Contents

1 Basic **1**
1.1 vimrc . . . 1
1.2 Compilation Argument . . . 1
1.3 Checker . . . 1
1.4 Fast Integer Input . . . 1
1.5 Increase stack size . . . 2
1.6 Pragma optimization . . . 2

2 Flow **2**
2.1 Dinic . . . 2
2.2 ISAP . . . 2
2.3 Minimum-cost flow . . . 2
2.4 Gomory-Hu Tree . . . 3
2.5 Stoer–Wagner Minimum Cut . . . 3
2.6 Kuhn–Munkres Algorithm . . . 3
2.7 Flow Model . . . 3

3 Data Structure **4**
3.1 Disjoint Set . . . 4
3.2 <ext/pbds> . . . 4
3.3 Li Chao Tree . . . 4

4 Graph **5**
4.1 Link-Cut Tree . . . 5
4.2 Heavy-Light Decomposition . . . 5
4.3 Centroid Decomposition . . . 6
4.4 Minimum Mean Cycle . . . 6
4.5 Minimum Steiner Tree . . . 6
4.6 Directed Minimum Spanning Tree . . . 7
4.7 Maximum Matching on General Graph . . . 7
4.8 Maximum Weighted Matching on General Graph . . . 8
4.9 Maximum Clique . . . 9
4.10 Tarjan's Articulation Point . . . 9
4.11 Tarjan's Bridge . . . 10
4.12 Dominator Tree . . . 10
4.13 System of Difference Constraints . . . 10

5 String **10**
5.1 Knuth–Morris–Pratt Algorithm . . . 10
5.2 Z Algorithm . . . 10
5.3 Manacher's Algorithm . . . 11
5.4 Aho-Corasick Automaton . . . 11
5.5 Suffix Automaton . . . 11
5.6 Suffix Array . . . 11
5.7 Lexicographically Smallest Rotation . . . 12

6 Math **12**
6.1 Fast Fourier Transform . . . 12
6.2 Number Theoretic Transform . . . 12
6.2.1 NTT Prime List . . . 13
6.3 Polynomial Division . . . 13
6.4 Fast Walsh-Hadamard Transform . . . 13
6.5 Simplex Algorithm . . . 14
6.5.1 Construction . . . 14
6.6 Lagrange Interpolation . . . 14
6.7 Miller Rabin . . . 14
6.8 Pollard's Rho . . . 15
6.9 Meissel–Lehmer Algorithm . . . 15
6.10 Gaussian Elimination . . . 15
6.11 Linear Equations (full pivoting) . . . 15
6.12 μ function . . . 15
6.13 $\lfloor \frac{n}{i} \rfloor$ Enumeration . . . 16
6.14 De Bruijn Sequence . . . 16
6.15 Extended GCD . . . 16
6.16 Chinese Remainder Theorem . . . 16
6.17 Kirchhoff's Theorem . . . 16
6.18 Tutte Matrix . . . 16
6.19 Primes . . . 16

7 Dynamic Programming **16**
7.1 Convex Hull (monotone) . . . 16
7.2 Convex Hull (non-monotone) . . . 16
7.3 $1D/1D$ Convex Optimization . . . 17
7.4 Conditon . . . 17
7.4.1 totally monotone (concave/convex) . . . 17
7.4.2 monge condition (concave/convex) . . . 17

8 Geometry **17**
8.1 Basic . . . 17
8.2 KD Tree . . . 18
8.3 Delaunay Triangulation . . . 18
8.4 Sector Area . . . 19
8.5 Polygon Area . . . 19
8.6 Half Plane Intersection . . . 19
8.7 Rotating Sweep Line . . . 19
8.8 Triangle Center . . . 19
8.9 Polygon Center . . . 20
8.10 Maximum Triangle . . . 20
8.11 Point in Polygon . . . 20
8.12 Circle-Line Intersection . . . 20
8.13 Circle-Triangle Intersection . . . 21
8.14 Polygon Diameter . . . 21
8.15 Minimun Distance of 2 Polygons . . . 21
8.16 2D Convex Hull . . . 21
8.17 3D Convex Hull . . . 21
8.18 Rotating Caliper . . . 22
8.19 Minimum Enclosing Circle . . . 22
8.20 Closest Pair . . . 23

9 Miscellaneous / Problems **23**
9.1 Bitwise Hack . . . 23
9.2 Hilbert's Curve (faster Mo's algorithm) . . . 23
9.3 Java . . . 23
9.4 Offline Dynamic MST . . . 23
9.5 Manhattan Distance MST . . . 24
9.6 "Dynamic" Kth Element (parallel binary search) . . . 25
9.7 Dynamic Kth Element (persistent segment tree) . . . 25

1 Basic

1.1 vimrc

```
se nu rnu
syn on
colo desert
se bs=2 ai ru mouse=a cin et ts=4 sw=4 sts=4
inoremap {<CR> {<CR>}<Esc>O
```

1.2 Compilation Argument

```
g++ -W -Wall -Wextra -O2 -std=c++14 -fsanitize=address
    -fsanitize=undefined -fsanitize=leak
```

1.3 Checker

```
for ((i = 0; i < 100; i++))
do
  ./gen > in
  ./ac < in > out1
  ./tle < in > out2
  diff out1 out2 || break
done
```

1.4 Fast Integer Input

```
#define getchar gtx
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;
}
template <typename T, typename ...Args>
inline bool rit(T& x, Args& ...args) { return rit(x) &&
     rit(args...); }
```

1.5 Increase stack size

```
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));
```

1.6 Pragma optimization

```
#pragma GCC optimize("Ofast", "no-stack-protector", "no
    -math-errno", "unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.2,
    popcnt,abm,mmx,avx,tune=native,arch=core-avx2,tune=
    core-avx2")
#pragma warning(disable:4996)
#pragma GCC ivdep
```

2 Flow

2.1 Dinic

```
struct dinic {
  static const int inf = 1e9;
  struct edge {
    int dest, cap, rev;
    edge(int d, int c, int r): dest(d), cap(c), rev(r)
    {}
  };
  vector<edge> g[maxn];
  int qu[maxn], ql, qr;
  int lev[maxn];
  void init() {
    for (int i = 0; i < maxn; ++i)
      g[i].clear();
  }
  void add_edge(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size() - 0);
    g[b].emplace_back(a, 0, g[a].size() - 1);
  }
  bool bfs(int s, int t) {
    memset(lev, -1, sizeof(lev));
    lev[s] = 0;
    ql = qr = 0;
    qu[qr++] = s;
    while (ql < qr) {
      int x = qu[ql++];
      for (edge &e : g[x]) if (lev[e.dest] == -1 && e.
    cap > 0) {
        lev[e.dest] = lev[x] + 1;
        qu[qr++] = e.dest;
      }
    }
    return lev[t] != -1;
  }
  int dfs(int x, int t, int flow) {
    if (x == t) return flow;
    int res = 0;
    for (edge &e : g[x]) if (e.cap > 0 && lev[e.dest]
    == lev[x] + 1) {
      int f = dfs(e.dest, t, min(e.cap, flow - res));
      res += f;
      e.cap -= f;
      g[e.dest][e.rev].cap += f;
    }
    if (res == 0) lev[x] = -1;
    return res;
  }
  int operator()(int s, int t) {
    int flow = 0;
    for (; bfs(s, t); flow += dfs(s, t, inf));
    return flow;
  }
};
```

2.2 ISAP

```
struct isap {
  static const int inf = 1e9;
  struct edge {
    int dest, cap, rev;
    edge(int a, int b, int c): dest(a), cap(b), rev(c)
     {}
  };
  vector<edge> g[maxn];
  int it[maxn], gap[maxn], d[maxn];
  void add_edge(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size() - 0);
    g[b].emplace_back(a, 0, g[a].size() - 1);
  }
  int dfs(int x, int t, int tot, int flow) {
    if (x == t) return flow;
    for (int &i = it[x]; i < g[x].size(); ++i) {
      edge &e = g[x][i];
      if (e.cap > 0 && d[e.dest] == d[x] - 1) {
        int f = dfs(e.dest, t, tot, min(flow, e.cap));
        if (f) {
          e.cap -= f;
          g[e.dest][e.rev].cap += f;
          return f;
        }
      }
    }
    if ((--gap[d[x]]) == 0) d[x] = tot;
    else d[x]++, it[x] = 0, ++gap[d[x]];
    return 0;
  }
  int operator()(int s, int t, int tot) {
    memset(it, 0, sizeof(it));
    memset(gap, 0, sizeof(gap));
    memset(d, 0, sizeof(d));
    int r = 0;
    gap[0] = tot;
    for (; d[s] < tot; r += dfs(s, t, tot, inf));
    return r;
  }
};
```

2.3 Minimum-cost flow

```
struct mincost {
  struct edge {
    int dest, cap, w, rev;
    edge(int a, int b, int c, int d): dest(a), cap(b),
    w(c), rev(d) {}
  };
  vector<edge> g[maxn];
  int d[maxn], p[maxn], ed[maxn];
  bool inq[maxn];
  void init() {
    for (int i = 0; i < maxn; ++i) g[i].clear();
  }
  void add_edge(int a, int b, int c, int d) {
    g[a].emplace_back(b, c, +d, g[b].size() - 0);
    g[b].emplace_back(a, 0, -d, g[a].size() - 1);
  }
  bool spfa(int s, int t, int &f, int &c) {
    for (int i = 0; i < maxn; ++i) {
      d[i] = inf;
      p[i] = ed[i] = -1;
      inq[i] = false;
    }
    d[s] = 0;
    queue<int> q;
    q.push(s);
    while (q.size()) {
      int x = q.front(); q.pop();
      inq[x] = false;
      for (int i = 0; i < g[x].size(); ++i) {
        edge &e = g[x][i];
        if (e.cap > 0 && d[e.dest] > d[x] + e.w) {
          d[e.dest] = d[x] + e.w;
          p[e.dest] = x;
          ed[e.dest] = i;
```

```
        if (!inq[e.dest]) q.push(e.dest), inq[e.dest]
     = true;
      }
    }
  }
  if (d[t] == inf) return false;
  int dlt = inf;
  for (int x = t; x != s; x = p[x]) dlt = min(dlt, g[
  p[x]][ed[x]].cap);
  for (int x = t; x != s; x = p[x]) {
    edge &e = g[p[x]][ed[x]];
    e.cap -= dlt;
    g[e.dest][e.rev].cap += dlt;
  }
  f += dlt; c += d[t] * dlt;
  return true;
 }
 pair<int, int> operator()(int s, int t) {
  int f = 0, c = 0;
  while (spfa(s, t, f, c));
  return make_pair(f, c);
 }
};
```

2.4 Gomory-Hu Tree

```
int g[maxn];
vector<edge> GomoryHu(int n){
  vector<edge> rt;
  for(int i=1;i<=n;++i)g[i]=1;
  for(int i=2;i<=n;++i){
    int t=g[i];
    flow.reset(); // clear flows on all edge
    rt.push_back({i,t,flow(i,t)});
    flow.walk(i); // bfs points that connected to i (
    use edges not fully flow)
    for(int j=i+1;j<=n;++j){
      if(g[j]==t && flow.connect(j))g[j]=i; // check if
     i can reach j
    }
  }
  return rt;
}
```

2.5 Stoer–Wagner Minimum Cut

```
const int maxn = 500 + 5;
int w[maxn][maxn], g[maxn];
bool v[maxn], del[maxn];

void add_edge(int x, int y, int c) {
  w[x][y] += c;
  w[y][x] += c;
}

pair<int, int> phase(int n) {
  memset(v, false, sizeof(v));
  memset(g, 0, sizeof(g));
  int s = -1, t = -1;
  while (true) {
    int c = -1;
    for (int i = 0; i < n; ++i) {
      if (del[i] || v[i]) continue;
      if (c == -1 || g[i] > g[c]) c = i;
    }
    if (c == -1) break;
    v[c] = true;
    s = t, t = c;
    for (int i = 0; i < n; ++i) {
      if (del[i] || v[i]) continue;
      g[i] += w[c][i];
    }
  }
  return make_pair(s, t);
}

int mincut(int n) {
  int cut = 1e9;
  memset(del, false, sizeof(del));
  for (int i = 0; i < n - 1; ++i) {
    int s, t; tie(s, t) = phase(n);
    del[t] = true;
    cut = min(cut, g[t]);
    for (int j = 0; j < n; ++j) {
      w[s][j] += w[t][j];
      w[j][s] += w[j][t];
    }
  }
  return cut;
}
```

2.6 Kuhn–Munkres Algorithm

```
int w[maxn][maxn], lx[maxn], ly[maxn];
int match[maxn], slack[maxn];
bool vx[maxn], vy[maxn];

bool dfs(int x) {
  vx[x] = true;
  for (int i = 0; i < n; ++i) {
    if (vy[i]) continue;
    if (lx[x] + ly[i] > w[x][i]) {
      slack[i] = min(slack[i], lx[x] + ly[i] - w[x][i])
    ;
      continue;
    }
    vy[i] = true;
    if (match[i] == -1 || dfs(match[i])) {
      match[i] = x;
      return true;
    }
  }
  return false;
}

int solve() {
  fill_n(match, n, -1);
  fill_n(lx, n, -inf);
  fill_n(ly, n, 0);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) lx[i] = max(lx[i], w[i
    ][j]);
  }
  for (int i = 0; i < n; ++i) {
    fill_n(slack, n, inf);
    while (true) {
      fill_n(vx, n, false);
      fill_n(vy, n, false);
      if (dfs(i)) break;
      int dlt = inf;
      for (int j = 0; j < n; ++j) if (!vy[j]) dlt = min
    (dlt, slack[j]);
      for (int j = 0; j < n; ++j) {
        if (vx[j]) lx[j] -= dlt;
        if (vy[j]) ly[j] += dlt;
        else slack[j] -= dlt;
      }
    }
  }
  int res = 0;
  for (int i = 0; i < n; ++i) res += w[match[i]][i];
  return res;
}
```

2.7 Flow Model

- Maximum flow with lower/upper bound from s to t
  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 $in(v)$ as 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 $-in(v)$
  5. Denote f as the maximum flow of the current graph from S to T

6. Connect $t \to s$ with capacity ∞, increment f by the maximum flow from S to T
7. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution
8. Otherwise, the solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow 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
- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x, y, c), connect $x \to y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost, cap) = (-c, 1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost, cap) = (0, d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost, cap) = (0, -d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C + K$
- 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 $f < T|V|$

