# ACM Template

Zeng Xiaocan September 16, 2019

## Contents

1	Stri	$\mathbf{u}\mathbf{g}$
	1.1	STL
	1.2	Max/Min-Expression
	1.3	KMP
	1.4	EXKMP
	1.5	Hash
	1.6	Trie
	1.7	AC-Automaton
	1.8	Manacher
	1.9	Palindromic-Tree
	1.10	Suffix-Array
	1.10	1.10.1 Usage
	1 11	Suffix-Automaton
	1.11	
		1.11.1 Usage
		1.11.2 Memo
<b>2</b>	Prol	olemSet 19
_	2.1	Trie
	2.1	2.1.1 区间异或最大值
		2.1.2 路径异或最大值 2.1.2 2.1
		2.1.3 子树异或最大值 25
		2.1.4 编辑距离
	0.0	2.1.5 trie 树上 dp
	2.2	Pam
		2.2.1 双向插入 28
	2.0	2.2.2 next 和 fail 统计贡献
	2.3	Sa
		2.3.1 可重第 k 小子串
		2.3.2 单调栈 + 边界
		2.3.3 单调栈
	2.4	Sam
		2.4.1 出现 k 次子串
		2.4.2 不同子串个数
		2.4.3 长度大于等于 m 不同子串个数
		2.4.4 字典序第 k 小子串
		2.4.5 循环字典序第 k 小
	2.5	其他
		2.5.1 k 短路
		2.5.2 k 小团
		2.5.3 floyd 最小环
		2.5.4 dijk 费用流 +vector 建图
		2.5.5 spaf 费用流
		2.5.6 分层图最短路 55
		2.5.7 日期公式
		2.5.8 二维 RMQ
		2.5.9 左偏树 + 并查集
		2.5.10 左偏树 +dfs
		2.5.11 左偏树 +dfs
		2001 I /II իրդ իրդ իրդ լագարի և և և և և և և և և և և և և և և և և և և

## ACM Template By Zeng Xiaocan

2.5.12	树上启发式合并	64
2.5.13	线段树合并 + 树上 LIS	66
2.5.14	单调队列	68
2.5.15	康拓展开 + 字典序第 k 小全排列	69
2.5.16	fhq-treap	70
2.5.17	LCT 维护 MST	73
2.5.18	LCT 维护路径异或最大值	78

## 1 String

#### 1.1 STL

```
reverse(s.begin(), s.end());
transform(s.begin(), s.end(), s.begin(), ::toupper); (::tolower)
//字符串和数字互转
int a;
stringstream(s) » a;
char s[100];
sprint(s, "%d", a);
string(v.begin(),v.end());
//返回 pos 开始的长度为 len 的字符串
substr(pos,len);
//在 pos 位置插入字符串 s
insert(int pos,string s)
//从索引 pos 开始往后删 num 个, num 为空表示全删除
erase(pos,num);
//删除迭代器 it 指向的字符, 返回删除后迭代器的位置
erase(it);
//删除迭代器 [first, last) 之间的所有字符, 返回删除后迭代器的位置
erase(first, last);
//从 pos 开始查找字符 c/字符串 s 在当前字符串的位置
int find(c/s,pos);
```

### 1.2 Max/Min-Expression

```
//求循环字符串 s 的最小/最大表示
//i, j: 当前比较两个字符串的起始位置
//k: 这两个字符串已比较的长度
int getMin(char s[]){
   int n=strlen(s);
   int i=0, j=1, k=0;
   while(i<n && j<n && k<n){
       int t=s[(i+k)\%n]-s[(j+k)\%n];
       if(!t){
          k++;
       }else{
           if(t>0){
              //如果是求最大表示则为 j+=k+1
              i+=k+1;
          }else{
              j+=k+1;
          if(i==j){
              j++;
          }
          k=0;
       }
   }
```

```
return min(i,j);
}
1.3
     KMP
//nex[i]:表示前 i 个字符的最长相同前后缀长度
void getNext(char s[],int n){
   int i=0, j=-1;
   nex[0]=-1;
   while(i<n){
       if(j==-1 || s[i]==s[j]){
           nex[++i]=++j;
       }else{
           j=nex[j];
       }
   }
}
//前 i 个字符的最小循环节长度: i-nex[i], 个数: i/(i-nex[i])
int kmp(char s[],int n,char p[],int m){
   int i=0, j=0;
   // int cnt=0;
   getNext(p,m);
   while(i<n && j<m){
       if(j==-1 || s[i]==p[j]){
           i++;
           j++;
       }else{
           j=nex[j];
       if(j==m){
           //匹配位置
           return i-j+1;
           //匹配个数
           //cnt++;
           //不可重叠
           //j=0;
           //可重叠
           //j=nex[j];
       }
   }
   //return cnt;
}
     EXKMP
1.4
//nex[i] 表示 t 串中以 i 开始的后缀与 t 串的最长公共前缀
//ext[i] 表示 s 串中以 i 开始的后缀与 t 串的最长公共前缀
void getNex(char *t,int len){
   int a=0;
   while (a < len-1 \&\& t[a] == t[a+1])
```

```
a++;
    }
   nex[1]=a;
    int po=1;
    for(int i=2;i<len;i++){</pre>
       int p=po+nex[po]-1;
       int l=nex[i-po];
       if(1>=p-i+1){
            int j=max(0,p-i+1);
           while (i+j \le k t[i+j] == t[j])
               j++;
           }
           nex[i]=j;
           po=i;
       }else{
           nex[i]=1;
       }
    }
}
void getExt(char *s,int n,char *t,int m){
    int a=0;
    getNex(t,m);
    int mlen=min(n,m);
    //计算 ext[0]
    while (a \le m \le s = t = t = t)
       a++;
    }
    ext[0]=a;
    //po 表示当前最右的 i+ext[i]-1 所对应的 i
    int po=0;
    for(int i=1;i<n;i++){</pre>
        //p 表示最右的 i+ext[i]-1
       int p=po+ext[po]-1;
        //此时前面已匹配的 s[po...p] == t[O...p-po], 即 s[i...p] == t[i-po...p-po]
       //所以 l 就是表示 t[i-po...m-1] 和 t[0..m-1] 的 lcp
       //也就是 s[i..p] 和 t[0..m-1] 的 ** 部分 **lcp
       //得看 l 和 p-i+1(ext[i] 可能的最大值) 哪个大
       int l=nex[i-po];
       if(1>=p-i+1){
           //l 大, 那么从 p-i+1(目前可以保证的 ext[i] 的值) 继续暴力往下匹配
           int j=\max(0,p-i+1);
           while(i+j \le n \&\& j \le m \&\& s[i+j] == t[j]){
           }
           ext[i]=j;
           po=i;
       }else{
            //p-i+1 大, 那么 ext[i] 就只能是 l 了
           ext[i]=1;
```

```
}
    }
}
     Hash
1.5
//单哈希很容易卡;取模很慢
ull seeds[]={27,146527,19260817,91815541};
ull mods[]={1000000009,998244353,4294967291ull,21237044013013795711};
struct Hash{
    ull seed, mod;
    ull bas[N];
    ull sum[N];
    void init(int sidx,int midx,int len,char *s){
        seed=seeds[sidx];
        mod=mods[midx];
        bas[0]=1;
        for(int i=1;i<=len;i++){</pre>
           bas[i]=bas[i-1]*seed%mod;
        }
        for(int i=1;i<=len;i++){</pre>
            sum[i]=(sum[i-1]*seed%mod+s[i])%mod;
        }
    }
    ull getHash(int l,int r){
        return (sum[r]-sum[l-1]*bas[r-l+1]%mod+mod)%mod;
    }
}hs;
     Trie
1.6
//val[u] 表示 u 节点处保存的单词数
struct Trie{
    int cnt,tr[N][26],val[N];
    void insert(char *s){
        int len=strlen(s);
        int now=0;
        for(int i=0;i<len;i++){</pre>
            int id=s[i]-'a';
            if(!tr[now][id]){
                tr[now][id]=++cnt;
            }
            now=tr[now][id];
        }
        val[now]++;
    }
}T;
```

#### 1.7 AC-Automaton

```
//fail[x] 指向以 x 为结尾的后缀在 ** 其他模式串中 ** 所能匹配的最长前缀
//当 tr[now][i] 失配时, 就可以跳转到以已匹配的这部分后缀作为前缀的其他模式串。
struct ACM{
   int tr[N][26],val[N],fail[N],cnt;
   void insert(char *s){
       int len=strlen(s);
       int now=0;
       for(int i=0;i<len;i++){</pre>
           int id=s[i]-'a';
           if(!tr[now][id]){
               tr[now][id]=++cnt;
           now=tr[now][id];
       val[now]++;
   }
   //比 Trie 树多了构建 fail 指针
   void build(){
       queue<int> q;
       //初始化第一层
       for(int i=0;i<26;i++){</pre>
           if(tr[0][i]){
               fail[tr[0][i]]=0;
               q.push(tr[0][i]);
           }
       }
       while(!q.empty()){
           int u=q.front();
           q.pop();
           for(int i=0;i<26;i++){</pre>
               if(tr[u][i]){
                   fail[tr[u][i]]=tr[fail[u]][i];
                   q.push(tr[u][i]);
               }else{
                   tr[u][i]=tr[fail[u]][i];
               }
           }
       }
   }
   //查询所有模式串出现的总次数
   int query(char *s){
       int len=strlen(s);
       int ans=0;
       int now=0;
       for(int i=0;i<len;i++){</pre>
           int id=s[i]-'a';
           now=tr[now][id];
           //打标记暴力跳 fail, 避免重复计数
           for(int t=now;t && val[t]!=-1; t=fail[t]){
```

```
ans+=val[t];
             val[t]=-1;
          }
      }
      return ans;
   }
}ac;
    Manacher
1.8
//ma[]: 新字符串 (ma, mp 都注意要开两倍空间!)
//mp[i]:表示以 i 为中心的回文子串的半径 (包括特殊字符)
//ma: 能延伸到最右端的位置
//id: 能延伸到最右端的回文串中心位置
void manacher(char s[],int len){
   //构造新字符串,两个字符之间插入一个其他字符,第 0 个字符忽略 (即加入另一种字符)
   int 1=0;
   ma[1++]='$';
   ma[1++]='#';
   for(int i=0;i<len;i++){</pre>
      ma[1++]=s[i];
      ma[1++]='#';
   }
   ma[1] = ' \setminus 0';
   int mx=0,id=0;
   for(int i=1;i<1;i++){</pre>
      //若 mx>i: mp[2*id-i] 表示 i 关于 id 的对称点的最长回文半径
      //不能超出 mx, 所以和 mx-i 取 min
      //若 mx<i: mp[i]=1
      mp[i]=mx>i?min(mp[2*id-i],mx-i):1;
      //往两边更新
      while (ma[i+mp[i]] == ma[i-mp[i]]) {
          mp[i]++;
      }
      //更新全局 mac 和 id
      if(i+mp[i]>mx){
          mx=i+mp[i];
          id=i;
      }
   }
}
    Palindromic-Tree
1.9
struct PT{
      //回文树中每个节点表示一个回文串, 所以有偶数长度的树和奇数长度的树两棵
      //next 指针 next[u][i] 表示 u 节点左右添加字符 i 之后得到的回文串节点
      int next[N][26];
      //fail 指针 失配后跳转到最长后缀回文串对应的节点
      int fail[N];
```

```
//节点对应回文串在原串中出现次数,需先调用 count 函数
   int cnt[N];
   //num[i] 表示 ** 以节点 i 所表示的回文串右端点结尾 ** 的回文串个数 (包括自身)
//即 fail 指针的深度
   int num[N];
   //节点对应回文串的长度
   int len[N]:
   //存放添加的字符
   int S[N];
   //上一个字符所在节点
   int last;
   //节点对应的最新字符位置,反向映射 last(可以改成 vector<int>[])
   int id[N]:
   //字符数,不等于节点数
   int n;
   //回文树总结点数,包括奇偶两个空节点,节点编号为 0 到 p-1
   //不同回文子串个数 p-2 回文子串个数 \sum num[i]
   //创建长度为 1 的新节点
   int newnode(int 1){
         for(int i=0;i<26;i++){</pre>
               next[p][i]=0;
         cnt[p]=0;
         num[p]=0;
         len[p]=1;
         return p++;
   }
   //初始化
   void init(){
         p=0;
         //奇偶空节点, 先偶再奇
         newnode(0);
         newnode(-1);
         last=0;
         n=0;
         S[n] = -1;
         //偶根 fail 指向奇根
         fail[0]=1;
   //找到新插入字符 c 的回文匹配位置
   int getFail(int x){
         //在节点 x 对应串的后面加上一个字符, 就判断 x 前面字符是否相同
         //若相同直接构成新的回文串, 不同就跳到 fail, 即最长回文后缀
         //S[n-len[x]-1] 就是新加的字符 (S[n]) 关于 x 串的对称字符
         while (S[n-len[x]-1]!=S[n]) {
               x=fail[x];
         }
         return x;
   }
```

```
//插入字符 c
      void add(int c){
             c-= 'a';
             S[++n]=c;
             //找到当前回文串匹配位置,也就是当前回文串节点的父节点
             int cur=getFail(last);
             if(!next[cur][c]){
                   //出现了一个新的本质不同的回文串
                   int now=newnode(len[cur]+2);
                   //类似于 AC 自动机, 往上跳直到找到满足条件的串节点
                   //qetFail 其实就是不断比较当前加入的字符和 x 节点对称的那个字符
                   fail[now] = next[getFail(fail[cur])][c];
                   //fail 指针深度加 1
                   num[now] = num[fail[now]]+1;
                   //这句要放最后, 前面的指针关系处理好再连上子节点
                   next[cur][c]=now;
             //最新回文串节点
             last=next[cur][c];
             cnt[last]++;
             id[last] = n;
      }
      //统计每个节点回文串出现次数
      void count(){
             //从子节点逆推
             for(int i=p-1;i>=0;i--){
                   //i 节点出现, 说明其最长回文后缀 fail[i] 也出现
                   cnt[fail[i]]+=cnt[i]:
             }
      }
}ac;
     Suffix-Array
1.10
//"banana" 后缀为 [banana$ anana$ nana$ ana$ na$ a$ $]
//sa[i]: 排名第 i(从 O 开始) 小的后缀的首字符下标
//比如 [6 5 3 1 0 4 2]==>[$ a$ ana$ anana$ banana$ na$ nana$]
//rk[i]: 下标 i 开始的后缀 (不含 $) 的排名 (按字典序从小到大,相当于 sa 的逆)
//[4 3 6 2 5 1]
//h[i]: 排名为 i 的后缀和排名为 i-1 的后缀的最长公共前缀
//[1(ana$-a$) 3(anana$-ana$) 0 0 2]
//辅助数组: t[N],t2[N],c[N];
void build sa(int n, int m){
   //n 为字符串的长度,字符集的值为 0~m-1
   //相当于在后面加一个 $
   //有时候是数字数组而不是字符数组, 最好加上 s[n]=0
   n++;
   int *x=t, *y=t2;
   //基数排序
   for(int i=0;i<m;i++){</pre>
```

```
c[i]=0;
}
for(int i=0;i<n;i++){</pre>
    c[x[i]=s[i]]++;
}
for(int i=1;i<m;i++){</pre>
    c[i] += c[i-1];
}
//或者 ~i 表示 i!=-1
for(int i=n-1;i>=0;i--){
    sa[--c[x[i]]]=i;
}
for(int k=1; k<=n; k<<=1){</pre>
    int p=0;
    for(int i=n-k;i<n;i++){</pre>
         y[p++]=i;
    }
    for(int i=0;i<n;i++){</pre>
         if(sa[i]>=k){
             y[p++]=sa[i]-k;
    }
    //类似上面, 只是把 i 换成 y[i]
    for(int i=0;i<m;i++){</pre>
         c[i]=0;
    for(int i=0;i<n;i++){</pre>
         c[x[y[i]]]++;
    for(int i=1;i<m;i++){</pre>
         c[i] += c[i-1];
    for(int i=n-1;i>=0;i--){
         sa[--c[x[y[i]]]=y[i];
    swap(x, y);
    p=1;
    x[sa[0]]=0;
    for(int i=1;i<n;i++){</pre>
         x[sa[i]]=y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k]? p-1 : p++;
    }
    if (p>=n){
        break;
    }
    m = p;
}
//去掉 $
n--;
for(int i = 0; i <= n; i++){
    rk[sa[i]] = i;
```

```
}
    //计算 h
    int k=0;
    for(int i = 0; i < n; i++){
        if(k){
            k--;
        }
        int j = sa[rk[i] - 1];
        while(s[i + k] == s[j + k])\{
            k++;
        }
        h[rk[i]] = k;
    }
}
void debug(){
    //sa 0~n 包括一个特殊字符 从 0 计
    for(int i=0;i<=n;i++){</pre>
        printf("%d ",sa[i]);
    }
   printf("\n");
    //rk 0~n-1 后缀 [i...n-1] 的排名 从 1 计
    for(int i=0;i<n;i++){</pre>
        printf("%d ",rk[i]);
    }
   printf("\n");
    //h 1~n 排名为 i 的后缀与排名为 i-1 的后缀的 LCP
    for(int i=1;i<=n;i++){</pre>
        printf("%d ",h[i]);
   printf("\n");
}
/*
 * 使用 DC3 构建后缀数组 O(n) by Kuangbin 模板
 * 所有数组要开三倍
 * wa[N*3], wb[N*3], wv[N*3], wss[N*3]
#define F(x) ((x)/3+((x)\%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int c0(int *r,int a,int b){
    return r[a] == r[b] \&\& r[a+1] == r[b+1] \&\& r[a+2] == r[b+2];
}
int c12(int k,int *r,int a,int b){
    if(k == 2){
        return r[a] < r[b] \mid \mid (r[a] == r[b] \&\& c12(1,r,a+1,b+1));
        return r[a] < r[b] \mid \mid (r[a] == r[b] \&\& wv[a+1] < wv[b+1]);
    }
}
void sort(int *r,int *a,int *b,int n,int m){
    int i;
```

```
for(i = 0; i < n; i++){
        wv[i] = r[a[i]];
    }
    for(i = 0; i < m; i++){
        wss[i] = 0;
    }
    for(i = 0; i < n; i++){
        wss[wv[i]]++;
    }
    for(i = 1; i < m; i++){
        wss[i] += wss[i-1];
    for(i = n-1; i >= 0; i--){
        b[--wss[wv[i]]] = a[i];
    }
}
void dc3(int *r,int *sa,int n,int m){
    int i, j, *rn = r + n;
    int *san = sa + n, ta = 0, tb = (n+1)/3, tbc = 0, p;
    r[n] = r[n+1] = 0;
    for(i = 0; i < n; i++){
        if(i %3 != 0){
            wa[tbc++] = i;
        }
    }
    sort(r + 2, wa, wb, tbc, m);
    sort(r + 1, wb, wa, tbc, m);
    sort(r, wa, wb, tbc, m);
    for(p = 1, rn[F(wb[0])] = 0, i = 1;i < tbc;i++){
        rn[F(wb[i])] = c0(r, wb[i-1], wb[i]) ? p-1 : p++;
    }
    if(p < tbc){}
        dc3(rn,san,tbc,p);
    }else{
        for(i = 0; i < tbc; i++){
            san[rn[i]] = i;
        }
    for(i = 0; i < tbc; i++){
        if(san[i] < tb){
            wb[ta++] = san[i] * 3;
        }
    }
    if(n \% 3 == 1){
        wb[ta++] = n - 1;
    sort(r, wb, wa, ta, m);
    for(i = 0; i < tbc; i++){
        wv[wb[i] = G(san[i])] = i;
    }
```

```
for(i = 0, j = 0, p = 0;i < ta && j < tbc;p++){
        sa[p] = c12(wb[j] \% 3, r, wa[i], wb[j]) ? wa[i++] : wb[j++];
    }
    for(;i < ta;p++){
        sa[p] = wa[i++];
    }
    for(; j < tbc; p++){</pre>
        sa[p] = wb[j++];
    }
}
//str 和 sa 也要三倍
void da(int str[],int n,int m){
    for(int i = n; i < n*3; i++){
        str[i] = 0;
    }
    dc3(str, sa, n+1, m);
    int i,j,k = 0;
    for(int i = 0; i <= n; i++){
        rk[sa[i]] = i;
    }
    //计算 h
    for(int i = 0; i < n; i++){
        if(k){}
            k--;
        }
        int j = sa[rk[i] - 1];
        while(a[i + k] == a[j + k]){
            k++;
        h[rk[i]] = k;
    }
}
```

