Team Reference Document

Kilo_5723

2024 年 3 月 13 日

目录

1 数据结构

1.1 笛卡尔树

```
1
2  struct node {
3    array<node *, 2> ch;
4    int val, idx;
5    node(int val, int idx) : val(val), idx(idx) { ch[0] = ch[1] = NULL; }
6  };
7  node *build(vector<int> &a) {
8    vector<node *> stk;
9    for (int i = 0; i < a.size(); i++) {
10     node *last = NULL;
11     while (stk.size() && stk.back()->val > a[i]) {
12       last = stk.back();
13       stk.pop_back();
14     }
15     node *u = new node(a[i], i);
16     if (stk.size()) stk.back()->ch[1] = u;
17     if (last) u->ch[0] = last;
18     stk.push_back(u);
19   }
20   return stk[0];
21 }
```

1.2 Link-Cut Tree

```
1  struct lct {
2    vector<array<int, 2>> ch;
3    vector<int> fa;
4    vector<int> rev;
5    vector<int> siz;
6    lct(int n)
7        : ch(n + 1, {0, 0}), fa(n + 1, 0), rev(n + 1, false), siz(n + 1, 1) {
8      siz[0] = 0;  // initialize nil
9    }
10   void update(int u) { siz[u] = siz[ch[u][0]] + siz[ch[u][1]] + 1; }
11   void reverse(int u) {
12     if (u) rev[u] ^= 1, swap(ch[0], ch[1]);
13   }
14   void pushdown(int u) {
15     if (rev[u]) reverse(ch[u][0]), reverse(ch[u][1]);
```

```
16     rev[u] = false;
17   }
18   int chid(int u) {
19     if (ch[fa[u]][0] == u) return 0;
20     if (ch[fa[u]][1] == u) return 1;
21     return -1;
22   }
23   bool isroot(int u) { return chid(u) == -1; }
24   void rotate(int u) {
25     int v = fa[u], w = fa[v], k = chid(u), x = ch[u][!k];
26     if (!isroot(v)) ch[w][chid(v)] = u;
27     ch[u][!k] = v, ch[v][k] = x;
28     if (x) fa[x] = v;
29     fa[v] = u, fa[u] = w;
30     update(v), update(u);
31   }
32   void clearup(int u) {
33     if (!isroot(u)) clearup(fa[u]);
34     pushdown(u);
35   }
36   void splay(int u) {
37     clearup(u);
38     while (!isroot(u)) {
39       int f = fa[u];
40       if (!isroot(f)) rotate(chid(u) == chid(f) ? f : u);
41       rotate(u);
42     }
43   }
44   int access(int u) {
45     int v;
46     for (v = 0; u; v = u, u = fa[u]) splay(u), ch[u][1] = v, update(u);
47     return v;
48   }
49   void makeroot(int u) {
50     u = access(u);
51     reverse(u);
52   }
53   void link(int u, int v) {
54     makeroot(u);
55     splay(u);
56     fa[u] = v;
57   }
58   void cut(int u,int v){
59     makeroot(u);
60     access(v);
```

```
61      splay(v);
62      ch[v][0]=fa[u]=0;
63      update(v);
64    }
65  };
```

1.3 李超线段树

```
1   #include <optional>
2
3   #include "bits/stdc++.h"
4   using namespace std;
5   const char el = '\n';
6   typedef long long ll;
7
8   const ll inf = 1e18;
9   struct node {
10    node *ls, *rs;
11    int pl, pm, pr;
12    int xl, xm, xr;
13    ll k, b;
14    int tag;
15    node(int l, int r) {
16      pl = l, pr = r, pm = l + (r - l >> 1);
17      xl = xm = xr = 0;
18      k = 0, b = 0;
19      tag = 0;
20      if (l == r) {
21        ls = rs = NULL;
22        return;
23      }
24      ls = new node(l, pm);
25      rs = new node(pm + 1, r);
26    }
27    void addtag(int t) {
28      tag += t;
29      b -= k * t;
30      xl += t, xm += t, xr += t;
31    }
32    void pushtag() {
33      if (ls) ls->addtag(tag);
34      if (rs) rs->addtag(tag);
35      tag = 0;
36    }
37    void addfunc(int l, int r, ll _k, ll _b) {
38      if (r < pl || l > pr) return;
39      pushtag();
40      if (l <= pl && r >= pr) {
41        if (k * xm + b < _k * xm + _b) {
42          swap(k, _k);
43          swap(b, _b);
44        }
45        if (k * xl + b < _k * xl + _b && ls) ls->addfunc(l, r, _k, _b);
46        if (k * xr + b < _k * xr + _b && rs) rs->addfunc(l, r, _k, _b);
47        return;
48      }
49      if (ls) ls->addfunc(l, r, _k, _b);
50      if (rs) rs->addfunc(l, r, _k, _b);
51    }
52    void addx(int p, int v) {
53      if (pr < p) return;
54      if (pl >= p) {
55        addtag(v);
56        return;
57      }
58      if (pl >= p) xl += v;
59      if (pm >= p) xm += v;
60      if (pr >= p) xr += v;
61      if (ls) ls->addfunc(pl, pr, k, b);
62      if (rs) rs->addfunc(pl, pr, k, b);
63      pushtag();
64      if (ls) ls->addx(p, v);
65      if (rs) rs->addx(p, v);
66      k = 0, b = 0;
67    }
68    optional<pair<int, ll>> query(int p) {
69      if (p < pl || p > pr) return nullopt;
70      if (pl == pr) return pair<int, ll>{xm, k * xm + b};
71      pushtag();
72      auto res = ls->query(p);
73      if (!res) res = rs->query(p);
74      auto [x, y] = *res;
75      return pair<int, ll>{x, max(y, k * x + b)};
76    }
77  };
78  int main() {
79    ios::sync_with_stdio(false);
80    cin.tie(0);
81    cout << setprecision(15);
```

```
82    int tt;
83    cin >> tt;
84    while (tt--) {
85      int n, m;
86      cin >> n >> m;
87      vector<vector<int>> a(m + 1);
88      vector<int> c(m + 1);
89      while (n--) {
90        int l, r;
91        cin >> l >> r;
92        a[l].push_back(r);
93        c[r]++;
94      }
95      vector<ll> p(m + 1);
96      for (int i = 1; i <= m; i++) cin >> p[i];
97      node *rt = new node(0, m + 1);
98      for (int i = 1; i <= m; i++) {
99        auto [x, y] = *rt->query(i);
100       rt->addx(i, -c[i - 1]);
101       sort(a[i].begin(), a[i].end());
102       for (auto j : a[i]) rt->addx(j + 1, 1);
103       rt->addfunc(0, m + 1, p[i], y);
104     }
105     cout << rt->query(m + 1)->second << el;
106   }
107   return 0;
108 }
109 // https://codeforces.com/contest/1830/problem/F
```

1.4 可持久化线段树

```
1  struct node {
2    int l, r;
3    node *ls, *rs;
4    int tag;
5    node() : l(0), r(0), ls(nullptr), rs(nullptr), tag(0) {}
6    static void* operator new(size_t count) {
7      static node *begin = nullptr, *end = nullptr;
8      if (begin == end) begin = (node*)malloc(count * 1000), end = begin + 1000;
9      return begin++;
10   }
11 };
12 node* build(int l, int r) {
13   node* u = new node();
14   u->l = l, u->r = r;
15   if (l == r) return u;
16   int m = l + (r - l) / 2;
17   u->ls = build(l, m);
18   u->rs = build(m + 1, r);
19   return u;
20 }
21 node* addtag(node* u, int val) {
22   u = new node(*u);
23   u->tag += val;
24   return u;
25 }
26 node* pushdown(node* u) {
27   u = new node(*u);
28   if (u->tag) {
29     u->ls = addtag(u->ls, u->tag);
30     u->rs = addtag(u->rs, u->tag);
31     u->tag = 0;
32   }
33   return u;
34 }
35 node* add(node* u, int l, int r, int val) {
36   if (u->l > r || u->r < l) return u;
37   if (l <= u->l && u->r <= r) return addtag(u, val);
38   u = pushdown(u);
39   u->ls = add(u->ls, l, r, val);
40   u->rs = add(u->rs, l, r, val);
41   return u;
42 }
43 node* merge(node* u, node* v, int p) {
44   if (u->r <= p) return u;
45   if (v->l > p) return v;
46   u = pushdown(u), v = pushdown(v);
47   node* w = new node(*u);
48   w->ls = merge(u->ls, v->ls, p);
49   w->rs = merge(u->rs, v->rs, p);
50   return w;
51 }
52 int val(node* u, int p) {
53   if (u->r < p || u->l > p) return 0;
54   if (u->l == u->r) return u->tag;
55   return u->tag + val(u->ls, p) + val(u->rs, p);
56 }
```

1.5 区间处理

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct range {
  int l, r;
  bool valid() const { return r >= l; }
  bool cover(range rg) const { return l <= rg.l && r >= rg.r; }
  bool cross(range rg) const { return l <= rg.r && r >= rg.l; }
};
bool operator<(range a, range b) {
  return a.l < b.l || a.l == b.l && a.r > b.r;
}
struct rngseq {
  static const int inf = 1e9 + 7;
  set<range> seq;
  rngseq() : seq({{-inf, -inf}, {inf, inf}}) {}
  void add(range rg) {
    if (!rg.valid()) return;
    seq.insert(rg);
    rt->qadd(rg.l, rg.r, 1);
  }
  void rmv(range rg) {
    if (!rg.valid()) return;
    seq.erase(rg);
    rt->qadd(rg.l, rg.r, -1);
  }
  void insert(range rg) {
    auto it = seq.upper_bound(rg);
    auto tmp = *prev(it);
    if (tmp.cover(rg)) return;
    if (tmp.cross(rg)) {
      rg.l = tmp.l;
      rmv(tmp);
    }
    while (rg.cross(*it)) {
      tmp = *it++;
      rg.r = max(rg.r, tmp.r);
      rmv(tmp);
    }
    add(rg);
  }
  void erase(range rg) {
    auto it = seq.upper_bound(rg);
    auto tmp = *prev(it);
    if (tmp.cover(rg)) {
      rmv(tmp);
      add({tmp.l, rg.l - 1}), add({rg.r + 1, tmp.r});
      return;
    }
    if (tmp.cross(rg)) {
      rmv(tmp);
      add({tmp.l, rg.l - 1});
    }
    while (rg.cross(*it)) {
      tmp = *it++;
      rmv(tmp);
      add({rg.r + 1, tmp.r});
    }
  }
};
```

1.6 圆方树

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

const int MN = 100005;

int N, M, cnt;
std::vector<int> G[MN], T[MN * 2];

int dfn[MN], low[MN], dfc;
int stk[MN], tp;

void Tarjan(int u) {
  printf("  Enter : #%d\n", u);
  low[u] = dfn[u] = ++dfc;                     // low 初始化为当前节点 dfn
  stk[++tp] = u;                               // 加入栈中
  for (int v : G[u]) {                         // 遍历 u 的相邻节点
    if (!dfn[v]) {                             // 如果未访问过
      Tarjan(v);                               // 递归
      low[u] = std::min(low[u], low[v]);       // 未访问的和 low 取 min
      if (low[v] == dfn[u]) {                  // 标志着找到一个以 u 为根的点双连通分量
        ++cnt;                                 // 增加方点个数
        printf("  Found a New BCC #%d.\n", cnt - N);
                                               // 将点双中除了 u 的点退栈，并在圆方树中连边
```

```cpp
25        for (int x = 0; x != v; --tp) {
26          x = stk[tp];
27          T[cnt].push_back(x);
28          T[x].push_back(cnt);
29          printf("    BCC #%d has vertex #%d\n", cnt - N, x);
30        }
```
```cpp
32        T[cnt].push_back(u);
33        T[u].push_back(cnt);
34        printf("    BCC #%d has vertex #%d\n", cnt - N, u);
35      }
36    } else
37      low[u] = std::min(low[u], dfn[v]);  // 已访问的和 dfn 取 min
38  }
39  printf("  Exit : #%d : low = %d\n", u, low[u]);
40  printf("  Stack:\n    ");
41  for (int i = 1; i <= tp; ++i) printf("%d, ", stk[i]);
42  puts("");
43 }
44
45 int main() {
46   scanf("%d%d", &N, &M);
47   cnt = N;  // 点双 / 方点标号从 N 开始
48   for (int i = 1; i <= M; ++i) {
49     int u, v;
50     scanf("%d%d", &u, &v);
51     G[u].push_back(v);  // 加双向边
52     G[v].push_back(u);
53   }
54   // 处理非连通图
55   for (int u = 1; u <= N; ++u)
56     if (!dfn[u]) Tarjan(u), --tp;
57   // 注意到退出 Tarjan 时栈中还有一个元素即根，将其退栈
58   return 0;
59 }
```

