Algorithm Library

lonlyn

September 4, 2021

Contents

1 基础类

1.1 常用函数与类模板

1.1.1 二分查找

lower_bound() 第一个**大于等于**值的元素的位置

upper_bound() 第一个**大于**值的元素的位置

1.1.2 二进制

___builtin_ffs(x) 返回 x 的最后一位 1 是从后往前第几位

___builtin_clz(x) 返回 x 的二进制下前导 0 个数

___builtin_ctz(x) 返回 x 的二进制下末尾 0 个数

___builtin_popcount(x) 返回 x 的二进制下 1 的个数

___builtin_parity(x) 返回 x 的二进制下 1 的个数的奇偶性

1.2 快速输入，输出

注：I_int 后面的元素根据实际更改即可

```
#define I_int int
inline I_int read(){
  I_int x=0,f=1; char c=getchar();
  while(c<'0'||c>'9') {if(c=='-') f=-1; c=getchar();}
  while(c>='0'&&c<='9') {x=x*10+c-'0'; c=getchar();}
  return x*f;
}
char F[200];
inline void write(I_int x){
  I_int tmp=x>0?x:-x;
  if(x<0) putchar('-');
  int cnt=0;
  while(tmp>0){
     F[cnt++]=tmp%10+'0';
     tmp/=10;
  }
    while(cnt>0) putchar(F[--cnt]);
}
#undef I_int
```

1.3 string

getline(cin,str) 读取一行并存储在 str。(需要引用 string 库)
begin(),end() 起始/末尾的迭代器
empty(),size(),length() 是否为空及长度
insert(pos,count,ch) 在 pos 插入 count 个 ch
insert(pos,str) 在 pos 插入 str
erase(first,last) 删除 [first,last) 的字符串
push_back(),pop_back() 后附/移除末尾字符
append(str) 末尾添加字符串

1.4 高精度

```
struct Wint:vector<int>{
    Wint(int n=0){
        //if the num is too large then use long long instead of int
        //if still small,try just cin
        push_back(n); check();
    }
    Wint& check(){ //turn to a normal num
        while(!empty()&&!back())pop_back();//delete leading zero
        if(empty())return *this;
        for(int i=1; i<size(); ++i){
            (*this)[i]+=(*this)[i-1]/10; (*this)[i-1]%=10;
        }
        while(back()>=10){
            push_back(back()/10);
            (*this)[size()-2]%=10;
        }
        return *this;
    }
};
istream& operator>>(istream &is,Wint &n){ //cin
    string s;
    is>>s;
    n.clear();
    for(int i=s.size()-1; i>=0; --i)n.push_back(s[i]-'0');
    return is;
}
ostream& operator<<(ostream &os,const Wint &n){ //cout
    if(n.empty())os<<0;
    for(int i=n.size()-1; i>=0; --i)os<<n[i];
    return os;
}
```

```
32 bool operator!=(const Wint &a,const Wint &b){
33     if(a.size()!=b.size())return 1;
34     for(int i=a.size()-1; i>=0; --i)
35         if(a[i]!=b[i])return 1;
36     return 0;
37 }
38 bool operator==(const Wint &a,const Wint &b){
39     return !(a!=b);
40 }
41 bool operator<(const Wint &a,const Wint &b){
42     if(a.size()!=b.size())return a.size()<b.size();
43     for(int i=a.size()-1; i>=0; --i)
44         if(a[i]!=b[i])return a[i]<b[i];
45     return 0;
46 }
47 bool operator>(const Wint &a,const Wint &b){
48     return b<a;
49 }
50 bool operator<=(const Wint &a,const Wint &b){
51     return !(a>b);
52 }
53 bool operator>=(const Wint &a,const Wint &b){
54     return !(a<b);
55 }
56 Wint& operator+=(Wint &a,const Wint &b){
57     if(a.size()<b.size())a.resize(b.size());
58     for(int i=0; i!=b.size(); ++i)a[i]+=b[i];
59     return a.check();
60 }
61 Wint operator+(Wint a,const Wint &b){
62     return a+=b;
63 }
64 Wint& operator-=(Wint &a,Wint b){
65     if(a<b)swap(a,b);
66     for(int i=0; i!=b.size(); a[i]-=b[i],++i)
67         if(a[i]<b[i]){
68             int j=i+1;
69             while(!a[j]) ++j;
70             while(j>i){
71                 --a[j]; a[--j]+=10;
72             }
73         }
74     return a.check();
75 }
```

```
76  Wint operator-(Wint a,const Wint &b){
77      return a-=b;
78  }
79  Wint operator*(const Wint &a,const Wint &b){
80      Wint n;
81      n.assign(a.size()+b.size()-1,0);
82      for(int i=0; i!=a.size(); ++i)
83          for(int j=0; j!=b.size(); ++j)
84              n[i+j]+=a[i]*b[j];
85      return n.check();
86  }
87  Wint& operator*=(Wint &a,const Wint &b){
88      return a=a*b;
89  }
90  Wint divmod(Wint &a,const Wint &b){
91      Wint ans;
92      for(int t=a.size()-b.size(); a>=b; --t){
93          Wint d;
94          d.assign(t+1,0); d.back()=1;
95          Wint c=b*d;
96          while(a>=c){
97              a-=c; ans+=d;
98          }
99      }
100     return ans;
101 }
102 Wint operator/(Wint a,const Wint &b){
103     return divmod(a,b);
104 }
105 Wint& operator/=(Wint &a,const Wint &b){
106     return a=a/b;
107 }
108 Wint& operator%=(Wint &a,const Wint &b){
109     divmod(a,b); return a;
110 }
111 Wint operator%(Wint a,const Wint &b){
112     return a%=b;
113 }
114 Wint ksm(const Wint &n,const Wint &k){
115     if(k.empty())return 1;
116     if(k==2)return n*n;
117     if(k.back()%2)return n*ksm(n,k-1);
118     return ksm(ksm(n,k/2),2);
119 }
```

1.5 离散化

```
// int a[],b[]
// a[1..n]->b[1..n]
sort(a+1,a+1+n);
len=unique(a+1,a+n+1)-a-1;
b[i]=lower_bound(a+1,a+len+1,x)-a;
// vector<int> a,b
sort(a.begin(),a.end());
a.erase(unique(a.begin(),a.end()),a.end());
for (int i=0;i<n;++i){
    b[i] = std::lower_bound(a.begin(), a.end(), b[i]) - a.begin();
}
```

1.6 C++ STL

1.6.1 序列式容器：vector,deque,list

(1)vector：

构造:

vector<int> a;

vector<int> a(size[,value]); 创建有 size 个元素的 vector 并把 value 作为初值（默认 0)。

访问与修改:

注: 所有增减元素在中间的线性复杂度是根据操作位置与**最后元素**的距离而言的。

a.assign(size,const T& value) 将 size 份 value 覆盖到 a 里。$O(n)$

a.begin() a.end() 返回起始地址和**末元素后一位**的元素的地址。

a.clear() 将 a 中元素全部擦除。$O(n)$

a.insert(iterator pos,const T&value) 在 pos 前插入 value。$O(n)$

a.erase(iterator pos) 删除 pos 处元素。$O(n)$

a.erase(iterator first,iterator last) 删除 [first;last) 中的元素。$O(n)$

(2)deque：

构造:

deque<int> a;

deque<int> a(size[,value]); 创建有 size 个元素的 deque 并把 value 作为初值（默认 0)。

访问与修改:

注: 所有增减元素在中间的线性复杂度是根据操作位置与**两端最近的**距离而言的。

a.front(); a.back(); 返回头/尾元素的值。

clear insert erase begin end 同上

a.pushfront(const T& value); a.pushback(const T& value); 头部与末尾插入元素。

a.popfront(); a.popback(const T& value); 头部与末尾弹出元素。

(3)list：

构造：

list<int> a;

list<int> a(size[,value]); 创建有 size 个元素的 list 并把 value 作为初值（默认 0）。

访问与修改：

a.front(); a.back(); 返回头/尾元素的值。

a.begin() a.end() 返回起始地址和**末元素后一位**的元素的地址。

a.clear() 将 a 中元素全部擦除。复杂度 $O(n)$

a.insert(iterator pos,const T&value) 在 pos 前插入 value。$O(1)$

a.erase(iterator pos) 删除 pos 处元素。$O(1)$

a.erase(iterator first,iterator last) 删除 [first;last) 中的元素。

a.pushfront(const T& value); a.pushback(const T& value); 头部与末尾插入元素。

a.popfront(); a.popback(const T& value); 头部与末尾弹出元素。

a.sort([comp]); 排序，comp 为 **bool comp(const Type &_a,const Type &b){}**$O(n \log n)$

a.reverse(); 翻转顺序。$O(n)$

a.unique(); 去除所有**相继**的重复元素。$O(n)$

a.merge(list b[,comp]); 合并**有序列表** a,b（归并排序）。$O(n)$

1.6.2 关联式容器：(multi)set,(multi)map

(1)set：

构造：

set<Type> a;

set<Type,TypeComp> a;

注：这里的 TypeComp 需为 struct 而不是 operator。如：

```
// example of map
struct node {
    int x, y;
} cur;
struct cmp {
    bool operator () (const node &a, const node &b) const {
        return (a.x == b.x) ? a.y < b.y : a.x < b.x;
    }
};
map<node, int, cmp> mp;
```

访问与修改：

a.clear() 清空元素。$O(n)$

a.insert(value) 插入元素。$O(\log n)$

a.erase(iterator pos) 清除位于 pos 的元素。$O(1)$

a.erase(const key_type& key); 清除关键等于 key 的元素。$O(\log n)$

a.count(value) 返回是否存在该元素（0/1）。$O(\log n)$

a.find(value) 返回值为 value 的元素的**迭代器**，没有则为 a.end()。$O(\log n)$

a.lower_bound(value) 返回指向首个**不小于** value 的迭代器。$O(\log n)$

a.upper_bound(value) 返回指向首个**大于** value 的迭代器。$O(\log n)$

(2)multiset：

构造：

与 set 相似。略。

访问与修改：（与 set 功能相同的已经省略）

a.erase(const key_type& key); 清除关键字等于 key 的**所有**元素。$O(\log n + a.count(key))$

a.count(value) 返回键值为 key 的元素有多少个。$O(\log n)$

(3)map：

构造：

map<Type1 a,Type2 b> mp;

map<Type1 a,Type2 b,Type1Comp> mp;

这里的 Type1Comp 是针对 Type1 的，且需为 struct 而不是 operator。

访问与修改：

mp.clear() 清除。$O(n)$

mp.insert(pair<Type1,Type2>(a,b)); 添加元素，不过可以直接 mp[Type1]=Type2。$O(\log n)$

如果重复插入相同键值 (Type1) 则**后序插入的被忽略**。

mp.insertorassign(pair<Type1,Type2>(a,b)); 添加元素，$O(\log n)$

如果重复插入相同键值 (Type1) 则**后序插入值覆盖原来的值**。

mp.erase(iterator pos) 清除位于 pos 的元素。$O(1)$

mp.erase(iterator first,iterator last); 清除 [first;last) 的元素。

mp.erase(const key_type& key); 清除关键等于 key 的元素。$O(\log n)$

mp.count(const Key& key); 返回键值为 key 的元素个数 (0/1)。$O(\log n)$

mp.find(const Key& key); 返回键值为 key 的元素的 iterator。否则 mp.end()。$O(\log n)$

mp.lower_bound(const Key& key); 返回首个**不小于** k 的键值所在元素的 iterator。

mp.upper_bound(const Key& key); 同上，不过是**大于**。两者 $O(\log n)$

(4)multimap：

因为 multimap 允许多个元素拥有同一键的特点，multimap 并**没有提供给出键访问其对应值的方法**。

构造：

同 map 构造。

访问及修改（不同的）：

mp.erase(const key_type& key); 清除关键等于 key 的**所有**元素。$O(\log n)$

mp.count(const Key& key); 返回键值为 key 的元素个数。$O(\log n)$

1.6.3 无序关联式容器：unrodered_(multi)set,map

所有元素哈希处理。但遇到 cf 赛事时注意防卡 hash。

(1)unordered_set：

构造：

unordered_set<Type> st;

unordered_set<Type,Hash,Keyequal> st;

Type 表示类型，Hash 表示哈希方法，Keyequal 表示判定键值相同方法。后两者用 struct 判断。

如果 Type 不是自定义类型的话就不用写 Keyequal，或者写上更难 hack。

下面给出一个 hash 与 equal 栗子（摘自 OI-Wiki）

```
struct Node{
    int x,y;
}
struct NodeHash {
  static uint64_t splitmix64(uint64_t x) { //很复杂的式子
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(Type _a) const {
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().time_since_epoch().count();
    //加入未确定时间因素更难 hack
    return splitmix64(_a.x*233+_a.y + FIXED_RANDOM);
  }
};
struct NodeEqual{
    bool operator () {const Node& _a,const Node& _b} const {
        return _a.x==_b.x&&_a.y==_b.y;
    }
}
```

(2)unordered_multiset：

同上，不多赘述。

(3)unordered_map：

构造：

unordered_map<Type1,Type2> mp;

unordered_map<Type1,Type2,Type1Hash,Type1Equal>;

Type1Hash 与 Type1Equal 规则和上文一样。不再赘述。

(4)unordered_multiset：

同上，不多赘述。

2 数论

2.1 数论相关定理和规律

- 费马小定理：当 $a,p \in Z$ 且 p 为质数，且 $a \not\equiv 0 \ (mod\ p)$ ，有 $a^{p-1} \equiv 1 \ (mod\ p)$ 。所以 $a^b \equiv a^{b\ mod\ (p-1)} \ (mod\ p)$ 。
- 欧拉定理：当 $a,m \in Z$ ，且 $gcd(a,m)=1$ 时，有 $a^{\varphi(m)} \equiv 1 \ (mod\ m)$ 。所以 $a^b \equiv a^{b\ mod\ \varphi(m)} \ (mod\ m)$ 。
- 扩展欧拉定理：不需要互质啦：

$$a^c \equiv \begin{cases} a^{c\ mod\ \varphi(m)} & ,gcd(a,m)=1 \\ a^c & ,gcd(a,m)\neq 1, c<\varphi(m) \\ a^{(c\ mod\ \varphi(m))+\varphi(m)} & ,gcd(a,m)\neq 1, c\geq \varphi(m) \end{cases} \tag{1}$$

- 二项式定理：$(x+y)^n = \sum_{k=0}^{n} C_n^k x^{n-k} y^k$
- 第二类斯特林数 $S(n,k)$：将 n 个互不相同的元素分成 k 个互不区分非空子集的方案数。如果区分先后子集，注意乘上阶乘。

  $S(n,0) = [n=0]$

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

  $S(n,k) = \sum_{i=0}^{m} \frac{(-1)^{m-i} i^n}{i!(m-i)!}$
- 错排公式：
  - 递推公式：$D(n) = (n-1)[D(n-2)+D(n-1)]$，特殊的，$d[1]=0, d[2]=1$。关于 $d[0]$ 的取值：如果参与乘法运算考虑是否更改为 1，正常情况为 0。
  - 整体公式：$D(n) = n![\frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} + ... + \frac{(-1)^n}{n!}]$
  - 简化公式：$D(n) = \lfloor \frac{n!}{e} + 0.5 \rfloor$
- 卢卡斯定理：$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$
- $\sum_{d|n} \mu(d) = [n=1], \mu * I = \epsilon, \sum_{d|n} \varphi(d) = n, \varphi * I = id$
- 同余问题难以解决时，考虑移项。遇到倍数问题，尝试同除 gcd 得到互质，从而消去（乘互质的数对倍数无贡献）。
- $(a\ and\ b) * 2 = (a\ xor\ b) + (a+b)$

- 二阶线性递推数列:$x_n = a_1x_{n-1}+a_2x_{n-2}$ 的特征方程是 $x^2-a_1x-a_2 = 0$ 。求得两个解后，通项为 $x_n = c_1{x_1}^n + c_2{x_2}^n$ ，c_1, c_2 通过带入求解。例如斐波那契数列 $x_n = x_{n-1} + x_{n-2}$ ，特征方程 $x^2 - x - 1 = 0$ ，求出两个解，同时带入特殊值即可求出通项：$\frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n]$
- n 个数快速因式分解：如果在值域较小的情况下，预处理 d[i] 数组代表 i 的最小质因子，省去挪指针的时间。例求解 35 的因子，d[35]=5，d[35/5]=d[7]=[7]，便得到了 35 包含的质因子 3,7。
- 低阶等幂求和公式:
  1. $\sum_{i=1}^{n} i^1 = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$
  2. $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$
  3. $\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2 = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$
  4. $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$
  5. $\sum_{i=1}^{n} i^5 = \frac{n^2(n+1)^2(2n+1)(2n^2+2n-1)}{12} = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$

2.2 线性筛

```cpp
//phi and prime
int prime[maxn],phi[maxn];
int notprime[maxn];
int tot;
void init(){
    notprime[1]=1;
    for (int i=2;i<=maxn;++i){
        if (!notprime[i]){
            prime[++tot]=i;
            phi[i]=i-1;
        }
        for (int j=1;j<=tot&&i*prime[j]<=maxn;++j){
            notprime[i*prime[j]]=1;
            if (!(i%prime[j])){
                phi[i*prime[j]]=phi[i]*prime[j];
                break;
            }else{
                phi[i*prime[j]]=phi[i]*(prime[j]-1);
            }
        }
```

```
    }
}
/*
calulate how many divisors are there in n
example:d[6]=4
*/
int notprime[maxn];
int prime[maxn],d[maxn],num[maxn];
//num[i]:the minest prime divisor
int tot;
void init(){
    cnt=0;
    d[1]=1;
    for (int i=2;i<=maxn;++i){
        if (!notprime[i]){
            prime[++tot]=i; num[i]=1;
            d[i]=2;
        }
        for (int j=1;j<=tot&&i*prim[j]<=maxn;++j){
            notprime[i*prime[j]]=1;
            if (!(i%prim[j])){
                num[i*prime[j]]=num[i]+1;
                d[i*prime[j]]=d[i]/(num[i]+1)*(num[i*prime[j]]+1);
                break;
            }
            d[i*prime[j]]=d[i]*d[prime[j]];
            num[i*prime[j]]=1;
        }
    }
}
/*
calculate the sum of divisor of n
example:sd[6]=1+2+3+6=12
*/
int notprime[maxn];
int prime[maxn];
long long sd[N],sp[N];
int tot;
void init(){
    cnt=0; sd[1]=1;
    for (int i=2;i<=maxn;++i){
        if (!notprime[i]){
            prim[++tot]=i; sd[i]=i+1; sp[i]=i+1;
        }
```

```
        for (int j=0;j<cnt&&i*prime[j]<=maxn;++j){
            mark[i*prime[j]]=1;
            if (!(i%prime[j])){
                sp[i*prime[j]]=sp[i]*prime[j]+1;
                sd[i*prime[j]]=sd[i]/sp[i]*sp[i*prime[j]];
                break;
            }
            sd[i*prime[j]]=sd[i]*sd[prime[j]];
            sp[i*prime[j]]=1+prime[j];
        }
    }
}
```

2.3 逆元

2.3.1 单个逆元

$$a^{p-2} \equiv \frac{1}{a}(mod\ p)$$

2.3.2 线性阶乘逆元

考虑到 $\frac{1}{(n+1)!} \times (n+1) = \frac{1}{n!}$，求出最大的往下递推即可。

2.3.3 线性逆元

```
int inv[maxn];
inv[1]=1;
for (int i=2;i<p;++i){
    inv[i]=(p-p/i)*inv[p%i]%p;
}
```

2.3.4 任意 n 个数的逆元

```
int s[maxn],sv[maxn];
/*
s[]: prefix product
sv[]: inverse of s[]
*/
int inv[maxn];
s[0]=1;
for (int i=1;i<=n;++i) s[i]=s[i-1]*a[i]%p;
sv[n]=ksm(s[n],p-2);
for (int i=n;i>=1;--i) inv[i-1]=inv[i]*a[i]%p;
for (int i=1;i<=n;++i) inv[i]=sv[i]*s[i-1]%p;
```

2.4 Exgcd 与线性同余方程

2.4.1 Exgcd

返回一组满足 $ax + by = c$ 的 (x,y)

```
#define ll long long
ll exgcd(ll a,ll b,ll &x,ll &y){ //ax+by=c->ax=c(mod b)
    if (!b){
        x=1; y=0; return a;
    }
    ll d=exgcd(b,a%b,x,y);
    ll tmp=x; x=y; y=tmp-a/b*y;
    return d;
}
```

2.4.2 线性同余方程

返回 x 满足 $ax \equiv c(mod\ b)$，转化为 $ax + by = c$ 解决。有解需要满足 $gcd(a, b)|c$

```
ll liner(ll a,ll b,ll c){ //ax=c(mod b)
    ll x,y;
    ll d=exgcd(a,b,x,y);
    if (c%d) return -1; //there is no answer
    x*=c/d; ll t=b/d;
    return (x%t+t)%t; //the minest positive answer
}
```

2.5 线性同余方程组及中国剩余定理

2.5.1 CRT

求得 $x \equiv a_i(mod\ m_i)$ 需要**保证模数** $m_1, m_2, \cdots, m_n$ **两两互质**。

```
ll a[maxn],m[maxn];
ll crt(){ //x=a[](mod m[])
    ll ans=0,M=1;
    for (int i=1;i<=n;++i) M*=m[i];
    for (int i=1;i<=n;++i){
        ll x,y,Mi=M/m[i];
        exgcd(Mi,m[i],x,y); //attention this is Mi not M
        ans=(ans+Mi*x*a[i])%M;
        //ans=(ans+step_mul(Mi*x,a[i],M))%M;
        //this is safe,but it's very slow
    }
    if (ans<0) ans+=M;
```

```
    return ans;
}
```

2.5.2 EXCRT

不保证两两互质也是可以的。

```
ll excrt(){
    ll M=m[1],ans=a[1];
    for (int i=2;i<=n;++i){
        ll tmpa=M,tmpb=m[i],tmpc;
        tmpc=(a[i]-ans%tmpb+tmpb)%tmpb;
        ll x,y;
        ll d=exgcd(tmpa,tmpb,x,y);
        if (tmpc%d) return -1; //there is no answer
        x=(x*tmpc/d)%(tmpb/d);
        //x=step_mul(x,tmpc/d,tmpb/d);
        //also safe but slow
        ans+=x*M; M*=tmpb/d; ans=(ans%M+M)%M;
    }
    return (ans%M+M)%M;
}
```

2.5.3 StepMul

```
ll step_mul(ll aa,ll bb,ll mod){
    ll tmp=0;
    while (bb){
        if (bb&1) tmp=(tmp+aa)%mod;
        aa=(aa+aa)%mod; bb>>=1;
    }
    return tmp%mod;
}
```

2.6 BSGS

求得 x ，使得 $a^x = b(mod\ p)$。(**要求 a,p 互质，不一定 p 为质数**)

```
unordered_map<ll,ll> mp; //if use map then add a log
ll bsgs(ll a,ll b){ //a^x=b(%p)
    if (!a){
        if (!b) return 0;
        else if (b==1) return 0;
        else return -1;
    }
    mp.clear();
```

```
    if (b==1) return 0;
    ll ans=-1,m=sqrt(p)+1,s=b;
    for (ll i=0;i<m;++i){
        mp[s]=i; s=s*a%p;
    }
    ll tmp=ksm(a,m); s=1;
    for (ll i=1;i<=m+1;++i){
        s=s*tmp%p;
        if (mp.count(s)){
            ans=i*m-mp[s];
            break;
        }
    }
    return ans;
}
```

2.7 二次剩余

求解 x 使得 $x^2 \equiv a \pmod p$，即模意义下开方。p 是奇素数。二次剩余有 $\frac{p-1}{2}$ 个，非二次剩余有 $\frac{p-1}{2}$ 个。

勒让德符号：$\left(\frac{n}{p}\right)$，值为 1 时 n 是 p 的二次剩余且 $n \not| p$，0 时 $p \mid n$，-1 时 n 不是 p 的二次剩余且 $n \not| p$。

欧拉判别准则：$\left(\frac{n}{p}\right) \equiv n^{\frac{p-1}{2}} \pmod p$，通过判断 $n^{\frac{p-1}{2}} \equiv \pm 1 \pmod p$ 即可得知。

cipolla 求解 $x^2 \equiv a \pmod p$，随机找到 a 满足 $a^2 - n$ 为**非二次剩余**（期望为 2 次），定义 $i^2 \equiv a^2 - n \pmod p$。虽然 $a^2 - n$ 不是二次剩余，可以扩范围到复数。

考虑到 $(a+b)^p \equiv a^p + b^p \pmod p$，$i^p \equiv i \cdot (i^2)^{\frac{p-1}{2}} \equiv -i \pmod p$，$a^p \equiv a \pmod p$。可以得到结论：

$$x \equiv (a+i)^{\frac{p+1}{2}} \equiv n^{\frac{1}{2}} \pmod p$$

得知一个解以后，另一个解便是其模意义上的相反数 $p - x$。同时别忘了 0 的存在。

```
namespace cipolla{
    ll I2;
    struct cp{
        ll rel,img;
        cp(ll _r=0,ll _i=0):rel(_r),img(_i){}
        bool operator == (cp _a){
            return rel==_a.rel&&img==_a.img;
        }
```

```
        cp operator * (cp _a){
            return cp((rel*_a.rel+I2*img%mod*_a.img)%mod,
                      (img*_a.rel+rel*_a.img)%mod);
        }
    };
    cp ksm(cp x,ll k){
        cp res(1,0);
        while (k){
            if (k&1) res=res*x;
            x=x*x; k>>=1;
        }
        return res;
    }
    ll ksm(ll x,ll k){
        ll res=1;
        while (k){
            if (k&1) res=res*x%mod;
            x=x*x%mod; k>>=1;
        }
        return res;
    }
    bool check(int x){
        return ksm(cp(x,0),(mod-1)>>1)==cp(1,0);
    }
    ll solve(ll n){
        if (!n) return 0; //special judge 0
        if (ksm(n,(mod-1)>>1)==mod-1) return -1; //no answer
        ll tmp=rand()%mod;
        while (!tmp||check((tmp*tmp-n+mod)%mod)){
            tmp=rand()%mod;
        }
        I2=(tmp*tmp-n+mod)%mod;
        return ksm(cp(tmp,1),(mod+1)>>1).rel;
    }
}
```

