ICPC Templates

Yelson

November 13, 2021

Contents

1 字符串

1.1 STL-String

```
/*

- `find(ch, start = 0)` 查找并返回从 `start` 开始的字符 `ch` 的位置；`rfind(ch)` 从末尾开始，查找并返回第一个找到的字符 `ch` 的位置（皆从 `0` 开始）（如果查找不到，返回 `-1`）。

- `substr(start, len)` 可以从字符串的 `start`（从 `0` 开始）截取一个长度为 `len` 的字符串（缺省 `len` 时代码截取到字符串末尾）。

- `append(s)` 将 `s` 添加到字符串末尾。

- `append(s, pos, n)` 将字符串 `s` 中，从 `pos` 开始的 `n` 个字符连接到当前字符串结尾。


- `replace(pos, n, s)` 删除从 `pos` 开始的 `n` 个字符，然后在 `pos` 处插入串 `s`。


- `erase(pos, n)` 删除从 `pos` 开始的 `n` 个字符。


- `insert(pos, s)` 在 `pos` 位置插入字符串 `s`。例：s.insert(0," ");



- string 转换为int：`atoi(str.c_str())` （同样还有float型atof(),long型atol()等）

- int转换为string ：`to_string(int)`

*/
```

1.2 Manacher

```
#include <bits/stdc++.h>
#define maxn 2000005
using namespace std;
int mp[maxn];
string str;
char c[maxn];
void Manacher(string s,int len){
    int l=0,R=0,C=0;;
    c[l++]='$', c[l++]='#';
    for(int i=0;i<len;i++){
        c[l++]=s[i], c[l++]='#';
    }
    for(int i=0;i<l;i++){
        mp[i]=R>i?min(mp[2*C-i],R-i):1;
        while(i+mp[i]<l&&i-mp[i]>0){
            if(c[i+mp[i]]==c[i-mp[i]]) mp[i]++;
            else break;
```

```
        }
        if(i+mp[i]>R){
            R=i+mp[i], C=i;
        }
    }
}
int main()
{
    int cnt=0;
    while(cin>>str){
        if(str=="END") break;
        int len=str.length();
        Manacher(str,len);
        int ans=0;
        for(int i=0;i<2*len+4;i++){
            ans=max(ans,mp[i]-1);
        }
        printf("Case %d: %d\n",++cnt,ans);
    }
    return 0;
}

/*
求以任一点为中心的回文半径
O(n)
*/
const int maxl=1100005;
int p[2*maxl+5]; //p[i]-1表示以i为中点的回文串长度

int Manacher(string &s)
{
    string now;
    int len=s.size();
    for(int i=0;i<len;i++) //将原串处理成%a%b%c%形式，保证长度为奇数
    {
       now+='%';
       now+=s[i];
    }
    now+='%';
    len=now.size();
    int pos=0,R=0;
    for (int i=0;i<len;i++)
    {
        if (i<R) p[i]=min(p[2*pos-i],R-i); else p[i]=1;
        while (0<=i-p[i]&&i+p[i]<len&&now[i-p[i]]==now[i+p[i]]) p[i]++;
        if (i+p[i]>R) {pos=i;R=i+p[i];}
    }
    int MAX=0;
    for (int i=0;i<len;i++)
    {
        //p[i]-1为now串中以i为中点的回文半径，即是s中最长回文串的长度
        cout<<i<<" : "<<p[i]-1<<"\n";
        cout<<now.substr(i-p[i]+1,2*p[i]-1)<<"\n";
```

```
71          MAX=max(MAX,p[i]-1);
72      }
73      return MAX; //最长回文子串长度
74  }
75
76  /*
77  罗老师版马拉车
78  求以任一点为中心的回文半径
79  O(n)
80  */
81  const int maxn = 2e5;
82
83  string Mnc(string &s)
84  {
85      string t = "$#";
86      for (int i = 0; i < s.length(); ++i) //构造辅助串
87      {
88          t += s[i];
89          t += '#';
90      }
91      int ml = 0, p = 0, R = 0, M = 0;
92      //最大长度，最长回文中心，当前最大回文串右端，当前最长回文中心
93      int len = t.length();
94      vector<int> P(len, 0); //回文长度数组
95      for (int i = 0; i < len; ++i)
96      {
97          P[i] = R > i ? min(P[2 * M - i], R - i) : 1; //转移方程
98
99          while (t[i + P[i]] == t[i - P[i]]) //长度扩张
100             ++P[i];
101
102         if (i + P[i] > R) //更新右端和中心
103         {
104             R = i + P[i];
105             M = i;
106         }
107         if (ml < P[i]) //记录极大
108         {
109             ml = P[i];
110             p = i;
111         }
112     }
113     return s.substr((p - ml) / 2, ml - 1); //返回回文串
114 }
```

1.3 字符串 Hash

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  #define ull unsigned long long
5  const int N=1e6+10;
```

```
6  int n,m;
7
8  //一维Hash【自然溢出】
9  struct HashStr1d
10 {
11     const static int N=1e6+10,p=131;
12     ull bs[N],h[N];
13     string s;
14
15     void init(int n)
16     {
17         bs[0]=1;
18         for(int i=1;i<=n;i++) bs[i]=bs[i-1]*p;
19     }
20
21     //s下标从0开始; hash下标从1开始
22     void sethash(string &s)
23     {
24         h[0]=0;
25         for(int i=1;i<=(int)s.size();i++)
26             h[i] = h[i-1]*p + (s[i-1]-'a'+1);
27     }
28
29     ull gethash(int l,int r)
30     {
31         return h[r]-h[l-1]*bs[r-l+1];
32     }
33 };
34
35 //二维Hash【自然溢出】 - 常用于计算矩阵中子矩阵出现次数相关问题
36 struct HashStr02
37 {
38     //!N记得根据具体题目改
39     const static int N=2e3+10,px=131,py=13331;
40     ull bsx[N],bsy[N],h[N][N];
41     string s[N];
42     int mx,my;
43
44     void init(int n,int m)
45     {
46         bsx[0]=bsy[0]=1;
47         mx=n,my=m;
48         for(int i=1;i<=n;i++) bsx[i]=bsx[i-1]*px;
49         for(int i=1;i<=m;i++) bsy[i]=bsy[i-1]*py;
50     }
51
52     //s[i][j]: i下标从1开始, j下标从0开始; hash下标从1开始
53     void sethash()
54     {
55         for(int i=1;i<=mx;i++)
56         {
57             h[i][0] = 0;
58             for(int j=1;j<=my;j++)
```

```
            h[i][j] = h[i-1][j]*px + h[i][j-1]*py - h[i-1][j-1]*px*py
                   + (s[i][j-1]-'a'+1);
        }
    }

    ull gethash(int x1,int y1,int x2,int y2)
    {
        ull ret = h[x2][y2];
        ret -= h[x1-1][y2] * bsx[x2-x1+1];
        ret -= h[x2][y1-1] * bsy[y2-y1+1];
        ret += h[x1-1][y1-1] * bsx[x2-x1+1] * bsy[y2-y1+1];
        return ret;
    }
};

HashStr02 hstr;

/*
链接：https://ac.nowcoder.com/acm/problem/13610
来源：牛客网

给出一个n * m的矩阵。让你从中发现一个最大的正方形。使得这样子的正方形在矩阵中出现了至少两次。输出最大正方形的边长。
*/

bool check(int len)
{
    unordered_map<ull,int> hs;
    for(int i=1;i<=n-len+1;i++)
    {
        for(int j=1;j<=m-len+1;j++)
        {
            ull val = hstr.gethash(i,j,i+len-1,j+len-1);
            if(hs[val]) return true;
            hs[val]++;
        }
    }
    return false;
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    hstr.init(n,m);
    for(int i=1;i<=n;i++) cin>>hstr.s[i];
    hstr.sethash();

    int l=1,r=min(n,m);
    while(l<r)
    {
        int mid=l+r+1>>1;
        if(check(mid)) l=mid;
```

```
111         else r=mid-1;
112     }
113     cout<<l<<"\n";
114 
115     return 0;
116 }
117 
118 
119 //=====双质数======
120 /**
121  * 【双质数hash】
122 */
123 #include<bits/stdc++.h>
124 using namespace std;
125 
126 #define ll long long
127 const int N=4e5+100;
128 
129 //**朴素版**
130 // const int base1=13331,base2=233333,mod1=1019260817,mod2=1e9+7;
131 // ll bs1[N],bs2[N],hs1[N],hs2[N];
132 
133 // void init()
134 // {
135 // bs1[0]=bs2[0]=1;
136 // for(int i=1;i<N;i++)
137 // bs1[i]=bs1[i-1]*base1%mod1,bs2[i]=bs2[i-1]*base2%mod2;
138 // }
139 
140 // void sethash(string &s)
141 // {
142 // hs1[0]=hs2[0]=0;
143 // for(int i=1;i<=s.size();i++)
144 // {
145 // hs1[i]=hs1[i-1]*base1+(s[i-1]-'a'+1);
146 // hs1[i]%=mod1;
147 
148 // hs2[i]=hs2[i-1]*base2+(s[i-1]-'a'+1);
149 // hs2[i]%=mod2;
150 // }
151 // }
152 
153 // ll gethash(int l,int r)
154 // {
155 // ll h1=(hs1[r]-hs1[l-1]*bs1[r-l+1]%mod1+mod1)%mod1;
156 // ll h2=(hs2[r]-hs2[l-1]*bs2[r-l+1]%mod2+mod2)%mod2;
157 // return h1*mod2+h2;
158 // }
159 
160 //**封装版**
161 struct hsh{
162     static const int mod1=1019260817,mod2=1000000007;
163     int x,y;
```

```
164     hsh(int x=0,int y=0):x(x),y(y){}
165     hsh operator + (const hsh&a)const{
166         return hsh((x+a.x)%mod1,(y+a.y)%mod2);
167     }
168     hsh operator - (const hsh&a)const{
169         return hsh((x-a.x+mod1)%mod1,(y-a.y+mod2)%mod2);
170     }
171     hsh operator * (const hsh&a)const{
172         return hsh((ll)x*a.x%mod1,(ll)y*a.y%mod2);
173     }
174     ll val(){return (ll)x*mod2+y;}
175 }base(63485839,54958295);
176 
177 struct hsh2
178 {
179     hsh bs[N],hs[N];
180     void init()
181     {
182         bs[0]={1,1};
183         for(int i=1;i<N;i++) bs[i]=bs[i-1]*base;
184     }
185 
186     void sethash(string &s)
187     {
188         hs[0]={0,0};
189         for(int i=1;i<=s.size();i++)
190             hs[i]=hs[i-1]*base+hsh(s[i-1]-'a'+1,s[i-1]-'a'+1);
191     }
192 
193     ll gethash(int l,int r)
194     {
195         return (hs[r]-hs[l-1]*bs[r-l+1]).val();
196     }
197 };
198 
199 hsh2 hs2;
200 
201 int n;
202 string s;
203 unordered_map<ll,ll> cthave,ctneed;
204 
205 int main()
206 {
207     ios::sync_with_stdio(0);cin.tie(0);
208     hs2.init();
209     ll ans=0;
210     cin>>n;
211     for(int k=1;k<=n;k++)
212     {
213         cin>>s;
214         ll len=s.size();
215         hs2.sethash(s);
216         ll cur=hs2.gethash(1,len);
```

```
        ans+=ctneed[cur];
        cthave[cur]++;
        ctneed[cur]++;
        for(ll i=1;i<=len/2;i++)
        {
            if(i+1>len-i) break;
            ll l=hs2.gethash(1,i),r=hs2.gethash(len-i+1,len);
            if(l==r)
            {
                ans+=cthave[hs2.gethash(i+1,len-i)];
                ctneed[hs2.gethash(i+1,len-i)]++;
                // cout<<gethash(i+1,len-i)<<"\n";
            }
        }
    }
    cout<<ans<<"\n";

    return 0;
}
```

1.4 KMP

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

const int N=2e6+10;
int nx[N];

//nx[0]=nx[1]=0,t下标从1开始
void getnx00(string &t,int *nx)
{
    int n=t.size();
    for(int i=2,j=0;i<n;i++)
    {
        while(j && t[i]!=t[j+1]) j=nx[j];
        if(t[i]==t[j+1]) j++;
        nx[i]=j;
    }
}

//nx[0]=0,t下标从0开始
void getnx01(string &t,int *nx)
{
    int n=t.length();
    for(int i=1,j=0;i<n;i++)
    {
        while(j && t[i]!=t[j]) j=nx[j-1];
        if(t[i]==t[j]) j++;
        nx[i]=j;
    }
}

```

```cpp
void kmp01(string &s,string &t,int *nx)
{
    int cnt=0;
    int n=s.length(),m=t.length();
    for(int i=0,j=0;i<n;i++)
    {
        while(j && s[i]!=t[j]) j=nx[j-1];
        if(s[i]==t[j]) j++;

        if(j==m)
        {
            cout<<(i-j+1)+1<<"\n";
            cnt++;
            j=nx[j-1];
        }
    }
}

//基于前缀函数的KMP
void kmp00(string &s,string &t,int *nx)
{
    s = t + '#' + s;
    int n=s.length();
    int m=t.length();
    getnx02(s,nx);

    for(int i=m+1;i<n;i++)
        if(nx[i]==m)
            cout<<(i-2*m+1)<<"\n";

    for(int i=0;i<m;i++)
        cout<<nx[i]<<(i==m-1?"\n":" ");
}

//nx[0]=-1的写法：注意，nx[i]表示前缀[0,i-1]中后缀与前缀相等的最大长度
void getnx02(string &t,int *nx)
{
    memset(nx,0,sizeof nx);
    int len=t.length();
    int j = nx[0] = -1;
    for (int i = 0; i < len; i++)
    {
        while(j!=-1 && t[i]!=t[j]) j=nx[j];
        nx[i+1]=++j;
    }
}

void kmp02(string &s,string &t,int *nx)
{
    int cnt=0;
    int n=s.length(),m=t.length();
    for(int i=0,j=0;i<n;i++)
    {
```

```
        while(j!=-1 && s[i]!=t[j]) j=nx[j];
        j++;

        if(j==m)
        {
            cout<<(i-j+1)+1<<"\n";
            cnt++;
            j=nx[j];
        }
    }
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    string s,t;
    cin>>s>>t;
    kmp00(s,t,nx);

    return 0;
}



/* KMP求循环节
字符串S的最小循环节长度为 L =S.length() - nx[S.length()-1].

- 若S.length()能被L整除，
则说明S可以完全由该循环节组成，循环周期T=S.length()/L.

- 若不能整除，则说明还需要在末尾补充一段字符串，
才能将S变成完全由循环节组成的形式。
补充的长度为ADD= L - S.length()%L.

AcWing 141. 周期
多组测试样例，
每组给定一字符串，输出具有循环节的前缀的长度 i 和其对应的最大循环节个数K（K>1）。
*/

#include<iostream>
using namespace std;

const int N=1e6+10;
int n;
string s;
int nex[N];

//下标从1开始
void getnex()
{
    for(int i=2,j=0;i<=n;i++)
    {
        while(j&&s[i]!=s[j+1]) j=nex[j];
```

```
        if(s[i]==s[j+1]) j++;
        nex[i]=j;
    }
}

int main()
{
    int t=0;
    while(cin>>n)
    {
        if(n==0) break;
        cout<<"Test case #"<<++t<<"\n";
        cin>>s;
        s=' '+s;
        getnex();
        for(int i=2;i<=n;i++)
        {
            int len=i-nex[i];
            //判断循环节存在的条件:len!=i是因为要求K>1
            if(len!=i && i%len==0)
                cout<<i<<" "<<i/len<<"\n";
        }
        cout<<"\n";
    }
    return 0;
}


/* 失配树
Border(s) 为对于 i i∈[1,∈s∈), 满足 pre_i = suf_i的字符串 pre_i的集合。
Border(s) 中的每个元素都称之为字符串 s 的 border。
有 m 组询问, 每组询问给定 p,q, 求 s 的 p 前缀 和 q 前缀 的 最长公共 border 的长度。

SOLUTION: 建 Fail树, 然后 倍增LCA - O(n+mlogn)
*/
#include<bits/stdc++.h>
using namespace std;

const int N=1e6+100;
string s;
int m,p,q;
int nx[N];
vector<int> e[N];
int fa[N][22],dep[N];

void solve_simplify()
{
    int n=s.size();
    s.insert(0," ");
    for(int i=2,j=0;i<=n;i++)
    {
        while(j && s[i]!=s[j+1]) j=nx[j];
        if(s[i]==s[j+1]) j++;
```

```
        nx[i]=j;
        fa[i][0]=j,dep[i]=dep[j]+1;
    }

    for(int i=1;i<=20;i++)
        for(int j=1;j<=n;j++)
            fa[j][i]=fa[fa[j][i-1]][i-1];
    cin>>m;
    while (m--)
    {
        cin>>p>>q;
        if(dep[p]<dep[q]) swap(p,q);
        for(int i=20;i>=0;i--) if(dep[fa[p][i]]>=dep[q]) p=fa[p][i];
        for(int i=20;i>=0;i--) if(fa[p][i]!=fa[q][i]) p=fa[p][i],q=fa[q][i];
        cout<<fa[p][0]<<"\n";
    }
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>s;
    solve_simplify();

    return 0;
}


/*
【最小覆盖子矩阵】
https://ac.nowcoder.com/acm/contest/20323/K

1.KMP求最小覆盖子矩阵大小
对行和列分别hash，然后分别kmp，
得到关于行的最小循环节长度x和关于列的的最小循环节长度y
x,y 即为最小覆盖子矩阵大小

同时，矩阵中任意大小为 x*y 的子矩阵均能作为覆盖子矩阵

2.单调栈求所有大小为x*y的矩阵中的最大值
*/
#include <bits/stdc++.h>
using namespace std;

#define ull unsigned long long
const int N = 1e6 + 100, p = 131;
int n, m;
string s;
vector<vector<int>> a;
vector<vector<int>> miy;
int nx[N];

ull hx[N],hy[N];
```

```cpp

int kmp(ull *h,int n)
{
    for(int i=2,j=0;i<=n;i++)
    {
        while(j && h[i]!=h[j+1]) j=nx[j];
        if(h[i]==h[j+1]) j++;
        nx[i]=j;
    }
    return n-nx[n];
}

vector<int> getmx(vector<int> &a,int n,int len)
{
    deque<int> q;
    vector<int> ret(n+1);
    for(int i=1;i<=n;i++)
    {
        while(q.size() && i-q.front()>=len) q.pop_front();
        while(q.size() && a[q.back()]<a[i]) q.pop_back();
        q.push_back(i);
        ret[i]=a[q.front()];
    }
    return ret;
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> s, s.insert(0, " ");
        for(int j=1;j<=m;j++) hx[i]=hx[i]*p+(s[j]-'a'+1),hy[j]=hy[j]*p+(s[j]-'a'+1);
    }

    a.resize(n+1,vector<int>(m+1));
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> a[i][j];

    //**求最小覆盖子矩阵大小
    int x=kmp(hx,n), y=kmp(hy,m);
    // cout<<x<<" "<<y<<"\n";

    //**求覆盖子矩阵内的最小最大值
    //注：求所有大小为 x*y 的子矩阵的最大值中的最小值即可。
    //可看出性质，所有大小为 x*y 的子矩阵（无论是不是上面求出的最小覆盖子矩阵本身），
    //其无限扩展后，必能包含原最小覆盖子矩阵，
    //故所有大小为 x*y 的子矩阵都是合法的选择。
    miy.resize(n+1);
    for(int i=1;i<=n;i++) miy[i]=getmx(a[i],m,y);
```

```
    int mi=2e9;
    for(int j=y;j<=m;j++)
    {
        vector<int> tmp(1,0),vc;
        for(int i=1;i<=n;i++) tmp.push_back(miy[i][j]);
        vc = getmx(tmp,n,x);
        for(int i=x;i<=n;i++) mi=min(mi,vc[i]);
    }

    // cout<<mi<<"\n";
    cout<<(1ll*mi*(x+1)*(y+1))<<"\n";

    return 0;
}
```

1.5 Trie

```
//========Trie==========
const int N=1e5+10;
int trie[N][26],idx;
int cnt[N]; // 以该结点结尾的字符串的存在数量，不存在则为0

void insert(string &t)
{
  int p=0;
  for(int i=0;i<t.length();i++)
  {
    int &s=trie[p][t[i]-'a'];
    if(!s) s=++idx;
    p=s;
  }
  cnt[p]++;
}

int find(string &t)
{
  int p=0;
  for(int i=0;i<t.length();i++)
  {
    int &s=trie[p][t[i]-'a'];
    if(!s) return 0;
    p=s;
  }
  return cnt[p]; //返回该字符串在字典中出现的次数
}

//========01Trie==========
/*
在给定的 N 个整数 A1，A2……AN 中选出两个进行 xor（异或）运算，得到的结果最大是多少
*/
#include<iostream>
using namespace std;
```

```cpp

const int N=1e5+10;
int n,a[N];
int ans;
int trie[N*30][2],idx;

//从高位插入，优先比较高位，从而计算最大值
void insert(int t)
{
    int p=0;
    for(int i=30;~i;i--) //int类型位数为31，最高右移30位（30>=i>=0）
    {
        int &s=trie[p][t>>i&1];
        if(!s) s=++idx;
        p=s;
    }
}

int query(int t)
{
    int p=0,ret=0;
    for(int i=30;~i;i--)
    {
        int &s=trie[p][!(t>>i&1)];
        if(s)
        {
            ret+=1<<i;
            p=s;
        }
        else p=trie[p][t>>i&1];
    }
    return ret;
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>a[i],insert(a[i]);
    for(int i=0;i<n;i++)
        ans=max(ans,query(a[i]));
    cout<<ans<<"\n";
    return 0;
}

/*
给定一个非负整数数列 a，初始长度为 N。
请在所有长度不超过 M 的连续子数组中，找出子数组异或和的最大值。
子数组的异或和即为子数组中所有元素按位异或得到的结果。
注意：子数组可以为空。

SOLUTION: 异或前缀和+01Trie删点
```

```
$S[]$数组为异或前缀和;
通过删除$S[i-m-1]$维护Trie中只包含${S[i-m],...,S[i]}$
$S[i] ^ S[j] = a[i+1]^...^ a[j] $ ，^ 为异或符号。
故该问题转化为 求从${S[i-m-1],...,S[i]}$取一组$S[i]$,$S[j]$（i-m-1≤j≤i）$所得的最大异或值。

sum[]为异或前缀和
sum[0]^sum[m] = a1^a2^...^am
sum[i]^sum[j] = a(i+1)^a(i+2)^...^aj
*/

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

const int N=1e5+10;
int n,m;
int a[N],sum[N],ans;
int tr[N*31][2],idx;
int cnt[N*31];

void insert(int x,int c)
{
    int p=0;
    for(int i=31;i>=0;i--)
    {
        int &s=tr[p][x>>i&1];
        if(!s) s=++idx;
        p=s;
        cnt[p]+=c; //记录节点出现次数，删点即减少该节点出现次数
    }
}

int query(int x)
{
    int ret=0,p=0;
    for(int i=31;i>=0;i--)
    {
        int &s=tr[p][!(x>>i&1)];
        if(s && cnt[s]) //若该节点存在且出现次数不为0，则可以选择该该节点
        {
            ret^=1<<i;
            p=s;
        }else
            p=tr[p][x>>i&1];
    }
    return ret;
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        cin>>a[i],sum[i]=sum[i-1]^a[i];
```

```
142
143     insert(sum[0],1);
144
145     for(int i=1;i<=n;i++)
146     {
147         insert(sum[i],1);
148
149         //维护Trie中包含S[i-m]...S[i]
150         if(i>m) insert(sum[i-m-1],-1);
151         ans=max(ans,query(sum[i]));
152     }
153     cout<<ans;
154
155     return 0;
156 }
157
158 /*
159 给定一颗树和每条边两个相邻节点的异或值，和每个节点的取值范围 Li～Ri，求有多少种取点的合法方案。
160 n个限制：li<=x^ai<=ri；求合法的x的个数
161 */
162 #include<bits/stdc++.h>
163 using namespace std;
164
165 const int N=1e5+10,M=N*100;
166 int n;
167 int l[N],r[N],w[N];
168 vector<pair<int,int>> e[N];
169
170 int tr[M][2],idx;
171 int sum[M],cnt[M];
172
173 //插入x
174 void insert(int p,int pos,int id,int f) //f判断按哪条路径插入
175 {
176     if(pos<0)
177     {
178         sum[p]++;
179         return;
180     }
181     int cl=l[id],cr=r[id],cw=w[id];
182     int vl=cl>>pos&1;
183     int vr=cr>>pos&1;
184     int vw=cw>>pos&1;
185
186     int &s0=tr[p][vw]; //x^wi为0时，x该位的值
187     int &s1=tr[p][!vw]; //x^wi为1时，x该位的值
188
189     if(f==1) //沿l的路径插入
190     {
191         if(vl==0) //l该位为0时，x^wi可取0/1，且取1时，无论后面位如何选择都合法
192         {
193             if(!s1) s1=++idx;
194             sum[s1]++;
```

```
195
196             if(!s0) s0=++idx;
197             insert(s0,pos-1,id,f);
198         }
199         else if(vl==1) //l该位为1时，x^wi只可取1
200         {
201             if(!s1) s1=++idx;
202             insert(s1,pos-1,id,f);
203         }
204     }
205     else if(f==2) //沿r的路径插入
206     {
207         if(vr==0) //r该位为0时，x^wi只可取0
208         {
209             if(!s0) s0=++idx;
210             insert(s0,pos-1,id,f);
211         }
212         else if(vr==1) //r该位为1时，x^wi可取0/1，且取0时，无论后面位如何选择都合法
213         {
214             if(!s0) s0=++idx;
215             sum[s0]++;
216
217             if(!s1) s1=++idx;
218             insert(s1,pos-1,id,f);
219         }
220     }
221     else //沿l和r插入，当l和r第一次出现位置上的值不同时，分解成沿l插入和沿r插入
222     {
223         if(vl==0 && vr==0) //l和r该位同值时，x^wi取值唯一
224         {
225             if(!s0) s0=++idx;
226             insert(s0,pos-1,id,f);
227         }
228         else if(vl==1 && vr==1)
229         {
230             if(!s1) s1=++idx;
231             insert(s1,pos-1,id,f);
232         }
233         else if(vl==0 && vr==1) //当l和r第一次出现位置上的值不同时，分解成沿l插入和沿r插入
234         {
235             if(!s0) s0=++idx;
236             insert(s0,pos-1,id,1);
237
238             if(!s1) s1=++idx;
239             insert(s1,pos-1,id,2);
240         }
241     }
242 }
243
244 int query(int p,int pos,int x)
245 {
246     // if(pos<0) cout<<">:"<<x<<" - "<<sum[p]<<"\n";
247     if(sum[p]==n) return (1<<pos+1); //当前结点被覆盖n次时，返回长度
```

```
248     if(pos<0) return 0;
249     int ret=0;
250     if(tr[p][0])
251     {
252         sum[tr[p][0]] += sum[p];
253         ret+=query(tr[p][0],pos-1,x);
254     }
255     if(tr[p][1])
256     {
257         sum[tr[p][1]] += sum[p];
258         ret+=query(tr[p][1],pos-1,x^(1<<pos));
259     }
260     return ret;
261 }
262 
263 void dfs(int u,int fa)
264 {
265     for(auto i:e[u])
266     {
267         int v=i.first,cw=i.second;
268         if(v==fa) continue;
269         w[v]=w[u]^cw;
270         dfs(v,u);
271     }
272 }
273 
274 int main()
275 {
276     ios::sync_with_stdio(0);cin.tie(0);
277     cin>>n;
278     for(int i=1;i<=n;i++) cin>>l[i]>>r[i];
279     for(int i=0,u,v,w;i<n-1;i++)
280     {
281         cin>>u>>v>>w;
282         e[u].push_back({v,w});
283         e[v].push_back({u,w});
284     }
285     dfs(1,0);
286     for(int i=1;i<=n;i++)
287         insert(0,29,i,0);
288 
289     cout<<query(0,29,0)<<"\n";
290 
291     return 0;
292 }
293 
294 
295 //========可持久化01Trie=========
296 /*
297 给定一个非负整数序列 a，初始长度为 N。
298 有 M 个操作，有以下两种操作类型：
299 A x：添加操作，表示在序列末尾添加一个数 x，序列的长度 N 增大 1。
300 Q l r x：询问操作，你需要找到一个位置 p，满足 l≤p≤r，使得：a[p] xor a[p+1] xor … xor a[N]
```

```
    xor x 最大，输出这个最大值。
301
302 推公式：ans = max{ s[p-1]^s[n]^x } (l<=p<=r)
303 其中s[n]^x为定值，故只需在[l-1,r-1]区间查询满足要求的s[p-1]即可；
304
305 查询方法为：
306 在root[r-1]版本的01Trie中查询所有id>=l-1的路径，返回一个最大异或值。
307
308 */
309
310 #include<bits/stdc++.h>
311 using namespace std;
312
313 const int N=6e5+10, M=N*25;
314 int n,m;
315 int s[N];
316
317 int tr[M][2],idx;
318 int cnt[M],max_id[M];
319 int root[N];
320
321 //递归
322 void insert(int i,int k,int p,int q)
323 {
324     if(k<0)
325     {
326         max_id[q]=i;
327         return ;
328     }
329
330     int v=s[i]>>k&1;
331     if(p) tr[q][v^1] = tr[p][v^1];
332     tr[q][v] = ++idx;
333     insert(i,k-1,tr[p][v],tr[q][v]);
334
335     max_id[q] = max(max_id[tr[q][0]], max_id[tr[q][1]]);
336 }
337
338 //非递归
339 void insert(int k,int p,int q)
340 {
341     max_id[q] = k;
342     for(int i=23;i>=0;i--)
343     {
344         int v = s[k] >> i & 1;
345         if(p) tr[q][v^1] = tr[p][v^1];
346         tr[q][v] = ++idx;
347
348         q = tr[q][v];
349         p = tr[p][v];
350         max_id[q] = k;
351     }
352 }
```

```
353
354 int query(int root,int c,int l)
355 {
356     int p=root;
357     for(int i=23;i>=0;i--)
358     {
359         int v=c>>i&1;
360         if(max_id[tr[p][v^1]] >= l) p=tr[p][v^1];
361         else p=tr[p][v];
362     }
363
364     return c ^ s[max_id[p]];
365 }
366
367
368 int main()
369 {
370     ios::sync_with_stdio(0);cin.tie(0);
371     cin>>n>>m;
372
373     max_id[0]=-1;
374     root[0]=++idx;
375     //insert(0,23,0,root[0]);
376     insert(0,0,root[0]);
377
378     int a;
379     for(int i=1;i<=n;i++)
380     {
381         cin>>a;
382         root[i]=++idx;
383         s[i]=s[i-1]^a;
384         //insert(i,23,root[i-1],root[i]);
385         insert(i,root[i-1],root[i]);
386     }
387
388     char op;
389     while(m--)
390     {
391         cin>>op;
392         if(op=='A')
393         {
394             n++;
395             cin>>a;
396             s[n] = s[n-1]^a;
397             root[n] = ++idx;
398             //insert(n,23,root[n-1],root[n]);
399             insert(n,root[n-1],root[n]);
400         }else if(op=='Q')
401         {
402             int l,r,x;
403             cin>>l>>r>>x;
404             cout<<query(root[r-1],s[n]^x,l-1)<<"\n";
405         }
```

```
406     }
407 
408     return 0;
409 }
410 
411 
412 //========XorMST=========
413 /*
414 2020ICPCMacau C.Club Assignment
415 有n个数，现在要把他们拆分成两个集合，假设S为集合，有如下定义：
416 f(S)={min(x⊕y)⊕x,y⊕S and x≠y}
417 将n个数拆分为两个集合A,B，要求最大化min(f(A),f(B)),并输出划分集合的方案，如果有多种，则输出任意
        一个.
418 */
419 #include<bits/stdc++.h>
420 using namespace std;
421 
422 #define ll long long
423 #define pi pair<int,int>
424 const int N=2e5+10,M=N*31;
425 int tr[M][2],idx;
426 int l[M],r[M];
427 
428 int tr1[M][2],tr2[M][2],idx1,idx2;
429 
430 int t,n;
431 pi a[N];
432 vector<int> as,vs;
433 vector<int> vs1,vs2;
434 
435 vector<int> e[N];
436 
437 inline void insert1(int x)
438 {
439     int p=0;
440     for(int i=30;i>=0;i--)
441     {
442         vs1.push_back(p);
443         int &s=tr1[p][x>>i&1];
444         if(!s) s=++idx1;
445         p=s;
446     }
447     vs1.push_back(p);
448 }
449 inline void insert2(int x)
450 {
451     int p=0;
452     for(int i=30;i>=0;i--)
453     {
454         vs2.push_back(p);
455         int &s=tr2[p][x>>i&1];
456         if(!s) s=++idx2;
457         p=s;
```

```
458     }
459     vs2.push_back(p);
460 }
461
462 inline int find1(int x)
463 {
464     int ret=0,p=0;
465     for(int i=30;i>=0;i--)
466     {
467         int v=x>>i&1;
468         if(tr1[p][v]) p=tr1[p][v];
469         else if(tr1[p][v^1]) p=tr1[p][v^1],ret^=1<<i;
470         else return ret;
471     }
472     return ret;
473 }
474 inline int find2(int x)
475 {
476     int ret=0,p=0;
477     for(int i=30;i>=0;i--)
478     {
479         int v=x>>i&1;
480         if(tr2[p][v]) p=tr2[p][v];
481         else if(tr2[p][v^1]) p=tr2[p][v^1],ret^=1<<i;
482         else return ret;
483     }
484     return ret;
485 }
486
487 inline void insert(int x,int id)
488 {
489     int p=0;
490     for(int i=30;i>=0;i--)
491     {
492         vs.push_back(p);
493         int &s=tr[p][x>>i&1];
494         if(!s) s=++idx;
495         p=s;
496         if(!l[p]) l[p]=id;
497         r[p]=id;
498     }
499     vs.push_back(p);
500 }
501
502 inline int queryId(int p,int k,int x)
503 {
504     for(int i=k;i>=0;i--)
505     {
506         int v=x>>i&1;
507         if(tr[p][v]) p=tr[p][v];
508         else if(tr[p][v^1]) p=tr[p][v^1];
509         else return 0;
510     }
```

```
511     return r[p];
512 }
513
514 void xorMst(int p,int k)
515 {
516     int x=tr[p][0],y=tr[p][1];
517     if(x && y)
518     {
519         ll mi=1e17;
520         int u,v;
521         for(int i=l[x];i<=r[x];i++)
522         {
523             int id=queryId(y,k-1,a[i].first);
524             ll tmp=a[i].first^a[id].first;
525             if(tmp<mi) mi=tmp,u=i,v=id;
526         }
527         e[u].push_back(v);
528         e[v].push_back(u);
529         xorMst(x,k-1);
530         xorMst(y,k-1);
531     }
532     else if(x) xorMst(x,k-1);
533     else if(y) xorMst(y,k-1);
534
535     if(k==0)
536     {
537         if(x)
538             for(int i=l[x]+1;i<=r[x];i++)
539                 e[i].push_back(l[x]),e[l[x]].push_back(i);
540         if(y)
541             for(int i=l[y]+1;i<=r[y];i++)
542                 e[i].push_back(l[y]),e[l[y]].push_back(i);
543     }
544 }
545
546 void cls()
547 {
548     idx=0;
549     for(int i=0;i<vs.size();i++)
550         tr[vs[i]][0]=tr[vs[i]][1]=l[vs[i]]=r[vs[i]]=0;
551     idx1=0;
552     for(int i=0;i<vs1.size();i++)
553         tr1[vs1[i]][0]=tr1[vs1[i]][1]=0;
554     idx2=0;
555     for(int i=0;i<vs2.size();i++)
556         tr2[vs2[i]][0]=tr2[vs2[i]][1]=0;
557     vs.clear();
558     vs1.clear();
559     vs2.clear();
560     for(int i=1;i<=n;i++) e[i].clear();
561 }
562
563 void dfs(int u,int fa,int f)
```

```
{
    as[u]=f;
    for(auto v:e[u]) if(v!=fa) dfs(v,u,!f);
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>t;
    while(t--)
    {
        cin>>n;
        int ans=INT_MAX;
        as.resize(n+1,0);
        for(int i=1;i<=n;i++) cin>>a[i].first,a[i].second=i;
        sort(a+1,a+1+n);

        for(int i=1;i<=n;i++) insert(a[i].first,i);
        xorMst(0,30);

        dfs(1,0,1);
        for(int i=1;i<=n;i++) as[i]+=1;

        int c1=0,c2=0;
        for(int i=1;i<=n;i++)
            if(as[i]==1)
            {
                int tmp=find1(a[i].first);
                if(c1>0)
                    ans=min(ans,tmp);
                insert1(a[i].first);
                c1++;
            }else if(as[i]==2)
            {
                int tmp=find2(a[i].first);
                if(c2>0)
                    ans=min(ans,tmp);
                insert2(a[i].first);
                c2++;
            }

        vector<int> out(n+1);
        for(int i=1;i<=n;i++)
            if(as[i]==1) out[a[i].second]=1;
            else out[a[i].second]=2;

        cout<<ans<<"\n";
        for(int i=1;i<=n;i++)
            cout<<out[i];
        cout<<"\n";

        cls();
    }
```

```
617
618     return 0;
619 }
```

