National Taiwan University

Ashes

Chieh-Tu Yu, Tung-Chun Lin, Chan-Yu Yeh

Contents

1 Basic

1.1 Fast Integer Input

```cpp
inline int gtx() {
  const int N = 4096;
  static char buffer[N];
  static char *p = buffer, *end = buffer;
  if (p == end) {
    if ((end = buffer + fread(buffer, 1, N, stdin)) == buffer)
        return EOF;
    p = buffer;
  }
  return *p++;
}
template <typename T>
inline bool rit(T& x) {
  char c = 0; bool flag = false;
  while (c = getchar(), (c < '0' && c != '-') || c > '9') if (c
        == -1) return false;
  c == '-' ? (flag = true, x = 0) : (x = c - '0');
  while (c = getchar(), c >= '0' && c <= '9') x = x * 10 + c -
       '0';
  if (flag) x = -x;
  return true;
}
```

1.2 Increase stack size

```cpp
const int size = 256 << 20;
register long rsp asm("rsp");
char *p = (char*)malloc(size) + size, *bak = (char*)rsp;
__asm__("movq %0, %%rsp\n"::"r"(p));
// main
__asm__("movq %0, %%rsp\n"::"r"(bak));
```

1.3 Default Code

```cpp
#include <bits/stdc++.h>
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")
using namespace std;
#define DBG(x)   cout << (#x " = ") << x << endl;
#define ALL(x) x.begin(), x.end()
#define push_back emplace_back
#define spIO ios::sync_with_stdio(false);cin.tie(0)
#define mem(arr, value) memset(arr, value, sizeof(arr))
#define for0(i, n) for(int i = 0; i < n; i++)
#define for1(i, n) for(int i = 1; i <= n; i++)
typedef long long ll;
typedef long double ld;
const ll mod = 1e9 + 7;
inline ll pow2(ll target, ll p, ll MOD = mod)
{
    ll ret = 1;
    while (p)
    {
        if (p & 1)
            (ret *= target) %= MOD;
        p >>= 1;
        (target *= target) %= MOD;
    }
    return ret;
}
inline ll inv(ll x, ll MOD = mod)
{
    return pow2(x, MOD - 2, MOD);
}
inline ll gcd(ll x, ll y)
{
    if (!y)
        return x;
    if ((x & 1) && (y & 1))
    {
        if (x < y)
            swap(x, y);
        return gcd(y, x - y);
    }
    if (x & 1)
        return gcd(x, y >> 1);
    if (y & 1)
        return gcd(y, x >> 1);
    return 2 * gcd(x >> 1, y >> 1);
}
inline ll sub(ll x, ll y, ll MOD = mod)
{
    return (x - y < 0 ? x - y + MOD : x - y);
}
inline ll add(ll x, ll y, ll MOD = mod)
{
    return (x + y >= MOD ? x + y - MOD : x + y);
}
vector<pair<int, int>> directions = {
     {1,0},{1,1},{0,1},{-1,1},{-1,0},{-1,-1},{0,-1},{1,-1} };
```

1.4 Big Integer

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

struct Int {
  static const int inf = 1e9;
  std::vector<int> dig;
  bool sgn;
  Int() {
    dig.push_back(0);
    sgn = true;
  }
  Int(int n) {
    sgn = n >= 0;
    while (n) {
      dig.push_back(n % 10);
      n /= 10;
    }
    if (dig.size() == 0) dig.push_back(0);
  }
  Int(std::string s) {
    int i = 0; sgn = true;
    if (s[i] == '-') sgn = false, ++i;
    for (;i < s.length(); ++i) dig.push_back(s[i] - '0');
    reverse(dig.begin(), dig.end());
    if (dig.size() == 1 && dig[0] == '0') sgn = true;
  }
  Int(const std::vector<int>& d, const bool& s = true) {
```

