University of Dhaka

DU_Antifragile

Asif Jawad, Shakil Muhit, Yeamin Kaiser

Notebook for ICPC Dhaka Regional 2023

Updated: November 2023

Contents

1 DP

1.1 AlienTrick

Description: Divide n elements into k groups and minimize the sum of a cost function for each group. If $f_n(k) \geq f_n(k+1)$ (monotone) and $f_n(k-1) - f_n(k) \geq f_n(k) - f_n(k+1)$ (convex) we can apply alien trick. It means with the increase of k, answer becomes more optimal, but the rate of becoming optimal slows down. We define $g_n = f_n(k) + kC$. That means we need to add a penalty C for each group we use. We can binary search on C as g is convex. Compute g_n and the optimal k on each iteration – k decreases when C increases.
Time: $\mathcal{O}(n \log \max)$

1.2 CHT

Description:

- If m is decreasing:
  - for min : bad(s-3, s-2, s-1), x increasing
  - for max : bad(s-1, s-2, s-3), x decreasing
- If m is increasing:
  - for max : bad(s-3, s-2, s-1), x increasing
  - for min : bad(s-1, s-2, s-3), x decreasing
- If x isn't monotonic, then do Ternary Search or keep intersections and do binary search

```
struct CHT {
  vector<ll> m, b;
  int ptr = 0;
  bool bad(int l1, int l2, int l3) { // returns intersect(l1, l3) <= intersect(l1, l2)
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2]) <= 1.0 * (b[l2] - b[l1]) * (m[l1] - m[l3]);
  }
  void insert_line(ll _m, ll _b) {
    m.push_back(_m);
    b.push_back(_b);
    int s = m.size();
    while (s >= 3 && bad(s - 1, s - 2, s - 3)) {
      s--;
      m.erase(m.end() - 2);
      b.erase(b.end() - 2);
    }
  }
  ll f(int i, ll x) { return m[i] * x + b[i]; }
  ll eval(ll x) {
    if (ptr >= m.size()) ptr = m.size() - 1;
    while (ptr < m.size() - 1 && f(ptr + 1, x) > f(ptr, x)) ptr++;
    return f(ptr, x);
  }
};
```

1.3 DnC Optimization

Description:

- $dp[i][j] = \min_{k<j}\{dp[i-1][k] + C[k][j]\}$
- $A[i][j] \leq A[i][j+1]$
- $O(kn^2)$ to $O(kn\log n)$
- sufficient: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$, $a \leq b \leq c \leq d$ (QI)

```
void compute(int L, int R, int optL, int optR) {
  if (L > R) return;
  int M = L + R >> 1;
  pair<ll, int> best(1LL << 60, -1);
  for (int k = optL; k <= min(M, optR); k++) {
    best = min(best, {dp[prv][k] + C[k + 1][M], k});
  }
  dp[now][M] = best.ff;
  compute(L, M - 1, optL, best.ss);
  compute(M + 1, R, best.ss, optR);
}
```

1.4 Dynamic CHT

```
const ll is_query = -LLONG_MAX;
struct Line {
  ll m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line&rhs) const {
    if (rhs.b != is_query) return m < rhs.m;
    const Line * s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
struct HullDynamic : public multiset<Line> {
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end()) return y->m == x->m && y->b <= x->b;
    //may need to use __int128 instead of ld if supported
    return ld(x->b - y->b) * (z->m - y->m) >= ld(y->b - z->b) * (y->m - x->m);
  }
  void insert_line(ll m, ll b) {
    auto y = insert({ -m, -b }); //change here for max
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
    y->succ = [ = ] { return next(y) == end() ? 0 : &*next(y); };
    while (y != begin() && bad(prev(y))) erase(prev(y));
    if (y != begin()) prev(y)->succ = [ = ] { return &*y; };
  }
  ll eval(ll x) {
    auto l = *lower_bound((Line) { x, is_query });
    return -(l.m * x + l.b); //change here for max
  }
} hull;
```

1.5 Knuth Optimization

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \leq f(a,d)$ and $f(a,c) + f(b,d) \leq f(a,d) + f(b,c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer, monotone queues, ternary search.

Time: $\mathcal{O}(N^2)$

1.6 Li Chao Tree

Description: Add line segment, query minimum y at some x. Provide list of all query x points to constructor (offline solution). Use add_segment(line, l,r) to add a line segment $y = ax + b$ defined by $x \in [l,r)$. Use query(x) to get min at x.

Time: $\mathcal{O}(\log n)$

```
struct LiChaoTree {
  using Line = pair <ll, ll>;
  const ll linf = numeric_limits<ll>::max();
  int n; vector<ll> xl; vector<Line> dat;
  LiChaoTree(const vector<ll>& _xl) : xl(_xl) {
    n = 1; while(n < xl.size())n <<= 1;
    xl.resize(n,xl.back());
    dat = vector<Line>(2 * n - 1,Line(0,linf));
  }
  ll eval(Line f,ll x){return f.first * x + f.second;}
  void _add_line(Line f,int k,int l,int r){
    while (l != r) {
      int m = (l + r) / 2;
      ll lx = xl[l],mx = xl[m],rx = xl[r - 1];
      Line &g = dat[k];
      if(eval(f,lx) < eval(g,lx) && eval(f,rx) < eval(g,rx)){
        g = f; return;
      }
      if(eval(f,lx) >= eval(g,lx) && eval(f,rx) >= eval(g,rx))return;
      if(eval(f,mx) < eval(g,mx))swap(f,g);
      if(eval(f,lx) < eval(g,lx)) k = k * 2 + 1, r = m;
      else k = k * 2 + 2, l = m;
    }
  }
  void add_line(Line f){_add_line(f,0,0,n);}
  void add_segment(Line f,ll lx,ll rx){
    int l = lower_bound(xl.begin(), xl.end(),lx) - xl.begin();
    int r = lower_bound(xl.begin(), xl.end(),rx) - xl.begin();
    int a0 = l, b0 = r, sz = 1; l += n;r += n;
    while(l < r){
      if(r & 1) r--, b0 -= sz, _add_line(f,r - 1,b0,b0 + sz);
      if(l & 1) _add_line(f,l - 1,a0,a0 + sz), l++, a0 += sz;
      l >>= 1, r >>= 1, sz <<= 1;
    }
  }
  ll query(ll x) {
    int i = lower_bound(xl.begin(), xl.end(),x) - xl.begin();
    i += n - 1; ll res = eval(dat[i],x);
    while (i) i = (i - 1) / 2, res = min(res, eval(dat[i], x));
    return res;
  }
};
```

1.7 SOS DP

Description: Fast sum over subsets $g_i = \sum_{j\subseteq i} f_j$ and supersets $g_i = \sum_{i\subseteq j} f_j$.

Time: $\mathcal{O}(n2^n)$

```
for (int mask = 0; mask < 1 << n; mask++) f[mask] = a[mask];
// sum over subsets
for (int i = 0; i < n; ++i) {
  for (int mask = 0; mask < 1 << n; ++mask) {
    if (mask & 1 << i) f[mask] += f[mask ^ 1 << i];
  }
}
// sum over supersets
for (int i = 0; i < n; ++i) {
  for (int mask = (1 << n) - 1; mask >= 0; --mask) {
    if (~mask & 1 << i) f[mask] += f[mask | 1 << i];
  }
}
```

2 Data Structures

2.1 BIT 2D with Range Update and Range Query

```
using namespace std;
const int N = 1010;
struct BIT2D {
  long long M[N][N][2], A[N][N][2];
  BIT2D() {
    memset(M, 0, sizeof M);
    memset(A, 0, sizeof A);
  }
  void upd2(long long t[N][N][2], int x, int y, long long mul, long long add) {
    for(int i = x; i < N; i += i & -i) {
      for(int j = y; j < N; j += j & -j) {
        t[i][j][0] += mul;
        t[i][j][1] += add;
      }
    }
  }
  void upd1(int x, int y1, int y2, long long mul, long long add) {
    upd2(M, x, y1, mul, -mul * (y1 - 1));
    upd2(M, x, y2, -mul, mul * y2);
    upd2(A, x, y1, add, -add * (y1 - 1));
    upd2(A, x, y2, -add, add * y2);
  }
  void upd(int x1, int y1, int x2, int y2, long long val) {
    upd1(x1, y1, y2, val, -val * (x1 - 1));
```

```
    upd1(x2, y1, y2, -val, val * x2);
  }
  long long query2(long long t[N][N][2], int x, int y) {
    long long mul = 0, add = 0;
    for(int i = y; i > 0; i -= i & -i) {
      mul += t[x][i][0];
      add += t[x][i][1];
    }
    return mul * y + add;
  }
  long long query1(int x, int y) {
    long long mul = 0, add = 0;
    for(int i = x; i > 0; i -= i & -i) {
      mul += query2(M, i, y);
      add += query2(A, i, y);
    }
    return mul * x + add;
  }
  long long query(int x1, int y1, int x2, int y2) {
    return query1(x2, y2) - query1(x1 - 1, y2) - query1(x2,
    ↪ y1 - 1) + query1(x1 - 1, y1 - 1);
  }
} t;
int main() {
  int n, m;
  cin >> n >> m;
  for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= m; j++) {
      int k;
      cin >> k;
      t.upd(i, j, i, j, k);
    }
  }
  int q;
  cin >> q;
  while(q--) {
    int ty, x1, y1, x2, y2;
    cin >> ty;
    if(ty == 1) {
      long long val;
      cin >> x1 >> y1 >> x2 >> y2 >> val;
      t.upd(x1, y1, x2, y2, val); // add val from
      ↪ top-left(x1, y1) to bottom-right (x2, y2);
    } else {
      cin >> x1 >> y1 >> x2 >> y2;
      cout << t.query(x1, y1, x2, y2) << '\n'; // output
      ↪ sum from top-left(x1, y1) to bottom-right (x2,
      ↪ y2);
    }
  }
  return 0;
}
```

2.2 BIT

```
//1-based
void update(int n, int idx, int v){
  while(idx <= n) tree[idx] += v, idx += idx & (-idx);
}
int query(int idx){
  int sum = 0;
  while(idx > 0) sum += tree[idx], idx -= idx & (-idx);
  return sum;
}
```

2.3 Block Cut Tree

Description: finds all vertex-biconnected components and compresses them into a tree.
Time: $\mathcal{O}(V+E)$

```
bitset <N> art, good;
vector <int> g[N], tree[N], st, comp[N];
int n, m, ptr, cur, in[N], low[N], id[N];
void dfs (int u, int from = -1) {
  in[u] = low[u] = ++ptr;
  st.emplace_back(u);
  for (int v : g[u]) if (v ^ from) {
    if (!in[v]) {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] >= in[u]) {
        art[u] = in[u] > 1 or in[v] > 2;
        comp[++cur].emplace_back(u);
        while (comp[cur].back() ^ v) {
          comp[cur].emplace_back(st.back());
          st.pop_back();
        }
      }
    } else { low[u] = min(low[u], in[v]); }
  }
}
void buildTree() {
  ptr = 0;
  for (int i = 1; i <= n; ++i) {
    if (art[i]) id[i] = ++ptr;
  }
  for (int i = 1; i <= cur; ++i) {
    int x = ++ptr;
    for (int u : comp[i]) {
      if (art[u]) {
        tree[x].emplace_back(id[u]);
        tree[id[u]].emplace_back(x);
      } else { id[u] = x; }
    }
  }
}
int main() {
  for (int i = 1; i <= n; ++i) { if (!in[i]) dfs(i); }
  buildTree();
}
```

2.4 Bridge Tree

Description: finds all edge-biconnected components and compresses them into a tree.
Time: $\mathcal{O}(V+E)$

```
vector <int> g[N], tree[N];
int n, m, in[N], low[N], ptr, compID[N];
void go (int u, int par = -1) {
  in[u] = low[u] = ++ptr;
  for (int v : g[u]) {
    if (in[v]) {
      if (v == par) par = -1;
      else low[u] = min(low[u], in[v]);
    } else {
      go(v, u); low[u] = min(low[u], low[v]);
    }
  }
}
void shrink (int u, int id) {
  compID[u] = id;
  for (int v : g[u]) if (!compID[v]) {
    if (low[v] > in[u]) {
      tree[id].emplace_back(++ptr);
      shrink(v, ptr);
    } else { shrink(v, id); }
  }
}
int main() {
  cin >> n >> m;
  while (m--) {
    int u, v;
    scanf("%d %d", &u, &v);
    g[u].emplace_back(v);
    g[v].emplace_back(u);
  }
  for (int i = 1; i <= n; ++i) if (!in[i]) go(i);
  vector <int> roots; ptr = 0;
  for (int i = 1; i <= n; ++i) if (!compID[i]) {
    roots.emplace_back(++ptr);
    shrink(i, ptr);
  }
}
```

2.5 Centroid Decomposition

Description: all path problem to paths through root problem with $O(\log n)$ overhead.
Time: $\mathcal{O}(n \log n)$

```
bool bad[N]; ll ans, h[N];
vector <pair <int, ll>> g[N];
int n, sub[N], flat[N], ptr, in[N], out[N];
void trav (int u, int par = -1) {
  sub[u] = 1;
  for (auto [v, w] : g[u]) if (!bad[v] and v != par) {
    trav(v, u); sub[u] += sub[v];
  }
}
int getCentroid (int u, int tot, int par = -1) {
  for (auto [v, w] : g[u]) if (!bad[v] and v != par and
  ↪ sub[v] > tot / 2) {
    return getCentroid(v, tot, u);
  } return u;
}
void dfs (int u, int par = -1, ll far = 0) {
  flat[++ptr] = u, in[u] = ptr, h[u] = far;
  for (auto [v, w] : g[u]) if (!bad[v] and v != par) {
    dfs(v, u, far + w);
  } out[u] = ptr;
}
void decompose (int u = 1) {
  trav(u);
  int cen = getCentroid(u, sub[u]);
  ptr = 0; dfs(cen);
  for (auto [u, w] : g[cen]) if (!bad[u]) {
    for (int i = in[u]; i <= out[u]; ++i) {
      int v = flat[i];
      // update ans with v
    }
    for (int i = in[u]; i <= out[u]; ++i) {
      int v = flat[i];
      // insert v in data structure
    }
  }
  bad[cen] = 1;
  for (auto [v, w] : g[cen]) if (!bad[v]) decompose(v);
}
int main() {
  // input graph
  decompose();
}
```

2.6 HLD

Description: Transforms tree paths into ranges with $O(\log N)$ overhead. Subtree of v corresponds to segment [in[v], out[v]) and the path from v to the last vertex in ascending heavy path from v (which is nxt[v]) will be [in[nxt[v]], in[v]]. Use appropriate DS to handle stuff.
Time: $\mathcal{O}(\log^2 N)$

```
const int N = 1e5 + 5;
int par[N], nxt[N], in[N], out[N], sz[N], h[N];
int n, timer;
vector<int> g[N];
void dfs_sz(int u = 0, int p = -1, int d = 0) {
  par[u] = p, sz[u] = 1, h[u] = d;
  for (auto &v : g[u]) {
    if (v ^ p) {
      dfs_sz(v, u, d + 1);
      sz[u] += sz[v];
      if (sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
    }
  }
}
void dfs_hld(int u = 0, int p = -1) {
  update(1, 0, n - 1, timer, val[u]);
  in[u] = timer++;
  for (auto v : g[u]) {
    if (v ^ p) {
```

```cpp
      nxt[v] = (v == g[u][0]) ? nxt[u] : v;
      dfs_hld(v, u);
    }
  }
  out[u] = timer;
}
int hld_query(int u, int v) {
  int ret = 0;
  while (nxt[u] != nxt[v]) {
    if (h[nxt[u]] > h[nxt[v]]) swap(u, v);
    ret = merge(ret, query(1, 0, n - 1, in[nxt[v]], in[v]));
    v = par[nxt[v]];
  }
  if (h[u] > h[v]) swap(u, v);
  //in[u] -> in[u] + 1 in case of edge values
  ret = merge(ret, query(1, 0, n - 1, in[u], in[v]));
  return ret;
}
```

2.7 Interval Container

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

```cpp
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
 if (L == R) return is.end();
 auto it = is.lower_bound({L, R}), before = it;
 while (it != is.end() && it->first <= R) {
  R = max(R, it->second); before = it = is.erase(it);
 }
 if (it != is.begin() && (--it)->second >= L) {
   L = min(L, it->first); R = max(R, it->second);
   ↪ is.erase(it);
 } return is.insert(before, {L,R});
}
void removeInterval(set<pii>& is, int L, int R) {
 if (L == R) return;
 auto it = addInterval(is, L, R); auto r2 = it->second;
 if (it->first == L) is.erase(it); else (int&)it->second =
   ↪ L;
 if (R != r2) is.emplace(R, r2);
}
```

2.8 Interval Cover

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

```cpp
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
 vi S(sz(I)), R; iota(all(S), 0);
 sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
 T cur = G.first; int at = 0;
 while (cur < G.second) { // (A)
  pair<T, int> mx = make_pair(cur, -1);
  while (at < sz(I) && I[S[at]].first <= cur) {
   mx = max(mx, make_pair(I[S[at]].second, S[at])); at++;
  }
  if (mx.second == -1) return {};
  cur = mx.first; R.push_back(mx.second);
 } return R;
}
```

2.9 LCA

```cpp
const int N = 1e5 + 5;
const int LOGN = 18;
int h[N], table[LOGN][N];
std::vector<int> g[N];
void dfs(int u = 0, int p = -1, int d = 0){
  table[0][u] = p, h[u] = d;
  for(int i = 1; i < LOGN; i++){
    if(table[i - 1][u] ^ -1) {
      table[i][u] = table[i - 1][table[i - 1][u]];
    }
    else table[i][u] = -1;
  }
  for(auto v : g[u]){
    if(v ^ p) dfs(v, u, d + 1);
  }
}
int get_lca(int u, int v) {
  if (h[u] < h[v]) swap(u, v);
  for (int i = LOGN - 1; i >= 0; i--) {
    if (h[u] - (1 << i) >= h[v]) u = table[i][u];
  }
  if (u == v) return u;
  for (int i = LOGN - 1; i >= 0; i--) {
    if (table[i][u] != table[i][v]) {
      u = table[i][u]; v = table[i][v];
    }
  }
  return table[0][u];
}
```

2.10 Link Cut Tree

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.
Time: All operations take amortized $\mathcal{O}(\log N)$.

```cpp
struct Node { // Splay tree. Root's pp contains tree's
  ↪ parent.
 Node *p = 0, *pp = 0, *c[2];
 bool flip = 0;
 Node() { c[0] = c[1] = 0; fix(); }
 void fix() {
  if (c[0]) c[0]->p = this;
  if (c[1]) c[1]->p = this;
  // (+ update sum of subtree elements etc. if wanted)
 }
 void pushFlip() {
  if (!flip) return;
  flip = 0; swap(c[0], c[1]);
  if (c[0]) c[0]->flip ^= 1;
  if (c[1]) c[1]->flip ^= 1;
 }
 int up() { return p ? p->c[1] == this : -1; }
 void rot(int i, int b) {
  int h = i ^ b;
  Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
  if ((y->p = p)) p->c[up()] = y;
  c[i] = z->c[i ^ 1];
  if (b < 2) {
   x->c[h] = y->c[h ^ 1];
   z->c[h ^ 1] = b ? x : this;
  }
  y->c[i ^ 1] = b ? this : x;
  fix(); x->fix(); y->fix();
  if (p) p->fix(); swap(pp, y->pp);
 }
 void splay() {
  for (pushFlip(); p; ) {
   if (p->p) p->p->pushFlip();
   p->pushFlip(); pushFlip();
   int c1 = up(), c2 = p->up();
   if (c2 == -1) p->rot(c1, 2);
   else p->p->rot(c2, c1 != c2);
  }
 }
 Node* first() {
  pushFlip();
  return c[0] ? c[0]->first() : (splay(), this);
 }
};
struct LinkCut {
 vector<Node> node;
 LinkCut(int N) : node(N) {}
 void link(int u, int v) { // add an edge (u, v)
  assert(!connected(u, v));
  makeRoot(&node[u]);
  node[u].pp = &node[v];
 }
 void cut(int u, int v) { // remove an edge (u, v)
  Node *x = &node[u], *top = &node[v];
  makeRoot(top); x->splay();
  assert(top == (x->pp ?: x->c[0]));
  if (x->pp) x->pp = 0;
  else { x->c[0] = top->p = 0; x->fix(); }
 }
 bool connected(int u, int v) { // are u, v in the same
  ↪ tree?
  Node* nu = access(&node[u])->first();
  return nu == access(&node[v])->first();
 }
 void makeRoot(Node* u) {
  access(u); u->splay();
  if(u->c[0]) {
   u->c[0]->p = 0; u->c[0]->flip ^= 1;
   u->c[0]->pp = u; u->c[0] = 0; u->fix();
  }
 }
 Node* access(Node* u) {
  u->splay();
  while (Node* pp = u->pp) {
   pp->splay(); u->pp = 0;
   if (pp->c[1]) {
    pp->c[1]->p = 0; pp->c[1]->pp = pp; }
   pp->c[1] = u; pp->fix(); u = pp;
  } return u;
 }
};
```

2.11 MO's Algorithm

```cpp
//Hilbert Ordering for Mo's Algorithm
inline int64_t hilbertOrder(int x, int y, int pow, int
  ↪ rotate) {
  if (pow == 0) {
    return 0;
  }
  int hpow = 1 << (pow - 1);
  int seg = (x < hpow) ? ((y < hpow) ? 0 : 3) :
            ((y < hpow) ? 1 : 2);
  seg = (seg + rotate) & 3;
  const int rotateDelta[4] = {3, 0, 0, 1};
  int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
  int nrot = (rotate + rotateDelta[seg]) & 3;
  int64_t subSquareSize = int64_t(1) << (2 * pow - 2);
  int64_t ans = seg * subSquareSize;
  int64_t add = hilbertOrder(nx, ny, pow - 1, nrot);
  ans += (seg == 1 || seg == 2) ? add : (subSquareSize -
    ↪ add - 1);
  return ans;
}
struct Query {
  int l, r, idx; // queries
  int64_t ord; // Gilbert order of a query
  // call query[i].calcOrder() to calculate the Gilbert
    ↪ orders
  inline void calcOrder() {
    ord = hilbertOrder(l, r, 21, 0);
  }
};
// sort the queries based on the Gilbert order
inline bool operator<(const Query &a, const Query &b) {
  return a.ord < b.ord;
}
int curL = 0, curR = -1;
for(int i = 0; i < Q.sz; i++){
  while(curL > Q[i].L){
    curL--; add(curL);
  }
  while(curR < Q[i].R){
    curR++; add(curR);
  }
  while(curL < Q[i].L){
```

```
    remove(curL); curL++;
  }
  while(curR > Q[i].R){
    remove(curR); curR--;
  }
}
```

2.12 Matrix Expo

```
const int MOD = 998244353;
typedef vector<int> row;
typedef vector<row> matrix;
inline int add(const int &a, const int &b) {
  int c = a + b;
  if (c >= MOD) c -= MOD;
  return c;
}
inline int mult(const int &a, const int &b) {
  return (long long)a * b % MOD;
}
matrix operator*(const matrix &m1, const matrix &m2) {
  int r = m1.size();
  int m = m1.back().size();
  int c = m2.back().size();
  matrix ret(r, row(c, 0));
  for (int i = 0; i < r; i++) {
    for (int k = 0; k < m; k++) {
      for (int j = 0; j < c; j++) {
        ret[i][j] = add(ret[i][j], mult(m1[i][k], m2[k][j]));
      }
    }
  }
  return ret;
}
matrix one(int dim) {
  matrix ret(dim, row(dim, 0));
  for (int i = 0; i < dim; i++) {
    ret[i][i] = 1;
  }
  return ret;
}
matrix operator^(const matrix &m, const int &e) {
  if (e == 0) return one(m.size());
  matrix sqrtm = m ^ (e / 2);
  matrix ret = sqrtm * sqrtm;
  if (e & 1) ret = ret * m;
  return ret;
}
```

2.13 Mo On Tree

Description: Build Euler order of $2N$ size - write node ID when entering AND exiting. Path (u,v) with $\text{in}[u] < \text{in}[v]$ is now range. If u is LCA, then range is $[\text{in}[u], \text{in}[v]]$. If not, then range is $[\text{out}[u], \text{in}[v]] \cup [\text{in}[LCA], \text{in}[LCA]]$. Nodes that appear exactly once (not 0 or 2 times) on these ranges are relevant, maintain them during Mo.