2.8 卢卡斯定理及其扩展

2.8.1 Lucas

Lucas 定理：要求 p 是质数（实际应用要求 p 不算大，在 10^5 左右）。

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

（以下代码 **p 均为全局变量**）

```cpp
ll Comp(ll a,ll b,ll p){
    if (a<b) return 0;
    if (a==b) return 1;
    if (b>a-b) b=a-b;
    ll ans=1,ca=1,cb=1;
    for (int i=0;i<b;++i){
        ca=ca*(a-i)%p;
        cb=cb*(b-i)%p;
    }
    ans=ca*ksm(cb,p-2)%p;
    return ans;
}
ll lucas(ll n,ll m,ll p){
    if (!m) return 1;
    return (Comp(n%p,m%p,p)*lucas(n/p,m/p,p))%p;
}
```

2.8.2 ExLucas

扩展 Lucas：p 可以不为质数： 将 p 分解质因数：$p = {p_1}^{k_1}{p_2}^{k_2}\cdots{p_n}^{k_n}$，对于每个 $C_n^m \equiv a_i \pmod{{p_i}^{k_i}}$ 可以构成一组中国剩余定理。对于 $C_n^m \bmod p^t$，转化考虑 $n! \bmod p^t$ 的值。我们把所有 n 及其倍数的因子提出来，发现提出来的因子为 $p^{\lfloor \frac{n}{p} \rfloor} \cdot (\lfloor \frac{n}{p} \rfloor)!$，可以递推计算。前半部分以 p^t 为周期，后半部分单独乘上即可。

```cpp
LL CRT(int n, LL* a, LL* m) {
  LL M = 1, p = 0;
  for (int i = 1; i <= n; i++) M = M * m[i];
  for (int i = 1; i <= n; i++) {
    LL w = M / m[i], x, y;
    exgcd(w, m[i], x, y);
    p = (p + a[i] * w * x % mod) % mod;
  }
  return (p % mod + mod) % mod;
}
LL calc(LL n, LL x, LL P) {
  if (!n) return 1;
  LL s = 1;
  for (int i = 1; i <= P; i++)
    if (i % x) s = s * i % P;
  s = Pow(s, n / P, P);
  for (int i = n / P * P + 1; i <= n; i++)
    if (i % x) s = s * i % P;
  return s * calc(n / x, x, P) % P;
```

```
}
LL multilucas(LL m, LL n, LL x, LL P) {
  int cnt = 0;
  for (int i = m; i; i /= x) cnt += i / x;
  for (int i = n; i; i /= x) cnt -= i / x;
  for (int i = m - n; i; i /= x) cnt -= i / x;
  return Pow(x, cnt, P) % P * calc(m, x, P) % P *
   ↪  inverse(calc(n, x, P), P) %
        P * inverse(calc(m - n, x, P), P) % P;
  //inverse是 return the inverse(niyuan)
}
LL exlucas(LL m, LL n, LL P) {
  int cnt = 0;
  LL p[20], a[20];
  for (LL i = 2; i * i <= P; i++) {
    if (P % i == 0) {
      p[++cnt] = 1;
      while (P % i == 0) p[cnt] = p[cnt] * i, P /= i;
      a[cnt] = multilucas(m, n, i, p[cnt]);
    }
  }
  if (P > 1) p[++cnt] = P, a[cnt] = multilucas(m, n, P, P);
  return CRT(cnt, a, p);
}
```

2.9 数论分块

2.9.1 性质

对于 $i \in \left[x, \lfloor \frac{n}{\lfloor \frac{n}{x} \rfloor} \rfloor\right]$，值 $\lfloor \frac{n}{i} \rfloor = \lfloor \frac{n}{x} \rfloor$ 是相同的。

设 $j = \lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \rfloor$，有 $j = \lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \rfloor \leq \frac{n}{\lfloor \frac{n}{i} \rfloor} < j+1$，所以 $\lfloor \frac{n}{i} \rfloor \leq \frac{n}{j}$，且 $\frac{n}{j+1} < \lfloor \frac{n}{i} \rfloor$。$j$ 为最大取值。

如果遇到 $\sum i \times f[i]$ 的问题，考虑 $f[i]$ 能否用数列性质得出。

2.9.2 代码

以下代码求解的是 $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$。

```
for (ll l=1;l<=n;l=r+1){
    r=n/(n/l);
    ans=1*(r-l+1)*(n/l);
}
```

2.10 莫比乌斯反演

$a|b : b \ mod \ a = 0$

$$F(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$$

$$F(n) = \sum_{n|d} f(d) \Rightarrow f(n) = \sum_{n|d} \mu(\frac{d}{n}) F(d)$$

注 (1)：一般遇到 $\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j) = d]$ 的问题，我们一般设 $f(d) = \sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j) = d]$，然后设 $F(k) = \sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j) = k, 2k, 3k, \cdots]$，则显然 $F(k) = \sum_{k|d} f(d)$。又因为 $F(k) = \lfloor \frac{n}{k} \rfloor \cdot \lfloor \frac{m}{k} \rfloor$ 容易得出，所以答案 $f(d) = \sum_{d|k} \mu(\frac{k}{d}) \cdot \lfloor \frac{n}{k} \rfloor \cdot \lfloor \frac{m}{k} \rfloor$ 可以方便求出。

注 (2):为了方便我们一般把 $\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j) = d]$ 的问题转化为 $\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor}\sum_{j=1}^{\lfloor \frac{m}{d} \rfloor}[gcd(i,j) = 1]$ 的问题，求解 $f(1) = \sum_{k=1}^{min(n,m)} \mu(k) \cdot \lfloor \frac{n}{kd} \rfloor \cdot \lfloor \frac{m}{kd} \rfloor$

注 (3)：莫比乌斯函数的一个重要性质：$\sum_{d|n} \mu(d) = [n = 1]$ ，在 gcd 不是非常裸（比如与其他项相乘）的时候，借助此性质巧妙转换：$\sum_{d|gcd(i,j)} \mu(d) = [gcd(i,j) = 1]$，然后我们将枚举 gcd 的约数转化为枚举约数即可。

2.10.1 mu 函数及其前缀和

```
int prime[maxn],tot;
bool notprime[maxn];
int mu[maxn],sum[maxn];
void get_mu(){ //get mu and sum of mu
    notprime[1]=true; mu[1]=1;
    for (int i=2;i<=50000;++i){
        if (!notprime[i]){
            prime[++tot]=i;
            mu[i]=-1;
        }
        for (int j=1;j<=tot&&i*prime[j]<=50000;++j){
            notprime[i*prime[j]]=true;
            if (i%prime[j]==0) break;
            else mu[i*prime[j]]=-mu[i];
        }
```

```
    }
    for (int i=1;i<=50000;++i){
        sum[i]=sum[i-1]+mu[i];
    }
}
```

2.10.2 数论分块计算答案前缀和

计算完前缀和一般就能处理区间问题。$\sum_{k=1}^{min(x,y)} \mu(k)\cdot\lfloor\frac{x}{kd}\rfloor\cdot\lfloor\frac{y}{kd}\rfloor$ 为例。

```
ll calc(int x,int y){
    ll tmp=0;
    int mx=min(x,y)/k;
    //if k>min(x,y)/k then F(k)=(x/(k*d))*(y/(k*d))=0
    for (int l=1,r;l<=mx;l=r+1){
        r=min(x/(x/l),y/(y/l));
        //if there is only x then x/(x/l)
        tmp+=(ll)(x/(l*k))*(ll)(y/(l*k))*(sum[r]-sum[l-1]);
    }
    return tmp;
}
```

2.11 亚线性筛

2.11.1 积性函数

$\forall a\perp b, f(ab)=f(a)f(b)$，若对不互质的成立则为完全积性函数。

常见的积性函数：

公式	解释
$d(x)=\sum_{i\|n}1$	约数个数
$\sigma(x)=\sum_{i\|n}i$	约数和
$\varphi(x)=\sum_{i=1}^{x}[gcd(x,i)=1]$	欧拉函数
$\mu(x)$	莫比乌斯函数
$\epsilon(x)=[x=1]$	元函数（完全积性）
$I(x)=1$	恒等函数（完全积性）
$id(x)=x$	单位函数（完全积性）

积性函数的性质：若 $f(x),g(x)$ 是积性函数，则以下函数也是积性函数。

1. $h(x)=f(x^p)$
2. $h(x)=f^p(x)$
3. $h(x)=f(x)g(x)$

4. $h(x)=\sum\limits_{d|x}f(d)g(\frac{x}{d})$

2.11.2 杜教筛

使用条件:

1. 复杂度 $O(n^{\frac{2}{3}})$ 。
2. $f(x)$ 为积性函数。
3. 能找到一个积性函数 $g(x)$，设 $h=f*g$，满足以下特点:
   (a) $h(x)$ 的前缀和很好求。

假设要计算 $\sum\limits_{i=1}^{n}f(i)$。我们设另外两个积性函数 h,g ，使得 $h=f*g$ 。记 $S(n)=\sum\limits_{i=1}^{n}f(i)$, 求得 $\sum\limits_{i=1}^{n}h(i)=\sum\limits_{d=1}^{n}g(d)\cdot S(\lfloor\frac{n}{d}\rfloor)$ 。进一步变形得到:

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

如果 $h(i)=(f*g)(i)$ 前缀和很好求得，那么对后面的式子进行整除分块，求 $s(n)$ 复杂度就是 $O(n^{\frac{2}{3}})$ 。下面给出栗子:

1. 求 $S(n)=\sum\limits_{i=1}^{n}\mu(i)$

设 g 为 I ，则 $h=f*g=\epsilon$ 。则 $S(n)=1-\sum\limits_{d=2}^{n}S(\lfloor\frac{n}{d}\rfloor)$ 。

2. 求 $S(n)=\sum\limits_{i=1}^{n}\varphi(i)$ 。

设 g 为 I ，则 $h=f*g=id$ 。则 $S(n)=\sum\limits_{i=1}^{n}i-\sum\limits_{d=2}^{n}S(\lfloor\frac{n}{d}\rfloor)$ 。

3. 求 $S(n)=\sum\limits_{i=1}^{n}i*\varphi(i)$

$h(n)=\sum_{d|n}d\cdot\varphi(d)\cdot g(\frac{n}{d})$ 。考虑将 d 消去，不妨设 $g=I$ ，则 $h(n)=\sum_{d|n}\varphi(d)\cdot n=n\sum_{d|n}\varphi(d)=n^2$ 。所以: $S(n)=\sum_{i=1}^{n}i^2-\sum_{d=2}^{n}d\cdot S(\lfloor\frac{n}{d}\rfloor)$ 。另外，$1^2+2^2+3^2+\cdots+n^2=\frac{n(n+1)(2n+1)}{6}$

具体实现：

- 先线性筛出前 $n^{\frac{2}{3}}$ 项, 剩余部分时间复杂度为 $O(\int_0^{n^{\frac{1}{3}}}\sqrt{\frac{n}{x}}\,dx)=O(n^{\frac{2}{3}})$
- 剩下的东西 Hash 存储，或者直接 unordered_map（如果用 map 则多一个 log）
- 注意取模以及**常数因子**的影响（过多的取模和 long long 的不必要使用）

以下代码是 1，2 问的。

```
int T; ll n;
int limit;
int prime[maxn],tot;
int notprime[maxn];
int mu[maxn],phi[maxn];
ll sum_mu[maxn],sum_phi[maxn];
void linear_sieve(){
    phi[1]=mu[1]=1; notprime[1]=1;
    for (int i=2;i<=limit;++i){
        if (!notprime[i]){
            prime[++tot]=i;
            mu[i]=-1; phi[i]=i-1;
        }
        for (int j=1;j<=tot&&i*prime[j]<=limit;++j){
            notprime[i*prime[j]]=1;
            if (!(i%prime[j])){
                phi[i*prime[j]]=phi[i]*prime[j];
                break;
            }else{
                phi[i*prime[j]]=phi[i]*(prime[j]-1);
                mu[i*prime[j]]=-mu[i];
            }
        }
    }
    for (int i=1;i<=limit;++i){
        sum_mu[i]=sum_mu[i-1]+mu[i];
        sum_phi[i]=sum_phi[i-1]+phi[i];
    }
}
unordered_map<ll,ll> map_mu;
unordered_map<ll,ll> map_phi;
ll askmu(ll x){
    if (x<=limit) return sum_mu[x];
    if (map_mu[x]) return map_mu[x];
    ll ans=1;
    for (ll l=2,r;l>=0&&l<=x;l=r+1){
        r=x/(x/l);
        ans-=(r-l+1)*askmu(x/l);
    }
    return map_mu[x]=ans;
}
ll askphi(ll x){
```

```
43      if (x<=limit) return sum_phi[x];
44      if (map_phi[x]) return map_phi[x];
45      ll ans=(x*(x+1))>>1;
46      for (ll l=2,r;l<=x;l=r+1){
47          r=x/(x/l);
48          ans-=(r-l+1)*askphi(x/l);
49      }
50      return map_phi[x]=ans;
51  }
52  int main(){
53      scanf("%d",&T);
54      limit=(int)(pow(2147483647,0.6666666666667));
55      linear_sieve();
56      while (T--){
57          scanf("%lld",&n);
58          printf("%lld %lld\n",askphi(n),askmu(n));
59      }
60      return 0;
61  }
```

2.11.3 Min25 筛

使用条件：

1. 复杂度 $O(\frac{n^{\frac{3}{4}}}{\log n})$ 。
2. $f(x)$ 为积性函数。
3. 设 p 为质数，则 $f(p), f(p^c)$ 可以快速计算。

过程

求解质数位置 $f(x)$ 和
设 p_k 为第 k 小的质数，约定 $p_0 = 1$。
设 $s(n)$ 为 n 的最小质因子，约定 $s(1) = 1$。
定义函数 $f'(x)$，满足以下特点：

1. $f'(x)$ 是完全积性函数。
2. 质数位置值与 $f(x)$ 相同，即 $\forall x \in prime, f(x) = f'(x)$。
3. $f'(x)$ 可以快速求得前缀和。

设 $g(n,k) = \sum_{i=2}^{n} [isprime(i) \ or \ s(i) > p_k] f'(i)$,先预处理出所有的 $g(n,0) = \sum_{i=2}^{n} f'(i)$ 。

设 m 为 $p_{m+1} > \sqrt{n}$ 的第一个位置，那么 $g(n,m)$ 就是所有质数的 $f'(x)$ 和，也即 $f(x)$ 和。

考虑写出 $g(n,m)$ 的递推式：

$$\begin{aligned} g(n,j) &= g(n,j-1) - \sum_{i=2}^{n} [i \neq p_j, s(i) = p_j] f'(i) \\ &= g(n,j-1) - \left[g(\lfloor \frac{n}{p_j} \rfloor, j-1) - g(p_j, j-1) \right] f'(p_j) \end{aligned}$$

因为 $\forall x \in prime, f(x) = f'(x)$ ，所以我们得到了 $x \in prime$ 的所有 $f(x)$ 的和。

求解所有位置 $f(x)$ 的和

设 $S(n,k) = \sum\limits_{i=p_k}^{n} [s(i) \geq p_k] f(i)$ ，将其分为两类：质数和合数。

质数的和：$g(n,m) - \sum\limits_{i=1}^{k-1} f(p_i)$ 。

偶数的和：通过枚举最小质因子及其次数方式解决，分为只有一个质因子的合数和有两个及以上质因子的合数。

有两个质因子及以上：假设枚举到的最小质因子为 p_k ，枚举到次数为 e ，则和可以表示为：

$\sum\limits_{p_k^{e+1} \leq n} f(p_k^e) S(\lfloor \frac{n}{p_k^e} \rfloor, k+1)$

只有一个质因子：设值为 $p_k^e (e \neq 1, p_k^e \leq n)$ ，则和可以表示为：

$\sum\limits_{e \neq 1, p_k^e \leq n} f(p_k^e) = \sum\limits_{p_k^{e+1}} f(p_k^{e+1})$

我们把两类合数加起来，再加上质数，得到式子递推即可：

$$S(n,t) = g(n,m) - \sum_{i=1}^{t-1} f(p_i) + \sum_{k=t}^{m} \sum_{p_k^{e+1} \leq n} \left[f(p_k^e) S(\lfloor \frac{n}{p_k^e} \rfloor, k+1) + f(p_k^{e+1}) \right]$$

例：求解质数的前缀和

发现 Min25 筛的前半部分就是对质数位置的函数值求和，不妨设 $f'(x) = x$ ，那么 $g(n,m)$ 就是所求的答案。

```
namespace Min25 {
    const int maxn = 1000010;
    int newN;
    int prime[maxn], totp, tot;
    ll sumprime[maxn];
    int notprime[maxn];
    ll quotient[maxn]; int totq;
    int id1[maxn], id2[maxn];
    ll g[maxn];
```

```
10      void linear_seive(int n) {
11          notprime[1] = 1;
12          for (int i = 2; i <= n; ++i) {
13              if (!notprime[i]) {
14                  prime[++tot] = i;
15                  sumprime[tot] = sumprime[tot-1] + i;
16              }
17              for (int j = 1; j <= tot &&
18                              i * prime[j] <= n; ++j) {
19                  notprime[i * prime[j]] = 1;
20                  if (i % prime[j] == 0) break;
21              }
22          }
23      }
24      ll calc(ll x) {
25          return x * (x + 1) / 2 - 1;
26      }
27      int ID(ll n, ll x) {
28          return x <= newN ? id1[x] : id2[n / x];
29      }
30      ll solve(ll n) {
31          if (n <= 1) return n;
32          newN = sqrt(n + 0.5);
33          /*
34              This make sure the linear seive be done
35              only once when there're multiple querys.
36          */
37          if (!tot) linear_seive(sqrt(10000000010));
38          totq = 0;
39          for (ll l = 1; l <= n; l = n / (n / l) + 1) {
40              quotient[++totq] = n / l;
41              if (quotient[totq] <= newN) {
42                  id1[quotient[totq]] = totq;
43              } else {
44                  id2[n / quotient[totq]] = totq;
45              }
46              g[totq] = calc(quotient[totq]);
47          }
48          for (int i = 1; i <= tot
49              && 1ll * prime[i] * prime[i] <= n; ++i) {
50              for (int j = 1; j <= totq &&
51                  1ll * prime[i] * prime[i] <= quotient[j];
52                  ++j) {
53                  g[j] = g[j] - prime[i] *
```

```
                        (g[ID(n, quotient[j] / prime[i])] -
                         sumprime[i - 1]);
            }
        }
        return g[ID(n, n)];
    }
}

// get sum of prime from 2 to n:
ll sum = Min25::solve(n)
```

2.12 高斯消元

```
const double eps=1e-9;
double abss(double x){
    return (x<0)?-x:x;
}
int check(){
    for (int i=1;i<=n;++i){
        int tot=0;
        for (int j=1;j<=n;++j){
            if (!a[i][j]) tot++;
        }
        if (tot==n&&abs(a[i][n+1])>eps) noans=true;
        if (tot==n&&abs(a[i][n+1])<=eps) manyans=true;
    }
    if (noans) return -1;
    if (manyans) return 0;
    return 1;
    /*
    -1:no answer,0:infinity answer,1:only one answer.
    judge no answer before infinity answer
    */
}
void gauss(){
    for (int i=1;i<=n;++i){
        int r=i;
        for (int j=i+1;j<=n;++j){
            if (abss(a[j][i])>abss(a[r][i])) r=j;
        }
        if (r!=i) swap(a[r],a[i]);
        if (abss(a[i][i])<=eps) continue;
        for (int j=1;j<=n;++j){
            if (i==j) continue;
```

```
32              double tmp=a[j][i]/a[i][i];
33              for (int k=1;k<=n+1;++k){
34                  a[j][k]-=tmp*a[i][k];
35              }
36          }
37      }
38      int flag=check();
39      if (flag==-1){ //no answer
40          printf("-1"); return;
41      }
42      if (flag==0){ //infinity answer
43          printf("0"); return;
44      }
45      for (int i=n;i>=1;--i){
46          a[i][n+1]/=a[i][i];
47      }
48      for (int i=1;i<=n;++i){
49          printf("x%d=",i);
50          if (abss(a[i][n+1])<=eps) printf("0\n");
51          else printf("%.2lf\n",a[i][n+1]);
52      }
53  }
```

2.13 拉格朗日插值

2.13.1 获得完整多项式

参考**高斯消元**。复杂度 $O(n^3)$

2.13.2 求特定 $f(k)$ 值

$f(k)=\sum_{i=1}^{n} y_i \times (\prod_{j\neq i} \frac{k-x_j}{x_i-x_j})$ 。复杂度 $O(n^2)$ 。

```
for (int i = 1; i <= n; i++){
    s1 = y[i]%mod; s2 = 1ll;
    for (int j = 1; j <= n; j++)
        if (i != j)
            s1 = s1 * (k - x[j]) % mod, s2 = s2 * ((x[i] -
             ↪  x[j] % mod) % mod) % mod;
    ans += s1 * inv(s2) % mod;
    ans = (ans + mod) % mod;
}
```

2.13.3 特殊情况：x 值连续

设 $pre[i]=\prod_{j=0}^{i} k-j, suf[i]=\prod_{j=i}^{n} k-j$ ，即前后缀积。设 $fac[i]$ 代表 i 的阶乘。那么式子变成：$f(k)=\sum_{i=0}^{n} y_i \frac{pre[i-1]*suf[i+1]}{fac[i]*fac[n-i]}$

注意：(1)$n-i$ 为奇数时分母为负。(2) 阶乘逆元线性求。复杂度 $O(n)$。

参考代码摘自KobeDuu：(其中的 $fac[i]$ 已经取了逆元)

```
LL Lagrange(LL f[],int n,int k){
    if(k <= n) return f[k];
    per[0] = suf[n+1] = 1;
    for(int i = 0;i <= n; ++i) per[i+1] = per[i]*(k-i)%mod;
    for(int i = n;i >= 0; --i) suf[i] = suf[i+1]*(k-i)%mod;
    LL fk = 0;
    for(int i = 0;i <= n; ++i){
        LL tep = f[i]*per[i]%mod*suf[i+1]%mod*fac[i]%mod*fac[n-i]%mod;
        if((n-i)&1) fk = (fk-tep+mod)%mod;
        else fk = (fk+tep)%mod;
    }
    return fk;
}
```

2.14 多项式

2.14.1 FFT

复杂度 $O(n\log n)$，一般用来解决两类问题。

1. 两个多项式的乘积，比如高精度乘法。
2. 快速求解 $\forall k, \sum_{i=1}^{n} f(i)g(i+k)$ 的所有值。设 $h(i)=f(n-i)$，那么所求就是 $\sum_{i=1}^{n} h(n-i)g(i+k)=h\times g(n+k)$，直接 FFT 即可。

此处模板下标从 0 开始
空间开的小，直接火葬场。推荐空间开 $(n+m)\times 2$ 或更多

```
#define cp complex_number
const double PI = acos(-1);
struct complex_number{
    double rel,img;
    complex_number(double x=0,double y=0){
        rel=x; img=y;
    }
};
```

```
cp operator + (cp a,cp b){
    return cp(a.rel+b.rel,a.img+b.img);
}
cp operator - (cp a,cp b){
    return cp(a.rel-b.rel,a.img-b.img);
}
cp operator * (cp a,cp b){
    return cp(a.rel*b.rel-a.img*b.img,a.rel*b.img+b.rel*a.img);
}
void fft(cp *a,int len,int inv)
{
    for (int i=0;i<len;++i){
        if (i<rev[i]) swap(a[i],a[rev[i]]);
    }
    for (int mid=1;mid<len;mid<<=1){
        cp tmp=cp(cos(PI/mid),inv*sin(PI/mid));
        for (int i=0;i<len;i+=mid<<1){
            cp omega=cp(1,0);
            for (int j=0;j<mid;++j,omega=omega*tmp){
                cp x=a[i+j],y=omega*a[i+j+mid];
                a[i+j]=x+y; a[i+j+mid]=x-y;
            }
        }
    }
}
//proceduce:
int limits=1,m,n,lim;
/*
limits=2^lim,ans's finally length
*/
cp aa[maxn],bb[maxn];
int rev[maxn];
void work(){
    limits=1; lim=0;
    while (limits<=m+n) limits<<=1,++lim;
    for (int i=0;i<=n;++i) aa[i].rel=read();
    for (int i=0;i<=m;++i) bb[i].rel=read();
    for (int i=0;i<limits;++i){
        rev[i]=(rev[i>>1]>>1)|((i&1)<<(lim-1));
    }//transfer to position
    fft(aa,limits,1); fft(bb,limits,1);
    for (int i=0;i<=limits;++i){
        aa[i]=aa[i]*bb[i];
    }
```

```
    fft(aa,limits,-1);
    for (int i=0;i<=n+m;++i){
        printf("%d ",(int)(aa[i].rel/limits+0.5));
    }
}
```

2.14.2 NTT

此处模板下标从 0 开始

原根 原根是 3 的也就有 998244353, 1004535809 几个。
原根一定是如下形式：$2, 4, p^a, 2p^a$ 。其中 p 为奇素数，a 为正整数。