1.7 线段树

```cpp
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 const ll inf = 1e18;
6 struct node {
7   node *ls, *rs;
8   int l, r;
9   ll val, sum;
10  pair<ll, int> mxm, mnm;
11  static node *newnd() {
12    static const int buff = 1000;
13    static node *ptr = new node[buff], *cur = ptr;
14    if (cur == ptr + buff) ptr = new node[buff], cur = ptr;
15    return cur++;
16  }
17  void update() {
18    sum = ls->sum + rs->sum;
19    mxm = max(ls->mxm, rs->mxm);
20    mnm = min(ls->mnm, rs->mnm);
21  }
22  void build(int _l, int _r, vector<ll> &a) {
23    l = _l, r = _r;
24    if (l == r) {
25      val = a[l];
26      mxm = mnm = {val, l};
27      sum = abs(val);
28      return;
29    }
30    int m = (l + r) / 2;
31    ls = new node(), ls->build(l, m, a);
32    rs = new node(), rs->build(m + 1, r, a);
33    update();
34  }
35  void modify(int p, int a) {
36    if (p < l || p > r) return;
37    if (l == r) {
38      val = val + a;
39      mxm = mnm = {val, l};
40      sum = abs(val);
41      return;
42    }
43    ls->modify(p, a), rs->modify(p, a);
44    update();
45  }
46  int ppos(int _l, int _r) {
47    if (_r < l || _l > r) return -1;
48    if (_l <= l && _r >= r) {
49      if (mxm.first < 0) return -1;
50      if (l == r) return l;
51    }
```

```
52      auto res = ls->ppos(_l, _r);
53      if (~res)
54        return res;
55      else
56        return rs->ppos(_l, _r);
57    }
58    pair<ll, int> qmxm(int _l, int _r) {
59      if (_r < l || _l > r) return {-inf, -1};
60      if (_l <= l && _r >= r) return mxm;
61      return max(ls->qmxm(_l, _r), rs->qmxm(_l, _r));
62    }
63    pair<ll, int> qmnm(int _l, int _r) {
64      if (_r < l || _l > r) return {inf, -1};
65      if (_l <= l && _r >= r) return mnm;
66      return min(ls->qmnm(_l, _r), rs->qmnm(_l, _r));
67    }
68    ll qsum(int _l, int _r) {
69      if (_r < l || _l > r) return 0;
70      if (_l <= l && _r >= r) return sum;
71      return ls->qsum(_l, _r) + rs->qsum(_l, _r);
72    }
73  };
74  auto newnd = node::newnd;
75
76  const ll inf = 1e18;
77  int ls(int u) { return u << 1; }
78  int rs(int u) { return u << 1 | 1; }
79  struct segtree {
80    vector<ll> mnm;
81    int l, r;
82    void build(int l_, int r_) {
83      l = l_, r = r_;
84      int k = 1;
85      while (k < r - l + 1) k <<= 1;
86      mnm.resize(k << 1);
87    }
88    void update(int u) { mnm[u] = min(mnm[ls(u)], mnm[rs(u)]); }
89    void init(int mode, int l = -1, int r = -1, int u = 1) {
90      if (!~l) l = this->l, r = this->r;
91      if (l == r) {
92        mnm[u] = (~mode ? mode ? inf : l : -l);
93        return;
94      }
95      int m = l + (r - l) / 2;
96      init(mode, l, m, ls(u));
```

```
97      init(mode, m + 1, r, rs(u));
98      update(u);
99    }
100   void modify(int p, ll a, int l = -1, int r = -1, int u = 1) {
101     if (!~l) l = this->l, r = this->r;
102     if (p < l || p > r) return;
103     if (l == r) {
104       mnm[u] = a;
105       return;
106     }
107     int m = l + (r - l) / 2;
108     modify(p, a, l, m, ls(u));
109     modify(p, a, m + 1, r, rs(u));
110     update(u);
111   }
112   ll qmnm(int l_, int r_, int l = -1, int r = -1, int u = 1) {
113     if (!~l) l = this->l, r = this->r;
114     if (r_ < l || l_ > r) return inf;
115     if (l_ <= l && r_ >= r) return mnm[u];
116     int m = l + (r - l) / 2;
117     return min(qmnm(l_, r_, l, m, ls(u)), qmnm(l_, r_, m + 1, r, rs(u)));
118   }
119 };
```

1.8 Treap

```
1   struct node {
2     array<node *, 2> ch;
3     cplx val, sum;
4     int rank;
5     node(cplx val) : val(val) {
6       ch[0] = ch[1] = NULL;
7       sum = val;
8       rank = rand();
9     }
10    void update() {
11      sum = val;
12      if (ch[0]) sum = ch[0]->sum + sum;
13      if (ch[1]) sum = sum + ch[1]->sum;
14    }
15  };
16  bool operator<(cplx a, cplx b) {
17    auto d = det(a, b);
18    if (d) return d < 0;
```

```
19 |     if (a.x != b.x) return a.x < b.x;
20 |     if (a.y != b.y) return a.y < b.y;
21 |     if (a.z != b.z) return a.z < b.z;
22 |     return false;
23 | }
24 | void rotate(node *&u, int c) {
25 |   auto v = u->ch[c];
26 |   u->ch[c] = v->ch[!c];
27 |   v->ch[!c] = u;
28 |   u->update();
29 |   v->update();
30 |   u = v;
31 | }
32 | node *insert(node *rt, cplx val) {
33 |   if (!rt) return new node(val);
34 |   auto c = rt->val < val;
35 |   rt->ch[c] = insert(rt->ch[c], val);
36 |   if (rt->ch[c]->rank < rt->rank) rotate(rt, c);
37 |   rt->update();
38 |   return rt;
39 | }
40 | void erase(node *&rt, cplx val) {
41 |   if (val < rt->val) {
42 |     erase(rt->ch[0], val);
43 |     rt->update();
44 |     return;
45 |   }
46 |   if (rt->val < val) {
47 |     erase(rt->ch[1], val);
48 |     rt->update();
49 |     return;
50 |   }
51 |   auto tmp = rt;
52 |   if (rt->ch[0]) {
53 |     if (rt->ch[1]) {
54 |       auto c = rt->ch[0]->rank > rt->ch[1]->rank;
55 |       rotate(rt, c);
56 |       erase(rt->ch[!c], val);
57 |       rt->update();
58 |     } else {
59 |       rt = rt->ch[0];
60 |       delete tmp;
61 |     }
62 |   } else {
63 |     if (rt->ch[1]) {
```

```
64 |       rt = rt->ch[1];
65 |       delete tmp;
66 |     } else {
67 |       rt = NULL;
68 |       delete tmp;
69 |     }
70 |   }
71 | }
72 |
73 |
74 | struct node {
75 |   array<node *, 2> ch;
76 |   ui p, n, avg;
77 |   ui sl, sr, tag, sum;
78 |   ui rank;
79 |   node(int l, int r) {
80 |     ch[0] = ch[1] = NULL;
81 |     sl = p = l, sr = n = r;
82 |     avg = tag = sum = 0;
83 |     rank = rand();
84 |   }
85 |   void addtag(ui t) {
86 |     tag += t, avg += t;
87 |     sum += (sr - sl) * t;
88 |   }
89 |   void pushdown() {
90 |     if (ch[0]) ch[0]->addtag(tag);
91 |     if (ch[1]) ch[1]->addtag(tag);
92 |     tag = 0;
93 |   }
94 |   void update() {
95 |     sum = (n - p) * avg;
96 |     sl = ch[0] ? ch[0]->sl : p;
97 |     sr = ch[1] ? ch[1]->sr : n;
98 |     if (ch[0]) sum += ch[0]->sum;
99 |     if (ch[1]) sum += ch[1]->sum;
100 |   }
101 |   void add(ui l, ui r, ui v) {
102 |     if (l >= sr || r <= sl) return;
103 |     if (l <= sl && r >= sr) {
104 |       addtag(v);
105 |       return;
106 |     }
107 |     pushdown();
108 |     if (l <= p && r >= n) avg += v;
```

```
109        if (ch[0]) ch[0]->add(l, r, v);
110        if (ch[1]) ch[1]->add(l, r, v);
111        update();
112      }
113      ui qsum(ui l, ui r) {
114        if (l >= sr || r <= sl) return 0;
115        if (l <= sl && r >= sr) return sum;
116        pushdown();
117        ui res = 0;
118        if (l <= p && r >= n) res += (n - p) * avg;
119        if (ch[0]) res += ch[0]->qsum(l, r);
120        if (ch[1]) res += ch[1]->qsum(l, r);
121        return res;
122      }
123    };
124    node *merge(node *u, node *v) {
125      if (!u) return v;
126      if (!v) return u;
127      if (u->rank < v->rank) {
128        u->pushdown();
129        u->ch[1] = merge(u->ch[1], v);
130        u->update();
131        return u;
132      } else {
133        v->pushdown();
134        v->ch[0] = merge(u, v->ch[0]);
135        v->update();
136        return v;
137      }
138    }
139    pair<node *, node *> split(node *u, ui p) {
140      if (!u) return {NULL, NULL};
141      u->pushdown();
142      if (u->p <= p) {
143        auto [l, r] = split(u->ch[1], p);
144        u->ch[1] = l;
145        u->update();
146        return {u, r};
147      } else {
148        auto [l, r] = split(u->ch[0], p);
149        u->ch[0] = r;
150        u->update();
151        return {l, u};
152      }
153    }
```

```
154    node *insert(node *u, ui p) {
155      auto [l, r] = split(u, p);
156      auto *v = l;
157      while (v->ch[1]) v = v->ch[1];
158      if (v->p == p) return merge(l, r);
159      auto w = new node(p, v->n);
160      v->n = p;
161      w->avg = v->avg;
162      v->update(), w->update();
163      w = merge(l, w), w = merge(w, r);
164      return w;
165    }
```

1.9   Trie

```
1
2    struct node {
3      array<node *, alpha> ch;
4      array<node *, sizf> fa;
5      int dep;
6      ll val;
7      node() {
8        for (auto &v : ch) v = NULL;
9        for (auto &v : fa) v = NULL;
10       dep = 0;
11       val = 0;
12     }
13   };
14   node *rt;
15   struct cmp {
16     bool operator()(node *a, node *b) {
17       if (a == b) return false;
18       if (!a) return true;
19       if (!b) return false;
20       if (a->dep != b->dep) return a->dep > b->dep;
21       for (int i = sizf - 1; i >= 0; i--)
22         if (a->fa[i] != b->fa[i]) a = a->fa[i], b = b->fa[i];
23       node *u = a->fa[0];
24       for (int i = 0; i < alpha; i++) {
25         if (a == u->ch[i]) return false;
26         if (b == u->ch[i]) return true;
27       }
28     }
29     bool operator()(pair<node *, int> a, pair<node *, int> b) {
```

```
30        return cmp()(a.first, b.first);
31    }
32 };
33 node *add(node *u, int n) {
34    if (n / 10) u = add(u, n / 10);
35    n %= 10;
36    if (u->ch[n]) return u->ch[n];
37    u->ch[n] = new node();
38    auto v = u->ch[n];
39    v->fa[0] = u;
40    for (int i = 1; i < sizf; i++) v->fa[i] = v->fa[i - 1]->fa[i - 1];
41    v->dep = u->dep + 1;
42    v->val = (u->val * 10 + n) % mod;
43    return v;
44 }
```

1.10 虚树

```
1  inline bool cmp(const int x, const int y) { return id[x] < id[y]; }
2
3  void build() {
4    sort(h + 1, h + k + 1, cmp);
5    sta[top = 1] = 1, g.sz = 0, g.head[1] = -1;
6    // 1 号节点入栈，清空 1 号节点对应的邻接表，设置邻接表边数为 1
7    for (int i = 1, l; i <= k; ++i)
8      if (h[i] != 1) {
9        // 如果 1 号节点是关键节点就不要重复添加
10       l = lca(h[i], sta[top]);
11       // 计算当前节点与栈顶节点的 LCA
12       if (l != sta[top]) {
13         // 如果 LCA 和栈顶元素不同，则说明当前节点不再当前栈所存的链上
14         while (id[l] < id[sta[top - 1]])
15           // 当次大节点的 Dfs 序大于 LCA 的 Dfs 序
16           g.push(sta[top - 1], sta[top]), top--;
17         // 把与当前节点所在的链不重合的链连接掉并且弹出
18         if (id[l] > id[sta[top - 1]])
19           // 如果 LCA 不等于次大节点（这里的大于其实和不等于没有区别）
20           g.head[l] = -1, g.push(l, sta[top]), sta[top] = l;
21         // 说明 LCA 是第一次入栈，清空其邻接表，连边后弹出栈顶元素，并将 LCA
22         // 入栈
23         else
24           g.push(l, sta[top--]);
25         // 说明 LCA 就是次大节点，直接弹出栈顶元素
26       }
```

```
27         g.head[h[i]] = -1, sta[++top] = h[i];
28         // 当前节点必然是第一次入栈，清空邻接表并入栈
29      }
30    for (int i = 1; i < top; ++i)
31      g.push(sta[i], sta[i + 1]);  // 剩余的最后一条链连接一下
32    return;
33 }
```

2 博弈

2.1 国际象棋

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const char el = '\n';
4  typedef long long ll;
5  typedef unsigned long long ull;
6  array<ull, 64> rook, bishop, knight, queen, arch, chan, maha;
7  void write(ull t, ull p) {
8    for (int i = 0; i < 8; i++, cout << el)
9      for (int j = 0; j < 8; j++) {
10       int x = i << 3 | j;
11       cout << (p >> x & 1 ? "o " : t >> x & 1 ? "  " : "x ");
12     }
13   cout << el;
14 }
15 void init() {
16   for (int i = 0; i < 8; i++)
17     for (int j = 0; j < 8; j++) {
18       auto &t = rook[i << 3 | j];
19       t = 0;
20       for (int x = 7; x >= 0; x--)
21         for (int y = 7; y >= 0; y--) t = t << 1 | (x == i || y == j);
22     }
23   for (int i = 0; i < 8; i++)
24     for (int j = 0; j < 8; j++) {
25       auto &t = bishop[i << 3 | j];
26       t = 0;
27       for (int x = 7; x >= 0; x--)
28         for (int y = 7; y >= 0; y--)
29           t = t << 1 | (i + j == x + y || i - j == x - y);
30     }
31   for (int i = 0; i < 8; i++)
```