Time: $\mathcal{O}\left(N\sqrt{Q}\right)$

2.14 Mo With Updates

Description: Sort queries by tuple $(\lfloor l/B \rfloor, \lfloor r/B \rfloor, t)$ where t is the time of query. If DS has answer for $[L,R]$ at time T now, then we have to adjust range (add/remove like usual Mo). For time, we need to make some updates if $t < T$ and rollback some updates if $t > T$.

Time: $\mathcal{O}\left(QN^{2/3}\right)$

2.15 Ordered Set

```
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type,
less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
// T.insert(x)
// *T.find_by_order(k) -> kth element
// T.order_of_key(x) -> position of 1st element >= x
```

2.16 Persistent Segment Tree

Description: RMQ, point updates.
Time: $\mathcal{O}(\log N)$

```
namespace PersistentTree {
  ll t[M];
  int L[M], R[M], root[N], arr[N], ptr = 0;
  void update(int x, int y, int p, int v, int b = 1, int e = n) {
    if (b == e) return void(t[y] = v);
    int mid = b + e >> 1;
    if (p <= mid) R[y] = R[x], L[y] = ++ptr, update(L[x], L[y], p, v, b, mid);
    else L[y] = L[x], R[y] = ++ptr, update(R[x], R[y], p, v, mid + 1, e);
    t[y] = min(t[L[y]], t[R[y]]);
  }
  void init() {
    for (int i = 1; i <= n; ++i) arr[i] = INT_MAX;
    root[0] = 0, t[0] = INF;
    for (int i = 1; i <= n; ++i) {
      root[i] = root[i - 1];
      for (auto & [r, w] : who[i]) if (w < arr[r]) {
        int new_root = ++ptr;
        arr[r] = w, update(root[i], new_root, r, w);
        root[i] = new_root;
      }
    }
  }
  ll query(int u, int l, int r, int b = 1, int e = n) {
    if (!u or b > r or e < l) return INF;
    if (b >= l and e <= r) return t[u];
    int mid = b + e >> 1;
    return min(query(L[u], l, r, b, mid), query(R[u], l, r, mid + 1, e));
  }
  inline ll getMin(int l, int r) {
    return query(root[l], r, n);
  }
}
```

2.17 Persistent Trie

```
using namespace std;
// Description: find maximum value (x^a[j]) in the range (l,r) where l<=j<=r
const int N = 1e5 + 100;
const int K = 15;
struct node_t;
typedef node_t * pnode;
struct node_t {
  int time;
  pnode to[2];
  node_t() : time(0) {
    to[0] = to[1] = 0;
  }
  bool go(int l) const {
    if (!this) return false;
    return time >= l;
  }
  pnode clone() {
    pnode cur = new node_t();
    if (this) {
      cur->time = time;
      cur->to[0] = to[0];
      cur->to[1] = to[1];
    }
    return cur;
  }
};
pnode last;
pnode version[N];
void insert(int a, int time) {
  pnode v = version[time] = last = last->clone();
  for (int i = K - 1; i >= 0; --i) {
    int bit = (a >> i) & 1;
    pnode &child = v->to[bit];
    child = child->clone();
    v = child;
    v->time = time;
  }
}
int query(pnode v, int x, int l) {
  int ans = 0;
  for (int i = K - 1; i >= 0; --i) {
    int bit = (x >> i) & 1;
    if (v->to[bit]->go(l)) { // checking if this bit was inserted before the range
      ans |= 1 << i;
      v = v->to[bit];
    } else {
      v = v->to[bit ^ 1];
    }
  }
  return ans;
}
void solve() {
  int n, q;
  scanf("%d %d", &n, &q);
  last = 0;
  for (int i = 0; i < n; ++i) {
    int a;
    scanf("%d", &a);
    insert(a, i);
  }
  while (q--) {
    int x, l, r;
    scanf("%d %d %d", &x, &l, &r);
    --l, --r;
    printf("%d\n", query(version[r], ~x, l));
    // Trie version[r] contains the trie for [0...r] elements
  }
}
```

2.18 Segment Tree Beats

Description: For update $A_i \to A_i \bmod x$ and similar, keep range min, max in node and lazily update whenever min = max. For update $A_i \to \min(A_i, x)$ and similar, keep range max, second max in node and lazily update whenever $x >$ second max.

Time: $\mathcal{O}\left(\log^2 N\right)$, $\mathcal{O}(\log N)$

2.19 Segment Tree

Description: RMQ. Iterative, 0-indexed, point update, query $[l,r)$.
Time: $\mathcal{O}(\log N)$

```
const int N = 500010;
int n, a[N], tree[N << 1];
void init() {
  for (int i = 0; i < n; ++i) tree[n + i] = a[i];
  for (int i = n - 1; i >= 0; --i) {
    tree[i] = min(tree[i << 1], tree[i << 1 | 1]);
  }
}
void update(int p, int v) {
  for (tree[p += n] = v; p > 1; p >>= 1) {
    tree[p >> 1] = min(tree[p], tree[p ^ 1]);
  }
}
int query(int l, int r) {
  int ret = INT_MAX;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l & 1) ret = min(ret, tree[l++]);
    if (r & 1) ret = min(ret, tree[--r]);
  } return ret;
}
```

2.20 Sparse Table

Description: Sparse table for fast RMQ.
Time: $\mathcal{O}(1)$

```
int a[sz], spar[20][sz];
int rmq(int l, int r) {    // 1 - index
  int h = 31 - __builtin_clz(r - l + 1);
  return min(spar[h][l], spar[h][r + 1 - (1 << h)]);
}
int main() {
  for (int i = 1; i <= n; i++) spar[0][i] = a[i];
  for (int h = 0; 2 << h <= n; h++) {
    for (int i = 1, j = i + (1 << h); i <= n; i++, j++) {
      int &w = spar[h + 1][i] = spar[h][i];
      if (j <= n) w = min(w, spar[h][j]);
    }
  }
}
```

2.21 Treap

Description: Implicit treap - array with fast cut/join operation support. Below code supports finding root of each tree.
Time: $\mathcal{O}(\log N)$

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
struct node {
  int heap, tot; ll value, sum; node *l, *r, *par;
  node (ll value) : value(value), sum(value), tot(0), heap(rng()), l(nullptr), r(nullptr), par(nullptr) {}
};
inline int size (node *t) {return t ? t->tot : 0;}
inline ll sum (node *t) {return t ? t->sum : 0;}
inline void refresh (node *t) {
  if (t) {
    t->tot = 1 + size(t->l) + size(t->r);
    t->sum = sum(t->l) + sum(t->r) + t->value;
    if (t->l) t->l->par = t;
    if (t->r) t->r->par = t;
    t -> par = nullptr;
  }
}
void split (node *t, node* &l, node* &r, int key, int add = 0) {
  if (!t) return void(l = r = nullptr);
  int cur_key = add + size(t->l);
  if (cur_key >= key) split(t->l, l, t->l, key, add), r = t;
  else split(t->r, t->r, r, key, cur_key + 1), l = t;
  refresh(t);
}
void merge (node* &t, node *l, node *r) {
  if (!l or !r) t = l ? l : r;
  else if (l->heap > r->heap) merge(l->r, l->r, r), t = l;
  else merge(r->l, l, r->l), t = r;
  refresh(t);
}
inline node* getRoot (node *u) {
  while (u->par) u = u -> par;
  return u;
}
int main() {
  int q; cin >> q;
  vector <node*> a(q + 1);
  for (int i = 1; i <= q; ++i) {
    int cmd, x, y;
    scanf("%d %d", &cmd, &x);
    if (cmd == 1) {
      a[i] = new node(x);
    } else if (cmd == 2) {
      scanf("%d", &y);
      node *one = getRoot(a[x]), *two = getRoot(a[y]), *root = nullptr;
      if (one != two) merge(root, one, two);
    } else if (cmd == 3) {
      scanf("%d", &y);
      node *l, *r, *root = getRoot(a[x]);
      split(root, l, r, y);
    } else {
      printf("%lld\n", sum(getRoot(a[x])));
    }
  }
  return 0;
}
```

2.22 Trie

```
struct Trie {
  const int L = 26; int N;
  vector< vector<int> > next;
  vector<int> cnt;
  Trie() : N(0) {node(); }
  int node() {
    next.emplace_back(L, 0);
    cnt.emplace_back();
    return N++;
  }
  //insert or delete
  void insert(const string &s, int a = 1){
    int u = 0, c;
    for (int i = 0; i < s.length(); i++) {
      c = s[i] - 'a';
      if(!next[u][c]) next[u][c] = node();
      cnt[u = next[u][c]] += a;
    }
  }
};
```

3 Geometry

3.1 2D

3.1.1 Circle2D

```
struct Circle {
  Point o;
  double r;
  Circle () {}
  Circle (Point o, double r = 0): o(o), r(r) {}
  void read () { o.read(), scanf("%lf", &r); }
  Point point(double rad) { return Point(o.x + cos(rad)*r, o.y + sin(rad)*r); }
  double getArea (double rad) { return rad * r * r / 2; }
  //area of the circular sector cut by a chord with central angle alpha
  double sector(double alpha) {return r * r * 0.5 * (alpha - sin(alpha));}
};
namespace Circular {
  using namespace Linear;
  using namespace Vectorial;
  using namespace Triangular;
  int getLineCircleIntersection (Point p, Point q, Circle O, double& t1, double& t2, vector<Point>& sol) {
    Vector v = q - p;
    //sol.clear();
    double a = v.x, b = p.x - O.o.x, c = v.y, d = p.y - O.o.y;
    double e = a*a+c*c, f = 2*(a*b+c*d), g = b*b+d*d-O.r*O.r;
    double delta = f*f - 4*e*g;
    if (dcmp(delta) < 0) return 0;
    if (dcmp(delta) == 0) {
      t1 = t2 = -f / (2 * e);
      sol.push_back(p + v * t1);
      return 1;
    }
    t1 = (-f - sqrt(delta)) / (2 * e); sol.push_back(p + v * t1);
    t2 = (-f + sqrt(delta)) / (2 * e); sol.push_back(p + v * t2);
    return 2;
  }
  // signed area of intersection of circle(c.o, c.r) and
  // triangle(c.o, s.a, s.b) [cross(a-o, b-o)/2]
  double areaCircleTriIntersection(Circle c, Segment s){
    using namespace Linear;
    double OA = getLength(c.o - s.a);
    double OB = getLength(c.o - s.b);
    // sector
    if (dcmp(getDistanceToSegment(c.o, s.a, s.b) - c.r) >= 0)
      return fix_acute(getSignedAngle(s.a - c.o, s.b - c.o)) * (c.r*c.r) / 2.0;
    // triangle
    if (dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)
      return getCross(c.o-s.b,s.a-s.b) / 2.0;
    // three part: (A, a) (a, b) (b, B)
    vector<Point>Sect; double t1,t2;
    getLineCircleIntersection(s.a, s.b, c, t1, t2, Sect);
    return areaCircleTriIntersection(c, Segment(s.a, Sect[0]))
      + areaCircleTriIntersection(c, Segment(Sect[0], Sect[1]))
      + areaCircleTriIntersection(c, Segment(Sect[1], s.b));
  }
  // area of intersecion of circle(c.o, c.r) and simple polyson(p[])
  // Tested : ZOJ 2675 - Little Mammoth
  double areaCirclePolygon(Circle c, Polygon p){
    double res = .0;
    int n = p.size();
    for (int i = 0; i < n; ++ i)
      res += areaCircleTriIntersection(c, Segment(p[i], p[(i+1)%n]));
    return fabs(res);
  }
  // interior          (d < R - r)         ----> -2
  // interior tangents (d = R - r)         ----> -1
  // concentric        (d = 0)
  // secants           (R - r < d < R + r) ---->  0
  // exterior tangents (d = R + r)         ---->  1
  // exterior          (d > R + r)         ---->  2
  int getPos(Circle o1, Circle o2) {
    using namespace Vectorial;
    double d = getLength(o1.o - o2.o);
    int in = dcmp(d - fabs(o1.r - o2.r)), ex = dcmp(d - (o1.r + o2.r));
    return in<0 ? -2 : in==0? -1 : ex==0 ? 1 : ex>0? 2 : 0;
  }
  int getCircleCircleIntersection (Circle o1, Circle o2, vector<Point>& sol) {
    double d = getLength(o1.o - o2.o);
    if (dcmp(d) == 0) {
      if (dcmp(o1.r - o2.r) == 0) return -1;
      return 0;
    }
    if (dcmp(o1.r + o2.r - d) < 0) return 0;
    if (dcmp(fabs(o1.r-o2.r) - d) > 0) return 0;
    Vector v = o2.o - o1.o;
    double co = (o1.r*o1.r + getPLength(v) - o2.r*o2.r) / (2 * o1.r * getLength(v));
    double si = sqrt(fabs(1.0 - co*co));
    Point p1 = scale(cw(v,co, si), o1.r) + o1.o;
    Point p2 = scale(ccw(v,co, si), o1.r) + o1.o;
    sol.push_back(p1);
    if (p1 == p2) return 1;
    sol.push_back(p2);
    return 2;
  }
  double areaCircleCircle(Circle o1, Circle o2){
    Vector AB = o2.o - o1.o;
    double d = getLength(AB);
    if(d >= o1.r + o2.r) return 0;
    if(d + o1.r <= o2.r) return pi * o1.r * o1.r;
    if(d + o2.r <= o1.r) return pi * o2.r * o2.r;
    double alpha1 = acos((o1.r * o1.r + d * d - o2.r * o2.r) / (2.0 * o1.r * d));
    double alpha2 = acos((o2.r * o2.r + d * d - o1.r * o1.r) / (2.0 * o2.r * d));
    return o1.sector(2*alpha1) + o2.sector(2*alpha2);
```

```
}
int getTangents (Point p, Circle o, Vector* v) {
  Vector u = o.o - p;
  double d = getLength(u);
  if (d < o.r) return 0;
  else if (dcmp(d - o.r) == 0) {
    v[0] = rotate(u, pi / 2);
    return 1;
  } else {
    double ang = asin(o.r / d);
    v[0] = rotate(u, -ang);
    v[1] = rotate(u, ang);
    return 2;
  }
}
int getTangentPoints (Point p, Circle o, vector<Point>& v) {
  Vector u = p - o.o ;
  double d = getLength(u);
  if (d < o.r) return 0;
  else if (dcmp(d - o.r) == 0) {
    v.push_back(o.o+u);
    return 1;
  } else {
    double ang = acos(o.r / d);
    u = u / getLength(u) * o.r;
    v.push_back(o.o+rotate(u, -ang));
    v.push_back(o.o+rotate(u, ang));
    return 2;
  }
}
int getTangents (Circle o1, Circle o2, Point* a, Point* b) {
  int cnt = 0;
  if (dcmp(o1.r-o2.r) < 0) { swap(o1, o2); swap(a, b); }
  double d2 = getPLength(o1.o - o2.o);
  double rdif = o1.r - o2.r, rsum = o1.r + o2.r;
  if (dcmp(d2 - rdif * rdif) < 0) return 0;
  if (dcmp(d2) == 0 && dcmp(o1.r - o2.r) == 0) return -1;
  double base = getAngle(o2.o - o1.o);
  if (dcmp(d2 - rdif * rdif) == 0) {
    a[cnt] = o1.point(base); b[cnt] = o2.point(base); cnt++;
    return cnt;
  }
  double ang = acos( (o1.r - o2.r) / sqrt(d2) );
  a[cnt] = o1.point(base+ang); b[cnt] = o2.point(base+ang); cnt++;
  a[cnt] = o1.point(base-ang); b[cnt] = o2.point(base-ang); cnt++;
  if (dcmp(d2 - rsum * rsum) == 0) {
    a[cnt] = o1.point(base); b[cnt] = o2.point(pi+base); cnt++;
  }
  else if (dcmp(d2 - rsum * rsum) > 0) {
    double ang = acos( (o1.r + o2.r) / sqrt(d2) );
    a[cnt] = o1.point(base+ang); b[cnt] = o2.point(pi+base+ang); cnt++;
    a[cnt] = o1.point(base-ang); b[cnt] = o2.point(pi+base-ang); cnt++;
  }
  return cnt;
}
Circle CircumscribedCircle(Point p1, Point p2, Point p3) {
  double Bx = p2.x - p1.x, By = p2.y - p1.y;
  double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
  double D = 2 * (Bx * Cy - By * Cx);
  double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x;
  double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y;
  Point p = Point(cx, cy);
  return Circle(p, getLength(p1 - p));
}
Circle InscribedCircle(Point p1, Point p2, Point p3) {
  double a = getLength(p2 - p3);
  double b = getLength(p3 - p1);
  double c = getLength(p1 - p2);
  Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
  return Circle(p, getDistanceToLine(p, p1, p2));
}
//distance From P : distance from Q =  rp : rq
Circle getApolloniusCircle(const Point& P,const Point& Q, double rp, double rq ){
  rq *= rq ;
  rp *= rp ;
  double a = rq - rp ;
  assert(dcmp(a));
  double g = rq * P.x - rp * Q.x ; g /= a ;
  double h = rq * P.y - rp * Q.y ; h /= a ;
  double c = rq*P.x*P.x-rp*Q.x*Q.x+rq*P.y*P.y-rp*Q.y*Q.y ;
  c /= a ;
  Point o(g,h);
  double R = g*g +h*h - c ;
  R = sqrt(R);
  return Circle(o,R);
 }
};
```

3.1.2 Line2D

```
typedef Point Vector;
struct Line {
  double a, b, c;
  Line (double a = 0, double b = 0, double c = 0): a(a), b(b), c(c) {}
};
struct Segment{
  Point a, b;
  Segment(){}
  Segment(Point aa,Point bb) {a=aa,b=bb;}
};
struct DirLine {
  Point p;
  Vector v;
  double ang;
  DirLine () {}
  DirLine (Point p, Vector v): p(p), v(v) { ang = atan2(v.y, v.x); }
  bool operator < (const DirLine& u) const { return ang < u.ang; }
};
namespace Vectorial {
  double getDot (Vector a, Vector b) { return a.x * b.x + a.y * b.y; }
  double getCross (Vector a, Vector b) { return a.x * b.y - a.y * b.x; }
  double getLength (Vector a) { return sqrt(getDot(a, a)); }
  double getPLength (Vector a) { return getDot(a, a); }
  double getAngle (Vector u) { return atan2(u.y, u.x); }
  double getSignedAngle (Vector a, Vector b) {return getAngle(b)-getAngle(a);}
  Vector rotate (Vector a, double rad) { return Vector(a.x*cos(rad)-a.y*sin(rad), a.x*sin(rad)+a.y*cos(rad)); }
  Vector ccw(Vector a, double co, double si) {return Vector(a.x*co-a.y*si, a.y*co+a.x*si);}
  Vector cw (Vector a, double co, double si) {return Vector(a.x*co+a.y*si, a.y*co-a.x*si);}
  Vector scale(Vector a, double s = 1.0) {return a / getLength(a) * s;}
  Vector getNormal (Vector a) { double l = getLength(a); return Vector(-a.y/l, a.x/l); }
};
namespace Linear {
  using namespace Vectorial;
  Line getLine (double x1, double y1, double x2, double y2) { return Line(y2-y1, x1-x2, y1*x2-x1*y2); }
  Line getLine (double a, double b, Point u) { return Line(a, -b, u.y * b - u.x * a); }
  bool getIntersection (Line p, Line q, Point& o) {
    if (fabs(p.a * q.b - q.a * p.b) < eps)
      return false;
    o.x = (q.c * p.b - p.c * q.b) / (p.a * q.b - q.a * p.b);
    o.y = (q.c * p.a - p.c * q.a) / (p.b * q.a - q.b * p.a);
    return true;
  }
  bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {
    if (dcmp(getCross(v, w)) == 0) return false;
    Vector u = p - q;
    double k = getCross(w, u) / getCross(v, w);
    o = p + v * k;
    return true;
  }
  double getDistanceToLine (Point p, Point a, Point b) { return fabs(getCross(b-a, p-a) / getLength(b-a)); }
  double getDistanceToSegment (Point p, Point a, Point b) {
    if (a == b) return getLength(p-a);
    Vector v1 = b - a, v2 = p - a, v3 = p - b;
    if (dcmp(getDot(v1, v2)) < 0) return getLength(v2);
    else if (dcmp(getDot(v1, v3)) > 0) return getLength(v3);
    else return fabs(getCross(v1, v2) / getLength(v1));
  }
  double getDistanceSegToSeg (Point a,Point b,Point c,Point d){
    double Ans=INT_MAX;
    Ans=min(Ans,getDistanceToSegment(a,c,d));
    Ans=min(Ans,getDistanceToSegment(b,c,d));
    Ans=min(Ans,getDistanceToSegment(c,a,b));
    Ans=min(Ans,getDistanceToSegment(d,a,b));
    return Ans;
  }
  Point getPointToLine (Point p, Point a, Point b) { Vector v = b-a; return a+v*(getDot(v, p-a) / getDot(v,v)); }
  bool onSegment (Point p, Point a, Point b) { return dcmp(getCross(a-p, b-p)) == 0 && dcmp(getDot(a-p, b-p)) <= 0; }
  bool haveIntersection (Point a1, Point a2, Point b1, Point b2) {
    if(onSegment(a1,b1,b2)) return true;
    if(onSegment(a2,b1,b2)) return true;
    if(onSegment(b1,a1,a2)) return true;
    if(onSegment(b2,a1,a2)) return true;  //Case of touch
    double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1, b2-a1), c3=getCross(b2-b1, a1-b1), c4=getCross(b2-b1,a2-b1);
    return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
  }
  bool onLeft(DirLine l, Point p) { return dcmp(l.v * (p-l.p)) >= 0; }
}
```

3.1.3 Point2D

