Team Note of Deobureo Minkyu Party

koosaga, hyea, alex9801

Compiled on December 17, 2017

Contents

럭스를 럭스답게 든든한 연습헬팟 더불어민규당

hyea: Prove by solving.

koosaga: Locality to the rescue!

alex9801: Isshoman Beenzino

1 Flows, Matching

1.1 Hopcroft-Karp Bipartite Matching

```
const int MAXN = 50005, MAXM = 50005;
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){
  for(auto &i : gph[x]){
    if(r[i] == -1 || (!vis[r[i]] && dis[r[i]] == dis[x] + 1 &&
    dfs(r[i]))){
      vis[r[i]] = 1; 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;
}
```

1.2 Dinic's Algorithm

```
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 match(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 Min Cost Max Flow

```
const int MAXN = 100;
struct mincostflow{
  struct edg{ int pos, cap, rev, 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, int c){
    gph[s].push_back({e, x, (int)gph[e].size(), c});
    gph[e].push_back({s, 0, (int)gph[s].size()-1, -c});
  }
  int dist[MAXN], pa[MAXN], pe[MAXN];
  bool inque[MAXN];
  bool spfa(int src, int sink){
    memset(dist, 0x3f, sizeof(dist));
    memset(inque, 0, sizeof(inque));
    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;
  }
  int match(int src, int sink){
    int ret = 0;
    while(spfa(src, sink)){
      int cap = 1e9;
      for(int pos = sink; pos != src; pos = pa[pos]){
        cap = min(cap, gph[pa[pos]][pe[pos]].cap);
      }
      ret += dist[sink] * cap;
      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;
```

1.4 Hell-Joseon style MCMF

```
const int MAXN = 100;
struct mincostflow{
  struct edg{ int pos, cap, rev, 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, int c){
    gph[s].push_back({e, x, (int)gph[e].size(), c});
    gph[e].push_back({s, 0, (int)gph[s].size()-1, -c});
  }
  int phi[MAXN], inque[MAXN], dist[MAXN];
  void prep(int src, int sink){
    memset(phi, 0x3f, sizeof(phi));
    memset(dist, 0x3f, sizeof(dist));
    queue<int> que;
    que.push(src);
    inque[src] = 1;
    while(!que.empty()){
      int x = que.front();
      que.pop();
      inque[x] = 0;
      for(auto &i : gph[x]){
        if(i.cap > 0 && phi[i.pos] > phi[x] + i.cost){
          phi[i.pos] = phi[x] + i.cost;
          if(!inque[i.pos]){
            inque[i.pos] = 1;
            que.push(i.pos);
          }
        }
      }
    }
    for(int i=0; i<MAXN; i++){
      for(auto &j : gph[i]){
        if(j.cap > 0) j.cost += phi[i] - phi[j.pos];
      }
    }
    priority_queue<pi, vector<pi>, greater<pi> > pq;
    pq.push(pi(0, src));
    dist[src] = 0;
    while(!pq.empty()){
      auto l = pq.top();
      pq.pop();
      if(dist[l.second] != l.first) continue;
      for(auto &i : gph[l.second]){
        if(i.cap > 0 && dist[i.pos] > l.first + i.cost){
```

