Standard-Code-Library

> 最大的罪行往往源自贪婪而非贫穷。

目录

Standard-Code-Library
- 目录

图论
- 仙人掌DP
- 动态最小生成树
- 可持久化左偏树求k短路
- 二分图最大权匹配
  - KM
- 一般图最大匹配
  - 高斯消元
  - 带花树
- 最大流
  - Dinic
  - ISAP
  - 最高标号预流推进（HLPP）
- 网络流原理
  - 最小割
    - 最小割输出一种方案
    - 最小割的可行边与必须边
      - 可行边
      - 必须边
  - 二分图
    - 最大匹配的可行边与必须边
      - 可行边
      - 必须边
    - 独立集

字符串
- AC自动机
- 后缀数组
  - SAMSA
- 后缀自动机
- 回文树
  - 广义回文树
- Manacher / 马拉车
- ex-KMP

数学
- FWT
- 插值
  - 牛顿插值
  - 拉格朗日插值
- 多项式
  - FFT
  - NTT
  - 多项式操作
  - 拉格朗日反演

- 分治FFT
  - 半在线卷积
- 单纯形
- 线性基
- 牛顿迭代

数论

- $O(n)$预处理逆元
- 杜教筛
- 线性筛
- Miller-Rabin
- Pollard's Rho

数据结构

- 线段树
  - 非递归线段树
  - 主席树
- 平衡树
  - Treap
  - Splay
- 树的性质
  - 重心
  - 直径
- 树分治
  - 动态树分治
  - 紫荆花之恋
- LCT
  - 不换根
  - 换根 / 维护生成树（GREALD07 加强版）
  - 维护子树信息
    - 模板题：动态QTREE4
- 长链剖分
- 可并堆（左偏树）
- 自带数据结构
  - STL
    - vector
    - list
  - pb_ds
    - 哈希表
    - 堆
    - 平衡树
    - Rope
- 常见根号思路
  - 通用
  - 序列
  - 树
  - 字符串

DP

- 决策单调性（在线）
- 斜率优化
  - 陈丹琦分治维护（NOI2007 货币兑换Cash）

杂の算法

- $O(1)$快速乘
- $O(n^2)$高精度
- xorshift
  - xorshift128 Plus
  - xorshift128

杂の参考资料
常见数列
斐波那契数
性质
周期性
卡特兰数
伯努利数
斯特林数
第一类斯特林数
第二类斯特林数
上升幂、普通幂与下降幂的转换
贝尔数
常见数列打表
卡特兰数
贝尔数
斐波那契数
Lucas数列
五边形数
错位排列数
无标号有根树计数
常用NTT素数及原根
常用素数
C++11新特性
Lambda表达式
capture 捕获子句
编译选项
Linux命令行
__builtin
打表和分段打表
注意事项
Linux注意事项
Linux中禁止使用的变量名
编译检查事项
场外相关
做题策略与心态调节

图论

仙人掌DP

```cpp
#include <bits/stdc++.h>

using namespace std;

const int maxn = 200005;

struct edge{
    int to, w, prev;
}e[maxn * 2];

vector<pair<int, int> > v[maxn];

```

```
13  vector<long long> d[maxn];
14
15  stack<int> stk;
16
17  int p[maxn];
18
19  bool vis[maxn], vise[maxn * 2];
20
21  int last[maxn], cnte;
22
23  long long f[maxn], g[maxn], sum[maxn];
24
25  int n, m, cnt;
26
27  void addedge(int x, int y, int w) {
28      // printf("%d -> %d  w = %d\n", x, y, w);
29
30      v[x].push_back(make_pair(y, w));
31  }
32
33  void dfs(int x) {
34      // printf("dfs(%d) p[x] = %d\n", x, p[x]);
35
36      vis[x] = true;
37
38      for (int i = last[x]; ~i; i = e[i].prev) {
39          if (vise[i ^ 1])
40              continue;
41
42          int y = e[i].to, w = e[i].w;
43
44          vise[i] = true;
45
46          if (!vis[y]) {
47              stk.push(i);
48              p[y] = x;
49              dfs(y);
50
51              if (!stk.empty() && stk.top() == i) {
52                  stk.pop();
53                  addedge(x, y, w);
54              }
55          }
56
57          else {
58              // printf("x = %d y = %d\n", x, y);
59              cnt++;
60              // assert(cnt <= 2 * n);
61              long long tmp = w;
62              while (!stk.empty()) {
63                  int i = stk.top();
64                  stk.pop();
65
66                  int yy = e[i].to, ww = e[i].w;
67
```

```
                addedge(cnt, yy, 0);
                // printf("cnt = %d tmp = %lld\n", cnt, tmp);
                d[cnt].push_back(tmp);

                tmp += ww;

                if (e[i ^ 1].to == y)
                    break;
            }

            addedge(y, cnt, 0);
            // tmp += w;
            sum[cnt] = tmp;
        }
    }
}

void dp(int x) {

    // printf("dp(%d) sum[x] = %lld\n", x, sum[x]);

    for (auto o : v[x]) {
        int y = o.first, w = o.second;
        dp(y);
    }

    if (x <= n) {
        for (auto o : v[x]) {
            int y = o.first, w = o.second;

            f[x] += 2 * w + f[y];
        }

        g[x] = f[x];

        for (auto o : v[x]) {
            int y = o.first, w = o.second;

            g[x] = min(g[x], f[x] - f[y] - 2 * w + g[y] + w);
        }
    }
    else {
        f[x] = sum[x];
        for (auto o : v[x]) {
            int y = o.first;

            f[x] += f[y];
        }

        g[x] = f[x];

        for (int i = 0; i < (int)v[x].size(); i++) {
            int y = v[x][i].first;

            // printf("x = %d  y = %d d[x][i] = %lld\n", x, y, d[x][i]);
```

```
123
124             g[x] = min(g[x], f[x] - f[y] + g[y] + min(d[x][i], sum[x] -
    d[x][i]));
125         }
126     }
127
128     // printf("x = %d  sum[x] = %lld  f[x] = %lld  g[x] = %lld\n", x,
    sum[x], f[x], g[x]);
129 }
130
131 signed main() {
132
133     memset(last, -1, sizeof(last));
134
135     ios::sync_with_stdio(false);
136     // scanf("%d%d", &n, &m);
137     cin >> n >> m;
138
139     // assert(n <= 100000 && m <= 2 * n - 2);
140
141     cnt = n;
142
143     while (m--) {
144         int x, y, w;
145         cin >> x >> y >> w;
146
147         e[cnte].to = y;
148         e[cnte].w = w;
149         e[cnte].prev = last[x];
150         last[x] = cnte++;
151
152         e[cnte].to = x;
153         e[cnte].w = w;
154         e[cnte].prev = last[y];
155         last[y] = cnte++;
156
157         // assert(cnte <= 400000);
158     }
159
160     dfs(1);
161     dp(1);
162
163     cout << g[1] << endl;
164
165     return 0;
166 }
```

动态最小生成树

```
1 // Dynamic Minimal Spinning Tree 动态最小生成树 O(nlog^2n)
2 // By AntiLeaf
3 // 通过题目：Bzoj2001 [Hnoi2010] City城市建设
```

```

// 动态最小生成树的离线算法比较容易，而在线算法通常极为复杂
// 一个跑得比较快的离线做法是对时间分治，在每层分治时找出一定在/不在MST上的边，只带着不确定边继续递归
// 简单起见，找确定边的过程用Kruskal算法实现，过程中的两种重要操作如下：
// - Reduction：待修改边标为+INF，跑MST后把非树边删掉，减少无用边
// - Contraction：待修改边标为-INF，跑MST后缩除待修改边之外的所有MST边，计算必须边
// 每轮分治需要Reduction-Contraction，借此减少不确定边，从而保证复杂度
// 复杂度证明：假设当前区间有k条待修改边，n和m表示点数和边数，那么最坏情况下R-C的效果为(n, m) -> (n, n + k - 1) -> (k + 1, 2k)


// 全局结构体与数组定义
struct edge { //边的定义
    int u, v, w, id; // id表示边在原图中的编号
    bool vis; // 在Kruskal时用，记录这条边是否是树边
    bool operator < (const edge &e) const { return w < e.w; }
} e[20][maxn], t[maxn]; // 为了便于回滚，在每层分治存一个副本


// 用于存储修改的结构体，表示第id条边的权值从u修改为v
struct A {
    int id, u, v;
} a[maxn];


int id[20][maxn]; // 每条边在当前图中的编号
int p[maxn], size[maxn], stk[maxn], top; // p和size是并查集数组，stk是用来撤销的栈
int n, m, q; // 点数，边数，修改数


// 方便起见，附上可能需要用到的预处理代码
for (int i = 1; i <= n; i++) { // 并查集初始化
    p[i] = i;
    size[i] = 1;
}

for (int i = 1; i <= m; i++) { // 读入与预标号
    scanf("%d%d%d", &e[0][i].u, &e[0][i].v, &e[0][i].w);
    e[0][i].id = i;
    id[0][i] = i;
}

for (int i = 1; i <= q; i++) { // 预处理出调用数组
    scanf("%d%d", &a[i].id, &a[i].v);
    a[i].u = e[0][a[i].id].w;
    e[0][a[i].id].w = a[i].v;
}

for(int i = q; i; i--)
    e[0][a[i].id].w = a[i].u;

CDQ(1, q, 0, m, 0); // 这是调用方法

```

```
56
57  // 分治主过程 O(nlog^2n)
58  // 需要调用Reduction和Contraction
59  void CDQ(int l, int r, int d, int m, long long ans) { // CDQ分治
60      if (l == r) { // 区间长度已减小到1，输出答案，退出
61          e[d][id[d][a[l].id]].w = a[l].v;
62          printf("%lld\n", ans + Kruskal(m, e[d]));
63          e[d][id[d][a[l].id]].w=a[l].u;
64          return;
65      }
66
67      int tmp = top;
68
69      Reduction(l, r, d, m);
70      ans += Contraction(l, r, d, m); // R-C
71
72      int mid = (l + r) / 2;
73
74      copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
75      for (int i = 1; i <= m; i++)
76          id[d + 1][e[d][i].id] = i; // 准备好下一层要用的数组
77
78      CDQ(l, mid, d + 1, m, ans);
79
80      for (int i = l; i <= mid; i++)
81          e[d][id[d][a[i].id]].w = a[i].v; // 进行左边的修改
82
83      copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
84      for (int i = 1; i <= m; i++)
85          id[d + 1][e[d][i].id] = i; // 重新准备下一层要用的数组
86
87      CDQ(mid + 1, r, d + 1, m, ans);
88
89      for (int i = top; i > tmp; i--)
90          cut(stk[i]);//撤销所有操作
91      top = tmp;
92  }
93
94
95  // Reduction（减少无用边）：待修改边标为+INF，跑MST后把非树边删掉，减少无用边
96  // 需要调用Kruskal
97  void Reduction(int l, int r, int d, int &m) {
98      for (int i = l; i <= r; i++)
99          e[d][id[d][a[i].id]].w = INF;//待修改的边标为INF
100
101     Kruskal(m, e[d]);
102
103     copy(e[d] + 1, e[d] + m + 1, t + 1);
104
105     int cnt = 0;
106     for (int i = 1; i <= m; i++)
107         if (t[i].w == INF || t[i].vis){ // 非树边扔掉
108             id[d][t[i].id] = ++cnt; // 给边重新编号
109             e[d][cnt] = t[i];
110         }
```

```
111
112     for (int i = r; i >= l; i--)
113         e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改回去
114
115     m=cnt;
116 }
117
118
119 // Contraction（缩必须边）：待修改边标为-INF，跑MST后缩除待修改边之外的所有树边
120 // 返回缩掉的边的总权值
121 // 需要调用Kruskal
122 long long Contraction(int l, int r, int d, int &m) {
123     long long ans = 0;
124
125     for (int i = l; i <= r; i++)
126         e[d][id[d][a[i].id]].w = -INF; // 待修改边标为-INF
127
128     Kruskal(m, e[d]);
129     copy(e[d] + 1, e[d] + m + 1, t + 1);
130
131     int cnt = 0;
132     for (int i = 1; i <= m ; i++) {
133
134         if (t[i].w != -INF && t[i].vis) { // 必须边
135             ans += t[i].w;
136             mergeset(t[i].u, t[i].v);
137         }
138         else { // 不确定边
139             id[d][t[i].id]=++cnt;
140             e[d][cnt]=t[i];
141         }
142     }
143
144     for (int i = r ; i >= l; i--) {
145         e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改回去
146         e[d][id[d][a[i].id]].vis = false;
147     }
148
149     m = cnt;
150
151     return ans;
152 }
153
154
155 // Kruskal算法 O(mlogn)
156 // 方便起见，这里直接沿用进行过缩点的并查集，在过程结束后撤销即可
157 long long Kruskal(int m, edge *e) {
158     int tmp = top;
159     long long ans = 0;
160
161     sort(e + 1, e + m + 1); // 比较函数在结构体中定义过了
162
163     for (int i = 1; i <= m; i++) {
164         if (findroot(e[i].u) != findroot(e[i].v)) {
165             e[i].vis = true;
```

```
            ans += e[i].w;
            mergeset(e[i].u, e[i].v);
        }
        else
            e[i].vis = false;
    }

    for(int i = top; i > tmp; i--)
        cut(stk[i]); // 撤销所有操作
    top = tmp;

    return ans;
}


// 以下是并查集相关函数
int findroot(int x) { // 因为需要撤销，不写路径压缩
    while (p[x] != x)
        x = p[x];

    return x;
}

void mergeset(int x, int y) { // 按size合并，如果想跑得更快就写一个按秩合并
    x = findroot(x); // 但是按秩合并要再开一个栈记录合并之前的秩
    y = findroot(y);

    if (x == y)
        return;

    if (size[x] > size[y])
        swap(x, y);

    p[x] = y;
    size[y] += size[x];
    stk[++top] = x;
}

void cut(int x) { // 并查集撤销
    int y = x;

    do
        size[y = p[y]] -= size[x];
    while (p[y]! = y);

    p[x] = x;
}
```

可持久化左偏树求k短路

```
//Kth Shortest Path via Persistable Mergeable Heap
//可持久化可并堆求k短路 O(SSSP+(m+k)\log n)
//By AntiLeaf
//通过题目：USACO Mar08 牛跑步（板子题）
```

```

//注意这是个多项式算法，在k比较大时很有优势，但k比较小时最好还是用A*
//DAG和有环的情况都可以，有重边或自环也无所谓，但不能有零环
//以下代码以Dijkstra+可持久化左偏树为例

const int maxn=1005,maxe=10005,maxm=maxe*30;//点数，边数，左偏树结点数

//需要用到的结构体定义
struct A{//用来求最短路
    int x,d;
    A(int x,int d):x(x),d(d){}
    bool operator<(const A &a)const{return d>a.d;}
};

struct node{//左偏树结点
    int w,i,d;//i：最后一条边的编号 d：左偏树附加信息
    node *lc,*rc;
    node(){}
    node(int w,int i):w(w),i(i),d(0){}
    void refresh(){d=rc->d+1;}
}null[maxm],*ptr=null,*root[maxn];

struct B{//维护答案用
    int x,w;//x是结点编号，w表示之前已经产生的权值
    node *rt;//这个答案对应的堆顶，注意可能不等于任何一个结点的堆
    B(int x,node *rt,int w):x(x),w(w),rt(rt){}
    bool operator<(const B &a)const{return w+rt->w>a.w+a.rt->w;}
};

//全局变量和数组定义
vector<int>G[maxn],W[maxn],id[maxn];//最开始要存反向图，然后把G清空作为儿子列表
bool vis[maxn],used[maxe];//used表示边是否在最短路树上
int u[maxe],v[maxe],w[maxe];//存下每条边，注意是有向边
int d[maxn],p[maxn];//p表示最短路树上每个点的父边
int n,m,k,s,t;//s,t分别表示起点和终点

//以下是主函数中较关键的部分
for(int i=0;i<=n;i++)root[i]=null;//一定要加上！！！
//(读入&建反向图)
Dijkstra();
//(清空G,W,id)
for(int i=1;i<=n;i++)
    if(p[i]){
        used[p[i]]=true;//在最短路树上
        G[v[p[i]]].push_back(i);
    }
for(int i=1;i<=m;i++){
    w[i]-=d[u[i]]-d[v[i]];//现在的w[i]表示这条边能使路径长度增加多少
    if(!used[i])
        root[u[i]]=merge(root[u[i]],newnode(w[i],i));
}
dfs(t);
priority_queue<B>heap;
heap.push(B(s,root[s],0));//初始状态是找贡献最小的边加进去
printf("%d\n",d[s]);//第1短路需要特判
```

```
while(--k){//其余k-1短路径用二叉堆维护
    if(heap.empty())printf("-1\n");
    else{
        int x=heap.top().x,w=heap.top().w;
        node *rt=heap.top().rt;
        heap.pop();
        printf("%d\n",d[s]+w+rt->w);
        if(rt->lc!=null||rt->rc!=null)
            heap.push(B(x,merge(rt->lc,rt->rc),w));//pop掉当前边，换成另一条贡献大一点的边
        if(root[v[rt->i]]!=null)
            heap.push(B(v[rt->i],root[v[rt->i]],w+rt->w));//保留当前边，往后面再接上另一条边
    }
}
//主函数到此结束

//Dijkstra预处理最短路 O(m\log n)
void Dijkstra(){
    memset(d,63,sizeof(d));
    d[t]=0;
    priority_queue<A>heap;
    heap.push(A(t,0));
    while(!heap.empty()){
        int x=heap.top().x;
        heap.pop();
        if(vis[x])continue;
        vis[x]=true;
        for(int i=0;i<(int)G[x].size();i++)
            if(!vis[G[x][i]]&&d[G[x][i]]>d[x]+W[x][i]){
                d[G[x][i]]=d[x]+W[x][i];
                p[G[x][i]]=id[x][i];
                heap.push(A(G[x][i],d[G[x][i]]));
            }
    }
}

//dfs求出每个点的堆 总计O(m\log n)
//需要调用merge，同时递归调用自身
void dfs(int x){
    root[x]=merge(root[x],root[v[p[x]]]);
    for(int i=0;i<(int)G[x].size();i++)
        dfs(G[x][i]);
}

//包装过的new node() O(1)
node *newnode(int w,int i){
    *++ptr=node(w,i);
    ptr->lc=ptr->rc=null;
    return ptr;
}

//带可持久化的左偏树合并 总计O(\log n)
//递归调用自身
node *merge(node *x,node *y){
```

```
    if(x==null)return y;
    if(y==null)return x;
    if(x->w>y->w)swap(x,y);
    node *z=newnode(x->w,x->i);
    z->lc=x->lc;
    z->rc=merge(x->rc,y);
    if(z->lc->d>z->rc->d)swap(z->lc,z->rc);
    z->refresh();
    return z;
}
```

