1 テンプレート

1.1 bash

```
##### gdb #####
*`setxkbmap -option ctrl:nocaps`
gdbcmd
```
r
bt
```
gdb a.out -x gdbcmd
echo -ne "r\nbt" | gdb ./a.out 2>&1 | grep -n2 $(basename `pwd`).cpp

##### build env #####
alias g++="g++ -std=c++1y -O2 -Wall -Wshadow -D_GLIBCXX_DEBUG -g"
mkdir {a..l}

vim template.cpp

for d in {a..l}
do
cp template.cpp $d/$d.cpp
done
```

1.2 .vimrc

```
set number
syntax on
set title
set cindent
set si
set sta
set ts=4 sw=4 ai noet

inoremap <C-b> <left>
inoremap <C-f> <right>
inoremap <C-l> <right>
inoremap <C-a> <esc>^i
inoremap <C-e> <end>

set backspace=start,eol,indent
```

1.3 CPP

```
#include <bits/stdc++.h>
#define rep(i,n) for(int i=0;(i)<(n);++(i))
using namespace std;

template<class T>bool chmax(T &a, const T &b) { return (a<b)?(a=b,1):0;}
template<class T>bool chmin(T &a, const T &b) { return (b<a)?(a=b,1):0;}

// 万が一
//#define _overload(_1,_2,_3,name,...) name
//#define _rep(i,n) _range(i,0,n)
//#define _range(i,a,b) for(int i=int(a);i<int(b);++i)
//#define rep(...) _overload(__VA_ARGS__,_range,_rep,)(__VA_ARGS__)

//#define uniq(arg) sort(_all(arg)),(arg).erase(unique(_all(arg)),end(arg))
//#define getidx(ary,key) lower_bound(_all(ary),key)-begin(ary)

int main(void){
    return 0;
}
```

2 ユーティリディ

2.1 XorShift

```
// Description: 乱数生成 XorShift
// TimeComplexity: O(1)
// Verifyed: NNERC15 J

struct Xorshift {
    uint32_t x = 4;
    uint32_t next() {
        x = x ^ (x << 13);
        x = x ^ (x >> 17);
        x = x ^ (x << 15);
        return x;
    }
};
```

2.2 DICE

```
// Description: 六面体サイコロ
// Verifyed: Many Diffrent Problem
```

```cpp

struct Dice {
  int l, r, f, b, d, u;
  Dice(int l, int r, int f, int b, int u, int d): l(l), r(r), f(f), b(b), u(u),
    d(d) {}

#define rotation(a,b,c,d) swap(a,b),swap(b,c),swap(c,d);

  void rot_f() { rotation(f, u, b, d);}
  void rot_b() { rotation(b, u, f, d);}
  void rot_l() { rotation(l, u, r, d);}
  void rot_r() { rotation(r, u, l, d);}
};
```

3 グラフ

3.1 グラフの定義

```cpp
// Description: グラフの型定義
// Verifyed: Many Diffrent Problem

// 実装の際にはよく考えて定義すること
using W = ll;
using edge = struct {int to, rev; W cap, flow, cost;};
using G = vector<vector<edge>>;

void add_edge(G &graph, int from, int to, W cap, W cost) {
  graph[from].push_back({to, int(graph[to].size()) , cap , 0 , cost});
  graph[to].push_back({from, int(graph[from].size()) - 1, 0 , 0, -cost});
}
```

3.2 全域木

3.2.1 最小有向全域木

```cpp
// Description: 連結グラフに対する最小有向全域木
// TimeComplexity: O(EV)
// Verifyed: AOJ GRL_2_B

G chu_liu(const G &graph, int root) {
  using W = int; const W inf = 1 << 28;
  const int n = graph.size();

  vector<int> cost(n, inf), pv(n, -1);
  rep(v, n) for (auto &e : graph[v]) {
    if (chmin(cost[e.to], e.cost)) pv[e.to] = v;
  }

  int led = -1, unuse = -1;
  vector<int> comp(n), used(n, -1);
  iota(begin(comp), end(comp), 0);

  rep(v, n) if (used[v] == -1 and pv[v] != -1 and v != root) {
    int cur = used[v] = v;
    for (cur = pv[cur]; cur != root and used[cur] == -1; cur = pv[cur]) {
      used[cur] = v;
    }
    if (used[cur] != v) continue;
    led = cur, comp[cur] = cur, v = n;
    for (int w = pv[cur]; w != cur; w = pv[w]) {
      comp[w] = cur;
    }
  }

  G ngraph(n), ret(n);
  using E = tuple<int, int, int>; map<E, E> dict;

  if (led == -1) {
    rep(v, n) {
      if (v != root and pv[v] != -1) {
        add_edge(ret, pv[v], v, cost[v]);
      }
    }
  } else {
    rep(v, n) for (auto &e : graph[v]) {
      int a = comp[v], b = comp[e.to];
      if (a == b) continue;
      int c = e.cost - (comp[e.to] == led) * cost[e.to];
      dict[E(a, b, c)] = E(v, e.to, e.cost);
      add_edge(ngraph, a, b, c);
    }

    G ntree = chu_liu(ngraph, comp[root]);

    rep(v, n) for (auto &e : ntree[v]) {
      int a, b, c;
      tie(a, b, c) = dict[E(v, e.to, e.cost)];
      if (comp[b] == led) unuse = b;
      add_edge(ret, a, b, c);
    }

    rep(v, n) {
      if (comp[v] == led and v != unuse) {
```

```
        add_edge(ret, pv[v], v, cost[v]);
      }
    }
  }
  return ret;
}
```

3.2.2 最小シュタイナー木 Dreyfus-Wagner 法

```
// Description: 連結グラフに対する最小シュタイナー木
// TimeComplexity: O(V^3 + V^2 2^T + V3^T) T はターミナル数
// Verifyed: AOJ 1040

auto Dreyfus_wagner(const G &graph, const auto &term) {
  using W = int; W inf = 1 << 28;
  const int n = int(graph.size()), t = int(term.size());
  if (t <= 1) return 0;
  auto dist = graph;
  vector<vector<W>> opt(1 << t, vector<W>(n, inf));

  rep(k, n)rep(i, n)rep(j, n) chmin(dist[i][j], dist[i][k] + dist[k][j]);

  map<int, int> dict;
  rep(i, t) {
    dict[term[i]] = 1 << i;
    rep(j, n) opt[1 << i][j] = dist[term[i]][j];
    rep(j, t) opt[(1 << i) | (1 << j)][term[j]] = dist[term[i]][term[j]];
  }

  auto next_combination = [&](int s) {
    int x = s & -s, y = s + x;
    return ((s & ~y) / x >> 1) | y;
  };

  rep(k, 2, t) {
    for (int s = (1 << k) - 1; s < (1 << t); s = next_combination(s)) {
      rep(i, n) {
        int ns = s | dict[i];
        if (dict[i] and s == ns) continue;
        for (int u = (s - 1)&s; u != 0; u = (u - 1)&s) {
          int c = s & (~u);
          chmin(opt[ns][i], opt[u][i] + opt[c][i]);
        }
      }
    }
    for (int s = (1 << k) - 1; s < (1 << t); s = next_combination(s)) {
      rep(i, n) {
        int ns = s | dict[i];
        if (dict[i] and s == ns) continue;
        rep(j, n) chmin(opt[ns][i], opt[s][j] + dist[j][i]);
      }
    }
  }

  const int mask = (1 << t) - 1;
  W ans = opt[mask][0];
  rep(i, n) chmin(ans, opt[mask][i]);
  return ans;
}
```

3.3 連結成分

3.3.1 Low-Link

```
// Description: 無向グラフに対する Low_Link
// TimeComplexity: O(V + E)
// Verifyed: AOJ GRL_3_A GRL_3_B

auto low_link(const G& graph) {
  int n = graph.size(), k = 0;
  vector<int> par(n), ord(n, -1), low(n), root(n, 0);

  auto dfs = [&](int v, int p, int &k) {
    auto func = [&](int v, int p, int &k, auto func)->void{
      ord[v] = k++, low[v] = ord[v], par[v] = p;
      for (auto &e : graph[v]) {
        if (e.to == p) continue;
        if (ord[e.to] == -1)
          func(e.to, v, k, func), chmin(low[v], low[e.to]);
        else
          chmin(low[v], ord[e.to]);
      }
    };
    return func(v, p, k, func);
  };

  rep(v, n) if (ord[v] == -1) dfs(v, -1, k), root[v] = 1;
  return make_tuple(par, ord, low, root);
}
```

3.3.2 関節点

```cpp
// Description: 無向グラフに対する関節点
// TimeComplexity: O(V + E)
// Verifyed: AOJ GRL_3_A
// Required: Low-link

auto articulation_point(const G& graph) {
    const int n = graph.size();
    vector<int> par, ord, low, root;
    tie(par, ord, low, root) = low_link(graph);

    vector<int> res;
    rep(v, n) {
        if (root[v]) {
            int degree = 0;
            for (auto &e : graph[v]) if (v == par[e.to]) degree++;
            if (degree >= 2) res.push_back(v);
        } else {
            for (auto &e : graph[v]) {
                if (v == par[e.to] && ord[v] <= low[e.to]) {
                    res.push_back(v);
                    break;
                }
            }
        }
    }
    return res;
}
```