```
        dist[i.pos] = l.first + i.cost;
        pq.push(pi(dist[i.pos], i.pos));
      }
    }
  }
}
bool vis[MAXN];
int ptr[MAXN];
int dfs(int pos, int sink, int flow){
  vis[pos] = 1;
  if(pos == sink) return flow;
  for(; ptr[pos] < gph[pos].size(); ptr[pos]++){
    auto &i = gph[pos][ptr[pos]];
    if(!vis[i.pos] && dist[i.pos] == i.cost + dist[pos] && i.cap >
    0){
      int ret = dfs(i.pos, sink, min(i.cap, flow));
      if(ret != 0){
        i.cap -= ret;
        gph[i.pos][i.rev].cap += ret;
        return ret;
      }
    }
  }
  return 0;
}
int match(int src, int sink, int sz){
  prep(src, sink);
  for(int i=0; i<sz; i++) dist[i] += phi[sink] - phi[src];
  int ret = 0;
  while(true){
    memset(ptr, 0, sizeof(ptr));
    memset(vis, 0, sizeof(vis));
    int tmp = 0;
    while((tmp = dfs(src, sink, 1e9))){
      ret += dist[sink] * tmp;
      memset(vis, 0, sizeof(vis));
    }
    tmp = 1e9;
    for(int i=0; i<sz; i++){
      if(!vis[i]) continue;
      for(auto &j : gph[i]){
        if(j.cap > 0 && !vis[j.pos]){
          tmp = min(tmp, (dist[i] + j.cost) - dist[j.pos]);
        }
      }
    }
    if(tmp > 1e9 - 200) break;
    for(int i=0; i<sz; i++){
      if(!vis[i]) dist[i] += tmp;
    }
  }
  return ret;
}
}mcmf;
```

1.5 Circulation Problem

```
struct circ{
  maxflow mf;
  lint lsum;
  void clear(){
    lsum = 0;
    mf.clear();
  }
  void add_edge(int s, int e, int l, int r){
    lsum += l;
    mf.add_edge(s + 2, e + 2, r - l);
    mf.add_edge(0, e + 2, l);
    mf.add_edge(s + 2, 1, l);
  }
  bool solve(int s, int e){
    mf.add_edge(e+2, s+2, 1e9); // to reduce as maxflow with lower
    bounds, in circulation problem skip this line
    return lsum == mf.match(0, 1);
    // to get maximum LR flow, run maxflow from s+2 to e+2 again
  }
}circ;
```

1.6 Min Cost Circulation (WIP)

Should be **added**.

1.7 2-Commodity Flows (WIP)

Should be **added**.

1.8 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;
}
```

1.9 Stable Marriage

```
vector<vector<int>> rev; // 0-based
vector<int> solve(int n, vector<vector<int>> &a, vector<vector<int>>
&b){
  rev.clear(); rev.resize(n, vector<int>(n, 0));
  vector<int> ret(n), pnt(n), mat(n, -1);
  queue<int> que;
  for(int i=0; i<n; i++){
    for(int j=0; j<n; j++) rev[i][b[i][j]] = j;
    que.push(i);
  }
  while(!que.empty()){
    int x = que.front(); que.pop();
    int y = a[x][pnt[x]++];
    if(mat[y] == -1) mat[y] = x, ret[x] = y;
    else{
      if(rev[y][mat[y]] > rev[y][x]){
        que.push(mat[y]);
        mat[y] = x, ret[x] = y;
      }
      else que.push(x);
    }
  }
  return ret; // optimal matching of A side (x)
}
```

1.10 Edmond's Blossom Algorithm for General Matching (WIP)

Should be **added**.

2 Graph

2.1 2-SAT

```
strongly_connected scc;
int n; // = number of clauses
void init(int _n){ scc.clear(); n = _n; }
int NOT(int x){ return x >= n ? (x - n) : (x + n); }
```

```
void add_edge(int x, int y){ // input ~x to denote NOT
  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(vector<bool> &res){
  res.resize(n);
  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;
}
```

2.2 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);
  }
}
```

2.3 Splay Tree + Link-Cut Tree

```
// Checklist 1. Is it link cut, or splay?
// Checklist 2. In link cut, is son always root?
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(!x->p) root = x; // IF YOU ARE SPLAY TREE
  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);
}
```

2.4 Offline Dynamic MST (WIP)

Should be **added**.

2.5 Dominator Tree (WIP)

Should be **added**.

2.6 Global Min-Cut

```
namespace stoer_wagner{
  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;
  }
};
```

2.7 K-shortest path (WIP)

Should be **added**.

2.8 Edmond's Directed MST (WIP)

Should be **added**.

2.9 Vizing Theorem for Edge Coloring (WIP)

Should be **added**.

3 Strings

3.1 Aho-Corasick Algorithm

```
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 Suffix Array

Should be **revised**.

```
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);
  }
}
```

3.3 Manacher's Algorithm

```
const int MAXN = 1000005;
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;
    }
  }
}
```

3.4 Suffix Automaton (WIP)

Should be **added**.

