超飛上

koosaga, tlwpdus, khsoo01

Compiled on July 25, 2022

Contents

1 Combinatorial Optimization

1.1 Circulation (Push-relabel)

```
// code written by https://loj.ac/u/teapotd

template<class flow_t> struct HLPP {
  struct Edge {
    int to, inv;
    flow_t rem, cap;
  };

  vector<vector<Edge>> G; // Don't use basic_string here, it will fuck up your entire
  life
  vector<flow_t> excess;
  vector<int> hei, arc, prv, nxt, act, bot;
  queue<int> Q;
  int n, high, cut, work;

  // Initialize for n vertices
  HLPP(int k) : G(k) {}

  int addEdge(int u, int v,
        flow_t cap, flow_t rcap = 0) {
    G[u].push_back({ v, sz(G[v]), cap, cap });
    G[v].push_back({ u, sz(G[u])-1, rcap, rcap });
    return sz(G[u])-1;
  }

  void raise(int v, int h) {
    prv[nxt[prv[v]] = nxt[v]] = prv[v];
    hei[v] = h;
    if (excess[v] > 0) {
      bot[v] = act[h]; act[h] = v;
      high = max(high, h);
    }
    if (h < n) cut = max(cut, h+1);
    nxt[v] = nxt[prv[v] = h += n];
    prv[nxt[nxt[h] = v]] = v;
  }

  void global(int s, int t) {
    hei.assign(n, n*2);
    act.assign(n*2, -1);
    iota(all(prv), 0);
    iota(all(nxt), 0);
    hei[t] = high = cut = work = 0;
    hei[s] = n;
    for (int x : {t, s})
      for (Q.push(x); !Q.empty(); Q.pop()) {
        int v = Q.front();
        for(auto &e : G[v]){
          if (hei[e.to] == n*2 &&
            G[e.to][e.inv].rem)
            Q.push(e.to), raise(e.to,hei[v]+1);
        }
      }
  }

  void push(int v, Edge& e, bool z) {
    auto f = min(excess[v], e.rem);
    if (f > 0) {
      if (z && !excess[e.to]) {
        bot[e.to] = act[hei[e.to]];
        act[hei[e.to]] = e.to;
      }
      e.rem -= f; G[e.to][e.inv].rem += f;
      excess[v] -= f; excess[e.to] += f;
    }
  }

  void discharge(int v) {
    int h = n*2, k = hei[v];

    for(int j = 0; j < sz(G[v]); j++){
      auto& e = G[v][arc[v]];
      if (e.rem) {
        if (k == hei[e.to]+1) {
          push(v, e, 1);
          if (excess[v] <= 0) return;
        } else h = min(h, hei[e.to]+1);
      }
      if (++arc[v] >= sz(G[v])) arc[v] = 0;
    }

    if (k < n && nxt[k+n] == prv[k+n]) {
      for(int j = k; j < cut; j++){
      while (nxt[j+n] < n)
        raise(nxt[j+n], n);
      }
      cut = k;
    } else raise(v, h), work++;
  }

  // Compute maximum flow from src to dst
  flow_t flow(int src, int dst) {
    excess.assign(n = sz(G), 0);
    arc.assign(n, 0);
    prv.assign(n*3, 0);
    nxt.assign(n*3, 0);
    bot.assign(n, 0);
    for(auto &e : G[src]){
      excess[src] = e.rem, push(src, e, 0);
    }

    global(src, dst);

    for (; high; high--)
      while (act[high] != -1) {
        int v = act[high];
        act[high] = bot[v];
        if (v != src && hei[v] == high) {
          discharge(v);
          if (work > 4*n) global(src, dst);
        }
      }

    return excess[dst];
  }

  // Get flow through e-th edge of vertex v
  flow_t getFlow(int v, int e) {
    return G[v][e].cap - G[v][e].rem;
  }

  // Get if v belongs to cut component with src
  bool cutSide(int v) { return hei[v] >= n; }
};

template <class T> struct Circulation {
  const T INF = numeric_limits<T>::max() / 2;
  T lowerBoundSum = 0;
  HLPP<T> mf;

  // Initialize for n vertices
  Circulation(int k) : mf(k + 2) {}
  void addEdge(int s, int e, T l, T r){
    mf.addEdge(s + 2, e + 2, r - l);
    if(l > 0){
      mf.addEdge(0, e + 2, l);
      mf.addEdge(s + 2, 1, l);
      lowerBoundSum += l;
    }
    else{
      mf.addEdge(0, s + 2, -l);
      mf.addEdge(e + 2, 1, -l);
      lowerBoundSum += -l;
    }
  }
  bool solve(int s, int e){
 //       mf.addEdge(e+2, s+2, INF); // to reduce as maxflow with lower bounds, in
 circulation problem skip this line
    return lowerBoundSum == mf.flow(0, 1);
    // to get maximum LR flow, run maxflow from s+2 to e+2 again
  }
};
```

1.2 Circulation (Dinic)

```
const int MAXN = 505;
struct edg{ int pos, cap, rev; };
vector<edg> gph[MAXN];
void clear(){ for(int i=0; i<MAXN; i++) gph[i].clear(); }
void add_edge(int s, int e, int x){
    gph[s].push_back({e, x, (int)gph[e].size()});
    gph[e].push_back({s, 0, (int)gph[s].size()-1});
}
int dis[MAXN], pnt[MAXN];
bool bfs(int src, int sink){
    memset(dis, 0, sizeof(dis));
    memset(pnt, 0, sizeof(pnt));
    queue<int> que;
    que.push(src);
    dis[src] = 1;
    while(!que.empty()){
        int x = que.front();
        que.pop();
        for(auto &e : gph[x]){
            if(e.cap > 0 && !dis[e.pos]){
                dis[e.pos] = dis[x] + 1;
                que.push(e.pos);
            }
        }
    }
    return dis[sink] > 0;
}
int dfs(int x, int sink, int f){
    if(x == sink) return f;
    for(; pnt[x] < gph[x].size(); pnt[x]++){
        edg e = gph[x][pnt[x]];
        if(e.cap > 0 && dis[e.pos] == dis[x] + 1){
            int w = dfs(e.pos, sink, min(f, e.cap));
            if(w){
                gph[x][pnt[x]].cap -= w;
                gph[e.pos][e.rev].cap += w;
                return w;
            }
        }
    }
    return 0;
}
lint flow(int src, int sink){
    lint ret = 0;
    while(bfs(src, sink)){
        int r;
        while((r = dfs(src, sink, 2e9))) ret += r;
    }
    return ret;
}
```

1.3 Minimum Cost Circulation (Goldberg-Tarjan)

```
template <class T> struct MinCostCirculation {
  const int SCALE = 3; // scale by 1/(1 << SCALE)
  const T INF = numeric_limits<T>::max() / 2;
  struct EdgeStack {
    int s, e;
    T l, r, cost;
  };
  struct Edge {
    int pos, rev;
    T rem, cap, cost;
  };
  int n;
  vector<EdgeStack> estk;
```

```
Circulation<T> circ;
vector<vector<Edge>> gph;
vector<T> p;
MinCostCirculation(int k) : circ(k), gph(k), p(k) { n = k; }
void addEdge(int s, int e, T l, T r, T cost){
  estk.push_back({s, e, l, r, cost});
}
pair<bool, T> solve(){
  for(auto &i : estk){
    if(i.s != i.e) circ.addEdge(i.s, i.e, i.l, i.r);
  }
  if(!circ.solve(-1, -1)){
    return make_pair(false, T(0));
  }
  vector<int> ptr(n);
  T eps = 0;
  for(auto &i : estk){
    T curFlow;
    if(i.s != i.e) curFlow = i.r - circ.mf.G[i.s + 2][ptr[i.s]].rem;
    else curFlow = i.r;
    int srev = sz(gph[i.e]);
    int erev = sz(gph[i.s]);
    if(i.s == i.e) srev++;
    gph[i.s].push_back({i.e, srev, i.r - curFlow, i.r, i.cost * (n + 1)});
    gph[i.e].push_back({i.s, erev, -i.l + curFlow, -i.l, -i.cost * (n + 1)});
    eps = max(eps, abs(i.cost) * (n + 1));
    if(i.s != i.e){
      ptr[i.s] += 2;
      ptr[i.e] += 2;
    }
  }
  while(true){
    auto cost = [&](Edge &e, int s, int t){
      return e.cost + p[s] - p[t];
    };
    eps = 0;
    for(int i = 0; i < n; i++){
      for(auto &e : gph[i]){
        if(e.rem > 0) eps = max(eps, -cost(e, i, e.pos));
      }
    }
    if(eps <= T(1)) break;
    eps = max(T(1), eps >> SCALE);
    bool upd = 1;
    for(int it = 0; it < 5 && upd; it++){
      upd = false;
      for(int i = 0; i < n; i++){
        for(auto &e : gph[i]){
          if(e.rem > 0 && p[e.pos] > p[i] + e.cost + eps){
            p[e.pos] = p[i] + e.cost + eps;
            upd = true;
          }
        }
      }
      if(!upd) break;
    }
    if(!upd) continue;
    vector<T> excess(n);
    queue<int> que;
    auto push = [&](Edge &e, int src, T flow){
      e.rem -= flow;
      gph[e.pos][e.rev].rem += flow;
      excess[src] -= flow;
      excess[e.pos] += flow;
      if(excess[e.pos] <= flow && excess[e.pos] > 0){
        que.push(e.pos);
      }
    };
    vector<int> ptr(n);
    auto relabel = [&](int v){
      ptr[v] = 0;
      p[v] = -INF;
      for(auto &e : gph[v]){
        if(e.rem > 0){
          p[v] = max(p[v], p[e.pos] - e.cost - eps);
        }
      }
    };
    for(int i = 0; i < n; i++){
      for(auto &j : gph[i]){
        if(j.rem > 0 && cost(j, i, j.pos) < 0){
          push(j, i, j.rem);
        }
      }
    }
    while(sz(que)){
      int x = que.front();
      que.pop();
      while(excess[x] > 0){
        for(; ptr[x] < sz(gph[x]); ptr[x]++){
          Edge &e = gph[x][ptr[x]];
          if(e.rem > 0 && cost(e, x, e.pos) < 0){
            push(e, x, min(e.rem, excess[x]));
            if(excess[x] == 0) break;
          }
        }
        if(excess[x] == 0) break;
        relabel(x);
      }
    }
  }
  T ans = 0;
  for(int i = 0; i < n; i++){
    for(auto &j : gph[i]){
      j.cost /= (n + 1);
      ans += j.cost * (j.cap - j.rem);
    }
  }
  return make_pair(true, ans / 2);
}
void bellmanFord(){
  fill(all(p), T(0));
  bool upd = 1;
  while(upd){
    upd = 0;
    for(int i = 0; i < n; i++){
      for(auto &j : gph[i]){
        if(j.rem > 0 && p[j.pos] > p[i] + j.cost){
          p[j.pos] = p[i] + j.cost;
          upd = 1;
        }
      }
    }
  }
};
```

1.4 Minimum Cost Circulation (Johnson+SSSP)

```
struct mcmf{
  struct edg{ int pos, cap, rev; lint cost; };
  vector<edg> gph[MAXN];
  void clear(){
    for(int i=0; i<MAXN; i++) gph[i].clear();
  }
  void add_edge(int s, int e, int x, lint c){
    gph[s].push_back({e, x, (int)gph[e].size(), c});
    gph[e].push_back({s, 0, (int)gph[s].size()-1, -c});
  }
  lint dist[MAXN];
  int pa[MAXN], pe[MAXN];
  bool inque[MAXN];
  bool spfa(int src, int sink, int n){
    memset(dist, 0x3f, sizeof(dist[0]) * n);
    memset(inque, 0, sizeof(inque[0]) * n);
    queue<int> que;
    dist[src] = 0;
    inque[src] = 1;
    que.push(src);
    bool ok = 0;
    while(!que.empty()){
      int x = que.front();
      que.pop();
      if(x == sink) ok = 1;
      inque[x] = 0;
      for(int i=0; i<gph[x].size(); i++){
        edg e = gph[x][i];
        if(e.cap > 0 && dist[e.pos] > dist[x] + e.cost){
          dist[e.pos] = dist[x] + e.cost;
          pa[e.pos] = x;
          pe[e.pos] = i;
          if(!inque[e.pos]){
            inque[e.pos] = 1;
            que.push(e.pos);
          }
        }
      }
    }
    return ok;
  }
  lint new_dist[MAXN];
  pair<bool, lint> dijkstra(int src, int sink, int n){
    priority_queue<pi, vector<pi>, greater<pi> > pq;
    memset(new_dist, 0x3f, sizeof(new_dist[0]) * n);
    new_dist[src] = 0;
    pq.emplace(0, src);
    bool isSink = 0;
    while(sz(pq)){
      auto tp = pq.top(); pq.pop();
      if(new_dist[tp.second] != tp.first) continue;
      int v = tp.second;
      if(v == sink) isSink = 1;
      for(int i = 0; i < gph[v].size(); i++){
        edg e = gph[v][i];
        lint new_weight = e.cost + dist[v] - dist[e.pos];
        if(e.cap > 0 && new_dist[e.pos] > new_dist[v] + new_weight){
          new_dist[e.pos] = new_dist[v] + new_weight;
          pa[e.pos] = v;
          pe[e.pos] = i;
          pq.emplace(new_dist[e.pos], e.pos);
        }
      }
    }
    return make_pair(isSink, new_dist[sink]);
  }

  lint match(int src, int sink, int n){
    spfa(src, sink, n);
    pair<bool, lint> path;
    lint ret = 0;
    while((path = dijkstra(src, sink, n)).first){
      for(int i = 0; i < n; i++) dist[i] += min(lint(2e15), new_dist[i]);
      lint cap = 1e18;
      for(int pos = sink; pos != src; pos = pa[pos]){
        cap = min(cap, (lint)gph[pa[pos]][pe[pos]].cap);
      }
      ret += cap * (dist[sink] - dist[src]);
      for(int pos = sink; pos != src; pos = pa[pos]){
        int rev = gph[pa[pos]][pe[pos]].rev;
        gph[pa[pos]][pe[pos]].cap -= cap;
        gph[pos][rev].cap += cap;
      }
    }
    return ret;
  }
}mcmf;
```

1.5 Gomory-Hu Tree

```
struct edg{ int s, e, x; };
vector<edg> edgs;
maxflow mf;
void clear(){ edgs.clear(); }
```

```
void add_edge(int s, int e, int x){ edgs.push_back({s, e, x}); }
bool vis[MAXN];
void dfs(int x){
  if(vis[x]) return;
  vis[x] = 1;
  for(auto &i : mf.gph[x]) if(i.cap > 0) dfs(i.pos);
}
vector<pi> solve(int n){ // i - j cut : i - j minimum edge cost. 0 based.
  vector<pi> ret(n); // if i > 0, stores pair(parent,cost)
  for(int i=1; i<n; i++){
    for(auto &j : edgs){
      mf.add_edge(j.s, j.e, j.x);
      mf.add_edge(j.e, j.s, j.x);
    }
    ret[i].first = mf.match(i, ret[i].second);
    memset(vis, 0, sizeof(vis));
    dfs(i);
    for(int j=i+1; j<n; j++){
      if(ret[j].second == ret[i].second && vis[j]){
        ret[j].second = i;
      }
    }
    mf.clear();
  }
  return ret;
}
```

1.6 Bipartite Matching

```
const int MAXN = 100005, MAXM = 100005;
vector<int> gph[MAXN];
int dis[MAXN], l[MAXN], r[MAXM], vis[MAXN];
void clear(){ for(int i=0; i<MAXN; i++) gph[i].clear();  }
void add_edge(int l, int r){ gph[l].push_back(r); }
bool bfs(int n){
  queue<int> que;
  bool ok = 0;
  memset(dis, 0, sizeof(dis));
  for(int i=0; i<n; i++){
    if(l[i] == -1 && !dis[i]){
      que.push(i);
      dis[i] = 1;
    }
  }
  while(!que.empty()){
    int x = que.front();
    que.pop();
    for(auto &i : gph[x]){
      if(r[i] == -1) ok = 1;
      else if(!dis[r[i]]){
        dis[r[i]] = dis[x] + 1;
        que.push(r[i]);
      }
    }
  }
  return ok;
}
bool dfs(int x){
  if(vis[x]) return 0;
  vis[x] = 1;
  for(auto &i : gph[x]){
    if(r[i] == -1 || (!vis[r[i]] && dis[r[i]] == dis[x] + 1 && dfs(r[i]))){
      l[x] = i; r[i] = x;
      return 1;
    }
  }
  return 0;
}
int match(int n){
  memset(l, -1, sizeof(l));
  memset(r, -1, sizeof(r));
  int ret = 0;
  while(bfs(n)){
    memset(vis, 0, sizeof(vis));
    for(int i=0; i<n; i++) if(l[i] == -1 && dfs(i)) ret++;
  }
  return ret;
}
bool chk[MAXN + MAXM];
void rdfs(int x, int n){
  if(chk[x]) return;
  chk[x] = 1;
  for(auto &i : gph[x]){
    chk[i + n] = 1;
    rdfs(r[i], n);
  }
}
vector<int> getcover(int n, int m){ // solve min. vertex cover
  match(n);
  memset(chk, 0, sizeof(chk));
  for(int i=0; i<n; i++) if(l[i] == -1) rdfs(i, n);
  vector<int> v;
  for(int i=0; i<n; i++) if(!chk[i]) v.push_back(i);
  for(int i=n; i<n+m; i++) if(chk[i]) v.push_back(i);
  return v;
}
```

1.7 General Matching (Short)

```
const int MAXN = 2020 + 1;
// 1-based Vertex index
int vis[MAXN], par[MAXN], orig[MAXN], match[MAXN], aux[MAXN], t, N;
vector<int> conn[MAXN];
queue<int> Q;
void addEdge(int u, int v) {
  conn[u].push_back(v); conn[v].push_back(u);
}
void init(int n) {
  N = n; t = 0;
  for(int i=0; i<=n; ++i) {
    conn[i].clear();
    match[i] = aux[i] = par[i] = 0;
  }
}
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 v, int w) {
  ++t;
  while(true) {
    if(v) {
      if(aux[v] == t) return v; aux[v] = t;
      v = orig[par[match[v]]];
    }
    swap(v, w);
  }
}
void blossom(int v, int w, int a) {
  while(orig[v] != a) {
    par[v] = w; w = match[v];
    if(vis[w] == 1) Q.push(w), vis[w] = 0;
    orig[v] = orig[w] = a;
    v = par[w];
  }
}
bool bfs(int u) {
  fill(vis+1, vis+1+N, -1); iota(orig + 1, orig + N + 1, 1);
  Q = queue<int> (); Q.push(u); vis[u] = 0;
  while(!Q.empty()) {
    int v = Q.front(); Q.pop();
    for(int x: conn[v]) {
      if(vis[x] == -1) {
        par[x] = v; vis[x] = 1;
        if(!match[x]) return augment(u, x), true;
        Q.push(match[x]); vis[match[x]] = 0;
      }
      else if(vis[x] == 0 && orig[v] != orig[x]) {
        int a = lca(orig[v], orig[x]);
        blossom(x, v, a); blossom(v, x, a);
      }
    }
  }
  return false;
}
int Match() {
  int ans = 0;
  // find random matching (not necessary, constant improvement)
  vector<int> V(N-1); iota(V.begin(), V.end(), 1);
  shuffle(V.begin(), V.end(), mt19937(0x94949));
  for(auto x: V) if(!match[x]){
    for(auto y: conn[x]) if(!match[y]) {
      match[x] = y, match[y] = x;
      ++ans; break;
    }
  }
  for(int i=1; i<=N; ++i) if(!match[i] && bfs(i)) ++ans;
  return ans;
}
```