```cpp
    dig = std::vector<int>(d.begin(), d.end());
    sgn = s;
  }
  Int(const Int& n) {
    sgn = n.sgn;
    dig = n.dig;
  }
  bool operator<(const Int& rhs) const {
    if (sgn && !rhs.sgn) return true;
    if (!sgn && rhs.sgn) return false;
    if (!sgn && !rhs.sgn) return Int(dig) > Int(rhs.dig);
    if (dig.size() < rhs.dig.size()) return true;
    if (dig.size() > rhs.dig.size()) return false;
    for (int i = dig.size() - 1; i >= 0; --i) {
      if (dig[i] != rhs.dig[i]) return dig[i] < rhs.dig[i];
    }
    return false;
  }
  bool operator==(const Int& rhs) const {
    if (sgn != rhs.sgn) return false;
    return dig == rhs.dig;
  }
  bool operator>(const Int& rhs) const {
    return !(*this < rhs) && !(*this == rhs);
  }
  bool operator<(const int& n) const {
    return *this < Int(n);
  }
  bool operator>(const int& n) const {
    return *this > Int(n);
  }
  bool operator==(const int& n) const {
    return *this == Int(n);
  }
  Int operator-() const {
    return Int(dig, !sgn);
  }
  Int operator+(const Int& rhs) const {
    bool res = true;
    if (!sgn && !rhs.sgn) res = false;
    else if (!sgn && rhs.sgn) return rhs - (-*this);
    else if (sgn && !rhs.sgn) return *this - -rhs;
    std::vector<int> v1 = dig, v2 = rhs.dig;
    if (v2.size() > v1.size()) swap(v1, v2);
    int car = 0;
    std::vector<int> nvec;
    for (int i = 0; i < v2.size(); ++i) {
      int k = v1[i] + v2[i] + car;
      nvec.push_back(k % 10);
      car = k / 10;
    }
    for (int i = v2.size(); i < v1.size(); ++i) {
      int k = v1[i] + car;
      nvec.push_back(k % 10);
      car = k / 10;
    }
    return Int(nvec, res);
  }
  Int operator-(const Int& rhs) const {
    if (*this < rhs) {
      std::vector<int> nvec = (rhs - *this).dig;
      return Int(nvec, false);
    }
    if (*this == rhs) return Int(0);
    std::vector<int> v1 = dig, v2 = rhs.dig;
    std::vector<int> nvec;
    for (int i = 0; i < v2.size(); ++i) {
      int k = v1[i] - v2[i];
      if (k < 0) {
        for (int j = i + 1; j < v1.size(); ++j) if (v1[j] > 0)
            {
          --v1[j]; k += 10;
          break;
        }
      }
      nvec.push_back(k);
    }
    int rind = v1.size() - 1;
    while (rind >= v2.size() && v1[rind] == 0) --rind;
    for (int i = v2.size(); i <= rind; ++i) {
      nvec.push_back(v1[i]);
    }
    return Int(nvec);
  }
  Int operator*(const Int& rhs) const {
    if (sgn && !rhs.sgn || !sgn && rhs.sgn) return -(Int(dig,
        true) * Int(rhs.dig, true));
    if (*this == 0) return Int();
    if (rhs == 0) return Int();
    std::vector<int> v1 = dig, v2 = rhs.dig;
    if (v1.size() < v2.size()) swap(v1, v2);
    std::vector<int> res(v1.size() * v2.size(), 0);
    for (int i = 0; i < v2.size(); ++i) {
      int car = 0;
      for (int j = 0; j < v1.size(); ++j) {
        int k = car + v1[j] * v2[i];
        res[j + i] += k % 10;
        car = k / 10;
      }
    }
    int car = 0;
    for (int i = 0; i < res.size(); ++i) {
      int k = car + res[i];
      res[i] = k % 10;
      car = k / 10;
    }
    while (car) {
      res.push_back(car % 10);
      car /= 10;
    }
    int ind = res.size() - 1;
    while (ind >= 0 && res[ind] == 0) --ind;
    std::vector<int> nvec;
    for (int i = 0; i <= ind; ++i) nvec.push_back(res[i]);
    return Int(nvec);
  }
  Int operator+(const int& n) const {
    return *this + Int(n);
  }
  Int operator-(const int& n) const {
    return *this - Int(n);
  }
  Int& operator+=(const Int& n) {
    *this = (*this + n);
    return *this;
  }
  Int& operator-=(const Int& n) {
    *this = (*this - n);
    return *this;
  }
  Int& operator+=(const int& n) {
    *this += Int(n);
    return *this;
  }
  Int& operator-=(const int& n) {
    *this -= Int(n);
    return *this;
  }
  Int& operator*=(const Int& n) {
    *this = *this * n;
    return *this;
  }
  Int& operator*=(const int& n) {
    *this *= Int(n);
    return *this;
  }
  Int& operator++(int) {
    *this += 1;
    return *this;
  }
  Int& operator--(int) {
    *this -= 1;
    return *this;
  }
  friend std::istream& operator>>(std::istream& in, Int& n) {
    std::string s; in >> s;
    n = Int(s);
    return in;
  }
  friend std::ostream& operator<<(std::ostream& out, const Int&
      n) {
    if (!n.sgn) out << "-";
    for (int i = n.dig.size() - 1; i >= 0; --i) out << n.dig[i
        ];
    return out;
  }
};
```

2 Flows, Matching

2.1 Dinic's Algorithm

```
struct Edge {
  int to, cap, rev;
  Edge(int a, int b, int c) : to(a), cap(b), rev(c) {}
};
int Flow(vector<vector<Edge>> &g, int s, int t){
  int n = g.size(), res = 0;
  vector<int> lev(n, -1), iter(n);
  while(true){
    vector<int> que(1, s);
    fill(lev.begin(), lev.end(), -1);
    fill(iter.begin(), iter.end(), 0);
    lev[s] = 0;
    for (int i = 0; i < (int)que.size(); i++){
      int x = que[i];
      for (Edge &e : g[x]){
        if (e.cap > 0 && lev[e.to] == -1){
          lev[e.to] = lev[x] + 1;
          que.push_back(e.to);
        }
      }
    }
    if (lev[t] == -1) break;
    auto Dfs = [&](auto dfs, int x, int f = 1e9){
      if (x == t) return f;
      int res = 0;
      for (int &i = iter[x]; i < (int)g[x].size(); i++){
        Edge &e = g[x][i];
        if (e.cap > 0 && lev[e.to] == lev[x] + 1){
          int p = dfs(dfs, e.to, min(f - res, e.cap));
          res += p;
          e.cap -= p;
          g[e.to][e.rev].cap += p;
        }
      }
      if (res == 0) lev[x] = -1;
      return res;
    };
    res += Dfs(Dfs, s);
  }
  return res;
}
auto Add = [&](int a, int b, int c){
  g[a].emplace_back(b, c, (int)g[b].size());
  g[b].emplace_back(a, 0, (int)g[a].size()-1);
};
```

2.2 Minimum Cost Maximum Flow