3.5 Palindromic Tree (WIP)

Should be **added**.

3.6 Circular LCS

```
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;
  s1 = '#' + 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;
}
```

4 Geometry

4.1 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 $M = \frac{1}{2}x, \quad L = -\frac{1}{2}y$.

Line integral of circle parametrized by $(x, y) = (x_C + r_C \cos\theta,\ y_C + r_C \sin\theta)$ 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.: $\frac{1}{2}(x_1 y_2 - x_2 y_1)$.

```
inline double arc_area(double x, double y, double r, double s,
double e){ //s and e are line integral theta value.
  return (r*(x*(sin(e)-sin(s))-y*(cos(e)-cos(s)))+(e-s)*r*r)*0.5;
}

inline double polygon_area(double x1, double y1, double x2, double
y2){ //Shoelace formula
  return (x1*y2-x2*y1)*0.5;
}
```

4.2 Smallest Enclosing Circle / Sphere

```
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);
  }
};
```

4.3 3D Convex Hull

```
// code credit : https://gist.github.com/msg555/4963794
struct vec3{
  ll x, y, z;
  vec3(): x(0), y(0), z(0) {}
  vec3(ll a, ll b, ll c): x(a), y(b), z(c) {}
  vec3 operator*(const vec3& v) const{ return vec3(y*v.z-z*v.y,
  z*v.x-x*v.z, x*v.y-y*v.x); }
  vec3 operator-(const vec3& v) const{ return vec3(x-v.x, y-v.y,
  z-v.z); }
  vec3 operator-() const{ return vec3(-x, -y, -z); }
  ll dot(const vec3 &v) const{ return x*v.x+y*v.y+z*v.z; }
};

struct twoset {
  int a, b;
  void insert(int x) { (a == -1 ? a : b) = x; }
  bool contains(int x) { return a == x || b == x; }
  void erase(int x) { (a == x ? a : b) = -1; }
  int size() { return (a != -1) + (b != -1); }
} E[MAXN][MAXN]; // i < j

struct face{
```

```
  vec3 norm;
  ll disc;
  int I[3];
};

face make_face(int i, int j, int k, int ii, vector<vec3> &A){ // p^T
* norm < disc
  E[i][j].insert(k); E[i][k].insert(j); E[j][k].insert(i);
  face f; f.I[0]=i, f.I[1]=j, f.I[2]=k;
  f.norm = (A[j]-A[i])*(A[k]-A[i]);
  f.disc = f.norm.dot(A[i]);
  if(f.norm.dot(A[ii])>f.disc){
    f.norm = -f.norm;
    f.disc = -f.disc;
  }
  return f;
}

vector<face> get_hull(vector<vec3> &A){
  int N = A.size();
  vector<face> faces; memset(E, -1, sizeof(E));
  faces.push_back(make_face(0,1,2,3,A));
  faces.push_back(make_face(0,1,3,2,A));
  faces.push_back(make_face(0,2,3,1,A));
  faces.push_back(make_face(1,2,3,0,A));
  for(int i=4; i<N; ++i){
    for(int j=0; j<faces.size(); ++j){
      face f = faces[j];
      if(f.norm.dot(A[i])>f.disc){
        E[f.I[0]][f.I[1]].erase(f.I[2]);
        E[f.I[0]][f.I[2]].erase(f.I[1]);
        E[f.I[1]][f.I[2]].erase(f.I[0]);
        faces[j--] = faces.back();
        faces.pop_back();
      }
    }
    int nf = faces.size();
    for(int j=0; j<nf; ++j){
      face f=faces[j];
      for(int a=0; a<3; ++a) for(int b=a+1; b<3; ++b){
        int c=3-a-b;
        if(E[f.I[a]][f.I[b]].size()==2) continue;
        faces.push_back(make_face(f.I[a], f.I[b], i, f.I[c], A));
      }
    }
  }
  return faces;
}
```

4.4 Dynamic Convex Hull Trick

```
// code credit : https://github.com/niklasb/contest-algos/
// blob/master/convex_hull/dynamic.cpp
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;
```

4.5 Half-plane Intersection

```
const double eps = 1e-8;
typedef pair<long double, long double> pi;
namespace hpi{
  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;
  }
};
```

4.6 Polygon tangent (WIP)

Should be **added**.