```
const double pi = 4 * atan(1);
const double eps = 1e-6;
inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }
double fix_acute(double th) {return th<-pi ? (th+2*pi): th>pi ? (th-2*pi) : th;}
inline double getDistance (double x, double y) { return sqrt(x * x + y * y); }
inline double torad(double deg) { return deg / 180 * pi; }
struct Point {
  double x, y;
  Point (double x = 0, double y = 0): x(x), y(y) {}
  void read () { scanf("%lf%lf", &x, &y); }
  void write () { printf("%lf %lf", x, y); }
  bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }
  bool operator != (const Point& u) const { return !(*this == u); }
  bool operator < (const Point& u) const { return dcmp(x - u.x) < 0 || (dcmp(x-u.x)==0 && dcmp(y-u.y) < 0); }
```

```cpp
  bool operator > (const Point& u) const { return u <
  ↪  *this; }
  bool operator <= (const Point& u) const { return *this <
  ↪  u || *this == u; }
  bool operator >= (const Point& u) const { return *this >
  ↪  u || *this == u; }
  Point operator + (const Point& u) { return Point(x + u.x,
  ↪  y + u.y); }
  Point operator - (const Point& u) { return Point(x - u.x,
  ↪  y - u.y); }
  Point operator * (const double u) { return Point(x * u, y
  ↪  * u); }
  Point operator / (const double u) { return Point(x / u, y
  ↪  / u); }
  double operator * (const Point& u) { return x*u.y -
  ↪  y*u.x; }
};
inline double getDistance (Point a, Point b) { double
↪  x=a.x-b.x, y=a.y-b.y; return sqrt(x*x + y*y); }
```

3.1.4 Polygon2D

```cpp
typedef vector<Point> Polygon;
namespace Polygonal {
  using namespace Vectorial;
  using namespace Linear;
  using namespace Triangular;
  double getSignedArea (Point* p, int n) {
    double ret = 0;
    for (int i = 0; i < n-1; i++)
      ret += (p[i]-p[0]) * (p[i+1]-p[0]);
    return ret/2;
  }
  int isPointInPolygon (Point o, Point* p, int n) {
    int wn = 0;
    for (int i = 0; i < n; i++) {
      int j = (i + 1) % n;
      if (onSegment(o, p[i], p[j]) || o == p[i]) return 0;
      int k = dcmp(getCross(p[j] - p[i], o-p[i]));
      int d1 = dcmp(p[i].y - o.y);
      int d2 = dcmp(p[j].y - o.y);
      if (k > 0 && d1 <= 0 && d2 > 0) wn++;
      if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    return wn ? -1 : 1;
  }
  void rotatingCalipers(Point *p, int n, vector&lt;Segment&gt;&
  ↪  sol) {
    sol.clear();
    int j = 1; p[n] = p[0];
    for (int i = 0; i < n; i++) {
      while (getCross(p[j+1]-p[i+1], p[i]-p[i+1]) >
      ↪  getCross(p[j]-p[i+1], p[i]-p[i+1]))
        j = (j+1) % n;
      sol.push_back(Segment(p[i],p[j]));
      sol.push_back(Segment(p[i + 1],p[j + 1]));
    }
  }
  void rotatingCalipersGetRectangle (Point *p, int n,
  ↪  double& area, double& perimeter) {
    p[n] = p[0];
    int l = 1, r = 1, j = 1;
    area = perimeter = 1e20;
    for (int i = 0; i < n; i++) {
      Vector v = (p[i+1]-p[i]) / getLength(p[i+1]-p[i]);
      while (dcmp(getDot(v, p[r%n]-p[i]) - getDot(v,
      ↪  p[(r+1)%n]-p[i])) < 0) r++;
      while (j < r || dcmp(getCross(v, p[j%n]-p[i]) -
      ↪  getCross(v,p[(j+1)%n]-p[i])) < 0) j++;
      while (l < j || dcmp(getDot(v, p[l%n]-p[i]) -
      ↪  getDot(v, p[(l+1)%n]-p[i])) > 0) l++;
      double w = getDot(v, p[r%n]-p[i])-getDot(v,
      ↪  p[l%n]-p[i]);
      double h = getDistanceToLine (p[j%n], p[i], p[i+1]);
      area = min(area, w * h);
      perimeter = min(perimeter, 2 * w + 2 * h);
    }
  }
  Polygon cutPolygon (Polygon u, Point a, Point b) {
    Polygon ret;
    int n = u.size();
    for (int i = 0; i < n; i++) {
      Point c = u[i], d = u[(i+1)%n];
      if (dcmp((b-a)*(c-a)) >= 0) ret.push_back(c);
      if (dcmp((b-a)*(d-c)) != 0) {
        Point t;
        getIntersection(a, b-a, c, d-c, t);
        if (onSegment(t, c, d))
          ret.push_back(t);
      }
    }
    return ret;
  }
  int halfPlaneIntersection(DirLine* li, int n, Point*
  ↪  poly) {
    sort(li, li + n);
    int first, last;
    Point* p = new Point[n];
    DirLine* q = new DirLine[n];
    q[first=last=0] = li[0];
    for (int i = 1; i < n; i++) {
      while (first < last && !onLeft(li[i], p[last-1]))
      ↪  last--;
      while (first < last && !onLeft(li[i], p[first]))
      ↪  first++;
      q[++last] = li[i];
      if (dcmp(q[last].v * q[last-1].v) == 0) {
        last--;
        if (onLeft(q[last], li[i].p)) q[last] = li[i];
      }
      if (first < last)
        getIntersection(q[last-1].p, q[last-1].v,
        ↪  q[last].p, q[last].v, p[last-1]);
    }
    while (first < last && !onLeft(q[first], p[last-1]))
    ↪  last--;
    if (last - first <= 1) { delete [] p; delete [] q;
    ↪  return 0; }
    getIntersection(q[last].p, q[last].v, q[first].p,
    ↪  q[first].v, p[last]);
    int m = 0;
    for (int i = first; i <= last; i++) poly[m++] = p[i];
    delete [] p; delete [] q;
    return m;
  }
  Polygon simplify (const Polygon& poly) {
    Polygon ret;
    int n = poly.size();
    for (int i = 0; i < n; i++) {
      Point a = poly[i];
      Point b = poly[(i+1)%n];
      Point c = poly[(i+2)%n];
      if (dcmp((b-a)*(c-b)) != 0 && (ret.size() == 0 || b
      ↪  != ret[ret.size()-1]))
        ret.push_back(b);
    }
    return ret;
  }
  Point ComputeCentroid( Point* p,int n){
    Point c(0,0);
    double scale = 6.0 * getSignedArea(p,n);
    for (int i = 0; i < n; i++){
      int j = (i+1) % n;
      c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
  }
  // Tested : https://www.spoj.com/problems/INOROUT
  // pt must be in ccw order with no three collinear points
  // returns inside = 1, on = 0, outside = -1
  int pointInConvexPolygon(Point* pt, int n, Point p){
    assert(n >= 3);
    int lo = 1 , hi = n - 1 ;
    while(hi - lo > 1){
      int mid = (lo + hi) / 2;
      if(getCross(pt[mid] - pt[0], p - pt[0]) > 0) lo = mid;
      else hi = mid;
    }
    bool in = pointInTriangle(pt[0], pt[lo], pt[hi], p);
    if(!in) return -1;
    if(getCross(pt[lo] - pt[lo-1], p - pt[lo-1]) == 0)
    ↪  return 0;
    if(getCross(pt[hi] - pt[lo], p - pt[lo]) == 0) return 0;
    if(getCross(pt[hi] - pt[(hi+1)%n], p - pt[(hi+1)%n]) ==
    ↪  0) return 0;
    return 1;
  }
  // Tested : https://toph.co/p/cover-the-points
  // Calculate [ACW, CW] tangent pair from an external point
  #define CW   -1
  #define ACW   1
  int direction(Point st, Point ed, Point q)      {return
  ↪  dcmp(getCross(ed - st, q - ed));}
  bool isGood(Point u, Point v, Point Q, int dir)  {return
  ↪  direction(Q, u, v) != -dir;}
  Point better(Point u, Point v, Point Q, int dir) {return
  ↪  direction(Q, u, v) == dir ? u : v;}
  Point tangents(Point* pt, Point Q, int dir, int lo, int
  ↪  hi){
    while(hi - lo > 1){
      int mid = (lo + hi)/2;
      bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);
      bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);
      if(pvs && nxt) return pt[mid];
      if(!(pvs || nxt)){
        Point p1 = tangents(pt, Q, dir, mid+1, hi);
        Point p2 = tangents(pt, Q, dir, lo, mid - 1);
        return better(p1, p2, Q, dir);
      }
      if(!pvs){
        if(direction(Q, pt[mid], pt[lo]) == dir)  hi = mid
        ↪  - 1;
        else if(better(pt[lo], pt[hi], Q, dir) == pt[lo])
        ↪  hi = mid - 1;
        else lo = mid + 1;
      }
      if(!nxt){
        if(direction(Q, pt[mid], pt[lo]) == dir)  lo = mid
        ↪  + 1;
        else if(better(pt[lo], pt[hi], Q, dir) == pt[lo])
        ↪  hi = mid - 1;
        else lo = mid + 1;
      }
    }
    Point ret = pt[lo];
    for(int i = lo + 1; i <= hi; i++) ret = better(ret,
    ↪  pt[i], Q, dir);
    return ret;
  }
  // [ACW, CW] Tangent
  pair<Point, Point> get_tangents(Point* pt, int n, Point
  ↪  Q){
    Point acw_tan = tangents(pt, Q, ACW, 0, n - 1);
    Point cw_tan = tangents(pt, Q, CW, 0, n - 1);
    return make_pair(acw_tan, cw_tan);
  }
};
```

3.1.5 Star2D

```cpp
struct Star{
  int n;    // number of side of the star
  double r; // radius of the circum-circle
  Star(int n,double r) {this->n=n; this->r=r;}
  double getArea(){
```

```
    double theta=pi/n;
    double s=2*r*sin(theta);
    double R=0.5*s/tan(theta);
    double a=0.5*n*s*R;
    double a2=0.25*s*s/tan(1.5*theta);
    return a-n*a2;
  }
};
```

3.1.6 Triangle2D

```
namespace Triangular {
  using namespace Vectorial;
  double getAngle (double a, double b, double c) { return acos((a*a+b*b-c*c) / (2*a*b)); }
  double getArea (double a, double b, double c) { double s =(a+b+c)/2; return sqrt(s*(s-a)*(s-b)*(s-c)); }
  double getArea (double a, double h) { return a * h / 2; }
  double getArea (Point a, Point b, Point c) { return fabs(getCross(b - a, c - a)) / 2; }
  double getDirArea (Point a, Point b, Point c) { return getCross(b - a, c - a) / 2;}
  //ma/mb/mc = length of median from side a/b/c
  double getArea_(double ma,double mb,double mc) {double s=(ma+mb+mc)/2; return 4/3.0 * sqrt(s*(s-ma)*(s-mb)*(s-mc));}
  //ha/hb/hc = length of perpendicular from side a/b/c
  double get_Area(double ha,double hb,double hc){
    double H=(1/ha+1/hb+1/hc)/2; double _A_ = 4 * sqrt(H * (H-1/ha)*(H-1/hb)*(H-1/hc)); return 1.0/_A_;
  }
  bool pointInTriangle(Point a, Point b, Point c, Point p){
    double s1 = getArea(a,b,c);
    double s2 = getArea(p,b,c) + getArea(p,a,b) + getArea(p,c,a);
    return dcmp(s1 - s2) == 0;
  }
};
```

3.2 3D

3.2.1 Line3D

```
typedef Point3D Vector3D;
typedef vector<Point> Polygon;
typedef vector<Point3D> Polyhedron;
namespace Vectorial{
  double getDot (Vector3D a, Vector3D b)  {return a.x*b.x+a.y*b.y+a.z*b.z;}
  Vector3D getCross(Vector3D a, Vector3D b) {return Point3D(a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z, a.x*b.y-a.y*b.x);}
  double getLength (Vector3D a)          {return sqrt(getDot(a, a)); }
  double getPLength (Vector3D a)         {return getDot(a, a); }
  Vector3D unitVector(Vector3D v)          {return v/getLength(v);}
  double getUnsignedAngle(Vector3D u,Vector3D v){
    double cosTheta = getDot(u,v)/getLength(u)/getLength(v);
    cosTheta = max(-1.0,min(1.0,cosTheta));
    return acos(cosTheta);
  }
  Vector3D rotate(Vector3D v,Vector3D a,double rad){
    a = unitVector(a);
    return v * cos(rad) + a * (1 - cos(rad)) * getDot(a,v) + getCross(a,v) * sin(rad);
  }
}
struct Line3D{
  Vector3D v; Point3D o;
  Line3D() {};
  Line3D(Vector3D v,Point3D o):v(v),o(o){}
  Point3D getPoint(double t) {return o + v*t;}
};
namespace Linear{
  using namespace Vectorial;
  double getDistSq(Line3D l, Point3D p)          {return getPLength(getCross(l.v,p-l.o))/getPLength(l.v);}
  double getDistLinePoint(Line3D l, Point3D p)   {return sqrt(getDistSq(l,p));}
  bool cmp(Line3D l,Point3D p, Point3D q)         {return getDot(l.v,p) < getDot(l.v,q);}
  Point3D projection(Line3D l,Point3D p)          {return l.o + l.v * getDot(l.v,p-l.o)/getPLength(l.v);}
  Point3D reflection(Line3D l,Point3D p)          {return projection(l,p)+projection(l,p)-p;}
  double getAngle(Line3D l,Line3D m)             {return getUnsignedAngle(l.v,m.v);}
  bool isParallel(Line3D p,Line3D q)             {return dcmp(getPLength(getCross(p.v,q.v))) == 0;}
  bool isPerpendicular(Line3D p,Line3D q)        {return dcmp(getDot(p.v,q.v)) == 0;}
  double getDist(Line3D l, Line3D m){
    Vector3D n = getCross(l.v, m.v);
    if(getPLength(n) == 0) return getDistLinePoint(l,m.o);
    else return fabs(getDot(m.o-l.o , n)) / getLength(n);
  }
  Point3D getClosestPointOnLine1(Line3D l,Line3D m){
    Vector3D n = getCross(l.v, m.v);
    Vector3D n2 = getCross(m.v, n);
    return l.o + l.v * getDot(m.o-l.o, n2) / getDot(l.v, n2);
  }
}
```

3.2.2 Plane3D

```
struct Plane{
  Vector3D n; //normal n
  double d; //getDot(n,p) = d for any point p on the plane
  Plane() {}
  Plane(Vector3D n, double d) : n(n), d(d) {}
  Plane(Vector3D n, Point3D p) : n(n), d(Vectorial :: getDot(n,p)) {}
  Plane(const Plane &p) : n(p.n), d(p.d) {}
};
namespace Planar{
  using namespace Vectorial;
  Plane getPlane(Point3D a,Point3D b,Point3D c) {return Plane(getCross(b-a,c-a),a);}
  Plane translate(Plane p,Vector3D t)       {return Plane(p.n, p.d+getDot(p.n,t));}
  Plane shiftUp(Plane p,double dist)       {return Plane(p.n, p.d+dist*getLength(p.n));}
  Plane shiftDown(Plane p,double dist)     {return Plane(p.n, p.d-dist*getLength(p.n));}
  double getSide(Plane p,Point3D a)      {return getDot(p.n,a)-p.d;}
  double getDistance(Plane p,Point3D a) {return fabs(getSide(p,a))/getLength(p.n);}
  Point3D projection(Plane p,Point3D a)   {return a-p.n*getSide(p,a)/getPLength(p.n);}
  Point3D reflection(Plane p,Point3D a)   {return a-p.n*getSide(p,a)/getPLength(p.n)*2;}
  bool intersect(Plane p, Line3D l, Point3D& a){
    if(dcmp(getDot(p.n,l.v)) == 0) return false;
    a = l.o - l.v * getSide(p,l.o) / getDot(p.n,l.v);
    return true;
  }
  bool intersect(Plane p,Plane q,Line3D& l){
    l.v = getCross(p.n,q.n);
    if(dcmp(getPLength(l.v)) == 0) return false;
    l.o = getCross(q.n*p.d - p.n*q.d , l.v) / getPLength(l.v);
    return true;
  }
  double getAngle(Plane p,Plane q)      {return getUnsignedAngle(p.n,q.n);}
  bool isParallel(Plane p,Plane q)      {return dcmp(getPLength(getCross(p.n,q.n))) == 0;}
  bool isPerpendicular(Plane p,Plane q) {return dcmp(getDot(p.n,q.n)) == 0;}
  bool getAngle(Plane p,Line3D l)         {return pi/2.0 - getUnsignedAngle(p.n,l.v);}
  bool isParallel(Plane p,Line3D l)       {return dcmp(getDot(p.n,l.v)) == 0;}
  bool isPerpendicular(Plane p,Line3D l)  {return dcmp(getPLength(getCross(p.n,l.v))) == 0;}
  Line3D perpThrough(Plane p,Point3D a)     {return Line3D(p.n,a);}
  Plane perpThrough(Line3D l,Point3D a)     {return Plane(l.v,a);}
  //Modify p.n if necessary with respect to the reference point
  Vector3D rotateCCW90(Plane p,Vector3D d)  {return getCross(p.n,d);}
  Vector3D rotateCW90(Plane p,Vector3D d)   {return getCross(d,p.n);}
  pair<Point3D, Point3D> TwoPointsOnPlane(Plane p){
    Vector3D N = p.n; double D = p.d;
    assert(dcmp(N.x) != 0 || dcmp(N.y) != 0 || dcmp(N.z) != 0);
    if(dcmp(N.x) == 0 && dcmp(N.y) == 0) return {Point3D(1,0,D/N.z), Point3D(0,1,D/N.z)};
    if(dcmp(N.y) == 0 && dcmp(N.z) == 0) return {Point3D(D/N.x,1,0), Point3D(D/N.x,0,1)};
    if(dcmp(N.z) == 0 && dcmp(N.x) == 0) return {Point3D(1,D/N.y,0), Point3D(0,D/N.y,1)};
    if(dcmp(N.x) == 0) return {Point3D(1,D/N.y,0), Point3D(0,0,D/N.z)};
    if(dcmp(N.y) == 0) return {Point3D(0,1,D/N.z), Point3D(D/N.x,0,0)};
    if(dcmp(N.z) == 0) return {Point3D(D/N.x,0,1), Point3D(0,D/N.y,0)};
    if(dcmp(D)!=0) return {Point3D(D/N.x,0,0), Point3D(0,D/N.y,0)};
    return {Point3D(N.y,-N.x,0), Point3D(-N.y,N.x,0)};
  }
  Point From3Dto2D(Plane p, Point3D a){
    assert( dcmp(getSide(p,a)) == 0 );
    auto Pair = TwoPointsOnPlane(p);
    Point3D A = Pair.first;
    Point3D B = Pair.second;
    Vector3D Z = p.n;                    Z = Z / getLength(Z);
    Vector3D X = B - A;                  X = X / getLength(X);
    Vector3D Y = getCross(Z,X);
    Vector3D v = a - A;
    assert( dcmp(getDot(v,Z)) == 0);
    return Point(getDot(v,X),getDot(v,Y));
  }
  Point3D From2Dto3D(Plane p, Point a){
    auto Pair = TwoPointsOnPlane(p);
    Point3D A = Pair.first;
    Point3D B = Pair.second;
    Vector3D Z = p.n;                    Z = Z / getLength(Z);
    Vector3D X = B - A;                  X = X / getLength(X);
    Vector3D Y = getCross(Z,X);
    return A + X * a.x + Y * a.y;
  }
}
```

3.2.3 Point3D

```
using namespace std;
const double pi = 4 * atan(1);
const double eps = 1e-10;
inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }
inline double torad(double deg) { return deg / 180 * pi; }
struct Point{
  double x, y;
  Point (double x = 0, double y = 0): x(x), y(y) {}
```

```
  Point operator + (const Point& u) { return Point(x + u.x,
  ↪  y + u.y); }
  Point operator - (const Point& u) { return Point(x - u.x,
  ↪  y - u.y); }
  Point operator * (const double u) { return Point(x * u, y
  ↪  * u); }
  Point operator / (const double u) { return Point(x / u, y
  ↪  / u); }
  double operator * (const Point& u) { return x*u.y -
  ↪  y*u.x; }
};
struct Point3D{
  double x, y, z;
  Point3D() {}
  void read () {cin>>x>>y>>z;}
  void write () {cout<<x<<" --- "<<y<<" --- "<<z<<"\n";}
  Point3D(double x, double y, double z) : x(x), y(y), z(z)
  ↪  {}
  Point3D(const Point3D &p) : x(p.x), y(p.y), z(p.z) {}
  Point3D operator +(Point3D b)  {return
  ↪  Point3D(x+b.x,y+b.y, z+b.z);}
  Point3D operator -(Point3D b)  {return
  ↪  Point3D(x-b.x,y-b.y, z-b.z);}
  Point3D operator *(double b) {return Point3D(x*b,y*b,
  ↪  z*b);}
  Point3D operator /(double b) {return Point3D(x/b,y/b,
  ↪  z/b);}
  bool operator  <(Point3D b)  {return
      make_pair(make_pair(x,y),z) <
  ↪  make_pair(make_pair(b.x,b.y),b.z);}
  bool operator ==(Point3D b)  {return dcmp(x-b.x)==0 &&
  ↪  dcmp(y-b.y) == 0 && dcmp(z-b.z) == 0;}
};
```

3.2.4 Poygon3D

```
namespace Poly{
  using namespace Vectorial;
  Sphere SmallestEnclosingSphere(Polyhedron p){
    int n = p.size();
    Point3D C(0,0,0);
    for(int i=0; i<n; i++) C = C + p[i];
    C = C / n;
    double P = 0.1;
    int pos = 0;
    int Accuracy = 70000;
    for (int i = 0; i < Accuracy; i++) {
      pos = 0;
      for (int j = 1; j < n; j++){
        if(getPLength(C - p[j]) > getPLength(C - p[pos]))
        ↪  pos = j;
      }
      C = C + (p[pos] - C)*P;
      P *= 0.998;
    }
    return Sphere(C, getPLength(C - p[pos]));
  }
}
struct Pyramid{
  int n;     //number of side of the pyramid
  double l;  //length of each side
  double ang;
  Pyramid(int n, double l) {this->n=n; this->l=l; ang=pi/n;}
  double getBaseArea() {return l * l * n / (4* tan(ang));}
  double getHeight() {return l * sqrt(1 - 1 /
  ↪  (4*sin(ang)*sin(ang)) );}
  double getVolume() {return getBaseArea() * getHeight() /
  ↪  3;}
};
```

3.2.5 Sphere3D

