Columbia University: CU Later Team Reference Document

Kevin Yang, Innokentiy Kaurov, Eric Yuang Shao

May 21th 2024

Contents

Templates

Ken's template

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  #define all(v) (v).begin(), (v).end()
4  typedef long long ll;
5  typedef long double ld;
6  #define pb push_back
7  #define sz(x) (int)(x).size()
8  #define fi first
9  #define se second
10 #define endl '\n'
```

Kevin's template

```
1  // paste Kaurov's Template, minus last line
2  typedef vector<int> vi;
3  typedef vector<ll> vll;
4  typedef pair<int, int> pii;
5  typedef pair<ll, ll> pll;
6  const char nl = '\n';
7  #define forn(i, n) for (int i = 0; i < int(n); i++)
8  ll k, n, m, u, v, w, x, y, z;
9  string s, t;
10
11 bool multiTest = 1;
12 void solve(int tt){
13 }
14
15 int main(){
16   ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
17   cout<<fixed<< setprecision(14);
18
19   int t = 1;
20   if (multiTest) cin >> t;
21   forn(ii, t) solve(ii);
22 }
```

Kevin's Template Extended

- to type after the start of the contest

```
1  typedef pair<double, double> pdd;
2  const ld PI = acosl(-1);
3  const ll mod7 = 1e9 + 7;
4  const ll mod9 = 998244353;
5  const ll INF = 2*1024*1024*1023;
6  #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
7  #include <ext/pb_ds/assoc_container.hpp>
8  #include <ext/pb_ds/tree_policy.hpp>
9  using namespace __gnu_pbds;
10 template<class T> using ordered_set = tree<T, null_type,
   ↪  less<T>, rb_tree_tag,
   ↪  tree_order_statistics_node_update>;
11 vi d4x = {1, 0, -1, 0};
12 vi d4y = {0, 1, 0, -1};
13 vi d8x = {1, 0, -1, 0, 1, 1, -1, -1};
14 vi d8y = {0, 1, 0, -1, 1, -1, 1, -1};
15 mt19937
   ↪  rng(chrono::steady_clock::now().time_since_epoch().count());
```

Geometry

- Basic stuff

```
1  template<typename T>
2  struct TPoint{
3    T x, y;
4    int id;
5    static constexpr T eps = static_cast<T>(1e-9);
6    TPoint() : x(0), y(0), id(-1) {}
7    TPoint(const T& x_, const T& y_) : x(x_), y(y_),
   ↪  id(-1) {}
8    TPoint(const T& x_, const T& y_, const int id_) :
   ↪  x(x_), y(y_), id(id_) {}
9
10   TPoint operator + (const TPoint& rhs) const {
11     return TPoint(x + rhs.x, y + rhs.y);
12   }
13   TPoint operator - (const TPoint& rhs) const {
14     return TPoint(x - rhs.x, y - rhs.y);
15   }
16   TPoint operator * (const T& rhs) const {
17     return TPoint(x * rhs, y * rhs);
18   }
19   TPoint operator / (const T& rhs) const {
20     return TPoint(x / rhs, y / rhs);
21   }
22   TPoint ort() const {
23     return TPoint(-y, x);
24   }
25   T abs2() const {
26     return x * x + y * y;
27   }
28   T len() const {
29     return sqrtl(abs2());
30   }
31   TPoint unit() const {
32     return TPoint(x, y) / len();
33   }
34 };
35 template<typename T>
36 bool operator< (TPoint<T>& A, TPoint<T>& B){
37   return make_pair(A.x, A.y) < make_pair(B.x, B.y);
38 }
39 template<typename T>
40 bool operator== (TPoint<T>& A, TPoint<T>& B){
41   return abs(A.x - B.x) <= TPoint<T>::eps && abs(A.y -
   ↪  B.y) <= TPoint<T>::eps;
42 }
43 template<typename T>
44 struct TLine{
45   T a, b, c;
46   TLine() : a(0), b(0), c(0) {}
47   TLine(const T& a_, const T& b_, const T& c_) : a(a_),
   ↪  b(b_), c(c_) {}
48   TLine(const TPoint<T>& p1, const TPoint<T>& p2){
49     a = p1.y - p2.y;
50     b = p2.x - p1.x;
51     c = -a * p1.x - b * p1.y;
52   }
53 };
54 template<typename T>
55 T det(const T& a11, const T& a12, const T& a21, const T&
   ↪  a22){
56   return a11 * a22 - a12 * a21;
57 }
58 template<typename T>
59 T sq(const T& a){
60   return a * a;
61 }
62 template<typename T>
63 T smul(const TPoint<T>& a, const TPoint<T>& b){
64   return a.x * b.x + a.y * b.y;
65 }
66 template<typename T>
67 T vmul(const TPoint<T>& a, const TPoint<T>& b){
68   return det(a.x, a.y, b.x, b.y);
69 }
70 template<typename T>
71 bool parallel(const TLine<T>& l1, const TLine<T>& l2){
72   return abs(vmul(TPoint<T>(l1.a, l1.b), TPoint<T>(l2.a,
   ↪  l2.b))) <= TPoint<T>::eps;
73 }
74 template<typename T>
75 bool equivalent(const TLine<T>& l1, const TLine<T>& l2){
76   return parallel(l1, l2) &&
77   abs(det(l1.b, l1.c, l2.b, l2.c)) <= TPoint<T>::eps &&
78   abs(det(l1.a, l1.c, l2.a, l2.c)) <= TPoint<T>::eps;
79 }
```

- Intersection

```
1  template<typename T>
2  TPoint<T> intersection(const TLine<T>& l1, const
   ↪  TLine<T>& l2){
3    return TPoint<T>(
4      det(-l1.c, l1.b, -l2.c, l2.b) / det(l1.a, l1.b,
   ↪  l2.a, l2.b),
5      det(l1.a, -l1.c, l2.a, -l2.c) / det(l1.a, l1.b,
   ↪  l2.a, l2.b)
6    );
7  }
8  template<typename T>
9  int sign(const T& x){
10   if (abs(x) <= TPoint<T>::eps) return 0;
11   return x > 0? +1 : -1;
12 }
```

- Area

```
template<typename T>
T area(const vector<TPoint<T>>& pts){
  int n = sz(pts);
  T ans = 0;
  for (int i = 0; i < n; i++){
    ans += vmul(pts[i], pts[(i + 1) % n]);
  }
  return abs(ans) / 2;
}
template<typename T>
T dist_pp(const TPoint<T>& a, const TPoint<T>& b){
  return sqrt(sq(a.x - b.x) + sq(a.y - b.y));
}
template<typename T>
TLine<T> perp_line(const TLine<T>& l, const TPoint<T>& p){
  T na = -l.b, nb = l.a, nc = - na * p.x - nb * p.y;
  return TLine<T>(na, nb, nc);
}
```

• Projection

```
template<typename T>
TPoint<T> projection(const TPoint<T>& p, const TLine<T>& l){
  return intersection(l, perp_line(l, p));
}
template<typename T>
T dist_pl(const TPoint<T>& p, const TLine<T>& l){
  return dist_pp(p, projection(p, l));
}
template<typename T>
struct TRay{
  TLine<T> l;
  TPoint<T> start, dirvec;
  TRay() : l(), start(), dirvec() {}
  TRay(const TPoint<T>& p1, const TPoint<T>& p2){
    l = TLine<T>(p1, p2);
    start = p1, dirvec = p2 - p1;
  }
};
template<typename T>
bool is_on_line(const TPoint<T>& p, const TLine<T>& l){
  return abs(l.a * p.x + l.b * p.y + l.c) <=
  ↪  TPoint<T>::eps;
}
template<typename T>
bool is_on_ray(const TPoint<T>& p, const TRay<T>& r){
  if (is_on_line(p, r.l)){
    return sign(smul(r.dirvec, TPoint<T>(p - r.start)))
  ↪  != -1;
  }
  else return false;
}
template<typename T>
bool is_on_seg(const TPoint<T>& P, const TPoint<T>& A,
  ↪  const TPoint<T>& B){
  return is_on_ray(P, TRay<T>(A, B)) && is_on_ray(P,
  ↪  TRay<T>(B, A));
}
```

```
}
template<typename T>
T dist_pr(const TPoint<T>& P, const TRay<T>& R){
  auto H = projection(P, R.l);
  return is_on_ray(H, R)? dist_pp(P, H) : dist_pp(P,
  ↪  R.start);
}
template<typename T>
T dist_ps(const TPoint<T>& P, const TPoint<T>& A, const
  ↪  TPoint<T>& B){
  auto H = projection(P, TLine<T>(A, B));
  if (is_on_seg(H, A, B)) return dist_pp(P, H);
  else return min(dist_pp(P, A), dist_pp(P, B));
}
```

• acw

```
template<typename T>
bool acw(const TPoint<T>& A, const TPoint<T>& B){
  T mul = vmul(A, B);
  return mul > 0 || abs(mul) <= TPoint<T>::eps;
}
```

• cw

```
template<typename T>
bool cw(const TPoint<T>& A, const TPoint<T>& B){
  T mul = vmul(A, B);
  return mul < 0 || abs(mul) <= TPoint<T>::eps;
}
```

• Convex Hull

```
template<typename T>
vector<TPoint<T>> convex_hull(vector<TPoint<T>> pts){
  sort(all(pts));
  pts.erase(unique(all(pts)), pts.end());
  vector<TPoint<T>> up, down;
  for (auto p : pts){
    while (sz(up) > 1 && acw(up.end()[-1] -
  ↪  up.end()[-2], p - up.end()[-2])) up.pop_back();
    while (sz(down) > 1 && cw(down.end()[-1] -
  ↪  down.end()[-2], p - down.end()[-2]))
  ↪  down.pop_back();
    up.pb(p), down.pb(p);
  }
  for (int i = sz(up) - 2; i >= 1; i--) down.pb(up[i]);
  return down;
}
```

• in_triangle