4.7 kd-tree

```
typedef pair<int, int> pi;
struct node{
  pi pnt;
  int spl, sx, ex, sy, ey;
}tree[270000];
```

```
pi a[100005];
int n, ok[270000];

lint sqr(int x){ return 1ll * x * x; }
bool cmp1(pi a, pi b){ return a < b; }
bool cmp2(pi a, pi b){ return pi(a.second, a.first) < pi(b.second,
b.first); }

// init(0, n-1, 1) : Initialize kd-tree
// set dap = INF, and call solve(1, P). dap = (closest point from P)

void init(int s, int e, int p){ // Initialize kd-tree
  int minx = 1e9, maxx = -1e9, miny = 1e9, maxy = -1e9;
  int m = (s+e)/2;
  for(int i=s; i<=e; i++){
    minx = min(minx, a[i].first);
    miny = min(miny, a[i].second);
    maxx = max(maxx, a[i].first);
    maxy = max(maxy, a[i].second);
  }
  tree[p].spl = (maxx - minx < maxy - miny);
  sort(a+s, a+e+1, [&](const pi &a, const pi &b){
    return tree[p].spl ? cmp2(a, b) : cmp1(a, b);
  });
  ok[p] = 1;
  tree[p] = {a[m], tree[p].spl, minx, maxx, miny, maxy};
  if(s <= m-1) init(s, m-1, 2*p);
  if(m+1 <= e) init(m+1, e, 2*p+1);
}

lint dap = 3e18;

void solve(int p, pi x){ // find closest point from point x (L^2)
  if(x != tree[p].pnt) dap = min(dap, sqr(x.first -
  tree[p].pnt.first) + sqr(x.second - tree[p].pnt.second));
  if(tree[p].spl){
    if(!cmp2(tree[p].pnt, x)){
      if(ok[2*p]) solve(2*p, x);
      if(ok[2*p+1] && sqr(tree[2*p+1].sy - x.second) < dap)
      solve(2*p+1, x);
    }
    else{
      if(ok[2*p+1]) solve(2*p+1, x);
      if(ok[2*p] && sqr(tree[2*p].ey - x.second) < dap) solve(2*p,
      x);
    }
  }
  else{
    if(!cmp1(tree[p].pnt, x)){
      if(ok[2*p]) solve(2*p, x);
      if(ok[2*p+1] && sqr(tree[2*p+1].sx - x.first) < dap)
      solve(2*p+1, x);
    }
    else{
      if(ok[2*p+1]) solve(2*p+1, x);
      if(ok[2*p] && sqr(tree[2*p].ex - x.first) < dap) solve(2*p,
      x);
    }
  }
}
```