```
struct Sphere{
  Point3D c;
  double r;
  Sphere() {}
  Sphere(Point3D c, double r) : c(c), r(r) {}
  //Spherical cap with polar angle theta
  double Height(double alpha)      {return
  ↪  r*(1-cos(alpha));}
  double BaseRadius(double alpha)  {return r*sin(alpha);}
  double Volume(double alpha)      {double h =
  ↪  Height(alpha); return pi*h*h*(3*r-h)/3.0;}
  double SurfaceArea(double alpha) {double h =
  ↪  Height(alpha); return 2*pi*r*h;}
};
namespace Spherical{
  using namespace Vectorial;
  using namespace Planar;
  using namespace Linear;
  Sphere CircumscribedSphere(Point3D a,Point3D b,Point3D
  ↪  c,Point3D d){
    assert( dcmp(getSide(getPlane(a,b,c), d)) != 0);
    Plane U = Plane(a-b, (a+b)/2);
    Plane V = Plane(b-c, (b+c)/2);
    Plane W = Plane(c-d, (c+d)/2);
    Line3D l1,l2;
    bool ret1 = intersect(U,V,l1);
    bool ret2 = intersect(V,W,l2);
    assert(ret1 == true && ret2 == true);
    assert( dcmp(getDist(l1,l2)) == 0);
    Point3D C = getClosestPointOnLine1(l1,l2);
    return Sphere(C, getLength(C-a));
  }
  pair<double,double> SphereSphereIntersection(Sphere
  ↪  s1,Sphere s2){
    double d = getLength(s1.c-s2.c);
    if(dcmp(d - s1.r -s2.r) >= 0) return {0,0};
    double R1 = max(s1.r,s2.r); double R2 = min(s1.r,s2.r);
    double y = R1 + R2 - d;
    double x = (R1*R1 - R2*R2 + d*d) / (2*d);
    double h1 = R1 - x;
    double h2 = y - h1;
    double Volume      = pi*h1*h1*(3*R1-h1)/3.0 +
    ↪  pi*h2*h2*(3*R2-h2)/3.0;
    double SurfaceArea = 2*pi*R1*h1 + 2*pi*R2*h2;
    return make_pair(SurfaceArea,Volume);
  }
  Point3D getPointOnSurface(double r,double Lat,double Lon){
    Lat = torad(Lat);  //North-South
    Lon = torad(Lon);  //East-West
    return Point3D(r*cos(Lat)*cos(Lon),
    ↪  r*cos(Lat)*sin(Lon), r*sin(Lat));
  }
  int intersect(Sphere s,Line3D l, vector<Point3D>& ret){
    double h2 = s.r*s.r - getDistSq(l,s.c);
    if(dcmp(h2)<0) return 0;
    Point3D p = projection(l,s.c);
    if(dcmp(h2) == 0) {ret.push_back(p); return 1;}
    Vector3D h = l.v * sqrt(h2) / getLength(l.v);
    ret.push_back(p-h); ret.push_back(p+h); return 2;
  }
  double GreatCircleDistance(Sphere s,Point3D a,Point3D b){
    return s.r * getUnsignedAngle(a-s.c, b-s.c);
  }
}
```

3.3 Closest Pair

Finds the closest pair of points.
Time: $O(n \log n)$

```
#include "Point.h"
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p +
    ↪  d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

3.4 Convex Hull

```
typedef pair <ll, ll> point;
inline ll area (point a, point b, point c) {
  return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x -
  ↪  a.x);
}
vector <point> convexHull (vector <point> p) {
  int n = p.size(), m = 0;
  if (n < 3) return p;
  vector <point> hull(n + n);
  sort(p.begin(), p.end());
  for (int i = 0; i < n; ++i) {
    while (m > 1 and area(hull[m - 2], hull[m - 1], p[i])
    ↪  <= 0) --m;
    hull[m++] = p[i];
  }
  for (int i = n - 2, j = m + 1; i >= 0; --i) {
    while (m >= j and area(hull[m - 2], hull[m - 1], p[i])
    ↪  <= 0) --m;
    hull[m++] = p[i];
  }
  hull.resize(m - 1); return hull;
}
```

3.5 Minkowski Sum

Description: Convex hull of minkowski sum of two convex polygons P and Q.

```
void reorder_polygon(vector<pt> & P){
  size_t pos = 0;
  for(size_t i = 1; i < P.size(); i++){
    if(P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x <
    ↪  P[pos].x))
      pos = i;
  }
  rotate(P.begin(), P.begin() + pos, P.end());
}
vector<pt> minkowski(vector<pt> P, vector<pt> Q){
  // the first vertex must be the lowest
  reorder_polygon(P); reorder_polygon(Q);
  // we must ensure cyclic indexing
  P.push_back(P[0]); P.push_back(P[1]);
  Q.push_back(Q[0]); Q.push_back(Q[1]);
  // main part
  vector<pt> result;
  size_t i = 0, j = 0;
  while(i < P.size() - 2 || j < Q.size() - 2){
    result.push_back(P[i] + Q[j]);
    auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
    if(cross >= 0 && i < P.size() - 2) ++i;
    if(cross <= 0 && j < Q.size() - 2) ++j;
  }
  return result;
}
```

3.6 Polar Sort

Description: Sorts points CCW.

```
inline bool up (point p) {
  return p.y > 0 or (p.y == 0 and p.x >= 0);
}
sort(v.begin(), v.end(), [] (point a, point b) {
  return up(a) == up(b) ? a.x * b.y > a.y * b.x : up(a) < up(b);
});
```

3.7 Polyhedron Volume

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

```
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilist) {
 double v = 0;
 for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
 return v / 6;
}
```

3.8 Spherical Distance

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

```
double sphericalDistance(double f1, double t1,
  double f2, double t2, double radius) {
 double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
 double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
 double dz = cos(t2) - cos(t1);
 double d = sqrt(dx*dx + dy*dy + dz*dz);
 return radius*2*asin(d/2);
}
```

4 Graph

4.1 2-SAT

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a\|\|b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&...$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (x). Usage: TwoSat ts(number of boolean variables); ts.either(0,); // Var 0 is true or var 3 is false ts.set_value(2); // Var 2 is true ts.at_most_one(0,,2); // <= 1 of vars 0, and 2 are true ts.solve(); // Returns true iff it is solvable ts.values[0..N-1] holds the assigned values to the vars
Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

```
struct TwoSat {
  int N;
  vector<vector<int>> gr;
  vector<int> values; // 0 = false, 1 = true
  TwoSat(int n = 0) : N(n), gr(2 * n) {}
  int add_var() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }
  void either(int f, int j) {
    f = max(2 * f, -1 - 2 * f);
    j = max(2 * j, -1 - 2 * j);
    gr[f].push_back(j ^ 1);
    gr[j].push_back(f ^ 1);
  }
  void set_value(int x) { either(x, x); }
  void at_most_one(const vector<int> &li) { // (optional)
    if ((int)li.size() <= 1) return;
    int cur = ~li[0];
    for (int i = 2; i < (int)li.size(); i++) {
      int next = add_var();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }
  vector<int> val, comp, z;
  int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x;
    z.push_back(i);
    for (auto &e : gr[i])
      if (!comp[e]) low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back();
        z.pop_back();
        comp[x] = low;
        if (values[x >> 1] == -1) values[x >> 1] = x & 1;
      } while (x != i);
    return val[i] = low;
  }
  bool solve() {
    values.assign(N, -1);
    val.assign(2 * N, 0);
    comp = val;
    for (int i = 0; i < 2 * N; i++)
      if (!comp[i]) dfs(i);
    }
    for (int i = 0; i < N; i++) {
      if (comp[2 * i] == comp[2 * i + 1]) return 0;
    }
    return 1;
  }
};
```

4.2 AP and Bridge

```
using namespace std;
const int N = 1e5 + 10;
vector<int> g[N];
int vis[N], low[N], cut[N], now = 0, n, m;
void dfs(int u, int p) {
  low[u] = vis[u] = ++now; int ch = 0;
  for(int v : g[u]){
    if(v ^ p) {
      if(vis[v]) low[u] = min(low[u], vis[v]);
      else {
        ch++; dfs(v, u);
        low[u] = min(low[u], low[v]);
        if(p + 1 && low[v] >= vis[u]) cut[u] = 1;
        if(low[v] > vis[u]) {
          printf("Bridge %d -- %d\n", u, v);
        }
      }
    }
  } if(p == -1 && ch > 1) cut[u] = 1;
}
void fuck() {
  memset(vis, 0, sizeof vis);
  memset(low, 0, sizeof low);
  memset(cut, 0, sizeof cut);
  now = 0;
  for(int i = 0; i < n; i++) {
    if(!vis[i]) dfs(i, -1);
  }
}
```

4.3 CompressTree

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S|-1$) pairwise LCAs and compressing edges. Tree is stored in virt[] with edge costs at cost[].
Time: $\mathcal{O}(|S|\log|S|)$

```
vector <int> g[N], virt[N], cost[N];
int n, m, ptr, h[N], in[N], stk[N];
void add_edge (int u, int v) {
  if (u == v) return;
  virt[u].emplace_back(v);
  virt[v].emplace_back(u);
  int w = abs(h[u] - h[v]);
  cost[u].emplace_back(w);
  cost[v].emplace_back(w);
}
void buildTree (vector <int> &nodes) {
  if (nodes.size() <= 1) return;
  sort(nodes.begin(), nodes.end(), [] (int x, int y) {return in[x] < in[y];});
  int root = get_lca(nodes[0], nodes.back()), sz = nodes.size();
  ptr = 0, stk[ptr++] = root;
  for (int i = 0; i < sz; ++i) {
    int u = nodes[i], lca = get_lca(u, stk[ptr - 1]);
    if (lca == stk[ptr - 1]) {
      stk[ptr++] = u;
    } else {
      while (ptr > 1 and h[stk[ptr - 2]] >= h[lca]) {
        add_edge(stk[ptr - 2], stk[ptr - 1]), --ptr;
      }
      if (stk[ptr - 1] != lca) {
        add_edge(lca, stk[--ptr]);
        stk[ptr++] = lca, nodes.emplace_back(lca);
      } stk[ptr++] = u;
    }
  }
  if (find(nodes.begin(), nodes.end(), root) == nodes.end()) nodes.emplace_back(root);
  for (int j = 0; j + 1 < ptr; ++j) add_edge(stk[j], stk[j + 1]);
}
int main() {
  cin >> m; vector <int> nodes(m);
  for (int i = 0; i < m; ++i) cin >> nodes[i];
  buildTree(nodes);
}
```

4.4 DSU On Tree

Description: DSU on tree. Count number of vertices of color c in a subtree.
Time: $\mathcal{O}(n\log n)$

```
int cnt[N];
void dfs (int u, int par, bool keep) {
  int mx = -1, bigChild = -1;
  for (auto v : g[u])
    if (v != par and sz[v] > mx)
      mx = sz[v], bigChild = v;
  for (auto v : g[u]) if (v != par and v != bigChild) {
    dfs(v, u, 0); // small child, clear them before exiting
  }
  if (bigChild != -1)
    dfs(bigChild, u, 1); // don't clear big child
  for (auto v : g[u]) if (v != par and v != bigChild) {
    for (int i = in[v]; i <= out[v]; ++i)
      ++cnt[col[flat[i]]];
  }
  ++cnt[col[u]];
  // now cnt[c] is the number of vertices in subtree of vertex u with color c, answer queries
  if (!keep) { // clear this subtree
    for (int i = in[u]; i <= out[u]; ++i)
      --cnt[col[flat[i]]];
  }
}
```

4.5 Dinic

Description: Lower bound on capacity – create a supersource, a supersink. If $u \to v$ has a lower bound of L, give an edge from supersource to v with capacity L. Give an edge from u to supersink with capacity L. Give an edge from normal sink to normal source with capacity infinity. If max flow here is equal to L, then the lower bound can be satisfied. For minimum flow satisfying lower bounds, binary search on the capacity from normal sink to normal source (instead of assigning inf). For maximum flow satisfying bounds, just add another source to normal source and binary search on capacity.

Time: $\mathcal{O}\left(V^2E\right)$, (on unit graphs $\mathcal{O}\left(E\sqrt{V}\right)$)

```cpp
struct edge {
  int u, v; ll cap, flow;
  edge () {}
  edge (int u, int v, ll cap) : u(u), v(v), cap(cap),
    flow(0) {}
};
struct Dinic {
  int N; vector <edge> E;
  vector <vector <int>> g;
  vector <int> d, pt;
  Dinic (int N) : N(N), E(0), g(N), d(N), pt(N) {}
  void AddEdge (int u, int v, ll cap) {
    if (u ^ v) {
      E.emplace_back(u, v, cap);
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(v, u, 0);
      g[v].emplace_back(E.size() - 1);
    }
  }
  bool BFS (int S, int T) {
    queue <int> q({S});
    fill(d.begin(), d.end(), N + 1); d[S] = 0;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k : g[u]) {
        edge &e = E[k];
        if (e.flow < e.cap and d[e.v] > d[e.u] + 1) {
          d[e.v] = d[e.u] + 1; q.emplace(e.v);
        }
      }
    } return d[T] != N + 1;
  }
  ll DFS (int u, int T, ll flow = -1) {
    if (u == T or flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      edge &e = E[g[u][i]];
      edge &oe = E[g[u][i] ^ 1];
      if (d[e.v] == d[e.u] + 1) {
        ll amt = e.cap - e.flow;
        if (flow != -1 and amt > flow) amt = flow;
        if (ll pushed = DFS(e.v, T, amt)) {
          e.flow += pushed; oe.flow -= pushed;
          return pushed;
        }
      }
    } return 0;
  }
  ll MaxFlow (int S, int T) {
    ll total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (ll flow = DFS(S, T)) total += flow;
    } return total;
  }
};
```

4.6 Directed MST

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

Time: $\mathcal{O}(E\log V)$

```cpp
struct Edge { int a, b; ll w; };
struct Node {
 Edge key; Node *l, *r; ll delta;
 void prop() {
  key.w += delta;
  if (l) l->delta += delta;
  if (r) r->delta += delta;
  delta = 0;
 } Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
 if (!a || !b) return a ?: b;
 a->prop(), b->prop();
 if (a->key.w > b->key.w) swap(a, b);
 swap(a->l, (a->r = merge(b, a->r)));
 return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
 RollbackUF uf(n);
 vector<Node*> heap(n);
 for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
 ll res = 0; vi seen(n, -1), path(n), par(n);
 seen[r] = r; vector<Edge> Q(n), in(n, {-1,-1}), comp;
 deque<tuple<int, int, vector<Edge>>> cycs;
 rep(s,0,n) {
  int u = s, qi = 0, w;
  while (seen[u] < 0) {
   if (!heap[u]) return {-1,{}};
   Edge e = heap[u]->top();
   heap[u]->delta -= e.w, pop(heap[u]);
   Q[qi] = e, path[qi++] = u, seen[u] = s;
   res += e.w, u = uf.find(e.a);
   if (seen[u] == s) {
    Node* cyc = 0;
    int end = qi, time = uf.time();
    do cyc = merge(cyc, heap[w = path[--qi]]);
    while (uf.join(u, w));
    u = uf.find(u), heap[u] = cyc, seen[u] = -1;
    cycs.push_front({u, time, {&Q[qi], &Q[end]}});
   }
  } rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
 }
 for (auto& [u,t,comp] : cycs) { // restore sol (optional)
  uf.rollback(t); Edge inEdge = in[u];
  for (auto& e : comp) in[uf.find(e.b)] = e;
  in[uf.find(inEdge.b)] = inEdge;
 } rep(i,0,n) par[i] = in[i].a; return {res, par};
}
```

4.7 Dominator Tree

Description: A node u is ancestor of node v in the dominator tree if all the the paths from source to node v contain node u. If a problem asks for edge disjoint paths, for every edge, take a new node w and turn the edge $(u \to v)$ to $(u \to w \to v)$ and find node disjoint path now. 1-based directed graph input. dtree is the edge list of the dominator tree. Clear everything at the start of each test case. Only the nodes reachable from source will be in the dominator tree.

Time: construction $\mathcal{O}(V+E)$

```cpp
vector <int> g[sz], rg[sz], dtree[sz], bucket[sz];
int sdom[sz], par[sz], dom[sz], dsu[sz], label[sz];
int arr[sz], rev[sz], ts, source;
void dfs(int u) {
    ts++; arr[u] = ts; rev[ts] = u;
    label[ts] = sdom[ts] = dsu[ts] = ts;
    for(int &v : g[u]) {
        if(!arr[v]) { dfs(v); par[arr[v]] = arr[u]; }
        rg[arr[v]].push_back(arr[u]);
    }
}
inline int root(int u, int x = 0) {
    if(u == dsu[u]) return x ? -1 : u;
    int v = root(dsu[u], x + 1);
    if(v < 0) return u;
    if(sdom[label[dsu[u]]] < sdom[label[u]]) label[u] =
      label[dsu[u]];
    dsu[u] = v; return x ? v : label[u];
}
void build(int n) {
    dfs(source);
    for(int i=n; i; i--) {
        for(int j : rg[i]) sdom[i] =
          min(sdom[i],sdom[root(j)]);
        if(i > 1) bucket[sdom[i]].push_back(i);
        for(int w : bucket[i]) {
            int v = root(w);
            if(sdom[v] == sdom[w]) dom[w] = sdom[w];
            else dom[w] = v;
        } if(i > 1) dsu[i] = par[i];
    }
    for(int i=2; i<=n; i++) {
        int &dm = dom[i];
        if(dm ^ sdom[i]) dm = dom[dm];
        dtree[rev[i]].push_back(rev[dm]);
        dtree[rev[dm]].push_back(rev[i]);
    }
}
int main() {
    // input graph of n nodes in g[], assign "source" vertex
    ts = 0; build(n); // clear stuff before calling
}
```

4.8 Edge Coloring

Description: Given a simple, undirected graph with max degree D, computes a $(D+1)$-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

Time: $\mathcal{O}(NM)$

```cpp
vi edgeColoring(int N, vector<pii> eds) {
 vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
 for (pii e : eds) ++cc[e.first], ++cc[e.second];
 int u, v, ncols = *max_element(all(cc)) + 1;
 vector<vi> adj(N, vi(ncols, -1));
 for (pii e : eds) {
  tie(u, v) = e;
  fan[0] = v;
  loc.assign(ncols, 0);
  int at = u, end = u, d, c = free[u], ind = 0, i = 0;
  while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
   loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
  cc[loc[d]] = c;
  for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
   swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
  while (adj[fan[i]][d] != -1) {
   int left = fan[i], right = fan[++i], e = cc[i];
   adj[u][e] = left;
   adj[left][e] = u;
   adj[right][e] = -1;
   free[right] = e;
  }
  adj[u][d] = fan[i];
  adj[fan[i]][d] = u;
  for (int y : {fan[0], u, end})
   for (int& z = free[y] = 0; adj[y][z] != -1; z++);
 }
 rep(i,0,sz(eds))
  for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
 return ret;
}
```

4.9 Euler Walk

Description: On directed graph, circuit (or edge disjoint directed cycles) exists iff each node satisfies in_degree = out_degree and the graph is strongly connected; path exists iff at most one vertex has in_degree - out_degree = 1 and at most one vertex has out_degree - in_degree = 1 and all other vertices have in_degree = out_degree, and graph is weakly connected. Push edge ID in circ if edges needed.

Time: $\mathcal{O}(V+E)$

```
bitset <N> bad;
vector <int> g[N], circ;
int n, m, deg[N], U[N], V[N];
void hierholzer (int src) {
  if (!deg[src]) return;
  vector <int> path;
  path.push_back(src);
  int at = src;
  while (!path.empty()) {
    if (deg[at]) {
      path.push_back(at);
      while (bad[g[at].back()]) g[at].pop_back();
      int e = g[at].back(), nxt = U[e] ^ at ^ V[e];
      bad[e] = 1, --deg[at], --deg[nxt], at = nxt;
    } else {
      circ.push_back(at);
      at = path.back(), path.pop_back();
    }
  } reverse(circ.begin(), circ.end());
}
int main() {
  cin >> n >> m;
  for (int i = 1; i <= m; ++i) {
    scanf("%d %d", U + i, V + i);
    g[U[i]].push_back(i);
    g[V[i]].push_back(i);
    ++deg[U[i]], ++deg[V[i]];
  }
  hierholzer(1); // run loop if not connected
  for (int x : circ) printf("%d ", x); puts("");
}
```

4.10 GeneralMatching

```
using namespace std;
const int N = 505;
struct Blossom {
  queue <int> q;
  vector <int> g[N];
  int t, n, vis[N], par[N], orig[N], match[N], aux[N];
  Blossom (int _n = 0) {
    n = _n, t = 0;
    for (int i = 0; i <= n; ++i) {
      g[i].clear(), match[i] = aux[i] = par[i] = 0;
    }
  }
  inline void addEdge (int u, int v) {
    g[u].push_back(v), g[v].push_back(u);
  }
  void augment (int u, int v) {
    int pv = v, nv;
    do {
      pv = par[v], nv = match[pv];
      match[v] = pv, match[pv] = v, v = nv;
    } while (u ^ pv);
  }
  int lca (int u, int v) {
    ++t;
    while (true) {
      if (u) {
        if (aux[u] == t) return u; aux[u] = t;
        u = orig[par[match[u]]];
      }
      swap(u, v);
    }
  }
  void blossom (int u, int v, int x) {
    while (orig[u] ^ x) {
      par[u] = v, v = match[u];
      if (vis[v] == 1) q.emplace(v), vis[v] = 0;
      orig[u] = orig[v] = x, u = par[v];
    }
  }
  bool bfs (int src) {
    fill(vis + 1, vis + n + 1, -1);
    iota(orig + 1, orig + n + 1, 1);
    while (!q.empty()) q.pop();
    q.emplace(src), vis[src] = 0;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      for (int v : g[u]) {
        if (vis[v] == -1) {
          par[v] = u, vis[v] = 1;
          if (!match[v]) return augment(src, v), 1;
          q.emplace(match[v]), vis[match[v]] = 0;
        } else if (vis[v] == 0 and orig[u] ^ orig[v]) {
          int x = lca(orig[u], orig[v]);
          blossom(v, u, x), blossom(u, v, x);
        }
      }
    } return 0;
  }
  int maxMatch() {
    int ans = 0;
    vector <int> vec(n - 1);
    iota(vec.begin(), vec.end(), 1);
    shuffle(vec.begin(), vec.end(), mt19937(69));
    for (int u : vec) if (!match[u]) {
      for (int v : g[u]) if (!match[v]) {
        match[u] = v, match[v] = u;
        ++ans; break;
      }
    }
    for (int i = 1; i <= n; ++i) if (!match[i] and bfs(i))
    ↪ ++ans;
    return ans;
  }
};
int n, m;
int main() {
  cin >> n >> m;
  Blossom yo(n);
  while (m--) {
    int u, v;
    scanf("%d %d", &u, &v);
    ++u, ++v;
    yo.addEdge(u, v);
  }
  cout << yo.maxMatch() << '\n';
  for (int i = 1; i <= n; ++i) {
    if (yo.match[i] > i) printf("%d %d\n", i - 1,
    ↪ yo.match[i] - 1);
  }
  return 0;
}
```

4.11 GeneralMatching

Description: Matching for general graphs. Fails with probability N/mod.