3 Data Structure

3.1 Disjoint Set

```
struct DisjointSet {
  int p[maxn], sz[maxn], n, cc;
  vector<pair<int*, int>> his;
  vector<int> sh;
  void init(int _n) {
    n = _n; cc = n;
    for (int i = 0; i < n; ++i) sz[i] = 1, p[i] = i;
    sh.clear(); his.clear();
  }
  void assign(int *k, int v) {
    his.emplace_back(k, *k);
    *k = v;
  }
  void save() {
    sh.push_back((int)his.size());
  }
  void undo() {
    int last = sh.back(); sh.pop_back();
    while (his.size() != last) {
      int *k, v;
      tie(k, v) = his.back(); his.pop_back();
      *k = v;
    }
  }
  int find(int x) {
    if (x == p[x]) return x;
    return find(p[x]);
  }
  void merge(int x, int y) {
    x = find(x); y = find(y);
    if (x == y) return;
    if (sz[x] > sz[y]) swap(x, y);
    assign(&sz[y], sz[x] + sz[y]);
    assign(&p[x], y);
    assign(&cc, cc - 1);
  }
} dsu;
```

3.2 <ext/pbds>

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
#include <ext/rope>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
#include <ext/pb_ds/assoc_container.hpp>
typedef tree<int, null_type, std::less<int>,
    rb_tree_tag, tree_order_statistics_node_update>
    tree_set;
typedef cc_hash_table<int, int> umap;
typedef priority_queue<int> heap;

int main() {
  // rb tree
  tree_set s;
  s.insert(71); s.insert(22);
  assert(*s.find_by_order(0) == 22); assert(*s.
    find_by_order(1) == 71);
  assert(s.order_of_key(22) == 0); assert(s.
    order_of_key(71) == 1);
  s.erase(22);
  assert(*s.find_by_order(0) == 71); assert(s.
    order_of_key(71) == 0);
  // mergable heap
  heap a, b; a.join(b);
  // persistant
  rope<char> r[2];
  r[1] = r[0];
  std::string st = "abc";
  r[1].insert(0, st.c_str());
  r[1].erase(1, 1);
  std::cout << r[1].substr(0, 2) << std::endl;
  return 0;
}
```

3.3 Li Chao Tree

```
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, lc[maxc * 4], rc[maxc * 4];
  int gnode() {
    st[sz] = line(1e9, 1e9);
    lc[sz] = -1, rc[sz] = -1;
    return 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) {
      if (lc[o] == -1) lc[o] = gnode();
      add(l, (l + r) / 2, tl, lc[o]);
    } else {
      if (rc[o] == -1) rc[o] = gnode();
      add((l + r) / 2, r, tl, rc[o]);
    }
  }
  long long query(int l, int r, int x, int o) {
    if (r - l == 1) return st[o](x);
    if (x < (l + r) / 2) {
      if (lc[o] == -1) return st[o](x);
      return min(st[o](x), query(l, (l + r) / 2, x, lc[
    o]));
    } else {
      if (rc[o] == -1) return st[o](x);
      return min(st[o](x), query((l + r) / 2, r, x, rc[
    o]));
```

```
    }
  }
}
```

4 Graph

4.1 Link-Cut Tree

```
struct node {
  node *ch[2], *fa, *pfa;
  int sum, v, rev;
  node(int s): v(s), sum(s), rev(0), fa(nullptr), pfa(
    nullptr) {
    ch[0] = nullptr;
    ch[1] = nullptr;
  }
  int relation() {
    return this == fa->ch[0] ? 0 : 1;
  }
  void push() {
    if (!rev) return;
    swap(ch[0], ch[1]);
    if (ch[0]) ch[0]->rev ^= 1;
    if (ch[1]) ch[1]->rev ^= 1;
    rev = 0;
  }
  void pull() {
    sum = v;
    if (ch[0]) sum += ch[0]->sum;
    if (ch[1]) sum += ch[1]->sum;
  }
  void rotate() {
    if (fa->fa) fa->fa->push();
    fa->push(), push();
    swap(pfa, fa->pfa);
    int d = relation();
    node *t = fa;
    if (t->fa) t->fa->ch[t->relation()] = this;
    fa = t->fa;
    t->ch[d] = ch[d ^ 1];
    if (ch[d ^ 1]) ch[d ^ 1]->fa = t;
    ch[d ^ 1] = t;
    t->fa = this;
    t->pull(), pull();
  }
  void splay() {
    while (fa) {
      if (!fa->fa) {
        rotate();
        continue;
      }
      fa->fa->push(), fa->push();
      if (relation() == fa->relation()) fa->rotate(),
    rotate();
      else rotate(), rotate();
    }
  }
  void evert() {
    access();
    splay();
    rev ^= 1;
  }
  void expose() {
    splay(), push();
    if (ch[1]) {
      ch[1]->fa = nullptr;
      ch[1]->pfa = this;
      ch[1] = nullptr;
      pull();
    }
  }
  bool splice() {
    splay();
    if (!pfa) return false;
    pfa->expose();
    pfa->ch[1] = this;
    fa = pfa;
    pfa = nullptr;
    fa->pull();
    return true;
  }
  void access() {
    expose();
    while (splice());
  }
  int query() {
    return sum;
  }
};

namespace lct {
  node *sp[maxn];
  void make(int u, int v) {
    // create node with id u and value v
    sp[u] = new node(v, u);
  }
  void link(int u, int v) {
    // u become v's parent
    sp[v]->evert();
    sp[v]->pfa = sp[u];
  }
  void cut(int u, int v) {
    // u was v's parent
    sp[u]->evert();
    sp[v]->access(), sp[v]->splay(), sp[v]->push();
    sp[v]->ch[0]->fa = nullptr;
    sp[v]->ch[0] = nullptr;
    sp[v]->pull();
  }
  void modify(int u, int v) {
    sp[u]->splay();
    sp[u]->v = v;
    sp[u]->pull();
  }
  int query(int u, int v) {
    sp[u]->evert(), sp[v]->access(), sp[v]->splay();
    return sp[v]->query();
  }
}
```

4.2 Heavy-Light Decomposition

```
struct HeavyLightDecomp {
  vector<int> G[maxn];
  int tin[maxn], top[maxn], dep[maxn], maxson[maxn], sz
    [maxn], p[maxn], n, clk;
  void dfs(int now, int fa, int d) {
    dep[now] = d;
    maxson[now] = -1;
    sz[now] = 1;
    p[now] = fa;
    for (int u : G[now]) if (u != fa) {
      dfs(u, now, d + 1);
      sz[now] += sz[u];
      if (maxson[now] == -1 || sz[u] > sz[maxson[now]])
     maxson[now] = u;
    }
  }
  void link(int now, int t) {
    top[now] = t;
    tin[now] = ++clk;
    if (maxson[now] == -1) return;
    link(maxson[now], t);
    for (int u : G[now]) if (u != p[now]) {
      if (u == maxson[now]) continue;
      link(u, u);
    }
  }
  HeavyLightDecomp(int n): n(n) {
    clk = 0;
    memset(tin, 0, sizeof(tin)); memset(top, 0, sizeof(
    top)); memset(dep, 0, sizeof(dep));
    memset(maxson, 0, sizeof(maxson)); memset(sz, 0,
    sizeof(sz)); memset(p, 0, sizeof(p));
  }
  void add_edge(int a, int b) {
    G[a].push_back(b);
    G[b].push_back(a);
```

```
  }
  void solve() {
    dfs(0, -1, 0);
    link(0, 0);
  }
  int lca(int a, int b) {
    int ta = top[a], tb = top[b];
    while (ta != tb) {
      if (dep[ta] < dep[tb]) {
        swap(ta, tb); swap(a, b);
      }
      a = p[ta]; ta = top[a];
    }
    if (a == b) return a;
    return dep[a] < dep[b] ? a : b;
  }
  vector<pair<int, int>> get_path(int a, int b) {
    int ta = top[a], tb = top[b];
    vector<pair<int, int>> ret;
    while (ta != tb) {
      if (dep[ta] < dep[tb]) {
        swap(ta, tb); swap(a, b);
      }
      ret.push_back(make_pair(tin[ta], tin[a]));
      a = p[ta]; ta = top[a];
    }
    ret.push_back(make_pair(min(tin[a], tin[b]), max(
    tin[a], tin[b])));
    return ret;
  }
};
```

4.3 Centroid Decomposition

```
vector<pair<int, int>> G[maxn];
int sz[maxn], mx[maxn];
bool v[maxn];
vector<int> vtx;

void get_center(int now) {
  v[now] = true; vtx.push_back(now);
  sz[now] = 1; mx[now] = 0;
  for (int u : G[now]) if (!v[u]) {
    get_center(u);
    mx[now] = max(mx[now], sz[u]);
    sz[now] += sz[u];
  }
}

void get_dis(int now, int d, int len) {
  dis[d][now] = cnt;
  v[now] = true;
  for (auto u : G[now]) if (!v[u.first]) {
    get_dis(u, d, len + u.second);
  }
}

void dfs(int now, int fa, int d) {
  get_center(now);
  int c = -1;
  for (int i : vtx) {
    if (max(mx[i], (int)vtx.size() - sz[i]) <= (int)vtx
    .size() / 2) c = i;
    v[i] = false;
  }
  get_dis(c, d, 0);
  for (int i : vtx) v[i] = false;
  v[c] = true; vtx.clear();
  dep[c] = d; p[c] = fa;
  for (auto u : G[c]) if (u.first != fa && !v[u.first])
    {
    dfs(u.first, c, d + 1);
  }
}
```

4.4 Minimum Mean Cycle

```
// d[i][j] == 0 if {i,j} !in E
long long d[1003][1003],dp[1003][1003];

pair<long long,long long> MMWC(){
  memset(dp,0x3f,sizeof(dp));
  for(int i=1;i<=n;++i)dp[0][i]=0;
  for(int i=1;i<=n;++i){
    for(int j=1;j<=n;++j){
      for(int k=1;k<=n;++k){
        dp[i][k]=min(dp[i-1][j]+d[j][k],dp[i][k]);
      }
    }
  }
  long long au=1ll<<31,ad=1;
  for(int i=1;i<=n;++i){
    if(dp[n][i]==0x3f3f3f3f3f3f3f3f)continue;
    long long u=0,d=1;
    for(int j=n-1;j>=0;--j){
      if((dp[n][i]-dp[j][i])*d>u*(n-j)){
        u=dp[n][i]-dp[j][i];
        d=n-j;
      }
    }
    if(u*ad<au*d)au=u,ad=d;
  }
  long long g=__gcd(au,ad);
  return make_pair(au/g,ad/g);
}
```

4.5 Minimum Steiner Tree

```
namespace steiner {
  const int maxn = 64, maxk = 10;
  const int inf = 1e9;
  int w[maxn][maxn], dp[1 << maxk][maxn], off[maxn];
  void init(int n) {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) w[i][j] = inf;
      w[i][i] = 0;
    }
  }
  void add_edge(int x, int y, int d) {
    w[x][y] = min(w[x][y], d);
    w[y][x] = min(w[y][x], d);
  }
  int solve(int n, vector<int> mark) {
    for (int k = 0; k < n; ++k) {
      for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) w[i][j] = min(w[i][
    j], w[i][k] + w[k][j]);
      }
    }
    int k = (int)mark.size();
    assert(k < maxk);
    for (int s = 0; s < (1 << k); ++s) {
      for (int i = 0; i < n; ++i) dp[s][i] = inf;
    }
    for (int i = 0; i < n; ++i) dp[0][i] = 0;
    for (int s = 1; s < (1 << k); ++s) {
      if (__builtin_popcount(s) == 1) {
        int x = __builtin_ctz(s);
        for (int i = 0; i < n; ++i) dp[s][i] = w[mark[x
    ]][i];
        continue;
      }
      for (int i = 0; i < n; ++i) {
        for (int sub = s & (s - 1); sub; sub = s & (sub
    - 1)) {
          dp[s][i] = min(dp[s][i], dp[sub][i] + dp[s ^
    sub][i]);
        }
      }
      for (int i = 0; i < n; ++i) {
        off[i] = inf;
        for (int j = 0; j < n; ++j) off[i] = min(off[i
    ], dp[s][j] + w[j][i]);
      }
      for (int i = 0; i < n; ++i) dp[s][i] = min(dp[s][
    i], off[i]);
    }
```

```
    int res = inf;
    for (int i = 0; i < n; ++i) res = min(res, dp[(1 <<
     k) - 1][i]);
    return res;
  }
}
```

4.6 Directed Minimum Spanning Tree

```
template <typename T> struct DMST {
  T g[maxn][maxn], fw[maxn];
  int n, fr[maxn];
  bool vis[maxn], inc[maxn];
  void clear() {
    for(int i = 0; i < maxn; ++i) {
      for(int j = 0; j < maxn; ++j) g[i][j] = inf;
      vis[i] = inc[i] = false;
    }
  }
  void addedge(int u, int v, T w) {
    g[u][v] = min(g[u][v], w);
  }
  double operator()(int root, int _n) {
    n = _n;
    if (dfs(root) != n) return -1;
    T ans = 0;
    while (true) {
      for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] =
      i;
      for (int i = 1; i <= n; ++i) if (!inc[i]) {
        for (int j = 1; j <= n; ++j) {
          if (!inc[j] && i != j && g[j][i] < fw[i]) {
            fw[i] = g[j][i];
            fr[i] = j;
          }
        }
      }
      int x = -1;
      for (int i = 1; i <= n; ++i) if (i != root && !
    inc[i]) {
        int j = i, c = 0;
        while (j != root && fr[j] != i && c <= n) ++c,
    j = fr[j];
        if (j == root || c > n) continue;
        else { x = i; break; }
      }
      if (!~x) {
        for (int i = 1; i <= n; ++i) if (i != root && !
    inc[i]) ans += fw[i];
        return ans;
      }
      int y = x;
      for (int i = 1; i <= n; ++i) vis[i] = false;
      do { ans += fw[y]; y = fr[y]; vis[y] = inc[y] =
    true; } while (y != x);
      inc[x] = false;
      for (int k = 1; k <= n; ++k) if (vis[k]) {
        for (int j = 1; j <= n; ++j) if (!vis[j]) {
          if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
          if (g[j][k] < inf && g[j][k] - fw[k] < g[j][x
    ]) g[j][x] = g[j][k] - fw[k];
        }
      }
    }
    return ans;
  }
  int dfs(int now) {
    int r = 1;
    vis[now] = true;
    for (int i = 1; i <= n; ++i) if (g[now][i] < inf &&
     !vis[i]) r += dfs(i);
    return r;
  }
};
```

4.7 Maximum Matching on General Graph