```
32        for (int j = 0; j < 8; j++) {
33          auto &t = knight[i << 3 | j];
34          t = 0;
35          for (int x = 7; x >= 0; x--)
36            for (int y = 7; y >= 0; y--)
37              t = t << 1 | ((x != i && y != j && abs(x - i) + abs(y - j) == 3) ||
38                            x == i && y == j);
39        }
40    for (int i = 0; i < 64; i++) queen[i] = rook[i] | bishop[i];
41    for (int i = 0; i < 64; i++) arch[i] = bishop[i] | knight[i];
42    for (int i = 0; i < 64; i++) chan[i] = rook[i] | knight[i];
43    for (int i = 0; i < 64; i++) maha[i] = queen[i] | knight[i];
44  }
45  array<ull, 64> piece(char ch) {
46    switch (ch) {
47      case 'R':
48        return rook;
49      case 'B':
50        return bishop;
51      case 'Q':
52        return queen;
53      case 'A':
54        return arch;
55      case 'C':
56        return chan;
57      case 'M':
58        return maha;
59    }
60  }
61  int cnt = 100;
62  bool win(string s, int p, ull safe, ull used) {
63    if (p == s.size()) return false;
64    bool res = false;
65    const auto &pce = piece(s[p]);
66    for (int i = 0; i < 64 && !res; i++)
67      if ((safe >> i & 1) && !(used & pce[i]))
68        res = res || !win(s, p + 1, safe & ~pce[i], used | (1ull << i));
69    return res;
70  }
71  int main() {
72    ios::sync_with_stdio(false);
73    cin.tie(0);
74    cout << setprecision(15);
75    init();
76    string s;
77    cin >> s;
78    cout << (win(s, 0, -1ull, 0) ? "Alice" : "Bob") << el;
79    return 0;
80  }
81  // https://contest.ucup.ac/contest/1399/problem/7635
```

2.2 Nimber

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const char el = '\n';
4  typedef long long ll;
5  typedef unsigned long long ull;
6  array<ull, 64> rook, bishop, knight, queen, arch, chan, maha;
7  void write(ull t, ull p) {
8    for (int i = 0; i < 8; i++, cout << el)
9      for (int j = 0; j < 8; j++) {
10       int x = i << 3 | j;
11       cout << (p >> x & 1 ? "o " : t >> x & 1 ? "  " : "x ");
12     }
13   cout << el;
14 }
15 void init() {
16   for (int i = 0; i < 8; i++)
17     for (int j = 0; j < 8; j++) {
18       auto &t = rook[i << 3 | j];
19       t = 0;
20       for (int x = 7; x >= 0; x--)
21         for (int y = 7; y >= 0; y--) t = t << 1 | (x == i || y == j);
22     }
23   for (int i = 0; i < 8; i++)
24     for (int j = 0; j < 8; j++) {
25       auto &t = bishop[i << 3 | j];
26       t = 0;
27       for (int x = 7; x >= 0; x--)
28         for (int y = 7; y >= 0; y--)
29           t = t << 1 | (i + j == x + y || i - j == x - y);
30     }
31   for (int i = 0; i < 8; i++)
32     for (int j = 0; j < 8; j++) {
33       auto &t = knight[i << 3 | j];
34       t = 0;
35       for (int x = 7; x >= 0; x--)
36         for (int y = 7; y >= 0; y--)
```

```
37          t = t << 1 | ((x != i && y != j && abs(x - i) + abs(y - j) == 3) ||
38                        x == i && y == j);
39      }
40      for (int i = 0; i < 64; i++) queen[i] = rook[i] | bishop[i];
41      for (int i = 0; i < 64; i++) arch[i] = bishop[i] | knight[i];
42      for (int i = 0; i < 64; i++) chan[i] = rook[i] | knight[i];
43      for (int i = 0; i < 64; i++) maha[i] = queen[i] | knight[i];
44  }
45  array<ull, 64> piece(char ch) {
46      switch (ch) {
47        case 'R':
48          return rook;
49        case 'B':
50          return bishop;
51        case 'Q':
52          return queen;
53        case 'A':
54          return arch;
55        case 'C':
56          return chan;
57        case 'M':
58          return maha;
59      }
60  }
61  int cnt = 100;
62  bool win(string s, int p, ull safe, ull used) {
63      if (p == s.size()) return false;
64      bool res = false;
65      const auto &pce = piece(s[p]);
66      for (int i = 0; i < 64 && !res; i++)
67        if ((safe >> i & 1) && !(used & pce[i]))
68          res = res || !win(s, p + 1, safe & ~pce[i], used | (1ull << i));
69      return res;
70  }
71  int main() {
72      ios::sync_with_stdio(false);
73      cin.tie(0);
74      cout << setprecision(15);
75      init();
76      string s;
77      cin >> s;
78      cout << (win(s, 0, -1ull, 0) ? "Alice" : "Bob") << el;
79      return 0;
80  }
81  // https://contest.ucup.ac/contest/1399/problem/7635
```

2.3 一般博弈

```
1
2   unordered_map<ll, int> result;
3   struct status {
4     array<int, 10> a;
5     status() { fill(a.begin(), a.end(), 0); }
6     status flip() {
7       status b;
8       for (int i = 0; i < 5; i++) b.a[i] = a[i + 5];
9       for (int i = 0; i < 5; i++) b.a[i + 5] = a[i];
10      return b;
11    }
12    ll num() {
13      ll res = 0;
14      for (int i = 0; i < 10; i++) res = (res << 5) + a[i];
15      return res;
16    }
17    vector<status> from() {
18      vector<status> res;
19      for (int i = 0; i < 5; i++)
20        if (a[i])
21          for (int j = 0; j < 5; j++)
22            if (a[j + 5])
23              if (i && j - i) {
24                auto b = *this;
25                b.a[j + 5]--;
26                b.a[(j + 5 - i) % 5 + 5]++;
27                res.push_back(b.flip());
28              }
29    // cout << "from " << res.size() << el;
30      return res;
31    }
32    vector<status> to() {
33      vector<status> res;
34      for (int i = 0; i < 5; i++)
35        if (a[i])
36          for (int j = 0; j < 5; j++)
37            if (a[j + 5])
38              if (i && j) {
39                auto b = *this;
40                b.a[i]--;
```

```
41              b.a[(i + j) % 5]++;
42              res.push_back(b.flip());
43            }
44      // cout << "to: " << res.size() << el;
45      return res;
46    }
47    bool win() {
48      if (result.count(num())) return result[num()] == 1;
49      for (auto v : to())
50        if (result.count(v.num()) && result[v.num()] == -1) return true;
51      return false;
52    }
53    bool lose() {
54      if (result.count(num())) return result[num()] == -1;
55      for (auto v : to())
56        if (!result.count(v.num()) || result[v.num()] != 1) return false;
57      return true;
58    }
59  };
60  bool operator<(const status &a, const status &b) { return a.a < b.a; }
61  status read() {
62    status a;
63    for (int i = 0; i < 8; i++) {
64      int t;
65      cin >> t;
66      a.a[t]++;
67    }
68    for (int i = 0; i < 8; i++) {
69      int t;
70      cin >> t;
71      a.a[t + 5]++;
72    }
73    return a;
74  }
75
76  set<status> losestat(int t) {
77    if (!t) {
78      status tmp;
79      tmp.a[5] = 8;
80      return {tmp};
81    }
82    set<status> res;
83    auto tmp = losestat(t - 1);
84    for (auto v : tmp) {
85      for (int i = 0; i < 5; i++) {
86        auto t = v;
87        t.a[i]++;
88        res.insert(t);
89      }
90    }
91    return res;
92  }
93  set<status> winstat(int t) {
94    if (!t) {
95      status tmp;
96      tmp.a[0] = 8;
97      return {tmp};
98    }
99    set<status> res;
100   auto tmp = winstat(t - 1);
101   for (auto v : tmp) {
102     for (int i = 0; i < 5; i++) {
103       auto t = v;
104       t.a[i + 5]++;
105       res.insert(t);
106     }
107   }
108   return res;
109 }
110 void preprocess() {
111   auto wst = winstat(8), lst = losestat(8);
112   vector<status> win, lose;
113   for (auto v : wst) {
114     win.push_back(v);
115     result[v.num()] = 1;
116   }
117   for (auto v : lst) {
118     lose.push_back(v);
119     result[v.num()] = -1;
120   }
121   while (win.size() || lose.size()) {
122     if (win.size()) {
123       for (auto wn : win)
124         for (auto ls : wn.from()) {
125           if (!result.count(ls.num()) && ls.lose()) {
126             lose.push_back(ls);
127             result[ls.num()] = -1;
128           }
129         }
130       win.clear();
```

12

```
131      } else {
132        for (auto ls : lose)
133          for (auto wn : ls.from()) {
134            if (!result.count(wn.num()) && wn.win()) {
135              win.push_back(wn);
136              result[wn.num()] = 1;
137            }
138          }
139        lose.clear();
140      }
141    }
142  }
```

3 图论

3.1 点分树

```
1  #include "bits/stdc++.h"
2  using namespace std;
3  const char el = '\n';
4  typedef long long ll;
5  const ll inf = 1e18;
6  struct graph {
7    vector<vector<pair<int, ll>>> e;
8    vector<ll> r;
9    graph(int n) : e(n + 1), r(n + 1) {}
10   void adde(int u, int v, ll w) {
11     e[u].push_back({v, w});
12     e[v].push_back({u, w});
13   }
14   vector<int> siz;
15   int calcsiz(int u, int f) {
16     siz[u] = 1;
17     for (auto [v, w] : e[u])
18       if (v != f && ~siz[v]) siz[u] += calcsiz(v, u);
19     return siz[u];
20   }
21   vector<vector<pair<int, ll>>> fa;
22   vector<vector<pair<ll, int>>> ch;
23   void init(int u, int f, int rt, ll d) {
24     fa[u].push_back({rt, d});
25     ch[rt].push_back({r[u] - d, u});
26     for (auto [v, w] : e[u])
27       if (v != f && ~siz[v]) init(v, u, rt, d + w);
28   }
29   int split(int u) {
30     int n = calcsiz(u, u) / 2;
31     while (true) {
32       bool flg = true;
33       for (auto [v, w] : e[u])
34         if (siz[v] < siz[u] && siz[v] > n) {
35           u = v;
36           flg = false;
37           break;
38         }
39       if (flg) break;
40     }
41     siz[u] = -1;
42     init(u, u, u, 0);
43     sort(ch[u].begin(), ch[u].end());
44     for (auto [v, w] : e[u])
45       if (~siz[v]) v = split(v);
46     return u;
47   }
48   int rt;
49   void ctrdecomp() {
50     int n = e.size();
51     siz.assign(n, 0);
52     fa.assign(n, {});
53     ch.assign(n, {});
54     rt = split(1);
55   }
56   vector<int> solve() {
57     vector<int> dis(e.size(), -1);
58     dis[1] = 0;
59     queue<int> que;
60     que.push(1);
61     while (que.size()) {
62       int u = que.front();
63       que.pop();
64       for (auto [f, d] : fa[u]) {
65         while (ch[f].size()) {
66           auto [w, v] = ch[f].back();
67           if (d > w) break;
68           ch[f].pop_back();
69           if (~dis[v]) continue;
70           que.push(v);
71           dis[v] = dis[u] + 1;
```

```
72            }
73          }
74        }
75        return dis;
76      }
77    };
78    int main() {
79      ios::sync_with_stdio(false);
80      cin.tie(0);
81      cout << setprecision(15);
82      int tt;
83      cin >> tt;
84      while (tt--) {
85        int n;
86        cin >> n;
87        graph g(n);
88        for (int i = 2; i <= n; i++) cin >> g.r[i];
89        for (int i = 1; i < n; i++) {
90          ll u, v, w;
91          cin >> u >> v >> w;
92          g.adde(u, v, w);
93        }
94        g.ctrdecomp();
95        auto res = g.solve();
96        for (int i = 2; i <= n; i++) cout << res[i] << ' ';
97        cout << el;
98      }
99      return 0;
100   }
```

3.2 树链剖分

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   struct graph {
4     vector<vector<int>> e;
5     vector<int> fa, dep, siz, hvs, top, dfn, rnk;
6     vector<bit> seg;
7     graph(int n) : seg(maxd, n) {
8       e.resize(n + 1);
9       fa.resize(n + 1);
10      dep.resize(n + 1);
11      siz.resize(n + 1);
12      hvs.resize(n + 1);
```

```
13      top.resize(n + 1);
14      dfn.resize(n + 1);
15      rnk.resize(n + 1);
16    }
17    void adde(int u, int v) {
18      e[u].push_back(v);
19      e[v].push_back(u);
20    }
21    void subts(int u, int f, int d) {
22      fa[u] = f, dep[u] = d;
23      siz[u] = 1;
24      hvs[u] = -1;
25      for (auto v : e[u])
26        if (v != f) {
27          subts(v, u, d + 1);
28          siz[u] += siz[v];
29          if (!~hvs[u] || siz[v] > siz[hvs[u]]) hvs[u] = v;
30        }
31    }
32    int tot;
33    void ordfs(int u, int t) {
34      top[u] = t;
35      tot++, rnk[tot] = u, dfn[u] = tot;
36      if (~hvs[u]) ordfs(hvs[u], t);
37      for (auto v : e[u])
38        if (v != fa[u] && v != hvs[u]) ordfs(v, v);
39    }
40    void hvydecomp(int rt) {
41      subts(rt, -1, 0);
42      tot = 0;
43      ordfs(rt, rt);
44    }
45    void modify(int u, int v, int k, int d) {
46      while (top[u] != top[v]) {
47        if (dep[top[u]] < dep[top[v]]) swap(u, v);
48        seg[d].add(dfn[top[u]], dfn[u], k);
49        u = fa[top[u]];
50      }
51      if (dep[u] < dep[v]) swap(u, v);
52      seg[d].add(dfn[v], dfn[u], k);
53      u = v;
54      while (d--) {
55        seg[d].add(dfn[u], dfn[u], k);
56        u = fa[u];
57        seg[d].add(dfn[u], dfn[u], k);
```

```
58        }
59    }
60    int query(int u) {
61        int ans = 0;
62        for (int i = 0; i < maxd; i++) {
63            ans += seg[i].query(dfn[u]);
64            u = fa[u];
65        }
66        return ans;
67    }
68 };
```

3.3 费用流

```
1  #pragma once
2
3  #include <algorithm>
4  #include <optional>
5  #include <queue>
6  #include <vector>
7
8  namespace costflow {
9
10 typedef long long flow_t;
11 typedef long long cost_t;
12
13 const flow_t inf_flow = 1e18;
14 const cost_t inf_cost = 1e18;
15
16 struct CostEdge {
17     int from, to;
18     cost_t cost;
19     flow_t cap, low = 0, flow = 0;
20 };
21
22 int num_node(const std::vector<CostEdge> &edges) {
23     int n = 0;
24     for (const auto &e : edges) n = std::max({n, e.from, e.to});
25     return n;
26 }
27 std::pair<flow_t, cost_t> get_flow(const std::vector<CostEdge> &edges, int s) {
28     flow_t flow = 0;
29     cost_t cost = 0;
30     for (const auto &e : edges) {
```