```
template<typename T>
bool in_triangle(TPoint<T>& P, TPoint<T>& A, TPoint<T>&
  ↪  B, TPoint<T>& C){
  if (is_on_seg(P, A, B) || is_on_seg(P, B, C) ||
  ↪  is_on_seg(P, C, A)) return true;
  return cw(P - A, B - A) == cw(P - B, C - B) &&
  cw(P - A, B - A) == cw(P - C, A - C);
}
```

• prep_convex_poly

```
template<typename T>
void prep_convex_poly(vector<TPoint<T>>& pts){
  rotate(pts.begin(), min_element(all(pts)), pts.end());
}
```

• in_convex_poly:

```
// 0 - Outside, 1 - Exclusively Inside, 2 - On the
↪  Border
template<typename T>
int in_convex_poly(TPoint<T>& p, vector<TPoint<T>>&
  ↪  pts){
  int n = sz(pts);
  if (!n) return 0;
  if (n <= 2) return is_on_seg(p, pts[0], pts.back());
  int l = 1, r = n - 1;
  while (r - l > 1){
    int mid = (l + r) / 2;
    if (acw(pts[mid] - pts[0], p - pts[0])) l = mid;
    else r = mid;
  }
  if (!in_triangle(p, pts[0], pts[l], pts[l + 1]))
↪  return 0;
  if (is_on_seg(p, pts[l], pts[l + 1]) ||
    is_on_seg(p, pts[0], pts.back()) ||
    is_on_seg(p, pts[0], pts[1])
  ) return 2;
  return 1;
}
```

• in_simple_poly

```
// 0 - Outside, 1 - Exclusively Inside, 2 - On the
↪  Border
template<typename T>
int in_simple_poly(TPoint<T> p, vector<TPoint<T>>& pts){
  int n = sz(pts);
  bool res = 0;
  for (int i = 0; i < n; i++){
    auto a = pts[i], b = pts[(i + 1) % n];
    if (is_on_seg(p, a, b)) return 2;
    if (((a.y > p.y) - (b.y > p.y)) * vmul(b - p, a - p)
↪  > TPoint<T>::eps){
      res ^= 1;
    }
  }
  return res;
}
```

• minkowski_rotate

```
template<typename T>
void minkowski_rotate(vector<TPoint<T>>& P){
  int pos = 0;
  for (int i = 1; i < sz(P); i++){
    if (abs(P[i].y - P[pos].y) <= TPoint<T>::eps){
```

```cpp
6        if (P[i].x < P[pos].x) pos = i;
7      }
8      else if (P[i].y < P[pos].y) pos = i;
9    }
10   rotate(P.begin(), P.begin() + pos, P.end());
11 }
```

- minkowski_sum

```cpp
1  // P and Q are strictly convex, points given in
   ↪   counterclockwise order
2  template<typename T>
3  vector<TPoint<T>> minkowski_sum(vector<TPoint<T>> P,
   ↪   vector<TPoint<T>> Q){
4    minkowski_rotate(P);
5    minkowski_rotate(Q);
6    P.pb(P[0]);
7    Q.pb(Q[0]);
8    vector<TPoint<T>> ans;
9    int i = 0, j = 0;
10   while (i < sz(P) - 1 || j < sz(Q) - 1){
11     ans.pb(P[i] + Q[j]);
12     T curmul;
13     if (i == sz(P) - 1) curmul = -1;
14     else if (j == sz(Q) - 1) curmul = +1;
15     else curmul = vmul(P[i + 1] - P[i], Q[j + 1] -
   ↪   Q[j]);
16     if (abs(curmul) < TPoint<T>::eps || curmul > 0) i++;
17     if (abs(curmul) < TPoint<T>::eps || curmul < 0) j++;
18   }
19   return ans;
20 }
21 using Point = TPoint<ll>; using Line = TLine<ll>; using
   ↪   Ray = TRay<ll>; const ld PI = acos(-1);
```

Strings

```cpp
1  vector<int> prefix_function(string s){
2    int n = sz(s);
3    vector<int> pi(n);
4    for (int i = 1; i < n; i++){
5      int k = pi[i - 1];
6      while (k > 0 && s[i] != s[k]){
7        k = pi[k - 1];
8      }
9      pi[i] = k + (s[i] == s[k]);
10   }
11   return pi;
12 }
13 vector<int> kmp(string s, string k){
14   string st = k + "#" + s;
15   vector<int> res;
16   auto pi = prefix_function(st);
17   for (int i = 0; i < sz(st); i++){
18     if (pi[i] == sz(k)){
19       res.pb(i - 2 * sz(k));
20     }
```

```cpp
21   }
22   return res;
23 }
24 vector<int> z_function(string s){
25   int n = sz(s);
26   vector<int> z(n);
27   int l = 0, r = 0;
28   for (int i = 1; i < n; i++){
29     if (r >= i) z[i] = min(z[i - l], r - i + 1);
30     while (i + z[i] < n && s[z[i]] == s[i + z[i]]){
31       z[i]++;
32     }
33     if (i + z[i] - 1 > r){
34       l = i, r = i + z[i] - 1;
35     }
36   }
37   return z;
38 }
```

Manacher's algorithm

```cpp
1  string longest_palindrome(string& s) {
2    // init "abc" -> "^$a#b#c$"
3    vector<char> t{'^', '#'};
4    for (char c : s) t.push_back(c), t.push_back('#');
5    t.push_back('$');
6    // manacher
7    int n = t.size(), r = 0, c = 0;
8    vector<int> p(n, 0);
9    for (int i = 1; i < n - 1; i++) {
10     if (i < r + c) p[i] = min(p[2 * c - i], r + c - i);
11     while (t[i + p[i] + 1] == t[i - p[i] - 1]) p[i]++;
12     if (i + p[i] > r + c) r = p[i], c = i;
13   }
14   // s[i] -> p[2 * i + 2] (even), p[2 * i + 2] (odd)
15   // output answer
16   int index = 0;
17   for (int i = 0; i < n; i++)
18     if (p[index] < p[i]) index = i;
19   return s.substr((index - p[index]) / 2, p[index]);
20 }
```

Flows

$O(N^2M)$, on unit networks $O(N^{1/2}M)$

```cpp
1  struct FlowEdge {
2      int v, u;
3      ll cap, flow = 0;
4      FlowEdge(int v, int u, ll cap) : v(v), u(u),
   ↪   cap(cap) {}
5  };
6  struct Dinic {
7      const ll flow_inf = 1e18;
8      vector<FlowEdge> edges;
9      vector<vector<int>> adj;
```

```cpp
10     int n, m = 0;
11     int s, t;
12     vector<int> level, ptr;
13     queue<int> q;
14     Dinic(int n, int s, int t) : n(n), s(s), t(t) {
15         adj.resize(n);
16         level.resize(n);
17         ptr.resize(n);
18     }
19     void add_edge(int v, int u, ll cap) {
20         edges.emplace_back(v, u, cap);
21         edges.emplace_back(u, v, 0);
22         adj[v].push_back(m);
23         adj[u].push_back(m + 1);
24         m += 2;
25     }
26     bool bfs() {
27         while (!q.empty()) {
28             int v = q.front();
29             q.pop();
30             for (int id : adj[v]) {
31                 if (edges[id].cap - edges[id].flow < 1)
32                     continue;
33                 if (level[edges[id].u] != -1)
34                     continue;
35                 level[edges[id].u] = level[v] + 1;
36                 q.push(edges[id].u);
37             }
38         }
39         return level[t] != -1;
40     }
41     ll dfs(int v, ll pushed) {
42         if (pushed == 0)
43             return 0;
44         if (v == t)
45             return pushed;
46         for (int& cid = ptr[v]; cid <
   ↪   (int)adj[v].size(); cid++) {
47             int id = adj[v][cid];
48             int u = edges[id].u;
49             if (level[v] + 1 != level[u] ||
   ↪   edges[id].cap - edges[id].flow < 1)
50                 continue;
51             ll tr = dfs(u, min(pushed, edges[id].cap -
   ↪   edges[id].flow));
52             if (tr == 0)
53                 continue;
54             edges[id].flow += tr;
55             edges[id ^ 1].flow -= tr;
56             return tr;
57         }
58         return 0;
59     }
60     ll flow() {
61         ll f = 0;
62         while (true) {
63             fill(level.begin(), level.end(), -1);
```

```
    level[s] = 0;
    q.push(s);
    if (!bfs())
      break;
    fill(ptr.begin(), ptr.end(), 0);
    while (ll pushed = dfs(s, flow_inf)) {
      f += pushed;
    }
  }
  return f;
  }
};
// To recover flow through original edges: iterate over
//   even indices in edges.
// To recover minimum cut: DFS from s using ALL of the
//   edges in the Dinic.edges vector for which flow <
//   cap.
```

MCMF – maximize flow, then minimize its cost. $O(mn + Fm \log n)$.

