-

Team Note of NewTrend

arnold, jupiter, Karuna

Compiled on September 13, 2024

Contents

1 String

1.1 Z, Manacher

```
// S[i - P[i] ... i + P[i]] is palindrome
// manacher(S = "abcbcba") = [ 0, 0, 1, 3, 1, 0, 0 ]
vector<int> manacher(string S) {
  int n = S.size();
  vector<int> a(n);
  for (int i = 0, j = 0; i < n; i++) {
    if (j + a[j] > i) a[i] = min(a[2 * j - i], j + a[j] - i);
    while (i + a[i] < n && i >= a[i] && S[i + a[i]] == S[i -
    a[i]]) ++a[i];
    if (i + a[i] > j + a[j]) j = i;
  }
  for (int i = 0; i < n; i++) --a[i];
  return a;
}
// Z[i] = LCP(S[i...], S)
// get_z(S = "abababacaca") = [ -, 0, 3, 0, 1, 0, 1, 0, 1 ]
vector<int> get_z(string S) {
  int n = S.size();
  vector<int> a(n);
  for (int i = 1, j = 0; i < n; i++) {
    if (j + a[j] > i) a[i] = min(j + a[j] - i, a[i - j]);
    while (i + a[i] < n && S[a[i]] == S[i + a[i]]) ++a[i];
    if (i + a[i] > j + a[j]) j = i;
  }
  return a;
}
```

1.2 Aho Corasick (1858B)

```
// 0-based, O(|S| + 26 * \sum|T_i|)
// v.fail = failure link of node v
// v.suf = longest suffix of node v corresponding to whole
inserted string (if none, root)
// v.chd[c] = c-th children of node v in trie
// v.jmp[c] = c-th next match of node v in automaton
// call init() first, and insert strings using insert(S)
// then call calc() to calculate fail, suf, chd, jump
// AhoCorasick(S, TV) : Find occurences of TV[0], TV[1], ... in
S (ending position)
// AhoCorasick(S = "mississippi", TV = [ "ss", "sis", "ippi",
"pp" ]) = [ 3, 5, 6, 9, 10 ]
struct Node {
  int par, fail, suf;
  vector<int> chd, jmp;
```

```
  Node() {
    par = fail = suf = -1;
    chd = vector<int>(26, -1);
    jmp = vector<int>(26, -1);
  }
};
int root;
vector<Node> NS;
int newNode() { NS.push_back(Node()); return NS.size() - 1; }
void init() {
  NS.clear();
  root = newNode();
  NS[root].par = root;
  NS[root].fail = root;
  NS[root].suf = root;
}
void insert(const string &S) {
  int now = root;
  for (auto c : S) {
    if (NS[now].chd[c - 'a'] == -1) {
      int nxt = NS[now].chd[c - 'a'] = newNode();
      NS[nxt].par = now;
    }
    now = NS[now].chd[c - 'a'];
  }
  NS[now].suf = now;
}
void calc() {
  queue<int> Q;
  Q.push(root);
  while (!Q.empty()) {
    int now = Q.front();
    Q.pop();
    for (int i = 0; i < 26; i++) if (NS[now].chd[i] != -1) {
      int nxt = NS[now].chd[i];
      NS[now].jmp[i] = nxt;
      if (now == root) NS[nxt].fail = root;
      else {
        int p;
        for (p = NS[now].fail; p != root && NS[p].chd[i] == -1;
        p = NS[p].fail);
        if (NS[p].chd[i] == -1) NS[nxt].fail = root;
        else NS[nxt].fail = NS[p].chd[i];
      }
      if (NS[nxt].suf == -1) NS[nxt].suf =
      NS[NS[nxt].fail].suf;
      Q.push(nxt);
    }

    for (int i = 0; i < 26; i++) {
      if (NS[now].chd[i] != -1) NS[now].jmp[i] =
      NS[now].chd[i];
      else {
        if (now == root) NS[now].jmp[i] = now;
        else NS[now].jmp[i] = NS[NS[now].fail].jmp[i];
      }
    }
  }
}
```

```
vector<int> AhoCorasick(string S, vector<string> TV) {
  vector<int> ans;
  for (auto &T : TV) insert(T);
  calc();

  int now = root;
  for (int i = 0; i < S.size(); i++) {
    now = NS[now].jmp[S[i] - 'a'];
    // now is matching node in trie with S[0...i]
    // your code goes here
    if (NS[now].suf != root) ans.push_back(i);
  }
  return ans;
}
```

1.3 Suffix Array (2395B)

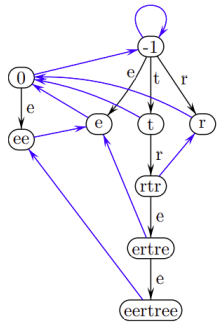
```
// 0-based, O(n log n)
// S   = "abcababc"
// sa  = [ 3, 5, 0, 4, 6, 1, 7, 2 ]
// lcp = [ -, 2, 3, 0, 1, 2, 0, 1 ]
void suffix_array(string S, vector<int> &sa, vector<int> &lcp)
{
  int n = S.size();
  vector<int> r(n), k(n), cnt(n), ord(n);
  sa.resize(n);
  lcp.resize(n);

  iota(sa.begin(), sa.end(), 0);
  sort(sa.begin(), sa.end(), [&](int i, int j) { return S[i] <
  S[j]; });

  r[sa[0]] = 1;
  for (int i = 1; i < n; i++) r[sa[i]] = r[sa[i - 1]] + (S[sa[i
  - 1]] != S[sa[i]]);

  for (int d = 1; d < n; d *= 2) {
    if (r[sa[n - 1]] == n) break;

    fill(cnt.begin(), cnt.end(), 0);
    for (int i = 0; i < n; i++) cnt[i + d < n ? r[i + d] :
    0]++;
    for (int i = 1; i < n; i++) cnt[i] += cnt[i - 1];
    for (int i = 0; i < n; i++) ord[--cnt[i + d < n ? r[i + d]
    : 0]] = i;

    fill(cnt.begin(), cnt.end(), 0);
    for (int i = 0; i < n; i++) cnt[r[i]]++;
    for (int i = 1; i < n; i++) cnt[i] += cnt[i - 1];
    for (int i = n - 1; i >= 0; i--) sa[--cnt[r[ord[i]]]] =
    ord[i];

    k[sa[0]] = 1;
    for (int i = 1; i < n; i++) {
      if (r[sa[i - 1]] != r[sa[i]]) k[sa[i]] = k[sa[i - 1]] +
      1;
      else {
        int p = sa[i - 1] + d < n ? r[sa[i - 1] + d] : 0;
        int q = sa[i] + d < n ? r[sa[i] + d] : 0;
        k[sa[i]] = k[sa[i - 1]] + (p != q);
      }
    }
    r = k;
  }
  for (int i = 0; i < n; i++) --r[i];
  for (int i = 0, p = 0; i < n; i++) if (r[i]) {
    int j = sa[r[i] - 1];
    while ((i + p < n ? S[i + p] : -1) == (j + p < n ? S[j + p]
    : -2)) ++p;
    lcp[r[i]] = p;
    p = max(p - 1, 0);
  }
}

// O(1), call init(S) first
// get_lcp(a, b) = LCP(S[a ...], S[b ...])
// compare(l1, r1, l2, r2) =
//  1 if S[l1 ... r1] > S[l2 ... r2]
//  0 if S[l1 ... r1] = S[l2 ... r2]
// -1 if S[l1 ... r1] < S[l2 ... r2]
// l1 <= r1, l2 <= r2 are NOT necessary
int N;
string S;
vector<int> sa, lcp, r;
vector<vector<int>> sp;
void init(string _S) {
  S = _S; N = S.size();
  r = vector<int>(N);
  suffix_array(S, sa, lcp);
  for (int i = 0; i < N; i++) r[sa[i]] = i;
  sp = vector<vector<int>>(__lg(N) + 1, vector<int>(N)); sp[0]
  = lcp;
  for (int i = 1; i < sp.size(); i++)
    for (int j = 0; j < N; j++) sp[i][j] = min(sp[i - 1][j],
    sp[i - 1][min(N - 1, j + (1 << (i - 1)))]);
}
int get_lcp(int a, int b) {
  if (a == b) return N - a;
  a = r[a]; b = r[b];
  if (a > b) swap(a, b);
  int len = b - a, k = __lg(len);
  return min(sp[k][a + 1], sp[k][b - (1 << k) + 1]);
}
int compare(int l1, int r1, int l2, int r2) {
  r1 = max(r1, l1 - 1);
  r2 = max(r2, l2 - 1);
  int len = min(r1 - l1 + 1, r2 - l2 + 1), x = get_lcp(l1, l2);
  if (x < len) return (S[l1 + x] > S[l2 + x]) - (S[l1 + x] <
  S[l2 + x]);
  else return (r1 - l1 > r2 - l2) - (r1 - l1 < r2 - l2);
}
```

1.4 Suffix Automaton (2205B)

```
// 0-based, O(26 * |S|)
// v.link = suffix link of v
// v.len = length of longest substring corresponding to v
// v.pos = one element of endpos(v)
// v.chd[c] = c-th children node of v in suffix automaton
```

```
// e.x = child node of edge e in suffix tree
// [e.l, e.r] = substring of S corresponding to edge e in
suffix tree
// v.flag = true if v corresponds exactly to suffix (v.pos)
// call init() first, and build suffix automaton by calling
build(S)
// call init() first, and build suffix tree by calling
suffix_tree(S)
// all nodes in nd are used in both suffix automaton and suffix
tree
// nodes in suffix tree CAN have ONE child
const int CH = 26;
struct node {
  int link, len, pos;
  vector<int> chd;
  bool flag;
  node() {
    link = len = pos = 0;
    flag = false;
    chd = vector<int>(CH);
  }
};
vector<node> nd;
void init() { nd.clear(); }
int new_node() {
  nd.push_back(node());
  return (int)nd.size() - 1;
}
void build(string S) {
  new_node();
  nd[0].link = -1;
  int prv = 0;
  for (int i = 0; i < S.size(); i++) {
    int c = S[i] - 'a';
    int cur = new_node();
    nd[cur].len = nd[prv].len + 1;
    nd[cur].pos = i;
    nd[cur].flag = true;
    for (; prv != -1 && !nd[prv].chd[c]; prv = nd[prv].link)
    nd[prv].chd[c] = cur;
    if (prv != -1) {
      int x = nd[prv].chd[c];
      if (nd[x].len == nd[prv].len + 1) nd[cur].link = x;
      else {
        int y = new_node();
        nd[y].len = nd[prv].len + 1;
        nd[y].pos = i;
        nd[y].link = nd[x].link;
        nd[y].chd = nd[x].chd;
        nd[x].link = nd[cur].link = y;
        for (; prv != -1 && nd[prv].chd[c] == x; prv =
        nd[prv].link) nd[prv].chd[c] = y;
      }
    }
    prv = cur;
  }
}
struct edge { int x, l, r; };
vector<vector<edge>> g;
void suffix_tree(string S) {
  reverse(S.begin(), S.end());
  build(S);
  reverse(S.begin(), S.end());
  int n = nd.size();
  g.clear(); g.resize(n);
  for (int v = 0; v < n; v++) nd[v].pos = (int)S.size() - 1 -
  nd[v].pos;
  for (int v = 1; v < n; v++) {
    int x = nd[v].link;
    int l = nd[v].pos + nd[x].len;
    int r = nd[v].pos + nd[v].len - 1;
    g[x].push_back({v, l, r});
  }
  for (int v = 0; v < n; v++) sort(g[v].begin(), g[v].end(),
  [&](edge e, edge f) { return S[e.l] < S[f.l]; });
}
void dfs(int v, vector<int> &sa) {
  if (nd[v].flag) sa.push_back(nd[v].pos);
  for (auto [x, l, r] : g[v]) dfs(x, sa);
}
void suffix_array(string S, vector<int> &sa, vector<int> &lcp)
{
  suffix_tree(S);
  dfs(0, sa);
  int n = S.size();
  lcp.resize(n);
  vector<int> r(n);
  for (int i = 0; i < n; i++) r[sa[i]] = i;
  for (int i = 0, p = 0; i < n; i++) if (r[i]) {
    int j = sa[r[i] - 1];
    while ((i + p < n ? S[i + p] : -1) == (j + p < n ? S[j + p]
    : -2)) ++p;
    lcp[r[i]] = p;
    p = max(p - 1, 0);
  }
}
```

1.5 eertree (1015B)




```
// 0-based, O(26 N)
// nd[x].pos - nd[x].len + 1 ~ nd[x].pos is palindrome for node
x
// nd[S].chd[c] represents the string cSc
// nd[S].link represents the longest palindromic suffix of S
// suf[i] represents the longest palindromic suffix of S[0 ...
i]
struct node {
  int link, len, par, pos;
  vector<int> chd;
  node() {
    link = len = par = pos = 0;
    chd = vector<int>(26, 1);
  }
};
vector<node> nd;
vector<int> suf;
int new_node() { nd.push_back(node()); return (int)nd.size() -
1; }
void init() { nd.clear(); suf.clear(); }
void build(string S) {
  new_node();  // node 0 represents length -1 string
  new_node();  // node 1 represents length 0 string
  nd[0].link = 0; nd[0].len = -1;
  nd[1].link = 0; nd[1].len = 0;
  for (int i = 0, x = 0; i < S.size(); i++) {
    while (x && (i == nd[x].len || S[i] != S[i - nd[x].len -
    1])) x = nd[x].link;
    int c = S[i] - 'a';
    if (nd[x].chd[c] == 1) {
      int r = new_node();
      nd[r].len = nd[x].len + 2;
      int v = nd[x].link;
      while (v && (i == nd[v].len || S[i] != S[i - nd[v].len -
      1])) v = nd[v].link;
      nd[r].link = nd[v].chd[c];
      nd[x].chd[c] = r;
      nd[r].par = x;
      nd[r].pos = i;
    }
    x = nd[x].chd[c];
    suf.push_back(x);
  }
}
```

1.6 duval (410B)

```
// 0-based, O(N)
// Get length of lexicographically smallest suffix for all
prefix
// lyndon(S = "ababca") = [1, 2, 1, 2, 5, 1]
vector<int> lyndon(string S) {
  int N = S.size();
  vector<int> dp(N);
  for (int i = 0; i < N;) {
    dp[i] = 1;
    int j = i + 1, k = i;
    for (; j < N; j++) {
      if (S[j] == S[k]) {
        dp[j] = dp[k];
        k++;
      }
      else if (S[j] > S[k]) {
        dp[j] = j - i + 1;
        k = i;
```

```
      }
      else break;
    }
    for (; i <= k; i += j - k);
  }
  return dp;
}
```