二分图最大权匹配

KM

```
// KM Weighted Bio-Graph Matching KM二分图最大权匹配
// By AntiLeaf
// O(n^3)

const long long INF = 0x3f3f3f3f3f3f3f3f;

long long w[maxn][maxn], lx[maxn], ly[maxn], slack[maxn];
// 边权 顶标 slack
// 如果要求最大权完美匹配就把不存在的边设为-INF 否则所有边对0取max

bool visx[maxn], visy[maxn];

int boy[maxn], girl[maxn], p[maxn], q[maxn], head, tail; // p : pre

int n, m, N, e;

// 增广
bool check(int y) {
    visy[y] = true;

    if (boy[y]) {
        visx[boy[y]] = true;
        q[tail++] = boy[y];
        return false;
    }

    while (y) {
        boy[y] = p[y];
        swap(y, girl[p[y]]);
    }

    return true;
}

// bfs每个点
void bfs(int x) {
    memset(q, 0, sizeof(q));
    head = tail = 0;

```

```
    q[tail++] = x;
    visx[x] = true;

    while (true) {
        while (head != tail) {
            int x = q[head++];

            for (int y = 1; y <= N; y++)
                if (!visy[y]) {
                    long long d = lx[x] + ly[y] - w[x][y];

                    if (d < slack[y]) {
                        p[y] = x;
                        slack[y] = d;

                        if (!slack[y] && check(y))
                            return;
                    }
                }
        }

        long long d = INF;
        for (int i = 1; i <= N; i++)
            if (!visy[i])
                d = min(d, slack[i]);

        for (int i = 1; i <= N; i++) {
            if (visx[i])
                lx[i] -= d;

            if (visy[i])
                ly[i] += d;
            else
                slack[i] -= d;
        }

        for (int i = 1; i <= N; i++)
            if (!visy[i] && !slack[i] && check(i))
                return;
    }
}

// 主过程
long long KM() {
    for (int i = 1; i <= N; i++) {
        // lx[i] = 0;
        ly[i] = -INF;
        // boy[i] = girl[i] = -1;

        for (int j = 1; j <= N; j++)
            ly[i] = max(ly[i], w[j][i]);
    }

    for (int i = 1; i <= N; i++) {
        memset(slack, 0x3f, sizeof(slack));
```

```
        memset(visx, 0, sizeof(visx));
        memset(visy, 0, sizeof(visy));
        bfs(i);
    }

    long long ans = 0;
    for (int i = 1; i <= N; i++)
        ans += w[i][girl[i]];
    return ans;
}

// 为了方便贴上主函数
int main() {

    scanf("%d%d%d", &n, &m, &e);
    N = max(n, m);

    while (e--) {
        int x, y, c;
        scanf("%d%d%d", &x, &y, &c);
        w[x][y] = max(c, 0);
    }

    printf("%lld\n", KM());

    for (int i = 1; i <= n; i++) {
        if (i > 1)
            printf(" ");
        printf("%d", w[i][girl[i]] > 0 ? girl[i] : 0);
    }
    printf("\n");

    return 0;
}
```

一般图最大匹配

高斯消元

```
// Graph Matching Based on Linear Algebra 基于线性代数的一般图匹配 O(n^3)
// By AntiLeaf
// 通过题目：UOJ#79 一般图最大匹配

// 这个算法基于Tutte定理和高斯消元，思维难度相对小一些，也更方便进行可行边的判定
// 注意这个算法复杂度是满的，并且常数有点大，而带花树通常是跑不满的
// 以及，根据Tutte定理，如果求最大匹配的大小的话直接输出Tutte矩阵的秩/2即可
// 需要输出方案时才需要再写后面那些乱七八糟的东西


// 复杂度和常数所限，1s之内500已经是这个算法的极限了
const int maxn = 505, p = 1000000007;//p可以是任意10^9以内的质数

// 全局数组和变量定义
int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn], id[maxn], a[maxn];
```

```
bool row[maxn] = {false}, col[maxn] = {false};
int n, m, girl[maxn]; // girl是匹配点，用来输出方案

// 为了方便使用，贴上主函数
// 需要调用高斯消元和eliminate
int main() {
    srand(19260817); // 膜蛤专用随机种子，换一个也无所谓

    scanf("%d%d", &n, &m); // 点数和边数
    while (m--) {
        int x, y;
        scanf("%d%d", &x, &y);
        A[x][y] = rand() % p;
        A[y][x] = -A[x][y]; // Tutte矩阵是反对称矩阵
    }

    for (int i = 1; i <= n; i++)
        id[i] = i; // 输出方案用的，因为高斯消元的时候会交换列
    memcpy(t, A, sizeof(t));
    Gauss(A, NULL, n);

    m = n;
    n = 0; // 这里变量复用纯属个人习惯……

    for (int i = 1; i <= m; i++)
        if (A[id[i]][id[i]])
            a[++n] = i; // 找出一个极大满秩子矩阵

    for (int i = 1;i <= n; i++)
        for (int j = 1; j <= n; j++)
            A[i][j]=t[a[i]][a[j]];

    Gauss(A,B,n);

    for (int i = 1; i <= n; i++)
        if (!girl[a[i]])
            for (int j = i + 1; j <= n; j++)
                if (!girl[a[j]] && t[a[i]][a[j]] && B[j][i]) {
                    // 注意上面那句if的写法，现在t是邻接矩阵的备份，
                    // 逆矩阵j行i列不为0当且仅当这条边可行
                    girl[a[i]] = a[j];
                    girl[a[j]] = a[i];
                    eliminate(i, j);
                    eliminate(j, i);
                    break;
                }

    printf("%d\n", n >> 1);
    for (int i = 1; i <= m; i++)
        printf("%d ", girl[i]);

    return 0;
}

// 高斯消元 O(n^3)
```

```
// 在传入B时表示计算逆矩阵，传入NULL则只需计算矩阵的秩
void Gauss(int A[][maxn], int B[][maxn], int n){
    if(B) {
        memset(B, 0, sizeof(t));
        for (int i = 1; i <= n; i++)
            B[i][i] = 1;
    }

    for (int i = 1; i <= n; i++) {
        if (!A[i][i]) {
            for (int j = i + 1; j <= n; j++)
                if (A[j][i]) {
                    swap(id[i], id[j]);
                    for (int k = i; k <= n; k++)
                        swap(A[i][k], A[j][k]);

                    if (B)
                        for (int k = 1; k <= n; k++)
                            swap(B[i][k], B[j][k]);
                    break;
                }

            if (!A[i][i])
                continue;
        }

        int inv = qpow(A[i][i], p - 2);

        for (int j = 1; j <= n; j++)
            if (i != j && A[j][i]){
                int t = (long long)A[j][i] * inv % p;

                for (int k = i; k <= n; k++)
                    if (A[i][k])
                        A[j][k] = (A[j][k] - (long long)t * A[i][k]) % p;

                if (B)
                    for (int k = 1; k <= n; k++)
                        if (B[i][k])
                            B[j][k] = (B[j][k] - (long long)t * B[i]
[k])%p;
            }
    }

    if (B)
        for (int i = 1; i <= n; i++) {
            int inv = qpow(A[i][i], p - 2);

            for (int j = 1; j <= n; j++)
                if (B[i][j])
                    B[i][j] = (long long)B[i][j] * inv % p;
        }
}

// 消去一行一列 O(n^2)
```

```
void eliminate(int r, int c) {
    row[r] = col[c] = true; // 已经被消掉

    int inv = qpow(B[r][c], p - 2);

    for (int i = 1; i <= n; i++)
        if (!row[i] && B[i][c]) {
            int t = (long long)B[i][c] * inv % p;

            for (int j = 1; j <= n; j++)
                if (!col[j] && B[r][j])
                    B[i][j] = (B[i][j] - (long long)t * B[r][j]) % p;
        }
}
```

一般图判定必须点/边和可行点/边时一般不会卡常，所以只提供基于Tutte矩阵的做法。

设图G的Tutte矩阵是$\tilde{A}$，则$(\tilde{A}^{-1})_{i,j} \neq 0$当且仅当$G-\{v_i,v_j\}$有完美匹配。

带花树

```
// Blossom 带花树 O(nm)
// By AntiLeaf
// 通过题目：UOJ#79 一般图最大匹配

// 带花树通常比高斯消元快很多，但在只需要求最大匹配大小的时候并没有高斯消元好写
// 当然输出方案要方便很多

// 全局数组与变量定义
vector<int> G[maxn];
int girl[maxn], f[maxn], t[maxn], p[maxn], vis[maxn], tim, q[maxn], head, tail;
int n, m;


// 封装好的主过程 O(nm)
int blossom() {
    int ans = 0;

    for (int i = 1; i <= n; i++)
        if (!girl[i])
            ans += bfs(i);

    return ans;
}


// bfs找增广路 O(m)
bool bfs(int s) {
    memset(t, 0, sizeof(t));
    memset(p, 0, sizeof(p));

    for (int i = 1; i <= n; i++)
        f[i] = i; // 并查集
```

```
33
34      head = tail = 0;
35      q[tail++] = s;
36      t[s] = 1;
37
38      while (head != tail){
39          int x = q[head++];
40          for (int y : G[x]){
41              if (findroot(y) == findroot(x) || t[y] == 2)
42                  continue;
43
44              if (!t[y]){
45                  t[y] = 2;
46                  p[y] = x;
47
48                  if (!girl[y]){
49                      for (int u = y, t; u; u = t) {
50                          t = girl[p[u]];
51                          girl[p[u]] = u;
52                          girl[u] = p[u];
53                      }
54                      return true;
55                  }
56                  t[girl[y]] = 1;
57                  q[tail++] = girl[y];
58              }
59              else if (t[y] == 1) {
60                  int z = LCA(x, y);
61                  shrink(x, y, z);
62                  shrink(y, x, z);
63              }
64          }
65      }
66
67      return false;
68  }
69
70  //缩奇环 O(n)
71  void shrink(int x, int y, int z) {
72      while (findroot(x) != z){
73          p[x] = y;
74          y = girl[x];
75
76          if (t[y] == 2) {
77              t[y] = 1;
78              q[tail++] = y;
79          }
80
81          if(findroot(x) == x)
82              f[x] = z;
83          if(findroot(y) == y)
84              f[y] = z;
85
86          x = p[y];
87      }
```

```
}

//暴力找LCA O(n)
int LCA(int x, int y) {
    tim++;
    while (true) {
        if (x) {
            x = findroot(x);

            if (vis[x] == tim)
                return x;
            else {
                vis[x] = tim;
                x = p[girl[x]];
            }
        }
        swap(x, y);
    }
}

//并查集的查找 O(1)
int findroot(int x) {
    return x == f[x] ? x : (f[x] = findroot(f[x]));
}
```

最大流

Dinic

```
// Dinic Maximum Flow 最大流（Dinic版本） O(n^2 m)
// By AntiLeaf
// 注意Dinic适用于二分图或分层图，对于一般稀疏图ISAP更优，稠密图则HLPP更优


struct edge{int to,cap,prev;}e[maxe<<1];


int last[maxn],len=0,d[maxn],cur[maxn],q[maxn];
memset(last,-1,sizeof(last));


void addedge(int x,int y,int z){
    AddEdge(x,y,z);
    AddEdge(y,x,0);
}


void AddEdge(int x,int y,int z){
    e[len].to=y;
    e[len].cap=z;
    e[len].prev=last[x];
    last[x]=len++;
}

```

```
26
27  int Dinic(){
28      int flow=0;
29      while(bfs(),d[t]!=-1){
30          memcpy(cur,last,sizeof(int)*(t+5));
31          flow+=dfs(s,(~0u)>>1);
32      }
33      return flow;
34  }
35
36
37  void bfs(){
38      int head=0,tail=0;
39      memset(d,-1,sizeof(int)*(t+5));
40      q[tail++]=s;
41      d[s]=0;
42      while(head!=tail){
43          int x=q[head++];
44          for(int i=last[x];i!=-1;i=e[i].prev)
45              if(e[i].cap>0&&d[e[i].to]==-1){
46                  d[e[i].to]=d[x]+1;
47                  q[tail++]=e[i].to;
48              }
49      }
50  }
51
52
53  int dfs(int x,int a){
54      if(x==t||!a)return a;
55      int flow=0,f;
56      for(int &i=cur[x];i!=-1;i=e[i].prev)
57          if(e[i].cap>0&&d[e[i].to]==d[x]+1&&
    (f=dfs(e[i].to,min(e[i].cap,a)))){
58              e[i].cap-=f;
59              e[i^1].cap+=f;
60              flow+=f;
61              a-=f;
62              if(!a)break;
63          }
64      return flow;
65  }
```

ISAP

```
1  // Improved Shortest Augment Path Algorighm 最大流（ISAP版本） O(n^2 m)
2  // By AntiLeaf
3  // 注意ISAP适用于一般稀疏图，对于二分图或分层图情况Dinic比较优，稠密图则HLPP更优
4
5
6  // 边的定义
7  // 这里没有记录起点和反向边，因为反向边即为正向边xor 1，起点即为反向边的终点
8  struct edge{
9      int to, cap, prev;
10 } e[maxe * 2];
```

```
11
12
13  // 全局变量和数组定义
14  int last[maxn], cnte = 0, d[maxn], p[maxn], c[maxn], cur[maxn], q[maxn];
15  int n, m, s, t; // s, t一定要开成全局变量
16
17
18  // 重要！！！
19  // main函数最前面一定要加上如下初始化
20  memset(last, -1, sizeof(last));
21
22
23  // 加边函数 O(1)
24  // 包装了加反向边的过程，方便调用
25  // 需要调用AddEdge
26  void addedge(int x, int y, int z) {
27      AddEdge(x, y, z);
28      AddEdge(y, x, 0);
29  }
30
31
32  // 真·加边函数 O(1)
33  void AddEdge(int x, int y, int z){
34      e[cnte].to = y;
35      e[cnte].cap = z;
36      e[cnte].prev = last[x];
37      last[x] = cnte++;
38  }
39
40
41  // 主过程 O(n^2 m)
42  // 返回最大流的流量
43  // 需要调用bfs、augment
44  // 注意这里的n是编号最大值，在这个值不为n的时候一定要开个变量记录下来并修改代码
45  // 非递归
46  int ISAP() {
47      bfs();
48
49      memcpy(cur, last, sizeof(cur));
50
51      int x = s, flow = 0;
52
53      while (d[s] < n) {
54          if (x == t) {//如果走到了t就增广一次，并返回s重新找增广路
55              flow += augment();
56              x = s;
57          }
58
59          bool ok = false;
60          for (int &i = cur[x]; ~i; i = e[i].prev)
61              if (e[i].cap && d[x] == d[e[i].to] + 1) {
62                  p[e[i].to] = i;
63                  x = e[i].to;
64
65                  ok = true;
```

```
                break;
            }

        if (!ok) { // 修改距离标号
            int tmp = n - 1;
            for (int i = last[x]; ~i; i = e[i].prev)
                if (e[i].cap)
                    tmp = min(tmp, d[e[i].to] + 1);

            if (!--c[d[x]])
                break; // gap优化，一定要加上

            c[d[x] = tmp]++;
            cur[x] = last[x];

            if(x != s)
                x = e[p[x] ^ 1].to;
        }
    }
    return flow;
}

// bfs函数 O(n+m)
// 预处理到t的距离标号
// 在测试数据组数较少时可以省略，把所有距离标号初始化为0
void bfs(){
    memset(d, -1, sizeof(d));

    int head = 0, tail = 0;
    d[t] = 0;
    q[tail++] = t;

    while (head != tail) {
        int x = q[head++];
        c[d[x]]++;

        for (int i = last[x]; ~i; i = e[i].prev)
            if (e[i ^ 1].cap && d[e[i].to] == -1) {
                d[e[i].to] = d[x] + 1;
                q[tail++] = e[i].to;
            }
    }
}

// augment函数 O(n)
// 沿增广路增广一次，返回增广的流量
int augment() {
    int a = (~0u) >> 1; // INT_MAX

    for (int x = t; x != s; x = e[p[x] ^ 1].to)
        a = min(a, e[p[x]].cap);

    for (int x = t; x != s; x = e[p[x] ^ 1].to){
        e[p[x]].cap -= a;
        e[p[x] ^ 1].cap += a;
```

```
    }

    return a;
}
```

最高标号预流推进（HLPP）

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using std::min;
using std::vector;
using std::queue;
using std::priority_queue;
const int N=2e4+5,M=2e5+5,inf=0x3f3f3f3f;
int n,s,t,tot;
int v[M<<1],w[M<<1],first[N],next[M<<1];
int h[N],e[N],gap[N<<1],inq[N];//节点高度是可以到达2n-1的
struct cmp
{
    inline bool operator()(int a,int b) const
    {
        return h[a]<h[b];//因为在优先队列中的节点高度不会改变，所以可以直接比较
    }
};
queue<int> Q;
priority_queue<int,vector<int>,cmp> pQ;
inline void add_edge(int from,int to,int flow)
{
    tot+=2;
    v[tot+1]=from;v[tot]=to;w[tot]=flow;w[tot+1]=0;
    next[tot]=first[from];first[from]=tot;
    next[tot+1]=first[to];first[to]=tot+1;
    return;
}
inline bool bfs()
{
    int now;
    register int go;
    memset(h+1,0x3f,sizeof(int)*n);
    h[t]=0;Q.push(t);
    while(!Q.empty())
    {
        now=Q.front();Q.pop();
        for(go=first[now];go;go=next[go])
            if(w[go^1]&&h[v[go]]>h[now]+1)
                h[v[go]]=h[now]+1,Q.push(v[go]);
    }
    return h[s]!=inf;
}
inline void push(int now)//推送
{
    int d;
```

```
48      register int go;
49      for(go=first[now];go;go=next[go])
50          if(w[go]&&h[v[go]]+1==h[now])
51          {
52              d=min(e[now],w[go]);
53              w[go]-=d;w[go^1]+=d;e[now]-=d;e[v[go]]+=d;
54              if(v[go]!=s&&v[go]!=t&&!inq[v[go]])
55                  pQ.push(v[go]),inq[v[go]]=1;
56              if(!e[now])//已经推送完毕可以直接退出
57                  break;
58          }
59      return;
60  }
61  inline void relabel(int now)//重贴标签
62  {
63      register int go;
64      h[now]=inf;
65      for(go=first[now];go;go=next[go])
66          if(w[go]&&h[v[go]]+1<h[now])
67              h[now]=h[v[go]]+1;
68      return;
69  }
70  inline int hlpp()
71  {
72      int now,d;
73      register int i,go;
74      if(!bfs())//s和t不连通
75          return 0;
76      h[s]=n;
77      memset(gap,0,sizeof(int)*(n<<1));
78      for(i=1;i<=n;i++)
79          if(h[i]<inf)
80              ++gap[h[i]];
81      for(go=first[s];go;go=next[go])
82          if(d=w[go])
83          {
84              w[go]-=d;w[go^1]+=d;e[s]-=d;e[v[go]]+=d;
85              if(v[go]!=s&&v[go]!=t&&!inq[v[go]])
86                  pQ.push(v[go]),inq[v[go]]=1;
87          }
88      while(!pQ.empty())
89      {
90          inq[now=pQ.top()]=0;pQ.pop();push(now);
91          if(e[now])
92          {
93              if(!--gap[h[now]])//gap优化，因为当前节点是最高的所以修改的节点一定不
    在优先队列中，不必担心修改对优先队列会造成影响
94                  for(i=1;i<=n;i++)
95                      if(i!=s&&i!=t&&h[i]>h[now]&&h[i]<n+1)
96                          h[i]=n+1;
97              relabel(now);++gap[h[now]];
98              pQ.push(now);inq[now]=1;
99          }
100     }
101     return e[t];
```