3.3.3 橋

```cpp
// Description: 無向グラフに対する橋
// TimeComplexity: O(V + E)
// Verifyed: AOJ GRL_3_B
// Required: Low-link

auto bridge(const G& graph) {
    const int n = graph.size();
    vector<int> par, ord, low, root;
    tie(par, ord, low, root) = low_link(graph);

    using state = tuple<int, int>;
    vector<state> res;
    rep(v, n) {
        if (par[v] == -1) continue;
        if (ord[v] < low[par[v]] || ord[par[v]] < low[v]) {
            auto in = state(v, par[v]);
            if (get<1>(in) <= get<0>(in)) swap(get<0>(in), get<1>(in));
            res.push_back(in);
        }
    }
    sort(_all(res));
    return res;
}
```

3.3.4 強連結成分分解・2-SAT

```cpp
// Description: 2-SAT
// Verifyed: Many Diffrent Problem
// Required: 有向グラフに対する強連結成分

using edge = struct {int to;};
using G = vector<vector<edge>>;
void add_edge(G &graph, int from, int to) {
    graph[from].push_back({to});
}

// x&1 == 1 True
// x&1 == 0 False
void closure_or(G &graph, int a, int b) {
    add_edge(graph, a ^ 1, b);
    add_edge(graph, b ^ 1, a);
}

auto strongly_connected_components(const G& graph) {
    const int n = graph.size();

    vector<int> used(n, 0), order, scc(n, 0);

    auto dfs = [&](int v) {
        auto func = [&](int v, auto func)->void{
            used[v] = true;
            for (auto &e : graph[v]) if (!used[e.to]) func(e.to, func);
            order.push_back(v);
        };
        return func(v, func);
    };

    rep(v, n) if (used[v] == false) dfs(v);

    G rgraph(n);
    rep(v, n) for (auto &e : graph[v]) add_edge(rgraph, e.to, v);
```

```cpp
  int total = 0;

  auto rdfs = [&](int v) {
    auto func = [&](int v, auto func)->void{
      used[v] = true, scc[v] = total;
      for (auto &e : rgraph[v]) if (!used[e.to]) func(e.to, func);
    };
    return func(v, func);
  };

  used.assign(2 * n, false);
  reverse(begin(order), end(order));

  for (auto &v : order) if (used[v] == false) rdfs(v), total++;
  return scc;
}

vector<int> get_variable(G &graph) {
  const int n = graph.size() / 2;
  vector<int> ret(n, 0);

  vector<int> scc;
  scc = strongly_connected_components(graph);

  rep(i, n) {
    if (scc[2 * i] == scc[2 * i + 1])
      ret[0] = -1;
    else
      ret[i] = (scc[2 * i] < scc[2 * i + 1]);
  }
  return ret;
}
```

3.3.5 無向グラフの全域最小カット

```cpp
const int vmax=210;
int graph[vmax][vmax];

// Stoer-Wagner Uva 10989

int minimum_cut(int n){
  int h[vmax][vmax];
  rep(i,n)rep(j,n) h[i][j]=graph[i][j];
  vi vertex(n);rep(i,n) vertex[i]=i;

  int cut=inf;
  for(int m=n;m>1;m--){
    vi ws(m,0);
    int cur=-1,prev=-1;
    int w;
    rep(loop,m){
      prev=cur;
      cur=max_element(ws.begin(),ws.end())-ws.begin();
      w=ws[cur];ws[cur]=-1;
      rep(i,m) if(ws[i]>=0) ws[i]+=h[vertex[cur]][vertex[i]];
      //併合によるコスト加算
    }
    rep(i,m){
      // prev に cur を併合
      h[vertex[i]][vertex[prev]]+=h[vertex[i]][vertex[cur]];
      h[vertex[prev]][vertex[i]]+=h[vertex[cur]][vertex[i]];
    }
    vertex.erase(vertex.begin()+cur);
    cut=min(cut,w);
  }
  return cut;
}
```

3.4 パス・サイクル

3.4.1 トポロジカルソート

```cpp
// Description: 有向グラフに対するトポロジカルソート
// TimeComplexity: O(V + E)
// Verifyed: AOJ GRL_4_B

auto topological_sort(const G& graph) {
  const int n = graph.size();
  vector<int> used(n, 0), order;

  auto dfs = [&](int v) {
    auto func = [&](int v, auto func)->void{
      used[v] = true;
      for (auto &e : graph[v]) if (!used[e.to]) func(e.to, func);
      order.push_back(v);
    };
    return func(v, func);
  };

  rep(v, n)if (!used[v]) dfs(v);
  reverse(_all(order));
  return order;
}
```

3.5 木

3.5.1 Heavy-Light Decomposition

```cpp
// Verified: AOJ 2450
template <int V> struct HLD {
  int par[V], depth[V], heavy[V];
  int head[V], vid[V], inv[V];

  int dfs(const G& graph, int v, int p) {
    int sz = 1, smax = 0;
    for (auto &e : graph[v]) {
      if (e.to == p) continue;
      par[e.to] = v, depth[e.to] = depth[v] + 1;

      int sub_sz = dfs(graph, e.to, v);
      sz += sub_sz;

      if (smax < sub_sz) {
        smax = sub_sz, heavy[v] = e.to;
      }
    }
    return sz;
  }

  void init(const G& graph) {
    const int n = graph.size();
    fill_n(heavy, n, -1);
    dfs(graph, 0, -1);

    int id = 0;
    queue<int> q({ 0 });
    while (!q.empty()) {
      int h = q.front(); q.pop();
      for (int v = h; v != -1; v = heavy[v]) {
        inv[id] = v, vid[v] = id++, head[v] = h;
        for (auto &e : graph[v]) {
          if (e.to == par[v] or e.to == heavy[v]) continue;
          q.push(e.to);
        }
      }
    }
  }

  // 頂点属性の for_each （有向 f の 3 番目の引数には順方向なら 0、逆方向なら 1 が渡される
  void for_each_v_directed(int u, int v, auto &f) {
    if (vid[u] > vid[v]) {
      f(max(vid[head[u]], vid[v]), vid[u] + 1, 1);
      if (head[u] != head[v]) for_each_v_directed(parent[head[u]], v, f);
    } else {
      f(max(vid[head[v]], vid[u]), vid[v] + 1, 0);
      if (head[u] != head[v]) for_each_v_directed(u, parent[head[v]], f);
    }
  }

  // 辺属性の for_each （有向 f の 3 番目の引数には順方向なら 0、逆方向なら 1 が渡される
  void for_each_e_directed(int u, int v, auto &f) {
    if (vid[u] > vid[v]) {
      if (head[u] != head[v]) {
        f(vid[head[u]], vid[u], 1);
        for_each_e(par[head[u]], v, f);
      } else {
        if (u != v) f(vid[v] + 1, vid[u], 1);
      }
    } else {
      if (head[u] != head[v]) {
        f(vid[head[v]], vid[v], 0);
        for_each_e_directed(u, par[head[v]], f);
      } else {
        if (u != v) f(vid[u] + 1, vid[v], 0);
      }
    }
  }

  // 頂点 u の d 個上の頂点を求める（存在しないなら 0 を返す）
  int ancestor(int u, int d) {
    while (1) {
      if (depth[head[u]] <= depth[u] - d) break;
      d -= depth[u] - depth[head[u]] + 1;
      if (head[u] == 0) return 0;
      u = parent[head[u]];
    }
    return inv[vid[u] - d];
  }

  // 頂点 u と頂点 v の LCA を求める
  int lca(int u, int v) {
    if (vid[u] > vid[v]) swap(u, v);
    if (head[u] == head[v]) return u;
    return lca(u, parent[head[v]]);
  }

  // 頂点 u と頂点 v の距離を求める
  int dist(int u, int v) { return depth[u] + depth[v] - 2 * depth[lca(u, v)];}
```

```cpp
};

HLD<nmax> hld;

int main(void{
    hld.init(tree);
    auto func = [&](int u, int v, int d) {
        seg.range_update(u, v);
    };
    hld.for_each_vertex_directed(a, b, func);
    return 0;
}
```

3.5.2 重心分解

```cpp
bool used[limit];
int sz[limit];

int calc_sub(const G &graph, int v, int p) {
    sz[v] = 1;
    for (auto &e : graph[v]) {
        if (e.to == p or used[e.to]) continue;
        sz[v] += calc_sub(graph, e.to, v);
    }
    return sz[v];
}

int serach_centroid(const G &graph, int v, int p, int total) {
    for (auto &e : graph[v]) {
        if (e.to == p or used[e.to]) continue;
        if (sz[e.to] > total / 2) return serach_centroid(graph, e.to, v, total);
    }
    return v;
}

void calc_dist(const G &graph, int v, int p, ll d, vector<int> &res) {
    res.push_back(d);
    for (auto &e : graph[v]) {
        if (e.to == p or used[e.to]) continue;
        calc_dist(graph, e.to, v, d + e.cost, res);
    }
}


void solve(const G &graph, int v) {
    calc_sub(graph, v, -1);
    const int c = serach_centroid(graph, v, -1, sz[v]);
    used[c] = true;

    vector<vector<int>> dist;
    for (auto &e : graph[c]) {
        if (used[e.to]) continue;
        dist.push_back(vector<int>());
        calc_dist(graph, e.to, c, e.cost, dist.back());
    }

    for (auto &e : graph[c]) {
        if (used[e.to]) continue;
        solve(graph, e.to);
    }

    // processing here
}
```