Time: $\mathcal{O}(N^3)$

```
vector<pii> generalMatching(int N, vector<pii>& ed) {
 vector<vector<ll>> mat(N, vector<ll>(N)), A;
 for (pii pa : ed) {
  int a = pa.first, b = pa.second, r = rand() % mod;
  mat[a][b] = r, mat[b][a] = (mod - r) % mod;
 }
 int r = matInv(A = mat), M = 2*N - r, fi, fj;
 assert(r % 2 == 0);
 if (M != N) do {
  mat.resize(M, vector<ll>(M));
  rep(i,0,N) {
   mat[i].resize(M);
   rep(j,N,M) {
    int r = rand() % mod;
    mat[i][j] = r, mat[j][i] = (mod - r) % mod;
   }
  }
 } while (matInv(A = mat) != M);
 vi has(M, 1); vector<pii> ret;
 rep(it,0,M/2) {
  rep(i,0,M) if (has[i])
   rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
    fi = i; fj = j; goto done;
  } assert(0); done:
  if (fj < N) ret.emplace_back(fi, fj);
  has[fi] = has[fj] = 0;
  rep(sw,0,2) {
   ll a = modpow(A[fi][fj], mod-2);
   rep(i,0,M) if (has[i] && A[i][fj]) {
    ll b = A[i][fj] * a % mod;
    rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
   } swap(fi,fj);
  }
 } return ret;
}
```

4.12 Global MinCut

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.

Time: $\mathcal{O}(V^3)$

```
pair<int, vi> globalMinCut(vector<vi> mat) {
 pair<int, vi> best = {INT_MAX, {}};
 int n = sz(mat); vector<vi> co(n);
 rep(i,0,n) co[i] = {i};
 rep(ph,1,n) {
  vi w = mat[0];
  size_t s = 0, t = 0;
  rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio. queue
   w[t] = INT_MIN;
   s = t, t = max_element(all(w)) - w.begin();
   rep(i,0,n) w[i] += mat[t][i];
  }
  best = min(best, {w[t] - mat[t][t], co[t]});
  co[s].insert(co[s].end(), all(co[t]));
  rep(i,0,n) mat[s][i] += mat[t][i];
  rep(i,0,n) mat[i][s] = mat[s][i];
  mat[0][t] = INT_MIN;
 }
 return best;
}
```

4.13 Hopcroft Karp

Description: Fast bipartite matching. Clear match[] first. Graph is normal, undirected.

Time: $\mathcal{O}(E\sqrt{V})$

```
bool bfs() {
  queue <int> q;
  for (int i = 1; i <= n; ++i) {
    if (!match[i]) dist[i] = 0, q.emplace(i);
    else dist[i] = INF;
  }
  dist[0] = INF;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    if (!u) continue;
    for (int v : g[u]) {
      if (dist[match[v]] == INF) {
        dist[match[v]] = dist[u] + 1, q.emplace(match[v]);
      }
    }
  } return dist[0] != INF;
}
bool dfs (int u) {
  if (!u) return 1;
  for (int v : g[u]) {
    if (dist[match[v]] == dist[u] + 1 and dfs(match[v])) {
      match[u] = v, match[v] = u; return 1;
    }
  } dist[u] = INF; return 0;
}
int hopcroftKarp() {
  int ret = 0;
  while (bfs()) {
    for (int i = 1; i <= n; ++i) {
```

```cpp
      ret += !match[i] and dfs(i);
    }
  } return ret;
}
```

4.14 MCMF

```cpp
namespace mcmf {
using T = int;
const T INF = ?;     // 0x3f3f3f3f or 0x3f3f3f3f3f3f3f3fLL
const int MAX = ?;     // maximum number of nodes
int n, src, snk;
T dis[MAX], mCap[MAX];
int par[MAX], pos[MAX];
bool vis[MAX];
struct Edge {
  int to, rev_pos;
  T cap, cost, flow;
};
vector<Edge> ed[MAX];
void init(int _n, int _src, int _snk) {
  n = _n, src = _src, snk = _snk;
  for (int i = 1; i <= n; i++) ed[i].clear();
}
void addEdge(int u, int v, T cap, T cost) {
  Edge a = {v, (int)ed[v].size(), cap, cost, 0};
  Edge b = {u, (int)ed[u].size(), 0, -cost, 0};
  ed[u].push_back(a);
  ed[v].push_back(b);
}
inline bool SPFA() {
  memset(vis, false, sizeof vis);
  for (int i = 1; i <= n; i++) mCap[i] = dis[i] = INF;
  queue<int> q;
  dis[src] = 0;
  vis[src] = true;
  q.push(src);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = false;
    for (int i = 0; i < (int)ed[u].size(); i++) {
      Edge &e = ed[u][i];
      int v = e.to;
      if (e.cap > e.flow && dis[v] > dis[u] + e.cost) {
        dis[v] = dis[u] + e.cost;
        par[v] = u;
        pos[v] = i;
        mCap[v] = min(mCap[u], e.cap - e.flow);
        if (!vis[v]) {
          vis[v] = true;
          q.push(v);
        }
      }
    }
  }
  return (dis[snk] != INF);
}
inline pair<T, T> solve() {
  T F = 0, C = 0, f;
  int u, v;
  while (SPFA()) {
    u = snk;
    f = mCap[u];
    F += f;
    while (u != src) {
      v = par[u];
      ed[v][pos[u]].flow += f;            // edge of v-->u increases
      ed[u][ed[v][pos[u]].rev_pos].flow -= f;
      u = v;
    }
    C += dis[snk] * f;
  }
  return make_pair(F, C);
}
}
```

4.15 MST Boruvka

Description: While there are more than one components, Find the closest weight edge that connects this component to any other component and Add this closest edge to MST if not already added.

4.16 Manhattan MST

```cpp
using namespace std;
using ll = long long;
struct UnionFind {
  vector<int> UF; int cnt; UnionFind(int N) : UF(N, -1), cnt(N) {}
  int find(int v) { return UF[v] < 0 ? v : UF[v] = find(UF[v]); }
  bool join(int v, int w) {
    if ((v = find(v)) == (w = find(w))) return false;
    if (UF[v] > UF[w]) swap(v, w);
    UF[v] += UF[w]; UF[w] = v; cnt--; return true;
  }
  bool connected(int v, int w) { return find(v) == find(w); }
  int getSize(int v) { return -UF[find(v)]; }
};
template <class T> struct KruskalMST {
  using Edge = tuple<int, int, T>;
  T mstWeight; vector<Edge> mstEdges; UnionFind uf;
  KruskalMST(int V, vector<Edge> edges) : mstWeight(), uf(V) {
    sort(edges.begin(), edges.end(), [&] (const Edge &a, const Edge &b) {
      return get<2>(a) < get<2>(b);
    });
    for (auto &&e : edges) {
      if (int(mstEdges.size()) >= V - 1) break;
      if (uf.join(get<0>(e), get<1>(e))) {
        mstEdges.push_back(e); mstWeight += get<2>(e);
      }
    }
  }
};
template <class T> struct ManhattanMST : public KruskalMST<T> {
  using Edge = typename KruskalMST<T>::Edge;
  static vector<Edge> generateCandidates(vector<pair<T, T>> P) {
    vector<int> id(P.size()); iota(id.begin(), id.end(), 0); vector<Edge> ret;
    ret.reserve(P.size() * 4); for (int h = 0; h < 4; h++) {
      sort(id.begin(), id.end(), [&] (int i, int j) {
        return P[i].first - P[j].first < P[j].second - P[i].second;
      });
      map<T, int> M; for (int i : id) {
        auto it = M.lower_bound(-P[i].second);
        for (; it != M.end(); it = M.erase(it)) {
          int j = it->second;
          T dx = P[i].first - P[j].first, dy = P[i].second - P[j].second;
          if (dy > dx) break;
          ret.emplace_back(i, j, dx + dy);
        }
        M[-P[i].second] = i;
      }
      for (auto &&p : P) {
        if (h % 2) p.first = -p.first;
        else swap(p.first, p.second);
      }
    }
    return ret;
  }
  ManhattanMST(const vector<pair<T, T>> &P)
      : KruskalMST<T>(P.size(), generateCandidates(P)) {}
};
int main() {
  int N; cin >> N;
  vector<pair<ll, ll>> P(N);
  for (auto &&p : P) cin >> p.first >> p.second;
  ManhattanMST mst(P);
  cout << mst.mstWeight << '\n';
  for (auto &&[v, w, weight] : mst.mstEdges) cout << v << ' ' << w << '\n';
  return 0;
}
```

4.17 Maximum Clique

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.

Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

```cpp
typedef vector<bitset<200>> vb;
struct Maxclique {
 double limit=0.025, pk=0;
 struct Vertex { int i, d=0; };
 typedef vector<Vertex> vv;
 vb e; vv V; vector<vi> C; vi qmax, q, S, old;
 void init(vv& r) {
  for (auto& v : r) v.d = 0;
  for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
  sort(all(r), [](auto a, auto b) { return a.d > b.d; });
  int mxD = r[0].d;
  rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
 }
 void expand(vv& R, int lev = 1) {
  S[lev] += S[lev - 1] - old[lev];
  old[lev] = S[lev - 1];
  while (sz(R)) {
   if (sz(q) + R.back().d <= sz(qmax)) return;
   q.push_back(R.back().i);
   vv T;
   for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
   if (sz(T)) {
    if (S[lev]++ / ++pk < limit) init(T);
    int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
    C[1].clear(), C[2].clear();
    for (auto v : T) {
     int k = 1;
     auto f = [&](int i) { return e[v.i][i]; };
     while (any_of(all(C[k]), f)) k++;
     if (k > mxk) mxk = k, C[mxk + 1].clear();
     if (k < mnk) T[j++].i = v.i;
     C[k].push_back(v.i);
    }
    if (j > 0) T[j - 1].d = 0;
    rep(k,mnk,mxk + 1) for (int i : C[k])
     T[j].i = i, T[j++].d = k;
    expand(T, lev + 1);
   } else if (sz(q) > sz(qmax)) qmax = q;
   q.pop_back(), R.pop_back();
  }
 }
 vi maxClique() { init(V), expand(V); return qmax; }
 Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
  rep(i,0,sz(e)) V.push_back({i});
 }
};
```

4.18 Maximum Independent Set

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

4.19 MinCut

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

4.20 Minimum Vertex Cover

Description: Suppose you have left and right parts in bipartite graph and you found the maximum matching M. Let's define orientation of edges. Those edges that belong to M will go from right to left, all other edges will go from left to right. Now run DFS starting at all left vertices that are not incident to edges in M. Some vertices of the graph will become visited during this DFS and some not-visited. To get minimum vertex cover you take all visited right vertices of M, and all not-visited left vertices of M.

4.21 SCC

```cpp
// col[u] stores the component number u belongs to
vector<int> adj[N], trans[N];
int col[n], vis[n], idx = 0, n, m;
stack<int> st;
void dfs(int u) {
  vis[u] = 1;
  for(int v : adj[u]) if(!vis[v]) dfs(v);
  st.push(u);
}
void dfs2(int u) {
  col[u] = idx;
  for(int v : trans[u]) if(!col[v]) dfs2(v);
}
void scc() {
  for(int i = 1; i <= n; i++){
    if(!vis[i]) dfs(i);
  }
  while(!st.empty()) {
    int u = st.top(); st.pop();
    if(col[u]) continue;
    idx++; dfs2(u);
  }
}
```

4.22 TreeHashAllRoot

Description: Find hashes of a tree when rooted at each possible node (unrooted tree isomorphism test).
Time: $\mathcal{O}(n)$

```cpp
const int sz = 2e5+5, mod = 1e9+7;
ll hval[sz], h[sz], dp[sz], rans[sz];
void dfs(vector <vector<int>> &g, int u = 0, int p = -1, int val = 0, int up = 0) {
   vector <int> cv, cht;       // current child values & heights
   if(u > 0) {
      cv.push_back(val);
      cht.push_back(up);
   }
   for(int v : g[u]) if(v - p) {
      cv.push_back(dp[v]);
      cht.push_back(1 + h[v]);
   }
   sort(cht.begin(), cht.end(), greater<int>());
   if(cv.size() > 1) {
      ll ret[] = {1, 1};     // for biggest & 2nd-biggest heights
      for(int i=0; i<2; i++)
         for(int value : cv)
            ret[i] = ret[i] * (hval[cht[i]] + value) % mod;
      rans[u] = ret[0];   // biggest is hash for this root
      for(int v : g[u]) if(v - p) {
         int id = 1;
         if(cht[0] - 1 - h[v]) id = 0;   // v is not on the biggest height path
         val = ret[id] * invmod((hval[cht[id]] + dp[v]) % mod) % mod;
         /* division of v subtree hash value */
         dfs(g, v, u, val, cht[id] + 1);
      }
   }
   else if(cv.size()) {  // Leaf node u OR vertex - 1 has only one child
      if(!up) val = 1;
      else val = (val + hval[up]) % mod;
      rans[u] = val;
      for(int v : g[u]) if(v - p) dfs(g, v, u, val, up + 1);
   }
}
ll get(vector <vector<int>> &g, int u = 0, int p = -1) {
   h[u] = 0;
   vector <ll> childs;
   for(int v : g[u]) if(v - p) {
      childs.push_back(get(g, v, u));
      h[u] = max(h[u], 1 + h[v]);
   }
   ll ret = 1;
   for(int value : childs) ret = ret * (hval[h[u]] + value) % mod;
   return dp[u] = ret;
}
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int main() { // can remove the g as param, can change to 1-index, multi-test works, no further change
      get(g); dfs(g);
      tree[k] = rans[0] = dp[0];
      // rans[i] = tree hash with i as root
}
```

4.23 Weighted Matching

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost.
Time: $\mathcal{O}(N^2M)$

```cpp
pair<int, vi> hungarian(const vector<vi> &a) {
 if (a.empty()) return {0, {}};
 int n = sz(a) + 1, m = sz(a[0]) + 1;
 vi u(n), v(m), p(m), ans(n - 1);
 rep(i,1,n) {
  p[0] = i;
  int j0 = 0; // add "dummy" worker 0
  vi dist(m, INT_MAX), pre(m, -1);
  vector<bool> done(m + 1);
  do { // dijkstra
   done[j0] = true;
   int i0 = p[j0], j1, delta = INT_MAX;
   rep(j,1,m) if (!done[j]) {
    auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
    if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
    if (dist[j] < delta) delta = dist[j], j1 = j;
   }
   rep(j,0,m) {
    if (done[j]) u[p[j]] += delta, v[j] -= delta;
    else dist[j] -= delta;
   }
   j0 = j1;
  } while (p[j0]);
  while (j0) { // update alternating path
   int j1 = pre[j0];
   p[j0] = p[j1], j0 = j1;
  }
 }
 rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
 return {-v[0], ans}; // min cost
}
```

5 Math

5.1 CRT

Description: Solves for $x \equiv a_i \pmod{m_i}$.

```cpp
ll normalize(ll x, ll mod) {
  x %= mod;
  return (x < 0) ? x + mod : x;
}
pair<ll, ll> CRT(vector<ll> a, vector<ll> m) {
  int n = a.size();
  for(int i = 0; i < n; i++) normalize(a[i], m[i]);
  ll ans = a[0], lcm = m[0];
  for(int i = 1; i < n; i++) {
    auto [d, x1, y1] = exgcd(lcm, m[i]);
    if((a[i] - ans) % d) return {-1, -1};
    ans = normalize(ans + x1 * (a[i] - ans) / d % (m[i] / d) * lcm, lcm * m[i] / d);
    lcm = lcm * m[i] / d;
  }
  return {ans, lcm};
}
```

5.2 Diophantine

Description: For any solution (x_0, y_0), all solutions are of the form $x = x_0 + k\frac{b}{g}, y = y_0 + k\frac{a}{g}$

```cpp
// (d, x, y) s.t ax + by = gcd(a, b) = d
tuple<ll, ll, ll> exgcd(ll a, ll b) {
  if(b == 0) return {a, 1, 0};
  auto [d, _x, _y] = exgcd(b, a % b);
  ll x = _y, y = _x - (a / b) * _y;
  return {d, x, y};
}
tuple<bool, ll, ll> diophantine(ll a, ll b, ll c) {
  auto [d, _x, _y] = exgcd(a, b);
  if(c % d) return {false, 0, 0};
  else {
    ll x = (c / d) * _x, y = (c / d) * _y;
    return {true, x, y};
  }
}
void shift_solution(ll &x, ll &y, ll a, ll b, ll cnt) {
  x += cnt * b;
  y -= cnt * a;
}
// returns the number of solutions where x is in the range[minx, maxx] and y is in the range[miny, maxy]
ll find_all_solutions(ll a, ll b, ll c, ll minx, ll maxx, ll miny,ll maxy) {
  ll g = __gcd(a, b);
  auto [res, x, y] = diophantine(a, b, c);
  if (res == false) return 0;
  if (a == 0 and b == 0) {
    assert(c == 0);
    return 1LL * (maxx - minx + 1) * (maxy - miny + 1);
  }
  if (a == 0) {
    return (maxx - minx + 1) * (miny <= c / b and c / b <= maxy);
  }
  if (b == 0) {
    return (maxy - miny + 1) * (minx <= c / a and c / a <= maxx);
  }
  a /= g, b /= g;
  ll sign_a = a > 0 ? +1 : -1;
  ll sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx) shift_solution(x, y, a, b, sign_b);
  if (x > maxx) return 0;
  ll lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx) shift_solution (x, y, a, b, -sign_b);
```

```cpp
  ll rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny) shift_solution (x, y, a, b, -sign_a);
  if (y > maxy) return 0;
  ll lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy) shift_solution(x, y, a, b, sign_a);
  ll rx2 = x;
  if (lx2 > rx2) swap (lx2, rx2);
  ll lx = max(lx1, lx2);
  ll rx = min(rx1, rx2);
  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}
```

5.3 Discrete Log

```cpp
// Returns minimum non-negative  st a^x = b (mod m) or -1 if doesn't exist
int discreteLog(int a, int b, int M) {
  a %= M, b %= M;
  int k = 1, add = 0, g;
  while ((g = gcd(a, M)) > 1) {
    if (b == k) return add;
    if (b % g) return -1;
    b /= g, M /= g, ++add;
    k = (1LL * k * a / g) % M;
  }
  int RT = sqrt(M) + 1, aRT = 1;
  for (int i = 0; i < RT; i++) aRT = (aRT * 1LL * a) % M;
  gp_hash_table<int, int> vals;
  for (int i = 0, cur = b; i <= RT; i++) {
    vals[cur] = i;
    cur = (cur * 1LL * a)%M;
  }
  for (int i = 1, cur = k; i <= M / RT + 1; i++) {
    cur = (cur * 1LL * aRT) % M;
    if (vals.find(cur) != vals.end())
      return RT * i - vals[cur] + add;
  }
  return -1;
}
```

5.4 FWHT

```cpp
using namespace std;
typedef long long ll;
const int N = 1 << 20;
// apply modulo if necessary
void fwht_xor (int *a, int n, int dir = 0) {
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = x + y, a[j + h] = x - y;
        if (dir) a[j] >>= 1, a[j + h] >>= 1;
      }
    }
  }
}
void fwht_or (int *a, int n, int dir = 0) {
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = x, a[j + h] = dir ? y - x : x + y;
      }
    }
  }
}
void fwht_and (int *a, int n, int dir = 0) {
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = dir ? x - y : x + y, a[j + h] = y;
      }
    }
  }
}
int n, a[N], b[N], c[N];
int main() {
  n = 1 << 16;
  for (int i = 0; i < n; ++i) {
    a[i] = rand() & 7;
    b[i] = rand() & 7;
  }
  fwht_xor(a, n), fwht_xor(b, n);
  for (int i = 0; i < n; ++i) {
    c[i] = a[i] * b[i];
  }
  fwht_xor(c, n, 1);
  for (int i = 0; i < n; ++i) {
    cout << c[i] << " ";
  }
  cout << '\n';
  return 0;
}
```

5.5 FastFourierTransform

Description: Use multiplyMod for arbitrary mod multiplication.
Time: $\mathcal{O}(N \log N)$

```cpp
struct cplx {
  ld a, b;
  cplx (ld a = 0, ld b = 0) : a(a), b(b) {}
  const cplx operator + (const cplx &c) const {
    return cplx(a + c.a, b + c.b); }
  const cplx operator - (const cplx &c) const {
    return cplx(a - c.a, b - c.b); }
  const cplx operator * (const cplx &c) const {
    return cplx(a * c.a - b * c.b, a * c.b + b * c.a); }
  const cplx conj() const { return cplx(a, -b); }
};
const ld PI = acosl(-1);
const int MOD = 1e9 + 7;
const int N = (1 << 20) + 5;
int rev[N]; cplx w[N];
void prepare (int n) {
  int sz = __builtin_ctz(n);
  for (int i = 1; i < n; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz - 1));
  w[0] = 0, w[1] = 1, sz = 1;
  while (1 << sz < n) {
    ld ang = 2 * PI / (1 << (sz + 1));
    cplx w_n = cplx(cosl(ang), sinl(ang));
    for (int i = 1 << (sz - 1); i < (1 << sz); ++i) {
      w[i << 1] = w[i], w[i << 1 | 1] = w[i] * w_n;
    } ++sz;
  }
}
void fft (cplx *a, int n) {
  for (int i = 1; i < n - 1; ++i) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int h = 1; h < n; h <<= 1) {
    for (int s = 0; s < n; s += h << 1) {
      for (int i = 0; i < h; ++i) {
        cplx &u = a[s + i], &v = a[s + i + h], t = v * w[h + i];
        v = u - t, u = u + t;
      }
    }
  }
}
static cplx f[N], g[N], u[N], v[N];
void multiply (int *a, int *b, int n, int m) {
  int sz = n + m - 1;
  while (sz & (sz - 1)) sz = (sz | (sz - 1)) + 1;
  prepare(sz);
  for (int i = 0; i < sz; ++i) f[i] = cplx(i < n ? a[i] : 0, i < m ? b[i] : 0);
  fft(f, sz);
  for (int i = 0; i <= (sz >> 1); ++i) {
    int j = (sz - i) & (sz - 1);
    cplx x = (f[i] * f[i] - (f[j] * f[j]).conj()) * cplx(0, -0.25);
    f[j] = x, f[i] = x.conj();
  } fft(f, sz);
  for(int i = 0; i < sz; ++i) a[i] = f[i].a / sz + 0.5;
}
void multiplyMod (int *a, int *b, int n, int m) {
  int sz = 1; while (sz < n + m - 1) sz <<= 1;
  prepare(sz);
  for (int i = 0; i < sz; ++i) {
    f[i] = i < n ? cplx(a[i] & 32767, a[i] >> 15) : cplx(0, 0);
    g[i] = i < m ? cplx(b[i] & 32767, b[i] >> 15) : cplx(0, 0);
  }
  fft(f, sz), fft(g, sz);
  for (int i = 0; i < sz; ++i) {
    int j = (sz - i) & (sz - 1);
    static cplx da, db, dc, dd;
    da = (f[i] + f[j].conj()) * cplx(0.5, 0);
    db = (f[i] - f[j].conj()) * cplx(0, -0.5);
    dc = (g[i] + g[j].conj()) * cplx(0.5, 0);
    dd = (g[i] - g[j].conj()) * cplx(0, -0.5);
    u[j] = da * dc + da * dd * cplx(0, 1);
    v[j] = db * dc + db * dd * cplx(0, 1);
  }
  fft(u, sz), fft(v, sz);
  for(int i = 0; i < sz; ++i) {
    int da = (ll) (u[i].a / sz + 0.5) % MOD;
    int db = (ll) (u[i].b / sz + 0.5) % MOD;
    int dc = (ll) (v[i].a / sz + 0.5) % MOD;
    int dd = (ll) (v[i].b / sz + 0.5) % MOD;
    a[i] = (da + ((ll) (db + dc) << 15) + ((ll) dd << 30)) % MOD;
  }
}
```