```
namespace matching {
  int fa[maxn], match[maxn], aux[maxn], orig[maxn], v[
    maxn], tk;
  vector<int> g[maxn];
  queue<int> q;
  void init() {
    for (int i = 0; i < maxn; ++i) {
      g[i].clear();
      match[i] = -1;
      fa[i] = -1;
      aux[i] = 0;
    }
    tk = 0;
  }
  void add_edge(int x, int y) {
    g[x].push_back(y);
    g[y].push_back(x);
  }
  void augment(int x, int y) {
    int a = y, b = -1;
    do {
      a = fa[y], b = match[a];
      match[y] = a, match[a] = y;
      y = b;
    } while (x != a);
  }
  int lca(int x, int y) {
    ++tk;
    while (true) {
      if (~x) {
        if (aux[x] == tk) return x;
        aux[x] = tk;
        x = orig[fa[match[x]]];
      }
      swap(x, y);
    }
  }
  void blossom(int x, int y, int a) {
    while (orig[x] != a) {
      fa[x] = y, y = match[x];
      if (v[y] == 1) q.push(y), v[y] = 0;
      orig[x] = orig[y] = a;
      x = fa[y];
    }
  }
  bool bfs(int s) {
    for (int i = 0; i < maxn; ++i) {
      v[i] = -1;
      orig[i] = i;
    }
    q = queue<int>();
    q.push(s);
    v[s] = 0;
    while (q.size()) {
      int x = q.front(); q.pop();
      for (const int &u : g[x]) {
        if (v[u] == -1) {
          fa[u] = x, v[u] = 1;
          if (!~match[u]) return augment(s, u), true;
          q.push(match[u]);
          v[match[u]] = 0;
        } else if (v[u] == 0 && orig[x] != orig[u]) {
          int a = lca(orig[x], orig[u]);
          blossom(u, x, a);
          blossom(x, u, a);
        }
      }
    }
    return false;
  }
  int solve(int n) {
    int ans = 0;
    vector<int> z(n);
    iota(z.begin(), z.end(), 0);
    random_shuffle(z.begin(), z.end());
    for (int x : z) if (!~match[x]) {
      for (int y : g[x]) if (!~match[y]) {
        match[y] = x;
        match[x] = y;
        ++ans;
        break;
      }
```

```
    }
    for (int i = 0; i < n; ++i) if (!~match[i] && bfs(i
    )) ++ans;
    return ans;
  }
}
```

4.8 Maximum Weighted Matching on General Graph

```
struct WeightGraph {
  static const int inf = INT_MAX;
  static const int maxn = 514;
  struct edge {
    int u, v, w;
    edge(){}
    edge(int u, int v, int w): u(u), v(v), w(w) {}
  };
  int n, n_x;
  edge g[maxn * 2][maxn * 2];
  int lab[maxn * 2];
  int match[maxn * 2], slack[maxn * 2], st[maxn * 2],
    pa[maxn * 2];
  int flo_from[maxn * 2][maxn + 1], S[maxn * 2], vis[
    maxn * 2];
  vector<int> flo[maxn * 2];
  queue<int> q;
  int e_delta(const edge &e) {
    return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
  }
  void update_slack(int u, int x) {
    if (!slack[x] || e_delta(g[u][x]) < e_delta(g[slack
    [x]][x])) slack[x] = u;
  }
  void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u)
      if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
        update_slack(u, x);
  }
  void q_push(int x) {
    if (x <= n) q.push(x);
    else for (size_t i = 0; i < flo[x].size(); i++)
    q_push(flo[x][i]);
  }
  void set_st(int x, int b) {
    st[x] = b;
    if (x > n) for (size_t i = 0; i < flo[x].size(); ++
    i) set_st(flo[x][i], b);
  }
  int get_pr(int b, int xr) {
    int pr = find(flo[b].begin(), flo[b].end(), xr) -
    flo[b].begin();
    if (pr % 2 == 1) {
      reverse(flo[b].begin() + 1, flo[b].end());
      return (int)flo[b].size() - pr;
    }
    return pr;
  }
  void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    edge e = g[u][v];
    int xr = flo_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i < pr; ++i) set_match(flo[u][i],
    flo[u][i ^ 1]);
    set_match(xr, v);
    rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].
    end());
  }
  void augment(int u, int v) {
    for (; ; ) {
      int xnv = st[match[u]];
      set_match(u, v);
      if (!xnv) return;
      set_match(xnv, st[pa[xnv]]);
      u = st[pa[xnv]], v = xnv;
    }
  }
  int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
      if (u == 0) continue;
      if (vis[u] == t) return u;
      vis[u] = t;
      u = st[match[u]];
      if (u) u = st[pa[u]];
    }
    return 0;
  }
  void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo[b].push_back(y = st[
    match[x]]), q_push(y);
    reverse(flo[b].begin() + 1, flo[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo[b].push_back(y = st[
    match[x]]), q_push(y);
    set_st(b, b);
    for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].
    w = 0;
    for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
    for (size_t i = 0; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      for (int x = 1; x <= n_x; ++x)
        if (g[b][x].w == 0 || e_delta(g[xs][x]) <
    e_delta(g[b][x]))
          g[b][x] = g[xs][x], g[x][b] = g[x][xs];
      for (int x = 1; x <= n; ++x)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slack(b);
  }
  void expand_blossom(int b) {
    for (size_t i = 0; i < flo[b].size(); ++i)
      set_st(flo[b][i], flo[b][i]);
    int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b,
     xr);
    for (int i = 0; i < pr; i += 2) {
      int xs = flo[b][i], xns = flo[b][i + 1];
      pa[xs] = g[xns][xs].u;
      S[xs] = 1, S[xns] = 0;
      slack[xs] = 0, set_slack(xns);
      q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
  }
  bool on_found_edge(const edge &e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
      pa[v] = e.u, S[v] = 1;
      int nu = st[match[v]];
      slack[v] = slack[nu] = 0;
      S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
      int lca = get_lca(u, v);
      if (!lca) return augment(u,v), augment(v,u), true
    ;
      else add_blossom(u, lca, v);
    }
    return false;
  }
  bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
      if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0,
     q_push(x);
```

```
    if (q.empty()) return false;
    for (; ; ) {
      while (q.size()) {
        int u = q.front(); q.pop();
        if (S[st[u]] == 1) continue;
        for (int v = 1; v <= n; ++v)
          if (g[u][v].w > 0 && st[u] != st[v]) {
            if (e_delta(g[u][v]) == 0) {
              if (on_found_edge(g[u][v])) return true;
            } else update_slack(u, st[v]);
          }
      }
      int d = inf;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b && S[b] == 1) d = min(d, lab[b]
 / 2);
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x]) {
          if (S[x] == -1) d = min(d, e_delta(g[slack[x
]][x]));
          else if (S[x] == 0) d = min(d, e_delta(g[
slack[x]][x]) / 2);
        }
      for (int u = 1; u <= n; ++u) {
        if (S[st[u]] == 0) {
          if (lab[u] <= d) return 0;
          lab[u] -= d;
        } else if (S[st[u]] == 1) lab[u] += d;
      }
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b) {
          if (S[st[b]] == 0) lab[b] += d * 2;
          else if (S[st[b]] == 1) lab[b] -= d * 2;
        }
      q = queue<int>();
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x] && st[slack[x]] != x
 && e_delta(g[slack[x]][x]) == 0)
          if (on_found_edge(g[slack[x]][x])) return
 true;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b && S[b] == 1 && lab[b] == 0)
 expand_blossom(b);
    }
    return false;
  }
  pair<long long, int> solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; ++u) st[u] = u, flo[u].
 clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v) {
        flo_from[u][v] = (u == v ? u : 0);
        w_max = max(w_max, g[u][v].w);
      }
    for (int u = 1; u <= n; ++u) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
      if (match[u] && match[u] < u)
        tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
  }
  void add_edge(int ui, int vi, int wi) {
    g[ui][vi].w = g[vi][ui].w = wi;
  }
  void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; ++u)
      for (int v=1; v <= n; ++v)
        g[u][v] = edge(u, v, 0);
  }
};
```

4.9 Maximum Clique

```
struct MaxClique {
  int n, deg[maxn], ans;
  bitset<maxn> adj[maxn];
  vector<pair<int, int>> edge;
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) adj[i].reset();
    for (int i = 0; i < n; ++i) deg[i] = 0;
    edge.clear();
  }
  void add_edge(int a, int b) {
    edge.emplace_back(a, b);
    ++deg[a]; ++deg[b];
  }
  int solve() {
    vector<int> ord;
    for (int i = 0; i < n; ++i) ord.push_back(i);
    sort(ord.begin(), ord.end(), [&](const int &a,
 const int &b) { return deg[a] < deg[b]; });
    vector<int> id(n);
    for (int i = 0; i < n; ++i) id[ord[i]] = i;
    for (auto e : edge) {
      int u = id[e.first], v = id[e.second];
      adj[u][v] = adj[v][u] = true;
    }
    bitset<maxn> r, p;
    for (int i = 0; i < n; ++i) p[i] = true;
    ans = 0;
    dfs(r, p);
    return ans;
  }
  void dfs(bitset<maxn> r, bitset<maxn> p) {
    if (p.count() == 0) return ans = max(ans, (int)r.
 count()), void();
    if ((r | p).count() <= ans) return;
    int now = p._Find_first();
    bitset<maxn> cur = p & ~adj[now];
    for (now = cur._Find_first(); now < n; now = cur.
 _Find_next(now)) {
      r[now] = true;
      dfs(r, p & adj[now]);
      r[now] = false;
      p[now] = false;
    }
  }
};
```

4.10 Tarjan's Articulation Point

```
vector<pair<int, int>> g[maxn];
int low[maxn], tin[maxn], t;
int bcc[maxn], sz;
int a[maxn], b[maxn], deg[maxn];
bool cut[maxn], ins[maxn];

vector<int> ed[maxn];

stack<int> st;

void dfs(int x, int p) {
  tin[x] = low[x] = ++t;
  int ch = 0;
  for (auto u : g[x]) if (u.first != p) {
    if (!ins[u.second]) st.push(u.second), ins[u.second
 ] = true;
    if (tin[u.first]) {
      low[x] = min(low[x], tin[u.first]);
      continue;
    }
    ++ch;
    dfs(u.first, x);
    low[x] = min(low[x], low[u.first]);
    if (low[u.first] >= tin[x]) {
      cut[x] = true;
      ++sz;
      while (true) {
        int e = st.top(); st.pop();
        bcc[e] = sz;
        if (e == u.second) break;
      }
```

```
    }
  }
  if (ch == 1 && p == -1) cut[x] = false;
}
```

4.11 Tarjan's Bridge

```
vector<pair<int, int>> g[maxn];
int tin[maxn], low[maxn], t;
int a[maxn], b[maxn];
int bcc[maxn], sz;
bool br[maxn];

stack<int> st;

void dfs(int x, int p) {
  tin[x] = low[x] = ++t;
  st.push(x);
  for (auto u : g[x]) if (u.first != p) {
    if (tin[u.first]) {
      low[x] = min(low[x], tin[u.first]);
      continue;
    }
    dfs(u.first, x);
    low[x] = min(low[x], low[u.first]);
    if (low[u.first] == tin[u.first]) br[u.second] =
    true;
  }
  if (tin[x] == low[x]) {
    ++sz;
    while (st.size()) {
      int u = st.top(); st.pop();
      bcc[u] = sz;
      if (u == x) break;
    }
  }
}
```

4.12 Dominator Tree

```
namespace dominator {
  vector<int> g[maxn], r[maxn], rdom[maxn];
  int dfn[maxn], rev[maxn], fa[maxn], sdom[maxn], dom[
    maxn], val[maxn], rp[maxn], tk;
  void add_edge(int x, int y) {
    g[x].push_back(y);
  }
  void dfs(int x) {
    rev[dfn[x] = tk] = x;
    fa[tk] = sdom[tk] = val[tk] = tk;
    tk++;
    for (const int &u : g[x]) {
      if (dfn[u] == -1) dfs(u), rp[dfn[u]] = dfn[x];
      r[dfn[u]].push_back(dfn[x]);
    }
  }
  void merge(int x, int y) {
    fa[x] = y;
  }
  int find(int x, int c = 0) {
    if (fa[x] == x) return x;
    int p = find(fa[x], 1);
    if (p == -1) return c ? fa[x] : val[x];
    if (sdom[val[x]] > sdom[val[fa[x]]]) val[x] = val[
    fa[x]];
    fa[x] = p;
    return c ? p : val[x];
  }
  vector<int> build(int s) {
    memset(dfn, -1, sizeof(dfn));
    memset(rev, -1, sizeof(rev));
    memset(fa, -1, sizeof(fa));
    memset(val, -1, sizeof(val));
    memset(sdom, -1, sizeof(sdom));
    memset(rp, -1, sizeof(rp));
    memset(dom, -1, sizeof(dom));
    tk = 0, dfs(s);
    for (int i = tk - 1; i >= 0; --i) {
      for (const int &u : r[i]) sdom[i] = min(sdom[i],
    sdom[find(u)]);
      if (i) rdom[sdom[i]].push_back(i);
      for (const int &u : rdom[i]) {
        int p = find(u);
        if (sdom[p] == i) dom[u] = i;
        else dom[u] = p;
      }
      if (i) merge(i, rp[i]);
    }
    vector<int> p(maxn, -1);
    for (int i = 1; i < tk; ++i) if (sdom[i] != dom[i])
     dom[i] = dom[dom[i]];
    for (int i = 1; i < tk; ++i) p[rev[i]] = rev[dom[i
    ]];
    return p;
  }
}
```

4.13 System of Difference Constraints

Given m constrains on n variables $x_1, x_2, \dots, x_n$ of form $x_i - x_j \le w$ (resp, $x_i - x_j \ge w$), connect $i \to j$ with weight w. Then connect $0 \to i$ for all i with weight 0 and find the shortest path (resp, longest path) on the graph. $dis(i)$ will be the maximum (resp, minimum) solution to x_i.

5 String

5.1 Knuth–Morris–Pratt Algorithm

```
vector<int> kmp(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 - 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 = kmp(t), res;
  int k = 0;
  for (int i = 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] = longest common prefix 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 Manacher's Algorithm

```
int z[maxn];

int manacher(const string& s) {
  string t = ".";
  for (int i = 0; i < s.length(); ++i) t += s[i], t +=
     '.';
  int l = 0, r = 0;
  for (int i = 1; i < t.length(); ++i) {
    z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
    while (i - z[i] >= 0 && i + z[i] < t.length() && t[
    i - z[i]] == t[i + z[i]]) ++z[i];
    if (i + z[i] > r) r = i + z[i], l = i;
  }
  int ans = 0;
  for (int i = 1; i < t.length(); ++i) ans = max(ans, z
     [i] - 1);
  return ans;
}
```

