_u) + \sum_{v \in son_u} (w(u, v) + f(v) + f(u))}{d(u)} \\ &= \frac{w(u, p_u) + \sum_{v \in son_u} (w(u, v) + f(v)) + (d(u) - 1)f(u)}{d(u)} \\ &= w(u, p_u) + \sum_{v \in son_u} (w(u, v) + f(v)) \\ &= \sum_{(u, t) \in E} w(u, t) + \sum_{v \in son_u} f(v) \end{aligned}$$

对于叶子结点 l ，初始状态为 $f(l) = w(p_l, l)$ 。

当树上所有边的边权都为 1 时，上式可化为：

$$f(u) = d(u) + \sum_{v \in son_u} f(v)$$

即 u 子树的所有结点的度数和，也即 u 子树大小的两倍 -1 （每个结点连向其父亲的边都有且只有一条，除 u 与 p_u 之间的边只有 1 点度数的贡献外，每条边会产生 2 点度数的贡献）。

向子结点走的期望距离

设 $g(u)$ 代表 p_u 结点走到其子结点 u 的期望距离，则有：

$$g(u) = \frac{w(p_u, u) + (w(p_u, p_{p_u}) + g(p_u) + g(u)) + \sum_{s \in sibling_u} (w(p_u, s) + f(s) + g(u))}{d(p_u)}$$

分子中的第一部分代表直接走向了子结点 u ，第二部分代表先走向了父结点再由父结点走回来然后再向 u 结点走，第三部分代表先走向 u 结点的兄弟结点再由其走回来然后再向 u 结点走；分母 $d(p_u)$ 代表从 p_u 结点走向其任何邻接点的概率相同。

化简如下：

$$\begin{aligned} g(u) &= \frac{w(p_u, u) + (w(p_u, p_{p_u}) + g(p_u) + g(u)) + \sum_{s \in sibling_u} (w(p_u, s) + f(s) + g(u))}{d(p_u)} \\ &= \frac{w(p_u, u) + w(p_u, p_{p_u}) + g(p_u) + \sum_{s \in sibling_u} (w(p_u, s) + f(s)) + (d(p_u) - 1)g(u)}{d(p_u)} \\ &= w(p_u, u) + w(p_u, p_{p_u}) + g(p_u) + \sum_{s \in sibling_u} (w(p_u, s) + f(s)) \\ &= \sum_{(p_u, t) \in E} w(p_u, t) + g(p_u) + \sum_{s \in sibling_u} f(s) \\ &= \sum_{(p_u, t) \in E} w(p_u, t) + g(p_u) + \left(f(p_u) - \sum_{(p_u, t) \in E} w(p_u, t) - f(u) \right) \\ &= g(p_u) + f(p_u) - f(u) \end{aligned}$$

初始状态为 $g(\text{root}) = 0$ 。

代码实现（以无权树为例）

```
vector<int> G[MAXN];

void dfs1(int u, int p) {
    f[u] = G[u].size();
    for (auto v : G[u]) {
        if (v == p) continue;
        dfs1(v, u);
        f[u] += f[v];
    }
}

void dfs2(int u, int p) {
    if (u != root) g[u] = g[p] + f[p] - f[u];
    for (auto v : G[u]) {
        if (v == p) continue;
        dfs2(v, u);
    }
}
```

图上随机游走

本页介绍图上随机游走问题。主要从网格图、稀疏图、一般图三个角度进行探究，介绍了解决这类问题的各种方法，并对比了它们在解决各种问题时的优缺点。

定义

给定一张有向简单图 $G = (V, E)$ ($V = \{v_1, v_2, \dots, v_{|V|}\}$) 和起点 $s \in V$, 终点 $t \in V$, 每条边 $e = (x, y)$ 有正权值 w_e , 满足 $\forall x \in V \setminus \{t\}$, $\sum_{(x,y) \in E} w_{(x,y)} = 1$, 且对于任意点 x 都存在一条从 x 出发到达 t 的路径。有一枚棋子从起点出发, 每秒从当前所在点 x 以 $w_{(x,y)}$ 的概率选择出边 (x, y) 并走向 y , 到达终点则停止, 求期望花费时间。

事实上, 这个问题也可以写成矩阵的形式。定义矩阵 P :

$$P_{x,y} = \begin{cases} w_{(x,y)} & \text{if } (x, y) \in E \text{ and } x \neq t \\ 0 & \text{if } (x, y) \notin E \text{ or } x = t \end{cases}$$

要求的答案即为:

$$\sum_{k \geq 0} k \times (P^k)_{s,t}$$

其中 $(P^k)_{s,t}$ 表示走了 k 步第一次到达终点的概率。当图有限且所有点都能到达终点时, 由 P 的定义可以证明其特征值都小于 1, 所以答案一定是收敛的。

