Team Note of 1_Hoeaeng_2_Hawawang

Seo Taesoo,Kim Junkyeom,Roh Seyun

Compiled on November 1, 2022

Contents

1 Graph

1.1 Euler Walk

Usage: Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.

Time Complexity: linear

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0)
{
  int n = sz(gr);
  vi D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler paths, not just cycles
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
    if (it == end){ ret.push_back(x); s.pop_back(); continue;
    }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1) return
  {};
  return {ret.rbegin(), ret.rend()};
}
```

1.2 Hungarian Method

Usage: D_{ij}가 주어질 때, $A_i + B_j + D_{ij} \geq 0$으로 만드는 최소 $sum(A_i, B_j)$를 구하는 데에도 사용

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

```
int in[505][505], mats[505], matt[505], Ls[505], Lt[505],
revs[505], revt[505], valt[505];
bool chks[505], chkt[505];

vector <int> Vu;
void vpush(int p, int N) {
  chks[p] = true;
  for (int i = 1; i <= N; i++) {
    if (!valt[i]) continue;
    if (valt[i] > Ls[p] + Lt[i] - in[p][i]) {
      valt[i] = Ls[p] + Lt[i] - in[p][i];
      revt[i] = p;
      if (!valt[i]) Vu.push_back(i);
    }
  }
}
int main() {
  int N, i, j, k;
  scanf("%d", &N);
  for (i = 1; i <= N; i++) {
    for (j = 1; j <= N; j++) {
      scanf("%d", &in[i][j]); // if minimum matching,
      in[i][j]=-in[i][j]
    }
  }

  for (i = 1; i <= N; i++) Lt[i] = -INF;
```

```
for (i = 1; i <= N; i++) for (j = 1; j <= N; j++) Lt[j] =
max(Lt[j], in[i][j]);
for (i = 1; i <= N; i++) {
  for (j = 1; j <= N; j++) chks[j] = chkt[j] = false;
  for (j = 1; j <= N; j++) valt[j] = INF;
  for (j = 1; j <= N; j++) revs[j] = revt[j] = 0;

  int p = 0;
  for (j = 1; j <= N; j++) if (!mats[j]) break;
  p = j;
  vpush(p, N);

  while (1) {
    if (!Vu.empty()) {
      int t = Vu.back();
      Vu.pop_back();
      chkt[t] = true;

      if (!matt[t]) {
        vector <int> Vu2;
        Vu2.push_back(t);
        while (1) {
          Vu2.push_back(revt[Vu2.back()]);
          if (Vu2.back() == p) break;
          Vu2.push_back(revs[Vu2.back()]);
        }
        reverse(all(Vu2));
        for (j = 0; j < Vu2.size(); j += 2) {
          int s = Vu2[j], t = Vu2[j + 1];
          mats[s] = t;
          matt[t] = s;
        }
        break;
      }
      else {
        int s = matt[t];
        revs[s] = t;
        vpush(s, N);
      }
    }
    else {
      int mn = INF;
      for (j = 1; j <= N; j++) if (!chkt[j]) mn = min(mn,
      valt[j]);
      for (j = 1; j <= N; j++) {
        if (chks[j]) Ls[j] -= mn;
        if (chkt[j]) Lt[j] += mn;
        else {
          valt[j] -= mn;
          if (valt[j] == 0) Vu.push_back(j);
        }}}}
  Vu.clear();
}

int ans = 0;
  for (i = 1; i <= N; i++) ans += Ls[i] + Lt[i];
  return !printf("%d\n", ans); // if minimum matching, print
  -ans
}
```

1.3 SCC / 2-SAT

Time Complexity: $\mathcal{O}(N+M)$

```
const int MAXN=100005;
vector<int> v[MAXN];
vector<vector<int>> SCC;
int num[MAXN], low[MAXN], sn[MAXN];
bool chk[MAXN];
stack<int> st;
int cnt, SN;
void dfs(int n){
  chk[n] = 1;
  st.push(n);
  num[n] = ++cnt;
  low[n] = cnt;
  for (int next : v[n]){
    if (num[next] == 0){
      dfs(next);
      low[n] = min(low[n], low[next]);
    }
    else if (chk[next])
      low[n] = min(low[n], num[next]);
  }
  if (num[n] == low[n]){
    vector<int> scc;
    while (!st.empty()){
      int x = st.top();
      st.pop();
      sn[x] = SN;
      chk[x] = 0;
      scc.push_back(x);
      if (n == x)
        break;
    }
    SCC.push_back(scc);
    SN++;
  }
}
// 2-SAT
vector<pair<int,int>> ord;
for(int i=1 ; i<=n ; i++)ord.push_back({-sn[i], i});
sort(ord.begin(), ord.end());
for(int i=1 ; i<=2*n ; i++){
  int now = ord[i].second;
  if (ans[now / 2] != -1)continue;
  ans[now / 2] = now & 1;
}
```

1.4 BCC

Usage: in main function, if(!num[i])dfs(i,-1), if(!vis[i])color(i,0)
Articulation point
u is root : equal or more than 2 children
u is not root : (u,v) is Tree edge and low[v] $\geq$ num[u]
Articulation bridge
(u,v)is Tree edge and low[v] ¿ num[u]
Time Complexity: $\mathcal{O}(N+M)$

```
void color(int x, int c){
   if(c > 0) bcc[x].push_back(c);
   vis[x] = 1;
   for(int w : graph[x]){
      if(vis[w]) continue;
      if(dfn[x] <= low[w]){
         bcc[x].push_back(++cpiv);
         color(w, cpiv);
      }
      else{
         color(w, c);
      }
   }
}
```

1.5 Dinic's Algorithm

Time Complexity: $\mathcal{O}(V^2E)$ but much faster

```
struct Edge {
  int to, r;
  Edge* ori;
  Edge* rev;
  Edge(int T, int R){
    to = T, r = R;
  }
};
vector<Edge *> v[503];
void addedge(int f, int t, int r)
{
  Edge* ori = new Edge(t, r);
  Edge* rev = new Edge(f, 0);
  ori->rev = rev;
  rev->rev = ori;
  v[f].push_back(ori);
  v[t].push_back(rev);
}
const int S = 501, T = 502;
int level[503],work[503];
bool bfs() {
  memset(level, -1, sizeof(level));
  level[S] = 0;
  queue<int> q;
  q.push(S);
  while (!q.empty()) {
    int x = q.front();
```

```
    q.pop();
    for (auto& nn : v[x]) {
      int next = nn->to;
      if (level[next] == -1 && nn->r > 0) {
        level[next] = level[x] + 1;
        q.push(next);
      }
    }
  }
  return level[T] != -1;
}
int dfs(int N, int des, int flow) {
  if (N == des) return flow;
  for (int &i = work[N]; i<v[N].size(); i++) {
    int next = v[N][i]->to;
    if (level[next] == level[N] + 1 && v[N][i]->r > 0) {
      int df = dfs(next, des, min(v[N][i]->r, flow));
      if (df > 0) {
        v[N][i]->r -= df;
        v[N][i]->rev->r += df;
        return df;
      }
    }
  }
  return 0;
}
int match(){
  int res = 0;
  while (bfs())
  {
    memset(work,0,sizeof(work));
    while (1)
    {
      int f = dfs(S, T, INF);
      if (f == 0)break;
      res += f;
    }
  }
  return res;
}
```

1.6 Hopcroft-Karp Bipartite Matching

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

```
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){
   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 Hell-Joseon MCMF

```
const int MAXN = 100;
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<pair<int,int>, vector<pair<int,int>>,
  greater<pair<int,int>> > pq;
  pq.push(pair<int,int>(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(pair<int,int>(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;
}
```

1.8 General Matching

Time Complexity: $\mathcal{O}(V^3)$ but fast in practice

```
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;
}
```

1.9 General Weighted Matching

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

```cpp
    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<ll,int> solve(){
  memset(match+1,0,sizeof(int)*n);
  n_x=n;
  int n_matches=0;
  ll 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 General Min Cut

Usage: addedge (u,v) cost z : g[u][v]+=z, g[v][u]+=z
Time Complexity: $\mathcal{O}(V^3)$

```
const int MAXN=505;
int g[MAXN][MAXN], dst[MAXN], chk[MAXN], del[MAXN];
int n, m;
int minCutPhase(int &s, int &t){
    memset(dst, 0, sizeof dst);
    memset(chk, 0, sizeof chk);
    int mincut = 0;
    for(int i=1; i<=n; i++){
        int k = -1, mx = -1;
        for(int j=1; j<=n; j++){
            if(del[j] || chk[j]) continue;
            if(dst[j] > mx) k = j, mx = dst[j];
        }
        if(k == -1) return mincut;
        s = t,t = k;
        mincut = mx, chk[k] = 1;
        for(int j=1; j<=n; j++){
            if(!del[j] && !chk[j]) dst[j] += g[k][j];
        }
    }
    return mincut;
}
int getMinCut(){
    int mincut = 1e9+7;
    for(int i=1; i<n; i++){
        int s, t;
        int now = minCutPhase(s, t);
        del[t] = 1;
        mincut = min(mincut, now);
        if(mincut == 0) return 0;
        for(int j=1; j<=n; j++){
            if(!del[j]) g[s][j] = (g[j][s] += g[j][t]);
        }
    }
    return mincut;
}
```

1.11 Dominator Tree

Usage: s in solve function is root
Time Complexity: $\mathcal{O}(N+M)$

```
const int MAXN=200002;
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();
  }
  RE[n+1].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;
}
```

1.12 Perfect Elimination Ordering

Usage: in Chordal Graph

Time Complexity: $\mathcal{O}(N+M)$

```
const int N_ = 101000;
namespace chordal{
   vector<int> v[N_];
   struct Set{
      list<int> L;
      int last;
   };
   list<Set> w;
   list<Set>::iterator Where[N_];
   list<int>::iterator Addr[N_];
   unordered_map<int, int>Edge[N_];

   void add_edge(int a,int b){
      v[a].push_back(b);
      v[b].push_back(a);
      Edge[a][b]=Edge[b][a]=1;
   }
   list<int> TP;
   vector<int> get_order(int n){ // 0-based
      vector<int> vis(n),Res(n+1),ord(n);
      for(int i = 0; i < n; i++) TP.push_back(i);
      w.push_back({ TP,0 });
      for (int i = 0; i < n; i++) Where[i] = w.begin();
      for (auto t=w.front().L.begin(); t !=
      w.front().L.end(); t++) {
         Addr[*t]=t;
      }
      int cnt = 0;
      while (!w.empty()) {
         auto cur = w.begin();
         if (cur->L.empty()) {
            w.erase(w.begin());
            continue;
         }
         int x = cur->L.front();
         Res[++cnt] = x, ord[x] = cnt, vis[x] = 1;
         cur->L.pop_front();
         for (auto &u : v[x]) {
            if (vis[u])continue;
            if (Where[u]->last != cnt) {
               auto it = Where[u];
               list<int>new_list;
               new_list.push_back(u);
               w.insert(it, { new_list, 0 });
               Where[u]->L.erase(Addr[u]);
               Where[u]->last = cnt; Where[u]--;
               Addr[u] = Where[u]->L.begin();
```

```
            }
            else {
                auto it = Where[u];
                Where[u]->L.erase(Addr[u]); Where[u]--;
                Where[u]->L.push_back(u);
                Addr[u] = Where[u]->L.end(); Addr[u]--;
            }
        }
    }
    for (int x = 0; x < n; x++) {
        int Max = -1;
        for (auto &u : v[x]) {
            if (ord[u] < ord[x])Max = max(Max, ord[u]);
        }
        if (Max == -1)continue;
        int pv = Res[Max];
        for (auto &u : v[x]) {
            if (u != pv && ord[u] < ord[x] &&
            !Edge[pv].count(u)) {
                return vector<int>(); //if no such order
            }
        }
    }
    reverse(Res.begin(),Res.end());
    Res.pop_back();
    return Res;
  }
}
```

1.13 Directed MST

Time Complexity: $\mathcal{O}(M \log M)$