5.4 Aho-Corasick Automaton

```
struct AC {
  static const int maxn = 1e5 + 5;
  int sz, ql, qr, root;
  int cnt[maxn], q[maxn], ed[maxn], el[maxn], ch[maxn
     ][26], f[maxn];
  int gnode() {
    for (int i = 0; i < 26; ++i) ch[sz][i] = -1;
    f[sz] = -1;
    ed[sz] = 0;
    cnt[sz] = 0;
    return sz++;
  }
  void init() {
    sz = 0;
    root = gnode();
  }
  int add(const string &s) {
    int now = root;
    for (int i = 0; i < s.length(); ++i) {
      if (ch[now][s[i] - 'a'] == -1) ch[now][s[i] - 'a'
    ] = gnode();
      now = ch[now][s[i] - 'a'];
    }
    ed[now] = 1;
    return now;
  }
  void build_fail() {
    ql = qr = 0; q[qr++] = root;
    while (ql < qr) {
      int now = q[ql++];
      for (int i = 0; i < 26; ++i) if (ch[now][i] !=
    -1) {
        int p = ch[now][i], fp = f[now];
        while (fp != -1 && ch[fp][i] == -1) fp = f[fp];
        int pd = fp != -1 ? ch[fp][i] : root;
        f[p] = pd;
        el[p] = ed[pd] ? pd : el[pd];
        q[qr++] = p;
      }
    }
  }
  void build(const string &s) {
    build_fail();
    int now = root;
    for (int i = 0; i < s.length(); ++i) {
      while (now != -1 && ch[now][s[i] - 'a'] == -1)
    now = f[now];
      now = now != -1 ? ch[now][s[i] - 'a'] : root;
      ++cnt[now];
    }
    for (int i = qr - 1; i >= 0; --i) cnt[f[q[i]]] +=
    cnt[q[i]];
  }
};
```

5.5 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;
  }
};
```

5.6 Suffix Array

```
int sa[maxn], tmp[2][maxn], c[maxn], hi[maxn], r[maxn];
// sa[i]: sa[i]-th suffix is the i-th lexigraphically
    smallest suffix.
```

```
// hi[i]: longest common prefix of suffix sa[i] and suffix sa[i - 1].
void build(const string &s) {
  int *rnk = tmp[0], *rkn = tmp[1];
  for (int i = 0; i < 256; ++i) c[i] = 0;
  for (int i = 0; i < s.size(); ++i) c[rnk[i] = s[i]]++;
  for (int i = 1; i < 256; ++i) c[i] += c[i - 1];
  for (int i = s.size() - 1; i >= 0; --i) sa[--c[s[i]]] = i;
  int sigma = 256;
  for (int n = 1; n < s.size(); n *= 2) {
    for (int i = 0; i < sigma; ++i) c[i] = 0;
    for (int i = 0; i < s.size(); ++i) c[rnk[i]]++;
    for (int i = 1; i < sigma; ++i) c[i] += c[i - 1];
    int *sa2 = rkn;
    int r = 0;
    for (int i = s.size() - n; i < s.size(); ++i) sa2[r++] = i;
    for (int i = 0; i < s.size(); ++i) {
      if (sa[i] >= n) sa2[r++] = sa[i] - n;
    }
    for (int i = s.size() - 1; i >= 0; --i) sa[--c[rnk[sa2[i]]]] = sa2[i];
    rkn[sa[0]] = r = 0;
    for (int i = 1; i < s.size(); ++i) {
      if (!(rnk[sa[i - 1]] == rnk[sa[i]] && sa[i - 1] + n < s.size() && rnk[sa[i - 1] + n] == rnk[sa[i] + n])) r++;
      rkn[sa[i]] = r;
    }
    swap(rnk, rkn);
    if (r == s.size() - 1) break;
    sigma = r + 1;
  }
  for (int i = 0; i < s.size(); ++i) r[sa[i]] = i;
  int ind = 0; hi[0] = 0;
  for (int i = 0; i < s.size(); ++i) {
    if (!r[i]) { ind = 0; continue; }
    while (i + ind < s.size() && s[i + ind] == s[sa[r[i] - 1] + ind]) ++ind;
    hi[r[i]] = ind ? ind-- : 0;
  }
}
```

5.7 Lexicographically Smallest Rotation

```
string rotate(const string &s) {
  int n = s.length();
  string t = s + s;
  int i = 0, j = 1;
  while (i < n && j < n) {
    int k = 0;
    while (k < n && t[i + k] == t[j + k]) ++k;
    if (t[i + k] <= t[j + k]) j += k + 1;
    else i += k + 1;
    if (i == j) ++j;
  }
  int pos = (i < n ? i : j);
  return t.substr(pos, n);
}
```

6 Math

6.1 Fast Fourier Transform

```
struct cplx {
  double re, im;
  cplx(): re(0), im(0) {}
  cplx(double r, double i): re(r), im(i) {}
  cplx operator+(const cplx &rhs) const { return cplx(re + rhs.re, im + rhs.im); }
  cplx operator-(const cplx &rhs) const { return cplx(re - rhs.re, im - rhs.im); }
  cplx operator*(const cplx &rhs) const { return cplx(re * rhs.re - im * rhs.im, re * rhs.im + im * rhs.re); }
  cplx conj() const { return cplx(re, -im); }
};

const int maxn = 262144;
const double pi = acos(-1);
cplx omega[maxn + 1];

void prefft() {
  for (int i = 0; i <= maxn; ++i)
    omega[i] = cplx(cos(2 * pi * i / maxn), sin(2 * pi * i / maxn));
}

void bitrev(vector<cplx> &v, int n) {
  int z = __builtin_ctz(n) - 1;
  for (int i = 0; i < n; ++i) {
    int x = 0;
    for (int j = 0; (1 << j) < n; ++j) x ^= (((i >> j & 1)) << (z - j));
    if (x > i) swap(v[x], v[i]);
  }
}

void fft(vector<cplx> &v, int n) {
  bitrev(v, n);
  for (int s = 2; s <= n; s <<= 1) {
    int z = s >> 1;
    for (int i = 0; i < n; i += s) {
      for (int k = 0; k < z; ++k) {
        cplx x = v[i + z + k] * omega[maxn / s * k];
        v[i + z + k] = v[i + k] - x;
        v[i + k] = v[i + k] + x;
      }
    }
  }
}

void ifft(vector<cplx> &v, int n) {
  fft(v, n);
  reverse(v.begin() + 1, v.end());
  for (int i = 0; i < n; ++i) v[i] = v[i] * cplx(1. / n, 0);
}

vector<int> conv(const vector<int> &a, const vector<int> &b) {
  int sz = 1;
  while (sz < a.size() + b.size() - 1) sz <<= 1;
  vector<cplx> v(sz);
  for (int i = 0; i < sz; ++i) {
    double re = i < a.size() ? a[i] : 0;
    double im = i < b.size() ? b[i] : 0;
    v[i] = cplx(re, im);
  }
  fft(v, sz);
  for (int i = 0; i <= sz / 2; ++i) {
    int j = (sz - i) & (sz - 1);
    cplx x = (v[i] + v[j].conj()) * (v[i] - v[j].conj()) * cplx(0, -0.25);
    if (j != i) v[j] = (v[j] + v[i].conj()) * (v[j] - v[i].conj()) * cplx(0, -0.25);
    v[i] = x;
  }
  ifft(v, sz);
  vector<int> c(sz);
  for (int i = 0; i < sz; ++i) c[i] = round(v[i].re);
  while (c.size() && c.back() == 0) c.pop_back();
  return c;
}
```

6.2 Number Theoretic Transform

```
const int maxn = 262144;
const long long mod = 2013265921, root = 31;
long long omega[maxn + 1];

long long fpow(long long a, long long n) {
  (n += mod - 1) %= mod - 1;
  long long r = 1;
  for (; n; n >>= 1) {
```

```
    if (n & 1) (r *= a) %= mod;
    (a *= a) %= mod;
  }
  return r;
}

void prentt() {
  long long x = fpow(root, (mod - 1) / maxn);
  omega[0] = 1;
  for (int i = 1; i <= maxn; ++i)
    omega[i] = omega[i - 1] * x % mod;
}

void bitrev(vector<long long> &v, int n) {
  int z = __builtin_ctz(n) - 1;
  for (int i = 0; i < n; ++i) {
    int x = 0;
    for (int j = 0; j <= z; ++j) x ^= ((i >> j & 1) <<
    (z - j));
    if (x > i) swap(v[x], v[i]);
  }
}

void ntt(vector<long long> &v, int n) {
  bitrev(v, n);
  for (int s = 2; s <= n; s <<= 1) {
    int z = s >> 1;
    for (int i = 0; i < n; i += s) {
      for (int k = 0; k < z; ++k) {
        long long x = v[i + k + z] * omega[maxn / s * k
    ] % mod;
        v[i + k + z] = (v[i + k] + mod - x) % mod;
        (v[i + k] += x) %= mod;
      }
    }
  }
}

void intt(vector<long long> &v, int n) {
  ntt(v, n);
  reverse(v.begin() + 1, v.end());
  long long inv = fpow(n, mod - 2);
  for (int i = 0; i < n; ++i) (v[i] *= inv) %= mod;
}

vector<long long> conv(vector<long long> a, vector<long
     long> b) {
  int sz = 1;
  while (sz < a.size() + b.size() - 1) sz <<= 1;
  vector<long long> c(sz);
  while (a.size() < sz) a.push_back(0);
  while (b.size() < sz) b.push_back(0);
  ntt(a, sz), ntt(b, sz);
  for (int i = 0; i < sz; ++i) c[i] = a[i] * b[i] % mod
    ;
  intt(c, sz);
  while (c.size() && c.back() == 0) c.pop_back();
  return c;
}
```

6.2.1 NTT Prime List

Prime	Root	Prime	Root
7681	17	167772161	3
12289	11	104857601	3
40961	3	985661441	3
65537	3	998244353	3
786433	10	1107296257	10
5767169	3	2013265921	31
7340033	3	2810183681	11
23068673	3	2885681153	3
469762049	3	605028353	3

6.3 Polynomial Division

```
vector<int> inverse(const vector<int> &v, int n) {
  vector<int> q(1, fpow(v[0], mod - 2));
  for (int i = 2; i <= n; i <<= 1) {
    vector<int> fv(v.begin(), v.begin() + i);
    vector<int> fq(q.begin(), q.end());
    fv.resize(2 * i), fq.resize(2 * i);
    ntt(fq, 2 * i), ntt(fv, 2 * i);
    for (int j = 0; j < 2 * i; ++j) {
      fv[j] = fv[j] * 1ll * fq[j] % mod * 1ll * fq[j] %
     mod;
    }
    intt(fv, 2 * i);
    vector<int> res(i);
    for (int j = 0; j < i; ++j) {
      res[j] = mod - fv[j];
      if (j < (i >> 1)) (res[j] += 2 * q[j] % mod) %=
    mod;
    }
    q = res;
  }
  return q;
}

vector<int> divide(const vector<int> &a, const vector<
    int> &b) {
  // leading zero should be trimmed
  int n = (int)a.size(), m = (int)b.size();
  int k = 2;
  while (k < n - m + 1) k <<= 1;
  vector<int> ra(k), rb(k);
  for (int i = 0; i < min(n, k); ++i) ra[i] = a[n - i -
    1];
  for (int i = 0; i < min(m, k); ++i) rb[i] = b[m - i -
    1];
  vector<int> rbi = inverse(rb, k);
  vector<int> res = conv(rbi, ra);
  res.resize(n - m + 1);
  reverse(res.begin(), res.end());
  return res;
}
```

6.4 Fast Walsh-Hadamard Transform

```
void xorfwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  xorfwt(v, l, m), xorfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) {
    int x = v[i] + v[j];
    v[j] = v[i] - v[j], v[i] = x;
  }
}

void xorifwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  for (int i = l, j = m; i < m; ++i, ++j) {
    int x = (v[i] + v[j]) / 2;
    v[j] = (v[i] - v[j]) / 2, v[i] = x;
  }
  xorifwt(v, l, m), xorifwt(v, m, r);
}

void andfwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  andfwt(v, l, m), andfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[i] += v[j];
}

void andifwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  andifwt(v, l, m), andifwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[i] -= v[j];
}

void orfwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  orfwt(v, l, m), orfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[j] += v[i];
}

void orifwt(int v[], int l, int r) {
  if (r - l == 1) return;
```

```cpp
  int m = l + r >> 1;
  orifwt(v, l, m), orifwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[j] -= v[i];
}
```

6.5 Simplex Algorithm

```cpp
namespace simplex {
  // maximize c^Tx under Ax <= B
  // return vector<double>(n, -inf) if the solution doesn't exist
  // return vector<double>(n, +inf) if the solution is unbounded
  const double eps = 1e-9;
  const double inf = 1e+9;
  int n, m;
  vector<vector<double>> d;
  vector<int> p, q;
  void pivot(int r, int s) {
    double inv = 1.0 / d[r][s];
    for (int i = 0; i < m + 2; ++i) {
      for (int j = 0; j < n + 2; ++j) {
        if (i != r && j != s) d[i][j] -= d[r][j] * d[i][s] * inv;
      }
    }
    for (int i = 0; i < m + 2; ++i) if (i != r) d[i][s] *= -inv;
    for (int j = 0; j < n + 2; ++j) if (j != s) d[r][j] *= +inv;
    d[r][s] = inv;
    swap(p[r], q[s]);
  }
  bool phase(int z) {
    int x = m + z;
    while (true) {
      int s = -1;
      for (int i = 0; i <= n; ++i) {
        if (!z && q[i] == -1) continue;
        if (s == -1 || d[x][i] < d[x][s]) s = i;
      }
      if (d[x][s] > -eps) return true;
      int r = -1;
      for (int i = 0; i < m; ++i) {
        if (d[i][s] < eps) continue;
        if (r == -1 || d[i][n + 1] / d[i][s] < d[r][n + 1] / d[r][s]) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }
  vector<double> solve(const vector<vector<double>> &a, const vector<double> &b, const vector<double> &c) {
    m = b.size(), n = c.size();
    d = vector<vector<double>>(m + 2, vector<double>(n + 2));
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j) d[i][j] = a[i][j];
    }
    p.resize(m), q.resize(n + 1);
    for (int i = 0; i < m; ++i) p[i] = n + i, d[i][n] = -1, d[i][n + 1] = b[i];
    for (int i = 0; i < n; ++i) q[i] = i, d[m][i] = -c[i];
    q[n] = -1, d[m + 1][n] = 1;
    int r = 0;
    for (int i = 1; i < m; ++i) if (d[i][n + 1] < d[r][n + 1]) r = i;
    if (d[r][n + 1] < -eps) {
      pivot(r, n);
      if (!phase(1) || d[m + 1][n + 1] < -eps) return vector<double>(n, -inf);
      for (int i = 0; i < m; ++i) if (p[i] == -1) {
        int s = min_element(d[i].begin(), d[i].end() - 1) - d[i].begin();
        pivot(i, s);
      }
    }
    if (!phase(0)) return vector<double>(n, inf);
    vector<double> x(n);
    for (int i = 0; i < n; ++i) if (p[i] < n) x[p[i]] = d[i][n + 1];
    return x;
  }
}
```

6.5.1 Construction

Standard form: maximize $\sum_{1\le i\le n} c_i x_i$ such that for all $1 \le j \le m$, $\sum_{1\le i\le n} A_{ji}x_i \le b_j$.and $x_i \ge 0$ for all $1 \le i \le n$.

1. In case of minimization, let $c'_i = -c_i$
2. $\sum_{1\le i\le n} A_{ji}x_i \ge b_j \rightarrow \sum_{1\le i\le n} -A_{ji}x_i \le -b_j$
3. $\sum_{1\le i\le n} A_{ji}x_i = b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \le b_j$
   - $\sum_{1\le i\le n} A_{ji}x_i \ge b_j$
4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.6 Lagrange Interpolation