1.8 General Matching (Fucking)

```
class MaximumMatching {
  /*
    Maximum Cardinality Matching in General Graphs.
    - O(\sqrt{n} m \log_{\max\{2, 1 + m/n\}} n) time
    - O(n + m) space
    Note: each vertex is 1-indexed.
    Ref:
      Harold N. Gabow,
      "The Weighted Matching Approach to Maximum Cardinality Matching" (2017)
      (https://arxiv.org/abs/1703.03998)
  */
public:
  struct Edge { int from, to; };
  static constexpr int Inf = 1 << 30;

private:
  enum Label {
    kInner = -1, // should be < 0
    kFree = 0    // should be 0
  };
  struct Link { int from, to; };
  struct Log { int v, par; };

  struct LinkedList {
    LinkedList() {}
    LinkedList(int N, int M) : N(N), next(M) { clear(); }
    void clear() { head.assign(N, -1); }
    void push(int h, int u) { next[u] = head[h], head[h] = u; }
    int N;
    vector<int> head, next;
  };

  template <typename T>
  struct Queue {
    Queue() {}
    Queue(int N) : qh(0), qt(0), data(N) {}
    T operator [] (int i) const { return data[i]; }
    void enqueue(int u) { data[qt++] = u; }
    int dequeue() { return data[qh++]; }
    bool empty() const { return qh == qt; }
    void clear() { qh = qt = 0; }
    int size() const { return qt; }
    int qh, qt;
    vector<T> data;
  };

  struct DisjointSetUnion {
    DisjointSetUnion() {}
```

```
    DisjointSetUnion(int N) : par(N) {
      for (int i = 0; i < N; ++i) par[i] = i;
    }
    int find(int u) { return par[u] == u ? u : (par[u] = find(par[u])); }
    void unite(int u, int v) {
      u = find(u), v = find(v);
      if (u != v) par[v] = u;
    }
    vector<int> par;
  };

public:
  MaximumMatching(int N, const vector<Edge>& in)
      : N(N), NH(N >> 1), ofs(N + 2, 0), edges(in.size() * 2) {

    for (auto& e : in) ofs[e.from + 1] += 1, ofs[e.to + 1] += 1;
    for (int i = 1; i <= N + 1; ++i) ofs[i] += ofs[i - 1];
    for (auto& e : in) {
      edges[ofs[e.from]++] = e; edges[ofs[e.to]++] = {e.to, e.from};
    }
    for (int i = N + 1; i > 0; --i) ofs[i] = ofs[i - 1];
    ofs[0] = 0;
  }

  int maximum_matching() {
    initialize();
    int match = 0;
    while (match * 2 + 1 < N) {
      reset_count();
      bool has_augmenting_path = do_edmonds_search();
      if (!has_augmenting_path) break;
      match += find_maximal();
      clear();
    }
    return match;
  }

private:
  void reset_count() {
    time_current_ = 0; time_augment_ = Inf;
    contract_count_ = 0; outer_id_ = 1;
    dsu_changelog_size_ = dsu_changelog_last_ = 0;
  }

  void clear() {
    que.clear();
    for (int u = 1; u <= N; ++u) potential[u] = 1;
    for (int u = 1; u <= N; ++u) dsu.par[u] = u;
    for (int t = time_current_; t <= N / 2; ++t) list.head[t] = -1;
    for (int u = 1; u <= N; ++u) blossom.head[u] = -1;
  }

  // first phase

  inline void grow(int x, int y, int z) {
    label[y] = kInner;
    potential[y] = time_current_; // visited time
    link[z] = {x, y}; label[z] = label[x];
    potential[z] = time_current_ + 1;
    que.enqueue(z);
  }

  void contract(int x, int y) {
    int bx = dsu.find(x), by = dsu.find(y);
    const int h = -(++contract_count_) + kInner;
    label[mate[bx]] = label[mate[by]] = h;
    int lca = -1;
    while (1) {
      if (mate[by] != 0) swap(bx, by);
      bx = lca = dsu.find(link[bx].from);
      if (label[mate[bx]] == h) break;
      label[mate[bx]] = h;
    }
    for (auto bv : {dsu.par[x], dsu.par[y]}) {
      for (; bv != lca; bv = dsu.par[link[bv].from]) {
        int mv = mate[bv];
        link[mv] = {x, y}; label[mv] = label[x];
        potential[mv] = 1 + (time_current_ - potential[mv]) + time_current_;
        que.enqueue(mv);
        dsu.par[bv] = dsu.par[mv] = lca;
        dsu_changelog[dsu_changelog_last_++] = {bv, lca};
        dsu_changelog[dsu_changelog_last_++] = {mv, lca};
      }
    }
  }

  bool find_augmenting_path() {
    while (!que.empty()) {
      int x = que.dequeue(), lx = label[x], px = potential[x], bx = dsu.find(x);
      for (int eid = ofs[x]; eid < ofs[x + 1]; ++eid) {
        int y = edges[eid].to;
        if (label[y] > 0) { // outer blossom/vertex
          int time_next = (px + potential[y]) >> 1;
          if (lx != label[y]) {
            if (time_next == time_current_) return true;
            time_augment_ = min(time_next, time_augment_);
          } else {
            if (bx == dsu.find(y)) continue;
            if (time_next == time_current_) {
              contract(x, y); bx = dsu.find(x);
            } else if (time_next <= NH) list.push(time_next, eid);
          }
        } else if (label[y] == kFree) { // free vertex
          int time_next = px + 1;
          if (time_next == time_current_) grow(x, y, mate[y]);
          else if (time_next <= NH) list.push(time_next, eid);
        }
      }
    }
    return false;
  }

  bool adjust_dual_variables() {
    // Return true if the current matching is maximum.
    const int time_lim = min(NH + 1, time_augment_);
    for (++time_current_; time_current_ <= time_lim; ++time_current_) {
      dsu_changelog_size_ = dsu_changelog_last_;
      if (time_current_ == time_lim) break;
      bool updated = false;
      for (int h = list.head[time_current_]; h >= 0; h = list.next[h]) {
        auto& e = edges[h]; int x = e.from, y = e.to;
        if (label[y] > 0) {
          // Case: outer -- (free => inner => outer)
          if (potential[x] + potential[y] != (time_current_ << 1)) continue;
          if (dsu.find(x) == dsu.find(y)) continue;
          if (label[x] != label[y]) { time_augment_ = time_current_; return false; }
          contract(x, y); updated = true;
        } else if (label[y] == kFree) {
          grow(x, y, mate[y]); updated = true;
        }
      }
      list.head[time_current_] = -1;
      if (updated) return false;
    }
    return time_current_ > NH;
  }

  bool do_edmonds_search() {
    label[0] = kFree;
    for (int u = 1; u <= N; ++u) {
      if (mate[u] == 0) {
        que.enqueue(u); label[u] = u; // component id
      } else label[u] = kFree;
    }
    while (1) {
      if (find_augmenting_path()) break;
      bool maximum = adjust_dual_variables();
      if (maximum) return false;
      if (time_current_ == time_augment_) break;
    }
    for (int u = 1; u <= N; ++u) {
      if (label[u] > 0) potential[u] -= time_current_;
      else if (label[u] < 0) potential[u] = 1 + (time_current_ - potential[u]);
    }
    return true;
  }

  // second phase

  void rematch(int v, int w) {
    int t = mate[v]; mate[v] = w;
    if (mate[t] != v) return;
    if (link[v].to == dsu.find(link[v].to)) {
      mate[t] = link[v].from;
      rematch(mate[t], t);
    } else {
      int x = link[v].from, y = link[v].to;
      rematch(x, y); rematch(y, x);
    }
  }

  bool dfs_augment(int x, int bx) {
    int px = potential[x], lx = label[bx];
    for (int eid = ofs[x]; eid < ofs[x + 1]; ++eid) {
      int y = edges[eid].to;
      if (px + potential[y] != 0) continue;
      int by = dsu.find(y), ly = label[by];
      if (ly > 0) { // outer
        if (lx >= ly) continue;
        int stack_beg = stack_last_;
        for (int bv = by; bv != bx; bv = dsu.find(link[bv].from)) {
          int bw = dsu.find(mate[bv]);
          stack[stack_last_++] = bw; link[bw] = {x, y};
          dsu.par[bv] = dsu.par[bw] = bx;
        }
        while (stack_last_ > stack_beg) {
          int bv = stack[--stack_last_];
          for (int v = blossom.head[bv]; v >= 0; v = blossom.next[v]) {
            if (!dfs_augment(v, bx)) continue;
            stack_last_ = stack_beg;
            return true;
          }
        }
      } else if (ly == kFree) {
        label[by] = kInner; int z = mate[by];
        if (z == 0) { rematch(x, y); rematch(y, x); return true; }
        int bz = dsu.find(z);
        link[bz] = {x, y}; label[bz] = outer_id_++;
        for (int v = blossom.head[bz]; v >= 0; v = blossom.next[v]) {
          if (dfs_augment(v, bz)) return true;
        }
      }
    }
    return false;
  }

  int find_maximal() {
    // discard blossoms whose potential is 0.
    for (int u = 1; u <= N; ++u) dsu.par[u] = u;
    for (int i = 0; i < dsu_changelog_size_; ++i) {
      dsu.par[dsu_changelog[i].v] = dsu_changelog[i].par;
    }
    for (int u = 1; u <= N; ++u) {
      label[u] = kFree; blossom.push(dsu.find(u), u);
    }
    int ret = 0;
    for (int u = 1; u <= N; ++u) if (!mate[u]) {
      int bu = dsu.par[u];
      if (label[bu] != kFree) continue;
      label[bu] = outer_id_++;
      for (int v = blossom.head[bu]; v >= 0; v = blossom.next[v]) {
```

```
        if (!dfs_augment(v, bu)) continue;
        ret += 1;
        break;
      }
    }
    assert(ret >= 1);
    return ret;
  }

  // init

  void initialize() {
    que = Queue<int>(N);

    mate.assign(N + 1, 0);
    potential.assign(N + 1, 1);
    label.assign(N + 1, kFree);
    link.assign(N + 1, {0, 0});

    dsu_changelog.resize(N);

    dsu = DisjointSetUnion(N + 1);
    list = LinkedList(NH + 1, edges.size());

    blossom = LinkedList(N + 1, N + 1);
    stack.resize(N); stack_last_ = 0;
  }

private:
  const int N, NH;
  vector<int> ofs; vector<Edge> edges;

  Queue<int> que;

  vector<int> mate, potential;
  vector<int> label; vector<Link> link;

  vector<Log> dsu_changelog; int dsu_changelog_last_, dsu_changelog_size_;

  DisjointSetUnion dsu;
  LinkedList list, blossom;
  vector<int> stack; int stack_last_;

  int time_current_, time_augment_;
  int contract_count_, outer_id_;
};

using Edge = MaximumMatching::Edge;
```

1.9 General Weighted Matching

```
// N^3 (but fast in practice)
static const int INF = INT_MAX;
static const int N = 514;
struct edge{
  int u,v,w; edge(){}
  edge(int ui,int vi,int wi)
    :u(ui),v(vi),w(wi){}
};
int n,n_x;
edge g[N*2][N*2];
int lab[N*2];
int match[N*2],slack[N*2],st[N*2],pa[N*2];
int flo_from[N*2][N+1],S[N*2],vis[N*2];
vector<int> flo[N*2];
queue<int> q;
int e_delta(const edge &e){
  return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
}
void update_slack(int u,int x){
  if(!slack[x]||e_delta(g[u][x]) < e_delta(g[slack[x]][x])){
  slack[x]=u;
  }
}
void set_slack(int x){
  slack[x]=0;
  for(int u=1;u<=n;++u)
    if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
      update_slack(u,x);
}
void q_push(int x){
  if(x<=n)q.push(x);
  else for(size_t i=0;i<flo[x].size();i++)
    q_push(flo[x][i]);
}
void set_st(int x,int b){
  st[x]=b;
  if(x>n)for(size_t i=0;i<flo[x].size();++i)
    set_st(flo[x][i],b);
}
int get_pr(int b,int xr){
  int pr= find(flo[b].begin(),flo[b].end(),xr) - flo[b].begin();
  if(pr%2==1){
    reverse(flo[b].begin()+1,flo[b].end());
    return (int)flo[b].size()-pr;
  }else return pr;
}
void set_match(int u,int v){
  match[u]=g[u][v].v;
  if(u<=n) return;
  edge e=g[u][v];
  int xr=flo_from[u][e.u],pr=get_pr(u,xr);
  for(int i=0;i<pr;++i) set_match(flo[u][i],flo[u][i^1]);
  set_match(xr,v);
  rotate(flo[u].begin(), flo[u].begin()+pr, flo[u].end());
}
void augment(int u,int v){
  for(;;){
    int xnv=st[match[u]];
    set_match(u,v);
    if(!xnv)return;
    set_match(xnv,st[pa[xnv]]);
    u=st[pa[xnv]],v=xnv;
  }
}
int get_lca(int u,int v){
  static int t=0;
  for(++t;u||v;swap(u,v)){
    if(u==0)continue;
    if(vis[u]==t)return u;
    vis[u]=t;
    u=st[match[u]];
    if(u)u=st[pa[u]];
  }
  return 0;
}
void add_blossom(int u,int lca,int v){
  int b=n+1;
  while(b<=n_x&&st[b])++b;
  if(b>n_x)++n_x;
  lab[b]=0,S[b]=0;
  match[b]=match[lca];
  flo[b].clear();
  flo[b].push_back(lca);
  for(int x=u,y;x!=lca;x=st[pa[y]])
    flo[b].push_back(x), flo[b].push_back(y=st[match[x]]), q_push(y);
  reverse(flo[b].begin()+1,flo[b].end());
  for(int x=v,y;x!=lca;x=st[pa[y]])
    flo[b].push_back(x), flo[b].push_back(y=st[match[x]]), q_push(y);
  set_st(b,b);
  for(int x=1;x<=n_x;++x)g[b][x].w=g[x][b].w=0;
  for(int x=1;x<=n;++x)flo_from[b][x]=0;
  for(size_t i=0;i<flo[b].size();++i){
    int xs=flo[b][i];
    for(int x=1;x<=n_x;++x)
      if(g[b][x].w==0 || e_delta(g[xs][x])<e_delta(g[b][x]))
        g[b][x]=g[xs][x], g[x][b]=g[x][xs];
    for(int x=1;x<=n;++x)
      if(flo_from[xs][x]) flo_from[b][x]=xs;
  }
  set_slack(b);
}
void expand_blossom(int b){
  for(size_t i=0;i<flo[b].size();++i)
    set_st(flo[b][i],flo[b][i]);
  int xr=flo_from[b][g[b][pa[b]].u], pr=get_pr(b,xr);
  for(int i=0;i<pr;i+=2){
    int xs=flo[b][i],xns=flo[b][i+1];
    pa[xs]=g[xns][xs].u;
    S[xs]=1,S[xns]=0;
    slack[xs]=0,set_slack(xns);
    q_push(xns);
  }
  S[xr]=1,pa[xr]=pa[b];
  for(size_t i=pr+1;i<flo[b].size();++i){
    int xs=flo[b][i];
    S[xs]=-1,set_slack(xs);
  }
  st[b]=0;
}
bool on_found_edge(const edge &e){
  int u=st[e.u],v=st[e.v];
  if(S[v]==-1){
    pa[v]=e.u,S[v]=1;
    int nu=st[match[v]];
    slack[v]=slack[nu]=0;
    S[nu]=0,q_push(nu);
  }else if(S[v]==0){
    int lca=get_lca(u,v);
    if(!lca)return augment(u,v),augment(v,u),true;
    else add_blossom(u,lca,v);
  }
  return false;
}
bool matching(){
  memset(S+1,-1,sizeof(int)*n_x);
  memset(slack+1,0,sizeof(int)*n_x);
  q=queue<int>();
  for(int x=1;x<=n_x;++x)
    if(st[x]==x&&!match[x]) pa[x]=0,S[x]=0,q_push(x);
  if(q.empty())return false;
  for(;;){
    while(q.size()){
      int u=q.front();q.pop();
      if(S[st[u]]==1)continue;
      for(int v=1;v<=n;++v)
        if(g[u][v].w>0&&st[u]!=st[v]){
          if(e_delta(g[u][v])==0){
            if(on_found_edge(g[u][v])) return true;
          }else update_slack(u,st[v]);
        }
    }
    int d=INF;
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b&&S[b]==1) d=min(d,lab[b]/2);
    for(int x=1;x<=n_x;++x)
      if(st[x]==x&&slack[x]){
        if(S[x]==-1) d=min(d,e_delta(g[slack[x]][x]));
        else if(S[x]==0) d=min(d,e_delta(g[slack[x]][x])/2);
      }
    for(int u=1;u<=n;++u){
      if(S[st[u]]==0){
        if(lab[u]<=d)return 0;
        lab[u]-=d;
      }else if(S[st[u]]==1)lab[u]+=d;
    }
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b){
        if(S[st[b]]==0) lab[b]+=d*2;
        else if(S[st[b]]==1) lab[b]-=d*2;
      }
    q=queue<int>();
```

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

1.10 Simplex Algorithm

```
using T = long double;
const int N = 410, M = 30010;
const T eps = 1e-7;
int n, m;
int Left[M], Down[N];
// time complexity: exponential. fast $O(MN^2)$ in experiment. dependent on the
modeling.
// Ax <= b, max c^T x. 최댓값: v, 답 추적: sol[i]. 1 based
T a[M][N], b[M], c[N], v, sol[N];
bool eq(T a, T b) { return fabs(a - b) < eps;  }
bool ls(T a, T b) { return a < b && !eq(a, b); }
void init(int p, int q) {
  n = p; m = q; v = 0;
  for(int i = 1; i <= m; i++){
    for(int j = 1; j <= n; j++) a[i][j]=0;
  }
  for(int i = 1; i <= m; i++) b[i]=0;
  for(int i = 1; i <= n; i++) c[i]=sol[i]=0;
}
void pivot(int x,int y) {
  swap(Left[x], Down[y]);
  T k = a[x][y]; a[x][y] = 1;
  vector<int> nz;
  for(int i = 1; i <= n; i++){
    a[x][i] /= k;
    if(!eq(a[x][i], 0)) nz.push_back(i);
  }
  b[x] /= k;

  for(int i = 1; i <= m; i++){
    if(i == x || eq(a[i][y], 0)) continue;
    k = a[i][y]; a[i][y] = 0;
    b[i] -= k*b[x];
    for(int j : nz) a[i][j] -= k*a[x][j];
  }
  if(eq(c[y], 0)) return;
  k = c[y]; c[y] = 0;
  v += k*b[x];
  for(int i : nz) c[i] -= k*a[x][i];
}
// 0: found solution, 1: no feasible solution, 2: unbounded
int solve() {
  for(int i = 1; i <= n; i++) Down[i] = i;
  for(int i = 1; i <= m; i++) Left[i] = n+i;
  while(1) { // Eliminating negative b[i]
    int x = 0, y = 0;
    for(int i = 1; i <= m; i++) if (ls(b[i], 0) && (x == 0 || b[i] < b[x])) x = i;
    if(x == 0) break;
    for(int i = 1; i <= n; i++) if (ls(a[x][i], 0) && (y == 0 || a[x][i] < a[x][y]))
    y = i;
    if(y == 0) return 1;
    pivot(x, y);
  }
  while(1) {
    int x = 0, y = 0;
    for(int i = 1; i <= n; i++)
      if (ls(0, c[i]) && (!y || c[i] > c[y])) y = i;
    if(y == 0) break;
    for(int i = 1; i <= m; i++)
      if (ls(0, a[i][y]) && (!x || b[i]/a[i][y] < b[x]/a[x][y])) x = i;
    if(x == 0) return 2;
    pivot(x, y);
  }
  for(int i = 1; i <= m; i++) if(Left[i] <= n) sol[Left[i]] = b[i];
  return 0;
}
```

2 Graph Theory

2.1 BCC

```
void color(int x, int p){
  if(p){
    bcc[p].push_back(x);
    cmp[x].push_back(p);
  }
  for(auto &i : gph[x]){
    if(cmp[i].size()) continue;
    if(low[i] >= dfn[x]){
      bcc[++c].push_back(x);
      cmp[x].push_back(c);
      color(i, c);
    }
    else color(i, p);
  }
}
```

2.2 SCC, 2-SAT

```
struct strongly_connected{
  vector<vector<int>> gph;

  void init(int n){
    gph.clear();
    gph.resize(n);
  }

  void add_edge(int s, int e){
    gph[s].push_back(e);
  }

  vector<int> val, comp, z, cont;
  int Time, ncomps;
  template<class G, class F> int dfs(int j, G& g, F f) {
    int low = val[j] = ++Time, x; z.push_back(j);
    for(auto e : g[j]) if (comp[e] < 0)
      low = min(low, val[e] ?: dfs(e,g,f));

    if (low == val[j]) {
      do {
        x = z.back(); z.pop_back();
        comp[x] = ncomps;
        cont.push_back(x);
      } while (x != j);
      f(cont); cont.clear();
      ncomps++;
    }
    return val[j] = low;
  }
  template<class G, class F> void scc(G& g, F f) {
    int n = sz(g);
    val.assign(n, 0); comp.assign(n, -1);
    Time = ncomps = 0;
    for(int i=0; i<n; i++) if (comp[i] < 0) dfs(i, g, f);
  }

  int piv;
  void get_scc(int n){
    scc(gph, [&](vector<int> &v){});
    for(int i=0; i<n; i++){
      comp[i] = ncomps - comp[i];
    }
    piv = ncomps;
  }
}scc;

struct twosat{
  strongly_connected scc;
  int n;
  void init(int _n){ scc.init(2 * _n); n = _n; }
  int NOT(int x){ return x >= n ? (x - n) : (x + n); }
  void add_edge(int x, int y){
    if((x >> 31) & 1) x = (~x) + n;
    if((y >> 31) & 1) y = (~y) + n;
    scc.add_edge(x, y), scc.add_edge(NOT(y), NOT(x));
  }
  bool satisfy(int *res){
    scc.get_scc(2*n);
    for(int i=0; i<n; i++){
      if(scc.comp[i] == scc.comp[NOT(i)]) return 0;
      if(scc.comp[i] < scc.comp[NOT(i)]) res[i] = 0;
      else res[i] = 1;
    }
    return 1;
  }
}twosat;
```