```
31        if (e.from == s) flow += e.flow;
32        cost += e.flow * e.cost;
33    }
34    return {flow, cost};
35 }
36
37 struct CostFlow {
38     struct Edge {
39         int from, to;
40         cost_t cost;
41         flow_t cap;
42     };
43     int n;
44     std::vector<std::vector<int>> eid;
45     std::vector<Edge> edge;
46     void build(const std::vector<CostEdge> &edges) {
47         n = num_node(edges);
48         eid.assign(n + 1, {});
49         edge.clear();
50         int num_edges = 0;
51         for (const auto &e : edges) {
52             eid[e.from].push_back(num_edges++);
53             edge.push_back({e.from, e.to, e.cost, e.cap - e.flow});
54             eid[e.to].push_back(num_edges++);
55             edge.push_back({e.to, e.from, -e.cost, e.flow});
56         }
57     }
58     std::vector<cost_t> dis;
59     std::vector<int> pre;
60     bool spfa(int s, int t) {
61         if (s > n || t > n) return false;
62         dis.assign(n + 1, inf_cost);
63         pre.assign(n + 1, 0);
64         std::vector<bool> inque(n + 1);
65         std::queue<int> que;
66         dis[s] = 0;
67         que.push(s);
68         inque[s] = true;
69         while (que.size()) {
70             int u = que.front();
71             // cerr << 'u' << ' ' << u << endl;
72             que.pop();
73             inque[u] = false;
74             for (auto i : eid[u]) {
75                 const auto &e = edge[i];
```

```
76        if (e.cap && dis[e.to] > dis[u] + e.cost) {
77          dis[e.to] = dis[u] + e.cost;
78          pre[e.to] = i;
79          if (!inque[e.to]) {
80            que.push(e.to);
81            inque[e.to] = true;
82          }
83        }
84      }
85    }
86    return dis[t] < inf_cost;
87  }
88  std::pair<flow_t, cost_t> maxflow(int s, int t) {
89    flow_t flow = 0;
90    cost_t cost = 0;
91    while (spfa(s, t)) {
92      flow_t detf = inf_flow;
93      cost_t detc = 0;
94      for (int u = t, i = pre[u]; u != s; u = edge[i].from, i = pre[u]) {
95        detf = std::min(detf, edge[i].cap);
96        detc += edge[i].cost;
97      }
98      for (int u = t, i = pre[u]; u != s; u = edge[i].from, i = pre[u]) {
99        edge[i].cap -= detf;
100       edge[i ^ 1].cap += detf;
101     }
102     flow += detf;
103     cost += detf * detc;
104   }
105   return {flow, cost};
106 }
107 std::vector<CostEdge> to_edge() {
108   std::vector<CostEdge> edges;
109   for (int i = 0; i < edge.size(); i += 2)
110     edges.push_back({
111         .from = edge[i].from,
112         .to = edge[i].to,
113         .cost = edge[i].cost,
114         .cap = edge[i].cap + edge[i ^ 1].cap,
115         .flow = edge[i ^ 1].cap,
116     });
117   return edges;
118 }
119 };
120
```

```
121 struct Processor {
122   std::vector<bool> neg;
123   std::vector<flow_t> low;
124   std::vector<flow_t> excess;
125   void init(std::vector<CostEdge> &edges) {
126     int n = num_node(edges);
127     neg.clear();
128     neg.reserve(edges.size());
129     low.clear();
130     low.reserve(edges.size());
131     excess.assign(n + 1, 0);
132   }
133   void rmv_low(std::vector<CostEdge> &edges) {
134     for (auto &e : edges) {
135       low.push_back(e.low);
136       if (e.flow >= e.low) {
137         e.flow -= e.low;
138       } else {
139         excess[e.from] -= e.low - e.flow;
140         excess[e.to] += e.low - e.flow;
141         e.flow = 0;
142       }
143       e.cap -= e.low;
144       e.low = 0;
145     }
146   }
147   void add_low(std::vector<CostEdge> &edges) {
148     reverse(low.begin(), low.end());
149     for (auto &e : edges) {
150       e.low = low.back();
151       e.flow += e.low;
152       e.cap += e.low;
153       low.pop_back();
154     }
155   }
156   void rmv_neg(std::vector<CostEdge> &edges) {
157     for (auto &e : edges) {
158       neg.push_back(e.cost < 0);
159       if (e.cost < 0) {
160         excess[e.from] -= e.cap - e.flow;
161         excess[e.to] += e.cap - e.flow;
162         e.flow = e.cap;
163       }
164       if (e.cost > 0) {
165         excess[e.from] += e.flow;
```

16

```cpp
166          excess[e.to] -= e.flow;
167          e.flow = 0;
168        }
169      }
170    }
171  };
172
173  bool excess_flow(std::vector<CostEdge> &edges,
174                   const std::vector<flow_t> &excess) {
175    int n = num_node(edges), m = edges.size();
176    for (int i = 1; i <= n; i++) {
177      if (excess[i] > 0)
178        edges.push_back({.from = n + 1, .to = i, .cost = 0, .cap = excess[i]});
179      if (excess[i] < 0)
180        edges.push_back({.from = i, .to = n + 2, .cost = 0, .cap = -excess[i]});
181    }
182    CostFlow g;
183    g.build(edges);
184    g.maxflow(n + 1, n + 2);
185    edges = g.to_edge();
186    for (int i = m; i < edges.size(); i++)
187      if (edges[i].flow != edges[i].cap) return false;
188    edges.resize(m);
189    return true;
190  }
191
192  std::optional<std::pair<flow_t, cost_t>> feasible_flow(
193      std::vector<CostEdge> &edges, int s = 0, int t = 0) {
194    if (s && t) edges.push_back({.from = t, .to = s, .cost = 0, .cap = inf_flow});
195    Processor p;
196    p.init(edges);
197    p.rmv_low(edges);
198    p.rmv_neg(edges);
199    if (!excess_flow(edges, p.excess)) return std::nullopt;
200    if (s && t) edges.pop_back();
201    p.add_low(edges);
202    return get_flow(edges, s);
203  }
204
205  std::optional<std::pair<flow_t, cost_t>> maximum_flow(
206      std::vector<CostEdge> &edges, int s, int t) {
207    edges.push_back({.from = t, .to = s, .cost = 0, .cap = inf_flow});
208    Processor p;
209    p.init(edges);
210    p.rmv_low(edges);
211    p.rmv_neg(edges);
212    if (!excess_flow(edges, p.excess)) return std::nullopt;
213    edges.pop_back();
214    CostFlow g;
215    g.build(edges);
216    g.maxflow(s, t);
217    edges = g.to_edge();
218    p.add_low(edges);
219    return get_flow(edges, s);
220  }
221
222  std::optional<std::pair<flow_t, cost_t>> minimum_flow(
223      std::vector<CostEdge> &edges, int s, int t) {
224    edges.push_back({.from = t, .to = s, .cost = 0, .cap = inf_flow});
225    Processor p;
226    p.init(edges);
227    p.rmv_low(edges);
228    p.rmv_neg(edges);
229    if (!excess_flow(edges, p.excess)) return std::nullopt;
230    edges.pop_back();
231    CostFlow g;
232    for (auto &e : edges) e.cost = -e.cost;
233    Processor q;
234    q.rmv_neg(edges);
235    excess_flow(edges, q.excess);
236    g.build(edges);
237    g.maxflow(t, s);
238    edges = g.to_edge();
239    for (auto &e : edges) e.cost = -e.cost;
240    p.add_low(edges);
241    return get_flow(edges, s);
242  }
243
244  } // namespace costflow
```

3.4 点分树

```cpp
1  #include "bits/stdc++.h"
2  using namespace std;
3  const char el = '\n';
4  typedef long long ll;
5  const ll inf = 1e18;
6  struct graph {
7    vector<vector<pair<int, ll>>> e;
```

```cpp
    vector<ll> r;
    graph(int n) : e(n + 1), r(n + 1) {}
    void adde(int u, int v, ll w) {
      e[u].push_back({v, w});
      e[v].push_back({u, w});
    }
    vector<int> siz;
    int calcsiz(int u, int f) {
      siz[u] = 1;
      for (auto [v, w] : e[u])
        if (v != f && ~siz[v]) siz[u] += calcsiz(v, u);
      return siz[u];
    }
    vector<vector<pair<int, ll>>> fa;
    vector<vector<pair<ll, int>>> ch;
    void init(int u, int f, int rt, ll d) {
      fa[u].push_back({rt, d});
      ch[rt].push_back({r[u] - d, u});
      for (auto [v, w] : e[u])
        if (v != f && ~siz[v]) init(v, u, rt, d + w);
    }
    int split(int u) {
      int n = calcsiz(u, u) / 2;
      while (true) {
        bool flg = true;
        for (auto [v, w] : e[u])
          if (siz[v] < siz[u] && siz[v] > n) {
            u = v;
            flg = false;
            break;
          }
        if (flg) break;
      }
      siz[u] = -1;
      init(u, u, u, 0);
      sort(ch[u].begin(), ch[u].end());
      for (auto [v, w] : e[u])
        if (~siz[v]) v = split(v);
      return u;
    }
    int rt;
    void ctrdecomp() {
      int n = e.size();
      siz.assign(n, 0);
      fa.assign(n, {});
      ch.assign(n, {});
      rt = split(1);
    }
    vector<int> solve() {
      vector<int> dis(e.size(), -1);
      dis[1] = 0;
      queue<int> que;
      que.push(1);
      while (que.size()) {
        int u = que.front();
        que.pop();
        for (auto [f, d] : fa[u]) {
          while (ch[f].size()) {
            auto [w, v] = ch[f].back();
            if (d > w) break;
            ch[f].pop_back();
            if (~dis[v]) continue;
            que.push(v);
            dis[v] = dis[u] + 1;
          }
        }
      }
      return dis;
    }
};
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << setprecision(15);
  int tt;
  cin >> tt;
  while (tt--) {
    int n;
    cin >> n;
    graph g(n);
    for (int i = 2; i <= n; i++) cin >> g.r[i];
    for (int i = 1; i < n; i++) {
      ll u, v, w;
      cin >> u >> v >> w;
      g.adde(u, v, w);
    }
    g.ctrdecomp();
    auto res = g.solve();
    for (int i = 2; i <= n; i++) cout << res[i] << ' ';
    cout << el;
```

18

```
98       }
99       return 0;
100   }
```

3.5   强连通分量分解

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   struct graph {
4     vector<vector<int>> e, r;
5     graph(int n) {
6       e.resize(n + 1);
7       r.resize(n + 1);
8     }
9     void adde(int u, int v) {
10      e[u].push_back(v);
11      r[v].push_back(u);
12    }
13    vector<bool> vis;
14    vector<int> ord;
15    vector<int> col;
16    void travel(int u, int f) {
17      vis[u] = true;
18      for (auto v : e[u])
19        if (!vis[v]) travel(v, u);
20      ord.push_back(u);
21    }
22    void color(int u, int f, int c) {
23      col[u] = c;
24      for (auto v : r[u])
25        if (!col[v]) color(v, u, c);
26    }
27    vector<int> decomp() {
28      vis.assign(e.size(), false);
29      ord.clear();
30      for (int i = 1; i < e.size(); i++)
31        if (!vis[i]) travel(i, i);
32      reverse(ord.begin(), ord.end());
33      col.assign(e.size(), 0);
34      for (auto v : ord)
35        if (!col[v]) color(v, v, v);
36      return col;
37    }
38  };
```

3.6   拓扑排序

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   struct graph {
4     vector<vector<int>> e;
5     graph(int n) : e(n + 1) {}
6     void adde(int u, int v) { e[u].push_back(v); }
7     optional<vector<int>> toposort() {
8       int n = e.size() - 1;
9       vector<int> d(n + 1);
10      queue<int> que;
11      for (int u = 1; u <= n; u++)
12        for (auto v : e[u]) d[v]++;
13      for (int i = 1; i <= n; i++)
14        if (!d[i]) que.push(i);
15      vector<int> res;
16      while (!que.empty()) {
17        auto u = que.front();
18        res.push_back(u);
19        que.pop();
20        for (auto v : e[u]) {
21          if (!--d[v]) que.push(v);
22        }
23      }
24      if (res.size() != n) return nullopt;
25      return res;
26    }
27  };
```

3.7   二分图匹配

```
1   #pragma once
2
3   #include <vector>
4
5   #include "maxflow.h"
6
7   namespace match {
8
9   std::vector<std::pair<int, int>> max_match(
```

```cpp
      const std::vector<std::pair<int, int>> &edges) {
  int n = 1, m = 1;
  for (auto [l, r] : edges) {
    n = std::max(n, l);
    m = std::max(m, r);
  }
  std::vector<maxflow::FlowEdge> fedge;
  int s = n + m + 1, t = n + m + 2;
  for (int i = 1; i <= n; i++) fedge.push_back({.from = s, .to = i, .cap = 1});
  for (int i = 1; i <= m; i++)
    fedge.push_back({.from = i + n, .to = t, .cap = 1});
  for (auto [l, r] : edges) fedge.push_back({.from = l, .to = r + n, .cap = 1});
  maxflow::maximum_flow(fedge, s, t);
  std::vector<std::pair<int, int>> res;
  for (auto e : fedge)
    if (e.from != s && e.to != t && e.flow == 1)
      res.push_back({e.from, e.to - n});
  return res;
}

void dfs(int u, std::vector<bool> &vl, std::vector<bool> &vr,
         std::vector<std::vector<int>> &ltr, std::vector<int> &rtl) {
  if (vl[u]) return;
  vl[u] = true;
  for (auto v : ltr[u]) {
    vr[v] = true;
    dfs(rtl[v], vl, vr, ltr, rtl);
  }
}

std::pair<std::vector<int>, std::vector<int>> min_cover(
    const std::vector<std::pair<int, int>> &edges) {
  int n = 1, m = 1;
  for (auto [l, r] : edges) {
    n = std::max(n, l);
    m = std::max(m, r);
  }
  auto match = max_match(edges);
  std::vector<std::vector<int>> ltr(n + 1);
  std::vector<int> rtl(m + 1);
  std::vector<bool> vis(n + 1);
  for (auto [l, r] : match) {
    rtl[r] = l;
    vis[l] = true;
  }
```