1.7 Bitset LCS (1112B)

```
// O(NM/64)
#define get(arr, x) (((arr[x >> 6] >> (x & 63)) & 1) == 1)
#define set(arr, x) (arr[x >> 6] |= 1llu << (x & 63))
using ull = unsigned long long;
// return lcs
int lcs(string A, string B) {
  int N = A.size(), M = B.size();
  int sz = (M >> 6) + 1;
  vector<ull> S[256];
  for(int c = 0; c < 256; c++) S[c].resize(sz);
  for(int j = 0; j < M; j++) set(S[B[j]], j);
  vector<ull> row(sz);
  for(int j = 0; j < M; j++) if(A[0] == B[j]) { set(row, j);
  break; }
  for(int i = 1; i < N; i++) {
    ull shl_carry = 1;
    ull minus_carry = 0;
    for(int k = 0; k < sz; k++) {
      // Compute k-th block of x == S[A[i]] OR D[i-1]
      ull x_k = S[A[i]][k] | row[k];
      // Compute k-th block of "(D[i-1] << 1) | 1"
      ull term_1 = (row[k] << 1) | shl_carry;
      shl_carry = row[k] >> 63;
      // Compute k-th block of "x - ((D[i-1] << 1) | 1)"
      auto sub_carry = [](ull &x, ull y){
        ull tmp = x;
        return (x = tmp - y) > tmp;
      };
      ull term_2 = x_k;
      minus_carry = sub_carry(term_2, minus_carry);
      minus_carry += sub_carry(term_2, term_1);
      row[k] = x_k & (x_k ^ term_2);
    }
    row[M >> 6] &= (1llu << (M & 63)) - 1;
  }
  int ret = 0;
  for(int j = 0; j < M; j++) if(get(row, j)) ret += 1;
  return ret;
}
```

2 Optimization

2.1 Dinic (1579B)

$\min(\mathcal{O}(V^2E), \mathcal{O}(Ef))$. If all vertices except the source and sink have unique indegree or outdegree with unit capacity, $\mathcal{O}(E\sqrt{V})$. If all edges are unit, $\min(\mathcal{O}(E\sqrt{E}), \mathcal{O}(EV^{2/3}))$.

```
// 0-based, min( O(V^2 E), O(EF) ), directed graph
// call init() first, add_edge(u, v, c), then flow(source,
sink)
struct Dinic {
  typedef long long T;
  const static int SV = 1200;
  const T INF = numeric_limits<T>::max();

  struct Edge { int v; T c; int r; };
  int N, src, snk;
  vector<Edge> gph[SV];
  int lvl[SV], pos[SV];
  T lim;
  void init(int _N) {
    N = _N;
    for (int i = 0; i < N; i++) gph[i] = vector<Edge>();
  }
  void add_edge(int u, int v, T c, bool dir = true) {
    // directed edge : dir = true, undirected edge : dir =
    false
    gph[u].push_back({v, c, gph[v].size()});
    gph[v].push_back({u, dir ? 0 : c, gph[u].size() - 1});
  }
  bool bfs() {
    for (int i = 0; i < N; i++) lvl[i] = 0;
    queue<int> Q;
    Q.push(src);
    lvl[src] = 1;
    while (!Q.empty()) {
      int now = Q.front(); Q.pop();
      for (auto [nxt, c, r] : gph[now]) {
        if (lvl[nxt] || c < lim) continue;
        Q.push(nxt);
        lvl[nxt] = lvl[now] + 1;
      }
    }
    return lvl[snk];
  }
  T dfs(int now, T flow) {
    if (now == snk) return flow;
    for (; pos[now] < gph[now].size(); pos[now]++) {
      auto &[nxt, c, r] = gph[now][pos[now]];
      if (lvl[nxt] != lvl[now] + 1 || !c) continue;
      T f = dfs(nxt, min(flow, c));
      if (f) {
        c -= f;
        gph[nxt][r].c += f;
        return f;
      }
    }
    return 0;
  }
  T flow(int _src, int _snk) {
    src = _src; snk = _snk;
    T ans = 0;
    for (lim = INF; lim > 0; lim >>= 1) {
      while (bfs()) {
        for (int i = 0; i < N; i++) pos[i] = 0;
        while (1) {
          T t = dfs(src, INF);
          if (!t) break;
          ans += t;
        }
      }
    }
    return ans;
  }
};
```

2.2 HLPP (1771B)

```
// 0-based, O(V^2 \sqrt{E}), directed graph
// call init() first, add_edge(u, v, c), then flow(source,
sink)
struct HLPP {
  typedef long long T;
  const static int SV = 1200;

  struct edge { int y; T c; int r; };
  int n;
  vector<edge> gph[SV];
  T ex[SV];
  vector<int> act[SV * 2], st[SV];
  int h[SV], mxh;
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) {
      ex[i] = h[i] = 0;
      st[i].clear();
      gph[i].clear();
    }
    for (int i = 0; i < n * 2; i++) act[i].clear();
    mxh = 0;
  }
  void add_edge(int x, int y, T c, bool dir = true) {
    // directed edge : dir = true, undirected edge : dir =
    false
    gph[x].push_back({y, c, (int)gph[y].size()});
    gph[y].push_back({x, dir ? 0 : c, (int)gph[x].size() - 1});
  }
  void push(int v, int e) {
    auto &w = gph[v][e];
    T t = min(ex[v], w.c);
    if (!t || h[w.y] >= h[v]) return;
    if (!ex[w.y]) act[h[w.y]].push_back(w.y);
    w.c -= t;
    gph[w.y][w.r].c += t;
    ex[v] -= t;
    ex[w.y] += t;
    if (w.c) st[v].push_back(e);
  }
  void relabel(int v) {
    ++h[v];
    act[h[v]].push_back(v);
    mxh = max(mxh, h[v]);
    st[v].clear();
    for (int i = 0; i < (int)gph[v].size(); ++i) {
      if (gph[v][i].c && h[gph[v][i].y] < h[v]) {
        st[v].push_back(i);
      }
    }
  }
  T flow(int s, int t) {
    h[s] = n;
```

```
    for (auto &v : gph[s]) {
      if (v.c && !ex[v.y]) act[0].push_back(v.y);
      ex[s] -= v.c;
      ex[v.y] += v.c;
      gph[v.y][v.r].c += v.c;
      v.c = 0;
    }
    while (mxh >= 0) {
      if (act[mxh].empty()) {
        --mxh;
        continue;
      }
      int x = act[mxh].back();
      act[mxh].pop_back();
      if (x == t) continue;
      while (ex[x] > 0 && st[x].size()) {
        int y = st[x].back();
        st[x].pop_back();
        push(x, y);
      }
      if (ex[x] > 0) relabel(x);
    }
    return ex[t];
  }
};
```

2.3 Hopcroft Karp (1182B)

```
// 1-based, O(E \sqrt{V}), unweighted bipartite graph
// call init() first, then call find_matching()
// if left vertex x is not included in a matching, a[x] = 0.
// Otherwise a[x] stores the vertex x is matched with
// if right vertex x is not included in a matching, b[x] = 0.
// Otherwise b[x] stores the vertex x is matched with
const int SZ = 101010;
int n, m, a[SZ], b[SZ];
vector<int> gph[SZ];
void init(int _n, int _m) {
  n = _n;
  m = _m;
  for (int i = 1; i <= n; i++) gph[i].clear();
  fill(a, a + 1 + n, 0);
  fill(b, b + 1 + m, 0);
}
void add_edge(int u, int v) {
  gph[u].push_back(v);  // left vertex u -> right vertex v
}
int ds[SZ], sp[SZ];
void bfs() {
  fill(ds, ds + 1 + n, 0);
  queue<int> Q;
  for (int i = 1; i <= n; i++) {
    if (!a[i]) {
      ds[i] = 0;
      Q.push(i);
    }
  }
  while (!Q.empty()) {
    int v = Q.front();
    Q.pop();
    for (int x : gph[v]) {
      if (b[x] && ds[b[x]] == 0) {
        ds[b[x]] = ds[v] + 1;
        Q.push(b[x]);
      }
    }
  }
}
bool match(int v) {
  for (; sp[v] < gph[v].size(); sp[v]++) {
    int x = gph[v][sp[v]];
    if (!b[x] || (ds[b[x]] == ds[v] + 1 && match(b[x]))) {
      b[x] = v; a[v] = x;
      return true;
    }
  }
  return false;
}
int find_matching() {
  int ret = 0;
  while (true) {
    fill(sp, sp + 1 + n, 0);
    bfs();
    int cur = 0;
    for (int i = 1; i <= n; i++) if (!a[i]) cur += match(i);
    if (cur > 0) ret += cur;
    else break;
  }
  return ret;
}
```

2.4 MCMF (1780B)

```
// 0-based, O(EF), directed graph
// call init() first, add_edge(u, v, c(capcity), w(cost)), then
flow(source, sink)
// returns (max flow, min cost when max flow)
struct MCMF {
  typedef long long T;
  const static int SV = 400;
  const T INF = numeric_limits<T>::max();

  struct Edge { int v; T c, w; int r; };
  int N, src, snk;
  vector<Edge> gph[SV];
  T dist[SV];
  int par[SV];
  bool inq[SV];

  void init(int _N) {
    N = _N;
    for (int i = 0; i < N; i++) gph[i].clear();
  }
  void add_edge(int u, int v, T c, T w) {
    gph[u].push_back({v, c, w, gph[v].size()});
    gph[v].push_back({u, 0, -w, gph[u].size() - 1});
  }
  bool shortest_path() {
    fill(dist, dist + N, INF);
    fill(par, par + N, -1);
    fill(inq, inq + N, false);
    deque<int> Q;
    dist[src] = 0;
    inq[src] = true;
    Q.push_back(src);
    int cnt = 0;
    while (Q.size()) {
      int x = Q.front();
      Q.pop_front();
      inq[x] = false;
      for (auto [y, c, w, r] : gph[x]) {
        if (c == 0) continue;
        if (dist[y] > dist[x] + w) {
          dist[y] = max(dist[x] + w, -INF);
          par[y] = r;
          if (!inq[y]) {
            inq[y] = true;
            if (Q.size() && dist[Q[0]] > dist[y])
            Q.push_front(y);
            else Q.push_back(y);
          }
        }
      }
    }
    return dist[snk] != INF;
  }
  pair<T, T> flow(int _src, int _snk) {
    src = _src;
    snk = _snk;
    T cost = 0, flow = 0;
    while (shortest_path()) {
      T val = INF;
      for (int x = snk; x != src;) {
        auto [y, c, w, r] = gph[x][par[x]];
        val = min(val, gph[y][r].c);
        x = y;
      }
      for (int x = snk; x != src;) {
        auto [y, c, w, r] = gph[x][par[x]];
        gph[y][r].c -= val;
        gph[x][par[x]].c += val;
        x = y;
      }
      cost += dist[snk] * val;
      flow += val;
    }
    return {flow, cost};
  }
};
```

2.5 Hungarian (1291B)

```
// 1-based, O(V^3), weighted bipartite graph
// solves min cost matching of a perfect bipartite graph
// call init() first, add_edge(u, v, w), then call
min_cost_matching()
// edges can be negative -> for max cost just negate all costs
then negate the answer
// multiple edges are NOT allowed
// left vertex v is matched with right vertex a[v], a[v] = 0 if
not matched
```

```
// right vertex v is matched with left vertex b[v], b[v] = 0 if
not matched
typedef long long T;
const int SZ = 505;
int n, a[SZ], b[SZ], m[SZ], p[SZ], vis[SZ];
T w[SZ][SZ], x[SZ], y[SZ];
void init(int _n) {
  n = _n;
  // it must hold that x[i] + y[j] <= w[i][j] initially
  for (int i = 1; i <= n; i++) {
    x[i] = y[i] = -1e9;
    a[i] = b[i] = 0;
    for (int j = 1; j <= n; j++) w[i][j] = 0;
  }
}
void add_edge(int u, int v, T c) { w[u][v] = c; }
T f(int i, int j) { return w[i][j] - x[i] - y[j]; }
int dfs(int v) {
  vis[v] = 1;
  for (int i = 1; i <= n; i++) if (m[i] == 0 || f(m[i], i) >
  f(v, i)) m[i] = v;
  for (int i = 1; i <= n; i++) if (f(v, i) == 0 && !p[i]) {
    p[i] = v;
    if (int r = b[i] ? dfs(b[i]) : i; r) return r;
  }
  return 0;
}
void go(int v) {
  for (int i = 1; i <= n; i++) m[i] = p[i] = vis[i] = 0;
  int r = dfs(v);
  while (!r) {
    for (int i = 1; i <= n; i++) if (!p[i]) {
      if (r == 0 || f(m[r], r) > f(m[i], i)) r = i;
    }
    T a = f(m[r], r);
    for (int i = 1; i <= n; i++) {
      if (vis[i]) x[i] += a;
      if (p[i]) y[i] -= a;
    }
    p[r] = m[r];
    r = b[r] ? dfs(b[r]) : r;
  }
  while (r) swap(r, a[b[r] = p[r]]);
}
T min_cost_matching() {
  T ret = 0;
  for (int i = 1; i <= n; i++) go(i);
  for (int i = 1; i <= n; i++) ret += x[i] + y[i];
  return ret;
}
```

2.6 Blossom (1547B)

```
// 1-based, O(V^3), but practically O(VE), undirected graph
// call init() first, add_edge(u, v), then call
general_matching()
// match[v] = 0 if v is not matched
// otherwise v is matched with match[v]
const int SV = 2020;
int n, t;
int match[SV], par[SV], vis[SV], org[SV], aux[SV];
vector<int> gph[SV], S;
void init(int _n) {
  n = _n; t = 0;
  for (int i = 1; i <= n; i++) {
    match[i] = par[i] = vis[i] = org[i] = aux[i] = 0;
    gph[i].clear();
  }
}
void add_edge(int u, int v) {
  gph[u].push_back(v);
  gph[v].push_back(u);
}
int lca(int x, int y) {
  ++t;
  while (true) {
    if (x) {
      if (aux[x] == t) return x;
      aux[x] = t;
      x = org[par[match[x]]];
    }
    swap(x, y);
  }
}
void blossom(int x, int y, int z) {
  while (org[x] != z) {
    par[x] = y; y = match[x];
    if (vis[y] == 1) { vis[y] = 0; S.push_back(y); }
    org[x] = org[y] = z;
    x = par[y];
  }
}
void augment(int x) {
  while (x) {
    int px = par[x], nx = match[px];
    match[x] = px; match[px] = x;
    x = nx;
  }
}
bool bfs(int s) {
  for (int i = 1; i <= n; i++) vis[i] = -1, org[i] = i;
  S.clear(); S.push_back(s);
  vis[s] = 0;
  while (!S.empty()) {
    int v = S.back(); S.pop_back();
    for (int x : gph[v]) {
      if (vis[x] == -1) {
        vis[x] = 1; par[x] = v;
        if (!match[x]) { augment(x); return true; }
        vis[match[x]] = 0;
        S.push_back(match[x]);
      }
      else if (vis[x] == 0 && org[x] != org[v]) {
        int z = lca(org[x], org[v]);
        blossom(v, x, z); blossom(x, v, z);
      }
    }
  }
  return false;
}
int general_matching() {
  int cnt = 0;
  for (int v = 1; v <= n; v++) if (!match[v]) cnt += bfs(v);
  return cnt;
}
```