```cpp
#include <ext/pb_ds/priority_queue.hpp>
template <typename T, typename C>
class MCMF {
 public:
  static constexpr T eps = (T) 1e-9;

  struct edge {
    int from;
    int to;
    T c;
    T f;
    C cost;
  };

  int n;
  vector<vector<int>> g;
  vector<edge> edges;
  vector<C> d;
  vector<C> pot;
  __gnu_pbds::priority_queue<pair<C, int>> q;
  vector<typename decltype(q)::point_iterator> its;
  vector<int> pe;
  const C INF_C = numeric_limits<C>::max() / 2;

  explicit MCMF(int n_) : n(n_), g(n), d(n), pot(n, 0),
    its(n), pe(n) {}

  int add(int from, int to, T forward_cap, C edge_cost,
    T backward_cap = 0) {
    assert(0 <= from && from < n && 0 <= to && to < n);
    assert(forward_cap >= 0 && backward_cap >= 0);
    int id = static_cast<int>(edges.size());
    g[from].push_back(id);
    edges.push_back({from, to, forward_cap, 0,
    edge_cost});
    g[to].push_back(id + 1);
    edges.push_back({to, from, backward_cap, 0,
    -edge_cost});
    return id;
  }

  void expath(int st) {
    fill(d.begin(), d.end(), INF_C);
    q.clear();
    fill(its.begin(), its.end(), q.end());
    its[st] = q.push({pot[st], st});
    d[st] = 0;
    while (!q.empty()) {
      int i = q.top().second;
      q.pop();
      its[i] = q.end();
      for (int id : g[i]) {
        const edge &e = edges[id];
        int j = e.to;
        if (e.c - e.f > eps && d[i] + e.cost < d[j]) {
          d[j] = d[i] + e.cost;
          pe[j] = id;
          if (its[j] == q.end()) {
            its[j] = q.push({pot[j] - d[j], j});
          } else {
            q.modify(its[j], {pot[j] - d[j], j});
          }
        }
      }
    }
    swap(d, pot);
  }

  pair<T, C> max_flow(int st, int fin) {
    T flow = 0;
    C cost = 0;
    bool ok = true;
    for (auto& e : edges) {
      if (e.c - e.f > eps && e.cost + pot[e.from] -
    pot[e.to] < 0) {
        ok = false;
        break;
      }
    }
    if (ok) {
      expath(st);
    } else {
      vector<int> deg(n, 0);
      for (int i = 0; i < n; i++) {
        for (int eid : g[i]) {
          auto& e = edges[eid];
          if (e.c - e.f > eps) {
            deg[e.to] += 1;
          }
        }
      }
      vector<int> que;
      for (int i = 0; i < n; i++) {
        if (deg[i] == 0) {
          que.push_back(i);
        }
      }
      for (int b = 0; b < (int) que.size(); b++) {
        for (int eid : g[que[b]]) {
          auto& e = edges[eid];
          if (e.c - e.f > eps) {
            deg[e.to] -= 1;
            if (deg[e.to] == 0) {
              que.push_back(e.to);
            }
          }
        }
      }
      fill(pot.begin(), pot.end(), INF_C);
      pot[st] = 0;
      if (static_cast<int>(que.size()) == n) {
        for (int v : que) {
          if (pot[v] < INF_C) {
            for (int eid : g[v]) {
              auto& e = edges[eid];
              if (e.c - e.f > eps) {
                if (pot[v] + e.cost < pot[e.to]) {
                  pot[e.to] = pot[v] + e.cost;
                  pe[e.to] = eid;
                }
              }
            }
          }
        }
      } else {
        que.assign(1, st);
        vector<bool> in_queue(n, false);
        in_queue[st] = true;
        for (int b = 0; b < (int) que.size(); b++) {
          int i = que[b];
          in_queue[i] = false;
          for (int id : g[i]) {
            const edge &e = edges[id];
            if (e.c - e.f > eps && pot[i] + e.cost <
    pot[e.to]) {
              pot[e.to] = pot[i] + e.cost;
              pe[e.to] = id;
              if (!in_queue[e.to]) {
                que.push_back(e.to);
                in_queue[e.to] = true;
              }
            }
          }
        }
      }
    }
    while (pot[fin] < INF_C) {
      T push = numeric_limits<T>::max();
      int v = fin;
      while (v != st) {
        const edge &e = edges[pe[v]];
```

```
146        push = min(push, e.c - e.f);
147        v = e.from;
148      }
149      v = fin;
150      while (v != st) {
151        edge &e = edges[pe[v]];
152        e.f += push;
153        edge &back = edges[pe[v] ^ 1];
154        back.f -= push;
155        v = e.from;
156      }
157      flow += push;
158      cost += push * pot[fin];
159      expath(st);
160    }
161    return {flow, cost};
162  }
163 };
164
165 // Examples: MCMF<int, int> g(n); g.add(u,v,c,w,0);
  ↪ g.max_flow(s,t).
166 // To recover flow through original edges: iterate over
  ↪ even indices in edges.
```

Graphs

Kuhn's algorithm for bipartite matching

```
1  /*
2  The graph is split into 2 halves of n1 and n2 vertices.
3  Complexity: O(n1 * m). Usually runs much faster. MUCH
  ↪ FASTER!!!
4  */
5  const int N = 305;
6
7  vector<int> g[N]; // Stores edges from left half to
  ↪ right.
8  bool used[N]; // Stores if vertex from left half is
  ↪ used.
9  int mt[N]; // For every vertex in right half, stores to
  ↪ which vertex in left half it's matched (-1 if not
  ↪ matched).
10
11 bool try_dfs(int v){
12   if (used[v]) return false;
13   used[v] = 1;
14   for (auto u : g[v]){
15     if (mt[u] == -1 || try_dfs(mt[u])){
16       mt[u] = v;
17       return true;
18     }
19   }
20   return false;
21 }
22
23 int main(){
24 // ......
```

```
27  for (int i = 1; i <= n2; i++) mt[i] = -1;
28  for (int i = 1; i <= n1; i++) used[i] = 0;
29  for (int i = 1; i <= n1; i++){
30    if (try_dfs(i)){
31      for (int j = 1; j <= n1; j++) used[j] = 0;
32    }
33  }
34  vector<pair<int, int>> ans;
35  for (int i = 1; i <= n2; i++){
36    if (mt[i] != -1) ans.pb({mt[i], i});
37  }
38 }
```

```
// Finding maximal independent set: size = # of nodes -
  ↪ # of edges in matching.
// To construct: launch Kuhn-like DFS from unmatched
  ↪ nodes in the left half.
// Independent set = visited nodes in left half +
  ↪ unvisited in right half.
// Finding minimal vertex cover: complement of maximal
  ↪ independent set.
```

Hungarian algorithm for Assignment Problem

- Given a 1-indexed $(n \times m)$ matrix A, select a number in each row such that each column has at most 1 number selected, and the sum of the selected numbers is minimized.

```
1  int INF = 1e9; // constant greater than any number in
  ↪ the matrix
2  vector<int> u(n+1), v(m+1), p(m+1), way(m+1);
3  for (int i=1; i<=n; ++i) {
4    p[0] = i;
5    int j0 = 0;
6    vector<int> minv (m+1, INF);
7    vector<bool> used (m+1, false);
8    do {
9      used[j0] = true;
10     int i0 = p[j0],  delta = INF,  j1;
11     for (int j=1; j<=m; ++j)
12       if (!used[j]) {
13         int cur = A[i0][j]-u[i0]-v[j];
14         if (cur < minv[j])
15           minv[j] = cur,  way[j] = j0;
16         if (minv[j] < delta)
17           delta = minv[j],  j1 = j;
18       }
19     for (int j=0; j<=m; ++j)
20       if (used[j])
21         u[p[j]] += delta,  v[j] -= delta;
22       else
23         minv[j] -= delta;
24     j0 = j1;
25   } while (p[j0] != 0);
26   do {
```

```
27      int j1 = way[j0];
28      p[j0] = p[j1];
29      j0 = j1;
30    } while (j0);
31 }
32 vector<int> ans (n+1); // ans[i] stores the column
  ↪ selected for row i
33 for (int j=1; j<=m; ++j)
34   ans[p[j]] = j;
35 int cost = -v[0]; // the total cost of the matching
```

Dijkstra's Algorithm

```
priority_queue<pair<ll, ll>, vector<pair<ll, ll>>,
  ↪ greater<pair<ll, ll>>> q;
dist[start] = 0;
q.push({0, start});
while (!q.empty()){
    auto [d, v] = q.top();
    q.pop();
    if (d != dist[v]) continue;
    for (auto [u, w] : g[v]){
      if (dist[u] > dist[v] + w){
        dist[u] = dist[v] + w;
        q.push({dist[u], u});
      }
    }
}
```

Eulerian Cycle DFS

```
1  void dfs(int v){
2    while (!g[v].empty()){
3      int u = g[v].back();
4      g[v].pop_back();
5      dfs(u);
6      ans.pb(v);
7    }
8  }
```

SCC and 2-SAT

```
1  void scc(vector<vector<int>>& g, int* idx) {
2    int n = g.size(), ct = 0;
3    int out[n];
4    vector<int> ginv[n];
5    memset(out, -1, sizeof out);
6    memset(idx, -1, n * sizeof(int));
7    function<void(int)> dfs = [&](int cur) {
8      out[cur] = INT_MAX;
9      for(int v : g[cur]) {
10       ginv[v].push_back(cur);
11       if(out[v] == -1) dfs(v);
12     }
13     ct++; out[cur] = ct;
```

```cpp
  };
  vector<int> order;
  for(int i = 0; i < n; i++) {
    order.push_back(i);
    if(out[i] == -1) dfs(i);
  }
  sort(order.begin(), order.end(), [&](int& u, int& v){
    return out[u] > out[v];
  });
  ct = 0;
  stack<int> s;
  auto dfs2 = [&](int start) {
    s.push(start);
    while(!s.empty()) {
      int cur = s.top();
      s.pop();
      idx[cur] = ct;
      for(int v : ginv[cur])
        if(idx[v] == -1) s.push(v);
    }
  };
  for(int v : order) {
    if(idx[v] == -1) {
      dfs2(v);
      ct++;
    }
  }
}

// 0 => impossible, 1 => possible
pair<int,vector<int>> sat2(int n, vector<pair<int,int>>&
↪  clauses) {
  vector<int> ans(n);
  vector<vector<int>> g(2*n + 1);
  for(auto [x, y] : clauses) {
    x = x < 0 ? -x + n : x;
    y = y < 0 ? -y + n : y;
    int nx = x <= n ? x + n : x - n;
    int ny = y <= n ? y + n : y - n;
    g[nx].push_back(y);
    g[ny].push_back(x);
  }
  int idx[2*n + 1];
  scc(g, idx);
  for(int i = 1; i <= n; i++) {
    if(idx[i] == idx[i + n]) return {0, {}};
    ans[i - 1] = idx[i + n] < idx[i];
  }
  return {1, ans};
}
```

Finding Bridges

```cpp
/*
Bridges.
Results are stored in a map "is_bridge".
For each connected component, call "dfs(starting vertex,
↪  starting vertex)".
```

```cpp
*/
const int N = 2e5 + 10; // Careful with the constant!