2.3 Bipolar Orientation

```
// Given a 2-connected graph, compute an acyclic ordering such that $s$ is the only
0-indegree vertex and $t$ is the only 0-outdegree vertex.
namespace STOrder{
  int n;
  vector<pi> gph[MAXN];
  vector<pi> backedg[MAXN];
  int par[MAXN], ord[MAXN], stk[MAXN], piv;

  void dfs(int x, int p){
    stk[x] = 1;
    ord[x] = ++piv;
    for(auto &i : gph[x]){
      if(i.first == p) continue;
      if(!ord[i.second]){
        par[i.second] = x;
        dfs(i.second, i.first);
      }
      else if(stk[i.second]){
        backedg[ord[i.second]].emplace_back(x, i.second);
      }
    }
    stk[x] = 0;
  }

  void clear(){
    for(int i=1; i<=n; i++){
      gph[i].clear();
      backedg[i].clear();
```

```
      ord[i] = 0;
    }
    piv = 0;
  }
  vector<int> solve(int _n, int s, int t, vector<pi> E){
    n = _n;
    gph[s].emplace_back(sz(E), t);
    gph[t].emplace_back(sz(E), s);
    for(int i=0; i<sz(E); i++){
      gph[E[i].first].emplace_back(i, E[i].second);
      gph[E[i].second].emplace_back(i, E[i].first);
    }
    dfs(s, -1);
    if(piv != n) return {};
    vector<vector<int>> ears;
    vector<int> mark(n + 1);
    for(int i=1; i<=n; i++){
      for(auto &j : backedg[i]){
        vector<int> v = {j.second, j.first};
        for(int k=j.first; k!=j.second; k=par[k]){
          if(mark[k]) break;
          mark[k] = 1;
          v.push_back(par[k]);
        }
        ears.push_back(v);
      }
    }
    if(sz(ears) == 0 || ears[0].front() != ears[0].back()) return {};
    for(int i=1; i<sz(ears); i++){
      if(ears[i].front() == ears[i].back()) return {};
    }
    for(int i=1; i<=n; i++){
      if(i != s && !mark[i]) return {};
    }
    vector<int> dp(n + 1);
    vector<pi> intv(n + 1);
    for(int i=sz(ears)-1; i>=1; i--){
      for(int j=1; j+1<sz(ears[i]); j++){
        dp[ears[i][0]] += 1 + dp[ears[i][j]];
      }
    }
    for(int j=0; j+1<sz(ears[0]); j++){
      intv[ears[0][j]].second = intv[ears[0][j]].first + dp[ears[0][j]] + 1;
      if(j + 2 < sz(ears[0])) intv[ears[0][j+1]].first = intv[ears[0][j]].second;
    }
    for(int i=1; i<sz(ears); i++){
      if(intv[ears[i][0]] < intv[ears[i].back()]){
        pi curIntv = intv[ears[i][0]];
        for(int j=sz(ears[i])-2; j>=1; j--){
          intv[ears[i][j]] = pi(curIntv.second - dp[ears[i][j]] - 1, curIntv.second);
          curIntv.second -= dp[ears[i][j]] + 1;
        }
        intv[ears[i][0]] = curIntv;
      }
      else{
        pi curIntv = intv[ears[i][0]];
        for(int j=sz(ears[i])-2; j>=1; j--){
          intv[ears[i][j]] = pi(curIntv.first, curIntv.first + dp[ears[i][j]] + 1);
          curIntv.first += dp[ears[i][j]] + 1;
        }
        intv[ears[i][0]] = curIntv;
      }
    }
    vector<int> dap(n);
    for(int i=1; i<=n; i++){
      assert(intv[i].first + 1 == intv[i].second);
      dap[intv[i].first] = i;
    }
    return dap;
  }
}
```

2.4 Directed MST

```
#include <bits/stdc++.h>
#define sz(v) ((int)(v).size())
#define all(v) (v).begin(), (v).end()
using namespace std;
using lint = long long;
using pi = pair<lint, lint>;

mt19937 rng(0x69420);
int randint(int lb, int ub){ return uniform_int_distribution<int>(lb, ub)(rng); }

struct edge{
  int s, e;
  lint x;
  bool operator>(const edge &e)const{
    return x > e.x;
  }
};

namespace dmst{
  struct node{
    node *l, *r;
    edge val;
    lint lazy;
    void add(lint v){
      val.x += v;
      lazy += v;
    }
    void push(){
      if(l) l->add(lazy);
      if(r) r->add(lazy);
      lazy = 0;
    }
    node(){
      l = r = NULL;
      lazy = 0;
    }
    node(edge e){
      l = r = NULL;
      val = e;
      lazy = 0;
    }
  };

  node* merge(node *x, node *y){
    if(!x) return y;
    if(!y) return x;
    x->push();
    y->push();
    if(x->val.x > y->val.x) swap(x, y);
    if(randint(0, 1)) x->l = merge(x->l, y);
    else x->r = merge(x->r, y);
    return x;
  }
  edge top(node *x){
    return x->val;
  }
  node *pop(node *x){
    x->push();
    return merge(x->l, x->r);
  }

  struct disj{
    vector<int> pa, rk, mx;
    vector<pair<int*, int>> event;
    void init(int n){
      event.clear();
      pa.resize(n + 1);
      rk.resize(n + 1);
      mx.resize(n + 1);
      iota(all(pa), 0);
      iota(all(mx), 0);
      fill(all(rk), 0);
    }
    int time(){ return sz(event); }
    int find(int x){
      return pa[x] == x ? x : find(pa[x]);
    }
    bool uni(int p, int q){
      p = find(p);
      q = find(q);
      if(p == q) return 0;
      if(rk[p] < rk[q]) swap(p, q);
      event.emplace_back(&pa[q], pa[q]);
      event.emplace_back(&mx[p], mx[p]);
      pa[q] = p;
      mx[p] = max(mx[p], mx[q]);
      if(rk[p] == rk[q]){
        event.emplace_back(&rk[p], rk[p]);
        rk[p]++;
      }
      return 1;
    }
    void rollback(int t){
      while(sz(event) > t){
        *event.back().first = event.back().second;
        event.pop_back();
      }
    }
    int getidx(int x){ return mx[find(x)]; }
  };
  vector<edge> solve(int n, int r, vector<edge> e){
    vector<edge> parent(n);
    vector<node*> gph(n);
    for(auto &i : e){
      gph[i.e] = merge(gph[i.e], new node(i));
    }
    disj dsu1, dsu2;
    dsu1.init(n*2);
    dsu2.init(n*2);
    vector<int> when;
    auto isLoop = [&](edge e){
      return dsu2.find(e.s) == dsu2.find(e.e);
    };
    int auxNode = n;
    for(int x = 0; x < auxNode; x++){
      if(x == r) continue;
      while(isLoop(top(gph[x]))) gph[x] = pop(gph[x]);
      parent[x] = top(gph[x]);
      gph[x] = pop(gph[x]);
      if(!dsu1.uni(x, parent[x].s)){
        vector<int> cycle = {x};
        for(int i = dsu2.getidx(parent[x].s); i != x; i = dsu2.getidx(parent[i].s)){
          cycle.push_back(i);
        }
        node* merged = NULL;
        when.push_back(dsu2.time());
        for(auto &i : cycle){
          dsu2.uni(i, auxNode);
          if(gph[i] != NULL){
            gph[i]->add(-parent[i].x);
            merged = merge(merged, gph[i]);
          }
        }
        gph.push_back(merged);
        parent.resize(auxNode + 1);
        dsu1.uni(x, auxNode);
        auxNode++;
      }
    }
    for(int i = auxNode - 1; i >= n; i--){
      dsu2.rollback(when.back());
      when.pop_back();
      int target = dsu2.getidx(parent[i].e);
      parent[i].x += parent[target].x;
      parent[target] = parent[i];
    }
    parent.resize(n);
    parent[r].x = 0;
```

```
    parent[r].s = r;
    return parent;
  }
};
```

2.5 Dominator Tree

```
vector<int> E[MAXN], RE[MAXN], rdom[MAXN];

int S[MAXN], RS[MAXN], cs;
int par[MAXN], val[MAXN], sdom[MAXN], rp[MAXN], dom[MAXN];

void clear(int n) {
  cs = 0;
  for(int i=0;i<=n;i++) {
    par[i] = val[i] = sdom[i] = rp[i] = dom[i] = S[i] = RS[i] = 0;
    E[i].clear(); RE[i].clear(); rdom[i].clear();
  }
}
void add_edge(int x, int y) { E[x].push_back(y); }
void Union(int x, int y) { par[x] = y; }
int Find(int x, int c = 0) {
  if(par[x] == x) return c ? -1 : x;
  int p = Find(par[x], 1);
  if(p == -1) return c ? par[x] : val[x];
  if(sdom[val[x]] > sdom[val[par[x]]]) val[x] = val[par[x]];
  par[x] = p;
  return c ? p : val[x];
}
void dfs(int x) {
  RS[ S[x] = ++cs ] = x;
  par[cs] = sdom[cs] = val[cs] = cs;
  for(int e : E[x]) {
    if(S[e] == 0) dfs(e), rp[S[e]] = S[x];
    RE[S[e]].push_back(S[x]);
  }
}
int solve(int s, int *up) { // Calculate idoms
  dfs(s);
  for(int i=cs;i;i--) {
    for(int e : RE[i]) sdom[i] = min(sdom[i], sdom[Find(e)]);
    if(i > 1) rdom[sdom[i]].push_back(i);
    for(int e : rdom[i]) {
      int p = Find(e);
      if(sdom[p] == i) dom[e] = i;
      else dom[e] = p;
    }
    if(i > 1) Union(i, rp[i]);
  }
  for(int i=2;i<=cs;i++) if(sdom[i] != dom[i]) dom[i] = dom[dom[i]];
  for(int i=2;i<=cs;i++) up[RS[i]] = RS[dom[i]];
  return cs;
}
```

2.6 Dynamic MST Offline

```
struct disj{
  int pa[MAXN], rk[MAXN];
  void init(int n){
    iota(pa, pa + n + 1, 0);
    memset(rk, 0, sizeof(rk));
  }
  int find(int x){
    return pa[x] == x ? x : find(pa[x]);
  }
  bool uni(int p, int q, vector<pi> &snapshot){
    p = find(p);
    q = find(q);
    if(p == q) return 0;
    if(rk[p] < rk[q]) swap(p, q);
    snapshot.push_back({q, pa[q]});
    pa[q] = p;
    if(rk[p] == rk[q]){
      snapshot.push_back({p, -1});
      rk[p]++;
    }
    return 1;
  }
  void revert(vector<pi> &snapshot){
    reverse(snapshot.begin(), snapshot.end());
    for(auto &x : snapshot){
      if(x.second < 0) rk[x.first]--;
      else pa[x.first] = x.second;
    }
    snapshot.clear();
  }
}disj;

int n, m, q;
int st[MAXN], ed[MAXN], cost[MAXN], chk[MAXN];
pi qr[MAXN];

bool cmp(int &a, int &b){ return pi(cost[a], a) < pi(cost[b], b); }

void contract(int s, int e, vector<int> v, vector<int> &must_mst, vector<int>
&maybe_mst){
  sort(v.begin(), v.end(), cmp);
  vector<pi> snapshot;
  for(int i=s; i<=e; i++) disj.uni(st[qr[i].first], ed[qr[i].first], snapshot);
  for(auto &i : v) if(disj.uni(st[i], ed[i], snapshot)) must_mst.push_back(i);
  disj.revert(snapshot);
  for(auto &i : must_mst) disj.uni(st[i], ed[i], snapshot);
  for(auto &i : v) if(disj.uni(st[i], ed[i], snapshot)) maybe_mst.push_back(i);
  disj.revert(snapshot);
}

void solve(int s, int e, vector<int> v, lint cv){
  if(s == e){
    cost[qr[s].first] = qr[s].second;
    if(st[qr[s].first] == ed[qr[s].first]){
      printf("%lld\n", cv);
      return;
    }
    int minv = qr[s].second;
    for(auto &i : v) minv = min(minv, cost[i]);
    printf("%lld\n",minv + cv);
    return;
  }
  int m = (s+e)/2;
  vector<int> lv = v, rv = v;
  vector<int> must_mst, maybe_mst;
  for(int i=m+1; i<=e; i++){
    chk[qr[i].first]--;
    if(chk[qr[i].first] == 0) lv.push_back(qr[i].first);
  }
  vector<pi> snapshot;
  contract(s, m, lv, must_mst, maybe_mst);
  lint lcv = cv;
  for(auto &i : must_mst) lcv += cost[i], disj.uni(st[i], ed[i], snapshot);
  solve(s, m, maybe_mst, lcv);
  disj.revert(snapshot);
  must_mst.clear(); maybe_mst.clear();
  for(int i=m+1; i<=e; i++) chk[qr[i].first]++;
  for(int i=s; i<=m; i++){
    chk[qr[i].first]--;
    if(chk[qr[i].first] == 0) rv.push_back(qr[i].first);
  }
  lint rcv = cv;
  contract(m+1, e, rv, must_mst, maybe_mst);
  for(auto &i : must_mst) rcv += cost[i], disj.uni(st[i], ed[i], snapshot);
  solve(m+1, e, maybe_mst, rcv);
  disj.revert(snapshot);
  for(int i=s; i<=m; i++) chk[qr[i].first]++;
}

int main(){
  scanf("%d %d",&n,&m);
  vector<int> ve;
  for(int i=0; i<m; i++){
    scanf("%d %d %d",&st[i],&ed[i],&cost[i]);
  }
  scanf("%d",&q);
  for(int i=0; i<q; i++){
    scanf("%d %d",&qr[i].first,&qr[i].second);
    qr[i].first--;
    chk[qr[i].first]++;
  }
  disj.init(n);
  for(int i=0; i<m; i++) if(!chk[i]) ve.push_back(i);
  solve(0, q-1, ve, 0);
}
```

2.7 Dynamic Connectivity in Tree (LCT)

```
struct node{
  node *l, *r, *p, *pp;
  node(){ l = r = p = pp = NULL; }
}*root;
void push(node *x){
  // if there's lazy stuff
}
void pull(node *x){

}
void rotate(node *x){
  if(!x->p) return;
  push(x->p);  // if there's lazy stuff
  push(x);
  node *p = x->p;
  bool is_left = (p->l == x);
  node *b = (is_left ? x->r : x->l);
  x->p = p->p;
  if(x->p && x->p->l == p) x->p->l = x;
  if(x->p && x->p->r == p) x->p->r = x;
  if(is_left){
    if(b) b->p = p;
    p->l = b;
    p->p = x;
    x->r = p;
  }
  else{
    if(b) b->p = p;
    p->r = b;
    p->p = x;
    x->l = p;
  }
  pull(p); // if there's something to pull up
  pull(x);
  if(p->pp){ // IF YOU ARE LINK CUT TREE
    x->pp = p->pp;
    p->pp = NULL;
  }
}
void splay(node *x){
  while(x->p){
    node *p = x->p;
    node *g = p->p;
    if(g){
      if((p->l == x) ^ (g->l == p)) rotate(x);
      else rotate(p);
    }
    rotate(x);
  }
}
void access(node *x){
  splay(x);
  push(x);
  if(x->r){
    x->r->pp = x;
    x->r->p = NULL;
    x->r = NULL;
  }
  pull(x);
  while(x->pp){
```

```
    node *nxt = x->pp;
    splay(nxt);
    push(nxt);
    if(nxt->r){
      nxt->r->pp = nxt;
      nxt->r->p = NULL;
      nxt->r = NULL;
    }
    nxt->r = x;
    x->p = nxt;
    x->pp = NULL;
    pull(nxt);
    splay(x);
  }
}
node *root(node *x){
  access(x);
  while(x->l){
    push(x);
    x = x->l;
  }
  access(x);
  return x;
}
node *par(node *x){
  access(x);
  if(!x->l) return NULL;
  push(x);
  x = x->l;
  while(x->r){
    push(x);
    x = x->r;
  }
  access(x);
  return x;
}
node *lca(node *s, node *t){
  access(s);
  access(t);
  splay(s);
  if(s->pp == NULL) return s;
  return s->pp;
}
void link(node *par, node *son){
  access(par);
  access(son);
  son->rev ^= 1;  // remove if needed
  push(son);
  son->l = par;
  par->p = son;
  pull(son);
}
void cut(node *p){
  access(p);
  push(p);
  if(p->l){
    p->l->p = NULL;
    p->l = NULL;
  }
  pull(p);
}
```

2.8 Edge Coloring in Bipartite Graph, Optimal

```
struct edge_color{ // must use 1-based
  int deg[2][MAXN];
  pi has[2][MAXN][MAXN];
  int color[MAXM];
  int c[2];
  void clear(int n){
    for(int t=0; t<2; t++){
      for(int i=0; i<=n; i++){
        deg[t][i] = 0;
        for(int j=0; j<=n; j++){
          has[t][i][j] = pi(0, 0);
        }
      }
    }
  }
  void dfs(int x, int p) {
    auto i = has[p][x][c[!p]];
    if (has[!p][i.first][c[p]].second) dfs(i.first,!p);
    else has[!p][i.first][c[!p]] = pi(0,0);
    has[p][x][c[p]] = i;
    has[!p][i.first][c[p]] = pi(x,i.second);
    color[i.second] = c[p];
  }
  int solve(vector<pi> v, vector<int> &cv){
    int m = sz(v);
    int ans = 0;
    for (int i=1;i<=m;i++) {
      int x[2];
      x[0] = v[i-1].first;
      x[1] = v[i-1].second;
      for (int d=0;d<2;d++) {
        deg[d][x[d]]+=1;
        ans = max(ans,deg[d][x[d]]);
        for (c[d]=1; has[d][x[d]][c[d]].second; c[d]++);
      }
      if (c[0]!=c[1]) dfs(x[1],1);
      for (int d=0;d<2;d++) has[d][x[d]][c[0]] = pi(x[!d],i);
      color[i] = c[0];
    }
    cv.resize(m);
    for(int i=1; i<=m; i++){
      cv[i-1] = color[i];
      color[i] = 0;
    }
    return ans;
  }
}EC;
```

2.9 Edge Coloring in General Graph, Almost Optimal

```
namespace Vizing{ // returns edge coloring in adjacent matrix G. 1 - based
  int C[MAXN][MAXN], G[MAXN][MAXN];
  void clear(int N){
    for(int i=0; i<=N; i++){
      for(int j=0; j<=N; j++) C[i][j] = G[i][j] = 0;
    }
  }
  void solve(vector<pi> &E, int N, int M){
    int X[MAXN] = {}, a;
    auto update = [&](int u){ for(X[u] = 1; C[u][X[u]]; X[u]++); };
    auto color = [&](int u, int v, int c){
      int p = G[u][v];
      G[u][v] = G[v][u] = c;
      C[u][c] = v; C[v][c] = u;
      C[u][p] = C[v][p] = 0;
      if( p ) X[u] = X[v] = p;
      else update(u), update(v);
      return p;
    };
    auto flip = [&](int u, int c1, int c2){
      int p = C[u][c1];
      swap(C[u][c1], C[u][c2]);
      if( p ) G[u][p] = G[p][u] = c2;
      if( !C[u][c1] ) X[u] = c1;
      if( !C[u][c2] ) X[u] = c2;
      return p;
    };
    for(int i = 1; i <= N; i++) X[i] = 1;
    for(int t = 0; t < E.size(); t++){
      int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c = c0, d;
      vector<pi> L;
      int vst[MAXN] = {};
      while(!G[u][v0]){
        L.emplace_back(v, d = X[v]);
        if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c = color(u, L[a].first,
        c);
        else if(!C[u][d]) for(a=(int)L.size()-1;a>=0;a--)
        color(u,L[a].first,L[a].second);
        else if( vst[d] ) break;
        else vst[d] = 1, v = C[u][d];
      }
      if( !G[u][v0] ){
        for(;v; v = flip(v, c, d), swap(c, d));
        if(C[u][c0]){
          for(a = (int)L.size()-2; a >= 0 && L[a].second != c; a--);
          for(; a >= 0; a--) color(u, L[a].first, L[a].second);
        } else t--;
      }
    }
  }
}
```

2.10 Global Minimum Cut

```
int minimum_cut_phase(int n, int &s, int &t, vector<vector<int>> &adj, vector<int>
vis){
  vector<int> dist(n);
  int mincut = 1e9;
  while(true){
    int pos = -1, cur = -1e9;
    for(int i=0; i<n; i++){
      if(!vis[i] && dist[i] > cur){
        cur = dist[i];
        pos = i;
      }
    }
    if(pos == -1) break;
    s = t;
    t = pos;
    mincut = cur;
    vis[pos] = 1;
    for(int i=0; i<n; i++){
      if(!vis[i]) dist[i] += adj[pos][i];
    }
  }
  return mincut; // optimal s-t cut here is, {t} and V \ {t}
}
int solve(int n, vector<vector<int>> adj){
  if(n <= 1) return 0;
  vector<int> vis(n);
  int ans = 1e9;
  for(int i=0; i<n-1; i++){
    int s, t;
    ans = min(ans, minimum_cut_phase(n, s, t, adj, vis));
    vis[t] = 1;
    for(int j=0; j<n; j++){
      if(!vis[j]){
        adj[s][j] += adj[t][j];
        adj[j][s] += adj[j][t];
      }
    }
    adj[s][s] = 0;
  }
  return ans;
}
```