```cpp
namespace lagrange {
  long long pf[maxn], nf[maxn];
  void init() {
    pf[0] = nf[0] = 1;
    for (int i = 1; i < maxn; ++i) {
      pf[i] = pf[i - 1] * i % mod;
      nf[i] = nf[i - 1] * (mod - i) % mod;
    }
  }
  // given y: value of f(a), a = [0, n], find f(x)
  long long solve(int n, vector<long long> y, long long x) {
    if (x <= n) return y[x];
    long long all = 1;
    for (int i = 0; i <= n; ++i) (all *= (x - i + mod)) %= mod;
    long long ans = 0;
    for (int i = 0; i <= n; ++i) {
      long long z = all * fpow(x - i, -1) % mod;
      long long l = pf[i], r = nf[n - i];
      (ans += y[i] * z % mod * fpow(l * r, -1)) %= mod;
    }
    return ans;
  }
}
```

6.7 Miller Rabin

```cpp
// n < 4759123141   chk = [2, 7, 61]
// n < 1122004669633 chk = [2, 13, 23, 1662803]
// n < 2^64       chk = [2, 325, 9375, 28178, 450775, 9780504, 1795265022]
//
vector<long long> chk = { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 };

bool check(long long a, long long u, long long n, int t) {
  a = fpow(a, u, n);
  if (a == 0) return true;
  if (a == 1 || a == n - 1) return true;
  for (int i = 0; i < t; ++i) {
    a = fmul(a, a, n);
    if (a == 1) return false;
    if (a == n - 1) return true;
  }
  return false;
}

bool is_prime(long long n) {
  if (n < 2) return false;
```

```
  if (n % 2 == 0) return n == 2;
  long long u = n - 1; int t = 0;
  for (; u & 1; u >>= 1, ++t);
  for (long long i : chk) {
    if (!check(i, u, n, t)) return false;
  }
  return true;
}
```

6.8 Pollard's Rho

```
long long f(long long x, long long n, int p) { return (
    fmul(x, x, n) + p) % n; }

map<long long, int> cnt;

void pollard_rho(long long n) {
  if (n == 1) return;
  if (prime(n)) return ++cnt[n], void();
  if (n % 2 == 0) return pollard_rho(n / 2), ++cnt[2],
    void();
  long long x = 2, y = 2, d = 1, p = 1;
  while (true) {
    if (d != n && d != 1) {
      pollard_rho(n / d);
      pollard_rho(d);
      return;
    }
    if (d == n) ++p;
    x = f(x, n, p); y = f(f(y, n, p), n, p);
    d = __gcd(abs(x - y), n);
  }
}
```

6.9 Meissel–Lehmer Algorithm

```
int prc[maxn];
long long phic[msz][nsz];

void sieve() {
  bitset<maxn> v;
  pr.push_back(0);
  for (int i = 2; i < maxn; ++i) {
    if (!v[i]) pr.push_back(i);
    for (int j = 1; i * pr[j] < maxn; ++j) {
      v[i * pr[j]] = true;
      if (i % pr[j] == 0) break;
    }
  }
  for (int i = 1; i < pr.size(); ++i) prc[pr[i]] = 1;
  for (int i = 1; i < maxn; ++i) prc[i] += prc[i - 1];
}

long long p2(long long, long long);

long long phi(long long m, long long n) {
  if (m < msz && n < nsz && phic[m][n] != -1) return
    phic[m][n];
  if (n == 0) return m;
  if (pr[n] >= m) return 1;
  long long ret = phi(m, n - 1) - phi(m / pr[n], n - 1)
    ;
  if (m < msz && n < nsz) phic[m][n] = ret;
  return ret;
}

long long pi(long long m) {
  if (m < maxn) return prc[m];
  long long n = pi(cbrt(m));
  return phi(m, n) + n - 1 - p2(m, n);
}

long long p2(long long m, long long n) {
  long long ret = 0;
  long long lim = sqrt(m);
  for (int i = n + 1; pr[i] <= lim; ++i) ret += pi(m /
    pr[i]) - pi(pr[i]) + 1;
  return ret;
}
```

6.10 Gaussian Elimination

```
void gauss(vector<vector<double>> &d) {
  int n = d.size(), m = d[0].size();
  for (int i = 0; i < m; ++i) {
    int p = -1;
    for (int j = i; j < n; ++j) {
      if (fabs(d[j][i]) < eps) continue;
      if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) p =
    j;
    }
    if (p == -1) continue;
    for (int j = 0; j < m; ++j) swap(d[p][j], d[i][j]);
    for (int j = 0; j < n; ++j) {
      if (i == j) continue;
      double z = d[j][i] / d[i][i];
      for (int k = 0; k < m; ++k) d[j][k] -= z * d[i][k
    ];
    }
  }
}
```

6.11 Linear Equations (full pivoting)

```
void linear_equation(vector<vector<double>> &d, vector<
    double> &aug, vector<double> &sol) {
  int n = d.size(), m = d[0].size();
  vector<int> r(n), c(m);
  iota(r.begin(), r.end(), 0);
  iota(c.begin(), c.end(), 0);
  for (int i = 0; i < m; ++i) {
    int p = -1, z = -1;
    for (int j = i; j < n; ++j) {
      for (int k = i; k < m; ++k) {
        if (fabs(d[r[j]][c[k]]) < eps) continue;
        if (p == -1 || fabs(d[r[j]][c[k]]) > fabs(d[r[p
    ]][c[z]])) p = j, z = k;
      }
    }
    if (p == -1) continue;
    swap(r[p], r[i]), swap(c[z], c[i]);
    for (int j = 0; j < n; ++j) {
      if (i == j) continue;
      double z = d[r[j]][c[i]] / d[r[i]][c[i]];
      for (int k = 0; k < m; ++k) d[r[j]][c[k]] -= z *
    d[r[i]][c[k]];
      aug[r[j]] -= z * aug[r[i]];
    }
  }
  vector<vector<double>> fd(n, vector<double>(m));
  vector<double> faug(n), x(n);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) fd[i][j] = d[r[i]][c[j
    ]];
    faug[i] = aug[r[i]];
  }
  d = fd, aug = faug;
  for (int i = n - 1; i >= 0; --i) {
    double p = 0.0;
    for (int j = i + 1; j < n; ++j) p += d[i][j] * x[j
    ];
    x[i] = (aug[i] - p) / d[i][i];
  }
  for (int i = 0; i < n; ++i) sol[c[i]] = x[i];
}
```

6.12 μ function

```
int mu[maxn], pi[maxn];
vector<int> prime;

void sieve() {
  mu[1] = pi[1] = 1;
  for (int i = 2; i < maxn; ++i) {
```

```
    if (!pi[i]) {
      pi[i] = i;
      prime.push_back(i);
      mu[i] = -1;
    }
    for (int j = 0; i * prime[j] < maxn; ++j) {
      pi[i * prime[j]] = prime[j];
      mu[i * prime[j]] = -mu[i];
      if (i % prime[j] == 0) {
        mu[i * prime[j]] = 0;
        break;
      }
    }
  }
}
```

6.13 $\lfloor \frac{n}{i} \rfloor$ Enumeration

$T_0 = 1, T_i = \lfloor \frac{n}{\lfloor \frac{n}{T_{i-1}+1} \rfloor} \rfloor$

6.14 De Bruijn Sequence

```
int res[maxn], aux[maxn], a[maxn], sz;

void db(int t, int p, int n, int k) {
  if (sz >= tg) return;
  if (t > n) {
    if (n % p == 0) {
      for (int i = 1; i <= p && sz < tg; ++i) res[sz++]
       = aux[i];
    }
  } else {
    aux[t] = aux[t - p];
    db(t + 1, p, n, k);
    for (int i = aux[t - p] + 1; i < k; ++i) {
      aux[t] = i;
      db(t + 1, t, n, k);
    }
  }
}

int de_bruijn(int k, int n) {
  // return cyclic string of length k^n such that every
      string of length n using k character appears as a
     substring.
  if (k == 1) {
    res[0] = 0;
    return 1;
  }
  for (int i = 0; i < k * n; i++) aux[i] = 0;
  sz = 0;
  db(1, 1, n, k);
  return sz;
}
```

6.15 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);
}
```

6.16 Chinese Remainder Theorem

Given $x \equiv a_i \mod n_i \forall 1 \leq i \leq k$, where n_i are pairwise coprime, find x.
Let $N = \prod_{i=1}^{k} n_i$ and $N_i = N/n_i$, there exist integer M_i and m_i such that $M_i N_i + m_i n_i = 1$.
A solution to the system of congruence is $x = \sum_{i=1}^{k} a_i M_i N_i$.

6.17 Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G, where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G.

- The number of undirected spanning in G is $|\det(L^*)|$, where L^* is the $(n-1) \times (n-1)$ matrix by removing row x and column x for some arbitrary x in L
- The number of directed spanning tree rooted at r in G is $|\det(L_r)|$, where L_r is the $(n-1) \times (n-1)$ matrix by removing row r and column r in L

6.18 Tutte Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniform randomly) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.

6.19 Primes

$97, 101, 131, 487, 593, 877, 1087, 1187, 1487, 1787, 3187, 12721, 13331, 14341, 75577, 123457, 222557, 556679, 999983, 1097774749, 1076767633, 100102021, 999997771, 1001010013, 1000512343, 987654361, 999991231, 999888733, 98789101, 987777733, 999991921, 1000000007, 1000000087, 1000000123, 1010101333, 1010102101, 1000000000039, 1000000000000037, 2305843009213693951, 4611686018427387847, 9223372036854775783, 18446744073709551557$

7 Dynamic Programming

7.1 Convex Hull (monotone)

```
struct line {
  double a, b;
  inline double operator()(const double &x) const {
    return a * x + b; }
  inline bool checkfront(const line &l, const double &x
    ) const { return (*this)(x) < l(x); }
  inline double intersect(const line &l) const { return
     (l.b - b) / (a - l.a); }
  inline bool checkback(const line &l, const line &
    pivot) const { return pivot.intersect((*this)) <=
    pivot.intersect(l); }
};

void solve() {
  for (int i = 1; i < maxn; ++i) dp[0][i] = inf;
  for (int i = 1; i <= k; ++i) {
    deque<line> dq; dq.push_back((line){ 0.0, dp[i -
    1][0] });
    for (int j = 1; j <= n; ++j) {
      while (dq.size() >= 2 && dq[1].checkfront(dq[0],
    invt[j])) dq.pop_front();
      dp[i][j] = st[j] + dq.front()(invt[j]);
      line nl = (line){ -s[j], dp[i - 1][j] - st[j] + s
    [j] * invt[j] };
      while (dq.size() >= 2 && nl.checkback(dq[dq.size
    () - 1], dq[dq.size() - 2])) dq.pop_back();
      dq.push_back(nl);
    }
  }
}
```

7.2 Convex Hull (non-monotone)

```
struct line {
  int m, y;
  int l, r;
  line(int m = 0,int y = 0, int l = -5, int r =
     1000000009): m(m), y(y), l(l), r(r) {}
  int get(int x) const { return m * x + y; }
  int useful(line le) const {
    return (int)(get(l) >= le.get(l)) + (int)(get(r) >=
      le.get(r));
  }
};

int magic;
bool operator < (const line &a, const line &b) {
  if (magic) return a.m < b.m;
  return a.l < b.l;
}

set<line> st;

void addline(line l) {
  magic = 1;
  auto it = st.lower_bound(l);
  if (it != st.end() && it->useful(l) == 2) return;
  while (it != st.end() && it->useful(l) == 0) it = st.
     erase(it);
  if (it != st.end() && it->useful(l) == 1) {
    int L = it->l, R = it->r, M;
    while (R > L) {
      M = (L + R + 1) >> 1;
      if (it->get(M) >= l.get(M)) R = M - 1;
      else L = M;
    }
    line cp = *it;
    st.erase(it);
    cp.l = L + 1;
    if (cp.l <= cp.r) st.insert(cp);
    l.r = L;
  }
  else if (it != st.end()) l.r = it->l - 1;
  it = st.lower_bound(l);
  while (it != st.begin() && prev(it)->useful(l) == 0)
    it = st.erase(prev(it));
  if (it != st.begin() && prev(it)->useful(l) == 1) {
    --it;
    int L = it->l, R = it->r, M;
    while (R > L) {
      M = (L + R) >> 1;
      if (it->get(M) >= l.get(M)) L = M + 1;
      else R = M;
    }
    line cp = *it;
    st.erase(it);
    cp.r = L - 1;
    if (cp.l <= cp.r) st.insert(cp);
    l.l = L;
  }
  else if (it != st.begin()) l.l = prev(it)->r + 1;
  if (l.l <= l.r) st.insert(l);
}

int getval(int d) {
  magic = 0;
  return (--st.upper_bound(line(0, 0, d, 0)))->get(d);
}
```

7.3 $1D/1D$ Convex Optimization

```
struct segment {
  int i, l, r;
  segment() {}
  segment(int a, int b, int c): i(a), l(b), r(c) {}
};

inline long long f(int l, int r) {
  return dp[l] + w(l + 1, r);
}

void solve() {
  dp[0] = 0ll;
  deque&lt;segment&gt; deq; deq.push_back(segment(0, 1, n));
  for (int i = 1; i <= n; ++i) {
    dp[i] = f(deq.front().i, i);
    while (deq.size() && deq.front().r < i + 1) deq.
    pop_front();
    deq.front().l = i + 1;
    segment seg = segment(i, i + 1, n);
    while (deq.size() && f(i, deq.back().l) < f(deq.
    back().i, deq.back().l)) deq.pop_back();
    if (deq.size()) {
      int d = 1048576, c = deq.back().l;
      while (d >>= 1) if (c + d <= deq.back().r) {
        if (f(i, c + d) > f(deq.back().i, c + d)) c +=
    d;
      }
      deq.back().r = c; seg.l = c + 1;
    }
    if (seg.l <= n) deq.push_back(seg);
  }
}
```