```
}
int m;
signed main()
{
    int u,v,w;
    scanf("%d%d%d%d",&n,&m,&s,&t);
    while(m--)
    {
        scanf("%d%d%d",&u,&v,&w);
        add_edge(u,v,w);
    }
    printf("%d\n",hlpp());
    return 0;
}
```

网络流原理

最小割

最小割输出一种方案

在残量网络上从源点开始floodfill，如果一条边的起点从源点可达，而终点不可达，那么这条边就在最小割中。

最小割的可行边与必须边

可行边

满流且残量网络上不存在起点到终点的路径，也就是起点和终点不在同一SCC中。

必须边

满流且残量网络上源点可达起点，终点可达汇点。

二分图

最大匹配的可行边与必须边

可行边

一条边的两个端点在同一个SCC中，不论是正边还是反边。

必须边

一条属于当前最大匹配的边，且两个端点不在同一个SCC中。

独立集

二分图独立集可以看成最小割问题，割掉最少的点使得剩下的点都在独立集中。

（割点等价于割掉它与源点或汇点相连的边）

所以独立集输出方案就是求出不在最小割中的点，独立集的必须点/可行点就是最小割的不可行点/非必须点。

字符串

AC自动机

```
// Aho-Corasick Automata AC自动机
// By AntiLeaf
// 通过题目：bzoj3881 Divljak


// 全局变量与数组定义
int ch[maxm][26] = {{0}}, f[maxm][26] = {{0}}, q[maxm] = {0}, sum[maxm] =
{0}, cnt = 0;


// 在字典树中插入一个字符串 O(n)
int insert(const char *c) {
    int x = 0;
    while (*c) {
        if (!ch[x][*c - 'a'])
            ch[x][*c - 'a'] = ++cnt;
        x = ch[x][*c++ - 'a'];
    }
    return x;
}


// 建AC自动机 O(n*sigma)
void getfail() {
    int x, head = 0, tail = 0;

    for (int c = 0; c < 26; c++)
        if (ch[0][c])
            q[tail++] = ch[0][c]; // 把根节点的儿子加入队列

    while (head != tail) {
        x = q[head++];

        G[f[x][0]].push_back(x);
        fill(f[x] + 1, f[x] + 26, cnt + 1);

        for (int c = 0; c < 26; c++) {
            if (ch[x][c]) {
                int y = f[x][0];

                while (y&&!ch[y][c])
                    y=f[y][0];

                f[ch[x][c]][0] = ch[y][c];
                q[tail++] = ch[x][c];
            }
            else
                ch[x][c] = ch[f[x][0]][c];
        }
    }
    fill(f[0], f[0] + 26, cnt + 1);
}
```

后缀数组

SAMSA

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=100005;
void expand(int);
void dfs(int);
int root,last,cnt=0,val[maxn<<1]={0},par[maxn<<1]={0},go[maxn<<1][26]={{0}};
bool vis[maxn<<1]={0};
char s[maxn];
int n,id[maxn<<1]={0},ch[maxn<<1][26]={{0}},height[maxn],tim=0;
int main(){
    root=last=++cnt;
    scanf("%s",s+1);
    n=strlen(s+1);
    for(int i=n;i;i--){
        expand(s[i]-'a');
        id[last]=i;
    }
    vis[1]=true;
    for(int i=1;i<=cnt;i++)if(id[i])for(int x=i,pos=n;x&&!vis[x];x=par[x]){
        vis[x]=true;
        pos-=val[x]-val[par[x]];
        ch[par[x]][s[pos+1]-'a']=x;
    }
    dfs(root);
    printf("\n");
    for(int i=1;i<n;i++)printf("%d ",height[i]);
    return 0;
}
void expand(int c){
    int p=last,np=++cnt;
    val[np]=val[p]+1;
    while(p&&!go[p][c]){
        go[p][c]=np;
        p=par[p];
    }
    if(!p)par[np]=root;
    else{
        int q=go[p][c];
        if(val[q]==val[p]+1)par[np]=q;
        else{
            int nq=++cnt;
            val[nq]=val[p]+1;
            memcpy(go[nq],go[q],sizeof(go[q]));
            par[nq]=par[q];
            par[np]=par[q]=nq;
            while(p&&go[p][c]==q){
                go[p][c]=nq;
```

```
                p=par[p];
            }
        }
    }
    last=np;
}
void dfs(int x){
    if(id[x]){
        printf("%d ",id[x]);
        height[tim++]=val[last];
        last=x;
    }
    for(int c=0;c<26;c++)if(ch[x][c])dfs(ch[x][c]);
    last=par[x];
}
```

后缀自动机

```
//Suffix Automaton 后缀自动机 O(n)
//By AntiLeaf
//通过题目：Bzoj3473 字符串

//在字符集比较小的时候可以直接开go数组，否则需要用map或者哈希表替换
//注意！！！结点数要开成串长的两倍

//全局变量与数组定义
int last,val[maxn],par[maxn],go[maxn][26],cnt;
int c[maxn],q[maxn];//用来桶排序

//在主函数开头加上这句初始化
last=cnt=1;

//以下是按val进行桶排序的代码
for(int i=1;i<=cnt;i++)c[val[i]+1]++;
for(int i=1;i<=n;i++)c[i]+=c[i-1];//这里n是串长
for(int i=1;i<=cnt;i++)q[++c[val[i]]]=i;

//加入一个字符 均摊O(1)
void extend(int c){
    int p=last,np=++cnt;
    val[np]=val[p]+1;
    while(p&&!go[p][c]){
        go[p][c]=np;
        p=par[p];
    }
    if(!p)par[np]=1;
    else{
        int q=go[p][c];
        if(val[q]==val[p]+1)par[np]=q;
        else{
            int nq=++cnt;
            val[nq]=val[p]+1;
            memcpy(go[nq],go[q],sizeof(go[q]));
            par[nq]=par[q];
```

```
            par[np]=par[q]=nq;
            while(p&&go[p][c]==q){
                go[p][c]=nq;
                p=par[p];
            }
        }
    }
    last=np;
}
```

回文树

```
//Palindromic Tree/EERTREE 回文树 O(n)
//By AntiLeaf
//通过题目：APIO2014 回文串

//定理：一个字符串本质不同的回文子串个数是O(n)的
//注意回文树只需要开一倍结点，另外结点编号是一个可用的bfs序

//全局数组定义
int val[maxn],par[maxn],go[maxn][26],last,cnt;
char s[maxn];

//重要！在主函数最前面一定要加上以下初始化
par[0]=cnt=1;
val[1]=-1;

//extend函数 均摊O(1)
//向后扩展一个字符
//传入对应下标
void extend(int n){
    int p=last,c=s[n]-'a';
    while(s[n-val[p]-1]!=s[n])p=par[p];
    if(!go[p][c]){
        int q=++cnt,now=p;
        val[q]=val[p]+2;
        do p=par[p];while(s[n-val[p]-1]!=s[n]);
        par[q]=go[p][c];
        last=go[now][c]=q;
    }
    else last=go[p][c];
    a[last]++;
}

```

广义回文树

```
#include <bits/stdc++.h>

using namespace std;

constexpr int maxn = 1000005, mod = 1000000007;

```

```
int val[maxn], par[maxn], go[maxn][26], fail[maxn][26], pam_last[maxn], pam_cnt;
int weight[maxn], pow_26[maxn];

int trie[maxn][26], trie_cnt, d[maxn], mxd[maxn], son[maxn], top[maxn], len[maxn], sum[maxn];
char chr[maxn];
int f[25][maxn], log_tbl[maxn];
vector<int> v[maxn];

vector<int> queries[maxn];

char str[maxn];
int n, m, ans[maxn];

int add(int x, int c) {
    if (!trie[x][c]) {
        trie[x][c] = ++trie_cnt;
        f[0][trie[x][c]] = x;
        chr[trie[x][c]] = c + 'a';
    }

    return trie[x][c];
}

int del(int x) {
    return f[0][x];
}

void dfs1(int x) {
    mxd[x] = d[x] = d[f[0][x]] + 1;

    for (int i = 0; i < 26; i++)
        if (trie[x][i]) {
            int y = trie[x][i];

            dfs1(y);

            mxd[x] = max(mxd[x], mxd[y]);
            if (mxd[y] > mxd[son[x]])
                son[x] = y;
        }
}

void dfs2(int x) {
    if (x == son[f[0][x]])
        top[x] = top[f[0][x]];
    else
        top[x] = x;

    for (int i = 0; i < 26; i++)
        if (trie[x][i]) {
            int y = trie[x][i];
            dfs2(y);
        }
```

```
60
61      if (top[x] == x) {
62          int u = x;
63          while (top[son[u]] == x)
64              u = son[u];
65
66          len[x] = d[u] - d[x];
67
68          for (int i = 0; i < len[x]; i++) {
69              v[x].push_back(u);
70              u = f[0][u];
71          }
72
73          u = x;
74          for (int i = 0; i < len[x]; i++) { // 梯子剖分，要延长一倍
75              v[x].push_back(u);
76              u = f[0][u];
77          }
78      }
79  }
80
81  int get_anc(int x, int k) {
82      if (!k)
83          return x;
84      if (k > d[x])
85          return 0;
86
87      x = f[log_tbl[k]][x];
88      k ^= 1 << log_tbl[k];
89
90      return v[top[x]][d[top[x]] + len[top[x]] - d[x] + k];
91  }
92
93  char get_char(int x, int k) { // 查询x前面k个的字符是哪个
94      return chr[get_anc(x, k)];
95  }
96
97  int getfail(int x, int p) {
98      if (get_char(x, val[p] + 1) == chr[x])
99          return p;
100     return fail[p][chr[x] - 'a'];
101 }
102
103 int extend(int x) {
104
105     int p = pam_last[f[0][x]], c = chr[x] - 'a';
106
107     p = getfail(x, p);
108
109     int new_last;
110
111     if (!go[p][c]) {
112         int q = ++pam_cnt, now = p;
113         val[q] = val[p] + 2;
114
```

```
        p = getfail(x, par[p]);

        par[q] = go[p][c];
        new_last = go[now][c] = q;

        for (int i = 0; i < 26; i++)
            fail[q][i] = fail[par[q]][i];

        if (get_char(x, val[par[q]]) >= 'a')
            fail[q][get_char(x, val[par[q]]) - 'a'] = par[q];

        if (val[q] <= n)
            weight[q] = (weight[par[q]] + (long long)(n - val[q] + 1) *
pow_26[n - val[q]]) % mod;
        else
            weight[q] = weight[par[q]];
    }
    else
        new_last = go[p][c];

    pam_last[x] = new_last;

    return weight[pam_last[x]];
}

void bfs() {

    queue<int> q;

    q.push(1);

    while (!q.empty()) {
        int x = q.front();
        q.pop();

        sum[x] = sum[f[0][x]];
        if (x > 1)
            sum[x] = (sum[x] + extend(x)) % mod;

        for (int i : queries[x])
            ans[i] = sum[x];

        for (int i = 0; i < 26; i++)
            if (trie[x][i])
                q.push(trie[x][i]);
    }

}

int main() {

    pow_26[0] = 1;
    log_tbl[0] = -1;

    for (int i = 1; i <= 1000000; i++) {
```

```
        pow_26[i] = 26ll * pow_26[i - 1] % mod;
        log_tbl[i] = log_tbl[i / 2] + 1;
    }

    int T;
    scanf("%d", &T);

    while (T--) {
        scanf("%d%d%s", &n, &m, str);

        trie_cnt = 1;
        chr[1] = '#';

        int last = 1;
        for (char *c = str; *c; c++)
            last = add(last, *c - 'a');

        queries[last].push_back(0);

        for (int i = 1; i <= m; i++) {
            int op;
            scanf("%d", &op);

            if (op == 1) {
                char c;
                scanf(" %c", &c);

                last = add(last, c - 'a');
            }
            else
                last = del(last);

            queries[last].push_back(i);
        }

        dfs1(1);
        dfs2(1);

        for (int j = 1; j <= log_tbl[trie_cnt]; j++)
            for (int i = 1; i <= trie_cnt; i++)
                f[j][i] = f[j - 1][f[j - 1][i]];

        par[0] = pam_cnt = 1;


        for (int i = 0; i < 26; i++)
            fail[0][i] = fail[1][i] = 1;

        val[1] = -1;
        pam_last[1] = 1;

        bfs();

        for (int i = 0; i <= m; i++)
            printf("%d\n", ans[i]);
```

```
224
225         for (int j = 0; j <= log_tbl[trie_cnt]; j++)
226             memset(f[j], 0, sizeof(f[j]));
227
228         for (int i = 1; i <= trie_cnt; i++) {
229             chr[i] = 0;
230             d[i] = mxd[i] = son[i] = top[i] = len[i] = pam_last[i] =
sum[i] = 0;
231             v[i].clear();
232             queries[i].clear();
233
234             memset(trie[i], 0, sizeof(trie[i]));
235         }
236         trie_cnt = 0;
237
238         for (int i = 0; i <= pam_cnt; i++) {
239             val[i] = par[i] = weight[i];
240
241             memset(go[i], 0, sizeof(go[i]));
242             memset(fail[i], 0, sizeof(fail[i]));
243         }
244         pam_cnt = 0;
245
246     }
247
248     return 0;
249 }
```

Manacher / 马拉车

```
1  //Manacher O(n)
2  //By AntiLeaf
3  //通过题目：51Nod1089 最长回文子串V2
4
5  //n为串长，回文半径输出到p数组中
6  //数组要开串长的两倍
7  void manacher(const char *t, int n) {
8      static char s[maxn * 2];
9
10     for (int i = n; i; i--)
11         s[i * 2] = t[i];
12     for (int i = 0; i <= n; i++)
13         s[i * 2 + 1]='#';
14
15     s[0] = '$';
16     s[(n + 1) * 2] = '\0';
17     n = n * 2 + 1;
18
19     int mx = 0, j = 0;
20
21     for (int i = 1; i <= n; i++) {
22         p[i] = (mx > i ? min(p[j * 2 - i], mx - i) : 1);
```

```
        while (s[i - p[i]] == s[i + p[i]])
            p[i]++;

        if(i + p[i] > mx){
            mx = i + p[i];
            j = i;
        }
    }
}
```

ex-KMP

```
//Extended KMP 扩展KMP
//By AntiLeaf
//通过题目：小作业OJ 4182

//全局变量与数组定义
char s[maxn], t[maxn];
int n, m, a[maxn];

//主过程 O(n + m)
//把t的每个后缀与s的LCP输出到a中，s的后缀和自己的LCP存在nx中
//0-based，s的长度是m，t的长度是n
void exKMP(const char *s, const char *t, int *a) {
    static int nx[maxn];

    memset(nx, 0, sizeof(nx));

    int j = 0;
    while (j + 1 < m && s[j] == s[j + 1])
        j++;
    nx[1] = j;

    for (int i = 2, k = 1;i < m; i++) {
        int pos = k + nx[k], len = nx[i - k];

        if (i + len < pos)
            nx[i] = len;
        else {
            j = max(pos - i, 0);
            while (i + j < m && s[j] == s[i + j])
                j++;

            nx[i] = j;
            k = i;
        }
    }

    j = 0;
    while (j < n && j < m && s[j] == t[j])
        j++;
    a[0] = j;

    for (int i = 1, k = 0; i < n; i++) {
```

```
        int pos = k + a[k], len = nx[i - k];
        if (i + len < pos)
            a[i] = len;
        else {
            j = max(pos - i, 0);
            while(j < m && i + j < n && s[j] == t[i + j])
                j++;

            a[i] = j;
            k = i;
        }
    }
}
```

数学

FWT

```
//Fast Walsh-Hadamard Transform 快速沃尔什变换 O(n\log n)
//By AntiLeaf
//通过题目：COGS上几道板子题

//注意FWT常数比较小，这点与FFT/NTT不同
//以下代码均以模质数情况为例，其中n为变换长度，tp表示正/逆变换

//按位或版本
void FWT_or(int *A,int n,int tp){
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                if(tp>0)A[i+j+(k>>1)]=(A[i+j+(k>>1)]+A[i+j])%p;
                else A[i+j+(k>>1)]=(A[i+j+(k>>1)]-A[i+j]+p)%p;
            }
}

//按位与版本
void FWT_and(int *A,int n,int tp){
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                if(tp>0)A[i+j]=(A[i+j]+A[i+j+(k>>1)])%p;
                else A[i+j]=(A[i+j]-A[i+j+(k>>1)]+p)%p;
            }
}

//按位异或版本
void FWT_xor(int *A,int n,int tp){
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                int a=A[i+j],b=A[i+j+(k>>1)];
                A[i+j]=(a+b)%p;
                A[i+j+(k>>1)]=(a-b+p)%p;
```

```
            }
    if(tp<0){
        int inv=qpow(n%p,p-2);//n的逆元，在不取模时需要用每层除以2代替
        for(int i=0;i<n;i++)A[i]=A[i]*inv%p;
    }
}
```

插值

牛顿插值

牛顿插值的原理是**二项式反演**。

二项式反演：

$$f(n)=\sum_{k=0}^{n}\binom{n}{k}g(k)\ \Leftrightarrow\ g(n)=\sum_{k=0}^{n}(-1)^{n-k}\binom{n}{k}f(k)$$

可以用e^x和e^{-x}的麦克劳林展开式证明。

套用二项式反演的结论即可得到牛顿插值：

$$f(n)=\sum_{i=0}^{k}\binom{n}{i}r_i$$

$$r_i=\sum_{j=0}^{i}(-1)^{i-j}\binom{i}{j}f(j)$$

其中k表示$f(n)$的最高次项系数。

实现时可以用k次差分替代右边的式子。

```
for(int i=0;i<=k;i++)
    r[i]=f(i);
for(int j=0;j<k;j++)
    for(int i=k;i>j;i--)
        r[i]-=r[i-1];
```

注意到预处理r_i的式子满足卷积形式，必要时可以用FFT优化至$O(k\log k)$预处理。

拉格朗日插值

$$f(x)=\sum_i f(x_i)\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}$$

多项式

FFT

```
//Fast Fourier Transform 快速傅里叶变换 O(n\log n)
//By AntiLeaf
//通过题目：COGS2294 释迦（作为拆系数FFT的组成部分）
//使用时一定要注意double的精度是否足够（极限大概是10^14）

const double pi=acos((double)-1.0);
```

```

//手写复数类
//支持加减乘三种运算
//+=运算符如果用的不多可以不重载
struct Complex{
    double a,b;//由于long double精度和double几乎相同，通常没有必要用long double
    Complex(double a=0.0,double b=0.0):a(a),b(b){}
    Complex operator+(const Complex &x)const{return Complex(a+x.a,b+x.b);}
    Complex operator-(const Complex &x)const{return Complex(a-x.a,b-x.b);}
    Complex operator*(const Complex &x)const{return Complex(a*x.a-b*x.b,a*x.b+b*x.a);}
    Complex &operator+=(const Complex &x){return *this=*this+x;}
}w[maxn],w_inv[maxn];

//FFT初始化 O(n)
//需要调用sin、cos函数
void FFT_init(int n){
    for(int i=0;i<n;i++)//根据单位根的旋转性质可以节省计算单位根逆元的时间
        w[i]=w_inv[n-i-1]=Complex(cos(2*pi/n*i),sin(2*pi/n*i));
    //当然不存单位根也可以，只不过在FFT次数较多时很可能会增大常数
}

//FFT主过程 O(n\log n)
void FFT(Complex *A,int n,int tp){
    for(int i=1,j=0,k;i<n-1;i++){
        k=n;
        do j^=(k>>=1);while(j<k);
        if(i<j)swap(A[i],A[j]);
    }
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                Complex a=A[i+j],b=(tp>0?w:w_inv)[n/k*j]*A[i+j+(k>>1)];
                A[i+j]=a+b;
                A[i+j+(k>>1)]=a-b;
            }
    if(tp<0)for(int i=0;i<n;i++)A[i].a/=n;
}
```