```
struct Edge {
  int to, cap, rev, w;
  Edge(int t, int c, int r, int w) : to(t), cap(c), rev(r), w(w
      ) {}
};
pair<int, int> Flow(vector<vector<Edge>> g, int s, int t) {
  int N = g.size();
  vector<int> dist(N), ed(N), pv(N);
  vector<bool> inque(N);
  int flow = 0, cost = 0;
  while (true) {
    dist.assign(N, kInf);
    inque.assign(N, false);
    pv.assign(N, -1);
    dist[s] = 0;
    queue<int> que;
    que.push(s);
    while (!que.empty()) {
      int x = que.front(); que.pop();
      inque[x] = false;
      for (int i = 0; i < g[x].size(); ++i) {
        Edge &e = g[x][i];
        if (e.cap > 0 && dist[e.to] > dist[x] + e.w) {
          dist[e.to] = dist[x] + e.w;
          pv[e.to] = x;
          ed[e.to] = i;
          if (!inque[e.to]) {
            inque[e.to] = true;
            que.push(e.to);
          }
        }
      }
    }
    if (dist[t] == kInf) break;
    int f = kInf;
    for (int x = t; x != s; x = pv[x]) f = min(f, g[pv[x]][ed[x
        ]].cap);
    for (int x = t; x != s; x = pv[x]) {
      Edge &e = g[pv[x]][ed[x]];
      e.cap -= f;
      g[e.to][e.rev].cap += f;
    }
    flow += f;
    cost += f * dist[t];
  }
  return make_pair(flow, cost);
}
auto AddEdge = [&](int from, int to, int cap, int weight) {
  g[from].emplace_back(to, cap, g[to].size(), weight);
  g[to].emplace_back(from, 0, g[from].size() - 1, -weight);
};
```

2.3 Bipartite Matching

```
class matching {
 public:
  vector<vector<int>> g;
  vector<int> pa;
  vector<int> pb;
  vector<int> was;
  int n, m;
  int res;
  int iter;

  matching(int _n, int _m) : n(_n), m(_m) {
    assert(0 <= n && 0 <= m);
    pa = vector<int>(n, -1);
    pb = vector<int>(m, -1);
    was = vector<int>(n, 0);
    g.resize(n);
    res = 0;
    iter = 0;
  }

  void add(int from, int to) {
    assert(0 <= from && from < n && 0 <= to && to < m);
    g[from].push_back(to);
  }

  bool dfs(int v) {
    was[v] = iter;
    for (int u : g[v]) {
      if (pb[u] == -1) {
        pa[v] = u;
        pb[u] = v;
        return true;
      }
    }
    for (int u : g[v]) {
      if (was[pb[u]] != iter && dfs(pb[u])) {
        pa[v] = u;
        pb[u] = v;
        return true;
      }
    }
    return false;
  }

  int solve() {
    while (true) {
      iter++;
      int add = 0;
      for (int i = 0; i < n; i++) {
        if (pa[i] == -1 && dfs(i)) {
          add++;
        }
      }
      if (add == 0) {
        break;
      }
      res += add;
    }
    return res;
  }

  int run_one(int v) {
    if (pa[v] != -1) {
      return 0;
    }
    iter++;
    return (int) dfs(v);
  }
};
```

2.4 Weighted Bipartite Match

```
const int kInf = 1e9;
long long KuhnMunkres(vector<vector<int>> W) {
  int N = W.size();
  vector<int> fl(N, -1), fr(N, -1), hr(N), hl(N);
  for (int i = 0; i < N; ++i) {
    hl[i] = *max_element(W[i].begin(), W[i].end());
  }
  auto Bfs = [&](int s) {
    vector<int> slk(N, kInf), pre(N);
    vector<bool> vl(N, false), vr(N, false);
    queue<int> que;
    que.push(s);
    vr[s] = true;
    auto Check = [&](int x) -> bool {
      if (vl[x] = true, fl[x] != -1) {
        que.push(fl[x]);
        return vr[fl[x]] = true;
      }
      while (x != -1) swap(x, fr[fl[x] = pre[x]]);
      return false;
    };
    while (true) {
      while (!que.empty()) {
        int y = que.front(); que.pop();
        for (int x = 0, d = 0; x < N; ++x) {
          if (!vl[x] && slk[x] >= (d = hl[x] + hr[y] - W[x][y])
              ) {
            if (pre[x] = y, d) slk[x] = d;
            else if (!Check(x)) return;
          }
        }
      }
      int d = kInf;
      for (int x = 0; x < N; ++x) {
        if (!vl[x] && d > slk[x]) d = slk[x];
      }
      for (int x = 0; x < N; ++x) {
        if (vl[x]) hl[x] += d;
        else slk[x] -= d;
        if (vr[x]) hr[x] -= d;
      }
      for (int x = 0; x < N; ++x) {
        if (!vl[x] && !slk[x] && !Check(x)) return;
      }
    }
  };
  for (int i = 0; i < N; ++i) Bfs(i);
  long long res = 0;
  for (int i = 0; i < N; ++i) res += W[i][fl[i]];
  return res;
}
```

2.5 Maximum Matching on General Graph

