Standard Code Library

Your TeamName

Your School

September 7, 2023

Contents

Start

Makefile

```
1  # ICPC
2  % : %.cpp
3      g++ -Wall -Wfatal-errors -Wshadow -g -std=c++2a $< -o $@ -O2
```

Template

```cpp
1   #include "bits/stdc++.h"
2   using namespace std;
3   using ll = long long;
4   using pii = pair<int, int>;
5   using pil = pair<int, ll>;
6   using pli = pair<ll, int>;
7   using pll = pair<ll, ll>;
8
9   inline void yes() { cout  << "YES" << "\n"; return; }
10  inline void no() { cout << "NO" << "\n"; return; }
11
12  template<typename T>
13  bool chmin(T &x, T val) { if (val < x) { x = val; return true; } return false; }
14  template<typename T>
15  bool chmax(T &x, T val) { if (x < val) { x = val; return true; } return false; }
16
17  #define DEBUG 1
18  #define MULTI_TEST 1
19
20  void solve() {
21      int n;
22      cin >> n;
23      vector<int> a(n);
24      for (auto &o: a) {
25          cin >> o;
26      }
27  }
28
29  int main() {
30      cin.tie(0)->sync_with_stdio(0);
31
32      if (MULTI_TEST) {
33          int t;
34          cin >> t;
35          while (t--) {
36              solve();
37          }
38      }
39      else {
40          solve();
41      }
42  }
```

Data Structure

Segment Tree

```cpp
1   template<class Info>
2   struct SegmentTree {
3       int n;
4       std::vector<Info> info;
5
6       SegmentTree() : n(0) {}
7       SegmentTree(int n_, Info v_ = Info()) {
8           init(n_, v_);
9       }
10      template<class T>
11      SegmentTree(std::vector<T> init_) {
12          init(init_);
```

```cpp
13          }
14
15          void init(int n_, Info v_ = Info()) {
16              init(std::vector(n_, v_));
17          }
18          template<class T>
19          void init(std::vector<T> init_) {
20              n = init_.size();
21              info.assign(4 << std::__lg(n), Info());
22              std::function<void(int, int, int)> build = [&](int p, int l, int r) {
23                  if (r - l == 1) {
24                      info[p] = init_[l];
25                      return;
26                  }
27                  int m = (l + r) / 2;
28                  build(2 * p, l, m);
29                  build(2 * p + 1, m, r);
30                  pull(p);
31              };
32              build(1, 0, n);
33          }
34          void pull(int p) {
35              info[p] = info[2 * p] + info[2 * p + 1];
36          }
37          void modify(int p, int l, int r, int x, const Info &v) {
38              if (r - l == 1) {
39                  info[p] = v;
40                  return;
41              }
42              int m = (l + r) / 2;
43              if (x < m) {
44                  modify(2 * p, l, m, x, v);
45              } else {
46                  modify(2 * p + 1, m, r, x, v);
47              }
48              pull(p);
49          }
50          void modify(int p, const Info &v) {
51              modify(1, 0, n, p, v);
52          }
53          Info rangeQuery(int p, int l, int r, int x, int y) {
54              if (l >= y || r <= x) {
55                  return Info();
56              }
57              if (l >= x && r <= y) {
58                  return info[p];
59              }
60              int m = (l + r) / 2;
61              return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
62          }
63          Info rangeQuery(int l, int r) {
64              return rangeQuery(1, 0, n, l, r);
65          }
66          template<class F>
67          int findFirst(int p, int l, int r, int x, int y, F pred) {
68              if (l >= y || r <= x || !pred(info[p])) {
69                  return -1;
70              }
71              if (r - l == 1) {
72                  return l;
73              }
74              int m = (l + r) / 2;
75              int res = findFirst(2 * p, l, m, x, y, pred);
76              if (res == -1) {
77                  res = findFirst(2 * p + 1, m, r, x, y, pred);
78              }
79              return res;
80          }
81          template<class F>
82          int findFirst(int l, int r, F pred) {
83              return findFirst(1, 0, n, l, r, pred);
```

```cpp
        }
        template<class F>
        int findLast(int p, int l, int r, int x, int y, F pred) {
            if (l >= y || r <= x || !pred(info[p])) {
                return -1;
            }
            if (r - l == 1) {
                return l;
            }
            int m = (l + r) / 2;
            int res = findLast(2 * p + 1, m, r, x, y, pred);
            if (res == -1) {
                res = findLast(2 * p, l, m, x, y, pred);
            }
            return res;
        }
        template<class F>
        int findLast(int l, int r, F pred) {
            return findLast(1, 0, n, l, r, pred);
        }
};

constexpr ll inf = 1E18;

struct Info {
    ll cnt = 0;
    ll sum = 0;
    ll min = inf;
};

Info operator+(Info a, Info b) {
    Info c;
    c.cnt = a.cnt + b.cnt;
    c.sum = a.sum + b.sum;
    c.min = std::min(a.min, b.min);
    return c;
}
```

Lazy Segment Tree

```cpp
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
```

```cpp
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
```

```cpp
            push(p);
            int res = findFirst(2 * p, l, m, x, y, pred);
            if (res == -1) {
                res = findFirst(2 * p + 1, m, r, x, y, pred);
            }
            return res;
        }
        template<class F>
        int findFirst(int l, int r, F pred) {
            return findFirst(1, 0, n, l, r, pred);
        }
        template<class F>
        int findLast(int p, int l, int r, int x, int y, F pred) {
            if (l >= y || r <= x || !pred(info[p])) {
                return -1;
            }
            if (r - l == 1) {
                return l;
            }
            int m = (l + r) / 2;
            push(p);
            int res = findLast(2 * p + 1, m, r, x, y, pred);
            if (res == -1) {
                res = findLast(2 * p, l, m, x, y, pred);
            }
            return res;
        }
        template<class F>
        int findLast(int l, int r, F pred) {
            return findLast(1, 0, n, l, r, pred);
        }
};

struct Tag {
    ll add = 0;

    void apply(Tag t) {
        add += t.add;
    }
};

struct Info {
    ll mn = 1E18;

    void apply(Tag t) {
        mn += t.add;
    }
};

Info operator+(Info a, Info b) {
    return {std::min(a.mn, b.mn)};
}
```