vector<int> g[N];
int tin[N], fup[N], timer;
map<pair<int, int>, bool> is_bridge;

void dfs(int v, int p){
  tin[v] = ++timer;
  fup[v] = tin[v];
  for (auto u : g[v]){
    if (!tin[u]){
      dfs(u, v);
      if (fup[u] > tin[v]){
        is_bridge[{u, v}] = is_bridge[{v, u}] = true;
      }
      fup[v] = min(fup[v], fup[u]);
    }
    else{
      if (u != p) fup[v] = min(fup[v], tin[u]);
    }
  }
}
```

Virtual Tree

```cpp
// order stores the nodes in the queried set
sort(all(order), [&] (int u, int v){return tin[u] <
↪  tin[v];});
int m = sz(order);
for (int i = 1; i < m; i++){
    order.pb(lca(order[i], order[i - 1]));
}
sort(all(order), [&] (int u, int v){return tin[u] <
↪  tin[v];});
order.erase(unique(all(order)), order.end());
vector<int> stk{order[0]};
for (int i = 1; i < sz(order); i++){
    int v = order[i];
    while (tout[stk.back()] < tout[v]) stk.pop_back();
    int u = stk.back();
    vg[u].pb({v, dep[v] - dep[u]});
    stk.pb(v);
}
```

HLD on Edges DFS

```cpp
void dfs1(int v, int p, int d){
  par[v] = p;
  for (auto e : g[v]){
    if (e.fi == p){
      g[v].erase(find(all(g[v]), e));
      break;
    }
  }
  dep[v] = d;
  sz[v] = 1;
```

```cpp
  for (auto [u, c] : g[v]){
    dfs1(u, v, d + 1);
    sz[v] += sz[u];
  }
  if (!g[v].empty()) iter_swap(g[v].begin(),
↪  max_element(all(g[v]), comp));
}
void dfs2(int v, int rt, int c){
  pos[v] = sz(a);
  a.pb(c);
  root[v] = rt;
  for (int i = 0; i < sz(g[v]); i++){
    auto [u, c] = g[v][i];
    if (!i) dfs2(u, rt, c);
    else dfs2(u, u, c);
  }
}
int getans(int u, int v){
  int res = 0;
  for (; root[u] != root[v]; v = par[root[v]]){
    if (dep[root[u]] > dep[root[v]]) swap(u, v);
    res = max(res, rmq(0, 0, n - 1, pos[root[v]],
↪  pos[v]));
  }
  if (pos[u] > pos[v]) swap(u, v);
  return max(res, rmq(0, 0, n - 1, pos[u] + 1, pos[v]));
}
```

Centroid Decomposition

```cpp
vector<char> res(n), seen(n), sz(n);
function<int(int, int)> get_size = [&](int node, int fa)
↪  {
  sz[node] = 1;
  for (auto& ne : g[node]) {
    if (ne == fa || seen[ne]) continue;
    sz[node] += get_size(ne, node);
  }
  return sz[node];
};
function<int(int, int, int)> find_centroid = [&](int
↪  node, int fa, int t) {
  for (auto& ne : g[node])
    if (ne != fa && !seen[ne] && sz[ne] > t / 2) return
↪  find_centroid(ne, node, t);
  return node;
};
function<void(int, char)> solve = [&](int node, char
↪  cur) {
  get_size(node, -1); auto c = find_centroid(node, -1,
↪  sz[node]);
  seen[c] = 1, res[c] = cur;
  for (auto& ne : g[c]) {
    if (seen[ne]) continue;
    solve(ne, char(cur + 1)); // we can pass c here to
↪  build tree
```

```
21       }
22  };
```

Math

Binary exponentiation

```
1  ll power(ll a, ll b){
2    ll res = 1;
3    for (; b; a = a * a % MOD, b >>= 1){
4      if (b & 1) res = res * a % MOD;
5    }
6    return res;
7  }
```

Matrix Exponentiation: $O(n^3 \log b)$

```
1  const int N = 100, MOD = 1e9 + 7;
2
3  struct matrix{
4    ll m[N][N];
5    int n;
6    matrix(){
7      n = N;
8      memset(m, 0, sizeof(m));
9    };
10   matrix(int n_){
11     n = n_;
12     memset(m, 0, sizeof(m));
13   };
14   matrix(int n_, ll val){
15     n = n_;
16     memset(m, 0, sizeof(m));
17     for (int i = 0; i < n; i++) m[i][i] = val;
18   };
19
20   matrix operator* (matrix oth){
21     matrix res(n);
22     for (int i = 0; i < n; i++){
23       for (int j = 0; j < n; j++){
24         for (int k = 0; k < n; k++){
25           res.m[i][j] = (res.m[i][j] + m[i][k] *
   ↪ oth.m[k][j]) % MOD;
26         }
27       }
28     }
29     return res;
30   }
31  };
32
33  matrix power(matrix a, ll b){
34    matrix res(a.n, 1);
35    for (; b; a = a * a, b >>= 1){
36      if (b & 1) res = res * a;
37    }
```

```
38    return res;
39  }
```

Extended Euclidean Algorithm

```
1  // gives (x, y) for ax + by = g
2  // solutions given (x0, y0): a(x0 + kb/g) + b(y0 - ka/g)
   ↪ = g
3  int gcd(int a, int b, int& x, int& y) {
4    x = 1, y = 0; int sum1 = a;
5    int x2 = 0, y2 = 1, sum2 = b;
6    while (sum2) {
7      int q = sum1 / sum2;
8      tie(x, x2) = make_tuple(x2, x - q * x2);
9      tie(y, y2) = make_tuple(y2, y - q * y2);
10     tie(sum1, sum2) = make_tuple(sum2, sum1 - q * sum2);
11   }
12   return sum1;
13  }
```

Linear Sieve

- Mobius Function

```
1  vector<int> prime;
2  bool is_composite[MAX_N];
3  int mu[MAX_N];
4
5  void sieve(int n){
6    fill(is_composite, is_composite + n, 0);
7    mu[1] = 1;
8    for (int i = 2; i < n; i++){
9      if (!is_composite[i]){
10       prime.push_back(i);
11       mu[i] = -1; //i is prime
12     }
13     for (int j = 0; j < prime.size() && i * prime[j] < n;
   ↪  j++){
14       is_composite[i * prime[j]] = true;
15       if (i % prime[j] == 0){
16         mu[i * prime[j]] = 0; //prime[j] divides i
17         break;
18       } else {
19         mu[i * prime[j]] = -mu[i]; //prime[j] does not
   ↪  divide i
20       }
21     }
22   }
23  }
```

- Euler's Totient Function

```
1  vector<int> prime;
2  bool is_composite[MAX_N];
3  int phi[MAX_N];
4
5  void sieve(int n){
6    fill(is_composite, is_composite + n, 0);
```

```
7    phi[1] = 1;
8    for (int i = 2; i < n; i++){
9      if (!is_composite[i]){
10       prime.push_back (i);
11       phi[i] = i - 1; //i is prime
12     }
13     for (int j = 0; j < prime.size () && i * prime[j] < n;
   ↪  j++){
14       is_composite[i * prime[j]] = true;
15       if (i % prime[j] == 0){
16         phi[i * prime[j]] = phi[i] * prime[j]; //prime[j]
   ↪  divides i
17         break;
18       } else {
19         phi[i * prime[j]] = phi[i] * phi[prime[j]];
   ↪  //prime[j] does not divide i
20       }
21     }
22   }
23  }
```

Gaussian Elimination

```
1  bool is_0(Z v) { return v.x == 0; }
2  Z abs(Z v) { return v; }
3  bool is_0(double v) { return abs(v) < 1e-9; }
4
5  // 1 => unique solution, 0 => no solution, -1 =>
   ↪  multiple solutions
6  template <typename T>
7  int gaussian_elimination(vector<vector<T>> &a, int
   ↪  limit) {
8    if (a.empty() || a[0].empty()) return -1;
9    int h = (int)a.size(), w = (int)a[0].size(), r = 0;
10   for (int c = 0; c < limit; c++) {
11     int id = -1;
12     for (int i = r; i < h; i++) {
13       if (!is_0(a[i][c]) && (id == -1 || abs(a[id][c]) <
   ↪  abs(a[i][c]))) {
14         id = i;
15       }
16     }
17     if (id == -1) continue;
18     if (id > r) {
19       swap(a[r], a[id]);
20       for (int j = c; j < w; j++) a[id][j] = -a[id][j];
21     }
22     vector<int> nonzero;
23     for (int j = c; j < w; j++) {
24       if (!is_0(a[r][j])) nonzero.push_back(j);
25     }
26     T inv_a = 1 / a[r][c];
27     for (int i = r + 1; i < h; i++) {
28       if (is_0(a[i][c])) continue;
29       T coeff = -a[i][c] * inv_a;
30       for (int j : nonzero) a[i][j] += coeff * a[r][j];
31     }
```

```
32        ++r;
33      }
34      for (int row = h - 1; row >= 0; row--) {
35        for (int c = 0; c < limit; c++) {
36          if (!is_0(a[row][c])) {
37            T inv_a = 1 / a[row][c];
38            for (int i = row - 1; i >= 0; i--) {
39              if (is_0(a[i][c])) continue;
40              T coeff = -a[i][c] * inv_a;
41              for (int j = c; j < w; j++) a[i][j] += coeff *
   ↪ a[row][j];
42            }
43            break;
44          }
45        }
46      } // not-free variables: only it on its line
47      for(int i = r; i < h; i++) if(!is_0(a[i][limit]))
   ↪ return 0;
48      return (r == limit) ? 1 : -1;
49  }
50
51  template <typename T>
52  pair<int,vector<T>> solve_linear(vector<vector<T>> a,
   ↪ const vector<T> &b, int w) {
53    int h = (int)a.size();
54    for (int i = 0; i < h; i++) a[i].push_back(b[i]);
55    int sol = gaussian_elimination(a, w);
56    if(!sol) return {0, vector<T>()};
57    vector<T> x(w, 0);
58    for (int i = 0; i < h; i++) {
59      for (int j = 0; j < w; j++) {
60        if (!is_0(a[i][j])) {
61          x[j] = a[i][w] / a[i][j];
62          break;
63        }
64      }
65    }
66    return {sol, x};
67  }
```

is__prime

- (Miller–Rabin primality test)