```cpp
  for (auto [l, r] : edges) ltr[l].push_back(r);
  std::vector<bool> vl(n + 1), vr(m + 1);
  for (int i = 1; i <= n; i++)
    if (!vis[i]) dfs(i, vl, vr, ltr, rtl);
  std::pair<std::vector<int>, std::vector<int>> res;
  for (int i = 1; i <= n; i++)
    if (!vl[i]) res.first.push_back(i);
  for (int i = 1; i <= m; i++)
    if (vr[i]) res.second.push_back(i);
  return res;
}


} // namespace match
```

3.8 最大流

```cpp
#pragma once

#include <algorithm>
#include <optional>
#include <queue>
#include <vector>

namespace maxflow {

typedef long long flow_t;
const flow_t inf_flow = 1e18;
const int inf_dep = 1e9;

struct FlowEdge {
  int from, to;
  flow_t cap, low = 0, flow = 0;
};

int num_node(const std::vector<FlowEdge> &edges) {
  int n = 0;
  for (const auto &e : edges) n = std::max({n, e.from, e.to});
  return n;
}

flow_t get_flow(const std::vector<FlowEdge> &edges, int s) {
  flow_t flow = 0;
  for (const auto &e : edges) {
    if (e.from == s) flow += e.flow;
```

```
29  |     }
30  |     return flow;
31  | }
32  |
33  | struct MaxFlow {
34  |   struct Edge {
35  |     int from, to;
36  |     flow_t cap;
37  |   };
38  |   int n;
39  |   std::vector<std::vector<int>> eid;
40  |   std::vector<Edge> edge;
41  |   void build(const std::vector<FlowEdge> &edges) {
42  |     n = num_node(edges);
43  |     eid.assign(n + 1, {});
44  |     edge.clear();
45  |     int num_edges = 0;
46  |     for (const auto &e : edges) {
47  |       eid[e.from].push_back(num_edges++);
48  |       edge.push_back({e.from, e.to, e.cap - e.flow});
49  |       eid[e.to].push_back(num_edges++);
50  |       edge.push_back({e.to, e.from, e.flow});
51  |     }
52  |   }
53  |
54  |   std::vector<int> dis;
55  |   std::vector<int> cur;
56  |   bool bfs(int s, int t) {
57  |     if (s > n || t > n) return false;
58  |     dis.assign(n + 1, inf_dep);
59  |     cur.assign(n + 1, 0);
60  |     std::queue<int> que;
61  |     dis[s] = 0;
62  |     que.push(s);
63  |     while (que.size()) {
64  |       int u = que.front();
65  |       que.pop();
66  |       for (auto i : eid[u]) {
67  |         const auto &e = edge[i];
68  |         if (e.cap && dis[e.to] > dis[u] + 1) {
69  |           dis[e.to] = dis[u] + 1;
70  |           que.push(e.to);
71  |         }
72  |       }
73  |     }
74  |     return dis[t] < inf_dep;
75  |   }
76  |
77  |   flow_t dfs(int s, int t, flow_t flim) {
78  |     if (s == t) return flim;
79  |     flow_t flow = 0;
80  |     for (int &i = cur[s]; i < eid[s].size() && flow < flim; i++) {
81  |       auto &e = edge[eid[s][i]];
82  |       if (dis[e.to] == dis[s] + 1 && e.cap) {
83  |         auto detf = dfs(e.to, t, std::min(flim - flow, e.cap));
84  |         flow += detf;
85  |         e.cap -= detf;
86  |         edge[eid[s][i] ^ 1].cap += detf;
87  |       }
88  |       if (flow == flim) break;
89  |     }
90  |     return flow;
91  |   }
92  |   flow_t maxflow(int s, int t) {
93  |     flow_t flow = 0;
94  |     while (bfs(s, t)) {
95  |       flow += dfs(s, t, inf_flow);
96  |     }
97  |     return flow;
98  |   }
99  |
100 |   std::vector<FlowEdge> to_edge() {
101 |     std::vector<FlowEdge> edges;
102 |     for (int i = 0; i < edge.size(); i += 2)
103 |       edges.push_back({
104 |           .from = edge[i].from,
105 |           .to = edge[i].to,
106 |           .cap = edge[i].cap + edge[i ^ 1].cap,
107 |           .low = 0,
108 |           .flow = edge[i ^ 1].cap,
109 |       });
110 |     return edges;
111 |   }
112 | };
113 |
114 | struct Processor {
115 |   std::vector<bool> neg;
116 |   std::vector<flow_t> low;
117 |   std::vector<flow_t> excess;
118 |   void init(std::vector<FlowEdge> &edges) {
```

21

```
119        int n = num_node(edges);                              164      for (int i = m; i < edges.size(); i++)
120        neg.clear();                                          165        if (edges[i].flow != edges[i].cap) return false;
121        neg.reserve(edges.size());                            166      edges.resize(m);
122        low.clear();                                          167      return true;
123        low.reserve(edges.size());                            168    }
124        excess.assign(n + 1, 0);                              169
125      }                                                       170    std::optional<flow_t> feasible_flow(std::vector<FlowEdge> &edges, int s = 0,
126      void rmv_low(std::vector<FlowEdge> &edges) {            171                                            int t = 0) {
127        for (auto &e : edges) {                               172      if (s && t) edges.push_back({.from = t, .to = s, .cap = inf_flow});
128          low.push_back(e.low);                               173      Processor p;
129          if (e.flow >= e.low) {                              174      p.init(edges);
130            e.flow -= e.low;                                  175      p.rmv_low(edges);
131          } else {                                            176      if (!excess_flow(edges, p.excess)) return std::nullopt;
132            excess[e.from] -= e.low - e.flow;                 177      if (s && t) edges.pop_back();
133            excess[e.to] += e.low - e.flow;                   178      p.add_low(edges);
134            e.flow = 0;                                       179      return get_flow(edges, s);
135          }                                                   180    }
136          e.cap -= e.low;                                     181
137          e.low = 0;                                          182    std::optional<flow_t> maximum_flow(std::vector<FlowEdge> &edges, int s, int t) {
138        }                                                     183      edges.push_back({.from = t, .to = s, .cap = inf_flow});
139      }                                                       184      Processor p;
140      void add_low(std::vector<FlowEdge> &edges) {            185      p.init(edges);
141        reverse(low.begin(), low.end());                      186      p.rmv_low(edges);
142        for (auto &e : edges) {                               187      if (!excess_flow(edges, p.excess)) return std::nullopt;
143          e.low = low.back();                                 188      edges.pop_back();
144          e.flow += e.low;                                    189      MaxFlow g;
145          e.cap += e.low;                                     190      g.build(edges);
146          low.pop_back();                                     191      g.maxflow(s, t);
147        }                                                     192      edges = g.to_edge();
148      }                                                       193      p.add_low(edges);
149    };                                                        194      return get_flow(edges, s);
150                                                              195    }
151    bool excess_flow(std::vector<FlowEdge> &edges,            196
152                     const std::vector<flow_t> &excess) {     197    std::optional<flow_t> minimum_flow(std::vector<FlowEdge> &edges, int s, int t) {
153      int n = num_node(edges), m = edges.size();              198      edges.push_back({.from = t, .to = s, .cap = inf_flow});
154      for (int i = 1; i <= n; i++) {                          199      Processor p;
155        if (excess[i] > 0)                                    200      p.init(edges);
156          edges.push_back({.from = n + 1, .to = i, .cap = excess[i]});  201      p.rmv_low(edges);
157        if (excess[i] < 0)                                    202      if (!excess_flow(edges, p.excess)) return std::nullopt;
158          edges.push_back({.from = i, .to = n + 2, .cap = -excess[i]});  203      edges.pop_back();
159      }                                                       204      MaxFlow g;
160      MaxFlow g;                                              205      Processor q;
161      g.build(edges);                                         206      excess_flow(edges, q.excess);
162      g.maxflow(n + 1, n + 2);                                207      g.build(edges);
163      edges = g.to_edge();                                    208      g.maxflow(t, s);
```

```
209    edges = g.to_edge();
210    p.add_low(edges);
211    return get_flow(edges, s);
212  }
213
214  }  // namespace maxflow
```

3.9 换根 DP

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  struct graph {
5    vector<vector<int>> e;
6    ll k, c;
7    graph(int n) { e.resize(n + 1); }
8    void adde(int u, int v) {
9      e[u].push_back(v);
10     e[v].push_back(u);
11   }
12   vector<int> dis, dep;
13   ll ans;
14   void setup(int u, int f) {
15     dep[u] = 0;
16     for (auto v : e[u])
17       if (v != f) {
18         dis[v] = dis[u] + 1;
19         setup(v, u);
20         dep[u] = max(dep[u], dep[v] + 1);
21       }
22   }
23   void reroot(int u, int f) {
24     int mxm1 = -1, mxm2 = -1;
25     for (auto v : e[u]) {
26       mxm2 = max(mxm2, dep[v]);
27       if (mxm2 > mxm1) swap(mxm1, mxm2);
28     }
29     dep[u] = mxm1 + 1;
30     ans = max(ans, dep[u] * k - dis[u] * c);
31     for (auto v : e[u])
32       if (v != f) {
33         dep[u] = (dep[v] == mxm1 ? mxm2 : mxm1) + 1;
34         reroot(v, u);
35       }
```

```
36   }
37   ll solve(int s) {
38     dis.resize(e.size());
39     dep.resize(e.size());
40     dep[s] = 0;
41     setup(s, s);
42     ans = 0;
43     reroot(s, s);
44     return ans;
45   }
46 };
```

3.10 树哈希

```
1  /**
2   * Author: Boboge adapted from peehs_moorhsum
3   * Date: 23-02-06
4   * Description: return the hash value of every subtree.
5   * Time: O(N).
6   * Status: tested on https://uoj.ac/problem/763
7   */
8  std::mt19937_64 rnd(std::chrono::steady_clock::now().time_since_epoch().count());
9
10 struct treeHash {
11     using ull = unsigned long long;
12     ull bas = rnd();
13
14     ull H(ull x) {
15         return x * x * x * 114514 + 19260817;
16     }
17
18     ull F(ull x) {
19         return H(x & ((1ll << 32) - 1)) + H(x >> 32);
20     }
21
22     std::vector<ull> h;
23
24     treeHash(std::vector<std::vector<int>> &adj, int rt = 0) : h(adj.size()) {
25         auto dfs = [&](auto dfs, int u, int fa) -> void {
26             h[u] = bas;
27             for (int v: adj[u]) {
28                 if (v == fa) continue;
29                 dfs(dfs, v, u);
30                 h[u] += F(h[v]);
```

```
31               }
32          };
33          dfs(dfs, rt, -1);
34      }
35  };
```

4 数学

4.1 数论分块

```
1  for (ll l = 1; l <= n;) {
2    ll r = n / (n / l);
3    ...  // for l<=i<=r, n/i = n/l
4    l = r + 1;
5  }
```

4.2 FWT

```
1
2  const int k = 17, n = 1 << k;
3  const int mod = 1e9 + 7, inv2 = mod / 2 + 1;
4  struct modint {
5    int n;
6    modint(int n = 0) : n(n) {}
7  };
8  modint operator+(modint a, modint b) {
9    return (a.n += b.n) >= mod ? a.n - mod : a.n;
10 }
11 modint operator-(modint a, modint b) {
12   return (a.n -= b.n) < 0 ? a.n + mod : a.n;
13 }
14 modint operator*(modint a, modint b) { return 1ll * a.n * b.n % mod; }
15 vector<modint> fib(n);
16 vector<modint> fwt(vector<modint> a, void (*opr)(modint &, modint &)) {
17   // cout << 1 << el;
18   int n = a.size();
19   for (int i = 0; 1 << i < n; i++)
20     for (int j = 0; j < n; j++)
21       if (j >> i & 1) opr(a[j - (1 << i)], a[j]);
22   // cout << 2 << el;
23   return a;
24 }
```

```
25 void fwtand(modint &a, modint &b) { a = a + b; }
26 void revand(modint &a, modint &b) { a = a - b; }
27 void fwtor(modint &a, modint &b) { b = b + a; }
28 void revor(modint &a, modint &b) { b = b - a; }
29 void fwtxor(modint &a, modint &b) {
30   modint x = a + b, y = a - b;
31   a = x, b = y;
32 }
33 void revxor(modint &a, modint &b) {
34   modint x = a + b, y = a - b;
35   a = x * inv2, b = y * inv2;
36 }
37 vector<modint> add(vector<modint> a, vector<modint> b) {
38   for (int i = 0; i < a.size(); i++) a[i] = a[i] + b[i];
39   return a;
40 }
41 vector<modint> mul(vector<modint> a, vector<modint> b) {
42   for (int i = 0; i < a.size(); i++) a[i] = a[i] * b[i];
43   return a;
44 }
45 vector<int> popc(n);
46 vector<modint> filter(vector<modint> a, int k) {
47   for (int i = 0; i < n; i++)
48     if (popc[i] != k) a[i] = 0;
49   return a;
50 }
51 int main() {
52   init();
53   fib[0] = 0, fib[1] = 1;
54   for (int i = 2; i < n; i++) fib[i] = fib[i - 1] + fib[i - 2];
55   popc[0] = 0;
56   for (int i = 1; i < n; i++) popc[i] = popc[i >> 1] + (i & 1);
57   int m;
58   cin >> m;
59   vector<int> s(m);
60   for (auto &v : s) cin >> v;
61   vector<modint> a(n);
62   for (auto v : s) a[v] = a[v] + 1;
63   auto mid = mul(a, fib),
64        rhs = mul(fwt(mul(fwt(a, fwtxor), fwt(a, fwtxor)), revxor), fib);
65   vector b(k + 1, vector<modint>());
66   vector c(k + 1, vector<modint>(n));
67   vector<modint> lhs(n);
68   for (int i = 0; i <= k; i++) b[i] = fwt(filter(a, i), fwtor);
69   for (int i = 0; i <= k; i++)
```

```
70        for (int j = 0; i + j <= k; j++) c[i + j] = add(c[i + j], mul(b[i], b[j]));
71     for (int i = 0; i <= k; i++)
72       lhs = add(lhs, filter(mul(fwt(c[i], revor), fib), i));
73     auto res = fwt(mul(mul(fwt(lhs, fwtand), fwt(mid, fwtand)), fwt(rhs, fwtand)),
74                     revand);
75     modint ans;
76     for (int i = 0; i < k; i++) ans = ans + res[1 << i];
77     cout << ans.n << el;
78     return 0;
79   }
```