1.6 AC 自动机

```
1  //基础模板 - 可重叠与不可重叠匹配个数
2  #include<bits/stdc++.h>
3  using namespace std;
4
5  #define dg 1
6  #define ll long long
7  const int N=5e5/dg+100;
8  ll T,n,m;
9  string s,t;
10
11 struct ACAM
12 {
13     int idx;
14     int tr[N][26],fail[N],idv[N];
15     int dep[N];
16     int pre[N];
17     int cnt[2][N]; //cnt[0][]:可重叠; cnt[1][]:不可重叠
18
19     vector<int> e[N];
20
21     ACAM() { idx=0; }
22     void init()
23     {
24         for(int i=0;i<=idx;i++)
25         {
26             memset(tr[i],0,sizeof(tr[i]));
27             fail[i]=0;
28             dep[i]=pre[i]=0;
29             cnt[0][i]=cnt[1][i]=0;
30         }
31         idx=0;
32     }
33     void insert(string &s, int id)
34     {
35         int p=0;
36         for(int i=0;i<s.size();i++)
37         {
38             int &v=tr[p][s[i]-'a'];
39             if(!v) v=++idx;
40             dep[v]=i+1;
41             p=v;
42         }
43         idv[id]=p;
44     }
45     void build_tree()
46     {
```

```
47      queue<int> q;
48      for(int i=0;i<26;i++) if(tr[0][i]) q.push(tr[0][i]);
49      while(q.size())
50      {
51          int p=q.front(); q.pop();
52          for(int i=0;i<26;i++)
53          {
54              int u=tr[p][i];
55              if(!u) continue;
56
57              int j=fail[p];
58              while(j && !tr[j][i]) j=fail[j];
59              if(tr[j][i]) j=tr[j][i];
60              fail[u]=j;
61              q.push(u);
62          }
63      }
64  }
65  void build_treph()
66  {
67      queue<int> q;
68      for(int i=0;i<26;i++) if(tr[0][i]) q.push(tr[0][i]);
69      while(q.size())
70      {
71          int p=q.front(); q.pop();
72          for(int i=0;i<26;i++)
73          {
74              int u=tr[p][i];
75              if(u) fail[u] = tr[fail[p]][i], q.push(u);
76              else tr[p][i] = tr[fail[p]][i];
77          }
78      }
79  }
80  void build_failtree()
81  {
82      for(int i=1;i<=idx;i++) e[fail[i]].push_back(i);
83  }
84  void query(string &s)
85  {
86      int p=0;
87      for(int i=0;i<=idx;i++) pre[i]=-1;
88      for(int i=0;i<s.size();i++)
89      {
90          p=tr[p][s[i]-'a'];
91          for(int j=p;j;j=fail[j])
92          {
93              //重叠
94              cnt[0][j]++;
95              //不重叠
96              if(pre[j]==-1 || i-pre[j]>=dep[j])
97                  cnt[1][j]++, pre[j]=i;
98          }
99      }
```

```
    }
};

ACAM ac;

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>T;
    while (T--)
    {
        cin>>s;
        ac.init();
        cin>>n;
        for(int i=1;i<=n;i++) cin>>t,ac.insert(t,i);
        ac.build_treph();
        ac.query(s);
        for(int i=1;i<=n;i++)
            cout<<ac.cnt[1][ac.idv[i]]<<"\n";
    }

    return 0;
}

//luogu模板 - 二次加强版
//给你一个文本串 S 和 n 个模式串 T_{1..n} ，请你分别求出每个模式串 T_i 在 S 中出现的次数。
#include<bits/stdc++.h>
using namespace std;

const int N=2e5+10;
int tr[N][26],idx;
int cnt[N],sum[N];
int nx[N];

void insert(string &t,int x)
{
    int p=0;
    for(int i=0;i<t.length();i++)
    {
        int &s=tr[p][t[i]-'a'];
        if(!s) s=++idx;
        p=s;
    }
    cnt[x]=p;
}

void build()
{
    queue<int> q;
    for(int i=0;i<26;i++)
        if(tr[0][i]) q.push(tr[0][i]);

    while(q.size())
```

```
153     {
154         int p=q.front();q.pop();
155         for(int i=0;i<26;i++)
156         {
157             int px=tr[p][i];
158             if(px) nx[px] = tr[nx[p]][i], q.push(px);
159             else tr[p][i] = tr[nx[p]][i];
160         }
161     }
162 }
163 
164 void query(string &t)
165 {
166     int p=0;
167     for(int i=0;i<t.length();i++)
168     {
169         p=tr[p][t[i]-'a'];
170         sum[p]++;
171     }
172 }
173 
174 int n;
175 vector<int> e[N];
176 
177 /*
178 对于当前节点P，若P已被匹配，则P回溯nx[]树，也可被匹配。
179 而在查询时，若每次都进行回溯，则会超时，故可进行优化，即在查询操作完之后统一进行回溯。
180 使用DFS进行回溯计算
181 */
182 void dfs(int x)
183 {
184     for(auto i:e[x]) dfs(i), sum[x] += sum[i];
185 }
186 
187 int main()
188 {
189     ios::sync_with_stdio(0);cin.tie(0);
190     cin>>n;
191     string s;
192     for(int i=1;i<=n;i++) cin>>s,insert(s,i);
193     build();
194     cin>>s;
195     query(s);
196 
197     for(int i=1;i<=idx;i++) e[nx[i]].push_back(i);
198     dfs(0);
199 
200     for(int i=1;i<=n;i++)
201         cout<<sum[cnt[i]]<<"\n";
202 
203     return 0;
204 }
205 
```

```cpp

/*
16th黑龙江省赛 E. Elastic Search
ACAM + 简单DP
给定N个串Si，定义序列Q{q1,q2,q3...}，满足S{qi}是S{q(i-1)}的子串，
求满足该要求的最长序列长度。

S{i}是S{i-1}的子串，表示：S{i}是S{i-1}的前缀或后缀。
考虑ACAM的结构：对于节点u，fa[u]为其前缀，fail[u]为其后缀。
故建立ACAM后，遍历Trie,DP即可。，
*/
#include<bits/stdc++.h>
using namespace std;

#define dg 1
#define ll long long
const int N=5e5/dg+100;
ll t,n,m;

struct ACAM
{
    int idx;
    int tr[N][26],fail[N],sz[N];
    ll dp[N];

    ACAM() { idx=0; }
    void insert(string &s)
    {
        int p=0;
        for(int i=0;i<s.size();i++)
        {
            int &v=tr[p][s[i]-'a'];
            if(!v) v=++idx;
            p=v;
        }
        sz[p]++;
    }
    void build_tree()
    {
        queue<int> q;
        for(int i=0;i<26;i++) if(tr[0][i]) q.push(tr[0][i]);
        while(q.size())
        {
            int p=q.front(); q.pop();
            for(int i=0;i<26;i++)
            {
                int u=tr[p][i];
                if(!u) continue;

                int j=fail[p];
                while(j && !tr[j][i]) j=fail[j];
                if(tr[j][i]) j=tr[j][i];
                fail[u]=j;
```

```
259             q.push(u);
260         }
261     }
262 }
263 void build_treph()
264 {
265     queue<int> q;
266     for(int i=0;i<26;i++) if(tr[0][i]) q.push(tr[0][i]);
267     while(q.size())
268     {
269         int p=q.front(); q.pop();
270         for(int i=0;i<26;i++)
271         {
272             int u=tr[p][i];
273             if(u) fail[u] = tr[fail[p]][i], q.push(u);
274             else tr[p][i] = tr[fail[p]][i];
275         }
276     }
277 }
278 void solve()
279 {
280     ll ans=0;
281     queue<int> q;
282     q.push(0);
283     while(q.size())
284     {
285         int p=q.front(); q.pop();
286         for(int i=0;i<26;i++)
287         {
288             int u=tr[p][i];
289             if(!u) continue;
290 
291             dp[u]=max(dp[p],dp[fail[u]])+sz[u];
292             ans=max(ans,dp[u]);
293             q.push(u);
294         }
295     }
296     cout<<ans<<"\n";
297 }
298 };
299 
300 ACAM ac;
301 string s;
302 
303 int main()
304 {
305     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
306     cin>>n;
307     for(int i=1;i<=n;i++) cin>>s,ac.insert(s);
308     ac.build_tree();
309     ac.solve();
310 
311     return 0;
```

```
}


/*
P4052 [JSOI2007]文本生成器
AC自动机 + DP

给定N个串Si，给定长度M，求有多少种长度为M的串T，满足至少有一个Si为T的子串。(答案取模)
考虑容斥：ANS = 26^M - sub
考虑如何计算sub：
对N个串建AC自动机，
定义dp[i][u]：长度为i的串匹配到节点u时，不满足要求的串的个数
状态转移：
若当前节点u不为终点节点，则有 dp[i][u] += dp[i-1][fa[u]]
注意：AC自动机建fail指针时更新终点节点的标记
最后，对于所有节点u：sub += dp[m][u]
*/
#include<bits/stdc++.h>
using namespace std;

const int N=110,N60=N*60,base=26,P=1e4+7;
int bs[N];
int n,m;

void init()
{
    bs[0]=1;
    for(int i=1;i<110;i++) bs[i]=(bs[i-1]*26)%P;
}

struct ACAM
{
    int idx;
    int tr[N60][26],fail[N60];
    int ed[N60];
    int dp[N][N60];

    ACAM() { idx=0; }
    void insert(string &s)
    {
        int p=0;
        for(int i=0;i<s.size();i++)
        {
            int &v=tr[p][s[i]-'A'];
            if(!v) v=++idx;
            p=v;
        }
        ed[p]=1;
    }
    void build_treph()
    {
        queue<int> q;
        for(int i=0;i<26;i++) if(tr[0][i]) q.push(tr[0][i]);
```

```
        while(q.size())
        {
            int p=q.front(); q.pop();
            for(int i=0;i<26;i++)
            {
                int u=tr[p][i];
                if(u) fail[u] = tr[fail[p]][i], ed[u] |= ed[fail[u]], q.push(u);
                else tr[p][i] = tr[fail[p]][i];
            }
        }
    }
    int calc()
    {
        dp[0][0]=1;
        for(int i=1;i<=m;i++)
        {
            for(int j=0;j<=idx;j++)
            {
                for(int k=0;k<26;k++)
                {
                    int u=tr[j][k];
                    if(!ed[u])
                        dp[i][u] = (dp[i][u] + dp[i-1][j]) % P;
                }
            }
        }
        int sub=0;
        for(int i=0;i<=idx;i++) sub = (sub+dp[m][i]) %P;
        return (bs[m]-sub+P) %P;
    }
};

ACAM ac;
string s;

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    init();
    cin>>n>>m;
    for(int i=0;i<n;i++) cin>>s,ac.insert(s);
    ac.build_treph();
    cout<<ac.calc()<<"\n";

    return 0;
}
```

1.7 后缀数组

```
//待补完
string s;
int sa[N], t1[N], t2[N], c[N], rk[N], height[N];
```

```
4  //sa[],height[]下标从1开始，rk[]下标从0开始
5  void getsa(int m, int n)
6  { //n为字符串的长度,字符集的值为0~m-1
7     n++;
8     int *x = t1, *y = t2;
9     //基数排序
10    for (int i = 0; i < m; i++) c[i] = 0;
11    for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
12    for (int i = 1; i < m; i++) c[i] += c[i - 1];
13    for (int i = 0; i < n; i++) sa[--c[x[i]]] = i;
14    for (int k = 1; k <= n; k <<= 1)
15    { //直接利用sa数组排序第二关键字
16       int p = 0;
17       for (int i = n - k; i < n; i++) y[p++] = i;
18       for (int i = 0; i < n; i++)
19          if (sa[i] >= k)
20             y[p++] = sa[i] - k;
21       //基数排序第一关键字
22       for (int i = 0; i < m; i++) c[i] = 0;
23       for (int i = 0; i < n; i++) c[x[y[i]]]++;
24       for (int i = 1; i < m; i++) c[i] += c[i - 1];
25       for (int i = n - 1; ~i; i--) sa[--c[x[y[i]]]] = y[i];
26       //根据sa和y数组计算新的x数组
27       swap(x, y);
28       p = 1;
29       x[sa[0]] = 0;
30       for (int i = 1; i < n; i++)
31          x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p - 1
                : p++;
32       if (p >= n)
33          break; //以后即使继续倍增，sa也不会改变，退出
34       m = p; //下次基数排序的最大值
35    }
36    n--;
37    int k = 0;
38    for (int i = 0; i <= n; i++)
39       rk[sa[i]] = i;
40    for (int i = 0; i < n; i++)
41    {
42       if (k) k--;
43       int j = sa[rk[i] - 1];
44       while (s[i + k] == s[j + k])
45          k++;
46       height[rk[i]] = k;
47    }
48 }
49
50 int st[N][30];
51 void initrmq(int n)
52 {
53    for (int i = 1; i <= n; i++)
54       st[i][0] = height[i];
55    for (int j = 1; (1 << j) <= n; j++)
```

```
56        for (int i = 1; i + (1 << j) - 1 <= n; i++)
57            st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
58    }
59    int rmq(int l, int r)
60    {
61        int k = 31 - __builtin_clz(r - l + 1);
62        return min(st[l][k], st[r - (1 << k) + 1][k]);
63    }
64    int lcp(int a, int b)
65    { // 求两个后缀的最长公共前缀
66        a = rk[a], b = rk[b];
67        if (a > b)
68        swap(a, b);
69        return rmq(a + 1, b);
70    }
```

1.8 后缀自动机

1.8.1 本质不同子串

```
1   /*
2   本质不同子串 : \sum_i^n {mxlen[i]-mxlen[link[i]]}
3   SDOI2016 生成魔咒
4   https://vjudge.net/problem/LibreOJ-2033
5   */
6
7   #include<bits/stdc++.h>
8   using namespace std;
9
10  #define ll long long
11  const int N=1e5+100;
12  int n,x;
13  ll ans=0;
14
15  template<int N> struct SAM
16  {
17      int idx,last;
18      map<int,int> nx[N<<1];
19      int link[N<<1],mxlen[N<<1];
20
21      void init(int n)
22      {
23          idx = last = 1;
24          for(int i=1;i<2*n+10;i++)
25              nx[i].clear(),link[i]=mxlen[i]=0;
26      }
27      void add_int(int c)
28      {
29          int p=last,cur=last=++idx;
30          mxlen[cur] = mxlen[p]+1;
31          for(; p && nx[p].find(c)==nx[p].end(); p=link[p])
32              nx[p][c] = cur;
33          if(!p) link[cur] = 1;
```

```
        else
        {
            int x=nx[p][c];
            if(mxlen[x]==mxlen[p]+1) link[cur]=x;
            else
            {
                int y=++idx;
                mxlen[y] = mxlen[p]+1;
                nx[y] = nx[x];
                link[y] = link[x];
                link[cur] = link[x] = y;
                for(; p && nx[p].find(c)!=nx[p].end() && nx[p][c]==x; p=link[p])
                    nx[p][c]=y;
            }
        }
        ans+=mxlen[cur]-mxlen[link[cur]];
    }
};

SAM<N> sam;

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    sam.init(n);
    for(int i=0;i<n;i++) cin>>x,sam.add_int(x),cout<<ans<<"\n";
    return 0;
}
```

1.8.2 单次询问字典序第 k 小子串

```
/*
P3975 [TJOI2015]弦论
字典序第k小子串
0.不同位置的相同子串算作一个
1.不同位置的相同子串算作多个
*/

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

#define ll long long
const int N=5e5+100;
string s;
ll t,k;

template<int N,int chsize> struct SAM
{
    int idx,last;
    int nx[N<<1][chsize],mxlen[N<<1],link[N<<1];
    ll cntpath[N<<1];
```

```
21  ll staSize[N<<1]; //每个状态所包含的子串的出现次数，也即 |endpos(substr(sta))|.
22  vector<int> e[N<<1]; //link树
23  bool vis[N<<1]; //dfsDAG时记忆化搜索
24
25  void init(int n)
26  {
27      idx = last = 1;
28      // memset(nx,0,(n*2+10)*chsize*sizeof(int));
29  }
30  void add_char(int c)
31  {
32      int p=last,cur=last=++idx;
33      mxlen[cur]=mxlen[p]+1;
34      for(; p && !nx[p][c]; p=link[p])
35          nx[p][c] = cur;
36      if(!p) link[cur]=1;
37      else
38      {
39          int x=nx[p][c];
40          if(mxlen[x]==mxlen[p]+1) link[cur]=x;
41          else
42          {
43              int y=++idx;
44              mxlen[y]=mxlen[p]+1;
45              memcpy(nx[y],nx[x],chsize*sizeof(int));
46              link[y]=link[x];
47              link[cur]=link[x]=y;
48              for(; p && nx[p][c]==x; p=link[p])
49                  nx[p][c]=y;
50          }
51      }
52      staSize[cur]=1;
53  }
54  void dfs1(int u)
55  {
56      for(auto v:e[u])
57          dfs1(v),staSize[u]+=staSize[v];
58  }
59  void dfs2(int type,int u)
60  {
61      if(vis[u]) return ;
62      vis[u]=1;
63      cntpath[u] = u==1?0:(type==1?staSize[u]:1); //根节点初值为0
64      for(int i=0;i<chsize;i++)
65      {
66          if(!nx[u][i]) continue;
67          dfs2(type,nx[u][i]);
68          cntpath[u]+=cntpath[nx[u][i]];
69      }
70  }
71  void add_string(int type,string &s)
72  {
73      for(auto ch:s) add_char(ch-'a');
```

```
        for(int i=2;i<=idx;i++) e[link[i]].push_back(i);
        dfs1(1);
        dfs2(type,1);
    }
    void find(int type,int k) //type=0-情况1;type=1-情况2
    {
        if(cntpath[1]<k)
        {
            cout<<-1<<"\n";
            return ;
        }
        int p=1;
        while(k>0)
        {
            for(int i=0;i<26;i++)
            {
                int v=nx[p][i];
                if(!v) continue;
                if(k>cntpath[v]) k-=cntpath[v];
                else
                {
                    cout<<char('a'+i);
                    p=v;
                    break;
                }
            }
            k -= type==1?staSize[p]:1;
        }
    }
};

SAM<N,26> sam;

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>s;
    sam.init(s.size());
    cin>>t>>k;
    sam.add_string(t,s);
    sam.find(t,k);

    return 0;
}
```