#### 1.10.1 Usage

0 循环字符串字典序第 k 小

将原串拼接在最后,再加一个大于字符集最大值的字符,计算 sa, sa 本身就是对后缀进行排序,按顺序枚举 k 个有效 (sa[i] 在 0-n) 的后缀即可。

#### 1.11 Suffix-Automaton

```
int len[N*2];
//状态对应 endpos 大小,即子串出现次数
int num[N*2];
//总节点数
int cnt;
//上一个节点
int lst;
int newnode(int 1,int s){
   for(int i=0;i<26;i++){</pre>
      next[cnt][i]=0;
   }
   len[cnt]=1;
   num[cnt]=s;
   return cnt++;
}
//初始化
void init(){
   cnt=0;
   lst=newnode(0,0);
   fa[lst]=-1;
void add(int c){
   c-= 'a';
   int p=lst;
   int cur=newnode(len[p]+1,1);
   //假设当前 sam 为"aabb",起点 S 为空串,节点 5 是 {b},节点 4 是 {aabb,abb,bb}
   //定义 suffix-path 为当前字符串的所有后缀的状态,即 S[1..i], S[2..i]...
   //此时的 s-p 就是 S-5-4, (b 这个后缀因为 endpos 大于其他, 所以在节点 5)
   //每插入一个字符, s-p 的遍历是从后往前, 根据 fa 边
   //插入的字符是 a, 而 s-p 上 5 和 4 节点都没有 a, 因此将节点 5 和 4 fa 节点 6
   //节点 6 此时为 {aabba,abba,bba,ba}
   //当路径上的节点没有 a
   while(p!=-1 && !next[p][c]){
      next[p][c]=cur;
      p=fa[p];
   }
   if(p==-1){
      //对应上面整个路径都没有 a 的情况
      fa[cur]=0;
   }else{
      //路径上找到一个有 a, 往前肯定都有 a
      int q=next[p][c];
      if(len[q] == len[p] + 1){
          //这里节点 S(p) 为空串, 而节点 I(q) 为 \{a\}, 因此将新节点 6 fa 节点 1
          fa[cur]=q;
      }else{
          //st[q].len>st[p].len+1
          //假设当前 sam 为"aab",起点 S 为空串,节点 4 是 {aab,ab,b}
          //此时的 s-p 就是 S-3, 要插入的字符是 b, 路径上 S 节点有 b, 指向节点 3
          //而 st[3].len>st[S].len+1, 因此需要将节点 3 拆分
```

```
//把从节点 S+b 得到的后缀 {b} 分给新的节点 5
             //将 q 拆成两个节点, p->cl->new
             int cl=newnode(len[p]+1,0);
             fa[cl]=fa[q];
             memcpy(next[cl],next[q],sizeof(next[cl]));
             while (p!=-1 \&\& next[p][c]==q)
                 //之前路径上所有 p 走向 q 的, 现在全部走向 q 拆出的新节点
                 next[p][c]=c1;
                 p=fa[p];
             //q 和新节点都 fa 向拆出节点
             fa[q]=fa[cur]=cl;
          }
      }
       //更新最后一个节点
      lst=cur;
   }
}ac;
```

#### 1.11.1 Usage

0 判断模式串是否是原串的子串

从起点 S 按模式串的每个字符进行转移, 无法转移则不是。

1 字符串最小循环移位

对字符串 s+s 建立 sam, 从起点贪心向最小的字符转移。

2 不同子串个数

-(1)-所有的状态节点就保存了所有不同子串,枚举每个状态,计算  $\sum (len[i] - len[fa[i]])$  即可。

推广到长度大于等于 m 的不同子串个数, 答案即  $\sum max(0, len[i] - max(len[fa[i]], m-1))$ 。 每添加一个字符, 所增加的不同子串为 len[lst] - len[fa[lst]]

-(2)-建立 sam 后直接从根节点 (0)dfs 搜索,dp[u] 表示 u 为起点的路径数, $dp[u]+=\sum dp[v]$ ,注意计算过的 dp[v] 不要重复计算,最后答案是 dp[0]-1(或初始化 dp[i] 为 1, dp[0] 为 0)。

dfs 也可以改用拓扑排序, 从后往前递推。

3 不同字串长度之和

即不同路径的长度之和, ans[u] 表示 u 为起点的路径长度和,  $ans[u] = \sum (ans[v] + dp[v])$ ,即 (u,v) 这条边对每条路径都有一个长度字符的贡献。

4 字典序第 k 小子串 (相同子串算 1 个)

从根节点 (0) 往下走,根据求出的 dp[i] 和 k 大小比较,判断走哪一条边,并输出该字符 (k 也要减 1),递归继续判断。

5 出现次数 k 次的不同子串个数。

子串出现的次数即 endpos 的大小, 因此求出 endpos 大小然后枚举所有状态即可。

从 S 开始的反向 fa 连接可以看成一个 parent 树,由 endpos 的性质, $|endpos(u)| = \sum |endpos(v)| + 1/0$ ,是否需要加上 1 取决于该节点对应的 substrings 是否包含原串的某个前缀 (即非分解出来的状态节点 cl)。

拓扑 (桶?) 排序后从后往前推,累加 |endpos|,节点 0 代表空串,|endpos|=0。

6 字典序第 k 小子串 (相同子串算多个)

结合上述第 4 和第 5, 定义 pd[u] 表示节点 u 为起点的子串数 (可相同), 初始化 pd[i] = |endpos(i)|(i>0), 而  $pd[u] + = \sum pd[v]$ 。

求解的时候,找到满足的字符 (pd[v]>=k),直接跳过相同的前缀个数 (k-num[u]),递归边界同样是判断 (k<=num[u])。

#### 1.11.2 Memo

```
//1
//s+=s build...
void solve(int n){
    int p=0;
    for (int i=0;i<n;i++) {</pre>
        auto t=next[p].begin();
        p=t->second;
        printf("%c",t->first+'a');
    }
    printf("\n");
}
//2 3
//dfs(0) dp[0] ans[0]...
void dfs(int u){
    dp[u]=u==0?0:111;
    for(int i=0;i<26;i++){</pre>
        int v=next[u][i];
        if(v){
             if(!dp[v]){
                 dfs(v);
             dp[u] += dp[v];
             ans[u] += ans[v] + dp[v];
        }
    }
}
//5
//topo(len(str)) go() num[i] = |endpos(i)|
void topo(int 1){
    for(int i=0;i<=1;i++){</pre>
        w[i] = 0;
    }
    for(int i=1;i<cnt;i++){</pre>
        w[len[i]]++;
    }
    for(int i=2;i<=1;i++){</pre>
        w[i] += w[i-1];
    for(int i=cnt-1;i>=1;i--){
        tp[w[len[i]]--]=i;
    }
}
void go(){
    for(int i=cnt-1;i>=1;i--){
```

```
num[fa[tp[i]]]+=num[tp[i]];
    }
    //S 状态是空串
    num[0]=0;
}
1/4 6
//get dp[] pd[] solve(0,k) ...
void solve1(int u,int k){
    if(k \le 0) \{ //k \le num[u] 
        return;
    }
    for(int i=0;i<26;i++){
        int v=next[u][i];
        if(v){
            if(dp[v]>=k){\frac{//pd[v]>=k}{}}
                 printf("%c",i+'a');
                 solve1(v,k-1); //solve2(v,k-num[u])
                 break;
            }else{
                 k=dp[v]; //k=pd[v]
        }
    }
}
```