5.6 FindRoots

Description: Finds the real roots to a polynomial.
Usage: `polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`
Time: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$

```cpp
struct Poly {
  vector <double> a;
  double operator() (double x) const {
    double val = 0;
    for (int i = a.size(); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    for (int i = 1; i < a.size(); ++i) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=a.size()-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
vector<double> polyRoots(Poly p, double xmin=-1e9, double xmax=1e9) {
  if (p.a.size() == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p; der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(dr.begin(), dr.end());
  for (int i = 0; i < dr.size()-1; ++i) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      for(int it = 0; it < 60; ++it) { // while (h-l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m; else h = m;
```

```
    } ret.push_back((l + h) / 2);
  }
  } return ret;
}
```

5.7 Floor Sum of AP

```
ll sum(ll n){
  return n * (n - 1) >> 1;
}
// sum [(ai + b) / m] for 0 <= i < n
ll floorSumAP (ll a, ll b, ll m, ll n) {
  ll res = a / m * sum(n) + b / m * n;
  a %= m, b %= m; if (!a) return res;
  ll to = (n * a + b) / m;
  return res + (n - 1) * to - floorSumAP(m, m - 1 - b, a, to);
}
// sum (a + di) % m for 0 <= i < n
ll modSumAP (ll a, ll b, ll m, ll n) {
  b = ((b % m) + m) % m, a = ((a % m) + m) % m;
  return n * b + a * sum(n) - m * floorSumAP(a, b, m, n);
}
```

5.8 Gaussian Elimination

```
using namespace std;
typedef long long ll;
typedef long double ld;
const int N = 505;
const ld EPS = 1e-10;
const int MOD = 998244353;
ll bigMod (ll a, ll e, ll mod) {
  if (e == -1) e = mod - 2;
  ll ret = 1;
  while (e) {
    if (e & 1) ret = ret * a % mod;
    a = a * a % mod, e >>= 1;
  }
  return ret;
}
pair <int, ld> gaussJordan (int n, int m, ld eq[N][N], ld res[N]) {
  ld det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (fabs(eq[k][j]) > fabs(eq[piv][j])) piv = k;
    if (fabs(eq[piv][j]) < EPS) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = -det; det *= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i) {
      ld x = eq[k][j] / eq[i][j];
      for (int l = j; l <= m; ++l) eq[k][l] -= x * eq[i][l];
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[pos[i]][m] / eq[pos[i]][i];
  }
  for (int i = 0; i < n; ++i) {
    ld cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += eq[i][j] * res[j];
    if (fabs(cur) > EPS) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
pair <int, int> gaussJordanModulo (int n, int m, int eq[N][N], int res[N], int mod) {
  int det = 1;
  vector <int> pos(m, -1);
  const ll mod_sq = (ll) mod * mod;
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j] > eq[piv][j]) piv = k;
    if (!eq[piv][j]) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = det ? MOD - det : 0; det = (ll) det * eq[i][j] % MOD;
    for (int k = 0; k < n; ++k) if (k ^ i and eq[k][j]) {
      ll x = eq[k][j] * bigMod(eq[i][j], -1, mod) % mod;
      for (int l = j; l <= m; ++l) if (eq[i][l]) eq[k][l] = (eq[k][l] + mod_sq - x * eq[i][l]) % mod;
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[pos[i]][m] * bigMod(eq[pos[i]][i], -1, mod) % mod;
  }
  for (int i = 0; i < n; ++i) {
    ll cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += (ll) eq[i][j] * res[j], cur %= mod;
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
pair <int, int> gaussJordanBit (int n, int m, bitset <N> eq[N], bitset <N> &res) {
  int det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j]) {
      piv = k; break;
    }
    if (!eq[piv][j]) continue; pos[j] = i, swap(eq[piv], eq[i]), det &= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i and eq[k][j]) eq[k] ^= eq[i]; ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[pos[i]][m];
  }
  for (int i = 0; i < n; ++i) {
    int cur = eq[i][m];
    for (int j = 0; j < m; ++j) cur ^= eq[i][j] & res[j];
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
```

5.9 Integrate Adaptive

Description: Fast integration using an adaptive Simpson's rule.
Usage: `double sphereVolume = quad(-1, 1, [](double x) { return quad(-1, 1, [&](double y) { return quad(-1, 1, [&](double z) { return x*x + y*y + z*z < 1; });});});`

```
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
template <class F>
d rec(F& f, d a, d b, d eps, d S) {
 d c = (a + b) / 2;
 d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
 if (abs(T - S) <= 15 * eps || b - a < 1e-10)
  return T + (T - S) / 15;
 return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
 return rec(f, a, b, eps, S(a, b));
}
```

5.10 Inverse Exp Log

Description: Computes P^{-1}, $\exp(P)$, $\log(P)$ fast. $P^k = \exp(k \log P)$. Log is in-place. Value of n should be power of 2 larger than polynomial size.
Time: $\mathcal{O}(n \log n)$

```
void prepare (int n);
void ntt (int *a, int n, int dir = 0);
// TAKE THE ABOVE PARTS FROM NumberTheoreticTransform.h
void multiply (int *a, int *b, int n) {
  prepare(n << 1), ntt(a, n << 1), ntt(b, n << 1);
  for (int i = 0; i < n << 1; ++i) a[i] = (ll) a[i] * b[i] % MOD;
  ntt(a, n << 1, 1); for (int i = n; i < n << 1; ++i) a[i] = 0;
}
static int f[N], g[N], h[N]; int inv[N]; // 1/i modulo MOD
void inverse (int *a, int n, int *b) {
  b[0] = inv[a[0]], b[1] = 0;
  for (int m = 2; m <= n; m <<= 1) {
    for (int i = 0; i < m; ++i) f[i] = a[i], f[i + m] = b[i + m] = 0;
    prepare(m << 1), ntt(f, m << 1), ntt(b, m << 1);
    for (int i = 0; i < m << 1; ++i) b[i] = (ll) b[i] * (MOD + 2 - (ll) b[i] * f[i] % MOD) % MOD;
    ntt(b, m << 1, 1); for (int i = m; i < m << 1; ++i) b[i] = 0;
  }
}
void log (int *a, int n) {
  inverse(a, n, g);
  for (int i = 0; i + 1 < n; ++i) a[i] = (i + 1LL) * a[i + 1] % MOD;
  multiply(a, g, n);
  for (int i = n - 1; i; --i) a[i] = (ll) a[i - 1] * inv[i] % MOD;
  a[0] = 0;
}
void exp (int *a, int n, int *b) {
  b[0] = 1, b[1] = 0;
  for (int m = 2; m <= n; m <<= 1) {
    for (int i = 0; i < m; ++i) h[i] = b[i];
    log(h, m);
    for (int i = 0; i < m; ++i) h[i] = (a[i] - h[i] + MOD) % MOD;
    ++h[0], h[0] %= MOD;
    for (int i = m; i < m << 1; ++i) b[i] = h[i] = 0;
    multiply(b, h, m); for (int i = m; i < m << 1; ++i) b[i] = 0;
  }
}
```

5.11 Lagrange Interpolation

Description: A polynomial of degree d can be uniquely identified given its values on d + 1 unique points. O(n) to pre-calculate given the first n points (x=0 to n-1) of the polynomial. Then answer each query to interpolate the x'th term in O(n). All values are done modulo mod, which needs to be a prime as we need its inverse modulo. Also includes an additional helper function called find_degree(terms, mod). Given at least the first d+2 points of a polynomial of degree d, it finds d in roughly O(n log d). Note, n should not exceed mod due to the way modular inverse is used. In such cases, we can use interpolation without modulo in big integers and take the remainder later
Time: $\mathcal{O}(n)$ *

```
using namespace std;
struct Lagrange{
  vector<int> terms, dp;
  int mod, n;
  Lagrange() {}
  Lagrange(const vector<int>& terms, int mod) : terms(terms), mod(mod){
    n = terms.size();
```

```
    assert(n <= mod);
    int i, v, f;
    for (f = 1, i = 1; i < n; i++) f = (long long)f * i %
     ↪ mod;
    v = expo(f, mod - 2);
    vector<int> inv(n, v);
    for (i = n - 1; i > 0; i--){
      inv[i - 1] = (long long)inv[i] * i % mod;
    }
    dp.resize(n, 1);
    for (i = 0; i < n; i++){
      dp[i] = (long long)inv[i] * inv[n - i - 1] % mod;
      dp[i] = (long long)dp[i] * terms[i] % mod;
    }
  }
  int expo(int a, int b){
    int res = 1;
    while (b){
      if (b & 1) res = (long long)res * a % mod;
      a = (long long)a * a % mod;
      b >>= 1;
    }
    return res;
  }
  int interpolate(long long x){
    if (x < n) return terms[x] % mod;
    x %= mod;
    int i, w;
    vector<int> X(n, 1), Y(n, 1);
    for (i = 1; i < n; i++){
      X[i] = (long long)X[i - 1] * (x - i + 1) % mod;
      if (X[i] < 0) X[i] += mod;
    }
    for (i = n - 2; i >= 0; i--){
      Y[i] = (long long)Y[i + 1] * (x - i - 1) % mod;
      if (Y[i] < 0) Y[i] += mod;
    }
    long long res = 0;
    for (i = 0; i < n; i++){
      w = ((long long)X[i] * Y[i] % mod) * dp[i] % mod;
      if ((n - i + 1) & 1) w = mod - w;
      res += w;
    }
    return res % mod;
  }
};
vector<int> get_terms(const vector<int>& terms, int mod,
 ↪ int l, int r){
  auto lagrange = Lagrange(terms, mod);
  vector<int> res;
  for (int i = l; i <= r; i++){
    res.push_back(lagrange.interpolate(i));
  }
  return res;
}
int find_degree(const vector<int>& terms, int mod){
  long long v = mod;
  int k = 1, n = terms.size();
  while (v < INT_MAX){
    v *= mod;
    k++;
  }
  int l = 1 << 30, r = l + k - 1;
  auto expected = get_terms(terms, mod, l, r);
  int low = 1, high = n - 1;
  while ((low + 1) < high){
    int mid = (low + high) >> 1;
    vector<int> v(terms.begin(), terms.begin() + mid);
    if (get_terms(v, mod, l, r) == expected) high = mid;
    else low = mid;
  }
  for (int d = low; d <= high; d++){
    vector<int> v(terms.begin(), terms.begin() + d);
    if (get_terms(v, mod, l, r) == expected) return d - 1;
  }
  return -1;
}
int main(){
  const int mod = 1000000007;
  vector <int> terms = vector<int> {0, 1, 5, 14, 30};
  auto lagrange = Lagrange(terms, mod);
  assert(lagrange.interpolate(5) == 55);
  assert(lagrange.interpolate(6) == 91);
  assert(lagrange.interpolate(1 << 30) == 300663155);
  assert(lagrange.interpolate(1LL << 60) == 717860166);
  assert(find_degree(terms, mod) == 3);
  terms.pop_back();
  assert(find_degree(terms, mod) == -1);
  return 0;
}
```

5.12 Linear Recurrence

Description: Get k-th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0\ldots \geq n-1]$ and $tr[0\ldots n-1]$.

Usage: `linearRec({0, 1}, {1, 1}, k) // k-th Fibonacci number`

Time: $\mathcal{O}\left(n^2 \log k\right)$

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = tr.size();
  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    for (int i = 0; i <= n; ++i)
      for (int j = 0; j <= n; ++j)
        res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i)
      for (int j = 0; j < n; ++j)
        res[i-1-j] = (res[i-1-j] + res[i] * tr[j]) % mod;
    res.resize(n + 1); return res;
  };
  Poly pol(n + 1), e(pol); pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }
  ll res = 0;
  for (int i = 0; i < n; ++i)
    res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

5.13 Matrix Inverse

Description: Inverts matrix A, stores in A. Returns rank.

Time: $\mathcal{O}\left(n^3\right)$

```
int matInv(vector<vector<ll>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<ll>> tmp(n, vector<ll>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;
  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n) if (A[j][k]) {
      r = j; c = k; goto found;
    }
    return i;
  found:
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n) swap(A[j][i], A[j][c]), swap(tmp[j][i],
     ↪ tmp[j][c]);
    swap(col[i], col[c]);
    ll v = bigMod(A[i][i], -1);
    rep(j,i+1,n) {
      ll f = A[j][i] * v % mod;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
      rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) %
       ↪ mod;
    }
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
  }
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    ll v = A[j][i];
    rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod;
  }
  rep(i,0,n) rep(j,0,n)
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0 ?
     ↪ mod : 0);
  return n;
}
```

5.14 Miller Rabin

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.

Time: 7 times the complexity of $a^b \mod c$.

```
bool isPrime(ull n) {
 if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
 ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
  ↪ 1795265022},
     s = __builtin_ctzll(n-1), d = n >> s;
 for (ull a : A) {   // ^ count trailing zeroes
  ull p = modpow(a%n, d, n), i = s;
  while (p != 1 && p != n - 1 && a % n && i--)
   p = modmul(p, p, n);
  if (p != n-1 && i != s) return 0;
 }
 return 1;
}
```

5.15 ModLog

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod{m}$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.

Time: $\mathcal{O}\left(\sqrt{m}\right)$

```
ll modLog(ll a, ll b, ll m) {
 ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
 unordered_map<ll, ll> A;
 while (j <= n && (e = f = e * a % m) != b % m)
  A[e * b % m] = j++;
 if (e == b % m) return j;
 if (__gcd(m, e) == __gcd(m, b))
  rep(i,2,n+2) if (A.count(e = e * f % m))
   return n * i - A[e];
 return -1;
}
```

5.16 ModMulLL

Description: Calculate $a \cdot b \bmod c$ for $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$.

Time: $\mathcal{O}(1)$ for `modmul`, $\mathcal{O}(\log b)$ for `modpow`

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
 ll ret = a * b - M * ull(1.L / M * a * b);
 return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
 ull ans = 1;
 for (; e; b = modmul(b, b, mod), e /= 2)
 if (e & 1) ans = modmul(ans, b, mod);
 return ans;
}
```

5.17 ModSqrt

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).

Time: $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p

```
ll sqrt(ll a, ll p) {
 a %= p; if (a < 0) a += p;
 if (a == 0) return 0;
 assert(bigMod(a, (p-1)/2, p) == 1); // else no solution
 if (p % 4 == 3) return bigMod(a, (p+1)/4, p);
 // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
 ll s = p - 1, n = 2;
 int r = 0, m;
 while (s % 2 == 0)
  ++r, s /= 2;
 while (bigMod(n, (p - 1) / 2, p) != p - 1) ++n;
 ll x = bigMod(a, (s + 1) / 2, p);
 ll b = bigMod(a, s, p), g = bigMod(n, s, p);
 for (;; r = m) {
  ll t = b;
  for (m = 0; m < r && t != 1; ++m) t = t * t % p;
  if (m == 0) return x;
  ll gs = bigMod(g, 1LL << (r - m - 1), p);
  g = gs * gs % p; x = x * gs % p; b = b * g % p;
 }
}
```

5.18 NumberTheoreticTransform

Description: Modulo M should be of form $c \cdot 2^n + 1$ where 2^n is bigger than maximum polynomial size under computation (or as a result). G should be a primitive root of M. To find G, factor $M-1$ and get the distinct primes p_i. If $G^{(M-1)/p_i} \neq 1 \pmod M$ for each p_i then G is a valid root. Try all G until a hit is found (usually found very quick).

Time: $\mathcal{O}(N \log N)$

```
const int G = 3;
const int MOD = 998244353;
const int N = (1 << 20) + 5;
int rev[N], w[N], inv_n;
// G = primitive_root(MOD)
int primitive_root (int p) {
  vector <int> factor;
  int tmp = p - 1;
  for (int i = 2; i * i <= tmp; ++i) {
    if (tmp % i == 0) {
      factor.emplace_back(i);
      while (tmp % i == 0) tmp /= i;
    }
  }
  if (tmp != 1) factor.emplace_back(tmp);
  for (int root = 1; ; ++root) {
    bool flag = true;
    for (int i = 0; i < (int) factor.size(); ++i) {
      if (bigMod(root, (p - 1) / factor[i], p) == 1) {
        flag = false; break;
      }
    }
    if (flag) return root;
  }
}
void prepare (int n) {
  int sz = abs(31 - __builtin_clz(n));
  int r = bigMod(G, (MOD - 1) / n, MOD);
  inv_n = bigMod(n, MOD - 2, MOD), w[0] = w[n] = 1;
  for (int i = 1; i < n; ++i) w[i] = (ll) w[i - 1] * r %
   MOD;
  for (int i = 1; i < n; ++i) rev[i] = (rev[i >> 1] >> 1) |
   ((i & 1) << (sz - 1));
}
void ntt (int *a, int n, int dir) {
  for (int i = 1; i < n - 1; ++i) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int m = 2; m <= n; m <<= 1) {
    for (int i = 0; i < n; i += m) {
      for (int j = 0; j < (m >> 1); ++j) {
        int &u = a[i + j], &v = a[i + j + (m >> 1)];
        int t = (ll) v * w[dir ? n - n / m * j : n / m * j]
         % MOD;
        v = u - t < 0 ? u - t + MOD : u - t;
        u = u + t >= MOD ? u + t - MOD : u + t;
      }
    }
  } if (dir) for (int i = 0; i < n; ++i) a[i] = (ll) a[i] *
   inv_n % MOD;
}
int f_a[N], f_b[N];
vector <int> multiply (vector <int> a, vector <int> b) {
  int sz = 1, n = a.size(), m = b.size();
  while (sz < n + m - 1) sz <<= 1; prepare(sz);
  for (int i = 0; i < sz; ++i) f_a[i] = i < n ? a[i] : 0;
  for (int i = 0; i < sz; ++i) f_b[i] = i < m ? b[i] : 0;
  ntt(f_a, sz, 0); ntt(f_b, sz, 0);
  for (int i = 0; i < sz; ++i) f_a[i] = (ll) f_a[i] *
   f_b[i] % MOD;
  ntt(f_a, sz, 1); return vector <int> (f_a, f_a + n + m -
   1);
}
```

5.19 Online Convolution

```
struct OnlineConvolution {
  vector<int> a, b, c;
  int k;
  OnlineConvolution(int n): a(n), b(n), c(n), k(0) {}
  // poly c = poly a * poly b
  // add a[i] = x and b[i] = y and it will return c[i]
  int extend(int i, int x, int y) {
   assert(i == k);
    a[k] = x; b[k] = y;
    int s = k + 2;
    for (int w = 1; s % w == 0 && w < s; w <<= 1) {
      for (int ri = 0; ri < 2; ri++) {
        if (ri == 0 || w * 2 != s) {
          vector<int> f(w), g(w);
          for (int i = 0; i < w; i++) f[i] = a[w - 1 + i],
           g[i] = b[k - w + 1 + i];
          f = multiply(f, g);
          for (int i = 0, j = k; i < f.size() && j <
           c.size(); i++, j++) {
            c[j] += f[i];
            if (c[j] >= MOD) c[j] -= MOD;
          }
        }
        swap(a, b);
      }
    }
    return c[k++];
  }
};
int main() {
  int n = 5;
  OnlineConvolution oc(n);
  vector<int> f(n + 1), g(n);
  for(int i = 0; i < n; i++) g[i] = n - i;
  // f[0] = 1
  // f[i] = f[j] * g[k] s.t j + k = i - 1
  f[0] = 1;
  for(int i = 1; i <= n; i++) {
    oc.extend(i - 1, f[i - 1], g[i - 1]);
    f[i] = oc.c[i - 1];
  }
}
```

5.20 Pollard Rho

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

```
typedef long long ll;
typedef unsigned long long ull;
namespace Rho {
  ull mul (ull a, ull b, ull mod) {
    ll ret = a * b - mod * (ull) (1.L / mod * a * b);
    return ret + mod * (ret < 0) - mod * (ret >= (ll) mod);
  }
  ull bigMod (ull a, ull e, ull mod) {
    ull ret = 1;
    while (e) {
      if (e & 1) ret = mul(ret, a, mod);
      a = mul(a, a, mod), e >>= 1;
    }
    return ret;
  }
  bool isPrime (ull n) {
    if (n < 2 or n % 6 % 4 != 1) return (n | 1) == 3;
    ull a[] = {2, 325, 9375, 28178, 450775, 9780504,
     1795265022};
    ull s = __builtin_ctzll(n - 1), d = n >> s;
    for (ull x : a) {
      ull p = bigMod(x % n, d, n), i = s;
      while (p != 1 and p != n - 1 and x % n and i--) p =
       mul(p, p, n);
      if (p != n - 1 and i != s) return 0;
    }
    return 1;
  }
  ull pollard (ull n) {
    auto f = [&] (ull x) {return mul(x, x, n) + 1;};
    ull x = 0, y = 0, t = 0, prod = 2, i = 1, q;
    while (t++ % 40 or __gcd(prod, n) == 1) {
      if (x == y) x = ++i, y = f(x);
      if ((q = mul(prod, max(x, y) - min(x, y), n))) prod =
       q;
      x = f(x), y = f(f(y));
    }
    return __gcd(prod, n);
  }
  vector <ull> factor (ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), r.begin(), r.end());
    return l;
  }
};
```

5.21 Power Sum

```
ll ncr[N][N], S[N][N], B[N], d[N][N];
ll fact[N], inv[N];
ll powerSum(ll n, ll k) {
  ll ret = 0LL;
  for (int i = 0; i <= k + 1; ++i) {
    ret += d[k][i] * bigMod(n, i);
    ret %= MOD;
  }
  return ret;
}
void init() {
  fact[0] = 1;
  for (int i = 1; i < N; ++i) { fact[i] = (fact[i - 1] * i)
   % MOD; }
  inv[1] = 1;
  for (int i = 2; i < N; ++i) {
    inv[i] = MOD - (MOD / i) * inv[MOD % i] % MOD;
  }
  S[0][0] = 1;
  for (int i = 1; i < N; ++i) {
    S[i][0] = 0;
    for (int j = 1; j <= i; ++j) {
      S[i][j] = j * S[i - 1][j] + S[i - 1][j - 1];
      S[i][j] %= MOD;
    }
```

```cpp
  }
  for (int i = 1; i < N; ++i) {
    ncr[i][0] = ncr[i][i] = 1;
    for (int j = 1; j < i; ++j) {
      ncr[i][j] = ncr[i - 1][j] + ncr[i - 1][j - 1];
      ncr[i][j] %= MOD;
    }
  }
  for (int i = 0; i < 15; ++i) {
    for (int j = 0; j <= i; ++j) {
      B[i] += ((j & 1) ? -1 : 1) * ((fact[j] * S[i][j]) % MOD)
              * inv[j + 1];
      B[i] %= MOD;
    }
  }
  for (int i = 0; i < 15; ++i) {
    for (int j = 0; j <= i; ++j) {
      d[i][i + 1 - j] = (ncr[i + 1][j] * abs(B[j])) % MOD;
      d[i][i + 1 - j] *= inv[i + 1];
      d[i][i + 1 - j] %= MOD;
    }
  }
  d[0][0] = 1; // 0^0 = 1
}
```

5.22 Simplex

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: `vvd A = {{1,-1}, {-1,1}, {-1,-2}};`
`vd b = {1,1,-4}, c = {-1,-1}, x;`
`T val = LPSolver(A, b, c).solve(x);`
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.