为了方便描述, 在本页中如无特殊说明, 均用 n 代指 $|V|$, m 代指 $|E|$ 。

另外, 在本页中, 稀疏图指边数和点数同阶的图。

网格图

例题 1 Circles of Waiting

有一枚棋子起始被放在平面直角坐标系的 $(0, 0)$ 点。每秒棋子会随机移动。假设它当前在 (x, y) , 它下一秒有 p_1 的概率移动到 $(x - 1, y)$, p_2 的概率移动到 $(x, y - 1)$, p_3 的概率移动到 $(x + 1, y)$, p_4 的概率移动到 $(x, y + 1)$ 。保证 $p_1 + p_2 + p_3 + p_4 = 1$ 。

求期望经过多少时间它会移动到一个离原点的欧几里得距离大于 R 的位置。 $0 \leq R \leq 50$, $p_1, p_2, p_3, p_4 > 0$, 答案对 $10^9 + 7$ 取模。

朴素做法

记 $f(i, j)$ 表示在棋子在 (i, j) 时移动到一个离原点的欧几里得距离大于 R 的位置的期望时间, 转移方程为:

$$f(i, j) = \begin{cases} p_1 f(i - 1, j) + p_2 f(i, j - 1) + p_3 f(i + 1, j) + p_4 f(i, j + 1) + 1 & i^2 + j^2 \leq R^2 \\ 0 & i^2 + j^2 > R^2 \end{cases}$$

由于转移并不存在拓扑序, 需要使用高斯消元求解。时间复杂度 $O(R^6)$, 无法通过本题。

直接消元法

注意到需要消元的方程的系数大多数都是 0, 消元时只对值非 0 的位置进行计算就可以降低复杂度。

考虑消元的过程, 将方程按照在坐标系中从上到下, 同一层中从左到右的顺序进行消元, 将已经消元过的方程染成黄色, 与黄色点相邻的点染成绿色, 其余点染成黑色, 如下图:

接下来要对下一个绿色格子对应的方程进行消元。在这个方程中, 只有绿色格子和它下方的第一个黑色格子对应的变量系数可能不为 0; 而只有绿色格子和它下方的第一个黑色格子对应的方程中, 当前格子对应的变量系数可能不为 0。

注意到绿色格子只有 $O(R)$ 个，所以单个方程消元的时间复杂度为 $O(R^2)$ 。一共只有 $O(R^2)$ 个方程，所以时间复杂度降低为 $O(R^4)$ ，可以通过本题。

主元法

方程和变量都有 $O(R^2)$ 个，如果能将规模缩小至 $O(R)$ ，那么朴素的高斯消元就能通过了。

将每行从左到右第一个格子对应的变量设为主元，共 $2R + 1$ 个，设法将其他格子对应的变量用关于这些主元的线性函数表示。从左到右逐列考虑，对于当前列的每个格子 (i, j) ，注意到

$f(i, j), f(i - 1, j), f(i, j - 1), f(i, j + 1)$ 都是已知的关于主元的线性函数，将转移方程移项，有：

$$f(i + 1, j) = \frac{f(i, j) - p_1 f(i - 1, j) - p_2 f(i, j - 1) - p_4 f(i, j + 1) - 1}{p_3}$$

这样我们就能得到 $f(i + 1, j)$ 关于主元的线性函数表示。如果 $(i + 1, j)$ 已经离原点欧几里得距离超过 R 了，则可以得到一个方程： $f(i + 1, j) = 0$ 。最终，会得到 $2R + 1$ 个方程，对这些方程进行高斯消元即可。

在递推关于主元的线性函数的阶段，共有 $O(R^2)$ 个变量，递推单个变量需要花费 $O(R)$ 的时间；之后，将问题的规模缩减到了 $O(R)$ 。两部分的时间复杂度均为 $O(R^3)$ ，总的时间复杂度也为 $O(R^3)$ ，可以通过本题。

两种做法的对比

下面从多种方面对比两种做法：

从时间复杂度方面，主元法在网格图上的最坏时间复杂度为 $O(n\sqrt{n})$ （当网格图长和宽都为 $O(\sqrt{n})$ 级别时时间复杂度最高），直接消元法在网格图上最坏时间复杂度为 $O(n^2)$ ，主元法较优。

从精度方面，对于一些需要进行实数计算而不是取模的题目，直接消元法的精度优于主元法。

从适用性方面，两种做法适用于不同的方面。

当网格图中存在障碍或者走某些边的概率为 0 时，对于每个障碍或者概率为 0 的边主元法需要增加一个主元，当障碍或者概率为 0 的边的数量多于 $O(R)$ 时，主元法的时间复杂度会增加，而直接消元法的时间复杂度仍然不变。