#### 2 ProblemSet

#### 2.1 Trie

#### 2.1.1 区间异或最大值

int insert(int pre,int x){

```
#include <bits/stdc++.h>
using namespace std;
const int N=3e5+50;
int n,q,a[N];
char s[2];
int 1,r,x;
//在序列后添加一个数 询问 [1,r] 中某一个 p 使得 (sum(n)^sum(p-1))^x 最大,也就是 (sum(n)^sum(p-1))^x
//转化为求区间里与 x 异或最大的那个值 类似于线段树求全局 主席树求区间 这里 trie 树求全局
//为了方便求前缀和,设第一个数 a[0] 为 0 而且为了方便建可持久化树 (第一个根 rt[0] 是个空标
int p[N];
int rt[N];
struct Trie{
   int cnt,tr[N*30][2],val[N*30];
   void init(){
      cnt=0;
      memset(tr,0,sizeof(tr));
      memset(val,0,sizeof(val));
```

```
int rt=++cnt;
        int now=rt;
        for(int i=31;i>=0;i--){
            int id=(x>>i)&1;
            if(!tr[now][id]){
                tr[now][id]=++cnt;
                tr[now] [id^1] = tr[pre] [id^1];
                val[tr[now][id]]=val[tr[pre][id]];
            pre=tr[pre][id];
            now=tr[now][id];
            val[now]++;
        }
        return rt;
    }
    int query(int 1,int r,int x){
        int ans=0;
        for(int i=31;i>=0;i--){
            int id=(x>>i)&1;
            if(val[tr[r][id^1]]-val[tr[l][id^1]]){
                ans=ans*2+1;
                l=tr[l][id^1];
                r=tr[r][id^1];
            }else{
                ans=ans*2;
                l=tr[l][id];
                r=tr[r][id];
            }
        }
        return ans;
    }
}T;
int main(void){
    freopen("in.txt","r",stdin);
    scanf("%d%d",&n,&q);
    T.init();
    rt[1]=T.insert(0,0);
    n++;
    for(int i=2;i<=n;i++){</pre>
        scanf("%d",&a[i]);
        p[i]=p[i-1]^a[i];
        rt[i]=T.insert(rt[i-1],p[i]);
    }
    while(q--){
        scanf("%s",s);
        if(s[0]=='A'){
            n++;
            scanf("%d",&a[n]);
            p[n]=p[n-1]^a[n];
            rt[n]=T.insert(rt[n-1],p[n]);
```

```
else if(s[0]=='Q'){
            scanf("%d%d%d", &1, &r, &x);
            printf("d\n",T.query(rt[l-1],rt[r],p[n]^x));
        }
    }
    return 0;
}
      路径异或最大值
2.1.2
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+50;
vector<int> g[N];
int a[N];
int n,m,u,v,x;
int rt[N];
struct Trie{
    int cnt,tr[N*50][2],val[N*50];
    int dep[N],fa[N][20];
    void init(){
        cnt=0;
        memset(tr,0,sizeof(tr));
        memset(val,0,sizeof(val));
        memset(dep,0,sizeof(dep));
        memset(fa,0,sizeof(fa));
    }
    int ins(int pre,int x){
        int rt=++cnt;
        int now=rt;
        for(int i=16;i>=0;i--){
            int id=(x>>i)&1;
            if(!tr[now][id]){
                tr[now][id]=++cnt;
                tr[now] [id^1] = tr[pre] [id^1];
                val[tr[now][id]]=val[tr[pre][id]];
            pre=tr[pre][id];
            now=tr[now][id];
            val[now]++;
        }
        return rt;
    void dfs(int u){
        for(int i=1;(1<<i)<=dep[u];i++){</pre>
            fa[u][i]=fa[fa[u][i-1]][i-1];
        }
        rt[u]=ins(rt[fa[u][0]],a[u]);
        int siz=g[u].size();
        for(int i=0;i<siz;i++){</pre>
```

```
int v=g[u][i];
       if(v==fa[u][0]){
           continue;
       fa[v][0]=u;
       dep[v]=dep[u]+1;
       dfs(v);
   }
}
int lca(int x,int y){
   if(dep[x]<dep[y]){</pre>
       swap(x,y);
   }
   //深度大的 x 先跳
   int t=dep[x]-dep[y];
   for(int i=0;(1<<i)<=t;i++){</pre>
       if(t &(1<<i)){
           x=fa[x][i];
       }
   }
   if(x==y){
       return x;
   }
   //同深度,一起跳
   for(int i=16;i>=0;i--){
       if(fa[x][i]!=fa[y][i]){
           x=fa[x][i];
           y=fa[y][i];
       }
   }
   //fa[x][i]==fa[y][i];
   return fa[x][0];
}
int solve(int u,int v,int x){
   //lca 单独考虑, 因为下面从根到 u 到 v 的路径再减去两倍 lca(必须减去两倍)
   int lc=lca(u,v);
   int ans=a[lc]^x;
   int t=0;
   u=rt[u];
   v=rt[v];
   lc=rt[lc];
   //每一位考虑异或最大
   for(int i=16;i>=0;i--){
       int id=(x>>i)&1;
       //普通可持久化 Trie 只要考虑 val[tr[r][id^1]]-val[tr[l][id^1]] 是否大于 0
       //这里就要考虑 u 到 v 路径上的 val 值是否大于 O (注意负数)
       if(val[tr[u][id^1]]+val[tr[v][id^1]]-2*val[tr[lc][id^1]]>0){
           t+=(1<<ii);
           id^=1;
       }
```

```
u=tr[u][id];
            v=tr[v][id];
            lc=tr[lc][id];
        }
        //printf("t %d\n",t);
        return max(ans,t);
    }
}T;
int main(void){
    // freopen("in.txt", "r", stdin);
    while (\simscanf("%d%d",&n,&m)){
        for(int i=1;i<=n;i++){</pre>
            g[i].clear();
            scanf("%d",&a[i]);
        }
        memset(rt,0,sizeof(rt));
        for(int i=0;i<n-1;i++){</pre>
            scanf("%d%d",&u,&v);
            g[u].push_back(v);
            g[v].push_back(u);
        }
        T.init();
        T.dfs(1);
        //printf("%d %d\n", T.lca(1,2), T.lca(2,3));
        //printf("%d %d %d\n", T. dep[1], T. dep[2], T. dep[3]);
        for(int i=0;i<m;i++){</pre>
            scanf("%d%d%d",&u,&v,&x);
            int ans=T.solve(u,v,x);
            printf("%d\n",ans);
        }
    }
    return 0;
}
2.1.3 子树异或最大值
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+50;
int n,q,a[N],f,u,x;
vector<int> g[N];
int cnt,tr[N*40][2],val[N*40];
int rt[N];
int insert(int pre,int x){
    int rt=++cnt;
    int now=rt;
    for(int i=31;i>=0;i--){
        int id=(x>>i)&1;
        if(!tr[now][id]){
            tr[now][id]=++cnt;
```

```
tr[now] [id^1] = tr[pre] [id^1];
            val[tr[now][id]]=val[tr[pre][id]];
        }
        pre=tr[pre][id];
        now=tr[now][id];
        val[now]++;
    }
    return rt;
}
int query(int 1,int r,int x){
    int ans=0;
    for(int i=31;i>=0;i--){
        int id=(x>>i)&1;
        if(val[tr[r][id^1]]-val[tr[l][id^1]]>0){
            l=tr[l][id^1];
            r=tr[r][id^1];
            ans=ans*2+1;
        }else{
            l=tr[l][id];
            r=tr[r][id];
            ans=ans*2;
        }
    }
    return ans;
}
int idx,in[N],ot[N],mp[N];
void dfs(int u,int f){
    in[u]=++idx;
    mp[idx]=u;
    int siz=g[u].size();
    for(int i=0;i<siz;i++){</pre>
        int v=g[u][i];
        if(v==f){
            continue;
        }
        dfs(v,u);
    }
    ot[u]=idx;
}
void init(){
    cnt=0;
    memset(tr,0,sizeof(tr));
    memset(val,0,sizeof(val));
    memset(rt,0,sizeof(rt));
    idx=0;
    memset(in,0,sizeof(in));
    memset(ot,0,sizeof(ot));
}
int main(void){
    // freopen("in.txt", "r", stdin);
```

```
while (-\operatorname{scanf}("%d%d", &n, &q)){
        init();
        for(int i=1;i<=n;i++){</pre>
            g[i].clear();
             scanf("%d",&a[i]);
        }
        for(int i=2;i<=n;i++){</pre>
            scanf("%d",&f);
            g[f].push back(i);
            g[i].push back(f);
        }
        dfs(1,0);
        for(int i=1;i<=n;i++){</pre>
            rt[i]=insert(rt[i-1],a[mp[i]]);
        }
        while(q--){
             scanf("%d%d",&u,&x);
             int ans=query(rt[in[u]-1],rt[ot[u]],x);
            printf("%d\n",ans);
        }
    return 0;
}
2.1.4 编辑距离
#include <bits/stdc++.h>
using namespace std;
const int N=2e5+50;
int ans,len;
bool yes, vis[N];
char s[40];
//编辑距离三种操作
//delete 索引 +1 节点不变
                                (del s[idx+1])
//add
           索引不变 节点 nex
                                (add in s[idx] idx \rightarrow idx + 1)
//replace 索引 +1 节点 nex
struct Trie{
    int cnt,tr[N][26],val[N];
    void init(){
        cnt=0;
        memset(tr,0,sizeof(tr));
        memset(val,0,sizeof(val));
    void insert(char *s){
        int len=strlen(s);
        int now=0;
        for(int i=0;i<len;i++){</pre>
             int id=s[i]-'a';
             if(!tr[now][id]){
                 tr[now][id]=++cnt;
```

```
}
       now=tr[now][id];
   }
   val[now]=1;
}
int query(){
   len=strlen(s+1);
   ans=0;
   yes=false;
   memset(vis,false,sizeof(vis));
   dfs(0,0,0);
   if(yes){
       return -1;
   }else{
       return ans;
   }
}
//trie 树根节点是 0, 所以考虑将字符串从 1 计数
//字符串下标, trie 数节点, 是否修改 (编辑距离为 1)
void dfs(int idx,int u,int f){
   if(idx==len){
        if(val[u]){
           if(f){
               if(!vis[u]){
                   ans++;
                   vis[u]=true;
               }
           }else{
               yes=true;
           return;
       }
   }
   if(!f){
       //delete 删除 s[idx+1]
       dfs(idx+1,u,1);
       for(int i=0;i<26;i++){</pre>
           if(tr[u][i]){
               //add 在 idx 前添加'a'+i
               dfs(idx,tr[u][i],1);
               if(i!=s[idx+1]-'a'){
                   //replace 将 s[idx+1] 换成'a'+i
                   dfs(idx+1,tr[u][i],1);
               }
           }
       }
   }
   //无操作
   int id=s[idx+1]-'a';
   if(tr[u][id]){
```

```
dfs(idx+1,tr[u][id],f);
        }
    }
}T;
int n,m;
int main(void){
    //freopen("in.txt", "r", stdin);
    scanf("%d%d",&n,&m);
    T.init();
    for(int i=0;i<n;i++){</pre>
        scanf("%s",s);
        T.insert(s);
    }
    for(int i=0;i<m;i++){</pre>
        scanf("%s",s+1);
        printf("%d\n",T.query());
    }
    return 0;
}
2.1.5 trie 树上 dp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=3e5+50;
const 11 mod=20071027;
char s[N];
int n;
int tr[N*2][26],cnt,val[N*2];
11 dp[N*2];
char w[105];
void init(){
    memset(tr,0,sizeof(tr));
    cnt=0;
    memset(val,0,sizeof(val));
    memset(dp,0,sizeof(dp));
}
void insert(char *s){
    int len=strlen(s);
    int now=0;
    for(int i=0;i<len;i++){</pre>
        int id=s[i]-'a';
        if(!tr[now][id]){
            tr[now][id]=++cnt;
        }
        now=tr[now][id];
    }
    val[now]++;
}
```

```
void query(char *s,int len,int x){
    int now=0;
    //因为包含一个已确定字符 s[0], 所以是 <=len
    for(int i=0;i<=len;i++){</pre>
        int id=s[i]-'a';
       if(!tr[now][id]){
           break:
       }else if(val[tr[now][id]]){
           dp[x]=(dp[x]+dp[x+i+1])\%mod;
       }
       now=tr[now][id];
    }
}
int main(void){
    // freopen("in.txt", "r", stdin);
    int cas=1;
    while(~scanf("%s",s)){
        init();
       scanf("%d",&n);
       int len=strlen(s);
       for(int i=0;i<n;i++){</pre>
            scanf("%s",w);
            insert(w);
       //dp[i] 表示 s[i...len-1] 这个串由单词表示的方案数
        //dp[i]+=dp[j] s[i...j-1] 是单词
       dp[len]=1;
       for(int i=len-1;i>=0;i--){
            //所以这里要查询以 s[i] 开头,长度为 min(len-i,100) 的单词数,累加其方案数
           query(s+i,min(100,len-i+1),i);
       }
       printf("Case %d: %lld\n", cas++, (dp[0] %mod+mod) %mod);
    }
    return 0;
}
2.2
     Pam
2.2.1 双向插入
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=2e5+50;
struct PT{
    int next[N][26],fail[N];
    int cnt[N],num[N],len[N];
    int S[N*2],last[2],L,R,id[N],n,p;
   ll ans;
    int newnode(int 1){
       for(int i=0;i<26;i++){
```

```
next[p][i]=0;
   }
   cnt[p]=0;
   num[p]=0;
   len[p]=1;
   return p++;
void init(int allLen){
   ans=0;
   p=0;
   newnode(0);
   newnode(-1);
   fail[0]=1;
   //两个 last 分别维护前端和后端插入
   last[0]=last[1]=0;
   //普通的后端插入是 n 从 o 开始, 然后 S[++n]
   //这里分为前后端插入,将 S 扩大两倍,后端插入的放在 S[allLen...],前端插入的放在
   //因为添加的时候是 S[++R] 和 S[--L], 所以 L 初值为 allLen, R 初值为 allLen-1
   L=allLen;
   R=allLen-1;
   memset(S,-1,sizeof(S));
}
int getFail(int x,int d){
   if(d){
       //后端添加
       while (S[R-len[x]-1]!=S[R]) {
           x=fail[x];
       }
   }else{
       //前端添加
       while (S[L+len[x]+1]!=S[L]) {
           x=fail[x];
       }
   }
   return x;
}
void add(int c,int d){
   c-='a';
   if(d){
       S[++R]=c;
   }else{
       S[--L]=c;
   }
   int cur=getFail(last[d],d);
   if(!next[cur][c]){
       int now=newnode(len[cur]+2);
       fail[now] = next[getFail(fail[cur],d)][c];
       num[now] = num[fail[now]]+1;
       next[cur][c]=now;
   }
```

```
last[d]=next[cur][c];
        cnt[last[d]]++;
        //添加字符之后当前整个串为回文, 修改另一个 last
        if(len[last[d]] == R-L+1){
            last[d^1] = last[d];
        }
        ans+=111*num[last[d]];
    }
    void count(){
        for(int i=p-1;i>=0;i--){
            cnt[fail[i]]+=cnt[i];
        }
    }
}ac;
int n,op;
char s[3];
int main(){
      freopen("in.txt", "r", stdin);
    while(~scanf("%d",&n)){
        ac.init(n);
        for(int i=0;i<n;i++){</pre>
            scanf("%d", &op);
            if(op==1){
                scanf("%s",s);
                ac.add(s[0],0);
            }else if(op==2){
                scanf("%s",s);
                ac.add(s[0],1);
            }else if(op==3){
                //不同回文子串个数
                printf("%d\n",ac.p-2);
            }else{
                //回文子串个数
                printf("%lld\n",ac.ans);
            }
        }
    }
    return 0;
}
2.2.2 next 和 fail 统计贡献
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=1e5+50;
int vis[N],ndp[N],fdp[N];
struct PT{
    int next[N][26],fail[N],cnt[N],num[N],len[N];
    int S[N],last,id[N],n,p;
```

```
int newnode(int 1){
    for(int i=0;i<26;i++){</pre>
        next[p][i]=0;
    }
    cnt[p]=num[p]=0;
    len[p]=1;
    return p++;
}
void init(){
    p=0;
    newnode(0);
    newnode(-1);
    last=0;
    n=0;
    S[n]=-1;
    fail[0]=1;
}
int getFail(int x){
    while (S[n-len[x]-1]!=S[n]) {
        x=fail[x];
    }
    return x;
}
void add(int c){
    c-='a';
    S[++n]=c;
    int cur=getFail(last);
    if(!next[cur][c]){
        int now=newnode(len[cur]+2);
        fail[now] = next[getFail(fail[cur])][c];
        num[now] = num[fail[now]]+1;
        next[cur][c]=now;
    }
    last=next[cur][c];
    cnt[last]++;
    id[last]=n;
}
void count(){
    for(int i=p-1;i>=0;i--){
        cnt[fail[i]]+=cnt[i];
    }
}
int dfs(int u){
    ndp[u]=1;
    fdp[u]=0;
    //计算向上跳的 fail 指针次数, vis 保证不重复 (比如 bb 跳的 fail 指针, bbbb 不能再
    for(int t=u;!vis[t] && t>1;t=fail[t]){
        vis[t]=u;
        fdp[u]++;
    }
```

```
for(int i=0;i<26;i++){</pre>
           if(next[u][i]){
               ndp[u]+=dfs(next[u][i]);
           }
       }
       //清空标记
       for(int t=u; vis[t] == u && t>1; t=fail[t]){
           vis[t]=0;
       return ndp[u];
    }
   11 solve(){
       //从两个根 dfs
       dfs(0);
       dfs(1);
       ll ans=0;
       for(int i=2;i<p;i++){</pre>
           //除去根,每个节点的贡献 (作为另一个回文子串的子串) 为
           //比如对于样例 abba, 回文节点 bb 的 next 指针指向 abba, fail 指针指向 b
           //因此 ndp 和 fdp 都为 2, 贡献为 2*2-1=3
           //即 (b,bb) (b,abba) (bb,bb) (bb,abba), 减 1 就是要减掉本身
           ans+=111*ndp[i]*fdp[i]-1;
       }
       return ans;
   };
}ac;
int T;
char s[N];
int main(void){
    // freopen("in.txt", "r", stdin);
    scanf("%d",&T);
    for(int cas=1;cas<=T;cas++){</pre>
       scanf("%s",s);
       ac.init();
       int len=strlen(s);
       for(int i=0;i<len;i++){</pre>
           ac.add(s[i]);
       }
       ac.count();
       memset(vis,0,sizeof(vis));
       printf("Case #%d: %lld\n",cas,ac.solve());
    }
   return 0;
}
2.3
     Sa
2.3.1 可重第 k 小子串
#include <bits/stdc++.h>
using namespace std;
```

```
const int N=1e5+50;
char s[N];
int k;
int sa[N],rk[N],h[N];
int t[N],t2[N],c[N];
void build(int n,int m=128){
   //后缀数组
}
//a[i] 记录第 i 个后缀目前枚举到第几个子串
int a[N];
void solve(int n){
   memset(a, 0, sizeof(a));
   //从排名第一的后缀开始
   int r=1;
   //复杂度不会超过 K
   while(k){
       a[r]++;
       //大于后缀的长度, 下一个后缀
       if(a[r]>n-sa[r]){
           r++;
           continue;
       }
       k--;
       //枚举 r 之后的所有后缀,h[i] > = a[r] 即扫过所有相同子串 (当前 a[r] 长度)
       for(int i=r+1;i<=n && h[i]>=a[r] && k;i++){
           a[i]++;
           k--;
       }
   }
   //第 k 小子串在第 r 个后缀里,长度为 a[r]
   for(int i=0;i<a[r];i++){</pre>
       printf("%c",s[sa[r]+i]);
   }
   printf("\n");
}
int main(void){
   scanf("%s",s);
   scanf("%d",&k);
   int n=strlen(s);
   build(n);
   if(k>111*n*(n+1)/2){
       printf("No such line.\n");
       return 0;
   }
   solve(n);
   return 0;
}
```