```
#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(chrono::steady_clock::now().time_since_epoch().count());
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;
 }
};
```

1.14 Kirchhoff's Theorem

Usage: 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. You can use Berlekamp-Massey to calculate determinant faster.

Time Complexity: $\mathcal{O}(VE)$

1.15 Circulation

Usage: 새로운 소스와 새로운 싱크를 만든다. 기존 싱크에서 기존 소스로 capacity가 INF인 간선을 생성한다. a에서 b로의 간선이 하한이 l 상한이 r 일 때 a에서 새로운 싱크로 (l) , 새로운 소스에서 b로(l) , a에서b로 $(r - l)$ 간선을 만들어 준 뒤 새로운 소스에서 새로운 싱크로 maximum flow를 확인하여 l들의 합과 같은지 여부를 확인. 이 때의 maxflow를 구하고 싶다면 기존 소스에서 기존 싱크로 가는 플로우를 찾을 수 있을 때 까지 계속 찾으면 된다.

1.16 Hall's Theorem

어떤 이분 그래프 $G = (L \cup R, E)$가 주어졌다고 하자. 어떤 부분집합 $S \subseteq L$에 대해서 S에 인접한 정점들의 집합을 $N(S) \subseteq R$이라 할 때, L의 모든 정점이 참여하는 matching이 존재하는 필요충분조건은 모든 L의 부분집합 S가 $|S| \le |N(S)|$를 만족하는 것이다.

1.17 De Brujin Sequence

Usage: 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;
}
```

1.18 Maximum Clique

```
int n,cur;
ll G[50];
Pi p[50];
void add_edge(int x,int y){
  G[x]|=(1LL<<y);
  G[y]|=(1LL<<x);
}
void get_clique(int R=0,ll P=(1LL<<n)-1,ll x=0,ll now = 0){
  if((P|x)==0){
    cur=max(cur,R);
    return;
  }
  int u=__builtin_ctzll(P|x);
  ll c=P&~G[u];
  while(c){
    int v=__builtin_ctzll(c);
    get_clique(R+1,P&G[v],x&G[v], now | (1ll<<v));
    P^=1LL<<v;
    x|=1LL<<v;
    c^=1LL<<v;
  }
}
```

1.19 Find All Triangles

Usage: Find all cycles of length 3

Time Complexity: $\mathcal{O}(N + M\sqrt{M})$

```
vector< tuple<int,int,int> > find_all_triangles(
        int n,
        vector<pair<int,int>> edges) {
    // Remove duplicated edges
    sort(edges.begin(), edges.end());
    edges.erase(unique(edges.begin(), edges.end()),
    edges.end());

    // Compute degs
    vector<int> deg(n, 0);
    for (const auto& [u, v] : edges) {
        if (u == v) continue;
        ++deg[u], ++deg[v];
    }

    // Add edge (u, v) where u is 'lower' than v
    vector<vector<int>> adj(n);
    for (auto [u, v] : edges) {
        if (u == v) continue;
        if (deg[u] > deg[v] || (deg[u] == deg[v] && u > v))
        swap(u, v);
        adj[u].push_back(v);
    }

    // Find all triplets.
    // If it's too slow, remove vector res and compute answer
    directly
    vector<tuple<int,int,int>> res;
    vector<bool> good(n, false);
    for (int i = 0; i < n; i++) {
        for (auto j : adj[i]) good[j] = true;
        for (auto j : adj[i]) {
            for (auto k : adj[j]) {
                if (good[k]) {
                    res.emplace_back(i, j, k);
                }
            }
        }
        for (auto j : adj[i]) good[j] = false;
    }
    return res;
}
```

1.20 Maximum weighted independent set in bipartite graph

그래프 왼쪽에 S, 오른쪽에 T 두고 S에서 왼쪽에 정점 가중치만큼 유량을 가진 간선 연결, T에서 오른쪽에 정점 가중치만큼 유량을 가진 간선 연결, 왼쪽에서 오른쪽으로 가중치 무한인 간선 연결. (가중치의 총합) - (최대 유량) 구하면 됨

2 Data Structure

2.1 Splay Tree / Link-Cut Tree

Usage: To init splay tree/link-cut tree, use `splay_init(element num)` `lct_init(vertex num)`. Don't forget to define appropriate `inf` and modify function `update` and `push`. **Link Cut Tree is not verified yet.**

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

```
const int SPLAY_TREE = 1;
const int LINK_CUT_TREE = 2;
struct LinkCutNode{
    LinkCutNode *l, *r, *p, *pp;
    ll sz, v, mn, flip, dummy;
    LinkCutNode() : LinkCutNode(0) {}
    LinkCutNode(ll _v) : LinkCutNode(_v, nullptr) {}
    LinkCutNode(ll _v, LinkCutNode *_p){
        p = _p; pp = l = r = nullptr;
        sz = 1; v = mn = _v; flip = dummy = 0;
    }
    ~LinkCutNode(){ if(l) delete l; if(r) delete r; }
};
struct LinkCutTree{
    LinkCutNode *root, *nd[1010101]; int type;
    LinkCutTree() : root() { memset(nd, 0, sizeof nd); }
    ~LinkCutTree(){ if(root) delete root; }
    void splay_init(int n){
        type = SPLAY_TREE;
        if(root) delete root;
        auto *now = root = new LinkCutNode(-inf); //left
        dummy node
        for(int i=1; i<=n; i++){
            nd[i] =  now->r = new LinkCutNode(i, now);
            now = now->r;
        }
        now->r = new LinkCutNode(inf, now); //right dummy
        node
        root->dummy = now->r->dummy = 1;
        for(int i=n; i>=1; i--) update(nd[i]);
    }
    void lct_init(int n){
        type = LINK_CUT_TREE;
        for(int i=1; i<=n; i++) nd[i] = new LinkCutNode(i);
    }
    void update(LinkCutNode *x){
        x->mn = x->v; x->sz = 1;
        if(x->l) x->mn = min(x->mn, x->l->mn), x->sz +=
        x->l->sz;
        if(x->r) x->mn = min(x->mn, x->r->mn), x->sz +=
        x->r->sz;
    }
    void push(LinkCutNode *x){
        if(!x->flip) return;
        swap(x->l, x->r); x->flip = 0;
        if(x->l) x->l->flip ^= 1;
        if(x->r) x->r->flip ^= 1;
    }
    void rotate(LinkCutNode *x){
        if(!x->p) return;
        LinkCutNode *p = x->p, *y; push(p); push(x);
        if(x == p->l) p->l = y = x->r, x->r = p;
        else p->r = y = x->l, x->l = p;
        x->p = p->p; p->p = x;
        if(y) y->p = p;
        if(x->p && p == x->p->l) x->p->l = x;
        else if(x->p && p == x->p->r) x->p->r = x;
        else if(type == SPLAY_TREE) root = x;
        update(p); update(x);
        if(type == LINK_CUT_TREE && p->pp){
            x->pp = p->pp; p->pp = nullptr;
        }
    }
    LinkCutNode* splay(LinkCutNode *x, LinkCutNode *g =
    nullptr){
        while(x->p != g){
            LinkCutNode *p = x->p;
            if(p->p == g){ rotate(x); break; }
            auto pp = p->p;
            if((p->l == x) == (pp->l == p)) rotate(p),
            rotate(x);
            else rotate(x), rotate(x);
        }
        if(type == LINK_CUT_TREE || !g) return root = x;
        return root;
    }
    LinkCutNode* splay_kth(int k){ // 1-based, return kth
    element
        auto now = root; push(now);
        while(1){
            while(now->l && now->l->sz > k){
                now = now->l; push(now);
            }
            if(now->l) k -= now->l->sz;
            if(!k) break; k--;
            now = now->r; push(now);
        }
        return splay(now);
    }
    LinkCutNode* splay_gather(int s, int e){ // gather range
    [s, e]
        auto a = splay_kth(e+1), b = splay_kth(s-1);
        return splay(a, b)->r->l;
    }
    void splay_flip(int s, int e){ // flip range [s, e]
        LinkCutNode *x = splay_gather(s, e);
        x->flip = !x->flip;
    }
    LinkCutNode* splay_shift(int s, int e, int k){ //
    right_shift(k) range [s, e]
        LinkCutNode *range = splay_gather(s, e);
        if(k >= 0){
            k %= (e - s + 1); if(!k) return range;
            splay_flip(s, e); splay_flip(s, s+k-1);
            splay_flip(s+k, e);
        }else{
            k *= -1; k %= (e - s + 1); if(!k) return range;
            splay_flip(s, e); splay_flip(s, e-k);
            splay_flip(e-k+1, e);
        }
        return splay_gather(s, e);
    }
    // get node index(position)
    int splay_getidx(int k){ return splay(nd[k])->l->sz; }

    void access(LinkCutNode *x){
        splay(x); push(x);
        if(x->r){
            x->r->pp = x;
            x->r = x->r->p = nullptr;
        }
        update(x);
        while(x->pp){
            auto *nxt = x->pp;
            splay(nxt); push(nxt);
            if(nxt->r){
                nxt->r->pp = nxt;
                nxt->r = nxt->r->p = nullptr;
            }
            nxt->r = x; x->p = nxt;
            x->pp = nullptr;
            update(nxt); splay(x);
        }
    }
    LinkCutNode* lct_root(int _x){
        auto x = nd[_x]; access(x); push(x);
        while(x->l){ x = x->l; push(x); }
        access(x); return x;
    }
    LinkCutNode* lct_par(int _x){
        auto x = nd[_x]; access(x); push(x);
        if (!x->l) return nullptr;
        x = x->l; push(x);
        while(x->r){ x = x->r; push(x); }
        access(x); return x;
    }
    LinkCutNode* lct_lca(int _s, int _t){
        auto s = nd[_s], t = nd[_t]; access(s); access(t);
        splay(s);
        if(!s->pp) return s;
        return s->pp;
    }
    void lct_link(int _son, int _par){
        auto son = nd[_son], par = nd[_par];
        access(par); access(son);
        son->flip ^= 1; // remove if needed
        push(son);
```

```
        son->l = par; par->p = son;
        update(son);
    }
    void lct_cut(int _son){
        auto son = nd[_son]; access(son); push(son);
        if(son->l){ son->l = son->l->p = nullptr; }
        update(son);
    }
    void inorder(LinkCutNode *x){
        push(x);
        if(x->l) inorder(x->l);
        if(!x->dummy) print(x);
        if(x->r) inorder(x->r);
    }
};
```

2.2 Union Find and Rollback

```
struct RollbackUF {
  vi e; vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

2.3 Conex Hull Trick

Usage: CHT joa
Time Complexity: $\mathcal{O}(n)$

```
struct CHTLinear {
    struct Line {
        long long a, b;
        long long y(long long x) const { return a * x + b; }
    };
    vector<Line> stk;
    int qpt;
    CHTLinear() : qpt(0) { }
    // when you need maximum :  (previous l).a < (now l).a
    // when you need minimum :  (previous l).a > (now l).a
    void pushLine(const Line& l) {
        while (stk.size() > 1) {
            Line& l0 = stk[stk.size() - 1];
            Line& l1 = stk[stk.size() - 2];
            if ((l0.b - l.b) * (l0.a - l1.a) > (l1.b - l0.b)
            * (l.a - l0.a)) break;
            stk.pop_back();
        }
        stk.push_back(l);
    }
    // (previous x) <= (current x)
    // it calculates max/min at x
    long long query(long long x) {
        while (qpt + 1 < stk.size()) {
            Line& l0 = stk[qpt];
            Line& l1 = stk[qpt + 1];
            if (l1.a - l0.a > 0 && (l0.b - l1.b) > x * (l1.a
            - l0.a)) break;
            if (l1.a - l0.a < 0 && (l0.b - l1.b) < x * (l1.a
            - l0.a)) break;
            ++qpt;
        }
        return stk[qpt].y(x);
    }
};
```

2.4 Li-Chao Tree

Usage: Return maximum y coordinate for given x as `query`. You may update lines by `update`. You must call function `init` with leftmost and rightmost coordinate for x. **Be careful for setting inf and leftmost/rightmost coordinate to not make overflow.**.
Time Complexity: $\mathcal{O}(\log n)$ for query, $\mathcal{O}(n)$ memory complexity.