1.8.3 多次询问字典序第 k 小子串

```
/*
*HDU 5008 Boring String Problem
*【多次询问字典序第k小子串】
*对逆串建SAM，构建后缀树，对后缀树节点按字典序排序；
*DFS求出前缀和数组，每次询问二分K出答案。
```

```
*/

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

#define ll long long
const int N=2e5+1000;
string s;
ll k;

struct SAM
{
    int idx,last;
    int nx[N][26],mxlen[N],link[N];

    //拓扑序数组 (比开vector建parent树跑dfs要快一点)
    int c[N],b[N];

    vector<int> e[N];
    int fp[N];
    vector<ll> sum,poi;

    void init()
    {
        for(int i=1;i<=idx;i++)
        {
            memset(nx[i],0,sizeof(int)*26);
            link[i]=fp[i]=0;
            e[i].clear();
        }
        idx=last=1;
        sum.clear();poi.clear();
    }
    void add_char(int c)
    {
        int p=last,cur=last=++idx;
        fp[cur]=mxlen[p]; //位置下标从0开始
        mxlen[cur]=mxlen[p]+1;
        for(; p && !nx[p][c]; p=link[p]) nx[p][c]=cur;
        if(!p) link[cur]=1;
        else
        {
            int x=nx[p][c];
            if(mxlen[x]==mxlen[p]+1) link[cur]=x;
            else
            {
                int y=++idx;
                mxlen[y]=mxlen[p]+1;
                memcpy(nx[y],nx[x],26*sizeof(int));
                link[y]=link[x];
                link[cur]=link[x]=y;
                for(; p && nx[p][c]==x; p=link[p]) nx[p][c]=y;
                fp[y]=fp[x];
```

```
59          }
60      }
61  }
62  void topu() //计数排序
63  {
64      for(int i=0;i<=idx;i++) c[i]=0;
65      for(int i=1;i<=idx;i++) ++c[mxlen[i]];
66      for(int i=1;i<=idx;i++) c[i]+=c[i-1];
67      for(int i=1;i<=idx;i++) b[c[mxlen[i]]--]=i;
68  }
69  void dfs(int u)
70  {
71      sum.push_back(mxlen[u]-mxlen[link[u]]); //按字典序从小到大的子串个数
72      poi.push_back(u); //sum[i]对应的节点u
73      for(auto v:e[u]) dfs(v);
74  }
75  void add_string(string &s)
76  {
77      for(auto &ch:s) add_char(ch-'a');
78      topu();
79      for(int i=2;i<=idx;i++) e[link[i]].push_back(i);
80      for(int i=idx;i>=1;i--)
81      {
82          int u=b[i],fa=link[u];
83          fp[fa]=max(fp[fa],fp[u]);
84      }
85      //对parent树每个节点的子节点按字典序排序
86      for(int i=1;i<=idx;i++)
87          sort(e[i].begin(),e[i].end(),
88              [&](int a,int b)
89              {return s[fp[a]-mxlen[link[a]]] < s[fp[b]-mxlen[link[a]]];}
90              );
91      dfs(1);
92      for(int i=1;i<sum.size();i++) sum[i]+=sum[i-1];
93  }
94  void query(ll &L,ll &R,ll v)
95  {
96      ll k=(L^R^v)+1; //强制在线题
97      if(k>sum.back())
98      {
99          L=R=0;
100         cout<<0<<" "<<0<<"\n";
101         return ;
102     }
103     int pos=lower_bound(sum.begin(),sum.end(),k)-sum.begin();
104     int p=poi[pos];
105     L = s.size() - fp[p];
106     R = L + mxlen[link[p]] + k - (pos?sum[pos-1]:0) -1;
107     cout<<L<<" "<<R<<"\n";
108 }
109 };
110
111 SAM sam;
```

```
112
113 int main()
114 {
115     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
116     while (cin>>s)
117     {
118         reverse(s.begin(),s.end());
119         sam.init();
120         sam.add_string(s);
121
122         int q;
123         cin>>q;
124         ll L=0,R=0;
125         while(q--)
126             cin>>k,sam.query(L,R,k);
127     }
128
129     return 0;
130 }
```

1.8.4 LCSS-1

```
1  /*
2  Longest Common Substring - 1
3  两个字符串的最长公共子串的长度
4
5  SOLUTION:
6  对第一个串建SAM,
7  然后将第二个串放到第一个串上用link数组当失配数组跑匹配,
8  失配就跳link数组，匹配就状态转移且len++;
9  len维护的就是：以当前位置为结尾的前缀中，可匹配的最长后缀的长度;
10 然后维护一个最长长度即可。
11 */
12 #include<bits/stdc++.h>
13 using namespace std;
14
15 const int N=2e5+100;
16 string s,t;
17
18 template <int N> struct SAM
19 {
20     int idx,last;
21     int nx[N<<1][26],mxlen[N<<1],link[N<<1];
22
23     void init()
24     {
25         idx = last = 1;
26     }
27     void add_char(int c)
28     {
29         int p=last,cur=last=++idx;
30         mxlen[cur]=mxlen[p]+1;
```

```
        for(; p && !nx[p][c]; p=link[p])
            nx[p][c]=cur;
        if(!p) link[cur]=1;
        else
        {
            int x=nx[p][c];
            if(mxlen[x]==mxlen[p]+1) link[cur]=x;
            else
            {
                int y=++idx;
                mxlen[y]=mxlen[p]+1;
                memcpy(nx[y],nx[x],26*sizeof(int));
                link[y]=link[x];
                link[cur]=link[x]=y;
                for(; p && nx[p][c]==x; p=link[p])
                    nx[p][c]=y;
            }
        }
    }
    void add_string(string &s)
    {
        for(auto ch:s) add_char(ch-'a');
    }
    //*LCS*
    int lcs(string &t)
    {
        //遍历T的每个位置，维护以当前位置为结尾的前缀的在S中匹配的最长后缀的长度
        int len=0,v=1,l=0,pos=0;
        for(int i=0;i<(int)t.size();i++)
        {
            int c=t[i]-'a';

            //失配跳link数组
            while(v>1 && !nx[v][c])
                v=link[v],l=mxlen[v];

            //边转移
            if(nx[v][c])
            {
                v=nx[v][c],l++;
                //更新答案
                if(l>len)
                    len=l,pos=i;
            }
        }

        return len;
    }
};

SAM<N> sam;

int main()
```

```
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>s>>t;
    sam.init();
    sam.add_string(s);
    cout<<sam.lcs(t)<<"\n";

    return 0;
}
```

1.8.5 LCSS-2

```
/*
Longest Common Substring - 2
多个字符串的最长公共子串的长度

SOLUTION: - 单串SAM
对第一个串建SAM,
然后将后面串放到第一个串上用link数组当失配数组跑匹配。
对第一个串的SAM上的每个节点都维护一个其它串在该节点匹配的最短长度lcslen;

同时，需要注意:
当前节点可匹配，则意味着parent树上该节点的祖先节点都可匹配，
故需要先更新一下祖先节点的最短长度，然后再对所有节点的lcslen取一个max.

如，对于节点u及其子节点v,
A串在节点u匹配长度为L1，在节点v匹配长度为L2; (L1<L2)
B串在节点v匹配长度为L3，在节点v匹配长度为L4；(L3>L2>L1;L4<L2)
若不更新父节点，则节点u的答案为L1，节点v的答案为L4;
而实际上，节点u的答案应为L2，节点v的答案为L4。

注意：在每次匹配的更新祖先节点阶段，需采用剪枝，不然无法保证时间复杂度

SOLUTION: - GSAM待补完

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

#define ll long long
const int N=2e5+100;
ll t,n,m;
string s;

struct SAM
{
    int idx,last;
    int nx[N][26],mxl[N],link[N];

    //拓扑序数组 (比开vector建parent树跑dfs要快一点)
    // int c[N],b[N];

```

```
41    int mlen[N],clen[N];
42    bool flag[N];
43    vector<int> e[N];
44
45    void init()
46    {
47        idx=last=1;
48    }
49    void add_char(int c)
50    {
51        int p=last,cur=last=++idx;
52        mxl[cur]=mxl[p]+1;
53        for(; p && !nx[p][c]; p=link[p]) nx[p][c]=cur;
54        if(!p) link[cur]=1;
55        else
56        {
57            int x=nx[p][c];
58            if(mxl[x]==mxl[p]+1) link[cur]=x;
59            else
60            {
61                int y=++idx;
62                mxl[y]=mxl[p]+1;
63                memcpy(nx[y],nx[x],26*sizeof(int));
64                link[y]=link[x];
65                link[cur]=link[x]=y;
66                for(; p && nx[p][c]==x; p=link[p]) nx[p][c]=y;
67            }
68        }
69    }
70    void add_string(string &s)
71    {
72        for(auto &ch:s) add_char(ch-'a');
73        for(int i=2;i<=idx;i++) e[link[i]].push_back(i);
74        //初始化clen[]
75        for(int i=1;i<=idx;i++) clen[i]=mxl[i];
76    }
77    //剪枝优化
78    void dfs_mc(int u)
79    {
80        if(u>1 && flag[u]) return ;
81        for(auto v:e[u])
82            dfs_mc(v),mlen[u]=max(mlen[u],min(mxl[u],mlen[v])),mlen[v]=0;
83        clen[u]=min(clen[u],mlen[u]);
84        if(clen[u]<=mxl[link[u]]) flag[u]=1;
85    }
86    // void topu() //计数排序
87    // {
88    // for(int i=1;i<=idx;i++) ++c[mxl[i]];
89    // for(int i=1;i<=idx;i++) c[i]+=c[i-1];
90    // for(int i=1;i<=idx;i++) b[c[mxl[i]]--]=i;
91    // }
92    void match(string &t)
93    {
```

```
94        int v=1,len=0;
95        for(int i=0;i<(int)t.size();i++)
96        {
97            int c=t[i]-'a';
98            while(v>1 && !nx[v][c])
99                v=link[v],len=mxl[v];
100           if(nx[v][c])
101           {
102               len++;
103               v=nx[v][c];
104               mlen[v]=max(mlen[v],len);
105           }
106       }
107       //当匹配次数足够多时，每次都更新整个SAM显然时间复杂度不够优秀，故需采用dfs剪枝
108       // topu();
109       // for(int i=idx;i>=1;i--)
110       // {
111       // int u=b[i],fa=link[u];
112       // mlen[fa] = max(mlen[fa], min(mlen[u],mxl[fa]));
113       // clen[u]=min(clen[u],mlen[u]);
114       // mlen[u]=0;
115       // }
116       dfs_mc(1);
117   }
118   void getAns(int u,int &x)
119   {
120       if(u>1 && flag[u]) return ;
121       x=max(x,clen[u]);
122       for(auto v:e[u]) getAns(v,x);
123   }
124};
125
126SAM sam;
127
128int main()
129{
130   ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
131   cin>>s;
132   sam.init();
133   sam.add_string(s);
134   while(cin>>s) sam.match(s);
135
136   int mxlcslen=0;
137   sam.getAns(1,mxlcslen);
138   cout<<mxlcslen<<"\n";
139
140   return 0;
141}
```

1.8.6 多个串字典序第 k 小公共子串

```
1 /*
```

```
n个串的字典序第k小公共子串
*/
#include<bits/stdc++.h>
using namespace std;

#define dg 1
#define ll long long
const int N=4e5/dg+100;
int t,n,q,k;
string s;

struct SAM
{
    int idx,last;
    int nx[N][26],mxl[N],link[N];

    vector<int> e[N];
    vector<ll> sum,po;
    int fp[N];
    bool flag[N];

    //mlen[]单次匹配的最长长度; clen[]所有串的最短公共匹配长度
    int mlen[N],clen[N];

    void init()
    {
        for(int i=0;i<=idx;i++)
        {
            memset(nx[i],0,sizeof(nx[i]));
            link[i]=fp[i]=0;
            e[i].clear();
            clen[i]=INT_MAX;
            flag[i]=0;
            mlen[i]=0;
        }
        idx=last=1;
        sum.clear();po.clear();
    }
    void add_char(int c)
    {
        int p=last,cur=last=++idx;
        mxl[cur]=mxl[p]+1;
        for(; p && !nx[p][c]; p=link[p]) nx[p][c]=cur;
        if(!p) link[cur]=1;
        else
        {
            int x=nx[p][c];
            if(mxl[x]==mxl[p]+1) link[cur]=x;
            else
            {
                int y=++idx;
                mxl[y]=mxl[p]+1;
                memcpy(nx[y],nx[x],sizeof(nx[x]));
```

```
55              link[y]=link[x];
56              link[cur]=link[x]=y;
57              for(; p && nx[p][c]==x; p=link[p]) nx[p][c]=y;
58          }
59      }
60
61  }
62  void dfs_fp(int u,string &s)
63  {
64      for(auto v:e[u]) dfs_fp(v,s),fp[u]=max(fp[u],fp[v]);
65      sort(e[u].begin(),e[u].end(),
66          [&](int a,int b)
67          { return s[fp[a]-mxl[link[a]]]<s[fp[b]-mxl[link[a]]]; });
68  }
69  void add_string(string &s)
70  {
71      for(auto ch:s) add_char(ch-'a');
72
73      int p=1;
74      for(int i=0;i<s.size();i++) p=nx[p][s[i]-'a'],fp[p]=i;
75      //建parent树
76      for(int i=2;i<=idx;i++) e[link[i]].push_back(i);
77      dfs_fp(1,s);
78  }
79
80  void dfs_mc(int u)
81  {
82      //flag[]标记数组，剪枝
83      if(u>1 && flag[u]) return ;
84      //更新 clen[],重置 mlen[];
85      for(auto v:e[u])
86      {
87          dfs_mc(v);
88          mlen[u]=max(mlen[u],min(mxl[u],mlen[v]));
89          mlen[v]=0;
90      }
91      clen[u]=min(clen[u],mlen[u]);
92      //若当前节点的公共匹配长度小于自身的minsub时，说明该节点及其子树不会对答案有贡献，标记剪枝
93      if(clen[u]<=mxl[link[u]]) flag[u]=1;
94  }
95
96  //SAM上的匹配：len表示对于S中的每个位置i，S在SAM中匹配的最长公共子串的长度
97  void match(string &s)
98  {
99      int u=1,len=0;
100     for(int i=0;i<s.size();i++)
101     {
102         int c=s[i]-'a';
103         while(u>1 && !nx[u][c]) u=link[u],len=mxl[u];
104         if(nx[u][c])
105         {
106             len++;
107             u=nx[u][c];
```

```
108             mlen[u]=max(mlen[u],len);
109         }
110     }
111     dfs_mc(1);
112 }
113
114 void dfs(int u)
115 {
116     if(u>1 && flag[u]) return ;
117     if(clen[u]>mxl[link[u]])
118     {
119         //!注意只有一个串的情况，此时clen为inf
120         sum.push_back(min(clen[u],mxl[u])-mxl[link[u]]);
121         po.push_back(u);
122     }
123     for(auto v:e[u]) dfs(v);
124 }
125 void build()
126 {
127     sum.push_back(0);
128     po.push_back(0);
129     dfs(1);
130     for(int i=1;i<sum.size();i++) sum[i]+=sum[i-1];
131 }
132
133 void query(int k,string &s)
134 {
135     if(k>sum.back())
136     {
137         cout<<-1<<"\n";
138         return ;
139     }
140     int pos=lower_bound(sum.begin(),sum.end(),k)-sum.begin();
141     int p=po[pos];
142     int l=s.size()-fp[p]-1;
143     int r=l+mxl[link[p]] + k-(pos?sum[pos-1]:0);
144     cout<<l<<" "<<r<<"\n";
145 }
146 };
147
148 SAM sam;
149
150 int main()
151 {
152     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
153     memset(sam.clen,0x3f,sizeof(sam.clen));
154     cin>>t;
155     while(t--)
156     {
157         cin>>n;
158         string rs;
159         cin>>rs;
160         reverse(rs.begin(),rs.end());
```

```
        sam.init();
        sam.add_string(rs);
        for(int i=2;i<=n;i++)
        {
            cin>>s;
            reverse(s.begin(),s.end());
            sam.match(s);
        }
        sam.build();
        cin>>q;
        while (q--)
            cin>>k,sam.query(k,rs);
    }

    return 0;
}
```

1.8.7 SAM+ 倍增 + 启发式合并

```
/*
T组样例,
给定长度为 n 的字符串 S 和 q 次询问, 每次询问给定整数 l,r ;
要求对每次询问输出 S[l,r] 在S中的一次或多次出现是否发生重叠。

n总共不超过6e5; q总共不超过3e6。

解法: O(n(logn)^2)
SAM + 倍增 + 启发式合并
对于每次询问, 倍增找到该子串在SAM上对应的状态,
判断该状态的 endpos{}集合中的最小间距 是否大于子串长度。
*/
#include<bits/stdc++.h>
using namespace std;

const int N=1e5+100;
int t,n,q;

template <int N> struct SAM
{
    int idx,last;
    int nx[N<<1][26],mxlen[N<<1],link[N<<1];
    vector<int> e[N<<1];
    set<int> endpos[N<<1];
    int plen[N<<1]; //状态sta所包含的子串的 重复出现的最小间距 。
    int fa[N<<1][21]; //link树上倍增找点
    int id[N];

    void init(int n)
    {
        idx = last = 1;
        for(int i=0;i<2*n+10;i++)
        {
```

```
34          memset(nx[i],0,26*sizeof(int));
35          e[i].clear();
36          endpos[i].clear();
37      }
38  }
39  void add_char(int c)
40  {
41      int p=last,cur=last=++idx;
42      endpos[cur].insert(mxlen[p]+1);
43      mxlen[cur]=mxlen[p]+1;
44      for(; p && !nx[p][c]; p=link[p])
45          nx[p][c]=cur;
46      if(!p) link[cur]=1;
47      else
48      {
49          int x=nx[p][c];
50          if(mxlen[x]==mxlen[p]+1) link[cur]=x;
51          else
52          {
53              int y=++idx;
54              mxlen[y]=mxlen[p]+1;
55              memcpy(nx[y],nx[x],26*sizeof(int));
56              link[y]=link[x];
57              link[cur]=link[x]=y;
58              for(; p && nx[p][c]==x; p=link[p])
59                  nx[p][c]=y;
60              endpos[y]=endpos[x];
61          }
62      }
63  }
64  void dfs(int u)
65  {
66      plen[u]=0x3f3f3f3f;
67      //!倍增数组赋值
68      for(int i=1;i<=20;i++) fa[u][i]=fa[fa[u][i-1]][i-1];
69      for(auto v:e[u])
70      {
71          dfs(v);
72          if(u==1) continue;
73          //维护最小间距
74          plen[u]=min(plen[u],plen[v]);
75          //!启发式合并
76          if(endpos[u].size()>=endpos[v].size())
77          {
78              while(endpos[v].size())
79              {
80                  auto it=*endpos[v].begin();
81                  endpos[v].erase(it);
82                  auto np=endpos[u].insert(it).first;
83                  if((*np)!=*endpos[u].begin())
84                  {
85                      int tmp=*np;
86                      plen[u]=min(plen[u],abs(tmp-*(--np)));
```

```
87                  ++np;
88              }
89              if((*np)!=*endpos[u].rbegin())
90              {
91                  int tmp=*np;
92                  plen[u]=min(plen[u],abs(tmp-*(++np)));
93                  --np;
94              }
95          }
96      }else
97      {
98          while(endpos[u].size())
99          {
100             auto it=*endpos[u].begin();
101             endpos[u].erase(it);
102             auto np=endpos[v].insert(it).first;
103             if((*np)!=*endpos[v].begin())
104             {
105                 int tmp=*np;
106                 plen[u]=min(plen[u],abs(tmp-*(--np)));
107                 ++np;
108             }
109             if((*np)!=*endpos[v].rbegin())
110             {
111                 int tmp=*np;
112                 plen[u]=min(plen[u],abs(tmp-*(++np)));
113                 --np;
114             }
115         }
116         endpos[u]=move(endpos[v]);
117     }
118 }
119 }
120
121 void add_string(string &s)
122 {
123     for(auto ch:s) add_char(ch-'a' );
124     for(int i=2;i<=idx;i++)
125         fa[i][0]=link[i],e[link[i]].push_back(i);
126     dfs(1);
127     id[0]=1;
128     for(int i=1;i<=s.size();i++)
129         id[i]=nx[id[i-1]][s[i-1]-'a'];
130 }
131 bool check(int l,int r,string &s)
132 {
133     int p=id[r], len=r-l+1;
134     //!倍增找第一个包含该子串的节点
135     for(int i=20;i>=0;i--)
136     {
137         int pre=fa[p][i];
138         if(pre>1 && mxlen[pre]>=len) p=pre;
139     }
```

```
        return plen[p]<len;
    }
};

SAM<N> sam;
string s;

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>t;
    while(t--)
    {
        cin>>n>>q>>s;
        sam.init(n);
        sam.add_string(s);
        while (q--)
        {
            int l,r;
            cin>>l>>r;
            cout<<(sam.check(l,r,s)?"Yes\n":"No\n");
        }
    }

    return 0;
}
```

1.8.8 SAM+ 线段树合并

```
/*
CF1037H. Security
给定一个基串S，q次询问。
每次询问给定参数l,r,字符串x,
求S[l,r]中字典序第一个大于x的子串，若不存在输出-1.

SOLUTION - SAM + 线段树合并
SAM上每个节点开一个线段树维护 该节点的endpos集合
对于每次询问，在S上跑x的匹配,
每到一个节点，判断该节点的endpos集合中是否存在pos,使得pos属于[l,r],
若存在则说明该点可以转移，然后贪心匹配即可（注意细节）。

*/

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

#define ll long long
const int N=2e5+100,M=4e6+100; //线段树合并的空间要计算好
string s;
ll q,l,r;
string x;
int slen;
```

```
24
25  //==SGT==
26  struct SGT
27  {
28      int ls[M],rs[M],node;
29      void change(int &x,const int &L,const int &R,const int &pos){
30          if(!x)x=++node;
31          if(L==R){return;}
32          int mid=(L+R)>>1;
33          if(pos<=mid)change(ls[x],L,mid,pos);
34          else change(rs[x],mid+1,R,pos);
35      }
36      //查询【L，R】内是否存在x，使得x属于【ql,qr】
37      bool query(const int &x,const int &L,const int &R,const int &ql,const int &qr){
38          if(!x)return 0;
39          if(L>=ql&&R<=qr)return 1;
40          int mid=(L+R)>>1;
41          int res=0;
42          if(ql<=mid)res=query(ls[x],L,mid,ql,qr);
43          if(mid<qr)res|=query(rs[x],mid+1,R,ql,qr);
44          return res;
45      }
46      int merge(const int &x,const int &y){
47          if(!x||!y)return x|y;
48          int p=++node;
49          ls[p]=merge(ls[x],ls[y]);
50          rs[p]=merge(rs[x],rs[y]);
51          return p;
52      }
53  };
54  SGT sgt;
55  //==SGT==
56
57  struct SAM
58  {
59      int idx,last;
60      int nx[N][26],mxl[N],link[N],fp[N],rt[N];
61      vector<int> e[N];
62
63      void init()
64      {
65          idx=last=1;
66      }
67      void add_char(int c)
68      {
69          int p=last,cur=last=++idx;
70          // fp[cur]=mxl[p]+1; //位置下标从1开始
71          mxl[cur]=mxl[p]+1;
72          for(; p && !nx[p][c]; p=link[p]) nx[p][c]=cur;
73          if(!p) link[cur]=1;
74          else
75          {
76              int x=nx[p][c];
```

```
77          if(mxl[x]==mxl[p]+1) link[cur]=x;
78          else
79          {
80              int y=++idx;
81              mxl[y]=mxl[p]+1;
82              memcpy(nx[y],nx[x],sizeof nx[x]);
83              link[y]=link[x];
84              link[cur]=link[x]=y;
85              for(; p && nx[p][c]==x; p=link[p]) nx[p][c]=y;
86              // fp[y]=fp[x];
87          }
88      }
89  }
90  void dfs(int u)
91  {
92      if(fp[u]) sgt.change(rt[u],1,slen,fp[u]);
93      for(auto v:e[u]) dfs(v),rt[u]=sgt.merge(rt[u],rt[v]);
94  }
95  void add_string(string &s)
96  {
97      for(auto ch:s) add_char(ch-'a');
98      for(int i=2;i<=idx;i++) e[link[i]].push_back(i);
99      //在不需要对fp合并时，采用只记录终点节点的fp更方便；同时，使用线段树开点时也更节省空间
100     int p=1;
101     for(int i=0;i<s.size();i++) p=nx[p][s[i]-'a'],fp[p]=i+1;
102     dfs(1);
103 }
104
105 void match(string &t,int l,int r)
106 {
107     int u=1,len=0;
108     vector<int> path{1},ans;
109     for(int i=0;i<t.size();i++)
110     {
111         int c=t[i]-'a',j=c;
112         for(;j<26;j++)
113         {
114             int v=nx[u][j];
115             if(!v) continue;
116             //判断下一个点是否可以转移
117             if(sgt.query(rt[v],1,slen,l+len,r))
118             {
119                 path.push_back(v);
120                 ans.push_back(j);
121                 u=v;
122                 len++;
123                 break;
124             }
125         }
126         if(j==26) //没有下一个符合转移条件的节点
127             break;
128         if(j>c) //当前路径的字符串字典序已经大于x
129         {
```

```
            for(auto ch:ans) cout<<char(ch+'a'); cout<<"\n";
            return ;
        }
    }

    //回溯 - 寻找第一个字典序大于当前路径的串
    reverse(path.begin(),path.end());
    int ed=len; t+='a'-1;
    for(auto u:path)
    {
        for(int j=t[ed]-'a'+1;j<26;j++)
        {
            int v=nx[u][j];
            if(!v) continue;
            if(sgt.query(rt[v],1,slen,l+len,r))
            {
                ans.push_back(j);
                for(auto ch:ans) cout<<char(ch+'a'); cout<<"\n";
                return ;
            }
        }
        ed--;
        len--;
        ans.pop_back();
    }
    cout<<-1<<"\n";
  }
};

SAM sam;

int main()
{
  ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
  cin>>s;
  slen=s.size();
  sam.init();
  sam.add_string(s);
  cin>>q;
  while(q--)
    cin>>l>>r>>x, sam.match(x,l,r);

  return 0;
}
```

1.8.9 SAMparent 树按 dfs 序建主席树

```
/*
给定一个长为 n 的串 s，有 q 次询问；
每次询问子串 s[l,r] 在原串中第 k 次出现的位置，若无答案，输出 -1。
(n, q, k<= 1e5)

```

```
6  SOLUTION: SAM的parent树上DFS序上建可持久化线段树
7
8  SAM上对每个节点记录其第一次出现的位置-fp;
9  则问题可转化为:
10 对于子串s[l,r],求包含该子串的节点的子树中的第k大值。
11 故可对parent树按DFS序建主席树, 查第k大。
12
13 注意: 对于复制节点要特殊处理。
14
15 */
16 #include<bits/stdc++.h>
17 using namespace std;
18
19 //!
20 const int N=1e5+100;
21 int t,n,q;
22 string s;
23
24 //======SAM
25 int idx,last;
26 int nx[N<<1][26],mxlen[N<<1],link[N<<1];
27 int staSize[N<<1];
28 int firstpos[N<<1];
29 vector<int> e[N<<1];
30 int fa[N<<1][33];
31 int id[N];
32 bool is_clone[N<<1];
33
34 void init(int n)
35 {
36    idx=last=1;
37    for(int i=0;i<2*n+10;i++)
38    {
39       memset(nx[i],0,26*sizeof(int));
40       e[i].clear();
41       staSize[i]=0;
42       is_clone[i]=0;
43    }
44 }
45 void add_char(int c)
46 {
47    int p=last,cur=last=++idx;
48    firstpos[cur]=mxlen[p]+1;
49    mxlen[cur]=mxlen[p]+1;
50    for(; p && !nx[p][c]; p=link[p])
51       nx[p][c]=cur;
52    if(!p) link[cur]=1;
53    else
54    {
55       int x=nx[p][c];
56       if(mxlen[x]==mxlen[p]+1) link[cur]=x;
57       else
58       {
```

```
59          int y=++idx;
60          mxlen[y]=mxlen[p]+1;
61          memcpy(nx[y],nx[x],26*sizeof(int));
62          link[y]=link[x];
63          link[cur]=link[x]=y;
64          for(; p && nx[p][c]==x; p=link[p])
65             nx[p][c]=y;
66          firstpos[y]=firstpos[x];
67          is_clone[y]=1;
68       }
69    }
70    staSize[cur]=1;
71 }
72 void dfs(int u)
73 {
74    for(int i=1;i<=30;i++) fa[u][i]=fa[fa[u][i-1]][i-1];
75    for(auto v:e[u])
76    {
77       dfs(v);
78       staSize[u]+=staSize[v];
79    }
80 }
81 void add_string(string &s)
82 {
83    for(auto ch:s) add_char(ch-'a');
84    for(int i=2;i<=idx;i++)
85       fa[i][0]=link[i],e[link[i]].push_back(i);
86    dfs(1);
87    id[0]=1;
88    for(int i=1;i<=s.size();i++)
89       id[i]=nx[id[i-1]][s[i-1]-'a'];
90 }
91 //======SAM
92
93 //主席树
94 struct node
95 {
96    int l,r;
97    int s;
98 };
99 node tr[N*40];
100 int rot[N];
101 int tot;
102 int build(int l,int r)
103 {
104    int cur=++tot;
105    tr[cur].s=0;
106    if(l==r)return cur;
107    int mid=l+r>>1;
108    tr[cur].l=build(l,mid);
109    tr[cur].r=build(mid+1,r);
110    return cur;
111 }
```

```
112 int update(int rt,int l,int r,int pos,int v)
113 {
114     int cur=++tot;
115     tr[cur]=tr[rt];
116     tr[cur].s+=v;
117     if(l==r)
118     {
119         return cur;
120     }
121     int mid=l+r>>1;
122     if(pos<=mid)tr[cur].l=update(tr[rt].l,l,mid,pos,v);
123     else tr[cur].r=update(tr[rt].r,mid+1,r,pos,v);
124     return cur;
125 }
126 int query(int lrt,int rrt,int k,int l,int r)
127 {
128     if(l==r)return l;
129     int mid=l+r>>1;
130     int ds=tr[tr[rrt].l].s-tr[tr[lrt].l].s;
131     if(ds>=k)
132         return query(tr[lrt].l,tr[rrt].l,k,l,mid);
133     else
134         return query(tr[lrt].r,tr[rrt].r,k-ds,mid+1,r);
135 }
136
137 //DFS序
138 int dfn1[N<<1],id1[N<<1],ed1[N<<1],cnt;
139 void dfs1(int u)
140 {
141     if(u!=1)
142         dfn1[u]=++cnt;
143     id1[cnt]=firstpos[u];
144     for(auto v:e[u]) dfs1(v);
145     ed1[u]=cnt;
146 }
147
148 int ppre[N];
149 int main()
150 {
151     ios::sync_with_stdio(0);cin.tie(0);
152     cin>>t;
153     while(t--)
154     {
155         cin>>n>>q>>s;
156         init(n);
157         add_string(s);
158         cnt=tot=0;
159
160         dfs1(1);
161         rot[0]=build(1,n+1);
162         for(int i=1;i<=cnt;i++) ppre[id1[i]]=i;
163         for(int i=1;i<=cnt;i++)
164         {
```

```
165             if(ppre[id1[i]]==i)
166                 rot[i]=update(rot[i-1],1,n+1,id1[i],1);
167             else
168                 rot[i]=rot[i-1];
169         }
170 
171         while (q--)
172         {
173             int l,r,k;
174             cin>>l>>r>>k;
175             int p1=id[r],len=r-l+1;
176             //倍增找到最前面的点
177             for(int i=30;i>=0;i--)
178             {
179                 int pre=fa[p1][i];
180                 if(pre>1 && mxlen[pre]>=len) p1=pre;
181             }
182             int t1=dfn1[p1]-1;
183             int t2=ed1[p1];
184             int tmp=p1;
185             int ans=query(rot[dfn1[p1]-1],rot[ed1[p1]],k,1,n+1);
186             if(ans<1 || ans>n) ans=-1;
187             else ans-=len-1;
188             cout<<ans<<"\n";
189         }
190     }
191 
192     return 0;
193 }
```