#### 2.3.2 单调栈 + 边界

```
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=1e5+50;
char str[N];
int s[N],n;
int sa[N],rk[N],h[N];
int t[N],t2[N],c[N];
int st[N][25];
int le[N],ri[N];
void build(int n,int m){
    //后缀数组
}
void init_rmq(){
    for(int i=0;i<=n;i++){</pre>
        st[i][0]=h[i];
    }
    for(int j=1;(1<<j)<=n;j++){
        for(int i=0;i+(1<< j)-1<=n;i++){
            st[i][j]=min(st[i][j-1],st[i+(1<<(j-1))][j-1]);
        }
    }
}
int rmq(int 1,int r){
    int k=0;
    while ((1 << (k+1)) <= r-1+1) {
    }
    return min(st[l][k],st[r-(1<<k)+1][k]);
}
int lcp(int l,int r){
    // printf("rk %d %d\n",l,r);
    if(l==r){
        return n-sa[1];
    }
    if(1>r){
        swap(1,r);
    }
    return rmq(l+1,r);
}
vector<pair<int,int> > ans;
void solve(){
    stack<int> sta;
    while(!sta.empty()){
        sta.pop();
    for(int i=1;i<=n;i++){</pre>
        while(!sta.empty() && h[i]<=h[sta.top()]){</pre>
            sta.pop();
```

```
}
        if(!sta.empty()){
            le[i]=sta.top()+1;
        }else{
            le[i]=1;
        }
        sta.push(i);
    }
    while(!sta.empty()){
        sta.pop();
    }
    for(int i=n;i>=1;i--){
        while(!sta.empty() && h[i] <= h[sta.top()]){</pre>
            sta.pop();
        }
        if(!sta.empty()){
            ri[i]=sta.top()-1;
        }else{
            ri[i]=n;
        }
        sta.push(i);
    }
    //枚举后缀长度
    int L,R;
    for(int len=1;len<n;len++){</pre>
        if(lcp(rk[0],rk[n-len])==len){}
            //即 h[rk[n-len]+1]...h[rk[0]] 的最小值为 len
            R=ri[rk[n-len]+1];
            if(h[rk[n-len]]<len){</pre>
                //必须有效 (按 h 分组后) 的后缀才能以最小值延伸,比如 len 为 3,但是 h[rl
                L=rk[n-len];
            }else{
                L=le[rk[n-len]];
            ans.push_back({len,R-L+1});
        }
    }
    //母串单独考虑
    ans.push_back(\{n,1\});
    int siz=ans.size();
    printf("%d\n",siz);
    for(int i=0;i<siz;i++){</pre>
        printf("%d %d\n",ans[i].first,ans[i].second);
    }
}
int main(){
    freopen("in.txt","r",stdin);
    scanf("%s",str);
    n=strlen(str);
    for(int i=0;i<n;i++){</pre>
```

```
s[i]=str[i]-'A'+1;
   }
   build(n,256);
   debug();
   init_rmq();
   solve();
   return 0;
}
2.3.3 单调栈
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
typedef long long ll;
const int N=3e5+50;
char a[N],b[N],s[N];
int sa[N],rk[N],h[N];
int t[N],t2[N],c[N];
int al,bl,n,k;
void build(int n,int m){
   //后缀数组
}
//答案就是对任意两个不同后缀 a[i...] 和 b[j...] 的 sum(lcp(ai,bj)-k+1)
I/两个单调栈, 一个维护 <math>h[i], 一个维护贡献之和
11 he[N],ct[N];
11 solve(){
   //考虑用单调栈优化到 O(n), 即对于每一个后缀求与前面后缀的 lcp 之和, 不重不漏
   ll ans=0;
   //当前后缀与前面每个后缀的 lcp 之和
   //由性质可知, 当前后缀和前面某一个后缀的 lcp 应该是之间的 h[i] 最小值
   //因此可以将递减的 h[i] 合并为最小的那个 h[min]*cnt
   ll sum=0;
   int tp=0;
   for(int i=2;i<=n;i++){</pre>
       if(h[i]<k){
          tp=0;
          sum=0;
          continue;
       }
       11 cnt=0;
       //维护单调栈, 由于 lcp 只跟区间 h 最小值有关, 将所有栈顶大于当前 h[i] 的都合并
       while(tp && he[tp]>h[i]){
          //减去无效栈顶的贡献 (h[i]-k+1)
          sum-=(he[tp]-k+1)*ct[tp];
          //暂时累计 cnt, 存储到新的栈顶
          cnt+=ct[tp];
          //栈顶出栈
          tp--;
```

```
}
       //入栈, 保持单调性
       he[++tp]=h[i];
       if(sa[i-1]<al) {
           //有效贡献的串,个数加 1
           cnt++;
       }
       ct[tp]=cnt;
       //累加栈顶贡献
       sum+=(he[tp]-k+1)*ct[tp];
       if(sa[i]>al){
           //将当前累加的贡献加到答案中, 即 b 串后缀与前面所有 a 串后缀的 lcp 之和
           ans+=sum;
       }
    }
    tp=sum=0;
    for(int i=2;i<=n;i++){</pre>
       if(h[i]<k){
           tp=0;
           sum=0;
           continue;
       }
       11 cnt=0;
       while(tp && he[tp]>h[i]){
           sum-=(he[tp]-k+1)*ct[tp];
           cnt+=ct[tp];
           tp--;
       }
       if(sa[i-1]>al){
           he[++tp]=h[i];
           ct[tp]=cnt+1;
           sum+=(he[tp]-k+1)*ct[tp];
       }else{
           he[++tp]=h[i];
           ct[tp]=cnt;
           sum+=(he[tp]-k+1)*ct[tp];
       }
       //累加 b 串后缀与前面所有 a 串后缀的 lcp 之和
       if(sa[i] < al) {</pre>
           ans+=sum;
       }
    }
   return ans;
}
int main(){
      freopen("in.txt", "r", stdin);
   while(-scanf("%d", &k) && k){
       scanf("%s",a);
       scanf("%s",b);
       al=strlen(a);
```

```
bl=strlen(b);
        for(int i=0;i<al;i++){</pre>
            s[i]=a[i];
        }
        s[al]='~';
        for(int i=0;i<bl;i++){</pre>
            s[al+1+i]=b[i];
        }
        n=al+bl+1;
        s[n]='\setminus 0';
        build(n,300);
        11 ans=solve();
        printf("%lld\n",ans);
    }
    return 0;
}
2.4
     Sam
2.4.1
     出现 k 次子串
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=2e6+50;
struct SAM{
    struct state{
        int len,fa;
        //状态对应 endpos 大小,即子串出现次数
        int siz;
        int next[26];
    }st[N*2];
    //后缀自动机
    void topo(int len){
        //按 len 从小到大排序 O(n)
        for(int i=0;i<=len;i++){</pre>
            ws[i]=0;
        }
        for(int i=1;i<cnt;i++){</pre>
            ws[st[i].len]++;
        }
        for(int i=1;i<=len;i++){</pre>
            ws[i] += ws[i-1];
        }
        for(int i=cnt-1;i>=1;i--){
            tp[ws[st[i].len]--]=i;
        //从叶子节点递推到 S,累加 siz,得到 endpos 的大小
        for(int i=cnt-1;i>0;i--){
            printf("%d add %d\n",st[tp[i]].fa,tp[i]);
            st[st[tp[i]].fa].siz+=st[tp[i]].siz;
```

```
}
        for(int i=0;i<cnt;i++){</pre>
           printf("%d %d %d\n",i,tp[i],st[i].siz);
        }
    }
    11 solve(int k){
       ll ans=0;
        for(int i=1;i<cnt;i++){</pre>
            //siz 表示状态的 endpos 大小,也就是子串集合出现的次数
            if(st[i].siz==k){
                //st[i].len-st[st[i].fa].len 表示该状态对应的子串数
                ans+=st[i].len-st[st[i].fa].len;
            }
        }
        return ans;
    }
}ac;
char s[N];
int T,k;
int main(void){
   freopen("in.txt","r",stdin);
    scanf("%d",&T);
    while(T--){
        scanf("%d",&k);
        scanf("%s",s);
        int n=strlen(s);
        ac.init();
        for(int i=0;i<n;i++){</pre>
            ac.add(s[i]);
        }
        ac.topo(n);
        11 ans=ac.solve(k);
       printf("%lld\n",ans);
    }
    return 0;
}
2.4.2 不同子串个数
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=2e5+50;
struct SAM{
    //后缀自动机
    //dp[u] 表示以 u 为起点的路径总数
    11 dp[N*2];
    void dfs(int u){
        dp[u]=u==0?0:111;
        for(int i=0;i<26;i++){</pre>
```

```
int v=next[u][i];
            if(v){
                if(!dp[v]){
                    dfs(v);
                dp[u] += dp[v];
            }
        }
    }
    11 solve(){
        dfs(0);
        return dp[0];
    }
}ac;
int n;
char s[N];
int main(){
      freopen("in.txt", "r", stdin);
    scanf("%d",&n);
    scanf("%s",s);
    ac.init();
    for(int i=0;i<n;i++){</pre>
        ac.add(s[i]);
    }
    11 ans=ac.solve();
    printf("%lld\n",ans);
    return 0;
}
      长度大于等于 m 不同子串个数
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=2e6+50;
struct SAM{
    struct state{
        int len,fa,next[26];
    }st[N*2];
    int cnt,lst;
    void init(){
        st[0]={0,-1};
        memset(st[0].next,0,sizeof(st[0].next));
        cnt++;
        lst=0;
    }
    void add(int c){
        c-='a';
        int cur=cnt++;
        int p=lst;
```

```
st[cur].len=st[p].len+1;
        while(p!=-1 && !st[p].next[c]){
            st[p].next[c]=cur;
            p=st[p].fa;
        }
        if(p==-1){
            st[cur].fa=0;
        }else{
            int q=st[p].next[c];
            if(st[q].len==st[p].len+1){
                st[cur].fa=q;
            }else{
                int cl=cnt++;
                st[cl]={st[p].len+1,st[q].fa};
                memcpy(st[cl].next,st[q].next,sizeof(st[c].next));
                while (p!=-1 \&\& st[p].next[c]==q){
                     st[p].next[c]=cl;
                     p=st[p].fa;
                st[q].fa=st[cur].fa=cl;
            }
        }
        lst=cur;
    }
    11 solve(int m){
        ll ans=0;
        for(int i=1;i<cnt;i++){</pre>
            ans+=(\max(0, st[i].len-\max(st[st[i].fa].len, m-1)));
        return ans;
    }
}ac;
int n,m;
char s[N];
int main(){
    scanf("%d%d",&n,&m);
    scanf("%s",s);
    ac.init();
    for(int i=0;i<n;i++){</pre>
        ac.add(s[i]);
    11 ans=ac.solve(m);
    printf("%lld\n",ans);
    return 0;
}
2.4.4 字典序第 k 小子串
#include <bits/stdc++.h>
using namespace std;
```

```
typedef long long 11;
const int N=2e6+50;
struct SAM{
    int next[N*2][26];
    int fa[N*2],len[N*2],num[N*2],cnt,lst;
    int newnode(int 1,int s){
        memset(next[cnt],0,sizeof(next[cnt]));
        len[cnt]=1;
        num[cnt]=s;
        return cnt++;
    }
    void init(){
        cnt=0;
        lst=newnode(0,0);
        fa[lst]=-1;
    }
    void add(int c){
        c-='a';
        int p=lst;
        int cur=newnode(len[p]+1,1);
        while(p!=-1 && !next[p][c]){
            next[p][c]=cur;
            p=fa[p];
        if(p==-1){
            fa[cur]=0;
        }else{
            int q=next[p][c];
            if(len[q] == len[p]+1){
                 fa[cur]=q;
            }else{
                 int cl=newnode(len[p]+1,0);
                 fa[cl]=fa[q];
                 memcpy(next[cl],next[q],sizeof(next[cl]));
                 while (p!=-1 \&\& next[p][c]==q){
                     next[p][c]=cl;
                     p=fa[p];
                 fa[q]=fa[cur]=cl;
            }
        }
        lst=cur;
    }
    11 dp[N*2],pd[N*2];
    int w[N],tp[N];
    void topo(int 1){
        for(int i=0;i<=1;i++){</pre>
            w[i] = 0;
        for(int i=1;i<cnt;i++){</pre>
```

```
w[len[i]]++;
    }
    for(int i=2;i<=1;i++){</pre>
        w[i] += w[i-1];
    }
    for(int i=cnt-1;i>=1;i--){
        tp[w[len[i]]--]=i;
    }
}
void go(){
    for(int i=cnt-1;i>=1;i--){
        num[fa[tp[i]]]+=num[tp[i]];
    }
    //S 状态是空串
    num[0]=0;
}
void dfs(int u){
    dp[u]=u==0?0:111;
    pd[u]=u==0?0:111*num[u];
    for(int i=0;i<26;i++){</pre>
        int v=next[u][i];
        if(v){
            if(!dp[v]){
                dfs(v);
            }
            dp[u] += dp[v];
            pd[u]+=pd[v];
        }
    }
}
//在以 u 节点开始的路径中查找第 k 小
void solve1(int u,int k){
    if(k<=0){
        return;
    for(int i=0;i<26;i++){</pre>
        int v=next[u][i];
        if(v){
            if(dp[v]>=k){
                printf("%c",i+'a');
                solve1(v,k-1);
                break;
            }else{
                k-=dp[v];
            }
        }
    }
}
void solve2(int u,int k){
    if(k<=num[u]){
```

```
return;
        }
        for(int i=0;i<26;i++){</pre>
             int v=next[u][i];
             if(v){
                 if(pd[v]>=k-num[u]){
                      printf("%c",i+'a');
                      solve2(v,k-num[u]);
                      break;
                 }else{