7.4 Conditon

7.4.1 totally monotone (concave/convex)

$$\forall i < i', j < j',\ B[i][j] \le B[i'][j] \implies B[i][j'] \le B[i'][j']$$
$$\forall i < i', j < j',\ B[i][j] \ge B[i'][j] \implies B[i][j'] \ge B[i'][j']$$

7.4.2 monge condition (concave/convex)

$$\forall i < i', j < j',\ B[i][j] + B[i'][j'] \ge B[i][j'] + B[i'][j]$$
$$\forall i < i', j < j',\ B[i][j] + B[i'][j'] \le B[i][j'] + B[i'][j]$$

8 Geometry

8.1 Basic

```
bool same(const double a,const double b){ return abs(a-
    b)<1e-9; }

struct Point{
  double x,y;
  Point():x(0),y(0){}
  Point(double x,double y):x(x),y(y){}
};
Point operator+(const Point a,const Point b){ return
    Point(a.x+b.x,a.y+b.y); }
Point operator-(const Point a,const Point b){ return
    Point(a.x-b.x,a.y-b.y); }
Point operator*(const Point a,const double b){ return
    Point(a.x*b,a.y*b); }
Point operator/(const Point a,const double b){ return
    Point(a.x/b,a.y/b); }
double operator^(const Point a,const Point b){ return a
    .x*b.y-a.y*b.x; }
double abs(const Point a){ return sqrt(a.x*a.x+a.y*a.y)
    ; }


struct Line{
  // ax+by+c=0
  double a,b,c;
  double angle;
  Point pa,pb;
  Line():a(0),b(0),c(0),angle(0),pa(),pb(){}
  Line(Point pa,Point pb):a(pa.y-pb.y),b(pb.x-pa.x),c(
    pa^pb),angle(atan2(-a,b)),pa(pa),pb(pb){}
};

Point intersect(Line la,Line lb){
  if(same(la.a*lb.b,la.b*lb.a))return Point(7122,7122);
  double bot=-la.a*lb.b+la.b*lb.a;
  return Point(-la.b*lb.c+la.c*lb.b,la.a*lb.c-la.c*lb.a
    )/bot;
}
```

8.2 KD Tree

```
namespace kdt {
  int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn], yl[
    maxn], yr[maxn];
  point p[maxn];
  int build(int l, int r, int dep = 0) {
    if (l == r) return -1;
    function<bool(const point &, const point &)> f = [
    dep](const point &a, const point &b) {
      if (dep & 1) return a.x < b.x;
      else return a.y < b.y;
    };
    int m = (l + r) >> 1;
    nth_element(p + l, p + m, p + r, f);
    xl[m] = xr[m] = p[m].x;
    yl[m] = yr[m] = p[m].y;
    lc[m] = build(l, m, dep + 1);
    if (~lc[m]) {
      xl[m] = min(xl[m], xl[lc[m]]);
      xr[m] = max(xr[m], xr[lc[m]]);
      yl[m] = min(yl[m], yl[lc[m]]);
      yr[m] = max(yr[m], yr[lc[m]]);
    }
    rc[m] = build(m + 1, r, dep + 1);
    if (~rc[m]) {
      xl[m] = min(xl[m], xl[rc[m]]);
      xr[m] = max(xr[m], xr[rc[m]]);
      yl[m] = min(yl[m], yl[rc[m]]);
      yr[m] = max(yr[m], yr[rc[m]]);
    }
    return m;
  }
  bool bound(const point &q, int o, long long d) {
    double ds = sqrt(d + 1.0);
    if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
      q.y < yl[o] - ds || q.y > yr[o] + ds) return
    false;
    return true;
  }
  long long dist(const point &a, const point &b) {
    return (a.x - b.x) * 1ll * (a.x - b.x) +
        (a.y - b.y) * 1ll * (a.y - b.y);
  }
  void dfs(const point &q, long long &d, int o, int dep
    = 0) {
    if (!bound(q, o, d)) return;
    long long cd = dist(p[o], q);
    if (cd != 0) d = min(d, cd);
    if ((dep & 1) && q.x < p[o].x || !(dep & 1) && q.y
    < p[o].y) {
      if (~lc[o]) dfs(q, d, lc[o], dep + 1);
      if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    } else {
      if (~rc[o]) dfs(q, d, rc[o], dep + 1);
      if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    }
  }
  void init(const vector<point> &v) {
    for (int i = 0; i < v.size(); ++i) p[i] = v[i];
    root = build(0, v.size());
  }
  long long nearest(const point &q) {
    long long res = 1e18;
    dfs(q, res, root);
    return res;
  }
}
```

8.3 Delaunay Triangulation

```
namespace triangulation {
  static const int maxn = 1e5 + 5;
  vector<point> p;
  set<int> g[maxn];
  int o[maxn];
  set<int> s;
  void add_edge(int x, int y) {
    s.insert(x), s.insert(y);
    g[x].insert(y);
    g[y].insert(x);
  }
  bool inside(point a, point b, point c, point p) {
    if (((b - a) ^ (c - a)) < 0) swap(b, c);
    function<long long(int)> sqr = [](int x) { return x
     * 1ll * x; };
    long long k11 = a.x - p.x, k12 = a.y - p.y, k13 =
    sqr(a.x) - sqr(p.x) + sqr(a.y) - sqr(p.y);
    long long k21 = b.x - p.x, k22 = b.y - p.y, k23 =
    sqr(b.x) - sqr(p.x) + sqr(b.y) - sqr(p.y);
    long long k31 = c.x - p.x, k32 = c.y - p.y, k33 =
    sqr(c.x) - sqr(p.x) + sqr(c.y) - sqr(p.y);
    long long det = k11 * (k22 * k33 - k23 * k32) - k12
     * (k21 * k33 - k23 * k31) + k13 * (k21 * k32 - k22
     * k31);
    return det > 0;
  }
  bool intersect(const point &a, const point &b, const
    point &c, const point &d) {
    return ((b - a) ^ (c - a)) * ((b - a) ^ (d - a)) <
    0 &&
        ((d - c) ^ (a - c)) * ((d - c) ^ (b - c)) < 0;
  }
  void dfs(int l, int r) {
    if (r - l <= 3) {
      for (int i = l; i < r; ++i) {
        for (int j = i + 1; j < r; ++j) add_edge(i, j);
      }
      return;
    }
    int m = (l + r) >> 1;
    dfs(l, m), dfs(m, r);
    int pl = l, pr = r - 1;
    while (true) {
      int z = -1;
      for (int u : g[pl]) {
        long long c = ((p[pl] - p[pr]) ^ (p[u] - p[pr])
    );
        if (c > 0 || c == 0 && abs(p[u] - p[pr]) < abs(
    p[pl] - p[pr])) {
          z = u;
          break;
        }
      }
      if (z != -1) {
        pl = z;
        continue;
      }
      for (int u : g[pr]) {
        long long c = ((p[pr] - p[pl]) ^ (p[u] - p[pl])
    );
        if (c < 0 || c == 0 && abs(p[u] - p[pl]) < abs(
    p[pr] - p[pl])) {
          z = u;
          break;
        }
      }
      if (z != -1) {
        pr = z;
        continue;
      }
      break;
    }
    add_edge(pl, pr);
    while (true) {
      int z = -1;
      bool b = false;
      for (int u : g[pl]) {
        long long c = ((p[pl] - p[pr]) ^ (p[u] - p[pr])
    );
        if (c < 0 && (z == -1 || inside(p[pl], p[pr], p
    [z], p[u]))) z = u;
      }
      for (int u : g[pr]) {
        long long c = ((p[pr] - p[pl]) ^ (p[u] - p[pl])
    );
        if (c > 0 && (z == -1 || inside(p[pl], p[pr], p
    [z], p[u]))) z = u, b = true;
      }
      if (z == -1) break;
      int x = pl, y = pr;
```

```
      if (b) swap(x, y);
      for (auto it = g[x].begin(); it != g[x].end(); )
    {
        int u = *it;
        if (intersect(p[x], p[u], p[y], p[z])) {
          it = g[x].erase(it);
          g[u].erase(x);
        } else {
          ++it;
        }
      }
      if (b) add_edge(pl, z), pr = z;
      else add_edge(pr, z), pl = z;
    }
  }
  vector<vector<int>> solve(vector<point> v) {
    int n = v.size();
    for (int i = 0; i < n; ++i) g[i].clear();
    for (int i = 0; i < n; ++i) o[i] = i;
    sort(o, o + n, [&](int i, int j) { return v[i] < v[
    j]; });
    p.resize(n);
    for (int i = 0; i < n; ++i) p[i] = v[o[i]];
    dfs(0, n);
    vector<vector<int>> res(n);
    for (int i = 0; i < n; ++i) {
      for (int j : g[i]) res[o[i]].push_back(o[j]);
    }
    return res;
  }
}
```

8.4 Sector Area

```
// calc area of sector which include a, b
double SectorArea(Point a, Point b, double r) {
  double theta = atan2(a.y, a.x) - atan2(b.y, b.x);
  while (theta <= 0) theta += 2 * pi;
  while (theta >= 2 * pi) theta -= 2 * pi;
  theta = min(theta, 2 * pi - theta);
  return r * r * theta / 2;
}
```

8.5 Polygon Area

```
// point sort in counterclockwise
double ConvexPolygonArea(vector<Point> &p, int n) {
  double area = 0;
  for (int i = 1; i < p.size() - 1; i++) area += Cross(
    p[i] - p[0], p[i + 1] - p[0]);
  return area / 2;
}
```

8.6 Half Plane Intersection

```
bool jizz(Line l1,Line l2,Line l3){
  Point p=intersect(l2,l3);
  return ((l1.pb-l1.pa)^(p-l1.pa))<-eps;
}

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

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

8.7 Rotating Sweep Line

```
void rotatingSweepLine(vector<pair<int,int>> &ps){
  int n=int(ps.size());
  vector<int> id(n),pos(n);
  vector<pair<int,int>> line(n*(n-1)/2);
  int m=-1;
  for(int i=0;i<n;++i)for(int j=i+1;j<n;++j)line[++m]=
    make_pair(i,j); ++m;
  sort(line.begin(),line.end(),[&](const pair<int,int>
    &a,const pair<int,int> &b)->bool{
    if(ps[a.first].first==ps[a.second].first)return 0;
    if(ps[b.first].first==ps[b.second].first)return 1;
    return (double)(ps[a.first].second-ps[a.second].
    second)/(ps[a.first].first-ps[a.second].first) < (
    double)(ps[b.first].second-ps[b.second].second)/(ps
    [b.first].first-ps[b.second].first);
  });
  for(int i=0;i<n;++i)id[i]=i;
  sort(id.begin(),id.end(),[&](const int &a,const int &
    b){ return ps[a]<ps[b]; });
  for(int i=0;i<n;++i)pos[id[i]]=i;

  for(int i=0;i<m;++i){
    auto l=line[i];
    // meow
    tie(pos[l.first],pos[l.second],id[pos[l.first]],id[
    pos[l.second]])=make_tuple(pos[l.second],pos[l.
    first],l.second,l.first);
  }
}
```

8.8 Triangle Center

```
Point TriangleCircumCenter(Point a, Point b, Point c) {
  Point res;
  double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
  double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
  double ax = (a.x + b.x) / 2;
  double ay = (a.y + b.y) / 2;
  double bx = (c.x + b.x) / 2;
  double by = (c.y + b.y) / 2;
  double r1 = (sin(a2) * (ax - bx) + cos(a2) * (by - ay
    )) / (sin(a1) * cos(a2) - sin(a2) * cos(a1));
  return Point(ax + r1 * cos(a1), ay + r1 * sin(a1));
}

Point TriangleMassCenter(Point a, Point b, Point c) {
  return (a + b + c) / 3.0;
}

Point TriangleOrthoCenter(Point a, Point b, Point c) {
  return TriangleMassCenter(a, b, c) * 3.0 -
    TriangleCircumCenter(a, b, c) * 2.0;
}

Point TriangleInnerCenter(Point a, Point b, Point c) {
  Point res;
  double la = len(b - c);
  double lb = len(a - c);
  double lc = len(a - b);
```

```
  res.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb +
     lc);
  res.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb +
     lc);
  return res;
}
```

8.9 Polygon Center

```
Point BaryCenter(vector<Point> &p, int n) {
  Point res(0, 0);
  double s = 0.0, t;
  for (int i = 1; i < p.size() - 1; i++) {
    t = Cross(p[i] - p[0], p[i + 1] - p[0]) / 2;
    s += t;
    res.x += (p[0].x + p[i].x + p[i + 1].x) * t;
    res.y += (p[0].y + p[i].y + p[i + 1].y) * t;
  }
  res.x /= (3 * s);
  res.y /= (3 * s);
  return res;
}
```

8.10 Maximum Triangle

```
double ConvexHullMaxTriangleArea(Point p[], int res[],
    int chnum) {
  double area = 0, tmp;
  res[chnum] = res[0];
  for (int i = 0, j = 1, k = 2; i < chnum; i++) {
    while (fabs(Cross(p[res[j]] - p[res[i]], p[res[(k +
     1) % chnum]] - p[res[i]])) > fabs(Cross(p[res[j]]
    - p[res[i]], p[res[k]] - p[res[i]]))) k = (k + 1) %
     chnum;
    tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] -
     p[res[i]]));
    if (tmp > area) area = tmp;
    while (fabs(Cross(p[res[(j + 1) % chnum]] - p[res[i
    ]], p[res[k]] - p[res[i]])) > fabs(Cross(p[res[j]]
    - p[res[i]], p[res[k]] - p[res[i]]))) j = (j + 1) %
     chnum;
    tmp = fabs(Cross(p[res[j]] - p[res[i]], p[res[k]] -
     p[res[i]]));
    if (tmp > area) area = tmp;
  }
  return area / 2;
}
```

8.11 Point in Polygon