BIT

```cpp
template <typename T>
struct BIT {
    int n;
    vector<T> a;

    BIT(int n = 0) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        a.assign(n, T());
    }

    void add(int x, int v) {
        while (x < n) {
```

```
17                  a[x] += v;
18                  x += x & -x;
19              }
20          }
21
22      T sum(int x) {
23          auto ret = T();
24          while (x > 0) {
25              ret += a[x];
26              x -= x & -x;
27          }
28          return ret;
29      }
30
31      T rangeSum(int l, int r) {
32          if (l == 0) return sum(r);
33          else return sum(r) - sum(l - 1);
34      }
35
36  };
```

DSU

```
1   struct DSU {
2       vector<int> f, siz, rnk;
3       int cc;
4
5       DSU() {}
6       DSU(int n) {
7           init(n);
8       }
9
10      void init(int n) {
11          f.resize(n);
12          iota(f.begin(), f.end(), 0);
13          siz.assign(n, 1);
14          rnk.assign(n, 0);
15          cc = n;
16      }
17
18      int find(int x) {
19          while (x != f[x]) {
20              x = f[x] = f[f[x]];
21          }
22          return x;
23      }
24
25      bool same(int x, int y) {
26          return find(x) == find(y);
27      }
28
29      int merge(int x, int y) {
30          x = find(x);
31          y = find(y);
32          if (x == y) return -1;
33          --cc;
34          if (rnk[x] > rnk[y]) swap(x, y);
35          siz[y] += siz[x];
36          f[x] = y;
37          if (rnk[x] == rnk[y]) rnk[y]++;
38          return y;
39      }
40
41      int size(int x) {
42          return siz[find(x)];
43      }
44  };
```

KD Tree

```cpp
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; }
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
    T dist2() const { return x*x + y*y; }
    double dist() const { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() const { return atan2(y, x); }
    P unit() const { return *this/dist(); } // makes dist()=1
    P perp() const { return P(-y, x); } // rotates +90 degrees
    P normal() const { return perp().unit(); }
    // returns point rotated 'a' radians ccw around the origin
    P rotate(double a) const {
        return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << "," << p.y << ")"; }
};

using T = ll;
using P = Point<T>;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
    P pt; // if this is a leaf, the single point in it
    T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
    Node *first = 0, *second = 0;

    T distance(const P& p) { // min squared distance to a point
        T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
        T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
        return (P(x,y) - p).dist2();
    }

    Node(vector<P>&& vp) : pt(vp[0]) {
        for (P p : vp) {
            x0 = min(x0, p.x); x1 = max(x1, p.x);
            y0 = min(y0, p.y); y1 = max(y1, p.y);
        }
        if (vp.size() > 1) {
            // split on x if width >= height (not ideal...)
            sort(begin(vp), end(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
            // divide by taking half the array for each child (not
            // best performance with many duplicates in the middle)
            int half = (int)size(vp)/2;
            first = new Node({vp.begin(), vp.begin() + half});
            second = new Node({vp.begin() + half, vp.end()});
        }
    }
};

struct KDTree {
    Node* root;
    KDTree(const vector<P>& vp) : root(new Node({begin(vp), end(vp)})) {}

    pair<T, P> search(Node *node, const P& p) {
```

```
70        if (!node->first) {
71            // uncomment if we should not find the point itself:
72            // if (p == node->pt) return {INF, P()};
73            return make_pair((p - node->pt).dist2(), node->pt);
74        }
75
76        Node *f = node->first, *s = node->second;
77        T bfirst = f->distance(p), bsec = s->distance(p);
78        if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
79
80        // search closest side first, other side if needed
81        auto best = search(f, p);
82        if (bsec < best.first)
83            best = min(best, search(s, p));
84        return best;
85    }
86
87    // find nearest point to a point, and its squared distance
88    // (requires an arbitrary operator< for Point)
89    pair<T, P> nearest(const P& p) {
90        return search(root, p);
91    }
92 };
```