2.11 k Shortest Paths

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 750005;
using lint = long long;
using pi = pair<lint, lint>;
#define sz(v) ((int)(v).size())
#define all(v) (v).begin(), (v).end()

mt19937 rng(0x69420);
int randint(int lb, int ub){ return uniform_int_distribution<int>(lb, ub)(rng); }

namespace Epp98{
```

```
struct node{
  node *son[2];
  pi val;
  node(){
    son[0] = son[1] = NULL;
    val = pi(-1e18, -1e18);
  }
  node(pi p){
    son[0] = son[1] = NULL;
    val = p;
  }
};

node* copy(node *x){
  if(x == NULL) return NULL;
  node *nd = new node();
  nd->son[0] = x->son[0];
  nd->son[1] = x->son[1];
  nd->val = x->val;
  return nd;
}

// precondition: x, y both points to new entity
node* merge(node *x, node *y){
  if(!x) return y;
  if(!y) return x;
  if(x->val > y->val) swap(x, y);
  int rd = randint(0, 1);
  if(x->son[rd]) x->son[rd] = copy(x->son[rd]);
  x->son[rd] = merge(x->son[rd], y);
  return x;
}

struct edg{
  int pos;
  lint weight;
  int idx;
};
vector<vector<edg>> gph, rev;
int idx;
void init(int n){
  gph.clear();
  rev.clear();
  gph.resize(n);
  rev.resize(n);
  idx = 0;
}
void add_edge(int s, int e, int x){
  gph[s].push_back((edg){e, x, idx});
  rev[e].push_back((edg){s, x, idx});
  idx++;
}
vector<int> par, pae;
vector<lint> dist;
vector<node*> heap;
void dijkstra(int snk){
  // replace this to SPFA if edge weight is negative
  int n = sz(gph);
  par.resize(n);
  pae.resize(n);
  dist.resize(n);
  heap.resize(n);
  fill(all(par), -1);
  fill(all(pae), -1);
  fill(all(dist), 2e18);
  fill(all(heap), (node*) NULL);
  priority_queue<pi, vector<pi>, greater<pi> > pq;
  auto enq = [&](int x, lint v, int pa, int pe){
    if(dist[x] > v){
      dist[x] = v;
      par[x] = pa;
      pae[x] = pe;
      pq.emplace(v, x);
    }
  };
  enq(snk, 0, -1, -1);
  vector<int> ord;
  while(sz(pq)){
    auto [w, v] = pq.top(); pq.pop();
    if(dist[v] != w) continue;
    ord.push_back(v);
    for(auto &e : rev[v]) enq(e.pos, e.weight + w, v, e.idx);
  }
  for(auto &v : ord){
    if(par[v] != -1){
      heap[v] = copy(heap[par[v]]);
    }
    for(auto &i : gph[v]){
      if(i.idx == pae[v]) continue;
      lint delay = dist[i.pos] + i.weight - dist[v];
      if(delay < 1e18){
        heap[v] = merge(heap[v], new node(pi(delay, i.pos)));
      }
    }
  }
}
vector<lint> ksp(int s, int e, int k){
  dijkstra(e);
  using state = pair<lint, node*>;
  priority_queue<state, vector<state>, greater<state>> pq;
  vector<lint> ans;
  if(dist[s] > 1e18){
    ans.resize(k);
    fill(all(ans), -1);
    return ans;
  }
  ans.push_back(dist[s]);
  if(heap[s]) pq.emplace(dist[s] + heap[s]->val.first, heap[s]);
  while(sz(pq) && sz(ans) < k){
    auto [cst, ptr] = pq.top();
    pq.pop();
    ans.push_back(cst);
    for(int j = 0; j < 2; j++){
      if(ptr->son[j]){
        pq.emplace(cst - ptr->val.first + ptr->son[j]->val.first, ptr->son[j]);
      }
    }
    int v = ptr->val.second;
    if(heap[v]) pq.emplace(cst + heap[v]->val.first, heap[v]);
  }
  while(sz(ans) < k) ans.push_back(-1);
  return ans;
 }
}
```

2.12 Perfect Elimination Ordering

```
// given simple, nonempty graph
// in O(m log n), returns permutation of vertices (positive) or empty vector
(negative) such that
// for p[i], all j > i such that p[j] is adjacent to p[i], constitutes cliques.
// for each undir edge (s, e), both s \in gph[e], e \in gph[s] should hold
vector<int> getPEO(vector<vector<int>> gph){
  int n = sz(gph);
  vector<int> cnt(n), idx(n);
  for(int i=0; i<n; i++) sort(gph[i].begin(), gph[i].end());
  priority_queue<pi> pq;
  for(int i=0; i<n; i++) pq.emplace(cnt[i], i);
  vector<int> ord;
  while(!pq.empty()){
    int x = pq.top().second, y = pq.top().first;
    pq.pop();
    if(cnt[x] != y || idx[x]) continue;
    ord.push_back(x);
    idx[x] = n + 1 - ord.size();
    for(auto &i : gph[x]){
      if(!idx[i]){
        cnt[i]++;
        pq.emplace(cnt[i], i);
      }
    }
  }
  reverse(ord.begin(), ord.end());
  for(auto &i : ord){
    int minBef = 1e9;
    for(auto &j : gph[i]){
      if(idx[j] > idx[i]) minBef = min(minBef, idx[j]);
    }
    minBef--;
    if(minBef < n){
      minBef = ord[minBef];
      for(auto &j : gph[i]){
        if(idx[j] > idx[minBef] && !binary_search(gph[minBef].begin(),
        gph[minBef].end(), j)){
          return vector<int>();
        }
      }
    }
  }
  return ord;
}
```

2.13 Tree Decomposition for $tw(G) \leq 2$

```
struct treeDecomp{
  bool valid;
  vector<int> par;
  vector<vector<int>> bags;
};

// Tree decomposition, width 2
treeDecomp tree_decomposition(int n, vector<pi> edges){
  vector<set<int>> gph(n);
  for(auto &[u, v] : edges){
    gph[u].insert(v);
    gph[v].insert(u);
  }
  treeDecomp ret;
  ret.valid = false;
  ret.par.resize(n, -1);
  ret.bags.resize(n);
  queue<int> que;
  for(int i = 0; i < n; i++){
    if(sz(gph[i]) <= 2) que.push(i);
  }
  auto rem_edge = [&](int u, int v){
    gph[u].erase(gph[u].find(v));
    gph[v].erase(gph[v].find(u));
  };
  vector<pair<int, int>> pcand(n, pi(-1, -1));
  vector<int> ord(n, -1);
  int piv = 0;
  while(sz(que)){
    int x = que.front();
    que.pop();
    if(ord[x] != -1) continue;
    ret.bags[x].push_back(x);
    ord[x] = piv++;
    if(sz(gph[x]) == 1){
      int y = *gph[x].begin();
      rem_edge(x, y);
      ret.bags[x].push_back(y);
      if(sz(gph[y]) <= 2) que.push(y);
      pcand[x] = pi(y, y);
    }
    if(sz(gph[x]) == 2){
      int u = *gph[x].begin();
      int v = *gph[x].rbegin();
      rem_edge(x, u);
      rem_edge(x, v);
      gph[u].insert(v);
      gph[v].insert(u);
      ret.bags[x].push_back(u);
```

```
      ret.bags[x].push_back(v);
      if(sz(gph[u]) <= 2) que.push(u);
      if(sz(gph[v]) <= 2) que.push(v);
      pcand[x] = pi(u, v);
    }
  }
  if(piv != n) return ret;
  ret.valid = true;
  int root = -1;
  for(int i = 0; i < n; i++){
    if(pcand[i].first == -1){
      if(root != -1) ret.par[i] = root;
      else root = i;
      continue;
    }
    if(ord[pcand[i].first] < ord[pcand[i].second]) swap(pcand[i].first,
    pcand[i].second);
    ret.par[i] = pcand[i].second;
  }
  return ret;
}
```

3 Strings

3.1 Aho-Corasick

```
const int MAXN = 100005, MAXC = 26;
int trie[MAXN][MAXC], fail[MAXN], term[MAXN], piv;
void init(vector<string> &v){
  memset(trie, 0, sizeof(trie));
  memset(fail, 0, sizeof(fail));
  memset(term, 0, sizeof(term));
  piv = 0;
  for(auto &i : v){
    int p = 0;
    for(auto &j : i){
      if(!trie[p][j]) trie[p][j] = ++piv;
      p = trie[p][j];
    }
    term[p] = 1;
  }
  queue<int> que;
  for(int i=0; i<MAXC; i++){
    if(trie[0][i]) que.push(trie[0][i]);
  }
  while(!que.empty()){
    int x = que.front();
    que.pop();
    for(int i=0; i<MAXC; i++){
      if(trie[x][i]){
        int p = fail[x];
        while(p && !trie[p][i]) p = fail[p];
        p = trie[p][i];
        fail[trie[x][i]] = p;
        if(term[p]) term[trie[x][i]] = 1;
        que.push(trie[x][i]);
      }
    }
  }
}
bool query(string &s){
  int p = 0;
  for(auto &i : s){
    while(p && !trie[p][i]) p = fail[p];
    p = trie[p][i];
    if(term[p]) return 1;
  }
  return 0;
}
```

3.2 Duval's Algorithm

```
vector<int> duval(vector<int> &s){
    int n = sz(s);
    vector<int> v;
    for(int i = 0; i < n; ){
        int j = i + 1, k = i;
        while(j < n && s[k] <= s[j]){
            if(s[k] == s[j]) j++, k++;
            else if(s[k] < s[j]) j++, k = i;
        }
        while(i <= k){
            i += j - k;
            v.push_back(i);
        }
    }
    return v;
}
```

3.3 LCS: Bitset, Hirschberg

```
struct bset{
  vector<word> wd;
  bset(){}
  bset(int n){
    wd.resize(n / 64 + 2);
  }
  void set(int x){
    wd[x >> 6] |= (word(1) << (x & 63));
  }
  void reset(int x){
      wd[x >> 6] = 0;
  }
  bool get(int x){
    return (wd[x >> 6] >> (x & 63)) & 1;
  }
  bset operator-(const bset &b){
    assert(sz(b.wd) == sz(wd));
    bset ret; ret.wd.resize(sz(wd));
    bool carry_bit = 0;
    for(int i=0; i<sz(wd); i++){
      ret.wd[i] = wd[i] - (b.wd[i] + carry_bit);
      carry_bit = (ret.wd[i] > wd[i] || (ret.wd[i] == wd[i] && carry_bit));
    }
    return ret;
  }
  bset operator^(const bset &b){
    assert(sz(b.wd) == sz(wd));
    bset ret; ret.wd.resize(sz(wd));
    for(int i=0; i<sz(wd); i++){
      ret.wd[i] = wd[i] ^ b.wd[i];
    }
    return ret;
  }
  bset operator|(const bset &b){
    bset ret; ret.wd.resize(sz(b.wd));
    for(int i=0; i<sz(b.wd); i++){
      ret.wd[i] = wd[i] | b.wd[i];
    }
    return ret;
  }
  bset operator&(const bset &b){
    assert(sz(b.wd) == sz(wd));
    bset ret; ret.wd.resize(sz(wd));
    for(int i=0; i<sz(wd); i++){
      ret.wd[i] = wd[i] & b.wd[i];
    }
    return ret;
  }
  void shift(){
    for(int i=sz(wd)-1; i>=0; i--){
      wd[i] <<= 1;
      if(i && (wd[i - 1] >> 63)) wd[i] ^= 1;
    }
  }
};

string s, t, ans;
bset alph[26];

vector<int> get_lcs(string s, string t){
  int n = sz(s);
  int m = sz(t);
  bool use[26] = {};
  for(int i=0; i<m; i++){
      use[t[i] - 'A'] = 1;
    alph[t[i] - 'A'].set(i + 1);
  }
  bset B(sz(t));
  for(int i=0; i<n; i++){
    bset y = B; y.shift(); y.set(0);
    bset x = (use[s[i] - 'A'] ? (alph[s[i] - 'A'] | B) : B);
    B = x ^ (x & (x - y));
  }
  vector<int> cur(m + 1);
  int cnt = 0;
  for(int i=0; i<m; i++){
    if(B.get(i + 1)) cnt++;
    cur[i + 1] = cnt;
  }
  for(int i=0; i<m; i++){
    alph[t[i] - 'A'].reset(i + 1);
  }
  return cur;
}

void solve(int l1, int r1, int l2, int r2){
  if(r1 - l1 == 1){
    if(find(t.begin() + l2, t.begin() + r2, s[l1]) != t.begin() + r2)
    ans.push_back(s[l1]);
    return;
  }
  int m = (l1 + r1) / 2;
  string x = s.substr(l1, m - l1);
  string y = s.substr(m, r1 - m);
  string z = t.substr(l2, r2 - l2);
  auto tab1 = get_lcs(x, z);
  reverse(all(y));
  reverse(all(z));
  auto tab2 = get_lcs(y, z);
  reverse(all(tab2));
  pi ans(-1e9, 1e9);
  for(int i=0; i<=sz(z); i++){
    ans = max(ans, pi(tab1[i] + tab2[i], i + l2));
  }
  solve(l1, m, l2, ans.second);
  solve(m, r1, ans.second, r2);
}

int main(){
  cin >> s >> t;
  for(int i=0; i<26; i++) alph[i] = bset(sz(t));
  solve(0, sz(s), 0, sz(t));
  cout << sz(ans) << endl;
  cout << ans << endl;
}
```

3.4 LCS: Circular

```
string s1, s2;
int dp[4005][2005];
int nxt[4005][2005];
int n, m;
void reroot(int px){
  int py = 1;
  while(py <= m && nxt[px][py] != 2) py++;
  nxt[px][py] = 1;
  while(px < 2 * n && py < m){
    if(nxt[px+1][py] == 3){
      px++;
      nxt[px][py] = 1;
    }
    else if(nxt[px+1][py+1] == 2){
      px++;
      py++;
      nxt[px][py] = 1;
```

```
    }
    else py++;
  }
  while(px < 2 * n && nxt[px+1][py] == 3){
    px++;
    nxt[px][py] = 1;
  }
}

int track(int x, int y, int e){ // use this routine to find LCS as string
  int ret = 0;
  while(y != 0 && x != e){
    if(nxt[x][y] == 1) y--;
    else if(nxt[x][y] == 2) ret += (s1[x] == s2[y]), x--, y--;
    else if(nxt[x][y] == 3) x--;
  }
  return ret;
}

int solve(string a, string b){
  n = a.size(), m = b.size();
  s1 = "#" + a + a;
  s2 = '#' + b;
  for(int i=0; i<=2*n; i++){
    for(int j=0; j<=m; j++){
      if(j == 0){
        nxt[i][j] = 3;
        continue;
      }
      if(i == 0){
        nxt[i][j] = 1;
        continue;
      }
      dp[i][j] = -1;
      if(dp[i][j] < dp[i][j-1]){
        dp[i][j] = dp[i][j-1];
        nxt[i][j] = 1;
      }
      if(dp[i][j] < dp[i-1][j-1] + (s1[i] == s2[j])){
        dp[i][j] = dp[i-1][j-1] + (s1[i] == s2[j]);
        nxt[i][j] = 2;
      }
      if(dp[i][j] < dp[i-1][j]){
        dp[i][j] = dp[i-1][j];
        nxt[i][j] = 3;
      }
    }
  }
  int ret = dp[n][m];
  for(int i=1; i<n; i++){
    reroot(i), ret = max(ret, track(n+i, m, i));
  }
  return ret;
}
```

3.5 Suffix Array

```
const int MAXN = 500005;
int ord[MAXN], nord[MAXN], cnt[MAXN], aux[MAXN];
void solve(int n, char *str, int *sfx, int *rev, int *lcp){
  int p = 1;
  memset(ord, 0, sizeof(ord));
  for(int i=0; i<n; i++){
    sfx[i] = i;
    ord[i] = str[i];
  }
  int pnt = 1;
  while(1){
    memset(cnt, 0, sizeof(cnt));
    for(int i=0; i<n; i++) cnt[ord[min(i+p, n)]]++;
    for(int i=1; i<=n || i<=255; i++) cnt[i] += cnt[i-1];
    for(int i=n-1; i>=0; i--)
      aux[--cnt[ord[min(i+p, n)]]] = i;
    memset(cnt, 0, sizeof(cnt));
    for(int i=0; i<n; i++) cnt[ord[i]]++;
    for(int i=1; i<=n || i<=255; i++) cnt[i] += cnt[i-1];
    for(int i=n-1; i>=0; i--)
      sfx[--cnt[ord[aux[i]]]] = aux[i];
    if(pnt == n) break;
    pnt = 1;
    nord[sfx[0]] = 1;
    for(int i=1; i<n; i++){
      if(ord[sfx[i-1]] != ord[sfx[i]] || ord[sfx[i-1] + p] != ord[sfx[i] + p]){
        pnt++;
      }
      nord[sfx[i]] = pnt;
    }
    memcpy(ord, nord, sizeof(int) * n);
    p *= 2;
  }
  for(int i=0; i<n; i++) rev[sfx[i]] = i;
  int h = 0;
  for(int i=0; i<n; i++){
    if(rev[i]){
      int prv = sfx[rev[i] - 1];
      while(str[prv + h] == str[i + h]) h++;
      lcp[rev[i]] = h;
    }
    h = max(h-1, 0);
  }
}
```

3.6 Palindrome Enumerate

```
const int MAXN = 1005;
int aux[2 * MAXN - 1];
void solve(int n, int *str, int *ret){
  // *ret : number of nonobvious palindromic character pair
  for(int i=0; i<n; i++){
    aux[2*i] = str[i];
    if(i != n-1) aux[2*i+1] = -1;
  }
  int p = 0, c = 0;
  for(int i=0; i<2*n-1; i++){
    int cur = 0;
    if(i <= p) cur = min(ret[2 * c - i], p - i);
    while(i - cur - 1 >= 0 && i + cur + 1 < 2*n-1 && aux[i-cur-1] == aux[i+cur+1]){
      cur++;
    }
    ret[i] = cur;
    if(i + ret[i] > p){
      p = i + ret[i];
      c = i;
    }
  }
}
```

3.7 Palindrome Tree

```
int nxt[MAXN][26];
int par[MAXN], len[MAXN], slink[MAXN], ptr[MAXN], diff[MAXN], series[MAXN], piv;
void clear(int n = MAXN){
  memset(par, 0, sizeof(int) * n);
  memset(len, 0, sizeof(int) * n);
  memset(slink, 0, sizeof(int) * n);
  memset(nxt, 0, sizeof(int) * 26 * n);
  piv = 0;
}
void init(int n, char *a){
  par[0] = 0;
  par[1] = 1;
  a[0] = -1;
  len[0] = -1;
  piv = 1;
  int cur = 1;
  for(int i=1; i<=n; i++){
    while(a[i] != a[i - len[cur] - 1]) cur = slink[cur];
    if(!nxt[cur][a[i]]){
      nxt[cur][a[i]] = ++piv;
      par[piv] = cur;
      len[piv] = len[cur] + 2;
      int lnk = slink[cur];
      while(a[i] != a[i - len[lnk] - 1]){
        lnk = slink[lnk];
      }
      if(nxt[lnk][a[i]]) lnk = nxt[lnk][a[i]];
      if(len[piv] == 1 || lnk == 0) lnk = 1;
      slink[piv] = lnk;
      diff[piv] = len[piv] - len[lnk];
      if(diff[piv] == diff[lnk]) series[piv] = series[lnk];
      else series[piv] = piv;
    }
    cur = nxt[cur][a[i]];
    ptr[i] = cur;
  }
}
int query(int s, int e){
  int pos = ptr[e];
  while(len[pos] >= e - s + 1){
        if(len[pos] % diff[pos] == (e - s + 1) % diff[pos] &&
           len[series[pos]] <= e - s + 1) return true;
        pos = series[pos];
        pos = slink[pos];
  }
  return false;
}
vector<pi> minimum_partition(int n){ // (odd min, even min)
  vector<pi> dp(n + 1);
  vector<pi> series_ans(n + 10);
  dp[0] = pi(1e9 + 1, 0);
  for(int i=1; i<=n; i++){
    dp[i] = pi(1e9 + 1, 1e9);
    for(int j=ptr[i]; len[j] > 0;){
      int slv = slink[series[j]];
      series_ans[j] = dp[i - (len[slv] + diff[j])];
      if(diff[j] == diff[slink[j]]){
        series_ans[j].first = min(series_ans[j].first, series_ans[slink[j]].first);
        series_ans[j].second = min(series_ans[j].second,
        series_ans[slink[j]].second);
      }
      auto val = series_ans[j];
      dp[i].first = min(dp[i].first, val.second + 1);
      dp[i].second = min(dp[i].second, val.first + 1);
      j = slv;
    }
  }
  return dp;
}
```

3.8 Run Enumerate

```
namespace ds{
    vector<int> sfx, rev, lcp;
    int spt[19][MAXN], lg[MAXN], n;

    int get_lcp(int s, int e){
        if(s == 2 * n + 1 || e == 2 * n + 1) return 0;
        s = rev[s]; e = rev[e];
        if(s > e) swap(s, e);
        int l = lg[e - s];
        return min(spt[l][e - 1], spt[l][s + (1<<l) - 1]);
    }

    int get_lcp_rev(int s, int e){
        return get_lcp(2*n+1-s, 2*n+1-e);
    }

    void prep(string str){
        n = sz(str);
        string s = str;
        string r = s; reverse(all(r));
        s = s + "#" + r;
        sfx = yosupo::suffix_array(s);
        lcp = yosupo::lcp_array(s, sfx);
        rev.resize(sz(sfx));
        for(int i=0; i<sz(sfx); i++) rev[sfx[i]] = i;
```