求 p 的原根：对 $p-1$ 素因子分解，若恒有 $g^{\frac{\varphi(p)}{p_i}} \not\equiv 1 \pmod p$ 则成立（当 p 是质数时 $\varphi(p) = p-1$ ）

```
#define mod 998244353
#define g 3
ll ksm(ll x,int k){
    ll res=1;
    while (k){
        if (k&1) res=(res*x)%mod;
        x=(x*x)%mod; k>>=1;
    }
    return res%mod;
}
void ntt(ll *a,int len,int inv)
{
    for (int i=0;i<len;++i){
        if (i<rev[i]) swap(a[i],a[rev[i]]);
    }
    for (int mid=1;mid<len;mid<<=1){
        ll tmp=ksm((inv==1)?g:ksm(g,mod-2),(mod-1)/(mid*2));
        for (int i=0;i<len;i+=mid<<1){
            ll omega=1;
            for (int j=0;j<mid;++j,omega=omega*tmp%mod){
                ll x=a[i+j],y=omega*a[i+j+mid]%mod;
                a[i+j]=(x+y)%mod; a[i+j+mid]=(x-y+mod)%mod;
            }
        }
    }
}
void work(){
    limits=1; lim=0;
    while (limits<=m+n) limits<<=1,++lim;
```

```
    for (int i=0;i<limits;++i){
        rev[i]=(rev[i>>1]>>1)|((i&1)<<(lim-1));
    }
    ntt(aa,limits,1); ntt(bb,limits,1);
    for (int i=0;i<=limits;++i){
        aa[i]=(aa[i]*bb[i])%mod;
    }
    ntt(aa,limits,-1);
    ll inv=ksm(limits,mod-2);
    for (int i=0;i<=n+m;++i){
        printf("%lld ",aa[i]*inv%mod);
    }
    //final ans is aa[i]*inv%mod
}
```

2.15 线性基

2.15.1 基本性质

- 1. 线性基中元素互相异或得到的值域与原数组互相异或得到的**值域相同**。
- 2. 线性基是满足性质 1 的**最小集合**。
- 3. 线性基**没有异或和为 0 的子集**。
- 4. 线性基中每个元素的**异或方案唯一**，即每个异或组合出来的数是互不相同的。
- 5. 每个元素的二进制最高位不同。

```
void linear_basis(){
    for (int i=1;i<=n;++i){ //every number
        for (int j=60;j>=0;--j){ //every bit
            if (!(a[i]>>j)) continue;
            if (!d[j]){
                d[j]=a[i]; break; //break!
            }
            a[i]^=d[j];
        }
    }
}
//example:find the maxnum xor sum
ll ans=0;
for (int j=60;j>=0;--j){
    if (ans^d[j]>ans) ans^=d[j];
```

```
}
//example:find out how many different numbers can be xor to
ll ans=1;
for (int j=0;j<=60;++j){
    if (d[j]) ans<<=1;
    ans%=mod; //if needed
}
```

2.15.2 前缀线性基

查询区间中任意元素异或得到的最大值。

对于第一个元素到第 n 个元素的组合，让每一个基尽量靠近右端点，如果基在范围内则可以进行相关处理。

```
void InsertNode(int x){
    ++n; a[n]=a[n-1]; int pos=n;
    for (int i=31;i>=0;--i){
        if (!(x>>i)) continue;
        if (!a[n].d[i]){
            a[n].d[i]=x;
            a[n].p[i]=pos;
            break;
        }
        if (a[n].p[i]<pos){
            swap(a[n].p[i],pos);
            swap(a[n].d[i],x);
        }
        x^=a[n].d[i];
    }
}
int QueryAns(int ql,int qr){
    int res=0;
    for (int i=31;i>=0;--i){
        if (a[qr].p[i]>=ql){
            res=max(res,res^a[qr].d[i]);
        }
    }
    return res;
}
```

2.16 组合数学

2.16.1 容斥原理

考虑到容斥的相邻个数选择方案异号，如果时间允许可以容易用 dfs 实现。一个例子：给定数组 a$[1\cdots m]$，求在 $1\cdots n$ 中有多少数不是 a$[1\cdots m]$ 的倍数。

```
ll ans,a[21],n;
int m;
void dfs(ll val,int pos,int f){
    if (val>n) return;
    if (pos>m){
        ans+=n/val*f;
        return;
    }
    dfs(val,pos+1,f);
    dfs(val/gcd(val,a[pos])*a[pos],pos+1,-f);
}
```

2.16.2 康托展开

定理 小于该排列的排列数：

$$X = a_n(n-1)! + a_{n-1}(n-2)! + \cdots + a_1 0!$$

其中 a_n 代表在未出现的数中，有几个比当前位置的排列数小。排列 $[2,5,3,4,1]$ 中，式子为

$$1\times 4! + 3\times 3! + 1\times 2! + 1\times 1! + 0\times 0! = 45$$

如果要计算排名，则：

$$Rank = X + 1$$

此外，康拓展开得到的排列与排名是双射，可以视为一种离散化。朴素做法为 $O(n)$，利用树状数组或线段树 $O(n\log n)$。

康托展开 $O(n\log n)$

```
int n,a[maxn];
ll jc[maxn];
int tree[maxn];
void AddValue(int x,int k){
    while (x<=n){
        tree[x]+=k;
        x+=x&-x;
```

```
    }
}
int QueryPrefixSum(int x){
    int res=0;
    while (x){
        res+=tree[x];
        x-=x&-x;
    }
    return res;
}
void CantorExpansion(int a[]){
    jc[0]=1; ll rank=0;
    for (int i=1;i<=n;++i){
        jc[i]=jc[i-1]*i%mod; AddValue(i,1);
    }
    for (int i=1;i<=n;++i){
        rank=(rank+jc[n-i]*(QueryPrefixSum(a[i])-1))%mod;
        AddValue(a[i],-1);
    }
    printf("%lld\n",(rank+1)%mod);
}
```

逆康托展开 考虑 $n! > \sum_0^{n-1} i \times i!$ ，所以可以利用类似进制的除 $n!$ 取余法。这里给出朴素求法，如果数据过大考虑线段树（考虑到 rank 过大，大概率结合高精度）。

```
vector<int> InvContor(int bits,int num){
    num--;
    vector<bool> vis(bits+1,false);
    vector<int> p(bits,-1);
    int n,r=num;
    for (int i=0;i<bits;++i){
        n=r/(fact[bits-i-1]);
        r=r%(fact[bits-i-1]);
        for (int j=1;j<=bits;++j){
            if (!vis[j]&&!(n--)){
                vis[j]=true;
                p[i]=j;
                break;
            }
        }
    }
    return p;
}
```

2.17 生成函数

2.17.1 普通生成函数 OGF

解决组合问题。

$$F(x)=\sum_{i=0}^{\infty}f_ix^i$$

其中 **OGF 乘积对应数列的卷积**，即:

$$F(x)G(x)=\sum_n(\sum_{i=0}^{n}f_ig_{n-i})x^n$$

数列	OGF
$<1,0,0,0,\cdots>$	1
$<1,1,1,1,\cdots>$	$\frac{1}{1-x}$
$<1,-1,1,-1,\cdots>$	$\frac{1}{1+x}$
$<1,c,c^2,c^3,\cdots>$	$\frac{1}{1-cx}$
$<1,2,3,\cdots>$	$\frac{1}{(1-x)^2}$
$\sum_i^{\infty}C_{i+k-1}^{k-1}x^i$	$\frac{1}{(1-x)^k}$
$<\binom{n}{0},\binom{n}{1},\binom{n}{2},\cdots>$	$(1+x)^n$
$<0,1,\frac{1}{2},\frac{1}{3},\cdots>$	$ln\frac{1}{1-x}$
$<0,1,-\frac{1}{2},\frac{1}{3},-\frac{1}{4},\cdots>$	$ln(1+x)$

2.17.2 指数型生成函数 EGF

解决排列问题。

$$F(x)=\sum_n f_n\frac{x^n}{n!}$$

其乘法运算:

$$F(x)G(x)=\sum_n(\sum_{i=0}^{n}\binom{n}{i}f_ig_{n-i})x^i$$

多出来的 $\binom{n}{i}$ 适用于组合。x_i 代表从 f,g 中选出 i 个物品的排列数。

数列	OGF
$<1,1,1,1,\cdots>$	e^x
$<0,1,2,3,\cdots>$	xe^x
$<1,c,c^2,c^3,\cdots>$	e^{cx}

2.18 博弈论

2.18.1 Nim

- 定理 1：没有后继状态的状态是必败状态。
- 定理 2：一个状态是必胜状态当且仅当存在至少一个必败状态为它的必败状态。
- 定理 3：一个状态是必败状态当且仅当它的所有后继状态均为必胜状态。

我们设 Nim 和 $= a_1 \oplus a_2 \cdots \oplus a_n$ 。当且仅当 Nim 和为 0 时必败，否则必胜。我们对照三个定理归纳：

- 定理 1：没有后继状态即结果均为 0，则结果为 0。
- 定理 2：设 $a_1 \oplus a_2 \cdots \oplus a_n = k \neq 0$，需要证明一定存在移动使得异或和为 0。如果要该状态变成必败状态则需要异或 k，根据交换律我们可以只更改一个 a_i。一定有元素满足 $a_i > a_i \oplus k$，所以合法，即必胜。
- 定理 3：设 $a_1 \oplus a_2 \cdots \oplus a_n = 0$，需证明无法减少一位中的数使得新式子为 0。我们只能异或 0，但是必须有减少，所以无法移动，即必败。

2.18.2 SG 函数

设 $mex(S) = min\{x | x \notin S, x \in N\}$。

设状态 x 由 k 个子状态 $x_1, x_2, \ldots, x_k$ 组成，则

$SG(x) = x_1 \oplus x_2 \oplus x_3 \oplus \cdots \oplus x_k$ 。

设状态 x 具有 k 个后继状态 $y_1, y_2, \ldots, y_k$，则

$SG(x) = mex\{SG(y_1), SG(y_2), \cdots, SG(y_k)\}$。

总结一下，就是：

$SG(x) = mex\{SG(y_{11} \oplus y_{12} \oplus \cdots), SG(y_{21} \oplus \cdots), \cdots, SG(y_{k1} \oplus \cdots)\}$

SG 定理：对于 n 个有向图游戏形成的组合游戏，设起点为 $s_1, s_2, \cdots, s_n$，则当且仅当 $SG(x_1) \oplus SG(x_2) \oplus \cdots \oplus SG(s_n) \neq 0$ 时，先手必胜。

我们一般找到几个特殊情况赋予初值，然后求出其他情况即可。

例：分石子问题

给定 n 堆石子，每堆 a_i 个。每次可以选择个数为 $x \neq 1$ 的一堆，将其分为 y 堆，每堆个数为 $\frac{x}{y}$ 。不能操作的输，问谁最后赢。

考虑不为 2 的质数：

$SG(p) = mex\{SG(1) \oplus SG(1) \oplus \cdots \oplus SG(1)\} = 1$

可以发现，分成的堆数状态相同且都是奇数，异或结果为 1 个的结果。

$$\begin{aligned} SG(p_1p_2\cdots p_n) &= mex\{SG(1), SG(p_1), SG(p_1p_2), \ldots, SG(p_1p_2\cdots p_{n-1})\} \\ &= mex\{0, 1, 2, \ldots, n-1\} \\ &= n \end{aligned}$$

考虑为 2 的次方：
可以看到分成的堆数都是偶数，在异或的时候都为 0。
$SG(2^p) = mex\{0\} = 1$

考虑所有数的结果：
$SG(p_1p_2\cdots p_n 2^m) = n + [m \geq 1]$

3 图论

3.1 树的直径和重心

3.1.1 树的直径

从任一点开始 dfs 求出最远的节点，则该节点一定是直径的一端。从得到的直径一端开始 dfs，最远的节点便是另一端。

3.1.2 树的重心

使得子树中最大的子树节点数最小的点。有一个或两个。

在所有点中，重心到其他所有点的距离和是最小的。如果有多个点，则所有重心的距离和一样。

一棵树上添加或删除一个叶子，重心最多只移动一条边。

若用一条边将两树相连，则新树的重心位于两树重心路径上。

```
int size[MAXN], weight[MAXN], centroid[2];
void GetCentroid(int cur, int fa) {
    size[cur] = 1;
    weight[cur] = 0;
    for (int i = head[cur]; i != -1; i = e[i].nxt) {
        if (e[i].to != fa) {
            GetCentroid(e[i].to, cur);
            size[cur] += size[e[i].to];
            weight[cur] = max(weight[cur], size[e[i].to]);
        }
    }
    weight[cur] = max(weight[cur], n - size[cur]);
    if (weight[cur] <= n / 2) {
        centroid[centroid[0] != 0] = cur;
    }
}
```

3.2 最小生成树

3.2.1 prim $O(N^2)$

设 MST 集合 V。先选择任意一点作为起始点加入 V，之后从集合中找到一点使得其到 V 中任意一点的距离最小。

转载自最小生成树之 prim 算法

```
1 void prim(int u){
2     int sum_mst = 0;
3     for(int i = 1; i <= n; i++){ //initial
4         lowcost[i] = Map[u][i];
```

```
5           mst[i] = u;
6       }
7       mst[u] = -1;
8       for(int i = 1; i < n; i++){
9           int minn = INF;
10          int v = -1;
11          //find the minest mst in lowcost
12          for(int j = 1; j <= n; j++){
13              if(mst[j] != -1 && lowcost[j] < minn){
14                  v = j; minn = lowcost[j];
15              }
16          }
17          if(v != -1){
18              printf("%d %d %d\n", mst[v], v, lowcost[v]);
19              //output path
20              mst[v] = -1;
21              sum_mst += lowcost[v];
22              for(int j = 1; j <= n; j++){
23                  if(mst[j] != -1 && lowcost[j] > Map[v][j]){
24                      lowcost[j] = Map[v][j];
25                      mst[j] = v;
26                  }
27              }
28          }
29      }
30      printf("weight of mst is %d\n", sum_mst);
31  }
```

3.2.2 kruskal $O(M \log M)$

```
1   struct line{
2       int from,to,val;
3       bool operator < (const line _a){
4           return val<_a.val;
5       }
6   };
7   vector<line> edge;
8   int fa[maxn];
9   int find_father(int x){
10      return (x==fa[x])?x:fa[x]=find_father(fa[x]);
11  }
12  bool addcheck(int x,int y,int v){
13      int fx=find_father(x),fy=find_father(y);
14      if (fx==fy){
```

```
15          return false;
16      }else{
17          fa[fx]=fy;
18          return true;
19      }
20  }
21  void work(){
22      sort(edge.begin(),edge.end());
23      int tot=0;
24      for (int i=0;i<edge.size();++i){
25          line e=edge[i];
26          if (addcheck(e.from,e.to,e.val)) ++tot;
27          if (tot==n-1) break;
28      }
29  }
```

3.3 最近公共祖先

3.3.1 倍增 LCA

```
1   vector<int> G[maxn];
2   int anc[maxn][20];
3   int dep[maxn];
4   void dfs(int x,int fa,int d){
5       dep[x]=d; anc[x][0]=fa;
6       for (int i=0;i<G[x].size();++i){
7           if (G[x][i]==fa) continue;
8           dfs(G[x][i],x,d+1);
9       }
10  }
11  void lca_prework(){
12      dfs(1,0,1);
13      for (int j=1;j<20;++j){
14          for (int i=1;i<=n;++i){
15              anc[i][j]=anc[anc[i][j-1]][j-1];
16          }
17      }
18  }
19  int single_up(int x,int k){
20      for (int j=0;k;++j){
21          if (k&1) x=anc[x][j];
22          k>>=1;
23      }
24      return x;
```

```
}
int lca(int x,int y){
    if (dep[x]<dep[y]) swap(x,y);
    x=single_up(x,dep[x]-dep[y]);
    if (x==y) return x;
    for (int j=19;j>=0;--j){
        if (anc[x][j]!=anc[y][j]){
            x=anc[x][j]; y=anc[y][j];
        }
    }
    return anc[x][0];
}
```

3.4 树上差分

3.4.1 点差分

$a \to b$ 路径节点加 k:
(1) 两端点权值 +k
(2)lca(a,b) 权值-k
(3)fa[lca(a,b)] 权值-k

```
void addpath(int a,int b,int k){
    cnt[a]+=k; cnt[b]+=k;
    int f=lca(a,b);
    cnt[f]-=k; cnt[anc[f][0]]-=k;
}
void dfs_sum(int x,int fa){
    for (int i=0;i<G[x].size();++i){
        if (G[x][i]==fa) continue;
        dfs_sum(G[x][i],x);
        cnt[x]+=cnt[G[x][i]];
    }
}
```

3.4.2 边差分

边差分同点差分相似，但处理方式有一些差异:
(1):$a \to b$ 路径节点加 k :
[1] 两端节点值 + k
[2]lca(a,b)-=2k
(2): 询问两点的距离:
[1] 预处理每个节点到根的距离 d[]
[2]dis(a,b)=d[a]+d[b]-2*d[lca(a,b)]

3.5 树链剖分

注：val[x]=n_val[dfn[x]]

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>
using namespace std;
#define ll long long
#define il inline
const int maxn=100010;
struct treenode{
    ll sum,lazy;
}tree[maxn<<2];
vector<int> G[maxn];
int n,m,root; ll mod;
ll val[maxn],n_val[maxn];
int opt,rx,ry; ll rz;
int fa[maxn],dep[maxn],son[maxn];
int siz[maxn],top[maxn],dfn[maxn],tot;
/*
treenode:sgt val[]->n_val[]:turn the order
first:dfs_getson(). then dfs_getlink()
*/
void dfs_getson(int u){
    siz[u]=1;
    for (int i=0;i<G[u].size();++i){
        int v=G[u][i];
        if (v==fa[u]) continue;
        fa[v]=u; dep[v]=dep[u]+1;
        dfs_getson(v); siz[u]+=siz[v];
        if (siz[v]>siz[son[u]]) son[u]=v;
    }
}
void dfs_getlink(int u,int tp){
    top[u]=tp; dfn[u]=++tot; n_val[tot]=val[u];
    if (son[u]) dfs_getlink(son[u],tp);
    for (int i=0;i<G[u].size();++i){
        int v=G[u][i];
        if (v==fa[u]||v==son[u]) continue;
        dfs_getlink(v,v);
    }
```

```
}
#define ls o<<1
#define rs ls|1
void sgt_build(int o,int l,int r){
    if (l==r){
        tree[o].sum=n_val[l];
        return;
    }
    int mid=(l+r)>>1;
    sgt_build(ls,l,mid); sgt_build(rs,mid+1,r);
    tree[o].sum=(tree[ls].sum+tree[rs].sum)%mod;
}
void sgt_pushdown(int o,int l,int r){
    tree[ls].lazy=(tree[ls].lazy+tree[o].lazy)%mod;
    tree[rs].lazy=(tree[rs].lazy+tree[o].lazy)%mod;
    int mid=(l+r)>>1;
    tree[ls].sum=(tree[ls].sum+tree[o].lazy*(mid-l+1)%mod)%mod;
    tree[rs].sum=(tree[rs].sum+tree[o].lazy*(r-mid)%mod)%mod;
    tree[o].lazy=0;
}
void sgt_add(int o,int l,int r,int ql,int qr,ll k){
    if (l>=ql&&r<=qr){
        tree[o].lazy=(tree[o].lazy+k)%mod;
        tree[o].sum=(tree[o].sum+(r-l+1)*k%mod)%mod;
        return;
    }
    if (tree[o].lazy) sgt_pushdown(o,l,r);
    int mid=(l+r)>>1;
    if (ql<=mid) sgt_add(ls,l,mid,ql,qr,k);
    if (qr>mid) sgt_add(rs,mid+1,r,ql,qr,k);
    tree[o].sum=(tree[ls].sum+tree[rs].sum)%mod;
}
ll sgt_querysum(int o,int l,int r,int ql,int qr){
    if (l>=ql&&r<=qr){
        return tree[o].sum;
    }
    int mid=(l+r)>>1;
    if (tree[o].lazy) sgt_pushdown(o,l,r);
    ll res=0;
    if (ql<=mid) res=(res+sgt_querysum(ls,l,mid,ql,qr))%mod;
    if (qr>mid) res=(res+sgt_querysum(rs,mid+1,r,ql,qr))%mod;
    return res;
}
#undef ls
```

```
#undef rs
void link_add(){
    while (top[rx]!=top[ry]){
        if (dep[top[rx]]>dep[top[ry]]) swap(rx,ry);
        sgt_add(1,1,tot,dfn[top[ry]],dfn[ry],rz);
        ry=fa[top[ry]];
    }
    if (dep[rx]>dep[ry]) swap(rx,ry);
    sgt_add(1,1,tot,dfn[rx],dfn[ry],rz);
}
ll link_query(){
    ll res=0;
    while (top[rx]!=top[ry]){
        if (dep[top[rx]]>dep[top[ry]]) swap(rx,ry);
        res=(res+sgt_querysum(1,1,tot,dfn[top[ry]],dfn[ry]))%mod;
        ry=fa[top[ry]];
    }
    if (dep[rx]>dep[ry]) swap(rx,ry);
    res=(res+sgt_querysum(1,1,tot,dfn[rx],dfn[ry]))%mod;
    return res;
}
void subtree_add(){
    sgt_add(1,1,tot,dfn[rx],dfn[rx]+siz[rx]-1,rz);
}
ll subtree_query(){
    return sgt_querysum(1,1,tot,dfn[rx],dfn[rx]+siz[rx]-1)%mod;
}
int main(){
    scanf("%d%d%d%lld",&n,&m,&root,&mod);
    for (int i=1;i<=n;++i){
        scanf("%lld",&val[i]);
    }
    for (int i=1;i<n;++i){
        scanf("%d%d",&rx,&ry);
        G[rx].push_back(ry);
        G[ry].push_back(rx);
    }
    dfs_getson(root); dfs_getlink(root,root);
    sgt_build(1,1,tot);
    for (int i=1;i<=m;++i){
        scanf("%d",&opt);
        if (opt==1){
            scanf("%d%d%lld",&rx,&ry,&rz);
            link_add();
```

```
130         }
131         if (opt==2){
132             scanf("%d%d",&rx,&ry);
133             printf("%lld\n",link_query());
134         }
135         if (opt==3){
136             scanf("%d%lld",&rx,&rz);
137             subtree_add();
138         }
139         if (opt==4){
140             scanf("%d",&rx);
141             printf("%lld\n",subtree_query());
142         }
143     }
144     return 0;
145 }
```

3.6 树分治

3.6.1 点分治

摘自【算法学习】点分治的两种写法与常见套路总结

点分治有 2 种写法。

1. 对于某个重心 u ，统计以 u 为根的所有路径，然后计算出所有组合情况。递归子树时，首先删除全部在一颗子树的路径，然后再进入子树递归求解。这样可以保证路径全部合法且不重不漏。

2. 对于某个重心 u ，先进入子树 v_1 ，求解出 u 到子树 v_1 所有节点的路径，然后进入子树 v_2 ，进入时先统计答案，然后再统计相关值。每次进入新的子树时，先统计答案，这样每次计算的路径一定是和之前统计过子树的相连而成的，没有不合法的答案，所以不用删除。最后，依次递归进子树，找出重心递归求解。

套路题型：

- 路径和等于或小于等于 k 的点对（路径条数）。
- 路径和为 k 的倍数。
- 路径和为 k 且路径的边数最少。
- 路径和 sum （mod M）后为某个值。
- 路径上经过不允许点的个数不超过某个值，且路径和最大。

1. 如若使用方法 1，一般开一个栈，保存路径上的距离等相关信息，排序后利用单调性或者二分找答案。

2. 如若使用方法 2，则一般处理这些问题时，都是开一个桶 $mess[i]$，表示距离为 i 的相关信息。

```
void get_root(int x,int fa){
    sz[x]=1; mxs[x]=0;
    for (int i=0;i<G[x].size();++i){
        line e=edge[G[x][i]];
        if (e.to==fa||vis[e.to]) continue;
        get_root(e.to,x);
        sz[x]+=sz[e.to];
        mxs[x]=max(mxs[x],sz[e.to]);
    }
    mxs[x]=max(mxs[x],sum-mxs[x]);
    if (mxs[x]<mxs[root]){
        root=x;
    }
}
void get_data(int x,int fa,int dis){
    sta[++top]=dis;
    for (int i=0;i<G[x].size();++i){
        line e=edge[G[x][i]];
        if (e.to==fa||vis[e.to]) continue;
        get_data(e.to,x,dis+e.val);
    }
}
void calc(int x,int flag,int dis){
    top=0; get_data(x,0,dis);
    for (int i=1;i<top;++i){
        for (int j=i+1;j<=top;++j){
            if (sta[i]+sta[j]>10000000) continue;
            ans[sta[i]+sta[j]]+=flag;
        }
    }
}
void solve(int x){
    calc(x,1,0); vis[x]=1;
    for (int i=0;i<G[x].size();++i){
        line e=edge[G[x][i]];
        if (vis[e.to]) continue;
        calc(e.to,-1,e.val);
        root=0; sum=sz[e.to];
        get_root(e.to,x); solve(root);
```

```
    }
}
```

3.7 最短路

3.7.1 floyed $O(N^3)$

```
int f[maxn][maxn];
memset(f,0x3f,sizeof(f));
for (int i=1;i<=n;++i) f[i][i]=0; //按需
for (int k=1;k<=n;++k){
    for (int i=1;i<=n;++i){
        for (int j=1;j<=n;++j){
            f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
        }
    }
}
```

3.7.2 spfa $O(kM)(k \approx 2)$

判断负环：$cnt[i]$ 表示从起点 s 到 i 的最短距离包含点的个数，初始化 $cnt[s]=1$ ，那么当我们能够用点 u 松弛点 v 时，松弛时同时更新 $cnt[v]=cnt[u]+1$，若发现此时 $cnt[v]>n$，那么就存在负环。更简单的做法：判断一个点是否入队 n 次以上。