Math

Mint

```
1  template<class T>
2  constexpr T power(T a, ll b) {
3      T res = 1;
4      for (; b; b /= 2, a *= a) {
5          if (b % 2) res *= a;
6      }
7      return res;
8  }
9
10 constexpr ll mul(ll a, ll b, ll p) {
11     ll res = a * b - ll(1.L * a * b / p) * p;
12     res %= p;
13     if (res < 0) res += p;
14     return res;
15 }
16
17 template<ll P>
18 struct MLong {
19     ll x;
20     constexpr MLong() : x{} {}
21     constexpr MLong(ll x) : x{norm(x % getMod())} {}
22
23     static ll Mod;
24     constexpr static ll getMod() {
25         if (P > 0) {
26             return P;
27         } else {
28             return Mod;
29         }
30     }
31     constexpr static void setMod(ll Mod_) {
32         Mod = Mod_;
33     }
34     constexpr ll norm(ll x) const {
35         if (x < 0) {
36             x += getMod();
37         }
38         if (x >= getMod()) {
39             x -= getMod();
40         }
41         return x;
42     }
```

```cpp
43
44        constexpr ll val() const { return x;}
45        explicit constexpr operator ll() const { return x; }
46
47        constexpr MLong operator-() const { MLong res; res.x = norm(getMod() - x); return res; }
48        constexpr MLong inv() const { assert(x != 0); return power(*this, getMod() - 2); }
49        constexpr MLong &operator*=(MLong rhs) & { x = mul(x, rhs.x, getMod()); return *this; }
50        constexpr MLong &operator+=(MLong rhs) & { x = norm(x + rhs.x); return *this; }
51        constexpr MLong &operator-=(MLong rhs) & { x = norm(x - rhs.x); return *this; }
52        constexpr MLong &operator/=(MLong rhs) & { return *this *= rhs.inv(); }
53
54        friend constexpr MLong operator*(MLong lhs, MLong rhs) { MLong res = lhs; res *= rhs; return res; }
55        friend constexpr MLong operator+(MLong lhs, MLong rhs) { MLong res = lhs; res += rhs; return res; }
56        friend constexpr MLong operator-(MLong lhs, MLong rhs) { MLong res = lhs; res -= rhs; return res; }
57        friend constexpr MLong operator/(MLong lhs, MLong rhs) { MLong res = lhs; res /= rhs; return res; }
58
59        friend constexpr std::istream &operator>>(std::istream &is, MLong &a) { ll v{}; is >> v; a = MLong(v); return is;
   ↪   }
60        friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a) { return os << a.val(); }
61        friend constexpr bool operator==(MLong lhs, MLong rhs) { return lhs.val() == rhs.val(); }
62        friend constexpr bool operator!=(MLong lhs, MLong rhs) { return lhs.val() != rhs.val(); }
63   };
64
65   template<>
66   ll MLong<0LL>::Mod = 1;
67
68   template<int P>
69   struct MInt {
70        int x;
71        constexpr MInt() : x{} {}
72        constexpr MInt(ll x) : x{norm(x % getMod())} {}
73
74        static int Mod;
75        constexpr static int getMod() {
76            if (P > 0) {
77                return P;
78            } else {
79                return Mod;
80            }
81        }
82        constexpr static void setMod(int Mod_) {
83            Mod = Mod_;
84        }
85        constexpr int norm(int x) const {
86            if (x < 0) {
87                x += getMod();
88            }
89            if (x >= getMod()) {
90                x -= getMod();
91            }
92            return x;
93        }
94
95        constexpr int val() const { return x; }
96        explicit constexpr operator int() const { return x; }
97
98        constexpr MInt operator-() const { MInt res; res.x = norm(getMod() - x); return res; }
99        constexpr MInt inv() const { assert(x != 0); return power(*this, getMod() - 2); }
100       constexpr MInt &operator*=(MInt rhs) & { x = 1LL * x * rhs.x % getMod(); return *this; }
101       constexpr MInt &operator+=(MInt rhs) & { x = norm(x + rhs.x); return *this; }
102       constexpr MInt &operator-=(MInt rhs) & { x = norm(x - rhs.x); return *this; }
103       constexpr MInt &operator/=(MInt rhs) & { return *this *= rhs.inv(); }
104
105       friend constexpr MInt operator*(MInt lhs, MInt rhs) { MInt res = lhs; res *= rhs; return res; }
106       friend constexpr MInt operator+(MInt lhs, MInt rhs) { MInt res = lhs; res += rhs; return res; }
107       friend constexpr MInt operator-(MInt lhs, MInt rhs) { MInt res = lhs; res -= rhs; return res; }
108       friend constexpr MInt operator/(MInt lhs, MInt rhs) { MInt res = lhs; res /= rhs; return res; }
109
110       friend constexpr std::istream &operator>>(std::istream &is, MInt &a) { ll v{}; is >> v; a = MInt(v); return is; }
111       friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) { return os << a.val(); }
112       friend constexpr bool operator==(MInt lhs, MInt rhs) { return lhs.val() == rhs.val(); }
```

```
113    friend constexpr bool operator!=(MInt lhs, MInt rhs) { return lhs.val() != rhs.val(); }
114  };
115
116  template<>
117  int MInt<0>::Mod = 1;
118
119  template<int V, int P>
120  constexpr MInt<P> CInv = MInt<P>(V).inv();
121
122  constexpr int P = 1000000007;
123  using Z = MInt<P>;
```