```
        for(int i=1; i<MAXN; i++){
            lg[i] = lg[i-1];
            while((2 << lg[i]) <= i) lg[i]++;
        }
        for(int i=0; i<sz(sfx)-1; i++) spt[0][i] = lcp[i];
        for(int i=1; i<19; i++){
            for(int j=0; j<sz(sfx); j++){
                spt[i][j] = spt[i-1][j];
                if(j >= (1<<(i-1))) spt[i][j] = min(spt[i][j],
                spt[i-1][j-(1<<(i-1))]);
            }
        }
    }
}

struct runs{
    int t, l, r;
    bool operator<(const runs &x)const{
        return make_tuple(t, l, r) < make_tuple(x.t, x.l, x.r);
    }
    bool operator==(const runs &x)const{
        return make_tuple(t, l, r) == make_tuple(x.t, x.l, x.r);
    }
};

vector<runs> run_enumerate(string s){
    int n = sz(s);
    vector<pi> v;
    auto get_interval = [&](string t){
        auto sfx = yosupo::suffix_array(t);
        vector<int> rev(n + 1);
        for(int i = 0; i < n; i++) rev[sfx[i]] = i;
        rev[n] = -1;
        vector<int> stk = {n}, ans(n);
        for(int i = n - 1; i >= 0; i--){
            while(sz(stk) && rev[stk.back()] > rev[i]) stk.pop_back();
            v.emplace_back(i, stk.back());
            stk.push_back(i);
        }
    };
    ds::prep(s);
    get_interval(s);
    for(auto &i : s) i = 'a' + 'z' - i;
    get_interval(s);
    vector<runs> ans;
    for(auto &[x, y] : v){
        int s = x - ds::get_lcp_rev(x, y);
        int e = y + ds::get_lcp(x, y);
        int p = y - x;
        if(e - s >= 2 * p){
            ans.push_back({p, s, e});
        }
    }
    sort(all(ans));
    ans.resize(unique(all(ans)) - ans.begin());
    return ans;
}
```

4 Mathematics

4.1 Polynomial

```
struct mint {
    int val;
    mint() { val = 0; }
    mint(const lint& v) {
        val = (-mod <= v && v < mod) ? v : v % mod;
        if (val < 0) val += mod;
    }

    friend ostream& operator<<(ostream& os, const mint& a) { return os << a.val; }
    friend bool operator==(const mint& a, const mint& b) { return a.val == b.val; }
    friend bool operator!=(const mint& a, const mint& b) { return !(a == b); }
    friend bool operator<(const mint& a, const mint& b) { return a.val < b.val; }

    mint operator-() const { return mint(-val); }
    mint& operator+=(const mint& m) { if ((val += m.val) >= mod) val -= mod; return
    *this; }
    mint& operator-=(const mint& m) { if ((val -= m.val) < 0) val += mod; return
    *this; }
    mint& operator*=(const mint& m) { val = (lint)val*m.val%mod; return *this; }
    friend mint ipow(mint a, lint p) {
        mint ans = 1; for (; p; p /= 2, a *= a) if (p&1) ans *= a;
        return ans;
    }
    friend mint inv(const mint& a) { assert(a.val); return ipow(a, mod - 2); }
    mint& operator/=(const mint& m) { return (*this) *= inv(m); }

    friend mint operator+(mint a, const mint& b) { return a += b; }
    friend mint operator-(mint a, const mint& b) { return a -= b; }
    friend mint operator*(mint a, const mint& b) { return a *= b; }
    friend mint operator/(mint a, const mint& b) { return a /= b; }
    operator int64_t() const {return val; }
};

namespace fft{
    using real_t = double;
    using base = complex<real_t>;

    void fft(vector<base> &a, bool inv){
        int n = a.size(), j = 0;
        vector<base> roots(n/2);
        for(int i=1; i<n; i++){
            int bit = (n >> 1);
            while(j >= bit){
                j -= bit;
                bit >>= 1;
            }
            j += bit;
            if(i < j) swap(a[i], a[j]);
        }
        real_t ang = 2 * acos(real_t(-1)) / n * (inv ? -1 : 1);
        for(int i=0; i<n/2; i++){
            roots[i] = base(cos(ang * i), sin(ang * i));
        }
        /*
           XOR Convolution : set roots[*] = 1.
           OR Convolution : set roots[*] = 1, and do following:
           if (!inv) {
           a[j + k] = u + v;
           a[j + k + i/2] = u;
           } else {
           a[j + k] = v;
           a[j + k + i/2] = u - v;
           }
         */
        for(int i=2; i<=n; i<<=1){
            int step = n / i;
            for(int j=0; j<n; j+=i){
                for(int k=0; k<i/2; k++){
                    base u = a[j+k], v = a[j+k+i/2] * roots[step * k];
                    a[j+k] = u+v;
                    a[j+k+i/2] = u-v;
                }
            }
        }
        if(inv) for(int i=0; i<n; i++) a[i] /= n; // skip for OR convolution.
    }
    template<typename T>
    void ntt(vector<T> &a, bool inv){
        const int prr = 3; // primitive root
        int n = a.size(), j = 0;
        vector<T> roots(n/2);
        for(int i=1; i<n; i++){
            int bit = (n >> 1);
            while(j >= bit){
                j -= bit;
                bit >>= 1;
            }
            j += bit;
            if(i < j) swap(a[i], a[j]);
        }
        T ang = ipow(T(prr), (mod - 1) / n);
        if(inv) ang = T(1) / ang;
        for(int i=0; i<n/2; i++){
            roots[i] = (i ? (roots[i-1] * ang) : T(1));
        }
        for(int i=2; i<=n; i<<=1){
            int step = n / i;
            for(int j=0; j<n; j+=i){
                for(int k=0; k<i/2; k++){
                    T u = a[j+k], v = a[j+k+i/2] * roots[step * k];
                    a[j+k] = u+v;
                    a[j+k+i/2] = u-v;
                }
            }
        }
        if(inv){
            T rev = T(1) / T(n);
            for(int i=0; i<n; i++) a[i] *= rev;
        }
    }
    template<typename T>
    vector<T> multiply_ntt(vector<T> &v, const vector<T> &w){
        vector<T> fv(all(v)), fw(all(w));
        int n = 2;
        while(n < sz(v) + sz(w)) n <<= 1;
        fv.resize(n); fw.resize(n);
        ntt(fv, 0); ntt(fw, 0);
        for(int i=0; i<n; i++) fv[i] *= fw[i];
        ntt(fv, 1);
        vector<T> ret(n);
        for(int i=0; i<n; i++) ret[i] = fv[i];
        return ret;
    }
    template<typename T>
    vector<T> multiply(vector<T> &v, const vector<T> &w){
        vector<base> fv(all(v)), fw(all(w));
        int n = 2;
        while(n < sz(v) + sz(w)) n <<= 1;
        fv.resize(n); fw.resize(n);
        fft(fv, 0); fft(fw, 0);
        for(int i=0; i<n; i++) fv[i] *= fw[i];
        fft(fv, 1);
        vector<T> ret(n);
        for(int i=0; i<n; i++) ret[i] = (T)llround(fv[i].real());
        return ret;
    }
    template<typename T>
    vector<T> multiply_mod(vector<T> v, const vector<T> &w){
        int n = 2;
        while(n < sz(v) + sz(w)) n <<= 1;
        vector<base> v1(n), v2(n), r1(n), r2(n);
        for(int i=0; i<v.size(); i++){
            v1[i] = base(v[i] >> 15, v[i] & 32767);
        }
        for(int i=0; i<w.size(); i++){
            v2[i] = base(w[i] >> 15, w[i] & 32767);
        }
        fft(v1, 0);
        fft(v2, 0);
        for(int i=0; i<n; i++){
            int j = (i ? (n - i) : i);
            base ans1 = (v1[i] + conj(v1[j])) * base(0.5, 0);
            base ans2 = (v1[i] - conj(v1[j])) * base(0, -0.5);
            base ans3 = (v2[i] + conj(v2[j])) * base(0.5, 0);
            base ans4 = (v2[i] - conj(v2[j])) * base(0, -0.5);
            r1[i] = (ans1 * ans3) + (ans1 * ans4) * base(0, 1);
            r2[i] = (ans2 * ans3) + (ans2 * ans4) * base(0, 1);
        }
        fft(r1, 1);
        fft(r2, 1);
        vector<T> ret(n);
```

```
        for(int i=0; i<n; i++){
            T av = llround(r1[i].real());
            T bv = llround(r1[i].imag()) + llround(r2[i].real());
            T cv = llround(r2[i].imag());
            av = av << 30;
            bv = bv << 15;
            ret[i] = av + bv + cv;
        }
        return ret;
    }
    template<typename T>
    vector<T> multiply_naive(vector<T> v, const vector<T> &w){
        if(sz(v) == 0 || sz(w) == 0) return vector<T>();
        vector<T> ret(sz(v) + sz(w) - 1);
        for(int i=0; i<sz(v); i++){
            for(int j=0; j<sz(w); j++){
                ret[i + j] += v[i] * w[j];
            }
        }
        return ret;
    }

}

template<typename T>
struct poly {
    vector<T> a;

    void normalize() { // get rid of leading zeroes
        while(!a.empty() && a.back() == T(0)) {
            a.pop_back();
        }
    }

    poly(){}
    poly(T a0){ a = {a0}; normalize(); }
    poly(vector<T> t) : a(t){ normalize(); }

    int deg() const{ return sz(a) - 1; } // -1 if empty
    T lead() const{ return sz(a) ? a.back() : T(0); }

    T operator [](int idx) const {
        return idx >= (int)a.size() || idx < 0 ? T(0) : a[idx];
    }

    T& coef(size_t idx) { // mutable reference at coefficient
        return a[idx];
    }

    poly reversed() const{
        vector<T> b = a;
        reverse(all(b));
        return poly(b);
    }
    poly trim(int n) const{
        n = min(n, sz(a));
        vector<T> b(a.begin(), a.begin() + n);
        return poly(b);
    }

    poly operator *= (const T &x) {
        for(auto &it: a) {
            it *= x;
        }
        normalize();
        return *this;
    }
    poly operator /= (const T &x) {
        return *this *= (T(1)/ T(x));
    }
    poly operator * (const T &x) const {return poly(*this) *= x;}
    poly operator / (const T &x) const {return poly(*this) /= x;}

    poly operator+=(const poly &p){
        a.resize(max(sz(a), sz(p.a)));
        for(int i=0; i<sz(p.a); i++){
            a[i] += p.a[i];
        }
        normalize();
        return *this;
    }
    poly operator-=(const poly &p){
        a.resize(max(sz(a), sz(p.a)));
        for(int i=0; i<sz(p.a); i++){
            a[i] -= p.a[i];
        }
        normalize();
        return *this;
    }
    poly operator*=(const poly &p){
        *this = poly(fft::asdf(a, p.a));
        normalize();
        return *this;
    }
    poly inv(int n){
        poly q(T(1) / a[0]);
        for(int i=1; i<n; i<<=1){
            poly p = poly(2) - q * trim(i * 2);
            q = (p * q).trim(i * 2);
        }
        return q.trim(n);
    }
    pair<poly, poly> divmod_slow(const poly &b) const { // when divisor or quotient
    is small
        vector<T> A(a);
        vector<T> res;
        while(A.size() >= b.a.size()) {
            res.push_back(A.back() / b.a.back());
            if(res.back() != T(0)) {
                for(size_t i = 0; i < b.a.size(); i++) {
                    A[A.size() - i - 1] -= res.back() * b.a[b.a.size() - i - 1];
                }
            }
            A.pop_back();
        }
        reverse(all(res));
        return {res, A};
    }

    poly operator/=(const poly &b){
        if(deg() < b.deg()) return *this = poly();
        if(min(deg(), b.deg()) < 256) return *this = divmod_slow(b).first;
        int k = deg() - b.deg() + 1;
        poly ra = reversed().trim(k);
        poly rb = b.reversed().trim(k).inv(k);
        *this = (ra * rb).trim(k);
        while(sz(a) < k) a.push_back(T(0));
        reverse(all(a));
        normalize();
        return *this;
    }
    poly operator%=(const poly &b){
        if(deg() < b.deg()) return *this;
        if(min(deg(), b.deg()) < 256) return *this = divmod_slow(b).second;
        poly foo = poly(a); foo /= b; foo *= b;
        *this = poly(*this) -= foo;
        normalize();
        return *this;
    }

    poly operator+(const poly &p)const{ return poly(*this) += p; }
    poly operator-(const poly &p)const{ return poly(*this) -= p; }
    poly operator*(const poly &p)const{ return poly(*this) *= p; }
    poly operator/(const poly &p)const{ return poly(*this) /= p; }
    poly operator%(const poly &p)const{ return poly(*this) %= p; }

    poly deriv() { // calculate derivative
        vector<T> res;
        for(int i = 1; i <= deg(); i++) {
            res.push_back(T(i) * a[i]);
        }
        return res;
    }
    poly integr() { // calculate integral with C = 0
        vector<T> res = {0};
        for(int i = 0; i <= deg(); i++) {
            res.push_back(a[i] / T(i + 1));
        }
        return res;
    }
    poly ln(int n){
        assert(sz(a) > 0 && a[0] == T(1));
        return (deriv() * inv(n)).integr().trim(n);
    }
    poly exp(int n){
        if(sz(a) == 0){
            return poly({T(1)});
        }
        assert(sz(a) > 0 && a[0] == T(0));
        poly q(1);
        for(int i=1; i<n; i<<=1){
            poly p = poly(1) + trim(2 * i) - q.ln(2 * i);
            q = (q * p).trim(2 * i);
        }
        return q.trim(n);
    }
    poly power(int n, int k){
        if(sz(a) == 0) return poly();
        if(k == 0) return poly(T(1)).trim(n);
        if(k == 1) return trim(n);
        int ptr = 0;
        while(ptr < sz(a) && a[ptr] == T(0)) ptr++;
        if(1ll * ptr * k >= n) return poly();
        n -= ptr * k;
        poly p(vector<T>(a.begin() + ptr, a.end()));
        T coeff = a[ptr];
        p /= coeff;
        p = p.ln(n);
        p *= k;
        p = p.exp(n);
        p *= ipow(coeff, k);
        vector<T> q(ptr * k, T(0));
        for(int i=0; i<=p.deg(); i++) q.push_back(p[i]);
        return poly(q);
    }
    poly root(int n, int k = 2){
        // NOT TESTED in K > 2
        assert(sz(a) > 0 && a[0] == T(1) && k >= 2);
        poly q(1);
        for(int i=1; i<n; i<<=1){
            if(k == 2) q += trim(2 * i) * q.inv(2 * i);
            else q = q * T(k - 1) + trim(2 * i) * power(q.inv(2 * i), k - 1, 2 * i);
            q = q.trim(2 * i) / T(k);
        }
        return q.trim(n);
    }
};
```

4.2 Algorithms on Polynomial

```
using pol = poly<mint>;

mint resultant(pol &a, pol &b){
    if(a.deg() == -1 || b.deg() == -1) return 0;
    if(a.deg() == 0 || b.deg() == 0){
        return ipow(a.lead(), b.deg()) * ipow(b.lead(), a.deg());
    }
    if(b.deg() > a.deg()){
        mint flag = (a.deg() % 2 && b.deg() % 2) ? -1 : 1;
        return resultant(b, a) * flag;
    }
    poly nxt = a % b;
```

```
    return ipow(b.lead(), a.deg() - nxt.deg()) * resultant(nxt, b);
}

pol interpolate(vector<mint> y, vector<mint> x){
    int n = sz(y);
    vector<mint> res(n), temp(n);
    for(int k = 0; k < n-1; k++){
        for(int i = k+1; i < n; i++){
            y[i] = (y[i] - y[k]) / (x[i] - x[k]);
        }
    }
    mint last = 0; temp[0] = 1;
    for(int k=0; k<n; k++){
        for(int i=0; i<n; i++){
            res[i] += y[k] * temp[i];
            swap(last, temp[i]);
            temp[i] -= last * x[k];
        }
    }
    return res;
}

pol interpolate_x0toxn(vector<mint> y){
    int n = sz(y);
    vector<mint> res(n), temp(n);
    vector<mint> x;
    for(int i = 0; i < n; i++){
        x.push_back(mint(1) / mint(i + 1));
    }
    for(int k = 0; k < n-1; k++){
        for(int i = k+1; i < n; i++){
            y[i] = (y[i] - y[k]) * x[i - k - 1];
        }
    }
    mint last = 0; temp[0] = 1;
    for(int k=0; k<n; k++){
        for(int i=0; i<n; i++){
            res[i] += y[k] * temp[i];
            swap(last, temp[i]);
            temp[i] -= last * mint(k);
        }
    }
    return res;
}

pol hessenberg(vector<vector<mint>> a){
    int n = sz(a);
    for(int i = 0; i+1 < n; i++){
        int j = i+1;
        while(j < n && a[j][i] == mint(0)) j++;
        if(j == n) continue;
        for(int k = 0; k < n; k++){
            swap(a[i+1][k], a[j][k]);
        }
        for(int k = 0; k < n; k++){
            swap(a[k][j], a[k][i+1]);
        }
        assert(a[i+1][i] != mint(0));
        for(int k = i+2; k < n; k++){
            mint gyesu = a[k][i] / a[i+1][i];
            for(int l = 0; l < n; l++){
                a[k][l] -= gyesu * a[i+1][l];
            }
            for(int l = 0; l < n; l++){
                a[l][i+1] += a[l][k] * gyesu;
            }
        }
    }
    vector<mint> interps(n + 1);
    for(int i = 0; i <= n; i++){
        auto b = a;
        mint gyesu = mint(1);
        for(int j = 1; j < n; j++){
            if(b[j-1][j-1] == mint(0)){
                gyesu *= -1;
                for(int k = 0; k < n; k++) swap(b[j-1][k], b[j][k]);
                continue;
            }
            mint inv = b[j][j-1] / b[j-1][j-1];
            for(int k = j-1; k < n; k++){
                b[j][k] -= inv * b[j-1][k];
            }
        }
        for(int j=0; j<n; j++) gyesu *= b[j][j];
        interps[i] = gyesu;
        for(int j = 0; j < n; j++){
            a[j][j] -= mint(1);
        }
    }
    return interpolate_x0toxn(interps);
}

using pol = poly<mint>;
vector<mint> chirpz_even(pol p, mint x, int n){
  if(p.deg() == -1){
    return vector<mint>(n, 0);
  }
  vector<mint> sq_list(p.deg() + 1 + n);
  {
    mint cur = x;
    sq_list[0] = 1;
    for(int i=1; i<sz(sq_list); i++){
      sq_list[i] = cur * sq_list[i - 1];
      cur *= x * x;
    }
  }
  for(int i=0; i<=p.deg(); i++){
    p.a[i] /= sq_list[i];
  }
  p = p.reversed();
  poly q(sq_list);
  poly ret = p * q;
  vector<mint> ans(n);
  for(int i=0; i<n; i++){
    ans[i] = ret[i + p.deg()] / sq_list[i];
  }
  return ans;
}

vector<mint> chirpz(pol p, mint x, int n){  // return first n terms of p(x^{2k})
  pol poly_ev = p;
  pol poly_od = p;
  mint cur = 1;
  for(int i=0; i<=p.deg(); i++){
    poly_od.a[i] *= cur;
    cur *= x;
  }
  auto rx = chirpz_even(poly_ev, x, (n + 2) / 2);
  auto ry = chirpz_even(poly_od, x, (n + 1) / 2);
  vector<mint> ret;
  for(int i=0; i<sz(rx) || i<sz(ry); i++){
    if(i < sz(rx)) ret.push_back(rx[i]);
    if(i < sz(ry)) ret.push_back(ry[i]);
  }
  return ret;
}
```

4.3 Recurrence Finder: Berlekamp-Massey

```
const int mod = 998244353;
using lint = long long;
lint ipow(lint x, lint p){
  lint ret = 1, piv = x;
  while(p){
    if(p & 1) ret = ret * piv % mod;
    piv = piv * piv % mod;
    p >>= 1;
  }
  return ret;
}
vector<int> berlekamp_massey(vector<int> x){
  vector<int> ls, cur;
  int lf, ld;
  for(int i=0; i<x.size(); i++){
    lint t = 0;
    for(int j=0; j<cur.size(); j++){
      t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
    }
    if((t - x[i]) % mod == 0) continue;
    if(cur.empty()){
      cur.resize(i+1);
      lf = i;
      ld = (t - x[i]) % mod;
      continue;
    }
    lint k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
    vector<int> c(i-lf-1);
    c.push_back(k);
    for(auto &j : ls) c.push_back(-j * k % mod);
    if(c.size() < cur.size()) c.resize(cur.size());
    for(int j=0; j<cur.size(); j++){
      c[j] = (c[j] + cur[j]) % mod;
    }
    if(i-lf+(int)ls.size()>=(int)cur.size()){
      tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
    }
    cur = c;
  }
  for(auto &i : cur) i = (i % mod + mod) % mod;
  return cur;
}
int get_nth(vector<int> rec, vector<int> dp, lint n){
  int m = rec.size();
  vector<int> s(m), t(m);
  s[0] = 1;
  if(m != 1) t[1] = 1;
  else t[0] = rec[0];
  auto mul = [&rec](vector<int> v, vector<int> w){
    int m = v.size();
    vector<int> t(2 * m);
    for(int j=0; j<m; j++){
      for(int k=0; k<m; k++){
        t[j+k] += 1ll * v[j] * w[k] % mod;
        if(t[j+k] >= mod) t[j+k] -= mod;
      }
    }
    for(int j=2*m-1; j>=m; j--){
      for(int k=1; k<=m; k++){
        t[j-k] += 1ll * t[j] * rec[k-1] % mod;
        if(t[j-k] >= mod) t[j-k] -= mod;
      }
    }
    t.resize(m);
    return t;
  };
  while(n){
    if(n & 1) s = mul(s, t);
    t = mul(t, t);
    n >>= 1;
  }
  lint ret = 0;
  for(int i=0; i<m; i++) ret += 1ll * s[i] * dp[i] % mod;
  return ret % mod;
}
int guess_nth_term(vector<int> x, lint n){
  if(n < x.size()) return x[n];
  vector<int> v = berlekamp_massey(x);
  if(v.empty()) return 0;
  return get_nth(v, x, n);
}
struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no duplicate please..
vector<int> get_min_poly(int n, vector<elem> M){
```