5 Math

5.1 FFT / NTT

```
namespace fft{
  typedef complex<double> base;
  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]);
    }
    double ang = 2 * acos(-1) / n * (inv ? -1 : 1);
```

```
  for(int i=0; i<n/2; i++){
    roots[i] = base(cos(ang * i), sin(ang * i));
  }
  /* In NTT, let prr = primitive root. Then,
  int ang = ipow(prr, (mod - 1) / n);
  if(inv) ang = ipow(ang, mod - 2);
  for(int i=0; i<n/2; i++){
    roots[i] = (i ? (1ll * roots[i-1] * ang % mod) : 1);
  }
  Others are same. If there is /= n, do *= ipow(n, mod - 2).
  In XOR convolution, roots[*] = 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++){
        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;
}

vector<lint> multiply(vector<lint> &v, vector<lint> &w){
  vector<base> fv(v.begin(), v.end()), fw(w.begin(), w.end());
  int n = 2; while(n < v.size() + w.size()) 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<lint> ret(n);
  for(int i=0; i<n; i++) ret[i] = (lint)round(fv[i].real());
  return ret;
}
vector<lint> multiply(vector<lint> &v, vector<lint> &w, lint mod){
  int n = 2; while(n < v.size() + w.size()) 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<lint> ret(n);
  for(int i=0; i<n; i++){
    lint av = (lint)round(r1[i].real());
    lint bv = (lint)round(r1[i].imag()) +
    (lint)round(r2[i].real());
    lint cv = (lint)round(r2[i].imag());
    av %= mod, bv %= mod, cv %= mod;
    ret[i] = (av << 30) + (bv << 15) + cv;
    ret[i] %= mod;
    ret[i] += mod;
    ret[i] %= mod;
  }
  return ret;
 }
}
```

5.2 Hell-Joseon style FFT

```
#include <smmintrin.h>
#include <immintrin.h>
#pragma GCC target("avx2")
#pragma GCC target("fma")
```

```
__m256d mult(__m256d a, __m256d b){
    __m256d c = _mm256_movedup_pd(a);
    __m256d d = _mm256_shuffle_pd(a, a, 15);
    __m256d cb = _mm256_mul_pd(c, b);
    __m256d db = _mm256_mul_pd(d, b);
    __m256d e = _mm256_shuffle_pd(db, db, 5);
    __m256d r = _mm256_addsub_pd(cb, e);
    return r;
}
void fft(int n, __m128d a[], bool invert){
    for(int i=1, j=0; i<n; ++i){
        int bit = n>>1;
        for(;j>=bit;bit>>=1) j -= bit;
        j += bit;
        if(i<j) swap(a[i], a[j]);
    }
    for(int len=2; len<=n; len<<=1){
        double ang = 2*3.14159265358979/len*(invert?-1:1);
        __m256d wlen; wlen[0] = cos(ang), wlen[1] = sin(ang);
        for(int i=0; i<n; i += len){
            __m256d w; w[0] = 1; w[1] = 0;
            for(int j=0; j<len/2; ++j){
                w = _mm256_permute2f128_pd(w, w, 0);
                wlen = _mm256_insertf128_pd(wlen, a[i+j+len/2], 1);
                w = mult(w, wlen);
                __m128d vw = _mm256_extractf128_pd(w, 1);
                __m128d u = a[i+j];
                a[i+j] = _mm_add_pd(u, vw);
                a[i+j+len/2] = _mm_sub_pd(u, vw);
            }
        }
    }
    if(invert){
        __m128d inv; inv[0] = inv[1] = 1.0/n;
        for(int i=0; i<n; ++i) a[i] = _mm_mul_pd(a[i], inv);
    }
}
vector<int64_t> multiply(vector<int64_t>& v, vector<int64_t>& w){
    int n = 2; while(n < v.size()+w.size()) n<<=1;
    __m128d* fv = new __m128d[n];
    for(int i=0; i<n; ++i) fv[i][0] = fv[i][1] = 0;
    for(int i=0; i<v.size(); ++i) fv[i][0] = v[i];
    for(int i=0; i<w.size(); ++i) fv[i][1] = w[i];
    fft(n, fv, 0); //(a+bi) is stored in FFT
    for(int i=0; i<n; i += 2){
        __m256d a;
        a = _mm256_insertf128_pd(a, fv[i], 0);
        a = _mm256_insertf128_pd(a, fv[i+1], 1);
        a = mult(a, a);
        fv[i] = _mm256_extractf128_pd(a, 0);
        fv[i+1] = _mm256_extractf128_pd(a, 1);
    }
    fft(n, fv, 1);
    vector<int64_t> ret(n);
    for(int i=0; i<n; ++i) ret[i] = (int64_t)round(fv[i][1]/2);
    delete[] fv;
    return ret;
}
```