4.3　高斯消元

```
1
2    const ll mod = 1e9 + 7, pri = 5;
3    const array<int, pri> inv = {0, 1, 3, 2, 4};
4    ll qpow(ll a, ll b, ll m = mod) {
5      ll res = 1;
6      while (b) {
7        if (b & 1) res = res * a % m;
8        a = a * a % m, b >>= 1;
9      }
10     return res;
11   }
12   vector<int> read() {
13     string s;
14     cin >> s;
15     vector<int> a(s.size());
16     for (int i = 0; i < s.size(); i++) a[i] = s[i] - 'a';
17     return a;
18   }
19   void flip(vector<vector<int>> &a) {
20     int n = a.size(), m = a[0].size();
21     vector b(m, vector<int>(n));
22     for (int i = 0; i < n; i++)
23       for (int j = 0; j < m; j++) b[j][i] = a[i][j];
24     swap(a, b);
25   }
26   int gaussian(vector<vector<int>> &a) {
27     int n = a.size(), m = a[0].size();
28     int r = 0;
29     for (int j = 0; j < m; j++) {
30       int p = -1;
31       for (int i = r; i < n; i++)
```

```
32         if (a[i][j]) p = i;
33       if (p == -1) continue;
34       swap(a[r], a[p]);
35       int t = inv[a[r][j]];
36       for (int k = 0; k < m; k++) a[r][k] = a[r][k] * t % pri;
37       for (int i = 0; i < n; i++)
38         if (i != r) {
39           int t = a[i][j];
40           for (int k = 0; k < m; k++)
41             a[i][k] = ((a[i][k] - a[r][k] * t) % pri + pri) % pri;
42         }
43       r++;
44     }
45     a.resize(r);
46     return r;
47   }
48   bool solve(vector<vector<int>> &a, vector<int> &b) {
49     int n = a.size(), m = a[0].size();
50     for (int i = 0; i < n; i++) {
51       int p = 0;
52       while (!a[i][p]) p++;
53       int t = b[p];
54       for (int j = 0; j < m; j++) b[j] = ((b[j] - a[i][j] * t) % pri + pri) % pri;
55     }
56     for (auto v : b)
57       if (v % pri) return false;
58     return true;
59   }
```

4.4　Min25 筛

```
1    const int det = 100;
2    const ll mod = 1e9 + 7;
3    template <typename T>
4    struct min25 {
5      ll n, m;
6      vector<ll> p;
7      vector<T> s;
8      void sieve(ll n) {
9        p = {1}, s = {T()};
10       vector<int> minf(n);
11       for (ll i = 2; i < n; i++) {
12         if (!minf[i]) {
13           minf[i] = p.size();
```

```
14          p.push_back(i);
15          s.push_back(s.back() + T::init(i));
16        }
17        for (ll j = 1; j <= minf[i] && i * p[j] < n; j++) minf[i * p[j]] = j;
18      }
19    }
20    vector<ll> lis;
21    vector<int> le, ge;
22    vector<T> g;
23    void init(ll n) {
24      le.resize(m + det), ge.resize(m + det);
25      for (ll i = 1, j; i <= n; i = n / j + 1) {
26        j = n / i;
27        int k = lis.size();
28        lis.push_back(j);
29        (j <= m ? le[j] : ge[n / j]) = k;
30        g.push_back(T::plug(j));
31      }
32    }
33    int id(ll v) { return v <= m ? le[v] : ge[n / v]; }
34    vector<ll> f;
35    void calcp() {
36      for (int k = 1; k < p.size(); k++) {
37        ll pk = p[k], sp = pk * pk;
38        for (int i = 0; i < lis.size() && lis[i] >= sp; i++) {
39          int j = id(lis[i] / pk);
40          g[i] = g[i] - (g[j] - s[k - 1]) * T::item(pk);
41        }
42      }
43      for (auto &v : g) f.push_back(v.val());
44    }
45    min25(ll n) : n(n) {
46      m = sqrt(n);
47      sieve(m + det);
48      init(n);
49      calcp();
50    }
51    ll query(ll n, int k = 1) {
52      if (n < p[k] || n <= 1) return 0;
53      const int i = id(n);
54      ll ans = f[i] - s[k - 1].val();
55      for (int i = k; i < p.size() && 1ll * p[i] * p[i] <= n; i++) {
56        ll mul = p[i];
57        for (int c = 1; mul * p[i] <= n; c++, mul *= p[i])
58          ans += T::f(p[i], c) * query(n / mul, i + 1) % mod + T::f(p[i], c + 1);
```

```
59      }
60      return ans % mod;
61    }
62  };
63  struct ez {
64    ll a, b;
65    ez(ll a = 0, ll b = 0) : a(a % mod), b(b % mod) {}
66    ez operator+(ez r) { return {(a + r.a) % mod, (b + r.b) % mod}; }
67    ez operator-(ez r) { return {(a - r.a) % mod, (b - r.b) % mod}; }
68    ez operator*(ez r) { return {a * r.a % mod, b * r.b % mod}; }
69    ll val() { return (a + b) % mod; }
70    static ll f(ll p, ll c) { return p ^ c; }
71    static ez init(ll p) { return {-1, p}; }
72    static ez plug(ll p) {
73      p %= mod;
74      return {-p + 1, (p + 2) * (p - 1) % mod * (mod + 1 >> 1) % mod};
75    }
76    static ez item(ll p) { return {1, p}; }
77  };
```

4.5 Pollard—ρ

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   using i64 = long long;
4   using i128 = __int128;
5   i64 fpow(i64 a, i64 t, i64 mod){
6           i64 r = 1;
7           for (; t; t >>= 1, a = (i128)a * a % mod) {
8                   if (t & 1) {
9                           r = (i128)r * a % mod;
10                  }
11          }
12          return r;
13  }
14  i64 gcd(i64 a, i64 b){
15          #define ctz __builtin_ctzll
16          int shift = ctz(a | b);
17          b >>= ctz(b);
18          while (a) {
19                  a >>= ctz(a);
20                  if (a < b) swap(a, b);
21                  a -= b;
22          }
```

```
23          return b << shift;
24  }
25  bool check_prime(i64 n){
26          static const int jp[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
27          if (n == 1) return false;
28          for (int p : jp) if (n % p == 0) return n == p;
29          i64 r = n - 1, x, y;
30          int e = 0;
31          while (~r & 1) r >>= 1, ++e;
32          for (int p : jp) {
33                  x = fpow(p, r, n);
34                  for (int t = 0; t < e && x > 1; ++t) {
35                          y = (i128)x * x % n;
36                          if (y == 1 && x != n - 1) return false;
37                          x = y;
38                  }
39                  if (x != 1) return false;
40          }
41          return true;
42  }
43  i64 find(i64 n){
44          static const int step = 1 << 7;
45          i64 x, y = rand() % n;
46          int c = rand() % n;
47          auto f = [=](i64 x){ return ((i128)x * x + c) % n; } ;
48          for (int l = 1; ; l <<= 1) {
49                  x = y;
50                  for (int i = 0; i < l; ++i) y = f(y);
51                  for (int k = 0; k < l; k += step) {
52                          int e = std::min(step, l - k);
53                          i64 g = 1, z = y;
54                          for (int i = 0; i < e; ++i) g = (i128)g * ((y = f(y)) + n - x) % n;
55                          g = gcd(g, n);
56                          if (g == 1) continue;
57                          if (g == n) for (g = 1, y = z; g == 1; ) y = f(y), g = gcd(y + n - x, n);
58                          return g;
59                  }
60          } //
61  }
62  void rho(i64 n, map<i64,int> &factor){
63          while (~n & 1) n >>= 1, ++factor[2];
64          if (n == 1) return ;
65          if (check_prime(n)) {
66                  ++factor[n];
67                  return ;
```

```
68          }
69          i64 d;
70          for (d = find(n); d == n; d = find(d));
71          rho(d, factor), rho(n / d, factor);
72  }
73  int T;
74  i64 n;
75  int main(){
76          for (cin >> T; T; --T) {
77                  map<i64, int> f;
78                  cin >> n;
79                  rho(n, f);
80                  if (f.size() > 1 || (--f.end())->second > 1) {
81                          cout << (--f.end())->first << '\n';
82                  } else {
83                          cout << "Prime\n";
84                  }
85          }
86  }
```

4.6 多项式

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   typedef long long ll;
4   const ll mod = 998244353;
5   typedef double ld;
6   typedef complex<ld> cplx;
7   typedef vector<ll> poly;
8
9   ll qpow(ll a, ll b) {
10      ll res = 1;
11      while (b) {
12          if (b & 1) res = res * a % mod;
13          a = a * a % mod;
14          b >>= 1;
15      }
16      return res;
17  }
18  ll inv(ll n) { return qpow(n, mod - 2); }
19
20  const auto pi = acosl(-1);
21  const int len = 15, mask = (1 << len) - 1;
22
```

27

```
23 | struct unitroot {
24 |   static vector<cplx> w;
25 |   static vector<cplx> get_root(int n) {
26 |     n = 1 << 32 - __builtin_clz(n);
27 |     if (n > w.size()) {
28 |       w.resize(n);
29 |       for (int i = 0; i < n; i++)
30 |         w[i] = cplx(cos(2 * i * pi / n), sin(2 * i * pi / n));
31 |     }
32 |     int m = w.size() / n;
33 |     vector<cplx> res(n);
34 |     for (int i = 0, j = 0; i < n; i++, j += m) res[i] = w[j];
35 |     return res;
36 |   }
37 | };
38 | vector<cplx> unitroot::w;
39 |
40 | void fft(vector<cplx> &p, const vector<cplx> &w) {
41 |   int n = w.size();
42 |   for (int i = 1, j = 0; i < n - 1; ++i) {
43 |     int s = n;
44 |     do {
45 |       s >>= 1;
46 |       j ^= s;
47 |     } while (~j & s);
48 |     if (i < j) swap(p[i], p[j]);
49 |   }
50 |   for (int d = 0; (1 << d) < n; ++d) {
51 |     int m = 1 << d, m2 = m * 2, rm = n >> (d + 1);
52 |     for (int i = 0; i < n; i += m2)
53 |       for (int j = 0; j < m; ++j) {
54 |         auto &p1 = p[i + j + m], &p2 = p[i + j];
55 |         auto t = w[rm * j] * p1;
56 |         p1 = p2 - t;
57 |         p2 = p2 + t;
58 |       }
59 |   }
60 | }
61 | poly operator+(const poly &a, const poly &b) {
62 |   poly c(max(a.size(), b.size()));
63 |   for (int i = 0; i < a.size(); i++) c[i] += a[i];
64 |   for (int i = 0; i < b.size(); i++) c[i] += b[i];
65 |   for (auto &v : c) v %= mod;
66 |   return c;
67 | }
```

```
68  | poly operator-(poly b) {
69  |   for (auto &v : b) v = v ? mod - v : 0;
70  |   return b;
71  | }
72  | poly operator-(const poly &a, const poly &b) { return a + -b; }
73  | poly operator*(const poly &a, const poly &b) {
74  |   vector<cplx> w = unitroot::get_root(a.size() + b.size() - 1);
75  |   int n = w.size();
76  |   vector<cplx> A(n), B(n), C(n), D(n);
77  |   for (int i = 0; i < a.size(); ++i) A[i] = cplx(a[i] >> len, a[i] & mask);
78  |   for (int i = 0; i < b.size(); ++i) B[i] = cplx(b[i] >> len, b[i] & mask);
79  |   fft(A, w), fft(B, w);
80  |   for (int i = 0; i < n; ++i) {
81  |     int j = (n - i) % n;
82  |     cplx da = (A[i] - conj(A[j])) * cplx(0, -0.5),
83  |          db = (A[i] + conj(A[j])) * cplx(0.5, 0),
84  |          dc = (B[i] - conj(B[j])) * cplx(0, -0.5),
85  |          dd = (B[i] + conj(B[j])) * cplx(0.5, 0);
86  |     C[j] = da * dd + da * dc * cplx(0, 1);
87  |     D[j] = db * dd + db * dc * cplx(0, 1);
88  |   }
89  |   fft(C, w), fft(D, w);
90  |   poly res(a.size() + b.size() - 1);
91  |   for (int i = 0; i < res.size(); ++i) {
92  |     ll da = (ll)(C[i].imag() / n + 0.5) % mod,
93  |        db = (ll)(C[i].real() / n + 0.5) % mod,
94  |        dc = (ll)(D[i].imag() / n + 0.5) % mod,
95  |        dd = (ll)(D[i].real() / n + 0.5) % mod;
96  |     res[i] = ((dd << (len * 2)) + ((db + dc) << len) + da) % mod;
97  |   }
98  |   return res;
99  | }
100 | poly inv(poly a) {
101 |   int n = a.size();
102 |   if (a.size() == 1) return {inv(a[0])};
103 |   poly b = inv({a.begin(), a.end() - n / 2});
104 |   auto c = a * b;
105 |   c.resize(n);
106 |   a = b * (poly{2} - c);
107 |   a.resize(n);
108 |   return a;
109 | }
110 | poly operator/(poly a, poly b) {
111 |   int n = a.size() + b.size() - 1;
112 |   b.resize(n);
```

28

```
113     a = a * inv(b);
114     a.resize(n);
115     return a;
116   }
```

4.7 GCD

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   unsigned long long gcd(unsigned long long a, unsigned long long b) {
5     int shift = __builtin_ctzll(a | b);
6     b >>= __builtin_ctzll(b);
7     while (a) {
8       a >>= __builtin_ctzll(a);
9       if (a < b) swap(a, b);
10      a -= b;
11    }
12    return b << shift;
13  }
```

5 字符串

5.1 哈希