1.9 广义后缀自动机

1.9.1 Trie 离线构造

```
1  /*
2  P6139 【模板】广义后缀自动机（广义 SAM）
3  */
4  #include<bits/stdc++.h>
5  using namespace std;
6 
7  #define ll long long
8  const int N=1e6+100;
9  ll n,ans;
10 string s;
11 
12 //Trie - 离线构建
13 struct GSA
14 {
15     int idx;
16     int nx[N<<1][26],link[N<<1],mxlen[N<<1];
17 
18     void init()
19     {
```

```
20          idx=1;
21      }
22      void insTrie(string &t)
23      {
24          int p=1;
25          for(int i=0;i<(int)t.size();i++)
26          {
27              int &s=nx[p][t[i]-'a'];
28              if(!s) s=++idx;
29              p=s;
30          }
31      }
32      int insSAM(int last,int c)
33      {
34          int cur=nx[last][c];
35          int p=link[last];
36          mxlen[cur]=mxlen[last]+1;
37
38          for(; p && !nx[p][c]; p=link[p])
39              nx[p][c]=cur;
40          if(!p) link[cur]=1;
41          else
42          {
43              int x=nx[p][c];
44              if(mxlen[x]==mxlen[p]+1) link[cur]=x;
45              else
46              {
47                  int y=++idx;
48                  mxlen[y]=mxlen[p]+1;
49                  for(int i=0;i<26;i++)
50                      nx[y][i] = mxlen[nx[x][i]]!=0? nx[x][i]:0;
51                  link[y]=link[x];
52                  link[cur]=link[x]=y;
53                  for(; p!=-1 && nx[p][c]==x; p=link[p])
54                      nx[p][c]=y;
55              }
56          }
57          return cur;
58      }
59      void build()
60      {
61          queue<pair<int,int>> q;
62          for(int i=0;i<26;i++)
63              if(nx[1][i]) q.push({i,1});
64          while(q.size())
65          {
66              auto cur=q.front();q.pop();
67              int last=insSAM(cur.second,cur.first);
68              for(int i=0;i<26;i++)
69                  if(nx[last][i]) q.push({i,last});
70          }
71      }
72  };
```

```
73
74  GSA gsa;
75
76  int main()
77  {
78      ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
79      gsa.init();
80      cin>>n;
81      for(int i=0;i<n;i++)
82          cin>>s,gsa.insTrie(s);
83      gsa.build();
84      for(int i=2;i<=gsa.idx;++i)
85          ans+=gsa.mxlen[i]-gsa.mxlen[gsa.link[i]];
86      cout<<ans<<"\n";
87
88      return 0;
89  }
```

1.9.2 在线构造

```
1   /*
2   P4081 [USACO17DEC]Standing Out from the Herd P
3   定义【独特值】表示只需要该字符串的本质不同的非空子串的个数，
4   如 “amy” 与 “tommy” 两个串，只属于 “amy” 的本质不同的子串为 “a” “am” “amy” 共 3
        个。
5   只属于 “tommy” 的本质不同的子串为 “t” “to” “tom” “tomm” “tommy” “o” “om” “omm
        ” “ommy” “mm” “mmy” 共 11 个。
6   所以 “amy” 的「独特值」为 3 ，“tommy” 的「独特值」为 11 。
7
8   染色
9   */
10  #include<bits/stdc++.h>
11  using namespace std;
12
13  #define ll long long
14  const int N=2e5+100;
15  ll n;
16  string s;
17  int color[N],ct[N],ans[N];
18
19  struct GSA
20  {
21      int idx,last;
22      int nx[N<<1][26],link[N<<1],mxlen[N<<1];
23
24      void init()
25      {
26          idx=last=1;
27      }
28      void insSAM(int c,int id) //在线GSAM
29      {
30          //特判
```

```
31      if(nx[last][c])
32      {
33          int p=last,x=nx[p][c];
34          if(mxlen[x]==mxlen[p]+1)
35          {
36              last=x;
37              //染色
38              int cur=last;
39              for(; cur && ct[cur]<=1 && color[cur]!=id; cur=link[cur])
40                  color[cur]=id, ct[cur]++;
41              return ;
42          }
43          else
44          {
45              int y=++idx;
46              mxlen[y]=mxlen[p]+1;
47              memcpy(nx[y],nx[x],26*sizeof(int));
48              link[y]=link[x];
49              link[x]=y;
50              for(; p!=-1 && nx[p][c]==x; p=link[p]) nx[p][c]=y;
51              color[y]=color[x];
52              ct[y]=ct[x];
53              last=y;
54              //染色
55              int cur=last;
56              for(; cur && ct[cur]<=1 && color[cur]!=id; cur=link[cur])
57                  color[cur]=id, ct[cur]++;
58              return ;
59          }
60      }
61
62      //普通SAM插入
63      int p=last,cur=last=++idx;
64      mxlen[cur]=mxlen[p]+1;
65      for(; p && !nx[p][c]; p=link[p]) nx[p][c]=cur;
66      if(!p) link[cur]=1;
67      else
68      {
69          int x=nx[p][c];
70          if(mxlen[x]==mxlen[p]+1) link[cur]=x;
71          else
72          {
73              int y=++idx;
74              mxlen[y]=mxlen[p]+1;
75              memcpy(nx[y],nx[x],26*sizeof(int));
76              link[y]=link[x];
77              link[cur]=link[x]=y;
78              for(; p!=-1 && nx[p][c]==x; p=link[p]) nx[p][c]=y;
79              color[y]=color[x];
80              ct[y]=ct[x];
81          }
82      }
83
```

```
        //染色
        for(; cur && ct[cur]<=1 && color[cur]!=id; cur=link[cur])
            color[cur]=id, ct[cur]++;
    }
    void add_string(int id)
    {
        last=1;
        cin>>s;
        for(auto ch:s) insSAM(ch-'a',id);
    }
};

GSA gsa;

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    gsa.init();
    cin>>n;
    for(int i=1;i<=n;i++)
        gsa.add_string(i);
    for(int i=2;i<=gsa.idx;i++)
        if(ct[i]==1) ans[color[i]]+=gsa.mxlen[i]-gsa.mxlen[gsa.link[i]];
    for(int i=1;i<=n;i++)
        cout<<ans[i]<<"\n";

    return 0;
}
```

1.9.3 GSAM+ 线段树合并

```
/*
CF666E. Forensic Examination
给定一个基串S，给定n个串T_i，q次询问。
每次询问给定参数l，r，pl，pr，
求【T_l,T_r】中S[pl,pr]出现次数做多的串的下标及最多出现次数。


SOLUTION:
线段树维护每个节点包含串的 出现id 和 对应id下的出现次数 【注意空间，不存l,r】

GSAM
每次询问:
倍增找到第一个包含S[pl,pr]的节点;
查询该节点 在区间[l,r]内的最值即可
*/

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

#define dd 1
#define ll long long
```

```
const int N=11e5/dd+100,M=12e6/dd+100;
ll n,m,q;
string s;

struct node
{
    int id,cnt;
};

node max(const node &a,const node &b)
{
    if(a.cnt==b.cnt)
        return a.id<b.id? a:b;
    return a.cnt>b.cnt? a:b;
}

//==sgt==
struct SGT
{
    struct TR
    {
        int ls,rs;node nd;
    };
    TR tr[M];
    int idx;

    void pushup(int x)
    {
        tr[x].nd = max(tr[tr[x].ls].nd,tr[tr[x].rs].nd);
    }
    void insert(int l,int r,int &x,int pos,int c)
    {
        if(!x) x=++idx;
        if(l==r)
        {
            tr[x].nd={pos,c};
            return;
        }
        int mid=l+r>>1;
        if(pos<=mid) insert(l,mid,tr[x].ls,pos,c);
        else insert(mid+1,r,tr[x].rs,pos,c);
        pushup(x);
    }
    node query(int l,int r,int L,int R,int x)
    {
        if(L>=l && R<=r)
            return tr[x].nd;
        int mid=L+R>>1;
        node ret={0,0};
        if(l<=mid) ret=max(ret, query(l,r,L,mid,tr[x].ls));
        if(r>mid) ret=max(ret, query(l,r,mid+1,R,tr[x].rs));
        return ret;
    }
```

```
75      int merge(int l,int r,int x,int y)
76      {
77          if(!x||!y) return x|y;
78          if(l==r)
79          {
80              int p=++idx;
81              tr[p].nd=tr[x].nd;
82              tr[p].nd.cnt+=tr[y].nd.cnt;
83              return p;
84          }
85          int p=++idx;
86          int mid=l+r>>1;
87          tr[p].ls=merge(l,mid,tr[x].ls,tr[y].ls);
88          tr[p].rs=merge(mid+1,r,tr[x].rs,tr[y].rs);
89          pushup(p);
90          return p;
91      }
92  };
93  SGT sgt;
94  //==sgt==
95
96  //==GSAM==
97  struct GSAM
98  {
99      int idx,last;
100     int nx[N][26],mxl[N],link[N];
101
102     int fa[N][23];
103     int rt[N];
104     map<int,map<int,int>> fp;
105     vector<int> e[N];
106     int pos[N];
107
108     void init()
109     {
110         idx=last=1;
111     }
112     void add_char(int c)
113     {
114         if(nx[last][c])
115         {
116             int p=last,x=nx[p][c];
117             if(mxl[x]==mxl[p]+1)
118             {
119                 last=x;
120                 return ;
121             }else
122             {
123                 int y=++idx;
124                 mxl[y]=mxl[p]+1;
125                 memcpy(nx[y],nx[x],sizeof(nx[x]));
126                 for(; p && nx[p][c]==x; p=link[p]) nx[p][c]=y;
127                 link[y]=link[x];
```

```
            link[x]=y;
            last=y;
            return ;
        }
    }

    int p=last,cur=last=++idx;
    mxl[cur]=mxl[p]+1;
    for(; p && !nx[p][c]; p=link[p]) nx[p][c]=cur;
    if(!p) link[cur]=1;
    else
    {
        int x=nx[p][c];
        if(mxl[x]==mxl[p]+1) link[cur]=x;
        else
        {
            int y=++idx;
            mxl[y]=mxl[p]+1;
            memcpy(nx[y],nx[x],sizeof(nx[x]));
            link[y]=link[x];
            link[cur]=link[x]=y;
            for(; p && nx[p][c]==x; p=link[p]) nx[p][c]=y;
        }
    }
}
void add_string(string &s,int id)
{
    last=1;
    for(auto ch:s) add_char(ch-'a');
    int p=1;
    for(int i=0;i<s.size();i++)
    {
        p=nx[p][s[i]-'a'];
        if(id==0) pos[i+1]=p;
        else fp[p][id]=1;
    }
}
void dfs(int u)
{
    for(int i=1;i<=22;i++) fa[u][i]=fa[fa[u][i-1]][i-1];
    for(auto it:fp[u])
        sgt.insert(1,m,rt[u],it.first,it.second);
    for(auto v:e[u]) dfs(v),rt[u]=sgt.merge(1,m,rt[u],rt[v]);
}
void build()
{
    for(int i=2;i<=idx;i++) fa[i][0]=link[i],e[link[i]].push_back(i);
    dfs(1);
}
void query(int l,int r,int ql,int qr)
{
    int p=pos[qr],len=qr-ql+1;
    for(int i=22;i>=0;i--)
```

```
        {
            int f=fa[p][i];
            if(f>1 && mxl[f]>=len) p=f;
        }
        auto ans=sgt.query(l,r,1,m,rt[p]);
        if(ans.cnt==0) ans.id=l;
        cout<<ans.id<<" "<<ans.cnt<<"\n";
    }
};
GSAM sam;
//==GSAM==

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>s;
    sam.init();
    sam.add_string(s,0);
    cin>>m;
    for(int i=1;i<=m;i++) cin>>s,sam.add_string(s,i);
    sam.build();
    cin>>q;
    while(q--)
    {
        int l,r,pl,pr;
        cin>>l>>r>>pl>>pr;
        sam.query(l,r,pl,pr);
    }

    return 0;
}


/*
Codeforces 547E. Mike and Friends
给定N个串，Q次询问。
每次询问给定参数l,r,k，问S_k在[S_l,S_r]内的出现次数。

SOLUTION1 - GSAM+线段树合并
对N个串建GSAM，对每个节点记录其在第几个串的出现次数。
对每个节点开一个线段树，记录其在每个出现的串中的出现次数。
（pos-串的下标，add-该串中的出现次数）
DFSparent树，进行线段树合并。
然后每次询问，查询第k个串所代表的节点的线段树中 区间[l,r] 内的总出现次数。

可参考CF666E

SOLUTION2 - AC自动机上dfs序建主席树（待补完）

*/

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

```
234
235 #define ll long long
236 #define pi pair<ll,ll>
237 const int N=4e5+100,M=N*20;
238 ll t,n,q;
239 string s;
240 ll a[N];
241
242 //==sgt==
243 struct SGT
244 {
245     struct TR
246     {
247         int ls,rs,sum;
248     };
249     TR tr[M];
250     int idx;
251
252     void pushup(int x)
253     {
254         tr[x].sum = tr[tr[x].ls].sum + tr[tr[x].rs].sum;
255     }
256     void insert(int l,int r,int &x,int pos,int c)
257     {
258         if(!x) x=++idx;
259         if(l==r)
260         {
261             tr[x].sum+=c;
262             return;
263         }
264         int mid=l+r>>1;
265         if(pos<=mid) insert(l,mid,tr[x].ls,pos,c);
266         else insert(mid+1,r,tr[x].rs,pos,c);
267         pushup(x);
268     }
269     int query(int l,int r,int L,int R,int x)
270     {
271         if(L>=l && R<=r)
272             return tr[x].sum;
273         int mid=L+R>>1;
274         int ret=0;
275         if(l<=mid) ret+=query(l,r,L,mid,tr[x].ls);
276         if(r>mid) ret+=query(l,r,mid+1,R,tr[x].rs);
277         return ret;
278     }
279     int merge(int l,int r,int x,int y)
280     {
281         if(!x||!y) return x|y;
282         int p=++idx;
283         if(l==r)
284         {
285             tr[p].sum = tr[x].sum + tr[y].sum;
286             return p;
```

```
287         }
288         int mid=l+r>>1;
289         tr[p].ls=merge(l,mid,tr[x].ls,tr[y].ls);
290         tr[p].rs=merge(mid+1,r,tr[x].rs,tr[y].rs);
291         pushup(p);
292         return p;
293     }
294 };
295 SGT sgt;
296 //==sgt==
297 
298 //==GSAM==
299 struct GSAM
300 {
301     int idx,last;
302     int nx[N][26],mxl[N],link[N];
303 
304     map<int,map<int,int>> fp;
305     vector<int> e[N];
306     int edp[N];
307     int rt[N];
308 
309     void init()
310     {
311         idx=last=1;
312     }
313     void add_char(int c)
314     {
315         if(nx[last][c])
316         {
317             int p=last,x=nx[p][c];
318             if(mxl[x]==mxl[p]+1)
319             {
320                 last=x;
321                 return ;
322             }else
323             {
324                 int y=++idx;
325                 mxl[y]=mxl[p]+1;
326                 memcpy(nx[y],nx[x],sizeof(nx[x]));
327                 for(; p && nx[p][c]==x; p=link[p]) nx[p][c]=y;
328                 link[y]=link[x];
329                 link[x]=y;
330                 last=y;
331                 return ;
332             }
333         }
334 
335         int p=last,cur=last=++idx;
336         mxl[cur]=mxl[p]+1;
337         for(; p && !nx[p][c]; p=link[p]) nx[p][c]=cur;
338         if(!p) link[cur]=1;
339         else
```

```
340         {
341             int x=nx[p][c];
342             if(mxl[x]==mxl[p]+1) link[cur]=x;
343             else
344             {
345                 int y=++idx;
346                 mxl[y]=mxl[p]+1;
347                 memcpy(nx[y],nx[x],sizeof(nx[x]));
348                 link[y]=link[x];
349                 link[cur]=link[x]=y;
350                 for(; p && nx[p][c]==x; p=link[p]) nx[p][c]=y;
351             }
352         }
353     }
354     void add_string(string &s,int id)
355     {
356         last=1;
357         for(auto ch:s) add_char(ch-'a');
358         edp[id]=last;
359         int p=1;
360         for(int i=0;i<s.size();i++)
361         {
362             p=nx[p][s[i]-'a'];
363             fp[p][id]=1;
364         }
365     }
366     void dfs(int u)
367     {
368         for(auto it:fp[u])
369             sgt.insert(1,n,rt[u],it.first,it.second);
370         for(auto v:e[u]) dfs(v),rt[u]=sgt.merge(1,n,rt[u],rt[v]);
371     }
372     void build()
373     {
374         for(int i=2;i<=idx;i++) e[link[i]].push_back(i);
375         dfs(1);
376     }
377 };
378 GSAM sam;
379 //==GSAM==
380 
381 int main()
382 {
383     ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
384     cin>>n>>q;
385     sam.init();
386     for(int i=1;i<=n;i++) cin>>s,sam.add_string(s,i);
387     sam.build();
388     while(q--)
389     {
390         int l,r,k;
391         cin>>l>>r>>k;
392         int p=sam.edp[k];
```

```
393         cout<<sgt.query(l,r,1,n,sam.rt[p])<<"\n";
394     }
395 
396     return 0;
397 }
```

1.10 回文树

1.10.1 PAM-base

```
1  /*
2  ref: https://www.cnblogs.com/lhm-/p/13293090.html
3  
4  Template format
5  len: 节点对应的回文字串的长度;
6  fail: fail指针, 指向该节点所对应的回文子串的最长回文后缀所对应的节点;
7  tr: 树的边转移, 转移为向当前回文子串两端加上一个字符。
8  
9  一个字符串的回文树由两棵树组成: 奇树, 偶树。
10 为方便处理, 奇树的根的len设为-1, fail为其本身; 偶树的根的len设为0, fail为奇树根。
11 增量法构造
12 */
13 #include<bits/stdc++.h>
14 using namespace std;
15 
16 const int N=1e6+100;
17 
18 struct PAM
19 {
20     int tr[N][26],len[N],fail[N];
21     int anc[N],dif[N];
22     int idx,last;
23     string s;
24     vector<int> e[N];
25     int sz[N];
26 
27     void init()
28     {
29         len[1]=-1;
30         fail[0]=fail[1]=idx=1;
31         last=0;
32         for(int i=0;i<26;i++) pe[0][i]=pe[1][i]=1;
33     }
34     //str下标从1开始
35     //==普通版==
36     void insert_base(int i)
37     {
38         int p=last,ch=s[i]-'a';
39         while(s[i]!=s[i-len[p]-1]) p=fail[p];
40         if(tr[p][ch])
41         {
42             last=tr[p][ch];
43             sz[last]++; //sz更新
```

```
44          return ;
45      }
46      int x=fail[p],cur=++idx;
47      while(s[i]!=s[i-len[x]-1]) x=fail[x];
48      fail[cur]=tr[x][ch];
49      len[cur]=len[p]+2;
50      tr[p][ch]=last=cur;
51      sz[last]++; //sz更新
52  }
53  //==常数优化版==
54  int getf(int p,int i)
55  {
56      while(s[i]!=s[i-len[p]-1])
57          if(s[i]==s[i-len[fail[p]]-1]) return fail[p];
58          else p=anc[p];
59      return p;
60  }
61  //str下标从1开始
62  void insert(int i)
63  {
64      int p=getf(last,i),ch=s[i]-'a';
65      if(tr[p][ch])
66      {
67          last=tr[p][ch];
68          return ;
69      }
70      int x=getf(fail[p],i),cur=++idx;
71      fail[cur]=tr[x][ch];
72      len[cur]=len[p]+2;
73      tr[p][ch]=last=cur;
74      //improve
75      dif[idx]=len[idx]-len[fail[idx]];
76      if(dif[idx]==dif[fail[idx]]) anc[idx]=anc[fail[idx]];
77      else anc[idx]=fail[idx];
78  }
79  void add_string()
80  {
81      s.insert(0," ");
82      int n=s.size()-1;
83      for(int i=1;i<=n;i++) insert(i);
84  }
85  void build()
86  {
87      //fail_Tree
88      // for(int i=0;i<=idx;i++) if(i!=1) e[fail[i]].push_back(i);
89      //本质不同回文子串个数即为回文树除两个根节点之外的节点个数
90      // int bnum = idx-1;
91      //每个节点对应回文子串的出现次数: fail树拓扑序更新
92      for(int i=idx;i>=0;i--) sz[fail[i]]+=sz[i];
93  }
94  void solve()
95  {
96      long long ans=0;
```

```
        for(int i=2;i<=idx;i++) ans=max(ans,1ll*sz[i]*len[i]);
        cout<<ans<<"\n";
    }
};
PAM pam;

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    pam.init();
    cin>>pam.s;
    pam.add_string();
    pam.build();
    pam.solve();

    return 0;
}
```

1.10.2 回文匹配

```
/*
P6216 回文匹配
https://www.luogu.com.cn/problem/P6216

给定字符串S1,S2，对于S1中所有长度为奇数的回文串S1[l,r]，
S1的分数会增加S2在S1[l,r]中的出现次数。（答案对2^32取模）
1.PAM求出S1所有长度为奇数的回文串
2.KMP初始化S2在S1[1,i]中的出现次数，
  然后对于S1的奇数长度回文串，O(1)求出S2的出现次数，累加即可。
  ct[l,r] = ct[r]-ct[l-1]
*/
#include<bits/stdc++.h>
using namespace std;

#define dg 1
const int N=3e6/dg+100;
int n,m;

struct PAM
{
    int tr[N][26],len[N],fail[N];
    int idx,last;
    string s,t;
    vector<int> e[N];
    unsigned int id[N],sz[N];
    unsigned int nx[N],ct[N];
    unsigned int ans=0;

    void init()
    {
        len[1]=-1;
        fail[0]=fail[1]=idx=1;
```

```
33          last=0;
34      }
35      //str下标从1开始
36      void insert(int i)
37      {
38          int p=last,ch=s[i]-'a';
39          while(s[i]!=s[i-len[p]-1]) p=fail[p];
40          if(tr[p][ch])
41          {
42              last=tr[p][ch];
43              return ;
44          }
45          int x=fail[p],cur=++idx;
46          while(s[i]!=s[i-len[x]-1]) x=fail[x];
47          fail[cur]=tr[x][ch];
48          len[cur]=len[p]+2;
49          tr[p][ch]=last=cur;
50      }
51      void add_string()
52      {
53          s.insert(0," "); t.insert(0," ");
54          for(int i=1;i<=n;i++)
55          {
56              insert(i),sz[last]++;
57              id[last]=i;
58          }
59      }
60      void kmp()
61      {
62          for(int i=2,j=0;i<=m;i++)
63          {
64              while(j && t[i]!=t[j+1]) j=nx[j];
65              if(t[i]==t[j+1]) j++;
66              nx[i]=j;
67          }
68          for(int i=1,j=0;i<=n;i++)
69          {
70              while(j && s[i]!=t[j+1]) j=nx[j];
71              if(s[i]==t[j+1]) j++;
72              ct[i]=ct[i-1];
73              if(j==m)
74              {
75                  ct[i]++;
76                  j=nx[j];
77              }
78          }
79      }
80      void solve()
81      {
82          for(int i=idx;i>=0;i--) sz[fail[i]]+=sz[i];
83          for(int i=idx;i>=2;i--)
84          {
85              if((len[i]&1) && len[i]>=m)
```

```
86          {
87              unsigned int cnt = ct[id[i]]-ct[id[i]-len[i]+m-1];
88              ans += cnt * sz[i];
89          }
90       }
91       cout<<ans<<"\n";
92    }
93 };
94 PAM pam;
95 
96 int main()
97 {
98    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
99    cin>>n>>m;
100   pam.init();
101   cin>>pam.s>>pam.t;
102   pam.add_string();
103   pam.kmp();
104   pam.solve();
105 
106   return 0;
107 }
```

1.10.3 小常数优化

```
1  /**
2   * 【45-ICPC-昆明-F.Generating Strings】
3   * 设有一个字符串生成器，能生成指定长度的字符串T(保证字符集为小写字母)
4   * 给定生成长度N，操作次数M，原始字符串S
5   * 定义V(T)为：每个生成的T的所有子串中，回文子串在S中出现次数的和。
6   * 定义操作：
7   * - 1.1 c：在S末尾添加字符c
8   * - 2.2 ：删去S的最后一个字符
9   * 输入的后m行为操作输入
10  * 现要求，初识V(T)，和每次操作后的V(T)。（答案取模）
11  *
12  * SOLUTION:
13  * 对S建PAM，维护一个add[]，
14  * add[p]：添加p这个节点后，该节点对答案的贡献。
15  * 考虑如何维护add[p]：
16  * 考虑插入PAM的时候，当前找到/生成的回文串出现次数+1，且其所有父亲节点的回文串出现次数+1.
17  * 故有以下递推：
18  * - Case1. N<len[p]: add[p]=add[fail[p]].
19  * 【当前回文串长超过N，故该节点的贡献只有串长小于N的部分】
20  * - Case2. N>=len[p]: add[p]=add[fail[p]]+ksm(26,N-len[p])*(N-len[p]+1);
21  * 【当前节点的贡献 = 其父亲节点的贡献 + 当前节点所代表的新回文串的贡献】
22  *
23  * ATTENTION:
24  * 建PAM需要优化常数。
25  */
26 #include<bits/stdc++.h>
27 using namespace std;
```

```
28
29  #define ll long long
30  const int N=1e6+100,P=1e9+7,base=26;
31  int t,n,m;
32  int bs[N],val[N];
33  string s;
34  int slen,ts;
35
36  void bsinit()
37  {
38      bs[0]=1;
39      for(int i=1;i<N;i++) bs[i]=(1ll*bs[i-1]*base)%P;
40  }
41
42  struct PAM
43  {
44      int tr[N][26],len[N],fail[N];
45      int idx,last;
46      int id[N],add[N];
47      int anc[N],dif[N]; //奇怪的优化
48
49      void init()
50      {
51          len[1]=-1;
52          fail[0]=fail[1]=1;
53          idx=1;
54          last=0;
55      }
56      void cls()
57      {
58          for(int i=0;i<=idx;i++)
59          {
60              memset(tr[i],0,sizeof(tr[i]));
61              len[i]=fail[i]=add[i]=0;
62              id[i]=0;
63              anc[i]=dif[i]=0;
64          }
65          len[1]=-1;
66          fail[0]=fail[1]=1;
67          idx=1;
68          last=0;
69      }
70      int getf(int p,int i)
71      {
72          while(s[i]!=s[i-len[p]-1])
73              if(s[i]==s[i-len[fail[p]]-1]) return fail[p];
74              else p=anc[p];
75          return p;
76      }
77      void insert(int i)
78      {
79          int p=getf(last,i),ch=s[i]-'a';
80          if(tr[p][ch])
```