但主元法还可以做类似于网格图的转移方程的消元，例如

$f(i, j) = p_1 f(i + 1, j) + p_2 f(i, j + 1) + p_3 f(\text{pre}(i, j)) + 1$, 其中

$\text{pre}(i, j) = (x, y) (x \leq i, y \leq j)$ 是问题给定的值，而直接消元法的复杂度分析在这种模型中并不适用。

除此以外，网格图邻接矩阵行列式的计算，也不能使用主元法，只能用直接消元法来优化时间复杂度。

综上，两种做法各有所长，需要根据具体题目分析采用不同的做法。

稀疏图

例题 2 Expected Value

给定一张简单无向连通稀疏图 $G = (V, E)$ ，有一枚棋子起始被放在 v_1 ，每秒棋子会从与当前点相连的边中等概率选择一条走到出边指向的点，求到达 v_n 的期望时间。 $n \leq 2000$ ，答案对 p 取模， p 是在区间 $[10^9, 1.01 \times 10^9]$ 内随机生成的一个质数。

基础知识

定义 4.1. 所有满足 $p(A) = 0$ 的多项式 $p(\lambda)$ 称为矩阵 A 的零化多项式。

定义 4.2. 记 I_n 表示 n 阶单位矩阵，定义一个 $n \times n$ 矩阵 A 的特征多项式为 $p(\lambda) = \det(\lambda I_n - A)$ ，其中 \det 表示一个矩阵的行列式。

不难发现，一个 n 阶矩阵 A 的特征多项式的次数不超过 n 。

定理 4.2. (Cayley–Hamilton 定理) 任意矩阵的特征多项式是它的零化多项式。

所以，一个 n 阶矩阵的次数最小的零化多项式的次数也不超过 n 。

求解原问题

注意到，期望走的时间 $E(t) = \sum_{i \geq 0} \Pr[t > i]$ ，如果我们能求出走了 i 步还没有结束的概率，对所有 $i \geq 0$ 求和即为答案。

记 $f(i, j)$ 表示走了 i 步，当前停留在 j ，且没有走到过 n 的概率，那么有：

$$f(i, j) = \sum_{(k,j) \in E} \frac{f(i-1, k)}{\deg_k} (j \neq n)$$

其中 \deg_k 表示 k 的度数。

注意到 f 的转移与 i 无关，可以认为一次转移是乘上了一个矩阵，即 $fi + 1 = f_i M$ 。由于 M 的最小零化多项式次数不超过 n ，所以 f 的最短递推式长度也不超过 n ，故 $\Pr[t > i] = \sum_{j=1}^{n-1} f(i, j)$ 的最短递推式长度也不超过 n 。我们可以在 $O(nm)$ 的时间求出 $\Pr[t > 0], \Pr[t > 1], \dots, \Pr[t > 3n]$ ，然后使用 Berlekamp-Massey 算法，在 $O(n^2)$ 的时间内求解出 $\Pr[t > i]$ 的最短递推式。

考虑求一个 k 阶线性递推序列 a 的生成函数。不妨设 $i \geq i_0$ 时 $a_i = \sum_{j=1}^k c_j a_{i-j}$ ，记 a 和 c 的生成函数为 $A(x)$ 和 $C(x)$ ，那么 $A(x) = A(x)C(x) + A_0(x)$ ，其中 $A_0(x)$ 是由 $i < i_0$ 的项决定的。

回到原问题，由于我们能求出 $\Pr[t > i]$ 的最短递推式，则我们可以求出 $C(x)$ 和 $A_0(x)$ （定义与上一段相同），移项得 $A(x) = \frac{A_0(x)}{1-C(x)}$ 。我们要求的是 $\sum_{i \geq 0} [x^i] A(x)$ ，不难发现这个值就等于 $A(1)$ ，将 $x = 1$ 带入原问题求解即可。由于模数是随机质数，可以认为分母不会为 0。

这样，我们就在 $O(nm + n^2)$ 的时间复杂度内解决了本题。如果图 G 的点数与边数同阶，本题中时间复杂度可以认为是 $O(n^2)$ 。

一般图

例题 3 Frank

给定一张简单强连通有向图 $G = (V, E)$ ，对于所有 $1 \leq s \leq n, 1 \leq t \leq n, s \neq t$ 。回答下面的问题：有一枚棋子起始被放在 v_s ，每秒棋子会从当前点的出边中等概率选择一条走到出边指向的点，求到达 v_t 的期望时间。 $3 \leq n \leq 400$ 。