```
1   typedef __int128_t i128;
2
3   i128 power(i128 a, i128 b, i128 MOD = 1, i128 res = 1) {
4     for (; b; b /= 2, (a *= a) %= MOD)
5       if (b & 1) (res *= a) %= MOD;
6     return res;
7   }
8
9   bool is_prime(ll n) {
10    if (n < 2) return false;
11    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17,
   ↪ 19, 23};
12    int s = __builtin_ctzll(n - 1);
13    ll d = (n - 1) >> s;
```

```
14    for (auto a : A) {
15      if (a == n) return true;
16      ll x = (ll)power(a, d, n);
17      if (x == 1 || x == n - 1) continue;
18      bool ok = false;
19      for (int i = 0; i < s - 1; ++i) {
20        x = ll((i128)x * x % n);  // potential overflow!
21        if (x == n - 1) {
22          ok = true;
23          break;
24        }
25      }
26      if (!ok) return false;
27    }
28    return true;
29  }
```

```
1   typedef __int128_t i128;
2
3   ll pollard_rho(ll x) {
4     ll s = 0, t = 0, c = rng() % (x - 1) + 1;
5     ll stp = 0, goal = 1, val = 1;
6     for (goal = 1;; goal *= 2, s = t, val = 1) {
7       for (stp = 1; stp <= goal; ++stp) {
8         t = ll(((i128)t * t + c) % x);
9         val = ll((i128)val * abs(t - s) % x);
10        if ((stp % 127) == 0) {
11          ll d = gcd(val, x);
12          if (d > 1) return d;
13        }
14      }
15      ll d = gcd(val, x);
16      if (d > 1) return d;
17    }
18  }
19
20  ll get_max_factor(ll _x) {
21    ll max_factor = 0;
22    function<void(ll)> fac = [&](ll x) {
23      if (x <= max_factor || x < 2) return;
24      if (is_prime(x)) {
25        max_factor = max_factor > x ? max_factor : x;
26        return;
27      }
28      ll p = x;
29      while (p >= x) p = pollard_rho(x);
30      while ((x % p) == 0) x /= p;
31      fac(x), fac(p);
32    };
33    fac(_x);
34    return max_factor;
35  }
```

Berlekamp-Massey

- Recovers any n-order linear recurrence relation from the first $2n$ terms of the sequence.

- Input s is the sequence to be analyzed.
- Output c is the shortest sequence $c_1, ..., c_n$, such that

$$s_m = \sum_{i=1}^{n} c_i \cdot s_{m-i}, \text{ for all } m \geq n.$$

- Be careful since c is returned in 0-based indexation.
- Complexity: $O(N^2)$

```
1   vector<ll> berlekamp_massey(vector<ll> s) {
2     int n = sz(s), l = 0, m = 1;
3     vector<ll> b(n), c(n);
4     ll ldd = b[0] = c[0] = 1;
5     for (int i = 0; i < n; i++, m++) {
6       ll d = s[i];
7       for (int j = 1; j <= l; j++) d = (d + c[j] * s[i -
   ↪ j]) % MOD;
8       if (d == 0) continue;
9       vector<ll> temp = c;
10      ll coef = d * power(ldd, MOD - 2) % MOD;
11      for (int j = m; j < n; j++){
12        c[j] = (c[j] + MOD - coef * b[j - m]) % MOD;
13        if (c[j] < 0) c[j] += MOD;
14      }
15      if (2 * l <= i) {
16        l = i + 1 - l;
17        b = temp;
18        ldd = d;
19        m = 0;
20      }
21    }
22    c.resize(l + 1);
23    c.erase(c.begin());
24    for (ll &x : c)
25      x = (MOD - x) % MOD;
26    return c;
27  }
```

Calculating k-th term of a linear recurrence

- Given the first n terms $s_0, s_1, ..., s_{n-1}$ and the sequence $c_1, c_2, ..., c_n$ such that

$$s_m = \sum_{i=1}^{n} c_i \cdot s_{m-i}, \text{ for all } m \geq n,$$

the function calc_kth computes s_k.
- Complexity: $O(n^2 \log k)$

```cpp
vector<ll> poly_mult_mod(vector<ll> p, vector<ll> q,
↪   vector<ll>& c){
  vector<ll> ans(sz(p) + sz(q) - 1);
  for (int i = 0; i < sz(p); i++){
    for (int j = 0; j < sz(q); j++){
      ans[i + j] = (ans[i + j] + p[i] * q[j]) % MOD;
    }
  }
  int n = sz(ans), m = sz(c);
  for (int i = n - 1; i >= m; i--){
    for (int j = 0; j < m; j++){
      ans[i - 1 - j] = (ans[i - 1 - j] + c[j] * ans[i])
↪   % MOD;
    }
  }
  ans.resize(m);
  return ans;
}

ll calc_kth(vector<ll> s, vector<ll> c, ll k){
  assert(sz(s) >= sz(c)); // size of s can be greater
↪   than c, but not less
  if (k < sz(s)) return s[k];
  vector<ll> res{1};
  for (vector<ll> poly = {0, 1}; k; poly =
↪   poly_mult_mod(poly, poly, c), k >>= 1){
    if (k & 1) res = poly_mult_mod(res, poly, c);
  }
  ll ans = 0;
  for (int i = 0; i < min(sz(res), sz(c)); i++) ans =
↪   (ans + s[i] * res[i]) % MOD;
  return ans;
}
```

Partition Function

- Returns number of partitions of n in $O(n^{1.5})$

```cpp
int partition(int n) {
  int dp[n + 1];
  dp[0] = 1;
  for (int i = 1; i <= n; i++) {
    dp[i] = 0;
    for (int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0;
↪   ++j, r *= -1) {
      dp[i] += dp[i - (3 * j * j - j) / 2] * r;
      if (i - (3 * j * j + j) / 2 >= 0) dp[i] += dp[i -
↪   (3 * j * j + j) / 2] * r;
    }
  }
  return dp[n];
}
```

NTT

```cpp
void ntt(vector<ll>& a, int f) {
  int n = int(a.size());
```

```cpp
  vector<ll> w(n);
  vector<int> rev(n);
  for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] / 2)
↪   | ((i & 1) * (n / 2));
  for (int i = 0; i < n; i++) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  ll wn = power(f ? (MOD + 1) / 3 : 3, (MOD - 1) / n);
  w[0] = 1;
  for (int i = 1; i < n; i++) w[i] = w[i - 1] * wn %
↪   MOD;
  for (int mid = 1; mid < n; mid *= 2) {
    for (int i = 0; i < n; i += 2 * mid) {
      for (int j = 0; j < mid; j++) {
        ll x = a[i + j], y = a[i + j + mid] * w[n / (2 *
↪   mid) * j] % MOD;
        a[i + j] = (x + y) % MOD, a[i + j + mid] = (x +
↪   MOD - y) % MOD;
      }
    }
  }
  if (f) {
    ll iv = power(n, MOD - 2);
    for (auto& x : a) x = x * iv % MOD;
  }
}
vector<ll> mul(vector<ll> a, vector<ll> b) {
  int n = 1, m = (int)a.size() + (int)b.size() - 1;
  while (n < m) n *= 2;
  a.resize(n), b.resize(n);
  ntt(a, 0), ntt(b, 0); // if squaring, you can save one
↪   NTT here
  for (int i = 0; i < n; i++) a[i] = a[i] * b[i] % MOD;
  ntt(a, 1);
  a.resize(m);
  return a;
}
```

FFT

```cpp
const ld PI = acosl(-1);
auto mul = [&](const vector<ld>& aa, const vector<ld>&
↪   bb) {
  int n = (int)aa.size(), m = (int)bb.size(), bit = 1;
  while ((1 << bit) < n + m - 1) bit++;
  int len = 1 << bit;
  vector<complex<ld>> a(len), b(len);
  vector<int> rev(len);
  for (int i = 0; i < n; i++) a[i].real(aa[i]);
  for (int i = 0; i < m; i++) b[i].real(bb[i]);
  for (int i = 0; i < len; i++) rev[i] = (rev[i >> 1] >>
↪   1) | ((i & 1) << (bit - 1));
  auto fft = [&](vector<complex<ld>>& p, int inv) {
    for (int i = 0; i < len; i++)
      if (i < rev[i]) swap(p[i], p[rev[i]]);
    for (int mid = 1; mid < len; mid *= 2) {
      auto w1 = complex<ld>(cos(PI / mid), (inv ? -1 :
↪   1) * sin(PI / mid));
```

```cpp
      for (int i = 0; i < len; i += mid * 2) {
        auto wk = complex<ld>(1, 0);
        for (int j = 0; j < mid; j++, wk = wk * w1) {
          auto x = p[i + j], y = wk * p[i + j + mid];
          p[i + j] = x + y, p[i + j + mid] = x - y;
        }
      }
    }
    if (inv == 1) {
      for (int i = 0; i < len; i++)
↪   p[i].real(p[i].real() / len);
    }
  };
  fft(a, 0), fft(b, 0);
  for (int i = 0; i < len; i++) a[i] = a[i] * b[i];
  fft(a, 1);
  a.resize(n + m - 1);
  vector<ld> res(n + m - 1);
  for (int i = 0; i < n + m - 1; i++) res[i] =
↪   a[i].real();
  return res;
};
```

MIT's FFT/NTT, Polynomial mod/log/exp Template

- For integers rounding works if $(|a| + |b|) \max(a, b) < {\sim}10^9$, or in theory maybe 10^6
- $\frac{1}{P(x)}$ in $O(n \log n)$, $e^{P(x)}$ in $O(n \log n)$, $\ln(P(x))$ in $O(n \log n)$, $P(x)^k$ in $O(n \log n)$, Evaluates $P(x_1), \cdots, P(x_n)$ in $O(n \log^2 n)$, Lagrange Interpolation in $O(n \log^2 n)$

```cpp
// use #define FFT 1 to use FFT instead of NTT (default)
// Examples:
// poly a(n+1); // constructs degree n poly
// a[0].v = 10; // assigns constant term a_0 = 10
// poly b = exp(a);
// poly is vector<num>
// for NTT, num stores just one int named v
// for FFT, num stores two doubles named x (real), y
↪   (imag)