```
int in_queue[maxn];
int dis[maxn];
void spfa(){
    memset(dis,0x3f,sizeof(dis));
    dis[s]=0;
    queue<int> q;
    q.push(s); in_queue[s]=1;
    while (!q.empty()){
        int u=q.front(); q.pop();
        in_queue[u]=0;
        for (int i=0;i<G[u].size();++i){
            line e=edge[G[u][i]];
            if (dis[u]+e.v<dis[e.to]){
                dis[e.to]=dis[u]+e.v;
                if (!in_queue[e.to]){
                    q.push(e.to); in_queue[e.to]=1;
                }
            }
        }
    }
```

```
}
//program below has been added slf
//considering deque is slow,so set apart
void spfa(){
    memset(dis,0x3f,sizeof(dis));
    dis[s]=0; in_queue[s]=1;
    deque<int> q; q.push_back(s);
    while (!q.empty()){
        int u=q.front(); q.pop_front();
        in_queue[u]=0;
        for (int i=0;i<G[u].size();++i){
            line e=edge[G[u][i]];
            if (dis[u]+e.v<dis[e.to]){
                dis[e.to]=dis[u]+e.v;
                if (!in_queue[e.to]){
                    in_queue[e.to]=1;
                    if (q.empty()||dis[e.to]>dis[q.front()])
                    ↪  q.push_back(e.to);
                    else q.push_front(e.to);
                }
            }
        }
    }
}
```

3.7.3 dijkstra

dijkstra 不支持负权边 !!!!!

~~暴力：$O(n^2+m)$~~
堆：$O((N+M)\ log\ N)$
优先队列：$O((N+M)\ log\ M)$
线段树（ZKW）：$O((N+M)\ log\ N)$
Fibonacci 堆：$O(Nlog\ N+M)$
这是个堆优化：

```
struct line{
    int to,v;
};
vector<line> edge;
vector<int> G[maxn];
struct node{
    int pos,dis;
    bool operator < (const node _a)const{
        return dis>_a.dis;
```

```
    }
};
priority_queue<node> q;
int dis[maxn];
bool vis[maxn];
void addedge(int from,int to,int val){
    edge.push_back((line){to,val});
    int m=edge.size();
    G[from].push_back(m-1);
}
void dij(){
    memset(dis,0x3f,sizeof(dis));
    dis[s]=0;
    q.push((node){s,0});
    while (!q.empty()){
        node u=q.top();
        q.pop();
        int now=u.pos,val=u.dis;
        if (!vis[now]){
            vis[now]=true;
            for (int i=0;i<G[now].size();++i){
                int to=edge[G[now][i]].to;
                int va=edge[G[now][i]].v;
                if (dis[to]>dis[now]+va){
                    dis[to]=dis[now]+va;
                    q.push((node){to,dis[to]});
                }
            }
        }
    }
}
```

3.7.4 最短路应用

- 求出两点间所有在最短路上的边：跑正反两个点的最短路，枚举每条边，满足 dis(S->u)+dis(T->v)+len(u,v) 的边便在集合中。

3.8 差分约束系统

(1) 如果涉及考虑输出最少 or 最大答案：如果求解的是**每两者差的最大值**则把每一个约束条件都转化为 $x[i]-x[j]\le c[k]$ ，建图后跑**最短路**；否则把条件转化为 $x[i]-x[j]\ge c[k]$ ，建图后跑**最长路**。

原型	转化	连边
$x_a - x_b \geq c$	$x_b - x_a \leq -c$	add(a,b,-c);
$x_a - x_b \leq c$	$x_a - x_b \leq c$	add(b,a,c);
$x_a = x_b$	$x_a - x_b \leq 0, x_b - x_a \leq 0$	add(b,a,0); add(a,b,0);
$x_a - x_b > c$	$x_b - x_a \leq -c - 1$	add(a,b,-c-1);

(2) 可以看做 $dis[y] \leq$ or $\geq dis[x] + z$ 的形式，从而对每一条：节点 j 向节点 i 连接长度为 $c[k]$ 的边。

(3) 判断是否有负环，从而判断无解 or 有解。

(4) 注意**图的连通性**问题：不确定时可以创建初始原点 0 ，向每一个边连一条边。如果有整体的初始条件，则考虑对原点 0 特殊赋值，并按照 中方法处理得到一组特殊解。

3.9 联通分量相关

3.9.1 强联通分量

```
int dfn[maxn],low[maxn],dfncnt;
int sta[maxn],tp; //stack
int scc[maxn],scc_tot;
//scc[]:every point belongs to which scc
int siz[maxn]; //the siz of each scc
int in_stack[maxn];
void tarjan(int u){
    dfn[u]=low[u]=++dfncnt;
    sta[++tp]=u; in_stack[u]=1;
    for (int i=0;i<G[u].size();++i){
        int v=G[u][i];
        if (!dfn[v]){
            tarjan(v); low[u]=min(low[u],low[v]);
        }else{
            if (in_stack[v]) low[u]=min(low[u],dfn[v]);
        }
    }
    if (dfn[u]==low[u]){
        ++scc_tot;
        while (sta[tp]!=u){
            scc[sta[tp]]=scc_tot; ++siz[scc_tot];
            in_stack[sta[tp--]]=0;
        }
        scc[sta[tp]]=scc_tot; ++siz[scc_tot];
        in_stack[sta[tp--]]=0;
    }
}
```

```
//to be used
for (int i=1;i<=n;++i){
    if (!dfn[i]) tarjan(i);
}
```

3.9.2 割点

```
int dfn[maxn],low[maxn],dfncnt;
int ans[maxn],anstot;
//anstot:tot of ans,ans[]=1 means ge dian
int vis[maxn];
vector<int> G[maxn];
void tarjan(int u,int fa){
    dfn[u]=low[u]=++dfncnt; vis[u]=1;
    int totchild=0;
    for (int i=0;i<G[u].size();++i){
        int v=G[u][i];
        if (!vis[v]){
            ++totchild; tarjan(v,u);
            low[u]=min(low[u],low[v]);
            if (u!=fa&&low[v]>=dfn[u]&&!ans[u]){
                anstot++; ans[u]=1;
            }
        }else{
            if (v==fa) continue;
            low[u]=min(low[u],dfn[v]);
        }
    }
    if (u==fa&&totchild>=2&&!ans[u]){
        ++anstot; ans[u]=1;
    }
}
```

3.9.3 割边 (桥)

```
int low[MAXN], dfn[MAXN], iscut[MAXN], dfs_clock;
bool isbridge[MAXN];
vector<int> G[MAXN];
int cnt_bridge;
int father[MAXN];
void tarjan(int u, int fa) {
  father[u] = fa;
  low[u] = dfn[u] = ++dfs_clock;
  for (int i = 0; i < G[u].size(); i++) {
```

```
10      int v = G[u][i];
11      if (!dfn[v]) {
12        tarjan(v, u);
13        low[u] = min(low[u], low[v]);
14        if (low[v] > dfn[u]) {
15          isbridge[v] = true;
16          ++cnt_bridge;
17        }
18      } else if (dfn[v] < dfn[u] && v != fa) {
19        low[u] = min(low[u], dfn[v]);
20      }
21    }
22  }
```

3.10 2-SAT

涉及 2*n 个物品，已经有 n 对两者能且只能选出一个（不能不选）的情况。给出其中互斥情况，询问是否有满足去 n 个人的情况。设 A1,A2 为一组，同理 B。若 A1 与 B1 互斥，则选 A1 后必选 B2，选 B1 后必选 A2；将两种情况连边，通过判断联通。若 A 组或 B 组中两个元素同属于一个连通分量则无解。若所有情况都不是无解则是有解。

```
1   namespace TwoSat{
2       vector<int> G[maxn<<1];
3       void addedge(int from,int to){
4           G[from].push_back(to);
5       }
6       int dfn[maxn<<1],low[maxn<<1],dfncnt;
7       int sta[maxn<<1],tp;
8       int scc[maxn<<1],scc_tot;
9       int siz[maxn<<1];
10      int in_stack[maxn<<1];
11      //if there are multi problem don't forget
12      void init(){
13          for (int i=0;i<(n<<1);++i) G[i].clear();
14          dfncnt=tp=scc_tot=0;
15          memset(dfn,0,sizeof(dfn));
16          memset(low,0,sizeof(low));
17          memset(sta,0,sizeof(sta));
18          memset(scc,0,sizeof(scc));
19          memset(siz,0,sizeof(siz));
20          memset(in_stack,0,sizeof(in_stack));
21      }
22      void tarjan(int u){
```

```
23          dfn[u]=low[u]=++dfncnt;
24          sta[++tp]=u; in_stack[u]=1;
25          for (int i=0;i<G[u].size();++i){
26              int v=G[u][i];
27              if (!dfn[v]){
28                  tarjan(v); low[u]=min(low[u],low[v]);
29              }else{
30                  if (in_stack[v]) low[u]=min(low[u],dfn[v]);
31              }
32          }
33          if (dfn[u]==low[u]){
34              ++scc_tot;
35              while (sta[tp]!=u){
36                  scc[sta[tp]]=scc_tot; ++siz[scc_tot];
37                  in_stack[sta[tp--]]=0;
38              }
39              scc[sta[tp]]=scc_tot; ++siz[scc_tot];
40              in_stack[sta[tp--]]=0;
41          }
42      }
43      bool solve(){
44          //adjust the range if necessary
45          for (int i=0;i<(n<<1);++i){
46              if (!dfn[i]) tarjan(i);
47          }
48          for (int i=0;i<(n<<1);i+=2){
49              if (scc[i]==scc[i+1]) return false;
50          }
51          return true;
52      }
53  }
```

3.11 拓扑排序

```
1   bool toposort() {
2       q = new queue();
3       for (i = 0; i < n; i++)
4           if (in_deg[i] == 0) q.push(i);
5       ans = new vector();
6       while (!q.empty()) {
7           u = q.pop();
8           ans.push_back(u);
9           for each edge(u, v) {
10              if (--in_deg[v] == 0) q.push(v);
```

```
11          }
12      }
13      if (ans.size() == n) {
14          for (i = 0; i < n; i++)
15              cout << ans[i] << endl;
16          return true;
17      } else {
18          return false;
19      }
20  }
```

3.12 图的匹配

3.12.1 基本性质

二分图极大匹配 在当前已完成的匹配下, 无法再通过增加未完成匹配的边的方式来增加匹配的边数。

二分图最大匹配 所有极大匹配当中边数最大的一个匹配。选择这样的边数最大的子集称为图的最大匹配问题。

二分图完美匹配（完备匹配） 一个图中所有的顶点都是匹配点的匹配，即 2|M| = |V|。完美匹配一定是最大匹配，但并非每个图都存在完美匹配。

二分图最优匹配 最优匹配又称为带权最大匹配，是指在带有权值边的二分图中，求一个匹配使得匹配边上的权值和最大。一般 X 和 Y 集合顶点个数相同，最优匹配也是一个完备匹配，即每个顶点都被匹配。如果个数不相等，可以通过补点加 0 边实现转化。一般使用 KM 算法解决该问题。

二分图最小覆盖 包括最小顶点覆盖和最小路径覆盖。

最小顶点覆盖是指最少的顶点数使得二分图 G 中的每条边都至少与其中一个点相关联。二分图的最小顶点覆盖数 = 二分图的最大匹配数。

最小路径覆盖也称为最小边覆盖，是指用尽量少的不相交简单路径覆盖二分图中的所有顶点。二分图的最小路径覆盖数 =|V|-二分图的最大匹配数。

二分图最大独立集 最大独立集是指寻找一个点集，使得其中任意两点在图中无对应边。二分图的来最大独立集 =|V|-二分图的最大匹配数。最大独立集 S 与最小覆盖集 T 互补。

3.12.2 二分图最大匹配（匈牙利算法）

可以直接利用网络流跑 dinic 即可，复杂度 $O(m\sqrt{n})$ 。这里使用匈牙利算法，复杂度 $O(nm)$。匈牙利算法可以忽略重边，而写 dinic 要注意重边问题。

```cpp
namespace hungarian{
    vector<int> G[maxn];
    int vis[maxn];
    int p[maxn]; //p[right]=left
    bool match(int u){
        for (int i=0;i<G[u].size();++i){
            int v=G[u][i];
            if (vis[v]) continue;
            vis[v]=1;
            if (!p[v]||match(p[v])){
                p[v]=u; return true;
            }
        }
        return false;
    }
    int hungarian(){
        int cnt=0;
        for (int i=1;i<=n;++i){
            memset(vis,0,sizeof(vis));
            if (match(i)){
                ++cnt;
            }
        }
        return cnt;
    }
}
```

3.12.3 二分图最大权匹配（KM 算法）

可以直接建图跑费用流。这里使用 KM 算法，复杂度 $O(n^3)$。

KM 算法是在满足**完美匹配**的前提下计算的，如果不满足存在完美匹配则需要增加虚点虚边。因为建图一般允许使用邻接矩阵（不能用邻接矩阵的也估计跑不了 $O(n^3)$），所以在默认中就增加了点。至少确保左侧点数量大于右侧点数量。

之后分两种情况处理：允许存在不完备匹配和特判不存在完备匹配情况并输出。

允许存在不完备匹配时，我们可以将所有的边和虚边默认权值设为 0，保证存在完美匹配且选择该边无影响。同时注意将给出的负边的权值设为 0，因为该边绝对不会被选取。

需要特判不存在完备匹配时，将所有边和虚边设为-inf。匹配之后判断对每个 px[] 匹配对应的边是否权值为-inf，若选择了权值为-inf 的边即无解。

注：px[x]：x（左侧）对应右侧的点是 px[x] 。py[y] 同理。

```cpp
namespace km{
```

```
    int px[maxn],py[maxn];
    int pre[maxn];
    int visx[maxn],visy[maxn];
    ll lx[maxn],ly[maxn];
    ll slack[maxn];
    ll g[maxn][maxn];
    queue<int> q;
    void init(){
        for (int i=0;i<maxn;++i){
            for (int j=0;j<maxn;++j){
                g[i][j]=-inf;
            }
        }
        memset(px,-1,sizeof(px));
        memset(py,-1,sizeof(py));
        memset(pre,0,sizeof(pre));
        memset(visx,0,sizeof(visx));
        memset(visy,0,sizeof(visy));
        memset(slack,0x3f,sizeof(slack));
    }
    void addedge(int u,int v,ll w){
        g[u][v]=w;
    }
    bool match(int v){
        visy[v]=1;
        if (py[v]!=-1){
            q.push(py[v]);
            visx[py[v]]=1;
            return false;
        }
        while (v!=-1){
            py[v]=pre[v];
            swap(v,px[pre[v]]);
        }
        return true;
    }
    void bfs(int x){
        while (!q.empty()) q.pop();
        q.push(x); visx[x]=1;
        while (true){
            while (!q.empty()){
                int u=q.front(); q.pop();
                for (int v=1;v<=n;++v){
                    if (visy[v]) continue;
```

```
46                      ll delta=lx[u]+ly[v]-g[u][v];
47                      if (slack[v]>=delta){
48                          pre[v]=u;
49                          if (delta){
50                              slack[v]=delta;
51                          }else{
52                              if (match(v)) return;
53                          }
54                      }
55                  }
56              }
57              ll a=inf;
58              for (int j=1;j<=n;++j){
59                  if (!visy[j]){
60                      a=min(a,slack[j]);
61                  }
62              }
63              for (int j=1;j<=n;++j){
64                  if (visx[j]) lx[j]-=a;
65                  if (visy[j]) ly[j]+=a;
66                  else slack[j]-=a;
67              }
68              for (int j=1;j<=n;++j){
69                  if (!visy[j]&&!slack[j]&&match(j)) return;
70              }
71          }
72      }
73      ll solve(){
74          for (int i=1;i<=n;++i) lx[i]=-inf;
75          memset(ly,0,sizeof(ly));
76          for (int i=1;i<=n;++i){
77              for (int j=1;j<=n;++j){
78                  lx[i]=max(lx[i],g[i][j]);
79              }
80          }
81          for (int i=1;i<=n;++i){
82              memset(slack,0x3f,sizeof(slack));
83              memset(visx,0,sizeof(visx));
84              memset(visy,0,sizeof(visy));
85              bfs(i);
86          }
87          ll res=0;
88          for (int i=1;i<=n;++i){
89              res+=lx[i]+ly[i];
```

```
        }
        return res;
    }
}
```

3.12.4 一般图最大匹配（带花树）

二分图其实就是包含 0 个或多个偶环的图。一般图包括奇环，需要使用缩花缩奇环，使用带花树。复杂度 $O(n^3)$ ，均摊 $O(n(nlog\ n+m))$ ，通常可以认为是 $O(nm)$ ，和网络流一样，复杂度一般跑不满。

lk[x] 代表与 x 匹配的编号（没有则是 0）。solve() 返回匹配最大值。

```
namespace FlowerTree{
    vector<int> G[maxn];
    int lk[maxn];
    int tag[maxn],pre[maxn];
    int fa[maxn];
    int dfn[maxn],dfncnt;
    queue<int> q;
    void addedge(int u,int v){
        G[u].push_back(v);
        G[v].push_back(u);
        if (!lk[u]&&!lk[v]){
            lk[u]=v; lk[v]=u; ++ans;
        }
    }
    int findx(int x){
        return (fa[x]==x)?x:fa[x]=findx(fa[x]);
    }
    void aug(int x){
        int nxt;
        while (x){
            nxt=lk[pre[x]];
            lk[x]=pre[x];
            lk[pre[x]]=x;
            x=nxt;
        }
    }
    int lca(int x,int y){
        ++dfncnt;
        while (dfn[x=findx(x)]!=dfncnt){
            if (x) dfn[x]=dfncnt;
            x=pre[lk[x]]; swap(x,y);
        }
        return x;
```

```
34      }
35      void shrink(int u,int v,int p){
36          while (findx(u)!=p){
37              pre[u]=v; v=lk[u];
38              fa[u]=fa[v]=p;
39              if (tag[v]==2){
40                  tag[v]=1; q.push(v);
41              }
42              u=pre[v];
43          }
44      }
45      int blossom(int u){
46          memset(tag,0,sizeof(tag));
47          memset(pre,0,sizeof(pre));
48          for (int i=1;i<=n;++i) fa[i]=i;
49          while (!q.empty()) q.pop();
50          q.push(u); tag[u]=1;
51          while (!q.empty()){
52              int u=q.front(); q.pop();
53              for (int i=0;i<G[u].size();++i){
54                  int v=G[u][i];
55                  if (tag[v]==1){
56                      int p=lca(u,v);
57                      shrink(u,v,p);
58                      shrink(v,u,p);
59                  }
60                  else if (tag[v]==0){
61                      pre[v]=u; tag[v]=2;
62                      if (!lk[v]){
63                          aug(v); return 1;
64                      }else{
65                          tag[lk[v]]=1;
66                          q.push(lk[v]);
67                      }
68                  }
69              }
70          }
71          return 0;
72      }
73      int solve(){
74          for (int i=1;i<=n;++i){
75              if (!lk[i]){
76                  ans+=FlowerTree::blossom(i);
77              }
```

```
78          }
79          return ans;
80      }
81  }
```

3.13 网络流

3.13.1 最大流最小割

Dinic $O(n^2m)$　注意：当问题是**二分图匹配**时复杂度降到 $O(m\sqrt{n})$

```
1   struct line{
2       int to;
3       int cap,flow;
4   };
5   vector<line> edge;
6   vector<int> G[maxn];
7   int cur[maxn];
8   bool vis[maxn];
9   int dis[maxn];
10  int n,m,s,t,ans;
11  int ru,rv,rw;
12  void addedge(int from,int to,int cap){
13      edge.push_back((line){to,cap,0});
14      edge.push_back((line){from,0,0});
15      int m=edge.size();
16      G[from].push_back(m-2);
17      G[to].push_back(m-1);
18  }
19  bool bfs(){
20      memset(vis,false,sizeof(vis));
21      queue<int> q;
22      vis[s]=true;
23      dis[s]=0;
24      q.push(s);
25      while (!q.empty()){
26          int now=q.front();
27          q.pop();
28          for (int i=0;i<G[now].size();++i){
29              line e=edge[G[now][i]];
30              if (!vis[e.to]&&e.cap>e.flow){
31                  vis[e.to]=true;
32                  dis[e.to]=dis[now]+1;
33                  q.push(e.to);
34              }
```

```
        }
    }
    return vis[t];
}
int dfs(int x,int a){
    if (a==0||x==t) return a;
    int flow=0,f;
    for (int &i=cur[x];i<G[x].size();++i){
        line &e=edge[G[x][i]];
        if (dis[e.to]==dis[x]+1&&(f=dfs(e.to,min(e.cap-e.flow,a)))){
            flow+=f;
            a-=f;
            e.flow+=f;
            edge[G[x][i]^1].flow-=f;
            if (!a) break;
        }
    }
    return flow;
}
int dinic(){
    while (bfs()){
        memset(cur,0,sizeof(cur));
        ans+=dfs(s,1<<30);
    }
}
```

HLPP $O(n^2\sqrt{m})$

```
const int inf=0x3f3f3f3f;
int n,m;
namespace hlpp{
    int S,T;
    struct line{
        int u,v; ll f;
    };
    vector<line> edge;
    vector<int> G[maxn];
    void AddFlow(int u,int v,ll f){
        edge.push_back((line){u,v,f});
        edge.push_back((line){v,u,0});
        int m=edge.size();
        G[u].push_back(m-2);
        G[v].push_back(m-1);
    }
```

```
17      int ht[maxn],gap[maxn]; ll ex[maxn];
18      bool bfs(){
19          memset(ht,0x3f,sizeof(ht));
20          queue<int> q;
21          q.push(T); ht[T]=0;
22          while (q.size()){
23              int u=q.front(); q.pop();
24              for (unsigned int i=0;i<G[u].size();++i){
25                  int v=edge[G[u][i]].v;
26                  if (edge[G[u][i]^1].f&&ht[v]>ht[u]+1){
27                      ht[v]=ht[u]+1; q.push(v);
28                  }
29              }
30          }
31          return ht[S]!=inf;
32      }
33      struct cmp{
34          bool operator () (int a,int b) const {
35              return ht[a]<ht[b];
36          }
37      };
38      priority_queue<int,vector<int>,cmp> pq;
39      bool inq[maxn];
40      int push(int u){
41          for (unsigned int i=0;i<G[u].size();++i){
42              line &e=edge[G[u][i]];
43              int v=e.v; ll w=e.f;
44              if (!w||ht[u]!=ht[v]+1) continue;
45              int k=min(w,ex[u]);
46              ex[u]-=k; ex[v]+=k;
47              e.f-=k; edge[G[u][i]^1].f+=k;
48              if (v!=S&&v!=T&&!inq[v]){
49                  pq.push(v); inq[v]=1;
50              }
51              if (!ex[u]) return 0;
52          }
53          return 1;
54      }
55      void relable(int u){
56          ht[u]=inf;
57          for (unsigned int i=0;i<G[u].size();++i){
58              line e=edge[G[u][i]];
59              if (e.f) ht[u]=min(ht[u],ht[e.v]);
60          }
```

```
61          ++ht[u];
62      }
63      ll hlpp(){
64          if (!bfs()) return 0;
65          ht[S]=n;
66          memset(gap,0,sizeof(gap));
67          for (int i=1;i<=n;++i){
68              if (ht[i]!=inf) ++gap[ht[i]];
69          }
70          for (unsigned int i=0;i<G[S].size();++i){
71              line &e=edge[G[S][i]];
72              int v=e.v; ll w=e.f;
73              if (!w) continue;
74              ex[S]-=w; ex[v]+=w;
75              e.f-=w; edge[G[S][i]^1].f+=w;
76              if (v!=S&&v!=T&&!inq[v]){
77                  pq.push(v);
78              }
79          }
80          while (pq.size()){
81              int u=pq.top();
82              pq.pop(); inq[u]=0;
83              while (push(u)){
84                  if (!--gap[ht[u]]){
85                      for (int i=1;i<=n;++i){
86                          if (i!=S&&i!=T&&ht[i]>ht[u]&&ht[i]<n+1){
87                              ht[i]=n+1;
88                          }
89                      }
90                  }
91                  relable(u); ++gap[ht[u]];
92              }
93          }
94          return ex[T];
95      }
96  }
```

3.13.2 费用流

```
1   struct line{
2       int from,to;
3       int cap,cost;
4   };
5   vector<line> edge;
```

```
vector<int> G[maxn];
void addedge(int from,int to,int cap,int cost){
    edge.push_back((line){from,to,cap,cost});
    edge.push_back((line){to,from,0,-cost});
    int m=edge.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
}
int n,m,s,t,mcost,mflow;
int u,v,w,f;
int dis[maxn],a[maxn],pre[maxn];
bool vis[maxn];
bool spfa(){
    memset(vis,false,sizeof(vis));
    memset(dis,0x3f,sizeof(dis));
    a[s]=2000000000; dis[s]=0;
    vis[s]=true; pre[s]=0;
    queue<int> q; q.push(s);
    while (!q.empty()){
        int u=q.front(); q.pop();
        vis[u]=false;
        for (int i=0;i<G[u].size();++i){
            line e=edge[G[u][i]];
            if (e.cap>0&&dis[e.to]>dis[u]+e.cost){
                dis[e.to]=dis[u]+e.cost;
                pre[e.to]=G[u][i];
                a[e.to]=min(a[u],e.cap);
                if (!vis[e.to]){
                    vis[e.to]=true; q.push(e.to);
                }
            }
        }
    }
    if (dis[t]==dis[maxn-1]) return false;
    mflow+=a[t]; mcost+=dis[t]*a[t];
    int u=t;
    while (u!=s){
        edge[pre[u]].cap-=a[t];
        edge[pre[u]^1].cap+=a[t];
        u=edge[pre[u]].from;
    }
    return true;
}
int main(){
```