NTT

```
// Number Theory Transform 快速数论变换 O(n\log n)
// By AntiLeaf
// 通过题目：UOJ#34 多项式乘法
// 要求模数为10^9以内的NTT模数

const int p = 998244353, g = 3; // p为模数，g为p的任意一个原根

void NTT(int *A, int n, int tp) { // n为变换长度，tp为1或-1，表示正/逆变换
    for (int i = 1, j = 0, k; i < n - 1; i++) { // O(n)旋转算法，原理是模拟二进制加一
        k = n;
        do
            j ^= (k >>= 1);
```

```
        while (j < k);

        if(i < j)
            swap(A[i], A[j]);
    }

    for (int k = 2; k <= n; k <<= 1) {
        int wn = qpow(g, (tp > 0 ? (p - 1) / k : (p - 1) / k * (long long)(p - 2) % (p - 1)));
        for (int i = 0; i < n; i += k) {
            int w = 1;
            for (int j = 0; j < (k >> 1); j++, w = (long long)w * wn % p){
                int a = A[i + j], b = (long long)w * A[i + j + (k >> 1)] % p;
                A[i + j] = (a + b) % p;
                A[i + j + (k >> 1)] = (a - b + p) % p;
            } // 更好的写法是预处理单位根的次幂，参照FFT的代码
        }
    }

    if (tp < 0) {
        int inv = qpow(n, p - 2); // 如果预处理过逆元的话就不用快速幂了
        for (int i = 0; i < n; i++)
            A[i] = (long long)A[i] * inv % p;
    }
}
```

多项式操作

```
//Polymial Operations 多项式操作
//By AntiLeaf
//通过题目：COGS2189 帕秋莉的超级多项式（板子题）

const int maxn=262200;//以下所有代码均为NTT版本
//以下所有代码均满足：A为输入（不进行修改），C为输出，n为所需长度

//多项式求逆 O(n\log n)
//要求A常数项不为0
void getinv(int *A,int *C,int n){
    static int B[maxn];
    memset(C,0,sizeof(int)*(n<<1));
    C[0]=qpow(A[0],p-2);//一般题目直接赋值为1就可以
    for(int k=2;k<=n;k<<=1){
        memcpy(B,A,sizeof(int)*k);
        memset(B+k,0,sizeof(int)*k);
        NTT(B,k<<1,1);
        NTT(C,k<<1,1);
        for(int i=0;i<(k<<1);i++)
            C[i]=((2-(long long)B[i]*C[i])%p*C[i]%p+p)%p;
        NTT(C,k<<1,-1);
        memset(C+k,0,sizeof(int)*k);
    }
}

```

```
//多项式开根 O(n\log n)
//要求A常数项可以开根/存在二次剩余
//需要调用多项式求逆，且需要预处理2的逆元
void getsqrt(int *A,int *C,int n){
    static int B[maxn],D[maxn];
    memset(C,0,sizeof(int)*(n<<1));
    C[0]=(int)(sqrt(A[0])+1e-7);//一般题目直接赋值为1就可以
    for(int k=2;k<=n;k<<=1){
        memcpy(B,A,sizeof(int)*k);
        memset(B+k,0,sizeof(int)*k);
        getinv(C,D,k);
        NTT(B,k<<1,1);
        NTT(D,k<<1,1);
        for(int i=0;i<(k<<1);i++)B[i]=(long long)B[i]*D[i]%p;
        NTT(B,k<<1,-1);
        for(int i=0;i<k;i++)C[i]=(long long)(C[i]+B[i])*inv_2%p;//inv_2是2的逆元
    }
}

//求导 O(n)
void getderivative(int *A,int *C,int n){
    for(int i=1;i<n;i++)C[i-1]=(long long)A[i]*i%p;
    C[n-1]=0;
}

//不定积分 O(n\log n)，如果预处理过逆元可以降到O(n)
void getintegrate(int *A,int *C,int n){
    for(int i=1;i<n;i++)C[i]=(long long)A[i-1]*qpow(i,p-2)%p;
    C[0]=0;//由于是不定积分，结果没有常数项
}

//多项式\ln O(n\log n)
//要求A常数项不为0/存在离散对数
//需要调用多项式求逆、求导、不定积分
void getln(int *A,int *C,int n){//通常情况下A常数项都是1
    static int B[maxn];
    getderivative(A,B,n);
    memset(B+n,0,sizeof(int)*n);
    getinv(A,C,n);
    NTT(B,n<<1,1);
    NTT(C,n<<1,1);
    for(int i=0;i<(n<<1);i++)B[i]=(long long)B[i]*C[i]%p;
    NTT(B,n<<1,-1);
    getintegrate(B,C,n);
    memset(C+n,0,sizeof(int)*n);
}

//多项式\exp O(n\log n)
//要求A没有常数项
//需要调用多项式\ln
//常数很大且总代码较长，在时间效率要求不高时最好替换为分治FFT
//分治FFT依据：设G(x)=\exp F(x)，则有g_i=\sum_{k=1}^i f_k g_{i-k}
void getexp(int *A,int *C,int n){
    static int B[maxn];
```

```
    memset(C,0,sizeof(int)*(n<<1));
    C[0]=1;
    for(int k=2;k<=n;k<<=1){
        getln(C,B,k);
        for(int i=0;i<k;i++){
            B[i]=A[i]-B[i];
            if(B[i]<0)B[i]+=p;
        }
        (++B[0])%=p;
        NTT(B,k<<1,1);
        NTT(C,k<<1,1);
        for(int i=0;i<(k<<1);i++)C[i]=(long long)C[i]*B[i]%p;
        NTT(C,k<<1,-1);
        memset(C+k,0,sizeof(int)*k);
    }
}

//多项式k次幂 O(n\log n)
//在A常数项不为1时需要转化
//需要调用多项式/exp、\ln
//常数较大且总代码较长，在时间效率要求不高时最好替换为暴力快速幂
void getpow(int *A,int *C,int n,int k){
    static int B[maxn];
    getln(A,B,n);
    for(int i=0;i<n;i++)B[i]=(long long)B[i]*k%p;
    getexp(B,C,n);
}
```

拉格朗日反演

用于求复合逆。

如果$f(x)$与$g(x)$互为复合逆，则有

$$g(x)=\frac{1}{n}[x^{n-1}]\left(\frac{x}{f(x)}\right)^n$$

$$h(g(x))=\frac{1}{n}[x^{n-1}]h'(x)\left(\frac{x}{f(x)}\right)^n$$

分治FFT

半在线卷积

模板题：给定结点的生成函数，求有根树的生成函数

```
void solve(int l, int r) {
    if (r <= m)
        return;

    if (r - l == 1) {
        if (l == m)
            f[l] = a[m];
        else
            f[l] = (long long)f[l] * inv[l - m] % p;

```

```
        for (int i = l, t = (long long)l * f[l] % p; i <= n; i += l)
            g[i] = (g[i] + t) % p;

        return;
    }

    int mid = (l + r) / 2;

    solve(l, mid);

    if (l == 0) {
        for (int i = 1; i < mid; i++) {
            A[i] = f[i];
            B[i] = (c[i] + g[i]) % p;
        }
        NTT(A, r, 1);
        NTT(B, r, 1);
        for (int i = 0; i < r; i++)
            A[i] = (long long)A[i] * B[i] % p;
        NTT(A, r, -1);

        for (int i = mid; i < r; i++) {
            f[i] = (f[i] + A[i]) % p;
        }
    }
    else {
        for (int i = 0; i < r - l; i++)
            A[i] = f[i];
        for (int i = l; i < mid; i++)
            B[i - l] = (c[i] + g[i]) % p;
        NTT(A, r - l, 1);
        NTT(B, r - l, 1);
        for (int i = 0; i < r - l; i++)
            A[i] = (long long)A[i] * B[i] %p;
        NTT(A, r - l, -1);

        for (int i = mid; i < r; i++) {
            f[i] = (f[i] + A[i - l]) % p;
        }

        memset(A, 0, sizeof(int) * (r - l));
        memset(B, 0, sizeof(int) * (r - l));

        for (int i = l; i < mid; i++)
            A[i - l] = f[i];
        for (int i = 0; i < r - l; i++)
            B[i] = (c[i] + g[i]) % p;
        NTT(A, r - l, 1);
        NTT(B, r - l, 1);
        for (int i = 0; i < r - l; i++)
            A[i] = (long long)A[i] * B[i] % p;
        NTT(A, r - l, -1);

        for (int i = mid; i < r; i++) {
            f[i] = (f[i] + A[i - l]) % p;
```

```
        }
    }

    memset(A, 0, sizeof(int) * (r - l));
    memset(B, 0, sizeof(int) * (r - l));

    solve(mid, r);
}
```

单纯形

```
//Simplex Method 单纯形方法求解线性规划
//By AntiLeaf
//通过题目：UOJ#179 线性规划（然而被hack了QAQ……）

//单纯形其实是指数算法，但实践中跑得飞快，所以复杂度什么的也就无所谓了


const double eps=1e-10;

double A[maxn][maxn],x[maxn];
int n,m,t,id[maxn<<1];

//方便起见，这里附上主函数
int main(){
    scanf("%d%d%d",&n,&m,&t);
    for(int i=1;i<=n;i++){
        scanf("%lf",&A[0][i]);
        id[i]=i;
    }
    for(int i=1;i<=m;i++){
        for(int j=1;j<=n;j++)scanf("%lf",&A[i][j]);
        scanf("%lf",&A[i][0]);
    }
    if(!initalize())printf("Infeasible");
    else if(!simplex())printf("Unbounded");
    else{
        printf("%.15lf\n",-A[0][0]);
        if(t){
            for(int i=1;i<=m;i++)x[id[i+n]]=A[i][0];
            for(int i=1;i<=n;i++)printf("%.15lf ",x[i]);
        }
    }
    return 0;
}

//初始化
//对于初始解可行的问题，可以把初始化省略掉
bool initalize(){
    for(;;){
        double t=0.0;
        int l=0,e=0;
        for(int i=1;i<=m;i++)if(A[i][0]+eps<t){
            t=A[i][0];
```

```
            l=i;
        }
        if(!l)return true;
        for(int i=1;i<=n;i++)if(A[l][i]<-eps&&(!e||id[i]<id[e]))e=i;
        if(!e)return false;
        pivot(l,e);
    }
}

//求解
bool simplex(){
    for(;;){
        int l=0,e=0;
        for(int i=1;i<=n;i++)if(A[0][i]>eps&&(!e||id[i]<id[e]))e=i;
        if(!e)return true;
        double t=1e50;
        for(int i=1;i<=m;i++)if(A[i][e]>eps&&A[i][0]/A[i][e]<t){
            l=i;
            t=A[i][0]/A[i][e];
        }
        if(!l)return false;
        pivot(l,e);
    }
}

//转轴操作，本质是
void pivot(int l,int e){
    swap(id[e],id[n+l]);
    double t=A[l][e];
    A[l][e]=1.0;
    for(int i=0;i<=n;i++)A[l][i]/=t;
    for(int i=0;i<=m;i++)if(i!=l){
        t=A[i][e];
        A[i][e]=0.0;
        for(int j=0;j<=n;j++)A[i][j]-=t*A[l][j];
    }
}
```

线性基

线性基是向量空间的一组基，通常可以解决有关异或的一些题目。

通俗一点的讲法就是由一个集合构造出来的另一个集合，它有以下几个性质：

- 线性基的元素能相互异或得到原集合的元素的所有相互异或得到的值。
- 线性基是满足性质 1 的最小的集合。
- 线性基没有异或和为 0 的子集。
- 线性基中每个元素的异或方案唯一，也就是说，线性基中不同的异或组合异或出的数都是不一样的。
- 线性基中每个元素的二进制最高位互不相同。

构造线性基的方法如下：

对原集合的每个数 p 转为二进制，从高位向低位扫，对于第 位是 1 的，如果 不存在，那么令 并结束扫描，如果存在，令 。

代码：

```
inline void insert(long long x) {
  for (int i = 55; i + 1; i--) {
    if (!(x >> i))  // x的第i位是0
      continue;
    if (!p[i]) {
      p[i] = x;
      break;
    }
    x ^= p[i];
  }
}
```

查询原集合内任意几个元素 xor 的最大值，就可以用线性基解决。

将线性基从高位向低位扫，若 xor 上当前扫到的 答案变大，就把答案异或上 。

为什么能行呢？因为从高往低位扫，若当前扫到第 位，意味着可以保证答案的第 位为 1，且后面没有机会改变第 位。

查询原集合内任意几个元素 xor 的最小值，就是线性基集合所有元素中最小的那个。

查询某个数是否能被异或出来，类似于插入，如果最后插入的数 被异或成了 0，则能被异或出来。

牛顿迭代

$x_{i+1}=x_i-\frac{f(x_i)}{f'(x_i)}$

求解平方根：$x_{i+1}=x_i-\frac{x_i^2-n}{2x_i}=\frac{x_i+\frac{n}{x_i}}{2}$

数论

$O(n)$预处理逆元

```
//Mutiply Inversation 预处理乘法逆元 O(n)
//By AntiLeaf
//要求p为质数

inv[0]=inv[1]=1;
for(int i=2;i<=n;i++)
    inv[i]=(long long)(p-(p/i))*inv[p%i]%p;//p为模数
//$i^{-1}\equiv-\left\lfloor\frac p i\right\rfloor\times(p\bmod i)^{-1}\pmod p$
//i^-1 = -(p/i) * (p%i)^-1
```

杜教筛

```
//Yuhao Du's Sieve 杜教筛 O(n^{2/3})
//By AntiLeaf
//通过题目：51Nod1239 欧拉函数之和

//用于求可以用狄利克雷卷积构造出好求和的东西的函数的前缀和（有点绕）
```

```
//有些题只要求n<=10^9，这时就没必要开long long了，但记得乘法时强转

//常量/全局变量/数组定义
const int maxn=5000005,table_size=5000000,p=1000000007,inv_2=(p+1)/2;
bool notp[maxn];
int prime[maxn/20],phi[maxn],tbl[100005];
//tbl用来顶替哈希表，其实开到n^{1/3}就够了，不过保险起见开成\sqrt n比较好
long long N;

//主函数前面加上这么一句
memset(tbl,-1,sizeof(tbl));

//线性筛预处理部分略去

//杜教筛主过程 总计O(n^{2/3})
//递归调用自身
//递推式还需具体情况具体分析，这里以求欧拉函数前缀和(mod 10^9+7)为例
int S(long long n){
    if(n<=table_size)return phi[n];
    else if(~tbl[N/n])return tbl[N/n];
    //原理：n除以所有可能的数的结果一定互不相同
    int ans=0;
    for(long long i=2,last;i<=n;i=last+1){
        last=n/(n/i);
        ans=(ans+(last-i+1)%p*S(n/i))%p;//如果n是int范围的话记得强转
    }
    ans=(n%p*((n+1)%p)%p*inv_2-ans+p)%p;//同上
    return tbl[N/n]=ans;
}
```

线性筛

```
//Extended Euler's Sieving 扩展线性筛 O(n)
//By AntiLeaf
//通过题目：51Nod1220 约数之和（预处理部分）

//此代码以计算约数之和函数\sigma_1（对10^9+7取模）为例
//适用于任何f(p^k)便于计算的积性函数
const int p=1000000007;

int prime[maxn/10],sigma_one[maxn],f[maxn],g[maxn];
//f：除掉最小质因子后剩下的部分
//g：最小质因子的幂次，在f(p^k)比较复杂时很有用，但f(p^k)可以递推时就可以省略了
//这里没有记录最小质因子，但根据线性筛的性质，每个合数只会被它最小的质因子筛掉
bool notp[maxn];//顾名思义

void get_table(int n){
    sigma_one[1]=1;//积性函数必有f(1)=1
    for(int i=2;i<=n;i++){
        if(!notp[i]){//质数情况
            prime[++prime[0]]=i;
            sigma_one[i]=i+1;
            f[i]=g[i]=1;
        }
```

```
        for(int j=1;j<=prime[0]&&i*prime[j]<=n;j++){
            notp[i*prime[j]]=true;
            if(i%prime[j]){//加入一个新的质因子，这种情况很简单
                sigma_one[i*prime[j]]=(long long)sigma_one[i]*(prime[j]+1)%p;
                f[i*prime[j]]=i;
                g[i*prime[j]]=1;
            }
            else{//再加入一次最小质因子，需要再进行分类讨论
                f[i*prime[j]]=f[i];
                g[i*prime[j]]=g[i]+1;
                //对于f(p^k)可以直接递推的函数，这里的判断可以改成
                //i/prime[j]%prime[j]!=0，这样可以省下f[]的空间，
                //但常数很可能会稍大一些
                if(f[i]==1)//质数的幂次，这里\sigma_1可以递推
                    sigma_one[i*prime[j]]=(sigma_one[i]+i*prime[j])%p;
                    //对于更一般的情况，可以借助g[]计算f(p^k)
                else sigma_one[i*prime[j]]=//否则直接利用积性，两半乘起来
                    (long long)sigma_one[i*prime[j]/f[i]]*sigma_one[f[i]]%p;
                break;
            }
        }
    }
}
```

Miller-Rabin

```
//Miller-Rabin Primality Test  Miller-Rabin素性检测算法
//By AntiLeaf
//通过题目：Bzoj4802 欧拉函数（作为Pollard's Rho的子算法）

//复杂度可以认为是常数

//封装好的函数体
//需要调用check
bool Miller_Rabin(long long n){
    if(n==1)return false;
    if(n==2)return true;
    if(n%2==0)return false;
    for(int i:{2,3,5,7,11,13,17,19,23,29,31,37}){
        if(i>n)break;
        if(!check(n,i))return false;
    }
    return true;
}

//用一个数检测
//需要调用long long快速幂和O(1)快速乘
bool check(long long n,long long b){//b是base
    long long a=n-1;
    int k=0;
    while(a%2==0){
        a>>=1;
```

```
        k++;
    }
    long long t=qpow(b,a,n);//这里的快速幂函数需要写O(1)快速乘
    if(t==1||t==n-1)return true;
    while(k--){
        t=mul(t,t,n);//mul是O(1)快速乘函数
        if(t==n-1)return true;
    }
    return false;
}
```