2.7 General Weighted Matching (6120B)

```
// 1-based, O(V^3) (but fast in practice), undirected graph
// call init() first, add_edge(u, v, w), then call solve()
// returns (weight of maximum matching, size of maximum
matching)
// if vertex x is not included in a matching, match[x] = 0.
// Otherwise match[x] stores the vertex x is matched with
typedef long long T;
static const T INF = numeric_limits<T>::max();
static const int SV = 514;
struct edge {
  int u, v; T w;
  edge() {}
  edge(int ui, int vi, T wi) : u(ui), v(vi), w(wi) {}
};
int n, n_x;
edge g[SV * 2][SV * 2];
T lab[SV * 2];
int match[SV * 2], slack[SV * 2], st[SV * 2], pa[SV * 2];
int flo_from[SV * 2][SV + 1], S[SV * 2], vis[SV * 2];
vector<int> flo[SV * 2];
queue<int> q;
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);
}
void add_edge(int ui, int vi, T wi) { g[ui][vi].w = g[vi][ui].w
= wi; }
T e_delta(const edge &e) { return lab[e.u] + lab[e.v] -
g[e.u][e.v].w * 2; }
void update_slack(int u, int x) { if (!slack[x] ||
e_delta(g[u][x]) < e_delta(g[slack[x]][x])) slack[x] = u; }
void set_slack(int x) {
  slack[x] = 0;
  for (int u = 1; u <= n; ++u) if (g[u][x].w > 0 && st[u] != x
  && S[st[u]] == 0) update_slack(u, x);
}
void q_push(int x) {
  if (x <= n) q.push(x);
  else for (size_t i = 0; i < flo[x].size(); i++)
  q_push(flo[x][i]);
}
void set_st(int x, int b) {
  st[x] = b;
  if (x > n) for (size_t i = 0; i < flo[x].size(); ++i)
  set_st(flo[x][i], b);
}
int get_pr(int b, int xr) {
  int pr = find(flo[b].begin(), flo[b].end(), xr) -
  flo[b].begin();
  if (pr % 2 == 1) {
```

```
    reverse(flo[b].begin() + 1, flo[b].end());
    return (int)flo[b].size() - pr;
  }
  else return pr;
}
void set_match(int u, int v) {
  match[u] = g[u][v].v;
  if (u <= n) return;
  edge e = g[u][v];
  int xr = flo_from[u][e.u], pr = get_pr(u, xr);
  for (int i = 0; i < pr; ++i) set_match(flo[u][i], flo[u][i ^
  1]);
  set_match(xr, v);
  rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].end());
}
void augment(int u, int v) {
  while (1) {
    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;
  while (1) {
    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]);
      }
    }
    T 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<T, int> solve() {
  memset(match + 1, 0, sizeof(int) * n);
  n_x = n;
  int n_matches = 0;
  T tot_weight = 0;
  for (int u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
  T 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);
}
```

2.8 Min Cost Circulation with Cost Scaling (3447B)

```
// Include any max flow algorithm here

// 0-based
// call init() first, add_edge(u, v, l, r), then flow(src, snk)
// gets maximum flow for given LR-flow graph
// returns true if feasible, false if infeasible
// for circulation, call flow(-1, -1)
// for maximum LR flow, run mf.flow(src, snk) again
// for minimum LR flow, binary search with INF
struct Circulation {
  typedef long long T;
  const T INF = numeric_limits<T>::max() / 2;
```

```
  int n;
  T lbsum = 0;
  HLPP mf; // any max flow algorithm
  // dummy src = n, dummy snk = n+1
  void init(int k) {
    n = k;
    mf.init(n + 2);
    lbsum = 0;
  }
  void add_edge(int s, int e, T l, T r) {
    mf.add_edge(s, e, r - l);
    if (l > 0) {
      mf.add_edge(n, e, l);
      mf.add_edge(s, n + 1, l);
      lbsum += l;
    } else {
      mf.add_edge(n, s, -l);
      mf.add_edge(e, n + 1, -l);
      lbsum += -l;
    }
  }
  bool flow(int src, int snk) {
    // to reduce as maxflow with lower bounds, in circulation
    problem skip this line
    // mf.add_edge(snk, src, INF);
    return lbsum == mf.flow(n, n + 1);
    // to get maximum LR flow, run mf.flow(src, snk) again
  }
};

// 0-based, O(V^3 log(VC)), directed graph
// call init() first, add_edge(u, v, l, r, cost), then solve()
// gets minimum cost circulation in given graph
// returns {false, 0} if infeasible
// otherwise, returns {true, ans}
// solution flow in edge (v, p) = gph[v][p].cap - gph[v][p].rem
struct MinCostCirculation {
  typedef long long T;
  const int SCALE = 3;  // scale by 1/(1 << SCALE)
  const T INF = numeric_limits<T>::max() / 2;
  struct EdgeStack {
    int s, e;
    T l, r, cost;
  };
  struct Edge {
    int pos, rev;
    T rem, cap, cost;
  };
  int n;
  vector<EdgeStack> estk;
  Circulation circ;
  vector<vector<Edge>> gph;
  vector<T> p;
  void init(int k) {
    n = k;
    circ.init(n);
    gph.clear(); gph.resize(n);
    p.clear(); p.resize(n);
  }
  void add_edge(int s, int e, T l, T r, T cost) {
    estk.push_back({s, e, l, r, cost});
  }
  pair<bool, T> solve() {
    for (auto &i : estk) if (i.s != i.e) circ.add_edge(i.s,
    i.e, i.l, i.r);
    if (!circ.flow(-1, -1)) return make_pair(false, T(0));
    vector<int> ptr(n);
    T eps = 0;
    for (auto &i : estk) {
      T curFlow;
      if (i.s != i.e) curFlow = i.r -
      circ.mf.gph[i.s][ptr[i.s]].c;
      else curFlow = i.r;
      int srev = gph[i.e].size();
      int erev = gph[i.s].size();
      if (i.s == i.e) srev++;
      gph[i.s].push_back({i.e, srev, i.r - curFlow, i.r, i.cost
      * (n + 1)});
      gph[i.e].push_back({i.s, erev, -i.l + curFlow, -i.l,
      -i.cost * (n + 1)});
      eps = max(eps, abs(i.cost) * (n + 1));
      if (i.s != i.e) {
        ptr[i.s] += 2;
        ptr[i.e] += 2;
      }
    }
    while (eps > 1) {
      eps = max(T(1), eps >> SCALE);
      auto cost = [&](Edge &e, int s, int t) { return e.cost +
      p[s] - p[t]; };
      vector<T> excess(n);
      queue<int> Q;
      auto push = [&](Edge &e, int src, T flow) {
        e.rem -= flow;
        gph[e.pos][e.rev].rem += flow;
        excess[src] -= flow;
        excess[e.pos] += flow;
        if (excess[e.pos] <= flow && excess[e.pos] > 0)
        Q.push(e.pos);
      };
      auto relabel = [&](int v) {
        p[v] = -INF;
        for (auto &e : gph[v]) {
          if (e.rem > 0) {
            p[v] = max(p[v], p[e.pos] - e.cost - eps);
          }
        }
      };
      for (int i = 0; i < n; i++) {
        for (auto &j : gph[i]) {
          if (j.rem > 0 && cost(j, i, j.pos) < 0) {
            push(j, i, j.rem);
          }
        }
      }
      while (!Q.empty()) {
        int x = Q.front();
        Q.pop();
        while (excess[x] > 0) {
          for (auto &e : gph[x]) {
            if (e.rem > 0 && cost(e, x, e.pos) < 0) {
              push(e, x, min(e.rem, excess[x]));
              if (excess[x] == 0) break;
            }
          }
          if (excess[x] == 0) break;
          relabel(x);
        }
      }
    }
    T ans = 0;
    for (int i = 0; i < n; i++) {
      for (auto &j : gph[i]) {
        j.cost /= (n + 1);
        ans += j.cost * (j.cap - j.rem);
      }
    }
    return make_pair(true, ans / 2);
  }
};
```

2.9 Global Min Cut (1557B)

```
// 0-based, O(V^3), undirected graph
// call init() first, add_edge(u, v, w), then call solve()
// gets global min cut of given graph
// cut[v] = {0, 1} which side the vertex belongs to
const int SV = 505;
typedef int T;
const T INF = numeric_limits<T>::max();
int n;
T g[SV][SV];
vector<int> cut;
void init(int _n) {
  n = _n;
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++)
  g[i][j] = 0;
}
void add_edge(int u, int v, T w) {
  if (u == v) return;
  g[u][v] += w;
  g[v][u] += w;
}
pair<T, pii> stMinCut(vector<int> &active) {
  vector<T> key(n);
  vector<int> v(n);
  int s = -1, t = -1;
  for (int i = 0; i < active.size(); i++) {
    T maxv = -1;
    int cur = -1;
    for (auto j : active) if (v[j] == 0 && maxv < key[j]) {
      maxv = key[j];
      cur = j;
    }
    t = s; s = cur;
    v[cur] = 1;
    for (auto j : active) key[j] += g[cur][j];
```

```
  }
  return {key[s], {s, t}};
}
T solve() {
  T res = INF;
  vector<vector<int>> grps;
  vector<int> active;
  cut.resize(n);
  for (int i = 0; i < n; i++) grps.push_back({1, i});
  for (int i = 0; i < n; i++) active.push_back(i);
  while (active.size() >= 2) {
    auto cur = stMinCut(active);
    if (cur.ff < res) {
      res = cur.ff;
      fill(cut.begin(), cut.end(), 0);
      for (auto v : grps[cur.ss.ff]) cut[v] = 1;
    }
    int s = cur.ss.ff, t = cur.ss.ss;
    if (grps[s].size() < grps[t].size()) swap(s, t);

    active.erase(find(active.begin(), active.end(), t));
    grps[s].insert(grps[s].end(), grps[t].begin(),
    grps[t].end());
    for (int i = 0; i < n; i++) { g[i][s] += g[i][t]; g[i][t] =
    0; }
    for (int i = 0; i < n; i++) { g[s][i] += g[t][i]; g[t][i] =
    0; }
    g[s][s] = 0;
  }
  return res;
}
```

2.10 Gomory-Hu Tree (802B)

```
// Include any max flow algorithm here

// 1-based
// MUST call init() before calling solve();
// vector<pii> par = solve();
// par represents 1-rooted tree. if i > 1, par[i] = (parent,
cost)
// i-j cut in original graph = i-j cut in gomory-hu tree
// time complexity: O(V * T(maxflow))
const int SZ = 101010;
typedef long long T;
struct edge { int u, v; T w; };
int n;
vector<edge> edges;
bool vis[SZ];
Dinic mf;
void init(int _n) { n = _n; edges.clear(); }
void add_edge(int u, int v, T w) { edges.push_back({u, v, w});
}
void dfs(int v) {
  if (vis[v]) return;
  vis[v] = true;
  for (auto &e : mf.gph[v]) if (e.c > 0) dfs(e.v);
}
vector<pair<int, T>> solve() {
  vector<pair<int, T>> ret(n + 1, {1, 0});
  for (int i = 2; i <= n; i++) {
    mf.init(n + 1);
    for (auto [u, v, w] : edges) mf.add_edge(u, v, w, false);
    ret[i].ss = mf.flow(i, ret[i].ff);
    for (int j = 1; j <= n; j++) vis[j] = 0;
    dfs(i);
    for (int j = i + 1; j <= n; j++) {
      if (ret[j].ff == ret[i].ff && vis[j]) {
        ret[j].ff = i;
      }
    }
  }
  return ret;
}
```

2.11 Simplex (2351B)

```
// Two-phase simplex algorithm for solving linear programs of
the form
//     maximize     c^T x
//     subject to   Ax <= b
//                   x >= 0
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be
stored
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
// To use this code, create an LPSolver object with A, b, and c
as
// arguments.  Then, call Solve(x).
// Time Complexity : exponential, but practically O(n^2m^2).
typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const double EPS = 1e-9;
struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;
  LPSolver(const VVD& A, const VD& b, const VD& c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n +
    2)) {
        assert(A.size() == b.size() && A[0].size() ==
        c.size());
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++)
    D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1;
    D[i][n + 1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m + 1][n] = 1;
  }
  void pivot(int r, int s) {
    double inv = 1.0 / D[r][s];
    for (int i = 0; i < m + 2; i++) if (i != r)
      for (int j = 0; j < n + 2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] * inv;
    for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
    for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *=
    -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }
  bool simplex(int phase) {
    int x = phase == 1 ? m + 1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s]
        && N[j] < N[s]) s = j;
      }
      if (D[x][s] > -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] /
        D[r][s] ||
          (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) &&
          B[i] < B[r]) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }
  double solve(VD& x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1])
    r = i;
    if (D[r][n + 1] < -EPS) {
      pivot(r, n);
      if (!simplex(1) || D[m + 1][n + 1] < -EPS)
        return -numeric_limits<double>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] ==
          D[i][s] && N[j] < N[s]) s = j;
        pivot(i, s);
      }
    }
    if (!simplex(2))
      return numeric_limits<double>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n
    + 1];
    return D[m][n + 1];
  }
};
```

3 Graph

3.1 BCC (1694B)

```


// 1-based, O(V + E), undirected graph
// call init() first, add_edge(u, v, i), then call get_bcc()
// bcnt : number of bcc
// bcc[c] : vertices of color c
// color[v] : colors of vertex v
// ecol[e] : color of edge e
```

```cpp
// connect (v, c + N) for block-cut tree
const int SV = 5e5 + 10;
const int SE = 5e5 + 10;
int N, M;
vector<pii> adj[SV];
int dfn[SV], low[SV], dcnt;
bool vis[SV];
int bcnt, ecol[SE];
vector<int> color[SV], bcc[SE];
void init(int _N, int _M) {
  N = _N; M = _M;
  for (int i = 1; i <= N; i++) {
    dfn[i] = low[i] = vis[i] = 0;
    adj[i] = vector<pii>();
    color[i] = vector<int>();
  }
  for (int i = 1; i <= M; i++) {
    ecol[i] = 0;
    bcc[i] = vector<int>();
  }
  dcnt = 0; bcnt = 0;
}
void add_edge(int u, int v, int i) {
  // i is edge number of edge (u, v)
  adj[u].push_back({v, i});
  adj[v].push_back({u, i});
}
void dfs(int now, int bef) {
  dfn[now] = low[now] = ++dcnt;
  for (auto [nxt, p] : adj[now]) {
    if (nxt == bef) continue;
    if (dfn[nxt]) low[now] = min(low[now], dfn[nxt]);
    else {
      dfs(nxt, now);
      low[now] = min(low[now], low[nxt]);
    }
  }
}
void dfs2(int now, int col) {
  if (col) {
    color[now].push_back(col);
    bcc[col].push_back(now);
  }
  vis[now] = true;
  for (auto [nxt, p] : adj[now]) {
    if (dfn[nxt] < dfn[now]) ecol[p] = col;
    if (vis[nxt]) continue;
    if (low[nxt] >= dfn[now]) {
      bcnt++;
      ecol[p] = bcnt;
      color[now].push_back(bcnt);
      bcc[bcnt].push_back(now);
      dfs2(nxt, bcnt);
    } else {
      ecol[p] = col;
      dfs2(nxt, col);
    }
  }
}
void get_bcc() {
  for (int i = 1; i <= N; i++) if (!dfn[i]) dfs(i, i);
  for (int i = 1; i <= N; i++) if (!vis[i]) dfs2(i, 0);
  // Comment this if isolated vertices does not form a bcc
  for (int i = 1; i <= N; i++) if (adj[i].empty()) {
    bcnt++;
    color[i].push_back(bcnt);
    bcc[bcnt].push_back(i);
  }
}
```