```
const ll inf = PRE_DEFINED_INF;
struct LiChaoTree {
  struct Line {
    ll a, b;
    ll f(ll x) { return a * x + b; }
    Line(ll a, ll b) : a(a), b(b) {}
    Line() : Line(0, -inf) {}
  };
  struct Node {
    Node() : l(-1), r(-1), v(0, -inf) {}
    int l, r; Line v;
  };
  vector<Node> nd;
  ll S, E;
  void init(ll _s, ll _e) { nd.emplace_back(); S = _s, E =
  _e; }
  void update(int node, ll s, ll e, Line v) {
    Line lo = nd[node].v, hi = v;
    if (lo.f(s) > hi.f(s)) swap(lo, hi);
    if (lo.f(e) <= hi.f(e)) { nd[node].v = hi; return; }
    ll m = s + e >> 1;
    if (lo.f(m) <= hi.f(m)) {
      nd[node].v = hi;
      if (nd[node].r == -1) nd[node].r = nd.size(),
      nd.emplace_back();
      update(nd[node].r, m + 1, e, lo);
    }
    else {
      nd[node].v = lo;
      if (nd[node].l == -1) nd[node].l = nd.size(),
      nd.emplace_back();
      update(nd[node].l, s, m, hi);
    }
  }
  void update(Line v) { update(0, S, E, v); }
  ll query(int node, ll s, ll e, ll x) {
    if (node == -1) return -inf;
    ll t = nd[node].v.f(x);
    ll m = s + e >> 1;
    if (x <= m) return max(t, query(nd[node].l, s, m, x));
    else return max(t, query(nd[node].r, m + 1, e, x));
  }
  ll query(ll x) { return query(0, S, E, x); }
};
```

2.5 Lazy Li-Chao Tree

Usage: Insert a segment, Add a segment($ax + b$), get point minimum, get range minimum. Notice that range minimum works only when you did not call $a \neq 0$ add function when $ax + b$.

Time Complexity: $\mathcal{O}(N)^2$ for insert and add. $\mathcal{O}(N)$ for both type of get.

```
#define ll long long
const ll inf = 4e18;
struct LiChao {
  struct Node {
    int l, r; ll a, b, mn, aa, bb;
    Node() { l = 0; r = 0; a = 0; b = inf; mn = inf; aa = 0;
    bb = 0; }
  };
  vector<Node> seg;
  ll _l, _r;
  LiChao(ll l, ll r) {
    seg.resize(2);
    _l = l; _r = r;
  }
  void propagate(int n, ll l, ll r) {
    if (seg[n].aa || seg[n].bb) {
      if (l != r) {
        if (seg[n].l == 0) seg[n].l = seg.size(),
        seg.push_back(Node());
        if (seg[n].r == 0) seg[n].r = seg.size(),
        seg.push_back(Node());
        seg[seg[n].l].aa += seg[n].aa, seg[seg[n].l].bb +=
        seg[n].bb;
```

```
      seg[seg[n].r].aa += seg[n].aa, seg[seg[n].r].bb +=
      seg[n].bb;
    }
    seg[n].mn += seg[n].bb;
    seg[n].a += seg[n].aa, seg[n].b += seg[n].bb;
    seg[n].aa = seg[n].bb = 0;
  }
}
void insert(ll L, ll R, ll a, ll b, int n, ll l, ll r) {
  if (r < L || R < l || L > R) return;
  if (seg[n].l == 0) seg[n].l = seg.size(),
  seg.push_back(Node());
  if (seg[n].r == 0) seg[n].r = seg.size(),
  seg.push_back(Node());
  propagate(n, l, r);
  seg[n].mn = min({seg[n].mn, a*max(l,L)+b, a*min(r,R)+b});
  ll m = l+r>>1;
  if (l < L || R < r) {
    if (L <= m) insert(L, R, a, b, seg[n].l, l, m);
    if (m+1 <= R) insert(L, R, a, b, seg[n].r, m+1, r);
    return;
  }
  ll &sa = seg[n].a, &sb = seg[n].b;
  if (a*l+b < sa*l+sb) swap(a, sa), swap(b, sb);
  if (a*r+b >= sa*r+sb) return;
  if (a*m+b < sa*m+sb) {
    swap(a, sa), swap(b, sb);
    insert(L, R, a, b, seg[n].l, l, m);
  }
  else insert(L, R, a, b, seg[n].r, m+1, r);
}
void add(ll L, ll R, ll a, ll b, int n, ll l, ll r) {
  if (r < L || R < l || L > R) return;
  if (seg[n].l == 0) seg[n].l = seg.size(),
  seg.push_back(Node());
  if (seg[n].r == 0) seg[n].r = seg.size(),
  seg.push_back(Node());
  propagate(n, l, r);
  ll m = l+r>>1;
  if (l < L || R < r) {
    insert(l, m, seg[n].a, seg[n].b, seg[n].l, l, m);
    insert(m+1, r, seg[n].a, seg[n].b, seg[n].r, m+1, r);
    seg[n].a = 0, seg[n].b = inf, seg[n].mn = inf;
    if (L <= m) add(L, R, a, b, seg[n].l, l, m);
    if (m+1 <= R) add(L, R, a, b, seg[n].r, m+1, r);
    seg[n].mn = min(seg[seg[n].l].mn, seg[seg[n].r].mn);
    return;
  }
  seg[n].aa += a, seg[n].bb += b;
  propagate(n, l, r);
}
ll get(ll x, int n, ll l, ll r) {
  if (n == 0) return inf;
  propagate(n, l, r);
  ll ret = seg[n].a*x + seg[n].b, m = l+r>>1;
  if (x <= m) return min(ret, get(x, seg[n].l, l, m));
  return min(ret, get(x, seg[n].r, m+1, r));
}
ll get(ll L, ll R, int n, ll l, ll r) {
  if (n == 0) return inf;
  if (r < L || R < l || L > R) return inf;
  propagate(n, l, r);
  if (L <= l && r <= R) return seg[n].mn;
  ll m = l+r>>1;
  return min({seg[n].a*max(l,L)+seg[n].b,
  seg[n].a*min(r,R)+seg[n].b, get(L, R, seg[n].l, l, m),
  get(L, R, seg[n].r, m+1, r)});
}
void insert(ll L, ll R, ll a, ll b) {
  insert(L, R, a, b, 1, _l, _r);
}
void add(ll L, ll R, ll a, ll b) {
  add(L, R, a, b, 1, _l, _r);
}
ll get(ll x) {
  return get(x, 1, _l, _r);
}
ll get(ll L, ll R) {
  return get(L, R, 1, _l, _r);
}
};
```

2.6 Persistent Segment Tree

Usage: You may call a constructer with number of nodes(0-indexed). You must call `init` function before use it(with unempty vector if you want to initialize). Default `update` runs with root with biggest root number for case that you update linearly. Be CAREFUL to modify `PST_MAX`, `size_v`, and `update` (especially when you use `kth` function.)

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

```
#define PST_MAX 101010
typedef ll size_v;
struct PSTNode {
  PSTNode* l, * r; size_v v;
  PSTNode() { l = r = nullptr; v = 0; }
};
struct PST {
  PSTNode* root[PST_MAX];
  int n, cnt;
  PST(int _n) : n(_n), cnt(0) { memset(root, 0, sizeof root);
  }
  void init(PSTNode* node, int s, int e, vector<size_v>& in)
  {
    if (s == e) { if (!in.empty()) node->v = in[s]; return; }
    int m = s + e >> 1;
    node->l = new PSTNode; node->r = new PSTNode;
    init(node->l, s, m, in); init(node->r, m + 1, e, in);
    node->v = node->l->v + node->r->v;
  }
  void init(vector<size_v>& in) { root[0] = new PSTNode;
  cnt++; init(root[0], 0, n - 1, in); }
  void init() { vector<size_v> tmp; init(tmp); }
  void update(PSTNode* prv, PSTNode* now, int s, int e, int
  x, size_v v) {
    if (s == e) { now->v = v; return; }
    // IF addition query:
    // DO if (s == e) { now->v = prv ? prv->v + v : v;
    return; }
    int m = s + e >> 1;
    if (x <= m) {
      now->l = new PSTNode; now->r = prv->r;
      update(prv->l, now->l, s, m, x, v);
    }
    else {
      now->r = new PSTNode; now->l = prv->l;
      update(prv->r, now->r, m + 1, e, x, v);
    }
    size_v t1 = now->l ? now->l->v : 0;
    size_v t2 = now->r ? now->r->v : 0;
    now->v = t1 + t2;
  }
  void update(int prv_idx, int x, size_v v) {
    root[cnt] = new PSTNode;
    update(root[prv_idx], root[cnt], 0, n - 1, x, v); cnt++;
  } void update(int x, size_v v) { update(cnt - 1, x, v); }
  size_v query(PSTNode* node, int s, int e, int l, int r) {
    if (r < s || e < l) return 0;
    if (l <= s && e <= r) return node->v;
    int m = s + e >> 1;
    return query(node->l, s, m, l, r) + query(node->r, m + 1,
    e, l, r);
  } size_v query(int root_idx, int l, int r) { return
  query(root[root_idx], 0, n - 1, l, r); }
  int kth(PSTNode* prv, PSTNode* now, int s, int e, int k) {
      //MUST be an addition query
    if (s == e) return s;
    int m = s + e >> 1;
    size_v diff = now->l->v - prv->l->v;
    if (k <= diff) return kth(prv->l, now->l, s, m, k);
    else return kth(prv->r, now->r, m + 1, e, k - diff);
  } int kth(int st, int en, int k) { return kth(root[st - 1],
  root[en], 0, n - 1, k); }
};
```

2.7 GomoryHuTree

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

```
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;
}
```

3 Geometry

3.1 Basic Implementations

```
inline int diff(double lhs, double rhs) {
  if (lhs - eps < rhs && rhs < lhs + eps) return 0;
  return (lhs < rhs) ? -1 : 1;
}
inline bool is_between(double check, double a, double b) {
  if (a < b) return (a - eps < check&& check < b + eps);
  else return (b - eps < check&& check < a + eps);
}
struct Point {
  double x, y;
  bool operator==(const Point& rhs) const { return diff(x,
  rhs.x) == 0 && diff(y, rhs.y) == 0; }
  // define <, <=, >, >=, +, -, * well. Good Luck Ryute!
};
struct Line {
  Point p, q;
  Point dir() const { return q - p; }
};
struct Circle {
  Point center; double r;
};

istream& operator >> (istream& in, Point& t) { in >> t.x >>
t.y; return in; }
ostream& operator << (ostream& out, Point t) { out << t.x <<
t.y; return out; }

inline double inner(const Point& a, const Point& b) { return
a.x * b.x + a.y * b.y; }
inline double outer(const Point& a, const Point& b) { return
a.x * b.y - a.y * b.x; }

inline int ccw_line(const Line& line, const Point& point) {
  return diff(outer(line.dir(), point - line.p), 0);
}
inline int ccw(const Point& a, const Point& b, const Point&
c) {
  return diff(outer(b - a, c - a), 0);
}
inline double dist(const Point& a, const Point& b) {
  return sqrt(inner(a - b, a - b));
}
inline double dist2(const Point& a, const Point& b) {
  return inner(a - b, a - b);
}

inline double dist(const Line& line, const Point& point, bool
segment = false) {
  double c1 = inner(point - line.p, line.dir());
  if (segment && diff(c1, 0) <= 0) return dist(line.p,
  point);
  double c2 = inner(line.dir(), line.dir());
  if (segment && diff(c2, c1) <= 0) return dist(line.q,
  point);
  return dist(line.q * (c1 / c2), point);
}

bool get_cross(const Line& a, const Line& b, Point& ret) {
  double mdet = outer(b.dir(), a.dir());
  if (diff(mdet, 0) == 0) return false;
  double t2 = outer(a.dir(), b.p - a.p) / mdet;
  ret = b.p + b.dir() * t2;
  return true;
}

bool get_segment_cross(const Line& a, const Line& b, Point&
ret) {
  double mdet = outer(b.dir(), a.dir());
  if (diff(mdet, 0) == 0) return false;
  double t1 = -outer(b.p - a.p, b.dir()) / mdet;
  double t2 = outer(a.dir(), b.p - a.p) / mdet;
  if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return
  false;
  ret = b.p + b.dir() * t2;
  return true;
}