```
50      scanf("%d%d%d%d",&n,&m,&s,&t);
51      while (m--){
52          scanf("%d%d%d%d",&u,&v,&w,&f);
53          addedge(u,v,w,f);
54      }
55      while (spfa());
56      printf("%d %d",mflow,mcost);
57      return 0;
58  }
```

3.13.3 上下界网络流

流量下界 $b(u,v)$ ，上界 $c(u,v)$

无源汇上下界可行流 先建立附加源点与附加汇点 S',T' 。对于原图中的 $u \to v$ ，连边流量为 $flow(u,v) = c(u,v) - b(u,v)$

假设一个点 x 初始 $inflow - outflow = M$ 。

- $M = 0$ ，流量平衡不用操作
- $M > 0$ ，出流量过大，$S' \to x$ 流量 M
- $M < 0$ ，入流量过大，$x \to T'$ 流量 $-M$

建新图完后跑 $S' \to T'$ 最大流，若 S' 连出去的边全满流，则存在可行流。此时原图中每一条边的流量为新图的流量 $+b(u,v)$

```
1   #include <iostream>
2   #include <cstdio>
3   #include <algorithm>
4   #include <cstring>
5   #include <cmath>
6   #include <queue>
7   #include <vector>
8   using namespace std;
9   #define ll long long
10  #define il inline
11  const int maxn = 210;
12  const int maxm = 11000;
13  const ll INF = 0x3f3f3f3f3f3f3f3f;
14  struct line{
15      int to;
16      ll cap, flow;
17  };
18  vector<line> edge;
```

```
19  vector<int> G[maxn];
20  void addedge(int from, int to, int cap) {
21      edge.push_back((line){to, cap, 0});
22      edge.push_back((line){from, 0, 0});
23      int m = edge.size();
24      G[from].push_back(m - 2);
25      G[to].push_back(m - 1);
26  }
27  int n, m, S, T, SS, TT;
28  ll dflow[maxn], totflow;
29  ll minflow[maxm];
30  int dep[maxn], vis[maxn], cur[maxn];
31  bool bfs(int tmpS, int tmpT) {
32      memset(vis, 0, sizeof(vis));
33      dep[tmpS] = 0; vis[tmpS] = 1;
34      queue<int> q; q.push(tmpS);
35      while (!q.empty()) {
36          int u = q.front(); q.pop();
37          for (int i = 0; i < G[u].size(); ++i) {
38              line &e = edge[G[u][i]];
39              if (!vis[e.to] && e.cap > e.flow) {
40                  dep[e.to] = dep[u] + 1;
41                  vis[e.to] = 1;
42                  q.push(e.to);
43              }
44          }
45      }
46      return vis[tmpT];
47  }
48  ll dfs(int x, ll a, int tmpT) {
49      if (x == tmpT || !a) return a;
50      ll flow = 0, f;
51      for (int &i = cur[x]; i < G[x].size(); ++i) {
52          line &e = edge[G[x][i]];
53          if (dep[e.to] == dep[x] + 1 &&
54              (f = dfs(e.to, min(e.cap - e.flow, a), tmpT)) > 0)
                ↪ {
55                  flow += f; a -= f;
56                  e.flow += f;
57                  edge[G[x][i] ^ 1].flow -= f;
58                  if (!a) break;
59              }
60      }
61      return flow;
```

```
}
ll dinic(int tmpS, int tmpT) {
    ll flow = 0;
    while (bfs(tmpS, tmpT)) {
        memset(cur, 0, sizeof(cur));
        flow += dfs(tmpS, INF, tmpT);
    }
    return flow;
}
int main() {
    scanf("%d%d", &n, &m);
    SS = n + 1; TT = n + 2;
    int ru, rv; ll rr;
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d%lld%lld", &ru, &rv, &minflow[i], &rr);
        addedge(ru, rv, rr - minflow[i]);
        dflow[ru] -= minflow[i];
        dflow[rv] += minflow[i];
    }
    for (int i = 1; i <= n; ++i) {
        if (!dflow[i]) continue;
        if (dflow[i] > 0) {
            addedge(SS, i, dflow[i]);
            totflow += dflow[i];
        } else {
            addedge(i, TT, -dflow[i]);
        }
    }
    if (totflow == dinic(SS, TT)) {
        printf("YES\n");
        for (int i = 1; i <= m; ++i) {
            printf("%lld\n", edge[(i - 1) * 2].flow +
            ↪  minflow[i]);
        }
    } else {
        printf("NO");
    }
    return 0;
}
```

有源汇上下界可行流 加入一条 $T \to S$ 上界为 ∞，下界为 0 的边，即让源汇点满足流量平衡。转化成无源汇问题。

若有解，$S \to T$ 的可行流流量等于 $T \to S$ 的附加边流量。

有源汇上下界最大流 先找一个可行流，找不到直接结束。

我们删去所有的附加边附加点，在**残量网络**上跑一次 $S \to T$ 最大流。(其实只需要删除 S', T' ，因为附加边已经满流了，我们直接在原图上跑一次 $S \to T$ 最大流即可）答案为可行流 + 残量最大流。

```
struct line{
    int to;
    ll cap,flow;
};
vector<line> edge;
vector<int> G[maxn];
inline void addedge(int from,int to,int cap){
    edge.push_back((line){to,cap,0});
    edge.push_back((line){from,0,0});
    int m=edge.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
}
int n,m,S,T,SS,TT;
int ru,rv; ll rl,rr;
ll dflow[maxn],totflow;
int vis[maxn],dep[maxn],cur[maxn];
bool bfs(int tmps,int tmpt){
    memset(vis,0,sizeof(vis));
    dep[tmps]=0; vis[tmps]=1;
    queue<int> q; q.push(tmps);
    while (!q.empty()){
        int x=q.front(); q.pop();
        for (int i=0;i<G[x].size();++i){
            line &e=edge[G[x][i]];
            if (!vis[e.to]&&e.cap>e.flow){
                dep[e.to]=dep[x]+1;
                vis[e.to]=1; q.push(e.to);
            }
        }
    }
    return vis[tmpt];
}
ll dfs(int x,ll a,int tmpt){
    if (x==tmpt||!a) return a;
    ll flow=0,f;
    for (int &i=cur[x];i<G[x].size();++i){
        line &e=edge[G[x][i]];
        if (dep[e.to]==dep[x]+1&&
```

```
            (f=dfs(e.to,min(e.cap-e.flow,a),tmpt))>0){
            flow+=f; a-=f;
            e.flow+=f;
            edge[G[x][i]^1].flow-=f;
            if (!a) break;
        }
    }
    return flow;
}
ll dinic(int tmps,int tmpt){
    ll flow=0;
    while (bfs(tmps,tmpt)){
        memset(cur,0,sizeof(cur));
        flow+=dfs(tmps,inf,tmpt);
    }
    return flow;
}
int main(){
    scanf("%d%d%d%d",&n,&m,&S,&T);
    SS=n+1; TT=n+2;
    for (int i=1;i<=m;++i){
        scanf("%d%d%lld%lld",&ru,&rv,&rl,&rr);
        addedge(ru,rv,rr-rl);
        dflow[ru]-=rl; dflow[rv]+=rl;
    }
    addedge(T,S,inf);
    for (int i=1;i<=n;++i){
        if (!dflow[i]) continue;
        if (dflow[i]>0){
            addedge(SS,i,dflow[i]);
            totflow+=dflow[i];
        }else{
            addedge(i,TT,-dflow[i]);
        }
    }
    if (totflow==dinic(SS,TT)){
        printf("%lld",dinic(S,T));
    }else{
        printf("please go home to sleep"); //no answer
    }
}
```

有源汇上下界最小流 先找一个可行流，找不到直接结束。

我们删去所有的附加边，在**残量网络**上跑一次 $T \to S$ 最大流。答案是可行流-残量最大流。

注意与有源汇上下界最大流的**方向相反**。

```
//just different here
if (totflow==dinic(SS,TT)){
    ll sum=edge[G[T][G[T].size()-1]].flow;
    G[S].pop_back(); G[T].pop_back();
    printf("%lld",sum-dinic(T,S)); //dont't write S,T
}else{
    printf("please go home to sleep");
}
```

上下界最小费用流的转化 1. 无源汇最小费用可行流同无源汇可行流建边。所有附加边费用为 0 。直接跑费用流即可。

2. 有源汇最小费用可行流与有源汇可行流建边类似。之后连边 $t \to s, c = 0, cap = +\infty$ 。跑超级源点到超级汇点的最小费用最大流，答案为求出的费用与原图中边的下界乘边的费用的和。

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cmath>
#include <vector>
#include <queue>
using namespace std;
#define ll long long
#define il inline
const int maxn = 310;
const int INF = 0x3f3f3f3f;
struct line {
    int from, to;
    int cap, cost;
};
vector<line> edge;
vector<int> G[maxn];
void addedge(int from, int to, int cap, int cost) {
    edge.push_back((line){from, to, cap, cost});
    edge.push_back((line){to, from, 0, -cost});
    int m = edge.size();
    G[from].push_back(m - 2);
    G[to].push_back(m - 1);
}
```

```
26 int dis[maxn], a[maxn], pre[maxn];
27 bool vis[maxn];
28 int mflow, mcost;
29 bool spfa(int tmpS, int tmpT) {
30     memset(vis, 0, sizeof(vis));
31     memset(dis, 0x3f, sizeof(dis));
32     a[tmpS] = INF; dis[tmpS] = 0;
33     vis[tmpS] = true; pre[tmpS] = 0;
34     queue<int> q; q.push(tmpS);
35     while (!q.empty()) {
36         int u = q.front(); q.pop();
37         vis[u] = false;
38         for (int i = 0; i < G[u].size(); ++i) {
39             line e = edge[G[u][i]];
40             if (e.cap > 0 && dis[e.to] > dis[u] + e.cost) {
41                 dis[e.to] = dis[u] + e.cost;
42                 pre[e.to] = G[u][i];
43                 a[e.to] = min(a[u], e.cap);
44                 if (!vis[e.to]) {
45                     vis[e.to] = true;
46                     q.push(e.to);
47                 }
48             }
49         }
50     }
51     if (dis[tmpT] == INF) return false;
52     mflow += a[tmpT];
53     mcost += dis[tmpT] * a[tmpT];
54     int u = tmpT;
55     while (u != tmpS) {
56         edge[pre[u]].cap -= a[tmpT];
57         edge[pre[u] ^ 1].cap += a[tmpT];
58         u = edge[pre[u]].from;
59     }
60     return true;
61 }
62 int n, S, T, SS, TT;
63 int dflow[maxn];
64 int totcost;
65 int main() {
66     scanf("%d", &n);
67     S = 1; T = n + 1;
68     SS = T + 1; TT = SS + 1;
69     int rk, rb, rt;
```

```
70      for (int i = 1; i <= n; ++i) {
71          scanf("%d", &rk);
72          while (rk--) {
73              scanf("%d%d", &rb, &rt);
74              addedge(i, rb, INF, rt);
75              --dflow[i]; ++dflow[rb];
76              // dflow is always 1 this place
77              // change dflow when necessary
78              totcost += rt * 1;
79              // totcost = lowestflow * cost;
80              // change when necessary
81          }
82      }
83      for (int i = 1; i <= n; ++i) {
84          addedge(i, T, INF, 0);
85      }
86      addedge(T, S, INF, 0);
87      for (int i = 1; i <= n + 1; ++i) {
88          if (!dflow[i]) continue;
89          if (dflow[i] > 0) {
90              addedge(SS, i, dflow[i], 0);
91          } else {
92              addedge(i, TT, -dflow[i], 0);
93          }
94      }
95      while (spfa(SS, TT));
96      totcost += mcost;
97      printf("%d", totcost);
98      return 0;
99  }
```

3.13.4 模型与应用

最大权闭合图 闭合图：任意点的任意后继一定也在闭合图中。反应出一定的必要条件关系。

最大权闭合图：每个点都有一个点权 w ，选出闭合图使得 $\sum_{v\in V} w_v$ 最大化。

构造：

1. 通过关联关系构造闭合图。若选 x 后必定选 y ，则加边 $x \to y$ 。流量 INF。
2. 根据点权值正负，构造超级源点汇点 S,T 。由 S 向权值为正的点连边，权值为负的向 T 连边，流量为绝对值。
3. 最大权值为所有正权值之和减去最小割。

3.14 Prufer 序列

摘自OI-Wiki

Prufer 序列可以将一个带标号 n 个结点的树用 $[1,n]$ 中的 $n-2$ 个整数表示。实现完全图的生成树与数列之间的双射（一一对应）。

3.14.1 Prufer 序列的构造

```cpp
//start from zero!
vector<vector<int>> adj;
vector<int> parent;
void dfs(int v) {
  for (int u : adj[v]) {
    if (u != parent[v]) parent[u] = v, dfs(u);
  }
}
vector<int> pruefer_code() {
  int n = adj.size();
  parent.resize(n), parent[n - 1] = -1;
  dfs(n - 1);
  int ptr = -1;
  vector<int> degree(n);
  for (int i = 0; i < n; i++) {
    degree[i] = adj[i].size();
    if (degree[i] == 1 && ptr == -1) ptr = i;
  }
  vector<int> code(n - 2);
  int leaf = ptr;
  for (int i = 0; i < n - 2; i++) {
    int next = parent[leaf];
    code[i] = next;
    if (--degree[next] == 1 && next < ptr) {
      leaf = next;
    } else {
      ptr++;
      while (degree[ptr] != 1) ptr++;
      leaf = ptr;
    }
  }
  return code;
}
```

3.14.2 Prufer 序列重构树

```cpp
vector<pair<int, int>> pruefer_decode(vector<int> const& code) {
  int n = code.size() + 2;
  vector<int> degree(n, 1);
  for (int i : code) degree[i]++;
  int ptr = 0;
  while (degree[ptr] != 1) ptr++;
  int leaf = ptr;
  vector<pair<int, int>> edges;
  for (int v : code) {
    edges.emplace_back(leaf, v);
    if (--degree[v] == 1 && v < ptr) {
      leaf = v;
    } else {
      ptr++;
      while (degree[ptr] != 1) ptr++;
      leaf = ptr;
    }
  }
  edges.emplace_back(leaf, n - 1);
  return edges;
}
```

3.14.3 Prufer 序列的应用

(1) **Cayley 定理：**完全图 K_n 有 n^{n-2} 棵生成树（当然无根树）。(任意一个长度为 $n-2$，值域为 $[1,n]$ 的整数序列都可以通过 Prufer 序列双射对应一个生成树。) (2) 图联通方案数：一个 n 个点 m 条边的带标号无向图有 k 个连通块。我们希望添加 $k-1$ 条边使得整个图连通。求方案数。设 s_i 表示每个连通块的数量。我们对 k 个连通块构造 Prufer 序列。设 d_i 为第 i 个连通块的度数。由于度数之和是边数的两倍，于是 $\sum_{i=1}^{k} d_i = 2k-2$。则对于给定的 d 序列构造 Prufer 序列的方案数是

$$\binom{k-2}{d_1-1, d_2-1, \cdots, d_k-1} = \frac{(k-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$$

对于第 i 个连通块，它的连接方式有 $s_i^{d_i}$ 种，因此对于给定 d 序列使图连通的方案数是

$$\binom{k-2}{d_1-1, d_2-1, \cdots, d_k-1} \cdot \prod_{i=1}^{k} {s_i}^{d_i}$$

现在我们要枚举 d 序列，式子变成

$$\sum_{d_i\geq 1\ \sum_{i=1}^{k} d_i=2k-2} \binom{k-2}{d_1-1,d_2-1,\cdots,d_k-1} \cdot \prod_{i=1}^{k} {s_i}^{d_i}$$

用二项式定理和换元，设 $e_i = d_i - 1$ ，则 $\sum_{i=1}^{k} e_i = k-2$ 。

$$\sum_{e_i\geq 0\ \sum_{i=1}^{k} e_i=k-2} \binom{k-2}{e_1,e_2,\cdots,e_k} \cdot \prod_{i=1}^{k} {s_i}^{e_i+1}$$

化简得

$$(s_1+s_2+\cdots+s_k)^{k-2} \cdot \prod_{i=1}^{k} s_i = n^{k-2} \cdot \prod_{i=1}^{k} s_i$$

3.15 欧拉图与哈密顿图

欧拉图针对的是所有的边，而哈密顿图针对的是所有的点。特定情况下哈密顿图可以转化为欧拉图判断。

注意判断连通性

3.15.1 欧拉图

欧拉通路：通过图中所有**边**恰好一次且行遍所有顶点的通路。

欧拉回路：通过图中所有**边**恰好一次且行遍所有顶点的**回路**。

欧拉图：具有欧拉回路的**无向图**。

半欧拉图：具有欧拉通路但不具有欧拉回路的**无向图**。

无向图判断：G 是欧拉图当且仅当 G **连通**且没有奇度点。G 是半欧拉图当且仅当 G **连通**且恰有两个奇度点。

有向图判断：G 中所有结点入度 = 出度则存在欧拉回路。G 中仅有两个点，一个入度 = 出度 +1，另一个入度 = 出度-1，其他入度 = 出度，则为欧拉通路。

混合图：摘自nextbin——图论——欧拉通路、欧拉回路（有向图无向图混合图）

存在**欧拉回路**的条件：

1. 将无向边随便定向，每个结点的入度与出度之差为偶数。

2. 进行网络流，若满流，则存在欧拉回路。

网络流构图：忽略有向边。对于随便定向的无向边，按照所定向连边，流量为 1 。对于入度大于出度的点 u，连边 $(u\ T)$，流量为 $abs((in[u]-out[u])/2)$。对于出度大于入度的点 v，连边 $(S\ v)$，流量为 $abs((in[v]-out[v])/2)$。

存在**欧拉通路**的条件：存在欧拉回路，或

1. 将无向边随便定向，有两个结点的入度与出度之差为奇数，其他结点的入度与出度之差为偶数。

2. 在两个奇度结点间连一条无向边，进行网络流，若满流，则存在欧拉通路。

网络流构图：同上。

求欧拉回路或欧拉路：(摘自OI Wiki——欧拉图)

Hierholzer 算法：算法流程为从一条回路开始，每次任取一条目前回路中的点，将其替换为一条简单回路，以此寻找到一条欧拉回路。如果从路开始的话，就可以寻找到一条欧拉路。时间复杂度：如果要求字典序最小则是 $\Theta(n+m\ log\ m)$。如果不用排序则是 $\Theta(n+m)$。存图推荐 **vector** 。

```cpp
#include <algorithm>
#include <cstdio>
#include <stack>
#include <vector>
using namespace std;

struct edge {
  int to;
  bool exists;
  int revref;

  bool operator<(const edge& b) const { return to < b.to; }
};

vector<edge> beg[505];
int cnt[505];

const int dn = 500;
stack<int> ans;

void Hierholzer(int x)  // 关键函数
{
  for (int& i = cnt[x]; i < (int)beg[x].size();) {
    if (beg[x][i].exists) {
      edge e = beg[x][i];
      beg[x][i].exists = 0;
      beg[e.to][e.revref].exists = 0;
      ++i;
      Hierholzer(e.to);
    } else {
      ++i;
    }
  }
  ans.push(x);
}

int deg[505];
```

```
38 int reftop[505];
39
40 int main() {
41   for (int i = 1; i <= dn; ++i) {
42     beg[i].reserve(1050);  // vector 用 reserve 避免动态分配空间，加快速度
43   }
44
45   int m;
46   scanf("%d", &m);
47   for (int i = 1; i <= m; ++i) {
48     int a, b;
49     scanf("%d%d", &a, &b);
50     beg[a].push_back((edge){b, 1, 0});
51     beg[b].push_back((edge){a, 1, 0});
52     ++deg[a];
53     ++deg[b];
54   }
55
56   for (int i = 1; i <= dn; ++i) {
57     if (!beg[i].empty()) {
58       sort(beg[i].begin(), beg[i].end());  // 为了要按字典序贪心，必须排序
59     }
60   }
61
62   for (int i = 1; i <= dn; ++i) {
63     for (int j = 0; j < (int)beg[i].size(); ++j) {
64       beg[i][j].revref = reftop[beg[i][j].to]++;
65     }
66   }
67
68   int bv = 0;
69   for (int i = 1; i <= dn; ++i) {
70     if (!deg[bv] && deg[i]) {
71       bv = i;
72     } else if (!(deg[bv] & 1) && (deg[i] & 1)) {
73       bv = i;
74     }
75   }
76
77   Hierholzer(bv);
78
79   while (!ans.empty()) {
80     printf("%d\n", ans.top());
81     ans.pop();
```

```
82    }
83  }
```

3.15.2 哈密顿图

哈密顿通路：通过图中所有**顶点**一次且仅一次的通路。

哈密顿回路：通过图中所有**顶点**一次且仅一次的回路。

判断：充要条件无解。我们只有充分条件：

(1) 若图的最小度不小于顶点数的一半，则图是哈密顿图；

(2) 若图中每一对不相邻的顶点的度数之和不小于顶点数，则图是哈密顿图。

4 数据结构

4.1 莫队

4.1.1 普通莫队

长度为 n 的数组，一共 m 次询问。复杂度 $O(n\sqrt{m})$ 。我们设块的大小为 S，则复杂度为 $\frac{n^2}{S}+mS$，则 $S=\frac{n}{\sqrt{m}}$。同时采用奇偶性排序：

```
bool cmp(node a, node b) {
    return pos[a.l] ^ pos[b.l] ?
        pos[a.l] < pos[b.l] : pos[a.l] & 1 ?
            a.r < b.r : a.r > b.r;
}
```

注：初始状态 $l=1,r=0$ 代表空集。当 $l>r+1$ 的时候，有些元素被多删除了一次但没有加入，这种情况有时会产生错误（如用 set 维护区间中数的时候，这样会导致删除一个不存在的元素），其中只有 6 种写法是正确的。一般推荐操作是前两步扩大空间（- -l, ++r），后两步缩小空间（++l, - -r）。

一个例子：求解 $\sum_{i=1}^{k} {c_i}^2$ ，其中 c_i 表示数字 i 在 l 到 r 中的出现次数。设 $cnt(x)$ 为 x 的出现次数，则 x 的个数 $cnt(x)$ 变成 $cnt(x)+1$ 时答案增加了 $2x+1$ ，减法类似。

```
const int maxn = 50010;
struct Q {
    int l, r, id;
} q[maxn];
int n, m, k;
ll ans, ans_array[maxn];
int block;
bool cmp(const Q a, const Q b) {
    return (a.l / block) ^ (b.l / block) ?
        a.l < b.l : (a.l / block) & 1 ?
            a.r < b.r : a.r > b.r;
}
int a[maxn], cnt[maxn];
il void add(int x) {
    ans += 2 * cnt[x] + 1;
    ++cnt[x];
}
il void del(int x) {
    ans -= 2 * cnt[x] - 1;
    --cnt[x];
}
int main() {
    scanf("%d%d%d", &n, &m, &k);
```

```
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d", &q[i].l, &q[i].r);
        q[i].id = i;
    }
    block = n / sqrt(m * 2 / 3);
    sort(q + 1, q + 1 + m, cmp);
    int l = 1, r = 0;
    for (int i = 1; i <= m; ++i) {
        int ql = q[i].l, qr = q[i].r;
        while (l > ql) add(a[--l]);
        while (r < qr) add(a[++r]);
        while (l < ql) del(a[l++]);
        while (r > qr) del(a[r--]);
        ans_array[q[i].id] = ans;
    }
    for (int i = 1; i <= m; ++i) {
        printf("%lld\n", ans_array[i]);
    }
    return 0;
}
```

4.1.2 带修改莫队

加上一维时间维即可。分块大小取 $S = n^{\frac{2}{3}}$，最终复杂度 $n^{\frac{5}{3}}$。

```
#include <bits/stdc++.h>
#define SZ (10005)
using namespace std;
template <typename _Tp>
inline void IN(_Tp& dig) {
  char c;
  dig = 0;
  while (c = getchar(), !isdigit(c))
    ;
  while (isdigit(c)) dig = dig * 10 + c - '0', c = getchar();
}
int n, m, sqn, c[SZ], ct[SZ], c1, c2;
int mem[SZ][3], ans, tot[1000005], nal[SZ];
struct query {
  int l, r, i, c;
  bool operator<(const query another) const {
    if (l / sqn == another.l / sqn) {
```

```
18        if (r / sqn == another.r / sqn) return i < another.i;
19        return r < another.r;
20      }
21      return l < another.l;
22    }
23  } Q[SZ];
24  void add(int a) {
25    if (!tot[a]) ans++;
26    tot[a]++;
27  }
28  void del(int a) {
29    tot[a]--;
30    if (!tot[a]) ans--;
31  }
32  char opt[10];
33  int main() {
34    IN(n), IN(m), sqn = pow(n, (double)2 / (double)3);
35    for (int i = 1; i <= n; i++) IN(c[i]), ct[i] = c[i];
36    for (int i = 1, a, b; i <= m; i++)
37      if (scanf("%s", opt), IN(a), IN(b), opt[0] == 'Q')
38        Q[c1].l = a, Q[c1].r = b,
39        Q[c1].i = c1, Q[c1].c = c2, c1++;
40      else
41        mem[c2][0] = a, mem[c2][1] = ct[a],
42        mem[c2][2] = ct[a] = b, c2++;
43    sort(Q, Q + c1), add(c[1]);
44    int l = 1, r = 1, lst = 0;
45    for (int i = 0; i < c1; i++) {
46      for (; lst < Q[i].c; lst++) {
47        if (l <= mem[lst][0] && mem[lst][0] <= r)
48          del(mem[lst][1]), add(mem[lst][2]);
49        c[mem[lst][0]] = mem[lst][2];
50      }
51      for (; lst > Q[i].c; lst--) {
52        if (l <= mem[lst - 1][0] && mem[lst - 1][0] <= r)
53          del(mem[lst - 1][2]), add(mem[lst - 1][1]);
54        c[mem[lst - 1][0]] = mem[lst - 1][1];
55      }
56      for (++r; r <= Q[i].r; r++) add(c[r]);
57      for (--r; r > Q[i].r; r--) del(c[r]);
58      for (--l; l >= Q[i].l; l--) add(c[l]);
59      for (++l; l < Q[i].l; l++) del(c[l]);
60      nal[Q[i].i] = ans;
61    }
```

```
62    for (int i = 0; i < c1; i++) printf("%d\n", nal[i]);
63    return 0;
64  }
```