```
81          {
82              last=tr[p][ch];
83              return ;
84          }
85          int x=getf(fail[p],i),cur=++idx;
86          fail[cur]=tr[x][ch];
87          len[cur]=len[p]+2;
88          tr[p][ch]=last=cur;
89          //improve
90          dif[idx]=len[idx]-len[fail[idx]];
91          if(dif[idx]==dif[fail[idx]]) anc[idx]=anc[fail[idx]];
92          else anc[idx]=fail[idx];
93          //calc
94          ll r=n-len[last];
95          if(r<0) add[last] = add[fail[last]];
96          else add[last] = (add[fail[last]]+bs[r]*(r+1)%P)%P;
97      }
98      void add_string()
99      {
100         for(int i=1;i<=slen;i++)
101             insert(i),id[i]=last,ts=(ts+add[last])%P;
102     }
103
104     void solve()
105     {
106         cout<<ts<<"\n";
107         while(m--)
108         {
109             ll op; char ch;
110             cin>>op;
111             if(op==1)
112             {
113                 cin>>ch;
114                 s.push_back(ch),slen++;
115                 insert(slen);
116                 id[slen]=last;
117
118                 ts=(ts+add[last])%P;
119                 cout<<ts<<"\n";
120
121             }else
122             {
123                 s.pop_back();
124                 ts=(ts-add[last]+P)%P;
125                 last=id[--slen];
126                 cout<<ts<<"\n";
127             }
128         }
129     }
130 };
131
132 PAM pam;
133
```

```
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    bsinit();
    cin>>t;
    pam.init();
    while(t--)
    {
        ts=0;
        cin>>n>>m>>s;
        slen=s.size(), s.insert(0," ");
        pam.add_string();
        pam.solve();
        pam.cls();
    }

    return 0;
}
```

1.11 最小表示法-循环同构

```
//字符串 S 的最小表示为与 S 循环同构的所有字符串中字典序最小的字符串.
// Lyndon分解求最小表示
// smallest_cyclic_string
string min_cyclic_string(string s)
{
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j])
        {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k) i += j - k;
    }
    return s.substr(ans, n / 2);
}

// 普通版
string get_min(string &s)
{
    int n=s.size();
    s+=s;

    int i=0,j=1;
```

```
    while(i<n && j<n)
    {
        int k=0;
        while(k<n && s[i+k]==s[j+k]) k++;
        if(k==n) break;
        if(s[i+k]>s[j+k]) i+=k+1;
        else j+=k+1;
        if(i==j) i++;
    }
    int pos=min(i,j);
    return s.substr(pos,n);
}
```

1.12 序列自动机

```
/**
 * 【序列自动机基础模板】
*/
#include<bits/stdc++.h>
using namespace std;

#define dg 1
#define ll long long
const int N=2e5/dg+100;
ll n,inf;

struct SeqAM
{
    int tr[N][26],len;
    string s;
    ll ct[N]; //ct[i]: 以i为起点的子序列数目。【可能很大，看是否取模】

    void build()
    {
        int n=s.size();
        inf=n+1;
        for(int j=0;j<26;j++) tr[n][j]=inf;
        for(int i=n;i>=1;i--)
        {
            for(int j=0;j<26;j++) tr[i-1][j]=tr[i][j];
            tr[i-1][s[i-1]-'a']=i;
        }
    }

    bool qry(string &t) //查询T是否是S的子序列
    {
        int u=0;
        for(int i=0;i<t.size();i++)
        {
            u=tr[u][t[i]-'a'];
            if(u==inf) return 0;
        }
```

```
38          return 1;
39      }
40
41      // dfs(0): 求本质不同子序列数目
42      ll dfs(int x)
43      {
44          if(ct[x]) return ct[x];
45          for(int i=0;i<26;i++)
46          {
47              int v=tr[x][i];
48              if(v==inf) continue;
49              ct[x]+=dfs(v);
50          }
51          if(x) ct[x]++;
52          return ct[x];
53      }
54  };
55
56  SeqAM s1,s2,s3;
57
58  //【求3个串的公共子序列数目】
59  //f[x][y][z]: 表示以x,y,z为起点的公共子序列的个数
60  //显然有转移: f[x][y][z]+=f[x1][y1][z1]; 同时可以记忆化搜索
61  const int M=200, P=998244353;
62  int f[M][M][M];
63  ll dfs(int x,int y,int z)
64  {
65      if(f[x][y][z]) return f[x][y][z];
66      for(int i=0;i<26;i++)
67      {
68          int x1=s1.tr[x][i], y1=s2.tr[y][i], z1=s3.tr[z][i];
69          if(x1!=inf && y1!=inf && z1!=inf)
70              (f[x][y][z]+=dfs(x1,y1,z1)) % P;
71      }
72      if(x||y||z) f[x][y][z]++;
73      return f[x][y][z]%P;
74  }
75
76  //【求字符串的回文子序列个数-未测试】
77  //对原串和逆串建SeqAM
78  int vis[M][M];
79  ll dfs(int x,int y)
80  {
81      if(vis[x][y]) return vis[x][y];
82      for(int i=0;i<26;i++)
83      {
84          int x1=s1.tr[x][i],y1=s2.tr[y][i];
85          if(x1!=inf && y1!=inf)
86          {
87              if(x1+y1>n+1) continue;
88              if(x1+y1<n+1) vis[x][y]++;
89              (vis[x][y]+=dfs(x1,y1)) %= P;
90          }
```

```
    }
    return ++vis[x][y];
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>s1.s;
    s1.build();
    cout<<s1.dfs(0)<<"\n";
    cin>>n;
    while(n--)
    {
        string t;
        cin>>t;
        cout<<(s1.qry(t)?"YES\n":"NO\n");
    }

    return 0;
}


/**
 * 【2021年中国大学生程序设计竞赛女生专场-B】
 * 【多次区间询问，求最短的不是S[l,r]的子序列的长度 - 倍增】
 * https://codeforces.com/gym/103389/problem/B
 *
*/
#include<bits/stdc++.h>
using namespace std;

#define dg 1
const int N=2e5/dg+100;
int m,n,q;

struct SeqAM
{
    string s;
    //tr[i][c]: 第i个位置往后的字符c的首次出现位置
    //mxd[i]: 第i个位置下一步所能跳的最远位置
    int tr[N][26],inf,mxd[N];
    //倍增数组
    int to[N][22];

    void build()
    {
        inf=n+1;
        // s.insert(0," ");
        for(int i=0;i<m;i++) tr[n][i]=inf;
        for(int i=n;i>=1;i--)
        {
            for(int j=0;j<m;j++) tr[i-1][j]=tr[i][j];
            tr[i-1][s[i-1]-'a']=i;
```

```
        }
        //build
        for(int i=0;i<=n;i++)
        {
            mxd[i]=0;
            for(int j=0;j<m;j++) mxd[i] = max(mxd[i],tr[i][j]);
            to[i][0]=mxd[i];
        }
        for(int i=0;i<=20;i++) to[inf][i]=inf;
        for(int i=1;i<=20;i++)
            for(int u=0;u<=n;u++)
                to[u][i]=to[to[u][i-1]][i-1];
    }

    void solve(int l,int r)
    {
        int u=l-1,ans=0;
        for(int i=20;i>=0;i--)
        {
            if(to[u][i]<=r)
            {
                u=to[u][i];
                ans+=1<<i;
            }
        }
        cout<<ans+1<<"\n";
    }
};

SeqAM sq;

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>m>>n>>sq.s;
    sq.build();
    cin>>q;
    while(q--)
    {
        int l,r;
        cin>>l>>r;
        sq.solve(l,r);
    }

    return 0;
}
```

2 FFT 与 NTT

2.1 FFT

```
/* 01串匹配
```

```
2  H.Rock Paper Scissors
3  */
4  #include <bits/stdc++.h>
5  using namespace std;
6
7  #define ll long long
8  int na, nb;
9
10 //* 板子 from https://github.com/KanadeSiina/StdLibrary/blob/master
11 namespace fft
12 {
13     struct num
14     {
15         double x,y;
16         num() {x=y=0;}
17         num(double x,double y):x(x),y(y){}
18     };
19     inline num operator+(num a,num b) {return num(a.x+b.x,a.y+b.y);}
20     inline num operator-(num a,num b) {return num(a.x-b.x,a.y-b.y);}
21     inline num operator*(num a,num b) {return num(a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x);}
22     inline num conj(num a) {return num(a.x,-a.y);}
23
24     int base=1;
25     vector<num> roots={{0,0},{1,0}};
26     vector<int> rev={0,1};
27     const double PI=acosl(-1.0);
28
29     void ensure_base(int nbase)
30     {
31         if(nbase<=base) return;
32         rev.resize(1<<nbase);
33         for(int i=0;i<(1<<nbase);i++)
34             rev[i]=(rev[i>>1]>>1)+((i&1)<<(nbase-1));
35         roots.resize(1<<nbase);
36         while(base<nbase)
37         {
38             double angle=2*PI/(1<<(base+1));
39             for(int i=1<<(base-1);i<(1<<base);i++)
40             {
41                 roots[i<<1]=roots[i];
42                 double angle_i=angle*(2*i+1-(1<<base));
43                 roots[(i<<1)+1]=num(cos(angle_i),sin(angle_i));
44             }
45             base++;
46         }
47     }
48
49     void fft(vector<num> &a,int n=-1)
50     {
51         if(n==-1) n=a.size();
52         assert((n&(n-1))==0);
53         int zeros=__builtin_ctz(n);
54         ensure_base(zeros);
```

```
55          int shift=base-zeros;
56          for(int i=0;i<n;i++)
57              if(i<(rev[i]>>shift))
58                  swap(a[i],a[rev[i]>>shift]);
59          for(int k=1;k<n;k<<=1)
60          {
61              for(int i=0;i<n;i+=2*k)
62              {
63                  for(int j=0;j<k;j++)
64                  {
65                      num z=a[i+j+k]*roots[j+k];
66                      a[i+j+k]=a[i+j]-z;
67                      a[i+j]=a[i+j]+z;
68                  }
69              }
70          }
71      }
72
73      vector<num> fa,fb;
74
75      vector<int> multiply(vector<int> &a, vector<int> &b)
76      {
77          int need=a.size()+b.size()-1;
78          int nbase=0;
79          while((1<<nbase)<need) nbase++;
80          ensure_base(nbase);
81          int sz=1<<nbase;
82          if(sz>(int)fa.size()) fa.resize(sz);
83          for(int i=0;i<sz;i++)
84          {
85              int x=(i<(int)a.size()?a[i]:0);
86              int y=(i<(int)b.size()?b[i]:0);
87              fa[i]=num(x,y);
88          }
89          fft(fa,sz);
90          num r(0,-0.25/sz);
91          for(int i=0;i<=(sz>>1);i++)
92          {
93              int j=(sz-i)&(sz-1);
94              num z=(fa[j]*fa[j]-conj(fa[i]*fa[i]))*r;
95              if(i!=j) fa[j]=(fa[i]*fa[i]-conj(fa[j]*fa[j]))*r;
96              fa[i]=z;
97          }
98          fft(fa,sz);
99          vector<int> res(need);
100         for(int i=0;i<need;i++) res[i]=fa[i].x+0.5;
101         return res;
102     }
103 };
104
105 int main()
106 {
107     ios::sync_with_stdio(0);cin.tie(0);
```

```
    cin >> na >> nb;
    vector<char> a(na,0),b(nb,0);
    for (int i = 0; i < na; i++)
        cin>>a[i];
    for (int i = 0; i < nb; i++)
        cin>>b[i];
    reverse(b.begin(),b.end());

    vector<int> vra(na,0),vpb(nb,0);
    for (int i = 0; i < na; i++)
        if(a[i]=='R') vra[i]=1;
    for (int i = 0; i < nb; i++)
        if(b[i]=='P') vpb[i]=1;

    vector<int> P = fft::multiply(vra,vpb);

    vector<int> vsa(na,0),vrb(nb,0);
    for (int i = 0; i < na; i++)
        if(a[i]=='S') vsa[i]=1;
    for (int i = 0; i < nb; i++)
        if(b[i]=='R') vrb[i]=1;

    vector<int> R = fft::multiply(vsa,vrb);

    vector<int> vpa(na,0),vsb(nb,0);
    for (int i = 0; i < na; i++)
        if(a[i]=='P') vpa[i]=1;
    for (int i = 0; i < nb; i++)
        if(b[i]=='S') vsb[i]=1;

    vector<int> S = fft::multiply(vpa,vsb);

    int ans = -1;
    for (int i = nb-1; i < P.size(); i++)
    {
        int t = P[i]+S[i]+R[i];
        ans = max(ans,t);
    }
    cout<<ans<<"\n";

    return 0;
}

//给定数组a，且有1≤|i-j|≤N,求差值|i-j|是否出现
/*
构造01数组A：$A[i]$表示$i$是否出现，其多项式意义为$X^i$是否存在；

构造01数组B：$B[i]$表示$-i$是否出现，由于数组下标不能为负数，转化为$B[N-i]$表示$-i$是否出现，其多项式意义为$X^{N-i}$是否存在。

卷积公式：

$$
```

```
C_{i+N-j}=\sum^k_{i+N-j=k}A_iB_{N-j}
$$

卷积数组C：$C[i+N-j]$表示$X^{i+N-j}$的系数，当该值不为0时，即意味着存在$i+N-j$。

显然，遍历C数组中区间$[N,2*N]$中的值，就可求得差值$i-N$是否出现.

同时，受绝对值的影响，以下两种遍历等价：

- $0≤i≤N$：$C[i]$表示差值$N-i$是否出现.

- $N≤i≤2*N$：$C[i]$表示差值$i-N$是否出现.
*/
/*
FFT板子
...
*/

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>n;
    vector<int> a(N+1,0);
    vector<int> b(N+1,0);
    for(int i=0,x;i<n;i++)
        cin>>x,a[x]=1,b[N-x]=1;
    auto res = fft::multiply(a,b);

    for(int i=0;i<=N;i++)
        if(res[i])
            vis[N-i]=1;
    // 下面写法等价
    // for(int i=N;i<=2*N;i++)
    // if(res[i])
    // vis[i-N]=1;

    int ans=0;
    for(int i=1;i<=N+1 && !ans;i++)
    {
        bool f=1;
        for(int j=i;j<=N && f;j+=i)
            if(vis[j])
                f=0;
        if(f)
            ans=i;
    }
    cout<<ans<<"\n";

    return 0;
}
```

2.2 多项式

```
/*
主要功能列表
using Poly = std::vector<int>;: 用 vector 表示多项式。
void dft/idft(int *a, n);: 长度为 n 的 dft/idft，需要保证 n=2^k。
void dft/idft(Poly &a);: 参数为多项式的dft/idft。
Poly operator*(Poly a, Poly b): 多项式乘法。
Poly operator+(Poly a, Poly b): 多项式加法。
Poly operator-(Poly a, Poly b): 多项式减法。
Poly operator/(Poly a, Poly b): 多项式除法。
std::pair<Poly, Poly> operator%(Poly a, Poly b): 多项式取模。
Poly inverse(Poly a);: 多项式乘法逆。
Poly sqrt(Poly a);: 多项式开根。
Poly deriv(Poly a);: 多项式求导。
Poly integ(Poly a);: 多项式积分。
Poly ln(Poly a);: 多项式对数函数。
Poly expNewton(Poly a);: 多项式指数函数（牛顿迭代，需要保证长度为 2 的幂）。
Poly exp2(Poly a);: 多项式指数函数（半在线卷积）。
Poly exp(Poly a);: 多项式指数函数（优化半在线卷积）。
Poly power(Poly a, int k);: 多项式幂函数。
Poly power(Poly a, int k1, int k2);: 多项式幂函数（模数对 P 和 φ(P) 取模）。
Poly divAt(Poly p, Poly q, LL n);: 计算 [x^n](P[x]/Q[x]).
*/

namespace Polynomial {
using Poly = std::vector<int>;
constexpr int P(998244353), G(3);
inline void inc(int &x, int y) { (x += y) >= P ? x -= P : 0; }
inline int mod(int64_t x) { return x % P; }
inline int fpow(int x, int k = P - 2) {
  int r = 1;
  for (; k; k >>= 1, x = 1LL * x * x % P)
    if (k & 1) r = 1LL * r * x % P;
  return r;
}
template <int N>
std::array<int, N> getOmega() {
  std::array<int, N> w;
  for (int i = N >> 1, x = fpow(G, (P - 1) / N); i; i >>= 1, x = 1LL * x * x % P) {
    w[i] = 1;
    for (int j = 1; j < i; j++) w[i + j] = 1LL * w[i + j - 1] * x % P;
  }
  return w;
}
auto w = getOmega<1 << 20>();
Poly &operator*=(Poly &a, int b) { for (auto &x : a) x = 1LL * x * b % P; return a; }
Poly operator*(Poly a, int b) { return a *= b; }
Poly operator*(int a, Poly b) { return b * a; }
Poly &operator/=(Poly &a, int b) { return a *= fpow(b); }
Poly operator/(Poly a, int b) { return a /= b; }
Poly &operator+=(Poly &a, Poly b) {
  a.resize(std::max(a.size(), b.size()));
```

```
52   for (int i = 0; i < b.size(); i++) inc(a[i], b[i]);
53   return a;
54 }
55 Poly operator+(Poly a, Poly b) { return a += b; }
56 Poly &operator-=(Poly &a, Poly b) {
57   a.resize(std::max(a.size(), b.size()));
58   for (int i = 0; i < b.size(); i++) inc(a[i], P - b[i]);
59   return a;
60 }
61 Poly operator-(Poly a, Poly b) { return a -= b; }
62 Poly operator-(Poly a) { for (auto &x : a) x ? x = P - x : 0; return a; }
63 Poly &operator>>=(Poly &a, int x) {
64   if (x >= (int)a.size()) {
65     a.clear();
66   } else {
67     a.erase(a.begin(), a.begin() + x);
68   }
69   return a;
70 }
71 Poly &operator<<=(Poly &a, int x) {
72   a.insert(a.begin(), x, 0);
73   return a;
74 }
75 Poly operator>>(Poly a, int x) { return a >>= x; }
76 Poly operator<<(Poly a, int x) { return a <<= x; }
77 inline Poly &dotEq(Poly &a, Poly b) {
78   assert(a.size() == b.size());
79   for (int i = 0; i < a.size(); i++) a[i] = 1LL * a[i] * b[i] % P;
80   return a;
81 }
82 inline Poly dot(Poly a, Poly b) { return dotEq(a, b); }
83 void norm(Poly &a) {
84   if (!a.empty()) {
85     a.resize(1 << std::__lg(a.size() * 2 - 1));
86   }
87 }
88 void dft(int *a, int n) {
89   assert((n & n - 1) == 0);
90   for (int k = n >> 1; k; k >>= 1) {
91     for (int i = 0; i < n; i += k << 1) {
92       for (int j = 0; j < k; j++) {
93         int y = a[i + j + k];
94         a[i + j + k] = 1LL * (a[i + j] - y + P) * w[k + j] % P;
95         inc(a[i + j], y);
96       }
97     }
98   }
99 }
100 void idft(int *a, int n) {
101   assert((n & n - 1) == 0);
102   for (int k = 1; k < n; k <<= 1) {
103     for (int i = 0; i < n; i += k << 1) {
104       for (int j = 0; j < k; j++) {
```

```
105         int x = a[i + j], y = 1LL * a[i + j + k] * w[k + j] % P;
106         a[i + j + k] = x - y < 0 ? x - y + P : x - y;
107         inc(a[i + j], y);
108       }
109     }
110   }
111   for (int i = 0, inv = P - (P - 1) / n; i < n; i++)
112     a[i] = 1LL * a[i] * inv % P;
113   std::reverse(a + 1, a + n);
114 }
115 void dft(Poly &a) { dft(a.data(), a.size()); }
116 void idft(Poly &a) { idft(a.data(), a.size()); }
117 Poly operator*(Poly a, Poly b) {
118   int len = a.size() + b.size() - 1;
119   if (a.size() <= 8 || b.size() <= 8) {
120     Poly c(len);
121     for (size_t i = 0; i < a.size(); i++)
122       for (size_t j = 0; j < b.size(); j++)
123         c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
124     return c;
125   }
126   int n = 1 << std::__lg(len - 1) + 1;
127   a.resize(n), b.resize(n);
128   dft(a), dft(b);
129   dotEq(a, b);
130   idft(a);
131   a.resize(len);
132   return a;
133 }
134 Poly invRec(Poly a) {
135   int n = a.size();
136   assert((n & n - 1) == 0);
137   if (n == 1) return {fpow(a[0])};
138   int m = n >> 1;
139   Poly b = invRec(Poly(a.begin(), a.begin() + m)), c = b;
140   b.resize(n);
141   dft(a), dft(b), dotEq(a, b), idft(a);
142   for (int i = 0; i < m; i++) a[i] = 0;
143   for (int i = m; i < n; i++) a[i] = P - a[i];
144   dft(a), dotEq(a, b), idft(a);
145   for (int i = 0; i < m; i++) a[i] = c[i];
146   return a;
147 }
148 Poly inverse(Poly a) {
149   int n = a.size();
150   norm(a);
151   a = invRec(a);
152   a.resize(n);
153   return a;
154 }
155 Poly operator/(Poly a, Poly b) { // return: c(len = n - m + 1), a = b * c + r
156   int n = a.size(), m = b.size();
157   if (n < m) return {0};
```

```
158   int k = 1 << std::__lg(n - m << 1 | 1);
159   std::reverse(a.begin(), a.end());
160   std::reverse(b.begin(), b.end());
161   a.resize(k), b.resize(k), b = invRec(b);
162   a = a * b;
163   a.resize(n - m + 1);
164   std::reverse(a.begin(), a.end());
165   return a;
166 }
167 std::pair<Poly, Poly> operator%(Poly a, Poly b) { // return: {c(len = n - m + 1), r(len =
        m - 1)}
168   int m = b.size();
169   Poly c = a / b;
170   b = b * c;
171   a.resize(m - 1);
172   for (int i = 0; i < m - 1; i++) inc(a[i], P - b[i]);
173   return {c, a};
174 }
175 Poly sqrt(Poly a) {
176   int raw = a.size();
177   int d = 0;
178   while (d < raw && !a[d]) d++;
179   if (d == raw) return a;
180   if (d & 1) return {};
181   norm(a >>= d);
182   int len = a.size();
183   Poly b(len), binv(1), bsqr{a[0]}, foo, bar; // sqrt, sqrt_inv, sqrt_sqr
184   auto sq = SqrtMod::sqrtMod(a[0], P);
185   if (sq.empty()) return {};
186   b[0] = sq[0], binv[0] = fpow(b[0]);
187   auto shift = [](int x) { return (x & 1 ? x + P : x) >> 1; }; // quick div 2
188   for (int m = 1, n = 2; n <= len; m <<= 1, n <<= 1) {
189     foo.resize(n), bar = binv;
190     for (int i = 0; i < m; i++) {
191       foo[i + m] = a[i] + a[i + m] - bsqr[i];
192       if (foo[i + m] >= P) foo[i + m] -= P;
193       if (foo[i + m] < 0) foo[i + m] += P;
194       foo[i] = 0;
195     }
196     binv.resize(n);
197     dft(foo), dft(binv), dotEq(foo, binv), idft(foo);
198     for (int i = m; i < n; i++) b[i] = shift(foo[i]);
199     // inv
200     if (n == len) break;
201     for (int i = 0; i < n; i++) foo[i] = b[i];
202     bar.resize(n), binv = bar;
203     dft(foo), dft(bar), bsqr = dot(foo, foo), idft(bsqr);
204     dotEq(foo, bar), idft(foo);
205     for (int i = 0; i < m; i++) foo[i] = 0;
206     for (int i = m; i < n; i++) foo[i] = P - foo[i];
207     dft(foo), dotEq(foo, bar), idft(foo);
208     for (int i = m; i < n; i++) binv[i] = foo[i];
209   }
```

```
  b <<= d / 2;
  b.resize(raw);
  return b;
}
Poly deriv(Poly a) {
  for (int i = 0; i + 1 < a.size(); i++) a[i] = (i + 1LL) * a[i + 1] % P;
  a.pop_back();
  return a;
}
std::vector<int> inv = {1, 1};
void updateInv(int n) {
  if ((int)inv.size() <= n) {
    int p = inv.size();
    inv.resize(n + 1);
    for (int i = p; i <= n; i++) inv[i] = 1LL * (P - P / i) * inv[P % i] % P;
  }
}
Poly integ(Poly a, int c = 0) {
  int n = a.size();
  updateInv(n);
  Poly b(n + 1);
  b[0] = c;
  for (int i = 0; i < n; i++) b[i + 1] = 1LL * inv[i + 1] * a[i] % P;
  return b;
}
Poly ln(Poly a) {
  int n = a.size();
  assert(a[0] == 1);
  a = inverse(a) * deriv(a);
  a.resize(n - 1);
  return integ(a);
}
// newton
// $O(n \log n)$, slower than exp2
Poly expNewton(Poly a) {
  int n = a.size();
  assert((n & n - 1) == 0);
  assert(a[0] == 0);
  if (n == 1) return {1};
  int m = n >> 1;
  Poly b = expNewton(Poly(a.begin(), a.begin() + m)), c;
  b.resize(n), c = ln(b);
  a.resize(n << 1), b.resize(n << 1), c.resize(n << 1);
  dft(a), dft(b), dft(c);
  for (int i = 0; i < n << 1; i++) a[i] = (1LL + P + a[i] - c[i]) * b[i] % P;
  idft(a);
  a.resize(n);
  return a;
}
// half-online conv
// $O(n\log^2n)$
// $b = e^a, b' = a'b$
// $(n+1)b_{n+1} = \sum_{i=0}^n a'_ib_{n-i}$
```

```
// $nb_n = \sum_{i=0}^{n-1} a'_ib_{n - 1 - i}$
Poly exp2(Poly a) {
  if (a.empty()) return {};
  assert(a[0] == 0);
  int n = a.size();
  updateInv(n);
  for (int i = 0; i + 1 < n; i++) {
    a[i] = a[i + 1] * (i + 1LL) % P;
  }
  a.pop_back();
  Poly b(n);
  b[0] = 1;
  for (int m = 1; m < n; m++) {
    int k = m & -m, l = m - k, r = std::min(m + k, n);
    Poly p(a.begin(), a.begin() + (r - l - 1));
    Poly q(b.begin() + l, b.begin() + m);
    p.resize(k * 2), q.resize(k * 2);
    dft(p), dft(q);
    dotEq(p, q);
    idft(p);
    for (int i = m; i < r; i++) inc(b[i], p[i - l - 1]);
    b[m] = 1LL * b[m] * inv[m] % P;
  }
  return b;
}
// half-online conv
// $O(\frac{n\log^2n}{\log\log n})$
// $nb_n = \sum_{i=0}^{n-1} a'_ib_{n - 1 - i}$
Poly exp(Poly a) {
  if (a.empty()) return {};
  assert(a[0] == 0);
  int n = a.size();
  updateInv(n);
  for (int i = 0; i + 1 < n; i++) {
    a[i] = a[i + 1] * (i + 1LL) % P;
  }
  a.pop_back();
  Poly b(n);
  b[0] = 1;
  std::vector<Poly> val_a[6], val_b(n);
  for (int m = 1; m < n; m++) {
    int k = 1, d = 0;
    while (!(m / k & 0xf)) k *= 16, d++;
    int l = m & ~(0xf * k), r = std::min(n, m + k);
    if (k == 1) {
      for (int i = m; i < r; i++) {
        for (int j = l; j < m; j++) {
          b[i] = (b[i] + 1LL * b[j] * a[i - j - 1]) % P;
        }
      }
    } else {
      assert(d < 6);
      if (val_a[d].empty()) val_a[d].resize(n);
```

```
316       val_b[m] = Poly(b.begin() + (m - k), b.begin() + m);
317       val_b[m].resize(k * 2);
318       dft(val_b[m]);
319       Poly res(k * 2);
320       for (; l < m; l += k) {
321         auto &p = val_a[d][m - l - k];
322         if (p.empty()) {
323           p = Poly(a.begin() + (m - l - k), a.begin() + (r - l - 1));
324           p.resize(2 * k);
325           dft(p);
326         }
327         auto &q = val_b[l + k];
328         for (int i = 0; i < k * 2; i++) res[i] = (res[i] + 1LL * p[i] * q[i]) % P;
329       }
330       idft(res);
331       for (int i = m; i < r; i++) inc(b[i], res[i - m + k - 1]);
332     }
333     b[m] = 1LL * b[m] * inv[m] % P;
334   }
335   return b;
336 }
337 Poly power(Poly a, int k) {
338   int n = a.size();
339   long long d = 0;
340   while (d < n && !a[d]) d++;
341   if (d == n) return a;
342   a >>= d;
343   int b = fpow(a[0]);
344   norm(a *= b);
345   a = exp(ln(a) * k) * fpow(b, P - 1 - k % (P - 1));
346   a.resize(n);
347   d *= k;
348   for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
349   d = std::min(d, 1LL * n);
350   for (int i = d; i; a[--i] = 0) ;
351   return a;
352 }
353 Poly power(Poly a, int k1, int k2) { // k1 = k % (P - 1), k2 = k % P
354   int n = a.size();
355   long long d = 0;
356   while (d < n && !a[d]) d++;
357   if (d == n) return a;
358   a >>= d;
359   int b = fpow(a[0]);
360   norm(a *= b);
361   a = exp(ln(a) * k2) * fpow(b, P - 1 - k1 % (P - 1));
362   a.resize(n);
363   d *= k1;
364   for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
365   d = std::min(d, 1LL * n);
366   for (int i = d; i; a[--i] = 0) ;
367   return a;
368 }
```