#define sz(x) ((int)x.size())
#define rep(i, j, k) for (int i = int(j); i < int(k);
↪   i++)
#define trav(a, x) for (auto &a : x)
#define per(i, a, b) for (int i = (b)-1; i >= (a); --i)
using ll = long long;
using vi = vector<int>;

namespace fft {
#if FFT
// FFT
```

```cpp
20   using dbl = double;
21   struct num {
22     dbl x, y;
23     num(dbl x_ = 0, dbl y_ = 0): x(x_), y(y_) {}
24   };
25   inline num operator+(num a, num b) {
26     return num(a.x + b.x, a.y + b.y);
27   }
28   inline num operator-(num a, num b) {
29     return num(a.x - b.x, a.y - b.y);
30   }
31   inline num operator*(num a, num b) {
32     return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y *
         b.x);
33   }
34   inline num conj(num a) { return num(a.x, -a.y); }
35   inline num inv(num a) {
36     dbl n = (a.x * a.x + a.y * a.y);
37     return num(a.x / n, -a.y / n);
38   }
39
40   #else
41   // NTT
42   const int mod = 998244353, g = 3;
43   // For p < 2~30 there is also (5 << 25, 3), (7 << 26,
         3),
44   // (479 << 21, 3) and (483 << 21, 5). Last two are >
         10~9.
45   struct num {
46     int v;
47     num(ll v_ = 0): v(int(v_ % mod)) {
48       if (v < 0) v += mod;
49     }
50     explicit operator int() const { return v; }
51   };
52   inline num operator+(num a, num b) { return num(a.v +
         b.v); }
53   inline num operator-(num a, num b) {
54     return num(a.v + mod - b.v);
55   }
56   inline num operator*(num a, num b) {
57     return num(1ll * a.v * b.v);
58   }
59   inline num pow(num a, int b) {
60     num r = 1;
61     do {
62       if (b & 1) r = r * a;
63       a = a * a;
64     } while (b >>= 1);
65     return r;
66   }
67   inline num inv(num a) { return pow(a, mod - 2); }
68
69   #endif
70   using vn = vector<num>;
71   vi rev({0, 1});
72   vn rt(2, num(1)), fa, fb;
73   inline void init(int n) {
```

```cpp
74     if (n <= sz(rt)) return;
75     rev.resize(n);
76     rep(i, 0, n) rev[i] = (rev[i >> 1] | ((i & 1) * n)) >>
         1;
77     rt.reserve(n);
78     for (int k = sz(rt); k < n; k *= 2) {
79       rt.resize(2 * k);
80   #if FFT
81       double a = M_PI / k;
82       num z(cos(a), sin(a)); // FFT
83   #else
84       num z = pow(num(g), (mod - 1) / (2 * k)); // NTT
85   #endif
86       rep(i, k / 2, k) rt[2 * i] = rt[i],
87                        rt[2 * i + 1] = rt[i] * z;
88     }
89   }
90   inline void fft(vector<num>& a, int n) {
91     init(n);
92     int s = __builtin_ctz(sz(rev) / n);
93     rep(i, 0, n) if (i < rev[i] >> s) swap(a[i], a[rev[i]
         >> s]);
94     for (int k = 1; k < n; k *= 2)
95       for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
96         num t = rt[j + k] * a[i + j + k];
97         a[i + j + k] = a[i + j] - t;
98         a[i + j] = a[i + j] + t;
99       }
100  }
101  // Complex/NTT
102  vn multiply(vn a, vn b) {
103    int s = sz(a) + sz(b) - 1;
104    if (s <= 0) return {};
105    int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1
         << L;
106    a.resize(n), b.resize(n);
107    fft(a, n);
108    fft(b, n);
109    num d = inv(num(n));
110    rep(i, 0, n) a[i] = a[i] * b[i] * d;
111    reverse(a.begin() + 1, a.end());
112    fft(a, n);
113    a.resize(s);
114    return a;
115  }
116  // Complex/NTT power-series inverse
117  // Doubles b as b[:n] = (2 - a[:n] * b[:n/2]) * b[:n/2]
118  vn inverse(const vn& a) {
119    if (a.empty()) return {};
120    vn b({inv(a[0])});
121    b.reserve(2 * a.size());
122    while (sz(b) < sz(a)) {
123      int n = 2 * sz(b);
124      b.resize(2 * n, 0);
125      if (sz(fa) < 2 * n) fa.resize(2 * n);
126      fill(fa.begin(), fa.begin() + 2 * n, 0);
127      copy(a.begin(), a.begin() + min(n, sz(a)),
         fa.begin());
```

```cpp
128      fft(b, 2 * n);
129      fft(fa, 2 * n);
130      num d = inv(num(2 * n));
131      rep(i, 0, 2 * n) b[i] = b[i] * (2 - fa[i] * b[i]) *
         d;
132      reverse(b.begin() + 1, b.end());
133      fft(b, 2 * n);
134      b.resize(n);
135    }
136    b.resize(a.size());
137    return b;
138  }
139  #if FFT
140  // Double multiply (num = complex)
141  using vd = vector<double>;
142  vd multiply(const vd& a, const vd& b) {
143    int s = sz(a) + sz(b) - 1;
144    if (s <= 0) return {};
145    int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1
         << L;
146    if (sz(fa) < n) fa.resize(n);
147    if (sz(fb) < n) fb.resize(n);
148    fill(fa.begin(), fa.begin() + n, 0);
149    rep(i, 0, sz(a)) fa[i].x = a[i];
150    rep(i, 0, sz(b)) fa[i].y = b[i];
151    fft(fa, n);
152    trav(x, fa) x = x * x;
153    rep(i, 0, n) fb[i] = fa[(n - i) & (n - 1)] -
         conj(fa[i]);
154    fft(fb, n);
155    vd r(s);
156    rep(i, 0, s) r[i] = fb[i].y / (4 * n);
157    return r;
158  }
159  // Integer multiply mod m (num = complex)
160  vi multiply_mod(const vi& a, const vi& b, int m) {
161    int s = sz(a) + sz(b) - 1;
162    if (s <= 0) return {};
163    int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1
         << L;
164    if (sz(fa) < n) fa.resize(n);
165    if (sz(fb) < n) fb.resize(n);
166    rep(i, 0, sz(a)) fa[i] =
167      num(a[i] & ((1 << 15) - 1), a[i] >> 15);
168    fill(fa.begin() + sz(a), fa.begin() + n, 0);
169    rep(i, 0, sz(b)) fb[i] =
170      num(b[i] & ((1 << 15) - 1), b[i] >> 15);
171    fill(fb.begin() + sz(b), fb.begin() + n, 0);
172    fft(fa, n);
173    fft(fb, n);
174    double r0 = 0.5 / n; // 1/2n
175    rep(i, 0, n / 2 + 1) {
176      int j = (n - i) & (n - 1);
177      num g0 = (fb[i] + conj(fb[j])) * r0;
178      num g1 = (fb[i] - conj(fb[j])) * r0;
179      swap(g1.x, g1.y);
180      g1.y *= -1;
```

```cpp
      if (j != i) {
        swap(fa[j], fa[i]);
        fb[j] = fa[j] * g1;
        fa[j] = fa[j] * g0;
      }
      fb[i] = fa[i] * conj(g1);
      fa[i] = fa[i] * conj(g0);
    }
    fft(fa, n);
    fft(fb, n);
    vi r(s);
    rep(i, 0, s) r[i] =
      int((ll(fa[i].x + 0.5) + (ll(fa[i].y + 0.5) % m << 15) +
          (ll(fb[i].x + 0.5) % m << 15) +
          (ll(fb[i].y + 0.5) % m << 30)) %
         m);
    return r;
  }
  #endif
} // namespace fft
// For multiply_mod, use num = modnum, poly =
//   vector<num>
using fft::num;
using poly = fft::vn;
using fft::multiply;
using fft::inverse;

poly& operator+=(poly& a, const poly& b) {
  if (sz(a) < sz(b)) a.resize(b.size());
  rep(i, 0, sz(b)) a[i] = a[i] + b[i];
  return a;
}
poly operator+(const poly& a, const poly& b) {
  poly r = a;
  r += b;
  return r;
}
poly& operator-=(poly& a, const poly& b) {
  if (sz(a) < sz(b)) a.resize(b.size());
  rep(i, 0, sz(b)) a[i] = a[i] - b[i];
  return a;
}
poly operator-(const poly& a, const poly& b) {
  poly r = a;
  r -= b;
  return r;
}
poly operator*(const poly& a, const poly& b) {
  return multiply(a, b);
}
poly& operator*=(poly& a, const poly& b) { return a = a * b; }

poly& operator*=(poly& a, const num& b) { // Optional
  trav(x, a) x = x * b;
  return a;
}