```
const int kN = 500;
namespace matching {
  int fa[kN], pre[kN], match[kN], s[kN], v[kN];
  vector<int> g[kN];
  queue<int> q;
  void Init(int n) {
    for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
    for (int i = 0; i < n; ++i) g[i].clear();
  }
  void AddEdge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
  }
  int Find(int u) {
    return u == fa[u] ? u : fa[u] = Find(fa[u]);
  }
  int LCA(int x, int y, int n) {
    static int tk = 0;
    tk++;
    x = Find(x), y = Find(y);
    for (; ; swap(x, y)) {
      if (x != n) {
        if (v[x] == tk) return x;
        v[x] = tk;
        x = Find(pre[match[x]]);
      }
    }
  }
  void Blossom(int x, int y, int l) {
    while (Find(x) != l) {
      pre[x] = y, y = match[x];
      if (s[y] == 1) q.push(y), s[y] = 0;
      if (fa[x] == x) fa[x] = l;
      if (fa[y] == y) fa[y] = l;
      x = pre[y];
    }
  }
  bool Bfs(int r, int n) {
    for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
    while (!q.empty()) q.pop();
    q.push(r);
    s[r] = 0;
    while (!q.empty()) {
      int x = q.front(); q.pop();
      for (int u : g[x]) {
        if (s[u] == -1) {
          pre[u] = x, s[u] = 1;
          if (match[u] == n) {
            for (int a = u, b = x, last; b != n; a = last, b =
                pre[a])
              last = match[b], match[b] = a, match[a] = b;
            return true;
          }
          q.push(match[u]);
          s[match[u]] = 0;
        } else if (!s[u] && Find(u) != Find(x)) {
          int l = LCA(u, x, n);
          Blossom(x, u, l);
          Blossom(u, x, l);
        }
      }
    }
    return false;
  }
  int Solve(int n) {
    int res = 0;
    for (int x = 0; x < n; ++x) {
      if (match[x] == n) res += Bfs(x, n);
    }
    return res;
  }
}
```

2.6 Flow Models

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x, y, l, u), connect $x \to y$ with capacity $u - l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
  1. Redirect every edge: $y \to x$ if $(x, y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u, v, w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u, v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.
- Project selection problem
  1. If $p_v > 0$, create edge (s, v) with capacity p_v; otherwise, create edge (v, t) with capacity $-p_v$.
  2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.

- 0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x\bar{y} + \sum_{xyx'y'} c_{xyx'y'}(x\bar{y} + x'\bar{y'})$$

can be minimized by the mincut of the following graph:

1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y.
2. Create edge (x, y) with capacity c_{xy}.
3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3 Data Structure

3.1 Disjoint Set Union

```
struct DSU {
  vector<int> p, sz;
  int n;
  vector<pair<int, int>> ops;
  int t;

  DSU(int _n): n(_n) {
    p.resize(n);
    sz.resize(n);
    for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
    t = 0;
  }
  int find(int x) {
    if (p[x] == x) return x;
    else return p[x] = find(p[x]);
  }
  void unite(int x, int y) {
    x = find(x); y = find(y);
    if (x == y) return;
    if (sz[x] > sz[y]) swap(x, y);

    ops.emplace_back(x, p[x]);
    p[x] = y;
    ops.emplace_back(~y, sz[y]);
    sz[y] += sz[x];

  }
  void save(){
    t = ops.size();
  }
  void rollback(){
    while ((int)ops.size() > t){
      int i = ops.back().first;
      int j = ops.back().second;
      ops.pop_back();
      if (i >= 0){
        p[i] = j;
      }else{
        sz[~i] = j;
      }
    }
  }
};
```

3.2 Segment Tree

```
template<class T> struct Seg { // comb(ID,b) = b
  const T ID = 1e18; T comb(T a, T b) { return min(a,b); }
  int n; vector<T> seg;
  void init(int _n) { n = _n; seg.assign(2*n,ID); }
  void pull(int p) { seg[p] = comb(seg[2*p],seg[2*p+1]); }
  void upd(int p, T val) { // set val at position p
    seg[p += n] = val; for (p /= 2; p; p /= 2) pull(p); }
  T query(int l, int r) { // min on interval [l, r]
    T ra = ID, rb = ID;
    for (l += n, r += n+1; l < r; l /= 2, r /= 2) {
      if (l&1) ra = comb(ra,seg[l++]);
      if (r&1) rb = comb(seg[--r],rb);
    }
    return comb(ra,rb);
  }
};
```

3.3 Lazy Segment Tree

```
const int maxN = 2e5+5;
int a[maxN];

struct node{
  ll val;
  ll lzAdd;
  ll lzSet;
  node(){};
}tree[maxN<<2];

#define lc p<<1
#define rc (p<<1)+1

inline void pushup(int p){
  tree[p].val = tree[lc].val + tree[rc].val;
  return;
}

void pushdown(int p, int l, int mid, int r){
  //lazy: range set
  if(tree[p].lzSet != 0){
    tree[lc].lzSet = tree[rc].lzSet = tree[p].lzSet;
    tree[lc].val = (mid-l+1) * tree[p].lzSet;
    tree[rc].val = (r-mid) * tree[p].lzSet;
    tree[lc].lzAdd = tree[rc].lzAdd = 0;
    tree[p].lzSet = 0;
  }else if(tree[p].lzAdd != 0){ //lazy: range add
    if(tree[lc].lzSet == 0)tree[lc].lzAdd += tree[p].lzAdd;
    else {
      tree[lc].lzSet += tree[p].lzAdd;
      tree[lc].lzAdd = 0;
    }
    if(tree[rc].lzSet == 0)tree[rc].lzAdd += tree[p].lzAdd;
    else{
      tree[rc].lzSet += tree[p].lzAdd;
      tree[rc].lzAdd = 0;
    }
    tree[lc].val += (mid-l+1) * tree[p].lzAdd;
    tree[rc].val += (r-mid) * tree[p].lzAdd;
    tree[p].lzAdd = 0;
  }
  return;
}

void build(int p, int l, int r){
  tree[p].lzAdd = tree[p].lzSet = 0;
  if(l == r){
    tree[p].val = a[l];
    return;
  }
  int mid = (l+r)>>1;
  build(lc,l,mid);
  build(rc,mid+1,r);
  pushup(p);
  return;
}

void add(int p, int l, int r, int a, int b, ll val){
  if(a > r || b < l)return;
  if(a <= l && r <= b){
    tree[p].val += (r-l+1) * val;
    if(tree[p].lzSet == 0)tree[p].lzAdd += val;
    else tree[p].lzSet += val;
    return;
  }
  int mid = (l+r)>>1;
  pushdown(p,l,mid,r);
  add(lc,l,mid,a,b,val);
  add(rc,mid+1,r,a,b,val);
  pushup(p);
  return;
}