4.2 单调栈与单调队列

单调栈：保证栈内元素单调。

单调队列：队列内元素单调。可以解决找出长度为 k 的所有区间的最大值最小值问题。

4.3 树状数组

部分代码摘自胡小兔——“高级” 数据结构——树状数组。

4.3.1 单点修改区间查询

- (1) 数组转化为差分数组
- (2) 差分数组前缀和即为当前点的值
- (3) 区间修改只需单点修改差分数组的两端

```
int lowbit(int x){
    return x&(-x);
}
void add_val(int pos,int val){
    while (pos<=n){
        tree[pos]+=val;
        pos+=lowbit(pos);
    }
}
int get_sum(int pos){
    int tmp=0;
    while (pos>0){
        tmp+=tree[pos];
        pos-=lowbit(pos);
    }
    return tmp;
}
```

4.3.2 区间修改区间查询

继续利用差分（数组 d ）的思路，我们区间查询利用前缀和。假设查 p 的位置，则 $\sum_{i=1}^{p} a[i] = \sum_{i=1}^{p} \sum_{j=1}^{i} d[j]$ ，拆一下式子，得到 $\sum_{i=1}^{p} a[i] = (p+1) \sum_{i=1}^{p} d[i] -$

$\sum_{i=1}^{p} d[i]*i$ 。所以维护两个数组的前缀和 $sum1[i] \to d[i], sum2[i] \to d[i]*i$ 。修改的话与上文差分思路相似。

```
void add(ll p,ll x){
    for(int i=p;i<=n;i+=i&-i)
        sum1[i]+=x,sum2[i]+=x*p;
}
void range_add(ll l,ll r,ll x){
    add(l,x), add(r+1,-x);
}
ll ask(ll p){
    ll res=0;
    for(int i=p;i;i-=i&-i)
        res+=(p+1)*sum1[i]-sum2[i];
    return res;
}
ll range_ask(ll l,ll r){
    return ask(r)-ask(l-1);
}
```

4.3.3 进阶：二维树状数组

单点修改单点查询 $tree[x][y]$ 相当于下端点 x ，右端点 y ，往上 $lowbit(x)$ ，往左 $lowbit(y)$ 的区间和。

```
void add(int x,int y,int z){ //add z to (x,y)
    int memo_y=y;
    while(x<=n){
        y=memo_y;
        while(y<=n)
            tree[x][y]+=z,y+=y&-y;
        x+=x&-x;
    }
}
void ask(int x,int y){ //calc sum of (1,1) to (x,y)
    int res=0,memo_y=y;
    while(x){
        y=memo_y;
        while(y)
            res+=tree[x][y],y-=y&-y;
        x-=x&-x;
    }
}
```

区间修改单点查询 根据二维前缀和：$d[i][j]=a[i][j]-(a[i-1][j]+a[i][j-1]-a[i-1][j-1])$。

```
void add(int x,int y,int z){
    int memo_y=y;
    while(x<=n){
        y=memo_y;
        while(y<=n)
            tree[x][y]+= z,y+=y&-y;
        x+=x&-x;
    }
}
void range_add(int xa,int ya,int xb,int yb,int z){
    add(xa,ya,z);
    add(xa,yb+1,-z);
    add(xb+1,ya,-z);
    add(xb+1,yb+1,z);
}
void ask(int x,int y){
    int res=0,memo_y=y;
    while(x){
        y=memo_y;
        while(y)
            res+=tree[x][y],y-=y&-y;
        x-=x&-x;
    }
}
```

区间修改区间查询

$$\begin{aligned}\sum_{i=1}^{x}\sum_{j=1}^{y}\sum_{k=1}^{i}\sum_{h=1}^{j}d[h][k]=&(x+1)*(y+1)*\sum_{i=1}^{x}\sum_{j=1}^{y}d[i][j]\\&-(y+1)*\sum_{i=1}^{x}\sum_{j=1}^{y}d[i][j]*i\\&-(x+1)*\sum_{i=1}^{x}\sum_{j=1}^{y}d[i][j]*j\\&+\sum_{i=1}^{x}\sum_{j=1}^{y}d[i][j]*i*j\end{aligned}\tag{2}$$

开四个数组分别维护以上东西即可。

```
ll t1[N][N],t2[N][N],t3[N][N],t4[N][N];
void add(ll x,ll y,ll z){
```

```
    for(int X=x;X<=n;X+=X&-X)
        for(int Y=y; Y<=m;Y+=Y&-Y){
            t1[X][Y]+=z;
            t2[X][Y]+=z*x;
            t3[X][Y]+=z*y;
            t4[X][Y]+=z*x*y;
        }
}
ll ask(ll x,ll y){
    ll res=0;
    for(int i=x;i;i-=i&-i)
        for(int j=y;j;j-=j&-j)
            res+=(x+1)*(y+1)*t1[i][j]
               -(y+1)*t2[i][j]
               -(x+1)*t3[i][j]
               +t4[i][j];
    return res;
}
ll range_ask(ll xa,ll ya,ll xb,ll yb){
    return ask(xb,yb)-ask(xb,ya-1)
          -ask(xa-1,yb)+ask(xa-1,ya-1);
}
```

4.4 线段树

四倍空间提醒！四倍空间提醒！四倍空间提醒！

解决区间能够合并的问题，比如 $min(l\dots r), max(l\dots r), sum(l\dots r), gcd(l\cdots r), \cdots$

这里放一个区间加乘，区间求和的板子。

```
#define ls o<<1
#define rs ls|1
struct treenode{
    ll sum;
    ll lazy_add,lazy_mul;
}sgt[maxn<<2];
void sgt_build(int o,int l,int r){
    sgt[o].lazy_mul=1;
    if (l==r){
        sgt[o].sum=a[l];
        return;
    }
    int mid=(l+r)>>1;
    sgt_build(ls,l,mid); sgt_build(rs,mid+1,r);
    sgt[o].sum=sgt[ls].sum+sgt[rs].sum;
```

```
16 }
17 void sgt_pushdown(int o,int l,int r){
18     //pushdown when query or modify
19     //frist mul then add
20     sgt[ls].lazy_mul=(sgt[ls].lazy_mul*sgt[o].lazy_mul)%p;
21     sgt[rs].lazy_mul=(sgt[rs].lazy_mul*sgt[o].lazy_mul)%p;
22     sgt[ls].lazy_add=(sgt[ls].lazy_add*sgt[o].lazy_mul)%p;
23     sgt[rs].lazy_add=(sgt[rs].lazy_add*sgt[o].lazy_mul)%p;
24     sgt[ls].sum=(sgt[ls].sum*sgt[o].lazy_mul)%p;
25     sgt[rs].sum=(sgt[rs].sum*sgt[o].lazy_mul)%p;
26     sgt[o].lazy_mul=1;
27     int mid=(l+r)>>1;
28     sgt[ls].lazy_add=(sgt[ls].lazy_add+sgt[o].lazy_add)%p;
29     sgt[rs].lazy_add=(sgt[rs].lazy_add+sgt[o].lazy_add)%p;
30     sgt[ls].sum=(sgt[ls].sum+(mid-l+1)*sgt[o].lazy_add)%p;
31     sgt[rs].sum=(sgt[rs].sum+(r-mid)*sgt[o].lazy_add)%p;
32     sgt[o].lazy_add=0;
33 }
34 void sgt_mul(int o,int l,int r,int ql,int qr,ll k){
35     if (l>=ql&&r<=qr){
36         sgt[o].lazy_add=(sgt[o].lazy_add*k)%p;
37         sgt[o].lazy_mul=(sgt[o].lazy_mul*k)%p;
38         sgt[o].sum=(sgt[o].sum*k)%p;
39         return;
40     }
41     int mid=(l+r)>>1;
42     if (sgt[o].lazy_mul!=1||sgt[o].lazy_add) sgt_pushdown(o,l,r);
43     if (ql<=mid) sgt_mul(ls,l,mid,ql,qr,k);
44     if (qr>mid) sgt_mul(rs,mid+1,r,ql,qr,k);
45     sgt[o].sum=(sgt[ls].sum+sgt[rs].sum)%p;
46 }
47 void sgt_add(int o,int l,int r,int ql,int qr,ll k){
48     if (l>=ql&&r<=qr){
49         sgt[o].lazy_add=(sgt[o].lazy_add+k)%p;
50         sgt[o].sum=(sgt[o].sum+(r-l+1)*k)%p;
51         return;
52     }
53     int mid=(l+r)>>1;
54     if (sgt[o].lazy_mul!=1||sgt[o].lazy_add) sgt_pushdown(o,l,r);
55     if (ql<=mid) sgt_add(ls,l,mid,ql,qr,k);
56     if (qr>mid) sgt_add(rs,mid+1,r,ql,qr,k);
57     sgt[o].sum=(sgt[ls].sum+sgt[rs].sum)%p;
58 }
59 ll sgt_querysum(int o,int l,int r,int ql,int qr){
```

```
60      if (l>=ql&&r<=qr){
61          return sgt[o].sum;
62      }
63      int mid=(l+r)>>1;
64      if (sgt[o].lazy_add||sgt[o].lazy_mul!=1) sgt_pushdown(o,l,r);
65      ll tmp_sum=0;
66      if (ql<=mid) tmp_sum=(tmp_sum+sgt_querysum(ls,l,mid,ql,qr))%p;
67      if (qr>mid) tmp_sum=(tmp_sum+sgt_querysum(rs,mid+1,r,ql,qr))%p;
68      return tmp_sum%p;
69  }
```

4.5 Splay

(1) 查询排名，查询前驱后继等平衡树固有属性。

(2) 序列相关操作，比如区间翻转，元素变换位置等。区间翻转 $[l,r]$：将 $l-1$ 旋转到根，$r+1$ 旋转到根的右儿子，那么 $r+1$ 的子树则是旋转区间。将每一个区间打标记，回头旋转即可。**记得哨兵节点**

4.5.1 基本操作

```
1   struct treenode{
2       int ch[2],fa;
3       int val,sz;
4       int cnt;
5   }splay[maxn];
6   int root,tot;
7   void splay_maintain(int x){
8       splay[x].sz=splay[splay[x].ch[0]].sz
9                   +splay[splay[x].ch[1]].sz
10                  +splay[x].cnt;
11  }
12  int splay_getchild(int x){
13      return x==splay[splay[x].fa].ch[1];
14  }
15  void splay_clear(int x){
16      splay[x].ch[0]=splay[x].ch[1]=splay[x].fa=0;
17      splay[x].val=splay[x].sz=0;
18      splay[x].cnt=0;
19  }
20  void splay_rotate(int x){
21      int y=splay[x].fa,z=splay[y].fa,chk=splay_getchild(x);
22      splay[y].ch[chk]=splay[x].ch[chk^1];
23      splay[splay[x].ch[chk^1]].fa=y;
24      splay[x].ch[chk^1]=y;
```

```
25      splay[y].fa=x; splay[x].fa=z;
26      if (z) splay[z].ch[y==splay[z].ch[1]]=x;
27      splay_maintain(x); splay_maintain(y);
28  }
29  void splay_splay(int x,int goal){
30      for (int f;(f=splay[x].fa)!=goal;splay_rotate(x)){
31          if (splay[f].fa!=goal)
32              splay_rotate(splay_getchild(x)==splay_getchild(f)?f:x);
33      }
34      if (!goal) root=x;
35  }
36  void splay_insert(int k){
37      if (!root){
38          splay[++tot].val=k; splay[tot].cnt++;
39          root=tot; splay_maintain(root);
40          return;
41      }
42      int cur=root,f=0;
43      while (1){
44          if (splay[cur].val==k){
45              splay[cur].cnt++;
46              splay_maintain(cur); splay_maintain(f);
47              splay_splay(cur,0);
48              break;
49          }
50          f=cur; cur=splay[cur].ch[splay[cur].val<k];
51          if (!cur){
52              splay[++tot].val=k; splay[tot].cnt++;
53              splay[tot].fa=f; splay[f].ch[splay[f].val<k]=tot;
54              splay_maintain(tot); splay_maintain(f);
55              splay_splay(tot,0); break;
56          }
57      }
58  }
59  int splay_findrank(int k){
60      int res=0,cur=root;
61      while (1){
62          if (k<splay[cur].val){
63              cur=splay[cur].ch[0];
64          }else{
65              res+=splay[splay[cur].ch[0]].sz;
66              if (k==splay[cur].val){
67                  splay_splay(cur,0); return res+1;
68              }
```

```
69              res+=splay[cur].cnt;
70              cur=splay[cur].ch[1];
71          }
72      }
73  }
74  int splay_findvalue(int k){
75      int cur=root;
76      while (1){
77          if (splay[cur].ch[0]&&k<=splay[splay[cur].ch[0]].sz){
78              cur=splay[cur].ch[0];
79          }else{
80              k-=splay[cur].cnt+splay[splay[cur].ch[0]].sz;
81              if (k<=0) return splay[cur].val;
82              /*
83              this just return the value
84              if want's the id in splay just return cur
85              */
86              cur=splay[cur].ch[1];
87          }
88      }
89  }
90  int splay_prenumber(){
91      int cur=splay[root].ch[0];
92      while (splay[cur].ch[1]) cur=splay[cur].ch[1];
93      return cur;
94  }
95  int splay_nxtnumber(){
96      int cur=splay[root].ch[1];
97      while (splay[cur].ch[0]) cur=splay[cur].ch[0];
98      return cur;
99  }
100 void splay_delete(int k){
101     splay_findrank(k);
102     if (splay[root].cnt>1){
103         splay[root].cnt--; splay_maintain(root);
104         return;
105     }
106     if (!splay[root].ch[0]&&!splay[root].ch[1]){
107         splay_clear(root); root=0;
108         return;
109     }
110     if (!splay[root].ch[0]){
111         int cur=root; root=splay[root].ch[1];
112         splay[root].fa=0; splay_clear(cur);
```

```
113             return;
114         }
115         if (!splay[root].ch[1]){
116             int cur=root; root=splay[root].ch[0];
117             splay[root].fa=0; splay_clear(cur);
118             return;
119         }
120         int cur=root,x=splay_prenumber();
121         splay_splay(x,0);
122         splay[splay[cur].ch[1]].fa=x;
123         splay[x].ch[1]=splay[cur].ch[1];
124         splay_clear(cur); splay_maintain(root);
125     }
126     //operations:
127     switch (opt){
128         //add a point val x
129         case 1:splay_insert(x); break;
130         //delete a point val x
131         case 2:splay_delete(x); break;
132         //return point's min rank which value is x
133         case 3:printf("%d\n",splay_findrank(x)); break;
134         //return point's value which rank is x
135         case 4:printf("%d\n",splay_findvalue(x)); break;
136         //return x's prenumber
137         case 5:
138             splay_insert(x);
139             printf("%d\n",splay[splay_prenumber()].val);
140             splay_delete(x);
141             break;
142         //retunr x's nxtnumber
143         case 6:
144             splay_insert(x);
145             printf("%d\n",splay[splay_nxtnumber()].val);
146             splay_delete(x);
147             break;
148     }
```

4.5.2 区间翻转

别忘了哨兵节点 !!! 同时，相对应的翻转区间 $[l,r]$ 变为 $[l+1,r+1]$

```
1   //initial build:(1,n+2,0)
2   int splay_build(int l,int r,int f){
3       if (l>r) return 0;
4       int mid=(l+r)>>1,tmp=++tot;
```

```
5       splay[tmp].val=a[mid];
6       splay[tmp].fa=f; splay[tmp].cnt++;
7       splay[tmp].ch[0]=splay_build(l,mid-1,tmp);
8       splay[tmp].ch[1]=splay_build(mid+1,r,tmp);
9       splay_maintain(tmp);
10      return tmp;
11  }
12  void splay_pushdown(int x){
13      if (x&&splay[x].tag){
14          splay[splay[x].ch[0]].tag^=1;
15          splay[splay[x].ch[1]].tag^=1;
16          splay[x].tag=0;
17          swap(splay[x].ch[0],splay[x].ch[1]);
18      }
19  }
20  void splay_rotate(int x){
21      int y=splay[x].fa,z=splay[y].fa;
22      //remember to pushdown
23      splay_pushdown(x); splay_pushdown(y);
24      int chk=splay_getchild(x);
25      splay[y].ch[chk]=splay[x].ch[chk^1];
26      splay[splay[x].ch[chk^1]].fa=y;
27      splay[x].ch[chk^1]=y;
28      splay[y].fa=x; splay[x].fa=z;
29      if (z) splay[z].ch[y==splay[z].ch[1]]=x;
30      splay_maintain(x); splay_maintain(y);
31  }
32  int splay_findvalue_p(int k){
33      int cur=root;
34      while (1){
35          //remember to pushdown
36          splay_pushdown(cur);
37          if (splay[cur].ch[0]&&k<=splay[splay[cur].ch[0]].sz){
38              cur=splay[cur].ch[0];
39          }else{
40              k-=splay[cur].cnt+splay[splay[cur].ch[0]].sz;
41              if (k<=0) return cur;
42              cur=splay[cur].ch[1];
43          }
44      }
45  }
46  void splay_reverse(int l,int r){
47      l--; r++;
48      l=splay_findvalue_p(l);
```

```
49      r=splay_findvalue_p(r);
50      splay_splay(l,0); splay_splay(r,l);
51      int tmp=splay[splay[root].ch[1]].ch[0];
52      splay[tmp].tag^=1;
53  }
54  void splay_print(int x){
55      //remember to pushdown
56      splay_pushdown(x);
57      if (splay[x].ch[0]) splay_print(splay[x].ch[0]);
58      if (splay[x].val!=-inf&&splay[x].val!=inf)
59          printf("%d ",splay[x].val);
60      if (splay[x].ch[1]) splay_print(splay[x].ch[1]);
61  }
```

4.6 主席树

一定注意空间设在 30 倍以上比较保险。

一个样例：求给定区间内第 k 大的数。

```
1  #include <iostream>
2  #include <cstdio>
3  #include <algorithm>
4  #include <cstring>
5  #include <cmath>
6  #include <queue>
7  #include <vector>
8  using namespace std;
9  const int maxn = 200010;
10 int n, m;
11 int len;
12 int a[maxn], b[maxn];
13
14 struct treenode {
15     int ls, rs;
16     int sum;
17 } tree[maxn << 5];
18 int root[maxn];
19 int cnt, tmp;
20 int rl, rr, rk;
21
22 void pushup(int o) {
23     tree[o].sum = tree[tree[o].ls].sum + tree[tree[o].rs].sum;
24 }
25
```

```
26 void build(int &o, int l, int r) {
27     o = ++cnt;
28     if (l == r) {
29         tree[o].sum = 0;
30         return;
31     }
32     int mid = (l + r) >> 1;
33     build(tree[o].ls, l, mid);
34     build(tree[o].rs, mid+1, r);
35     pushup(o);
36 }
37 
38 void update(int &o, int l, int r, int lst, int val) {
39     o = ++cnt;
40     tree[o].ls = tree[lst].ls;
41     tree[o].rs = tree[lst].rs;
42     if (l == r) {
43         tree[o].sum = tree[lst].sum + 1;
44         return;
45     }
46     int mid = (l + r) >> 1;
47     if (val <= mid)
48         update(tree[o].ls, l, mid, tree[lst].ls, val);
49     else
50         update(tree[o].rs, mid+1, r, tree[lst].rs, val);
51     pushup(o);
52 }
53 
54 int query(int oL, int oR, int l, int r, int val) {
55     if (l == r) {
56         return l;
57     }
58     int mid = (l + r) >> 1;
59     int tot = tree[tree[oR].ls].sum - tree[tree[oL].ls].sum;
60     if (val <= tot)
61         return query(tree[oL].ls, tree[oR].ls, l, mid, val);
62     else
63         return query(tree[oL].rs, tree[oR].rs, mid+1, r, val-tot);
64 }
65 
66 int main() {
67     scanf("%d%d", &n, &m);
68     for (int i = 1; i <= n; ++i) {
69         scanf("%d", &a[i]);
```

```
70          b[i] = a[i];
71      }
72      sort(b+1, b+1+n);
73      len = unique(b+1, b+1+n) - (b+1);
74
75      build(root[0], 1, len);
76      for (int i = 1; i <= n; ++i) {
77          tmp = lower_bound(b+1, b+1+len, a[i]) - b;
78          update(root[i], 1, len, root[i-1], tmp);
79      }
80      while (m--) {
81          scanf("%d%d%d", &rl, &rr, &rk);
82          printf("%d\n", b[query(root[rl-1], root[rr], 1, len, rk)]);
83      }
84      return 0;
85  }
```

4.7 Link Cut Tree

4.7.1 性质

深度唯一 每一个 Splay 维护的是一条从上到下按在原树中深度严格递增的路径，且中序遍历 Splay 得到的每个点的深度序列严格递增。

节点唯一 每个节点包含且仅包含于一个 Splay 中。

4.7.2 操作

pushup 维护更改后该节点的信息，通常是 siz[],sum[] 等

pushdown 下传标记，通常将标签传给子树

isroot 是否是**所在 splay** 的根

access 打通一条从**原树（可能已经改变过）根节点**到指定点的路径（在一个 splay 中）。

最短路径上的所有点都在该 splay 内。且 x 必定为深度最大的点。

返回值：最后一次虚实链变换时虚边父亲节点的编号。含义：

1. 连续两次 access 后，第二次 access 的返回值为该两点的 lca

2. 表示 x 到根的链所在的 splay 的根。该节点一定被旋转到根节点，父亲为空。

makeroot 将指定的点设为**原树**的根。为了防止一个 splay 中出现深度相同的点。

如果没有关于链值的相关修改，可以省略 makeroot 和其相关函数，减少常数。

split 拿出一棵 splay，维护 x 到 y 的路径。

makeroot(x) 后 x 的深度一定最小，access(y) 后 y 在该 splay 中深度最大且得到 x-y 的 splay 路径，将 y 旋到 splay 根意味着所有结点均在 y 左侧，成为了他的孩子。直接对 y 进行子树处理即可。

cut 剪短 x-y 的边（如果有的话）

findroot 找到当前 splay 的根。

4.7.3 模板

实现链上加一个值，乘一个值，断边连边，链上求和。

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
using namespace std;
#define ll long long
#define il inline
const int maxn=100010;
const ll mod=51061;
struct LinkCutTree{
    #define lc ch[x][0]
    #define rc ch[x][1]
    int ch[maxn][2],fa[maxn];
    ll val[maxn],siz[maxn],sum[maxn];
    int revtag[maxn];
    ll addtag[maxn],multag[maxn];
    int getch(int x){return ch[fa[x]][1]==x;}
    int isroot(int x){return ch[fa[x]][0]!=x&&ch[fa[x]][1]!=x;}
    void pushrev(int x){
        swap(lc,rc); revtag[x]^=1;
    }
    void pushadd(int x,ll a){
        val[x]=(val[x]+a)%mod;
        sum[x]=(sum[x]+a*siz[x])%mod;
        addtag[x]=(addtag[x]+a)%mod;
```

```
27      }
28      void pushmul(int x,ll a){
29          val[x]=(val[x]*a)%mod;
30          sum[x]=(sum[x]*a)%mod;
31          multag[x]=(multag[x]*a)%mod;
32          addtag[x]=(addtag[x]*a)%mod;
33      }
34      void pushdown(int x){
35          if (multag[x]!=1){
36              pushmul(lc,multag[x]);
37              pushmul(rc,multag[x]);
38              multag[x]=1;
39          }
40          if (addtag[x]){
41              pushadd(lc,addtag[x]);
42              pushadd(rc,addtag[x]);
43              addtag[x]=0;
44          }
45          if (revtag[x]){
46              if (lc) pushrev(lc);
47              if (rc) pushrev(rc);
48              revtag[x]^=1;
49          }
50      }
51      void pushup(int x){
52          sum[x]=(sum[lc]+sum[rc]+val[x])%mod;
53          siz[x]=siz[lc]+siz[rc]+1;
54      }
55      void rotate(int x){
56          int y=fa[x],z=fa[y],chk=getch(x),w=ch[x][chk^1];
57          if (!isroot(y)) ch[z][y==ch[z][1]]=x;
58          ch[x][chk^1]=y;
59          ch[y][chk]=w;
60          if (w) fa[w]=y;
61          fa[y]=x; fa[x]=z;
62          pushup(y);
63      }
64      int st[maxn];
65      void splay(int x){
66          int y=x,z=0;
67          st[++z]=y;
68          while (!isroot(y)) st[++z]=y=fa[y];
69          while (z) pushdown(st[z--]);
70          while (!isroot(x)){
```