Pollard's Rho

```
//Pollard's Rho Algorithm  Pollard's Rho质因数分解 O(n^{1/4})
//By AntiLeaf
//通过题目：Bzoj4802 欧拉函数

//注意，虽然Pollard's Rho的理论复杂度是O(n^{1/4})的，
//但实际跑起来比较慢，一般用于做long long范围内的质因数分解


//封装好的函数体
//需要调用solve
void factorize(long long n,vector<long long>&v){//v用于存分解出来的质因子，重复的会放多个
    for(int i:{2,3,5,7,11,13,17,19})
        while(n%i==0){
            v.push_back(i);
            n/=i;
        }
    solve(n,v);
    sort(v.begin(),v.end());//从小到大排序后返回
}

//递归过程
//需要调用Pollard's Rho主过程，同时递归调用自身
void solve(long long n,vector<long long>&v){
    if(n==1)return;
    long long p;
    do p=Pollards_Rho(n);while(!p);//p是任意一个非平凡因子
    if(p==n){
        v.push_back(p);//说明n本身就是质数
        return;
    }
    solve(p,v);//递归分解两半
    solve(n/p,v);
}

//Pollard's Rho主过程
//需要使用Miller-Rabin作为子算法
//同时需要调用O(1)快速乘和gcd函数
long long Pollards_Rho(long long n){
    assert(n>1);
    if(Miller_Rabin(n))return n;
```

```
    long long c=rand()%(n-2)+1,i=1,k=2,x=rand()%(n-3)+2,u=2;//注意这里rand函数需要重定义一下
    for(;;){
        i++;
        x=(mul(x,x,n)+c)%n;//mul是O(1)快速乘函数
        long long g=gcd((u-x+n)%n,n);
        if(g>1&&g<n)return g;
        if(u==x)return 0;//失败，需要重新调用
        if(i==k){
            u=x;
            k<<=1;
        }
    }
}
```

数据结构

线段树

非递归线段树

（让fstqwq手撕，下一个）

主席树

（参见GREALD07加强版）

平衡树

Treap

```
//Treap Minimum Heap Version 小根堆版本
//By AntiLeaf
//通过题目：普通平衡树

//注意：相同键值可以共存

struct node{//结点类定义
    int key,size,p;//分别为键值、子树大小、优先度
    node *ch[2];//0表示左儿子，1表示右儿子
    node(int key=0):key(key),size(1),p(rand()){}
    void refresh(){size=ch[0]->size+ch[1]->size+1;}//更新子树大小（和附加信息）
}null[maxn],*root=null,*ptr=null;//数组名叫做null是为了方便开哨兵节点
//如果需要删除而空间不能直接开下所有结点，则需要再写一个垃圾回收
//注意：数组里的元素一定不能delete，否则会导致RE

//重要！！！
//在主函数最开始一定要加上以下预处理：
null->ch[0]=null->ch[1]=null;
null->size=0;

//伪构造函数 O(1)
```

```
//为了方便，在结点类外面再定义一个伪构造函数
node *newnode(int x){//键值为x
    *++ptr=node(x);
    ptr->ch[0]=ptr->ch[1]=null;
    return ptr;
}

//插入键值 期望O(\log n)
//需要调用旋转
void insert(int x,node *&rt){//rt为当前结点，建议调用时传入root，下同
    if(rt==null){
        rt=newnode(x);
        return;
    }
    int d=x>rt->key;
    insert(x,rt->ch[d]);
    rt->refresh();
    if(rt->ch[d]->p<rt->p)rot(rt,d^1);
}

//删除一个键值 期望O(\log n)
//要求键值必须存在至少一个，否则会导致RE
//需要调用旋转
void erase(int x,node *&rt){
    if(x==rt->key){
        if(rt->ch[0]!=null&&rt->ch[1]!=null){
            int d=rt->ch[0]->p<rt->ch[1]->p;
            rot(rt,d);
            erase(x,rt->ch[d]);
        }
        else rt=rt->ch[rt->ch[0]==null];
    }
    else erase(x,rt->ch[x>rt->key]);
    if(rt!=null)rt->refresh();
}

//求元素的排名（严格小于键值的个数+1） 期望O(\log n)
//非递归
int rank(int x,node *rt){
    int ans=1,d;
    while(rt!=null){
        if((d=x>rt->key))ans+=rt->ch[0]->size+1;
        rt=rt->ch[d];
    }
    return ans;
}

//返回排名第k（从1开始）的键值对应的指针 期望O(\log n)
//非递归
node *kth(int x,node *rt){
    int d;
    while(rt!=null){
        if(x==rt->ch[0]->size+1)return rt;
        if((d=x>rt->ch[0]->size))x-=rt->ch[0]->size+1;
        rt=rt->ch[d];
```

```
    }
    return rt;
}

//返回前驱（最大的比给定键值小的键值）对应的指针 期望O(\log n)
//非递归
node *pred(int x,node *rt){
    node *y=null;
    int d;
    while(rt!=null){
        if((d=x>rt->key))y=rt;
        rt=rt->ch[d];
    }
    return y;
}

//返回后继（最小的比给定键值大的键值）对应的指针 期望O(\log n)
//非递归
node *succ(int x,node *rt){
    node *y=null;
    int d;
    while(rt!=null){
        if((d=x<rt->key))y=rt;
        rt=rt->ch[d^1];
    }
    return y;
}

//旋转（Treap版本） O(1)
//平衡树基础操作
//要求对应儿子必须存在，否则会导致后续各种莫名其妙的问题
void rot(node *&x,int d){//x为被转下去的结点，会被修改以维护树结构
    node *y=x->ch[d^1];
    x->ch[d^1]=y->ch[d];
    y->ch[d]=x;
    x->refresh();
    (x=y)->refresh();
}
```

Splay

（参见LCT板子）

树的性质

重心

- n为奇数时只有一个重心，n为偶数时可能有两个
- 重心的任意一个子树大小不超过$\frac{n}{2}$
- **重心是到所有点距离之和最小的点**

直径

- 任取一个点，距离它最远的点必定是直径的两个端点之一

- 对带权直径也适用，也是求直径的一种简单做法
- 当然带负权就不行了，要用链剖+线段树维护最大子段和

- **直径不一定经过重心**，甚至可以不存在经过重心的直径

树分治

动态树分治

```
//Dynamic Divide and Couquer on Tree 动态树分治 O(n\log n)-O(\log n)
//By AntiLeaf
//通过题目：COGS2278 树黑白

//为了减小常数，这里采用bfs写法（实测预处理比dfs快将近一半
//以下以维护一个点到每个黑点的距离之和为例

//全局数组定义
vector<int>G[maxn],W[maxn];
int size[maxn],son[maxn],q[maxn];
int p[maxn],depth[maxn],id[maxn][20],d[maxn][20];//id是对应层所在子树的根
int a[maxn],ca[maxn],b[maxn][20],cb[maxn][20];//维护距离和用的
bool vis[maxn]={false},col[maxn]={false};

//建树 总计O(n\log n)
//需要调用找重心、预处理距离，同时递归调用自身
void build(int x,int k,int s,int pr){//结点，深度，连通块大小，点分树上的父亲
    x=getcenter(x,s);
    vis[x]=true;
    depth[x]=k;
    p[x]=pr;
    for(int i=0;i<(int)G[x].size();i++)
        if(!vis[G[x][i]]){
            d[G[x][i]][k]=W[x][i];
            p[G[x][i]]=x;
            getdis(G[x][i],k,G[x][i]);
        }
    for(int i=0;i<(int)G[x].size();i++)
        if(!vis[G[x][i]])build(G[x][i],k+1,size[G[x][i]],x);
}

//找重心 O(n)
int getcenter(int x,int s){
    int head=0,tail=0;
    q[tail++]=x;
    while(head!=tail){
        x=q[head++];
        size[x]=1;
        son[x]=0;
        for(int i=0;i<(int)G[x].size();i++)
            if(!vis[G[x][i]]&&G[x][i]!=p[x]){
                p[G[x][i]]=x;
                q[tail++]=G[x][i];
            }
    }
    for(int i=tail-1;i;i--){
```

```
        x=q[i];
        size[p[x]]+=size[x];
        if(size[x]>size[son[p[x]]])son[p[x]]=x;
    }
    x=q[0];
    while(son[x]&&(size[son[x]]<<1)>=s)x=son[x];
    return x;
}

//预处理距离 O(n)
//方便起见，这里直接用了笨一点的方法，O(n\log n)全存下来
void getdis(int x,int k,int rt){
    int head=0,tail=0;
    q[tail++]=x;
    while(head!=tail){
        x=q[head++];
        size[x]=1;
        id[x][k]=rt;
        for(int i=0;i<(int)G[x].size();i++)
            if(!vis[G[x][i]]&&G[x][i]!=p[x]){
                p[G[x][i]]=x;
                d[G[x][i]][k]=d[x][k]+w[x][i];
                q[tail++]=G[x][i];
            }
    }
    for(int i=tail-1;i;i--)
        size[p[q[i]]]+=size[q[i]];
}

//修改 O(\log n)
void modify(int x){
    if(col[x])ca[x]--;
    else ca[x]++;//记得先特判自己作为重心的那层
    for(int u=p[x],k=depth[x]-1;u;u=p[u],k--){
        if(col[x]){
            a[u]-=d[x][k];
            ca[u]--;
            b[id[x][k]][k]-=d[x][k];
            cb[id[x][k]][k]--;
        }
        else{
            a[u]+=d[x][k];
            ca[u]++;
            b[id[x][k]][k]+=d[x][k];
            cb[id[x][k]][k]++;
        }
    }
    col[x]^=true;
}

//询问 O(\log n)
int query(int x){
    int ans=a[x];//特判自己是重心的那层
    for(int u=p[x],k=depth[x]-1;u;u=p[u],k--)
        ans+=a[u]-b[id[x][k]][k]+d[x][k]*(ca[u]-cb[id[x][k]][k]);
```

```
    return ans;
}
```

紫荆花之恋

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=100005;
const double alpha=0.7; // 平衡因子，当某个子树占比超过alpha时暴力重构
struct node{
    static int randint(){
        static int a=1213,b=97818217,p=998244353,x=751815431;
        x=a*x+b;x%=p;
        return x<0?(x+=p):x;
    }
    int data,size,p;
    node *ch[2];
    node(int d):data(d),size(1),p(randint()){}
    inline void refresh(){size=ch[0]->size+ch[1]->size+1;}
}*null=new node(0),*root[maxn],*root1[maxn][50];
void addnode(int,int);
void rebuild(int,int,int,int);
void dfs_getcenter(int,int,int&);
void dfs_getdis(int,int,int,int);
void dfs_destroy(int,int);
void insert(int,node*&);
int order(int,node*);
void destroy(node*&);
void rot(node*&,int);
vector<int>G[maxn],w[maxn];
int size[maxn]={0},siz[maxn][50]={0},son[maxn];
bool vis[maxn];
int depth[maxn],p[maxn],d[maxn][50],id[maxn][50];
int n,m,w[maxn],tmp;
long long ans=0;
int main(){
    freopen("flowera.in","r",stdin);
    freopen("flowera.out","w",stdout);
    null->size=0;
    null->ch[0]=null->ch[1]=null;
    scanf("%*d%d",&n);
    fill(vis,vis+n+1,true);
    fill(root,root+n+1,null);
    for(int i=0;i<=n;i++)fill(root1[i],root1[i]+50,null);
    scanf("%*d%*d%d",&w[1]);
    insert(-w[1],root[1]);
    size[1]=1;
    printf("0\n");
    for(int i=2;i<=n;i++){
        scanf("%d%d%d",&p[i],&tmp,&w[i]);
        p[i]^=(ans%(int)1e9);
        G[i].push_back(p[i]);
        w[i].push_back(tmp);
        G[p[i]].push_back(i);
```

```
        w[p[i]].push_back(tmp);
        addnode(i,tmp);
        printf("%lld\n",ans);
    }
    return 0;
}
void addnode(int x,int z){//wj-dj>=di-wi
    depth[x]=depth[p[x]]+1;
    size[x]=1;
    insert(-w[x],root[x]);
    int rt=0;
    for(int u=p[x],k=depth[p[x]];u;u=p[u],k--){
        if(u==p[x]){
            id[x][k]=x;
            d[x][k]=z;
        }
        else{
            id[x][k]=id[p[x]][k];
            d[x][k]=d[p[x]][k]+z;
        }
        ans+=order(w[x]-d[x][k],root[u])-order(w[x]-d[x][k],root1[id[x]
[k]][k]);
        insert(d[x][k]-w[x],root[u]);
        insert(d[x][k]-w[x],root1[id[x][k]][k]);
        size[u]++;
        siz[id[x][k]][k]++;
        if(siz[id[x][k]][k]>size[u]*alpha+5)rt=u;
    }
    id[x][depth[x]]=0;
    d[x][depth[x]]=0;
    if(rt){
        dfs_destroy(rt,depth[rt]);
        rebuild(rt,depth[rt],size[rt],p[rt]);
    }
}
void rebuild(int x,int k,int s,int pr){
    int u=0;
    dfs_getcenter(x,s,u);
    vis[x=u]=true;
    p[x]=pr;
    depth[x]=k;
    size[x]=s;
    d[x][k]=id[x][k]=0;
    destroy(root[x]);
    insert(-w[x],root[x]);
    if(s<=1)return;
    for(int i=0;i<(int)G[x].size();i++)if(!vis[G[x][i]]){
        p[G[x][i]]=0;
        d[G[x][i]][k]=w[x][i];
        siz[G[x][i]][k]=p[G[x][i]]=0;
        destroy(root1[G[x][i]][k]);
        dfs_getdis(G[x][i],x,G[x][i],k);
    }
    for(int i=0;i<(int)G[x].size();i++)if(!vis[G[x][i]])rebuild(G[x]
[i],k+1,size[G[x][i]],x);
```

```
}
void dfs_getcenter(int x,int s,int &u){
    size[x]=1;
    son[x]=0;
    for(int i=0;i<(int)G[x].size();i++)if(!vis[G[x][i]]&&G[x][i]!=p[x]){
        p[G[x][i]]=x;
        dfs_getcenter(G[x][i],s,u);
        size[x]+=size[G[x][i]];
        if(size[G[x][i]]>size[son[x]])son[x]=G[x][i];
    }
    if(!u||max(s-size[x],size[son[x]])<max(s-size[u],size[son[u]]))u=x;
}
void dfs_getdis(int x,int u,int rt,int k){
    insert(d[x][k]-w[x],root[u]);
    insert(d[x][k]-w[x],root1[rt][k]);
    id[x][k]=rt;
    siz[rt][k]++;
    size[x]=1;
    for(int i=0;i<(int)G[x].size();i++)if(!vis[G[x][i]]&&G[x][i]!=p[x]){
        p[G[x][i]]=x;
        d[G[x][i]][k]=d[x][k]+W[x][i];
        dfs_getdis(G[x][i],u,rt,k);
        size[x]+=size[G[x][i]];
    }
}
void dfs_destroy(int x,int k){
    vis[x]=false;
    for(int i=0;i<(int)G[x].size();i++)if(depth[G[x][i]]>=k&&G[x]
[i]!=p[x]){
        p[G[x][i]]=x;
        dfs_destroy(G[x][i],k);
    }
}
void insert(int x,node *&rt){
    if(rt==null){
        rt=new node(x);
        rt->ch[0]=rt->ch[1]=null;
        return;
    }
    int d=x>=rt->data;
    insert(x,rt->ch[d]);
    rt->refresh();
    if(rt->ch[d]->p<rt->p)rot(rt,d^1);
}
int order(int x,node *rt){
    int ans=0,d;
    x++;
    while(rt!=null){
        if((d=x>rt->data))ans+=rt->ch[0]->size+1;
        rt=rt->ch[d];
    }
    return ans;
}
void destroy(node *&x){
    if(x==null)return;
```

```
    destroy(x->ch[0]);
    destroy(x->ch[1]);
    delete x;
    x=null;
}
void rot(node *&x,int d){
    node *y=x->ch[d^1];
    x->ch[d^1]=y->ch[d];
    y->ch[d]=x;
    x->refresh();
    (x=y)->refresh();
}
```

LCT

不换根

```
//Link-Cut Trees without Changing Root LCT不换根版本 O((n+m)\log n)
//By AntiLeaf
//通过题目：弹飞绵羊

//常数较大，请根据数据范围谨慎使用

#define isroot(x) ((x)!=(x)->p->ch[0]&&(x)!=(x)->p->ch[1])//判断是不是Splay的根
#define dir(x) ((x)==(x)->p->ch[1])//判断它是它父亲的左/右儿子

struct node{//结点类定义
    int size;//Splay的子树大小
    node *ch[2],*p;
    node():size(1){}
    void refresh(){size=ch[0]->size+ch[1]->size+1;}//附加信息维护
}null[maxn];

//在主函数开头加上这句初始化
null->size=0;

//初始化结点
void initalize(node *x){x->ch[0]=x->ch[1]=x->p=null;}//

//Access 均摊O(\log n)
//LCT核心操作，把结点到根的路径打通，顺便把与重儿子的连边变成轻边
//需要调用splay
node *access(node *x){
    node *y=null;
    while(x!=null){
        splay(x);
        x->ch[1]=y;
        (y=x)->refresh();
        x=x->p;
    }
    return y;
}

```

```
//Link 均摊O(\log n)
//把x的父亲设为y
//要求x必须为所在树的根节点，否则会导致后续各种莫名其妙的问题
//需要调用splay
void link(node *x,node *y){
    splay(x);
    x->p=y;
}

//Cut 均摊O(\log n)
//把x与其父亲的连边断掉
//x可以是所在树的根节点，这时此操作没有任何实质效果
//需要调用access和splay
void cut(node *x){
    access(x);
    splay(x);
    x->ch[0]->p=null;
    x->ch[0]=null;
    x->refresh();
}

//Splay 均摊O(\log n)
//需要调用旋转
void splay(node *x){
    while(!isroot(x)){
        if(isroot(x->p)){
            rot(x->p,dir(x)^1);
            break;
        }
        if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
        else rot(x->p,dir(x)^1);
        rot(x->p,dir(x)^1);
    }
}

//旋转（LCT版本） O(1)
//平衡树基本操作
//要求对应儿子必须存在，否则会导致后续各种莫名其妙的问题
void rot(node *x,int d){
    node *y=x->ch[d^1];
    y->p=x->p;
    if(!isroot(x))x->p->ch[dir(x)]=y;
    if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
    (y->ch[d]=x)->p=y;
    x->refresh();
    y->refresh();
}
```

换根 / 维护生成树（GREALD07 加强版）

```
/*************************************************************
    Problem: 3514
    User: hzoier
    Language: C++
```