3.6 最大流

3.6.1 Dinic 法

```cpp
// Description: グラフに対する最大流
// TimeComplexity: O(EV^2) but fast
// Verifyed: AOJ GRL_6_A

W dinic(G &graph, int s, int t) {
    const W inf = 1LL << 50;
    const int n = graph.size();
    vector<int> level(n), iter(n);

    auto bfs = [&](int s, int t) {
        fill(begin(level), end(level), -1);
        queue<int> q;
        level[s] = 0, q.push(s);

        while (!q.empty()) {
            int v = q.front(); q.pop();
            for (auto &e : graph[v]) {
                if (level[e.to] == - 1 and e.cap > e.flow) {
                    level[e.to] = level[v] + 1;
                    q.push(e.to);
                }
            }
        }
        return (level[t] != -1);
    };

```

```
  auto dfs = [&](int v, int t, W f) {
    auto func = [&](int v, int t, W f, auto func)->W{

      if (v == t) return f;
      for (int &i = iter[v]; i < graph[v].size(); i++) {
        edge &e = graph[v][i];

        if (e.cap > e.flow and level[v] < level[e.to]) {

          W d = func(e.to, t, min(f, e.cap - e.flow), func);

          if (d > 0) {
            e.flow += d, graph[e.to][e.rev].flow -= d;
            return d;
          }
        }
      }

      return 0;
    };
    return func(v, t, f, func);
  };

  while (bfs(s, t)) {
    fill(begin(iter), end(iter), 0);
    while (dfs(s, t, inf) != 0 );
  }

  W ret = 0;
  for (auto &e : graph[s]) ret += e.flow;
  return ret;
}
```

3.6.2 残余グラフ

```
// Description: フローの復元
// TimeComplexity: O(E)
// Verifyed: not

auto positive_edges(G &graph) {
  vector<pair<int, int>> ret;
  const int n = graph.size();
  rep(v, n) for (auto &e : graph[v]) {
    if (e.cap - e.flow > 0) {
      ret.push_back(make_pair(v, e.to));
    }
  }
  return ret;
}
```

3.7 最小費用流

3.7.1 Primal-Dual 法

```
// Description: グラフに対する最小費用流
// TimeComplexity: O(FEV)
// Verifyed: AOJ GRL_6_B

W primal_dual(G &graph, int s, int t, int f) {
  const W inf = 1LL << 50;
  W res = 0;
  while (f) {
    int n = graph.size(), update;
    vector<W> dist(n, inf);
    vector<int> pv(n, 0), pe(n, 0);
    dist[s] = 0;

    rep(loop, n) {
      update = false;
      rep(v, n)rep(i, graph[v].size()) {
        edge &e = graph[v][i];
        if (e.cap > e.flow and chmin(dist[e.to], dist[v] + e.cost)) {
          pv[e.to] = v, pe[e.to] = i;
          update = true;
        }
      }
      if (!update) break;
    }

    if (dist[t] == inf) return -1;

    W d = f;

    for (int v = t; v != s; v = pv[v]){
      chmin(d, graph[pv[v]][pe[v]].cap - graph[pv[v]][pe[v]].flow);
    }

    f -= d, res += d * dist[t];

    for (int v = t; v != s; v = pv[v]) {
      edge &e = graph[pv[v]][pe[v]];
      e.flow += d;
      graph[v][e.rev].flow -= d;
    }
```

```
  }
  return res;
}
```

3.8 マッチング

3.8.1 2 部グラフの最大マッチング

```
// Description: 2部グラフに対する最大マッチング
// TimeComplexity: O(EV)
// Verifyed: AOJ GRL_7_A

int bipartite_matching(const G &graph) {
  int res = 0, n = graph.size();
  vector<int> match(n, -1), used(n, 0);

  auto dfs = [&](int v) {
    auto func = [&](int v, auto func)->int{
      used[v] = true;
      for (auto &e : graph[v]) {
        const int u = e.to, w = match[u];
        if (w < 0 || (!used[w] && func(w, func))) {
          match[v] = u, match[u] = v;
          return true;
        }
      }
      return false;
    };
    return func(v, func);
  };

  rep(v, n) {
    if (match[v] >= 0) continue;
    fill(_all(used), false);
    if (dfs(v)) res++;
  }
  return res;
}
```

3.9 極大独立集合

```
// Description: 無向グラフに対する最大独立集合
// TimeComplexity: O(1.466^V V)
// Verified: not

auto maximum_independent_sets(const G& graph) {
  const int n = graph.size();
  using T = bitset<40>;
  vector<T> edge(n);
  rep(i, n) for (auto &e : graph[i]) edge[i].set(e.to);

  auto dfs = [&](int v, T used) {
    auto func = [&](int v, T used, auto func)->T{
      if (v == n) return T("0");
      if (used[v]) return func(v + 1, used, func);

      used.set(v);
      T ret = func(v + 1, used | edge[v], func);
      ret.set(v);

      if (((~used)&edge[v]).count() >= 2) {
        T arg = func(v + 1, used, func); // not used v
        if (ret.count() <= arg.count()) ret = arg;
      }

      return ret;
    };
    return func(v, used, func);
  };
  return dfs(0, T());
}
```

3.10 Dominator Tree

```
// Description: Dominator Tree
// TimeComplexity: O(M log N)
// Verifyed: AOJ 0294

struct Union_find {
  Union_find(int n) {
    par.resize(n), iota(_all(par), 0);
    idx.resize(n), iota(_all(idx), 0);
    semi.assign(n, -1);
  }

  int find(int x) {
    if (par[x] == x) return x;
    int r = find(par[x]);
    if (semi[idx[par[x]]] < semi[idx[x]]) idx[x] = idx[par[x]];
    return par[x] = r;
  }

```

```
  void link(int a, int b) {par[b] = a;}
  int eval(int x) {find(x); return idx[x];}

  vector<int> par, idx, semi;
};

auto dominator_tree(G &graph, int root) {
  const int n = graph.size();

  G rgraph(n);
  rep(v, n) for (auto &e : graph[v]) add_edge(rgraph, e.to, v, e.cost);

  Union_find uf(n);

  vector<int> id(n, -1), par(n), u(n, -1);
  int total = 0;

  auto dfs = [&](int v, int p) {
    auto func = [&](int v, int p, auto func)->void{
      uf.semi[v] = total, id[total++] = v, par[v] = p;
      for (auto &e : graph[v]) {
        if (uf.semi[e.to] >= 0) continue;
        func(e.to, v, func);
      }
    };
    return func(v, p, func);
  };

  dfs(root, -1);
  vector<vector<int>> bucket(n);

  rrep(i, n, 1) {
    int v = id[i];
    for (auto &e : rgraph[v]) {
      int u = uf.eval(e.to);
      chmin(uf.semi[v], uf.semi[u]);
    }

    bucket[id[uf.semi[v]]].push_back(v);

    for (auto &nv : bucket[par[v]]) u[nv] = uf.eval(nv);
    bucket[par[v]].clear();
    uf.link(par[v], v);
  }

  vector<int> idom(n, 0);

  rep(i, 1, n) {
    int v = id[i], w = u[v];
    idom[v] = (uf.semi[v] == uf.semi[w]) ? uf.semi[v] : idom[w];
  }

  rep(i, 1, n) {
    int v = id[i];
    idom[v] = id[idom[v]];
  }

  idom[root] = -1;
  return idom;
}
```

4 動的計画法

4.1 Convex hull trick

次の性質を満たす漸化式の計算を $\mathcal{O}(n^2)$ から $\mathcal{O}(n)$ に高速化

$$\mathrm{dp}[i] = \min_{0 \le j < i} \mathrm{dp}[j] + \mathrm{a}[j] * \mathrm{x}[i] + \mathrm{b}[j] \tag{1}$$

```
// Description: DP using Convex hull Trick
// TimeComplexity: O(n)
// Verifyed: AOJ 2725

//Inequality sign min >=  max <=
template<typename T> class CHT {
public:
  T getval(T x) {
    while (deq.size() >= 2 && f(deq[0], x) <= f(deq[1], x)) deq.pop_front();
    return f(deq[0], x);
  }

  void push(T ca, T cb) {
    const int idx = a.size();
    a.push_back(ca), b.push_back(cb);
    while (deq.size() >= 2 && check(idx)) deq.pop_back();
    deq.push_back(idx);
  }

  int size() {return deq.size();}
private:
  vector<T> a, b;
  deque<int> deq;

  T f(int idx, T x) {return 1LL * a[idx] * x + b[idx];}
```

```
  bool check(int idx) {
    const int i = deq[size() - 2], j = deq[size() - 1], k = idx;
    return (b[j] - b[i]) * (a[k] - a[j]) <= (b[k] - b[j]) * (a[j] - a[i]);
  }
};
```