```
  // smallest poly P such that A^i = sum_{j < i} {A^j \times P_j}
  vector<int> rnd1, rnd2;
  mt19937 rng(0x14004);
  auto randint = [&rng](int lb, int ub){
    return uniform_int_distribution<int>(lb, ub)(rng);
  };
  for(int i=0; i<n; i++){
    rnd1.push_back(randint(1, mod - 1));
    rnd2.push_back(randint(1, mod - 1));
  }
  vector<int> gobs;
  for(int i=0; i<2*n+2; i++){
    int tmp = 0;
    for(int j=0; j<n; j++){
      tmp += 1ll * rnd2[j] * rnd1[j] % mod;
      if(tmp >= mod) tmp -= mod;
    }
    gobs.push_back(tmp);
    vector<int> nxt(n);
    for(auto &i : M){
      nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
      if(nxt[i.x] >= mod) nxt[i.x] -= mod;
    }
    rnd1 = nxt;
  }
  auto sol = berlekamp_massey(gobs);
  reverse(sol.begin(), sol.end());
  return sol;
}
lint det(int n, vector<elem> M){
  vector<int> rnd;
  mt19937 rng(0x14004);
  auto randint = [&rng](int lb, int ub){
    return uniform_int_distribution<int>(lb, ub)(rng);
  };
  for(int i=0; i<n; i++) rnd.push_back(randint(1, mod - 1));
  for(auto &i : M){
    i.v = 1ll * i.v * rnd[i.y] % mod;
  }
  auto sol = get_min_poly(n, M)[0];
  if(n % 2 == 0) sol = mod - sol;
  for(auto &i : rnd) sol = 1ll * sol * ipow(i, mod - 2) % mod;
  return sol;
}
```

4.4 Recurrence Finder: P-recursive

```
using i64 = long long;

int mod_inv(int a, int mod) {
  int b = mod, s = 1, t = 0, old_a = a;
  while (b) {
    int q = a / b;
    swap(a %= b, b);
    swap(s -= t * q, t);
  }
  if (a != 1) {
    fprintf(stderr,
      "Error: %d^{-1} mod %d does not exist.\n\n", old_a, mod);
    assert(0);
  }
  return s < 0 ? s + mod : s;
}

vector<int> extended(int n,
    const vector< vector<int> >& coeffs, const vector<int>& terms, int mod) {

  vector<int> ret(max<int>(n + 1, terms.size()));
  copy(terms.begin(), terms.end(), ret.begin());
  const int order = coeffs.size() - 1;
  const int deg = coeffs[0].size() - 1;
  assert((int) terms.size() >= order);
  for (int m = terms.size(); m <= n; ++m) {
    int s = 0;
    for (int i = 1; i <= order; ++i) {
      int k = m - i, t = ret[k];
      for (int d = 0; d <= deg; ++d) {
        s = (s + i64(t) * coeffs[i][d]) % mod;
        t = i64(t) * k % mod;
      }
    }
    int denom = 0, mpow = 1;
    for (int d = 0; d <= deg; ++d) {
      denom = (denom + i64(mpow) * coeffs[0][d]) % mod;
      mpow = i64(mpow) * m % mod;
    }
    ret[m] = i64(mod - s) * mod_inv(denom, mod) % mod;
  }
  return ret;
}

vector< vector<int> > find_recurrence_relation(
    const vector<int>& terms, const int deg, const int mod, bool verify=true) {

  const int n = terms.size();
  const int B = (n + 2) / (deg + 2); // number of blocks
  const int C = B * (deg + 1); // number of columns
  const int R = n - (B - 1); // number of rows
  assert(B >= 2); assert(R >= C - 1);

  auto mul = [mod] (int a, int b) { return i64(a) * b % mod; };
  auto fixed = [mod] (int a) { return (a %= mod) < 0 ? a + mod : a; };
  auto error = [] (int order, int deg) {
    fprintf(stderr,
      "Error: Could not find a recurrence relation "
      "of order <= %d and degree <= %d.\n\n",
      order, deg);
    assert(0);
  };

  vector< vector<int> > mat(R, vector<int>(C));
  for (int y = 0; y < R; ++y) {
    for (int b = 0; b < B; ++b) {
      for (int d = 0, v = fixed(terms[y + b]); d <= deg; ++d) {
        mat[y][b * (deg + 1) + d] = v;
        v = mul(v, y + b);
      }
    }
  }

  int rank = 0;
  for (int x = 0; x < C; ++x) {
    int pivot = -1;
    for (int y = rank; y < R; ++y) if (mat[y][x] != 0) {
      pivot = y; break;
    }
    if (pivot < 0) break;
    if (pivot != rank) swap(mat[rank], mat[pivot]);
    int inv = mod_inv(mat[rank][x], mod);
    for (int x2 = x; x2 < C; ++x2) mat[rank][x2] = mul(mat[rank][x2], inv);
    for (int y = rank + 1; y < R; ++y) if (mat[y][x]) {
      int c = mod - mat[y][x];
      for (int x2 = x; x2 < C; ++x2) {
        mat[y][x2] = (i64(mat[y][x2]) + i64(c) * mat[rank][x2]) % mod;
      }
    }
    ++rank;
  }

  if (rank == C) error(B - 1, deg);

  for (int y = rank - 1; y >= 0; --y) if (mat[y][rank]) {
    assert(mat[y][y] == 1);
    int c = mod - mat[y][rank];
    for (int y2 = 0; y2 < y; ++y2) {
      mat[y2][rank] = (mat[y2][rank] + i64(c) * mat[y2][y]) % mod;
    }
  }

  int order = rank / (deg + 1);
  vector< vector<int> > ret(order + 1, vector<int>(deg + 1));
  ret[0][rank % (deg + 1)] = 1;
  for (int y = rank - 1; y >= 0; --y) {
    int k = order - y / (deg + 1), d = y % (deg + 1);
    ret[k][d] = (mod - mat[y][rank]) % mod;
  }

  if (verify) {
    auto extended_terms = extended(n - 1, ret,
        vector<int>(terms.begin(), terms.begin() + order), mod);
    for (int i = 0; i < (int) terms.size(); ++i) {
      if (fixed(terms[i] - extended_terms[i]) != 0) error(B - 1, deg);
    }
  }

  auto verbose = [&] {
    int last = verify ? n - 1 : order + R - 1;
    fprintf(stderr,
      "[ Found a recurrence relation ]\n"
      "- order %d\n"
      "- degree %d\n"
      "- verified up to a(%d) (number of non-trivial terms: %d)\n",
      order, deg, last, (last + 1) - ((deg + 2) * (order + 1) - 2)
    );
    fprintf(stderr, "{\n");
    for (int k = 0; k <= order; ++k) {
      fprintf(stderr, "  {");
      for (int d = 0; d <= deg; ++d) {
        if (d) fprintf(stderr, ", ");
        fprintf(stderr, "%d", ret[k][d] <= mod / 2 ? ret[k][d] : ret[k][d] - mod);
      }
      fprintf(stderr, "}%s\n", k == order ? "" : ",");
    }
    fprintf(stderr, "}\n\n");
  };
  verbose();

  return ret;
}

void show_extended_sequence(int n, const vector<int>& terms, int degree, int mod) {
  auto coeffs = find_recurrence_relation(terms, degree, mod);
  auto extended_terms = extended(n, coeffs, terms, mod);
  for (int i = 0; i < (int) extended_terms.size(); ++i) {
    printf("%d %d\n", i, extended_terms[i]);
  }
  puts("");
}

int main() {
  const int mod = 1e9 + 7;

  // Ones
  show_extended_sequence(10, {1, 1, 1, 1, 1}, 0, mod);

  // Factorials
  show_extended_sequence(10, {1, 1, 2, 6, 24, 120}, 1, mod);

  // Catalan numbers
  show_extended_sequence(10, {1, 1, 2, 5, 14, 42}, 1, mod);

  // Error: (n + 1) a_n \equiv 0 (mod 2)
  // show_extended_sequence(10, {0, 1, 0, 1, 0, 1, 0, 1, 0, 1}, 1, 2);

  // binom(3 * i, i) ** 2 + binom(2 * i, i + 1): order 8, degree 5
  show_extended_sequence(128, {
    1, 10, 229, 7071, 245081,
    9018219, 344622888, 521041312, 917599501, 328470621,
    920199271, 726809819, 712906773, 531692419, 688496750,
    140388924, 514070772, 712606107, 333670208, 549905369,
    504023887, 34217948, 890190161, 703909490, 6403597,
    623962638, 685637246, 126160387, 956873888, 9766247,
```

```
    864866393, 563563889, 613532405, 710746029, 182520210,
    914377932, 648461424, 715143730, 918800735, 503145605,
    27402642, 282029583, 635728688, 91880493, 896737996,
    773282006, 625726102, 992524580, 494071629, 82874383,
    536460288, 218839718, 406647024, 248185000, 360613817,
    546217158, 925224608, 482921337, 928327434, 372559325,
    614987117, 601351833, 765504201, 230666863, 98348380,
  }, 5, mod);
  return 0;
}
```

4.5 Binomial

```
//https://judge.yosupo.jp/submission/65139
// Require: 1 <= b
// return: (g, x) s.t. g = gcd(a, b), xa = g MOD b, 0 <= x < b/g
template <typename Int> /* constexpr */ std::pair<Int, Int> inv_gcd(Int a, Int b) {
    a %= b;
    if (a < 0) a += b;
    if (a == 0) return {b, 0};
    Int s = b, t = a, m0 = 0, m1 = 1;
    while (t) {
        Int u = s / t;
        s -= t * u, m0 -= m1 * u;
        auto tmp = s;
        s = t, t = tmp, tmp = m0, m0 = m1, m1 = tmp;
    }
    if (m0 < 0) m0 += b / s;
    return {s, m0};
}

struct combination_prime_pow {
    int p, q, m;
    std::vector<int> fac, invfac;

    int _pow(int x, long long n) const {
        long long ans = 1;
        while (n) {
            if (n & 1) ans = ans * x % m;
            x = (long long)x * x % m;
            n >>= 1;
        }
        return ans;
    }

    long long _ej(long long n, int j) const {
        for (int t = 0; t < j + 1 and n; ++t) n /= p;
        long long ret = 0;
        while (n) ret += n, n /= p;
        return ret;
    }

    combination_prime_pow(int p_, int q_) : p(p_), q(q_), m(1) {
        for (int t = 0; t < q; ++t) m *= p;
        fac.assign(m, 1);
        invfac.assign(m, 1);
        for (int i = 1; i < m; ++i) fac[i] = (long long)fac[i - 1] * (i % p ? i : 1)
        % m;
        invfac[m - 1] = _pow(fac[m - 1], m / p * (p - 1) - 1);
        for (int i = m - 1; i; --i) invfac[i - 1] = (long long)invfac[i] * (i % p ? i
        : 1) % m;
    }

    int nCr(long long n, long long r) const {
        if (r < 0 or n < r) return 0;
        long long k = n - r;
        long long e0 = _ej(n, 0) - _ej(r, 0) - _ej(k, 0);
        if (e0 >= q) return 0;
        long long ret = _pow(p, e0);
        if ((p > 2 or q < 3) and (_ej(n, q - 1) - _ej(r, q - 1) - _ej(k, q - 1)) & 1)
        {
            ret = ret ? m - ret : 0;
        }
        while (n) {
            ret = __int128(ret) * fac[n % m] * ((long long)invfac[r % m] * invfac[k %
            m]) % m;
            n /= p, r /= p, k /= p;
        }
        return (int)ret;
    }
};

// nCr mod m
// Complexity: O(m) space worst (construction), O(polylog(n)) (per query)
// https://judge.yosupo.jp/problem/binomial_coefficient
struct combination {
    std::vector<combination_prime_pow> cpps;
    std::vector<int> ms, ims;

    template <class Map> combination(const Map &p2deg) {
        int m0 = 1;
        for (auto f : p2deg) {
            cpps.push_back(combination_prime_pow(f.first, f.second));
            int m1 = cpps.back().m;
            ms.push_back(m1);
            if (m0 < m1) std::swap(m0, m1);
            int im = inv_gcd<int>(m0, m1).second;
            ims.push_back(im);
            m0 *= m1;
        }
    }

    int operator()(long long n, long long r) const {
        if (r < 0 or n < r) return 0;
        int r0 = 0, m0 = 1;
        for (int i = 0; i < int(cpps.size()); ++i) {
            int r1 = cpps[i].nCr(n, r) % ms[i], m1 = ms[i];
            if (m0 < m1) {
                std::swap(r0, r1);
                std::swap(m0, m1);
            }
            int im = ims[i];
            int x = (r1 - r0) % m1 * im % m1;
            r0 += x * m0;
            m0 *= m1;
            if (r0 < 0) r0 += m0;
        }
        return r0;
        // std::vector<int> rs;
        // for (const auto &cpp : cpps) rs.push_back(cpp.nCr(n, r));
        // return crt(rs, ms).first;
    }
};


int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  int q; cin >> q;
  int m; cin >> m;
  map<int, int> mp;
  for(int i = 2; i * i <= m; i++){
    int cnt = 0;
    while(m % i == 0){
      m /= i;
      cnt++;
    }
    if(cnt) mp[i] = cnt;
  }
  if(m > 1) mp[m] = 1;
  combination solver(mp);
  while(q--){
  lint n, k; cin >> n >> k;
  cout << solver(n, k) << "\n";
  }
}
```

4.6 De Bruijn Sequence

```
// Create cyclic string of length k^n that contains every length n string as
substring. alphabet = [0, k - 1]
int res[10000]; // >= k^n
int aux[10000]; // >= k*n
int de_bruijn(int k, int n) { // Returns size (k^n)
  if(k == 1) {
    res[0] = 0;
    return 1;
  }
  for(int i = 0; i < k * n; i++)
    aux[i] = 0;
  int sz = 0;
  function<void(int, int)> db = [&](int t, int p) {
    if(t > n) {
      if(n % p == 0)
        for(int i = 1; i <= p; i++)
          res[sz++] = aux[i];
    }
    else {
      aux[t] = aux[t - p];
      db(t + 1, p);
      for(int i = aux[t - p] + 1; i < k; i++) {
        aux[t] = i;
        db(t + 1, t);
      }
    }
  };
  db(1, 1);
  return sz;
}
```

4.7 Extended GCD

```
namespace Euclid{
  lint gcd(lint x, lint y) { return y ? gcd(y, x%y) : x; }
  lint mod(lint a, lint b) { return ((a%b) + b) % b; }

  // returns g = gcd(a, b); finds x, y such that g = ax + by
  lint ext_gcd(lint a, lint b, lint &x, lint &y) {
    lint xx = y = 0;
    lint yy = x = 1;
    while (b) {
      lint q = a / b;
      lint t = b; b = a%b; a = t;
      t = xx; xx = x - q*xx; x = t;
      t = yy; yy = y - q*yy; y = t;
    }
    return a;
  }

  // computes b such that ab = 1 (mod n), returns -1 on failure
  lint mod_inverse(lint a, lint n) {
    lint x, y;
    lint g = ext_gcd(a, n, x, y);
    if (g > 1) return -1;
    return mod(x, n);
  }

  // Chinese remainder theorem: find z such that
  // z % m1 = r1, z % m2 = r2.  Here, z is unique modulo M = lcm(m1, m2).
  // Return (z, M).  On failure, M = -1.
  pair<lint, lint> CRT(lint m1, lint r1, lint m2, lint r2) {
    lint s, t;
    lint g = ext_gcd(m1, m2, s, t);
    if (r1%g != r2%g) return make_pair(0, -1);
    s = mod(s * r2, m2);
    t = mod(t * r1, m1);
    return make_pair(mod(s*(m1/g) + t*(m2/g), m1*(m2/g)), m1*(m2/g));
  }
}
```

4.8 Discrete Log

```
namespace discrete_log{
  lint product(lint x, lint y, lint MOD) {
    return x * y % MOD;
```

```
  }

  lint mod_pow(lint x, lint k, lint MOD) {
    if (!k) return 1;
    if (k&1) return product(x, mod_pow(x, k - 1, MOD), MOD);
    return mod_pow(product(x, x, MOD), k / 2, MOD);
  }

  lint totient(lint v) {
    lint tot = v;
    for (lint p = 2; p * p <= v; p++) if (v % p == 0) {
      tot = tot / p * (p - 1);
      while (v % p == 0) v /= p;
    }
    if (v > 1) tot = tot / v * (v - 1);
    return tot;
  }

  // https://judge.yosupo.jp/submission/32236
  /* Returns the smallest K >= 0 such that init * pow(x, K) === y modulo MOD.
   * Returns -1 if no such K exists. Runs in O(sqrt(MOD)).
   0 <= x, y < MOD, MOD not necessarily have to be prime
   */
  lint solve(lint x, lint y, lint MOD, lint init = 1) {
    if (x == 0)
      return y == 1 ? 0 : y == 0 ? MOD > 1 : -1;

    lint prefix = 0;
    while (init != y && gcd(init, MOD) != gcd(product(init, x, MOD), MOD)) {
      init = product(init, x, MOD);
      prefix++;
    }

    if (init == y)
      return prefix;

    if (gcd(init, MOD) != gcd(y, MOD))
      return -1;

    MOD = MOD / gcd(init, MOD);

    x %= MOD;
    y %= MOD;
    init %= MOD;

    lint subgroup_order = totient(MOD);

    y = product(y, mod_pow(init, subgroup_order - 1, MOD), MOD);

    lint step_size = 0;
    while (step_size * step_size < subgroup_order)
      step_size++;

    unordered_map<lint, lint> table;

    lint baby_step = 1;
    for (lint i = 0; i < step_size; i++) {
      table[baby_step] = i;
      baby_step = product(baby_step, x, MOD);
    }

    lint giant_step = mod_pow(x, subgroup_order - step_size, MOD);
    for (lint i = 0; i < step_size; i++) {
      auto it = table.find(y);
      if (it != table.end())
        return prefix + i * step_size + it->second;
      y = product(y, giant_step, MOD);
    }

    return -1;
  }
}
```

4.9 Discrete Kth Root

```
//https://judge.yosupo.jp/submission/23481
// p need to be prime.
// find solution to x^k == a mod p

namespace kth_root_mod {
    template <typename T>
        struct Memo {
            Memo(const T &g, int s, int _period){
                size = 1;
                while(2 * size <= min(s, _period)) size *= 2;
                mask = size - 1;
                period = _period;
                vs.resize(size);
                os.resize(size + 1);
                T x(1);
                for (int i = 0; i < size; ++i, x *= g) os[((lint)x) & mask]++;
                for (int i = 1; i < size; ++i) os[i] += os[i - 1];
                x = 1;
                for (int i = 0; i < size; ++i, x *= g) vs[--os[((lint)x) & mask]] =
                {x, i};
                gpow = x;
                os[size] = size;
            }
            int find(T x) const {
                for (int t = 0; t < period; t += size, x *= gpow) {
                    for (int m = (((lint)x) & mask), i = os[m]; i < os[m + 1]; ++i) {
                        if (x == vs[i].first) {
                            int ret = vs[i].second - t;
                            return ret < 0 ? ret + period : ret;
                        }
                    }
                }
                assert(0);
            }
            T gpow;
            int size, mask, period;
            vector<pair<T, int> > vs;
            vector<int> os;
        };


    lint inv(lint a, lint p) {
        lint b = p, x = 1, y = 0;
        while (a) {
            lint q = b / a;
            swap(a, b %= a);
            swap(x, y -= q * x);
        }
        assert(b == 1);
        return y < 0 ? y + p : y;
    }


    mint pe_root(lint c, lint pi, lint ei, lint p) {
        lint s = p - 1, t = 0;
        while (s % pi == 0) s /= pi, ++t;
        lint pe = 1;
        for (lint _ = 0; _ < ei; ++_) pe *= pi;

        lint u = inv(pe - s % pe, pe);
        mint mc = c, one = 1;
        mint z = ipow(mc, (s * u + 1) / pe);
        mint zpe = ipow(mc, s * u);
        if (zpe == one) return z;

        mint vs;
        {
            lint ptm1 = 1;
            for (lint _ = 0; _ < t - 1; ++_) ptm1 *= pi;
            for (mint v = 2;; v += one) {
                vs = ipow(v, s);
                if(ipow(vs, ptm1) != one) break;
            }
        }

        mint vspe = ipow(vs, pe);
        lint vs_e = ei;
        mint base = vspe;
        for (lint _ = 0; _ < t - ei - 1; _++) base = ipow(base, pi);
        Memo<mint> memo(base, (lint)(sqrt(t - ei) * sqrt(pi)) + 1, pi);

        while (zpe != one) {
            mint tmp = zpe;
            lint td = 0;
            while (tmp != one) ++td, tmp = ipow(tmp, pi);
            lint e = t - td;
            while (vs_e != e) {
                vs = ipow(vs, pi);
                vspe = ipow(vspe, pi);
                ++vs_e;
            }