```
    Result: Accepted
    Time:33584 ms
    Memory:93752 kb
*****************************************************************/
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<map>
#define isroot(x) ((x)->p==null||((x)->p->ch[0]!=(x)&&(x)->p->ch[1]!=(x)))
#define dir(x) ((x)==(x)->p->ch[1])
using namespace std;
const int maxn=200010;
struct node{
    int key,mn,pos;
    bool rev;
    node *ch[2],*p;
    node(int key=(~0u)>>1):key(key),mn(key),pos(-1),rev(false){}
    inline void pushdown(){
        if(!rev)return;
        ch[0]->rev^=true;
        ch[1]->rev^=true;
        swap(ch[0],ch[1]);
        if(pos!=-1)pos^=1;
        rev=false;
    }
    inline void refresh(){
        mn=key;
        pos=-1;
        if(ch[0]->mn<mn){
            mn=ch[0]->mn;
            pos=0;
        }
        if(ch[1]->mn<mn){
            mn=ch[1]->mn;
            pos=1;
        }
    }
}null[maxn<<1],*ptr=null;
node *newnode(int);
node *access(node*);
void makeroot(node*);
void link(node*,node*);
void cut(node*,node*);
node *getroot(node*);
node *getmin(node*,node*);
void splay(node*);
void rot(node*,int);
void build(int,int,int&,int);
void query(int,int,int,int);
int sm[maxn<<5]={0},lc[maxn<<5]={0},rc[maxn<<5]={0},root[maxn]={0},cnt=0;
map<node*,pair<node*,node*> >mp;
node *tmp;
int n,m,q,tp,x,y,k,l,r,t,ans=0;
int main(){
```

```
    null->ch[0]=null->ch[1]=null->p=null;
    scanf("%d%d%d%d",&n,&m,&q,&tp);
    for(int i=1;i<=n;i++)newnode((~0u)>>1);
    for(int i=1;i<=m;i++){
        scanf("%d%d",&x,&y);
        if(x==y){
            root[i]=root[i-1];
            continue;
        }
        if(getroot(null+x)!=getroot(null+y)){
            tmp=newnode(i);
            k=0;
        }
        else{
            tmp=getmin(null+x,null+y);
            cut(tmp,mp[tmp].first);
            cut(tmp,mp[tmp].second);
            k=tmp->key;
            tmp->key=i;
            tmp->refresh();
        }
        link(tmp,null+x);
        link(tmp,null+y);
        mp[tmp]=make_pair(null+x,null+y);
        build(0,m-1,root[i],root[i-1]);
    }
    while(q--){
        scanf("%d%d",&l,&r);
        if(tp){
            l^=ans;
            r^=ans;
        }
        ans=n;
        t=--l;
        query(0,m-1,root[r],root[l]);
        printf("%d\n",ans);
    }
    return 0;
}
node *newnode(int x){
    *++ptr=node(x);
    ptr->ch[0]=ptr->ch[1]=ptr->p=null;
    return ptr;
}
node *access(node *x){
    node *y=null;
    while(x!=null){
        splay(x);
        x->ch[1]=y;
        (y=x)->refresh();
        x=x->p;
    }
    return y;
}
void makeroot(node *x){
```

```
    access(x);
    splay(x);
    x->rev^=true;
}
void link(node *x,node *y){
    makeroot(x);
    x->p=y;
}
void cut(node *x,node *y){
    makeroot(x);
    access(y);
    splay(y);
    y->ch[0]->p=null;
    y->ch[0]=null;
    y->refresh();
}
node *getroot(node *x){
    x=access(x);
    while(x->pushdown(),x->ch[0]!=null)x=x->ch[0];
    splay(x);
    return x;
}
node *getmin(node *x,node *y){
    makeroot(x);
    x=access(y);
    while(x->pushdown(),x->pos!=-1)x=x->ch[x->pos];
    splay(x);
    return x;
}
void splay(node *x){
    x->pushdown();
    while(!isroot(x)){
        if(!isroot(x->p))x->p->p->pushdown();
        x->p->pushdown();
        x->pushdown();
        if(isroot(x->p)){
            rot(x->p,dir(x)^1);
            break;
        }
        if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
        else rot(x->p,dir(x)^1);
        rot(x->p,dir(x)^1);
    }
}
void rot(node *x,int d){
    node *y=x->ch[d^1];
    if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
    y->p=x->p;
    if(!isroot(x))x->p->ch[dir(x)]=y;
    (y->ch[d]=x)->p=y;
    x->refresh();
    y->refresh();
}
void build(int l,int r,int &rt,int pr){
    sm[rt=++cnt]=sm[pr]+1;
```

```
    if(l==r)return;
    lc[rt]=lc[pr];
    rc[rt]=rc[pr];
    int mid=(l+r)>>1;
    if(k<=mid)build(l,mid,lc[rt],lc[pr]);
    else build(mid+1,r,rc[rt],rc[pr]);
}
void query(int l,int r,int rt,int pr){
    if(!rt&&!pr)return;
    if(t>=r){
        ans-=sm[rt]-sm[pr];
        return;
    }
    int mid=(l+r)>>1;
    query(l,mid,lc[rt],lc[pr]);
    if(t>mid)query(mid+1,r,rc[rt],rc[pr]);
}
/*
如果是离线的话，我们可以LCT+莫队什么的乱搞是吧，但是在线就……
不过还是有一个很喵的做法——
我们用LCT维护一棵生成树，当加入一条边i的时候(i是其编号)，其连接的两个点可能已经联通，加入i之后会形成一个环，我们弹掉这个环上编号最小的边(也就是加入最早的边)，并记录其编号为ntr_i。特殊的，如果i没有弹掉任何边，我们记ntr_i=0。
对于一个询问[L,R](表示我们只保留e|e∈[L,R])，答案就是$n?\sum_{i=L}^R(ntr_i<L)$。这个就是主席树了。
——YouSiki

这做法简直是妙啊……
*/
```

维护子树信息

```
//Link-Cut Trees with subtree values LCT维护子树信息 O((n+m)\log n)
//By AntiLeaf
//通过题目：LOJ#558 我们的CPU遭到攻击（维护黑点到根距离和）

//这个东西虽然只需要抄板子但还是极其难写，常数极其巨大，没必要的时候就不要用
//如果维护子树最小值就需要套一个可删除的堆来维护，复杂度会变成O(n\log^2 n)
//注意由于这道题与边权有关，需要边权拆点变点权

//宏定义
#define isroot(x) ((x)->p==null||((x)!=(x)->p->ch[0]&&(x)!=(x)->p->ch[1]))
#define dir(x) ((x)==(x)->p->ch[1])

//节点类定义
struct node{//以维护子树中黑点到根距离和为例
    int w,chain_cnt,tree_cnt;
    long long sum,suml,sumr,tree_sum;//由于换根需要子树反转，需要维护两个方向的信息
    bool rev,col;
    node *ch[2],*p;
```

```
19
node():w(0),chain_cnt(0),tree_cnt(0),sum(0),suml(0),sumr(0),tree_sum(0),r
ev(false),col(false){}
20      inline void pushdown(){
21          if(!rev)return;
22          ch[0]->rev^=true;
23          ch[1]->rev^=true;
24          swap(ch[0],ch[1]);
25          swap(suml,sumr);
26          rev=false;
27      }
28      inline void refresh(){//不多解释了……这毒瘤题恶心的要死（我骂我自己.png
29          sum=ch[0]->sum+ch[1]->sum+w;
30          suml=(ch[0]->rev?ch[0]->sumr:ch[0]->suml)+(ch[1]->rev?ch[1]-
>sumr:ch[1]->suml)
31              +(tree_cnt+ch[1]->chain_cnt)*(ch[0]->sum+w)+tree_sum;
32          sumr=(ch[0]->rev?ch[0]->suml:ch[0]->sumr)+(ch[1]->rev?ch[1]-
>suml:ch[1]->sumr)
33              +(tree_cnt+ch[0]->chain_cnt)*(ch[1]->sum+w)+tree_sum;
34          chain_cnt=ch[0]->chain_cnt+ch[1]->chain_cnt+tree_cnt;
35      }
36  }null[maxn<<1];//如果没有边权变点权就不用乘2了
37
38  //封装构造函数
39  node *newnode(int w){
40      node *x=nodes.front();
41      nodes.pop();
42      initalize(x);
43      x->w=w;
44      x->refresh();
45      return x;
46  }
47
48  //封装初始化函数
49  //记得在进行操作之前对所有结点调用一遍
50  inline void initalize(node *x){
51      *x=node();
52      x->ch[0]=x->ch[1]=x->p=null;
53  }
54
55  //Access函数
56  //注意一下在Access的同时更新子树信息的方法
57  node *access(node *x){
58      node *y=null;
59      while(x!=null){
60          splay(x);
61          x->tree_cnt+=x->ch[1]->chain_cnt-y->chain_cnt;
62          x->tree_sum+=(x->ch[1]->rev?x->ch[1]->sumr:x->ch[1]->suml)-y-
>suml;
63          x->ch[1]=y;
64          (y=x)->refresh();
65          x=x->p;
66      }
67      return y;
68  }
```

```
69
70  //找到一个点所在连通块的根
71  //对比原版没有变化
72  node *getroot(node *x){
73      x=access(x);
74      while(x->pushdown(),x->ch[0]!=null)x=x->ch[0];
75      splay(x);
76      return x;
77  }
78
79  //换根，同样没有变化
80  void makeroot(node *x){
81      access(x);
82      splay(x);
83      x->rev^=true;
84      x->pushdown();
85  }
86
87  //连接两个点
88  //注意这里必须把两者都变成根，因为只能修改根结点
89  void link(node *x,node *y){
90      makeroot(x);
91      makeroot(y);
92      x->p=y;
93      y->tree_cnt+=x->chain_cnt;
94      y->tree_sum+=x->suml;
95      y->refresh();
96  }
97
98  //删除一条边
99  //对比原版没有变化
100 void cut(node *x,node *y){
101     makeroot(x);
102     access(y);
103     splay(y);
104     y->ch[0]->p=null;
105     y->ch[0]=null;
106     y->refresh();
107 }
108
109 //修改/询问一个点，这里以询问为例
110 //如果是修改就在换根之后搞一些操作
111 long long query(node *x){
112     makeroot(x);
113     return x->suml;
114 }
115
116 //Splay函数
117 //对比原版没有变化
118 void splay(node *x){
119     x->pushdown();
120     while(!isroot(x)){
121         if(!isroot(x->p))x->p->p->pushdown();
122         x->p->pushdown();
123         x->pushdown();
```

```
        if(isroot(x->p)){
            rot(x->p,dir(x)^1);
            break;
        }
        if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
        else rot(x->p,dir(x)^1);
        rot(x->p,dir(x)^1);
    }
}

//旋转函数
//对比原版没有变化
void rot(node *x,int d){
    node *y=x->ch[d^1];
    if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
    y->p=x->p;
    if(!isroot(x))x->p->ch[dir(x)]=y;
    (y->ch[d]=x)->p=y;
    x->refresh();
    y->refresh();
}
```

模板题：动态QTREE4

题意：加边删边修改边权，维护树上最远点对

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
#include<ext/pb_ds/priority_queue.hpp>

#define isroot(x) ((x)->p==null||((x)!=(x)->p->ch[0]&&(x)!=(x)->p->ch[1]))
#define dir(x) ((x)==(x)->p->ch[1])

using namespace std;
using namespace __gnu_pbds;

const int maxn=100010;
const long long INF=1000000000000000000ll;

struct binary_heap{
    __gnu_pbds::priority_queue<long long,less<long long>,binary_heap_tag>q1,q2;
    binary_heap(){}
    void push(long long x){if(x>(-INF)>>2)q1.push(x);}
    void erase(long long x){if(x>(-INF)>>2)q2.push(x);}
    long long top(){
        if(empty())return -INF;
        while(!q2.empty()&&q1.top()==q2.top()){
            q1.pop();
            q2.pop();
        }
        return q1.top();
```

```
    }
    long long top2(){
        if(size()<2)return -INF;
        long long a=top();
        erase(a);
        long long b=top();
        push(a);
        return a+b;
    }
    int size(){return q1.size()-q2.size();}
    bool empty(){return q1.size()==q2.size();}
}heap;//全局堆维护每条链的最大子段和
struct node{
    long long sum,maxsum,prefix,suffix;
    int key;
    binary_heap heap;//每个点的堆存的是它的子树中到它的最远距离，如果它是黑点的话还会包括自己
    node *ch[2],*p;
    bool rev;
    node(int k=0):sum(k),maxsum(-INF),prefix(-INF),suffix(-INF),key(k),rev(false){}
    inline void pushdown(){
        if(!rev)return;
        ch[0]->rev^=true;
        ch[1]->rev^=true;
        swap(ch[0],ch[1]);
        swap(prefix,suffix);
        rev=false;
    }
    inline void refresh(){
        pushdown();
        ch[0]->pushdown();
        ch[1]->pushdown();
        sum=ch[0]->sum+ch[1]->sum+key;
        prefix=max(ch[0]->prefix,ch[0]->sum+key+ch[1]->prefix);
        suffix=max(ch[1]->suffix,ch[1]->sum+key+ch[0]->suffix);
        maxsum=max(max(ch[0]->maxsum,ch[1]->maxsum),ch[0]->suffix+key+ch[1]->prefix);
        if(!heap.empty()){
            prefix=max(prefix,ch[0]->sum+key+heap.top());
            suffix=max(suffix,ch[1]->sum+key+heap.top());
            maxsum=max(maxsum,max(ch[0]->suffix,ch[1]->prefix)+key+heap.top());
            if(heap.size()>1){
                maxsum=max(maxsum,heap.top2()+key);
            }
        }
    }
}null[maxn<<1],*ptr=null;
void addedge(int,int,int);
void deledge(int,int);
void modify(int,int,int);
void modify_color(int);
node *newnode(int);
node *access(node*);
```

```
78  void makeroot(node*);
79  void link(node*,node*);
80  void cut(node*,node*);
81  void splay(node*);
82  void rot(node*,int);
83  queue<node*>freenodes;
84  tree<pair<int,int>,node*>mp;
85  bool col[maxn]={false};
86  char c;
87  int n,m,k,x,y,z;
88  int main(){
89      null->ch[0]=null->ch[1]=null->p=null;
90      scanf("%d%d%d",&n,&m,&k);
91      for(int i=1;i<=n;i++){
92          newnode(0);
93      }
94      heap.push(0);
95      while(k--){
96          scanf("%d",&x);
97          col[x]=true;
98          null[x].heap.push(0);
99      }
100     for(int i=1;i<n;i++){
101         scanf("%d%d%d",&x,&y,&z);
102         if(x>y)swap(x,y);
103         addedge(x,y,z);
104     }
105     while(m--){
106         scanf(" %c%d",&c,&x);
107         if(c=='A'){
108             scanf("%d",&y);
109             if(x>y)swap(x,y);
110             deledge(x,y);
111         }
112         else if(c=='B'){
113             scanf("%d%d",&y,&z);
114             if(x>y)swap(x,y);
115             addedge(x,y,z);
116         }
117         else if(c=='C'){
118             scanf("%d%d",&y,&z);
119             if(x>y)swap(x,y);
120             modify(x,y,z);
121         }
122         else modify_color(x);
123         printf("%lld\n",(heap.top()>0?heap.top():-1));
124     }
125     return 0;
126 }
127 void addedge(int x,int y,int z){
128     node *tmp;
129     if(freenodes.empty())tmp=newnode(z);
130     else{
131         tmp=freenodes.front();
132         freenodes.pop();
```

```
        *tmp=node(z);
    }
    tmp->ch[0]=tmp->ch[1]=tmp->p=null;heap.push(tmp->maxsum);
    link(tmp,null+x);
    link(tmp,null+y);
    mp[make_pair(x,y)]=tmp;
}
void deledge(int x,int y){
    node *tmp=mp[make_pair(x,y)];
    cut(tmp,null+x);
    cut(tmp,null+y);
    freenodes.push(tmp);
    heap.erase(tmp->maxsum);
    mp.erase(make_pair(x,y));
}
void modify(int x,int y,int z){
    node *tmp=mp[make_pair(x,y)];
    makeroot(tmp);
    tmp->pushdown();
    heap.erase(tmp->maxsum);
    tmp->key=z;
    tmp->refresh();
    heap.push(tmp->maxsum);
}
void modify_color(int x){
    makeroot(null+x);
    col[x]^=true;
    if(col[x])null[x].heap.push(0);
    else null[x].heap.erase(0);
    heap.erase(null[x].maxsum);
    null[x].refresh();
    heap.push(null[x].maxsum);
}
node *newnode(int k){
    *(++ptr)=node(k);
    ptr->ch[0]=ptr->ch[1]=ptr->p=null;
    return ptr;
}
node *access(node *x){
    splay(x);
    heap.erase(x->maxsum);
    x->refresh();
    if(x->ch[1]!=null){
        x->ch[1]->pushdown();
        x->heap.push(x->ch[1]->prefix);x->refresh();
        heap.push(x->ch[1]->maxsum);
    }
    x->ch[1]=null;
    x->refresh();
    node *y=x;
    x=x->p;
    while(x!=null){
        splay(x);
        heap.erase(x->maxsum);
        if(x->ch[1]!=null){
```

```
            x->ch[1]->pushdown();
            x->heap.push(x->ch[1]->prefix);
            heap.push(x->ch[1]->maxsum);
        }
        x->heap.erase(y->prefix);
        x->ch[1]=y;
        (y=x)->refresh();
        x=x->p;
    }
    heap.push(y->maxsum);
    return y;
}
void makeroot(node *x){
    access(x);
    splay(x);
    x->rev^=true;
}
void link(node *x,node *y){//新添一条虚边，维护y对应的堆
    makeroot(x);
    makeroot(y);
    x->pushdown();
    x->p=y;
    heap.erase(y->maxsum);
    y->heap.push(x->prefix);
    y->refresh();
    heap.push(y->maxsum);
}
void cut(node *x,node *y){//断开一条实边，一条链变成两条链，需要维护全局堆
    makeroot(x);
    access(y);
    splay(y);
    heap.erase(y->maxsum);
    heap.push(y->ch[0]->maxsum);
    y->ch[0]->p=null;
    y->ch[0]=null;
    y->refresh();
    heap.push(y->maxsum);
}
void splay(node *x){
    x->pushdown();
    while(!isroot(x)){
        if(!isroot(x->p))x->p->p->pushdown();
        x->p->pushdown();
        x->pushdown();
        if(isroot(x->p)){
            rot(x->p,dir(x)^1);
            break;
        }
        if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
        else rot(x->p,dir(x)^1);
        rot(x->p,dir(x)^1);
    }
}
void rot(node *x,int d){
    node *y=x->ch[d^1];
```

```
    if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
    y->p=x->p;
    if(!isroot(x))x->p->ch[dir(x)]=y;
    (y->ch[d]=x)->p=y;
    x->refresh();
    y->refresh();
}
```