4.2 Monge 性

コスト関数が次の条件を満たしているかに注意して使用すること Quadrangle inequality Monotonicity:

4.2.1 Divide and Conquer Optimization

次の性質を満たす漸化式の計算を $\mathcal{O}(n^2)$ から $\mathcal{O}(n)$ に高速化

$$\mathrm{dp}[i][j] = \min_{0 \le k < j} \mathrm{dp}[i-1][k] + \mathrm{cost}[k][j] \tag{2}$$

```
int dp[kmax][nmax];
void solve(int i, int L, int R, int optL, int optR) {
  if (L > R) return;
  int M = (L + R) / 2;
  int optM = -1;
  for (int k = optL; k <= optR; ++k)
    if (dp[i + 1][M] > dp[i][k] + cost[k][M])
      dp[i + 1][M] = dp[i][k] + cost[k][M] , optM = k;
  solve(i, L, M, optL, optM);
  solve(i, M, R, optM, optR);
  return;
}

int main(void) {
  for (int i = 0; i <= k; ++i)
    for (int j = 0; j <= n; ++j)
      dp[i][j] = inf;
  dp[0][0] = 0;
  for (int i = 0; i < k; ++i) solve(i, 0, n, 0, n);
  cout << dp[k][n] << endl;
  return 0;
}
```

4.2.2 Knuth Optimization

$$\mathrm{dp}[i][j] = \min_{i < k < j} \mathrm{dp}[i][k] + \mathrm{dp}[k][j] + \mathrm{cost}[i][j] \tag{3}$$

```
int dp[nmax][nmax];
int C[nmax][nmax];

int main(void) {
  for (int i = 0; i <= k; ++i)
    for (int j = 0; j <= n; ++j)
      dp[i][j] = inf;
  for (int i = 0; i < n; ++i) dp[i][i] = 0, C[i][i] = i;

  for (int d = 2; d <= n; ++d) {
    for (int i = 0; i + d - 1 < n; ++i) {
      int L = i, R = i + d - 1;
      int idx = C[L][R - 1];
      for (int j = C[L][R - 1]; j <= C[L + 1][R]; ++j) {
        if (dp[L][R] > dp[L][j] + dp[j + 1][R] + cost[L][R])
          dp[L][R] = dp[L][j] + dp[j + 1][R] + cost[L][R], idx = j;
      }
      C[L][R] = idx;
    }
  }
  cout << dp[0][n - 1] << endl;
  return 0;
}
```

5 データ構造

5.1 Union Find

```
// Description: 素集合を管理するデータ構造
// TimeComplexity: 初期化 O(n) 更新 O(log n)
// Verifyed: AOJ DSL_1_A

struct Union_find{
  Union_find(int n){par.resize(n),iota(_all(par),0);}
  int find(int x){return (par[x]==x)?x:par[x]=find(par[x]);}
  void unite(int a,int b){a=find(a),b=find(b);par[a]=b;}
  bool same(int a,int b){return find(a)==find(b);}
  vector<int> par;
};
```

5.2 マージの一般的なテクニック

2 つのデータ構造をマージする時はサイズの小さいデータ構造をサイズの大きいデータ構造に挿入する. $\mathcal{O}(\log n)$

5.3 セグメント木

5.3.1 RMQ

```
// Description: セグメント木 点更新 区間クエリ
// TimeComplexity: 初期化 O(n log n) 更新とクエリ O(log n)
// Verifyed: AOJ DSL_2_A

struct Segment_tree {
  using T = ll;

  int n;
  vector<T> data;
  const T out = (1LL << 31) - 1;

  inline T vmerge(T l, T r) {return min(l, r);}

  Segment_tree(int n): n(n) {data.assign(2 * n, out);}

  void update(int p, T x) { // set value at position p
    for (data[p += n] = x; p > 1; p >>= 1) data[p >> 1] = vmerge(data[p], data[p ^ 1]);
  }

  T query(int l, int r) {  // sum on interval [l, r)
    T resl = out, resr = out;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) resl = vmerge(data[l++], resl);
      if (r & 1) resr = vmerge(resr, data[--r]);
    }
    return vmerge(resl, resr);
  }
};
```

5.3.2 Starry Sky Tree

```
template <int depth>  struct Segment_tree {
  const static int h = depth;
  const static int n = 1 << h;

  using T = long long;
  T data[2 * n], lazy[2 * n];
  const T out = (1LL << 31) - 1;

  inline T vmerge(T l, T r) {return min(l, r);}

  void init() {
    fill_n(data, 2 * n, out);
    fill_n(lazy, 2 * n, out);
  }

  // lazy_evaluation
  void apply(T v, int p, int l, int r) {
    data[p] = v;
    lazy[p] = v;
  }

  void push(int p, int l, int r) {
    const int m = (l + r) / 2;
    if (lazy[p] == out) return;
    apply(lazy[p], 2 * p + 1, l, m);
    apply(lazy[p], 2 * p + 2, m, r);
    lazy[p] = out;
  }

  void range_update(int a, int b, T x, int k = 0, int l = 0, int r = n) {
    if (r <= a or b <= l) return;
    if (a <= l and r <= b) return apply(x, k, l, r);
    push(k, l, r);
    const int m = (l + r) / 2;
    range_update(a, b, x, k * 2 + 1, l, m);
    range_update(a, b, x, k * 2 + 2, m, r);
    data[k] = vmerge(data[k * 2 + 1], data[k * 2 + 2]);
  }

  T range_query(int a, int b, int k = 0, int l = 0, int r = n) {
    if (r <= a or b <= l) return out;
    if (a <= l and r <= b) return data[k];
    push(k, l, r);
    const int m = (l + r) / 2;
    T vl = range_query(a, b, k * 2 + 1, l, m);
    T vr = range_query(a, b, k * 2 + 2, m, r);
    return vmerge(vl, vr);
  }
};

Segment_tree<17> seg;
```

5.4 Binary Indexed Tree

```
// Description: [1,x] のクエリに対するデータ構造
// TimeComplexity: 更新 O(log n) クエリ O(log n)
// Verifyed: ARC 033 C
```

```cpp

template <int depth> struct Binary_indexed_tree {
  using T = int;

  const static int h = depth;
  const static int n = 1 << h;
  T data[n];

  void init() {
    fill_n(heavy, n, 0);
  }

  void update(int i, T x) {
    for (; i < n; i += i & -i) {
      data[i] += x;
    }
  }

  T query(int i) {
    T ret = 0;
    for (; i > 0; i -= i & -i) ret += data[i];
    return ret;
  }

  int lower_bound(T x) {
    if (x <= 0) return 0;
    int i = 0;
    for (int k = n; k > 0; k >>= 1) {
      if (i + k < n and data[i + k] < x)
        x -= data[i + k], i += k;
    }
    return i + 1;
  }
};
```

6 文字列

6.1 KMP

```cpp
// Description: 文字列のパターンマッチングオートマトン
// TimeComplexity: O(|S|)
// Verifyed: CF 201 B Todo

// kmp[i]=s[0,i-1] の prefix と suffix の最長共通文字列の長さ
auto init(const string &s) {
  int n = s.size(), j = -1;
  vector<int> kmp(n + 1, -1);

  rep(i, n) {
    while (j >= 0 && s[i] != s[j]) j = kmp[j];
    j++, kmp[i + 1] = (s[i] == s[j]) ? kmp[j] : j;
  }
  return kmp;
}

int match(string p, string s, const auto &kmp) {
  int cur = 0, m = p.size(), n = s.size();
  rep(i, n) {
    while (cur >= 0 && s[i] != p[cur]) cur = kmp[cur];
    cur++;
    if (cur >= m) return cur; // cur=kmp[cur];
  }
  return cur;
}
```

6.2 Aho-Corasick

```cpp
// Description: 複数文字列パターンマッチングオートマトン
// TimeComplexity: O(∑|S|)
// Verifyed: Todo

const int sigma = 26;
inline int convert(char& arg) {
  // 1-indexed
  return arg - 'a' + 1;
}

vector<vector<int>> fg; // 0 faliure otherwise goto
vector<vector<int>> ac;

auto set_union(const vector<int> &a, const vector<int> &b) {
  vector<int> res;
  set_union(begin(a), end(a), begin(b), end(b), back_inserter(res));
  return res;
}

void add_state() {
  fg.push_back(vector<int>(1 + sigma, 0));
  ac.push_back(vector<int>());
}

int build(vector<string> &pattern) {
  fg.clear();
```

```
    ac.clear();

    const int root = 1;
    rep(loop, 2) add_state();
    fg[root][0] = root; // root failure

    // Trie
    rep(i, pattern.size()) {
        int now = root;
        for (auto &c : pattern[i]) {
            int j = convert(c);

            if (fg[now][j] == 0) {
                fg[now][j] = int(fg.size());
                add_state();
            }
            now = fg[now][j];
        }
        ac[now].push_back(i);
    }

    // Aho-corasick
    queue<int> q;
    for (int i = 1; i <= sigma; ++i) {
        if (fg[root][i]) {
            fg[fg[root][i]][0] = root;
            int nxt = fg[root][i];
            q.push(nxt);
        } else
            fg[root][i] = root;
    }

    // abcと遷移した時にbcも検知できるようにしている.
    while (!q.empty()) {
        int now = q.front(); q.pop();
        for (int i = 1; i <= sigma; ++i) {
            if (fg[now][i]) {
                int nxt = fg[now][0];
                while (!fg[nxt][i]) nxt = fg[nxt][0];
                fg[fg[now][i]][0] = fg[nxt][i];
                ac[fg[now][i]] = set_union(ac[fg[now][i]], ac[fg[nxt][i]]);
                q.push(fg[now][i]);
            }
        }
    }
    return root;
}

vector<int> match(int root, string &s, vector<string> &pattern) {
    int now = root;
    vector<int> res(pattern.size(), 0);
    for (auto &c : s) {
        int i = convert(c);
        while (!fg[now][i]) now = fg[now][0];
        now = fg[now][i];
        for (auto &j : ac[now]) res[j]++;
    }
    return res;
}
```