Point ccw_perpendicular(Line& L) {
  Point t = L.dir();
  Point a = { -t.y,t.x }, b = { t.y,-t.x };
  Point np = a + L.p;
  if (ccw_line(L, np) >= 0) return a;
  else return b;
} // calculate per. vector which is ccw with line

Line moveLine_length(Line L, Point p, double d) {
  double k = dist(p, { 0,0 });
  p = p * (d / k);
  L.p = L.p + p, L.q = L.q + p;
  return L;
} // move L to dir p with length d

Point vector_reform(Point p, double val) {
  double k = dist(p, { 0,0 });
  return p * (val / k);
}

Point inner_center(const Point& a, const Point& b, const
Point& c) {
  double wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
  double w = wa + wb + wc;
  return Point{ (wa * a.x + wb * b.x + wc * c.x) / w, (wa *
  a.y + wb * b.y + wc * c.y) / w };
}

Point outer_center(const Point& a, const Point& b, const
Point& c) {
  Point d1 = b - a, d2 = c - a;
  double area = outer(d1, d2);
  double dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y
    + d1.y * d2.y * (d1.y - d2.y);
  double dy = d1.y * d1.y * d2.x - d2.y * d2.y * d1.x
    + d1.x * d2.x * (d1.x - d2.x);
  return Point{ a.x + dx / area / 2.0, a.y - dy / area / 2.0
  };
}

vector<Point> circle_line(const Circle& circle, const Line&
line) {
  vector<Point> result;
  double a = 2 * inner(line.dir(), line.dir());
  double b = 2 * (line.dir().x * (line.p.x - circle.center.x)
    + line.dir().y * (line.p.y - circle.center.y));
  double c = inner(line.p - circle.center, line.p -
  circle.center)
    - circle.r * circle.r;
  double det = b * b - 2 * a * c;
  int pred = diff(det, 0);
  if (pred == 0)
    result.push_back(line.p + line.dir() * (-b / a));
  else if (pred > 0) {
    det = sqrt(det);
    result.push_back(line.p + line.dir() * ((-b + det) / a));
```

```
    result.push_back(line.p + line.dir() * ((-b - det) / a));
  }
  return result;
}

vector<Point> circle_circle(const Circle& a, const Circle& b)
{
  vector<Point> result;
  int pred = diff(dist(a.center, b.center), a.r + b.r);
  if (pred > 0) return result;
  if (pred == 0) {
    result.push_back((a.center * b.r + b.center * a.r) * (1 /
    (a.r + b.r)));
    return result;
  }
  double aa = a.center.x * a.center.x + a.center.y *
  a.center.y - a.r * a.r;
  double bb = b.center.x * b.center.x + b.center.y *
  b.center.y - b.r * b.r;
  double tmp = (bb - aa) / 2.0;
  Point cdiff = b.center - a.center;
  if (diff(cdiff.x, 0) == 0) {
    if (diff(cdiff.y, 0) == 0)
      return result; // if (diff(a.r, b.r) == 0): same circle
    return circle_line(a, Line{ Point{ 0, tmp / cdiff.y },
    Point{ 1, tmp / cdiff.y } });
  }
  return circle_line(a,
    Line{ Point{ tmp / cdiff.x, 0 }, Point{ tmp / cdiff.x -
    cdiff.y, cdiff.x } });
}

Circle circle_from_3pts(const Point& a, const Point& b, const
Point& c) {
  Point ba = b - a, cb = c - b;
  Line p{ (a + b) * 0.5, (a + b) * 0.5 + Point{ ba.y, -ba.x }
  };
  Line q{ (b + c) * 0.5, (b + c) * 0.5 + Point{ cb.y, -cb.x }
  };
  Circle circle;
  if (!get_cross(p, q, circle.center)) circle.r = -1;
  else circle.r = dist(circle.center, a);
  return circle;
}

Circle circle_from_2pts_rad(const Point& a, const Point& b,
double r) {
  double det = r * r / dist2(a, b) - 0.25; Circle circle;
  if (det < 0) circle.r = -1;
  else {
    double h = sqrt(det);
    // center is to the left of a->b
    circle.center = (a + b) * 0.5 + Point{ a.y - b.y, b.x -
    a.x } *h;
    circle.r = r;
  }
  return circle;
}
```

3.2 Point In Convex Polygon Test

Usage: You need to reverse a vector if your convex hull is clockwise.
Time Complexity: $\mathcal{O}(\log n)$

```
bool pip_convex(const vector<Point>& v, Point pt) {
  int i = lower_bound(v.begin() + 1, v.end(), pt, [&](const
  Point& a, const Point& b) {
    int cw = ccw(v[0], a, b);
    if (cw) return cw > 0;
    return dist2(v[0], a) < dist2(v[0], b);
    }) - v.begin();
    if (i == v.size()) return 0;
    if (i == 1) return ccw(v[0], pt, v[1]) == 0 && v[0] <= pt
    && pt <= v[1];
    int t1 = ccw(v[0], pt, v[i]) * ccw(v[0], pt, v[i - 1]);
    int t2 = ccw(v[i], v[i - 1], v[0]) * ccw(v[i], v[i - 1],
    pt);
    if (t1 == -1 && t2 == -1) return 0;
    return ccw(v[0], pt, v[i - 1]) != 0;
}
```

3.3 Rotating Calipers

```
double rotating_calipers(vector<Point>& pt) {
  sort(pt.begin(), pt.end(), [](const Point& a, const Point&
  b) {
    return (a.x == b.x) ? a.y < b.y : a.x < b.x;
    });
  vector<Point> up, lo;
  for (const auto& p : pt) {
    while (up.size() >= 2 && ccw(*++up.rbegin(),
    *up.rbegin(), p) >= 0) up.pop_back();
    while (lo.size() >= 2 && ccw(*++lo.rbegin(),
    *lo.rbegin(), p) <= 0) lo.pop_back();
    up.emplace_back(p);
    lo.emplace_back(p);
  }
  if (up.size() <= 1) return 0; // only one point
  if (up.size() + lo.size() <= 4) return dist(up[0], up[1]);
  // points on line
  double ma = 0;
  for (int i = 0, j = (int)lo.size() - 1; i + 1 < up.size()
  || j > 0; ) {
    ma = max(ma, dist(up[i], lo[j])); // do something here!
    if (i + 1 == up.size()) --j;
    else if (j == 0) ++i;
    else if ((up[i + 1].y - up[i].y) * (lo[j].x - lo[j -
    1].x) > (up[i + 1].x - up[i].x) * (lo[j].y - lo[j -
    1].y))
      ++i;
    else --j;
  }
  return ma;
}
```

3.4 Half Plane Intersection

```
const long double eps = 1e-9, inf = 1e9;
struct Point {
    long double x, y;
    explicit Point(long double x = 0, long double y = 0) :
    x(x), y(y) {}
    // operator definition
};
struct Halfplane {
    Point p, pq;
    long double angle;
    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b -
    a) {angle =atan2l(pq.y, pq.x);    }
    bool out(const Point& r) { return cross(pq, r - p) <
    -eps; }
    bool operator < (const Halfplane& e) const { return angle
    < e.angle;}
    friend Point inter(const Halfplane& s, const Halfplane&
    t) {
        long double alpha = cross((t.p - s.p), t.pq) /
        cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};
vector<Point> hp_intersect(vector<Halfplane>& H) {
    Point box[4] = {  // Bounding box in CCW order
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };
    for(int i = 0; i<4; i++) { // Add bounding box
    half-planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {
        while (len > 1 && H[i].out(inter(dq[len-1],
        dq[len-2]))) {
            dq.pop_back();
            --len;
        }
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }
```

```
        if (len > 0 && fabsl(cross(H[i].pq, dq[len-1].pq)) <
        eps) {
            if (dot(H[i].pq, dq[len-1].pq) < 0.0)
                return vector<Point>();
            if (H[i].out(dq[len-1].p)) {
                dq.pop_back();
                --len;
            }
            else continue;
        }
        dq.push_back(H[i]);
        ++len;
    }
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2])))
    {
        dq.pop_back();
        --len;
    }
    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }
    if (len < 3) return vector<Point>();
    vector<Point> ret(len);
    for(int i = 0; i+1 < len; i++)
        ret[i] = inter(dq[i], dq[i+1]);
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

3.5 KD-tree

Usage: Query returns nearest point.
Time Complexity: $\mathcal{O}(n^2)$, $\mathcal{O}(n \log n)$ in average.

```
#define x first
#define y second
typedef pair<ll, ll> p;
inline ll dst(const p& a, const p& b) {
  ll dx = b.x - a.x, dy = b.y - a.y;
  return dx * dx + dy * dy;
}
struct KDNode {
  pll v; bool dir;
  ll sx, ex, sy, ey;
  KDNode() { sx = sy = inf; ex = ey = -inf; }
};
const auto xcmp = [](pll a, pll b) { return tie(a.x, a.y) <
tie(b.x, b.y); };
const auto ycmp = [](pll a, pll b) { return tie(a.y, a.x) <
tie(b.y, b.x); };
struct KDTree {
  // Segment Tree Size
  static const int S = 1 << 18;
  KDNode nd[S]; int chk[S];
  vector<pll> v;
  KDTree() { init(); }
  void init() { memset(chk, 0, sizeof chk); }
  void _build(int node, int s, int e) {
    chk[node] = 1;
    nd[node].sx = min_element(v.begin() + s, v.begin() + e +
    1, xcmp)->x;
    nd[node].ex = max_element(v.begin() + s, v.begin() + e +
    1, xcmp)->x;
    nd[node].sy = min_element(v.begin() + s, v.begin() + e +
    1, ycmp)->y;
    nd[node].ey = max_element(v.begin() + s, v.begin() + e +
    1, ycmp)->y;
    nd[node].dir = !nd[node / 2].dir;

    if (nd[node].dir) sort(v.begin() + s, v.begin() + e + 1,
    ycmp);
    else sort(v.begin() + s, v.begin() + e + 1, xcmp);

    int m = s + e >> 1; nd[node].v = v[m];
    if (s <= m - 1) _build(node << 1, s, m - 1);
    if (m + 1 <= e) _build(node << 1 | 1, m + 1, e);
  }
  void build(const vector<pll>& _v) {
    v = _v; sort(all(v));
    _build(1, 0, v.size() - 1);
  }
  ll query(pll t, int node = 1) {
    ll tmp, ret = inf;
    if (t != nd[node].v) ret = min(ret, dst(t, nd[node].v));
    bool x_chk = (!nd[node].dir && xcmp(t, nd[node].v));
    bool y_chk = (nd[node].dir && ycmp(t, nd[node].v));
    if (x_chk || y_chk) {
      if (chk[node << 1]) ret = min(ret, query(t, node <<
      1));
      if (chk[node << 1 | 1]) {
        if (nd[node].dir) tmp = nd[node << 1 | 1].sy - t.y;
        else tmp = nd[node << 1 | 1].sx - t.x;
        if (tmp * tmp < ret) ret = min(ret, query(t, node <<
        1 | 1));
      }
    }
    else {
      if (chk[node << 1 | 1]) ret = min(ret, query(t, node <<
      1 | 1));
      if (chk[node << 1]) {
        if (nd[node].dir) tmp = nd[node << 1].ey - t.y;
        else tmp = nd[node << 1].ex - t.x;
        if (tmp * tmp < ret) ret = min(ret, query(t, node <<
        1));
      }
    }
    return ret;
  }
};
```