                      k-=pd[v];
                 }
             }
        }
    }
    void debug(){
        for(int i=0;i<cnt;i++){</pre>
             printf("%d %d %lld %lld\n",i,num[i],dp[i],pd[i]);
             for(int j=0; j<26; j++){</pre>
                 if(next[i][j]){
                      printf("%c %d\n", 'a'+j, next[i][j]);
                 }
             }
        }
        for(int i=1;i<=dp[0];i++){</pre>
             printf("%d: ",i);
             solve1(0,i);
             printf("\n");
        }
        for(int i=1;i<=pd[0]-num[0];i++){</pre>
             printf("%d: ",i);
             solve2(0,i);
             printf("\n");
        }
    }
}ac;
int q;
11 k;
char s[N];
int main(){
      freopen("in.txt", "r", stdin);
    scanf("%s",s);
    int n=strlen(s);
    ac.init();
    for(int i=0;i<n;i++){</pre>
        ac.add(s[i]);
    ac.topo(n);
    ac.go();
    ac.dfs(0);
```

```
scanf("%d%1ld",&q,&k);
    if(q){
        //相同子串算多个
        if(k>ac.pd[0]){
            printf("-1\n");
            return 0;
        }
        ac.solve2(0,k);
        printf("\n");
    }else{
        //相同子串算一个
        if(k>ac.dp[0]){
            printf("-1\n");
            return 0;
        }
        ac.solve1(0,k);
        printf("\n");
    }
    return 0;
}
      循环字典序第 k 小
2.4.5
#include <bits/stdc++.h>
using namespace std;
const int N=6e5+05;
struct SAM{
   map<long long,int> next[N*2];
    //后缀自动机
    void solve(int n){
        //贪心找最小的转移边
        int p=0;
        for (int i=0;i<n;i++) {</pre>
            auto t=next[p].begin();
            p=t->second;
            printf("%d ",t->first);
        printf("\n");
    }
}ac;
int n;
long long a[N/2];
int main(){
    ac.init();
    scanf("%d",&n);
    for(int i=1;i<=n;i++){</pre>
        scanf("%lld",&a[i]);
    }
    for(int i=1;i<=2*n;i++){</pre>
        ac.add(a[(i-1)%n+1]);
```

```
}
    ac.solve(n);
    return 0;
}
      其他
2.5
2.5.1 k 短路
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=5e4+50;
struct Edge{
    int v;
    ll w;
    bool operator <(const Edge &rhs)const{</pre>
        return w<rhs.w;
    }
};
vector<Edge> g[N];
int T,n,m,q,k,u,v,w;
struct node{
    int u,id;
    11 w;
    bool operator <(const node &rhs)const{</pre>
        return w>rhs.w;
    }
};
int que[N];
ll ans[N];
priority_queue<node> pq;
int main(){
      freopen("in.txt", "r", stdin);
    scanf("%d",&T);
    while(T--){
        scanf("%d%d%d",&n,&m,&q);
        for(int i=1;i<=n;i++){</pre>
            g[i].clear();
        }
        for(int i=1;i<=m;i++){</pre>
             scanf("%d%d%d",&u,&v,&w);
            g[u].push_back(Edge{v,111*w});
        }
        int mx=0;
        for(int i=1;i<=q;i++){</pre>
            scanf("%d",&que[i]);
            mx=max(mx,que[i]);
        }
        while(!pq.empty()){
            pq.pop();
```

```
}
        for(int i=1;i<=n;i++){</pre>
            sort(g[i].begin(),g[i].end());
            if(g[i].size()>0){
                pq.push(node{i,0,g[i][0].w});
            }
        }
        int kk=0;
        //取一条加两条
        while(!pq.empty()){
            node tmp=pq.top();
           pq.pop();
           kk++;
            ans[kk]=tmp.w;
             * if(kk==que[i].k){
                    i++;
                    if(i>q){
                        break;
             * 这种写法是错的,如果有多个相同的 k,只会计算第一个的答案,后面死循环
             */
            if(kk==mx){
                break;
            int u=tmp.u;
            int id=tmp.id;
            int v=g[u][id].v;
            11 w=tmp.w;
            if(g[u].size()>id+1){
                pq.push(node{u,id+1,w-g[u][id].w+g[u][id+1].w});
            }
            if(g[v].size()>0){
                pq.push(node\{v,0,w+g[v][0].w\});
            }
        }
        for(int i=1;i<=q;i++){</pre>
           printf("%lld\n", ans[que[i]]);
        }
    }
    return 0;
}
2.5.2 k 小团
#include <bits/stdc++.h>
using namespace std;
const int N=105;
typedef long long 11;
```

```
typedef bitset<N> bs;
int n,k;
ll a[N];
char s[N];
struct node{
    //表示从 id 开始往下找的团, 为了保证不重复
    int id;
   ll val;
    //当前团的状态
   bs sta;
   bool operator <(const node& rhs)const{</pre>
        return val>rhs.val;
    }
};
bs g[N];
bs t;
priority_queue<node> pq;
int main(void){
    // freopen("in.txt","r",stdin);
    scanf("%d%d",&n,&k);
    for(int i=0;i<n;i++){</pre>
        scanf("%lld",&a[i]);
    }
    for(int i=0;i<n;i++){</pre>
        scanf("%s",s);
        for(int j=0; j<n; j++){</pre>
            g[i][j]=s[j]-'0';
        }
    }
    if(k==1){
        printf("0\n");
        return 0;
    }
   k--;
    for(int i=0;i<n;i++){</pre>
        t.reset();
        t[i]=1;
        pq.push(node{i,a[i],t});
    }
    while(!pq.empty()){
        node tmp=pq.top();
        pq.pop();
        if(k==1){
            printf("%lld\n",tmp.val);
            return 0;
        }
        k--;
        for(int i=tmp.id+1;i<n;i++){</pre>
            //如果一个点所连的所有点 (g[i] 刚好是当前团的所有点 sta, 那么加入 i 后仍然是图
            if((g[i]&tmp.sta)==tmp.sta){
```

```
bs now=tmp.sta;
                 now[i]=1;
                 pq.push(node{i,tmp.val+a[i],now});
             }
        }
    }
    printf("-1\n");
    return 0;
}
2.5.3 floyd 最小环
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=1e5+50;
const int INF=1e8+50;
int n;
11 x,a[N];
int bc[60];
int g[200][200],dis[200][200];
int main(){
      freopen("in.txt", "r", stdin);
    scanf("%d",&n);
    int k=0;
    for(int i=1;i<=n;i++){</pre>
        scanf("%lld",&x);
        if(x==0){
             continue;
        }
        a[++k]=x;
        for(int j=0; j<60; j++){</pre>
             if((x>>j)&1){
                 bc[j]++;
             }
        }
    for(int i=0;i<60;i++){</pre>
        if(bc[i]>=3){
             printf("3\n");
             return 0;
        }
    }
    //all bc[i]<=2
    for(int i=1;i<=k;i++){</pre>
        for(int j=1; j<=k; j++){</pre>
             dis[i][j]=i==j?0:INF;
             g[i][j]=i==j?0:INF;
        }
    }
```

```
for(int i=1;i<=k;i++){</pre>
        for(int j=i+1; j<=k; j++){</pre>
            if((a[i]&a[j])!=0){
                dis[i][j]=dis[j][i]=1;
                g[i][j]=g[j][i]=1;
            }
        }
    }
    int ans=INF;
    for(int p=1;p<=k;p++){</pre>
        for(int i=1;i<p;i++){</pre>
            for(int j=i+1; j<p; j++){</pre>
                //g 是原图的边,dis 是松弛后的边,且中间点是 O 到 k-1,因此形成环
                ans=min(ans,g[i][p]+g[p][j]+dis[i][j]);
            }
        }
        for(int i=1;i<=k;i++){</pre>
            for(int j=1; j<=k; j++){</pre>
                dis[i][j]=min(dis[i][j],dis[i][p]+dis[p][j]);
            }
        }
    }
    if(ans==INF){
        //no loop
        ans=-1;
    }
    printf("%d\n",ans);
    return 0;
}
2.5.4 dijk 费用流 +vector 建图
#include <bits/stdc++.h>
using namespace std;
const int N=5e6+50;
const int INF=0x3f3f3f3f;
int T,n,k,w[N],nc,s,t;
struct edge {
    int to, capacity, cost, rev;
};
int h[N],dis[N],PreV[N],p[N],a[N];
//vector < edge > G[N + 5];
//void init() {
      for (int i = 0; i < N; ++i)G[i].clear();
//
//}
//void add(int from, int to, int cap, int cost) {
      G[from].push_back({to, cap, cost, int(G[to].size())});
//
      G[to].push\ back(\{from, 0, -cost, int(G[from].size() - 1)\});
//}
struct Edge{
```

```
int u,v,w,c,next;
}e[N];
int cnt,head[N];
void init(){
    cnt=0;
   memset(head,-1,sizeof(head));
}
void add(int u,int v,int w,int c){
    e[cnt]=Edge{u,v,w,c,head[u]};
   head[u]=cnt++;
    e[cnt]=Edge{v,u,0,-c,head[v]};
   head[v]=cnt++;
}
struct node{
    int v,c;
    bool operator <(const node &rhs)const{</pre>
        return c>rhs.c;
    }
};
void mcmf(int &flow, int &cost) {
    flow=0;
    cost=0;
    //初始化势函数,也可以跑一遍 spfa 计算出 dis, 赋值给 h
    for(int i=1;i<=nc;i++){</pre>
        h[i]=0;
    }
    while(true){
        //重复跑 dijk 堆优化
        //跑费用的最短路,同时记录路径的最小流量
        priority_queue<node> q;
        for(int i=1;i<=nc;i++){</pre>
            dis[i]=INF;
        }
        dis[s]=0;
        p[s]=0;
        a[s]=INF;
        q.push(\{s,0\});
        while(!q.empty()){
            node now=q.top();
            q.pop();
            int u=now.v;
            if(dis[u] < now.c){</pre>
                continue;
            for(int i=head[u];i!=-1;i=e[i].next){
                int v=e[i].v;
                int w=e[i].w;
                int c=e[i].c;
//
              for (int i = 0; i < G[u].size(); ++i) {
//
                   edge \&e = G[u][i];
```

```
//
                   int v=e.to;
//
                   int w=e.capacity;
//
                   int c=e.cost;
                 if(w>0 \&\& dis[v]>dis[u]+c+h[u]-h[v]){
                     dis[v]=dis[u]+c+h[u]-h[v];
                     a[v]=min(a[u],w);
                     PreV[v] = u;
                     p[v]=i;
                     q.push({v,dis[v]});
                 }
            }
        }
        //无法再增广
        if(dis[t] == INF){
            break;
        }
        //更新势函数
        for(int i=1;i<=nc;i++){</pre>
            h[i]+=dis[i];
        }
        flow+=a[t];
        cost+=a[t]*h[t];
//
          for (int v = t; v != s; v = PreV[v]) {
//
               edge \&e = G[PreV[v]][p[v]];
               e.capacity -= a[t];
//
               G[v][e.rev].capacity += a[t];
//
          }
//
        for(int u=t;u!=s;u=e[p[u]].u){
            e[p[u]].w-=a[t];
            e[p[u]^1].w+=a[t];
        }
    }
}
int main(){
      freopen("in.txt", "r", stdin);
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&k);
        init();
        for(int i=1;i<=n;i++){</pre>
            scanf("%d",&w[i]);
        }
        s=2*n+1;
        int s1=2*n+2;
        int t1=2*n+3;
        t=2*n+4;
        add(s,s1,k,0);
        add(t1,t,k,0);
        for(int i=1;i<=n;i++){</pre>
            add(s1,i,1,-w[i]);
```

```
}
        for(int i=1;i<=n;i++){</pre>
             add(i,i+n,1,0);
        }
        for(int i=1;i<=n;i++){</pre>
             add(i+n,t1,1,0);
        }
        for(int i=1;i<=n;i++){</pre>
             for(int j=i+1; j<=n; j++) {</pre>
                 if(w[j]>=w[i]){
                     add(i+n,j,1,-w[j]);
                 }
             }
        }
        nc=2*n+4;
        int flow=0,cost=0;
        mcmf(flow,cost);
        printf("%d\n",-cost);
    }
    return 0;
}
2.5.5 spaf 费用流
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+50;
const int INF=0x3f3f3f3f;
int T,n,k,w[N];
struct Edge{
    int u,v,w,c,next;
}e[N];
int cnt,head[N];
void init(){
    cnt=0;
    memset(head, -1, sizeof(head));
void add(int u,int v,int w,int c){
    e[cnt]=Edge{u,v,w,c,head[u]};
    head[u]=cnt++;
    e[cnt]=Edge{v,u,0,-c,head[v]};
    head[v]=cnt++;
int d[N],inq[N],p[N],a[N];
int nc,s,t;
bool BF(int &flow,int &cost){
    for(int i=0;i<=nc;i++){</pre>
        d[i]=INF;
        inq[i]=false;
    }
```

```
d[s]=0;