6.3 Rolling Hash

```
using ull = unsigned long long;
const ull b = 1000000007;
const ull binv = 13499267949257065399U;
const int max_size = 1010;
char pattern[max_size][max_size];
char target[max_size][max_size];

ull hash[max_size][max_size], tmp[max_size][max_size];

void calc(char a[max_size][max_size], int n, int m, int r, int c) {
    const ull br = 1000000007;
    const ull bc = 1000000009;

    ull tc = 1;
    rep(i, c) tc *= bc;

    rep(i, n) {
        ull e = 0;
        rep(j, c) e = e * bc + a[i][j];
        rep(j, m) {
            tmp[i][j] = e;
            if (j + c >= m) break;
            e = e * bc - tc * a[i][j] + a[i][j + c];
        }
    }

    ull tr = 1;
    rep(j, r) tr *= br;

    rep(j, m - c + 1) {
        ull e = 0;
```

```
    rep(i, r) e = e * br + tmp[i][j];
    rep(i, n) {
      hash[i][j] = e;
      if (i + r >= n) break;
      e = e * br - tr * tmp[i][j] + tmp[i + r][j];
    }
  }
  return;
}
```

6.4 Suffix Array

```
// Description: 文字列の接尾辞配列
// TimeComplexity: O(|S| log^2 |S|)
// Verifyed: ARC050 D

auto suffix_array(string s) {
  const int n = s.size();
  vector<int> sa(n + 1), rnk(n + 1, 0), tmp(n + 1, 0);
  rep(i, n + 1) sa[i] = i, rnk[i] = (i < n ? s[i] : -1);
  for (int k = 1; k <= n; k <<= 1) {
    auto cmp = [&](int i, int j) {
      if (rnk[i] != rnk[j]) return rnk[i] < rnk[j];
      int ci = (i + k <= n) ? rnk[i + k] : -1;
      int cj = (j + k <= n) ? rnk[j + k] : -1;
      return ci < cj;
    };
    sort(begin(sa), end(sa), cmp);
    tmp[sa[0]] = 0;
    rep(i, n) tmp[sa[i + 1]] = tmp[sa[i]] + cmp(sa[i], sa[i + 1]);
    rep(i, n + 1) rnk[i] = tmp[i];
  }
  return make_tuple(rnk, sa);
}
```

6.5 LCP Array

```
// Description: 文字列の最長共通接頭辞
// TimeComplexity: O(|S|)
// Verifyed: ARC050 D

auto longest_common_prefix(string s, const auto &rnk, const auto &sa) {
  int n = s.size(), h = 0;
  vector<int> lcp(n + 1, 0);
  rep(i, n) {
    int j = sa[rnk[i] - 1];
    for (h = max(h - 1, 0); j + h < n && i + h < n; h++) if (s[j + h] != s[i + h])
      ↪ break;
    lcp[rnk[i] - 1] = h;
  }
  return lcp;
}
```

6.6 Z algorithm

```
// Description: Z[i]=s と s[i..] の最長共通部分列の長さ
// TimeComplexity: O(N)
// Verifyed: ARC 060 D

auto z_algorithm(const string &s) {
  int n = s.size(), i = 1, j = 0, k;
  vector<int> z(n, n);
  while (i < n) {
    while (i + j < n && s[i + j] == s[j]) j++;
    z[i] = j, k = !!j;
    while (i + k < n && k + z[k] < j) z[i + k] = z[k], k++;
    i += k + !j, j -= k;
  }
  return z;
}
```

6.7 manacher

```
auto manacher(const string &in) {
  int n = in.size();
  string s(2 * n - 1, '#');
  rep(i, n) s[2 * i] = in[i];
  n = 2 * n - 1;

  vector<int> r(n);
  int i = 0, j = 0, k;
  while (i < n) {
    while (0 <= i - j && i + j < n && s[i - j] == s[i + j])j++;
    r[i] = j, k = 1;
    while (0 <= i - k && i + k < n && k + r[i - k] < r[i])r[i + k] = r[i - k], k++;
    i += k, j -= k;
  }
  return r;
}
```

7 数学

7.1 素数判定

```
ll multiple(ll a, ll b, ll mod) {
  ll res = 0LL;
  for (; b; a = ADD(a, a, mod), b >>= 1) if (b & 1) res = ADD(res, a, mod);
  return res;
}

ll power(ll a, ll n, ll mod) {
  ll res = 1LL;
  for (; n; a = multiple(a, a, mod), n >>= 1)if (n & 1) res = multiple(res, a,
      mod);
  return res;
}

bool isprime(ll n) {
  if (n == 1) return false;
  if (n % 2 == 0) return (n == 2);
  vector<int> base = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}; // n < 2^64
  //vector<int> base = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41}; // n < 10^24

  ll s = __builtin_ctz(n - 1), d = (n - 1) >> s;
  rep(i, 12) {
    if (power(base[i], d, n) == 1) continue;
    bool ok = false;
    rep(j, s) if (power(base[i], d << j, n) == n - 1) ok = true;
    if (ok == false) return false;
  }
  return true;
}
```

7.2 EXTGCD 中国剰余定理

```
// Description: Chinese Remainder Theorm
// TimeComplexity: O(log |m|)
// Verifyed: CF 724 C

inline tuple<ll, ll, ll> extgcd(ll a, ll b) {
  ll x = 1LL, y = 0LL, g = a;
  if (b != 0) {
    tie(g, y, x) = extgcd(b, a % b);
    y -= a / b * x;
  }
  return make_tuple(g, x, y);
}

auto chinese_remainder(ll cb, ll cm, ll nb, ll nm) {
  ll g, x, y;
  tie(g, x, y) = extgcd(cm, nm);
  ll d = (nb - cb + nm) % nm;
  if (d % g != 0) return make_tuple(-1LL, -1LL);
  d /= g, cb += cm * x * d;
  ll rm = cm / g * nm, rb = (cb % rm + rm) % rm;
  return make_tuple(rb, rm);
}
```

7.3 中国剰余定理

```
// verify yukicoder 0187
// http://www.csee.umbc.edu/~lomonaco/s08/441/handouts/Garner-Alg-Example.pdf
ll chinese_remainder(vector<ll> b,vector<ll> m){
  int n=m.size();
  set<ll> prime;
  rep(i,n){
    ll cur=m[i];
    for(ll f=2;f*f<=cur;++f){
      if(cur%f) continue;
      while(cur%f==0) cur/=f;
      prime.insert(f);
    }
    if(cur>1) prime.insert(cur);
  }

  vector<ll> factor(n);
  for(auto &p:prime){
    int index=0;
    rep(i,n){
      factor[i]=1LL;
      ll cur=m[i];
      while(cur%p==0) factor[i]*=p,cur/=p;
      if(factor[i]>factor[index]) index=i;
    }
    rep(i,n)if(i!=index){
      if(factor[i]==1) continue;
      if(b[index]%factor[i]!=b[i]%factor[i]) return -1;
      m[i]/=factor[i],b[i]%=m[i];
    }
  }

  vector<ll> constant(n,0LL),coef(n,1LL),v(n,0LL);
```

```
  rep(i,n){
    v[i]=SUB(b[i],constant[i],m[i]);
    v[i]=DIV(v[i],coef[i],m[i]);
    rep(j,i+1,n){
      constant[j]=ADD(constant[j],MUL(v[i],coef[j],m[j]),m[j]);
      coef[j]=MUL(coef[j],m[i],m[j]);
    }
  }

  ll ans=0LL;
  rrep(i,n) ans=ADD(MUL(ans,m[i],mod),v[i],mod);
  return ans;
}
```

7.4 オイラーの ϕ 関数

$$\phi(n) = n \prod_{k=1}^{d} \frac{p_k - 1}{p_k} \tag{4}$$

7.5 メビウスの反転公式

```
int mobius[limit];
int prime[limit];

void init() {
  rep(i, limit) prime[i] = i;

  for (ll i = 2; i * i < limit; ++i) {
    if (prime[i] == i) {
      for (ll j = i * i; j < limit; j += i) {
        prime[j] = i;
      }
    }
  }

  mobius[1] = 1;
  for (ll i = 2; i < limit; ++i) {
    if (i == prime[i])
      mobius[i] = -1;
    else
      mobius[i] = -1 * mobius[i / prime[i]];
  }

  for (ll i = 2; i * i < limit; ++i) {
    if (i == prime[i]) {
      for (ll j = i * i; j < limit; j += i * i) {
        mobius[j] = 0;
      }
    }
  }
}
```