5.3 NTT Polynomial Division

```
vector<lint> get_inv(int n, const vector<lint> &p){
  vector<lint> q = {ipow(p[0], mod - 2)};
  for(int i=2; i<=n; i<<=1){
    vector<lint> res;
    vector<lint> fq(q.begin(), q.end()); fq.resize(2*i);
    vector<lint> fp(p.begin(), p.begin() + i); fp.resize(2*i);
    fft(fq, 0); fft(fp, 0);
    for(int j=0; j<2*i; j++){
      fp[j] *= fq[j] * fq[j] % mod;
      fp[j] %= mod;
    }
    fft(fp, 1);
    res.resize(i);
    for(int j=0; j<i; j++){
      res[j] = mod - fp[j];
      if(j < i/2) res[j] += 2 * q[j];
      res[j] %= mod;
    }
    q = res;
```

```
  }
  return q;
}
vector<lint> poly_divide(const vector<lint> &a, const vector<lint>
&b){
  assert(b.back() != 0); // please trim leading zero
  int n = a.size(), m = b.size();
  int k = 2; while(k < n-m+1) k <<= 1;
  vector<lint> rb(k), ra(k);
  for(int i=0; i<m && i<k; ++i) rb[i] = b[m-i-1];
  for(int i=0; i<n && i<k; ++i) ra[i] = a[n-i-1];
  vector<lint> rbi = get_inv(k, rb);
  vector<lint> res = multiply(rbi, ra);
  res.resize(n - m + 1);
  reverse(res.begin(), res.end());
  return res;
}
```

5.4 Simplex Algorithm

```
/* Ax <= b, max c^T x
 * Usage : Simplex(VVD A, VD b, VD c).solve(VD ans)
 * not feasible : -INF; unbounded : INF
 * accuracy ~ (size of ans) * EPS
 * EPS recommended 1e-9 on double, 1e-12 on long double
 * expected n ~ 100, 10ms. worst case is exponential */

using real_t = double;
using VD = vector<real_t>;
using VVD = vector<VD>;
const real_t EPS = 1e-9;

struct Simplex{
  int m, n;
  vector<int> B, N;
  VVD D;
  Simplex(const VVD& A, const VD& b, const VD &c)
    : m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, VD(n+2)){
      for(int i=0; i<m; ++i) for(int j=0; j<n; ++j) D[i][j] =
      A[i][j];
      for(int i=0; i<m; ++i) B[i] = n+i, D[i][n] = -1, D[i][n+1] =
      b[i];
      for(int j=0; j<n; ++j) N[j] = j, D[m][j] = -c[j];
      N[n] = -1; D[m+1][n] = 1;
  }
  void Pivot(int r, int s) {
    real_t inv = 1/D[r][s];
    for(int i=0; i<m+2; ++i){
      for(int j=0; j<n+2; ++j){
        if(i != r && j != s) D[i][j] -= D[r][j] * D[i][s] * inv;
      }
    }
    for(int i=0; i<m+2; ++i) if(i != r) D[i][s] *= -inv;
    for(int j=0; j<n+2; ++j) if(j != s) D[r][j] *= inv;
    D[r][s] = inv; swap(B[r], N[s]);
  }
  bool Phase(bool p) {
    int x = m + p;
    while(true) {
      int s = -1;
      for(int j=0; j<=n; ++j){
        if(!p && N[j] == -1) continue;
        if(s == -1 || D[x][j] < D[x][s]) s = j;
      }
      if(D[x][s] > -EPS) return true;
      int r = -1;
      for(int i=0; i<m; ++i){
        if(D[i][s] <= EPS) continue;
        if(r == -1 || D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s]) r =
        i;
      }
      if(r == -1) return false;
      Pivot(r, s);
    }
  }
  real_t solve(VD &x) {
    int r = 0;
    for(int i=1; i<m; ++i) if(D[i][n+1] < D[r][n+1]) r=i;
    if(D[r][n+1] < -EPS) {
      Pivot(r, n);
```

```
      if(!Phase(1) || D[m+1][n+1] < -EPS) return -1/0.0;
      for(int i=0; i<m; ++i) if(B[i] == -1) {
        int s = min_element(D[i].begin(), D[i].end() - 1) -
        D[i].begin();
        Pivot(i, s);
      }
    }
    if(!Phase(0)) return 1/0.0;
    x = VD(n);
    for(int i=0; i<m; ++i) if(B[i] < n) x[B[i]] = D[i][n+1];
    return D[m][n+1];
  }
};
```