    p[s]=0;
    a[s]=INF;
    queue<int> q;
    q.push(s);
    inq[s]=true;
    while(!q.empty()){
        int u=q.front();
        q.pop();
        inq[u]=false;
        for(int i=head[u];i!=-1;i=e[i].next){
             int v=e[i].v;
            int w=e[i].w;
            int c=e[i].c;
             if(w>0 \&\& d[v]>d[u]+c){
                 d[v]=d[u]+c;
                 p[v]=i;
                 a[v]=min(a[u],w);
                 if(!inq[v]){
                     q.push(v);
                     inq[v]=true;
                 }
            }
        }
    }
    if(d[t] == INF) {
        return false;
    }
    flow+=a[t];
    cost+=d[t]*a[t];
    for(int u=t;u!=s;u=e[p[u]].u){
        e[p[u]].w-=a[t];
        e[p[u]^1].w+=a[t];
    }
    return true;
}
void mcmf(int &flow,int &cost){
    flow=0;
    cost=0;
    while(BF(flow,cost));
}
int main(){
      freopen("in.txt", "r", stdin);
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&k);
        init();
        for(int i=1;i<=n;i++){</pre>
             scanf("%d",&w[i]);
        }
```

```
s=2*n+1;
        int s1=2*n+2;
        int t1=2*n+3;
        t=2*n+4;
        add(s,s1,k,0);
        add(t1,t,k,0);
        for(int i=1;i<=n;i++){</pre>
             add(s1,i,1,-w[i]);
        }
        for(int i=1;i<=n;i++){</pre>
             add(i,i+n,1,0);
        }
        for(int i=1;i<=n;i++){</pre>
             add(i+n,t1,1,0);
        for(int i=1;i<=n;i++){</pre>
            for(int j=i+1; j<=n; j++){</pre>
                 if(w[j]>=w[i]){
                     add(i+n,j,1,-w[j]);
                 }
            }
        }
        nc=2*n+4;
        int flow=0,cost=0;
        mcmf(flow,cost);
        printf("%d\n",-cost);
    }
    return 0;
}
2.5.6 分层图最短路
#include <bits/stdc++.h>
using namespace std;
const int N=1e4+50;
const int INF=0x3f3f3f3f;
int s,t,n,m,k,u,v,w;
bool vis[N][20];
//dis[v][i]: 到 v 使用 i 条免费路的最短路
int dis[N][20];
struct Edge{
    int v,w;
};
vector<Edge> g[N];
struct node{
    int v,dis,lev;
    bool operator <(const node& rhs)const{</pre>
        return dis>rhs.dis;
    }
};
```

```
int dijkstra(int s,int t){
    for(int i=0;i<=n;i++){</pre>
        for(int j=0; j<=k; j++){</pre>
             vis[i][j]=false;
             dis[i][j]=INF;
        }
    }
    dis[s][0]=0;
    priority_queue<node> pq;
    pq.push(node{s,0,0});
    while(!pq.empty()){
        node t=pq.top();
        pq.pop();
        int u=t.v;
        int lev=t.lev;
        if(vis[u][lev]){
             continue;
        }
        vis[u][lev]=true;
        int siz=g[u].size();
        for(int i=0;i<siz;i++){</pre>
             int v=g[u][i].v;
             int w=g[u][i].w;
             if(dis[u][lev]+w<dis[v][lev]){</pre>
                 dis[v][lev]=dis[u][lev]+w;
                 pq.push(node{v,dis[v][lev],lev});
             }
             if(lev<k && dis[u][lev]<dis[v][lev+1]){</pre>
                 //使用一条免费路
                 dis[v][lev+1]=dis[u][lev];
                 pq.push(node{v,dis[v][lev+1],lev+1});
             }
        }
    }
    int ans=INF;
    for(int i=0;i<=k;i++){</pre>
        ans=min(ans,dis[t][i]);
    }
    return ans;
}
int main(void){
    scanf("%d%d%d",&n,&m,&k);
    scanf("%d%d",&s,&t);
    for(int i=0;i<m;i++){</pre>
        scanf("%d%d%d",&u,&v,&w);
        g[u].push_back(Edge{v,w});
        g[v].push_back(Edge{u,w});
    }
    int ans=dijkstra(s,t);
    printf("%d\n",ans);
```

```
return 0;
}
2.5.7 日期公式
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+50;
int T,n;
int ms[]={0,31,28,31,30,31,30,31,30,31,30,31};
bool leap(int y){
    return ((y\%4==0 \&\& y\%100!=0) || y\%400==0);
}
//判断日期合法性
bool check(int y,int m,int d){
    if(m==0 || m>12 || d==0){
        return false;
    }
    int t=ms[m];
    if(leap(y) && m==2){
       t++;
    }
    return d<=t;
}
//基姆拉尔森公式
bool cl(int y,int m,int d){
    if(!check(y,m,d)){
       return false;
    }
    //题面...
    if(y<1600){</pre>
       return false;
    //1 2 月要转成上一年 13 14 月
    if(m<=2){
       m+=12;
    int w=((d+2*m+3*(m+1)/5+y+y/4-y/100+y/400+1)\%7+7)\%7;
    return w==5;
}
int main(void){
    //main
    return 0;
}
2.5.8 二维 RMQ
#include <bits/stdc++.h>
using namespace std;
const int Log=12;
```

```
const int N=610;
const int inf=1e9;
int n,m,Q;
int maxv[Log][Log][N][N];
int pre[N],val[N][N];
void init(){
    for(int i=1;i<=n;i++){</pre>
        for(int j=1; j<=m; j++){</pre>
            maxv[0][0][i][j]=val[i][j];
        }
    }
    pre[2]=pre[3]=1;
    for(int i=4,up=max(n,m);i<=up;i++){</pre>
        pre[i]=pre[i>>1]+1;
    }
    int up1=pre[n]+1,up2=pre[m]+1;
    for(int l1=0;l1<=up1;l1++){</pre>
        for(int 12=0;12<=up2;12++){</pre>
             if(!11&&!12){
                 continue;
             for(int i=1;(i+(1<<l1)-1)<=n;i++){
                 for(int j=1;(j+(1<<12)-1)<=m;j++){
                     if(12){
                          maxv[11][12][i][j]=max(maxv[11][12-1][i][j],
                                                  \max [11][12-1][i][j+(1<<(12-1))];
                     }else{
                          maxv[11][12][i][j]=max(maxv[11-1][12][i][j],
                                                  \max [11-1][12][i+(1<<(11-1))][j]);
                     }
                 }
            }
        }
    }
}
int query(int x1,int y1,int x2,int y2){
    int p=pre[x2-x1+1],q=pre[y2-y1+1];
    int ans=-inf;
    ans=max(maxv[p][q][x1][y1], maxv[p][q][x1][y2-(1<<q)+1]);
    ans=max(ans,max(maxv[p][q][x2-(1<<p)+1][y1],maxv[p][q][x2-(1<<p)+1][y2-(1<<q)+1]));
    return ans;
}
int x1,x2,y1,y2;
int main(){
    scanf("%d%d%d",&n,&m,&Q);
    for(int i=1;i<=n;i++){</pre>
        for(int j=1; j<=m; j++){</pre>
             scanf("%d",&val[i][j]);
        }
    }
```

```
init();
    for(int i=1;i<=Q;i++){</pre>
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        printf("\frac{n}{d}, query(x1,y1,x2,y2));
    }
    return 0;
}
      左偏树 + 并查集
2.5.9
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+50;
int n,m,o,x,y;
int dis[N],val[N],ls[N],rs[N],p[N];
void init(){
    for(int i=1;i<=n;i++){</pre>
        p[i]=i;
    }
}
int find(int x){
    return x==p[x]?p[x]:p[x]=find(p[x]);
}
int merge(int a,int b){
    if(!a || !b){
        return a+b;
    }
    //val[a]>val[b] 小根堆
    if(val[a]>val[b] || (val[a]==val[b] && a>b)){
        swap(a,b);
    rs[a]=merge(rs[a],b);
    if(dis[ls[a]] < dis[rs[a]]){</pre>
        swap(ls[a],rs[a]);
    }
    //更新并查集的根
    p[ls[a]]=p[rs[a]]=p[a]=a;
    dis[a]=dis[rs[a]]+1;
    return a;
//删除 x 所在堆顶元素
void pop(int x){
    val[x]=-1;
    //先断开,更新并查集的根
    p[ls[x]]=ls[x];
   p[rs[x]]=rs[x];
    //合并,更新并查集的根
    p[x] = merge(ls[x], rs[x]);
    ls[x]=rs[x]=dis[x]=0;
}
```

```
int main(){
    scanf("%d%d",&n,&m);
    init();
    for(int i=1;i<=n;i++){</pre>
        scanf("%d",&val[i]);
    }
    for(int i=0;i<m;i++){</pre>
        scanf("%d",&o);
        if(o==1){
            scanf("%d%d",&x,&y);
            int fx=find(x);
             int fy=find(y);
            if(val[x]==-1 \mid \mid val[y]==-1 \mid \mid fx==fy){
                 continue;
            }
            p[fx]=p[fy]=merge(fx,fy);
        }else{
            scanf("%d",&x);
            if(val[x]==-1){
                 printf("-1\n");
            }else{
                 int fx=find(x);
                 printf("%d\n",val[fx]);
                 pop(fx);
            }
        }
    }
    return 0 ;
}
2.5.10 左偏树 +dfs
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=1e5+50;
struct Edge{
    int v,next;
}e[N];
int cnt,head[N];
void init(){
    cnt=0;
    memset(head, -1, sizeof(head));
}
void add(int u,int v){
    e[cnt]=Edge{v,head[u]};
    head[u]=cnt++;
}
int n,f;
11 m,1[N],w[N],ans,sum[N];
```

```
int ls[N],rs[N],dis[N],siz[N];
int merge(int a,int b){
    if(!a || !b){
        return a+b;
    }
    if(w[a] < w[b]){</pre>
        swap(a,b);
    }
    //a 作为新根, 右儿子和 b 合并
    rs[a]=merge(rs[a],b);
    //维护左偏性质
    if(dis[ls[a]] < dis[rs[a]]){</pre>
        swap(ls[a],rs[a]);
    }
    dis[a]=dis[rs[a]]+1;
    return a;
}
int pop(int a){
    //弹出堆顶元素即把左右儿子合并
    int rt=merge(ls[a],rs[a]);
    ls[a]=rs[a]=dis[a]=0;
   return rt;
}
//在 u 的子树中找到 sum(w[v]) 小于 m, 使得 siz*l[u] 最大
int dfs(int u){
    int a=u;
    sum[u]=w[u];
    siz[u]=1;
    for(int i=head[u];i!=-1;i=e[i].next){
        int v=e[i].v;
        int b=dfs(v);
        a=merge(a,b);
        sum[u] += sum[v];
        siz[u]+=siz[v];
    while(sum[u]>m){
        sum[u] -= w[a];
        siz[u]--;
        a=pop(a);
    }
    ans=max(ans,1[u]*siz[u]);
    return a;
}
int main(){
      freopen("in.txt", "r", stdin);
    scanf("%d%lld",&n,&m);
    init();
    int root;
    for(int i=1;i<=n;i++){</pre>
        scanf("%d%lld%lld",&f,&w[i],&l[i]);
```

```
if(f){
            add(f,i);
        }else{
            root=i;
        }
    }
    dfs(root);
    printf("%lld\n",ans);
}
2.5.11 左偏树 +dfs
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=3e5+50;
struct Edge{
    int v,next;
}e[N],ct[N];
int cnt1,cnt2,head1[N],head2[N];
void init(){
    cnt1=0;
    cnt2=0;
    memset(head1,-1,sizeof(head1));
    memset(head2,-1,sizeof(head1));
}
void add(int u,int v,bool tr){
    if(tr){
        e[cnt1]=Edge{v,head1[u]};
        head1[u]=cnt1++;
    }else{
        ct[cnt2] = Edge{v,head2[u]};
        head2[u]=cnt2++;
    }
}
int n,m,fa,fi[N],sis[N],k[N],ls[N],rs[N],dis[N],dep[N];
11 f[N],ai[N],vi[N],g[N],ad[N],mu[N];
//对 a 子树计算标记
void fun(int a,ll add,ll mul){
    if(a){
        g[a]*=mul;
        g[a] += add;
        ad[a] *=mul;
        ad[a]+=add;
        mu[a] *=mul;
    }
}
void pushdown(int a){
    fun(ls[a],ad[a],mu[a]);
    fun(rs[a],ad[a],mu[a]);
```

```
ad[a]=0;
   mu[a]=1;
}
int merge(int a,int b){
   if(!a || !b){
       return a+b;
   }
   pushdown(a);
   pushdown(b);
   if(g[a]>g[b]){
       swap(a,b);
   }
   rs[a]=merge(rs[a],b);
   if(dis[ls[a]] < dis[rs[a]]){</pre>
       swap(ls[a],rs[a]);
   }
   dis[a]=dis[rs[a]]+1;
   return a;
}
int pop(int a){
   pushdown(a);
   return merge(ls[a],rs[a]);
}
int dfs(int u,int d){
   //因为是小根堆, 这里是 a=0, 如果是大根堆, a=u
   int a=0;
   dep[u]=d;
   //合并在这个城池开始的所有骑士
   for(int i=head2[u];i!=-1;i=ct[i].next){
       int v=ct[i].v;
       a=merge(a,v);
   //合并能从下面上来到这个城池的骑士
   for(int i=head1[u];i!=-1;i=e[i].next){
       int v=e[i].v;
       a=merge(a,dfs(v,d+1));
   }
   //攻击力不够的骑士死在这个城池,记录死的位置,通过深度可知占领的城池数
   while(a && g[a]<f[u]){
       k[a]=u;
       sis[u]++;
       a=pop(a);
   }
   //更新攻击力, 回溯到上一层城池进行攻击
   if(ai[u]){
       fun(a,0,vi[u]);
   }else{
       fun(a,vi[u],1);
   return a;
```

```
}
int main(){
      freopen("in.txt", "r", stdin);
//
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){</pre>
        scanf("%lld",&f[i]);
    }
    init();
    for(int i=2;i<=n;i++){</pre>
        scanf("%d%lld%lld", &fa, &ai[i], &vi[i]);