7.6 高速ゼータ・メビウス変換

```
// Fast Zeta Transfrom
// g(S) = \sum_{S \subseteq T} f(T)
rep(i, n) rep(j, 1 << n) if (!(j & (1 << i))) f[j] += f[j | (1 << i)];
// Fast Mobius Transfrom
// g(S) = \sum_{T \subseteq S} -1^{|S|-|T|} f(T)
rep(i, n) rep(j, 1 << n) if (j & (1 << i)) f[j] -= f[j ^ (1 << i)];
```

7.7 行列

```
using vec = valarray<R>;
using mat = valarray<vec>;

mat mul(mat a, mat b) {
  int m = a.size();
  mat c(vec(0.0, m), m);
  rep(i, m)rep(j, m) rep(k, m) c[i][j] += a[i][k] * b[k][j];
  return c;
}

mat power(mat a, int n) {
  int m = a.size();
  mat b(vec(0.0, m), m);
  rep(i, m) b[i][i] = 1.0;
  while (n) {
    if (n & 1) b = mul(b, a);
    a = mul(a, a);
    n >>= 1;
  }
  return b;
}
```

7.7.1 連立一次方程式

```
inline int pivoting(mat &a, int k, int &c) {
  int n = a.size(), m = a[0].size(), p = k, ret = 0;
  for (; c < m; ++c) {
```

```cpp
		R cmax = abs(a[k][c]);
		rep(i, k + 1, n) if (chmax(cmax, abs(a[i][c]))) p = i, ret = 1;
		if (cmax > eps) break;
	}
	if (k != p) swap(a[k], a[p]);
	return ret;
}

int forward(mat &a) {
	int n = a.size(), m = a[0].size(), ret = 0, c = 0;
	rep(i, n - 1) {
		ret += pivoting(a, i, c);
		if (abs(a[i][c]) < eps) break;
		rep(j, i + 1, n) {
			R coef = 1.0 * a[j][c] / a[i][c];
			rep(k, c, m) a[j][k] -= 1.0 * coef * a[i][k];
		}
	}
	return ret;
}

int rank(mat &a) {
	int n = a.size(), m = a[0].size(), ret = 0;
	rep(i, n)rep(j, m) if (abs(a[i][j]) > eps) ret = i + 1;
	return ret;
}

double det(mat &a, int sgn) {
	R ret = 1.0;
	int n = a.size(), m = a[0].size();
	rep(i, n) ret *= a[i][i];
	if (sgn & 1) ret *= -1.0;
	return ret;
}

//backward substitution

vec back(mat &a) {
	int n = a.size(), m = a[0].size();
	vec x(0.0, n);
	for (int i = n - 1; i >= 0; i--) {
		R sum = 0.0;
		if (i + 1 < n) rep(j, i + 1, n) sum += 1.0 * a[i][j] * x[j];
		x[i] = 1.0 * (a[i][m - 1] - sum) / a[i][i];
	}
	return x;
}

int answer(mat &a) {
	int n = a.size(), m = a[0].size();
	int arank = 0, brank = 0;
	rep(i, n)rep(j, m - 1) if (abs(a[i][j]) > eps) arank = i + 1;
	rep(i, n)rep(j, m) if (abs(a[i][j]) > eps) brank = i + 1;
	if (arank != brank) return 0;
	if (arank < n) return 2;
	return 1;
}

vec gauss_jordan(mat &a) {
	const int n = a.size(), m = a[0].size();
	rep(i, n) {
		int pivot = i;
		rep(j, i, n) if (abs(a[j][i]) > abs(a[pivot][i])) pivot = j;
		swap(a[i], a[pivot]);

		if (abs(a[i][i]) < eps) return vec();

		rep(j, i + 1, m) a[i][j] /= a[i][i];
		rep(j, n) if (i != j) rep(k, i + 1, m) a[j][k] -= a[j][i] * a[i][k];
	}

	vec x(0.0, n);
	rep(i, n) x[i] = a[i][n];
	return x;
}
```

7.8 ハンガリアン法

```cpp
vector<int> min_cost_match(vector<vector<int>> a) {
	int n = a.size(), m = a[0].size();
	vector<int> left(n, -1), right(m, -1);
	vector<int> ofsleft(n, 0), ofsright(m, 0);
	auto cost = [&](int i, int j) { return a[i][j] + ofsleft[i] + ofsright[j];};

	rep(r, n) {
		vector<bool> s(n, false), t(m, false);
		vector<int> trace(m, -1), edge(m, r);
		s[r] = true;

		int b = -1;
		while (1) {
			int d = numeric_limits<int>::max();
			rep(j, m) if (!t[j]) d = min(d, cost(edge[j], j));
```

```
            rep(i, n) if (s[i]) ofsleft[i] -= d;
            rep(j, m) if (t[j]) ofsright[j] += d;

            rep(j, m) if (!t[j] && cost(edge[j], j) == 0) b = j;
            trace[b] = edge[b];
            int c = right[b];
            if (c < 0) break;

            s[c] = t[b] = true;
            rep(j, m) if (cost(c, j) < cost(edge[j], j)) edge[j] = c;
        }
        while (b >= 0) {
            int a = trace[b], nb = left[a];
            right[b] = a, left[a] = b, b = nb;
        }
    }
    return left;
}
```

7.9 基底変換

7.9.1 高速フーリエ変換

```
using C = complex<R>;
// いざという時は複素数の掛け算を自分で定義する
void fft(vector<C> &a, bool inv) {
    const int n = a.size();
    const R sign = (inv ? -1.0 : 1.0);

    int rj = 0;
    rep(j, 1, n - 1) {
        for (int k = n >> 1; k > (rj ^= k); k >>= 1);
        if (j < rj) swap(a[j], a[rj]);
    }

    for (int m = 1; m < n; m <<= 1) {
        C wn = exp(C(0.0, sign * pi / m)), w = 1.0;
        rep(p, m) {
            for (int s = p; s < n; s += 2 * m) {
                C u = a[s], v = a[s + m] * w;
                a[s] = u + v, a[s + m] = u - v;
            }
            w *= wn;
        }
    }
    if (inv) rep(i, n) a[i] /= n;
}

vector<C> convolution(vector<C> a, vector<C> b) {
    int fft_size = 1;
    while (fft_size < a.size() + b.size()) fft_size <<= 1;

    a.resize(fft_size), fft(a, 0);
    b.resize(fft_size), fft(b, 0);

    vector<C> c(fft_size, 0.0);
    rep(i, fft_size) c[i] = a[i] * b[i];

    fft(c, 1);
    return c;
}
```

7.9.2 高速剰余変換

```
// 2^23 より大きく，primitive root に 3 を持つもの
// const ll mods[] = { 1224736769, 469762049, 167772161,
// 595591169, 645922817, 897581057, 998244353 };
// 1.5 * 10^5 * 10^4 = 150000 * 100
// 5 * 2^23 + 1 = 998244353

void ntt(vector<ll> &a, bool inv, ll mod) {
    const int n = a.size();

    ll base = POW(3LL, (mod - 1) / n, mod);
    if (inv) base = INV(base, mod);

    int rj = 0;
    rep(j, 1, n - 1) {
        for (int k = n >> 1; k > (rj ^= k); k >>= 1);
        if (j < rj) swap(a[j], a[rj]);
    }

    for (int m = 1; m < n; m <<= 1) {
        const ll wn = POW(base, n / 2 / m, mod);
        ll w = 1LL;

        rep(p, m) {
            for (int s = p; s < n; s += 2 * m) {
                ll u = a[s], v = MUL(a[s + m], w, mod);
                a[s] = ADD(u, v, mod), a[s + m] = SUB(u, v, mod);
            }
            w = MUL(w, wn, mod);
        }
    }
```

```
	const ll n_inv = INV(n, mod);
	if (inv) rep(i, n) a[i] = MUL(a[i], n_inv, mod);
}

vector<ll> convolution(vector<ll> a, vector<ll> b, ll mod = 998244353) {
	int ntt_size = 1;
	while (ntt_size < a.size() + b.size()) ntt_size <<= 1;

	a.resize(ntt_size), ntt(a, 0, mod);
	b.resize(ntt_size), ntt(b, 0, mod);

	vector<ll> c(ntt_size, 0LL);
	rep(i, ntt_size) c[i] = MUL(a[i], b[i], mod);

	ntt(c, 1, mod);
	return c;
}
```

7.9.3 高速アダマール変換

```
void fht(vector<int> &a, int l, int r) {
	if (r - l == 1) return;
	const int half = (r - l) / 2;
	const int m = (l + r) / 2;

	fht(a, l, m), fht(a, m, r);
	for (int i = l; i < m; i++) {
		int b[2] = {a[i], a[i + half]};
		a[i] = (b[0] + b[1]) % mod;
		a[i + half] = (b[0] + mod -  b[1]) % mod;
	}
}
```