3.2 SPFA (1548B)

```cpp
// 1-based, average O(E), directed graph
// call init() first, add_edge(u, v, w), then call
shortest_path(source)
// shortest path is stored in dist[v]
// returns true if negative cycle was not detected, false if
negative cycle was detected
// can NOT find negative cycles unreachable from source
typedef long long T;
const int SV = 101010;
// INF should be larger than M * |X|
const T INF = (T)1e18 + 100;
int n;
vector<pair<int, T>> gph[SV];
T dist[SV];
int par[SV], in[SV];
bool inq[SV];
void init(int _n) {
  n = _n;
  for (int i = 1; i <= n; ++i) gph[i].clear();
}
void add_edge(int u, int v, T w) { gph[u].push_back({v, w}); }
bool is_cycle() {
  fill(in + 1, in + n + 1, 0);
  for (int i = 1; i <= n; ++i) if (par[i] != 0) ++in[par[i]];
  queue<int> Q;
  for (int i = 1; i <= n; ++i) if (!in[i]) Q.push(i);
  while (Q.size()) {
    int x = Q.front(); Q.pop();
    if (par[x] != 0 && !--in[par[x]]) Q.push(par[x]);
  }
  // in order to restore cycle, choose any node s.t. in[i] !=
  0, then i belongs to cycle
  // then, follow par[i] until it returns to i
  for (int i = 1; i <= n; ++i) if (in[i]) return true;
  return false;
}
bool shortest_path(int s) {
  // if s = -1, start from all vertices (for only negative
  cycle detection)
  fill(dist + 1, dist + n + 1, INF);
  fill(par + 1, par + n + 1, 0);
  fill(inq + 1, inq + n + 1, false);
  deque<int> Q;
  if (s == -1) {
    for (int i = 1; i <= n; i++) {
      dist[i] = 0; inq[i] = true; Q.push_back(i);
    }
  } else {
    dist[s] = 0; inq[s] = true; Q.push_back(s);
  }
  int cnt = 0;
  while (Q.size()) {
    int x = Q.front();
    Q.pop_front();
    inq[x] = false;
    for (auto [y, w] : gph[x]) {
      if (dist[y] > dist[x] + w) {
        dist[y] = max(dist[x] + w, -INF);
        par[y] = x;
        if (!inq[y]) {
          inq[y] = true;
          if (Q.size() && dist[Q[0]] > dist[y])
          Q.push_front(y);
          else Q.push_back(y);
        }
      }
    }
    // comment this line if negative cycle detection is not
    required
    if (++cnt % n == 0 && is_cycle()) return false;
  }
  return true;
}
```

3.3 Dominator Tree (1433B)

```cpp
// 1-based, O(V + E), general directed graph
// call init() first, add_edge(u, v), then call calc(source)
// result stored in I[v]
// I[v] = 0 if not reachable, else I[v] = idom of vertex v
// I[v] -> v forms a tree rooted at the source
const int SV = 202020;
int n, sp, dfn[SV], ord[SV], par[SV];
int S[SV], I[SV];
vector<int> gph[SV], rev[SV], V[SV];
void init(int _n) {
  n = _n; sp = 0;
  for (int i = 1; i <= n; i++) {
    dfn[i] = 0, ord[i] = 0, S[i] = 1e9, I[i] = 0;
    gph[i].clear();
    rev[i].clear();
  }
}
void add_edge(int u, int v) {
  gph[u].push_back(v);
  rev[v].push_back(u);
}
struct dsu {
  int par[SV], bst[SV];
  void init() {
    iota(par + 1, par + 1 + n, 1);
    iota(bst + 1, bst + 1 + n, 1);
  }
  void merge(int p, int v) { par[v] = p; }
  int calc(int v) {
    if (par[v] == v) return bst[v];
```

```
    int k = calc(par[v]);
    if (S[bst[v]] > S[k]) bst[v] = k;
    par[v] = par[par[v]];
    return bst[v];
  }
} dsu;
void dfs(int v) {
  dfn[v] = ++sp;
  ord[sp] = v;
  for (int x : gph[v]) if (!dfn[x]) {
    par[x] = v;
    dfs(x);
  }
}
void calc(int src) {
  dfs(src);
  dsu.init();
  I[src] = src;
  for (int i = n; i > 1; i--) if (ord[i] != 0) {
    int v = ord[i];
    for (int x : rev[v]) if (dfn[x]) S[v] = min(S[v], dfn[x] >
    dfn[v] ? S[dsu.calc(x)] : dfn[x]);
    dsu.merge(par[v], v);
    V[ord[S[v]]].push_back(v);
    for (int x : V[par[v]]) {
      int y = dsu.calc(x);
      I[x] = (S[x] == S[y]) ? ord[S[x]] : y;
    }
    V[par[v]].clear();
  }
  for (int i = 2; i <= n; i++) if (ord[i] != 0) {
    int v = ord[i];
    I[v] = (I[v] == ord[S[v]]) ? I[v] : I[I[v]];
  }
}
```

3.4 Tree Isomorphism (1325B)

```
// 1-based, O(n log n)
// call hash_init() exactly ONCE before using
// then call tree_init(int n) per each tree before adding any
edges
// use unrooted_hash() if unrooted
// use rooted_hash(parent, vertex) for rooted/subtree hash
typedef unsigned long long ull;
const int SZ = 1010101;
int n, sub[SZ], par[SZ], dep[SZ];
vector<int> gph[SZ];

mt19937 rnd(1557);
ull rng() { return uniform_int_distribution<ull>(0, -1)(rnd); }

ull A[SZ], B[SZ];
void hash_init() {
  for (int i = 0; i < SZ; i++) {
    A[i] = rng();
    B[i] = rng();
  }
}
void tree_init(int _n) {
  n = _n;
  for (int i = 1; i <= n; i++) gph[i].clear();
}
void add_edge(int u, int v) {
  gph[u].push_back(v);
  gph[v].push_back(u);
}
void dfs(int p, int v) {
  sub[v] = 1;
  for (int x : gph[v]) if (p != x) {
    par[x] = v;
    dfs(v, x);
    sub[v] += sub[x];
  }
}
int centroid(int p, int v, int tot) {
  for (int x : gph[v]) if (p != x) {
    if (sub[x] * 2 >= tot) return centroid(v, x, tot);
  }
  return v;
}
ull rooted_hash(int p, int v) {
  vector<ull> V = {1};
  dep[v] = 0;
  for (int x : gph[v]) if (p != x) {
    ull r = rooted_hash(v, x);
    dep[v] = max(dep[v], dep[x] + 1);
    V.push_back(r);
  }
  sort(V.begin(), V.end());
  ull ret = 0;
  for (int i = 0; i < V.size(); i++) ret += V[i] ^ A[dep[v]] ^
  B[i];
  return ret;
}
ull unrooted_hash() {
  dfs(-1, 1);
  int x = centroid(-1, 1, sub[1]);
  if (2 * sub[x] == sub[1]) return rooted_hash(x, par[x]) *
  rooted_hash(par[x], x);
  else return rooted_hash(-1, x);
}
```

3.5 Enumerate triangle, Count 4-cycle (1204B)

```
// 1-based, O(E sqrt E), undirected graph
// at most O(E sqrt E) triangles, achieved when K_n
// at most O(E^2) 4-cycles, achieved when K_n
// call init() first, add_edge(u, v), then call
enumerate_triangles() or count_4_cycles()
void solve(int x, int y, int z);
const int SV = 101010;
int n, cnt[SV], ord[SV], rnk[SV];
vector<int> gph[SV];
void init(int _n) {
  n = _n;
  for (int i = 1; i <= n; i++) gph[i].clear();
}
void add_edge(int u, int v) {
  gph[u].push_back(v);
  gph[v].push_back(u);
}
void precalc() {
  iota(ord + 1, ord + 1 + n, 0);
  sort(ord + 1, ord + 1 + n, [&](int i, int j) { return
  gph[i].size() < gph[j].size(); });
  for (int i = 1; i <= n; i++) rnk[ord[i]] = i;
}
void enumerate_triangles() {
  precalc();
  for (int i = 1; i <= n; i++) {
    int x = ord[i];
    for (int y : gph[x]) cnt[y] = true;
    for (int y : gph[x]) if (rnk[y] < rnk[x]) {
      for (int z : gph[y]) if (cnt[z] && rnk[z] < rnk[y]) {
        // x, y, z forms a triangle
        solve(x, y, z);
      }
    }
    for (int y : gph[x]) cnt[y] = false;
  }
}
ll count_4_cycles() {
  precalc();
  ll ret = 0;
  for (int i = 1; i <= n; i++) {
    int x = ord[i];
    for (int y : gph[x]) if (rnk[y] < rnk[x]) {
      for (int z : gph[y]) if (rnk[z] < rnk[y]) {
        ret += cnt[z]++;
      }
    }
    for (int y : gph[x]) if (rnk[y] < rnk[x]) for (int z :
    gph[y]) cnt[z] = 0;
  }
  return ret;
}
```

3.6 Directed MST (2459B)

```
// 0-based, O(E log V)
// call dmst(n, r, E); where E is the edge list of the graph
// return the cost of r-rooted mst, and parent of each node (r
itself for parent of r)
// all nodes should be reachable from r. dmst() doesn't check
about it :<
typedef long long T;
typedef pair<T, T> ptt;
struct UF1 {
  int n;
  vector<int> par;
  UF1(int n) : n(n), par(n) { iota(par.begin(), par.end(), 0);
  }
  int fnd(int x) { return par[x] == x ? x : par[x] =
  fnd(par[x]); }
  bool uni(int x, int y) { x = fnd(x), y = fnd(y); if(x == y)
  return false; par[x] = y; return true; }
};
struct UF2 {
  int n;
```

```
  vector<int> par;
  vector<pii> his;
  UF2(int n) : n(n), par(n), his() { iota(par.begin(),
  par.end(), 0); }
  int fnd(int x) { return par[x] == x ? x : (his.push_back({x,
  par[x]}), par[x] = fnd(par[x])); }
  void uni(int x, int y) { x = fnd(x), y = fnd(y);
  his.push_back({x, par[x]}); par[x] = y; }
};
struct Edge{int u, v; T w;};
pair<T, vector<int>> dmst(int n, int r, vector<Edge> E) {
  int m = E.size();
  vector<priority_queue<ptt, vector<ptt>, greater<ptt>>> Q(n);
  vector<T> L(n);
  for(int i = 0; i < (int)E.size(); ++i) Q[E[i].v].push({
  E[i].w, i });
  T wgh = 0;
  UF1 uf1(n + m); UF2 uf2(n + m);
  vector<ptt> P(n, {-1, -1});
  vector<int> R;
  queue<int> V;
  for(int i = 0; i < n; ++i) if(i != r) V.push(i);
  while(V.size()) {
    int q = V.front(); V.pop();
    while(q == uf2.fnd(E[Q[q].top().ss].u)) Q[q].pop();
    auto [w, e] = Q[q].top(); Q[q].pop();
    w += L[q];
    wgh += w;
    P[q] = {e, w};
    if(!uf1.uni(E[e].u, E[e].v)) {
      int x = q, y = x;
      vector<int> cyc;
      do { cyc.push_back(y); y = uf2.fnd(E[P[y].ff].u); }
      while(x != y);
      int N = P.size();
      Q.emplace_back();
      L.push_back(0);
      P.push_back({-1, -1});
      R.push_back(uf2.his.size());
      V.push(N);
      uf1.uni(cyc[0], N);
      for(auto i : cyc) uf2.uni(i, N);
      for(int i = 1; i < (int)cyc.size(); ++i)
      if(Q[cyc[0]].size() < Q[cyc[i]].size()) swap(cyc[0],
      cyc[i]);
      swap(Q[cyc[0]], Q[N]);
      swap(L[cyc[0]], L[N]);
      L[N] -= P[cyc[0]].ss;
      for(int i = 1; i < (int)cyc.size(); ++i) {
        while(Q[cyc[i]].size()) {
          auto [_w, _e] = Q[cyc[i]].top(); Q[cyc[i]].pop();
          Q[N].push({ _w + L[cyc[i]] - L[N] - P[cyc[i]].ss, _e
          });
        }
      }
    }
  }
  for(int i = (int)P.size() - 1; i >= n; --i) {
    while(uf2.his.size() > R[i - n]) uf2.par[uf2.his.back().ff]
    = uf2.his.back().ss, uf2.his.pop_back();
    P[uf2.fnd(E[P[i].ff].v)].ff = P[i].ff;
  }
  vector<int> ret(n);
  for(int i = 0; i < n; ++i) ret[i] = (P[i].ff == -1 ? i :
  E[P[i].ff].u);
  return {wgh, ret};
}
```

3.7 K-th Shortest Path (3048B)

```
// call init(|V|) first, add_edge(u, v, x), then kth_walk(s, e,
K)
// O(|E|log|E|+klogk) in total
// if there is negative edge, O((SPFA time) + |E|log|E| +
klogk)
// multi-edges, loops are allowed.
// ALERT: NODES ARE 0-BASED!!!!!!!!
// ALERT: max answer = K|E||X| could be too large.
typedef long long T;
typedef pair<T, T> ptt;
const int SV = 505050, HSIZE = 20202020; // HSIZE >= 2|E|log|E|
const T INF = (T)1e18; // INF >= |E||X| (not need to be larger
than K|E||X|)
int n;
vector<pair<int, T>> gph[SV], rgph[SV];
struct Lheap {
  struct Node {
    ptt x;
    int l, r, s;
    Node(void) : x({0, 0}), l(0), r(0), s(0) {}
    Node(ptt x) : x(x), l(0), r(0), s(1) {}
  }h[HSIZE];
  int cnt = 1;
  int mk(ptt x) { h[cnt] = Node(x); return cnt++; }
  void norm(int x) {
    if(h[h[x].l].s < h[h[x].r].s) swap(h[x].l, h[x].r);
    h[x].s = h[h[x].r].s + 1;
  }
  int mrge(int x, int y) {
    if(!x || !y) return x ^ y;
    if(h[x].x > h[y].x) swap(x, y);
    int ret = mk(h[x].x);
    h[ret].l = h[x].l;
    h[ret].r = mrge(h[x].r, y);
    norm(ret);
    return ret;
  }
}hp;
void init(int _n) {
  n = _n;
  for(int i = 0; i < n; ++i) gph[i].clear();
  for(int i = 0; i < n; ++i) rgph[i].clear();
  hp.cnt = 1;
}
void add_edge(int u, int v, T x) {
  gph[u].push_back({v, x});
  rgph[v].push_back({u, x});
}
// return less than K elements if there is no such walk
vector<T> kth_walk(int s, int e, int K) {
  int nxt[n]; vector<int> top; bool vst[n]{};
  T dst[n]; fill(dst, dst + n, INF); dst[e] = 0;
  typedef tuple<T, int, int> tlii;
  // if there is no negative edge
  priority_queue<tlii, vector<tlii>, greater<tlii>> Q;
  Q.push({0, e, -1});
  while(Q.size()) {
    auto [d, x, p] = Q.top(); Q.pop();
    if(vst[x]) continue;
    vst[x] = true; nxt[x] = p;
    top.push_back(x);
    for(auto [y, c] : rgph[x]) if(!vst[y] && dst[y] > d + c)
    dst[y] = d + c, Q.push({d + c, y, x});
  }
  // if there is negative edge
  // nxt[e] = -1;
  // for(int t = 0; t < n; ++t) {
  //   for(int x = 0; x < n; ++x) for(auto [y, c] : rgph[x])
  if(dst[y] > dst[x] + c)
  //     dst[y] = dst[x] + c, nxt[y] = x;
  // }
  // // OR use SPFA
  // vector<int> ls[n];
  // for(int i = 0; i < n; ++i) if(nxt[i] != -1)
  ls[nxt[i]].push_back(i);
  // queue<int> Q; Q.push(e);
  // while(Q.size()) {
  //   int x = Q.front(); Q.pop();
  //   top.push_back(x);
  //   for(auto y : ls[x]) Q.push(y);
  // }
  if(dst[s] >= INF) return vector<T>();
  bool chc[n]{};
  int rt[n]{};
  for(auto x : top) if(dst[x] < INF) {
    if(nxt[x] != -1) rt[x] = rt[nxt[x]];
    for(auto [y, c] : gph[x]) if(dst[y] < INF) {
      if(!chc[x] && y == nxt[x] && dst[x] == c + dst[y]) {
      chc[x] = true; continue; }
      rt[x] = hp.mrge(rt[x], hp.mk({c + dst[y] - dst[x], y}));
    }
  }
  vector<T> ret({dst[s]});
  priority_queue<ptt, vector<ptt>, greater<ptt>> PQ;
  if(rt[s]) PQ.push({hp.h[rt[s]].x.ff, rt[s]});
  while((int)ret.size() < K && PQ.size()) {
    auto [d, x] = PQ.top(); PQ.pop();
    ret.push_back(dst[s] + d);
    if(rt[hp.h[x].x.ss]) PQ.push({d +
    hp.h[rt[hp.h[x].x.ss]].x.ff, rt[hp.h[x].x.ss]});
    if(hp.h[x].l) PQ.push({d - hp.h[x].x.ff +
    hp.h[hp.h[x].l].x.ff, hp.h[x].l});
    if(hp.h[x].r) PQ.push({d - hp.h[x].x.ff +
    hp.h[hp.h[x].r].x.ff, hp.h[x].r});
  }
  return ret;
}
```