void st(int p, int l, int r, int a, int b, ll val){
  if(a > r || b < l)return;
  if(a <= l && r <= b){
    tree[p].val = (r-l+1) * val;
    tree[p].lzAdd = 0;
    tree[p].lzSet = val;
    return;
  }
  int mid = (l+r)>>1;
  pushdown(p,l,mid,r);
  st(lc,l,mid,a,b,val);
  st(rc,mid+1,r,a,b,val);
  pushup(p);
  return;
}

ll query(int p, int l, int r, int a, int b){
  if(a > r || b < l)return 0;
  if(a <= l && r <= b)return tree[p].val;
  int mid = (l+r)>>1;
  pushdown(p,l,mid,r);
```

```
  return query(lc,l,mid,a,b) + query(rc,mid+1,r,a,b);
}
```

3.4 Treap

```
typedef struct item * pitem;
struct item {
  int prior, value, cnt;
  bool rev;
  pitem l, r;
};

int cnt (pitem it) {
  return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
  if (it)
    it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void push (pitem it) {
  if (it && it->rev) {
    it->rev = false;
    swap (it->l, it->r);
    if (it->l)  it->l->rev ^= true;
    if (it->r)  it->r->rev ^= true;
  }
}

void merge (pitem & t, pitem l, pitem r) {
  push (l);
  push (r);
  if (!l || !r)
    t = l ? l : r;
  else if (l->prior > r->prior)
    merge (l->r, l->r, r),  t = l;
  else
    merge (r->l, l, r->l),  t = r;
  upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add =
     0) {
  if (!t)
    return void( l = r = 0 );
  push (t);
  int cur_key = add + cnt(t->l);
  if (key <= cur_key)
    split (t->l, l, t->l, key, add),  r = t;
  else
    split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
  upd_cnt (t);
}

void reverse (pitem t, int l, int r) {
  pitem t1, t2, t3;
  split (t, t1, t2, l);
  split (t2, t2, t3, r-l+1);
  t2->rev ^= true;
  merge (t, t1, t2);
  merge (t, t, t3);
}

void output (pitem t) {
  if (!t)  return;
  push (t);
  output (t->l);
  printf ("%d ", t->value);
  output (t->r);
}
```

3.5 LiChaoTree

```
const int maxc = 1e6 + 10;
namespace lichao {
  struct line {
    long long a, b;
    line() : a(0), b(0) {}
    line(long long a, long long b) : a(a), b(b) {}
    long long operator()(int x) const { return a * x + b; }
  };
  line st[maxc * 4];
  int sz;
  void init() {
    sz = 0;
  }
  void add(int l, int r, line tl, int o) {
    bool lcp = st[o](l) > tl(l);
    bool mcp = st[o]((l + r) / 2) > tl((l + r) / 2);
    if (mcp) swap(st[o], tl);
    if (r - l == 1) return;
    if (lcp != mcp) {
      add(l, (l + r) / 2, tl, 2 * o + 1);
    }
    else {
      add((l + r) / 2, r, tl, 2 * o + 2);;
    }
  }
  long long query(int l, int r, int x, int o) {
    if (r - l == 1) return st[o](x);
    if (x < (l + r) / 2) {
      return min(st[o](x), query(l, (l + r) / 2, x, 2 * o + 1))
          ;
    }
    else {
      return min(st[o](x), query((l + r) / 2, r, x, 2 * o + 2))
          ;
    }
  }
}
```

4 Graph

4.1 Bi-Connected Component

```
vector<int> dfn(n), low(n);
int timer = 0, bcc = 0;
vector<int> id(n);
stack<int> stk;
function<void(int, int)> tarjan = [&](int u, int fa){
    dfn[u] = low[u] = timer++;
    stk.push(u);
    for (auto e:adj[u]){
        int v = e.first;
        int w = e.second;
        if (w == fa) continue;
        if (!dfn[v]){
            tarjan(v, w);
            low[u] = min(low[u], low[v]);
        }else{
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (low[u] == dfn[u]){
        while (true) {
            int v = stk.top();
      stk.pop();
            id[v] = bcc;
            if (v == u) break;
        }
        bcc++;
    }
};
```

4.2 Strongly Connected Component

```
vector<int> dfn(n), low(n), ins(n);
int timer = 0, scc = 0;
vector<int> id(n);
stack<int> stk;
void tarjan(int u){
  low[u] = dfn[u] = ++timer;
  ins[u] = 1;
  stk.push(u);
  for (int v:adj[u]){
    if (!dfn[v]){
      tarjan(v);
      low[u] = min(low[u], low[v]);
    }else if (ins[v]){
      low[u] = min(low[u], dfn[v]);
    }
  }
  if (low[u] == dfn[u]){
    int v;
    do {
      v = stk.top(); stk.pop();
      id[v] = scc;
      ins[v] = 0;
    } while(v != u);
    scc++;
  }
}
```

4.3 Lowest Common Ancestor