分析和转化

记 $p_{i,j}$ 表示棋子在 i 时，选择出边 (i, j) 走到 j 的概率，特别地，当出边不存在时概率为 0。记 $f_{i,j}$ 表示 i 随机游走到 j 的期望时间，特别地， $f_{i,i} = 0$ 。当 $i \neq j$ 时，转移方程为：

$$f_{i,j} = 1 + \sum_{1 \leq k \leq n} p_{i,k} f_{k,j}$$

当 $i = j$ 时，记 g_i 表示从 i 开始随机游走，第一次回到 i 的期望时间，那么：

$$f_{i,i} = 1 - g_i + \sum_{1 \leq k \leq n} p_{i,k} f_{k,i}$$

为了方便观察，我们将转移方程写成矩阵的形式。记 P 表示这个图的转移矩阵， F 表示答案矩阵， I 表示 n 阶单位矩阵， J 表示 n 阶全 1 矩阵， G 是一个 n 阶矩阵，满足 $G_{i,i} = g_i$ ，其他位置为 0，则：

$$F = J - G + PF$$

如果我们能求出 G ，那么我们只需要解方程：

$$(I - P)F = J - G$$

G 的求法

定义 5.1. 定义一个 n 阶转移矩阵 P 的稳态分布为一个 n 维向量 π , 满足 $\sum_{i=1}^n \pi_i = 1$, $\pi P = \pi$ 。其中, π 每一维的值都在区间 $[0, 1]$ 内。

我们很容易找出稳态分布的实际意义。如果某个时刻棋子有 π_i 的概率停留在 v_i , 则在之后的任意时刻, 棋子仍然满足这个概率分布。我们可以在 $O(n^3)$ 的时间通过高斯消元解方程来求出 π , 那么 π 与 G 有什么关系呢?

定理 5.1. 对于任意 $1 \leq i \leq n$, 有 $\pi_i g_i = 1$ 。

证明

由 $F = J - G + PF$, 移项得:

$$G = PF + J - F$$

两边同时在左边乘上 π 有:

$$\pi G = \pi PF + \pi J - \pi F$$

由 π 的定义有 $\pi P = \pi$, 故:

$$\pi G = \pi J$$

所以:

$$\pi_i g_i = \sum_{j=1}^n \pi_j = 1$$

原命题得证。

所以, 通过引入稳态分布, 我们可以在 $O(n^3)$ 的时间内求解 G 。

求解原问题

在解方程的过程中, 我们发现一个问题: $(I - P)$ 并不满秩, 不能通过乘逆矩阵的方法求解。

定义 5.2. 定义一个有向图 $G = (V, E)$ 的以 $r \in V$ 为根的有向生成树是 G 的一个子图 $T = (V, A)$, 满足:

1. 对于任意 $i \neq r$, i 的出度为 1。
2. r 的出度为 0。
3. T 中不存在环。

引理 5.1. (有向图上的矩阵树定理) 对于一个有向图 G , 记 D 表示其出度矩阵, 即 $D_{i,i} = d_i$, $D_{i,j} = 0(i \neq j)$, 其中 d_i 表示 i 的出度, 记 A 表示其邻接矩阵, 则其以 r 为根的有向生成树个数为 $D - A$ 去掉第 r 行第 r 列后的行列式。

定理 5.2. 对于一个强连通图 $G = (V, E)$ 的转移矩阵 P , $(I - P)$ 的秩为 $n - 1$ 。

证明

因为对矩阵某一行乘上一个非零常数其秩不改变, 所以我们将 $(I - P)$ 的第 i 行乘上 v_i 的出度, 得到一个新的矩阵 L , 只需证明 L 的秩为 $n - 1$ 即可。

由于 L 每行的和均为 0, 对 L 的所有列向量求和, 会得到零向量, 即这些向量线性相关, 所以 L 的秩不为 n 。

不难发现 L 等于图 G 的出度矩阵减去其邻接矩阵, 由引理 5.1 得 L 去掉第 i 行第 i 列后行列式表示以 v_i 为根的有向生成树个数。

由于 G 是强连通的, 所以以任意点为根的有向生成树个数均不为 0, 即 L 去掉第 i 行第 i 列之后仍然满秩。

因为加上一列秩不会变小，所以 L 去掉第 i 行后所有行向量线性无关。故 L 的秩为 $n - 1$ 。
 回到原问题，考虑求解原问题中的方程。为了方便，我们将方程写成 $AX = B$ 的形式，
 其中 A, B 已知，需要求解 X 。由于 A 不满秩，解有无数个，我们首先求出一组特解。
 将 A 和 B 一起做高斯消元。把 A 的前 $n - 1$ 行消成只有主对角线和第 n 列有值的形式，最后一行消成全
 0，即下列形式：