7.10 ラグランジュ補間

```
// Description: (d+1) 個の点から d 次式を補間
// TimeComplexity: O(N^2)
// Verifyed: AOJ 1328

R Lagrange_interpolation(vector<R> xi, vector<R> fi, R x) {
	const int n = xi.size();
	R f = 0.0;
	rep(i, n) {
		R li = 1.0;
		rep(j, n) {
			if (i == j) continue;
			li *= (x - xi[j]) / (xi[i] - xi[j]);
		}
		f += fi[i] * li;
	}
	return f;
}
```

7.11 公式集

- フェルマーの小定理: 素数 p，任意の整数 x に対し，$x^p \equiv x \pmod p$
- 中国剰余定理: k 個の整数 m_i がどの 2 つも互いに素ならば，任意に与えられる k 個の整数 a_i に対し、$x \equiv a_i \pmod{m_i}$ である x が一意に定まる．
- ポリアの数え上げ定理: すべてのパターンをちょうど同じ回数だけ数え上げ，重複回数で割ることで数え上げが可能
- シンプソン公式: 数値積分の公式 $\frac{b-a}{6}\left[f(a)+4f(\frac{a+b}{2})+f(b)\right]$ 本来は近似値だが，$f(x)$ が二次以下であれば厳密値が得られる．

8 幾何

8.1 注意事項

- sgn 関数の定義はちゃんと写す・istream を使う時は p の代入を忘れない
- 交点を求める前に交差判定が必要 iss(a,b) ill(a,b) == parallel(a,b)
- angle の定義には気をつける
- complex の比較関数は namespace std の中で書く

8.2 ベクトル

```
// Description: ベクトル
// Verifyed: various problem
using namespace placeholders;
using R = long double;
const R EPS = 1e-9L; // [-1000,1000]->EPS=1e-8 [-10000,10000]->EPS=1e-7
inline int sgn(const R& r) {return (r > EPS) - (r < -EPS);}
inline R sq(R x) {return sqrt(max(x, 0.0L));}

const R INF = 1E40L;
const R PI = acos(-1.0L);
using P = complex<R>;
using L = struct {P s, t;};
using VP = vector<P>;
using C = struct {P c; R r;};

#define at(a,i) (a[(i + a.size()) % a.size()])

```

```cpp
auto& operator >> (istream& is, P& p) { R x, y; is >> x >> y, p = P(x, y); return
  is;}
auto& operator << (ostream& os, P& p) { os << real(p) << " " << imag(p); return
  os;}

namespace std {
bool operator <  (const P& a, const P& b) { return sgn(real(a - b)) ? real(a - b) <
  0 : sgn(imag(a - b)) < 0;}
bool operator == (const P& a, const P& b) { return sgn(real(a - b)) == 0 &&
  sgn(imag(a - b)) == 0;}
}

inline R dot(P o, P a, P b) {return real(conj(a - o) * (b - o));}
inline R det(P o, P a, P b) {return imag(conj(a - o) * (b - o));}
inline P vec(L l) {return l.t - l.s;}
auto sdot = bind(sgn, bind(dot, _1, _2, _3));
auto sdet = bind(sgn, bind(det, _1, _2, _3));

//projection verify AOJ CGL_1_A
P proj(L l, P p) { R u = real((p - l.s) / vec(l)); return (1 - u) * l.s + u * l.t;}
}
```

8.3 点集合

8.3.1 凸包

```cpp
// convex_hull Verify AOJ CGL_4_A
VP convex_hull(VP pol) {
  int n = pol.size(), k = 0, t = 1;

  auto cmp_x = [](P a, P b)->bool{
    int sr = sgn(real(a - b)), si = sgn(imag(a - b));
    return sr ? sr < 0 : si < 0;
  };

  sort(begin(pol), end(pol), cmp_x);
  VP res(2 * n);

  auto push = [&](P p)->void{
    while (k > t and sdet(res[k - 1], res[k - 2], p) >= 1) k--;
    res[k++] = p;
  };

  for_each(begin(pol), end(pol), push);
  t = k;
  for_each(rbegin(pol) + 1, rend(pol), push);
  res.resize(k - 1);
  return res;
}
```

8.3.2 最近点対

```cpp
// closest point pair Verify AOJ CGL_5_A
R cpp(VP a, int flag = 1) {
  const int n = a.size(), m = n / 2;
  if (n <= 1) return INF;

  auto cmp_x = [](P a, P b)->bool{
    int sr = sgn(real(a - b)), si = sgn(imag(a - b));
    return sr ? sr < 0 : si < 0;
  };

  if (flag) sort(begin(a), end(a), cmp_x);

  VP b(begin(a), begin(a) + m), c(begin(a) + m, end(a));
  R x = real(a[m]), d = min(cpp(b, 0), cpp(c, 0));

  auto cmp_y = [](P a, P b)->bool{
    int sr = sgn(real(a - b)), si = sgn(imag(a - b));
    return si ? si < 0 : sr < 0;
  };

  sort(begin(a), end(a), cmp_y);
  deque<P> e;

  for (auto &p : a) {
    if (abs(real(p) - x) >= d) continue;

    for (auto &q : e) {
      if (imag(p - q) >= d) break;
      d = min(d, abs(p - q));
    }
    e.push_front(p);
  }
  return d;
}
```

8.3.3 最遠点対

```cpp
// farthest point pair Verify AOJ CGL_4_B
R fpp(VP pol) {
  int n = pol.size(), i = 0, j = 0;
  if (n <= 2) return abs(pol[0] - pol[1]);
```

```
    R res = 0.0;

    auto cmp_x = [](P a, P b)->bool{
        int sr = sgn(real(a - b)), si = sgn(imag(a - b));
        return sr ? sr < 0 : si < 0;
    };

    rep(k, n) {
        if (!cmp_x(pol[i], pol[k]))i = k;
        if (cmp_x(pol[j], pol[k]))j = k;
    }

    int si = i, sj = j;
    while (i != sj || j != si) {
        res = max(res, abs(pol[i] - pol[j]));
        P li = vec(L{at(pol, i), at(pol, i + 1)});
        P lj = vec(L{at(pol, j), at(pol, j + 1)});
        if (sdet(0, li, lj) < 0)
            i = (i + 1) % n;
        else
            j = (j + 1) % n;
    }
    return res;
}
```

8.4 直線と線分

```
// vertical parallel
// verified: AOJ CGL_2_A
bool vertical(L a, L b) {return sdot(0, vec(a), vec(b)) == 0;}
bool parallel(L a, L b) {return sdet(0, vec(a), vec(b)) == 0;}
bool eql(L a, L b) { return parallel(a, b) and sdet(a.s, a.t, b.s) == 0;}

// crossing determination
// verified: AOJ CGL_2_B
bool iss(L a, L b) {
    int sa = sdet(a.s, a.t, b.s) * sdet(a.s, a.t, b.t);
    int sb = sdet(b.s, b.t, a.s) * sdet(b.s, b.t, a.t);
    return max(sa, sb) < 0;
}

// crossing point
// verified: AOJ CGL_2_C
P cross(L a, L b) {
    R u = det(a.s, b.s, b.t) / det(0, vec(a), vec(b));
    return (1 - u) * a.s + u * a.t;
}

// distance
// verified: AOJ CGL_2_D
R dsp(L l, P p) {
    P h = proj(l, p);
    if (sdot(l.s, l.t, p) <= 0) h = l.s;
    if (sdot(l.t, l.s, p) <= 0) h = l.t;
    return abs(p - h);
}

R dss(L a, L b) {
    if(iss(a,b)) return 0;
    return min({dsp(a, b.s), dsp(a, b.t), dsp(b, a.s), dsp(b, a.t)});
}
```

8.5 多角形

```
// Polygon

// area
// verified: AOJ 1100 CGL_3_A
R area(const VP& pol) {
    R sum = 0.0;
    rep(i, pol.size()) sum += det(0, at(pol, i), at(pol, i + 1));
    return abs(sum / 2.0L);
}

// convex_polygon determination
// verified: CGL_3_B
bool is_convex(const VP& pol) {
    rep(i, pol.size()){
        if(sdet(at(pol, i), at(pol, i + 1), at(pol, i + 2)) < 0){
            return false;
        }
    }
    return true;
}

// polygon realation determination  in  2 on 1 out 0   (possible non-convex)
// verified: AOJ CGL_3-C
int in_polygon(const VP& pol, const P& p) {
    int res = 0;
    auto simag = [](const P & p) {return sgn(imag(p));};
    rep(i, pol.size()) {
        P a = at(pol, i), b = at(pol, i + 1);
```

```
    if (sdet(p, a, b) == 0 and sdot(p, a, b) <= 0) return 1;
    bool f = simag(p - a) >= 0, s = simag(p - b) < 0;
    if (simag(b - a)*sdet(a, b, p) == 1 and f == s) res += (2 * f - 1);
  }
  return res ? 2 : 0;
}

// polygon realation determination   (possible non-convex)
// verified: not AOJ 2514
bool in_polygon(const VP& pol, const L& l) {
  VP check = {l.s, l.t};
  rep(i, pol.size()) {
    L edge = {at(pol, i), at(pol, i + 1)};
    if (iss(l, edge)) check.emplace_back(cross(l, edge));
  }

  auto cmp_x = [](P a, P b)->bool{
    int sr = sgn(real(a - b)), si = sgn(imag(a - b));
    return sr ? sr < 0 : si < 0;
  };

  sort(begin(check), end(check), cmp_x);
  rep(i, check.size() - 1) {
    P m = (at(check, i) + at(check, i + 1)) / 2.0L;
    if (in_polygon(pol, m) == false) return false;
  }
  return true;
}

// convex_cut
// verified: AOJ CGL_4_C
VP convex_cut(const VP& pol, const L& l) {
  VP res;
  rep(i, pol.size()) {
    P a = at(pol, i), b = at(pol, i + 1);
    int da = sdet(l.s, l.t, a), db = sdet(l.s, l.t, b);
    if (da >= 0) res.emplace_back(a);
    if (da * db < 0) res.emplace_back(cross({a, b}, l));
  }
  return res;
}
```