```
const int mxN = 1e5+5, LOG = 18;
vector<int> adj[mxN];
int dep[mxN], up[mxN][LOG];
void dfs(int u, int p){
  for (int v:adj[u]){
    if (v == p)
      continue;
    dep[v] = dep[u] + 1;
    up[v][0] = u;
    for (int j = 1; j < LOG; j++){
      up[v][j] = up[up[v][j - 1]][j - 1];
    }
    dfs(v, u);
  }
}
int lca(int u, int v){
  if(dep[u] < dep[v])
    swap(u, v);
  int k = dep[u] - dep[v];
  for(int j = LOG - 1; j >= 0; j--) {
    if(k & (1 << j)) {
      u = up[u][j];
    }
  }
  if(u == v) {
    return u;
  }
  for(int j = LOG - 1; j >= 0; j--) {
    if(up[u][j] != up[v][j]) {
      u = up[u][j];
      v = up[v][j];
    }
  }
  return up[u][0];
}
```

5 String

5.1 Knuth–Morris–Pratt Algorithm

```
vector<int> Failure(const string &s) {
  vector<int> f(s.size(), 0);
  // f[i] = length of the longest prefix (excluding s[0:i])
      such that it coincides with the suffix of s[0:i] of the
      same length
  // i + 1 - f[i] is the length of the smallest recurring
      period of s[0:i]
  int k = 0;
  for (int i = 1; i < (int)s.size(); ++i) {
    while (k > 0 && s[i] != s[k]) k = f[k - 1];
    if (s[i] == s[k]) ++k;
    f[i] = k;
  }
  return f;
}
vector<int> Search(const string &s, const string &t) {
  // return 0-indexed occurrence of t in s
  vector<int> f = Failure(t), res;
  for (int i = 0, k = 0; i < (int)s.size(); ++i) {
    while (k > 0 && (k == (int)t.size() || s[i] != t[k])) k = f
        [k - 1];
    if (s[i] == t[k]) ++k;
    if (k == (int)t.size()) res.push_back(i - t.size() + 1);
  }
  return res;
}
```

5.2 Z Algorithm

```
int z[maxn];
// z[i] = LCP of suffix i and suffix 0
void z_function(const string& s) {
  memset(z, 0, sizeof(z));
  z[0] = (int)s.length();
  int l = 0, r = 0;
  for (int i = 1; i < s.length(); ++i) {
    z[i] = max(0, min(z[i - l], r - i + 1));
    while (i + z[i] < s.length() && s[z[i]] == s[i + z[i]]) {
      l = i; r = i + z[i];
      ++z[i];
    }
  }
}
```

5.3 Suffix Automaton

```
struct SAM {
  static const int maxn = 5e5 + 5;
  int nxt[maxn][26], to[maxn], len[maxn];
  int root, last, sz;
  int gnode(int x) {
    for (int i = 0; i < 26; ++i) nxt[sz][i] = -1;
    to[sz] = -1;
    len[sz] = x;
    return sz++;
  }
  void init() {
    sz = 0;
    root = gnode(0);
    last = root;
  }
  void push(int c) {
    int cur = last;
    last = gnode(len[last] + 1);
    for (; ~cur && nxt[cur][c] == -1; cur = to[cur]) nxt[cur][c
        ] = last;
    if (cur == -1) return to[last] = root, void();
    int link = nxt[cur][c];
    if (len[link] == len[cur] + 1) return to[last] = link, void
        ();
    int tlink = gnode(len[cur] + 1);
    for (; ~cur && nxt[cur][c] == link; cur = to[cur]) nxt[cur
        ][c] = tlink;
    for (int i = 0; i < 26; ++i) nxt[tlink][i] = nxt[link][i];
    to[tlink] = to[link];
    to[link] = tlink;
    to[last] = tlink;
  }
  void add(const string &s) {
    for (int i = 0; i < s.size(); ++i) push(s[i] - 'a');
  }
  bool find(const string &s) {
    int cur = root;
    for (int i = 0; i < s.size(); ++i) {
      cur = nxt[cur][s[i] - 'a'];
      if (cur == -1) return false;
    }
    return true;
  }
  int solve(const string &t) {
    int res = 0, cnt = 0;
    int cur = root;
    for (int i = 0; i < t.size(); ++i) {
      if (~nxt[cur][t[i] - 'a']) {
        ++cnt;
        cur = nxt[cur][t[i] - 'a'];
      } else {
        for (; ~cur && nxt[cur][t[i] - 'a'] == -1; cur = to[cur
            ]);
        if (~cur) cnt = len[cur] + 1, cur = nxt[cur][t[i] - 'a'
            ];
        else cnt = 0, cur = root;
      }
      res = max(res, cnt);
    }
    return res;
  }
};
```

5.4 Suffix Array

```
// sa[i]: sa[i]-th suffix is the i-th lexigraphically smallest
    suffix.
// lcp[i]: longest common prefix of suffix sa[i] and suffix sa[
    i - 1].
namespace sfx {
vector<int> Build(const string &s) {
  int n = s.size();
  vector<int> str(n * 2), sa(n * 2), c(max(n, 256) * 2), x(max(
      n, 256)), p(n), q(n * 2), t(n * 2);
  for (int i = 0; i < n; ++i) str[i] = s[i];
  auto Pre = [&](int *sa, int *c, int n, int z) {
    memset(sa, 0, sizeof(int) * n);
    memcpy(x.data(), c, sizeof(int) * z);
  };
  auto Induce = [&](int *sa, int *c, int *s, int *t, int n, int
       z) {
    memcpy(x.data() + 1, c, sizeof(int) * (z - 1));
    for (int i = 0; i < n; ++i) if (sa[i] && !t[sa[i] - 1]) sa[
        x[s[sa[i] - 1]]++] = sa[i] - 1;
    memcpy(x.data(), c, sizeof(int) * z);
```