3.6 Voronoi Diagram

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

```
typedef pair<int,int> pii;
typedef pair<double, double> pdd;
const double EPS = 1e-9;
int dcmp(double x){ return x < -EPS? -1 : x > EPS ? 1 : 0; }
double operator / (pdd a,    pdd b){ return a.first *
b.second - a.second * b.first; }
pdd    operator * (double b, pdd a){ return pdd(b * a.first,
b * a.second); }
pdd    operator + (pdd a,    pdd b){ return pdd(a.first +
b.first, a.second + b.second); }
pdd    operator - (pdd a,    pdd b){ return pdd(a.first -
b.first, a.second - b.second); }
double sq(double x){ return x*x; }
double size(pdd p){ return hypot(p.first, p.second); }
double sz2(pdd p){ return sq(p.first) + sq(p.second); }
pdd r90(pdd p){ return pdd(-p.second, p.first); }
pdd line_intersect(pdd a, pdd b, pdd u, pdd v){ return u +
(((a-u)/b) / (v/b))*v; }
pdd get_circumcenter(pdd p0, pdd p1, pdd p2){
  return line_intersect(0.5 * (p0+p1), r90(p0-p1), 0.5 *
  (p1+p2), r90(p1-p2));
}
double parabola_intersect(pdd left, pdd right, double
sweepline){
  if(dcmp(left.second - right.second) == 0) return
  (left.first + right.first) / 2.0;
  auto f2 = [](pdd left, pdd right, double sweepline){
    int sign = left.first < right.first ? 1 : -1;
    pdd m = 0.5 * (left+right);
    pdd v = line_intersect(m, r90(right-left), pdd(0,
    sweepline), pdd(1, 0));
    pdd w = line_intersect(m, r90(left-v), v, left-v);
    double l1 = size(v-w), l2 = sqrt(sq(sweepline-m.second) -
    sz2(m-w)), l3 = size(left-v);
    return v.first + (m.first - v.first) * l3 / (l1 + sign *
    l2);
  };
  if(fabs(left.second - right.second) < fabs(left.first -
  right.first) * EPS) return f2(left, right, sweepline);// */
  int sign = left.second < right.second ? -1 : 1;
  pdd v = line_intersect(left, right-left, pdd(0, sweepline),
  pdd(1, 0));
  double d1 = sz2(0.5 * (left+right) - v), d2 = sz2(0.5 *
  (left-right));
  return v.first + sign * sqrt(max(0.0, d1 - d2));
}
class Beachline{
  public:
    struct node{
      node(){}
```

```
    node(pdd point, int idx):point(point), idx(idx),
    end(0),
      link{0, 0}, par(0), prv(0), nxt(0) {}
    pdd point; int idx; int end;
    node *link[2], *par, *prv, *nxt;
  };
  node *root;
  double sweepline;
  Beachline() : sweepline(-1e20), root(NULL){ }
  inline int dir(node *x){ return x->par->link[0] != x; }
  void rotate(node *n){
    node *p = n->par;         int d = dir(n);
    p->link[d] = n->link[!d]; if(n->link[!d])
    n->link[!d]->par = p;
    n->par = p->par;          if(p->par)
    p->par->link[dir(p)] = n;
    n->link[!d] = p;          p->par = n;
  }
  void splay(node *x, node *f = NULL){
    while(x->par != f){
      if(x->par->par == f);
      else if(dir(x) == dir(x->par)) rotate(x->par);
      else rotate(x);
      rotate(x);
    }
    if(f == NULL) root = x;
  }
  void insert(node *n, node *p, int d){
    splay(p); node* c = p->link[d];
    n->link[d] = c; if(c) c->par = n;
    p->link[d] = n; n->par = p;

    node *prv = !d?p->prv:p, *nxt = !d?p:p->nxt;
    n->prv = prv;   if(prv) prv->nxt = n;
    n->nxt = nxt;   if(nxt) nxt->prv = n;
  }
  void erase(node* n){
    node *prv = n->prv, *nxt = n->nxt;
    if(!prv && !nxt){ if(n == root) root = NULL; return; }
    n->prv = NULL;   if(prv) prv->nxt = nxt;
    n->nxt = NULL;   if(nxt) nxt->prv = prv;
    splay(n);
    if(!nxt){
      root->par = NULL; n->link[0] = NULL;
      root = prv;
    }
    else{
      splay(nxt, n);     node* c = n->link[0];
      nxt->link[0] = c;  c->par = nxt;         n->link[0] =
      NULL;
      n->link[1] = NULL; nxt->par = NULL;
      root = nxt;
    }
  }
  bool get_event(node* cur, double &next_sweep){
    if(!cur->prv || !cur->nxt) return false;
    pdd u = r90(cur->point - cur->prv->point);
    pdd v = r90(cur->nxt->point - cur->point);
    if(dcmp(u/v) != 1) return false;
    pdd p = get_circumcenter(cur->point, cur->prv->point,
    cur->nxt->point);
    next_sweep = p.second + size(p - cur->point);
    return true;
  }
  node* find_beachline(double x){
    node* cur = root;
    while(cur){
      double left = cur->prv ?
      parabola_intersect(cur->prv->point, cur->point,
      sweepline) : -1e30;
      double right = cur->nxt ?
      parabola_intersect(cur->point, cur->nxt->point,
      sweepline) : 1e30;
      if(left <= x && x <= right){ splay(cur); return cur;
      }
      cur = cur->link[x > right];
    }
  }
}; using BeachNode = Beachline::node;
static BeachNode* arr;
static int sz;
static BeachNode* new_node(pdd point, int idx){
  arr[sz] = BeachNode(point, idx);
  return arr + (sz++);
}
struct event{
  event(double sweep, int idx):type(0), sweep(sweep),
  idx(idx){}
  event(double sweep, BeachNode* cur):type(1), sweep(sweep),
  prv(cur->prv->idx), cur(cur), nxt(cur->nxt->idx){}
  int type, idx, prv, nxt;
  BeachNode* cur;
  double sweep;
  bool operator>(const event &l)const{ return sweep >
  l.sweep; }
};
void VoronoiDiagram(vector<pdd> &input, vector<pdd> &vertex,
vector<pii> &edge, vector<pii> &area){
  Beachline beachline = Beachline();
  priority_queue<event, vector<event>, greater<event>>
  events;
  auto add_edge = [&](int u, int v, int a, int b, BeachNode*
  c1, BeachNode* c2){
    if(c1) c1->end = edge.size()*2;
    if(c2) c2->end = edge.size()*2 + 1;
    edge.emplace_back(u, v);
    area.emplace_back(a, b);
  };
  auto write_edge = [&](int idx, int v){ idx%2 == 0 ?
  edge[idx/2].first = v : edge[idx/2].second = v; };
  auto add_event = [&](BeachNode* cur){ double nxt;
  if(beachline.get_event(cur, nxt)) events.emplace(nxt, cur);
  };
  int n = input.size(), cnt = 0;
  arr = new BeachNode[n*4]; sz = 0;
  sort(input.begin(), input.end(), [](const pdd &l, const pdd
  &r){
      return l.second != r.second ? l.second < r.second :
      l.first < r.first;
      });
  BeachNode* tmp = beachline.root = new_node(input[0], 0),
  *t2;
  for(int i = 1; i < n; i++){
    if(dcmp(input[i].second - input[0].second) == 0){
      add_edge(-1, -1, i-1, i, 0, tmp);
      beachline.insert(t2 = new_node(input[i], i), tmp, 1);
      tmp = t2;
    }
    else events.emplace(input[i].second, i);
  }
  while(events.size()){
    event q = events.top(); events.pop();
    BeachNode *prv, *cur, *nxt, *site;
    int v = vertex.size(), idx = q.idx;
    beachline.sweepline = q.sweep;
    if(q.type == 0){
      pdd point = input[idx];
      cur = beachline.find_beachline(point.first);
      beachline.insert(site = new_node(point, idx), cur, 0);
      beachline.insert(prv = new_node(cur->point, cur->idx),
      site, 0);
      add_edge(-1, -1, cur->idx, idx, site, prv);
      add_event(prv); add_event(cur);
    }
    else{
      cur = q.cur, prv = cur->prv, nxt = cur->nxt;
      if(!prv || !nxt || prv->idx != q.prv || nxt->idx !=
      q.nxt) continue;
      vertex.push_back(get_circumcenter(prv->point,
      nxt->point, cur->point));
      write_edge(prv->end, v); write_edge(cur->end, v);
      add_edge(v, -1, prv->idx, nxt->idx, 0, prv);
      beachline.erase(cur);
      add_event(prv); add_event(nxt);
    }
  }
  delete arr;
}
```

3.7 3d Convex Hull

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

```
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;
}
```

3.8 Convex Tangent

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

```
int convex_tangent(vector<pii> &C, pii P, int up = 1){
  auto sign = [&](ll c){ return c > 0 ? up : c == 0 ? 0 :
  -up; };
  auto local = [&](pii P, pii a, pii b, pii c) {
    return sign((a - P) ^ (b - P)) <= 0 && sign((b - P) ^ (c
    - P)) >= 0;
  };

  assert(C.size() >= 2);
  int N = C.size()-1, s = 0, e = N, m;
  if( local(P, C[1], C[0], C[N-1]) ) return 0;
  //  for(int i = 1; i < N; i++) if( local(P, C[i-1], C[i],
  C[i+1])) return i;
  while(s+1 < e){
    m = (s+e) / 2;
    if( local(P, C[m-1], C[m], C[m+1]) ) return m;
    if( sign((C[s]-P) ^ (C[s+1]-P)) < 0 ){ // up
      if( sign((C[m]-P) ^ (C[m+1]-P)) > 0 ) e = m;
      else if( sign((C[m]-P) ^ (C[s]-P)) > 0 ) s = m;
      else e = m;
    }
    else{ // down
      if( sign((C[m]-P) ^ (C[m+1]-P)) < 0 ) s = m;
      else if( sign((C[m]-P) ^ (C[s]-P)) < 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;
}
```