8.6 円

```
// Circle // verified: AOJ 1183
enum RCC {OUT = 2, ON_OUT = 1, ISC = 0, ON_IN = -1, IN = -2};
int rcc(C a, C b) {
  R d = abs(a.c - b.c);
  return sgn(d - a.r - b.r) + sgn(d - abs(a.r - b.r));
}

// circle crossing determination
bool icp(C c, P p, int end = 0) {return sgn(abs(p - c.c) - c.r) <= -end;}
bool ics(C c, L s, int end = 0) {
  if (sgn(dsp(s, c.c) - c.r) > end) return false;
  if (icp(c, s.s, end) and icp(c, s.t, end)) return false;
  return true;
}
// common area between circles
R area(C a, C b) {
  int r = rcc(a, b);
  if (r >= ON_OUT) return 0.0L;
  if (r <= ON_IN) return min(norm(a.r), norm(b.r)) * PI;
  R d = abs(b.c - a.c), rc = (norm(d) + norm(a.r) - norm(b.r)) / (2.0 * d);
  R t = acos(rc / a.r), p = acos((d - rc) / b.r);
  return norm(a.r) * t + norm(b.r) * p - d * a.r * sin(t);
}

// cross point between circle and line
// verified: AOJ CGL_7_D
P cir(C c, R t) {return c.c + polar(c.r, t);}
VP cross(C c, L l) {
  P h = proj(l, c.c);
  P e = polar(sq(norm(c.r) - norm(h - c.c)), arg(vec(l)));
  return VP{h - e, h + e};
}

// cross point between circles
// verified: AOJ CGL_7_E
VP cross(C a, C b) {
  P d = b.c - a.c;
  P w = (norm(d) + norm(a.r) - norm(b.r)) / (2.0L * norm(d)) * d;
  return cross(a, {a.c + w, a.c + w + 1il * d});
}

// circle tangent
// verified: AOJ CGL_7_F
L tan(C c, P p) {return L{p, p + 1il * (p - c.c)};}

P helper(C c, P d, R r, P j) {
  P tmp = sq(norm(d) - norm(r)) * j;
  P dir = (r + tmp) / norm(d) * d;
  return c.c + c.r * dir;
```

```
}

VP contact(C c, P p) {
  VP ret;
  P d = p - c.c;
  for (P j : { -1il, 1il}) ret.emplace_back(helper(c, d, c.r, j));
  sort(begin(ret), end(ret));
  ret.erase(unique(begin(ret), end(ret)), end(ret));
  return ret;
}

// circle tangent
// Verified: AOJ CGL_7_G
VP contact(C a, C b) {
  VP ret;
  P d = b.c - a.c;
  for (int s : { -1, 1}) {
    if (rcc(a, b) >= s) {
      for (P j : { -1i, 1i}) {
        R r = a.r + s * b.r;
        ret.emplace_back(helper(a, d, r, j));
      }
    }
  }
  sort(begin(ret), end(ret));
  ret.erase(unique(begin(ret), end(ret)), end(ret));
  return ret;
}

// common area of circle and polygon
// verified: AOJ CGL_7_H
R area_helper(C c, P a, P b) {
  if (icp(c, a) and icp(c, b)) return det(0, a, b) / 2.0l;
  return norm(c.r) * arg(conj(a) * b) / 2.0l;
}

R area(C c, P a, P b) {
  L l = {a, b};

  if (sgn(min({c.r, abs(a), abs(b), abs(b - a)})) == 0) return 0.0;
  if (ics(c, l) == false) return area_helper(c, a, b);

  R res = 0.0; VP ary;
  ary.push_back(a);
  for (auto &p : cross(c, l)) if (sdot(p, a, b) < 0) ary.push_back(p);
  ary.push_back(b);

  rep(i, ary.size() - 1) res += area_helper(c, at(ary, i), at(ary, i + 1));
  return res;
}

R area(C c, VP pol) {
  R res = 0;
  rep(i, pol.size()) {
    P a = at(pol, i) - c.c , b = at(pol, i + 1) - c.c;
    res += area(C{0.0L, c.r}, a, b);
  }
  return res;
}
```

8.7 線分アレンジメント

```
// segments arrangement AOJ 1050
G segment_arrangement(const vector<L> &seg, vector<P> &point){
  int n=seg.size();
  rep(i,n){
    auto &l=seg[i];
    point.emplace_back({l.s,l.t});
    rep(j,i) if(iss(seg[i],seg[j],1)) point.emplace_back(cross(l,seg[j]));
  }

  uniq(point);
  int m=point.size();
  G graph(m);

  for(auto &l:seg){
    vector<int> idx;
    rep(j,m) if(sdot(point[j],l.s,l.t)<0)  idx.emplace_back(j);

    sort(_all(idx),[&](int i,int j){return norm(point[i]-l.s)<norm(point[j]-l.s)});
    rep(j,1,idx.size){
      int a=idx[j-1],b=idx[j];
      add_edge(graph,a,b,abs(point[a]-point[b]));
    }
  }
  return graph;
}
```

8.8 円アレンジメント

```
const int vmax=5010;
struct node{int to;R cost;};
```

```cpp
vector<node> graph[vmax];

// Points not verify
R toRagian(R degree){ return degree*PI/180.0;}
R ang (P p){return arg(p);}
R ang (P bs,P a,P b) {R res=arg((b-bs)/(a-bs));return res<0?res+2*PI:res;}
P rot (P bs,P a,R tht){P tar=a-bs;return bs+polar(abs(tar),arg(tar)+tht);}

const int vmax=5010;
struct node{int to;R cost;};
vector<node> graph[vmax];

inline void add_edge(int f,int t,R c){reg(graph[f],{t,c}),reg(graph[t],{f,c});}

// AOJ 1352

void circle_arrangement(const VC &circle,VP &point){
  VP candiate;
  auto can=[&](P p){
    for(auto &c:circle)if(icp(c,p,1)) return;
    reg(candiate,p);
  };

  auto check1=[&](P p){
    for(auto &c:circle)if(icp(c,p,1)) return false;
    return true;
  };

  auto check2=[&](L s){
    for(auto &c:circle)if(ics(c,s,1)) return false;
    return true;
  };

  for(auto &c1:circle){
    rep(j,4) can(cir(c1,j*PI/2.0));
    for(auto &p:point) for(auto &l:tan(c1,p)) can(proj(l,c1.c));
    for(auto &c2:circle){
      if(rcc(c1,c2)==ISC) for(auto &p:pcc(c1,c2)) can(p);
      for(auto &l:tan(c1,c2)) can(proj(l,c1.c)),can(proj(l,c2.c));
    }
  }

  uniq(candiate),move(_all(candiate),back_inserter(point));
  for(auto &c:circle){
    vector<pair<R,int>> idx;
    rep(i,point.size()){
      if(sgn(norm(c.c-point[i])-norm(c.r))==0)
        reg(idx,{arg(point[i]-c.c),i});
    }
    sort(_all(idx)),reg(idx,{idx[0].first+2*PI,idx[0].second});
    rep(i,1,idx.size()){
      R a1=idx[i-1].first,a2=idx[i].first;
      P mid=cir(c,(a1+a2)/2.0);
      if(check1(mid)) add_edge(idx[i-1].second,idx[i].second,c.r*(a2-a1));
    }
  }
  rep(i,point.size())rep(j,i){
    L l={point[i],point[j]};
    if(check2(l)) add_edge(i,j,abs(l.t-l.s));
  }
}
```

8.9 ボロノイ図

```cpp
VP normalize_polygon(VP pol) {
  rep(i, pol.size()) {
    if (ccw(pol[(i + n - 1) % n], pol[i], pol[(i + 1) % n]) == ON)
      pol.erase(begin(pol) + i--);
  }
  return pol;
}

L bisector(P a, P b) {
  const P mid = (a + b) / P(2, 0);
  return L{mid, mid + (b - a)*P(0, 1)};
}

VP voronoi_cell(VP pol, VP v, int s) {
  rep(i, v.size()) {
    if (i == s) continue;
    pol = convex_cut(pol, bisector(v[s], v[i]));
  }
  return pol;
}
```