```cpp
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
struct LPSolver {
  int m, n; vi N, B; vvd D;
  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }
  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv; swap(B[r], N[s]);
  }
  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false; pivot(r, s);
    }
  }
  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0; rep(j,1,n+1) ltj(D[i]); pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

5.23 Stirling

Description:

- Number of permutations of n elements with k disjoint cycles = Str1(n,k) = (n-1) * Str1(n-1,k) + Str1(n-1,k-1).
- n! = Sum(Str1(n,k)) (for all 0 <= k <= n).
- Ways to partition n labelled objects into k unlabelled subsets = Str2(n,k) = k * Str2(n-1,k) + Str2(n-1,k-1).
- Parity of Str2(n,k) : ((n-k) Floor((k-1)/2)) == 0). Ways to partition n labelled objects into k unlabelled subsets, with each subset containing at least r elements: SR(n,k) = k * SR(n-1,k) + C(n-1,r-1) * SR(n-r,k-1).
- Number of ways to partition n labelled objects 1,2,3, ... n into k non-empty subsets so that for any integers i and j in a given subset |i-j| >= d: Str2(n-d+1, k-d+1), n >= k >= d.

```cpp
NTT ntt(mod);
vector<ll> v[MAX];
//Stirling1 (n,k) = co-eff of x^k in x*(x+1)*(x+2)*....(x+n-1)
int Stirling1(int n, int r) {
  int nn = 1;
  while (nn < n) nn <<= 1;
  for (int i = 0; i < n; ++i) {v[i].push_back(i); v[i].push_back(1);}
  for (int i = n; i < nn; ++i) v[i].push_back(1);
  for (int j = nn; j > 1; j >>= 1) {
    int hn = j >> 1;
    for (int i = 0; i < hn; ++i) ntt.multiply(v[i], v[i + hn], v[i]);
  }
  return v[0][r];
}
NTT ntt(mod);
vector<ll> a, b, res;
//Stirling2 (n,k) = co-eff of x^k in product of polynomials A & B
//where A(i) = (-1)^i / i! and B(i) = i^n / i!
int Stirling2(int n, int r) {
  a.resize(n + 1); b.resize(n + 1);
  for (int i = 0; i <= n; i++) {
    a[i] = invfct[i];
    if (i % 2 == 1) a[i] = mod - a[i];
  }
  for (int i = 0; i <= n; i++) {
    b[i] = bigMod(i, n, mod);
    b[i] = (b[i] * invfct[i]) % mod;
  }
  NTT ntt(mod);
  ntt.multiply(a, b, res);
  return res[r];
}
```

5.24 Sum of Totient Function

```cpp
using namespace __gnu_pbds;
const int N = 3e5 + 9, mod = 998244353;
template <const int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other) const {
      int32_t c = this->value + other.value; return modint<MOD>(c >= MOD ? c - MOD : c); }
  inline modint<MOD> operator - (modint<MOD> other) const {
      int32_t c = this->value - other.value; return modint<MOD>(c <    0 ? c + MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other) const {
      int32_t c = (int64_t)this->value * other.value % MOD; return modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> & operator += (modint<MOD> other) {
      this->value += other.value; if (this->value >= MOD) this->value -= MOD; return *this; }
  inline modint<MOD> & operator -= (modint<MOD> other) {
      this->value -= other.value; if (this->value <    0) this->value += MOD; return *this; }
  inline modint<MOD> & operator *= (modint<MOD> other) {
      this->value = (int64_t)this->value * other.value % MOD; if (this->value < 0) this->value += MOD; return *this; }
  inline modint<MOD> operator - () const { return modint<MOD>(this->value ? MOD - this->value : 0); }
  modint<MOD> pow(uint64_t k) const { modint<MOD> x = *this, y = 1; for (; k; k >>= 1) { if (k & 1) y *= x; x *= x; } return y; }
  modint<MOD> inv() const { return pow(MOD - 2); }  // MOD must be a prime
  inline modint<MOD> operator /  (modint<MOD> other) const { return *this *  other.inv(); }
  inline modint<MOD> operator /= (modint<MOD> other) { return *this *= other.inv(); }
  inline bool operator == (modint<MOD> other) const { return value == other.value; }
  inline bool operator != (modint<MOD> other) const { return value != other.value; }
  inline bool operator < (modint<MOD> other) const { return value < other.value; }
  inline bool operator > (modint<MOD> other) const { return value > other.value; }
};
template <int32_t MOD> modint<MOD> operator * (int64_t value, modint<MOD> n) { return modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD> operator * (int32_t value, modint<MOD> n) { return modint<MOD>(value % MOD) * n; }
template <int32_t MOD> istream & operator >> (istream & in, modint<MOD> &n) { return in >> n.value; }
template <int32_t MOD> ostream & operator << (ostream & out, modint<MOD> n) { return out << n.value; }
using mint = modint<mod>;
namespace Dirichlet {
  //solution for f(x) = phi(x)
  const int T = 1e7 + 9;
  long long phi[T];
  gp_hash_table<long long, mint> mp;
  mint dp[T], inv;
  int sz, spf[T], prime[T];
  void init() {
    memset(spf, 0, sizeof spf);
    phi[1] = 1; sz = 0;
```

```
    for (int i = 2; i < T; i++) {
      if (spf[i] == 0) phi[i] = i - 1, spf[i] = i,
        prime[sz++] = i;
      for (int j = 0; j < sz && i * prime[j] < T &&
        prime[j] <= spf[i]; j++) {
        spf[i * prime[j]] = prime[j];
        if (i % prime[j] == 0) phi[i * prime[j]] = phi[i] *
          prime[j];
        else phi[i * prime[j]] = phi[i] * (prime[j] - 1);
      }
    }
    dp[0] = 0;
    for(int i = 1; i < T; i++) dp[i] = dp[i - 1] + phi[i] %
      mod;
    inv = 1; // g(1)
  }
  mint p_c(long long n) {
    if (n % 2 == 0) return n / 2 % mod * ((n + 1) % mod) %
      mod;
    return (n + 1) / 2 % mod * (n % mod) % mod;
  }
  mint p_g(long long n) {
    return n % mod;
  }
  mint solve (long long x) {
    if (x < T) return dp[x];
    if (mp.find(x) != mp.end()) return mp[x];
    mint ans = 0;
    for (long long i = 2, last; i <= x; i = last + 1) {
      last = x / (x / i);
      ans += solve (x / i) * (p_g(last) - p_g(i - 1));
    }
    ans = p_c(x) - ans;
    ans /= inv;
    return mp[x] = ans;
  }
};
```

5.25 Totient

```
int phi(int n) {
  int result = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0)
        n /= i;
      result -= result / i;
    }
  }
  if (n > 1)
    result -= result / n;
  return result;
}
void phi_1_to_n(int n) {
  vector<int> phi(n + 1);
  for (int i = 0; i <= n; i++)
    phi[i] = i;
  for (int i = 2; i <= n; i++)
    if (phi[i] == i) {
      for (int j = i; j <= n; j += i)
        phi[j] -= phi[j] / i;
    }
  }
}
```

5.26 Tridiagonal

Description: $x = \text{tridiagonal}(d,p,q,b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i, 1 \le i \le n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1,-1,-1,...,-1,1\},\{0,c_1,c_2,\dots,c_n\},$$
$$\{b_1,b_2,\dots,b_n,0\},\{a_0,d_1,d_2,\dots,d_n,a_{n+1}\}).$$

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>&
  super,
  const vector<T>& sub, vector<T> b) {
 int n = sz(b); vi tr(n);
 rep(i,0,n-1) {
  if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
   b[i+1] -= b[i] * diag[i+1] / super[i];
   if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
   diag[i+1] = sub[i]; tr[++i] = 1;
  } else {
   diag[i+1] -= super[i]*sub[i]/diag[i];
   b[i+1] -= b[i]*sub[i]/diag[i];
  }
 }
 for (int i = n; i--;) {
  if (tr[i]) {
   swap(b[i], b[i-1]); diag[i-1] = diag[i]; b[i] /=
     super[i-1];
  } else {
   b[i] /= diag[i]; if (i) b[i-1] -= b[i]*super[i-1];
  }
 } return b;
}
```

5.27 Xor Basis

```
void tryGauss(int mask) {
 for (int i = 0; i < n; i++) {
  if ((mask & 1 << i) == 0) continue;
  if (!basis[i]) {
   basis[i] = mask;
   ++sz;
   break;
  }
  mask ^= basis[i];
 }
}
```

6 Misc

6.1 Debug

```
template<typename A>
string to_string(A v) {
  bool first = true;
  string res = "{";
  for (const auto &x : v) {
    if (!first) res += ", ";
    first = false;
    res += to_string(x);
  }
  res += "}";
  return res;
}
void debug_out() { cerr << endl; }
template<typename Head, typename ... Tail>
void debug_out(Head H, Tail... T) {
  cerr << " " << to_string(H) << " |";
  debug_out(T ...);
}
#define debug(...) clog << "Line : " << __LINE__ << " : ["
  <<#__VA_ARGS__ << "]:", debug_out(__VA_ARGS__)
```

6.2 LIS

```
const int inf = 1e9;
vector<int> LIS(vector<int> a, int n){
  vector<int> d(n + 1, inf);
  for (int i = 0; i < n; i++) {
   *lower_bound(d.begin(), d.end(), a[i]) = a[i];
  }
  d.resize(lower_bound(d.begin(), d.end(), inf) -
    d.begin());
  return d;
}
```

6.3 Misc

```
// Pragmas
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("O3,unroll-loops,Ofast,fast-math")
#pragma GCC target("avx,avx2,fma")
// Custom Priority Queue
std::priority_queue<int, std::vector<int>,
  std::greater<int>> Q;   // increasing
//gp hash table https://codeforces.com/blog/entry/60737
using namespace __gnu_pbds;
const int RANDOM = chrono::high_resolution_clock::now().tim
  e_since_epoch().count();
struct chash {
  int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<key, int, chash> table;
//bitset
BS._Find_first()
BS._Find_next(x) //Return first set bit after xth bit, x on
  failure
//Gray Code, G(0) = 000, G(1) = 001, G(2) = 011, G(3) = 010
inline int g(int n) { return n ^ (n >> 1); }
//Inverse Gray Code
int rev_g(int g) {
  int n = 0;
  for ( ; g; g >>= 1) n ^= g;
  return n;
}
// Only for non-negative integers
// Returns the immediate next number with same count of one
  bits, -1 on failure
long long hakmemItem175(long long n) {
  if (!n) return -1;
  long long x = (n & -n);
  long long left = (x + n);
  long long right = ((n ^ left) / x) >> 2;
  long long res = (left | right);
  return res;
}
// Returns the immediate previous number with same count of
  one bits, -1 on failure
long long lol(long long n) {
  if (n < 2) return -1;
  long long res = ~hakmemItem175(~n);
  return (!res) ? -1 : res;
}
int __builtin_clz(int x);// number of leading zero
int __builtin_ctz(int x);// number of trailing zero
int __builtin_clzll(long long x);// number of leading zero
int __builtin_ctzll(long long x);// number of trailing zero
int __builtin_popcount(int x);// number of 1-bits in x
int __builtin_popcountll(long long x);// number of 1-bits
  in x
lsb(n): (n & -n); // last bit (smallest)
floor(log2(n)): 31 - __builtin_clz(n | 1);
floor(log2(n)): 63 - __builtin_clzll(n | 1);
// compute next perm. ex) 00111, 01011, 01101, 01110,
  10011, 10101..
long long next_perm(long long v){
  long long t = v | (v-1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) +
    1));
```

```
}
```

6.4 Random

```
// mt19937_64 in case of 64 bit
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
shuffle(v.begin(), v.end(), rng);
ll rnd(ll l, ll r) {
  return uniform_int_distribution<ll>(l, r) (rng);
}
```

6.5 Stress Testing

```
set -e
g++ code.cpp -o code
g++ gen.cpp -o gen
g++ brute.cpp -o brute
for((i = 1; ; ++i)); do
    ./gen $i > input_file
    ./code < input_file > myAnswer
    ./brute < input_file > correctAnswer
    echo "Passed test: "  $i
done
```

7 Strings

7.1 Aho Corasick

Description: Use add_pattern to add a bunch of strings, call compute (REMEMBER), then query/match text.
Time: linear

```
struct AC {
  int N, P; const int A = 26;
  vector <vector <int>> next;
  vector <int> link, out_link;
  vector <vector <int>> out;
  AC(): N(0), P(0) {node();}
  int node() {
    next.emplace_back(A, 0);
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++;
  }
  inline int get (char c) { return c - 'a'; }
  int add_pattern (const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][get(c)]) next[u][get(c)] = node();
      u = next[u][get(c)];
    } out[u].push_back(P); return P++;
  }
  void compute() {
    queue <int> q;
    for (q.push(0); !q.empty();) {
      int u = q.front(); q.pop();
      for (int c = 0; c < A; ++c) {
        int v = next[u][c];
        if (!v) next[u][c] = next[link[u]][c];
        else {
          link[v] = u ? next[link[u]][c] : 0;
          out_link[v] = out[link[v]].empty() ?
            out_link[link[v]] : link[v];
          q.push(v);
        }
      }
    }
  }
  int advance (int u, char c) {
    while (u and !next[u][get(c)]) u = link[u];
    u = next[u][get(c)]; return u;
  }
  void match (const string S) {
    int u = 0; for (auto c : S) {
      u = advance(u, c);
      for (int v = u; v; v = out_link[v]) {
        for (auto p : out[v]) cout << "match " << p << endl;
      }
    }
  }
};
```

7.2 Hash

```
const int N = 1e6 + 5;
const int MOD = int(1e9) + 7;
const ll P[] = {97, 1000003};
ll pwr[2][N], inv[2][N];
void initHash() {
  for (int it = 0; it < 2; ++it) {
    pwr[it][0] = inv[it][0] = 1;
    ll INV_P = bigMod(P[it], -1);
    for (int i = 1; i < N; ++i) {
      pwr[it][i] = pwr[it][i - 1] * P[it] % MOD;
      inv[it][i] = inv[it][i - 1] * INV_P % MOD;
    }
  }
}
struct RangeHash {
  vector<ll> h[2], rev[2];
  RangeHash(const string S, bool revFlag = 0) {
    for (int it = 0; it < 2; ++it) {
      h[it].resize(S.size() + 1, 0);
      for (int i = 0; i < S.size(); ++i) {
        h[it][i + 1] = (h[it][i] + pwr[it][i + 1] * (S[i] -
          'a' + 1)) % MOD;
      }
      if (revFlag) {
        rev[it].resize(S.size() + 1, 0);
        for (int i = 0; i < S.size(); ++i) {
          rev[it][i + 1] = (rev[it][i] + inv[it][i + 1] *
            (S[i] - 'a' + 1)) % MOD;
        }
      }
    }
  }
  inline ll get(int l, int r) {
    ll one = (h[0][r + 1] - h[0][l]) * inv[0][l + 1] % MOD;
    ll two = (h[1][r + 1] - h[1][l]) * inv[1][l + 1] % MOD;
    if (one < 0) one += MOD; if (two < 0) two += MOD;
    return one << 31 | two;
  }
  inline ll getReverse(int l, int r) {
    ll one = (rev[0][r + 1] - rev[0][l]) * pwr[0][r + 1] %
      MOD;
    ll two = (rev[1][r + 1] - rev[1][l]) * pwr[1][r + 1] %
      MOD;
    if (one < 0) one += MOD; if (two < 0) two += MOD;
    return one << 31 | two;
  }
};
int main() {
  initHash();
  string S; cin >> S;
  RangeHash machine(S);
  cout << machine.get(0, S.size() - 1) << '\n';
  return 0;
}
```

7.3 KMP

```
template<typename T>
vector<int> prefix_function(const T &s) {
  int n = (int)s.size();
  vector<int> p(n, 0);
  int k = 0;
  for (int i = 1; i < n; i++) {
    while (k > 0 && !(s[i] == s[k])) {
      k = p[k - 1];
    }
    if (s[i] == s[k]) {
      k++;
    }
    p[i] = k;
  }
  return p;
}
// Returns 0-indexed positions of occurrences of s in w
template<typename T>
vector<int> kmp_search(const T &s, const T &w) {
  int n = (int)s.size();
  int m = (int)w.size();
  const vector<int> p = prefix_function(s);
  assert(n >= 1 && (int) p.size() == n);
  vector<int> res;
  int k = 0;
  for (int i = 0; i < m; i++) {
    while (k > 0 && (k == n || !(w[i] == s[k]))) { k = p[k
      - 1]; }
    if (w[i] == s[k]) { k++; }
    if (k == n) { res.push_back(i - n + 1); }
  }
  return res;
}
template<typename T>
vector<vector<int>> prefix_function_automaton(const T &s,
  int alphabet_size, int smallest_alphabet) {
  int n = s.size();
  vector<int> pf = prefix_function(s);
  vector<vector<int>> automaton(n + 1,
    vector<int>(alphabet_size));
  for (int i = 0; i <= n; i++) {
    for (int c = 0; c < alphabet_size; c++) {
      if (i < n and s[i] == smallest_alphabet + c) {
        automaton[i][c] = i + 1;
      }
      else {
        automaton[i][c] = i == 0 ? 0 : automaton[pf[i -
          1]][c];
      }
    }
  }
  return automaton;
}
```

7.4 Manacher

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).
Time: $\mathcal{O}(N)$

```
typedef vector<int> vi;
array<vi, 2> manacher(const string& s) {
  int n = s.length();
  array<vi, 2> p = { vi(n + 1), vi(n)};
  for (int z = 0; z < 2; z++) for (int i = 0, l = 0, r = 0;
    i < n; i++) {
    int t = r - i + !z;
    if (i < r) p[z][i] = min(t, p[z][l + t]);
    int L = i - p[z][i], R = i + p[z][i] - !z;
    while (L >= 1 && R + 1 < n && s[L - 1] == s[R + 1])
      p[z][i]++, L--, R++;
    if (R > r) l = L, r = R;
  }
  return p;
}
```

7.5 MinRotation

Description: Finds the lexicographically smallest rotation of a string.
Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
Time: $\mathcal{O}(N)$

```
int minRotation(string s) {
  int a = 0, N = sz(s); s += s;
  rep(b, 0, N) rep(k, 0, N) {
    if (a + k == b || s[a + k] < s[b + k]) {b += max(0, k -
      1); break;}
    if (s[a + k] > s[b + k]) { a = b; break; }
  } return a;
}
```

7.6 Palindromic Tree

```
class PalindromicTree {
private:
  int A;
  string s;
  int last, ptr;
  vector<int> link, len, occ, depth;
  vector<vector<int>> nxt;
  void init(int sz) {
    link.resize(sz, 0), len.resize(sz, 0), occ.resize(sz, 0), depth.resize(sz, 0);
    nxt.resize(sz, vector<int>(A, 0));
    len[1] = -1, len[2] = 0, link[1] = link[2] = 1, last = ptr = 2;
  }
  void feed(int at) {
    while (s[at - len[last] - 1] != s[at]) last = link[last];
    int ch = s[at] - 'a', temp = link[last];
    while (s[at - len[temp] - 1] != s[at]) temp = link[temp];
    if (!nxt[last][ch]) {
      nxt[last][ch] = ++ptr, len[ptr] = len[last] + 2;
      link[ptr] = len[ptr] == 1 ? 2 : nxt[temp][ch];
      depth[ptr] = depth[link[ptr]] + 1;
      palindromes.emplace_back(at - len[ptr], at);
    }
    last = nxt[last][ch], ++occ[last];
  }
public:
  vector<pair<int, int>> palindromes;
  PalindromicTree(string s, int A = 26) {
    int n = s.length();
    this->s = '0' + s;
    this->A = A;
    init(n + 3);
    for (int i = 1; i <= n; ++i) feed(i);
  }
};
```

7.7 Suffix Array