        add(fa,i,true);
    }
    for(int i=1;i<=m;i++){</pre>
        scanf("%lld%d", &g[i], &fi[i]);
        add(fi[i],i,false);
    }
    dfs(1,1);
    for(int i=1;i<=n;i++){</pre>
        printf("%d\n",sis[i]);
    }
    for(int i=1;i<=m;i++){</pre>
        printf("%d\n",dep[fi[i]]-dep[k[i]]);
    }
    return 0;
}
2.5.12 树上启发式合并
//树上启发式合并, 求子树出现次数最多的数之和
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=1e5+50;
int n,a[N],u,v;
vector<int> g[N];
int sz[N],son[N];
void dfs1(int u,int f){
    sz[u]=1;
    int siz=g[u].size();
    for(int i=0;i<siz;i++){</pre>
        int v=g[u][i];
        if(v==f){
            continue;
        }
        dfs1(v,u);
        sz[u] += sz[v];
        if(sz[v]>sz[son[u]]){
            son[u]=v;
        }
    }
```

```
}
bool vis[N];
int maxv;
11 sum,ans[N];
int hsh[N];
void calc(int u,int f,int k){
   //a[u] 颜色 ++
   hsh[a[u]] += k;
   //更新众数, 计算答案
   if(k>0 && hsh[a[u]]==maxv){
       sum+=a[u];
   }
   if(k>0 && hsh[a[u]]>maxv){
       maxv=hsh[a[u]];
       sum=a[u];
   }
   int siz=g[u].size();
   for(int i=0;i<siz;i++){</pre>
       int v=g[u][i];
       //vis[v]=1 表示 hsh 数组中已经存了这棵子树
       //即暴力添加轻儿子信息
       if(v==f || vis[v]){
           continue;
       }
       calc(v,u,k);
   }
}
//z 表示 u 是否是重儿子
void dfs2(int u,int f,bool z){
   int siz=g[u].size();
   for(int i=0;i<siz;i++){</pre>
       int v=g[u][i];
       if(v==f || v==son[u]){
           continue;
       }
       //先走轻儿子,处理完不保留信息
       dfs2(v,u,0);
   }
   //再走重儿子
   if(son[u]){
       dfs2(son[u],u,1);
       vis[son[u]]=1;
   }
   //计算 u 为根的子树答案
   calc(u,f,1);
   ans[u]=sum;
   //重儿子等父节点信息统计完再清空
   if(son[u]){
       vis[son[u]]=0;
   }
```

```
if(!z){
       //擦除轻儿子贡献
       calc(u,f,-1);
       maxv=sum=0;
   }
}
int main(void){
   // freopen("in.txt", "r", stdin);
   scanf("%d",&n);
   for(int i=1;i<=n;i++){</pre>
       scanf("%d",&a[i]);
   }
   for(int i=0;i<n-1;i++){</pre>
       scanf("%d%d",&u,&v);
       g[u].push_back(v);
       g[v].push_back(u);
   }
   dfs1(1,0);
   dfs2(1,0,0);
   for(int i=1;i<=n;i++){</pre>
       printf("%lld%c",ans[i],i==n?'\n':' ');
   }
   return 0;
}
     线段树合并 + 树上 LIS
2.5.13
#include <bits/stdc++.h>
using namespace std;
#define lson i<<1
#define rson i << 1/1
#define mid (l+r)/2
const int N=1e5+50;
vector<int> g[N];
int ans;
int a[N],b[N],n,m,u,v;
//动态开点权值线段树
int cnt,sum[N*40],ls[N*40],rs[N*40];
//两个线段树, 维护节点对应 (值域) 区间的 lis/lds 最大值
//比如节点 [1,4] 维护的就是 1 的某个子孙节点到 1, 2 的某个子孙节点到 2...4 的某个子孙节点
int lis[N*40],lds[N*40];
int rt[N];
//now: 当前节点
//l,r: 权值线段树范围
//x: 要插入的值
//a: 本题中区别维护的是 LIS 还是 LDS
//c: 要插入的这个权值对应的权值 (lis/lds 长度)
//不同于主席树, 权值线段树合并是每个根节点 rt[i] 一个完全的线段树, 然后再合并
void insert(int &now,int l,int r,int x,int c,int *a){
   //动态开点
```

```
if(!now){
       now=++cnt;
   }
   //lis/lds 即维护最大值
   a[now] = max(a[now],c);
   if(l==r){
       return;
   }
   if(x<=mid){</pre>
       insert(ls[now],1,mid,x,c,a);
   }else{
       insert(rs[now],mid+1,r,x,c,a);
   }
}
//权值线段树合并
int merge(int a,int b){
   if(!a){
       return b;
   if(!b){
       return a;
   }
   lis[a]=max(lis[a],lis[b]);
   lds[a] = max(lds[a], lds[b]);
   //更新全局答案
   ans=max(ans,max(lis[ls[a]]+lds[rs[b]],lds[rs[a]]+lis[ls[b]]));
   ls[a]=merge(ls[a],ls[b]); rs[a]=merge(rs[a],rs[b]);
   return a;
}
//权值线段树查询最值 (lis 或 lds)
//查询 now(rt[v]) 对应权值线段树权值为 [ql,qr] 之间的 a 数组最大值
//即查询以 v(u) 的子节点) 结尾, 所对应子树权值为 [ql,qr] 之间的 lis 或 lds, 和 u 的权值结
int query(int now,int l,int r,int ql,int qr,int *a){
   //因为是动态开点, 递归到不存在的节点直接返回
   if(!now){
       return 0;
   }
   if(ql<=l && qr>=r){
       return a[now];
   }
   if(qr<=mid){</pre>
       return query(ls[now],1,mid,ql,qr,a);
   }
   if(ql>mid){
       return query(rs[now],mid+1,r,ql,qr,a);
   return max(query(ls[now],1,mid,q1,mid,a),query(rs[now],mid+1,r,mid+1,qr,a));
}
void dfs(int u,int f){
   int mlis=0,mlds=0;
```

```
int siz=g[u].size();
    for(int i=0;i<siz;i++){</pre>
       int v=g[u][i];
       if(v==f){
           continue;
       }
       dfs(v,u);
       //对于 u, 先递归查询每个子链, 再依次合并
       //这里就是查询从 v 的子孙节点到 v 路径上值域为 \emph{[1,a[u]-1]} 的 \emph{lis} 长度
       int ilis=query(rt[v],1,m,1,a[u]-1,lis);
       //同理
       int ilds=query(rt[v],1,m,a[u]+1,m,lds);
       //合并权值线段树
       rt[u]=merge(rt[u],rt[v]);
       //更新全局答案
       ans=max(ans,max(ilis+mlds,ilds+mlis)+1);
       //更新 u 节点暂时的值 (从 u 的子孙节点到 u 路径上, 以 u 结尾)
       mlis=max(mlis,ilis);
       mlds=max(mlds,ilds);
    }
    //节点 u 对应的一颗权值线段树,从 u 的某个子孙节点到 u 路径上某个值域区间的 lis/lds \dagger
    insert(rt[u],1,m,a[u],mlis+1,lis);
    insert(rt[u],1,m,a[u],mlds+1,lds);
}
int main(void){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){</pre>
       scanf("%d",&a[i]);
       b[i]=a[i];
    sort(b+1,b+1+n);
   m=unique(b+1,b+1+n)-b-1;
    for(int i=1;i<=n;i++){</pre>
       a[i]=lower_bound(b+1,b+1+m,a[i])-b;
    for(int i=0;i<n-1;i++){</pre>
       scanf("%d%d",&u,&v);
       g[u].push_back(v);
       g[v].push_back(u);
    }
    dfs(1,0);
   printf("%d\n",ans);
   return 0;
}
2.5.14 单调队列
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+50;
```

```
int n,m,k,a[N];
int mxq[N],mnq[N];
int main(void){
    // freopen("in.txt", "r", stdin);
    while (\simscanf ("%d%d%d", &n, &m, &k)){
       for(int i=1;i<=n;i++){</pre>
            scanf("%d",&a[i]);
       }
       int l=1;
       int 11=1,r1=0;
       int 12=1,r2=0;
       int ans=0;
       for(int r=1;r<=n;r++){</pre>
           //删除队尾元素 入队
           while (11 \le r1 \& a[r] \ge a[mxq[r1]])
               r1--;
           }
           mxq[++r1]=r;
           while (12 \le r2 \&\& a[r] \le a[mnq[r2]])
               r2--;
           mnq[++r2]=r;
            //队首元素只能用来维护 max 和 min 的最大差值
           while (1 \le r \&\& 11 \le r1 \&\& 12 \le r2 \&\& a[mxq[11]] - a[mnq[12]] > k)
               1++;
               while(l1<=r1 && mxq[l1]<1){
                   11++;
               }
               while(12<=r2 && mnq[12]<1){
                   12++;
               }
           if(11<=r1 && 12<=r2 && a[mxq[11]]-a[mnq[12]]>=m){
               ans=\max(ans,r-1+1);
            }
       }
       printf("%d\n",ans);
    }
   return 0;
}
       康拓展开 + 字典序第 k 小全排列
//给定一个排列 45231, 求排名
//每个数后面比该数小的数分别有 [3 3 1 1 0] 个
//所以排名为 3*4!+3*3!+1*2!+1*1+0*0!+1=94
//给定排名 96, 求对应排列 (n=5)
//96-1=95 95/4!=3...23 23/3!=3...5 5/2!=2...1 1/1!=0...0
//商就对应每个数后面比该数小的数的个数
//因为阶乘, 适合 n 小的情况
```

```
#include <bits/stdc++.h>
using namespace std;
const int N=1001;
int fac[]={1,1,2,6,24,120,720,5040,40320};
int n,m;
int a[N], vis[15];
void solve(int n,int m){
     m--;
     memset(a,0,sizeof(a));
     memset(vis,0,sizeof(vis));
     int k=m;
     for(int i=0;i<n;i++){</pre>
          int t=k/fac[n-i-1];
          int j=1;
          for(; j<=n; j++){
                if(!vis[j]){
                    if(t==0){
                        break;
                    }
                    t--;
                }
          }
          vis[j]=1;
          a[i]=j;
          k=k\%fac[n-i-1];
     }
}
int main(){
    while(~scanf("%d%d",&n,&m)){
        solve(n,m);
        for(int i=0;i<n;i++){</pre>
            printf("%d%c",a[i],i==n-1?'\n':' ');
        }
    }
    return 0;
}
2.5.16 fhq-treap
#include <bits/stdc++.h>
using namespace std;
const int N=1e6+50;
const int INF=0x3f3f3f3f;
struct FhqTreap{
    int tot,seed;
    struct node{
        int val,rk,lc,rc,siz;
    }tr[N];
    int Rand(){
        return seed=int(seed*48271111%2147483647);
```

```
}
void pushup(int rt){
   tr[rt].siz=tr[tr[rt].lc].siz+tr[tr[rt].rc].siz+1;
int newnode(int v){
   tr[++tot]=node{v,Rand(),0,0,1};
   return tot;
}
//将一颗 treap 分成两棵,小于等于 v 的在 a 树,大于 v 的在 b 树
void split(int rt,int &a,int &b,int v){
   if(!rt){
       a=b=0;
       return;
   if(tr[rt].val<=v){</pre>
       a=rt;
       split(tr[rt].rc,tr[a].rc,b,v);
   }else{
       b=rt;
       split(tr[rt].lc,a,tr[b].lc,v);
   pushup(rt);
}
//将两棵 treap 合并成一颗
void merge(int &rt,int a,int b){
    if(a==0 | b==0){
       //rt=a || rt=b
       rt=a+b;
       return;
   }
   if(tr[a].rk<tr[b].rk){</pre>
       //b 合并在 a 的右子树
       rt=a;
       merge(tr[a].rc,tr[a].rc,b);
   }else{
       //a 合并在 b 的左子树
       rt=b;
       merge(tr[b].lc,a,tr[b].lc);
   pushup(rt);
void ins(int &rt,int v){
   int x=0,y=0;
   int no=newnode(v);
    //将当前的树根据 v 分成两棵
   split(rt,x,y,v);
    //合并在 x 树上
   merge(x,x,no);
   //再重新合并到根上
   merge(rt,x,y);
```

```
}
void del(int &rt,int v){
   int x=0,y=0,z=0;
   //将值为 υ 的节点分开
   split(rt,x,y,v);
   split(x,x,z,v-1);
   //分出的三棵树再合并,z 树包含 v 节点,所以只合并 tr[z].lc 和 tr[z].rc
   merge(z,tr[z].lc,tr[z].rc);
   merge(x,x,z);
   merge(rt,x,y);
}
//第 k 大
int kth(int rt,int k){
   while(true){
       int t=tr[tr[rt].lc].siz+1;
       if(k<t){
           rt=tr[rt].lc;
       }else if(k>t){
           k-=t;
           rt=tr[rt].rc;
       }else{
           break;
   return tr[rt].val;
}
//v 的排名 (重复的数算第一个排名)
int rnk(int &rt,int v){
   int x=0,y=0;
   split(rt,x,y,v-1);
   int tmp=tr[x].siz+1;
   merge(rt,x,y);
   return tmp;
}
//前驱
int pre(int &rt,int v){
   int x=0, y=0;
   //将 v-1 分离, 查询前 k 大, 即最大一个小于 v
   split(rt,x,y,v-1);
   int tmp=kth(x,tr[x].siz);
   merge(rt,x,y);
   return tmp;
}
//后继
int nex(int &rt,int v){
   int x=0,y=0;
   //将 v 分离, 查询第 1 大, 即最小一个大于 v
   split(rt,x,y,v);
   int tmp=kth(y,1);
   merge(rt,x,y);
```

```
return tmp;
    }
    int init(){
       tot=0;
        seed=233;
        return newnode(INF);
}ac;
int n,o,v;
int main(){
    scanf("%d",&n);
    int rt=ac.init();
    for(int i=1;i<=n;i++){</pre>
        scanf("%d%d",&o,&v);
        if(o==1){
            ac.ins(rt,v);
        }else if(o==2){
            ac.del(rt,v);
        else if(o==3){
            printf("%d\n",ac.rnk(rt,v));
        }else if(o==4){
            printf("%d\n",ac.kth(rt,v));
        } else if(o==5){
            printf("%d\n",ac.pre(rt,v));
        }else if(o==6){
            printf("%d\n",ac.nex(rt,v));
        }
    }
    return 0;
}
2.5.17 LCT 维护 MST
#include<bits/stdc++.h>
using namespace std;
#define ls tr[u][0]
#define rs tr[u][1]
const int N=4e5+50;
struct Edge{
    int u,v,l,r;
    //按 r 值从大到小排序, LCT 维护 l 值最大值
    bool operator <(const Edge& rhs)const{</pre>
        return r>rhs.r;
    }
}e[N];
struct LCT{
    int fa[N],tr[N][2],xr[N],lz[N],mx[N];
    int q[N],top;
    //维护信息, 这里维护 1 最最大值的位置
    void pushup(int u){
```

```
mx[u]=u;
   if(ls && e[mx[ls]].1>e[mx[u]].1){
       mx[u]=mx[ls];
   }
   if(rs && e[mx[rs]].l>e[mx[u]].l){
       mx[u]=mx[rs];
   }
}
//区间翻转,左右子树交换
void change(int u){
   int temp=ls;
   ls=rs;
   rs=temp;
   lz[u]^=1;
}
//下传标记
void pushdown(int u){
   if(lz[u]){
       if(ls){
           change(ls);
       if(rs){
           change(rs);
       lz[u]=0;
   }
//判断 u 是否为 acplay 的根, 父节点是 Path father
bool isroot(int u){
   return (tr[fa[u]][0]!=u && tr[fa[u]][1]!=u);