3.9 Segment Intersections

Usage: Given N segments. Check and returns the indices if there are 2 segments intersect. NOTES: Must set Segment.id. Otherwise it will be impossible to debug

```
int cmp(int x, int y) {
    if (x == y) return 0;
    if (x < y) return -1;
    return 1;
}
struct Point {
    int x, y;

    Point() { x = y = 0; }
    Point(int x, int y) : x(x), y(y) {}

    Point operator - (const Point& a) const {
        return Point(x - a.x, y - a.y);
    }
    int operator % (const Point& a) const {
        return x*a.y - y*a.x;
    }
};
istream& operator >> (istream& cin, Point& p) {
    cin >> p.x >> p.y;
    return cin;
}

struct Segment {
    Point p, q;
    int id;

    double get_y(int x) const {
        if (p.x == q.x) return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (double) (q.x
        - p.x);
    }
};
istream& operator >> (istream& cin, Segment& s) {
    cin >> s.p >> s.q;
    return cin;
}

bool intersect1d(int l1, int r1, int l2, int r2) {
    if (l1 > r1) swap(l1, r1);
    if (l2 > r2) swap(l2, r2);

    return max(l1, l2) <= min(r1, r2);
}
int ccw(Point a, Point b, Point c) {
    return cmp((b - a) % (c - a), 0);
}

bool intersect(const Segment& a, const Segment& b) {
    return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x)
        && intersect1d(a.p.y, a.q.y, b.p.y, b.q.y)
        && ccw(a.p, a.q, b.p) * ccw(a.p, a.q, b.q) <= 0
        && ccw(b.p, b.q, a.p) * ccw(b.p, b.q, a.q) <= 0;
}
```

```cpp
bool operator < (const Segment& a, const Segment& b) {
    int x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - 1e-9;
}

struct Event {
    int x;
    int tp, id;

    Event() {}
    Event(int x, int tp, int id) : x(x), tp(tp), id(id) {}

    bool operator < (const Event& e) const {
        if (x != e.x) return x < e.x;
        return tp > e.tp;
    }
};

set&lt;Segment&gt; s;
vector< set&lt;Segment&gt; :: iterator> where;
set&lt;Segment&gt; :: iterator get_prev(set&lt;Segment&gt;::iterator it)
{
    return it == s.begin() ? s.end() : --it;
}

set&lt;Segment&gt; :: iterator get_next(set&lt;Segment&gt;::iterator it)
{
    return ++it;
}

pair<int,int> solve(const vector&lt;Segment&gt;& a) {
    int n = SZ(a);
    vector<Event> e;
    REP(i,n) {
        e.push_back(Event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(Event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(ALL(e));

    s.clear();
    where.resize(SZ(a));
    REP(i,SZ(e)) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set&lt;Segment&gt;::iterator next =
            s.lower_bound(a[id]), prev = get_prev(next);
            if (next != s.end() && intersect(*next, a[id])) {
                return make_pair(next->id, id);
            }
            if (prev != s.end() && intersect(*prev, a[id])) {
                return make_pair(prev->id, id);
            }
            where[id] = s.insert(next, a[id]);
        } else {
            set&lt;Segment&gt;::iterator next =
            get_next(where[id]), prev = get_prev(where[id]);
            if (next != s.end() && prev != s.end() &&
            intersect(*next, *prev)) {
                return make_pair(prev->id, next->id);
            }
            s.erase(where[id]);
        }
    }
    return make_pair(-1, -1);
}
```

3.10 Bulldozer

```cpp
struct Event //고압선
{
    int flag; /// 0: 수직벡터, 인덱스가 인접한 지 판단 /// 1:
    평행벡터, 양옆 인덱스 점까지 거리를 구함
    int q, w;
    pp v;
    bool operator<(const Event &r)
    {
        if (idx[q] == idx[r.q])
            return idx[w] < idx[r.w];
        return idx[q] < idx[r.q];
    }
};
vector<Event> e;
int main()
{
    int n; cin >> n; int i, j;
    for (i = 1; i <= n; i++)cin >> a[i].x >> a[i].y;
    sort(a + 1, a + n + 1);
    for (i = 1; i <= n; i++)
    {
        for (j = i + 1; j <= n; j++)
        {
            e.push_back({1, i, j, {a[j].x - a[i].x, a[j].y -
            a[i].y}});
            pp v = {a[i].y - a[j].y, a[j].x - a[i].x};
            if (a[i].y - a[j].y < 0 || a[j].y == a[i].y &&
            a[j].x - a[i].x < 0)
                v = {a[j].y - a[i].y, a[i].x - a[j].x};
            e.push_back({0, i, j, v});
        }
    }
    sort(e.begin(), e.end(), [&](Event e1, Event e2){
         if(ccw(e1.v,e2.v)>0)return true;
         return false; });
    for (i = 1; i <= n; i++)
    {
        b[i] = i;
        idx[i] = i;
    }
    int s = e.size();
    for (i = 0; i < s; i++)
    {
        j = i;
        vector<Event> ee;
        ee.push_back(e[i]);
        if (idx[e[i].q] > idx[e[i].w])
            swap(e[i].q, e[i].w);
        while (j + 1 < s && ccw(e[i].v, e[j + 1].v) == 0)
        {
            if (idx[e[j + 1].q] > idx[e[j + 1].w])
                swap(e[j + 1].q, e[j + 1].w);
            ee.push_back(e[j + 1]);
            j++;
        }
        sort(ee.begin(), ee.end());
        for (auto k : ee)
        {
            if (k.flag)
            {
                swap(b[idx[k.q]], b[idx[k.w]]);
                swap(idx[k.q], idx[k.w]);
            }
        }
        for (auto k : ee)
        {
            if (k.flag)
            {
                int lef = min(idx[k.q], idx[k.w]) - 1;
                int rig = max(idx[k.q], idx[k.w]) + 1;
                if (lef >= 1)
                    ans = max(ans, dist(a[k.q], a[k.w],
                    a[b[lef]]) / 2);
                if (rig <= n)
                    ans = max(ans, dist(a[k.q], a[k.w],
                    a[b[rig]]) / 2);
            }
            else if (abs(idx[k.q] - idx[k.w]) == 1)
                ans = max(ans, dis(a[k.q], a[k.w]) / 2);
        }
        i = j;
    }
    cout.precision(15);
    cout << ans;
    return 0;
}
```

4 String

4.1 KMP

```cpp
int kmp(const string &T, const string &P) {
    if (P.empty()) return 0;
    vector<int> pi(P.size(), 0);
    for (int i = 1, k = 0; i < P.size(); ++i) {
```

```
        while (k && P[k] != P[i]) k = pi[k - 1];
        if (P[k] == P[i]) ++k;
        pi[i] = k;
    }
    for (int i = 0, k = 0; i < T.size(); ++i) {
        while (k && P[k] != T[i]) k = pi[k - 1];
        if (P[k] == T[i]) ++k;
        if (k == P.size()) return i - k + 1; //더 찾으려면
        k=pi[k]
    }
    return -1;
}
```

4.2 Aho-Corasick

```
const int MAXN = 100005, MAXC = 26;
struct aho_corasick{
  int trie[MAXN][MAXC], piv; // trie
  int fail[MAXN]; // failure link
  int term[MAXN]; // output check
  void init(vector<string> &v){
    memset(trie, 0, sizeof(trie));
    memset(fail, 0, sizeof(fail));
    memset(term, 0, sizeof(term));
    for(auto &i: v)for(auto &j: i)j=j-'a';// lowercase
    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){
      for(auto &i: s)i=i-'a';
    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;
  }
}aho_corasick;
```

4.3 Suffix Array and LCP

```
// calculates suffix array.O(n*logn)
vector<int> suffix_array(const string& in) {
    int n = (int)in.size(), c = 0;
    vector<int> temp(n), pos2bckt(n), bckt(n), bpos(n),
    out(n);
    for (int i = 0; i < n; i++) out[i] = i;
    sort(out.begin(), out.end(), [&](int a, int b) { return
    in[a] < in[b]; });
    for (int i = 0; i < n; i++) {
        bckt[i] = c;
        if (i + 1 == n || in[out[i]] != in[out[i + 1]]) c++;
    }
    for (int h = 1; h < n && c < n; h <<= 1) {
        for (int i = 0; i < n; i++) pos2bckt[out[i]] =
        bckt[i];
        for (int i = n - 1; i >= 0; i--) bpos[bckt[i]] = i;
        for (int i = 0; i < n; i++)
            if (out[i] >= n - h) temp[bpos[bckt[i]]++] =
            out[i];
        for (int i = 0; i < n; i++)
            if (out[i] >= h) temp[bpos[pos2bckt[out[i] -
            h]]++] = out[i] - h;
        c = 0;
        for (int i = 0; i + 1 < n; i++) {
            int a = (bckt[i] != bckt[i + 1]) || (temp[i] >= n
            - h)
                || (pos2bckt[temp[i + 1] + h] !=
                pos2bckt[temp[i] + h]);
            bckt[i] = c;
            c += a;
        }
        bckt[n - 1] = c++;
        temp.swap(out);
    }
    return out;
}
// calculates lcp array. it needs SA & original sequence.
O(n)
vector<int> lcp_(const string& in, const vector<int>& sa) {
    int n = (int)in.size();
    if (n == 0) return vector<int>();
    vector<int> rank(n), height(n - 1);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    for (int i = 0, h = 0; i < n; i++) {
        if (rank[i] == 0) continue;
        int j = sa[rank[i] - 1];
        while (i + h < n && j + h < n && in[i + h] == in[j +
        h]) h++;
        height[rank[i] - 1] = h;
        if (h > 0) h--;
    }
    return height;
}
```

4.4 Manacher

Usage: Returns all palindromic largest span length. Dummy characters DO NOT inserted in start and end of the sequence. You may reference index of dummy characters to check even-length palindrome.
Time Complexity: $\mathcal{O}(n)$

```
const char dummy = '*';
vector<int> manacher(const string& s_str) {
    int r = -1, p = -1;
    string str;
    for (int i = 0; i < s_str.length() * 2 - 1; i++)
        str.push_back(i % 2 ? dummy : s_str[i / 2]);
    vector<int> plen(str.length());
    for (int i = 0; i < str.length(); ++i) {
        if (i <= r)
            plen[i] = min((2 * p - i >= 0) ? plen[2 * p - i]
            : 0, r - i);
        else
            plen[i] = 0;
        while (i - plen[i] - 1 >= 0 && i + plen[i] + 1 <
        str.length()
            && str[i - plen[i] - 1] == str[i + plen[i] + 1])
            {
            plen[i] += 1;
        }
        if (i + plen[i] > r) {
            r = i + plen[i];
            p = i;
        }
    }
    return plen;
}
```

4.5 Z

Usage: Calculates maximum k that $S[0..k] = S[i..i+k]$ for each i.
Time Complexity: $\mathcal{O}(n)$

```
vector<int> zf(string& s) {
    int n = s.size();
    vector<int> z(n);
    for (int i=1, l=0, r=0; i<n; ++i) { // [l, r)
        if (i < r) z[i] = min(r-i, z[i-l]);
        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) ++z[i];
```

```
        if (i+z[i] > r) l = i, r = i+z[i];
    }
    return z;// z[i]=longest common prefix of [i,n-1] and s
}
```

4.6 Eertree

```
struct node{
  int len,link;
  int ne[26]={0};
}tree[200002];
int cnt=2,last=2;
char s[200002];
void make_node(int c,int pos){
  int cur=last;
  while(1){
    if(pos-tree[cur].len>=1 &&
    c==s[pos-tree[cur].len-1]-'a')break;
    cur=tree[cur].link;
  }
  if(tree[cur].ne[c]){
    last=tree[cur].ne[c];
    return;
  }
  int next=last=tree[cur].ne[c]=++cnt;
  tree[next].len=tree[cur].len+2;
  if(tree[next].len==1){
    tree[next].link=2;
    return;
  }
  while(cur>1){
    cur=tree[cur].link;
    if(pos-tree[cur].len>=1 &&
    c==s[pos-tree[cur].len-1]-'a'){
      tree[next].link=tree[cur].ne[c];
      break;
    }
  }
}
void init(){
  tree[1].len=-1,tree[1].link=1;
  tree[2].len=0,tree[2].link=1;
  scanf("%s",s);
  int n = strlen(s);
  for(int i=0;i<n;i++)make_node(s[i]-'a', i);
}
```