            // BS-GS ... find (zpe * ( vspe ^ n ) ) ^( p_i ^ (td - 1) ) = 1
            mint base_zpe = mint(1) / zpe;
            for (lint _ = 0; _ < td - 1; _++) base_zpe = ipow(base_zpe, pi);
            lint bsgs = memo.find(base_zpe);

            z *= ipow(vs, bsgs);
            zpe *= ipow(vspe, bsgs);
        }
        return z;
    }

    lint kth_root(lint a, lint k, lint p) {
        mod = p;
        a %= p;
        if (k == 0) return a == 1 ? a : -1;
        if (a <= 1 || k <= 1) return a;

        assert(p > 2);
        lint g = gcd(p - 1, k);
        if (ipow(mint(a), (p - 1) / g) != mint(1)) return -1;
        a = (lint)ipow(mint(a), inv(k / g, (p - 1) / g));
        unordered_map<lint, int> fac;
        for (auto &f : factors::factorize(g)) fac[f]++;
        for (auto pp : fac)
            a = pe_root(a, pp.first, pp.second, p);
        return a;
    }
}
```

4.10 Factorization

```
lint mul(lint x, lint y, lint mod){ return (__int128) x * y % mod; }

lint ipow(lint x, lint y, lint p){
  lint ret = 1, piv = x % p;
  while(y){
    if(y&1) ret = mul(ret, piv, p);
    piv = mul(piv, piv, p);
    y >>= 1;
  }
  return ret;
}

namespace factors{
  bool miller_rabin(lint x, lint a){
    if(x % a == 0) return 0;
    lint d = x - 1;
    while(1){
      lint tmp = ipow(a, d, x);
      if(d&1) return (tmp != 1 && tmp != x-1);
      else if(tmp == x-1) return 0;
      d >>= 1;
    }
  }
  bool isprime(lint x){
    for(auto &i : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){
```

```
      if(x == i) return 1;
      if(x > 40 && miller_rabin(x, i)) return 0;
    }
    if(x <= 40) return 0;
    return 1;
  }
  lint f(lint x, lint n, lint c){
    return (c + mul(x, x, n)) % n;
  }
  void rec(lint n, vector<lint> &v){
    if(n == 1) return;
    if(n % 2 == 0){
      v.push_back(2);
      rec(n/2, v);
      return;
    }
    if(isprime(n)){
      v.push_back(n);
      return;
    }
    lint a, b, c;
    while(1){
      a = rand() % (n-2) + 2;
      b = a;
      c = rand() % 20 + 1;
      do{
        a = f(a, n, c);
        b = f(f(b, n, c), n, c);
      }while(gcd(abs(a-b), n) == 1);
      if(a != b) break;
    }
    lint x = gcd(abs(a-b), n);
    rec(x, v);
    rec(n/x, v);
  }
  vector<lint> factorize(lint n){
    vector<lint> ret;
    rec(n, ret);
    sort(ret.begin(), ret.end());
    return ret;
  }
  lint euler_phi(lint n){
    auto pf = factorize(n);
    pf.resize(unique(all(pf)) - pf.begin());
    for(auto &p : pf){
      n -= n / p;
    }
    return n;
  }
};
```

4.11 Nim Multiplication

```
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;

namespace nimbers {
    constexpr u32 n2f[16] = {0x0001u, 0x071cu, 0x6bd1u, 0x1224u, 0x6ba8u, 0x1333u,
    0x1553u, 0x0007u, 0x071eu, 0x0925u, 0xc586u, 0x5dbdu, 0xc463u, 0x5efdu, 0x2aa1u,
    0x155au},
                  f2n[16] = {0x0001u, 0x0102u, 0x0183u, 0x8041u, 0x015cu, 0x5f24u,
                  0xde2cu, 0x957eu, 0x01f4u, 0xf7d8u, 0x76b0u, 0x5d52u, 0xa977u,
                  0x20d1u, 0xc1a4u, 0x271fu};
    inline u32 nimber2field(u32 x) {u32 y = 0; for (; x; x &= x - 1) y ^=
    n2f[__builtin_ctz(x)]; return y;}
    inline u32 field2nimber(u32 x) {u32 y = 0; for (; x; x &= x - 1) y ^=
    f2n[__builtin_ctz(x)]; return y;}
    inline u32 __builtin_double(u32 x) {return x << 1 ^ (x < 32768 ? 0 : 0x16babu);}

    u16 ln[65536], exp[196605], *Hexp = exp + 62133, *H2exp = exp + 58731;

    inline void init() {
        int i;
        for (*exp = i = 1; i < 65535; ++i) exp[i] = __builtin_double(exp[i - 1]);
        for (i = 1; i < 65535; ++i) exp[i] = field2nimber(exp[i]), ln[exp[i]] = i;
        memcpy(exp + 65535, exp, 131070);
        memcpy(exp + 131070, exp, 131070);
    }

    inline u16 product(u16 A, u16 B) {return A && B ? exp[ln[A] + ln[B]] : 0;}
    inline u16 H(u16 A) {return A ? Hexp[ln[A]] : 0;}
    inline u16 H2(u16 A) {return A ? H2exp[ln[A]] : 0;}
    inline u16 Hproduct(u16 A, u16 B) {return A && B ? Hexp[ln[A] + ln[B]] : 0;}

    inline u32 product(u32 A, u32 B) {
        u16 a = A & 65535, b = B & 65535, c = A >> 16, d = B >> 16, e = product(a,
        b);
        return u32(product(u16(a ^ c), u16(b ^ d)) ^ e) << 16 | (Hproduct(c, d) ^ e);
    }

    inline u32 H(u32 A) {
        u16 a = A & 65535, b = A >> 16;
        return H(u16(a ^ b)) << 16 | H2(b);
    }

    inline u64 product(u64 A, u64 B) {
        u32 a = A & UINT_MAX, b = B & UINT_MAX, c = A >> 32, d = B >> 32, e =
        product(a, b);
        return u64(product(a ^ c, b ^ d) ^ e) << 32 | (H(product(c, d)) ^ e);
    }
}
```

4.12 Partition Number

```
// answer is P[n + 1]
vector<pair<int, int>> gp;
lint P[MAXN+1] = {};
gp.emplace_back(0, 0);
for(int i = 1; gp.back().second <= MAXN; i++) {
  gp.emplace_back(i % 2 ? 1 : -1, i * (3*i - 1) / 2);
  gp.emplace_back(i % 2 ? 1 : -1, i * (3*i + 1) / 2);
}
P[1] = 1;
for(int n = 2; n <= MAXN; n++) {
  for(auto it : gp) if(n >= it.second) P[n] += P[n - it.second] * it.first + MOD;
  P[n] %= MOD;
}
```

4.13 Prime Counting

```
vector<bool> prime;
vector<int> primes;
vector<int> prime_count;

void sieve(int maximum) {
    maximum = max(maximum, 2);
    prime.assign(maximum + 1, true);
    prime[0] = prime[1] = false;
    primes = {};

    for (long long p = 2; p <= maximum; p++)
        if (prime[p]) {
            primes.push_back(p);
            for (long long i = p * p; i <= maximum; i += p)
                prime[i] = false;
        }
    prime_count.assign(maximum + 1, 0);
    for (int i = 1; i <= maximum; i++)
        prime_count[i] = prime_count[i - 1] + prime[i];
}

const int K_MEMO = 50;
const int N_MEMO = 1e5;

// Warning: if N_MEMO >= 2^17, this must be changed to a 32-bit integer.
uint16_t memo[K_MEMO][N_MEMO];

long long count_non_multiples(long long n, int k) {
    if (n <= 1 || k < 0) return n;
    long long p = primes[k];
    if (n <= p)
        return 1;
    if (n < (int) prime.size() && n <= p * p)
        return prime_count[n] - k;
    bool save = k < K_MEMO && n < N_MEMO;
    if (save && memo[k][n])
        return memo[k][n];
    long long ret = count_non_multiples(n, k - 1) - count_non_multiples(n / p, k -
    1);
    if (save)
        memo[k][n] = ret;
    return ret;
}

// Returns the count of primes <= n. Runs in O(n^(3/4) / log n).
long long count_primes(long long n) {
    if (n < (int) prime.size())
        return prime_count[n];
    int s = (int) sqrt(n) + 1;
    assert(s < (int) prime.size());
    int k = prime_count[s];
    return count_non_multiples(n, k) + k;
}
```

4.14 Stern Brocot Tree

```
#define x first
#define y second
pair<long long, long long> solve(pair<int, int> L, pair<int, int> R){
//  printf("%.10Lf %.10Lf\n", L, R);
    pair<long long, long long> l(0, 1), r(1, 0);
    for(;;){
        pair<long long, long long> m(l.x+r.x, l.y+r.y);
        fflush(stdout);
        if(L.first*m.x<=L.second*m.y){// move to the right;
            long long kl=1, kr=1;
            while(L.first*(l.x+kr*r.x) <= L.second*(l.y+kr*r.y)) kr*=2;// exponential
            search
            while(kl!=kr){
                long long km = (kl+kr)/2;
                if(L.first*(l.x+km*r.x) <= L.second*(l.y+km*r.y)) kl=km+1;
                else kr=km;
            }
            l = make_pair(l.x+(kl-1)*r.x, l.y+(kl-1)*r.y);
        } else if (R.first*m.x >= R.second*m.y){//move to the left
            long long kl=1, kr=1;
            while(R.first*(r.x+kr*l.x) >= R.second*(r.y+kr*l.y))kr*=2;// exponential
            search
            while(kl!=kr){
                long long km = (kl+kr)/2;
                if(R.first*(r.x+km*l.x) >= R.second*(r.y+km*l.y)) kl = km+1;
                else kr = km;
            }
            r = make_pair(r.x+(kl-1)*l.x, r.y+(kl-1)*l.y);
        } else {
            return m;
        }
    }
}


// x < ans < y, all argument should be nonnegative, x != pi(0, 0), y != pi(0, 0).
// minimize y, in case of tie x.

pi minY(pi x, pi y){
  auto fuck =  solve(x, y);
  return pi(fuck.second, fuck.first);
}
```

4.15 Tetration

```
// a[0]^(a[1]^(a[2]^...)) mod positive integer. assumes all a[i] > 0

lint tetration(vector<lint> a, lint mod) {
  if(sz(a) > 128) a.resize(128); // 2 * log(maxA)
  vector<lint> mod_chain = {mod};
```

```
  while(mod_chain.back() > 1){
    lint newVal = factors::euler_phi(mod_chain.back());
    mod_chain.push_back(newVal);
  }
  int nsz = min(sz(a), sz(mod_chain));
  a.resize(nsz);
  mod_chain.resize(nsz);
  lint v = 1;
  auto ipow = [&](lint x, lint n, lint mod) -> lint {
    if (x >= mod) x = x % mod + mod;
    lint v = 1;
    do {
      if (n & 1) {
        v *= x;
        if (v >= mod) v = v % mod + mod;
      }
      x *= x;
      if (x >= mod) x = x % mod + mod;
      n /= 2;
    } while (n);
    return v;
  };

  for(int i = sz(a) - 1; i >= 0; i--){
    v = ipow(a[i], v, mod_chain[i]);
  }
  return v % mod;
}
```

4.16 Xudyh Sieve

```
// given multiplicative function f, and g where prefix sum of g and f * g is easy,
find the prefix sum of f
namespace moe{
  lint moe[MAXN], sum[MAXN];
  lint inv;
  lint f(lint x){  return moe[x]; }
  lint gs(lint x){ return x; }
  lint fgs(lint x){ return 1; }
  // if you take f(x) = mobius(x), g(x) = 1, then h(x) = sum f * g(x)  = 1
  void init(){
    moe[1] = 1;
    for(int i=1; i<MAXN; i++){
      for(int j=2*i; j<MAXN; j+=i){
        moe[j] -= moe[i];
      }
    }
    inv = gs(1);
    for(int i=1; i<MAXN; i++){
      sum[i] = sum[i-1] + f(i);
    }
  }
  unordered_map<lint, lint> mp;
  lint query(lint x){
    if(x < MAXN) return sum[x];
    if(mp.find(x) != mp.end()) return mp[x];
    lint ans = fgs(x);
    for(lint i=2; i<=x; ){
      lint cur = x / (x / i);
      ans -= (gs(cur) - gs(i - 1)) * query(x / i);
      i = cur + 1;
    }
    ans /= inv;
    return mp[x] = ans;
  }
};
```

5 Geometry

5.1 Convex Hull Trick

```
using line_t = double;
const line_t is_query = -1e18;

struct Line {
  line_t 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;
    line_t x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};

struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
  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;
    return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
  }
  void insert_line(line_t m, line_t b) {
    auto y = insert({ m, b });
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }
  line_t query(line_t x) {
    auto l = *lower_bound((Line) { x, is_query });
    return l.m * x + l.b;
  }
}H;
```

5.2 3D Convex Hull

```
using P3 = array<lint,3>;
P3& operator+=(P3& l, const P3& r) { for(int i = 0; i < 3; i++) l[i] += r[i];
  return l; }
P3& operator-=(P3& l, const P3& r) { for(int i = 0; i < 3; i++) l[i] -= r[i];
  return l; }
P3& operator*=(P3& l, const lint& r) { for(int i = 0; i < 3; i++) l[i] *= r;
  return l; }
P3& operator/=(P3& l, const lint& r) { for(int i = 0; i < 3; i++) l[i] /= r;
  return l; }
P3 operator-(P3 l) { l *= -1; return l; }
P3 operator+(P3 l, const P3& r) { return l += r; }
P3 operator-(P3 l, const P3& r) { return l -= r; }
P3 operator*(P3 l, const lint& r) { return l *= r; }
P3 operator/(P3 l, const lint& r) { return l /= r; }

lint dot(const P3& a, const P3& b) {
  lint sum = 0; for(int i = 0; i < 3; i++) sum += a[i]*b[i];
  return sum; }
P3 cross(const P3& a, const P3& b) {
  return {a[1]*b[2]-a[2]*b[1],a[2]*b[0]-a[0]*b[2],
      a[0]*b[1]-a[1]*b[0]}; }
P3 cross(const P3& a, const P3& b, const P3& c) {
  return cross(b-a,c-a); }

bool isMult(const P3& a, const P3& b) { // for long longs
  P3 c = cross(a,b); for(int i = 0; i < sz(c); i++) if (c[i] != 0) return 0;
  return 1; }
bool collinear(const P3& a, const P3& b, const P3& c) {
  return isMult(b-a,c-a); }

lint DC(const P3&a,const P3&b,const P3&c,const P3&p) {
  return dot(cross(a,b,c),p-a); }
bool coplanar(const P3&a,const P3&b,const P3&c,const P3&p) {
  return DC(a,b,c,p) == 0; }

lint above(const P3&a,const P3&b,const P3&c,const P3&p) {
  return DC(a,b,c,p) > 0;
} // is p strictly above plane

void prep(vector<P3>& p) { // rearrange points such that
  shuffle(all(p), mt19937(0x14004));
  int dim = 1;
  for(int i = 1; i < sz(p); i++){
    if (dim == 1) {
      if (p[0] != p[i]) swap(p[1],p[i]), ++dim;
    } else if (dim == 2) {
      if (!collinear(p[0],p[1],p[i]))
        swap(p[2],p[i]), ++dim;
    } else if (dim == 3) {
      if (!coplanar(p[0],p[1],p[2],p[i]))
        swap(p[3],p[i]), ++dim;
    }
  }
    assert(dim == 4);
}

using F = array<int,3>; // face
vector<F> hull3dFast(vector<P3>& p) {
  prep(p);
  int N = sz(p); vector<F> hull;
  vector<int> active; vector<vector<int>> rvis; vector<array<pi,3>> other;
  // whether face is active
  // points visible from each face
  // other face adjacent to each edge of face
  vector<vector<int>> vis(N); // faces visible from each point
  auto ad = [&](int a, int b, int c) {
    hull.push_back({a,b,c}); active.push_back(1); rvis.emplace_back();
    other.emplace_back();
  };
  auto ae = [&](int a, int b) { vis[b].push_back(a), rvis[a].push_back(b); };
  auto abv = [&](int a, int b) {
    F f=hull[a]; return above(p[f[0]],p[f[1]],p[f[2]],p[b]);};
  auto edge = [&](pi e) -> pi {
    return {hull[e.first][e.second],hull[e.first][(e.second+1)%3]}; };
  auto glue = [&](pi a, pi b) { // link two faces by an edge
    pi x = edge(a); assert(edge(b) == pi(x.second,x.first));
    other[a.first][a.second] = b;
    other[b.first][b.second] = a;
  }; // ensure face 0 is removed when i=3
  ad(0,1,2), ad(0,2,1);
  if (abv(1,3)) swap(p[1],p[2]);

  for(int i = 0; i < 3; i++) glue({0,i},{1,2-i});
  for(int i = 3; i < N; i++) ae(abv(1,i),i); // coplanar points go in rvis[0]
  vector<int> label(N,-1);
  for(int i = 3; i < N; i++){ // incremental construction
    vector<int> rem;
    for(auto &t : vis[i]) if (active[t]){
      active[t]=0, rem.push_back(t);
    }
    if (!sz(rem)) continue; // hull unchanged
    int st = -1;
    for(auto &r : rem){
      for(int j = 0; j < 3; j++){
        int o = other[r][j].first;
        if (active[o]) { // create new face!
          int a,b; tie(a,b) = edge({r,j}); ad(a,b,i); st = a;
          int cur = sz(rvis)-1; label[a] = cur;
          vector<int> tmp; set_union(all(rvis[r]),all(rvis[o]),
              back_inserter(tmp));
          // merge sorted vectors ignoring duplicates
          for(auto &x : tmp) if (abv(cur,x)) ae(cur,x);
          /// if no rounding errors then guaranteed that only x>i matters
          glue({cur,0},other[r][j]); // glue old w/ new face
        }
      }
    }
    for (int x = st, y; ; x = y) { // glue new faces together
      int X = label[x]; glue({X,1},{label[y=hull[X][1]],2});
      if (y == st) break;
    }
  }
  vector<F> ans;
```

```
  for(int i = 0; i < sz(hull); i++){
    if(active[i]) ans.push_back(hull[i]);
  }
  return ans;
}
```

5.3 Delaunay

```
using ll = long long;

bool ge(const ll& a, const ll& b) { return a >= b; }
bool le(const ll& a, const ll& b) { return a <= b; }
bool eq(const ll& a, const ll& b) { return a == b; }
bool gt(const ll& a, const ll& b) { return a > b; }
bool lt(const ll& a, const ll& b) { return a < b; }
int sgn(const ll& a) { return a >= 0 ? a ? 1 : 0 : -1; }

struct pt {
  ll x, y;
  pt() { }
  pt(ll _x, ll _y) : x(_x), y(_y) { }
  pt operator-(const pt& p) const {
    return pt(x - p.x, y - p.y);
  }
  ll cross(const pt& p) const {
    return x * p.y - y * p.x;
  }
  ll cross(const pt& a, const pt& b) const {
    return (a - *this).cross(b - *this);
  }
  ll dot(const pt& p) const {
    return x * p.x + y * p.y;
  }
  ll dot(const pt& a, const pt& b) const {
    return (a - *this).dot(b - *this);
  }
  ll sqrLength() const {
    return this->dot(*this);
  }
  bool operator==(const pt& p) const {
    return eq(x, p.x) && eq(y, p.y);
  }
};

const pt inf_pt = pt(1e18, 1e18);

struct QuadEdge {
  pt origin;
  QuadEdge* rot = nullptr;
  QuadEdge* onext = nullptr;
  bool used = false;
  QuadEdge* rev() const {
    return rot->rot;
  }
  QuadEdge* lnext() const {
    return rot->rev()->onext->rot;
  }
  QuadEdge* oprev() const {
    return rot->onext->rot;
  }
  pt dest() const {
    return rev()->origin;
  }
};

QuadEdge* make_edge(pt from, pt to) {
  QuadEdge* e1 = new QuadEdge;
  QuadEdge* e2 = new QuadEdge;
  QuadEdge* e3 = new QuadEdge;
  QuadEdge* e4 = new QuadEdge;
  e1->origin = from;
  e2->origin = to;
  e3->origin = e4->origin = inf_pt;
  e1->rot = e3;
  e2->rot = e4;
  e3->rot = e2;
  e4->rot = e1;
  e1->onext = e1;
  e2->onext = e2;
  e3->onext = e4;
  e4->onext = e3;
  return e1;
}

void splice(QuadEdge* a, QuadEdge* b) {
  swap(a->onext->rot->onext, b->onext->rot->onext);
  swap(a->onext, b->onext);
}

void delete_edge(QuadEdge* e) {
  splice(e, e->oprev());
  splice(e->rev(), e->rev()->oprev());
  delete e->rev()->rot;
  delete e->rev();
  delete e->rot;
  delete e;
}

QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
  QuadEdge* e = make_edge(a->dest(), b->origin);
  splice(e, a->lnext());
  splice(e->rev(), b);
  return e;
}

bool left_of(pt p, QuadEdge* e) {
  return gt(p.cross(e->origin, e->dest()), 0);
}

bool right_of(pt p, QuadEdge* e) {
  return lt(p.cross(e->origin, e->dest()), 0);
}

template <class T>
T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2, T c3) {
  return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) +
         a3 * (b1 * c2 - c1 * b2);
}

bool in_circle(pt a, pt b, pt c, pt d) {
  long double det = -det3<long double>(b.x, b.y, b.sqrLength(), c.x, c.y,
      c.sqrLength(), d.x, d.y, d.sqrLength());
  det += det3<long double>(a.x, a.y, a.sqrLength(), c.x, c.y, c.sqrLength(), d.x,
      d.y, d.sqrLength());
  det -= det3<long double>(a.x, a.y, a.sqrLength(), b.x, b.y, b.sqrLength(), d.x,
      d.y, d.sqrLength());
  det += det3<long double>(a.x, a.y, a.sqrLength(), b.x, b.y, b.sqrLength(), c.x,
      c.y, c.sqrLength());
  if(fabs(det) > 1e18) return det > 0;
  else{

    ll det = -det3<ll>(b.x, b.y, b.sqrLength(), c.x, c.y,
        c.sqrLength(), d.x, d.y, d.sqrLength());
    det += det3<ll>(a.x, a.y, a.sqrLength(), c.x, c.y, c.sqrLength(), d.x,
        d.y, d.sqrLength());
    det -= det3<ll>(a.x, a.y, a.sqrLength(), b.x, b.y, b.sqrLength(), d.x,
        d.y, d.sqrLength());
    det += det3<ll>(a.x, a.y, a.sqrLength(), b.x, b.y, b.sqrLength(), c.x,
        c.y, c.sqrLength());
    return (det > 0);
  }
}

pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<pt>& p) {
  if (r - l + 1 == 2) {
    QuadEdge* res = make_edge(p[l], p[r]);
    return make_pair(res, res->rev());
  }
  if (r - l + 1 == 3) {
    QuadEdge *a = make_edge(p[l], p[l + 1]), *b = make_edge(p[l + 1], p[r]);
    splice(a->rev(), b);
    int sg = sgn(p[l].cross(p[l + 1], p[r]));
    if (sg == 0)
      return make_pair(a, b->rev());
    QuadEdge* c = connect(b, a);
    if (sg == 1)
      return make_pair(a, b->rev());
    else
      return make_pair(c->rev(), c);
  }
  int mid = (l + r) / 2;
  QuadEdge *ldo, *ldi, *rdo, *rdi;
  tie(ldo, ldi) = build_tr(l, mid, p);
  tie(rdi, rdo) = build_tr(mid + 1, r, p);
  while (true) {
    if (left_of(rdi->origin, ldi)) {
      ldi = ldi->lnext();
      continue;
    }
    if (right_of(ldi->origin, rdi)) {
      rdi = rdi->rev()->onext;
      continue;
    }
    break;
  }
  QuadEdge* basel = connect(rdi->rev(), ldi);
  auto valid = [&basel](QuadEdge* e) { return right_of(e->dest(), basel); };
  if (ldi->origin == ldo->origin)
    ldo = basel->rev();
  if (rdi->origin == rdo->origin)
    rdo = basel;
  while (true) {
    QuadEdge* lcand = basel->rev()->onext;
    if (valid(lcand)) {
      while (in_circle(basel->dest(), basel->origin, lcand->dest(),
                lcand->onext->dest())) {
        QuadEdge* t = lcand->onext;
        delete_edge(lcand);
        lcand = t;
      }
    }
    QuadEdge* rcand = basel->oprev();
    if (valid(rcand)) {
      while (in_circle(basel->dest(), basel->origin, rcand->dest(),
                rcand->oprev()->dest())) {
        QuadEdge* t = rcand->oprev();
        delete_edge(rcand);
        rcand = t;
      }
    }
    if (!valid(lcand) && !valid(rcand))
      break;
    if (!valid(lcand) ||
      (valid(rcand) && in_circle(lcand->dest(), lcand->origin,
                    rcand->origin, rcand->dest())))
      basel = connect(rcand, basel->rev());
    else
      basel = connect(basel->rev(), lcand->rev());
  }
  return make_pair(ldo, rdo);
}

vector<tuple<pt, pt, pt>> delaunay(vector<pt> p) {
  sort(p.begin(), p.end(), [](const pt& a, const pt& b) {
    return lt(a.x, b.x) || (eq(a.x, b.x) && lt(a.y, b.y));
  });
  auto res = build_tr(0, (int)p.size() - 1, p);
  QuadEdge* e = res.first;
  vector<QuadEdge*> edges = {e};
  while (lt(e->onext->dest().cross(e->dest(), e->origin), 0))
    e = e->onext;
  auto add = [&p, &e, &edges]() {
```

```
    QuadEdge* curr = e;
    do {
      curr->used = true;
      p.push_back(curr->origin);
      edges.push_back(curr->rev());
      curr = curr->lnext();
    } while (curr != e);
  };
  add();
  p.clear();
  int kek = 0;
  while (kek < (int)edges.size()) {
    if (!(e = edges[kek++])->used)
      add();
  }
  vector<tuple<pt, pt, pt>> ans;
  for (int i = 0; i < (int)p.size(); i += 3) {
    ans.push_back(make_tuple(p[i], p[i + 1], p[i + 2]));
  }
  return ans;
}
```

5.4 Halfplane Intersection

```
const double eps = 1e-8;
typedef pair<long double, long double> pi;
bool z(long double x){ return fabs(x) < eps; }
struct line{
  long double a, b, c;
  bool operator<(const line &l)const{
    bool flag1 = pi(a, b) > pi(0, 0);
    bool flag2 = pi(l.a, l.b) > pi(0, 0);
    if(flag1 != flag2) return flag1 > flag2;
    long double t = ccw(pi(0, 0), pi(a, b), pi(l.a, l.b));
    return z(t) ? c * hypot(l.a, l.b) < l.c * hypot(a, b) : t > 0;
  }
  pi slope(){ return pi(a, b); }
};
pi cross(line a, line b){
  long double det = a.a * b.b - b.a * a.b;
  return pi((a.c * b.b - a.b * b.c) / det, (a.a * b.c - a.c * b.a) / det);
}
bool bad(line a, line b, line c){
  if(ccw(pi(0, 0), a.slope(), b.slope()) <= 0) return false;
  pi crs = cross(a, b);
  return crs.first * c.a + crs.second * c.b >= c.c;
}
bool solve(vector<line> v, vector<pi> &solution){ // ax + by <= c;
  sort(v.begin(), v.end());
  deque<line> dq;
  for(auto &i : v){
    if(!dq.empty() && z(ccw(pi(0, 0), dq.back().slope(), i.slope()))) continue;
    while(dq.size() >= 2 && bad(dq[dq.size()-2], dq.back(), i)) dq.pop_back();
    while(dq.size() >= 2 && bad(i, dq[0], dq[1])) dq.pop_front();
    dq.push_back(i);
  }
  while(dq.size() > 2 && bad(dq[dq.size()-2], dq.back(), dq[0])) dq.pop_back();
  while(dq.size() > 2 && bad(dq.back(), dq[0], dq[1])) dq.pop_front();
  vector<pi> tmp;
  for(int i=0; i<dq.size(); i++){
    line cur = dq[i], nxt = dq[(i+1)%dq.size()];
    if(ccw(pi(0, 0), cur.slope(), nxt.slope()) <= eps) return false;
    tmp.push_back(cross(cur, nxt));
  }
  solution = tmp;
  return true;
}
```

5.5 Smallest Enclosing Circle

```
namespace cover_2d{
  double eps = 1e-9;
  using Point = complex<double>;
  struct Circle{ Point p; double r; };
  double dist(Point p, Point q){ return abs(p-q); }
  double area2(Point p, Point q){ return (conj(p)*q).imag(); }
  bool in(const Circle& c, Point p){ return dist(c.p, p) < c.r + eps; }
  Circle INVAL = Circle{Point(0, 0), -1};
  Circle mCC(Point a, Point b, Point c){
    b -= a; c -= a;
    double d = 2*(conj(b)*c).imag(); if(abs(d)<eps) return INVAL;
    Point ans = (c*norm(b) - b*norm(c)) * Point(0, -1) / d;
    return Circle{a + ans, abs(ans)};
  }
  Circle solve(vector<Point> p) {
    mt19937 gen(0x94949); shuffle(p.begin(), p.end(), gen);
    Circle c = INVAL;
    for(int i=0; i<p.size(); ++i) if(c.r<0 ||!in(c, p[i])){
      c = Circle{p[i], 0};
      for(int j=0; j<=i; ++j) if(!in(c, p[j])){
        Circle ans{(p[i]+p[j])*0.5, dist(p[i], p[j])*0.5};
        if(c.r == 0) { c = ans; continue; }
        Circle l, r; l = r = INVAL;
        Point pq = p[j]-p[i];
        for(int k=0; k<=j; ++k) if(!in(ans, p[k])) {
          double a2 = area2(pq, p[k]-p[i]);
          Circle c = mCC(p[i], p[j], p[k]);
          if(c.r<0) continue;
          else if(a2 > 0 && (l.r<0||area2(pq, c.p-p[i]) > area2(pq, l.p-p[i]))) l =
          c;
          else if(a2 < 0 && (r.r<0||area2(pq, c.p-p[i]) < area2(pq, r.p-p[i]))) r =
          c;
        }
        if(l.r<0&&r.r<0) c = ans;
        else if(l.r<0) c = r;
        else if(r.r<0) c = l;
        else c = l.r<=r.r?l:r;
      }
    }
    return c;
  }
};
```

```
namespace cover_3d{
  double enclosing_sphere(vector<double> x, vector<double> y, vector<double> z){
    int n = x.size();
    auto hyp = [](double x, double y, double z){
      return x * x + y * y + z * z;
    };
    double px = 0, py = 0, pz = 0;
    for(int i=0; i<n; i++){
      px += x[i];
      py += y[i];
      pz += z[i];
    }
    px *= 1.0 / n;
    py *= 1.0 / n;
    pz *= 1.0 / n;
    double rat = 0.1, maxv;
    for(int i=0; i<10000; i++){
      maxv = -1;
      int maxp = -1;
      for(int j=0; j<n; j++){
        double tmp = hyp(x[j] - px, y[j] - py, z[j] - pz);
        if(maxv < tmp){
          maxv = tmp;
          maxp = j;
        }
      }
      px += (x[maxp] - px) * rat;
      py += (y[maxp] - py) * rat;
      pz += (z[maxp] - pz) * rat;
      rat *= 0.998;
    }
    return sqrt(maxv);
  }
};
```

5.6 Tangent on Convex Polygon

```
// by zigui
// C : counter_clockwise(C[0] == C[N]), N >= 2
// return highest point in C <- P(clockwise) or -1 if strictly in convex
// recommend : strongly convex, C[i] != P

int convex_tangent(vector<pi> &C, pi P, int up = 1){
  auto sign = [&](lint c){ return c > 0 ? up : c == 0 ? 0 : -up; };
  auto local = [&](pi P, pi a, pi b, pi c) {
    return sign(ccw(P, a, b)) <= 0 && sign(ccw(P, b, c)) >= 0;
  };
  int N = C.size()-1, s = 0, e = N, m;
  if( local(P, C[1], C[0], C[N-1]) ) return 0;
  while(s+1 < e){
    m = (s+e) / 2;
    if( local(P, C[m-1], C[m], C[m+1]) ) return m;
    if( sign(ccw(P, C[s], C[s+1])) < 0){ // up
      if( sign(ccw(P, C[m], C[m+1])) > 0) e = m;
      else if( sign(ccw(P, C[m], C[s])) > 0) s = m;
      else e = m;
    }
    else{ // down
      if( sign(ccw(P, C[m], C[m+1])) < 0) s = m;
      else if( sign(ccw(P, C[m], C[s])) < 0) s = m;
      else e = m;
    }
  }
  if( s && local(P, C[s-1], C[s], C[s+1]) ) return s;
  if( e != N && local(P, C[e-1], C[e], C[e+1]) ) return e;
  return -1;
}
```

6 Miscellaneous

6.1 Mathematics

- **Tutte Matrix**. For a simple undirected graph G, Let M be a matrix with entries $A_{i,j} = 0$ if $(i,j) \notin E$ and $A_{i,j} = -A_{j,i} = X$ if $(i,j) \in E$. X could be any random value. If the determinants are non-zero, then a perfect matching exists, while other direction might not hold for very small probability.

- **Cayley's Formula**. Given a degree sequence $d_1, d_2 \cdots, d_n$ for each labeled vertices, there exists $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees. Summing this for every possible degree sequence gives n^{n-2}.

- **Kirchhoff's Theorem**. For a multigraph G with no loops, define Laplacian matrix as $L = D - A$. D is a diagonal matrix with $D_{i,i} = deg(i)$, and A is an adjacency matrix. If you remove any row and column of L, the determinant gives a number of spanning trees.

- **Green's Theorem**. Let C is positive, smooth, simple curve. D is region bounded by C.
  $$\oint_C (Ldx + Mdy) = \iint_D \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right)$$
  To calculate area, $\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y} = 1$, common selection is $M = \frac{1}{2}x$, $L = -\frac{1}{2}y$.
  Line integral of circle parametrized by $(x,y) = (x_C + r_C\cos\theta,\ y_C + r_C\sin\theta)$, when $\theta = t\theta_i + (1-t)\theta_f$, is given as follows.: $\frac{1}{2}(r_C(x_C(\sin\theta_f - \sin\theta_i) - y_C(\cos\theta_f - \cos\theta_i)) + (\theta_f - \theta_i)r_C^2)$.
  Line integral of line parametrized by $(x,y) = t(x_1,y_1) + (1-t)(x_2,y_2)$ is given as follows.: $\frac{1}{2}(x_1y_2 - x_2y_1)$.

- **Burnside's lemma / Pólya enumeration theorem**. let G and H be groups of permutations of finite sets X and Y. Let $c_m(g)$ denote the number of cycles of length m in $g \in G$ when permuting X. The number of colorings of X into $|Y| = n$ colors with exactly r_i occurrences of the i-th color is the coefficient of $w_1^{r_1} \dots w_n^{r_n}$ in the following polynomial:
  $P(w_1, \dots, w_n) = \frac{1}{|H|} \sum_{h \in H} \frac{1}{|G|} \sum_{g \in G} \prod_{m \geq 1} (\sum_{h^m(b)=b} (w_b^m))^{c_m(g)}$

  When $H = \{I\}$ (No color permutation):
  $P(w_1, \dots, w_n) = \frac{1}{|G|} \sum_{g \in G} \prod_{m \geq 1} (w_1^m + \dots + w_n^m)^{c_m(g)}$

  Without the occurrence restriction:
  $P(1, \dots, 1) = \frac{1}{|G|} \sum_{g \in G} n^{c(g)}$
  where $c(g)$ could also be interpreted as the number of elements in X that are fixed up to g.

- **Pick's Theorem**. $A = i + \frac{b}{2} - 1$, where: P is a simple polygon whose vertices are grid points, A is area of P, i is # of grid points in the interior of P, and b is # of grid points on the boundary of P. If h is # of holes of P ($h + 1$ simple closed curves in total), $A = i + \frac{b}{2} + h - 1$.

```
// number of (x, y) : (0 <= x < n && 0 < y <= k/d x + b/d)
// argument should be positive
ll count_solve(ll n, ll k, ll b, ll d) {
  if (k == 0) {
    return (b / d) * n;
  }
  if (k >= d || b >= d) {
    return ((k / d) * (n - 1) + 2 * (b / d)) * n / 2 + count_solve(n, k % d, b
    % d, d);
  }
  return count_solve((k * n + b) / d, d, (k * n + b) % d, k);
}
```

- **Xudyh Sieve**. $F(n) = \sum_{d|n} f(d)$
  $S(n) = \sum_{i \leq n} f(i) = \sum_{i \leq n} F(i) - \sum_{d=2}^{n} S\left(\lfloor \frac{n}{d} \rfloor\right)$
  Preprocess $S(1)$ to $S(M)$ (Set $M = n^{\frac{2}{3}}$ for complexity)
  $S(n) = \sum f(i) = \sum_{i \leq n} \left[F(i) - \sum_{j|i, j \neq i} f(j)\right] = \sum F(i) - \sum_{i/j=d=2}^{n} \sum_{dj \leq n} f(j)$
  $S(n) = \sum if(i) = \sum_{i \leq n} i \left[F(i) - \sum_{j|i, j \neq i} f(j)\right] = \sum iF(i) - \sum_{i/j=d=2}^{n} \sum_{dj \leq n} djf(j)$
  $\sum_{d|n} \varphi(d) = n$ $\sum_{d|n} \mu(d) =$ if $(n > 1)$ then 0 else 1 $\sum_{d|n} (\mu(\frac{n}{d}) \sum_{e|d} f(e)) = f(n)$

- **Knuth's $O(n^2)$ Optimal BST**. minimize $D_{i,j} = Min_{i \leq k < j}(D_{i,k} + D_{k+1,j}) + C_{i,j}$. Quadrangle Inequality : $C_{a,c} + C_{b,d} \leq C_{a,d} + C_{b,c}$, $C_{b,c} \leq C_{a,d}$. Now monotonicity holds.

- **LP Duality**. max $c^T x$ sjt to $Ax \leq b$. Dual problem is min $b^T x$ sjt to $A^T x \geq c$. By strong duality, min max value coincides.

6.2 Fast LL Division / Modulo

```
inline void fasterLLDivMod(unsigned long long x, unsigned y, unsigned &out_d,
unsigned &out_m) {
  unsigned xh = (unsigned)(x >> 32), xl = (unsigned)x, d, m;
#ifdef __GNUC__
  asm(
    "divl %4; \n\t"
    : "=a" (d), "=d" (m)
    : "d" (xh), "a" (xl), "r" (y)
  );
#else
  __asm {
    mov edx, dword ptr[xh];
    mov eax, dword ptr[xl];
    div dword ptr[y];
    mov dword ptr[d], eax;
    mov dword ptr[m], edx;
  };
#endif
  out_d = d; out_m = m;
}
//x < 2^32 * MOD !
inline unsigned Mod(unsigned long long x){
  unsigned y = mod;
  unsigned dummy, r;
  fasterLLDivMod(x, y, dummy, r);
  return r;
}
```

6.3 Bit Twiddling Hack

```
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 Fast Integer IO

```
static char buf[1 << 19]; // size : any number geq than 1024
static int idx = 0;
static int bytes = 0;
static inline int _read() {
  if (!bytes || idx == bytes) {
    bytes = (int)fread(buf, sizeof(buf[0]), sizeof(buf), stdin);
    idx = 0;
  }
  return buf[idx++];
}
static inline int _readInt() {
  int x = 0, s = 1;
  int c = _read();
  while (c <= 32) c = _read();
  if (c == '-') s = -1, c = _read();
  while (c > 32) x = 10 * x + (c - '0'), c = _read();
  if (s < 0) x = -x;
  return x;
}
```

6.5 Nasty Stack Hacks

```
// 64bit ver.
int main2(){ return 0; }
int main(){
  size_t  sz = 1<<29;  // 512MB
  void* newstack = malloc(sz);
  void* sp_dest = newstack + sz - sizeof(void*);
  asm  __volatile__("movq %0, %%rax\n\t"
  "movq %%rsp , (%%rax)\n\t"
  "movq %0, %%rsp\n\t": : "r"(sp_dest): );
  main2();
  asm  __volatile__("pop %rsp\n\t");
  return  0;
}
```

6.6 C++ / Environment Overview

```
// vimrc : set nu sc ci si ai sw=4 ts=4 bs=2  mouse=a syntax on

// compile : g++ -o PROB PROB.cpp -std=c++11 -Wall -O2
// options : -fsanitize=address -Wfatal-errors

struct StupidGCCCantEvenCompileThisSimpleCode{
  pair<int, int> array[1000000];
}; // https://gcc.gnu.org/bugzilla/show_bug.cgi?id=68203

// how to use rand (in 2018)
mt19937 rng(0x14004);
int randint(int lb, int ub){ return uniform_int_distribution<int>(lb, ub)(rng); }

// comparator overload
auto cmp = [](seg a, seg b){ return a.func() < b.func(); };
set<seg, decltype(cmp)> s(cmp);
map<seg, int, decltype(cmp)> mp(cmp);
priority_queue<seg, vector<seg>, decltype(cmp)> pq(cmp); // max heap

// hash func overload
struct point{
    int x, y;
    bool operator==(const point &p)const{ return x == p.x && y == p.y; }
};
struct hasher {
    size_t operator()(const point &p)const{ return p.x * 2 + p.y * 3; }
};
unordered_map<point, int, hasher> hsh;
```