Combin

```
1    template<class T>
2    constexpr T power(T a, ll b) {
3        T res = 1;
4        for (; b; b /= 2, a *= a) {
5            if (b % 2) res *= a;
6        }
7        return res;
8    }
9
10   constexpr ll mul(ll a, ll b, ll p) {
11       ll res = a * b - ll(1.L * a * b / p) * p;
12       res %= p;
13       if (res < 0) res += p;
14       return res;
15   }
16
17   template<int P>
18   struct MInt {
19       int x;
20       constexpr MInt() : x{} {}
21       constexpr MInt(ll x) : x{norm(x % getMod())} {}
22
23       static int Mod;
24       constexpr static int getMod() {
25           if (P > 0) {
26               return P;
27           } else {
28               return Mod;
29           }
30       }
31       constexpr static void setMod(int Mod_) {
32           Mod = Mod_;
33       }
34       constexpr int norm(int x) const {
35           if (x < 0) {
36               x += getMod();
37           }
38           if (x >= getMod()) {
39               x -= getMod();
40           }
41           return x;
42       }
43
44       constexpr int val() const { return x; }
45       explicit constexpr operator int() const { return x; }
46
47       constexpr MInt operator-() const { MInt res; res.x = norm(getMod() - x); return res; }
48       constexpr MInt inv() const { assert(x != 0); return power(*this, getMod() - 2); }
49       constexpr MInt &operator*=(MInt rhs) & { x = 1LL * x * rhs.x % getMod(); return *this; }
50       constexpr MInt &operator+=(MInt rhs) & { x = norm(x + rhs.x); return *this; }
51       constexpr MInt &operator-=(MInt rhs) & { x = norm(x - rhs.x); return *this; }
52       constexpr MInt &operator/=(MInt rhs) & { return *this *= rhs.inv(); }
53
54       friend constexpr MInt operator*(MInt lhs, MInt rhs) { MInt res = lhs; res *= rhs; return res; }
55       friend constexpr MInt operator+(MInt lhs, MInt rhs) { MInt res = lhs; res += rhs; return res; }
56       friend constexpr MInt operator-(MInt lhs, MInt rhs) { MInt res = lhs; res -= rhs; return res; }
57       friend constexpr MInt operator/(MInt lhs, MInt rhs) { MInt res = lhs; res /= rhs; return res; }
```

```cpp
58
59        friend constexpr std::istream &operator>>(std::istream &is, MInt &a) { ll v{}; is >> v; a = MInt(v); return is; }
60        friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) { return os << a.val(); }
61        friend constexpr bool operator==(MInt lhs, MInt rhs) { return lhs.val() == rhs.val(); }
62        friend constexpr bool operator!=(MInt lhs, MInt rhs) { return lhs.val() != rhs.val(); }
63    };
64
65    template<>
66    int MInt<0>::Mod = 1;
67
68    template<int V, int P>
69    constexpr MInt<P> CInv = MInt<P>(V).inv();
70
71    constexpr int P = 1000000007;
72    using Z = MInt<P>;
73
74    struct Comb {
75        int n;
76        std::vector<Z> _fac;
77        std::vector<Z> _invfac;
78        std::vector<Z> _inv;
79
80        Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
81        Comb(int n) : Comb() {
82            init(n);
83        }
84
85        void init(int m) {
86            if (m <= n) return;
87            _fac.resize(m + 1);
88            _invfac.resize(m + 1);
89            _inv.resize(m + 1);
90
91            for (int i = n + 1; i <= m; i++) {
92                _fac[i] = _fac[i - 1] * i;
93            }
94            _invfac[m] = _fac[m].inv();
95            for (int i = m; i > n; i--) {
96                _invfac[i - 1] = _invfac[i] * i;
97                _inv[i] = _invfac[i] * _fac[i - 1];
98            }
99            n = m;
100        }
101
102        Z fac(int m) {
103            if (m > n) init(2 * m);
104            return _fac[m];
105        }
106        Z invfac(int m) {
107            if (m > n) init(2 * m);
108            return _invfac[m];
109        }
110        Z inv(int m) {
111            if (m > n) init(2 * m);
112            return _inv[m];
113        }
114        Z binom(int n, int m) {
115            if (n < m || m < 0) return 0;
116            return fac(n) * invfac(m) * invfac(n - m);
117        }
118        Z catalan(int n) {
119            return binom(2 * n, n) * inv(n + 1);
120        }
121        Z lucas(int n, int m) {
122            if (n % P < m % P) return 0;
123            if (n < P) return binom(n, m);
124            return 1ll * lucas(n / P, m / P) * binom(n % P, m % P);
125        }
126    } comb;
```

Sieve

```
1   std::vector<int> minp, primes;
2
3   void sieve(int n) {
4       minp.assign(n + 1, 0);
5       primes.clear();
6
7       for (int i = 2; i <= n; i++) {
8           if (minp[i] == 0) {
9               minp[i] = i;
10              primes.push_back(i);
11          }
12
13          for (auto p : primes) {
14              if (i * p > n) {
15                  break;
16              }
17              minp[i * p] = p;
18              if (p == minp[i]) {
19                  break;
20              }
21          }
22      }
23  }
```

Polynomial

```
1   std::vector<int> rev;
2   template<int P>
3   std::vector<MInt<P>> roots{0, 1};
4
5   template<int P>
6   constexpr MInt<P> findPrimitiveRoot() {
7       MInt<P> i = 2;
8       int k = __builtin_ctz(P - 1);
9       while (true) {
10          if (power(i, (P - 1) / 2) != 1) {
11              break;
12          }
13          i += 1;
14      }
15      return power(i, (P - 1) >> k);
16  }
17
18  template<int P>
19  constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
20
21  template<>
22  constexpr MInt<998244353> primitiveRoot<998244353> {31};
23
24  template<int P>
25  constexpr void dft(std::vector<MInt<P>> &a) {
26      int n = a.size();
27
28      if (int(rev.size()) != n) {
29          int k = __builtin_ctz(n) - 1;
30          rev.resize(n);
31          for (int i = 0; i < n; i++) {
32              rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
33          }
34      }
35
36      for (int i = 0; i < n; i++) {
37          if (rev[i] < i) {
38              std::swap(a[i], a[rev[i]]);
39          }
40      }
41      if (roots<P>.size() < n) {
42          int k = __builtin_ctz(roots<P>.size());
43          roots<P>.resize(n);
```

```
44          while ((1 << k) < n) {
45              auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
46              for (int i = 1 << (k - 1); i < (1 << k); i++) {
47                  roots<P>[2 * i] = roots<P>[i];
48                  roots<P>[2 * i + 1] = roots<P>[i] * e;
49              }
50              k++;
51          }
52      }
53      for (int k = 1; k < n; k *= 2) {
54          for (int i = 0; i < n; i += 2 * k) {
55              for (int j = 0; j < k; j++) {
56                  MInt<P> u = a[i + j];
57                  MInt<P> v = a[i + j + k] * roots<P>[k + j];
58                  a[i + j] = u + v;
59                  a[i + j + k] = u - v;
60              }
61          }
62      }
63  }
64
65  template<int P>
66  constexpr void idft(std::vector<MInt<P>> &a) {
67      int n = a.size();
68      std::reverse(a.begin() + 1, a.end());
69      dft(a);
70      MInt<P> inv = (1 - P) / n;
71      for (int i = 0; i < n; i++) {
72          a[i] *= inv;
73      }
74  }
75
76  template<int P = 998244353>
77  struct Poly : public std::vector<MInt<P>> {
78      using Value = MInt<P>;
79
80      Poly() : std::vector<Value>() {}
81      explicit constexpr Poly(int n) : std::vector<Value>(n) {}
82
83      explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a) {}
84      constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a) {}
85
86      template<class InputIt, class = std::::_RequireInputIter<InputIt>>
87      explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(first, last) {}
88
89      template<class F>
90      explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
91          for (int i = 0; i < n; i++) {
92              (*this)[i] = f(i);
93          }
94      }
95
96      constexpr Poly shift(int k) const {
97          if (k >= 0) {
98              auto b = *this;
99              b.insert(b.begin(), k, 0);
100             return b;
101         } else if (this->size() <= -k) {
102             return Poly();
103         } else {
104             return Poly(this->begin() + (-k), this->end());
105         }
106     }
107     constexpr Poly trunc(int k) const {
108         Poly f = *this;
109         f.resize(k);
110         return f;
111     }
112     constexpr friend Poly operator+(const Poly &a, const Poly &b) {
113         Poly res(std::max(a.size(), b.size()));
114         for (int i = 0; i < a.size(); i++) {
```