```
// Everything is 0-indexed
char s[N]; // Suffix array will be built for this string
int SA[N], iSA[N]; // SA is the suffix array, iSA[i] stores the rank of the i'th suffix
int cnt[N], nxt[N]; // Internal stuff
bool bh[N], b2h[N]; // Internal stuff
int lcp[N]; // Stores lcp of SA[i] and SA[i + 1]; lcp[n - 1] = 0
int lcpSparse[LOGN][N]; // lcpSparse[i][j] = min(lcp[j], ..., lcp[j - 1 + (1 << i)])
void buildSA(int n) {
  for (int i = 0; i < n; i++) SA[i] = i;
  sort(SA, SA + n, [](int i, int j) { return s[i] < s[j]; });
  for (int i = 0; i < n; i++) {
    bh[i] = i == 0 || s[SA[i]] != s[SA[i - 1]];
    b2h[i] = 0;
  }
  for (int h = 1; h < n; h <<= 1) {
    int tot = 0;
    for (int i = 0, j; i < n; i = j) {
      j = i + 1;
      while (j < n && !bh[j]) j++;
      nxt[i] = j; tot++;
    } if (tot == n) break;
    for (int i = 0; i < n; i = nxt[i]) {
      for (int j = i; j < nxt[i]; j++) iSA[SA[j]] = i;
      cnt[i] = 0;
    }
    cnt[iSA[n - h]]++;
    b2h[iSA[n - h]] = 1;
    for (int i = 0; i < n; i = nxt[i]) {
      for (int j = i; j < nxt[i]; j++) {
        int s = SA[j] - h;
        if (s < 0) continue;
        int head = iSA[s];
        iSA[s] = head + cnt[head]++;
        b2h[iSA[s]] = 1;
      }
      for (int j = i; j < nxt[i]; j++) {
        int s = SA[j] - h;
        if (s < 0 || !b2h[iSA[s]]) continue;
        for (int k = iSA[s] + 1; !bh[k] && b2h[k]; k++) b2h[k] = 0;
      }
    }
    for (int i = 0; i < n; i++) {
      SA[iSA[i]] = i;
      bh[i] |= b2h[i];
    }
  }
  for (int i = 0; i < n; i++) iSA[SA[i]] = i;
}
void buildLCP(int n) {
  for (int i = 0, k = 0; i < n; i++) {
    if (iSA[i] == n - 1) {
      k = 0;
      lcp[n - 1] = 0;
      continue;
    }
    int j = SA[iSA[i] + 1];
    while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
    lcp[iSA[i]] = k;
    if (k) k--;
  }
}
void buildLCPSparse(int n) {
  for (int i = 0; i < n; i++) lcpSparse[0][i] = lcp[i];
  for (int i = 1; i < LOGN; i++) {
    for (int j = 0; j < n; j++) {
      lcpSparse[i][j] = min(lcpSparse[i - 1][j], lcpSparse[i - 1][min(n - 1, j + (1 << (i - 1)))]);
    }
  }
}
pair<int, int> minLCPRange(int n, int from, int minLCP) {
  int r = from;
  for (int i = LOGN - 1; i >= 0; i--) {
    int jump = 1 << i;
    if (r + jump < n and lcpSparse[i][r] >= minLCP) r += jump;
  }
  int l = from;
  for (int i = LOGN - 1; i >= 0; i--) {
    int jump = 1 << i;
    if (l - jump >= 0 and lcpSparse[i][l - jump] >= minLCP) l -= jump;
  }
  return make_pair(l, r);
}
```

7.8 SuffixAutomaton

Description: Pattern matching – call run function. Do DP on DAG for number of different substrings (= different paths) or total length of different substrings. Lexicographically kth substring is kth path from root. Min rotation is smallest $|S|$ length path on automaton of $S+S$. Number of occurrence for a node – mark non-clone nodes with 1 then do cnt[link[u]]+ = cnt[u]. Occurrence position – maintain firstPos (maybe lastPos too?) for each endpos set. Shortest non-appearing string – DP on DAG, DP[u] = 1 if there's no transition with a character, $1+\min \mathrm{DP}[v]$ otherwise. Longest common substring of two strings S, T – construct automaton S, run through T, and climb up the suffix links of automaton until a transition is found through next character of T. Do similar stuff for multiple strings.
Time: $\mathcal{O}(n \log n)$, linear if array used

```
char s[N], p[N];
map <char, int> to[N << 1]; // use array maybe?
int len[N << 1], link[N << 1], sz, last;
inline void init() {
  len[0] = 0, link[0] = -1, sz = 1, last = 0, to[0].clear();
}
void feed (char c) {
  int cur = sz++, p = last;
  len[cur] = len[last] + 1, link[cur] = 0, to[cur].clear();
  while (~p and !to[p].count(c)) to[p][c] = cur, p = link[p];
  if (~p) {
    int q = to[p][c];
    if (len[q] - len[p] - 1) {
      int r = sz++;
      len[r] = len[p] + 1, to[r] = to[q], link[r] = link[q];
      while (~p and to[p][c] == q) to[p][c] = r, p = link[p];
      link[q] = link[cur] = r;
    } else link[cur] = q;
  } last = cur;
}
bool run() {
  int m = strlen(p);
  for (int i = 0, u = 0; i < m; ++i) {
    if (!to[u].count(p[i])) return 0;
    u = to[u][p[i]];
  } return 1;
}
int main() {
  init();
  for (int i = 0; i < n; ++i) feed(s[i]);
  run();
}
```

7.9 Z Algorithm

```
template<typename T>
vector<int> z_function(const T &s) {
  int n = (int)s.size();
  vector<int> z(n, n);
  int l = 0, r = 0;
  for (int i = 1; i < n; i++) {
    z[i] = (i > r ? 0 : min(r - i + 1, z[i - l]));
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
      z[i]++;
    }
    if (i + z[i] - 1 > r) {
      l = i;
      r = i + z[i] - 1;
    }
  }
  return z;
}
```

8 Notes

8.1 Equations

$$\begin{array}{l} ax+by=e \\ cx+dy=f \end{array} \Rightarrow \begin{array}{l} x=\dfrac{ed-bf}{ad-bc} \\ y=\dfrac{af-ec}{ad-bc} \end{array}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

8.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k + c_1 x^{k-1} + \cdots + c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

8.3 Trigonometry

$$(V+W)\tan(v-w)/2=(V-W)\tan(v+w)/2$$

where V,W are lengths of sides opposite angles v,w.

$$a\cos x+b\sin x=r\cos(x-\phi)$$

$$a\sin x+b\cos x=r\sin(x+\phi)$$

where $r=\sqrt{a^2+b^2}, \phi=\operatorname{atan2}(b,a)$.

8.4 Geometry

8.4.1 Triangles

Circumradius: $R=\frac{abc}{4A}$, Inradius: $r=\frac{A}{s}$

Length of median (divides triangle into two equal-area triangles): $m_a=\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$

Length of bisector (divides angles in two): $s_a=\sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$

Law of tangents: $\frac{a+b}{a-b}=\frac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

8.4.2 Quadrilaterals

With side lengths a,b,c,d, diagonals e,f, diagonals angle θ, area A and magic flux $F=b^2+d^2-a^2-c^2$:

$$4A=2ef\cdot\sin\theta=F\tan\theta=\sqrt{4e^2f^2-F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef=ac+bd$, and $A=\sqrt{(p-a)(p-b)(p-c)(p-d)}$.

8.4.3 Spherical coordinates

$$x=r\sin\theta\cos\phi \qquad r=\sqrt{x^2+y^2+z^2}$$

$$y=r\sin\theta\sin\phi \qquad \theta=\operatorname{acos}(z/\sqrt{x^2+y^2+z^2})$$

$$z=r\cos\theta \qquad \phi=\operatorname{atan2}(y,x)$$

8.4.4 Pick's Theorem.

Area of lattice polygon $A=i+b/2-1$, where i is the number of lattice points strictly inside, and b is the number of lattice points on boundary (including vertices).

8.5 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x=\frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x=-\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\tan x=1+\tan^2 x \qquad \frac{d}{dx}\arctan x=\frac{1}{1+x^2}$$

$$\int\tan ax=-\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax=\frac{\sin ax-ax\cos ax}{a^2}$$

$$\int e^{-x^2}=\frac{\sqrt{\pi}}{2}\operatorname{erf}(x) \qquad \int xe^{ax}dx=\frac{e^{ax}}{a^2}(ax-1)$$

8.6 Sums

$1^2+2^2+3^2+\cdots+n^2=\frac{n(2n+1)(n+1)}{6}$

$1^3+2^3+3^3+\cdots+n^3=\frac{n^2(n+1)^2}{4}$

$1^4+2^4+3^4+\cdots+n^4=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$

$\sum_{i=1}^{n} i^m=\frac{1}{m+1}\left[(n+1)^{m+1}-1-\sum_{i=1}^{n}\left((i+1)^{m+1}-i^{m+1}-(m+1)i^m\right)\right]$

$\sum_{i=1}^{n-1} i^m=\frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m+1-k}$

$\sum_{k=0}^{n} kx^k=(x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$

8.7 Series

$$e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots,(-\infty<x<\infty)$$

$$\ln(1+x)=x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots,(-1<x\le 1)$$

$$\sqrt{1+x}=1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\ldots,(-1\le x\le 1)$$

$$\sin x=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots,(-\infty<x<\infty)$$

$$\cos x=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots,(-\infty<x<\infty)$$

$$(x+a)^{-n}=\sum_{k=0}^{\infty}(-1)^k\binom{n+k-1}{k}x^k a^{-n-k}$$

8.8 Number Theory

8.8.1 Primes

$p=962592769$ is such that $2^{21}\mid p-1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p=2,a>2$, and there are $\phi(\phi(p^a))$ many. For $p=2,a>2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2\times\mathbb{Z}_{2^{a-2}}$.

8.8.2 Estimates

$\sum_{d|n} d=O(n\log\log n)$.

The number of divisors of n is at most around 100 for $n<5e4$, 500 for $n<1e7$, 2000 for $n<1e10$, 200 000 for $n<1e19$.

8.8.3 Carmichael numbers

A positive composite n is a Carmichael number ($a^{n-1}\equiv 1 \pmod n$ for all a: $\gcd(a,n)=1$), iff n is square-free, and for all prime divisors p of n, $p-1$ divides $n-1$.

8.8.4 Totient Function

- $\phi(n)=n\times\prod\left(1-\frac{1}{p}\right)$.
- $\phi(p)=p-1$. if $\gcd(m,n)=1, \phi(mn)=\phi(m)\times\phi(n)$.
- Sum of $m<=n$ s.t, $\gcd(m,n)=1$: $n\times\frac{\phi(n)}{2}$
- For a and b, $\phi(ab)=\phi(a)\phi(b)\frac{d}{\phi(d)}$
- $\phi(ip)=p\phi(i)$ whenever p is a prime and it divides i
- $\sum_{i=1}^{n}\gcd(i,n)=\sum_{d|n} d\phi(\frac{n}{d})$
- $\sum_{i=1} n\operatorname{lcm}(i,n)=\frac{n}{2}(\sum_{d|n}(d\phi(d))+1)$
- $a^{\phi(m)}\equiv 1 \pmod m$, if $\gcd(a,m)=1$

8.8.5 Mobius function

$\mu(1)=1$. $\mu(n)=0$, if n is not squarefree. $\mu(n)=(-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. If for all $n\in N$, $F(n)=\sum_{d|n}f(d)$, then $f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$, and vice versa. $\phi(n)=\sum_{d|n}\mu(d)\frac{n}{d}$. $\sum_{d|n}\mu(d)=1$.

If f is multiplicative, then $\sum_{d|n}\mu(d)f(d)=\prod_{p|n}(1-f(p))$, $\sum_{d|n}\mu(d)^2f(d)=\prod_{p|n}(1+f(p))$.

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)=1]=\sum_{k=1}^{n}\mu(k)\lfloor\frac{n}{k}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}gcd(i,j)=\sum_{k=1}^{n}k\sum_{l=1}^{\lfloor\frac{n}{k}\rfloor}\mu(l)\lfloor\frac{n}{kl}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}gcd(i,j)=\sum_{k=1}^{n}\left(\frac{\lfloor\frac{n}{k}\rfloor(1+\lfloor\frac{n}{k}\rfloor)}{2}\right)^2\sum_{d|k}\mu(d)kd$$

8.8.6 Quadratic Residue.

$(\frac{a}{p})$ is 0 if $p\mid a$, 1 if a is a quadratic residue, -1 otherwise. Euler: $(\frac{a}{p})=a^{(p-1)/2} \pmod p$ (prime). Jacobi: if $n=p_1^{e_1}\cdots p_k^{e_k}$ then $(\frac{a}{n})=\prod(\frac{a}{p_i})^{e_i}$.

8.8.7 Legendre symbol

If p is an odd prime, $a\in\mathbb{Z}$, then $\left(\frac{a}{p}\right)$ equals 0, if $p|a$; 1 if a is a quadratic residue modulo p; and -1 otherwise. Euler's criterion: $\left(\frac{a}{p}\right)=a^{\left(\frac{p-1}{2}\right)} \pmod p$.

8.8.8 Jacobi symbol

If $n=p_1^{a_1}\cdots p_k^{a_k}$ is odd, then $\left(\frac{a}{n}\right)=\prod_{i=1}^{k}\left(\frac{a}{p_i}\right)^{k_i}$.

8.8.9 Kronecker symbol

Let a be a positive integer, which is not a perfect square and $a\equiv 0$ or $1 \pmod 4$. $\left(\frac{a}{2}\right)=\{1$, if $a\equiv 1 \pmod 8$; -1, if $a\equiv 5 \pmod 8\}$.

$\left(\frac{a}{n}\right)=\prod_{j=1}^{k}p_j^{k_j}$, if $\gcd(a,n)\neq 1$ and $n=\prod p_i^{k_i}$. $\left(\frac{a}{n}\right)$ equals Jacobi symbol otherwise.

8.8.10 Primitive roots

If the order of g modulo m (min $n>0$: $g^n\equiv 1 \pmod m$) is $\phi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i=\operatorname{ind}_g(a)$ modulo $\phi(m)$, such that $g^i\equiv a \pmod m$. $\operatorname{ind}_g(a)$ has logarithm-like properties: $\operatorname{ind}(1)=0$, $\operatorname{ind}(ab)=\operatorname{ind}(a)+\operatorname{ind}(b)$.

If p is prime and a is not divisible by p, then congruence $x^n\equiv a \pmod p$ has $\gcd(n,p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)}\equiv 1 \pmod p$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i\equiv a \pmod p$, $g^u\equiv x \pmod p$. $x^n\equiv a \pmod p$ iff $g^{nu}\equiv g^i \pmod p$ iff $nu\equiv i \pmod p$.)

8.8.11 Discrete logarithm problem

Find x from $a^x\equiv b \pmod m$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n=\lceil\sqrt{m}\rceil$, and $x=ny-z$. Equation becomes $a^{ny}\equiv ba^z \pmod m$. Precompute all values that the RHS can take for $z=0,1,\ldots,n-1$, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

8.8.12 Postage stamps/McNuggets problem

Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax+by$ $(x,y\ge 0)$, and the largest is $(a-1)(b-1)-1=ab-a-b$.

8.8.13 Fermat's two-squares theorem

Odd prime p can be represented as a sum of two squares iff $p\equiv 1 \pmod 4$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p=4k+3$ occurs an even number of times in n's factorization.

8.8.14 Bezout's Theorem.

For $a\neq$, $b\neq 0$, $d=\gcd(a,b)$ is the smallest positive integer for which there are integer solutions to $ax+by=d$. If (x,y) is one solution, then all solutions are given by $(x+kb/d, y-ka/d), k\in\mathbb{Z}$. Find one solution with extended Euclidean algorithm.

8.9 Combinatorics

8.9.1 Biomial Coefficient

Hockey Stick Identity:

- (Left to right) Sum over n and k: $\sum_{k=0}^{m}\binom{n+k}{k}=\binom{n+m-1}{m}$
- (Right to left) Sum over n: $\sum_{m=0}^{n}\binom{m}{k}=\binom{n+1}{k+1}$

Sum of the squares: $\sum_{k=0}^{n}(\binom{n}{k})^2=\binom{2n}{n}$

Weighted sum: $\sum_{k=1}^{n}k\binom{n}{k}=n2^{n-1}$

Connection with the fibonacci numbers: $\sum_{k=0}\binom{n-k}{k}=F_{n+1}$

Parity: $\binom{n}{r}$ is odd if r is a submask of n ($n\&r=r$).

8.9.2 Fibonacci Numbers

$$\sum_{i=0}^{n}F_i^2=F_{n+1}F_n$$

$$\sum_{i=0}^{n}F_iF_{i+1}=F_{n+1}^2-(-1)^n$$

$$\sum_{i=0}^{n}F_iF_{i-1}=\sum_{i=0}^{n-1}F_iF_{i+1}$$

$$\gcd(F_m,F_n)=F_{\gcd(m,n)}$$

$$\gcd(F_n,F_{n+1})=\gcd(F_n,F_{n+2})=\gcd(F_{n+1},F_{n+2})=1$$

8.9.3 2nd Kaplansky's Lemma

The number of ways of selecting k objects, no two consecutive,from n labelled objects arrayed in a circle is $\frac{n}{k}\binom{n-k-1}{k-1}=\frac{n}{n-k}\binom{n-k}{k}$

8.9.4 Balls in Bins

- n distinct balls in r distinct bins (empty bins): r^n
- n distinct balls in r distinct bins (no empty bins): $r^n - \binom{r}{1}(r-1)^n + \binom{r}{2}(r-2)^n \ldots + (-1)^{n-1}\binom{r}{r}$
- n distinct balls in r identical bins (empty bins): Bell Number
- n distinct balls in r identical bins (no empty bins): Stirling Number 2
- n identical balls in r distinct bins (empty bins): $\binom{n+r-1}{r-1}$
- n identical balls in r distinct bins (empty bins): $\binom{n-1}{r-1}$

8.9.5 Ballot Problem

Suppose that in an election, candidate A receives a votes and candidate B receives b votes, where a kb for some positive integer k. Compute the number of ways the ballots can be ordered so that A maintains more than k times as many votes as B throughout the counting of the ballots : $\frac{a-kb}{a+b} \times \binom{a+b}{a}$

8.9.6 Coloring

- The number of labeled undirected graphs with n vertices, $G_n = 2^{\binom{n}{2}}$
- The number of labeled directed graphs with n vertices, $G_n = 2^{n(n-1)}$
- The number of connected labeled undirected graphs with n vertices, $C_n = 2^{\binom{n}{2}} - \frac{1}{n}\sum_{k=1}^{n-1} k\binom{n}{k} 2^{\binom{n-k}{2}} C_k = 2^{\binom{n}{2}} - \sum_{k=1}^{n-1}\binom{n-1}{k-1} 2^{\binom{n-k}{2}} C_k$
- The number of k-connected labeled undirected graphs with n vertices, $D[n][k] = \sum_{s=1}^{n}\binom{n-1}{s-1} C_s D[n-s][k-1]$
- **Cayley's formula:** the number of trees on n labeled vertices = the number of spanning trees of a complete graph with n labeled vertices = n^{n-2}
- Number of ways to color a graph using k color such that no two adjacent nodes have same color: **Complete graph** = $k(k-1)(k-2)\ldots(k-n+1)$, **Tree** = $k(k-1)^{n-1}$, **Cycle** = $(k-1)^n + (-1)^n(k-1)$
- Number of trees with n labeled nodes: n^{n-2}

8.9.7 Matrix Tree Theorem.

Create $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]-`, `mat[b][b]++` (and `mat[b][a]-`, `mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

8.9.8 Erdős–Gallai theorem.

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$, $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$.

8.9.9 Selected cycle lengths.

If $g_S(n)$ is number of n-permutations with cycle lengths from set S then $\sum_{n\geq 0} \frac{g_S(n)}{n!} x^n = \exp(\sum_{n\in S} \frac{x^n}{n})$.

8.9.10 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

8.9.11 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n\in S}\frac{x^n}{n}\right)$$

8.9.12 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

8.9.13 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k)$$

8.9.14 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k\in\mathbb{Z}\setminus\{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

8.9.15 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$

$$\textstyle\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

8.9.16 Stirling (second kind).

Ways to partition n items into k non-empty groups. $S(n,k) = S(n-1,k-1) + kS(n-1,k), S(n,1) = S(n,n) = 1$. Row generator:

$$e^{-x}\sum_{k\geq 0}\frac{k^n}{k!}x^k,\quad S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

8.9.17 Bell number.

Number of partitions of n distinct elements.

$$B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$$

For prime p, $B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$. $B(n) = \sum_{k\geq 0}\binom{n-1}{k}B(k) = \sum_{k\geq 0} S(n,k)$.

8.9.18 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

8.9.19 Bernoulli numbers

$\sum_{j=0}^{m}\binom{m+1}{j}B_j = 0$. $B_0 = 1, B_1 = -\frac{1}{2}$. $B_n = 0$, for all odd $n \neq 1$.

8.9.20 Lucas' Theorem.

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \ldots + n_1 p + n_0$ and $m = m_k p^k + \ldots + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod{p}$.

8.9.21 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

8.9.22 Catalan Convolution

$$C_n^{(k)} = \sum_{a_0+a_1+\ldots+a_k=n} C_{a_0}C_{a_1}\ldots C_{a_k}, n \geq 0; C_0 = 1$$

$$C_n^{(k)} = \frac{k+1}{n+k+1}\binom{2n+k}{n}$$

8.9.23 Labeled unrooted trees.

n^{n-2} trees on n vertices. $n_1 n_2 \cdots n_k n^{k-2}$ trees on k existing trees of size n_i. $(n-2)!/[(d_1-1)!\cdots(d_n-1)!]$ trees with degree sequence d_i.

8.10 Trivia

8.10.1 Pythagorean (coprime) triples.

$[a,b,c] = [m^2-n^2, 2mn, m^2+n^2]$ with $m > n > 0$ and m,n coprime, both not odd. Other triples are these scaled by $k > 0$. $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

8.10.2 Primes less than 10^6.

78498.

8.10.3 nth root congruence.

If p is prime and p doesn't divide a, then congruence $x^n \equiv a \pmod{p}$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod{p}$, and no solutions otherwise. There are $\phi(\phi(n))$ primitive roots for $n = 2, 4, p^k, 2p^k$, odd prime p.

8.10.4 Max number of divisors $\leq n$

. 100 for $n = 50{,}000$. 500 for $n = 10^7$. 2000 for $n = 10^{10}$. 2,00,000 for $n = 10^{19}$.

8.10.5 Two Square Theorem.

Odd prime p is a sum of two squares iff $p \equiv 1 \pmod{4}$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k+3$ occurs an even number of times in n's factorization.

8.10.6 Perfect Number.

Number that equals sum of its proper divisors. n is perfect iff $n = 2^{p-1}(2^p-1)$ and $2^p - 1$ is a (Mersenne) prime.

8.11 Inequalities

8.11.1 Titu's Lemma

For positive reals $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$,

$$\frac{{a_1}^2}{b_1} + \frac{{a_2}^2}{b_2} + \ldots + \frac{{a_n}^2}{b_n} \geq \frac{a_1 + a_2 + \ldots + {a_n}^2}{b_1 + b_2 + \ldots + b_n}$$

Equality holds if and only if $a_i = kb_i$ for a non-zero real constant k.

8.12 Games

8.12.1 Grundy numbers

For a two-player, normal-play (last to move wins) game on a graph (V,E): $G(x) = \text{mex}(\{G(y) : (x,y) \in E\})$, where $\text{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

8.12.2 Sums of games

- *Player chooses a game and makes a move in it* Grundy number of a position is xor of grundy numbers of positions in summed games.
- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them* A position is losing iff each game is in a losing position.
- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.
- *Player must move in all games, and loses if can't move in some game* A position is losing if any of the games is in a losing position.

8.12.3 Misère Nim

Misere nim is evaluated like normal nim if all piles are not of size 1. Otherwise depends on parity of number of piles. Sum of some normal impartial games:

- Pick non-empty subset of games and move in each: losing iff every game is losing.
- Pick non-empty proper subset of games: losing grundy numbers of all games are equal.