```
// [x^n](f / g)
// $O(m \log m \log n)$
int divAt(Poly f, Poly g, int64_t n) {
  assert(f.size() == g.size());
  int len = f.size(), m = 1 << std::__lg(len * 2 - 1);
  for (; n; n >>= 1) {
    f.resize(m * 2), g.resize(m * 2);
    dft(f), dft(g);
    for (int i = 0; i < m * 2; i++) f[i] = 1LL * f[i] * g[i ^ 1] % P;
    for (int i = 0; i < m; i++) g[i] = 1LL * g[i * 2] * g[i * 2 + 1] % P;
    g.resize(m);
    idft(f), idft(g);
    for (int i = 0, j = n & 1; i < len; i++, j += 2) f[i] = f[j];
    f.resize(len), g.resize(len);
  }
  return f[0];
}
} // namespace Polynomial
```

3 基础数据结构

3.1 逆序对

```
//逆序对：序列a[]中满足i<j且a_i>a_j的有序对。
/*
1.离散化 + 树状数组
*/
#include<iostream>
#include<algorithm>
using namespace std;

#define ll long long
#define pi pair<ll,int>
const int N=5e5+10;
ll n,cnt,tr[N];
pi a[N];

void add(int x,int c)
{
    for(int i=x;i<=n;i+=i&-i) tr[i]+=c;
}

int sum(int x)
{
    int ret=0;
    for(int i=x;i>0;i-=i&-i) ret+=tr[i];
    return ret;
}

int main()
{
    while(cin>>n)
```

```
    {
        if(!n) break;
        cnt=0;
        for(int i=1;i<=n;i++)
            cin>>a[i].first,a[i].second=i;
        sort(a+1,a+1+n);
        for(int i=1;i<=n;i++)
        {
            //从小到大遍历a[]，按其在原数组中的位置(下标)插入a[i]
            add(a[i].second,1);
            //逆序对个数 += 当前插入数的个数 - 在a[i]插入位置之前插入的数的个数
            //即原数组中在i之前比a[i]大的数的个数
            cnt+=i-sum(a[i].second);
        }
        cout<<cnt<<"\n";
        for(int i=1;i<=n;i++) tr[i]=0;
    }

    return 0;
}

/*
2.归并排序求逆序对
*/

#include<iostream>
using namespace std;

#define ll long long
const int N=5e5+10;
ll n,a[N],t[N];

ll msort(ll q[],int l,int r)
{
    if(l>=r) return 0;

    int mid=l+r>>1;
    ll res = msort(q,l,mid) + msort(q,mid+1,r);

    int k=l,i=l,j=mid+1;
    while(i<=mid && j<=r)
        if(q[i]<=q[j]) t[k++] = q[i++];
        else if(q[i]>q[j])
            t[k++]=q[j++],res+=mid-i+1;
        //关键代码：res += mid-i+1
        //[i,mid]区间内的数均与q[j]构成逆序对

    while(i<=mid) t[k++]=q[i++];
    while(j<=r) t[k++]=q[j++];

    for(i=l;i<=r;i++) q[i]=t[i];
    return res;
}
```

```
83
84  int main()
85  {
86      while(cin>>n,n)
87      {
88          for(int i=1;i<=n;i++) cin>>a[i];
89          cout<<msort(a,1,n)<<"\n";
90      }
91      return 0;
92  }
```

3.2 并查集

```
1   /*
2   并查集适用于维护含有传递关系的集合，根据使用姿势的不同，常见的有【边带权】，【扩展域】（拆点）两
        种。
3   */
4
5   //基础
6   const int N=1e6+7;
7   int fa[N];
8
9   void makeSet()
10  {
11      for(int i=1;i<=n;i++) fa[i] = i;
12  }
13  int find(int x)
14  {
15      return fa[x]==x? x:fa[x]=find(fa[x]);
16  }
17  void merge(int x,int y)
18  {
19      x=find(x), y=find(y);
20      if(x!=y) fa[x]=y;
21  }
22
23  //AcWing 238. 银河英雄传说 - 边带权并查集
24  /*
25  有一个划分为 N 列的星际战场，各列依次编号为 1,2,…,N。
26  有 N 艘战舰，也依次编号为 1,2,…,N，其中第 i 号战舰处于第 i 列。
27  有 T 条指令，每条指令格式为以下两种之一：
28  1. `M i j`，表示让第 i 号战舰所在列的全部战舰保持原有顺序，接在第 j 号战舰所在列的尾部。
29  2. `C i j`，表示询问第 i 号战舰与第 j 号战舰当前是否处于同一列中，如果在同一列中，它们之间间隔了
        多少艘战舰。
30  现在需要你编写一个程序，处理一系列的指令。
31
32  SOLUTION
33  带边权的并查集
34  `d[i]：i到其祖先节点的距离
35  siz[i]：祖先节点为i的集合的大小`
36  `find(x)：dfs，先更新父节点的d[fa[x]]，再用父节点的距离更新自己的d[x]；
37  merge(x,y)：考虑对各自的影响——x集合中d[x]距离应更新为siz[y]；y集合中siz[y]需要加上siz[x]。`
```

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

const int N=3e4+100;
int t;
int fa[N],siz[N],d[N];

int find(int x)
{
    if(fa[x]==x) return x;
    int rt = find(fa[x]);
    d[x] += d[fa[x]];
    return fa[x]=rt;
}

void merge(int x,int y)
{
    x=find(x),y=find(y);
    if(x==y) return ;
    d[x]=siz[y];
    siz[y]+=siz[x];
    fa[x]=y;
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    cin>>t;
    for(int i=1;i<N;i++) siz[i]=1,fa[i]=i;
    while(t--)
    {
        char op;
        int i,j;
        cin>>op>>i>>j;
        if(op=='M')
        {
            merge(i,j);
        }else
        {
            if(i==j)
                cout<<0<<"\n";
            else if(find(i)==find(j))
                cout<<abs(d[i]-d[j])-1<<"\n";
            else
                cout<<-1<<"\n";
        }
    }
    return 0;
}

//AcWing 239. 奇偶游戏 - 【边带权/扩展域】
/*
```

```
小 A 和小 B 在玩一个游戏。
首先，小 A 写了一个由 0 和 1 组成的序列 S，长度为 N。
然后，小 B 向小 A 提出了 M 个问题。
在每个问题中，小 B 指定两个数 l 和 r，小 A 回答 S[l⊕r] 中有奇数个 1 还是偶数个 1。
机智的小 B 发现小 A 有可能在撒谎。
例如，小 A 曾经回答过 S[1⊕3] 中有奇数个 1，S[4⊕6] 中有偶数个 1，现在又回答 S[1⊕6] 中有偶数个 1，显然这是自相矛盾的。
请你帮助小 B 检查这 M 个答案，并指出在至少多少个回答之后可以确定小 A 一定在撒谎。
即求出一个最小的 k，使得 01 序列 S 满足第 1⊕k 个回答，但不满足第 1⊕k+1 个回答。
*/
//AC代码
/*
SOLUTION-1
边带权并查集 - 使用异或运算维护奇偶性

1.s[l~r]有偶数个1,等价于sum[l-1]与sum[r]奇偶性相同
2.s[l~r]有奇数个1,等价于sum[l-1]与sum[r]奇偶性不同

d[x]=0表示x与fa[x]奇偶性相同; d[x]=1表示x与fa[x]奇偶性不同.
使用异或运算来更新d[x].

find(x): dfs,d[x]为x与其祖先节点路径上的异或和.
merge(x,y):
   fa[rx] = ry;
   x~y的路径 = x~rx + rx~ry + ry~y，即 flag = d[x]^d[rx]^d[y]，故有 d[rx] = d[x]^d[y]^flag.

SOLUTION-2
扩展域并查集 - 将每个节点X分解成奇数点oddX和偶数点evenX.
对于每次回答，根据答案，检查矛盾，并对oddX,evenX,oddY,evenY进行合并:
1.ans=0：合并oddX和oddY，evenX和evenY;
2.ans=1：合并oddX和evenY，evenX和oddY。
*/

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

const int N=1e4+100;
int n,m,k,idx;
int fa[N<<1],d[N<<1];

//解法1：边带权 - 异或运算更新边权
int l,r;
string s;
unordered_map<int,int> hs; //离散化

int find1(int x)
{
   if(fa[x]==x) return x;
   int rt=find1(fa[x]);
   d[x]^=d[fa[x]];
   return fa[x]=rt;
}
```

```
142
143 void solve1()
144 {
145     cin>>n>>m;
146     for(int i=1;i<=m;i++)
147     {
148         cin>>l>>r>>s;
149         int ans = s[0]=='e'?0:1;
150
151         if(hs[l-1]==0) hs[l-1]=++idx,fa[idx]=idx;
152         if(hs[r]==0) hs[r]=++idx,fa[idx]=idx;
153         int x=hs[l-1],y=hs[r];
154         int rx=find1(x),ry=find1(y);
155
156         if(rx==ry && (d[y]^d[x])!=ans) break;
157         if(rx!=ry)
158         {
159             fa[rx]=ry;
160             d[rx]=d[x]^d[y]^ans; //合并区间
161         }
162         k++;
163     }
164     cout<<k<<"\n";
165 }
166
167 //解法2: 扩展域 - 将每个节点X分解成奇数点oddX和偶数点evenX.
168 int L[N],R[N],ans[N],a[N<<1];
169
170 void pre_work() //离散化
171 {
172     cin>>n>>m;
173     for(int i=1;i<=m;i++)
174     {
175         cin>>L[i]>>R[i]>>s;
176         ans[i] = s[0]=='e'?0:1;
177         a[++idx]=L[i]-1;
178         a[++idx]=R[i];
179     }
180     sort(a+1,a+1+idx);
181     n = unique(a+1,a+1+idx) - (a+1);
182 }
183
184 int find(int x)
185 {
186     return fa[x]==x ? x:fa[x]=find(fa[x]);
187 }
188
189 void solve2()
190 {
191     pre_work();
192     for(int i=1;i<=2*n;i++) fa[i]=i;
193     for(int i=1;i<=m;i++)
194     {
```

```
195         int x=lower_bound(a+1,a+1+n,L[i]-1) - a;
196         int y=lower_bound(a+1,a+1+n,R[i]) - a;
197         int x_odd=x, x_even=x+n;
198         int y_odd=y, y_even=y+n;
199 
200         if(ans[i]==0)
201         {
202             if(find(x_odd)==find(y_even)) break;
203             fa[find(x_odd)] = find(y_odd);
204             fa[find(x_even)] = find(y_even);
205         }else
206         {
207             if(find(x_odd)==find(y_odd)) break;
208             fa[find(x_odd)] = find(y_even);
209             fa[find(x_even)] = find(y_odd);
210         }
211         k++;
212     }
213     cout<<k<<"\n";
214 }
215 
216 int main()
217 {
218     ios::sync_with_stdio(0);cin.tie(0);
219     solve2();
220     return 0;
221 }
222 
223 //AcWing 240. 食物链 - 【边带权/扩展域】
224 /*
225 动物王国中有三类动物A,B,C，这三类动物的食物链构成了有趣的环形。
226 A吃B， B吃C，C吃A。
227 现有N个动物，以1 - N编号。
228 每个动物都是A,B,C中的一种，但是我们并不知道它到底是哪一种。
229 有人用两种说法对这N个动物所构成的食物链关系进行描述：
230 第一种说法是”1 X Y”，表示X和Y是同类。
231 第二种说法是”2 X Y”，表示X吃Y。
232 此人对N个动物，用上述两种说法，一句接一句地说出K句话，这K句话有的是真的，有的是假的。
233 当一句话满足下列三条之一时，这句话就是假话，否则就是真话。
234 1） 当前的话与前面的某些真的话冲突，就是假话；
235 2） 当前的话中X或Y比N大，就是假话；
236 3） 当前的话表示X吃X，就是假话。
237 你的任务是根据给定的N和K句话，输出假话的总数。
238 
239 SOLUTION-1
240 并查集 - 扩展域
241 对于每个节点X，扩展3个域：天敌，同类，捕食.
242 
243 
244 SOLUTION-2
245 并查集 - 边带权
246 将关系转化为向量运算理解：
247 
```

```
0-同类；1-吃；2-被吃；
给定关系为 rel = D-1 = (a,b);

d[x] = 向量(x,rx)
1.rx=ry: (a,b) = (a,rx) - (b,ry)
       => rel = d[a] - d[b].
2.rx!=ry: 此时肯定符合要求，需合并集合——fa[rx]=ry, (rx,ry) = (a,b) - (a,rx) + (b,ry).
       => fa[rx]=ry, d[rx] = rel - d[a] + d[b].

在数值运算时，需要对运算结果对3取模.

*/

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

const int N=2e5+100;
int n,k,ans;
int D,x,y;
int fa[N];

//*********解法1-扩展域
int find(int x)
{
   return fa[x]==x ? x:fa[x]=find(fa[x]);
}

void solve1()
{
   cin>>n>>k;
   for(int i=1;i<=n*3;i++) fa[i]=i;
   for(int i=1;i<=k;i++)
   {
      cin>>D>>x>>y;
      if(x>n||y>n)
      {
         ans++;
         continue;
      }
      //天敌，同类，捕食
      int x1=x,x2=x+n,x3=x+2*n;
      int y1=y,y2=y+n,y3=y+2*n;

      if(D==1)
      {
         //矛盾：x的天敌是y的同类 / x的捕食是y的同类
         if(find(x1)==find(y2) || find(x3)==find(y2))
         {
            ans++;
            continue;
         }
         fa[find(x2)]=find(y2); //x和y同类继承
         fa[find(x1)]=find(y1);
```

```
301             fa[find(x3)]=find(y3);
302         }
303         else if(D==2)
304         {
305             //矛盾：x的同类是y的同类 / x的天敌是y的同类
306             if(find(x2)==find(y2) || find(x1)==find(y2))
307             {
308                 ans++;
309                 continue;
310             }
311             fa[find(x2)]=find(y1); //x的同类是y的天敌
312             fa[find(x1)]=find(y3); //x的天敌是y的捕食
313             fa[find(x3)]=find(y2); //x的捕食是y的同类
314         }
315     }
316     cout<<ans<<"\n";
317 }
318
319 //*********解法2-边带权
320 int d[N];
321
322 int find2(int x)
323 {
324     if(fa[x]==x) return x;
325     int rt=find2(fa[x]);
326     d[x]+=d[fa[x]];
327     return fa[x]=rt;
328 }
329
330 void solve2()
331 {
332     cin>>n>>k;
333     for(int i=1;i<=n;i++) fa[i]=i;
334     while(k--)
335     {
336         cin>>D>>x>>y;
337         if(x>n || y>n)
338         {
339             ans++;
340             continue;
341         }
342         int rx=find2(x),ry=find2(y);
343         int rel=D-1;
344
345         if(rx==ry)
346         {
347             //若(d[x]-d[y])<0，模3后依旧是负数，故需要再+3将其变为正数；即:(d[x]-d[y])%3+3
348             //若(d[x]-d[y])>0，(d[x]-d[y])%3+3后结果会大于3，故还要再模3，即:((d[x]-d[y])%3+3)
                    %3
349             if( rel != ((d[x]-d[y])%3+3)%3 )
350                 ans++;
351         }else
352         {
```

```
            fa[rx]=ry;
            d[rx] = rel - d[x] + d[y];
        }
    }
    cout<<ans<<"\n";
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    solve2();
    return 0;
}
```

3.3 单调栈-最大子矩阵与子矩阵个数

```
//前置 - AcWing 131. 直方图中最大的矩形
//AC代码
 /*
对于每个矩形
其所能形成的最大矩形为：（向左扩展的最大长度+向右扩展的最大长度）*矩形的高度
*/
#include<iostream>
#include<algorithm>
#include<stack>
using namespace std;

#define ll long long
const int N=1e5+10;
int n;
ll h[N],l[N],r[N]; //l[]存放第i个矩形向左扩展的最远下标；r[]存放第i个矩形向右扩展的最远下标；

void get(ll a[N]) //单调递增栈
{
    stack<int> stk;
    for(int i=0;i<=n;i++)
    {
        while(stk.size() && h[stk.top()]>=h[i]) stk.pop();
        if(stk.size()) a[i]=stk.top();
        stk.push(i);
    }
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);
    while(cin>>n,n)
    {
        h[0]=-1;
        for(int i=1;i<=n;i++) cin>>h[i];

        get(l);
```

```
37          reverse(h+1,h+1+n);
38          get(r);
39
40          ll ans=0;
41          for(int i=1,j=n;i<=n;i++,j--)
42              //坐标变换
43              ans=max(ans,((n-r[j]) - l[i])*h[j]);
44
45          cout<<ans<<"\n";
46      }
47
48      return 0;
49  }
50
51  //给定一个N*M的矩阵，其中有一些障碍点，求不含障碍点最大子矩阵。
52  //AC代码
53  /*
54  O(n^2)
55  【直方图中最大的矩形】 - 扩展
56  枚举每一行，以每一行为基准面做【直方图中最大的矩形】计算，最后取最大值即为答案。
57  */
58
59  #include<bits/stdc++.h>
60  using namespace std;
61
62  #define ll long long
63  const int N=1e3+10;
64  ll ans=0;
65  int n,m;
66  int f[N],l[N],r[N]; //l[]向左扩展的最大长度；r[]向右扩展的最大长度
67
68  void get(int *ar)
69  {
70      stack<int> stk;
71      for(int i=1;i<=m;i++) //单调递增栈
72      {
73          while(stk.size() && f[i]<=f[stk.top()]) stk.pop();
74          if(stk.size()) ar[i]=i-stk.top()-1;
75          else ar[i]=i-1;
76          stk.push(i);
77      }
78  }
79
80  int main()
81  {
82      ios::sync_with_stdio(0);cin.tie(0);
83      cin>>n>>m;
84      char ch;
85      for(int i=1;i<=n;i++)
86      {
87          for(int j=1;j<=m;j++)
88              (cin>>ch,ch) =='F' ? f[j]++:f[j]=0;
89
```

```
90          get(l); reverse(f+1,f+1+m);
91          get(r); reverse(f+1,f+1+m);
92
93          for(int j=1;j<=m;j++)
94              ans=max(ans,(ll)f[j]*(r[m-j+1]+l[j]+1));
95      }
96      cout<<(3*ans)<<"\n";
97
98      return 0;
99  }
100
101
102 //给定一个N*M的矩阵，其中有一些障碍点，求不含障碍点的子矩阵的个数。
103 /*
104 对每一个直方块的高度维护一个单调递增栈。
105 对于每个矩形，
106 以该矩形最下端的单位矩形为右下角的子矩形 个数为：
107
108 以往左的第一个不能扩展的矩形为右下角的子矩形个数 + （向左扩展的最大长度）* 矩形的高度.
109 */
110 #include<bits/stdc++.h>
111 using namespace std;
112
113 #define ll long long
114 const int N=5e3+10,M=1e6+10;
115
116 int t;
117 int n,m,k;
118 ll mp[N][N],h[N],sum[N];
119
120 int main()
121 {
122     ios::sync_with_stdio(0);cin.tie(0);
123     cin>>n>>m;
124     for(int i=1,a,b;i<=m;i++)
125         cin>>a>>b,mp[a][b]=1;
126
127     ll ans=0;
128     for(int i=1;i<=n;i++)
129     {
130         stack<int> stk;
131         for(int j=1;j<=n;j++)
132         {
133             sum[j]=0;
134             if(mp[i][j]) h[j]=0;
135             else h[j]++;
136
137             while(stk.size() && h[stk.top()]>=h[j]) stk.pop();
138             if(stk.size()) sum[j] += sum[stk.top()] + (j-stk.top()) * h[j];
139             else sum[j] += j*h[j];
140             ans += sum[j];
141             stk.push(j);
142         }
```

```
    }
    cout<<ans<<"\n";

    return 0;
}
```

3.4 莫队分块

```
/**
 * 【莫队分块】
 * https://ac.nowcoder.com/acm/contest/21592/G
*/
#include<bits/stdc++.h>
using namespace std;

#define ll long long
const int N=1e6+100;
int t,n,q,M;
int a[N],ans[N];

int pr[N],prc;
bool ispr[N];

vector<int> fs[N];
int cnt[N],num[N],mx;
//cnt[i]: 记录因子i出现次数
//num[ct]: 记录出现ct次的因子的个数

struct qry
{
    int l,r,id;
    bool operator<(const qry& x) const
    {
        if(l/M!=x.l/M) return l<x.l;
        return (l/M)&1 ? r<x.r: r>x.r;
    }
};
qry qs[N];

//欧式线性筛
void eluer()
{
    for (int i = 2; i < N; i++)
    {
        if(!ispr[i]) pr[prc++]=i;
        for (int j = 0; j < prc && 1ll*pr[j]*i<=N; j++)
        {
            ispr[pr[j]*i] = 1;
            if(i%pr[j]==0) break;
        }
    }
}
```

```cpp

//质因数分解
void div(int m)
{
    int t=m;
    for(int i=0;pr[i]<=t/pr[i];i++)
    {
        int p=pr[i];
        if(t%p==0)
        {
            while(t%p==0) t/=p;
            fs[m].push_back(p);
        }
    }
    if(t>1) fs[m].push_back(t);
}


void add(int m)
{
    if(!fs[m].size()) div(m);
    //calc
    for(auto &v:fs[m])
    {
        cnt[v]++;
        if((++num[cnt[v]])==1) mx++;
    }
}

void sub(int m)
{
    if(!fs[m].size()) div(m);
    //calc
    for(auto &v:fs[m])
    {
        if((--num[cnt[v]])==0) mx--;
        cnt[v]--;
    }
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    eluer();
    cin>>t;
    while(t--)
    {
        cin>>n>>q;
        for(int i=1;i<=n;i++) cin>>a[i];
        for(int i=1,l,r;i<=q;i++) cin>>l>>r,qs[i]={l,r,i};
        M=sqrt(n);
        sort(qs+1,qs+1+q);
        mx=0;
```

```
98        int i,pl,pr;
99        for( i=1,pl=1,pr=0;i<=q;i++)
100       {
101          int l=qs[i].l,r=qs[i].r,id=qs[i].id;
102          while(pl>l) add(a[--pl]); //add
103          while(pr<r) add(a[++pr]); //add
104          while(pl<l) sub(a[pl++]);
105          while(pr>r) sub(a[pr--]);
106          ans[id]=mx;
107       }
108       while(pl<=pr) sub(a[pl++]);
109       for(int i=1;i<=q;i++) cout<<ans[i]<<"\n";
110    }
111
112    return 0;
113 }
```

3.5 基础树套树

```
1  /**
2   * Icpc19南京F.Paper Grading
3   * Trie树dfs序 + 树状数组套动态开点线段树
4   * 树套树模型问题：
5   * 单点修改，求[l,r]内数值在[L,R]之间的数有多少个。
6  */
7  #include<bits/stdc++.h>
8  using namespace std;
9
10 #define ll long long
11 const int N=2e5+100,M=N*100;
12 ll n,m;
13 string s;
14
15 int tr[N][26],idx;
16 int vid[N];
17 int rid[N];
18 int rt[N];
19
20 //dfs_xu
21 int dfn[N],id[N],tot,sz[N];
22 void dfs(int u)
23 {
24    dfn[++tot]=u;
25    id[u]=tot;
26    sz[u]=1; //start[u]=id[u],end[u]=id[u]+sz[u]-1
27    for(int i=0;i<26;i++)
28    {
29       if(tr[u][i])
30       {
31          dfs(tr[u][i]);
32          sz[u]+=sz[tr[u][i]];
33       }
```

```
    }
}

//SGT
struct SGT
{
    #define mid (l+r>>1)
    int val[M],lc[M],rc[M];
    int idx;
    void upd(int &x,int l,int r,int pos,int c)
    {
        if(!x) x=++idx;
        val[x]+=c;
        if(l==r) return ;
        if(pos<=mid) upd(lc[x],l,mid,pos,c);
        else upd(rc[x],mid+1,r,pos,c);
    }
    int qry(int x,int l,int r,int L,int R)
    {
        if(!x) return 0;
        if(L<=l && r<=R) return val[x];
        int ret=0;
        if(L<=mid) ret+=qry(lc[x],l,mid,L,R);
        if(R>mid) ret+=qry(rc[x],mid+1,r,L,R);
        return ret;
    }
};
SGT sgt;

//LOW_TREE
void add(int x,int pos,int c)
{
    for(int i=x;i<=n;i+=i&-i) sgt.upd(rt[i],1,tot,pos,c);
}
int sum(int x,int L,int R)
{
    int ret=0;
    for(int i=x;i>0;i-=i&-i) ret+=sgt.qry(rt[i],1,tot,L,R);
    return ret;
}

int root;

//TRIE
int ins(string &s)
{
    int p=root;
    for(auto ch:s)
    {
        int &v=tr[p][ch-'a'];
        if(!v) v=++idx;
        p=v;
    }
```

```
87      return p;
88  }
89
90  void qry(string &s,int k,int l,int r)
91  {
92      int p=root;
93      //注意k=0的情况
94      for(int i=0;i<k;i++)
95      {
96          int v=tr[p][s[i]-'a'];
97          if(!v)
98          {
99              p=0;break;
100         }
101         p=v;
102     }
103     int ans=0;
104     if(p==0) ans=0;
105     else
106     {
107         int L=id[p],R=id[p]+sz[p]-1;
108         ans = sum(r,L,R)-sum(l-1,L,R);
109     }
110     cout<<ans<<"\n";
111 }
112
113 int main()
114 {
115     ios::sync_with_stdio(0);cin.tie(0);
116     cin>>n>>m;
117     root=++idx;
118     for(int i=1;i<=n;i++) cin>>s,rid[i]=ins(s);
119     dfs(root);
120     for(int i=1;i<=n;i++) add(i,id[rid[i]],1);
121
122     int op,k,l,r;
123     for(int i=1;i<=m;i++)
124     {
125         cin>>op;
126         if(op==1)
127         {
128             cin>>l>>r;
129             add(l,id[rid[l]],-1);
130             add(l,id[rid[r]],1);
131             add(r,id[rid[r]],-1);
132             add(r,id[rid[l]],1);
133             swap(rid[l],rid[r]);
134         }else
135         {
136             cin>>s>>k>>l>>r;
137             qry(s,k,l,r);
138         }
139     }
```

```

    return 0;
}
```

4 数学

4.1 高斯消元

```
/*
高斯消元法模板题
*/

#include<iostream>
#include<cmath>
#include<cstdio>
using namespace std;

const int N=110;
const double eps=1e-9;
int n;
double a[N][N],x[N];

// a[N][N]是增广矩阵
int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c ++ )
    {
        int t = r;
        for (int i = r; i < n; i ++ ) // 找到绝对值最大的行
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if (fabs(a[t][c]) < eps) continue;

        for (int i = c; i <= n; i ++ ) swap(a[t][i], a[r][i]); // 将绝对值最大的行换到最顶端
        for (int i = n; i >= c; i -- ) a[r][i] /= a[r][c]; // 将当前行的首位变成1
        for (int i = r + 1; i < n; i ++ ) // 用当前行将下面所有的列消成0
            if (fabs(a[i][c]) > eps)
                for (int j = n; j >= c; j -- )
                    a[i][j] -= a[r][j] * a[i][c];

        r ++ ;
    }

    if (r < n)
    {
        for (int i = r; i < n; i ++ )
            if (fabs(a[i][n]) > eps)
                return 2; // 无解
        return 1; // 有无穷多组解
    }
```

```
45
46      for (int i = n - 1; i >= 0; i -- )
47          for (int j = i + 1; j < n; j ++ )
48              a[i][n] -= a[i][j] * a[j][n];
49
50      return 0; // 有唯一解
51  }
52
53  int main()
54  {
55      cin>>n;
56      for (int i = 0; i < n; i++)
57          for (int j = 0; j <= n; j++)
58              cin>>a[i][j];
59      if(gauss())
60          printf("No Solution\n");
61      else
62      {
63          for(int i=0;i<n;i++)
64              printf("%.2lf\n",a[i][n]);
65      }
66
67      return 0;
68  }
69
70
71  #include<bits/stdc++.h>
72  using namespace std;
73
74  const int N=410,mod=1e9+7;
75
76  int ksm(int a,int b)
77  {
78      int ret=1;
79      while(b)
80      {
81          if(b&1) ret=1ll*ret*a%mod;
82          a=1ll*a*a%mod;
83          b>>=1;
84      }
85      return ret;
86  }
87
88  int n;
89  int a[N][N*2];
90
91  int gaussRevMod()
92  {
93      for(int i=0;i<n;i++)
94      {
95          // print();
96          for(int j=i;j<n;j++)
97              if(a[j][i])
```

```
98            {
99                swap(a[i],a[j]);
100               break;
101           }
102
103       if(!a[i][i]) return 1;//无解
104       int inv=ksm(a[i][i],mod-2); //逆元
105
106       for(int j=2*n-1;j>=i;j--) a[i][j]=1ll*a[i][j]*inv%mod;
107       for(int j=0;j<n;j++)
108           if(j!=i && a[j][i])
109               for(int k=2*n-1;k>=i;k--)
110                   a[j][k] = (a[j][k] - 1ll*a[j][i]*a[i][k]%mod + mod)%mod;
111   }
112   return 0;
113 }
114
115 int main()
116 {
117   ios::sync_with_stdio(0);cin.tie(0);
118   cin>>n;
119   for(int i=0;i<n;i++)
120       for(int j=0;j<n;j++)
121           cin>>a[i][j];
122
123   for(int i=0;i<n;i++) a[i][i+n]=1;
124
125   if(gaussRevMod())
126       cout<<"No Solution";
127   else
128       for(int i=0;i<n;i++)
129           for(int j=0;j<n;j++)
130               cout<<a[i][j+n]<<(j==n-1?"\n":" ");
131
132   return 0;
133 }
```