```
bool on(point a, point b, point c) {
  if (a.x == b.x) {
    if (c.x != a.x) return false;
    if (c.y >= min(a.y, b.y) && c.y <= max(a.y, b.y))
    return true;
    return false;
  }
  if (((a - c) ^ (b - c)) != 0) return false;
  if (a.x > b.x) swap(a, b);
  if (c.x < min(a.x, b.x) || c.x > max(a.x, b.x))
    return false;
  return ((a - b) ^ (a - c)) == 0;
}

int sgn(long long x) {
  if (x > 0) return 1;
  if (x < 0) return -1;
  return 0;
}

bool in(const vector<point> &c, point p) {
  int last = -2;
  int n = c.size();
  for (int i = 0; i < c.size(); ++i) {
    if (on(c[i], c[(i + 1) % n], p)) return true;
    int g = sgn((c[i] - p) ^ (c[(i + 1) % n] - p));
    if (last == -2) last = g;
    else if (last != g) return false;
  }
  return true;
}

bool in(point a, point b, point c, point p) {
  return in({ a, b, c }, p);
}

bool inside(const vector<point> &ch, point t) {
  point p = ch[1] - ch[0];
  point q = t - ch[0];
  if ((p ^ q) < 0) return false;
  if ((p ^ q) == 0) {
    if (p * q < 0) return false;
    if (q.len() > p.len()) return false;
    return true;
  }
  p = ch[ch.size() - 1] - ch[0];
  if ((p ^ q) > 0) return false;
  if ((p ^ q) == 0) {
    if (p * q < 0) return false;
    if (q.len() > p.len()) return false;
    return true;
  }
  p = ch[1] - ch[0];
  double ang = acos(1.0 * (p * q) / p.len() / q.len());
  int d = 20, z = ch.size() - 1;
  while (d--) {
    if (z - (1 << d) < 1) continue;
    point p1 = ch[1] - ch[0];
    point p2 = ch[z - (1 << d)] - ch[0];
    double tang = acos(1.0 * (p1 * p2) / p1.len() / p2.
    len());
    if (tang >= ang) z -= (1 << d);
  }
  return in(ch[0], ch[z - 1], ch[z], t);
}
```

8.12 Circle-Line Intersection

```
// remove second level if to get points for line (
    defalut: segment)
void CircleCrossLine(Point a, Point b, Point o, double
    r, Point ret[], int &num) {
  double x0 = o.x, y0 = o.y;
  double x1 = a.x, y1 = a.y;
  double x2 = b.x, y2 = b.y;
  double dx = x2 - x1, dy = y2 - y1;
  double A = dx * dx + dy * dy;
  double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
  double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 -
    y0) - r * r;
  double delta = B * B - 4 * A * C;
  num = 0;
  if (epssgn(delta) >= 0) {
    double t1 = (-B - sqrt(fabs(delta))) / (2 * A);
    double t2 = (-B + sqrt(fabs(delta))) / (2 * A);
    if (epssgn(t1 - 1.0) <= 0 && epssgn(t1) >= 0) ret[
    num++] = Point(x1 + t1 * dx, y1 + t1 * dy);
    if (epssgn(t2 - 1.0) <= 0 && epssgn(t2) >= 0) ret[
    num++] = Point(x1 + t2 * dx, y1 + t2 * dy);
  }
}

vector<Point> CircleCrossLine(Point a, Point b, Point o
    , double r) {
  double x0 = o.x, y0 = o.y;
  double x1 = a.x, y1 = a.y;
  double x2 = b.x, y2 = b.y;
  double dx = x2- x1, dy = y2 - y1;
  double A = dx * dx + dy * dy;
  double B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
  double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 -
    y0) - r * r;
  double delta = B * B - 4 * A * C;
  vector<Point> ret;
  if (epssgn(delta) >= 0) {
    double t1 = (-B - sqrt(fabs(delta))) / (2 * A);
```

```
    double t2 = (-B + sqrt(fabs(delta))) / (2 * A);
    if (epssgn(t1 - 1.0) <= 0 && epssgn(t1) >= 0) ret.
    emplace_back(x1 + t1 * dx, y1 + t1 * dy);
    if (epssgn(t2 - 1.0) <= 0 && epssgn(t2) >= 0) ret.
    emplace_back(x1 + t2 * dx, y1 + t2 * dy);
  }
  return ret;
}
```

8.13 Circle-Triangle Intersection

```
// calc area intersect by circle with radius r and
    triangle OAB
double Calc(Point a, Point b, double r) {
  Point p[2];
  int num = 0;
  bool ina = epssgn(len(a) - r) < 0, inb = epssgn(len(b
    ) - r) < 0;
  if (ina) {
    if (inb) return fabs(Cross(a, b)) / 2.0; //
    triangle in circle
    else { // a point inside and another outside: calc
    sector and triangle area
      CircleCrossLine(a, b, Point(0, 0), r, p, num);
      return SectorArea(b, p[0], r) + fabs(Cross(a, p
    [0])) / 2.0;
    }
  } else {
    CircleCrossLine(a, b, Point(0, 0), r, p, num);
    if (inb) return SectorArea(p[0], a, r) + fabs(Cross
    (p[0], b)) / 2.0;
    else {
      if (num == 2) return SectorArea(a, p[0], r) +
    SectorArea(p[1], b, r) + fabs(Cross(p[0], p[1])) /
    2.0; // segment ab has 2 point intersect with
    circle
      else return SectorArea(a, b, r); // segment has
    no intersect point with circle
    }
  }
}
```

8.14 Polygon Diameter

```
// get diameter of p[res[]] store opposite points in
    app
double Diameter(Point p[], int res[], int chnum, int
    app[][2], int &appnum) {
  double ret = 0, nowlen;
  res[chnum] = res[0];
  appnum = 0;
  for (int i = 0, j = 1; i < chnum; ++i) {
    while (Cross(p[res[i]] - p[res[i + 1]], p[res[j +
    1]] - p[res[i + 1]]) < Cross(p[res[i]] - p[res[i +
    1]], p[res[j]] - p[res[i + 1]])) {
      ++j;
      j %= chnum;
    }
    app[appnum][0] = res[i];
    app[appnum][1] = res[j];
    ++appnum;
    nowlen = dis(p[res[i]], p[res[j]]);
    if (nowlen > ret) ret = nowlen;
    nowlen = dis(p[res[i + 1]], p[res[j + 1]]);
    if (nowlen > ret) ret = nowlen;
  }
  return ret;
}
```

8.15 Minimun Distance of 2 Polygons

```
// p, q is convex
double TwoConvexHullMinDist(Point P[], Point Q[], int n
    , int m) {
  int YMinP = 0, YMaxQ = 0;
  double tmp, ans = 999999999;
  for (i = 0; i < n; ++i) if(P[i].y < P[YMinP].y) YMinP
     = i;
  for (i = 0; i < m; ++i) if(Q[i].y > Q[YMaxQ].y) YMaxQ
     = i;
  P[n] = P[0], Q[m] = Q[0];
  for (int i = 0; i < n; ++i) {
    while (tmp = Cross(Q[YMaxQ + 1] - P[YMinP + 1], P[
    YMinP] - P[YMinP + 1]) > Cross(Q[YMaxQ] - P[YMinP +
     1], P[YMinP] - P[YMinP + 1])) YMaxQ = (YMaxQ + 1)
    % m;
    if (tmp < 0) ans = min(ans, PointToSegDist(P[YMinP
    ], P[YMinP + 1], Q[YMaxQ]));
    else ans = min(ans, TwoSegMinDist(P[YMinP], P[YMinP
     + 1], Q[YMaxQ], Q[YMaxQ + 1]));
    YMinP = (YMinP + 1) % n;
  }
  return ans;
}
```

8.16 2D Convex Hull

```
vector<point> convex(vector<point> p) {
  sort(p.begin(), p.end());
  vector<point> ch;
  for (int i = 0; i < n; ++i) {
    while (ch.size() >= 2 && ((p[i] - ch[ch.size() -
    2]) ^ (ch[ch.size() - 1] - ch[ch.size() - 2])) >=
    0) ch.pop_back();
    ch.push_back(p[i]);
  }
  int t = ch.size();
  for (int i = n - 2; i >= 0; --i) {
    while (ch.size() > t && ((p[i] - ch[ch.size() - 2])
     ^ (ch[ch.size() - 1] - ch[ch.size() - 2])) >= 0)
    ch.pop_back();
    ch.push_back(p[i]);
  }
  ch.pop_back();
  return ch;
}
```

8.17 3D Convex Hull

```
double absvol(const Point a,const Point b,const Point c
    ,const Point d){
  return abs(((b-a)^(c-a))*(d-a))/6;
}

struct convex3D{
static const int maxn=1010;
struct Triangle{
  int a,b,c;
  bool res;
  Triangle(){}
  Triangle(int a,int b,int c,bool res=1):a(a),b(b),c(c)
    ,res(res){}
};
int n,m;
Point p[maxn];
Triangle f[maxn*8];
int id[maxn][maxn];
bool on(Triangle &t,Point &pt){
  return ((p[t.c]-p[t.b])^(p[t.a]-p[t.b]))*(pt-p[t.a])>
    eps;
}
void meow(int pi,int a,int b){
  int f2=id[a][b];
  if(f[f2].res){
    if(on(f[f2],p[pi]))dfs(pi,f2);
    else{
      id[pi][b]=id[a][pi]=id[b][a]=m;
      f[m++]=Triangle(b,a,pi,1);
    }
  }
}
void dfs(int pi,int now){
  f[now].res=0;
  meow(pi,f[now].b,f[now].a);
```

```
    meow(pi,f[now].c,f[now].b);
    meow(pi,f[now].a,f[now].c);
  }
  void operator()(){
    if(n<4)return;
    if([&]()->int{
      for(int i=1;i<n;++i){
        if(abs(p[0]-p[i])>eps){
          swap(p[1],p[i]);
          return 0;
        }
      }
      return 1;
    }())return;
    if([&]()->int{
      for(int i=2;i<n;++i){
        if(abs((p[0]-p[i])^(p[1]-p[i]))>eps){
          swap(p[2],p[i]);
          return 0;
        }
      }
      return 1;
    }())return;
    if([&]()->int{
      for(int i=3;i<n;++i){
        if(abs(((p[1]-p[0])^(p[2]-p[0]))*(p[i]-p[0]))>eps
    ){
          swap(p[3],p[i]);
          return 0;
        }
      }
      return 1;
    }())return;
    for(int i=0;i<4;++i){
      Triangle tmp((i+1)%4,(i+2)%4,(i+3)%4,1);
      if(on(tmp,p[i]))swap(tmp.b,tmp.c);
      id[tmp.a][tmp.b]=id[tmp.b][tmp.c]=id[tmp.c][tmp.a]=
    m;
      f[m++]=tmp;
    }
    for(int i=4;i<n;++i){
      for(int j=0;j<m;++j){
        if(f[j].res && on(f[j],p[i])){
          dfs(i,j);
          break;
        }
      }
    }
    int mm=m; m=0;
    for(int i=0;i<mm;++i){
      if(f[i].res)f[m++]=f[i];
    }
  }
  bool same(int i,int j){
    return !(absvol(p[f[i].a],p[f[i].b],p[f[i].c],p[f[j].
    a])>eps || absvol(p[f[i].a],p[f[i].b],p[f[i].c],p[f
    [j].b])>eps || absvol(p[f[i].a],p[f[i].b],p[f[i].c
    ],p[f[j].c])>eps);
  }
  int faces(){
    int rt=0;
    for(int i=0;i<m;++i){
      int iden=1;
      for(int j=0;j<i;++j){
        if(same(i,j))iden=0;
      }
      rt+=iden;
    }
    return rt;
  }
} tb;
```

8.18 Rotating Caliper

```
struct pnt {
  int x, y;
  pnt(): x(0), y(0) {};
  pnt(int xx, int yy): x(xx), y(yy) {};
} p[maxn];
pnt operator-(const pnt &a, const pnt &b) { return pnt(
    b.x - a.x, b.y - a.y); }
int operator^(const pnt &a, const pnt &b) { return a.x
    * b.y - a.y * b.x; } //cross
int operator*(const pnt &a, const pnt &b) { return (a -
    b).x * (a - b).x + (a - b).y * (a - b).y; } //
    distance
int tb[maxn], tbz, rsd;

int dist(int n1, int n2){
  return p[n1] * p[n2];
}
int cross(int t1, int t2, int n1){
  return (p[t2] - p[t1]) ^ (p[n1] - p[t1]);
}
bool cmpx(const pnt &a, const pnt &b) { return a.x == b
    .x ? a.y < b.y : a.x < b.x; }

void RotatingCaliper() {
  sort(p, p + n, cmpx);
  for (int i = 0; i < n; ++i) {
    while (tbz > 1 && cross(tb[tbz - 2], tb[tbz - 1], i
    ) <= 0) --tbz;
    tb[tbz++] = i;
  }
  rsd = tbz - 1;
  for (int i = n - 2; i >= 0; --i) {
    while (tbz > rsd + 1 && cross(tb[tbz - 2], tb[tbz -
     1], i) <= 0) --tbz;
    tb[tbz++] = i;
  }
  --tbz;
  int lpr = 0, rpr = rsd;
  // tb[lpr], tb[rpr]
  while (lpr < rsd || rpr < tbz - 1) {
    if (lpr < rsd && rpr < tbz - 1) {
      pnt rvt = p[tb[rpr + 1]] - p[tb[rpr]];
      pnt lvt = p[tb[lpr + 1]] - p[tb[lpr]];
      if ((lvt ^ rvt) < 0) ++lpr;
      else ++rpr;
    }
    else if (lpr == rsd) ++rpr;
    else ++lpr;
    // tb[lpr], tb[rpr]
  }
}
```

8.19 Minimum Enclosing Circle

```
pt center(const pt &a, const pt &b, const pt &c) {
  pt p0 = b - a, p1 = c - a;
  double c1 = norm2(p0) * 0.5, c2 = norm2(p1) * 0.5;
  double d = p0 ^ p1;
  double x = a.x + (c1 * p1.y - c2 * p0.y) / d;
  double y = a.y + (c2 * p0.x - c1 * p1.x) / d;
  return pt(x, y);
}

circle min_enclosing(vector<pt> &p) {
  random_shuffle(p.begin(), p.end());
  double r = 0.0;
  pt cent;
  for (int i = 0; i < p.size(); ++i) {
    if (norm2(cent - p[i]) <= r) continue;
    cent = p[i];
    r = 0.0;
    for (int j = 0; j < i; ++j) {
      if (norm2(cent - p[j]) <= r) continue;
      cent = (p[i] + p[j]) / 2;
      r = norm2(p[j] - cent);
      for (int k = 0; k < j; ++k) {
        if (norm2(cent - p[k]) <= r) continue;
        cent = center(p[i], p[j], p[k]);
        r = norm2(p[k] - cent);
      }
    }
  }
  return circle(cent, sqrt(r));
}
```

8.20 Closest Pair