```
1   #include "bits/stdc++.h"
2   using namespace std;
3   const char el = '\n';
4   typedef long long ll;
5   const ll mod1 = 1e9 + 7, mod2 = 1e9 + 9;
6   struct graph {
7     vector<vector<pair<int, int>>> e;
8     graph(int n) : e(n + 1) {}
9     void adde(int u, int v, int w) { e[u].push_back({v, w}); }
10    bool check() {
11      vector<int> deg(e.size());
12      for (auto &ed : e)
13        for (auto &[to, wt] : ed) deg[to]++;
14      for (int i = 1; i < e.size(); i++)
15        if (deg[i] != e[i].size()) return false;
16      return true;
17    }
```

```
18    void euler(int u, vector<int> &res) {
19      while (e[u].size()) {
20        auto [v, w] = e[u].back();
21        e[u].pop_back();
22        euler(v, res);
23        res.push_back(w);
24      }
25    }
26  };
27  struct hsh {
28    static const int k = 3;
29    array<ll, k> a;
30  };
31  const hsh zero{0, 0, 0}, one{1, 1, 1}, mul{31, 57, 71},
32      mod{998244353, 1e9 + 7, 1e9 + 9};
33
34  const int mul1 = 31, mul2 = 157;
35  pair<ll, ll> pmul(int n) {
36    static vector<pair<ll, ll>> res = {{1, 1}};
37    while (res.size() <= n) {
38      auto [p, q] = res.back();
39      res.push_back({p * mul1 % mod1, q * mul2 % mod2});
40    }
41    return res[n];
42  }
43  struct hshstr {
44    vector<pair<ll, ll>> hsh;
45    hshstr(string s = "") {
46      hsh.push_back({0, 0});
47      for (auto c : s) {
48        auto [p, q] = hsh.back();
49        hsh.push_back({(p * mul1 + c) % mod1, (q * mul2 + c) % mod2});
50      }
51    }
52    pair<ll, ll> hash(int l, int r) {
53      auto [p1, q1] = hsh[l];
54      auto [p2, q2] = hsh[r];
55      auto [mp, mq] = pmul(r - l);
56      return {(p1 * mp - p2 + mod1) % mod1, (q1 * mq - q2 + mod2) % mod2};
57    }
58  };
59  optional<pair<vector<int>, vector<int>>> solve(vector<hshstr> &a,
60                                                  vector<hshstr> &b, int n,
61                                                  int m) {
62    map<pair<ll, ll>, int> id1, id2;
```

```
63    int k = 0;
64    for (auto &s : a) {
65      pair<ll, ll> l = s.hash(0, m), r = s.hash(m, n);
66      if (!id1.count(l)) id1[l] = ++k;
67      if (!id2.count(r)) id2[r] = ++k;
68    }
69    for (auto &s : b) {
70      pair<ll, ll> l = s.hash(0, n - m), r = s.hash(n - m, n);
71      if (!id2.count(l)) return nullopt;
72      if (!id1.count(r)) return nullopt;
73    }
74    graph g(k);
75    int t = 0;
76    for (auto &s : a) {
77      pair<ll, ll> l = s.hash(0, m), r = s.hash(m, n);
78      g.adde(id1[l], id2[r], ++t);
79    }
80    t = 0;
81    for (auto &s : b) {
82      pair<ll, ll> l = s.hash(0, n - m), r = s.hash(n - m, n);
83      g.adde(id2[l], id1[r], ++t);
84    }
85    vector<int> res;
86    if (!g.check()) return nullopt;
87    g.euler(1, res);
88    if (res.size() < 2 * a.size()) return nullopt;
89    reverse(res.begin(), res.end());
90    vector<int> p, q;
91    for (int i = 0; i < res.size(); i += 2) p.push_back(res[i]);
92    for (int i = 1; i < res.size(); i += 2) q.push_back(res[i]);
93    return pair<vector<int>, vector<int>>{p, q};
94  }
95  int main() {
96    ios::sync_with_stdio(false);
97    cin.tie(0);
98    cout << setprecision(15);
99    int tt;
100   cin >> tt;
101   while (tt--) {
102     int n, m;
103     cin >> n >> m;
104     vector<hshstr> a(n), b(n);
105     for (auto &v : a) {
106       string s;
107       cin >> s;
```

```
108        v = hshstr(s);
109      }
110      for (auto &v : b) {
111        string s;
112        cin >> s;
113        v = hshstr(s);
114      }
115      bool flg = false;
116      for (int i = 0; i < m; i++)
117        if (auto res = solve(a, b, m, i)) {
118          flg = true;
119          for (auto v : res->first) cout << v << ' ';
120          cout << el;
121          for (auto v : res->second) cout << v << ' ';
122          cout << el;
123          break;
124        }
125      if (!flg) cout << -1 << el;
126    }
127    return 0;
128  }
```

5.2 KMP

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   vector<int> kmp(string s) {
5     vector<int> res = {-1};
6     for (auto c : s) {
7       int cur = res.back();
8       while (~cur && c != s[cur]) cur = res[cur];
9       res.push_back(cur + 1);
10    }
11    return res;
12  }
```

5.3 后缀自动机

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

```
4
5  const int alpha = 26;
6  struct sam {
7    struct node {
8      array<node *, alpha> next;
9      node *link;
10     int len;
11     int cnt;
12     node() {
13       fill(next.begin(), next.end(), (node *)NULL);
14       link = NULL;
15       len = 0;
16     }
17   };
18   node *nd, *end;
19   node *nnode() { return new (end++) node(); }
20   node *root, *last;
21   sam(int n) {
22     end = nd = (node *)new char[sizeof(node) * (n << 1)];
23     last = root = nnode();
24   }
25   void add(int c) {
26     node *cur = nnode();
27     cur->len = last->len + 1;
28     cur->cnt = 1;
29     node *p = last;
30     last = cur;
31     while (p && !p->next[c]) {
32       p->next[c] = cur;
33       p = p->link;
34     }
35     if (!p) {
36       cur->link = root;
37       return;
38     }
39     node *q = p->next[c];
40     if (q->len == p->len + 1) {
41       cur->link = q;
42       return;
43     }
44     node *clone = nnode();
45     clone->next = q->next;
46     clone->link = q->link;
47     clone->len = p->len + 1;
48     while (p && p->next[c] == q) {
```

```
49       p->next[c] = clone;
50       p = p->link;
51     }
52     cur->link = q->link = clone;
53   }
54   ll solve() {
55     vector<int> deg(end - nd);
56     for (auto u = nd + 1; u < end; u++) deg[u->link - nd]++;
57     queue<node *> que;
58     for (auto u = nd; u < end; u++)
59       if (!deg[u - nd]) que.push(u);
60     ll res = 0;
61     while (que.size()) {
62       auto u = que.front();
63       que.pop();
64       if (auto v = u->link) {
65         v->cnt += u->cnt;
66         if (!--deg[v - nd]) que.push(v);
67       }
68       if (u->cnt > 1) res = max(res, 1ll * u->cnt * u->len);
69     }
70     return res;
71   }
72 };
```

5.4 字典树

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const char el = '\n';
4  typedef long long ll;
5  const int alpha = 26;
6  struct node {
7    array<node *, alpha> ch;
8    int cnt;
9    node() : cnt(0) { ch.fill(NULL); }
10 };
11 void add(node *u, const string &s) {
12   for (auto c : s) {
13     c -= 'a';
14     if (!u->ch[c]) u->ch[c] = new node();
15     u = u->ch[c];
16     u->cnt++;
17   }
```

```
18  }
19  pair<node *, ll> run(node *u, const string &s) {
20    ll res = 0;
21    for (auto c : s) {
22      c -= 'a';
23      if (!u->ch[c]) return {NULL, res};
24      u = u->ch[c];
25      res += u->cnt;
26    }
27    return {u, res};
28  }
```

6   计算几何

6.1   三维计算几何

```
1   #include "bits/stdc++.h"
2   using namespace std;
3   const char el = '\n';
4   typedef long long ll;
5   typedef long double ld;
6   // typedef __int128 double;
7   struct cplx {
8     ld x, y, z;
9     ld abs() { return sqrt(1.0 * x * x + 1.0 * y * y + 1.0 * z * z); }
10  };
11  cplx operator+(cplx a, cplx b) { return {a.x + b.x, a.y + b.y, a.z + b.z}; }
12  cplx operator-(cplx a, cplx b) { return {a.x - b.x, a.y - b.y, a.z - b.z}; }
13  cplx operator*(cplx a, ld b) { return {a.x * b, a.y * b, a.z * b}; }
14  cplx operator*(ld b, cplx a) { return {a.x * b, a.y * b, a.z * b}; }
15  cplx det(cplx a, cplx b) {
16    return {a.y * b.z - b.y * a.z, a.z * b.x - b.z * a.x, a.x * b.y - b.x * a.y};
17  }
18  ld dot(cplx a, cplx b) { return a.x * b.x + a.y * b.y + a.z * b.z; }
19  const ld inf = 1e30, eps = 1e-8;
20  ld dist(array<cplx, 3> a, cplx b) {
21    cplx h = det(a[0] - a[1], a[0] - a[2]);
22    ld dt = dot(b, h);
23    if (dt < eps) return inf;
24    ld cp = dot(a[0], h);
25    cplx o = cp / dt * b;
26    ld sa = det(a[0] - o, a[1] - o).abs() + det(a[1] - o, a[2] - o).abs() +
27            det(a[2] - o, a[0] - o).abs(),
```

```
28          sb = h.abs();
29    if (sa / sb > 1 + eps) return inf;
30    return o.abs();
31  }
32  int main() {
33    ios::sync_with_stdio(false);
34    cin.tie(0);
35    cout << setprecision(3);
36    int n, m;
37    cin >> n >> m;
38    vector a(n, array<cplx, 3>());
39    for (auto &t : a) {
40      for (auto &[x, y, z] : t) cin >> x >> y >> z;
41      if (dot(det(t[0] - t[1], t[0] - t[2]), t[0]) < 0) swap(t[1], t[2]);
42    }
43    while (m--) {
44      cplx b;
45      cin >> b.x >> b.y >> b.z;
46      int p = -1;
47      ld val = inf / 2;
48      for (int i = 0; i < n; i++) {
49        auto tmp = dist(a[i], b);
50        if (tmp < val) {
51          p = i, val = tmp;
52        }
53      }
54      cout << p + 1 << el;
55    }
56    return 0;
57  }
```

6.2   圆凸包

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   const char el = '\n';
4   typedef long long ll;
5   typedef double ld;
6   const ld pi = acosl(-1);
7   const ld eps = 1e-9;
8   struct func {
9     ll x, y, z;
10    ld val(ld t) { return x * cos(t) + y * sin(t) + z; }
11    pair<ld, ld> zero() {
```

```
12        ld l = sqrt(x * x + y * y);
13        if (l < abs(z) - eps) return {0, 2 * pi};
14        ld a = atan2(y, x) + 2 * pi, b = acos(-z / l);
15        ld p = a - b, q = a + b;
16        while (p > 2 * pi) p -= 2 * pi;
17        while (q > 2 * pi) q -= 2 * pi;
18        if (p > q) swap(p, q);
19        return {p, q};
20    }
21  };
22  bool operator==(func a, func b) {
23    return a.x == b.x && a.y == b.y && a.z == b.z;
24  }
25  func operator+(func a, func b) { return {a.x + b.x, a.y + b.y, a.z + b.z}; }
26  func operator-(func a, func b) { return {a.x - b.x, a.y - b.y, a.z - b.z}; }
27  struct pcef {
28    vector<ld> a;
29    vector<func> f;
30    ld val(ld x) {
31      int p = upper_bound(a.begin(), a.end(), x - eps) - a.begin();
32      p = max(min(p - 1, (int)f.size() - 1), 0);
33      return f[p].val(x);
34    }
35    ld fnc(ld x) {
36      return max(val(x) + val(x + pi), val(x + pi / 2) + val(x + pi * 3 / 2));
37    }
38    ld mxm(ld l, ld r) {
39      ld res = min(fnc(l), fnc(r));
40      while (r - l > eps) {
41        ld m1 = (l + r) / 2, m2 = (m1 + r) / 2;
42        ld v1 = fnc(m1), v2 = fnc(m2);
43        res = min({res, v1, v2});
44        if (v1 < v2)
45          r = m2;
46        else
47          l = m1;
48      }
49      return res;
50    }
51  };
52  pcef operator+(pcef a, pcef b) {
53    pcef c = {{0}, {}};
54    int p = 0, q = 0;
55    ld xl = 0;
56    while (p < a.f.size() && q < b.f.size()) {
```

```
57        ld xr = min(a.a[p + 1], b.a[q + 1]);
58        auto [x1, x2] = (a.f[p] - b.f[q]).zero();
59        vector<ld> ax;
60        if (x1 > xl && x1 < xr) ax.push_back(x1);
61        if (x2 > xl && x2 < xr) ax.push_back(x2);
62        ax.push_back(xr);
63        for (auto xx : ax) {
64          ld x = (xl + xx) / 2;
65          auto f = a.f[p].val(x) > b.f[q].val(x) ? a.f[p] : b.f[q];
66          if (c.f.size() && c.f.back() == f)
67            c.a.back() = xx;
68          else
69            c.a.push_back(xx), c.f.push_back(f);
70          xl = xx;
71        }
72        if (a.a[p + 1] < xr + eps) p++;
73        if (b.a[q + 1] < xr + eps) q++;
74      }
75      return c;
76  };
77  pcef comb(const vector<func> &a, int l, int r) {
78    if (r - l == 1) return pcef({{0, 2 * pi}, {a[l]}});
79    int m = (l + r) / 2;
80    return comb(a, l, m) + comb(a, m, r);
81  }
```

6.3 凸包

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   typedef long long ll;
4   typedef long double ld;
5   struct cplx {
6     ll x, y;
7     ll abs2() { return x * x + y * y; }
8     ld abs() { return sqrt(abs2()); }
9   };
10  cplx operator+(const cplx &a, const cplx &b) { return {a.x + b.x, a.y + b.y}; }
11  cplx operator-(const cplx &a, const cplx &b) { return {a.x - b.x, a.y - b.y}; }
12  bool operator<(const cplx &a, const cplx &b) {
13    return a.x != b.x ? a.x < b.x : a.y < b.y;
14  }
15  bool operator!=(const cplx &a, const cplx &b) { return a < b || b < a; }
16  bool operator==(const cplx &a, const cplx &b) { return !(a != b); }
```