4.7 Rope

Usage: Insert,Delete,Concat,Split,Index,Report
Time Complexity: $\mathcal{O}(n)$, Report is $\mathcal{O}(length + \log n)$

```
#include <ext/rope>
using namespace __gnu_cxx;
    string s; cin>>s;
    crope rp;
    rp.append(s.c_str());

    rp=rp.substr(x,y-x+1)+rp.substr(0,x)+rp.substr(y+1,n-y-1);
    // move [x,y] to front  (0-indexed)

    rp=rp.substr(0,x)+rp.substr(y+1,n-y-1)+rp.substr(x,y-x+1);
    // move [x,y] to back
    cout<<rp.at(x)<<"\n"; // get s[x]
```

4.8 String Tokenizer

```
vector<string> split(const string &s, char dm){
  // Returns a vector of strings tokenized by dm.
  stringstream ss(s);
  string item; vector<string> tokens;
  while(getline(ss,item,dm)) tokens.push_back(item);
  return tokens;
}
```

5 Math

5.1 Triangles

변 길이 a, b, c; $p = \frac{a+b+c}{2}$
넓이 $A = \sqrt{p(p-a)(p-b)(p-c)}$
외접원 반지름 $R = \frac{abc}{4A}$
내접원 반지름 $r = \frac{A}{p}$
중선 길이 $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$
각 이등분선 길이 $s_a = \sqrt{bc(1 - (\frac{a}{b+c})^2)}$
사인 법칙 $\frac{\sin A}{a} = \frac{1}{2R}$
코사인 법칙 $a^2 = b^2 + c^2 - 2bc\cos A$
탄젠트 법칙 $\frac{a+b}{a-b} = \frac{\tan(A+B)/2}{\tan(A-B)/2}$
중심 좌표 $(\frac{\alpha x_a + \beta x_b + \gamma x_c}{\alpha+\beta+\gamma}, \frac{\alpha y_a + \beta y_b + \gamma y_c}{\alpha+\beta+\gamma})$ where

이름	α	β	γ	
외심	$a^2\mathcal{A}$	$b^2\mathcal{B}$	$c^2\mathcal{C}$	$\mathcal{A} = b^2 + c^2 - a^2$
내심	a	b	c	$\mathcal{B} = a^2 + c^2 - b^2$
무게중심	1	1	1	$\mathcal{C} = a^2 + b^2 - c^2$
수심	$\mathcal{BC}$	$\mathcal{AC}$	$\mathcal{AB}$	
방심(A)	$-a$	b	c	

5.2 Series And Calculus

$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$	$(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$
$(\tan x)' = 1 + \tan^2 x$	$(\arctan x)' = \frac{1}{1+x^2}$
$\int \tan ax = -\frac{\ln\lvert\cos ax\rvert}{a}$	$\int x \sin ax = (\sin ax - ax\cos ax)/a^2$

$$\oint_C (Ldx + Mdy) = \iint_D (\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y})dxdy$$

where C is positively oriented, piecewise smooth, simple, closed; D is the region inside C; L and M have continuous partial derivatives in D.

Newton's $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

5.3 Theorems

Kirchhoff's theorem: The number of spanning trees equals any cofactor of its Laplacian matrix.

Dilworth's theorem: In a finite partially ordered set, maximum antichain equals minimum chain cover.

Euler's theorem 1: For coprime a and n, $a^{\phi(n)} \equiv 1 \pmod n$.

Euler's theorem 2: Generally, $a^n \equiv a^{n-\phi(n)} \pmod n$.

Euler's theorem 3: For $m \geq \log_2 n$, $a^m \equiv a^{m\%\phi(n)+\phi(n)} \pmod n$.

Konig's theorem: In a bipartite graph, maximum matching equals minimum vertex cover.

Hall's theorem: A bipartite graph with partition (A, B) has a perfect matching iff for $X \subseteq A$, $|X| \leq |N_G(X)|$.

Pick's theorem: $A = i + \frac{b}{2} - 1$. A: 다각형 넓이, i: 변 위의 점 개수, b: 변 내부 점 개수

5.4 Combinatorics

Counts the # of functions $f : N \to K$, $|N| = n$, $|K| = k$. The elements in N and K can be distinguishable or indistinguishable, while f can be injective (one-to-one) of surjective (onto).

N	K	none	injective	surjective
dist	dist	k^n	$\frac{k!}{(k-n)!}$	$k!S(n,k)$
indist	dist	$\binom{n+k-1}{n}$	$\binom{k}{n}$	$\binom{n-1}{n-k}$
dist	indist	$\sum_{t=0}^{k} S(n,t)$	$[n \leq k]$	$S(n,k)$
indist	indist	$\sum_{t=1}^{k} p(n,t)$	$[n \leq k]$	$p(n,k)$

Here, $S(n,k)$ is the Stirling number of the second kind, and $p(n,k)$ is the partition number.

Derangement: $D(n) = (n-1)(D(n-1) + D(n-2))$
Sgn Stirling 1: $S_1(n,k) = (n-1)S_1(n-1,k) + S_1(n-1,k-1)$
Unsgn Stirling 1: $C_1(n,k) = (n-1)c_1(n-1,k) + C_1(n-1,k-1)$
Stirling 2: $S_2(n,k) = kS_2(n-1,k) + S_2(n-1,k-1)$
Stirling 2: $S_2(n,k) = \frac{1}{k!}\sum_{j=0} k(-1)^{k-j}\binom{k}{j}j^n$
Partition: $p(n,k) = p(n-1,k-1) + p(n-k,k)$
Partition: $p(n) = \sum(-1)^k p(n - k(3k-1)/2)$
Bell: $B(n) = \sum_{k=1}^{n} \binom{n-1}{k-1} B(n-k)$
Catalan: $C_n = \frac{1}{n+1}\binom{2n}{n}$
Catalan: $C_n = \binom{2n}{n} - \binom{2n}{n+1}$
Catalan: $C_n = \frac{(2n)!}{(n+1)!n!}$
Catalan: $C_n = \sum C_i C_{n-i}$

5.5 Composite and Prime

$n \le 1,000,000$ 약수 최대 240개 (720,720)
$n \le 1,000,000,000$ 최대 1,344개 (735,134,400)
up to 10,000: 소수 1,229개 (9,973)
up to 100,000: 소수 9,592개 (99,991)
up to 1,000,000: 소수 78,498개 (999,983)
up to 1,000,000,000: 소수 50,847,534개 (999,999,937)
10,007; 10,009; 10,111; 31,567; 70,001; 1,000,003; 1,000,033
99,999,989; 999,999,937; 1,000,000,007; 9,999,999,967
$998244353 = 119 \times 2^{23} + 1$, primitive 3
$1012924417 = 483 \times 2^{21} + 1$, primitive 5

5.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by $a = k(m^2 - n^2)$, $b = k(2mn)$, $c = k(m^2 + n^2)$ where $m > n > 0, k > 0, gcd(m, n) = 1$, and either m or n is even.

5.7 FFT / NTT

Time Complexity: $\mathcal{O}(nlogn)$

```
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);
  }
  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.
}

vector<ll> multiply(vector<ll> &v, vector<ll> &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<ll> ret(n);
  for(int i=0; i<n; i++) ret[i] = (ll)round(fv[i].real());
  return ret;
}
vector<ll> multiply(vector<ll> &v, vector<ll> &w, ll 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<ll> ret(n);
  for(int i=0; i<n; i++){
    ll av = (ll)round(r1[i].real());
    ll bv = (ll)round(r1[i].imag()) +
    (ll)round(r2[i].real());
    ll cv = (ll)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;
}
```

5.8 Berlekamp-Massey Algorithm

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

```
vector<int> berlekamp_massey(vector<int> x){
  vector<int> ls, cur;
  int lf, ld;
  for(int i=0; i<x.size(); i++){
    ll 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;
    }
    ll k = -(x[i] - t) * POW(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, ll 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;
  }
  ll 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, ll 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);
}

//Extra
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;
}
```

5.9 Miller-Rabin Test + Pollard Rho Factorization

```
namespace miller_rabin{
    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;
  }
  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;
  }
};
```

5.10 Exgcd / Modulo inverse

```
//find x, y : ax+by=gcd(a,b)
pair<ll,ll> exgcd(ll a, ll b){
  if (!b) return{ 1, 0 };
  auto [x,y] = exgcd(b, a%b);
  return{ y, x - (a / b)*y };
}
//find x in [0,m) s.t. ax ≡ gcd(a, m) (mod m)
ll modinv(ll a, ll m) {
  return (exgcd(a, m).first % m + m) % m;
}
```

5.11 Xudyh's sieve

Usage: moe : squared prime factor : 0, odd number of prime factor : -1, even number of prime factor : 1
$(f * g)(n) = \sum_{d|n} f(d)g(\frac{n}{d})$

```
ll moe[1000001], sum[1000001];
ll inv;
ll f(ll x){  return moe[x]; }
ll gs(ll x){ return x; }
ll fgs(ll x){ return 1; }
// if you take f(x) = mobius(x), g(x) = 1, then h(x) = sum f
* g(x)  = 1
// totient : f=phi, g=1, f*g=n && change init
void init(){
  moe[1] = 1;
  for(int i=1; i<=1000000; i++){
    for(int j=2*i; j<=1000000; j+=i){
      moe[j] -= moe[i];
    }
  }
  inv = gs(1);
  for(int i=1; i<=1000000; i++){
    sum[i] = sum[i-1] + f(i);
  }
}
unordered_map<ll, ll> M;
ll query(ll x){
  if(x <= 1000000) return sum[x];
  if(M.find(x) != M.end()) return M[x];
  ll ans = fgs(x);
  for(ll i=2; i<=x; ){
    ll cur = x / (x / i);
    ans -= (gs(cur) - gs(i - 1)) * query(x / i);
    i = cur + 1;
  }
  ans /= inv;
  return M[x] = ans;
}
```

5.12 Chinese Remainder Theorem

```
ll minv(ll a, ll b)
{
  if(a==0 && b==1) return 0;
  if(a==1) return 1;
  return b - minv(b%a, a) * b / a;
}

// x == A.first (mod A.second)
// x == B.first (mod B.second)
// returns solution as X == ans.first (mod ans.second)
// if no solution, returns (-1, -1)
// always a good idea to keep 0 <= ?.first < ?.second (for ?
: A, B, ans)
pair<ll, ll> solve(pair<ll, ll> A, pair<ll, ll> B)
{
  if(A.second == -1 || B.second == -1) return make_pair(-1,
  -1);
  if(A.second == 1) return B;
  if(B.second == 1) return A;
  ll g = gcd(A.second, B.second); // gcd
  ll l = A.second * (B.second / g); // lcm
  if((B.first-A.first)%g!=0) return make_pair(-1, -1); // no
  solution case

  ll a = A.second / g;
  ll b = B.second / g;
  ll mul = (B.first-A.first) / g;
  mul = (mul * minv(a%b, b)) % b; // this is now t
  ll ret = (mul * A.second + A.first); // n_1 t + a_1
  ret %= l; ret = (ret + l) % l; // take modulos
  return make_pair(ret, l);
}
```

5.13 Sum of Floor

```
//Σ i=0 to n-1, floor((a*i+b)/m)
ll sum_of_floor(ll n, ll m, ll a, ll b) {
    ll ans=0;
    if(a>=m) {
        ans+=(n-1)*n*(a/m)/2;
        a%=m;
    }
    if(b>=m) {
        ans+=n*(b/m);
        b%=m;
    }
    ll y_max=(a*n+b)/m, x_max=(y_max*m-b);
    if(y_max==0) return ans;
    ans+=(n-(x_max+a-1)/a)*y_max;
    ans+=sum_of_floor(y_max,a,m,(a-x_max%a)%a);
    return ans;
}
```