14

```
115                res[i] += a[i];
116            }
117            for (int i = 0; i < b.size(); i++) {
118                res[i] += b[i];
119            }
120            return res;
121        }
122        constexpr friend Poly operator-(const Poly &a, const Poly &b) {
123            Poly res(std::max(a.size(), b.size()));
124            for (int i = 0; i < a.size(); i++) {
125                res[i] += a[i];
126            }
127            for (int i = 0; i < b.size(); i++) {
128                res[i] -= b[i];
129            }
130            return res;
131        }
132        constexpr friend Poly operator-(const Poly &a) {
133            std::vector<Value> res(a.size());
134            for (int i = 0; i < int(res.size()); i++) {
135                res[i] = -a[i];
136            }
137            return Poly(res);
138        }
139        constexpr friend Poly operator*(Poly a, Poly b) {
140            if (a.size() == 0 || b.size() == 0) {
141                return Poly();
142            }
143            if (a.size() < b.size()) {
144                std::swap(a, b);
145            }
146            int n = 1, tot = a.size() + b.size() - 1;
147            while (n < tot) {
148                n *= 2;
149            }
150            if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
151                Poly c(a.size() + b.size() - 1);
152                for (int i = 0; i < a.size(); i++) {
153                    for (int j = 0; j < b.size(); j++) {
154                        c[i + j] += a[i] * b[j];
155                    }
156                }
157                return c;
158            }
159            a.resize(n);
160            b.resize(n);
161            dft(a);
162            dft(b);
163            for (int i = 0; i < n; ++i) {
164                a[i] *= b[i];
165            }
166            idft(a);
167            a.resize(tot);
168            return a;
169        }
170        constexpr friend Poly operator*(Value a, Poly b) {
171            for (int i = 0; i < int(b.size()); i++) {
172                b[i] *= a;
173            }
174            return b;
175        }
176        constexpr friend Poly operator*(Poly a, Value b) {
177            for (int i = 0; i < int(a.size()); i++) {
178                a[i] *= b;
179            }
180            return a;
181        }
182        constexpr friend Poly operator/(Poly a, Value b) {
183            for (int i = 0; i < int(a.size()); i++) {
184                a[i] /= b;
185            }
```

```cpp
186             return a;
187         }
188         constexpr Poly &operator+=(Poly b) {
189             return (*this) = (*this) + b;
190         }
191         constexpr Poly &operator-=(Poly b) {
192             return (*this) = (*this) - b;
193         }
194         constexpr Poly &operator*=(Poly b) {
195             return (*this) = (*this) * b;
196         }
197         constexpr Poly &operator*=(Value b) {
198             return (*this) = (*this) * b;
199         }
200         constexpr Poly &operator/=(Value b) {
201             return (*this) = (*this) / b;
202         }
203         constexpr Poly deriv() const {
204             if (this->empty()) {
205                 return Poly();
206             }
207             Poly res(this->size() - 1);
208             for (int i = 0; i < this->size() - 1; ++i) {
209                 res[i] = (i + 1) * (*this)[i + 1];
210             }
211             return res;
212         }
213         constexpr Poly integr() const {
214             Poly res(this->size() + 1);
215             for (int i = 0; i < this->size(); ++i) {
216                 res[i + 1] = (*this)[i] / (i + 1);
217             }
218             return res;
219         }
220         constexpr Poly inv(int m) const {
221             Poly x{(*this)[0].inv()};
222             int k = 1;
223             while (k < m) {
224                 k *= 2;
225                 x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
226             }
227             return x.trunc(m);
228         }
229         constexpr Poly log(int m) const {
230             return (deriv() * inv(m)).integr().trunc(m);
231         }
232         constexpr Poly exp(int m) const {
233             Poly x{1};
234             int k = 1;
235             while (k < m) {
236                 k *= 2;
237                 x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
238             }
239             return x.trunc(m);
240         }
241         constexpr Poly pow(int k, int m) const {
242             int i = 0;
243             while (i < this->size() && (*this)[i] == 0) {
244                 i++;
245             }
246             if (i == this->size() || 1LL * i * k >= m) {
247                 return Poly(m);
248             }
249             Value v = (*this)[i];
250             auto f = shift(-i) * v.inv();
251             return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
252         }
253         constexpr Poly sqrt(int m) const {
254             Poly x{1};
255             int k = 1;
256             while (k < m) {
```