poly operator*(const poly& a, const num& b) {
  poly r = a;
  r *= b;
  return r;
}
// Polynomial floor division; no leading 0's please
poly operator/(poly a, poly b) {
  if (sz(a) < sz(b)) return {};
  int s = sz(a) - sz(b) + 1;
  reverse(a.begin(), a.end());
  reverse(b.begin(), b.end());
  a.resize(s);
  b.resize(s);
  a = a * inverse(move(b));
  a.resize(s);
  reverse(a.begin(), a.end());
  return a;
}
poly& operator/=(poly& a, const poly& b) { return a = a / b; }
poly& operator%=(poly& a, const poly& b) {
  if (sz(a) >= sz(b)) {
    poly c = (a / b) * b;
    a.resize(sz(b) - 1);
    rep(i, 0, sz(a)) a[i] = a[i] - c[i];
  }
  return a;
}
poly operator%(const poly& a, const poly& b) {
  poly r = a;
  r %= b;
  return r;
}
// Log/exp/pow
poly deriv(const poly& a) {
  if (a.empty()) return {};
  poly b(sz(a) - 1);
  rep(i, 1, sz(a)) b[i - 1] = a[i] * i;
  return b;
}
poly integ(const poly& a) {
  poly b(sz(a) + 1);
  b[1] = 1; // mod p
  rep(i, 2, sz(b)) b[i] =
    b[fft::mod % i] * (-fft::mod / i); // mod p
  rep(i, 1, sz(b)) b[i] = a[i - 1] * b[i]; // mod p
  //rep(i,1,sz(b)) b[i]=a[i-1]*inv(num(i)); // else
  return b;
}
poly log(const poly& a) { // MUST have a[0] == 1
  poly b = integ(deriv(a) * inverse(a));
  b.resize(a.size());
  return b;
}
poly exp(const poly& a) { // MUST have a[0] == 0
  poly b(1, num(1));
  if (a.empty()) return b;
  while (sz(b) < sz(a)) {
    int n = min(sz(b) * 2, sz(a));
    b.resize(n);
    poly v = poly(a.begin(), a.begin() + n) - log(b);
    v[0] = v[0] + num(1);
    b *= v;
    b.resize(n);
  }
  return b;
}
poly pow(const poly& a, int m) { // m >= 0
  poly b(a.size());
  if (!m) {
    b[0] = 1;
    return b;
  }
  int p = 0;
  while (p < sz(a) && a[p].v == 0) ++p;
  if (1ll * m * p >= sz(a)) return b;
  num mu = pow(a[p], m), di = inv(a[p]);
  poly c(sz(a) - m * p);
  rep(i, 0, sz(c)) c[i] = a[i + p] * di;
  c = log(c);
  trav(v, c) v = v * m;
  c = exp(c);
  rep(i, 0, sz(c)) b[i + m * p] = c[i] * mu;
  return b;
}
// Multipoint evaluation/interpolation

vector<num> eval(const poly& a, const vector<num>& x) {
  int n = sz(x);
  if (!n) return {};
  vector<poly> up(2 * n);
  rep(i, 0, n) up[i + n] = poly({0 - x[i], 1});
  per(i, 1, n) up[i] = up[2 * i] * up[2 * i + 1];
  vector<poly> down(2 * n);
  down[1] = a % up[1];
  rep(i, 2, 2 * n) down[i] = down[i / 2] % up[i];
  vector<num> y(n);
  rep(i, 0, n) y[i] = down[i + n][0];
  return y;
}

poly interp(const vector<num>& x, const vector<num>& y) {
  int n = sz(x);
  assert(n);
  vector<poly> up(n * 2);
  rep(i, 0, n) up[i + n] = poly({0 - x[i], 1});
  per(i, 1, n) up[i] = up[2 * i] * up[2 * i + 1];
  vector<num> a = eval(deriv(up[1]), x);
  vector<poly> down(2 * n);
  rep(i, 0, n) down[i + n] = poly({y[i] * inv(a[i])});
  per(i, 1, n) down[i] =
    down[i * 2] * up[i * 2 + 1] + down[i * 2 + 1] * up[i * 2];
  return down[1];
}
```

Data Structures

Fenwick Tree

```cpp
ll sum(int r) {
    ll ret = 0;
    for (; r >= 0; r = (r & r + 1) - 1) ret += bit[r];
    return ret;
}
void add(int idx, ll delta) {
    for (; idx < n; idx |= idx + 1) bit[idx] += delta;
}
```

Lazy Propagation SegTree

```cpp
// Clear: clear() or build()
const int N = 2e5 + 10; // Change the constant!
template<typename T>
struct LazySegTree{
  T t[4 * N];
  T lazy[4 * N];
  int n;

  // Change these functions, default return, and lazy
  // mark.
  T default_return = 0, lazy_mark =
  numeric_limits<T>::min();
  // Lazy mark is how the algorithm will identify that
  // no propagation is needed.
  function<T(T, T)> f = [&] (T a, T b){
    return a + b;
  };
  // f_on_seg calculates the function f, knowing the
  // lazy value on segment,
  // segment's size and the previous value.
  // The default is segment modification for RSQ. For
  // increments change to:
  //     return cur_seg_val + seg_size * lazy_val;
  // For RMQ.  Modification: return lazy_val;
  //  Increments: return cur_seg_val + lazy_val;
  function<T(T, int, T)> f_on_seg = [&] (T cur_seg_val,
  int seg_size, T lazy_val){
    return seg_size * lazy_val;
  };
  // upd_lazy updates the value to be propagated to
  // child segments.
  // Default: modification. For increments change to:
  //     lazy[v] = (lazy[v] == lazy_mark? val : lazy[v]
  //  + val);
  function<void(int, T)> upd_lazy = [&] (int v, T val){
    lazy[v] = val;
  };
  // Tip: for "get element on single index" queries, use
  // max() on segment: no overflows.

  LazySegTree(int n_) : n(n_) {
    clear(n);
  }

  void build(int v, int tl, int tr, vector<T>& a){
    if (tl == tr) {
      t[v] = a[tl];
      return;
    }
    int tm = (tl + tr) / 2;
    // left child: [tl, tm]
    // right child: [tm + 1, tr]
    build(2 * v + 1, tl, tm, a);
    build(2 * v + 2, tm + 1, tr, a);
    t[v] = f(t[2 * v + 1], t[2 * v + 2]);
  }

  LazySegTree(vector<T>& a){
    build(a);
  }

  void push(int v, int tl, int tr){
    if (lazy[v] == lazy_mark) return;
    int tm = (tl + tr) / 2;
    t[2 * v + 1] = f_on_seg(t[2 * v + 1], tm - tl + 1,
    lazy[v]);
    t[2 * v + 2] = f_on_seg(t[2 * v + 2], tr - tm,
    lazy[v]);
    upd_lazy(2 * v + 1, lazy[v]), upd_lazy(2 * v + 2,
    lazy[v]);
    lazy[v] = lazy_mark;
  }

  void modify(int v, int tl, int tr, int l, int r, T
  val){
    if (l > r) return;
    if (tl == l && tr == r){
      t[v] = f_on_seg(t[v], tr - tl + 1, val);
      upd_lazy(v, val);
      return;
    }
    push(v, tl, tr);
    int tm = (tl + tr) / 2;
    modify(2 * v + 1, tl, tm, l, min(r, tm), val);
    modify(2 * v + 2, tm + 1, tr, max(l, tm + 1), r,
    val);
    t[v] = f(t[2 * v + 1], t[2 * v + 2]);
  }

  T query(int v, int tl, int tr, int l, int r) {
    if (l > r) return default_return;
    if (tl == l && tr == r) return t[v];
    push(v, tl, tr);
    int tm = (tl + tr) / 2;
    return f(
      query(2 * v + 1, tl, tm, l, min(r, tm)),
      query(2 * v + 2, tm + 1, tr, max(l, tm + 1), r)
    );
  }

  void modify(int l, int r, T val){
    modify(0, 0, n - 1, l, r, val);
  }

  T query(int l, int r){
    return query(0, 0, n - 1, l, r);
  }

  T get(int pos){
    return query(pos, pos);
  }

  // Change clear() function to t.clear() if using
  // unordered_map for SegTree!!!
  void clear(int n_){
    n = n_;
    for (int i = 0; i < 4 * n; i++) t[i] = 0, lazy[i] =
    lazy_mark;
  }

  void build(vector<T>& a){
    n = sz(a);
    clear(n);
    build(0, 0, n - 1, a);
  }
};
```

Sparse Table

```cpp
const int N = 2e5 + 10, LOG = 20; // Change the
// constant!
template<typename T>
struct SparseTable{
int lg[N];
T st[N][LOG];
int n;

// Change this function
function<T(T, T)> f = [&] (T a, T b){
  return min(a, b);
};

void build(vector<T>& a){
  n = sz(a);
  lg[1] = 0;
  for (int i = 2; i <= n; i++) lg[i] = lg[i / 2] + 1;

  for (int k = 0; k < LOG; k++){
    for (int i = 0; i < n; i++){
      if (!k) st[i][k] = a[i];
      else st[i][k] = f(st[i][k - 1], st[min(n - 1, i +
      (1 << (k - 1)))][k - 1]);
    }
  }
```

```
24   }
25
26   T query(int l, int r){
27     int sz = r - l + 1;
28     return f(st[l][lg[sz]], st[r - (1 << lg[sz]) +
    ↪ 1][lg[sz]]);
29   }
30 };
```

Suffix Array and LCP array

- (uses SparseTable above)

```
1  struct SuffixArray{
2    vector<int> p, c, h;
3    SparseTable<int> st;
4    /*
5    In the end, array c gives the position of each suffix
   ↪ in p
6    using 1-based indexation!
7    */
8
9    SuffixArray() {}
10
11   SuffixArray(string s){
12     buildArray(s);
13     buildLCP(s);
14     buildSparse();
15   }
16
17   void buildArray(string s){
18     int n = sz(s) + 1;
19     p.resize(n), c.resize(n);
20     for (int i = 0; i < n; i++) p[i] = i;
21     sort(all(p), [&] (int a, int b){return s[a] <
   ↪ s[b];});
22     c[p[0]] = 0;
23     for (int i = 1; i < n; i++){
24       c[p[i]] = c[p[i - 1]] + (s[p[i]] != s[p[i - 1]]);
25     }
26     vector<int> p2(n), c2(n);
27     // w is half-length of each string.
28     for (int w = 1; w < n; w <<= 1){
29       for (int i = 0; i < n; i++){
30         p2[i] = (p[i] - w + n) % n;
31       }
32       vector<int> cnt(n);
33       for (auto i : c) cnt[i]++;
34       for (int i = 1; i < n; i++) cnt[i] += cnt[i - 1];
35       for (int i = n - 1; i >= 0; i--){
36         p[--cnt[c[p2[i]]]] = p2[i];
37       }
38       c2[p[0]] = 0;
39       for (int i = 1; i < n; i++){
40         c2[p[i]] = c2[p[i - 1]] +
41         (c[p[i]] != c[p[i - 1]] ||
42         c[(p[i] + w) % n] != c[(p[i - 1] + w) % n]);
43       }
```

```
44       c.swap(c2);
45     }
46     p.erase(p.begin());
47   }
48
49   void buildLCP(string s){
50     // The algorithm assumes that suffix array is
   ↪ already built on the same string.
51     int n = sz(s);
52     h.resize(n - 1);
53     int k = 0;
54     for (int i = 0; i < n; i++){
55       if (c[i] == n){
56         k = 0;
57         continue;
58       }
59       int j = p[c[i]];
60       while (i + k < n && j + k < n && s[i + k] == s[j +
   ↪ k]) k++;
61       h[c[i] - 1] = k;
62       if (k) k--;
63     }
64     /*
65     Then an RMQ Sparse Table can be built on array h
66     to calculate LCP of 2 non-consecutive suffixes.
67     */
68   }
69
70   void buildSparse(){
71     st.build(h);
72   }
73
74   // l and r must be in 0-BASED INDEXATION
75   int lcp(int l, int r){
76     l = c[l] - 1, r = c[r] - 1;
77     if (l > r) swap(l, r);
78     return st.query(l, r - 1);
79   }
80 };
```

Aho Corasick Trie

- For each node in the trie, the suffix link points to the longest proper suffix of the represented string. The terminal-link tree has square-root height (can be constructed by DFS).