4 Math

4.1 Ultimate Mod (852B)

```
typedef unsigned int ui;
typedef unsigned long long ull;
const ull MOD = 0xffff'ffff'0000'0001;  // primitive root = 7
struct mint {
  ull x;
  mint() : x(0) {}
  mint(ull x) : x(x) {}
  mint operator + (ull ot) const { return x >= MOD - ot ? x -
  MOD + ot : x + ot; }
  mint operator - (ull ot) const { return x >= ot ? x - ot : x
  - ot + MOD; }
  mint operator - () const { return x ? MOD - x : 0; }
  mint operator * (ull ot) const {
    unsigned __int128 p = (unsigned __int128)x * ot;
    ull a = (ull)p;
    ull b = (ui)(p >> 64);
    ull c = p >> 96;
    return mint(a) - b - c + (b << 32);
  }
  mint operator += (ull ot) { return *this = *this + ot; }
  mint operator -= (ull ot) { return *this = *this - ot; }
  mint operator *= (ull ot) { return *this = *this * ot; }
  operator unsigned long long() const { return x; }
};
```

4.2 NTT (1266B)

```
const int MOD = 998244353;
const mint G = 3;  // primitive root
// or (MOD, G) = (0xffffffff00000001, 7)
// define mint, mpow
void dft(vector<mint> &F, bool inv) {
  int n = F.size();
  int b = __lg(n);
  vector<int> rev(n);
  for (int i = 0; i < n; i++) {
    rev[i] = (rev[i / 2] >> 1) | ((i & 1) << (b - 1));
    if (i < rev[i]) swap(F[i], F[rev[i]]);
  }
  vector<mint> w(n);
  w[0] = w[1] = 1;
  for (int k = 2; k < n; k <<= 1) {
    mint z[2] = {1, mpow(G, inv ? MOD - 1 - (MOD - 1) / k / 2 :
    (MOD - 1) / k / 2)};
    for (int i = k; i < (k << 1); ++i) w[i] = w[i >> 1] * z[i &
    1];
  }
  for (int d = 1; d < n; d <<= 1) {
    for (int i = 0; i < n; i += 2 * d) {
      for (int j = 0; j < d; j++) {
        mint b = F[i | j | d] * w[j | d];
        F[i | j | d] = F[i | j] - b;
        F[i | j] += b;
      }
    }
  }
  if (inv) {
    mint val = mpow(n, MOD - 2);
    for (int i = 0; i < n; i++) F[i] *= val;
  }
}
vector<mint> multiply(vector<mint> F, vector<mint> G) {
  int n = 1;
  int sz = F.size() + G.size() - 1;
  while (n < (int)F.size() + G.size()) n <<= 1;
  F.resize(n); G.resize(n);
  dft(F, false); dft(G, false);
  for (int i = 0; i < n; i++) F[i] *= G[i];
  dft(F, true);
  F.resize(sz);
  return F;
}
```

4.3 FFT (1956B)

```
typedef complex<double> cpx;
typedef __int128 dll;
// D^2 < |X|
const int D = 32000;
const double pi = acos((double)(-1));
int M, B;
vector<int> rev;
vector<cpx> _w;
// |F|+|G| <= 2^_B
void init(int _B) {
  B = _B;
  M = 1 << B;
  _w.resize(M / 2);
  rev.resize(M);
  for (int i = 0; i < M / 2; i++) _w[i] = {cos(pi * i / M),
  sin(pi * i / M)};
  for (int i = 0; i < M; i++) rev[i] = (rev[i / 2] >> 1) | ((i
  & 1) << (B - 1));
}
void dft(vector<cpx> &F, bool inv) {
  int n = F.size();
  int b = __lg(n);
  vector<cpx> w(n);
  w[0] = w[1] = 1;
  for (int k = 2; k < n; k <<= 1) {
    for (int i = 0; i < k; ++i) w[k | i] = _w[M / k / 2 * i];
  }
  if (inv) for (int i = 0; i < n; ++i) w[i] = conj(w[i]);
  for (int i = 0; i < n; i++) {
    int p = rev[i] >> (B - b);
    if (i < p) swap(F[i], F[p]);
  }
  for (int d = 1; d < n; d *= 2) {
    for (int i = 0; i < n; i += 2 * d) {
      for (int j = 0; j < d; j++) {
        cpx b = F[i | j | d] * w[j | d];
        F[i | j | d] = F[i | j] - b;
        F[i | j] += b;
      }
    }
  }
  if (inv) for (int i = 0; i < n; i++) F[i] /= n;
}
// Safe for sum(p[i]^2 + q[i]^2) lg2(n) < 9e14
vector<dll> multiply(vector<ll> F, vector<ll> G) {
  int n = 1;
  while (n < (int)F.size() + G.size()) n <<= 1;
  vector<cpx> P(n);
  vector<cpx> Q(n);
  for (int i = 0; i < F.size(); i++) P[i] = {F[i] / D, F[i] %
  D};
  for (int i = 0; i < G.size(); i++) Q[i] = {G[i] / D, G[i] %
  D};
  dft(P, false);
  dft(Q, false);
  vector<cpx> R(n), S(n);
  for (int i = 0; i < n; i++) {
    int j = (n - i) % n;
    R[i] = (P[i] + conj(P[j])) * Q[i] * cpx(0.5, 0);
    S[i] = (P[i] - conj(P[j])) * conj(Q[j]) * cpx(0.5, 0);
  }
  dft(R, true);
  dft(S, true);
  vector<ll> Z((int)F.size() + (int)G.size() - 1);
  for (int i = 0; i < (int)Z.size(); i++) Z[i] =
  (ll)round(R[i].real()) * D * D + (ll)round(R[i].imag() +
  S[i].imag()) * D + (ll)round(S[i].real());
  return Z;
}
```

4.4 ext-euclid (1167B)

```
typedef __int128 dll;

// REQUIRED : m != 0
// computes floor(x / m), ceil(x / m), x mod m
ll mfloor(ll x, ll m) { return x / m - ((x ^ m) < 0 && x % m);
}
ll mceil(ll x, ll m) { return x / m + ((x ^ m) > 0 && x % m); }
ll mmod(dll x, ll m) { return x % m + (x < 0 && x % m) *
abs(m); }

// REQUIRED : a, b > 0, gcd(a, b) = 1
// computes a^{-1} mod b
// 0 <= result < b
ll get_inv(ll a, ll b) {
  return (b == 1) ? 0 : mmod((1 - (dll)b * get_inv(b, a % b)) /
  (a % b), b);
}

// REQUIRED : 0 <= x1 < m1 and 0 <= x2 < m2, numbers less than
2 * 10^12
// solves x = x1 (mod m1), x = x2 (mod m2)
// if there is no solution, returns false
// otherwise x = a mod b, returns true
bool crt(ll x1, ll m1, ll x2, ll m2, ll &a, ll &b) {
  ll g = __gcd(m1, m2);
  if (x1 % g != x2 % g) return false;
  ll r = x1 % g;
  m1 /= g; m2 /= g; x1 /= g; x2 /= g;
  ll x = (dll)x1 * m2 * get_inv(m2, m1) + (dll)x2 * m1 *
  get_inv(m1, m2);
  ll m = m1 * m2;
  a = g * mmod(x, m) + r;
  b = g * m;
```

```
  return true;
}

// REQUIRED : (a, b) != (0, 0), numbers less than 9 * 10^18
// solves ax + by = c
// if there is no solution, returns false
// otherwise (x + sk, y - tk) describes all solutions
// satisfies s >= 0, 0 <= x < s
bool diophantine(ll a, ll b, ll c, ll &x, ll &y, ll &s, ll &t)
{
  if (!a && !b) assert(false);
  ll g = __gcd(abs(a), abs(b));
  if (c % g != 0) return false;
  if (!a) {
    x = 0; y = c / b; s = 1; t = 0;
    return true;
  }
  if (!b) {
    x = c / a; y = 0; s = 0, t = 1;
    return true;
  }
  a /= g; b /= g; c /= g;
  x = mmod((dll)get_inv(abs(a), abs(b)) * (a < 0 ? -c : c),
  abs(b));
  y = (c - (dll)x * a) / b;
  s = b; t = a;
  if (s < 0) s *= -1, t *= -1;
  return true;
}
```

4.5 RREF (477B)

```
const double eps = 1e-9;
void rref(vector<vector<double>> &A) {
  int n = A.size(), m = A[0].size();
  for (int i = 0, j = 0; j < m; ++j) {
    int t = i;
    while (t < n && abs(A[t][j]) < eps) ++t;
    if (t == n) continue;
    swap(A[i], A[t]);
    double p = A[i][j];
    for (int k = j; k < m; ++k) A[i][k] /= p;
    for (int l = 0; l < n; ++l) if (l != i) {
      p = A[l][j];
      for (int k = j; k < m; ++k) A[l][k] -= p * A[i][k];
    }
    ++i;
  }
}
```

4.6 Pollard-Rho, Miller-Rabin (1298B)

```
// call isPrime(n) for primality test : O(log n)
// call factorize(n) for factorization : O(n^{1/4} log n)
// result of factorize(n) may NOT be sorted
typedef long long ll;
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n - 1), d = n >> s;
  for (ull a : A) {  // ^ count trailing zeroes
    ull p = modpow(a % n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--) p = modmul(p,
    p, n);
    if (p != n - 1 && i != s) return 0;
  }
  return 1;
}
ull pollard(ull n) {
  auto f = [n](ull x) { return modmul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x, y) - min(x, y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), r.begin(), r.end());
  return l;
}
```

4.7 Tonelli-Shanks (775B)

```
// find one x such that x^2 = n (mod p)
// returns -1 if does not exist
// p should be a prime, 0 <= n < p
// O(log^2 p)
mt19937 rnd(1557);
ll rng(ll l, ll r) { return uniform_int_distribution<ll>(l,
r)(rnd); }
ll mpow(ll a, ll x, ll mod) {
  ll r = 1;
  while (x) {
    if (x & 1) r = r * a % mod;
    a = a * a % mod;
    x /= 2;
  }
  return r;
}
ll quadratic_residue(ll n, ll p) {
  if (n == 0) return 0;
  if (p == 2) return n;
  if (mpow(n, (p - 1) / 2, p) != 1) return -1;

  int q = p - 1, s = __builtin_ctz(q);
  q >>= s;
  ll x = mpow(n, (q + 1) / 2, p);
  ll t = mpow(n, q, p);
  ll z = rng(1, p - 1);
  while (mpow(z, (p - 1) / 2, p) == 1) z = rng(1, p - 1);

  while (t != 1) {
    ll i = 0, v = t;
    while (v != 1) v = v * v % p, ++i;
    ll b = mpow(z, q * (1ll << (s - i - 1)), p);
    x = x * b % p;
    t = t * b % p * b % p;
  }
  return x;
}
```

4.8 Linear Sieve (690B)

```
const int MAXN = 1010101;
bool prime[MAXN];
int phi[MAXN];
void calc() {
  fill(prime, prime + MAXN, true);
  prime[0] = prime[1] = false;
  vector<int> P;
  for (int n = 2; n < MAXN; n++) {
    if (prime[n]) {
      P.push_back(n);
      phi[n] = n - 1;
    }
    for (int pr : P) {
      ll x = (ll)n * pr;
      if (x >= MAXN) break;
      prime[x] = false;
      // for general multiplicative function, you should
      maintain the highest prime divisor and its order
      // mul[x] = mul[n] / mul[p ^ k] * mul[p ^ (k + 1)]
      if (n % pr == 0) {
        phi[x] = phi[n] * pr;
        break;
      } else {
        phi[x] = phi[n] * (pr - 1);
      }
    }
  }
}
```

4.9 min25 Sieve (2237B)

```
/**
Let f(x) be a multiplicative function and
when p is a prime, f(p) has a polynomial representation
and f(p^k) can be calculated quickly

We can compute the prefix sum of f(n) in O((n^(3/4)) / logn)
using Min_25 sieve
Takes ~1s for n = 10^10 (deg = 3) in CF
unfold the loops over poly to make the code faster
Tutorial: https://www.luogu.com.cn/problem/solution/P5325
**/
int primes[N], p, deg;
```

```
T pref[D][N];
T poly[D];  // polynomial representation of f(p)
void sieve(int n) {
  vector<bool> f(n + 1, false);
  p = 0;
  for (int i = 2; i <= n; i++) {
    if (!f[i]) {
      primes[++p] = i;
      T cur = 1;
      // pref[k][p] = sum of primes[i]^k s.t. i <= p
      for (int k = 0; k < deg; k++) {
        pref[k][p] = pref[k][p - 1] + cur;
        cur *= i;
      }
    }
    for (int j = 1; j <= p and primes[j] * i <= n; j++) {
      f[i * primes[j]] = 1;
      if (i % primes[j] == 0) break;
    }
  }
}
int r, id1[N], id2[N];
ll n;
int get_id(ll x) { return x <= r ? id1[x] : id2[n / x]; }
int tot;
T g[D][N];
ll a[N];
// f(p^k) in O(1)
inline T eval(int p, int k, ll pw) {  // pw = p^k, pw <= n
  T ans = pw % mod;
  return ans * (ans - 1);
}
// no memorization needed!
// sum of f(i) s.t. spf[i] > primes[j]
T yo(ll x, int j) {
  if (primes[j] >= x) return 0;
  int id = get_id(x);
  T ans = 0;
  // initialization: ans = g(x) - g(primes[j])
  for (int k = 0; k < deg; k++) ans += poly[k] * g[k][id];
  for (int k = 0; k < deg; k++) ans -= poly[k] * pref[k][j];
  for (int i = j + 1; i <= p and primes[i] <= x / primes[i];
  i++) {
    ll pw = primes[i];
    for (int e = 1; pw <= x; e++) {
      ans += eval(primes[i], e, pw) * (yo(x / pw, i) + (e !=
      1));
      if (!(pw <= x / primes[i])) break;
      pw *= primes[i];
    }
  }
  return ans;
}
// sum of f(i) for 1 <= i <= n
// pol is the polynomial representation of f(p)
T solve(ll _n, vector<T> pol) {
  n = _n;
  deg = pol.size();
  for (int i = 0; i < deg; i++) poly[i] = pol[i];
  r = sqrt(n);
  while (1LL * r * r < n) ++r;
  while (1LL * r * r > n) --r;
  sieve(r);
  tot = 0;
  ll i = 1;
  while (i <= n) {
    ll x = n / i;
    ll j = n / x;
    a[++tot] = x;
    // initialization g[k][tot] = sum of i^k for 2 <= i <= x
    T z = x % mod;  // remove this mod if not needed
    for (int k = 0; k < deg; k++) {
      if (k == 0) g[k][tot] = z - 1;
      if (k == 1) g[k][tot] = z * (z + 1) / 2 - 1;
      if (k == 2) g[k][tot] = z * (z + 1) * (2 * z + 1) / 6 -
      1;
      // compute for larger values using lagrange if needed
    }
    if (x <= r) id1[x] = tot;
    else id2[n / x] = tot;
    i = j + 1;
  }
  // an integer x belongs to the array a iff for some integer
  z, n / z = x
  // g[k][i] = sum of prime^k for prime <= a[i]
  for (int i = 1; i <= p; i++) {
    for (int j = 1; j <= tot && primes[i] <= a[j] / primes[i];
    j++) {
      int id = get_id(a[j] / primes[i]);
      T cur = 1;
      for (int k = 0; k < deg; k++) {
        g[k][j] -= cur * (g[k][id] - pref[k][i - 1]);
        cur *= primes[i];
      }
    }
  }
  return yo(n, 0) + 1;  // add f(1)
}
```