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & a_1 \\ 0 & 1 & 0 & \cdots & 0 & a_2 \\ 0 & 0 & 1 & \cdots & 0 & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & a_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} X = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & \cdots & b_{1,n-1} & b_{1,n} \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots & b_{2,n-1} & b_{2,n} \\ b_{3,1} & b_{3,2} & b_{3,3} & \cdots & b_{3,n-1} & b_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ b_{n-1,1} & b_{n-1,2} & b_{n-1,3} & \cdots & b_{n-1,n-1} & b_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

令 $X_{n,i} = 0$ ，可以解出一组特解，记为 Y 。接下来将特解调整为真正的解。

注意到 $X_{n,i} = 0$ ，考虑组合意义有 $Y_{i,j} = 1 + Y_{j,j} + P_{i,k}X_{k,j}$ ，不难解出 $X_{i,j} = Y_{i,j} - Y_{j,j}$ 。
 最终在 $O(n^3)$ 的时间复杂度内解决了这个问题。

5、鞅与停时定理

定义

- 随机过程：随机过程 X 为一随机变量族 $\{X_0, X_1, X_2, \dots\}$ ，其中 X_i 为随机变量（代表的是 i 次操作后的状态）。
- 鞅：称随机过程 X 为鞅，如果：
 - $\forall n \geq 0, E(|X_n|) < +\infty$
 - $\forall n \geq 0, E(X_{n+1}|X_0, \dots, X_n) = X_n$

称随机过程 Y 为随机过程 X 的鞅，如果：

- $\forall n \geq 0, E(|Y_n|) < +\infty$
- $\forall n \geq 0, E(Y_{n+1}|X_0, \dots, X_n) = Y_n$

第一个条件即期望有限（这点一般题目都能保证）。

第二个条件意味着已知 $X_0, \dots, X_n, E(X_{n+1}) = E(X_{n+2}) = \dots = X_n$ 。

此外，易知 $E(X_n) = E(X_0)$ 。

- 随机时刻：设随机变量 $T \in N \cup \{+\infty\}$ 与随机过程 X ，若事件 $T = n$ 的示性函数 $I_{T=n}$ 为关于 X_0, \dots, X_n 的函数（即仅由这些随机变量确定），则称 T 为 X 的随机时刻。
- 停时：若 T 为 X 的随机时刻，且 $P(T < +\infty) = 1$ ，则称 T 为 X 的停时。
- 停止过程：若 T 为 X 的随机时刻，则定义 X 的停止过程 \bar{X} 为：

$$\bar{X}_n = \begin{cases} X_n, & n \leq T \\ X_T, & n > T \end{cases}$$

- 性质：若 X 为鞅，则 \bar{X} 为关于 X 的鞅。
- 鞅的停时定理：若随机过程 X 与随机时刻 T 满足以下条件之一：
 - \bar{X} 一致有界（即每一个 \bar{X}_n 均有界）。
 - T 有界。
 - $E(T) < +\infty$ ，且存在 $M < \infty$ 使 $E(|X_{n+1} - X_n| | X_0, \dots, X_n) < M$ 。

则 $E(X_T) = E(X_0)$ 。

一般而言题目都能满足条件。

解题方法

目前遇到的题都比较套路。

1. 设计状态（一般而言状态就是题目中的状态）。
2. 尝试构造势函数 $\phi(X)$ ，使得 $E(\phi(Xn+1)|X0, \dots, Xn) = \phi(Xn) + 1$ 。这样一来 $\{\phi(Xn) - n\}$ 即为一个鞅，由停时定理得到 $E(\phi(XT)) - E(T) = E(\phi(X0))$ ，即可计算 $E(T) = E(\phi(XT)) - E(\phi(X0))$ 。
3. 解 $E(\phi(Xn+1)|X0, \dots, Xn) = \phi(Xn) + 1$ 方程并构造势函数。（主要步骤）

问题 1 (CF1025G Company Acquisitions)

- 现有 n 个人组成若干组，每组有且仅有一个组长。
- 接下来的每一时刻，随机有序选取两个不同的组长 A, B ，让 A 加入 B 的组内并将 A 原来的组解散（每个人各自组建一个新的小组）。
- 求第一次只剩下 1 个组的期望时间。

设大小为 n 的组的势函数为 $f(n)$ ，一个局面的势函数 $\phi(X_t) = \sum f(a_{t,i})$ ，其中 $a_{t,i}$ 表示的是 X_t 中每个组的大小。