```
    for (int i = n - 1; i >= 0; --i) if (sa[i] && t[sa[i] - 1])
        sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
  };
  auto SAIS = [&](auto self, int *s, int *sa, int *p, int *q,
      int *t, int *c, int n, int z) -> void {
    bool uniq = t[n - 1] = true;
    int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n, last =
        -1;
    memset(c, 0, sizeof(int) * z);
    for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
    for (int i = 0; i < z - 1; ++i) c[i + 1] += c[i];
    if (uniq) {
      for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
      return;
    }
    for (int i = n - 2; i >= 0; --i) t[i] = (s[i] == s[i + 1] ?
        t[i + 1] : s[i] < s[i + 1]);
    Pre(sa, c, n, z);
    for (int i = 1; i <= n - 1; ++i) if (t[i] && !t[i - 1]) sa
        [--x[s[i]]] = p[q[i] = nn++] = i;
    Induce(sa, c, s, t, n, z);
    for (int i = 0; i < n; ++i) if (sa[i] && t[sa[i]] && !t[sa[
        i] - 1]) {
      bool neq = last < 0 || memcmp(s + sa[i], s + last, (p[q[
          sa[i]] + 1] - sa[i]) * sizeof(int));
      ns[q[last = sa[i]]] = nmxz += neq;
    }
    self(self, ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz +
        1);
    Pre(sa, c, n, z);
    for (int i = nn - 1; i >= 0; --i) sa[--x[s[p[nsa[i]]]]] = p
        [nsa[i]];
    Induce(sa, c, s, t, n, z);
  };
  SAIS(SAIS, str.data(), sa.data(), p.data(), q.data(), t.data
      (), c.data(), n + 1, 256);
  return vector<int>(sa.begin() + 1, sa.begin() + n + 1);
}
vector<int> BuildLCP(const vector<int> &sa, const string &s) {
  int n = s.size();
  vector<int> lcp(n), rev(n);
  for (int i = 0; i < n; ++i) rev[sa[i]] = i;
  for (int i = 0, ptr = 0; i < n; ++i) {
    if (!rev[i]) {
      ptr = 0;
      continue;
    }
    while (i + ptr < n && s[i + ptr] == s[sa[rev[i] - 1] + ptr
        ]) ptr++;
    lcp[rev[i]] = ptr ? ptr-- : 0;
  }
  return lcp;
}}  // namespace sfx
```

6 Math

6.1 Compute Primes

```
void compPrimes(int N) {
  vector<int> primes, leastFac;
  for (int i = 0; i < N; i++) {
    leastFac.push_back(0);
  }
  leastFac[0] = 1; leastFac[1] = 1;
  for (int i = 2; i < N; i++) {
    if (leastFac[i] == 0) {
      primes.push_back(i);
      leastFac[i] = i;
    }
    for (int j = 0; j < primes.size() && i*primes[j] < N &&
        primes[j] <= leastFac[i]; j++) {
      leastFac[i*primes[j]] = primes[j];
    }
  }
}
```

6.2 Extended GCD

```
template <typename T> tuple<T, T, T> extgcd(T a, T b) {
  if (!b) return make_tuple(a, 1, 0);
  T d, x, y;
  tie(d, x, y) = extgcd(b, a % b);
  return make_tuple(d, y, x - (a / b) * y);
}
```

6.3 Primitive Root

```
int powmod (int a, int b, int p) {
  int res = 1;
  while (b)
    if (b & 1)
      res = int (res * 1ll * a % p),  --b;
    else
      a = int (a * 1ll * a % p),  b >>= 1;
  return res;
}
int generator (int p) {
  vector<int> fact;
  int phi = p-1,  n = phi;
  for (int i=2; i*i<=n; ++i)
    if (n % i == 0) {
      fact.push_back (i);
      while (n % i == 0)
        n /= i;
    }
  if (n > 1)
    fact.push_back (n);
  for (int res=2; res<=p; ++res) {
    bool ok = true;
    for (size_t i=0; i<fact.size() && ok; ++i)
      ok &= powmod (res, phi / fact[i], p) != 1;
    if (ok)  return res;
  }
  return -1;
}
```

6.4 Discrete Logarithm

```
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
  a %= m, b %= m;
  int k = 1, add = 0, g;
  while ((g = gcd(a, m)) > 1) {
    if (b == k)
      return add;
    if (b % g)
      return -1;
    b /= g, m /= g, ++add;
    k = (k * 1ll * a / g) % m;
  }

  int n = sqrt(m) + 1;
  int an = 1;
  for (int i = 0; i < n; ++i)
    an = (an * 1ll * a) % m;

  unordered_map<int, int> vals;
  for (int q = 0, cur = b; q <= n; ++q) {
    vals[cur] = q;
    cur = (cur * 1ll * a) % m;
  }

  for (int p = 1, cur = k; p <= n; ++p) {
    cur = (cur * 1ll * an) % m;
    if (vals.count(cur)) {
      int ans = n * p - vals[cur] + add;
      return ans;
    }
  }
  return -1;
}
```

7 Geometry

7.1 Basic

```
bool same(double a, double b) { return abs(a - b) < eps; }
```

```
struct P {
  double x, y;
  P() : x(0), y(0) {}
  P(double x, double y) : x(x), y(y) {}
  P operator + (P b) { return P(x + b.x, y + b.y); }
  P operator - (P b) { return P(x - b.x, y - b.y); }
  P operator * (double b) { return P(x * b, y * b); }
  P operator / (double b) { return P(x / b, y / b); }
  double operator * (P b) { return x * b.x + y * b.y; }
  double operator ^ (P b) { return x * b.y - y * b.x; }
  double abs() { return hypot(x, y); }
  P unit() { return *this / abs(); }
  P rot(double o) {
    double c = cos(o), s = sin(o);
```