```
17  ll dot(const cplx &a, const cplx &b) { return a.x * b.x + a.y * b.y; }
18  ll det(const cplx &a, const cplx &b) { return a.x * b.y - a.y * b.x; }
19  void diagsort(vector<cplx> &a, cplx o) {
20    sort(a.begin(), a.end(), [&](const cplx &p, const cplx &q) {
21      if ((o < p) != (o < q)) return o < p;
22      auto d = det(p - o, q - o);
23      if (d) return d > 0;
24      return (p - o).abs2() > (q - o).abs2();
25    });
26    a.resize(unique(a.begin(), a.end(),
27                [&](const cplx &p, const cplx &q) {
28                  return !det(p - o, q - o) && dot(p - o, q - o) > 0;
29                }) -
30            a.begin());
31  }
32  vector<cplx> convex(vector<cplx> a) {
33    sort(a.begin(), a.end());
34    auto o = a[0];
35    a.erase(a.begin());
36    diagsort(a, o);
37    vector<cplx> res = {o};
38    for (auto v : a) {
39      while (res.size() >= 2) {
40        auto p = res[res.size() - 2], q = res[res.size() - 1];
41        if (det(q - p, v - q) > 0) break;
42        res.pop_back();
43      }
44      res.push_back(v);
45    }
46    return res;
47  }
48  double diameter(const vector<cplx> &a) {
49    double res = (a.back() - a[0]).abs();
50    for (int i = 1; i < a.size(); i++) res += (a[i] - a[i - 1]).abs();
51    return res;
52  }
53
54  cplx find(vector<cplx> &conv, cplx a) {
55    int l = 0, r = conv.size() - 1;
56    while (r - 1) {
57      int m = (l + r) / 2;
58      if (det(conv[m + 1] - conv[m], a) < 0)
59        r = m;
60      else
61        l = m + 1;
```

```
62    }
63    return conv[l];
64  }
65  ll solve(vector<ll> &a) {
66    vector<ll> s = {0};
67    for (auto v : a) s.push_back(s.back() + v);
68    vector<cplx> conv;
69    for (int i = 0; i < s.size(); i++) {
70      cplx r = {i, s[i]};
71      while (conv.size() >= 2) {
72        cplx p = conv[conv.size() - 2], q = conv.back();
73        if (det(q - p, r - q) > 0) break;
74        conv.pop_back();
75      }
76      conv.push_back(r);
77    }
78    ll res = 0;
79    for (int i = 0; i < a.size(); i++) {
80      auto p = find(conv, {1, a[i]});
81      res = max(res, s[i] - p.y + (p.x - i) * a[i]);
82    }
83    return res;
84  }
```

6.4 扫描线

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   const char el = '\n';
4   typedef long long ll;
5   typedef __int128 li;
6   const int inf = 3e6;
7   struct frac {
8     li a, b;
9     frac(li _a = 0, li _b = 1) : a(_a), b(_b) {
10      if (b < 0) a = -a, b = -b;
11    }
12  };
13  // note that without gcd plus operation also enlarges the fractor.
14  frac operator+(frac a, frac b) { return {a.a * b.b + b.a * a.b, a.b * b.b}; }
15  frac operator-(frac a, frac b) { return {a.a * b.b - b.a * a.b, a.b * b.b}; }
16  frac operator*(frac a, frac b) { return {a.a * b.a, a.b * b.b}; }
17  frac operator/(frac a, frac b) { return {a.a * b.b, a.b * b.a}; }
18  bool operator<(frac a, frac b) { return a.a * b.b < b.a * a.b; }
```

```
19  bool operator>(frac a, frac b) { return b < a; }
20  bool operator>=(frac a, frac b) { return !(a < b); }
21  bool operator<=(frac a, frac b) { return !(b < a); }
22  bool operator!=(frac a, frac b) { return a < b || b < a; }
23  bool operator==(frac a, frac b) { return !(a != b); }
24  frac abs(frac a) { return {a.a < 0 ? -a.a : a.a, a.b}; }
25  struct cplx {
26    frac x, y;
27  };
28  cplx operator+(const cplx &a, const cplx &b) { return {a.x + b.x, a.y + b.y}; }
29  cplx operator-(const cplx &a, const cplx &b) { return {a.x - b.x, a.y - b.y}; }
30  bool operator<(const cplx &a, const cplx &b) {
31    return a.x == b.x ? a.y < b.y : a.x < b.x;
32  }
33  bool operator!=(const cplx &a, const cplx &b) { return a < b || b < a; }
34  bool operator==(const cplx &a, const cplx &b) { return !(a != b); }
35  struct sgmt {
36    cplx a, b;
37    sgmt(cplx _a, cplx _b) : a(_a), b(_b) {
38      if (b < a) swap(a, b);
39    }
40    frac cut(frac x) const {
41      if (x == a.x) return a.y;
42      if (x == b.x) return b.y;
43      auto t1 = (b.x - x) * a.y, t2 = (x - a.x) * b.y;
44      t1.a += t2.a;
45      auto t3 = b.x - a.x;
46      auto res = t1 / t3;
47      return res;
48    }
49  };
50  bool operator<(const sgmt &a, const sgmt &b) {
51    if (a.a != b.a) return a.a < b.a;
52    return a.b < b.b;
53  }
54  map<sgmt, vector<cplx>> solve(vector<sgmt> line, vector<cplx> vtx) {
55    sort(line.begin(), line.end(), [&](sgmt a, sgmt b) { return a.a < b.a; });
56    auto cmp1 = [&](sgmt a, sgmt b) {
57      if (a.b != b.b) return a.b < b.b;
58      return a.a < b.a;
59    };
60    set<sgmt, decltype(cmp1)> curl(cmp1);
61    frac cx;
62    auto cmp2 = [&](sgmt a, sgmt b) {
63      frac ay = a.cut(cx), by = b.cut(cx);
64      if (ay == by) return a.cut(cx + 1) < b.cut(cx + 1);
65      return ay < by;
66    };
67    set<sgmt, decltype(cmp2)> scan(cmp2);
68    auto findc = [&](cplx v) {
69      auto [x, y] = v;
70      return *scan.lower_bound({{x, y}, {x + 1, y - inf}});
71    };
72    auto cmp3 = [&](sgmt a, sgmt b) { return a.a.y < b.a.y; };
73    set<sgmt, decltype(cmp3)> hori(cmp3);
74    auto findh = [&](cplx v) -> optional<sgmt> {
75      auto [x, y] = v;
76      auto it = hori.upper_bound({{x, y}, {x, y}});
77      if (it == hori.begin()) return nullopt;
78      it = prev(it);
79      if (it->b.y < y) return nullopt;
80      return *it;
81    };
82    reverse(line.begin(), line.end());
83    reverse(vtx.begin(), vtx.end());
84    map<sgmt, vector<cplx>> res;
85    while (vtx.size()) {
86      hori.clear();
87      frac x = vtx.back().x;
88      vector<cplx> curv;
89      while (vtx.size() && vtx.back().x <= x)
90        curv.push_back(vtx.back()), vtx.pop_back();
91      while (curl.size() && curl.begin()->b.x <= x)
92        scan.erase(*curl.begin()), curl.erase(curl.begin());
93      cx = x;
94      while (line.size() && line.back().a.x <= x) {
95        auto l = line.back();
96        line.pop_back();
97        if (l.b.x < x) continue;
98        if (l.a.x == l.b.x)
99          hori.insert(l);
100       else
101         scan.insert(l), curl.insert(l);
102     }
103     for (auto v : curv)
104       if (auto t = findh(v))
105         res[*t].push_back(v);
106       else
107         res[findc(v)].push_back(v);
108   }
```

35

```
109    return res;
110  }
111  const int k = 20;
112  struct graph {
113    vector<vector<int>> e;
114    graph(int n) : e(n + 1) {}
115    void adde(int u, int v) {
116      e[u].push_back(v);
117      e[v].push_back(u);
118    }
119    vector<array<int, k>> f;
120    vector<int> d;
121    void makef(int u, int w) {
122      f[u][0] = w;
123      for (int i = 1; i < k; i++) f[u][i] = f[f[u][i - 1]][i - 1];
124      d[u] = d[w] + 1;
125      for (auto &v : e[u])
126        if (v == w) {
127          v = e[u].back(), e[u].pop_back();
128          break;
129        }
130      for (auto v : e[u]) makef(v, u);
131    }
132    int lca(int u, int v) {
133      if (d[u] < d[v]) swap(u, v);
134      for (int i = 0; i < k; i++)
135        if (d[u] - d[v] >> i & 1) u = f[u][i];
136      if (u == v) return u;
137      for (int i = k - 1; i >= 0; i--)
138        if (f[u][i] != f[v][i]) u = f[u][i], v = f[v][i];
139      return f[u][0];
140    }
141    vector<int> s, c;
142    void addup(int u) {
143      for (auto v : e[u]) s[v] += s[u], addup(v), c[u] += c[v];
144    }
145    vector<int> solve(vector<pair<int, int>> q) {
146      int n = e.size();
147      f.resize(n);
148      d.assign(n, 0);
149      makef(1, 0);
150      s.assign(n, 0);
151      c.assign(n, 0);
152      for (auto [u, v] : q) {
153        auto w = lca(u, v);
154        s[w]++;
155        c[u]++, c[v]++, c[w]--, c[f[w][0]]--;
156      }
157      addup(1);
158      vector<int> res;
159      for (auto [u, v] : q) {
160        auto w = lca(u, v);
161        res.push_back(s[u] + s[v] - 2 * s[w] + c[w] - 1);
162      }
163      return res;
164    }
165  };
166  li read() {
167    ll n;
168    cin >> n;
169    return n;
170  }
171  int main() {
172    ios::sync_with_stdio(false);
173    cin.tie(0);
174    cout << setprecision(3);
175    int n, m;
176    cin >> n >> m;
177    vector<cplx> a(n);
178    for (auto &[x, y] : a) x.a = read(), y.a = read();
179    a.insert(a.begin(), {0, 0});
180    vector<pair<int, int>> edge(n - 1);
181    for (auto &[u, v] : edge) cin >> u >> v;
182    vector<sgmt> line;
183    for (auto [u, v] : edge) line.push_back(sgmt(a[u], a[v]));
184    vector<pair<cplx, cplx>> qry(m);
185    for (auto &[a, b] : qry)
186      a.x.a = read(), a.x.b = read(), a.y.a = read(), a.y.b = read(),
187      b.x.a = read(), b.x.b = read(), b.y.a = read(), b.y.b = read();
188    map<cplx, int> id;
189    for (int i = 1; i <= n; i++) id[a[i]] = i;
190    vector<cplx> vtx;
191    for (auto [a, b] : qry)
192      for (auto v : {a, b})
193        if (!id.count(v)) vtx.push_back(v);
194    sort(vtx.begin(), vtx.end());
195    vtx.resize(unique(vtx.begin(), vtx.end()) - vtx.begin());
196    for (int i = 0; i < vtx.size(); i++) id[vtx[i]] = n + 1 + i;
197    auto vol = solve(line, vtx);
198    graph g(id.size());
```

```
199    for (const auto &[p, q] : line) {
200      auto arr = vol[{p, q}];
201      arr.insert(arr.begin(), p), arr.push_back(q);
202      for (int i = 1; i < arr.size(); i++) g.adde(id[arr[i - 1]], id[arr[i]]);
203    }
204    vector<pair<int, int>> q;
205    for (auto &[a, b] : qry) q.push_back({id[a], id[b]});
206    vector<int> res = g.solve(q);
207    for (auto v : res) cout << v << ' ';
208    cout << el;
209    return 0;
210  }
```

7 杂项

7.1 随机数生成

```
1
2  namespace myrand {
3  random_device r;
4  default_random_engine e(r());
5  std::uniform_int_distribution<int> g(0, 1e9);
6  }  // namespace myrand
```

7.2 STL 容器 +Lambda

```
1
2    auto cmp = [&](int u, int v) {
3      if (g.e[u].size() != g.e[v].size()) return g.e[u].size() < g.e[v].size();
4      return u < v;
5    };
6    set<int, decltype(cmp)> st(cmp);
```

7.3 子集枚举

```
1
2  for (int p = 0; p < n; p++)
```

```
3    for (int q = p; q; q = q - 1 & p) {
4      int k = ~p & (n - 1);
5      c[k + q] = max(c[k + q], a[k] + b[q]);
6    }
```

7.4 二项式反演

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

$$f(n) = \sum_{i=m}^{n} \binom{n}{i} g(i) \iff g(n) = \sum_{i=m}^{n} (-1)^{n-i} \binom{n}{i} f(i)$$

$f(n)$: n 选 i 个；$g(n)$: 恰好 n 个

$$f(n) = \sum_{i=n}^{m} \binom{i}{n} g(i) \iff g(n) = \sum_{i=n}^{m} (-1)^{i-n} \binom{i}{n} f(i)$$

$f(n)$: 钦定 n 个；$g(n)$: 恰好 n 个

7.5 莫比乌斯反演

$$f(n) = \sum_{d|n} g(d) \iff g(n) = \sum_{d|n} \mu(d) f(n/d)$$

$$f(n) = \sum_{n|d} g(d) \iff g(n) = \sum_{n|d} \mu(d/n) f(d)$$

7.6 容斥原理

$$\left| \bigcup_{i=1}^{n} S_i \right| = \sum_{m=1}^{n} (-1)^{m-1} \sum_{a_i < a_{i+1}} \left| \bigcap_{i=1}^{m} S_{a_i} \right|$$

$$\left| \bigcap_{i=1}^{n} S_i \right| = |U| - \left| \bigcup_{i=1}^{n} \overline{S_i} \right|$$

$$f(S) = \sum_{S \subseteq T} g(T) \iff g(S) = \sum_{S \subseteq T} (-1)^{|T|-|S|} f(T)$$

7.7 Min-Max 容斥

$$\max_{i \in S} x_i = \sum_{T \subseteq S} (-1)^{|T|-1} \min_{j \in T} x_j$$

$$\min_{i \in S} x_i = \sum_{T \subseteq S} (-1)^{|T|-1} \max_{j \in T} x_j$$