由于这道题组数会变，考虑每次只计算变化的。设无序选出的两个组长所在组的人数分别为 a, b 。

$$-f(a) - f(b) + \frac{1}{2}[f(b+1) + (a-1)f(1)] + \frac{1}{2}[f(a+1) + (b-1)f(1)] = 1$$

$$[(a-1)f(1) + \frac{1}{2}f(a+1) - f(a)] + [(b-1)f(1) + \frac{1}{2}f(b+1) - f(b)] = 1$$

$$\text{令 } (n-1)f(1) + \frac{1}{2}f(n+1) - f(n) = \frac{1}{2}, \text{ 即 } f(n) = 2f(n-1) - 2(n-1)f(1) + 1.$$

$$\text{令 } f(1) = 0, \text{ 即有 } f(n) = 2f(n-1) + 1 = 2^{n-1} - 1.$$

答案即为 $E(T) = E(\phi(X_T)) - E(\phi(X_0))$ ，时间复杂度 $O(n \log m)$ 。

问题 2 (CF1349D Slime and Biscuits)

- 现有 n 个人，第 i 个人有 a_i 个饼干。
- 每一时刻，随机选择一个饼干，将其随机分配给除了它现在所有者的其他 $n-1$ 个人之一。
- 求第一次出现一个人拥有所有饼干的期望时间。

$$\text{设 } m = \sum a_i.$$

$$\phi(X_{t+1}) = \sum_{i=1}^n \sum_{j \neq i} \frac{a_{t,i}}{m} \frac{1}{n-1} [f(a_{t,i}-1) + f(a_{t,j}+1) + \sum_{k \neq i,j} f(a_{t,k})]$$

$$= \sum_{i=1}^n \left[\frac{a_{t,i}}{m} f(a_{t,i}-1) + \frac{m-a_{t,i}}{m} \frac{1}{n-1} f(a_{t,j}+1) + \frac{m-a_{t,i}}{m} \frac{n-2}{n-1} f(a_{t,i}) \right]$$

令 $\phi(X_{t+1}) = \phi(X_t) + 1$ ，即 $\phi(X_t) = \phi(X_{t+1}) - 1$ 。这里把 1 拆成 $n \times \frac{1}{n}$ ，拆成 $\sum \frac{a_i}{m}$ 也可以（之后会这么拆）。

$$\sum_{i=1}^n f(a_{t,i}) = \sum_{i=1}^n \left[\frac{a_{t,i}}{m} f(a_{t,i}-1) + \frac{m-a_{t,i}}{m} \frac{1}{n-1} f(a_{t,j}+1) + \frac{m-a_{t,i}}{m} \frac{n-2}{n-1} f(a_{t,i}) \right] - 1$$

$$a_{t,i} = \frac{a_{t,i}}{m} f(a_{t,i}-1) + \frac{m-a_{t,i}}{m} \frac{1}{n-1} f(a_{t,j}+1) + \frac{m-a_{t,i}}{m} \frac{n-2}{n-1} f(a_{t,i}) - \frac{1}{n}$$

移项得：

$$f(a+1) = (n-1) \frac{m}{m-x} (f(x) - f(x-1)) - (n-2) f(x) + \frac{m}{m-x} \frac{n-1}{n}$$

线性递推即可，时间复杂度 $O(m)$ 。

6、Min_25理解

定义

从此种筛法的思想方法来说，其又被称为「Extended Eratosthenes Sieve」。

由于其由 [Min_25](#) 发明并最早开始使用，故称「Min_25 筛」。

性质

其可以在 $O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$ 或 $\Theta(n^{1-\epsilon})$ 的时间复杂度下解决一类 **积性函数** 的前缀和问题。

要求： $f(p)$ 是一个关于 p 可以快速求值的完全积性函数之和（例如多项式）； $f(p^c)$ 可以快速求值。

记号

- 如无特别说明，本节中所有记为 p 的变量的取值集合均为全体质数。
- $x/y := \left\lfloor \frac{x}{y} \right\rfloor$
- $\text{isprime}(n) := [\{d : d | n\}] = 2]$ ，即 n 为质数时其值为 1，否则为 0。
- p_k : 全体质数中第 k 小的质数（如： $p_1 = 2, p_2 = 3$ ）。特别地，令 $p_0 = 1$ 。
- $\text{lpf}(n) := [1 < n] \min\{p : p | n\} + [1 = n]$ ，即 n 的最小质因数。特别地， $n = 1$ 时，其值为 1
- $F_{\text{prime}}(n) := \sum_{2 \leq p \leq n} f(p)$
- $F_k(n) := \sum_{i=2}^n [p_k \leq \text{lpf}(i)] f(i)$

解释

观察 $F_k(n)$ 的定义，可以发现答案即为 $F_1(n) + f(1) = F_1(n) + 1$ 。

考虑如何求出 $F_k(n)$ 。通过枚举每个 i 的最小质因子及其次数可以得到递推式：

$$\begin{aligned} F_k(n) &= \sum_{i=2}^n [p_k \leq \text{lpf}(i)] f(i) \\ &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^c \leq n}} f(p_i^c) ([c > 1] + F_{i+1}(n/p_i^c)) + \sum_{\substack{k \leq i \\ p_i \leq n}} f(p_i) \\ &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^c \leq n}} f(p_i^c) ([c > 1] + F_{i+1}(n/p_i^c)) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) \\ &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^{c+1} \leq n}} (f(p_i^c) F_{i+1}(n/p_i^c) + f(p_i^{c+1})) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) \end{aligned}$$