16

```
257          k *= 2;
258          x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
259      }
260      return x.trunc(m);
261  }
262  constexpr Poly mulT(Poly b) const {
263      if (b.size() == 0) {
264          return Poly();
265      }
266      int n = b.size();
267      std::reverse(b.begin(), b.end());
268      return ((*this) * b).shift(-(n - 1));
269  }
270  std::vector<Value> eval(std::vector<Value> x) const {
271      if (this->size() == 0) {
272          return std::vector<Value>(x.size(), 0);
273      }
274      const int n = std::max(x.size(), this->size());
275      std::vector<Poly> q(4 * n);
276      std::vector<Value> ans(x.size());
277      x.resize(n);
278      std::function<void(int, int, int)> build = [&](int p, int l, int r) {
279          if (r - l == 1) {
280              q[p] = Poly{1, -x[l]};
281          } else {
282              int m = (l + r) / 2;
283              build(2 * p, l, m);
284              build(2 * p + 1, m, r);
285              q[p] = q[2 * p] * q[2 * p + 1];
286          }
287      };
288      build(1, 0, n);
289      std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
290          if (r - l == 1) {
291              if (l < int(ans.size())) {
292                  ans[l] = num[0];
293              }
294          } else {
295              int m = (l + r) / 2;
296              work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - l));
297              work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
298          }
299      };
300      work(1, 0, n, mulT(q[1].inv(n)));
301      return ans;
302  }
303  };
304
305  template<int P = 998244353>
306  Poly<P> berlekampMassey(const Poly<P> &s) {
307      Poly<P> c;
308      Poly<P> oldC;
309      int f = -1;
310      for (int i = 0; i < s.size(); i++) {
311          auto delta = s[i];
312          for (int j = 1; j <= c.size(); j++) {
313              delta -= c[j - 1] * s[i - j];
314          }
315          if (delta == 0) {
316              continue;
317          }
318          if (f == -1) {
319              c.resize(i + 1);
320              f = i;
321          } else {
322              auto d = oldC;
323              d *= -1;
324              d.insert(d.begin(), 1);
325              MInt<P> df1 = 0;
326              for (int j = 1; j <= d.size(); j++) {
327                  df1 += d[j - 1] * s[f + 1 - j];
```

```cpp
                }
                assert(df1 != 0);
                auto coef = delta / df1;
                d *= coef;
                Poly<P> zeros(i - f - 1);
                zeros.insert(zeros.end(), d.begin(), d.end());
                d = zeros;
                auto temp = c;
                c += d;
                if (i - temp.size() > f - oldC.size()) {
                    oldC = temp;
                    f = i;
                }
            }
        }
        c *= -1;
        c.insert(c.begin(), 1);
        return c;
}


template<int P = 998244353>
MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
    int m = q.size() - 1;
    while (n > 0) {
        auto newq = q;
        for (int i = 1; i <= m; i += 2) {
            newq[i] *= -1;
        }
        auto newp = p * newq;
        newq = q * newq;
        for (int i = 0; i < m; i++) {
            p[i] = newp[i * 2 + n % 2];
        }
        for (int i = 0; i <= m; i++) {
            q[i] = newq[i * 2];
        }
        n /= 2;
    }
    return p[0] / q[0];
}
```

Graph

SCC

```cpp
struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    SCC() {}
    SCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
```

```cpp
25          }
26
27          void dfs(int x) {
28              dfn[x] = low[x] = cur++;
29              stk.push_back(x);
30
31              for (auto y : adj[x]) {
32                  if (dfn[y] == -1) {
33                      dfs(y);
34                      low[x] = std::min(low[x], low[y]);
35                  } else if (bel[y] == -1) {
36                      low[x] = std::min(low[x], dfn[y]);
37                  }
38              }
39
40              if (dfn[x] == low[x]) {
41                  int y;
42                  do {
43                      y = stk.back();
44                      bel[y] = cnt;
45                      stk.pop_back();
46                  } while (y != x);
47                  cnt++;
48              }
49          }
50
51          std::vector<int> work() {
52              for (int i = 0; i < n; i++) {
53                  if (dfn[i] == -1) {
54                      dfs(i);
55                  }
56              }
57              return bel;
58          }
59      };
```

Centroid Decomposition

```cpp
1   void solve() {
2       int n, T;
3       cin >> n >> T;
4
5       vector<vector<array<int, 2>>> g(n);
6
7       for (int i = 1; i < n; i++) {
8           int u, v, w;
9           cin >> u >> v >> w;
10          u--, v--;
11          g[u].push_back({v, w});
12          g[v].push_back({u, w});
13      }
14
15      int q;
16      cin >> q;
17
18      vector<vector<array<int, 3>>> qry(n);
19      for (int i = 0; i < q; i++) {
20          int a, b;
21          cin >> a >> b;
22          a--, b--;
23          qry[0].push_back({a, b, i});
24      }
25      vector<ll> ans(q, inf);
26
27      vector<bool> vis(n);
28      vector<int> siz(n), bel(n);
29
30      auto dfs = [&](auto self, int x, int p) -> void {
31          siz[x] = 1;
32          for (auto [y, w] : g[x]) {
33              if (y == p) {
```

```
34                  continue;
35              }
36              self(self, y, x);
37              siz[x] += siz[y];
38          }
39      };
40      dfs(dfs, 0, -1);
41
42      vector<ll> dep(n);
43      auto solve = [&](auto &&self, int r) -> void {
44          auto Q = std::move(qry[r]);
45
46          auto find = [&](auto self, int x, int p, int s) -> int {
47              for (auto [y, _] : g[x]) {
48                  if (y == p || vis[y] || 2 * siz[y] <= s) {
49                      continue;
50                  }
51                  return self(self, y, x, s);
52              }
53              return x;
54          };
55          r = find(find, r, -1, siz[r]);
56          vis[r] = true;
57
58          auto dfs = [&](auto self, int x, int p) -> void {
59              siz[x] = 1;
60              for (auto [y, w] : g[x]) {
61                  if (y == p || vis[y]) {
62                      continue;
63                  }
64                  dep[y] = dep[x] + w;
65                  bel[y] = x == r ? y : bel[x];
66                  self(self, y, x);
67                  siz[x] += siz[y];
68              }
69          };
70          dfs(dfs, r, -1);
71
72          for (auto [a, b, i] : Q) {
73              ans[i] = min(ans[i], dep[a] + dep[b]);
74              if (bel[a] == bel[b]) {
75                  qry[bel[a]].push_back({a, b, i});
76              }
77          }
78          for (auto [y, _] : g[r]) {
79              if (!vis[y]) {
80                  self(self, y);
81              }
82          }
83      };
84      solve(solve, 0);
85
86      for (int i = 0; i < q; i++) {
87          cout << ans[i] << "\n";
88      }
89  }
90
```