```
71              y=fa[x]; z=fa[y];
72              if (!isroot(y))
73                  rotate((ch[y][0]==x)^(ch[z][0]==y)?x:y);
74              rotate(x);
75          }
76          pushup(x);
77      }
78      void access(int x){
79          for (int y=0;x;x=fa[y=x]){
80              splay(x); rc=y; pushup(x);
81          }
82      }
83      void makeroot(int x){
84          access(x); splay(x); pushrev(x);
85      }
86      int findroot(int x){
87          access(x); splay(x);
88          while (lc){
89              pushdown(x); x=lc;
90          }
91          splay(x); return x;
92      }
93      void split(int x,int y){
94          //here y is splay's root
95          makeroot(x);
96          access(y); splay(y);
97      }
98      void link(int x,int y){
99          //x's father is y
100         makeroot(x);
101         if (findroot(y)!=x) fa[x]=y;
102     }
103     /* if it's always legal
104     void link(int x,int y){
105         makeroot(x); fa[x]=y;
106     }
107     */
108     void cut(int x,int y){
109         makeroot(x);
110         if (x==findroot(y)&&fa[y]==x&&!ch[y][0]){
111             fa[y]=ch[x][1]=0;
112             pushup(x);
113         }
114         //below also ok:
```

```
115         //if (!(findroot(y)!=x||siz[x]>2)){}
116     }
117     /* if it's always legal
118     void cut(int x,int y){
119         split(x,y); fa[x]=ch[y][0]=0;
120         pushup(y);
121     }
122     */
123     #undef lc
124     #undef rc
125 }tree;
126 int n,q;
127 int rx,ry,ra,rb,rc,rd;
128 char opt;
129 int main(){
130     scanf("%d%d",&n,&q);
131     for (int i=1;i<=n;++i){
132         tree.val[i]=1;
133         tree.multag[i]=1;
134         tree.siz[i]=1;
135     }
136     for (int i=1;i<n;++i){
137         scanf("%d%d",&rx,&ry);
138         tree.link(rx,ry);
139     }
140     while (q--){
141         opt=getchar();
142         while (opt!='+'&&opt!='-'&&opt!='*'&&opt!='/')
143             opt=getchar();
144         switch (opt){
145             case '+': //link add ra->rb rc
146                 scanf("%d%d%d",&ra,&rb,&rc);
147                 tree.split(ra,rb);
148                 tree.pushadd(rb,rc);
149                 break;
150             case '-': //cut ra-rb
151                 scanf("%d%d",&ra,&rb);
152                 tree.cut(ra,rb);
153                 scanf("%d%d",&ra,&rb);
154                 tree.link(ra,rb);
155                 break;
156             case '*': //link mul ra->rb rc
157                 scanf("%d%d%d",&ra,&rb,&rc);
158                 tree.split(ra,rb);
```

```
159                 tree.pushmul(rb,rc);
160                 break;
161             case '/': //link query ra->rb
162                 scanf("%d%d",&ra,&rb);
163                 tree.split(ra,rb);
164                 printf("%lld\n",tree.sum[rb]);
165         }
166     }
167     return 0;
168 }
```

4.7.4 应用

维护树链信息 通过 split(x,y) 可以把树上 x-y 的路径提取到以 y 为根的 splay 内。然后方便进行类似线段树的修改。

维护连通性 通过 findroot() 可以动态判断两点是否连通，如果两者 root 相同，则在一棵树上。

维护双连通分量

维护边权（生成树）

维护子树信息

4.8 01 Trie

4.8.1 维护极值

对于异或和问题，可以处理前缀异或和，就转化为两点异或极值问题了。

对于所有出现的值，从高位到低位加入一个 Trie。之后若要查询与任意一个值的异或最大值，则在 Trie 树上尽量走不同的即可。

例：查找不小于 k 的最短异或区间。

```
1  const int maxn = 100010;
2  int T, n, k;
3  int a[maxn], sum[maxn];
4  int ansl, ansr;
5  int tree[maxn * 30][2], cnt;
6  int rt[maxn * 30];
7  void add(int pos) {
8      int cur = 0, tmp;
9      for (int i = 29; i >= 0; --i) {
10         tmp = (sum[pos] >> i) & 1;
11         if (!tree[cur][tmp]) {
```

```
12              tree[cur][tmp] = ++cnt;
13              tree[cnt][0] = tree[cnt][1] = 0;
14          }
15          cur = tree[cur][tmp];
16          rt[cur] = max(rt[cur], pos);
17      }
18  }
19  int query(int x) {
20      int cur = 0, tmp, pos = -1;
21      for (int i = 29; i >= 0; --i) {
22          tmp = (x >> i) & 1;
23          if ((k >> i) & 1) {
24              // k[i]=1: must different
25              cur = tree[cur][tmp^1];
26          } else {
27              // k[i]=0: different>k, same continue
28              if (tree[cur][tmp^1]) {
29                  pos = max(pos, rt[tree[cur][tmp^1]]);
30              }
31              cur = tree[cur][tmp];
32          }
33          if (!cur) break;
34      }
35      // same to end
36      if (cur) pos = max(pos, rt[cur]);
37      return pos;
38  }
39  int main() {
40      T = read();
41      while (T--) {
42          memset(rt, -1, sizeof(rt));
43          cnt = 0;
44          tree[0][0] = tree[0][1] = 0;
45          n = read(); k = read();
46          ansl = -1; ansr = n;
47          for (int i = 1; i <= n; ++i) {
48              a[i] = read();
49              sum[i] = sum[i-1] ^ a[i];
50          }
51          if (k == 0) {
52              printf("1 1\n");
53              continue;
54          }
55          add(0);
```

```
56          for (int i = 1; i <= n; ++i) {
57              int r = query(sum[i]);
58              if (r >= 0 && i - r < ansr - ansl) {
59                  ansr = i;
60                  ansl = r;
61              }
62              add(i);
63          }
64          if (ansl >= 0) {
65              printf("%d %d\n", ansl+1, ansr);
66          } else {
67              printf("-1\n");
68          }
69      }
70      return 0;
71  }
```

4.8.2 维护异或和

支持插入、删除、全局 +1。

从低位到高位建立 Trie。

例：给定有根树 (root=1)，每个节点有一个权值 v。设 x 的价值 $val(x)$ 表示为：子树内所有点 c 权值 $v_c + d(c,x)$ 的异或和，其中 $d(x,y)$ 代表 $x \to y$ 简单路径经过的边数。求 $\sum_{i=1}^{n} val(i)$。

从叶子合并到根即可。每次升高高度相当于子树全局 +1，通过合并操作处理合并到根。

```
1   const int maxn = 525020;
2   namespace Trie {
3       const int maxh = 25;
4       int root[maxn];
5       int ch[maxn * maxh][2];
6       int w[maxn * maxh];
7       int xorval[maxn * maxh];
8       int nodecnt = 0;
9       void maintain(int o) {
10          w[o] = xorval[o] = 0;
11          if (ch[o][0]) {
12              w[o] += w[ch[o][0]];
13              xorval[o] ^= xorval[ch[o][0]] << 1;
14          }
15          if (ch[o][1]) {
16              w[o] += w[ch[o][1]];
17              xorval[o] ^= (xorval[ch[o][1]] << 1) | (w[ch[o][1]] & 1);
```

```
18          }
19          w[o] &= 1;
20      }
21      void insert(int &o, int x, int dep) {
22          if (!o) {
23              o = ++nodecnt;
24              ch[o][0] = ch[o][1] = 0;
25              w[o] = 0;
26              xorval[o] = 0;
27          }
28          if (dep >= maxh) {
29              ++w[o]; return;
30          }
31          insert(ch[o][x & 1], x >> 1, dep + 1);
32          maintain(o);
33      }
34      void erase(int o, int x, int dep) {
35          if (dep > 20) {
36              --w[o];
37              return;
38          }
39          erase(ch[o][x & 1], x >> 1, dep + 1);
40          maintain(o);
41      }
42      void addallone(int o) {
43          // add one for all nodes
44          swap(ch[o][1], ch[o][0]);
45          if (ch[o][0]) addallone(ch[o][0]);
46          maintain(o);
47      }
48      int merge(int a, int b) {
49          // merge b to a
50          if (!a) return b;
51          if (!b) return a;
52          w[a] += w[b]; w[a] &= 1;
53          xorval[a] ^= xorval[b];
54          ch[a][0] = merge(ch[a][0], ch[b][0]);
55          ch[a][1] = merge(ch[a][1], ch[b][1]);
56          return a;
57      }
58  }
59  int n;
60  int v[maxn];
61  int rf;
```

```
62 int root[maxn];
63 vector<int> G[maxn];
64 ll ans;
65 void work(int u) {
66     for (int i = 0; i < G[u].size(); ++i) {
67         int v = G[u][i];
68         work(v);
69         root[u] = Trie::merge(root[u], root[v]);
70     }
71     Trie::addallone(root[u]);
72     Trie::insert(root[u], v[u], 0);
73     ans += Trie::xorval[root[u]];
74 }
75 int main() {
76     scanf("%d", &n);
77     for (int i = 1; i <= n; ++i) {
78         scanf("%d", &v[i]);
79     }
80     for (int i = 2; i <= n; ++i) {
81         scanf("%d", &rf);
82         G[rf].push_back(i);
83     }
84     work(1);
85     printf("%lld\n", ans);
86     return 0;
87 }
```

5 计算几何

5.1 常用定理与结论

- 判断直线 AB 与线段 CD 相交：由于叉积可以表示两向量的相对关系，可以通过判断向量 AC，AD 与 AB 的叉积的积的正负来判断。负数必然相交，正数不相交，0 特殊判断。如果判断线段 AB 与线段 CD 是否相交时，判断两项：直线 AB 与线段 CD 相交，直线 CD 与线段 AB 相交。
- 判断直线 AB 是否将两点 CD 分隔开：可通过上一个方法。另如果知道直线方程 ax+by+c=0 则可以将点 CD 的坐标带入左侧，若两者乘积为负说明在两侧。
- Pick 定理：给定顶点均为整点的简单多边形，皮克定理说明了其面积 A 和内部格点数目 i、边上格点数目 b 的关系：$A = i + \frac{b}{2} - 1$。

5.2 扫描线

扫描线用来支持快速计算多个矩阵的面积并，周长等等。思想是将一维排序，同时用线段树维护另一维的长度变化（线段树每个节点代表线段，与常规的线段树存在不同）。

```
const int maxn = 100010;
struct ScanLine {
    #define ls o << 1
    #define rs ls | 1
    int n;
    struct ScanLineNode {
        int x, y1, y2;
        int flag;
        bool operator < (const ScanLineNode o) {
            return x < o.x;
        }
    } p[maxn << 1];
    int mp[maxn << 1];
    int mpcnt;
    struct SegTreeNode {
        int dl, dr, sum;
        int cnt;
    } tree[maxn << 4];
    // attention: 16 times is preffered
    void readIn() {
        scanf("%d", &n);
        int rx, rxx, ry, ryy;
        for (int i = 1; i <= (n << 1); i += 2) {
```

```
            scanf("%d%d%d%d", &rx, &ry, &rxx, &ryy);
            p[i].x = rx; p[i].y1 = ry; p[i].y2 = ryy;
            p[i].flag = 1;
            p[i + 1].x = rxx; p[i + 1].y1 = ry; p[i + 1].y2 = ryy;
            p[i + 1].flag = -1;
            mp[i] = ry; mp[i + 1] = ryy;
        }
    }

    void build(int o, int l, int r) {
        tree[o].dl = mp[l];
        tree[o].dr = mp[r];
        tree[o].sum = 0;
        tree[o].cnt = 0;
        if (r - l > 1) {
            int mid = (l + r) >> 1;
            build(ls, l, mid);
            build(rs, mid, r);
        }
    }

    void init() {
        sort(mp + 1, mp + 1 + (n << 1));
        mpcnt = unique(mp + 1, mp + 1 + (n << 1)) - mp - 1;
        sort(p + 1, p + 1 + (n << 1));
        build(1, 1, mpcnt);
    }

    void pushup(int o) {
        if (tree[o].cnt > 0) {
            tree[o].sum = tree[o].dr - tree[o].dl;
        } else {
            tree[o].sum = tree[ls].sum + tree[rs].sum;
        }
    }

    void update(int o, int ql, int qr, int v) {
        // cout << o << " " << ql << " " << qr << endl;
        if (tree[o].dl >= ql && tree[o].dr <= qr) {
            tree[o].cnt += v;
            pushup(o);
        } else {
            if (tree[ls].dr > ql) update(ls, ql, qr, v);
            if (tree[rs].dl < qr) update(rs, ql, qr, v);
```

```
68              pushup(o);
69          }
70      }
71
72      ll calc() {
73          ll ans = 0;
74          update(1, p[1].y1, p[1].y2, p[1].flag);
75          for (int i = 2; i <= (n << 1); ++i) {
76              ans += 1ll * (p[i].x - p[i - 1].x) * tree[1].sum;
77              update(1, p[i].y1, p[i].y2, p[i].flag);
78          }
79          return ans;
80      }
81      #undef ls
82      #undef rs
83  } S;
84  int main() {
85      S.readIn();
86      S.init();
87      printf("%lld", S.calc());
88      return 0;
89  }
```

5.3 凸包

在平面上能**包含所有给定点**的**最小凸多边形**叫做凸包。同时是能包住所有点多边形中**周长最短**的。

```
1   struct vec{
2       //vector,also the point's position
3       double x,y;
4       bool operator <(const vec &_p){
5           return (x==_p.x)?y<_p.y:x<_p.x;
6       }
7       vec operator +(const vec &_p){
8           return (vec){x+_p.x,y+_p.y};
9       }
10      vec operator -(const vec &_p){
11          return (vec){x-_p.x,y-_p.y};
12      }
13  }p[maxn];
14  double cross(vec _a,vec _b){
15      return _a.x*_b.y-_a.y*_b.x;
16  }
```

```
int sta[maxn],top=0;
//instack=on convex hull
int used[maxn]; //if it is on convex hull
void Andrew(){
    sort(p+1,p+1+n);
    if (n<=2) return; //no answer
    sta[++top]=1;
    for (int i=2;i<=n;++i){
        while (top>=2&&
               cross(p[sta[top]]-p[sta[top-1]],p[i]-p[sta[top]])<=0){
            used[sta[top--]]=0;
        }
        used[i]=1; sta[++top]=i;
    }
    int tmp=top;
    for (int i=n-1;i>0;--i){
        if (used[i]) continue;
        //don't have influence on down convex hull
        while (top>tmp&&
               cross(p[sta[top]]-p[sta[top-1]],p[i]-p[sta[top]])<=0){
            used[sta[top--]]=0;
        }
        used[i]=1; sta[++top]=i;
    }
}
```

5.4 旋转卡壳

在原有凸包的基础上，$O(n)$ 扫描即可。

```
double dis(vec _a,vec _b){
    return sqrt((_a.x-_b.x)*(_a.x-_b.x)
                +(_a.y-_b.y)*(_a.y-_b.y));
}
double getdiam(){
    if (n<=2){
        return dis(p[1],p[2]);
    }
    double res=0.0;
    int j=3;
    for (int i=1;i<top;++i){
        while (cross(p[sta[i+1]]-p[sta[i]],p[sta[j]]-p[sta[i]])
              <cross(p[sta[i+1]]-p[sta[i]],p[sta[j+1]]-p[sta[i]])){
            ++j; if (j>top) j-=top;
```

```
15          }
16          res=max(res,dis(p[sta[i]],p[sta[j]]));
17          res=max(res,dis(p[sta[i+1]],p[sta[j]]));
18      }
19      return res;
20  }
```

6 字符串

6.1 哈希 Hash

```
const ll base=131;
const ll mod=1610612741;//or 1e9+7
//do hash
for (rg int i=1;i<=n;++i){
    s[i]=read();
    hs[i]=(hs[i-1]*base+s[i])%mod;
}
//get hash
il ll geth(un ll a[],int l,int r){
    return ((a[r]-a[l-1]*p[r-l+1])%mod+mod)%mod;
}
```

6.2 前缀函数

摘自OI Wiki——前缀函数与 KMP 算法

前缀函数：$\pi[i]=\max_{k=0\ldots i}\{k : s[0\ldots k-1]=s[i-(k-1)\ldots i]\}$

实现：复杂度 $O(N)$。

```
char s[maxn];
int pi[maxn];
void prefix_function(){
    int len=strlen(s);
    pi[0]=0;
    for (int i=1;i<n;++i){
        int j=pi[i-1];
        while (j>0&&s[i]!=s[j]) j=pi[j-1];
        if (s[i]==s[j]) j++;
        pi[i]=j;
    }
}
```

1. 字符串中查找子串（KMP）： 复杂度 $O(n+m)$

假设要在文本 t（长度 m）中查找字符串 s（长度 n），则我们构造新的字符串 $S=s+\#+t$。（$\#$ 不在 s，t 中出现），我们只需寻找 $\pi[i]=n$ 的位置即可。

2. 统计前缀出现次数 ：

一下代码是统计自身的。

```
vector<int> ans(n + 1);
for (int i = 0; i < n; i++) ans[pi[i]]++;
```

```
for (int i = n - 1; i > 0; i--) ans[pi[i - 1]] += ans[i];
for (int i = 0; i <= n; i++) ans[i]++;
```

统计非自身的方法可以参考 **1. 字符串中查找子串（KMP）**中的方法构造新字符串解决。

3. 一个字符串中本质不同子串的数目 给定一个长度为 n 的字符串 s，我们希望计算其本质不同子串的数目。

我们将迭代的解决该问题。换句话说，在知道了当前的本质不同子串的数目的情况下，我们要找出一种在 s 末尾添加一个字符后重新计算该数目的方法。

令 k 为当前 s 的本质不同子串数量。我们添加一个新的字符 c 至 s。显然，会有一些新的子串以字符 c 结尾。我们希望对这些以该字符结尾且我们之前未曾遇到的子串计数。

构造字符串 $t = s + c$ 并将其反转得到字符串 t' 。现在我们的任务变为计算有多少 t' 的前缀未在 t' 的其余任何地方出现。如果我们计算了 t' 的前缀函数最大值 π_{max} ，那么最长的出现在 s 中的前缀其长度为 π_{max} 。自然的，所有更短的前缀也出现了。

因此，当添加了一个新字符后新出现的子串数目为 $|s| + 1 - \pi_{max}$ 。

所以对于每个添加的字符，我们可以在 $O(n)$ 的时间内计算新子串的数目，故最终复杂度为 $O(n^2)$ 。

值得注意的是，我们也可以重新计算在头部添加一个字符，或者从尾或者头移除一个字符时的本质不同子串数目。

6.3 KMP

```
char t[maxn],s[maxn]; //find s in t
int slen,tlen; //don't use strlen in for()
int pi[maxn]; //pi[i]=next[i]
void prefix_function(){
    pi[0]=0;
    for (int i=1;i<slen;++i){
        int j=pi[i-1];
        while (j>0&&s[i]!=s[j]) j=pi[j-1];
        if (s[i]==s[j]) j++;
        pi[i]=j;
    }
}
vector<int> ans;
void kmp(){
    prefix_function();
    int j=0;
    for (int i=0;i<tlen;++i){
        while (j&&t[i]!=s[j]) j=pi[j-1];
```

```
19          if (t[i]==s[j]) ++j;
20          if (j==slen){
21              ans.push_back(i-slen+2);
22              j=pi[j-1];
23          }
24      }
25  }
```

6.4 Trie 树

```
1   struct trie {
2     int nex[100000][26], cnt;
3     bool exist[100000];
4     void insert(char *s, int l) {
5       int p = 0;
6       for (int i = 0; i < l; i++) {
7         int c = s[i] - 'a';
8         if (!nex[p][c]) nex[p][c] = ++cnt;
9         p = nex[p][c];
10      }
11      exist[p] = 1;
12    }
13    bool find(char *s, int l) {
14      int p = 0;
15      for (int i = 0; i < l; i++) {
16        int c = s[i] - 'a';
17        if (!nex[p][c]) return 0;
18        p = nex[p][c];
19      }
20      return exist[p];
21    }
22  };
```

6.5 AC 自动机

参考自OI-Wiki

AC 自动机的失配指针指向所有模式串的前缀中最长后缀状态。

下图展示的就是 hers,his,she,i 的表示。

初始将要查询的所有串 insert()，之后查询用 query() 返回匹配到的串的总数。**千万不要漏了 AC.build()**。

```
1   struct AC_automaton{
2       int trie[75*160][26],tot;
3       int fail[75*160];
4       int mp[75*160],cnt[160];
```

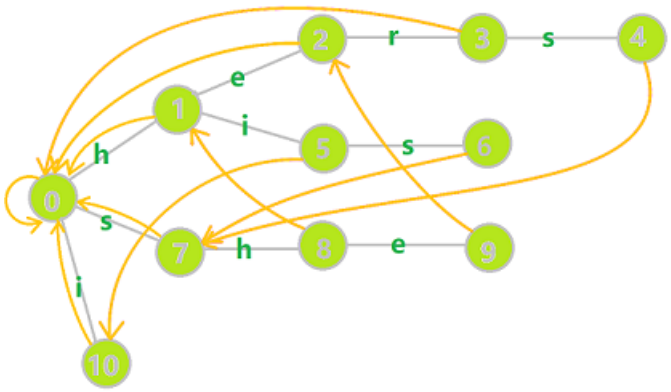

Figure 1: AC 自动机示意图

```
5       //mp:trie->origin string
6       //additional operation
7       int endword[maxn]; //how many times
8       //additional operation
9       void init(){
10          memset(trie,0,sizeof(trie)); tot=0;
11          memset(fail,0,sizeof(fail));
12          memset(cnt,0,sizeof(cnt));
13          memset(mp,0,sizeof(mp));
14      }
15      void insert(char *s,int id){
16          int u=0;
17          for (int i=1;s[i];++i){
18              if (!trie[u][s[i]-'a']) trie[u][s[i]-'a']=++tot;
19              u=trie[u][s[i]-'a'];
20          }
21          mp[u]=id; //addational:endword[u]++;
22      }
23      void build(){
24          queue<int> q;
25          for (int i=0;i<26;++i){
26              if (trie[0][i]) q.push(trie[0][i]);
27          }
28          while (!q.empty()){
29              int u=q.front(); q.pop();
30              for (int i=0;i<26;++i){
31                  if (trie[u][i]){
32                      fail[trie[u][i]]=trie[fail[u]][i];
33                      q.push(trie[u][i]);
```

```
                }else{
                    trie[u][i]=trie[fail[u]][i];
                }
            }
        }
    }
    int query(char *s){
        int u=0,ans=0;
        for (int i=1;s[i];++i){
            u=trie[u][s[i]-'a'];
            for (int j=u;j;j=fail[j]){
                if (mp[j]){
                    cnt[mp[j]]++; ans=max(ans,cnt[mp[j]]);
                }
                //additional:
                ans+=endword[j]; endword[j]=-1;
                //attention:in for judge:j&endword[j]!=-1
            }
        }
        return ans;
    }
}AC;
```

注意以下几点：

- AC 自动机求的是允许部分重合的子串出现次数，如果要求求相离的子串出现次数，可以通过记录上一个出现位置来判断。
- 该版本代码无法处理查询相同的子串（后面会覆盖前面），可以通过判断来支持。

```
int sameto[maxn];
struct AC_automaton {
    int lst[400010], dep[400010];
    void init(){
        memset(lst, -1, sizeof(lst));
        memset(dep, 0, sizeof(dep));
    }
    void insert(char *s, int id){
        // above is the same
        if (!mp[u]) mp[u] = id;
        else sameto[id] = mp[u];
    }
    void query(char *s) {
        int u = 0;
```

```
        for (int i = 1; s[i]; ++i){
            u = trie[u][s[i]-'a'];
            for (int j = u; j; j = fail[j]){
                // modify
                if (mp[j] && i - lst[j] >= dep[j]) {
                    ++cnt[mp[j]];
                    lst[j] = i;
                }
            }
        }
    }
} AC;

// output: same need to change
for (int i = 1; i <= n; ++i) {
    if (AC.cnt[i]) printf("%d\n", AC.cnt[i]);
    else printf("%d\n", AC.cnt[sameto[i]]);
}
```

6.6 后缀数组 SA

6.6.1 SA 求解

倍增

```
int n; char s[maxn]; //start from 1
int sa[maxn],rk[maxn];
int oldrk[maxn<<1],id[maxn],px[maxn],cnt[maxn];
bool cmp(int x,int y,int w){
    return (oldrk[x]==oldrk[y]&&oldrk[x+w]==oldrk[y+w]);
}
void get_sa(){
    int m=300,p;
    for (int i=1;i<=n;++i) ++cnt[rk[i]=s[i]];
    for (int i=1;i<=m;++i) cnt[i]+=cnt[i-1];
    for (int i=n;i>=1;--i) sa[cnt[rk[i]]--]=i;
    for (int w=1;w<n;w<<=1,m=p){
        p=0;
        for (int i=n;i>n-w;--i) id[++p]=i;
        for (int i=1;i<=n;++i){
            if (sa[i]>w) id[++p]=sa[i]-w;
        }
        memset(cnt,0,sizeof(cnt));
        for (int i=1;i<=n;++i) ++cnt[px[i]=rk[id[i]]];
        for (int i=1;i<=m;++i) cnt[i]+=cnt[i-1];
```

```
        for (int i=n;i>=1;--i) sa[cnt[px[i]]--]=id[i];
        memcpy(oldrk,rk,sizeof(rk));
        p=0;
        for (int i=1;i<=n;++i){
            rk[sa[i]]=cmp(sa[i],sa[i-1],w)?p:++p;
        }
    }
}
```

SA-IS 摘自诱导排序与 SA-IS 算法