最后一步推导基于这样一个事实：对于满足 $p_i^c \leq n < p_i^{c+1}$ 的 c ，有

$p_i^{c+1} > n \iff n/p_i^c < p_i < p_{i+1}$ ，故 $F_{i+1}(n/p_i^c) = 0$ 。

其边界值即为 $F_k(n) = 0 (p_k > n)$ 。

假设现在已经求出了所有的 $F_{\text{prime}}(n)$ ，那么有两种方式可以求出所有的 $F_k(n)$ ：

- 直接按照递推式计算。
- 从大到小枚举 p 转移，仅当 $p^2 < n$ 时转移增加值不为零，故按照递推式后缀和优化即可。

现在考虑如何计算 $F_{\text{prime}}(n)$ 。

观察求 $F_k(n)$ 的过程，容易发现 F_{prime} 有且仅有 $1, 2, \dots, \lfloor \sqrt{n} \rfloor, n/\sqrt{n}, \dots, n/2, n$ 这 $O(\sqrt{n})$ 处的点值是有用的。

一般情况下， $f(p)$ 是一个关于 p 的低次多项式，可以表示为 $f(p) = \sum a_i p^{c_i}$ 。

那么对于每个 p^{c_i} ，其对 $F_{\text{prime}}(n)$ 的贡献即为 $a_i \sum_{2 \leq p \leq n} p^{c_i}$ 。

分开考虑每个 p^{c_i} 的贡献，问题就转变为了：给定 $n, s, g(p) = p^s$ ，对所有的 $m = n/i$ ，求 $\sum_{p \leq m} g(p)$

。

Notice: $g(p) = p^s$ 是完全积性函数！

于是设 $G_k(n) := \sum_{i=2}^n [p_k < \text{lpf}(i) \vee \text{isprime}(i)]g(i)$ ，即埃筛第 k 轮筛完后剩下的数的 g 值之和。

对于一个合数 $x \leq n$ ，必定有 $\text{lpf}(x) \leq \sqrt{x} \leq \sqrt{n}$ 。设 $p_{\ell(n)}$ 为不大于 \sqrt{n} 的最大质数，则

$F_{\text{prime}}(n) = G_{\ell(n)}(n)$ ，即在埃筛进行 ℓ 轮之后剩下的均为质数。

考虑 G 的边界值，显然为 $G_0(n) = \sum_{i=2}^n g(i)$ 。（还记得吗？特别约定了 $p_0 = 1$ ）

对于转移，考虑埃筛的过程，分开讨论每部分的贡献，有：

1. 对于 $n < p_k^2$ 的部分， G 值不变，即 $G_k(n) = G_{k-1}(n)$ 。
2. 对于 $p_k^2 \leq n$ 的部分，被筛掉的数必有质因子 p_k ，即 $-g(p_k)G_{k-1}(n/p_k)$ 。
3. 对于第二部分，由于 $p_k^2 \leq n \iff p_k \leq n/p_k$ ，满足 $\text{lpf}(i) < p_k$ 的 i 会被额外减去。这部分应当加回来，即 $g(p_k)G_{k-1}(p_{k-1})$ 。

则有：

$$G_k(n) = G_{k-1}(n) - [p_k^2 \leq n]g(p_k)(G_{k-1}(n/p_k) - G_{k-1}(p_{k-1}))$$

复杂度分析

对于 $F_k(n)$ 的计算，其第一种方法的时间复杂度被证明为 $O(n^{1-\epsilon})$ （见 zzt 集训队论文 2.3）；

对于第二种方法，其本质即为洲阁筛的第二部分，在洲阁论文中也有提及（6.5.4），其时间复杂度被证明为 $O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$ 。

对于 $F_{\text{prime}}(n)$ 的计算，事实上，其实现与洲阁筛第一部分是相同的。

考虑对于每个 $m = n/i$ ，只有在枚举满足 $p_k^2 \leq m$ 的 p_k 转移时会对时间复杂度产生贡献，则时间复杂度可估计为：

$$\begin{aligned} T(n) &= \sum_{i^2 \leq n} O\left(\pi\left(\sqrt{i}\right)\right) + \sum_{i^2 \leq n} O\left(\pi\left(\sqrt{\frac{n}{i}}\right)\right) \\ &= \sum_{i^2 \leq n} O\left(\frac{\sqrt{i}}{\ln \sqrt{i}}\right) + \sum_{i^2 \leq n} O\left(\frac{\sqrt{\frac{n}{i}}}{\ln \sqrt{\frac{n}{i}}}\right) \\ &= O\left(\int_1^{\sqrt{n}} \frac{\sqrt{\frac{n}{x}}}{\log \sqrt{\frac{n}{x}}} dx\right) \\ &= O\left(\frac{n^{\frac{3}{4}}}{\log n}\right) \end{aligned}$$