}
int get(int x){
   return tr[fa[x]][1]==x;
}
int connect(int x,int f,int son){
   fa[x]=f;
   tr[f][son]=x;
}
//acplay 单旋
void rotate(int x){
   int y=fa[x];
   int z=fa[y];
   int xs=get(x);
   int ys=get(y);
   //区别于普通 acplay, 如果 y 已经是 acplay 的根, z 就不属于该 acplay
   if(!isroot(y)){
       connect(x,z,ys);
   }else{
       fa[x]=z;
   }
```

```
int b=tr[x][xs^1];
   connect(b,y,xs);
   connect(y,x,xs^1);
   pushup(y);
   pushup(x);
}
void splay(int x){
   //区别于普通 acplay
   top=1;
   q[top]=x;
   int i=x;
   while(!isroot(i)){
       q[++top]=fa[i];
       i=fa[i];
   }
   while(top){
       pushdown(q[top--]);
   }
   while(!isroot(x)){
       int y=fa[x];
       int z=fa[y];
       if(!isroot(y)){
          rotate(get(x)==get(y)?y:x);
       rotate(x);
   }
}
//打通根到 u 的实链, 得到一个中序遍历以根开始、以 u 结束的 acplay
void access(int u){
   int x=0;
   //u 不为整个 LCT 的根
   while(u){
       //将 u 旋到根
       splay(u);
       //断开右子树, 将 x 接在 u 的右子树
       connect(x,u,1);
       pushup(u);
       //往上爬
       x=u;
       u=fa[u];
   }
}
//将 u 变为整个 LCT 的根
void makeroot(int u){
   //access 后, u 成为所在 acplay 中深度最大的点
   access(u);
   //splay 后, u 旋至 acplay 的根, 没有右儿子
   splay(u);
   //翻转整个 acplay, u 没有左子树, 也就是 u 是深度最小的点, 也就是 LCT 的根
   change(u);
```

```
}
   //查找 u 所在原树的根
   int find(int u){
       //构造出新的 acplay 后,不断找左儿子,即深度小的
       access(u);
       //find(u) 会把 u 旋至 acplay 的根
       splay(u);
       while(ls){
           pushdown(u);
           u=ls;
       }
       return u;
   //拉出一条 u-v 的路径用 acplay 维护, 以 v 为根
   void split(int u,int v){
       makeroot(u);
       access(v);
       splay(v);
   }
   //断开边 (u,v)
   void cut(int u,int v){
       makeroot(u);
       //(u,v) 不一定合法
       //1. 边存在 2.v 是 u 的父节点 3.v 没有右儿子
       //find(v) 调用了 access(v), u 到 v 的实链构造一个 acplay, v 成了 acplay 的根
       if(find(v)==u && fa[u]==v && !rs){
           fa[u]=tr[v][0]=0;
           pushup(v);
       }
   }
   //连轻边 (x,y)
   void link(int u,int v){
       makeroot(u);
       if(find(v)!=u){
           fa[u]=v;
       }
   }
   //查询路径 max(li) 的位置
   int query(int u,int v){
       makeroot(v);
       access(u);
       splay(u);
       return mx[u];
   }
}ac;
vector<pair<int,int> > vec;
int n,m;
int main(){
   scanf("%d%d",&n,&m);
   for(int i=1;i<=m;i++){</pre>
```

```
scanf("%d%d%d%d", &e[i].u, &e[i].v, &e[i].1, &e[i].r);
}
sort(e+1,e+m+1);
//main
//r 从大到小枚举, 当前 e[i].r 就是已加入 LCT 的边中 r 值最小的边
//而 LCT 就必须维护最小的 (最大的 l 值), 也就是 max(li) 的最小生成树, [l,r] 就是这个l
for(int i=1;i<=m;i++){</pre>
   int u=e[i].u;
   int v=e[i].v;
   if(ac.find(m+u)==ac.find(m+v)){
       //查询实点, (m+1...m+n)
       int t=ac.query(m+u,m+v);
       if(e[i].l<e[t].l){
           //维护最小生成树 (删去原来的边, 加入更小的边)
           ac.cut(t,m+e[t].u);
           ac.cut(t,m+e[t].v);
           ac.link(i,m+u);
           ac.link(i,m+u);
       }
   }else{
       //u 和 v 未连通
       //LCT 中节点分为边的编号和图节点的编号
       //建立虚拟点, 将边权转化为 LCT 可以维护的点权
       //虚点 (1..m) 表示实际的边信息, 实点 (m+1..m+n) 只表示点
       ac.link(i,m+u);
       ac.link(i,m+v);
   }
   //1-n 连通, 查询路径最大 l 值的位置 (边的编号)
   if(ac.find(m+1)==ac.find(m+n)){
       int tmp=ac.query(m+1,m+n);
       if(e[tmp].l<=e[i].r){</pre>
           vec.push back({e[tmp].l,e[i].r});
       }
   }
}
int siz=vec.size();
//特判 0 卡 9%
if(siz==0){
   printf("0\n");
   return 0;
//线段并
sort(vec.begin(),vec.end());
int lst=vec[0].second+1;
int ans=vec[0].second-vec[0].first+1;
for(int i=1;i<siz;i++){</pre>
   ans+=vec[i].second-max(lst,vec[i].first)+1;
   lst=max(lst,vec[i].second+1);
printf("%d\n",ans);
```

```
return 0;
}
2.5.18 LCT 维护路径异或最大值
#include<bits/stdc++.h>
using namespace std;
#define ls tr[u][0]
#define rs tr[u][1]
const int N=3e5+7;
int n,m,a[N],o,x,y;
struct LCT{
   int fa[N],tr[N][2],xr[N],lz[N],q[N],top;
   //维护信息,这里维护路径点权异或值
   void pushup(int u){
       xr[u]=xr[ls]^xr[rs]^a[u];
   }
   //区间翻转, 左右子树交换
   void change(int u){
       int temp=ls;
       ls=rs;
       rs=temp;
       lz[u]^=1;
   }
   //下传标记
   void pushdown(int u){
       if(lz[u]){
           if(ls){
               change(ls);
           }
           if(rs){
               change(rs);
           }
           lz[u]=0;
       }
   }
   //判断 u 是否为 Splay 的根, 父节点是 Path father
   bool isroot(int u){
       return (tr[fa[u]][0]!=u && tr[fa[u]][1]!=u);
   }
   int get(int x){
       return tr[fa[x]][1]==x;
   int connect(int x,int f,int son){
       fa[x]=f;
       tr[f][son]=x;
   }
   //Splay 单旋
   void rotate(int x){
       int y=fa[x];
```

```
int z=fa[y];
   int xs=get(x);
   int ys=get(y);
   //区别于普通 Splay, 如果 y 已经是 Splay 的根, z 就不属于该 Splay
   if(!isroot(y)){
      connect(x,z,ys);
   }else{
      fa[x]=z;
   }
   int b=tr[x][xs^1];
   connect(b,y,xs);
   connect(y,x,xs^1);
   pushup(y);
   pushup(x);
}
void splay(int x){
   //区别于普通 Splay
   top=1;
   q[top]=x;
   int i=x;
   while(!isroot(i)){
      q[++top]=fa[i];
      i=fa[i];
   }
   while(top){
      pushdown(q[top--]);
   while(!isroot(x)){
      int y=fa[x];
      int z=fa[y];
      if(!isroot(y)){
          rotate(get(x) = get(y)?y:x);
      rotate(x);
   }
}
//打通根到 u 的实链,得到一个中序遍历以根开始、以 u 结束的 Splay
void access(int u){
   /**
    * 先将 x 旋至所在 Splay 的根, 然后断开右子树, 然后沿着偏爱路径往上爬
    * 每遇到一条虚边, 把虚边连向的节点 y 旋至所在 Splay 的根, 断开 y 的右子树, 并把
    * Splay 中右儿子 ==> 深度大,要打通的实链中 u 是深度最大的,从下往上
    * 因此每次将旋转后的 Splay 接到下一个 Splay 的右儿子上
    */
   int x=0;
   //u 不为整个 LCT 的根
   while(u){
      //将 u 旋到根
      splay(u);
      //断开右子树,将 x 接在 u 的右子树
```

```
connect(x,u,1);
      pushup(u);
      //往上爬
      x=u;
      u=fa[u];
   }
//将 u 变为整个 LCT 的根
void makeroot(int u){
   //access 后, u 成为所在 Splay 中深度最大的点
   access(u);
   //splay 后, u 旋至 Splay 的根,没有右儿子
   splay(u);
   //翻转整个 Splay, u 没有左子树, 也就是 u 是深度最小的点, 也就是 LCT 的根
   change(u);
}
//查找 u 所在原树的根
int find(int u){
   //构造出新的 Splay 后,不断找左儿子,即深度小的
   access(u);
   //find(u) 会把 u 旋至 Splay 的根
   splay(u);
   while(ls){
      pushdown(u);
      u=ls;
   }
   //加上这个就 WA 了一个点
   // splay(u);
   return u;
}
//拉出一条 u-v 的路径用 Splay 维护, 以 v 为根
void split(int u,int v){
   makeroot(u);
   access(v);
   splay(v);
}
//断开边 (u,v)
void cut(int u,int v){
   makeroot(u);
   //(u,v) 不一定合法
   //1. 边存在 2.v 是 u 的父节点 3.v 没有右儿子
   //find(v) 调用了 access(v), u 到 v 的实链构造一个 Splay, v 成了 Splay 的根
   if(find(v)==u && fa[u]==v && !rs){
      fa[u]=tr[v][0]=0;
      pushup(v);
   }
}
//连轻边 (x,y)
void link(int u,int v){
   makeroot(u);
```

```
if(find(v)!=u){
           fa[u]=v;
       }
   }
}T;
int main(){
   scanf("%d%d",&n,&m);
   for(int i=1;i<=n;i++){</pre>
       scanf("%d",&a[i]);
       T.xr[i]=a[i];
   }
   for(int i=1;i<=m;i++){</pre>
       scanf("%d%d%d", &o, &x, &y);
       if(o==0){
           T.split(x,y);
           printf("%d\n",T.xr[y]);
       }else if(o==1){
           T.link(x,y);
       else if(o==2){
           T.cut(x,y);
       }else if(o==3){
           //先旋到根, 再修改点权, pushup 更新信息
           T.splay(x);
           a[x]=y;
           T.pushup(x);
       }
   }
   return 0;
}
2.5.19 cdq 分治二维偏序
#include <bits/stdc++.h>
using namespace std;
typedef long long 11;
const int N=5e6+50;
11 n;
int ans[N];
11 get(int i,int j){
   //获取值
}
int val(ll x){
   int ans=0;
   while(x){
       ans+=x\%10;
       x/=10;
   }
   return ans;
}
```

```
int T,m,p,x,y,xb,yb,mx,cnt,c[N];
int lowbit(int x){}
void add(int i,int x){}
int sum(int i){}
void clr(int i){
   while(i<=mx){</pre>
       if(!c[i]){
           break;
       }
       c[i]=0;
        i+=lowbit(i);
    }
}
struct node{
    int type,x,y,w,id;
}a[N],b[N];
void cdq(int l,int r){
    if(l==r){
       return;
    }
    int mid=(1+r)/2;
    cdq(1,mid);
    cdq(mid+1,r);
    int i=1,j=mid+1,kk=1;
    //左边和右边单独的贡献已经递归算出
    //这里计算跨左右两边的贡献, 即左边加入 bit, 右边查询
    while (i \le mid \&\& j \le r)
       //两边此时都按 x, y 的顺序排序
       if(a[i].x \le a[j].x){
            if(a[i].type==1){
               //权值点, 加入 bit
               add(a[i].y,a[i].w);
            //按 x 归并排序
           b[kk++]=a[i++];
       }else{
            if(a[j].type==2){
               ans[a[j].id]+=sum(a[j].y);
            }else if(a[j].type==3){
               ans[a[j].id]-=sum(a[j].y);
           b[kk++]=a[j++];
       }
    }
    //处理剩下的
    while(i<=mid){</pre>
       if(a[i].type==1){
           add(a[i].y,1);
       b[kk++]=a[i++];
```

```
}
    while(j<=r){</pre>
        if(a[j].type==2){
            ans[a[j].id]+=sum(a[j].y);
        }else if(a[j].type==3){
            ans[a[j].id]-=sum(a[j].y);
        }
        b[kk++]=a[j++];
    }
    for(int i=1;i<=r;i++){</pre>
        clr(a[i].y);
        a[i]=b[i];
    }
}
int main(){
    scanf("%d",&T);
    while(T--){
        cnt=0, mx=0;
        scanf("%lld%d%d",&n,&m,&p);
        for(int i=1;i<=m;i++){</pre>
            //加入有价值的点
            scanf("%d%d",&x,&y);
            a[++cnt]=node{1,x,y,val(get(x,y)),i};
            mx=max(mx,y);
        }
        for(int i=1;i<=p;i++){</pre>
            //加入询问点
            scanf("%d%d%d",&x,&y,&xb,&yb);
            a[++cnt]=node{2,x-1,y-1,1,i};
            a[++cnt]=node{2,xb,yb,1,i};
            a[++cnt]=node{3,x-1,yb,1,i};
            a[++cnt]=node{3,xb,y-1,1,i};
            mx=max(mx,yb);
            ans[i]=0;
        }
        cdq(1,cnt);
        for(int i=1;i<=p;i++){</pre>
            printf("%d\n",ans[i]);
        }
    }
    return 0;
}
       树状数组二位偏序
2.5.20
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+50;
int n,m,l,r,c[N],a[N],p[N],ans[N];
struct node{
```

```
int o,id,l,r;
    bool operator<(const node& rhs)const{</pre>
        if(r==rhs.r){
             if(l==rhs.1){
                 //注意 l 和 r 都相同, 询问点要放在后面...
                 return o<rhs.o;</pre>
             }else{
                 return l>rhs.l;
        }else{
             return r<rhs.r;</pre>
        }
    }
};
vector<node> ns;
int lowbit(int x){
    return x\&(-x);
}
void add(int i,int x){
    while(i<=n){
        c[i]+=x;
        i+=lowbit(i);
    }
}
int sum(int i){
    int ans=0;
    while(i){
        ans+=c[i];
        i-=lowbit(i);
    }
    return ans;
}
int main(){
       freopen("in.txt", "r", stdin);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){</pre>
        scanf("%d",&a[i]);
        p[a[i]]=i;
    }
    for(int i=1;i<=n;i++){</pre>
        for(int j=i+i; j<=n; j+=i) {</pre>
             int a=p[i],b=p[j];
             if(a>b){
                 swap(a,b);
             }
             ns.push_back(\{1,0,a,b\});
        }
    }
    for(int i=1;i<=m;i++){</pre>
        scanf("%d%d",&l,&r);
```

```
ns.push_back(node{2,i,1,r});
    }
    sort(ns.begin(),ns.end());
    int ad=0;
    int siz=ns.size();
    for(int i=0;i<siz;i++){</pre>
        if(ns[i].o==1){
             add(ns[i].1,1);
             ad++;
        }else{
             ans[ns[i].id]=ad-sum(ns[i].l-1);
        }
    }
    for(int i=1;i<=m;i++){</pre>
        printf("%d\n",ans[i]);
    }
    return 0;
}
```