DSU On Tree

```
1  struct FreqBuckets {
2      vector<int> occ;
3      vector<int> freq;
4      FreqBuckets(int n, int maxC) : occ(maxC + 1, 0), freq(n + 1) { }
5
6      void add(int x, int mul) {
7          if (mul == +1) {
8              occ[x]++;
9              freq[occ[x]]++;
10         }
11         else if (mul == -1) {
```

```
12              freq[occ[x]]--;
13              occ[x]--;
14          }
15          else assert(false);
16      }
17  };
18
19  int main() {
20      ios::sync_with_stdio(false);
21      cin.tie(0);
22
23      int n,m; cin >> n >> m;
24
25      vector<int> c(n);
26      for (int i = 0; i < n; i++)
27          cin >> c[i];
28
29      vector<vector<int>> g(n);
30      for (int i = 0; i + 1 < n; i++) {
31          int u,v; cin >> u >> v; u--; v--;
32          g[u].emplace_back(v);
33          g[v].emplace_back(u);
34      }
35
36      vector<int> sz(n, 1);
37      function<void(int, int)> dfs_hld = [&](int u, int p) {
38          if (p != -1) {
39              auto it = find(g[u].begin(), g[u].end(), p);
40              assert(it != g[u].end());
41              g[u].erase(it);
42          }
43
44          for (auto& v : g[u]) {
45              dfs_hld(v, u);
46              sz[u] += sz[v];
47              if (sz[v] > sz[g[u][0]])
48                  swap(v, g[u][0]);
49          }
50      };
51      dfs_hld(0, -1);
52
53      vector<vector<pair<int, int>>> qry(n);
54      for (int i = 0; i < m; i++) {
55          int v,k; cin >> v >> k; v--;
56          qry[v].emplace_back(k, i);
57      }
58
59      const int maxC = 100000;
60      FreqBuckets cnt(n, maxC);
61      vector<int> ans(m, -1);
62
63      function<void(int, int)> dfs_addonly = [&](int u, int mul) {
64          cnt.add(c[u], mul);
65          for (auto& v : g[u])
66              dfs_addonly(v, mul);
67      };
68
69      function<void(int)> dfs_solve = [&](int u) {
70          for (auto& v : g[u]) {
71              if (v == g[u][0]) continue;
72              dfs_solve(v);
73              dfs_addonly(v, -1);
74          }
75
76          if (!g[u].empty())
77              dfs_solve(g[u][0]);
78          cnt.add(c[u], +1);
79          for (auto& v : g[u]) {
80              if (v == g[u][0]) continue;
81              dfs_addonly(v, +1);
82          }
```

```
83
84          for (auto& [k, i] : qry[u])
85              ans[i] = (k <= n ? cnt.freq[k] : 0);
86      };
87      dfs_solve(0);
88
89      for (int i = 0; i < m; i++)
90          cout << ans[i] << "\n";
91
92      return 0;
93  }
```

Flow

MaxFlow

```
1   constexpr int inf = 1E9;
2
3   template<class T>
4   struct MaxFlow {
5       struct _Edge {
6           int to;
7           T cap;
8           _Edge(int to, T cap) : to(to), cap(cap) {}
9       };
10
11      int n;
12      std::vector<_Edge> e;
13      std::vector<std::vector<int>> g;
14      std::vector<int> cur, h;
15
16      MaxFlow() {}
17      MaxFlow(int n) {
18          init(n);
19      }
20
21      void init(int n) {
22          this->n = n;
23          e.clear();
24          g.assign(n, {});
25          cur.resize(n);
26          h.resize(n);
27      }
28
29      bool bfs(int s, int t) {
30          h.assign(n, -1);
31          std::queue<int> que;
32          h[s] = 0;
33          que.push(s);
34          while (!que.empty()) {
35              const int u = que.front();
36              que.pop();
37              for (int i : g[u]) {
38                  auto [v, c] = e[i];
39                  if (c > 0 && h[v] == -1) {
40                      h[v] = h[u] + 1;
41                      if (v == t) {
42                          return true;
43                      }
44                      que.push(v);
45                  }
46              }
47          }
48          return false;
49      }
50
51      T dfs(int u, int t, T f) {
52          if (u == t) {
53              return f;
54          }
```

```
55          auto r = f;
56          for (int &i = cur[u]; i < int(g[u].size()); ++i) {
57              const int j = g[u][i];
58              auto [v, c] = e[j];
59              if (c > 0 && h[v] == h[u] + 1) {
60                  auto a = dfs(v, t, std::min(r, c));
61                  e[j].cap -= a;
62                  e[j ^ 1].cap += a;
63                  r -= a;
64                  if (r == 0) {
65                      return f;
66                  }
67              }
68          }
69          return f - r;
70      }
71      void addEdge(int u, int v, T c) {
72          g[u].push_back(e.size());
73          e.emplace_back(v, c);
74          g[v].push_back(e.size());
75          e.emplace_back(u, 0);
76      }
77      T flow(int s, int t) {
78          T ans = 0;
79          while (bfs(s, t)) {
80              cur.assign(n, 0);
81              ans += dfs(s, t, std::numeric_limits<T>::max());
82          }
83          return ans;
84      }
85
86      std::vector<bool> minCut() {
87          std::vector<bool> c(n);
88          for (int i = 0; i < n; i++) {
89              c[i] = (h[i] != -1);
90          }
91          return c;
92      }
93
94      struct Edge {
95          int from;
96          int to;
97          T cap;
98          T flow;
99      };
100     std::vector<Edge> edges() {
101         std::vector<Edge> a;
102         for (int i = 0; i < e.size(); i += 2) {
103             Edge x;
104             x.from = e[i + 1].to;
105             x.to = e[i].to;
106             x.cap = e[i].cap + e[i + 1].cap;
107             x.flow = e[i + 1].cap;
108             a.push_back(x);
109         }
110         return a;
111     }
112 };
```