4.2 线性基

```
1  //ACWing210. 异或运算 - 【线性基】
2  #include<bits/stdc++.h>
3  using namespace std;
4
5  #define ll long long
6  #define pi pair<ll,ll>
7  const int N=100;
8  ll t,n,k,m,cnt;
9  string s;
10 ll p[N];
11 bool flag;
12
13 //初始化线性基，也能求x是否在当前基的值域中，插入失败则在值域中
```

```cpp
bool insert(ll x)
{
    for(int i=60;i>=0;i--)
    {
        if(!(x>>i)) continue;
        if(!p[i]) {p[i]=x; return 1;}
        x^=p[i];
    }
    return 0;
}

//线性基独立化
void build()
{
    for(int i=0;i<=60;i++)
        for(int j=i+1;j<=60;j++)
            if(p[j]>>i&1) p[j]^=p[i];

    cnt=0;
    for(int i=0;i<=60;i++) if(p[i]) p[cnt++]=p[i];
}

//==补充==
ll qrymax()
{
    ll ret=0;
    for(int i=60;i>=0;i--) ret=max(ret,ret^p[i]);
    return ret;
}

//==补充==
ll qrymin()
{
    // 注意是否包含0
    if(flag) return 0;
    for(int i=0;i<=60;i++) if(p[i]) return p[i];
}

ll qrykth(ll k)
{
    // build() before use
    if(flag) k--;
    if(k>=(ll)1<<cnt) return -1;
    ll ret=0;
    for(int i=0;i<=60;i++) if(k>>i&1) ret^=p[i];
    return ret;
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>t;
    for(int ct=1;ct<=t;ct++)
```

```
67      {
68          cin>>n;
69          memset(p,0,sizeof(p));
70          flag=0;
71          for(int i=1;i<=n;i++)
72          {
73              ll x; cin>>x;
74              //插入失败说明当前数已在值域中，说明异或最小值为0
75              if(!insert(x)) flag=1;
76          }
77          build();
78
79          cout<<"Case #"<<ct<<":\n";
80          cin>>m;
81          while(m--)
82              cin>>k, cout<<qrykth(k)<<"\n";
83      }
84
85      return 0;
86  }
87
88  //====前缀线性基====
89  //【异或线性基 - 前缀线性基】
90  //多次询问，区间查询X是否在该区间内的数的异或值域中
91  #include<bits/stdc++.h>
92  using namespace std;
93
94  #define ll long long
95  const int N=4e5+100;
96  ll t,n,Q;
97  ll p[N][70],pos[N][70];
98
99  //初始化前缀线性基
100 bool insert(ll x,ll r)
101 {
102     ll cur=r;
103     memcpy(p[cur],p[cur-1],sizeof(p[cur]));
104     memcpy(pos[cur],pos[cur-1],sizeof(pos[cur]));
105     for(int i=60;i>=0;i--)
106     {
107         if(!(x>>i&1)) continue;
108         if(!p[r][i])
109         {
110             p[r][i]=x;
111             pos[r][i]=cur;
112             return 1;
113         }
114         if(cur>pos[r][i])
115         {
116             swap(p[r][i],x);
117             swap(pos[r][i],cur);
118         }
119         x^=p[r][i];
```

```
    }
    return 0;
}

bool qry(ll l,ll r,ll x)
{
    for(int i=60;i>=0;i--)
    {
        if(!(x>>i&1)) continue;
        if(pos[r][i]<l) continue;
        if(!p[r][i])
            return 1;
        x^=p[r][i];
    }
    return x==0;
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>n>>Q;
    for(ll i=1,x;i<=n;i++) cin>>x,insert(x,i);
    while(Q--)
    {
        ll l,r,x;
        cin>>l>>r>>x;
        cout<<(qry(l,r,x)?"Yes":"No")<<"\n";
    }

    return 0;
}

//====树上前缀线性基====
/**
 * P3292 [SCOI2016]幸运数字
 * https://www.luogu.com.cn/problem/P3292
 * 【树上-前缀线性基】
 * 题意:
 * 给定一颗树，多次询问，每次询问给点两个点x和y，
 * 求x到y的路径中所有节点的数字集合中的子集异或极值。
 *
 * SOLUTION:
 * 树上LCA，按深度记录前缀线性基;
 * 每次询问，求出x到lca(x,y)路径上的线性基和y到lca(x,y)路径上的线性基;
 * 然后进行线性基合并，查询极值即可。
*/
#include<bits/stdc++.h>
using namespace std;

#define ll long long
const int N=2e4+100;
ll t,n,m;
ll a[N];
```

```cpp
vector<int> e[N];
ll p[N][70],pos[N][70];
ll dep[N],fa[N][22];
ll vp[70];

bool insert(ll x,int r,int ff)
{
    ll cur=r;
    for(int i=0;i<=60;i++)
        p[cur][i]=p[ff][i], pos[cur][i]=pos[ff][i];
    for(int i=60;i>=0;i--)
    {
        if(!(x>>i)) continue;
        if(!p[r][i])
        {
            p[r][i]=x;
            pos[r][i]=cur;
            return 1;
        }
        if(dep[cur] > dep[pos[r][i]])
        {
            swap(p[r][i],x);
            swap(pos[r][i],cur);
        }
        x^=p[r][i];
    }
    return 0;
}

bool insert(ll x)
{
    for(int i=60;i>=0;i--)
    {
        if(!(x>>i)) continue;
        if(!vp[i])
        {
            vp[i]=x;
            return 1;
        }
        x^=vp[i];
    }
    return 0;
}

void dfs(int u,int ff)
{
    dep[u]=dep[ff]+1;
    insert(a[u],u,ff);
    fa[u][0]=ff;
    for(int i=1;i<=20;i++) fa[u][i]=fa[fa[u][i-1]][i-1];
    for(auto v:e[u]) if(v!=ff) dfs(v,u);
}

```

```cpp
int lca(int x,int y)
{
    if(dep[x]<dep[y]) swap(x,y);
    for(int i=20;i>=0;i--) if(dep[fa[x][i]]>=dep[y]) x=fa[x][i];
    for(int i=20;i>=0;i--) if(fa[x][i]!=fa[y][i]) x=fa[x][i],y=fa[y][i];
    return x!=y ? fa[x][0]:x;
}

void solve(int x,int y)
{
    int anc = lca(x,y);
    ll mx=0;
    for(int i=60;i>=0;i--) if(dep[pos[x][i]]>=dep[anc]) vp[i]=p[x][i]; else vp[i]=0;
    //直接插入合并线性基
    for(int i=60;i>=0;i--) if(dep[pos[y][i]]>=dep[anc]) insert(p[y][i]);
    for(int i=60;i>=0;i--) mx=max(mx,mx^vp[i]);

    cout<<mx<<"\n";
}

int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>a[i];
    for(int i=1,u,v;i<=n-1;i++)
    {
        cin>>u>>v;
        e[u].push_back(v), e[v].push_back(u);
    }
    dfs(1,0);
    while (m--)
    {
        int x,y;
        cin>>x>>y;
        solve(x,y);
    }

    return 0;
}
```

5 计算几何

5.1 ACW

```cpp
#include<algorithm>
#include<cstdio>
#include<cmath>


/*一：【准备工作】*/
#define LD double
```

```
8  #define LL long long
9  #define Re register int
10 #define Vector Point
11 using namespace std;
12 const int N=262144+3;
13 const LD eps=1e-8,Pi=acos(-1.0);
14 inline int dcmp(LD a){return a<-eps?-1:(a>eps?1:0);}//处理精度
15 inline LD Abs(LD a){return a*dcmp(a);}//取绝对值
16 struct Point{
17    LD x,y;Point(LD X=0,LD Y=0){x=X,y=Y;}
18    inline void in(){scanf("%lf%lf",&x,&y);}
19    inline void out(){printf("%.2lf %.2lf\n",x,y);}
20 };
21
22
23 /*二：【向量】*/
24 inline LD Dot(Vector a,Vector b){return a.x*b.x+a.y*b.y;}//【点积】
25 inline LD Cro(Vector a,Vector b){return a.x*b.y-a.y*b.x;}//【叉积】
26 inline LD Len(Vector a){return sqrt(Dot(a,a));}//【模长】
27 inline LD Angle(Vector a,Vector b){return acos(Dot(a,b)/Len(a)/Len(b));}//【两向量夹角】
28 inline Vector Normal(Vector a){return Vector(-a.y,a.x);}//【法向量】
29 inline Vector operator+(Vector a,Vector b){return Vector(a.x+b.x,a.y+b.y);}
30 inline Vector operator-(Vector a,Vector b){return Vector(a.x-b.x,a.y-b.y);}
31 inline Vector operator*(Vector a,LD b){return Vector(a.x*b,a.y*b);}
32 inline bool operator==(Point a,Point b){return !dcmp(a.x-b.x)&&!dcmp(a.y-b.y);}//两点坐标
       重合则相等
33
34
35 /*三：【点、向量的位置变换】*/
36
37 /*1.【点、向量的旋转】*/
38 inline Point turn_P(Point a,LD theta){//【点A\向量A顺时针旋转theta(弧度)】
39    LD x=a.x*cos(theta)+a.y*sin(theta);
40    LD y=-a.x*sin(theta)+a.y*cos(theta);
41    return Point(x,y);
42 }
43 inline Point turn_PP(Point a,Point b,LD theta){//【将点A绕点B顺时针旋转theta(弧度)】
44    LD x=(a.x-b.x)*cos(theta)+(a.y-b.y)*sin(theta)+b.x;
45    LD y=-(a.x-b.x)*sin(theta)+(a.y-b.y)*cos(theta)+b.y;
46    return Point(x,y);
47 }
48
49
50 /*四：【图形与图形之间的关系】*/
51
52 /*1.【点与线段】*/
53 inline int pan_PL(Point p,Point a,Point b){//【判断点P是否在线段AB上】
54    return !dcmp(Cro(p-a,b-a))&&dcmp(Dot(p-a,p-b))<=0;//做法一
55 // return !dcmp(Cro(p-a,b-a))&&dcmp(min(a.x,b.x)-p.x)<=0&&dcmp(p.x-max(a.x,b.x))<=0&&dcmp
       (min(a.y,b.y)-p.y)<=0&&dcmp(p.y-max(a.y,b.y))<=0;//做法二
56    //PA,AB共线且P在AB之间(其实也可以用len(p-a)+len(p-b)==len(a-b)判断，但是精度损失较大)
57 }
58 inline LD dis_PL(Point p,Point a,Point b){// 【点P到线段AB距离】
```

```
59      if(a==b)return Len(p-a);//AB重合
60      Vector x=p-a,y=p-b,z=b-a;
61      if(dcmp(Dot(x,z))<0)return Len(x);//P距离A更近
62      if(dcmp(Dot(y,z))>0)return Len(y);//P距离B更近
63      return Abs(Cro(x,z)/Len(z));//面积除以底边长
64  }
65
66  /*2.【点与直线】*/
67  inline int pan_PL_(Point p,Point a,Point b){//【判断点P是否在直线AB上】
68      return !dcmp(Cro(p-a,b-a));//PA,AB共线
69  }
70  inline Point FootPoint(Point p,Point a,Point b){//【点P到直线AB的垂足】
71      Vector x=p-a,y=p-b,z=b-a;
72      LD len1=Dot(x,z)/Len(z),len2=-1.0*Dot(y,z)/Len(z);//分别计算AP,BP在AB,BA上的投影
73      return a+z*(len1/(len1+len2));//点A加上向量AF
74  }
75  inline Point Symmetry_PL(Point p,Point a,Point b){//【点P关于直线AB的对称点】
76      return p+(FootPoint(p,a,b)-p)*2;//将PF延长一倍即可
77  }
78
79  /*3.【线与线】*/
80  inline Point cross_LL(Point a,Point b,Point c,Point d){//【两直线AB,CD的交点】
81      Vector x=b-a,y=d-c,z=a-c;
82      return a+x*(Cro(y,z)/Cro(x,y));//点A加上向量AF
83  }
84  inline int pan_cross_L_L(Point a,Point b,Point c,Point d){//【判断直线AB与线段CD是否相交】
85      return pan_PL(cross_LL(a,b,c,d),c,d);//直线AB与直线CD的交点在线段CD上
86  }
87  inline int pan_cross_LL(Point a,Point b,Point c,Point d){//【判断两线段AB,CD是否相交】
88      LD c1=Cro(b-a,c-a),c2=Cro(b-a,d-a);
89      LD d1=Cro(d-c,a-c),d2=Cro(d-c,b-c);
90      return dcmp(c1)*dcmp(c2)<0&&dcmp(d1)*dcmp(d2)<0;//分别在两侧
91  }
92
93  /*4.【点与多边形】*/
94  inline int PIP(Point *P,Re n,Point a){//【射线法】判断点A是否在任意多边形Poly以内
95      Re cnt=0;LD tmp;
96      for(Re i=1;i<=n;++i){
97          Re j=i<n?i+1:1;
98          if(pan_PL(a,P[i],P[j]))return 2;//点在多边形上
99          if(a.y>=min(P[i].y,P[j].y)&&a.y<max(P[i].y,P[j].y))//纵坐标在该线段两端点之间
100             tmp=P[i].x+(a.y-P[i].y)/(P[j].y-P[i].y)*(P[j].x-P[i].x),cnt+=dcmp(tmp-a.x)>0;//
                    交点在A右方
101     }
102     return cnt&1;//穿过奇数次则在多边形以内
103 }
104 inline int judge(Point a,Point L,Point R){//判断AL是否在AR右边
105     return dcmp(Cro(L-a,R-a))>0;//必须严格以内
106 }
107 inline int PIP_(Point *P,Re n,Point a){//【二分法】判断点A是否在凸多边形Poly以内
108     //点按逆时针给出
109     if(judge(P[1],a,P[2])||judge(P[1],P[n],a))return 0;//在P[1_2]或P[1_n]外
110     if(pan_PL(a,P[1],P[2])||pan_PL(a,P[1],P[n]))return 2;//在P[1_2]或P[1_n]上
```

```
111     Re l=2,r=n-1;
112     while(l<r){//二分找到一个位置pos使得P[1]_A在P[1_pos],P[1_(pos+1)]之间
113         Re mid=l+r+1>>1;
114         if(judge(P[1],P[mid],a))l=mid;
115         else r=mid-1;
116     }
117     if(judge(P[l],a,P[l+1]))return 0;//在P[pos_(pos+1)]外
118     if(pan_PL(a,P[l],P[l+1]))return 2;//在P[pos_(pos+1)]上
119     return 1;
120 }
121 
122 /*5.【线与多边形】*/
123 
124 /*6.【多边形与多边形】*/
125 inline int judge_PP(Point *A,Re n,Point *B,Re m){// 【判断多边形A与多边形B是否相离】
126     for(Re i1=1;i1<=n;++i1){
127         Re j1=i1<n?i1+1:1;
128         for(Re i2=1;i2<=m;++i2){
129             Re j2=i2<m?i2+1:1;
130             if(pan_cross_LL(A[i1],A[j1],B[i2],B[j2]))return 0;//两线段相交
131             if(PIP(B,m,A[i1])||PIP(A,n,B[i2]))return 0;//点包含在内
132         }
133     }
134     return 1;
135 }
136 
137 
138 /*五: 【图形面积】*/
139 
140 /*1.【任意多边形面积】*/
141 inline LD PolyArea(Point *P,Re n){// 【任意多边形P的面积】
142     LD S=0;
143     for(Re i=1;i<=n;++i)S+=Cro(P[i],P[i<n?i+1:1]);
144     return S/2.0;
145 }
146 
147 /*2.【圆的面积并】*/
148 
149 /*3.【三角形面积并】*/
150 
151 
152 /*六: 【凸包】*/
153 
154 /*1.【求凸包】*/
155 inline bool cmp1(Vector a,Vector b){return a.x==b.x?a.y<b.y:a.x<b.x;};//按坐标排序
156 inline int ConvexHull(Point *P,Re n,Point *cp){// 【Graham扫描法】求凸包
157     sort(P+1,P+n+1,cmp1);
158     Re t=0;
159     for(Re i=1;i<=n;++i){//下凸包
160         while(t>1&&dcmp(Cro(cp[t]-cp[t-1],P[i]-cp[t-1]))<=0)--t;
161         cp[++t]=P[i];
162     }
163     Re St=t;
```

```
164     for(Re i=n-1;i>=1;--i){//上凸包
165         while(t>St&&dcmp(Cro(cp[t]-cp[t-1],P[i]-cp[t-1]))<=0)--t;
166         cp[++t]=P[i];
167     }
168     return --t;//要减一
169 }
170 /*2.【旋转卡壳】*/
171 
172 /*3.【半平面交】*/
173 struct Line{
174     Point a,b;LD k;Line(Point A=Point(0,0),Point B=Point(0,0)){a=A,b=B,k=atan2(b.y-a.y,b.
        x-a.x);}
175     inline bool operator<(const Line &O)const{return dcmp(k-O.k)?dcmp(k-O.k)<0:judge(O.a,
        O.b,a);}//如果角度相等则取左边的
176 }L[N],Q[N];
177 inline Point cross(Line L1,Line L2){return cross_LL(L1.a,L1.b,L2.a,L2.b);}//获取直线L1,L2
    的交点
178 inline int judge(Line L,Point a){return dcmp(Cro(a-L.a,L.b-L.a))>0;}//判断点a是否在直线L的
    右边
179 inline int halfcut(Line *L,Re n,Point *P){//【半平面交】
180     sort(L+1,L+n+1);Re m=n;n=0;
181     for(Re i=1;i<=m;++i)if(i==1||dcmp(L[i].k-L[i-1].k))L[++n]=L[i];
182     Re h=1,t=0;
183     for(Re i=1;i<=n;++i){
184         while(h<t&&judge(L[i],cross(Q[t],Q[t-1])))--t;//当队尾两个直线交点不是在直线L[i]上或
            者左边时就出队
185         while(h<t&&judge(L[i],cross(Q[h],Q[h+1])))++h;//当队头两个直线交点不是在直线L[i]上或
            者左边时就出队
186         Q[++t]=L[i];
187     }
188     while(h<t&&judge(Q[h],cross(Q[t],Q[t-1])))--t;
189     while(h<t&&judge(Q[t],cross(Q[h],Q[h+1])))++h;
190     n=0;
191     for(Re i=h;i<=t;++i)P[++n]=cross(Q[i],Q[i<t?i+1:h]);
192     return n;
193 }
194 
195 /*4.【闵可夫斯基和】*/
196 Vector V1[N],V2[N];
197 inline int Mincowski(Point *P1,Re n,Point *P2,Re m,Vector *V){//【闵可夫斯基和】求两个凸包{
    P1},{P2}的向量集合{V}={P1+P2}构成的凸包
198     for(Re i=1;i<=n;++i)V1[i]=P1[i<n?i+1:1]-P1[i];
199     for(Re i=1;i<=m;++i)V2[i]=P2[i<m?i+1:1]-P2[i];
200     Re t=0,i=1,j=1;V[++t]=P1[1]+P2[1];
201     while(i<=n&&j<=m)++t,V[t]=V[t-1]+(dcmp(Cro(V1[i],V2[j]))>0?V1[i++]:V2[j++]);
202     while(i<=n)++t,V[t]=V[t-1]+V1[i++];
203     while(j<=m)++t,V[t]=V[t-1]+V2[j++];
204     return t;
205 }
206 
207 /*5.【动态凸包】*/
208 
209 /*七：【圆】*/
```

```
210
211 /*1.【三点确定一圆】*/
212 #define S(a) ((a)*(a))
213 struct Circle{Point O;LD r;Circle(Point P,LD R=0){O=P,r=R;}};
214 inline Circle getCircle(Point A,Point B,Point C){//【三点确定一圆】暴力解方程
215     LD x1=A.x,y1=A.y,x2=B.x,y2=B.y,x3=C.x,y3=C.y;
216     LD D=((S(x2)+S(y2)-S(x3)-S(y3))*(y1-y2)-(S(x1)+S(y1)-S(x2)-S(y2))*(y2-y3))/((x1-x2)*(
            y2-y3)-(x2-x3)*(y1-y2));
217     LD E=(S(x1)+S(y1)-S(x2)-S(y2)+D*(x1-x2))/(y2-y1);
218     LD F=-(S(x1)+S(y1)+D*x1+E*y1);
219     return Circle(Point(-D/2.0,-E/2.0),sqrt((S(D)+S(E)-4.0*F)/4.0));
220 }
221 inline Circle getcircle(Point A,Point B,Point C){//【三点确定一圆】向量垂心法
222     Point P1=(A+B)*0.5,P2=(A+C)*0.5;
223     Point O=cross_LL(P1,P1+Normal(B-A),P2,P2+Normal(C-A));
224     return Circle(O,Len(A-O));
225 }
226
227 /*2.【最小覆盖圆】*/
228 inline int PIC(Circle C,Point a){return dcmp(Len(a-C.O)-C.r)<=0;}//判断点A是否在圆C内
229 inline void Random(Point *P,Re n){for(Re i=1;i<=n;++i)swap(P[i],P[rand()%n+1]);}//随机一
        个排列
230 inline Circle Min_Circle(Point *P,Re n){//【求点集P的最小覆盖圆】
231 // random_shuffle(P+1,P+n+1);
232     Random(P,n);Circle C=Circle(P[1],0);
233     for(Re i=2;i<=n;++i)if(!PIC(C,P[i])){
234         C=Circle(P[i],0);
235         for(Re j=1;j<i;++j)if(!PIC(C,P[j])){
236             C.O=(P[i]+P[j])*0.5,C.r=Len(P[j]-C.O);
237             for(Re k=1;k<j;++k)if(!PIC(C,P[k]))C=getcircle(P[i],P[j],P[k]);
238         }
239     }
240     return C;
241 }
242
243 /*3.【三角剖分】*/
244 inline LD calc(Point A,Point B,Point O,LD R){//【三角剖分】
245     if(A==O||B==O)return 0;
246     Re op=dcmp(Cro(A-O,B-O))>0?1:-1;LD ans=0;
247     Vector x=A-O,y=B-O;
248     Re flag1=dcmp(Len(x)-R)>0,flag2=dcmp(Len(y)-R)>0;
249     if(!flag1&&!flag2)ans=Abs(Cro(A-O,B-O))/2.0;//两个点都在里面
250     else if(flag1&&flag2){//两个点都在外面
251         if(dcmp(dis_PL(O,A,B)-R)>=0)ans=R*R*Angle(x,y)/2.0;//完全包含了圆弧
252         else{//分三段处理 扇+圆弧+扇
253             if(dcmp(Cro(A-O,B-O))>0)swap(A,B);//把A换到左边
254             Point F=FootPoint(O,A,B);LD lenx=Len(F-O),len=sqrt(R*R-lenx*lenx);
255             Vector z=turn_P(F-O,Pi/2.0)*(len/lenx);Point B_=F+z,A_=F-z;
256             ans=R*R*(Angle(A-O,A_-O)+Angle(B-O,B_-O))/2.0+Cro(B_-O,A_-O)/2.0;
257         }
258     }
259     else{//一个点在里面，一个点在外面
260         if(flag1)swap(A,B);//使A为里面的点，B为外面的点
```

```
        Point F=FootPoint(O,A,B);LD lenx=Len(F-O),len=sqrt(R*R-lenx*lenx);
        Vector z=turn_P(F-O,Pi/2.0)*(len/lenx);Point C=dcmp(Cro(A-O,B-O))>0?F-z:F+z;
        ans=Abs(Cro(A-O,C-O))/2.0+R*R*Angle(C-O,B-O)/2.0;
    }
    return ans*op;
}

int main(){}
```