5.5 Range Prime Counting

```
// credit :
https://github.com/stjepang/snippets/blob/master/count_primes.cpp
// Primes up to 10^12 can be counted in ~1 second.
const int MAXN = 1000005; // MAXN is the maximum value of sqrt(N) +
2
bool prime[MAXN];
int prec[MAXN];
vector<int> P;

void init() {
  prime[2] = true;
  for (int i = 3; i < MAXN; i += 2) prime[i] = true;
  for (int i = 3; i*i < MAXN; i += 2){
    if (prime[i]){
      for (int j = i*i; j < MAXN; j += i+i) prime[j] = false;
    }
  }
  for(int i=1; i<MAXN; i++){
    if (prime[i]) P.push_back(i);
    prec[i] = prec[i-1] + prime[i];
  }
}

lint rec(lint N, int K) {
  if (N <= 1 || K < 0) return 0;
  if (N <= P[K]) return N-1;
  if (N < MAXN && 1ll * P[K]*P[K] > N) return N-1 - prec[N] +
  prec[P[K]];
  const int LIM = 250;
  static int memo[LIM*LIM][LIM];
  bool ok = N < LIM*LIM;
  if (ok && memo[N][K]) return memo[N][K];
  lint ret = N/P[K] - rec(N/P[K], K-1) + rec(N, K-1);
  if (ok) memo[N][K] = ret;
  return ret;
}

lint count_primes(lint N) { //less than or equal to
  if (N < MAXN) return prec[N];
  int K = prec[(int)sqrt(N) + 1];
  return N-1 - rec(N, K) + prec[P[K]];
}
```

5.6 Discrete Square Root

```
// https://github.com/tzupengwang/PECaveros/
// blob/master/codebook/math/DiscreteSqrt.cpp
void calcH(int &t, int &h, const int p) {
  int tmp=p-1; for(t=0;(tmp&1)==0;tmp/=2) t++; h=tmp;
}
// solve equation x^2 mod p = a
bool solve(int a, int p, int &x, int &y) {
  if(p == 2) { x = y = 1; return true; }
  int p2 = p / 2, tmp = mypow(a, p2, p);
  if (tmp == p - 1) return false;
  if ((p + 1) % 4 == 0) {
    x=mypow(a,(p+1)/4,p); y=p-x; return true;
  } else {
    int t, h, b, pb; calcH(t, h, p);
    if (t >= 2) {
      do {b = rand() % (p - 2) + 2;
      } while (mypow(b, p / 2, p) != p - 1);
      pb = mypow(b, h, p);
    } int s = mypow(a, h / 2, p);
```

```
    for (int step = 2; step <= t; step++) {
      int ss = (((lint)(s * s) % p) * a) % p;
      for(int i=0;i<t-step;i++) ss=(lint)ss*ss%p;;
      if (ss + 1 == p) s = (s * pb) % p;
      pb = ((lint)pb * pb) % p;
    } x = ((lint)s * a) % p; y = p - x;
  } return true;
}
```

5.7 Miller-Rabin Test + Pollard Rho Factorization

```
namespace miller_rabin{
  lint mul(lint a, lint b, lint p){
    lint ret = 0;
    while(a){
      if(a&1) ret = (ret + b) % p;
      a >>= 1;
      b = (b << 1) % p;
    }
    return ret;
  }
  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;
  }
  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;
  }

}

namespace pollard_rho{
  lint f(lint x, lint n, lint c){
    return (c + miller_rabin::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(miller_rabin::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;
}
};
```