```
#define L_TYPE 0
#define S_TYPE 1
inline bool is_lms_char(int *type, int x) {
    return x > 0 && type[x] == S_TYPE && type[x - 1] ==
    ↪ L_TYPE;
}
inline bool equal_substring(int *S, int x, int y, int *type) {
    do {
        if (S[x] != S[y])
            return false;
        x++, y++;
    } while (!is_lms_char(type, x) && !is_lms_char(type, y));

    return S[x] == S[y];
}
inline void induced_sort(int *S, int *SA, int *type, int
↪ *bucket, int *lbucket,
                         int *sbucket, int n, int SIGMA) {
    for (int i = 0; i <= n; i++)
        if (SA[i] > 0 && type[SA[i] - 1] == L_TYPE)
            SA[lbucket[S[SA[i] - 1]]++] = SA[i] - 1;
    for (int i = 1; i <= SIGMA; i++)
        sbucket[i] = bucket[i] - 1;
    for (int i = n; i >= 0; i--)
        if (SA[i] > 0 && type[SA[i] - 1] == S_TYPE)
            SA[sbucket[S[SA[i] - 1]]--] = SA[i] - 1;
}
// SAIS start from zero
// s:string,sigma:size of strings
static int *SAIS(int *S, int length, int SIGMA) {
    int n = length - 1;
    int *type = new int[n + 1];
    int *position = new int[n + 1];
```

```
32      int *name = new int[n + 1];
33      int *SA = new int[n + 1];
34      int *bucket = new int[SIGMA];
35      int *lbucket = new int[SIGMA];
36      int *sbucket = new int[SIGMA];
37      memset(bucket, 0, sizeof(int) * (SIGMA + 1));
38      for (int i = 0; i <= n; i++)
39          bucket[S[i]]++;
40      for (int i = 1; i <= SIGMA; i++) {
41          bucket[i] += bucket[i - 1];
42          lbucket[i] = bucket[i - 1];
43          sbucket[i] = bucket[i] - 1;
44      }
45      type[n] = S_TYPE;
46      for (int i = n - 1; i >= 0; i--) {
47          if (S[i] < S[i + 1])
48              type[i] = S_TYPE;
49          else if (S[i] > S[i + 1])
50              type[i] = L_TYPE;
51          else
52              type[i] = type[i + 1];
53      }
54      int cnt = 0;
55      for (int i = 1; i <= n; i++)
56          if (type[i] == S_TYPE && type[i - 1] == L_TYPE)
57              position[cnt++] = i;
58      fill(SA, SA + n + 1, -1);
59      for (int i = 0; i < cnt; i++)
60          SA[sbucket[S[position[i]]]--] = position[i];
61      induced_sort(S, SA, type, bucket, lbucket, sbucket, n,
        ↪  SIGMA);
62      fill(name, name + n + 1, -1);
63      int lastx = -1, namecnt = 1;
64      bool flag = false;
65      for (int i = 1; i <= n; i++) {
66          int x = SA[i];
67          if (is_lms_char(type, x)) {
68              if (lastx >= 0 && !equal_substring(S, x, lastx,
                ↪  type))
69                  namecnt++;
70              if (lastx >= 0 && namecnt == name[lastx])
71                  flag = true;
72              name[x] = namecnt;
73              lastx = x;
```

```
74          }
75      }
76      name[n] = 0;
77      int *S1 = new int[cnt];
78      int pos = 0;
79      for (int i = 0; i <= n; i++)
80          if (name[i] >= 0)
81              S1[pos++] = name[i];
82      int *SA1;
83      if (!flag) {
84          SA1 = new int[cnt + 1];
85          for (int i = 0; i < cnt; i++)
86              SA1[S1[i]] = i;
87      } else
88          SA1 = SAIS(S1, cnt, namecnt);
89      lbucket[0] = sbucket[0] = 0;
90      for (int i = 1; i <= SIGMA; i++) {
91          lbucket[i] = bucket[i - 1];
92          sbucket[i] = bucket[i] - 1;
93      }
94      fill(SA, SA + n + 1, -1);
95      for (int i = cnt - 1; i >= 0; i--)
96          SA[sbucket[S[position[SA1[i]]]]--] = position[SA1[i]];
97      induced_sort(S, SA, type, bucket, lbucket, sbucket, n,
        ↪ SIGMA);
98      return SA;
99  }
```

6.6.2 height 数组

基本定义 $lcp(u,v)$ 代表 u,v 两个字符串的最长公共前缀长度。

$LCP(i,j)$ 第 i 个后缀与第 j 个后缀的最长公共前缀长度。

$height[i] = lcp(sa[i], sa[i-1])$ 排名 i 与其前一名后缀的最长公共前缀长度。

$h[i] = height[rank[i]]$ 后缀 i 和排序后在他前一名的后缀的最长公共前缀长度。

重要性质：$h[i] \geq h[i-1] - 1$

height 数组求解 $O(n)$

```
1  for (i=1,k=0;i<=n;++i){
2      if (k) --k;
3      while (s[i+k]==s[sa[rk[i]-1]+k]) ++k;
4      ht[rk[i]]=k;
5  }
```

性质与应用

- $lcp(sa[i], sa[j]) = \min\{height[i+1 \dots j]\}$
- 比较字符串关系：
  $A = S[a \dots b], B = [c \dots d]$
  $if\ lcp(a,c) > \min\{|A|,|B|\}, then\ A < B \iff |A| < |B|$
  $else\ A < B \iff rk[a] < rk[b]$
- 不同子串数目：后缀的前缀，考虑枚举后缀，统计前缀。
  前缀总数 = 子串总数 $=\frac{n(n+1)}{2}$
  考虑去掉重复的：以每个后缀开始，重复了 lcp 个。
  答案：$\frac{n(n+1)}{2} - \sum_{i=2}^{n} height[i]$

6.7 后缀自动机 SAM

6.7.1 原理与实现

一个字符串 S 的自动机接受且仅接受 S 的后缀。
对于一个状态 st ，只保存以下要素：

数据	含义
$maxlen(st)$	st 包含的最长子串长度
$trans[st][\sum c]$	st 的转移函数，其中 $\sum c$ 为字符集
$link[st]$	st 的后缀链接

可能用到的性质：
0. 在 $|S| \geq 3$ 时，总状态数不超过 $2 \times |S| - 1$，转移数不超过 $3 \times |S| - 4$
1. 对于两个不同状态 u 和 v，包含的子串 substrings(u),substrings(v) 的交为空集;
2. 每个子串都恰好被一个状态包含。
3.minlen(st)=maxlen(link(st))+1
注：代码中的拓扑排序用来计算每个状态 st 的 endpos 大小 cnt[st]，也即该类子串出现了几次。别忘了初始化 SAM::init()。

```
namespace SAM{
    struct state{
        int mxlen,link;
        map<char,int> nxt;
    }st[maxn<<1];
    bool vis[maxn<<1];
    int ind[maxn<<1];
```

```
8       int cnt[maxn<<1];
9       int sz,lst;
10      void init(){
11          st[0].mxlen=0;
12          st[0].link=-1;
13          ++sz; lst=0;
14      }
15      void extend(char c){
16          int cur=sz++,p=lst; vis[cur]=true;
17          st[cur].mxlen=st[lst].mxlen+1;
18          while (p!=-1&&!st[p].nxt.count(c)){
19              st[p].nxt[c]=cur;
20              p=st[p].link;
21          }
22          if (p==-1){
23              st[cur].link=0;
24          }else{
25              int q=st[p].nxt[c];
26              if (st[p].mxlen+1==st[q].mxlen){
27                  st[cur].link=q;
28              }else{
29                  int tmp=sz++;
30                  st[tmp].mxlen=st[p].mxlen+1;
31                  st[tmp].nxt=st[q].nxt;
32                  st[tmp].link=st[q].link;
33                  while (p!=-1&&st[p].nxt[c]==q){
34                      st[p].nxt[c]=tmp;
35                      p=st[p].link;
36                  }
37                  st[q].link=st[cur].link=tmp;
38              }
39          }
40          lst=cur;
41      }
42      void toposort(){
43          for (int i=1;i<=sz;++i){
44              ind[st[i].link]++;
45          }
46          for (int i=1;i<=sz;++i){
47              if (ind[i]&&vis[i]) ++cnt[i];
48          }
49          queue<int> q;
50          for (int i=1;i<=sz;++i){
51              if (!ind[i]){
```

```
52                  q.push(i); ++cnt[i];
53              }
54          }
55          while (!q.empty()){
56              int u=q.front(); q.pop();
57              int v=st[u].link;
58              if (!v) continue;
59              cnt[v]+=cnt[u]; ind[v]--;
60              if (!ind[v]) q.push(v);
61          }
62      }
63  }
```

6.7.2 应用

- 求 S 的不同字串：对所有 st 求 $\sum_{st} maxlen(st) - minlen(st) + 1$
- 求子串的出现次数：每个子串存在且仅存在于一个状态 st 中，统计其 st 的 endpos 大小即是出现次数。而 endpos 大小为所有以他为 link 的 st 的和（如果包含原字符串前缀，即初始加入不用复制的状态，+1)。所以可以通过加入时标记（vis)，拓扑排序计算。

  衍生：求长度为 K 子串中出现次数最多的子串的出现次数。设长度为 x 的答案为 ans[x]，考虑到随 x 增大 ans[x] 单调递减，只需要通过更新每个 st 的最大值，然后对每个 ans 更新 ans[x]=max(ans[x],ans[x+1]) 即可。

```
void getans(){
    for (int i=1;i<=sz;++i){
        ans[st[i].mxlen]=max(ans[st[i].mxlen],cnt[i]);
    }
    for (int i=len-1;i>=1;--i){
        ans[i]=max(ans[i+1],ans[i]);
    }
}
```

6.8 Manacher

最长回文串，复杂度 $O(N)$。

一般将初始字符串左右端加上特殊字符，同时每个字符之间插上**同一特殊字符**'#'

```
1  int changestring(){
2      int len=strlen(s);
3      new_s[0]='$'; new_s[1]='#';
4      int j=2;
```

```
5       for (int i=0;i<len;++i){
6           new_s[j++]=s[i];
7           new_s[j++]='#';
8       }
9       new_s[j]='\0'; //don't forget
10      return j; //return new string's length
11  }
12  int manacher(){
13      int len=changestring(),ans=-1;
14      int l=-1,r=0;
15      for (int i=1;i<len;++i){
16          if (i<r){
17              p[i]=min(p[2*l-i],r-i);
18          }else{
19              p[i]=1;
20          }
21          while (new_s[i-p[i]]==new_s[i+p[i]]) ++p[i];
22          if (r<i+p[i]){
23              l=i; r=i+p[i];
24          }
25          ans=max(ans,p[i]-1);
26      }
27      return ans;
28  }
```

7 动态规划

7.1 线性 DP

7.1.1 LIS 问题

用 d 数组存储目前的最长上升 or 下降子序列。

```
for (int i=1;i<=n;++i){ //最长上升子序列（严格）
    if (!cnt||a[i]>d[cnt]){
        d[++cnt]=a[i];
    }else{
        int p=lower_bound(d+1,d+1+cnt,a[i])-d;
        d[p]=a[i];
    }
}
for (int i=1;i<=n;++i){ //最长不下降子序列
    if (!cnt||a[i]>=d[cnt]){
        d[++cnt]=a[i];
    }else{
        int p=upper_bound(d+1,d+1+cnt,a[i])-d;
        d[p]=a[i];
    }
}
for (int i=1;i<=n;++i){ //最长不上升子序列
    if (!cnt||a[i]<=d[cnt]){
        d[++cnt]=a[i];
    }else{
        int p=upper_bound(d+1,d+1+cnt,a[i],greater<int>())-d;
        d[p]=a[i];
    }
}
```

7.2 背包 DP

去看《**背包九讲**》即可。

7.3 区间 DP

合并：即将两个或多个部分进行整合，当然也可以反过来。

特征：能将问题分解为能**两两合并**的形式。如果是 k 个合并在一块，且**能转化为 1 个和 k-1 个合并的形式**，也是可以利用的。

求解：对整个问题设最优值，枚举合并点，将问题分解为左右两个部分，最后合并两个部分的最优值得到原问题的最优值。

例题：n 堆石子，一次只能合并**连续**的 $[l,r]$ 堆。合并价值为数量，求最小耗费。如果无法实现输出 0。

设 $f[i][j][k]$ 代表从 i 到 j 合并了 k 堆的最小花费，则得到下列表达式：

(1)$f[i][j][1] = min\{f[i][j][d] + sum[j] - sum[i-1]\}, d \in [l, r]$ 。能直接合并成一堆的肯定比两次合并更优。

(2)$f[i][j][k] = min\{f[i][x][1] + f[x+1][j][k-1]\}$ 。每一个 k 堆可以拆成 1 堆与 $k-1$ 堆的合并，相当于两两合并。

至于为什么不考虑由 d 堆和 $k-d$ 堆合并：枚举 d 堆明显增加一维复杂度。如果枚举 $f[i][x][d]$ ，其实相当于内部空间再由 $f[i][x'][1] + f[x'][x][d-1]$ 组合而成的。也就是说，分成 1 和 $k-1$ 实则已经考虑了所有情况，再重复枚举多余。

```
for (int i=1;i<=n;++i){
    f[i][i][1]=0;
}
for (int len=1;len<=n;++len){
    for (int i=1;i<=n;++i){
        int j=i+len-1;
        for (int k=len;k>=1;--k){
            if (k==1){
                for (int tmp=l;tmp<=r;++tmp){
                    f[i][j][k]=
                        min(f[i][j][k],f[i][j][tmp]+sum[j]-sum[i-1]);
                }
            }else{
                for (int tmp=i;tmp<j;++tmp){
                    f[i][j][k]=
                        min(f[i][j][k],f[i][tmp][1]+f[tmp+1][j][k-1]);
                }
            }
        }
    }
}
```

7.4 数位 dp

关于数字相关的问题考虑数位 dp。利用前缀和的思想处理 l 到 r 区间个数问题。

例题：从 1-n 每个数出现了几次。

```
ll f[maxn][2333][2][2]; //pos sum limit zero
ll a[maxn],len;
ll dfs(ll len,ll sum,ll limit,ll zero,ll x){
    if (!len) return sum;
    if (f[len][sum][limit][zero]!=-1){
        return f[len][sum][limit][zero];
    }
```

```
    ll n=limit?a[len]:9;
    ll t=0;
    for (ll i=0;i<=n;++i){
        t+=dfs(len-1,sum+((i==x)&&(!zero||i)),
               limit&&(i==n),zero&&(!i),x);
    }
    f[len][sum][limit][zero]=t;
    return t;
}
ll solve(ll n,ll x){
    memset(a,0,sizeof(a));
    len=0;
    while (n){
        a[++len]=n%10;
        n/=10;
    }
    memset(f,-1,sizeof(f));
    return dfs(len,0,1,1,x);
}
```

7.5 动态规划的优化

7.5.1 单调队列，单调栈优化

多重背包的优化 单调队列可以优化多重背包：有 n 个物品，每个物品重 w_i ，价值 v_i ，数量 k_i ，背包重量上限 m，求最大价值。

初始方程：$f_{i,j}=\max\limits_{k=0}^{k_i}\{f_{i-1,j-k\times w_i}+v_i\times k\}$ 。复杂度 $O(nW\sum k_i)$ 。

优化：设 $g_{x,y}=f_{i,x\times w_i+y}, g'_{x,y}=f_{i-1,x\times w_i+y}$ ，得到转移方程：$g_{x,y}=\max\limits_{k=0}^{k_i}\{g'_{x-k,y}+v_i\times k\}$ 。不妨设 $G_{x,y}=g'_{x,y}-v_i\times x$ ，则方程可以表示为：

$$g_{x,y}=\max_{k=0}^{k_i}\{G_{x-k,y}\}+v_i\times x$$

其中 $G_{x,y}$ 计算为常数，对于固定的 y 我们可以在 $O(\frac{W}{w_i})$ 时间内计算出 $g_{x,y}$ ，求出所有 g 总时间复杂度为 $O(\frac{W}{w_i})\times O(w_i)=O(w)$ 。转移总复杂度 $O(nW)$ 。

7.5.2 斜率优化

适用类型 对于转移方程 dp[i]=min/max{dp[j]+f(j)+g(i)} ，我们可以通过移项（把关于 i 的放外面，关于 j 的在内部）和维护最小值（单点 or 单调队列）实现 $O(1)$ 转移。

对于转移方程 dp[i]=min/max{dp[j]+f(i,j)} ，由于涉及了 i 和 j，所以不能拆开。考虑使用斜率优化。

两种思路 解决斜率优化问题考虑两种思路。

代数 考虑 i 确定时，两个决策点 $0 \le j_1 < j_2$ ，且后者 j_2 优于 j_1。我们假设取小为优，那么可以得到以下方程：$dp[j_1] + f(i, j_1) \ge dp[j_2] + f(i, j_2)$ 通过移项等操作，最终能化为 $h(i) \ge \frac{F(j_1)-F(j_2)}{G(j_1)-G(j_2)}$ 的形式。(h(i) 为关于 i 的常数项，F(j) 包含有 dp[j] 项，G(j) 为关于 j 的常数项，大于小于号不确定)。

确定好优劣之后，我们不妨找到三个决策点 A<B<C, AB 斜率 K1，BC 斜率 K2，K0=h(i)。若 K2<K1，分情况讨论最优（> 代表优于）：

- (1)K0<K2<K1:A>B>C
- (2)K2<=K0<K1:A>B,C>B
- (3)K2<K1<K0:C>B>A

结果无论如何 B 都不是最优决策点，于是便可以删除该点。

最终可以看出，最优决策点组成了上 or 下凸包，其斜率递增 or 减。关键在于维护凸包。

线性规划 将原方程 dp[i]=min/max{dp[j]+f(i,j)} 转化为 y=kx+b 的模式，其中 f(i,j) 看做斜率 k 乘 x（k 为 i 相关，x 为 j 相关），dp[i] 作为目标在 b 中（出去第一步与 i 相关的所有值也在内），y 代表所有与 j 相关的值(包括 dp[j])。若 x 的表达式单调递减，推荐两端同乘-1 使其单增。

最小化 dp[i] 可以转化为最小化 b，斜率一定时，使得 b 最小化的点也在凸包中。

两种结合 两种方法结合来看。

线性规划表示十分直观，且在某变量不满足单调性时能快速判断。通过线性规划也可以看出，**最优决策点的左右两边斜率符号相反，自身为极大 or 小值，**可以通过二分得到。

代数法可以转换思维方向，通过斜率比较方便去点。

决策单调性的结合 一般而言，通过单调队列维护凸包点集。复杂度 $O(n \log n)$ 。

- (1) 在凸包上找到最优决策点 j。
- (2) 通过 j 更新 dp[i]。
- (3) 将 i 作为决策点加入。
- 注：若 i 为 dp[i] 的决策点，则 (3) 放在第一步。

如果具有决策单调性，即最优决策点递增。我们在更新 dp[i] 的时候，可以以 i 为媒介从尾部挨个枚举两个元素 j 和 j-1，确定是否删除 j-1。找到的第一个满足 j-1 优于 j 的点 j-1 便是最优决策点。在加入一个新点的时候，为了满足凸包，在头部删除。每个元素入队出队一次，复杂度 $O(n)$ 。

代码：玩具装箱

```
double k(int i,int j){
    return
    ↪  ((double)(f[i]-f[j]+(i+s[i])*(i+s[i])-(j+s[j])*(j+s[j]))
            /(double)(2.0*(i+s[i]-j-s[j])));
}
//main
head=tail=1; q[1]=0;
    for (int i=1;i<=n;++i){
        while (head<tail&&
                k(q[head],q[head+1])<=(double)(i-1-l+s[i]))
                head++;
        f[i]=f[q[head]]
            +(i-q[head]-1-l+s[i]-s[q[head]])
            *(i-q[head]-1-l+s[i]-s[q[head]]);
        while (head<tail&&
                k(q[tail-1],q[tail])>=k(q[tail],i)) tail--;
        q[++tail]=i;
}
```

注意事项：

- 先判断能不能用斜率优化，即写成 $\frac{Y(j_2)-Y(j_1)}{X(j_2)-X(j_1)}$ 的形式。
- 注意判断上凸还是下凸。
- 若存在 X(j) 相等的情况，注意设 INF 且注意符号问题。如果用叉积解决的话可以解除干扰。比较斜率是推荐不等式中包含等号，解决重复问题。
- 注意初始点的问题，可能不为 0。
- 斜率最好使用精度高的 long double，或者使用向量叉积解决。

单调性相关讨论

决策点横坐标 X(j) 单调　最方便处理的类型，直接单调队列 $O(n)$ 解决。

决策点横坐标 X(j) 不单调　不能使用单调队列，通过平衡树或者 CDQ 分治维护凸包单调性。

待决策点 h(i) 函数（或理解为斜率）不单调 不保证具有决策单调性，不能弹走队首，同时答案也不能直接是队首，需要二分。

代码：任务安排。

```
ll st[maxn],sc[maxn],dp[maxn];
int Q[maxn],head=1,tail=0;
il ll X(int j){return sc[j];}
il ll Y(int j){return dp[j];}
il long double K(int i,int j){
    if (X(i)==X(j)){
        if (Y(j)>=Y(i)) return 1e18;
        else return -1e18;
    }else{
        return (long double)(Y(j)-Y(i))/(X(j)-X(i));
    }
}
int check(int k){
    if (head==tail) return Q[head];
    int l=head,r=tail,mid;
    while (l<r){
        mid=(l+r)>>1;
        if (K(Q[mid],Q[mid+1])<k) l=mid+1;
        else r=mid;
    }
    return Q[l];
}
void work(){
    Q[++tail]=0;
    for (int i=1;i<=n;++i){
        int j=check(s+st[i]);
        dp[i]=dp[j]+st[i]*(sc[i]-sc[j])+s*(sc[n]-sc[j]);
        while (head<tail&&
               K(Q[tail-1],Q[tail])>=K(Q[tail-1],i)) --tail;
        Q[++tail]=i;
    }
}
```

X(j),h(i) 都不单调 利用平衡树维护凸包，通过平衡树也能方便查到 h(i) 的前驱后继。如果用 CDQ 需加一维偏序。

7.5.3 四边形不等式优化

2D1D(区间类)动态规划应用 如以下形式的转移方程：$f_{l,r}=\min\limits_{k=l}^{r-1}\{f_{l,k}+f_{k+1,r}\}+w(l,r)$

朴素转移 $O(n^3)$ ，但若 $w(l,r)$ 满足以下性质时，我们可以通过**决策单调性**进行优化。(**决策单调性**：随状态增大，最优决策点非严格递增。)

- 1. **区间包含单调性**：对于任意 $l \le l' \le r' \le r$ ，都有 $w(l',r') \le w(l,r)$。
- 2. **四边形不等式**：对于任意 $l_1 \le l_2 \le r_1 \le r_2$ ，都有 $w(l_1,r_1)+w(l_2,r_2) \le w(l_1,r_2)+w(l_2,r_1)$ 成立。

相关定理：

- $w(l,r)$ 满足区间包含单调性的同时满足四边形不等式，则 $f_{l,r}$ 也满足四边形不等式。
- 若 f 满足四边形不等式，记 $m_{l,r}=\min\{k : f_{l,r}=f_{l,k}+f_{k+1,r}+w(l,r)\}$ 表示最优决策点，则 $m_{l,r-1} \le m_{l,r} \le m_{l+1,r}$ 。

因此在计算 $f_{l,r}$ 的时候可以记录最优决策点 $m_{l,r}$ ，那么对决策点 k 的枚举总量就将为 n^2 。

```
for (int len=2;len<=n;++len){
    for (int l=1,r=len;r<=len;++l,--r){
        f[l][r]=INF;
        for (int k=m[l][r-1];k<=m[l+1][r];++k){
            if (f[l][r]>f[l][r]+f[k+1][r]+w(l,r)){
                f[l][r]=f[l][k]+f[k+1][r]+w(l,r);
                m[l][r]=k;
            }
        }
    }
}
```

1D1D 动态规划应用 如以下形式的转移方程：$f_r=\min\limits_{l=1}^{r-1}\{f_l+w(l,r)\}$

若函数 $w(l,r)$ 满足四边形不等式，记 $m_r=\min\{l|f_r=f_l+w(l,r)\}$ ，则 $\forall r_1<r_2 : k_{r_1} \le k_{r_2}$。

我们只能得知枚举 l 的下界但无法确定上界。但可以解决一种特殊情况：$f_r=\min\limits_{l=1}^{r-1}\{w(l,r)\}$ 。我们定义 $dp(l,r,k_l,k_r)$ 表示求 $f_l \to f_r$ 的状态值，并且知道其最优决策点必然在 $[k_l,k_r]$ 中，可以使用分治算法解决。复杂度 $O(n\log n)$ 。

```
void dp(int l,int r,int kl,int kr){
    int mid=(l+r)>>1,k=kl;
    for (int i=kl;i<=min(kr,mid-1);++i){
        if (w(i,mid)<w(k,mid)) i=k;
    }
    f[mid]=w(k,mid);
```

```
    if (l<mid) dp(l,mid-1,kl,k);
    if (r>mid) dp(mid+1,r,k,k_r);
}
```

满足四边形不等式的常见性质

- 若 $w_1(l,r)$ 和 $w_2(l,r)$ 同时满足四边形不等式（或区间包含单调性），则 $c_1w_1 + c_2w_2$ 同样满足四边形不等式（或区间包含单调性）（$c_1 \geq 0, c_2 \geq 0$）
- 若存在函数 $f(x), g(x)$ 满足 $w(l,r) = f(r) - g(l)$ ，则 $w(l,r)$ 满足四边形恒等式（四边形不等式不等号变等号）。当函数 $f(x), g(x)$ 单调增加时，函数 $w(l,r)$ 还满足区间包含单调性。
- 设 $h(x)$ 是一个单增的凸函数（下凸函数，导数单调增加），若函数 $w(l,r)$ 满足四边形不等式且区间包含单调性，则函数 $h(w(l,r))$ 满足四边形不等式且区间包含单调性。
- 设 $h(x)$ 是一个凸函数，若函数 $w(l,r)$ 满足四边形不等式且区间包含单调性，则函数 $h(w(l,r))$ 满足四边形不等式。