4.10 FWHT (597B)

$$c_i = \sum_{j \star k = i} a_j \cdot a_k$$

$$T_{\oplus} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix} \cdot T_{\oplus} \begin{bmatrix} q_0 \\ q_1 \end{bmatrix} = T_{\oplus} \begin{bmatrix} p_0 q_0 + p_1 q_1 \\ p_0 q_1 + p_1 q_0 \end{bmatrix}$$

$$T_{\oplus} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, T_{\oplus}^{-1} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$T_{\&} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, T_{\&}^{-1} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$T_{|} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, T_{|}^{-1} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

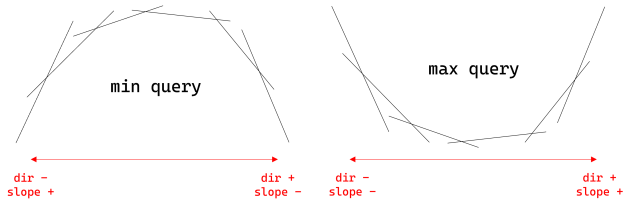
```
void DFT(vector<ll> &A, bool inv) {
  int N = A.size(), B = __lg(N);
  for (int i = 1; i < (1 << B); i <<= 1) {
    for (int j = 0; j < (1 << B); j += i + i) {
      for (int k = 0; k < i; k++) {
        ll a = A[j + k], b = A[j + i + k];
        // XOR
        A[j + k] = a + b, A[j + i + k] = a - b;
        if (inv) A[j + k] /= 2, A[j + i + k] /= 2;
        // AND
        A[j + k] = b, A[j + i + k] = a + b;
        if (inv) A[j + k] = -a + b, A[j + i + k] = a;
        // OR
        A[j + k] = a, A[j + i + k] = a + b;
        if (inv) A[j + k] = a, A[j + i + k] = -a + b;
      }
    }
  }
}
```

4.11 Polynomial

- Inverse: $B_k = A^{-1} \pmod{x^a}$일 때 $B_{k+1} = B_k(2 - AB_k) \pmod{x^{2a}}$로 놓으면 $B_{k+1} = A^{-1} (mod\ x^{2a})$. $O(n \log n)$
- Euclidean Division: $\deg A = n, \deg B = m$일 때, $B(x)D(x) + R(x) = A(x), \deg D \leq m - n$인 D 찾기; $D^R(x) = A^R(x)(B^R(x))^{-1} \pmod{x^{n-m+1}}$; 이 때 $A^R(x) = x^n A(x^{-1})$, $B^R(x) = x^m B(x^{-1})$, $D^R(x) = x^{m-n} D(x^{-1})$. $O(n)$
- Newton's Method: $F(P) = 0$인 polynomial P를 찾기; $F(Q_k) = 0 \pmod{x^a}$일 때 $Q_{k+1} = Q_k - \frac{F(Q_k)}{F'(Q_k)} \pmod{x^{2a}}$로 놓으면 $F(Q_{k+1}) = 0 \pmod{x^{2a}}$.
  Example : $Q = e^P$, $lnQ = P$, $F(Q) = lnQ - P$, $F'(Q) = \frac{1}{Q}$
- Logarithm: $\int \frac{P'(x)}{P(x)}$. $O(n \log n)$
- Exponent: $Q_{k+1} = Q_k(1 + P - lnQ_k)$. $O(n \log n)$
- Multipoint Evaluation: $A(x)$를 $(x - p_i)$로 나눈 나머지를 찾는 문제와 같으니, 분할정복으로 나머지 계산. $O(n \log^2 n)$
- Interpolation: $A(x) = \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$. 각각의 i에 대한 분모만 알면 분할정복을 할 수 있음. 분모를 알기 위해 $P(x) = (x - x_1)...(x - x_n)$이라고 하면 i번째 분모는 $P'(x_i)$. $O(n \log^2 n)$
- $\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$

5 Data Structure

5.1 CHT (1405B)



min/max	min	min	max	max
slope	+	-	+	-
dir	-	+	+	-

+ : inc
- : dec

```
typedef __int128 dll;
struct Line { ll a, b; };
struct Frac {
  ll u, d;
  Frac(ll _u, ll _d) : u(_u), d(_d) { if (d < 0) u = -u, d =
  -d; }
  bool operator<(const Frac &ot) const { return (dll)u * ot.d <
  (dll)ot.u * d; }
  bool operator>(const Frac &ot) const { return (dll)u * ot.d >
  (dll)ot.u * d; }
};
ll divfloor(ll u, ll d) { return u / d - ((u ^ d) < 0 && u %
d); }
ll divceil(ll u, ll d) { return u / d + ((u ^ d) > 0 && u % d);
}

// Get cross point of line p, q
// If all queries are integer, change 'Frac' to 'div'
Frac cross(const Line &p, const Line &q) { return Frac(p.b -
q.b, q.a - p.a); }  // dir + : divfloor, dir - : divceil

// min/max : ?, slope : ?, dir : ?
struct CHT {
  deque<Line> V;

  // Insert line p = ax+b
  // b must be increasing (or decreasing) ('slope')
  // cross(V[i-1], V[i]) < (or >) cross(V[i], V[i+1]) ('dir')
  void push(Line p) {
    if (!V.empty() && V.back().a == p.a) {
      if (V.back().b <= p.b) return;  // min : <= , max : >=
      V.pop_back();
    }
    while (V.size() > 1 && !(cross(V[V.size() - 2], V[V.size()
    - 1]) < cross(V[V.size() - 1], p))) V.pop_back();  // dir +
    : <, dir - : >
    V.push_back(p);
  }

  // Get min (or max) value at x in O(logN)
  ll query(ll x) {
    assert(!V.empty());
    int lo = 0, hi = V.size();
    while (lo + 1 < hi) {
      int mid = lo + hi >> 1;
      if (cross(V[mid - 1], V[mid]) < Frac(x, 1)) lo = mid;  //
      dir + : <, dir - : >
      else hi = mid;
    }
    return V[lo].a * x + V[lo].b;
  }

  // Get min (or max) value at x in ammortized O(1)
  // x must be increasing (or decreasing) ('dir')
  ll query2(ll x) {
    assert(!V.empty());
    while (V.size() > 1 && cross(V[0], V[1]) < Frac(x, 1))
    V.pop_front();  // dir + : <, dir - : >
    return V[0].a * x + V[0].b;
  }
};
```

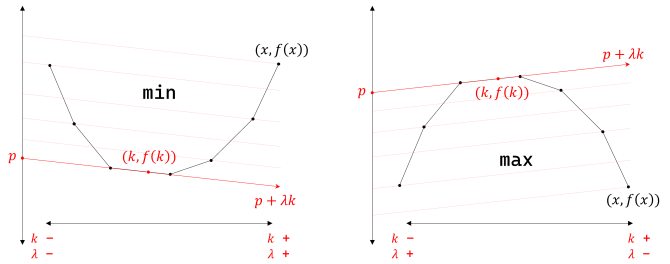
5.2 Li Chao Tree (1428B)

```
const ll INF = 4e18;
struct Line {
  ll a, b;
  Line() : a(0), b(INF) {}  // min : INF , max : -INF
  Line(ll _a, ll _b) : a(_a), b(_b) {}
  ll operator()(ll x) { return a * x + b; }
};
struct Node {
  Line f;
  int lc, rc;
  Node() : f(), lc(-1), rc(-1) {}
};
struct LiChao {
  int root;
  vector<Node> NS;
  LiChao() {
    NS = vector<Node>();
    root = newNode();
  }
  int newNode() {
    NS.push_back(Node());
    return NS.size() - 1;
  }
  // Insert line p in O(logN)
  void update(int node, ll tl, ll tr, Line p) {
    ll mid = tl + tr >> 1;
    if (p(mid) < NS[node].f(mid)) swap(p, NS[node].f);  // min
    : < , max : >
    if (tl == tr) return;
    if (p(tl) < NS[node].f(tl)) {  // min : < , max : >
      if (NS[node].lc == -1) NS[node].lc = newNode();
      update(NS[node].lc, tl, mid, p);
    }
    if (p(tr) < NS[node].f(tr)) {  // min : < , max : >
      if (NS[node].rc == -1) NS[node].rc = newNode();
      update(NS[node].rc, mid + 1, tr, p);
    }
  }
  // Get min (or max) value at x in O(logN)
  ll query(int node, ll tl, ll tr, ll x) {
    if (node == -1) return INF;  // min : INF , max : -INF
    ll mid = tl + tr >> 1;
    ll ret = NS[node].f(x);
    if (x <= mid) ret = min(ret, query(NS[node].lc, tl, mid,
    x));  // min : min() , max : max()
    else ret = min(ret, query(NS[node].rc, mid + 1, tr, x));
    // min : min() , max : max()
    return ret;
  }
};
```

5.3 LineContainer (1115B)

```
const ll INF = 9e18;
struct Line {
  mutable ll a, b, p;
  bool operator<(const Line& o) const { return a != o.a ? o.a <
  a : b < o.b; }  // min : < , max : > (both)
  bool operator<(ll x) const { return p < x; }
};
// LineContainer::LineContainer cht;
// multiset of Line, sorted in increasing order of cross points
struct LineContainer : multiset<Line, less<>> {
  // for doubles, use INF = 1/.0, div(a,b) = a/b
  ll div(ll u, ll d) { return u / d - ((u ^ d) < 0 && u % d); }
  bool isect(iterator x, iterator y) {
    if (y == end()) {
      x->p = INF;
      return 0;
    }
    if (x->a == y->a) x->p = x->b < y->b ? INF : -INF;  // min
    : < , max : >
    else x->p = div(y->b - x->b, x->a - y->a);
    return x->p >= y->p;
  }
  void push(ll a, ll b) {
    auto z = insert({a, b, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x,
    erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.a * x + l.b;
  }
};
```

5.4 Alien Trick (1508B)



$$dp[i][k] = min_{j<i}(dp[j][k-1] + cost(j,i))$$

$$dp2[i] = min_{j<i}(dp2[j] + cost(j,i)) - \lambda$$

$f(k) = dp[N][k]$라 정의하자. 적당한 λ에 대하여 $p = f(k) - \lambda k$를 최적화한다. $f(k) = p + \lambda k$이니, $(x, f(x))$를 선으로 이은 그래프에서 기울기 λ의 접선을 그었을 때의 y좌표가 최적의 p가 된다. $p = f(k) - \lambda k$는 고정된 λ에 대하여 최적의 $dp2[N]$을 의미한다. 답은 $dp2[N] + \lambda k$이다.

```
const int MAXN = 5e5;
int N;
// dp[i] : optimal dp value of i
// cnt[i] : how much transitions were used in optimal dp[i]
// memo[i] : previous transition position from i
```

```cpp
// V : restored optimal solution
ll dp[MAXN + 10];
int cnt[MAXN + 10], memo[MAXN + 10];
vector<int> V;
void init(int _N) { N = _N; }
// For given lambda, calculate dp, cnt, memo, V
// dp[i] = min(or max)_{j<i} (dp[j] + cost(j, i)*2 - lambda)
// changes dp, cnt, memo, V
void solve(ll lambda) {
  // initialize dp, cnt, memo, V, (other data structures)
  for (int i = 0; i <= N; i++) dp[i] = cnt[i] = memo[i] = 0;
  V.clear();
  for (int i = 1; i <= N; i++) {
    // get_opt(i), cost(p, q) must be implemented
    // opt = argmin(or max)_{j<i} (dp[j] + cost(j, i)*2)
    // your code goes here
    int opt = get_opt(i);
    dp[i] = dp[opt] + cost(opt, i) * 2 - lambda;  // Don't
    forget *2
    cnt[i] = cnt[opt] + 1;
    memo[i] = opt;
  }
  for (int i = N; i > 0;) {
    V.push_back(i);
    i = memo[i];
  }
  V.push_back(0);
  reverse(V.begin(), V.end());
}
// Get optimal dp[N][K] for given K
// Returns (answer, restored solution)
// dp[i][k] = min(or max)_{j<i} (dp[j][k-1] + cost(j, i))
pair<ll, vector<int>> alien(int K) {
  // lambda equals slope
  // minimum : K increase, lambda increase
  // maximum : K increase, lambda decrease
  ll lo = -1e18, hi = 1e18;  // range for lambda is [2*lo+1,
  2*hi+1]
  while (lo + 1 < hi) {
    ll mid = lo + hi >> 1;
    solve(2 * mid + 1);
    if (K <= cnt[N]) hi = mid;  // min : <= , max : >=
    else lo = mid;
  }
  vector<int> P1, P2, ansV;
  solve(2 * lo + 1); P1 = V;
  solve(2 * hi + 1); P2 = V;
  if (P1.size() > P2.size()) swap(P1, P2);
  if (P1.size() - 1 == K) ansV = P1;
  else if (P2.size() - 1 == K) ansV = P2;
  else {
    assert(P1.size() - 1 < K && K < P2.size() - 1);
    int x = K + 1 - P1.size();
    for (int i = 0; i + 1 < P1.size() && i + x + 1 < P2.size();
    i++) {
      if (P1[i] <= P2[i + x] && P2[i + x + 1] <= P1[i + 1]) {
        for (int j = 0; j <= i + x; j++) ansV.push_back(P2[j]);
        for (int j = i + 1; j < P1.size(); j++)
        ansV.push_back(P1[j]);
        break;
      }
    }
  }
  assert(ansV.size() == K + 1);
  solve(2 * hi);
  ll ans = dp[N] / 2 + hi * K;
  return {ans, ansV};
}
```

5.5 Monotone Queue Trick (1072B)

$$dp[i] = min_{j<i}(dp[j] + cost(j, i))$$

$cost(j, i)$가 monge 하니, $p \le q \le x_1 \le x_2$ 에 대해 $cost(p, x_1) - cost(q, x_1) \le cost(p, x_2) - cost(q, x_2)$. $f_p(x) = dp[p] + cost(p, x)$라 하면 $f_p(x_1) - f_q(x_1) \le f_p(x_2) - f_q(x_2)$. x가 증가함에 따라 $f_p(x)$가 최적이었다가, $f_q(x)$가 최적으로 바뀐다.