```
pt p[maxn];

double dis(const pt& a, const pt& b) {
  return sqrt((a - b) * (a - b));
}

double closest_pair(int l, int r) {
  if (l == r) return inf;
  if (r - l == 1) return dis(p[l], p[r]);
  int m = (l + r) >> 1;
  double d = min(closest_pair(l, m), closest_pair(m +
    1, r));
  vector<int> vec;
  for (int i = m; i >= l && fabs(p[m].x - p[i].x) < d;
    --i) vec.push_back(i);
  for (int i = m + 1; i <= r && fabs(p[m].x - p[i].x) <
     d; ++i) vec.push_back(i);
  sort(vec.begin(), vec.end(), [=](const int& a, const
    int& b) { return p[a].y < p[b].y; });
  for (int i = 0; i < vec.size(); ++i) {
    for (int j = i + 1; j < vec.size() && fabs(p[vec[j
    ]].y - p[vec[i]].y) < d; ++j) {
      d = min(d, dis(p[vec[i]], p[vec[j]]));
    }
  }
  return d;
}
```

9 Miscellaneous / Problems

9.1 Bitwise Hack

```
long long next_perm(long long v) {
  long long t = v | (v - 1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(
      v) + 1));
}

void subset(long long s) {
  long long sub = s;
  while (sub) {
    // do things
    sub = (sub - 1) & s;
  }
}
```

9.2 Hilbert's Curve (faster Mo's algorithm)

```
long long hilbert(int n, int x, int y) {
  long long res = 0;
  for (int s = n / 2; s; s >>= 1) {
    int rx = (x & s) > 0;
    int ry = (y & s) > 0;
    res += s * 1ll * s * ((3 * rx) ^ ry);
    if (ry == 0) {
      if (rx == 1) {
        x = s - 1 - x;
        y = s - 1 - y;
      }
      swap(x, y);
    }
  }
  return res;
}
```

9.3 Java

```
import java.io.*;
import java.util.*;
import java.lang.*;
import java.math.*;

public class filename{
  static Scanner in = new Scanner(System.in);
  public static void main(String[] args) {
    int t = 7122;
    while(in.hasNext()) {
      t = in.nextInt();
      float b = in.nextFloat();
      String str = in.nextLine(); // getline
      String stu = in.next(); // string
    }
    System.out.println("Case #" + t);
    System.out.printf("%d\n", 7122);
    int[] c = new int[5];
    int[][] d = {{7,1,2,2},{8,7}};
    int g = Integer.parseInt("-123");

    long f = (long)d[0][2];

    List<Integer> l = new ArrayList<>();
    Random rg = new Random();
    for (int i = 9; i >= 0; --i) {
      l.add(Integer.valueOf(rg.nextInt(100) + 1));
      l.add(Integer.valueOf((int)(Math.random() * 100)
+ 1));
    }
    Collections.sort(l, new Comparator<Integer>() {
      public int compare(Integer a, Integer b) {
        return a - b;
      }
    });
    for (int i = 0; i < l.size(); ++i) {
      System.out.print(l.get(i));
    }

    Set<String> s = new HashSet<String>(); // TreeSet
    s.add("jizz");
    System.out.println(s);
    System.out.println(s.contains("jizz"));

    Map<String, Integer> m = new HashMap<String,
    Integer>();
    m.put("lol", 7122);
    System.out.println(m);
    for(String key: m.keySet()) {
      System.out.println(key + " : " + m.get(key));
    }
    System.out.println(m.containsKey("lol"));
    System.out.println(m.containsValue(7122));

    System.out.println(Math.PI);
    System.out.println(Math.acos(-1));

    BigInteger bi = in.nextBigInteger(), bj = new
    BigInteger("-7122"), bk = BigInteger.valueOf(17171)
    ;
    bi = bi.add(bj);
    bi = bi.subtract(BigInteger.ONE);
    bi = bi.multiply(bj);
    bi = bi.divide(bj);
    bi = bi.and(bj);
    bi = bi.gcd(bj);
    bi = bi.max(bj);
    bi = bi.pow(10);
    int meow = bi.compareTo(bj); // -1 0 1
    String stz = "f5abd69150";
    BigInteger b16 = new BigInteger(stz, 16);
    System.out.println(b16.toString(2));
  }
}
```

9.4 Offline Dynamic MST

```
int cnt[maxn], cost[maxn], st[maxn], ed[maxn];
pair<int, int> qr[maxn];
// qr[i].first = id of edge to be changed, qr[i].second
     = weight after operation
// cnt[i] = number of operation on edge i
// call solve(0, q - 1, v, 0), where v contains edges i
     such that cnt[i] == 0
```

```cpp
void contract(int l, int r, vector<int> v, vector<int>
    &x, vector<int> &y) {
  sort(v.begin(), v.end(), [&](int i, int j) {
    if (cost[i] == cost[j]) return i < j;
    return cost[i] < cost[j];
  });
  djs.save();
  for (int i = l; i <= r; ++i) djs.merge(st[qr[i].first
    ], ed[qr[i].first]);
  for (int i = 0; i < (int)v.size(); ++i) {
    if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
      x.push_back(v[i]);
      djs.merge(st[v[i]], ed[v[i]]);
    }
  }
  djs.undo();
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) djs.merge(st[
    x[i]], ed[x[i]]);
  for (int i = 0; i < (int)v.size(); ++i) {
    if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
      y.push_back(v[i]);
      djs.merge(st[v[i]], ed[v[i]]);
    }
  }
  djs.undo();
}

void solve(int l, int r, vector<int> v, long long c) {
  if (l == r) {
    cost[qr[l].first] = qr[l].second;
    if (st[qr[l].first] == ed[qr[l].first]) {
      printf("%lld\n", c);
      return;
    }
    int minv = qr[l].second;
    for (int i = 0; i < (int)v.size(); ++i) minv = min(
    minv, cost[v[i]]);
    printf("%lld\n", c + minv);
    return;
  }
  int m = (l + r) >> 1;
  vector<int> lv = v, rv = v;
  vector<int> x, y;
  for (int i = m + 1; i <= r; ++i) {
    cnt[qr[i].first]--;
    if (cnt[qr[i].first] == 0) lv.push_back(qr[i].first
    );
  }
  contract(l, m, lv, x, y);
  long long lc = c, rc = c;
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) {
    lc += cost[x[i]];
    djs.merge(st[x[i]], ed[x[i]]);
  }
  solve(l, m, y, lc);
  djs.undo();
  x.clear(), y.clear();
  for (int i = m + 1; i <= r; ++i) cnt[qr[i].first]++;
  for (int i = l; i <= m; ++i) {
    cnt[qr[i].first]--;
    if (cnt[qr[i].first] == 0) rv.push_back(qr[i].first
    );
  }
  contract(m + 1, r, rv, x, y);
  djs.save();
  for (int i = 0; i < (int)x.size(); ++i) {
    rc += cost[x[i]];
    djs.merge(st[x[i]], ed[x[i]]);
  }
  solve(m + 1, r, y, rc);
  djs.undo();
  for (int i = l; i <= m; ++i) cnt[qr[i].first]++;
}
```

9.5 Manhattan Distance MST

```cpp
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e5 + 5;
int x[maxn], y[maxn], fa[maxn];
pair<int, int> bit[maxn];
vector<tuple<int, int, int>> ed;

void init() {
  for (int i = 0; i < maxn; ++i)
    bit[i] = make_pair(1e9, -1);
}

void add(int p, pair<int, int> v) {
  for (; p < maxn; p += p & -p)
    bit[p] = min(bit[p], v);
}

pair<int, int> query(int p) {
  pair<int, int> res = make_pair(1e9, -1);
  for (; p; p -= p & -p)
    res = min(res, bit[p]);
  return res;
}

void add_edge(int u, int v) {
  ed.emplace_back(u, v, abs(x[u] - x[v]) + abs(y[u] - y
    [v]));
}

void solve(int n) {
  init();
  vector<int> v(n), ds;
  for (int i = 0; i < n; ++i) {
    v[i] = i;
    ds.push_back(x[i] - y[i]);
  }
  sort(ds.begin(), ds.end());
  ds.resize(unique(ds.begin(), ds.end()) - ds.begin());
  sort(v.begin(), v.end(), [&](int i, int j) { return x
    [i] == x[j] ? y[i] > y[j] : x[i] > x[j]; });
  int j = 0;
  for (int i = 0; i < n; ++i) {
    int p = lower_bound(ds.begin(), ds.end(), x[v[i]] -
     y[v[i]]) - ds.begin() + 1;
    pair<int, int> q = query(p);
    if (~q.second) add_edge(v[i], q.second);
    add(p, make_pair(x[v[i]] + y[v[i]], v[i]));
  }
}

int find(int x) {
  if (x == fa[x]) return x;
  return fa[x] = find(fa[x]);
}

void merge(int x, int y) {
  fa[find(x)] = find(y);
}

int main() {
  int n; scanf("%d", &n);
  for (int i = 0; i < n; ++i) scanf("%d %d", &x[i], &y[
    i]);
  solve(n);
  for (int i = 0; i < n; ++i) swap(x[i], y[i]);
  solve(n);
  for (int i = 0; i < n; ++i) x[i] = -x[i];
  solve(n);
  for (int i = 0; i < n; ++i) swap(x[i], y[i]);
  solve(n);
  sort(ed.begin(), ed.end(), [](const tuple<int, int,
    int> &a, const tuple<int, int, int> &b) {
    return get<2>(a) < get<2>(b);
  });
  for (int i = 0; i < n; ++i) fa[i] = i;
  long long ans = 0;
  for (int i = 0; i < ed.size(); ++i) {
    int x, y, w; tie(x, y, w) = ed[i];
    if (find(x) == find(y)) continue;
    merge(x, y);
    ans += w;
  }
  printf("%lld\n", ans);
```

```
  return 0;
}
```

9.6 "Dynamic" Kth Element (parallel binary search)

```
struct query { int op, l, r, k, qid; };
// op = 1: insertion (l = pos, r = val)
// op = 2: deletion (l = pos, r = val)
// op = 3: query

void bs(vector<query> &qry, int l, int r) {
  // answer to queries in qry are from l to r
  if (l == r) {
    for (int i = 0; i < qry.size(); ++i) {
      if (qry[i].op == 3) ans[qry[i].qid] = l;
    }
    return;
  }
  if (qry.size() == 0) return;
  int m = l + r >> 1;
  for (int i = 0; i < qry.size(); ++i) {
    if (qry[i].op == 1 && qry[i].r <= m) bit.add(qry[i
    ].l, 1);
    else if (qry[i].op == 2 && qry[i].r <= m) bit.add(
    qry[i].l, -1);
    else if (qry[i].op == 3) tmp[qry[i].qid] += bit.qry
    (qry[i].r) - bit.qry(qry[i].l - 1);
  }
  vector<query> ql, qr;
  for (int i = 0; i < qry.size(); ++i) {
    if (qry[i].op == 3) {
      if (qry[i].k - tmp[qry[i].qid] > 0) qry[i].k -=
    tmp[qry[i].qid], qr.push_back(qry[i]);
      else ql.push_back(qry[i]);
      tmp[qry[i].qid] = 0;
      continue;
    }
    if (qry[i].r <= m) ql.push_back(qry[i]);
    else qr.push_back(qry[i]);
  }
  for (int i = 0; i < qry.size(); ++i) {
    if (qry[i].op == 1 && qry[i].r <= m) bit.add(qry[i
    ].l, -1);
    else if (qry[i].op == 2 && qry[i].r <= m) bit.add(
    qry[i].l, 1);
  }
  bs(ql, l, m), bs(qr, m + 1, r);
}
```

9.7 Dynamic Kth Element (persistent segment tree)

```
// segtree: persistant segment tree which supports
    range sum query

void init(int n) {
  segtree::sz = 0;
  bit[0] = segtree::build(0, ds.size());
  for (int i = 1; i <= n; ++i) bit[i] = bit[0];
}

void add(int p, int n, int x, int v) {
  for (; p <= n; p += p & -p)
    bit[p] = segtree::modify(0, ds.size(), x, v, bit[p
    ]);
}

vector<int> query(int p) {
  vector<int> z;
  for (; p; p -= p & -p)
    z.push_back(bit[p]);
  return z;
}

int dfs(int l, int r, vector<int> lz, vector<int> rz,
    int k) {
  if (r - l == 1) return l;
  int ls = 0, rs = 0;
  for (int i = 0; i < lz.size(); ++i) ls += segtree::st
    [segtree::lc[lz[i]]];
  for (int i = 0; i < rz.size(); ++i) rs += segtree::st
    [segtree::lc[rz[i]]];
  if (rs - ls >= k) {
    for (int i = 0; i < lz.size(); ++i) lz[i] = segtree
    ::lc[lz[i]];
    for (int i = 0; i < rz.size(); ++i) rz[i] = segtree
    ::lc[rz[i]];
    return dfs(l, (l + r) / 2, lz, rz, k);
  } else {
    for (int i = 0; i < lz.size(); ++i) lz[i] = segtree
    ::rc[lz[i]];
    for (int i = 0; i < rz.size(); ++i) rz[i] = segtree
    ::rc[rz[i]];
    return dfs((l + r) / 2, r, lz, rz, k - (rs - ls));
  }
}

int main() {
  int n, q; scanf("%d %d", &n, &q);
  for (int i = 1; i <= n; ++i) scanf("%d", &a[i]), ds.
    push_back(a[i]);
  for (int i = 0; i < q; ++i) {
    int a, b, c; scanf("%d %d %d", &a, &b, &c);
    vector<int> v = { a, b, c };
    if (a == 1) {
      int d; scanf("%d", &d);
      v.push_back(d);
    }
    qr.push_back(v);
  }
  for (int i = 0; i < q; ++i) if (qr[i][0] == 2) ds.
    push_back(qr[i][2]);
  sort(ds.begin(), ds.end()), ds.resize(unique(ds.begin
    (), ds.end()) - ds.begin());
  for (int i = 1; i <= n; ++i) a[i] = lower_bound(ds.
    begin(), ds.end(), a[i]) - ds.begin();
  for (int i = 0; i < q; ++i) if (qr[i][0] == 2) qr[i
    ][2] = lower_bound(ds.begin(), ds.end(), qr[i][2])
    - ds.begin();
  init(n);
  for (int i = 1; i <= n; ++i) add(i, n, a[i], 1);
  for (int i = 0; i < q; ++i) {
    if (qr[i][0] == 3) {
      puts("7122");
      continue;
    }
    if (qr[i][0] == 1) {
      vector<int> lz = query(qr[i][1] - 1);
      vector<int> rz = query(qr[i][2]);
      int ans = dfs(0, ds.size(), lz, rz, qr[i][3]);
      printf("%d\n", ds[ans]);
    } else {
      add(qr[i][1], n, a[qr[i][1]], -1);
      add(qr[i][1], n, qr[i][2], 1);
      a[qr[i][1]] = qr[i][2];
    }
  }
  return 0;
}
```