```cpp
    return P(c * x - s * y, s * x + c * y);
  }
  double angle() { return atan2(y, x); }
};

struct L {
  // ax + by + c = 0
  double a, b, c, o;
  P pa, pb;
  L() : a(0), b(0), c(0), o(0), pa(), pb() {}
  L(P pa, P pb) : a(pa.y - pb.y), b(pb.x - pa.x), c(pa ^ pb), o
       (atan2(-a, b)), pa(pa), pb(pb) {}
  P project(P p) { return pa + (pb - pa).unit() * ((pb - pa) *
       (p - pa) / (pb - pa).abs()); }
  P reflect(P p) { return p + (project(p) - p) * 2; }
  double get_ratio(P p) { return (p - pa) * (pb - pa) / ((pb -
       pa).abs() * (pb - pa).abs()); }
};

bool SegmentIntersect(P p1, P p2, P p3, P p4) {
  if (max(p1.x, p2.x) < min(p3.x, p4.x) || max(p3.x, p4.x) <
       min(p1.x, p2.x)) return false;
  if (max(p1.y, p2.y) < min(p3.y, p4.y) || max(p3.y, p4.y) <
       min(p1.y, p2.y)) return false;
  return sign((p3 - p1) ^ (p4 - p1)) * sign((p3 - p2) ^ (p4 -
       p2)) <= 0 &&
    sign((p1 - p3) ^ (p2 - p3)) * sign((p1 - p4) ^ (p2 - p4))
         <= 0;
}

bool parallel(L x, L y) { return same(x.a * y.b, x.b * y.a); }

P Intersect(L x, L y) { return P(-x.b * y.c + x.c * y.b, x.a *
     y.c - x.c * y.a) / (-x.a * y.b + x.b * y.a); }
```

7.2 Half Plane Intersection

```cpp
bool jizz(L l1,L l2,L l3){
  P p=Intersect(l2,l3);
  return ((l1.pb-l1.pa)^(p-l1.pa))<-eps;
}

bool cmp(const L &a,const L &b){
  return same(a.o,b.o)?(((b.pb-b.pa)^(a.pb-b.pa))>eps):a.o<b.o;
}

// availble area for L l is (l.pb-l.pa)^(p-l.pa)>0
vector<P> HPI(vector<L> &ls){
  sort(ls.begin(),ls.end(),cmp);
  vector<L> pls(1,ls[0]);
  for(int i=0;i<(int)ls.size();++i)if(!same(ls[i].o,pls.back().
       o))pls.push_back(ls[i]);
  deque<int> dq; dq.push_back(0); dq.push_back(1);
#define meow(a,b,c) while(dq.size()>1u && jizz(pls[a],pls[b],
     pls[c]))
  for(int i=2;i<(int)pls.size();++i){
    meow(i,dq.back(),dq[dq.size()-2])dq.pop_back();
    meow(i,dq[0],dq[1])dq.pop_front();
    dq.push_back(i);
  }
  meow(dq.front(),dq.back(),dq[dq.size()-2])dq.pop_back();
  meow(dq.back(),dq[0],dq[1])dq.pop_front();
  if(dq.size()<3u)return vector<P>(); // no solution or
       solution is not a convex
  vector<P> rt;
  for(int i=0;i<(int)dq.size();++i)rt.push_back(Intersect(pls[
       dq[i]],pls[dq[(i+1)%dq.size()]]));
  return rt;
}
```

7.3 Slope & Fraction

```cpp
struct P {
  int x, y, i;
  P() : x(0), y(0), i(-1) {}
};
struct Frac {
  int u, d;
  void norm() {
    if (d == 0) {
      u = u > 0 ? 1 : u < 0 ? -1 : 0;
      return;
    }
    int g = __gcd(u, d);
    u /= g;
    d /= g;
    if (d < 0) {
      d *= -1;
      u *= -1;
    }
  }
};
bool operator > (const Frac &a, const Frac &b) {
  return 1ll * a.u * b.d > 1ll * b.u * a.d;
}
bool operator >= (const Frac &a, const Frac &b) {
  return 1ll * a.u * b.d >= 1ll * b.u * a.d;
}
bool operator < (const Frac &a, const Frac &b) {
  return 1ll * a.u * b.d < 1ll * b.u * a.d;
}
bool operator <= (const Frac &a, const Frac &b) {
  return 1ll * a.u * b.d <= 1ll * b.u * a.d;
}
ostream& operator << (ostream &o, const Frac &f) {
  o << f.u << "/" << f.d;
  return o;
}

Frac Slope(P &a, P &b) {
  Frac f;
  f.u = b.y - a.y;
  f.d = b.x - a.x;
  f.norm();
  return f;
}
```

7.4 Convex order

```cpp
int quard(P p) {
    if (p.x > 0 && p.y >= 0) return 1;
    if (p.x <= 0 && p.y > 0) return 2;
    if (p.x < 0 && p.y <= 0) return 3;
    if (p.x >= 0 && p.y < 0) return 4;
    return -1;
}
P getcenter(vector<P>& p) {
    P res(0, 0); double n = (double)p.size();
    for (P it : p) res.x += it.x, res.y += it.y;
    res.x /= n; res.y /= n;
    return res;
}
void convex_order(vector<P>& p) {
    P center = getcenter(p);
    auto cmp = [&](P a, P b) {
        P tmpa = a - center,tmpb = b - center;
        int qa = quard(tmpa), qb = quard(tmpb);
        if (qa != qb) return qa < qb;
        return (tmpa ^ tmpb) > 0;
    };
    sort(ALL(p), cmp);
}
```

7.5 Area

```cpp
double area(const vector<P>& fig) {
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i++) {
        P p = i ? fig[i - 1] : fig.back();
        P q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;
}
```