```
1  const int S = 26;
2
3  // Function converting char to int.
4  int ctoi(char c){
5    return c - 'a';
6  }
7
8  // To add terminal links, use DFS
9  struct Node{
10   vector<int> nxt;
```

```
11   int link;
12   bool terminal;
13
14   Node() {
15     nxt.assign(S, -1), link = 0, terminal = 0;
16   }
17 };
18
19 vector<Node> trie(1);
20
21 // add_string returns the terminal vertex.
22 int add_string(string& s){
23   int v = 0;
24   for (auto c : s){
25     int cur = ctoi(c);
26     if (trie[v].nxt[cur] == -1){
27       trie[v].nxt[cur] = sz(trie);
28       trie.emplace_back();
29     }
30     v = trie[v].nxt[cur];
31   }
32   trie[v].terminal = 1;
33   return v;
34 }
35
36 /*
37 Suffix links are compressed.
38 This means that:
39   If vertex v has a child by letter x, then:
40     trie[v].nxt[x] points to that child.
41   If vertex v doesn't have such child, then:
42     trie[v].nxt[x] points to the suffix link of that
   ↪ child
43     if we would actually have it.
44 */
45 void add_links(){
46   queue<int> q;
47   q.push(0);
48   while (!q.empty()){
49     auto v = q.front();
50     int u = trie[v].link;
51     q.pop();
52     for (int i = 0; i < S; i++){
53       int& ch = trie[v].nxt[i];
54       if (ch == -1){
55         ch = v? trie[u].nxt[i] : 0;
56       }
57       else{
58         trie[ch].link = v? trie[u].nxt[i] : 0;
59         q.push(ch);
60       }
61     }
62   }
63 }
64
65 bool is_terminal(int v){
66   return trie[v].terminal;
```

```cpp
67  }
68
69  int get_link(int v){
70    return trie[v].link;
71  }
72
73  int go(int v, char c){
74    return trie[v].nxt[ctoi(c)];
75  }
```

Convex Hull Trick

- Allows to insert a linear function to the hull in (1) and get the minimum/maximum value of the stored function at a point in O(log n).
- NOTE: The lines must be added in the order of decreasing/increasing gradients. CAREFULLY CHECK THE SETUP BEFORE USING!
- IMPORTANT: THE DEFAULT VERSION SURELY WORKS. IF MODIFIED VERSIONS DON'T WORK, TRY TRANSFORMING THEM TO THE DEFAULT ONE BY CHANGING SIGNS.

```cpp
struct line{
  ll k, b;
  ll f(ll x){
    return k * x + b;
  };
};

vector<line> hull;

void add_line(line nl){
  if (!hull.empty() && hull.back().k == nl.k){
    nl.b = min(nl.b, hull.back().b); // Default:
    ↪ minimum. For maximum change "min" to "max".
    hull.pop_back();
  }
  while (sz(hull) > 1){
    auto& l1 = hull.end()[-2], l2 = hull.back();
    if ((nl.b - l1.b) * (l2.k - nl.k) >= (nl.b - l2.b) *
    ↪ (l1.k - nl.k)) hull.pop_back(); // Default:
    ↪ decreasing gradient k. For increasing k change the
    ↪ sign to <=.
    else break;
  }
  hull.pb(nl);
}

ll get(ll x){
  int l = 0, r = sz(hull);
  while (r - l > 1){
    int mid = (l + r) / 2;
    if (hull[mid - 1].f(x) >= hull[mid].f(x)) l = mid;
    ↪ // Default: minimum. For maximum change the sign to
    ↪ <=.
    else r = mid;
  }
  return hull[l].f(x);
}
```

Li-Chao Segment Tree

- allows to add linear functions in any order and query minimum/maximum value of those at a point, all in O(log n).
- Clear: clear()

```cpp
const ll INF = 1e18; // Change the constant!
struct LiChaoTree{
  struct line{
    ll k, b;
    line(){
      k = b = 0;
    };
    line(ll k_, ll b_){
      k = k_, b = b_;
    };
    ll f(ll x){
      return k * x + b;
    };
  };
  int n;
  bool minimum, on_points;
  vector<ll> pts;
  vector<line> t;

  void clear(){
    for (auto& l : t) l.k = 0, l.b = minimum? INF :
    ↪ -INF;
  }

  LiChaoTree(int n_, bool min_){ // This is a default
  ↪ constructor for numbers in range [0, n - 1].
    n = n_, minimum = min_, on_points = false;
    t.resize(4 * n);
    clear();
  };

  LiChaoTree(vector<ll> pts_, bool min_){ // This
  ↪ constructor will build LCT on the set of points you
  ↪ pass. The points may be in any order and contain
  ↪ duplicates.
    pts = pts_, minimum = min_;
    sort(all(pts));
    pts.erase(unique(all(pts)), pts.end());
    on_points = true;
    n = sz(pts);
    t.resize(4 * n);
    clear();
  };

  void add_line(int v, int l, int r, line nl){
    // Adding on segment [l, r)
    int m = (l + r) / 2;
    ll lval = on_points? pts[l] : l, mval = on_points?
    ↪ pts[m] : m;
    if ((minimum && nl.f(mval) < t[v].f(mval)) ||
    ↪ (!minimum && nl.f(mval) > t[v].f(mval))) swap(t[v],
    ↪ nl);
    if (r - l == 1) return;
    if ((minimum && nl.f(lval) < t[v].f(lval)) ||
    ↪ (!minimum && nl.f(lval) > t[v].f(lval))) add_line(2
    ↪ * v + 1, l, m, nl);
    else add_line(2 * v + 2, m, r, nl);
  }

  ll get(int v, int l, int r, int x){
    int m = (l + r) / 2;
    if (r - l == 1) return t[v].f(on_points? pts[x] :
    ↪ x);
    else{
      if (minimum) return min(t[v].f(on_points? pts[x] :
      ↪ x), x < m? get(2 * v + 1, l, m, x) : get(2 * v + 2,
      ↪ m, r, x));
      else return max(t[v].f(on_points? pts[x] : x), x <
      ↪ m? get(2 * v + 1, l, m, x) : get(2 * v + 2, m, r,
      ↪ x));
    }
  }

  void add_line(ll k, ll b){
    add_line(0, 0, n, line(k, b));
  }

  ll get(ll x){
    return get(0, 0, n, on_points? lower_bound(all(pts),
    ↪ x) - pts.begin() : x);
  }; // Always pass the actual value of x, even if LCT
  ↪ is on points.
};
```

Persistent Segment Tree

- for RSQ

```cpp
struct Node {
  ll val;
  Node *l, *r;

  Node(ll x) : val(x), l(nullptr), r(nullptr) {}
  Node(Node *ll, Node *rr) {
    l = ll, r = rr;
    val = 0;
    if (l) val += l->val;
    if (r) val += r->val;
  }
```

```
12          Node(Node *cp) : val(cp->val), l(cp->l), r(cp->r) {}
13    };
14    const int N = 2e5 + 20;
15    ll a[N];
16    Node *roots[N];
17    int n, cnt = 1;
18    Node *build(int l = 1, int r = n) {
19        if (l == r) return new Node(a[l]);
20        int mid = (l + r) / 2;
21        return new Node(build(l, mid), build(mid + 1, r));
22    }
23    Node *update(Node *node, int val, int pos, int l = 1,
      ↪  int r = n) {
24        if (l == r) return new Node(val);
25        int mid = (l + r) / 2;
26        if (pos > mid)
27            return new Node(node->l, update(node->r, val,
      ↪  pos, mid + 1, r));
28        else return new Node(update(node->l, val, pos, l,
      ↪  mid), node->r);
29    }
30    ll query(Node *node, int a, int b, int l = 1, int r = n)
      ↪  {
31        if (l > b || r < a) return 0;
32        if (l >= a && r <= b) return node->val;
33        int mid = (l + r) / 2;
34        return query(node->l, a, b, l, mid) + query(node->r,
      ↪  a, b, mid + 1, r);
35    }
```

Miscellaneous

Ordered Set

```
1    #include <ext/pb_ds/assoc_container.hpp>
2    #include <ext/pb_ds/tree_policy.hpp>
3    using namespace __gnu_pbds;
4    typedef tree<int, null_type, less<int>, rb_tree_tag,
      ↪  tree_order_statistics_node_update> ordered_set;
```

Measuring Execution Time

```
1    ld tic = clock();
2    // execute algo…
3    ld tac = clock();
4    // Time in milliseconds
5    cerr << (tac - tic) / CLOCKS_PER_SEC * 1000 << endl;
6    // No need to comment out the print because it's done to
      ↪  cerr.
```

Setting Fixed D.P. Precision

```
1    cout << setprecision(d) << fixed;
2    // Each number is rounded to d digits after the decimal
      ↪  point, and truncated.
```

Common Bugs and General Advice

- Check overflow, array bounds
- Check variable overloading
- Check special cases (n=1?)
- Do something instead of nothing, stay organized
- Write stuff down!
- Don't get stuck on one approach!