두 함수 $f_p(x)$, $f_q(x)$에 대하여 $cross(p, q)$를 $f_p(x) < f_q(x)$인 최대 x로 정의한다. 최적의 함수 $f_p(x) = dp[p] + cost(p, x)$들을 deque로 관리한다. $cross(DQ[i-1], DQ[i]) < cross(DQ[i], DQ[i+1])$를 강제한다.

```cpp
const int MAXN = 5e5;
int N;
// dp[i] : optimal dp value of i
// cnt[i] : how much transitions were used in optimal dp[i]
// memo[i] : previous transition position from i
// V : restored optimal solution
ll dp[MAXN + 10];
int cnt[MAXN + 10], memo[MAXN + 10];
vector<int> V;
void init(int _N) { N = _N; }
// cost(p, q) must be monge
// cost(a, d) + cost(b, c) > cost(a, c) + cost(b, d)
ll cost(int p, int q) {}
// Get maximum x that dp[p]+cost(p, x) < dp[q]+cost(q, x) (p is
optimal than q)
// p < q must hold
int cross(int p, int q) {
  int lo = q, hi = N + 1;
  while (lo + 1 < hi) {
    int mid = lo + hi >> 1;
    if (dp[p] + cost(p, mid) < dp[q] + cost(q, mid)) lo = mid;
    // min : <, max : >
    else hi = mid;
  }
  return lo;
}
// Get optimal dp[N]
// dp[i] = min(or max)_{j<i} (dp[j] + cost(j, i))
// changes dp, cnt, memo, V
void solve() {
  // initialize dp, cnt, memo, V, (other data structures)
  for (int i = 0; i <= N; i++) dp[i] = cnt[i] = memo[i] = 0;
  V.clear();
  deque<int> DQ;
  DQ.push_back(0);
  for (int i = 1; i <= N; i++) {
    while (DQ.size() > 1 && dp[DQ[0]] + cost(DQ[0], i) >=
    dp[DQ[1]] + cost(DQ[1], i)) DQ.pop_front();  // min : >=,
    max : <=
    // opt = argmin(or max)_{j<i} (dp[j] + cost(j, i))
    int opt = DQ.front();
    dp[i] = dp[opt] + cost(opt, i);
    cnt[i] = cnt[opt] + 1;
    memo[i] = opt;
    // cross(DQ[i-1], DQ[i]) < cross(DQ[i], DQ[i+1])
    while (DQ.size() > 1 && !(cross(DQ[DQ.size() - 2],
    DQ[DQ.size() - 1]) < cross(DQ[DQ.size() - 1], i)))
    DQ.pop_back();
    DQ.push_back(i);
  }
  for (int i = N; i > 0;) {
    V.push_back(i);
    i = memo[i];
  }
  V.push_back(0);
  reverse(V.begin(), V.end());
}
```

5.6 Kinetic Segment Tree (3023B)

```cpp
const ll INF = 7e18;
struct Line { ll a, b; };
ll divfloor(ll u, ll d) { return u / d - ((u ^ d) < 0 && u %
d); }
// p(0) <= q(0) must hold ( min : <= , max : >= )
// minimum t>0 s.t. p(t) > q(t) ( min : > , max : < )
ll cross(Line p, Line q) {
  assert(p.b <= q.b);          // min : <= , max : >=
  if (p.a <= q.a) return INF;  // min : <= , max : >=
  return divfloor(p.b - q.b, q.a - p.a) + 1;
}
struct SEG {
  vector<Line> tree;
  vector<ll> melt;  // minimum time left for next cross in
  subtree
  vector<ll> lazy;
  SEG(int N) {
    tree = vector<Line>(N * 4 + 10, {0, INF});  // min : INF ,
    max : -INF
    melt = vector<ll>(N * 4 + 10, INF);
    lazy = vector<ll>(N * 4 + 10, 0);
  }
  // Apply update value k to subtree of node
  // update to node itself, and to lazy value
  void apply(int node, ll k) {
    assert(k < melt[node]);
    tree[node].b += tree[node].a * k;
    melt[node] -= k;
    lazy[node] += k;
  }
  // Propagate lazy value to children, and initialize lazy
  // should be called before going down to children
  void prop(int node, int tl, int tr) {
    if (tl != tr) {
      apply(node * 2, lazy[node]);
      apply(node * 2 + 1, lazy[node]);
    }
    lazy[node] = 0;
```

```
  }
  // Recalculate value of node from children
  // lazy must be empty
  void recalc(int node) {
    assert(lazy[node] == 0);
    melt[node] = min(melt[node * 2], melt[node * 2 + 1]);
    if (tree[node * 2].b <= tree[node * 2 + 1].b) {  // min :
    <= , max : >=
      tree[node] = tree[node * 2];
      melt[node] = min(melt[node], cross(tree[node * 2],
      tree[node * 2 + 1]));
    }
    else {
      tree[node] = tree[node * 2 + 1];
      melt[node] = min(melt[node], cross(tree[node * 2 + 1],
      tree[node * 2]));
    }
  }
  // B[i]+=A[i]*t for all i in [l, r]
  // Ammortized O(Qlog^3N) if range heaten
  // Ammortized O(Qlog^2N) if entire heaten
  void heaten(int node, int tl, int tr, int l, int r, ll t) {
    if (r < tl || tr < l) return;
    if (l <= tl && tr <= r && melt[node] > t) {
      apply(node, t);
      return;
    }
    prop(node, tl, tr);
    int mid = tl + tr >> 1;
    heaten(node * 2, tl, mid, l, r, t);
    heaten(node * 2 + 1, mid + 1, tr, l, r, t);
    recalc(node);
  }
  // get minimum/maximum B[i] for i in [l, r]
  // O(QlogN)
  ll query(int node, int tl, int tr, int l, int r) {
    if (r < tl || tr < l) return INF;  // min : INF , max :
    -INF
    if (l <= tl && tr <= r) return tree[node].b;
    prop(node, tl, tr);
    int mid = tl + tr >> 1;
    return min(query(node * 2, tl, mid, l, r), query(node * 2 +
    1, mid + 1, tr, l, r));  // min : min() , max : max()
  }
  // update A[pos], B[pos]
  // O(QlogN)
  void update(int node, int tl, int tr, int pos, Line p) {
    if (tl == tr) {
      tree[node] = p;
      return;
    }
    prop(node, tl, tr);
    int mid = tl + tr >> 1;
    if (pos <= mid) update(node * 2, tl, mid, pos, p);
    else update(node * 2 + 1, mid + 1, tr, pos, p);
    recalc(node);
  }
};
```

5.7 Splay Tree (3438B)

```
struct Node {
  int sz;
  int par, lc, rc;
  // val : value stored in node, sum : query value stored in
  node
  // lazy : lazy value to be applied to subtree of node
  (already applied to val, sum)
  ll val, sum;
  ll lazy;
  Node(ll x) {
    sz = 1;
    par = lc = rc = 0;
    // your code goes here
    val = sum = x;
    lazy = 0;
  }
  Node() {
    sz = par = lc = rc = 0;
    val = sum = 0;
    lazy = 0;
  }
};
// root must be initialized
// NS[0] : NIL node
int root;
vector<Node> NS;
void init() { NS = vector<Node>(1); root = 0; }
int newNode(ll x) { NS.push_back(Node(x)); return NS.size() -
1; }

void recalc(int node) {
  if (node == 0) return;
  int l = NS[node].lc, r = NS[node].rc;
  NS[node].sz = NS[l].sz + NS[r].sz + 1;
  // your code goes here
  NS[node].sum = NS[node].val + NS[l].sum + NS[r].sum;
}
void apply(int node, ll upd) {
  if (node == 0) return;
  // your code goes here
  NS[node].lazy += upd;
  NS[node].val += upd;
  NS[node].sum += upd * NS[node].sz;
}
void prop(int node) {
  if (node == 0) return;
  if (NS[node].lazy == 0) return;
  apply(NS[node].lc, NS[node].lazy);
  apply(NS[node].rc, NS[node].lazy);
  // your code goes here
  NS[node].lazy = 0;
}

void prop_anc(int x) { if (NS[x].par != 0) prop_anc(NS[x].par);
prop(x); }
void rotate(int x) {
  assert(x != 0 && NS[x].par != 0);
  int p = NS[x].par, q;
  if (NS[p].lc == x) {
    NS[p].lc = q = NS[x].rc;
    NS[x].rc = p;
  }
  else {
    NS[p].rc = q = NS[x].lc;
    NS[x].lc = p;
  }
  NS[x].par = NS[p].par;
  NS[p].par = x;
  if (q) NS[q].par = p;
  if (NS[x].par != 0) {
    if (NS[NS[x].par].lc == p) NS[NS[x].par].lc = x;
    else NS[NS[x].par].rc = x;
  }
  recalc(p);
  recalc(x);
}
void splay(int x) {
  root = x;
  if (x == 0) return;
  prop_anc(x);
  while (NS[x].par) {
    int p = NS[x].par, q = NS[p].par;
    if (q) rotate((NS[p].lc == x) == (NS[q].lc == p) ? p : x);
    rotate(x);
  }
}
// Find kth node in subtree of node, and make it root
int find_kth(int node, int k, int &root) {
  if (node == 0) return 0;
  assert(1 <= k && k <= NS[node].sz);
  prop(node);
  if (k <= NS[NS[node].lc].sz) return find_kth(NS[node].lc, k,
  root);
  if (k == NS[NS[node].lc].sz + 1) { splay(node); return root =
  node; }
  return find_kth(NS[node].rc, k - NS[NS[node].lc].sz - 1,
  root);
}
// Insert node x after the kth node in subtree of node, and
make it root
void insert(int node, int k, int x, int &root) {
  assert(0 <= k && k <= NS[node].sz);
  prop(node);
  if (k <= NS[NS[node].lc].sz) {
    if (NS[node].lc == 0) {
      NS[node].lc = x;
      NS[x].par = node;
      splay(x);
      root = x;
    }
    else insert(NS[node].lc, k, x, root);
  }
  else {
    if (NS[node].rc == 0) {
      NS[node].rc = x;
      NS[x].par = node;
```

```
      splay(x);
      root = x;
    } else insert(NS[node].rc, k - NS[NS[node].lc].sz - 1, x,
    root);
  }
}
// Erase root of tree
void erase(int &root) {
  if (root == 0) return;
  prop(root);
  int p = NS[root].lc, q = NS[root].rc;
  if (p == 0 || q == 0) {
    root = p + q;
    NS[root].par = 0;
    return;
  }
  root = p;
  NS[root].par = 0;
  find_kth(root, NS[p].sz, root);
  NS[root].rc = q;
  NS[q].par = root;
  recalc(root);
}
// Merge [l, r]th nodes into a subtree (maybe
NS[NS[root].lc].rc), and return it
int interval(int l, int r, int &root) {
  assert(1 <= l && r <= NS[root].sz);
  int sz = NS[root].sz, ret, x;
  if (r < sz) {
    find_kth(root, r + 1, root);
    x = root;
    root = NS[x].lc;
    NS[root].par = 0;
  }
  if (l > 1) {
    find_kth(root, l - 1, root);
    ret = NS[root].rc;
  }
  else ret = root;
  if (r < sz) {
    NS[root].par = x;
    NS[x].lc = root;
    root = x;
  }
  return ret;
}
// Update val to range [l, r]
void update(int l, int r, ll val, int &root) {
  assert(1 <= l && r <= NS[root].sz);
  int p = interval(l, r, root);
  apply(p, val);
  recalc(NS[p].par);
  recalc(NS[NS[p].par].par);
}
// Query range [l, r]
Node query(int l, int r, int &root) {
  assert(1 <= l && r <= NS[root].sz);
  int p = interval(l, r, root);
  return NS[p];
}
```