5.2 FGG

```
/*
精度是几何很重要的一环，尽量选择不需要精度的判法，比如说用点距离来判？
精度要求不高且点不多的的话，可以考虑一些暴力做法，比如圆周拆成10000个点求凸包..
关键点思想，临界点
一堆点绕原点旋转，最后要统计某一维的差值，考虑以某一个点i为参考系，将向量投影到方向向量上。
*/
typedef double db;
const db eps=1e-6,pi=acos(-1.0);
int sgn(db x) {return (x>eps)-(x<-eps);}///判断和0关系
//点 线基础
struct point {
    db x,y;
    point():x(0),y(0){}
    point(db a,db b):x(a),y(b){}
    void in() {cin>>x>>y;}
    point operator + (const point &a) const {return point(x+a.x,y+a.y);}
    point operator - (const point &a) const {return point(x-a.x,y-a.y);}
    point operator * (const db &a) const {return point(x*a,y*a);}
    point operator / (const db &a) const {return point(x/a,y/a);}
    db len() {return sqrt(x*x+y*y);}
///以下可以不抄
    db len2() {return x*x+y*y;}
    point unit() {db w=len(); return point(x/w,y/w);}
    point vertical(bool anti=true) {return anti?point(-y,x):point(y,-x);}///默认返回逆时针
        旋转90度
    point rotate(db rad,point o=point(0,0)) {///点绕o逆时针旋转rad
        return point((x-o.x)*cos(rad)-(y-o.y)*sin(rad)+o.x,(x-o.x)*sin(rad)+(y-o.y)*cos(
            rad)+o.y);
    }
    int getP() const { return sgn(y)==1 || (sgn(y)==0 && sgn(x)==-1);}///是否在上半平面
    bool operator < (const point k) const {
        int a=sgn(x-k.x);
        if(a==-1) return 1;
        else if(a==1) return 0;
        else return sgn(y-k.y)==-1;
    }
};
struct line {
    point a,b;
    line(){}
    line(point x,point y):a(x),b(y){}
```

```
};

db dot(point a,point b) {return a.x*b.x+a.y*b.y;}///为0垂直
db cross(point a,point b) {return a.x*b.y-a.y*b.x;}///为0平行
db angle(point a,point b) {return acos(dot(a,b)/a.len()/b.len());}///求向量a,b夹角弧度
db dist(point a,point b) {return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));}///两点距离
db area(point a,point b,point c) {return fabs(cross(b-a,c-a)/2);} ///三角形面积
int ccw(point p0,point p1,point p2){ //p0->p2在p0->p1方位
    point a=p1-p0;
    point b=p2-p0;
    if(cross(a,b) > eps) return 1; //COUNTER_CLOCKWISE
    if(cross(a,b) < -eps) return -1; //CLOCKWISE
    if(dot(a,b) < -eps) return 2; //反向延长线
    if(a.len2()<b.len2()) return -2; //正向延长线
    return 0; //线段上
}

bool line_intersection(line l1,line l2,point &a) {//求直线l1和l2的交点
    if(!sgn(cross(l1.a-l1.b,l2.a-l2.b))) return 0;
    db s1=cross(l1.a-l2.a,l2.b-l2.a);
    db s2=cross(l1.b-l2.a,l2.b-l2.a);
    a=(l1.b*s1-l1.a*s2)/(s1-s2);
    return 1;
}
bool intersect(point a,point b,point c,point d) {//判断线段AB与CD相交不考虑端点
    db t1=cross(c-a,d-a)*cross(c-b,d-b); ///对于线段AB，ac叉乘ad和bc叉乘db是反向的
    db t2=cross(a-c,b-c)*cross(a-d,b-d); ///同理对于CD
    return sgn(t1)<0 && sgn(t2)<0;
}
bool strictintersect(point a,point b,point c,point d) {///AB与CD相交考虑端点，特判端点
    return sgn(max(a.x,b.x)-min(c.x,d.x))>=0
        && sgn(max(c.x,d.x)-min(a.x,b.x))>=0
        && sgn(max(a.y,b.y)-min(c.y,d.y))>=0
        && sgn(max(c.y,d.y)-min(a.y,b.y))>=0
        && sgn(cross(c-a,d-a)*cross(c-b,d-b))<=0
        && sgn(cross(a-c,b-c)*cross(a-d,b-d))<=0;
}
db disttoline(point a,point m,point n) {//注意线退化点情况
    if(m.x==n.x&&m.y==n.y) return dist(a,m);
    return fabs(cross(a-m,a-n)/dist(m,n));
}///点a到直线mn的距离
bool onsegment(point p,point s,point t) {return sgn(cross(p-s,t-s))==0 && sgn(dot(p-s,p-t))<=0;}///p严格在线段st上
point proj(point k1, point k2, point q) { // q 到直线 k1,k2 的投影
    point k=k2-k1;
    return k1+k*(dot(q-k1,k)/k.len2());
}
point reflect(point k1, point k2, point q) { return proj(k1, k2, q) * 2 - q; }//反射 画一下

///多边形 凸包相关
bool inpolygon(point a,vector<point>&p) {//点和多边形位置关系 为2pi表示在内部，pi表示在边上
```

```
90      db alpha=0;
91      int sz=p.size();
92      for(int i=0;i<sz;++i) alpha+=fabs(angle(p[i]-a,p[(i+1)%sz]-a));
93      return sgn(alpha-2*pi)==0;
94  }
95  int checkconvex(vector<point>&p) { // 确认是否逆时针
96      int n=p.size();
97      p.push_back(p[0]);
98      p.push_back(p[1]);
99      for(int i=0;i<n;++i) if(sgn(cross(p[i+1]-p[i],p[i+2]-p[i]))==-1) return 0;
100     return 1;
101 }
102 db polygonarea(vector<point>&p) {
103     db sum=0;
104     point O=point(0,0); //O可任取，由于叉积有正有负，取在多边形外都可以
105     int sz=p.size();
106     for(int i=0;i<sz;++i) sum+=cross(p[i]-O,p[(i+1)%sz]-O);
107     if(sum<0) sum=-sum;
108     return sum/2.0;
109 }
110 //旋转卡壳
111 db rotatecalipers(point *ch,int n) { //求多边形直径 此处length表示长度的平方
112     db res=-1e18;
113     ch[n]=ch[0];
114     int q=1;
115     for(int i=0;i<n;++i) {
116         while(sgn(cross(ch[q+1]-ch[i+1],ch[i]-ch[i+1])-cross(ch[q]-ch[i+1],ch[i]-ch[i+1]))
                >0) q=(q+1)%n;
117         res=max(res,max((ch[q]-ch[i]).len(),(ch[q+1]-ch[i+1]).len()));
118     }
119     return res;
120 }
121 vector<point> ConvexHull(vector<point>&p, int flag = 1) {//凸包 flag=0 不严格 flag=1 严格
122     int n=p.size();
123     vector<point>ans(n*2);
124     sort(p.begin(),p.end());
125     int now=-1;
126     for(int i=0;i<n;++i) {
127         while(now>0 && sgn(cross(ans[now]-ans[now-1],p[i] - ans[now - 1])) < flag)
128             now--;
129         ans[++now] = p[i];
130     }
131     int pre=now;
132     for (int i=n-2;i>=0;--i) {
133         while (now > pre && sgn(cross(ans[now] - ans[now - 1], p[i] - ans[now - 1])) <
              flag)
134             now--;
135         ans[++now] = p[i];
136     }
137     ans.resize(now);
138     return ans;
139 }
140 db convexDiameter(vector<point>&a) { // 凸包直径
```

```
    int now=0,n=a.size();
    db ans=0;
    for (int i=0;i<a.size();++i) {
        now = max(now, i);
        while (1) {
            db k1=dist(a[i],a[now%n]), k2 = dist(a[i],a[(now+1)%n]);
            ans = max(ans, max(k1, k2));
            if(k2>k1) now++;
            else break;
        }
    }
    return ans;
}

db seg(point P, point A, point B){// 求多边形的面积并
    if(sgn(B.x - A.x) == 0) return (P.y - A.y) / (B.y - A.y);
    return (P.x - A.x) / (B.x - A.x);
}
db polygonUnion(vector<vector<point>> &p){
    db res = 0;
    for(int i = 0; i < p.size(); i++){
        int sz1 = p[i].size();
        for(int j = 0; j < sz1; j++){
            vector<pair<db, int> > s;
            s.push_back({0, 0});
            s.push_back({1, 0});
            point a = p[i][j], b = p[i][(j+1)%sz1];
            for(int k = 0; k < p.size(); k++) if(k != i){
                int sz2 = p[k].size();
                for(int l = 0; l < sz2; l++){
                    point c = p[k][l], d = p[k][(l+1)%sz2];
                    int c1 = sgn(cross(b - a, c - a));
                    int c2 = sgn(cross(b - a, d - a));
                    if(c1 == 0 && c2 == 0){
                        if(sgn(dot(b - a, d - c))){
                            s.push_back({seg(c ,a, b), 1});
                            s.push_back({seg(c, a, b), -1});
                        }
                    }
                    else{
                        db s1 = cross(d - c, a - c);
                        db s2 = cross(d - c, b - c);
                        if(c1 >= 0 && c2 < 0) s.push_back({s1 / (s1 - s2), 1});
                        else if(c1 < 0 && c2 >= 0) s.push_back({s1 / (s1 - s2), -1});
                    }
                }
            }
            sort(s.begin(), s.end());
            db pre = min(max(s[0].first, 0.0), 1.0), now, sum = 0;
            int cov = s[0].second;
            for(int j = 1; j < s.size(); j++){
                now = min(max(s[j].first, 0.0), 1.0);
                if(!cov) sum += now - pre;
```

```
194             cov += s[j].second;
195             pre = now;
196         }
197         res += cross(a, b) * sum;
198       }
199   }
200   return fabs(res) / 2;
201 }
202 db dis_point_to_seg(point x,point m,point n){ //x到直线mn距离
203   db res=0;
204   if(onsegment(proj(m, n, x), m, n)){
205      res+=disttoline(x, m, n);
206   }else res+=min(dist(x,m),dist(x,n));
207   return res;
208 }
209 db v[3],ansx,ansy,ansz;
210 void calc(db a,db b,db c,db rad,db x,db y,db z) {///(x,y,z) rotate by (a,b,c)
211   db len=sqrtl(a*a+b*b+c*c);
212   a/=len,b/=len,c/=len;
213   db cs=cos(rad*pi/180.0),si=sin(rad*pi/180);
214   db dot=a*x+b*y+c*z;
215   v[0]=x*cs+(b*z-c*y)*si+a*dot*(1.0-cs);
216   v[1]=y*cs+(c*x-a*z)*si+b*dot*(1.0-cs);
217   v[2]=z*cs+(a*y-b*x)*si+c*dot*(1.0-cs);
218   if(v[2]>ansz) ansx=v[0],ansy=v[1],ansz=v[2];
219 }
```

5.3 LLS 二维几何

```
1  //===二维几何
2  #include <bits/stdc++.h>
3  using namespace std;
4  #define mp make_pair
5  #define fi first
6  #define se second
7  #define pb push_back
8  typedef double db;
9  const db eps = 1e-6;
10 const db pi = acos(-1.0);
11 int sign(db k)
12 {
13    if (k > eps)
14       return 1;
15    else if (k < -eps)
16       return -1;
17    return 0;
18 }
19 int cmp(db k1, db k2) { return sign(k1 - k2); }
20 int inmid(db k1, db k2, db k3) { return sign(k1 - k3) * sign(k2 - k3) <= 0; } // k3 在 [
       k1,k2] 内
21 struct point
22 {
```

```
23      db x, y;
24      point operator+(const point &k1) const { return (point){k1.x + x, k1.y + y}; }
25      point operator-(const point &k1) const { return (point){x - k1.x, y - k1.y}; }
26      point operator*(db k1) const { return (point){x * k1, y * k1}; }
27      point operator/(db k1) const { return (point){x / k1, y / k1}; }
28      point operator-() const { return (point){-x, -y}; }
29      int operator==(const point &k1) const { return cmp(x, k1.x) == 0 && cmp(y, k1.y) ==
            0; }
30      // 逆时针旋转
31      point turn(db k1) { return (point){x * cos(k1) - y * sin(k1), x * sin(k1) + y * cos(
            k1)}; }
32      point turn90() { return (point){-y, x}; }
33      point turn270() { return (point){y, -x}; }
34      bool operator<(const point k1) const
35      {
36          int a = cmp(x, k1.x);
37          if (a == -1)
38              return 1;
39          else if (a == 1)
40              return 0;
41          else
42              return cmp(y, k1.y) == -1;
43      }
44      db abs() { return sqrt(x * x + y * y); }
45      db abs2() { return x * x + y * y; }
46      db dis(point k1) { return ((*this) - k1).abs(); }
47      point unit()
48      {
49          db w = abs();
50          return (point){x / w, y / w};
51      }
52      friend istream &operator>>(istream &is, point &k)
53      {
54          is >> k.x >> k.y;
55          return is;
56      }
57      friend ostream &operator<<(ostream &os, const point &k)
58      {
59          os << fixed << setprecision(2) << "(" << k.x << "," << k.y << ")\n";
60          return os;
61      }
62      db getw() { return atan2(y, x); }
63      point getdel()
64      {
65          if (sign(x) == -1 || (sign(x) == 0 && sign(y) == -1))
66              return (*this) * (-1);
67          else
68              return (*this);
69      }
70      int getP() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) == -1); }
71  };
72  int inmid(point k1, point k2, point k3) { return inmid(k1.x, k2.x, k3.x) && inmid(k1.y,
        k2.y, k3.y); }
```

```
db cross(point k1, point k2) { return k1.x * k2.y - k1.y * k2.x; }
db dot(point k1, point k2) { return k1.x * k2.x + k1.y * k2.y; }
db rad(point k1, point k2) { return atan2(cross(k1, k2), dot(k1, k2)); }
// -pi -> pi
bool compareangle(const point &k1, const point &k2)
{
    return k1.getP() < k2.getP() || (k1.getP() == k2.getP() && sign(cross(k1, k2)) > 0);
}
pair<bool, pair<point, point>> getAA(point k1, point k2, point k3, point k4)
{ //求角度向量 [k1,k2]是否交角度向量[k3,k4]
    vector<tuple<point, int, int>> v = {make_tuple(k1, 0, 1), make_tuple(k2, 0, -1),
        make_tuple(k3, 1, 1), make_tuple(k4, 1, -1)};
    sort(v.begin(), v.end(), [](auto &k1, auto &k2) {
        if (get<0>(k1).getP() == get<0>(k2).getP() && sign(cross(get<0>(k1), get<0>(k2)))
            == 0)
            return get<2>(k1) > get<2>(k2);
        return compareangle(get<0>(k1), get<0>(k2));
    });
    int cnt = 0;
    vector<bool> meet(2);
    for (auto [angle, id, t] : v)
    {
        if (!meet[id] && t == -1)
            cnt++;
        meet[id] = true;
    }
    if (cnt == 2)
        return make_pair(true, make_pair(get<0>(v.back()), get<0>(v.front())));
    for (int i = 0; i < v.size(); i++)
    {
        cnt += get<2>(v[i]);
        if (cnt == 2)
            return make_pair(true, make_pair(get<0>(v[i]), get<0>(v[(i + 1) % v.size()])));
    }
    return make_pair(false, make_pair(k1, k2));
}
point proj(point k1, point k2, point q)
{ // q 到直线 k1,k2 的投影
    point k = k2 - k1;
    return k1 + k * (dot(q - k1, k) / k.abs2());
}
point reflect(point k1, point k2, point q) { return proj(k1, k2, q) * 2 - q; }
int clockwise(point k1, point k2, point k3)
{ // k1 k2 k3 逆时针 1 顺时针 -1 否则 0
    return sign(cross(k2 - k1, k3 - k1));
}
int checkLL(point k1, point k2, point k3, point k4)
{ // 求直线 (L) 线段 (S)k1,k2 和 k3,k4 的交点
    return cmp(cross(k3 - k1, k4 - k1), cross(k3 - k2, k4 - k2)) != 0;
}
point getLL(point k1, point k2, point k3, point k4)
{
    db w1 = cross(k1 - k3, k4 - k3), w2 = cross(k4 - k3, k2 - k3);
```

```
124     return (k1 * w2 + k2 * w1) / (w1 + w2);
125 }
126 int intersect(db l1, db r1, db l2, db r2)
127 {
128     if (l1 > r1)
129         swap(l1, r1);
130     if (l2 > r2)
131         swap(l2, r2);
132     return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;
133 }
134 int checkSS(point k1, point k2, point k3, point k4)
135 {
136     return intersect(k1.x, k2.x, k3.x, k4.x) && intersect(k1.y, k2.y, k3.y, k4.y) &&
137           sign(cross(k3 - k1, k4 - k1)) * sign(cross(k3 - k2, k4 - k2)) <= 0 &&
138           sign(cross(k1 - k3, k2 - k3)) * sign(cross(k1 - k4, k2 - k4)) <= 0;
139 }
140 db disSP(point k1, point k2, point q)
141 {
142     point k3 = proj(k1, k2, q);
143     if (inmid(k1, k2, k3))
144         return q.dis(k3);
145     else
146         return min(q.dis(k1), q.dis(k2));
147 }
148 db disSS(point k1, point k2, point k3, point k4)
149 {
150     if (checkSS(k1, k2, k3, k4))
151         return 0;
152     else
153         return min(min(disSP(k1, k2, k3), disSP(k1, k2, k4)), min(disSP(k3, k4, k1), disSP
              (k3, k4, k2)));
154 }
155 int onS(point k1, point k2, point q) //点q在点k1,k2之间
156 {
157     return inmid(k1, k2, q) && sign(cross(k1 - q, k2 - k1)) == 0;
158 }
```

5.4 LLS 多边形

```
1  //===多边形
2  db area(vector<point> A)
3  { // 多边形用 vector<point> 表示 , 逆时针
4      db ans = 0;
5      for (int i = 0; i < A.size(); i++)
6          ans += cross(A[i], A[(i + 1) % A.size()]);
7      return ans / 2;
8  }
9  int checkconvex(vector<point> A)
10 { // 逆时针
11     int n = A.size();
12     A.push_back(A[0]);
13     A.push_back(A[1]);
```

```
14      for (int i = 0; i < n; i++)
15          if (sign(cross(A[i + 1] - A[i], A[i + 2] - A[i])) == -1)
16              return 0;
17      return 1;
18  }
19  int contain(vector<point> A, point q)
20  { // 2 内部 1 边界 0 外部
21      int pd = 0;
22      A.push_back(A[0]);
23      for (int i = 1; i < A.size(); i++)
24      {
25          point u = A[i - 1], v = A[i];
26          if (onS(u, v, q))
27              return 1;
28          if (cmp(u.y, v.y) > 0)
29              swap(u, v);
30          if (cmp(u.y, q.y) >= 0 || cmp(v.y, q.y) < 0)
31              continue;
32          if (sign(cross(u - v, q - v)) < 0)
33              pd ^= 1;
34      }
35      return pd << 1;
36  }
37  vector<point> ConvexHull(vector<point> A, int flag = 1) //凸包
38  { // flag=0 不严格 flag=1 严格
39      int n = A.size();
40      vector<point> ans(n * 2);
41      sort(A.begin(), A.end());
42      int now = -1;
43      for (int i = 0; i < A.size(); i++)
44      {
45          while (now > 0 && sign(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) < flag
                )
46              now--;
47          ans[++now] = A[i];
48      }
49      int pre = now;
50      for (int i = n - 2; i >= 0; i--)
51      {
52          while (now > pre && sign(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) <
                flag)
53              now--;
54          ans[++now] = A[i];
55      }
56      ans.resize(now);
57      return ans;
58  }
59  db convexDiameter(vector<point> A)
60  { // 凸包直径
61      int n = A.size(), now = 1;
62      A.push_back(A.front());
63      db ans = 0;
64      for (int i = 0; i < A.size(); i++)
```

```
65      {
66          while (cross(A[i + 1] - A[i], A[now] - A[i]) < cross(A[i + 1] - A[i], A[now + 1] -
                A[i]))
67              now = (now + 1) % n;
68          ans = max({ans, A[i].dis(A[now]), A[i + 1].dis(A[now + 1])});
69      }
70      return ans;
71  }
72  vector<point> convexcut(vector<point> A, point k1, point k2)
73  {
74      // 保留 k1,k2,p 逆时针的所有点
75      int n = A.size();
76      A.push_back(A[0]);
77      vector<point> ans;
78      for (int i = 0; i < n; i++)
79      {
80          int w1 = clockwise(k1, k2, A[i]), w2 = clockwise(k1, k2, A[i + 1]);
81          if (w1 >= 0)
82              ans.push_back(A[i]);
83          if (w1 * w2 < 0)
84              ans.push_back(getLL(k1, k2, A[i], A[i + 1]));
85      }
86      return ans;
87  }
```

5.5 LLS 圆和线段

```
1   struct circle
2   {
3       point o;
4       db r;
5       void scan()
6       {
7           o.scan();
8           scanf("%lf", &r);
9       }
10      int inside(point k) { return cmp(r, o.dis(k)); }
11  };
12  struct line
13  {
14      // p[0]->p[1]
15      point p[2];
16      line(point k1, point k2)
17      {
18          p[0] = k1;
19          p[1] = k2;
20      }
21      point &operator[](int k) { return p[k]; }
22      int include(point k) { return sign(cross(p[1] - p[0], k - p[0])) > 0; }
23      point dir() { return p[1] - p[0]; }
24      line push()
25      { // 向外 ( 左手边 ) 平移 eps
```

```
26          const db eps = 1e-6;
27          point delta = (p[1] - p[0]).turn90().unit() * eps;
28          return {p[0] - delta, p[1] - delta};
29      }
30      line push(db k)
31      { //向内平移k
32          point delta = (p[1] - p[0]).turn90().unit() * k;
33          return {p[0] + delta, p[1] + delta};
34      }
35  };
36  point getLL(line k1, line k2) { return getLL(k1[0], k1[1], k2[0], k2[1]); }
37  int parallel(line k1, line k2) { return sign(cross(k1.dir(), k2.dir())) == 0; }
38  int sameDir(line k1, line k2) { return parallel(k1, k2) && sign(dot(k1.dir(), k2.dir()))
        == 1; }
39  int operator<(line k1, line k2)
40  {
41      if (sameDir(k1, k2))
42          return k2.include(k1[0]);
43      return compareangle(k1.dir(), k2.dir());
44  }
45  int checkpos(line k1, line k2, line k3) { return k3.include(getLL(k1, k2)); }
46
47  int checkposCC(circle k1, circle k2)
48  { // 返回两个圆的公切线数量
49      if (cmp(k1.r, k2.r) == -1)
50          swap(k1, k2);
51      db dis = k1.o.dis(k2.o);
52      int w1 = cmp(dis, k1.r + k2.r), w2 = cmp(dis, k1.r - k2.r);
53      if (w1 > 0)
54          return 4;
55      else if (w1 == 0)
56          return 3;
57      else if (w2 > 0)
58          return 2;
59      else if (w2 == 0)
60          return 1;
61      else
62          return 0;
63  }
64  vector<point> getCL(circle k1, point k2, point k3)
65  { // 沿着 k2->k3 方向给出 , 相切给出两个
66      point k = proj(k2, k3, k1.o);
67      db d = k1.r * k1.r - (k - k1.o).abs2();
68      if (sign(d) == -1)
69          return {};
70      point del = (k3 - k2).unit() * sqrt(max((db)0.0, d));
71      return {k - del, k + del};
72  }
73  vector<point> getCC(circle k1, circle k2)
74  { // 沿圆 k1 逆时针给出 , 相切给出两个
75      int pd = checkposCC(k1, k2);
76      if (pd == 0 || pd == 4)
77          return {};
```

```
78      db a = (k2.o - k1.o).abs2(), cosA = (k1.r * k1.r + a - k2.r * k2.r) / (2 * k1.r *
            sqrt(max(a, (db)0.0)));
79      db b = k1.r * cosA, c = sqrt(max((db)0.0, k1.r * k1.r - b * b));
80      point k = (k2.o - k1.o).unit(), m = k1.o + k * b, del = k.turn90() * c;
81      return {m - del, m + del};
82  }
83  vector<point> TangentCP(circle k1, point k2)
84  { // 沿圆 k1 逆时针给出
85      db a = (k2 - k1.o).abs(), b = k1.r * k1.r / a, c = sqrt(max((db)0.0, k1.r * k1.r - b
            * b));
86      point k = (k2 - k1.o).unit(), m = k1.o + k * b, del = k.turn90() * c;
87      return {m - del, m + del};
88  }
89  vector<line> TangentoutCC(circle k1, circle k2)
90  {
91      int pd = checkposCC(k1, k2);
92      if (pd == 0)
93          return {};
94      if (pd == 1)
95      {
96          point k = getCC(k1, k2)[0];
97          return {(line){k, k}};
98      }
99      if (cmp(k1.r, k2.r) == 0)
100     {
101         point del = (k2.o - k1.o).unit().turn90().getdel();
102         return {(line){k1.o - del * k1.r, k2.o - del * k2.r}, (line){k1.o + del * k1.r, k2
                .o + del * k2.r}};
103     }
104     else
105     {
106         point p = (k2.o * k1.r - k1.o * k2.r) / (k1.r - k2.r);
107         vector<point> A = TangentCP(k1, p), B = TangentCP(k2, p);
108         vector<line> ans;
109         for (int i = 0; i < A.size(); i++)
110            ans.push_back((line){A[i], B[i]});
111         return ans;
112     }
113 }
114 vector<line> TangentinCC(circle k1, circle k2)
115 {
116     int pd = checkposCC(k1, k2);
117     if (pd <= 2)
118         return {};
119     if (pd == 3)
120     {
121         point k = getCC(k1, k2)[0];
122         return {(line){k, k}};
123     }
124     point p = (k2.o * k1.r + k1.o * k2.r) / (k1.r + k2.r);
125     vector<point> A = TangentCP(k1, p), B = TangentCP(k2, p);
126     vector<line> ans;
127     for (int i = 0; i < A.size(); i++)
```

```
128         ans.push_back((line){A[i], B[i]});
129     return ans;
130 }
131 vector<line> TangentCC(circle k1, circle k2)
132 {
133     int flag = 0;
134     if (k1.r < k2.r)
135         swap(k1, k2), flag = 1;
136     vector<line> A = TangentoutCC(k1, k2), B = TangentinCC(k1, k2);
137     for (line k : B)
138         A.push_back(k);
139     if (flag)
140         for (line &k : A)
141             swap(k[0], k[1]);
142     return A;
143 }
144 db getarea(circle k1, point k2, point k3)
145 {
146     // 圆 k1 与三角形 k2 k3 k1.o 的有向面积交
147     point k = k1.o;
148     k1.o = k1.o - k;
149     k2 = k2 - k;
150     k3 = k3 - k;
151     int pd1 = k1.inside(k2), pd2 = k1.inside(k3);
152     vector<point> A = getCL(k1, k2, k3);
153     if (pd1 >= 0)
154     {
155         if (pd2 >= 0)
156             return cross(k2, k3) / 2;
157         return k1.r * k1.r * rad(A[1], k3) / 2 + cross(k2, A[1]) / 2;
158     }
159     else if (pd2 >= 0)
160     {
161         return k1.r * k1.r * rad(k2, A[0]) / 2 + cross(A[0], k3) / 2;
162     }
163     else
164     {
165         int pd = cmp(k1.r, disSP(k2, k3, k1.o));
166         if (pd <= 0)
167             return k1.r * k1.r * rad(k2, k3) / 2;
168         return cross(A[0], A[1]) / 2 + k1.r * k1.r * (rad(k2, A[0]) + rad(A[1], k3)) / 2;
169     }
170 }
171 circle getcircle(point k1, point k2, point k3)
172 {
173     // 三点求圆
174     db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 + b1 * b1) / 2;
175     db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 + b2 * b2) / 2;
176     db d = a1 * b2 - a2 * b1;
177     point o = (point){k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1 * c2 - a2 * c1) / d};
178     return (circle){o, k1.dis(o)};
179 }
180 circle getScircle(vector<point> A)
```

```
{
    //随机增量法 最小圆覆盖
    random_shuffle(A.begin(), A.end());
    circle ans = (circle){A[0], 0};
    for (int i = 1; i < A.size(); i++)
        if (ans.inside(A[i]) == -1)
        {
            ans = (circle){A[i], 0};
            for (int j = 0; j < i; j++)
                if (ans.inside(A[j]) == -1)
                {
                    ans.o = (A[i] + A[j]) / 2;
                    ans.r = ans.o.dis(A[i]);
                    for (int k = 0; k < j; k++)
                        if (ans.inside(A[k]) == -1)
                            ans = getcircle(A[i], A[j], A[k]);
                }
        }
    return ans;
}
```

5.6 LLS 其他

```
vector<line> getHL(vector<line> &L)
{ // 求半平面交 , 半平面是逆时针方向 , 输出按照逆时针
    sort(L.begin(), L.end());
    deque<line> q;
    for (int i = 0; i < (int)L.size(); i++)
    {
        if (i && sameDir(L[i], L[i - 1]))
            continue;
        while (q.size() > 1 && !checkpos(q[q.size() - 2], q[q.size() - 1], L[i]))
            q.pop_back();
        while (q.size() > 1 && !checkpos(q[1], q[0], L[i]))
            q.pop_front();
        q.push_back(L[i]);
    }
    while (q.size() > 2 && !checkpos(q[q.size() - 2], q[q.size() - 1], q[0]))
        q.pop_back();
    while (q.size() > 2 && !checkpos(q[1], q[0], q[q.size() - 1]))
        q.pop_front();
    vector<line> ans;
    for (int i = 0; i < q.size(); i++)
        ans.push_back(q[i]);
    return ans;
}
db closepoint(vector<point> &A, int l, int r)
{ // 最近点对 , 先要按照 x 坐标排序
    if (r - l <= 5)
    {
        db ans = 1e20;
        for (int i = l; i <= r; i++)
```

```
30          for (int j = i + 1; j <= r; j++)
31              ans = min(ans, A[i].dis(A[j]));
32      return ans;
33   }
34   int mid = ((l + r)) >> 1;
35   db ans = min(closepoint(A, l, mid), closepoint(A, mid + 1, r));
36   vector<point> B;
37   for (int i = l; i <= r; i++)
38      if (abs(A[i].x - A[mid].x) <= ans)
39         B.push_back(A[i]);
40   sort(B.begin(), B.end(), [](point k1, point k2) { return k1.y < k2.y; });
41   for (int i = 0; i < B.size(); i++)
42      for (int j = i + 1; j < B.size() && B[j].y - B[i].y < ans; j++)
43         ans = min(ans, B[i].dis(B[j]));
44   return ans;
45}
46vector<point> convexcut(vector<point> A, point k1, point k2)
47{
48   // 保留 k1,k2,p 逆时针的所有点
49   int n = A.size();
50   A.push_back(A[0]);
51   vector<point> ans;
52   for (int i = 0; i < n; i++)
53   {
54      int w1 = clockwise(k1, k2, A[i]), w2 = clockwise(k1, k2, A[i + 1]);
55      if (w1 >= 0)
56         ans.push_back(A[i]);
57      if (w1 * w2 < 0)
58         ans.push_back(getLL(k1, k2, A[i], A[i + 1]));
59   }
60   return ans;
61}
62int checkPoS(vector<point> A, point k1, point k2)
63{
64   // 多边形 A 和直线 ( 线段 )k1->k2 严格相交 , 注释部分为线段
65   struct ins
66   {
67      point m, u, v;
68      int operator<(const ins &k) const { return m < k.m; }
69   };
70   vector<ins> B;
71   //if (contain(A,k1)==2||contain(A,k2)==2) return 1;
72   vector<point> poly = A;
73   A.push_back(A[0]);
74   for (int i = 1; i < A.size(); i++)
75      if (checkLL(A[i - 1], A[i], k1, k2))
76      {
77         point m = getLL(A[i - 1], A[i], k1, k2);
78         if (inmid(A[i - 1], A[i], m) /*&&inmid(k1,k2,m)*/)
79            B.push_back((ins){m, A[i - 1], A[i]});
80      }
81   if (B.size() == 0)
82      return 0;
```

```
83      sort(B.begin(), B.end());
84      int now = 1;
85      while (now < B.size() && B[now].m == B[0].m)
86          now++;
87      if (now == B.size())
88          return 0;
89      int flag = contain(poly, (B[0].m + B[now].m) / 2);
90      if (flag == 2)
91          return 1;
92      point d = B[now].m - B[0].m;
93      for (int i = now; i < B.size(); i++)
94      {
95          if (!(B[i].m == B[i - 1].m) && flag == 2)
96              return 1;
97          int tag = sign(cross(B[i].v - B[i].u, B[i].m + d - B[i].u));
98          if (B[i].m == B[i].u || B[i].m == B[i].v)
99              flag += tag;
100         else
101             flag += tag * 2;
102     }
103     //return 0;
104     return flag == 2;
105 }
```