长链剖分

```
//Long-chain Subdivision 长链剖分 O(n)
//By AntiLeaf
//通过题目：vijos lxhgww的奇思妙想（板子题）、Codeforces 1009F

//顾名思义，长链剖分是取最深的儿子作为重儿子
//长链剖分的两个应用：
//O(1)在线求一个点的第k祖先
//O(n)维护以深度为下标的子树信息

//--------------------------------分割线--------------------------------

//在线求一个点的第k祖先 O(n\log n)-O(1)
//其中O(n\log n)预处理是因为需要用到倍增
//理论基础：任意一个点x的k级祖先y所在长链长度一定>=k

//全局数组定义
vector<int>G[maxn],v[maxn];
int d[maxn],mxd[maxn],son[maxn],top[maxn],len[maxn];
int f[maxn][19],log_tbl[maxn];

//在主函数中两遍dfs之后加上如下预处理
log_tbl[0]=-1;
for(int i=1;i<=n;i++)log_tbl[i]=log_tbl[i>>1]+1;
for(int j=1;(1<<j)<n;j++)
    for(int i=1;i<=n;i++)
        f[i][j]=f[f[i][j-1]][j-1];

//第一遍dfs，用于计算深度和找出重儿子
//递归调用自身
void dfs1(int x){
    mxd[x]=d[x];
    for(int i=0;i<(int)G[x].size();i++)
        if(G[x][i]!=f[x][0]){
            f[G[x][i]][0]=x;
            d[G[x][i]]=d[x]+1;
            dfs1(G[x][i]);
            mxd[x]=max(mxd[x],mxd[G[x][i]]);
            if(mxd[G[x][i]]>mxd[son[x]])son[x]=G[x][i];
        }
}

//第二遍dfs，用于进行剖分和预处理梯子剖分（每条链向上延伸一倍）数组
//递归调用自身
void dfs2(int x){
```

```
    top[x]=(x==son[f[x][0]]?top[f[x][0]]:x);
    for(int i=0;i<(int)G[x].size();i++)
        if(G[x][i]!=f[x][0])dfs2(G[x][i]);
    if(top[x]==x){
        int u=x;
        while(top[son[u]]==x)u=son[u];
        len[x]=d[u]-d[x];
        for(int i=0;i<len[x];i++,u=f[u][0])v[x].push_back(u);
        u=x;
        for(int i=0;i<len[x]&&u;i++,u=f[u][0])v[x].push_back(u);
    }
}

//在线询问x的k级祖先 O(1)
//不存在时返回0
int query(int x,int k){
    if(!k)return x;
    if(k>d[x])return 0;
    x=f[x][log_tbl[k]];
    k^=1<<log_tbl[k];
    return v[top[x]][d[top[x]]+len[top[x]]-d[x]+k];
}

//------------------------------分割线-----------------------------

//O(n)维护以深度为下标的子树信息

vector<int>G[maxn],v[maxn];
int n,p[maxn],h[maxn],son[maxn],ans[maxn];

//原题题意：求每个点的子树中与它距离是几的点最多，相同的取最大深度
//由于vector只能在后面加入元素，为了写代码方便，这里反过来存
void dfs(int x){
    h[x]=1;
    for(int i=0;i<(int)G[x].size();i++)
        if(G[x][i]!=p[x]){
            p[G[x][i]]=x;
            dfs(G[x][i]);
            if(h[G[x][i]]>h[son[x]])son[x]=G[x][i];
        }
    if(!son[x]){
        v[x].push_back(1);
        ans[x]=0;
        return;
    }
    //printf("x=%d h=%d son=%d\n",x,h[x],son[x]);
    h[x]=h[son[x]]+1;
    swap(v[x],v[son[x]]);
    if(v[x][ans[son[x]]]==1)ans[x]=h[x]-1;
    else ans[x]=ans[son[x]];
    v[x].push_back(1);
    int mx=v[x][ans[x]];
    for(int i=0;i<(int)G[x].size();i++)
        if(G[x][i]!=p[x]&&G[x][i]!=son[x]){
            for(int j=1;j<=h[G[x][i]];j++){
```

```
                v[x][h[x]-j-1]+=v[G[x][i]][h[G[x][i]]-j];
                int t=v[x][h[x]-j-1];
                if(t>mx||(t==mx&&h[x]-j-1>ans[x])){
                    mx=t;
                    ans[x]=h[x]-j-1;
                }
            }
            v[G[x][i]].clear();
        }
}
```

可并堆（左偏树）

（参见k短路板子）

自带数据结构

STL

（不要忘了可以用`rbegin()`和`rend()`，同时正反迭代器是可以转换的）

vector

```
vector(int nSize):创建一个vector,元素个数为nSize
vector(int nSize,const t& t):创建一个vector，元素个数为nSize,且值均为t
vector(begin,end):复制[begin,end)区间内另一个数组的元素到vector中

void assign(int n,const T& x):设置向量中前n个元素的值为x
void assign(const_iterator first,const_iterator last):向量中[first,last)中元素设置成当前向量元素
```

list

```
assign() 给list赋值
back() 返回最后一个元素
begin() 返回指向第一个元素的迭代器
clear() 删除所有元素
empty() 如果list是空的则返回true
end() 返回末尾的迭代器
erase() 删除一个元素
front() 返回第一个元素
get_allocator() 返回list的配置器
insert() 插入一个元素到list中
max_size() 返回list能容纳的最大元素数量
merge() 合并两个list
pop_back() 删除最后一个元素
pop_front() 删除第一个元素
push_back() 在list的末尾添加一个元素
push_front() 在list的头部添加一个元素
rbegin() 返回指向第一个元素的逆向迭代器
remove() 从list删除元素
remove_if() 按指定条件删除元素
rend() 指向list末尾的逆向迭代器
```

```
resize() 改变list的大小
reverse() 把list的元素倒转
size() 返回list中的元素个数
sort() 给list排序
splice() 合并两个list
swap() 交换两个list
unique() 删除list中重复的元素
```

pb_ds

哈希表

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace __gnu_pbds;

cc_hash_table<string,int>mp1;//拉链法
gp_hash_table<string,int>mp2;//查探法(快一些)
```

堆

```
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;

__gnu_pbds::priority_queue<int>q;
__gnu_pbds::priority_queue<int,greater<int>,pairing_heap_tag> pq;
```

常用操作：

- `push()`：向堆中压入一个元素，返回该元素位置的迭代器。
- `pop()`：将堆顶元素弹出。
- `top()`：返回堆顶元素。
- `size()` 返回元素个数。
- `empty()` 返回是否非空。
- `modify(point_iterator, const key)`：把迭代器位置的 `key` 修改为传入的 `key`，并对底层储存结构进行排序。
- `erase(point_iterator)`：把迭代器位置的键值从堆中擦除。
- `join(__gnu_pbds :: priority_queue &other)`：把 `other` 合并到 `*this` 并把 `other` 清空。

平衡树

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> t;

// rb_tree_tag 红黑树（还有splay_tree_tag和ov_tree_tag，后者不知道是什么）
```

注意第五个参数要填`tree_order_statistics_node_update`才能使用排名操作。

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

（注意平衡树不支持多重值，如果需要多重值，可以再开一个unordered_map来记录值出现的次数。将x<<32后加上出现的次数后插入tree。注意此时应该为long long类型。）

Rope

```
#include <ext/rope>

using namespace __gnu_cxx;

push_back(x);//在末尾添加x
insert(pos,x);//在pos插入x，自然支持整个char数组的一次插入
erase(pos,x);//从pos开始删除x个
copy(pos,len,x);//从pos开始到pos+len为止的部分，赋值给x
replace(pos,x);//从pos开始换成x
substr(pos,x);//提取pos开始x个
at(x)/[x];//访问第x个元素
```

常见根号思路

通用

- 出现次数大于$\sqrt{n}$的数不会超过$\sqrt{n}$个
- 对于带修改问题，如果不方便分治或者二进制分组，可以考虑对操作分块
  - 每次查询时暴力最后的$\sqrt{n}$个修改并更正答案
- **根号分治**：如果分治时每个子问题需要$O(N)$（N是全局问题的大小）的时间，而规模较小的子问题可以$O(n^2)$解决，则可以使用根号分治
  - 规模大于$\sqrt{n}$的子问题用$O(N)$的方法解决，规模小于$\sqrt{n}$的子问题用$O(n^2)$暴力
  - 规模大于$\sqrt{n}$的子问题最多只有$\sqrt{n}$个
  - 规模不大于$\sqrt{n}$的子问题大小的平方和也必定不会超过$n\sqrt{n}$
- 如果输入规模之和不大于n（例如给定多个小字符串与大字符串进行询问），那么规模超过$\sqrt{n}$的问题最多只有$\sqrt{n}$个

序列

- 某些维护序列的问题可以用分块/块状链表维护
- 对于静态区间询问问题，如果可以快速将左/右端点移动一位，可以考虑莫队
  - 如果强制在线可以分块预处理，但是一般空间需要$n\sqrt{n}$
    - 例题：询问区间中有几种数出现次数恰好为k，强制在线
  - 如果带修改可以试着想一想带修莫队，但是复杂度高达$n^{\frac{5}{3}}$

- 线段树可以解决的问题也可以用分块来做到$O(1)$询问或是$O(1)$修改，具体要看哪种操作更多

树

- 与序列类似，树上也有树分块和树上莫队
  - 树上带修莫队很麻烦，常数也大，最好不要先考虑
  - 树分块不要想当然
- 树分治也可以套根号分治，道理是一样的

字符串

- 循环节长度大于$\sqrt{n}$的子串最多只有$O(n)$个，如果是极长子串则只有$O(\sqrt{n})$个

DP

决策单调性（在线）

```cpp
#include <bits/stdc++.h>

using namespace std;

const int maxn = 300005;

int a[maxn], q[maxn], p[maxn], g[maxn]; // 存左端点，右端点就是下一个左端点 - 1

long long f[maxn], s[maxn];

int n, m;

long long calc(int l, int r) {
    if (r < l)
        return 0;

    int mid = (l + r) / 2;
    if ((r - l + 1) % 2 == 0)
        return (s[r] - s[mid]) - (s[mid] - s[l - 1]);
    else
        return (s[r] - s[mid]) - (s[mid - 1] - s[l - 1]);
}

int solve(long long tmp) {
    memset(f, 63, sizeof(f));
    f[0] = 0;

    int head = 1, tail = 0;

    // printf("---------- solve(%lld) ----------\n", tmp);

    for (int i = 1; i <= n; i++) {
        f[i] = calc(1, i);
        g[i] = 1;

```

```
        while (head < tail && p[head + 1] <= i)
            head++;
        if (head <= tail) {
            if (f[q[head]] + calc(q[head] + 1, i) < f[i]) {
                f[i] = f[q[head]] + calc(q[head] + 1, i);
                g[i] = g[q[head]] + 1;
            }
            while (head < tail && p[head + 1] <= i + 1)
                head++;
            if (head <= tail)
                p[head] = i + 1;
        }
        f[i] += tmp;
        // printf("f[%d] = %lld g[%d] = %d\n", i, f[i], i, g[i]);

        /*
        if (head <= tail && f[q[tail]] + calc(q[tail] + 1, n) <= f[i] + calc(i + 1, n))
            continue;
        */

        int r = n;

        while(head <= tail) {
            if (f[q[tail]] + calc(q[tail] + 1, p[tail]) > f[i] + calc(i + 1, p[tail])) {
                r = p[tail] - 1;
                tail--;
            }
            else if (f[q[tail]] + calc(q[tail] + 1, r) <= f[i] + calc(i + 1, r)) {
                if (r < n) {
                    q[++tail] = i;
                    p[tail] = r + 1;
                }
                break;
            }
            else {
                int L = p[tail], R = r;
                while (L < R) {
                    int M = (L + R) / 2;

                    if (f[q[tail]] + calc(q[tail] + 1, M) <= f[i] + calc(i + 1, M))
                        L = M + 1;
                    else
                        R = M;
                }

                q[++tail] = i;
                p[tail] = L;

                break;
            }
        }
```

```
        if (head > tail) {
            q[++tail] = i;
            p[tail] = i + 1;
        }
    }

    return g[n];
}

int main() {
    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        s[i] = s[i - 1] + a[i];
    }

    long long L = 0, R = 1e16;

    while (L < R) {
        long long M = (L + R) / 2;
        if (solve(M) > m)
            L = M + 1;
        else
            R = M;
    }

    solve(L);

    printf("%lld\n", f[n] - m * L);

    return 0;
}
```

斜率优化

陈丹琦分治维护（NOI2007 货币兑换Cash）

```
#include<cstdio>
#include<cstring>
#include<cmath>
#include<algorithm>
using namespace std;
const int maxn=100010;
const long double eps=1e-7;
void mergesort(int,int,int);
int CDQ(int,int,int);
long double getk(int,int);
long double w(int,int);
long double A[maxn],B[maxn],R[maxn],f[maxn],x[maxn],y[maxn],k[maxn];
double tmp;
int n,t[20][maxn],a[maxn],s[maxn];
int main(){
    freopen("cash.in","r",stdin);
```

```
    freopen("cash.out","w",stdout);
    scanf("%d%lf",&n,&tmp);
    f[1]=tmp;
    for(int i=1;i<=n;i++){
        scanf("%lf",&tmp);
        A[i]=tmp;
        scanf("%lf",&tmp);
        B[i]=tmp;
        scanf("%lf",&tmp);
        R[i]=tmp;
        k[i]=B[i]/A[i];
    }
    mergesort(1,n,0);
    CDQ(1,n,1);
    printf("%.3lf",(double)f[n]);
    return 0;
}
void mergesort(int l,int r,int d){
    if(l>=r){
        t[d][l]=l;
        return;
    }
    int mid=(l+r)>>1;
    mergesort(l,mid,d+1);
    mergesort(mid+1,r,d+1);
    int i=l,j=mid+1,p=l;
    while(i<=mid&&j<=r){
        if(k[t[d+1][i]]<=k[t[d+1][j]])t[d][p++]=t[d+1][i++];
        else t[d][p++]=t[d+1][j++];
    }
    while(i<=mid)t[d][p++]=t[d+1][i++];
    while(j<=r)t[d][p++]=t[d+1][j++];
}
int CDQ(int l,int r,int d){
    if(l>=r){
        a[l]=l;
        x[l]=f[l]/(A[l]*R[l]+B[l]);
        y[l]=-R[l]*x[l];
        f[l+1]=max(f[l+1],f[l]);
        return 1;
    }
    int mid=(l+r)>>1,cntl=CDQ(l,mid,d+1),i=l,j=mid+1;
    while(i<l+cntl-1&&j<=r){
        if(getk(a[i],a[i+1])-eps<k[t[d][j]])i++;
        else{
            f[t[d][j]]=max(f[t[d][j]],w(a[i],t[d][j]));
            j++;
        }
    }
    while(j<=r){
        f[t[d][j]]=max(f[t[d][j]],w(a[i],t[d][j]));
        j++;
    }
    int cntr=CDQ(mid+1,r,d+1),cnt=l-1;
    i=l;j=mid+1;
```

```
    while(i<l+cntl&&j<=mid+cntr){
        if(fabs(x[a[i]]-x[a[j]])<=eps){
            x[a[i]]=min(x[a[i]],x[a[j]]);
            j++;
        }
        else if(x[a[i]]<x[a[j]]){
            while(cnt>l&&getk(s[cnt-1],s[cnt])+eps>getk(s[cnt],a[i]))cnt--;
            s[++cnt]=a[i++];
        }
        else{
            while(cnt>l&&getk(s[cnt-1],s[cnt])+eps>getk(s[cnt],a[j]))cnt--;
            s[++cnt]=a[j++];
        }
    }
    while(i<l+cntl){
        while(cnt>l&&getk(s[cnt-1],s[cnt])+eps>getk(s[cnt],a[i]))cnt--;
        s[++cnt]=a[i++];
    }
    while(j<=mid+cntr){
        while(cnt>l&&getk(s[cnt-1],s[cnt])+eps>getk(s[cnt],a[j]))cnt--;
        s[++cnt]=a[j++];
    }
    copy(s+l,s+r+1,a+l);
    return cnt-l+1;
}
inline long double getk(int i,int j){return (y[i]-y[j])/(x[i]-x[j]);}
inline long double w(int j,int i){return f[j]*(A[i]*R[j]+B[i])/(A[j]*R[j]+B[j]);}
```

杂の算法

$O(1)$快速乘

```
// Quick Multiplication O(1)快速乘
// By AntiLeaf

// long double 快速乘
// 在两数直接相乘会爆long long时才有必要使用
// 常数比直接long long乘法+取模大很多，非必要时不建议使用
long long mul(long long a,long long b,long long p){
    a%=p;b%=p;
    return ((a*b-p*(long long)((long double)a/p*b+0.5))%p+p)%p;
}

// 指令集快速乘
// 试机记得测试能不能过编译
inline long long mul(const long long a, const long long b, const long long p) {
    long long ans;
    __asm__ __volatile__ ("\tmulq %%rbx\n\tdivq %%rcx\n" :  "=d"(ans) : "a"(a), "b"(b), "c"(p));
```

```
    return ans;
}
```