5.14 Power Tower

```
// calculate V[0]^(V[1]^(V[2]^...)) mod MOD,
O(sqrt(MOD)logMOD)
int power_tower(int MOD, vector<int> V) {
  int N=V.size(); vector<int> M(1,MOD);
  auto phi=[](int m) {
    int ret=1;
    for(int i=2;1LL*i*i<=m;i++) if(m%i==0) {
      m/=i; ret*=i-1;
      while(m%i==0) { m/=i; ret*=i; }
    }
    if(m>1) ret*=m-1;
    return ret;
  };
  auto fast_pow=[](int a, int b, int MOD) {
    int ret=1;
    for(;b;b>>=1) {
      if(b&1) ret=1LL*ret*a%MOD;
      a=1LL*a*a%MOD;
    }
    return ret;
  };
  function<int (int)> solve=[&](int c) {
    if(c+1==N || M[c]==1) return V[c];
    if(c+2==N) return fast_pow(V[c],V[c+1],M[c]);
    return fast_pow(V[c],2*M[c+1]+solve(c+1),M[c]);
  };
  while(M.back()>1) M.push_back(phi(M.back()));
  for(int i=0;i<N;i++) if(V[i]==1) {
    V.resize(i);
    break;
  }
  if(V.empty()) V.push_back(1);
  while(V.size()>1 && max(V.back(),V[V.size()-2])<9) {
    V[V.size()-2]=fast_pow(V[V.size()-2],V.back(),1000000000);
    V.pop_back();
  }
  N=V.size();
  return solve(0)%M[0];
}
```

5.15 Simplex

```
/*
n := number of variables
m := number of constraints
a[1~m][1~n] := constraints
b[1~m] := constraints value (b[i] can be negative)
c[1~n] := maximum coefficient
v := results
sol[i] := 등호조건, i번째 변수의 값
ex) Maximize p = 6x + 14y + 13z
    Constraints: 0.5x + 2y + z ≤ 24
                 x + 2y + 4z ≤ 60
    n = 2, m = 3, a = [[0.5, 2, 1], [1, 2, 4]],
    b = [24, 60], c = [6, 14, 13]
*/
namespace simplex {
  using T = long double;
  const int N = 410, M = 30010;
  const T eps = 1e-7;
  int n, m;
  int Left[M], Down[N];
  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;
  }
}
```

Primal LP
Maximize $c^T x$
Subject to $Ax \le b, x \ge 0$
LP Dual
Minimize $b^T y$
Subject to $A^T y \ge c, y \ge 0$

5.16 Poly Interpolation

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

```
// Given n points (x[i], y[i]), computes an n-1-degree
polynomial $p$ that passes through them: p(x) = a[0]*x^0 +
... + a[n-1]*x^{n-1}.
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd 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]);
    }
  }
  double 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;
}
```

5.17 Generating function

중복조합 (m종류 중복 포함 n개 뽑는 경우의 수) $= nCr(m+n-1, n) : 1/(1-x)^m$ (n=0 to inf)
이항계수 $= nCr(m, n) : (1+x)^m$ (n=0 to m)
각 식의 n차 항을 계산하면 됨

5.18 Pick's Theorem

볼록 다각형의 넓이를 S, 경계에 있는 격자점의 개수를 b라고 할 때 다각형 내부의 격자점의 개수는 $S - \frac{b}{2} + 1$개

5.19 Burnside Lemma

Burnside Lemma : r을 orbit의 갯수라고 할 때 $(r \cdot |G| = \sum_{g \in G} |X_g|)$ 이다.
X_g : g연산을 했을 때 자기 자신으로 돌아오는 원소의 집합

5.20 Integration

Usage: Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4, although in practice you will want to verify that the result is stable to desired precision when epsilon changes.
Time Complexity: $\mathcal{O}(h)$

```
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
  double h = (b - a) / 2 / n, v = f(a) + f(b);
  rep(i,1,n*2)
    v += f(a + i*h) * (i&1 ? 4 : 2);
  return v * h / 3;
}
```

5.21 Adaptive Integration

```
/* Usage:
  double sphereVolume = quad(-1, 1, [](double x) {
  return quad(-1, 1, [\&](double y) {
  return quad(-1, 1, [\&](double z) {
  return x*x + y*y + z*z < 1; });});});
*/

typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

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

6 Misc

6.1 Horn SAT

```
/*
n=5 (number of variance)
{}, 0     // x1
{0,2}, 1  // x1 & x3 => x2
{0}, 1,    // x1 => x2
```

```
{0,1,2}, -1  // x1 & x2 & x3 => 0
{0,2,3}, -1  // x1 & x3 & x4 => 0
{2,1}, 3  // x3 & x2 => x4
{3,4}, -1  // x4 & x5 => 0
output : {1,1,0,0,0}
*/

vector<int> HornSATsolver(int n, const vector<vector<int>>
&condition,const vector<int> &result){
  int N=condition.size();
  vector<int> solution(n), to_visit, margin(N);
  vector<vector<int>> adj(n);
  for(int i=0; i<N; i++){
    margin[i]=condition[i].size();
    if(condition[i].empty()) to_visit.push_back(i);
    for(int x: condition[i])adj[x].push_back(i);
  }
  while(!to_visit.empty()){
    int i=to_visit.back();
    to_visit.pop_back();
    int h=result[i];
    if(h<0)return vector<int>(); //no solution
    if(solution[h])continue;
    solution[h]=1;
    for(int x:adj[h]){
      if(--margin[x]==0)to_visit.push_back(x);
    }
  }
  return solution;
}
```

6.2 Simple DP optimizations

```
// Knuth Opt.
int dp[5005][5005];
int opt[5005][5005];
for(k=2 ; k<=n ; k++){
    for(j=0 ; j+k<=n ; j++){
        dp[k][j]=INF;
        for(l=opt[k-1][j];l<=opt[k-1][j+1] ; l++){
            if(dp[k][j]>dp[l-j][j]+dp[k-l+j][l]){
                dp[k][j]=dp[l-j][j]+dp[k-l+j][l];
                opt[k][j]=l;
            }
        }
        dp[k][j]+=s[j+k]-s[j];
    }
}
cout<<dp[n][0]<<"\n";

// DnC Opt.
void go(int i, int l, int r, int pl, int pr){
  if (l > r)return;
  int m = (l + r) / 2;
  d[i][m] = INF;
  for (int k = pl; k <= pr && k<m; k++)
    if (d[i][m] > d[i - 1][k] + cost(k, m)){
      d[i][m] = d[i - 1][k] + cost(k, m);
      p[i][m] = k;
    }
  go(i, l, m - 1, pl, p[i][m]);
  go(i, m + 1, r, p[i][m], pr);
}
for (int i = 1; i <= m; i++)
  d[1][i] = cost(1, i);
for (int i = 2; i <= n; i++)
  go(i, 1, m, 1, m);
printf("%lld", d[n][m]);
```

Convex Hull Trick
Recurrence : $DP[i] = \min(DP[j] + B[j] \times A[i])$
Complexity : $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$

Divide and Conquer Opt
Recurrence : $DP[i][j] = \min(DP[i-1][k] + C[k][j])$
Condition: Optimal Solution is monotone / Monge Array
Complexity : $\mathcal{O}(kn \log n)$

Knuth Opt
Recurrence : $DP[i][j] = \min(DP[i][k] + DP[i+1][j] + C[i][j])$
Condition: Monge Array AND $C[a][d] \geq C[b][c]$ for $a \leq b \leq c \leq d$
Complexity : $\mathcal{O}(n^2)$

6.3 Aliens+Monotone Queue Optimization

Usage: $DP[i][j] = Min_{k<j}(DP[i-1][k] + C[k+1][j])$ and C is Monge Array

```
ll s[50005],dp[50005],cnt[50005],ans;
int dq[50005],n,m;
int cross(int i,int j){
    int lef=j+1,rig=n,k=-1;
    while(rig>=lef){
        int mid=(lef+rig)/2;
        if(dp[i]+(s[mid]-s[i])*(mid-i) <
        dp[j]+(s[mid]-s[j])*(mid-j)){
            k=mid;
            lef=mid+1;
        }
        else rig=mid-1;
    }
    return k;
}
pair<ll,int> f(ll c){
    int l=1,r=0; dq[++r]=0;
    for(int i=1 ; i<=n ; i++){
        while(l<r && cross(dq[l],dq[l+1])<i)l++;
        int x=dq[l];
        dp[i]=dp[x]+(s[i]-s[x])*(i-x)+c;
        cnt[i]=cnt[x]+1;
        while(r>l &&
        cross(dq[r-1],dq[r])>=cross(dq[r],i))r--;
        dq[++r]=i;
    }
    return {dp[n],cnt[n]};
}
int main(){
    cin>>n>>m;
    for(int i=1 ; i<=n ; i++)s[i],s[i]+=s[i-1];
    ll lef=0,rig=1e14;
    while(rig>=lef){
        ll mid=(lef+rig)/2;
        auto x=f(mid);
        ans=max(ans,x.first-mid*m);
        if(x.second<=m)rig=mid-1;
        else lef=mid+1;
    }
    cout<<ans;
    return 0;
}
```

6.4 Aliens Trace

```
// given partition P1, P2(P1.size()>=P2.size()), return K
partition
// 1-based, first element should be zero (P[i-1],P[i]]
vector<int> alien_track(int K, vector<int> P1, vector<int>
P2)
{
   vector<int> ret;
   int j=1;
   for(int i=1;i<P1.size();i++) {
      while(j<P2.size() && P1[i-1]>P2[j]) j++;
      if(P1[i]<=P2[j] && i-j==K-(int)P2.size()+1) {
         for(int k=0;k<i;k++) ret.push_back(P1[k]);
         for(int k=j;k<P2.size();k++) ret.push_back(P2[k]);
         return ret;
      }
   }
   exit(-1);
}
```

6.5 SOS DP

Usage: $F[mask] = \sum_{i \subseteq mask} A[i]$

Time Complexity: $\mathcal{O}(N2^N)$

```
for(int i = 0; i < (1 << N); i++) F[i] = A[i];
for(int j = 0; j < N; j++) for(int i = 0; i < (1 << N); i++)
  if(i & (1 << j)) F[i] += F[i ^ (1 << j)];
```

6.6 Fast Knapsack

Time Complexity: $\mathcal{O}(N\max(w_i))$

```
//computes the maximum S <= t such that S is the sum of some
subset of the weights.
// Description: Given N non-negative integer weights w and a
non-negative target t

int knapsack(vector<int> w, int t){
  int a=0, b=0, x;
  while(b<w.size() && a+w[b]<=t) a+=w[b++];
  if (b==w.size()) return a;
  int m=*max_element(w.begin(), w.end());
  vector<int> u, v(2*m, -1);
  v[a+m-t] = b;
  for(int i=b; i<w.size(); i++) {
    u=v;
    for(int x=0; x<m; x++) v[x+w[i]]=max(v[x+w[i]], u[x]);
    for(x=2*m; --x>m;) for(int j=max(0,u[x]); j<v[x]; j++)
      v[x-w[j]] = max(v[x-w[j]], j);
  }
  for(a=t; v[a+m-t]<0; a--);
  return a;
}
```

6.7 FastIO

```
static char buf[1<<19];
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,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;
}
```

6.8 OSrank

```
#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
```

6.9 mt19937

```
int rand(mt19937 &rd, int l, int r){
    // mt19937 rd((unsigned)chrono::steady_clock::now().
    time_since_epoch().count());
    // mt19937 rd(0x1119);
    uniform_int_distribution<int> rnd(l, r);
    return rnd(rd);
}
```

6.10 Bits Hacks

```
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));
}
```

6.11 Fast 64bit Modular Division

```
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;
}
```

6.12 Nasty Stack Hack

Usage: Use this when your complier sucks and your stack explodes. **BOOM!**

```
int main2(){ //do something
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;
}
```

6.13 Pragma Optimizer

```
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")
```

문제를 잘못 읽어도, 코딩이 꼬여도, 서버가 터져도 항상 침착하고 자신있게!