对于空间复杂度，可以发现不论是 F_k 还是 F_{prime} ，其均只在 n/i 处取有效点值，共 $O(\sqrt{n})$ 个，仅记录有效值即可将空间复杂度优化至 $O(\sqrt{n})$ 。

首先，通过一次数论分块可以得到所有的有效值，用一个大小为 $O(\sqrt{n})$ 的数组 lis 记录。对于有效值 v ，记 $\text{id}(v)$ 为 v 在 lis 中的下标，易得：对于所有有效值 v ， $\text{id}(v) \leq \sqrt{n}$ 。

然后分开考虑小于等于 \sqrt{n} 的有效值和大于 \sqrt{n} 的有效值：对于小于等于 \sqrt{n} 的有效值 v ，用一个数组 le 记录其 $\text{id}(v)$ ，即 $\text{le}_v = \text{id}(v)$ ；对于大于 \sqrt{n} 的有效值 v ，用一个数组 ge 记录 $\text{id}(v)$ ，由于 v 过大所以借助 $v' = n/v < \sqrt{n}$ 记录 $\text{id}(v)$ ，即 $\text{ge}_{v'} = \text{id}(v)$ 。

这样，就可以使用两个大小为 $O(\sqrt{n})$ 的数组记录所有有效值的 id 并 $O(1)$ 查询。在计算 F_k 或 F_{prime} 时，使用有效值的 id 代替有效值作为下标，即可将空间复杂度优化至 $O(\sqrt{n})$ 。

过程

对于 $F_k(n)$ 的计算，我们实现时一般选择实现难度较低的第一种方法，其在数据规模较小时往往比第二种方法的表现要好；

对于 $F_{\text{prime}}(n)$ 的计算，直接按递推式实现即可。

对于 $p_k^2 \leq n$ ，可以用线性筛预处理出 $s_k := F_{\text{prime}}(p_k)$ 来替代 F_k 递推式中的 $F_{\text{prime}}(p_{k-1})$ 。相应地， G 递推式中的 $G_{k-1}(p_{k-1}) = \sum_{i=1}^{k-1} g(p_i)$ 也可以用此方法预处理。

用 Extended Eratosthenes Sieve 求 **积性函数** f 的前缀和时，应当明确以下几点：

- 如何快速（一般是线性时间复杂度）筛出前 \sqrt{n} 个 f 值；
- $f(p)$ 的多项式表示；
- 如何快速求出 $f(p^c)$ 。

明确上述几点之后按顺序实现以下几部分即可：

1. 筛出 $[1, \sqrt{n}]$ 内的质数与前 \sqrt{n} 个 f 值；
2. 对 $f(p)$ 多项式表示中的每一项筛出对应的 G ，合并得到 F_{prime} 的所有 $O(\sqrt{n})$ 个有用点值；
3. 按照 F_k 的递推式实现递归，求出 $F_1(n)$ 。

例题

求莫比乌斯函数的前缀和

求 $\sum_{i=1}^n \mu(i)$ 。

易知 $f(p) = -1$ 。则 $g(p) = -1, G_0(n) = \sum_{i=2}^n g(i) = -n + 1$ 。
直接筛即可得到 F_{prime} 的所有 $O(\sqrt{n})$ 个所需点值。

求欧拉函数的前缀和

求 $\sum_{i=1}^n \varphi(i)$ 。

首先易知 $f(p) = p - 1$ 。
对于 $f(p)$ 的一次项 (p)，有 $g(p) = p, G_0(n) = \sum_{i=2}^n g(i) = \frac{(n+2)(n-1)}{2}$ ；
对于 $f(p)$ 的常数项 (-1)，有 $g(p) = -1, G_0(n) = \sum_{i=2}^n g(i) = -n + 1$ 。
筛两次加起来即可得到 F_{prime} 的所有 $O(\sqrt{n})$ 个所需点值。

「LOJ #6053」简单的函数

给定 $f(n)$ ：

$$f(n) = \begin{cases} 1 & n = 1 \\ p \text{ xor } c & n = p^c \\ f(a)f(b) & n = ab \wedge a \perp b \end{cases}$$

易知 $f(p) = p - 1 + 2[p = 2]$ 。则按照筛 φ 的方法筛，对 2 讨论一下即可。