$O(n^2)$高精度

```
// 注意如果只需要正数运算的话
// 可以只抄英文名的运算函数
// 按需自取
// 乘法O(n ^ 2)，除法O(10 * n ^ 2)

const int maxn = 1005;

struct big_decimal {
    int a[maxn];
    bool negative;

    big_decimal() {
        memset(a, 0, sizeof(a));
        negative = false;
    }

    big_decimal(long long x) {
        memset(a, 0, sizeof(a));
        negative = false;

        if (x < 0) {
            negative = true;
            x = -x;
        }

        while (x) {
            a[++a[0]] = x % 10;
            x /= 10;
        }
    }

    big_decimal(string s) {
        memset(a, 0, sizeof(a));
        negative = false;

        if (s == "")
            return;

        if (s[0] == '-') {
            negative = true;
            s = s.substr(1);
        }
        a[0] = s.size();
        for (int i = 1; i <= a[0]; i++)
            a[i] = s[a[0] - i] - '0';

        while (a[0] && !a[a[0]])
            a[0]--;
```

```
    }

    void input() {
        string s;
        cin >> s;
        *this = s;
    }

    string str() const {
        if (!a[0])
            return "0";

        string s;
        if (negative)
            s = "-";

        for (int i = a[0]; i; i--)
            s.push_back('0' + a[i]);

        return s;
    }

    operator string () const {
        return str();
    }

    big_decimal operator - () const {
        big_decimal o = *this;
        if (a[0])
            o.negative ^= true;

        return o;
    }

    friend big_decimal abs(const big_decimal &u) {
        big_decimal o = u;
        o.negative = false;
        return o;
    }

    big_decimal &operator <<= (int k) {
        a[0] += k;

        for (int i = a[0]; i > k; i--)
            a[i] = a[i - k];

        for(int i = k; i; i--)
            a[i] = 0;

        return *this;
    }

    friend big_decimal operator << (const big_decimal &u, int k) {
        big_decimal o = u;
        return o <<= k;
```

```
    }

    big_decimal &operator >>= (int k) {
        if (a[0] < k)
            return *this = big_decimal(0);

        a[0] -= k;
        for (int i = 1; i <= a[0]; i++)
            a[i] = a[i + k];

        for (int i = a[0] + 1; i <= a[0] + k; i++)
            a[i] = 0;

        return *this;
    }

    friend big_decimal operator >> (const big_decimal &u, int k) {
        big_decimal o = u;
        return o >>= k;
    }

    friend int cmp(const big_decimal &u, const big_decimal &v) {
        if (u.negative || v.negative) {
            if (u.negative && v.negative)
                return -cmp(-u, -v);

            if (u.negative)
                return -1;

            if (v.negative)
                return 1;
        }

        if (u.a[0] != v.a[0])
            return u.a[0] < v.a[0] ? -1 : 1;

        for (int i = u.a[0]; i; i--)
            if (u.a[i] != v.a[i])
                return u.a[i] < v.a[i] ? -1 : 1;

        return 0;
    }

    friend bool operator < (const big_decimal &u, const big_decimal &v) {
        return cmp(u, v) == -1;
    }

    friend bool operator > (const big_decimal &u, const big_decimal &v) {
        return cmp(u, v) == 1;
    }

    friend bool operator == (const big_decimal &u, const big_decimal &v)
{
        return cmp(u, v) == 0;
    }
```

```
158
159     friend bool operator <= (const big_decimal &u, const big_decimal &v)
    {
160         return cmp(u, v) <= 0;
161     }
162
163     friend bool operator >= (const big_decimal &u, const big_decimal &v)
    {
164         return cmp(u, v) >= 0;
165     }
166
167     friend big_decimal decimal_plus(const big_decimal &u, const
    big_decimal &v) { // 保证u, v均为正数的话可以直接调用
168         big_decimal o;
169
170         o.a[0] = max(u.a[0], v.a[0]);
171
172         for (int i = 1; i <= u.a[0] || i <= v.a[0]; i++) {
173             o.a[i] += u.a[i] + v.a[i];
174
175             if (o.a[i] >= 10) {
176                 o.a[i + 1]++;
177                 o.a[i] -= 10;
178             }
179         }
180
181         if (o.a[o.a[0] + 1])
182             o.a[0]++;
183
184         return o;
185     }
186
187     friend big_decimal decimal_minus(const big_decimal &u, const
    big_decimal &v) { // 保证u, v均为正数的话可以直接调用
188         int k = cmp(u, v);
189
190         if (k == -1)
191             return -decimal_minus(v, u);
192         else if (k == 0)
193             return big_decimal(0);
194
195         big_decimal o;
196
197         o.a[0] = u.a[0];
198
199         for (int i = 1; i <= u.a[0]; i++) {
200             o.a[i] += u.a[i] - v.a[i];
201
202             if (o.a[i] < 0) {
203                 o.a[i] += 10;
204                 o.a[i + 1]--;
205             }
206         }
207
208         while (o.a[0] && !o.a[o.a[0]])
```

```
            o.a[0]--;

        return o;
    }

    friend big_decimal decimal_multi(const big_decimal &u, const
big_decimal &v) {
        big_decimal o;

        o.a[0] = u.a[0] + v.a[0] - 1;

        for (int i = 1; i <= u.a[0]; i++)
            for (int j = 1; j <= v.a[0]; j++)
                o.a[i + j - 1] += u.a[i] * v.a[j];

        for (int i = 1; i <= o.a[0]; i++)
            if (o.a[i] >= 10) {
                o.a[i + 1] += o.a[i] / 10;
                o.a[i] %= 10;
            }

        if (o.a[o.a[0] + 1])
            o.a[0]++;

        return o;
    }

    friend pair<big_decimal, big_decimal> decimal_divide(big_decimal u,
big_decimal v) { // 整除
        if (v > u)
            return make_pair(big_decimal(0), u);

        big_decimal o;
        o.a[0] = u.a[0] - v.a[0] + 1;

        int m = v.a[0];
        v <<= u.a[0] - m;

        for (int i = u.a[0]; i >= m; i--) {
            while (u >= v) {
                u = u - v;
                o.a[i - m + 1]++;
            }

            v >>= 1;
        }

        while (o.a[0] && !o.a[o.a[0]])
            o.a[0]--;

        return make_pair(o, u);
    }

    friend big_decimal operator + (const big_decimal &u, const
big_decimal &v) {
```

```
261         if (u.negative || v.negative) {
262             if (u.negative && v.negative)
263                 return -decimal_plus(-u, -v);
264 
265             if (u.negative)
266                 return v - (-u);
267 
268             if (v.negative)
269                 return u - (-v);
270         }
271 
272         return decimal_plus(u, v);
273     }
274 
275     friend big_decimal operator - (const big_decimal &u, const
    big_decimal &v) {
276         if (u.negative || v.negative) {
277             if (u.negative && v.negative)
278                 return -decimal_minus(-u, -v);
279 
280             if (u.negative)
281                 return -decimal_plus(-u, v);
282 
283             if (v.negative)
284                 return decimal_plus(u, -v);
285         }
286 
287         return decimal_minus(u, v);
288     }
289 
290     friend big_decimal operator * (const big_decimal &u, const
    big_decimal &v) {
291         if (u.negative || v.negative) {
292             big_decimal o = decimal_multi(abs(u), abs(v));
293 
294             if (u.negative ^ v.negative)
295                 return -o;
296             return o;
297         }
298 
299         return decimal_multi(u, v);
300     }
301 
302     big_decimal operator * (long long x) const {
303         if (x >= 10)
304             return *this * big_decimal(x);
305 
306         if (negative)
307             return -(*this * x);
308 
309         big_decimal o;
310 
311         o.a[0] = a[0];
312 
313         for (int i = 1; i <= a[0]; i++) {
```

```
            o.a[i] += a[i] * x;

            if (o.a[i] >= 10) {
                o.a[i + 1] += o.a[i] / 10;
                o.a[i] %= 10;
            }
        }

        if (o.a[a[0] + 1])
            o.a[0]++;

        return o;
    }

    friend pair<big_decimal, big_decimal> decimal_div(const big_decimal
&u, const big_decimal &v) {
        if (u.negative || v.negative) {
            pair<big_decimal, big_decimal> o = decimal_div(abs(u),
abs(v));

            if (u.negative ^ v.negative)
                return make_pair(-o.first, -o.second);
            return o;
        }

        return decimal_divide(u, v);
    }

    friend big_decimal operator / (const big_decimal &u, const
big_decimal &v) { // v不能是0
        if (u.negative || v.negative) {
            big_decimal o = abs(u) / abs(v);

            if (u.negative ^ v.negative)
                return -o;
            return o;
        }

        return decimal_divide(u, v).first;
    }

    friend big_decimal operator % (const big_decimal &u, const
big_decimal &v) {
        if (u.negative || v.negative) {
            big_decimal o = abs(u) % abs(v);

            if (u.negative ^ v.negative)
                return -o;
            return o;
        }

        return decimal_divide(u, v).second;
    }
};
```

xorshift

xorshift128 Plus

```
ull k1, k2;
const int mod = 10000000;
ull xorShift128Plus() {
    ull k3 = k1, k4 = k2;
    k1 = k4;
    k3 ^= (k3 << 23);
    k2 = k3 ^ k4 ^ (k3 >> 17) ^ (k4 >> 26);
    return k2 + k4;
}
void gen(ull _k1, ull _k2) {
    k1 = _k1, k2 = _k2;
    int x = xorShift128Plus() % threshold + 1;
    // do sth
}
```

xorshift128

```
uint32_t xor128(void) {
    static uint32_t x = 123456789;
    static uint32_t y = 362436069;
    static uint32_t z = 521288629;
    static uint32_t w = 88675123;
    uint32_t t;

    t = x ^ (x << 11);
    x = y; y = z; z = w;
    return w = w ^ (w >> 19) ^ (t ^ (t >> 8));
}
```

杂の参考资料

常见数列

斐波那契数

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

$$F_{2k} = F_k\left(2F_{k+1} - F_k\right) \; F_{2k+1} = F_{k+1}^2 + F_k^2$$

性质

1. 卡西尼性质（Cassini's identity）：$F_{n-1}F_{n+1} - F_n^2 = (-1)^n$。
2. 附加性质：$F_{n+k} = F_kF_{n+1} + F_{k-1}F_n$。
3. 取上一条性质中 $k = n$，我们得到 $F_{2n} = F_n(F_{n+1} + F_{n-1})$。
4. 由上一条性质可以归纳证明，$\forall k \in \mathbb{N}, F_n | F_{nk}$。
5. 上述性质可逆，即 $\forall F_a | F_b, a | b$。
6. GCD 性质：$(F_m, F_n) = F_{(m,n)}$。

周期性

考虑模 p 意义下的斐波那契数列，可以容易地使用抽屉原理证明，该数列是有周期性的。考虑模意义下前 p^2+1 个斐波那契数对（两个相邻数配对）：

$$(F_1,\ F_2),(F_2,\ F_3),\ \ldots,\ (F_{p^2+1},\ F_{p^2+2})$$

p 的剩余系大小为 p，意味着在前 p^2+1 个数对中必有两个相同的数对，于是这两个数对可以往后生成相同的斐波那契数列，那么他们就是周期性的。

事实上，我们有一个远比它要强的结论。模 n 意义下斐波那契数列的周期被称为皮萨诺周期，该数可以证明总是不超过 $6n$，且只有在满足 $n=2\times 5^k$ 的形式时才取到等号。

卡特兰数

以下问题属于 Catalan 数列：

1. 有 $2n$ 个人排成一行进入剧场。入场费 5 元。其中只有 n 个人有一张 5 元钞票，另外 n 人只有 10 元钞票，剧院无其它钞票，问有多少中方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零？
2. 一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作。每天她走 $2n$ 个街区去上班。如果他从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路？
3. 在圆上选择 $2n$ 个点，将这些点成对连接起来使得所得到的 n 条线段不相交的方法数？
4. 对角线不相交的情况下，将一个凸多边形区域分成三角形区域的方法数？
5. 一个栈（无穷大）的进栈序列为 $1,2,3,\cdots,n$ 有多少个不同的出栈序列？
6. n 个结点可构造多少个不同的二叉树？
7. n 个 $+1$ 和 n 个 -1 构成 $2n$ 项 $a_1,a_2,\cdots,a_{2n}$，其部分和满足 $a_1+a_2+\cdots+a_k\geq 0(k=1,2,3,\cdots,2n)$ 对与 n 该数列为？

递推式：

$$H_n=\frac{\binom{2n}{n}}{n+1}(n\geq 2,n\in\mathbf{N}_+)$$

关于 Catalan 数的常见公式：

$$H_n=\begin{cases}\sum_{i=1}^{n}H_{i-1}H_{n-i} & n\geq 2,n\in\mathbf{N}_+\\ 1 & n=0,1\end{cases}$$

$$H_n=\frac{H_{n-1}(4n-2)}{n+1}$$

$$H_n=\binom{2n}{n}-\binom{2n}{n-1}$$

伯努利数

$$B(x)=\sum_{i\geq 0}\frac{B_i x^i}{i!}=\frac{x}{e^x-1}$$

$$B_n=[n=0]-\sum_{i=0}^{n-1}\binom{n}{i}\frac{B_i}{n-k+1}$$

$$\sum_{i=0}^{n}\binom{n+1}{i}B_i=0$$

$$S_n(m)=\sum_{i=0}^{m-1}i^n=\sum_{i=0}^{n}\binom{n}{i}B_{n-i}\frac{m^{i+1}}{i+1}$$

斯特林数

第一类斯特林数

$\begin{bmatrix} n \\ k \end{bmatrix}$表示$n$个元素划分成$k$个轮换的方案数

求同一行：分治FFT $O(n \log^2 n)$

求同一列：用一个轮换的指数生成函数做k次幂

第二类斯特林数

$\begin{Bmatrix} n \\ k \end{Bmatrix}$表示$n$个元素划分成$k$个子集的方案数

求一个：容斥，狗都会做

求同一行：FFT，狗都会做

上升幂、普通幂与下降幂的转换

$$x^{\overline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

$$x^n = \sum_k \begin{Bmatrix} n \\ k \end{Bmatrix} (-1)^{n-k} x^{\overline{k}}$$

$$x^n = \sum_k \begin{Bmatrix} n \\ k \end{Bmatrix} x^{\underline{k}}$$

$$x^{\underline{n}} = \sum_k (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

贝尔数

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

$$B(x) = \frac{x}{e^x - 1}$$

常见数列打表

卡特兰数

```
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900,
2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420,
24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452,
18367353072152, 69533550916004, 263747951750360, 1002242216651368,
3814986502092304
```

贝尔数

```
1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597,
27644437, 190899322, 1382958545, 10480142147, 82864869804, 682076806159,
5832742205057, 51724158235372, 474869816156751, 4506715738447323,
44152005855084346, 445958869294805289, 4638590332229999353,
49631246523618756274
```

斐波那契数

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,
317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465,
14930352, 24157817, 39088169, 63245986, 10233415
```

Lucas数列

$(L_0 = 2, L_1 = 1)$

$$L_n = \left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n$$

```
2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207,
3571, 5778, 9349, 15127, 24476, 39603, 64079, 103682, 167761, 271443,
439204, 710647, 1149851, 1860498, 3010349, 4870847, 7881196, 12752043,
20633239, 33385282, 54018521, 87403803
```

五边形数

$n(3n-1)/2$

```
0, 1, 5, 12, 22, 35, 51, 70, 92, 117, 145, 176, 210, 247, 287, 330, 376,
425, 477, 532, 590, 651, 715, 782, 852, 925, 1001, 1080, 1162, 1247, 1335,
1426, 1520, 1617, 1717, 1820, 1926, 2035, 2147, 2262, 2380, 2501, 2625,
2752, 2882, 3015, 3151
```

错位排列数

$f_i = (i-1)\left(f_{i-1} + f_{i-2}\right)$

```
1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841,
2290792932, 32071101049, 481066515734, 7697064251745, 130850092279664,
2355301661033953, 44750731559645106, 895014631192902121,
18795307255050944540, 413496759611120779881, 9510425471055777937262
```

无标号有根树计数

```
0, 1, 1, 2, 4, 9, 20, 48, 115, 286, 719, 1842, 4766, 12486, 32973, 87811,
235381, 634847, 1721159, 4688676, 12826228, 35221832, 97055181, 268282855,
743724984, 2067174645, 5759636510, 16083734329, 45007066269, 126186554308,
354426847597
```

(无根树考虑只把重心作为树根即可，n为偶数的情况需要减掉两半完全对称的情况)

常用NTT素数及原根

(只整理了可能有用的，太大或者太小的没加进去)

(比较重要的模数和不寻常的原根用粗体标注)

$p=r\times 2^k+1$	r	k	最小原根
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
985661441	235	22	3
998244353	119	23	3
1004535809	479	21	3
1005060097	1917	19	**5**
2013265921	15	27	**31**
2281701377	17	27	3
31525197391593473	7	52	3
180143985094819841	5	55	**6**
1945555039024054273	27	56	**5**
4179340454199820289	29	57	3

注：

1005060097有点危险，在变换长度大于524288时不可使用。

常用素数

10^9+7，10^9+9，10^8+7，10^6+3，23333333333333333（16个3），10^4+7，19260817

（当然上面的NTT素数也是）

C++11新特性

`constexpr`是C++11的新特性（小声）

Lambda表达式

Lambda 表达式是能够捕获作用域中的变量的无名函数对象，我们可以将其理解为一个匿名的内联函数。下面是 Lambda 表达式的语法：

```
[capture] (parameters) -> return-type {statement}
```

capture 捕获子句

Lambda 表达式以 capture 子句开头，它指定哪些变量被捕获，以及捕获是通过值还是引用：有 `&` 符号前缀的变量通过引用访问，没有该前缀的变量通过值访问。空的 capture 子句 `[]` 指示 Lambda 表达式的主体不访问封闭范围中的变量。

我们也可以使用默认捕获模式：`&` 表示捕获到的所有变量都通过引用访问，`=` 表示捕获到的所有变量都通过值访问。之后我们可以为特定的变量 **显式** 指定相反的模式。

例如 Lambda 体要通过引用访问外部变量 `a` 并通过值访问外部变量 `b`，则以下子句等效：

- `[&a, b]`
- `[b, &a]`
- `[&, b]`
- `[b, &]`
- `[=, &a]`
- `[&a, =]`

默认捕获时，会捕获 Lambda 中提及的变量。

编译选项

- `-O2 -g -std=c++11`：狗都知道
- `-Wall -Wextra -Wconversion`：更多警告
- `-fsanitize=(address/undefined)`：检查整数溢出（算ub）/数组越界
  - 注意无符号类型溢出不算ub

Linux命令行

- `diff`：比较文件
- `time`：测时间

__builtin

- `__builtin_popcount` 统计1个数
- `__builtin_parity` 统计1个数奇偶性
- `__builtin_ctz` 查询末尾有几个0（0不能用）
- `__builtin_clz` 查询有几个前导0（0不能用）

后面加 `ll` 就可以查询long long了，例如 `__builtin_ctzll`。

打表和分段打表

没什么内容，但是你看见这个东西就会有想法

注意事项

Linux注意事项

- Linux中的 `rand()` 返回值是int范围，Windows下最多只有32767
- Linux下函数没有return很可能会报错，然而Windows很可能不会
- Linux下大多数情况 `python` 指的是Py2.x，如果要使用3.x需要 `python3`

Linux中禁止使用的变量名

```
pipe, size, time, next, map
```

编译检查事项

- 是否支持c++11

- 指令集快速乘
- `#include <bits/stdc++.h>`
- `pb_ds`, `rope`
- `__int128`, `__float128`

场外相关

- 安顿好之后查一下附近的咖啡店、打印店、便利店之类的位置，以备不时之需
- 热身赛记得检查一下编译注意事项中的代码能否过编译，还有熟悉比赛场地，清楚洗手间在哪儿，测试打印机（如果可以）
- 比赛前至少要翻一遍板子，尤其要看原理与例题
- 比赛前一两天不要摸鱼，要早睡，有条件最好洗个澡；比赛当天不要起太晚，维持好的状态
- 赛前记得买咖啡，最好直接安排三人份，记得要咖啡因比较足的；如果主办方允许，就带些巧克力之类的高热量零食
- 入场之后记得检查机器，尤其要逐个检查键盘按键有没有坏的；如果可以的话，调一下gedit设置
- 开赛之前调整好心态，比赛而已，不必心急。

做题策略与心态调节

- 拿到题后立刻按照商量好的顺序读题，前半小时最好跳过题意太复杂的题（除非被过穿了）
- 签到题写完不要激动，稍微检查一下最可能的下毒点再交，避免无谓的罚时
    - 一两行的那种傻逼题就算了
- 读完题及时输出题意，一方面避免重复读题，一方面也可以让队友有一个初步印象，方便之后决定开题顺序
- 一个题如果卡了很久又有其他题可以写，那不妨先放掉写更容易的题，不要在一棵树上吊死
    - 不要被一两道题搞得心态爆炸，一方面急也没有意义，一方面你很可能真的离AC就差一步
- 榜是不会骗人的，一个题如果被不少人过了就说明这个题很可能并没有那么难；如果不是有十足的把握就不要轻易开没什么人交的题；另外不要忘记最后一小时会封榜
- 想不出题/找不出毒自然容易犯困，一定不要放任自己昏昏欲睡，最好去洗手间冷静一下，没有条件就站起来踱步
- 思考的时候不要挂机，一定要在草稿纸上画一画，最好说出声来最不容易断掉思路
- 出完算法一定要check一下样例和一些trivial的情况，不然容易写了半天发现写了个假算法
- 上机前有时间就提前给需要思考怎么写的地方打草稿，不要浪费机时
- 查毒时如果最难的地方反复check也没有问题，就从头到脚仔仔细细查一遍，不要放过任何细节，即使是并查集和sort这种东西也不能想当然
- 后半场如果时间不充裕就不要冒险开难题，除非真的无事可做
    - 如果是没写过的东西也不要轻举妄动，在有其他好写的题的时候就等一会再说
- 大多数时候都要听队长安排，虽然不一定最正确但可以保持组织性
- 最好注意一下影响，就算忍不住嘴臭也不要太大声
- 任何时候都不要着急，着急不能解决问题，不要当吉吉国王
- 输了游戏，还有人生；赢了游戏，还有人生。