5.8 Highly Composite Numbers, Large Prime

< 10^k	number	divisors	2 3 5 71113171923293137
1	6	4	1 1
2	60	12	2 1 1
3	840	32	3 1 1 1
4	7560	64	3 3 1 1
5	83160	128	3 3 1 1 1
6	720720	240	4 2 1 1 1 1
7	8648640	448	6 3 1 1 1 1
8	73513440	768	5 3 1 1 1 1 1
9	735134400	1344	6 3 2 1 1 1 1
10	6983776800	2304	5 3 2 1 1 1 1 1
11	97772875200	4032	6 3 2 2 1 1 1 1
12	963761198400	6720	6 4 2 1 1 1 1 1 1
13	9316358251200	10752	6 3 2 1 1 1 1 1 1 1
14	97821761637600	17280	5 4 2 2 1 1 1 1 1 1
15	866421317361600	26880	6 4 2 1 1 1 1 1 1 1 1
16	8086598962041600	41472	8 3 2 2 1 1 1 1 1 1 1
17	74801040398884800	64512	6 3 2 2 1 1 1 1 1 1 1 1
18	897612484786617600	103680	8 4 2 2 1 1 1 1 1 1 1 1

< 10^k	prime	# of prime	< 10^k	prime
1	7	4	10	9999999967
2	97	25	11	99999999977
3	997	168	12	999999999989
4	9973	1229	13	9999999999971
5	99991	9592	14	99999999999973
6	999983	78498	15	999999999999989
7	9999991	664579	16	9999999999999937
8	99999989	5761455	17	99999999999999997
9	999999937	50847534	18	999999999999999989

NTT Prime:
$998244353 = 119 \times 2^{23} + 1$. Primitive root: 3.
$985661441 = 235 \times 2^{22} + 1$. Primitive root: 3.
$1012924417 = 483 \times 2^{21} + 1$. Primitive root: 5.

6 Miscellaneous

6.1 Popular Optimization Technique

- Convex Hull Trick (cf : 5.4 for dynamic slopes)
- Divide and Conquer Optimization
- Knuth's $O(n^2)$ Optimal BST : minimize $D_{i,j} = Min_{i \le k < j}(D_{i,k} + D_{k+1,j}) + C_{i,j}$. Quadrangle Inequality : $C_{a,c} + C_{b,d} \le C_{a,d} + C_{b,c}$, $C_{b,c} \le C_{a,d}$. Now monotonicity holds.
- Sqrt batch processing - Save queries in buffer, and update in every sqrt steps (cf : IOI 2011 Elephant. hyea calls it "ainta technique")
- Dynamic insertion in static set (Make $O(lgn)$ copy. Merge like binomial heap.)
- Offline insertion / deletion in insert-only set (Pair insertion-deletion operation, and regard it as range query)
- Mo's ~~algorithm~~ trick (on tree)
- Aliens trick : Partition n elements into k contiguous interval : Partition naively without k restriction. Penalize / Reward the partition by changing cost function. Mix this with binary search.

6.2 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));
}
```

6.3 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;
}
```

6.4 OSRank in g++

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

typedef
tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;

ordered_set X;
X.insert(1); X.insert(2); X.insert(4); X.insert(8); X.insert(16);

cout<<*X.find_by_order(1)<<endl; // 2
cout<<*X.find_by_order(2)<<endl; // 4
cout<<*X.find_by_order(4)<<endl; // 16
cout<<(end(X)==X.find_by_order(6))<<endl; // true

cout<<X.order_of_key(-5)<<endl;  // 0
cout<<X.order_of_key(1)<<endl;   // 0
cout<<X.order_of_key(3)<<endl;   // 2
cout<<X.order_of_key(4)<<endl;   // 2
cout<<X.order_of_key(400)<<endl; // 5
```

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
```

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

// how to use rand (in 2017)
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;
```