Min Cost Flow

```
1   constexpr int inf = 1E9;
2
3   template<class T>
4   struct MinCostFlow {
5       struct _Edge {
6           int to;
7           T cap;
8           T cost;
9           _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
10      };
```

```cpp
    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<T> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<T>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            T d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] != d) {
                continue;
            }
            for (int i : g[u]) {
                int v = e[i].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
                    dis[v] = d + h[u] - h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<T>::max();
    }
    MinCostFlow() {}
    MinCostFlow(int n_) {
        init(n_);
    }
    void init(int n_) {
        n = n_;
        e.clear();
        g.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost) {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    std::pair<T, T> flow(int s, int t) {
        T flow = 0;
        T cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) {
                h[i] += dis[i];
            }
            T aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = std::min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return std::make_pair(flow, cost);
    }
    struct Edge {
        int from;
        int to;
        T cap;
```

```
82          T cost;
83          T flow;
84      };
85      std::vector<Edge> edges() {
86          std::vector<Edge> a;
87          for (int i = 0; i < e.size(); i += 2) {
88              Edge x;
89              x.from = e[i + 1].to;
90              x.to = e[i].to;
91              x.cap = e[i].cap + e[i + 1].cap;
92              x.cost = e[i].cost;
93              x.flow = e[i + 1].cap;
94              a.push_back(x);
95          }
96          return a;
97      }
98  };
```

String

KMP

```
1   int main(){
2       string s;
3       cin >> s;
4       int n = (int)s.size();
5       s = " " + s;
6
7       vector<int> nxt(n + 1);
8       for (int i = 2, p = 0; i <= n; i++) {
9           while (p && s[p + 1] != s[i]) p = nxt[p];
10          p = nxt[i] = p + (s[p + 1] == s[i]);
11      }
12
13
14      string t;
15      cin >> t;
16      int m = (int)t.size();
17      t = " " + t;
18      for (int i = 1, p = 0; i <= m; i++) {
19          while (p && s[p + 1] != t[i]) p = nxt[p];
20          p += (s[p + 1] == t[i]);
21          if (p == n) {
22              cout << i - n + 1 << "\n";
23              p = nxt[p];
24          }
25      }
26
27      for (int i = 1; i <= n; i++) {
28          cout << nxt[i] << "\n";
29      }
30
31      return 0;
32  }
```

Trie

```
1   struct Trie {
2       const int S = 26;
3       int N;
4
5       int cnt = 0; // root = 0
6
7       vector<vector<int>> tr;
8       vector<int> pos;
9
10      Trie() : N(0) {}
11      Trie(int n) : N(2 * n + 5), tr(N, vector<int>(S)), pos(N) {}
12
```

```
13        void insert(string s, int id) {
14            int p = 0;
15            for (char it: s) {
16                if (!tr[p][it - 'a']) tr[p][it - 'a'] = ++cnt;
17                p = tr[p][it - 'a'];
18            }
19        }
20    };
```

GSAM

```
1     struct GSAM {
2         const int S = 26;
3         int N;
4
5         vector<vector<int>> son;
6         vector<int> len, fa;
7
8         int cnt = 1; // root = 1
9
10        GSAM() : N(0) {}
11        GSAM(int n) : N(2 * n + 5), son(N, vector<int>(S)), len(N), fa(N) {}
12
13        int insert(int p, int it) {
14            int cur = ++cnt;
15            len[cur] = len[p] + 1;
16            while (!son[p][it]) son[p][it] = cur, p = fa[p];
17            if (!p) return fa[cur] = 1, cur;
18            int q = son[p][it];
19            if (len[p] + 1 == len[q]) return fa[cur] = q, cur;
20            int cl = ++cnt;
21            son[cl] = son[q];
22            len[cl] = len[p] + 1, fa[cl] = fa[q], fa[q] = fa[cur] = cl;
23            while (son[p][it] == q) son[p][it] = cl, p = fa[p];
24            return cur;
25        }
26
27        void build(Trie &T) {
28            queue<int> Q;
29            Q.push(0), T.pos[0] = 1;
30            while (!Q.empty()) {
31                int cur = Q.front();
32                Q.pop();
33                for (int i = 0; i < S; i++) {
34                    int p = T.tr[cur][i];
35                    if (!p) continue;
36                    T.pos[p] = insert(T.pos[cur], i), Q.push(p);
37                }
38            }
39        }
40    };
```