5.8 Link Cut Tree (6653B)

```
struct Node {
  int par, lc, rc, lc2, rc2, pt;
  ll val;
  int hsz, lsz;
  ll hsum, lsum;
  ll hlazy, llazy;
  bool rev;
  Node(ll x) {
    par = lc = rc = lc2 = rc2 = pt = 0;
    val = x;
    hsz = 1; lsz = 0;
    hsum = x; lsum = 0;
    hlazy = llazy = 0;
    rev = false;
  }
  Node() {
    par = lc = rc = lc2 = rc2 = pt = 0;
    val = 0;
    hsz = lsz = 0;
    hsum = lsum = 0;
    hlazy = llazy = 0;
    rev = false;
  }
};
Node NS[MAXN + 10];

void recalc(int node) {
  if (node == 0) return;
  assert(NS[node].hlazy == 0 && NS[node].llazy == 0 &&
  !NS[node].rev);
  int l = NS[node].lc, r = NS[node].rc, l2 = NS[node].lc2, r2 =
  NS[node].rc2, p = NS[node].pt;
  NS[node].hsz = NS[l].hsz + NS[r].hsz + 1;
  NS[node].lsz = NS[l].lsz + NS[r].lsz + (NS[l2].hsz +
  NS[l2].lsz) + (NS[r2].hsz + NS[r2].lsz) + (NS[p].hsz +
  NS[p].lsz);
  NS[node].hsum = NS[l].hsum + NS[r].hsum + NS[node].val;
  NS[node].lsum = NS[l].lsum + NS[r].lsum + (NS[l2].hsum +
  NS[l2].lsum) + (NS[r2].hsum + NS[r2].lsum) + (NS[p].hsum +
  NS[p].lsum);
}
void apply(int node, ll hupd, ll lupd, bool rev) {
  if (node == 0) return;
  NS[node].rev ^= rev;
  NS[node].hlazy += hupd;
  NS[node].llazy += lupd;
  if (rev) swap(NS[node].lc, NS[node].rc);
  NS[node].val += hupd;
  NS[node].hsum += hupd * NS[node].hsz;
  if (NS[node].lsz) NS[node].lsum += lupd * NS[node].lsz;
}
void prop(int node) {
  if (node == 0) return;
  apply(NS[node].lc, NS[node].hlazy, NS[node].llazy,
  NS[node].rev);
  apply(NS[node].rc, NS[node].hlazy, NS[node].llazy,
  NS[node].rev);
  apply(NS[node].lc2, NS[node].llazy, NS[node].llazy, false);
  apply(NS[node].rc2, NS[node].llazy, NS[node].llazy, false);
  apply(NS[node].pt, NS[node].llazy, NS[node].llazy, false);
  NS[node].hlazy = NS[node].llazy = 0;
  NS[node].rev = false;
}

bool isRoot(int x) { return (NS[x].par == 0 ||
(NS[NS[x].par].lc != x && NS[NS[x].par].rc != x)); }
bool isRoot2(int x) { return (NS[x].par == 0 ||
NS[NS[x].par].pt == x); }
void prop_anc(int x) { if (!isRoot(x)) prop_anc(NS[x].par);
prop(x); }
void prop_anc2(int x) { if (!isRoot2(x)) prop_anc2(NS[x].par);
prop(x); }
void prop_anc3(int x) { if (NS[x].par) prop_anc3(NS[x].par);
prop(x); }
void rotate(int x) {
  assert(x != 0 && !isRoot(x));
  int p = NS[x].par, q;
  if (NS[p].lc == x) {
    NS[p].lc = q = NS[x].rc;
    NS[x].rc = p;
  }
  else {
    NS[p].rc = q = NS[x].lc;
    NS[x].lc = p;
  }
  NS[x].par = NS[p].par;
  NS[p].par = x;
  if (q) NS[q].par = p;
  if (NS[x].par) {
    if (NS[NS[x].par].lc == p) NS[NS[x].par].lc = x;
    else if (NS[NS[x].par].rc == p) NS[NS[x].par].rc = x;
    else if (NS[NS[x].par].lc2 == p) NS[NS[x].par].lc2 = x;
    else if (NS[NS[x].par].rc2 == p) NS[NS[x].par].rc2 = x;
    else if (NS[NS[x].par].pt == p) NS[NS[x].par].pt = x;
  }
  recalc(p);
  recalc(x);
}
void rotate2(int x) {
  assert(x != 0 && !isRoot2(x));
  int p = NS[x].par, q;
  if (NS[p].lc2 == x) {
    NS[p].lc2 = q = NS[x].rc2;
    NS[x].rc2 = p;
  }
  else {
    NS[p].rc2 = q = NS[x].lc2;
    NS[x].lc2 = p;
  }
  NS[x].par = NS[p].par;
  NS[p].par = x;
  if (q) NS[q].par = p;
  if (NS[x].par) {
    if (NS[NS[x].par].lc == p) NS[NS[x].par].lc = x;
```

```
    else if (NS[NS[x].par].rc == p) NS[NS[x].par].rc = x;
    else if (NS[NS[x].par].lc2 == p) NS[NS[x].par].lc2 = x;
    else if (NS[NS[x].par].rc2 == p) NS[NS[x].par].rc2 = x;
    else if (NS[NS[x].par].pt == p) NS[NS[x].par].pt = x;
  }
  recalc(p);
  recalc(x);
}
void splay(int x) {
  if (x == 0) return;
  prop_anc(x);
  int root = x;
  while (!isRoot(root)) root = NS[root].par;
  int l2 = NS[root].lc2, r2 = NS[root].rc2;
  NS[root].lc2 = NS[root].rc2 = 0;
  while (!isRoot(x)) {
    int p = NS[x].par, q = NS[p].par;
    if (!isRoot(p)) rotate((NS[p].lc == x) == (NS[q].lc == p) ?
    p : x);
    rotate(x);
  }
  NS[x].lc2 = l2;
  NS[x].rc2 = r2;
  recalc(x);
}
void splay2(int x) {
  if (x == 0) return;
  prop_anc2(x);
  while (!isRoot2(x)) {
    int p = NS[x].par, q = NS[p].par;
    if (!isRoot2(p)) rotate2((NS[p].lc2 == x) == (NS[q].lc2 ==
    p) ? p : x);
    rotate2(x);
  }
}
int find_right(int x) {
  assert(isRoot2(x));
  while (NS[x].rc2 != 0) {
    prop(x);
    x = NS[x].rc2;
  }
  splay2(x);
  return x;
}
void erase(int x) {
  assert(isRoot2(x));
  prop(x);
  int p = NS[x].lc2, q = NS[x].rc2;
  NS[x].lc2 = 0;
  NS[x].rc2 = 0;
  recalc(x);
  if (p == 0 || q == 0) {
    int root = p + q;
    if (root) NS[root].par = NS[x].par;
    if (NS[x].par) NS[NS[x].par].pt = root;
    recalc(NS[root].par);
    return;
  }
  int root = p;
  NS[root].par = 0;
  root = find_right(root);
  NS[root].rc2 = q;
  NS[q].par = root;
  NS[root].par = NS[x].par;
  if (NS[x].par) NS[NS[x].par].pt = root;
  recalc(root);
  recalc(NS[root].par);
}
void insert(int x, int y) {
  prop(x);
  if (NS[x].pt == 0) {
    NS[x].pt = y;
    NS[y].par = x;
  } else {
    int x2 = NS[x].pt;
    x2 = find_right(x2);
    NS[x2].rc2 = y;
    NS[y].par = x2;
    recalc(x2);
  }
  recalc(x);
}
int access(int x) {
  prop_anc3(x);
  splay(x);
  if (NS[x].rc != 0) insert(x, NS[x].rc);
  NS[x].rc = 0;
  recalc(x);
  int ret = x;
  while (NS[x].par) {
    splay2(x);
    int y = NS[x].par;
    ret = y;
    splay(y);
    erase(x);
    if (NS[y].rc != 0) insert(y, NS[y].rc);
    NS[y].rc = x;
    NS[x].par = y;
    recalc(x);
    recalc(y);
    splay(x);
  }
  prop(x);
  return ret;
}
void link(int x, int p) {
  access(x);
  access(p);
  NS[x].lc = p;
  NS[p].par = x;
  recalc(x);
}
void cut(int x) {
  access(x);
  NS[NS[x].lc].par = 0;
  NS[x].lc = 0;
  recalc(x);
}
int lca(int x, int y) {
  access(x);
  return access(y);
}
void reroot(int x) {
  access(x);
  apply(x, 0, 0, true);
}
int findRoot(int x) {
  access(x);
  while (NS[x].lc) x = NS[x].lc;
  splay(x);
  return x;
}
int findParent(int x) {
  access(x);
  x = NS[x].lc;
  if (x == 0) return 0;
  while (NS[x].rc) x = NS[x].rc;
  splay(x);
  return x;
}
int depth(int x) {
  access(x);
  return NS[NS[x].lc].hsz + 1;
}
//========================================
ll subtree_query(int r, int v) {
  reroot(r);
  access(v);
  int p = NS[v].pt;
  return NS[v].val + NS[p].hsum + NS[p].lsum;
}
void subtree_update(int r, int v, ll k) {
  reroot(r);
  access(v);
  apply(NS[v].pt, k, k, false);
  NS[v].val += k;
  recalc(v);
}
ll path_query(int u, int v) {
  reroot(u);
  access(v);
  return NS[v].hsum;
}
void path_update(int u, int v, ll k) {
  reroot(u);
  access(v);
  apply(v, k, 0, false);
}
```

5.9 Queue Undo Trick

큐의 원소들을 **빨간색**과 **파란색**으로 나눠서 스택에 저장. 파란색 원소들은 스택의 top에서부터 뽑을 때 first-in-first-out 이고, 빨간색 원소들은 last-in-

first-out임. pop() 연산이 들어왔을 때 스택의 top이 빨간색이라면, 원소들을 적당히 뽑아서 위 불변량을 유지하면서 변환함. (pseudocode 참고)

```
// pseudocode, O(n log n) amortized time
void push(x) {  S.insert(x, red); }
int pop() {
    if (S.top().col == blue) return S.pop();
    else {
        vector<int> R, B;
        while (!S.empty() && (size(R) != size(B) and S has
        blue)) {
            if (S.top().col == red) R.insert(S.pop());
            else B.insert(S.pop());
        }
        if (S.empty()) {
            reverse(R);
            for (x in R) x.col = blue, S.insert(x);
            for (x in B) S.insert(x);
        }
        else {
            for (x in R) S.insert(x);
            for (x in B) S.insert(x);
        }
    }
}
```

6 Geometry

6.1 Geometry Template (int)

```
bool point_on_line(point p, point q, point r) {
  ll x = (r - p) * (q - p);
  return ccw(p, q, r) == 0 && 0 <= x && x <= (q - p) * (q - p);
}
// 교점이 없으면 0, 교점이 선분의 끝점이면 1, 두 선분이 교차하면 2,
무수히 많으면 3
int line_intersect_verbose(point p, point q, point r, point s)
{
  if (ccw(p, q, r) == 0 && ccw(p, q, s) == 0) {
    ll u = (r - p) * (q - p);
    ll v = (s - p) * (q - p);
    ll l = (q - p) * (q - p);
    if (u > v) swap(u, v);
    // intersection of [0, l] and [u, v]
    if (l < u || v < 0) return 0;
    else if (l == u || v == 0) return 1;
    else return 3;
  }
  if (point_on_line(r, s, p) || point_on_line(r, s, q)) return
  1;
  if (point_on_line(p, q, r) || point_on_line(p, q, s)) return
  1;
  return (ccw(p, q, r) * ccw(p, q, s) < 0 && ccw(r, s, p) *
  ccw(r, s, q) < 0) ? 2 : 0;
}
```

6.2 Geometry Template (double)

```
const double eps = 1e-9;
point unit(point a) { return 1 / sqrt(a * a) * a; }
int ccw(point p, point q, point r) {
  double x = (q - p) / (r - p);
  return (x > eps) - (x < -eps);
}
bool point_on_line(point p, point q, point r) {
  double x = (r - p) * (q - p);
  return ccw(p, q, r) == 0 && -eps < x && x < (q - p) * (q - p)
  + eps;
}
point intersect(point p, point u, point q, point v) {
  return p + (((q - p) / v) / (u / v)) * u;
}
// 교점이 없으면 0, 교점이 선분의 끝점이면 1, 두 선분이 교차하면 2,
무수히 많으면 3
int line_intersect_verbose(point p, point q, point r, point s,
point &a, point &b) {
  const double eps = 1e-6;
  if (ccw(p, q, r) == 0 && ccw(p, q, s) == 0) {
    double u = (r - p) * (q - p);
    double v = (s - p) * (q - p);
    double l = (q - p) * (q - p);
    if (u > v) swap(u, v);
    // intersection of [0, l] and [u, v]
    // (u의 scale) / (자료형의 표현범위)가 eps보다 작도록.
    // (좌표의 차이) * (벡터의 크기) 가 eps보다 크도록.
    if (l < u - eps || 0 > v + eps) return 0;
    else if (abs(u - l) < eps) { a = p; return 1; }
  else if (abs(0 - v) < eps) { a = q; return 1; }
  else {
      a = p + (max(0.0, u) / l) * (q - p);
      b = p + (min(l, v) / l) * (q - p);
      return 3;
    }
  }
  if (point_on_line(r, s, p)) { a = p; return 1; }
  if (point_on_line(r, s, q)) { a = q; return 1; }
  if (point_on_line(p, q, r)) { a = r; return 1; }
  if (point_on_line(p, q, s)) { a = s; return 1; }
  if (ccw(p, q, r) * ccw(p, q, s) < 0 && ccw(r, s, p) * ccw(r,
  s, q) < 0) {
    a = intersect(p, q - p, r, s - r);
    return 2;
  }
  return 0;
}
```

6.3 Rotating Calipus (390B)

```
/* include 4 basic operator from int template */

// a must be a convex hull, counterclockwise, no colinear
points
vector<pair<point, point>> rotating_calipus(vector<point> a) {
  point vec = a[0] - a[n - 1];
  int i = 0, pos = 0;
  while (vec / (a[pos + 1] - a[pos]) > 0) ++pos;
  vector<pair<point, point>> R;
  while (i < n) {
    R.push_back({a[i], a[pos]});
    if ((a[i + 1] - a[i]) / (a[(pos + 1) % n] - a[pos]) > 0) {
      pos = (pos + 1) % n;
    } else
      ++i;
  }
  return R;
}
```

6.4 Convex Polygon (1768B)

```
// include 4 basic operators, ccw, point_on_line from int
template

// call init() first, then call add_point()
// P should be a convex polygon, counterclokwise, no colinear,
at least three points

// point_in_polygon(p) tests if p is strictly contained in the
interior of P
// tangent(p, dir) returns -1 if p is strictly inside
// otherwise, returns the vertex number v
// P[v] is the nearest point encountered on the tangent
(half)line

// dir = 0 (clockwise), dir = 1 (counterclockwise)
// 다각형 반대방향으로 반직선을 긋고 그 방향으로 돌릴 때 처음으로
만나는 점
struct convex_polygon {
  int n;
  vector<point> P;
  void init() { n = 0; P.clear(); }
  void add_point(point p) { ++n; P.push_back(p); }
  bool sgn(point p) {
    int k = ccw(P[n - 1], P[0], p);
    return k == 0 ? (P[n - 1] - P[0]) * (p - P[0]) >= 0 : k >
    0;
  }
  int get_region(point p) {
    if (p.x == P[0].x && p.y == P[0].y) return 0;
    bool k = sgn(p);
    int ss = 1, ee = n - 1;
    while (ss < ee) {
      int mid = (ss + ee) / 2;
      point u = k ? P[0] : P[mid];
      point v = k ? P[mid] : P[0];
      if (ccw(u, v, p) <= 0) ee = mid;
      else ss = mid + 1;
    }
    return k ? ss : -ss;
  }
  bool point_in_polygon(point p) {
    int k = get_region(p);
    if (k <= 0) return false;
    if (ccw(P[0], P[k], p) == 0) {
      if (k == 1 || k == n - 1) return false;
      else return (P[0] - p) * (P[0] - p) < (P[0] - P[k]) *
      (P[0] - P[k]);
    }
    else return ccw(P[k - 1], P[k], p) > 0;
  }
  int tangent(point p, bool dir) {
    if (point_in_polygon(p)) {
      return -1;
    }
    int k = get_region(p);
    if (k == 0) return dir ? 1 : n - 1;
    if (k > 0) {
```

```
      if (p.x == P[k].x && p.y == P[k].y) return dir ? (k + 1)
      % n : k - 1;
      if (point_on_line(P[k], P[k - 1], p)) return dir ? k : k
      - 1;
      if (point_on_line(P[0], P[n - 1], p)) return dir ? 0 : n
      - 1;
    }
    int ss = dir ? (k > 0 ? k : 0) : (k > 0 ? 0 : -k);
    int ee = dir ? (k > 0 ? n : -k) : (k > 0 ? k : n);
    --ee;
    while (ss < ee) {
      int mid = (ss + ee) / 2;
      int k = ccw(p, P[mid], P[mid + 1]);
      if (dir) {
        if (k >= 0) ee = mid;
        else ss = mid + 1;
      }
      else {
        if (k > 0) ss = mid + 1;
        else ee = mid;
      }
    }
    return ss;
  }
};
```

6.5 Halfplane Intersection (1516B)

```
typedef __int128_t dll;
/* include 4 basic operators from int template */
// 0-based, O(n log n), coordinates less than 10^12
const int INF = 1e6 + 10; // 좌표범위보다 크게
struct line { point s, d;  };
bool bad(line p, line q, line r) {
  dll S = (q.s - p.s) / q.d;
  dll T = p.d / q.d;
  if (T < 0) S *= -1, T *= -1;
  p.s = p.s - r.s;
  dll x = T * p.s.x + S * p.d.x;
  dll y = T * p.s.y + S * p.d.y;
  return x * r.d.y - y * r.d.x > 0;
}

vector<line> V;
void init() {
  point P[4];
  P[0] = {-INF, -INF}; P[1] = {INF, -INF};
  P[2] = {INF, INF}; P[3] = {-INF, INF};
  V.clear();
  for (int t = 0; t < 4; t++) V.push_back({P[t], P[(t + 1) % 4]
  - P[t]});
}
void add_line(point s, point d) { V.push_back({s, d}); } //
시점, 벡터

vector<line> get_hpi() {
  auto sgn = [&](point a) { return a.y > 0 || (a.y == 0 && a.x
  > 0); };
  sort(V.begin(), V.end(), [&](auto p, auto q) { return
  (sgn(p.d) == sgn(q.d)) ? p.d / q.d > 0 : sgn(p.d); });
  int n = V.size();
  line dq[n];
  int r = 0, s = 0;
  for (int i = 0; i < V.size(); i++) {
    while (s - r >= 2 && bad(dq[s - 2], dq[s - 1], V[i])) --s;
    while (s - r >= 2 && bad(dq[r], dq[r + 1], V[i])) ++r;

    if (r != s && dq[s - 1].d / V[i].d == 0) {
      if (dq[s - 1].d * V[i].d > 0) {
        if (V[i].d / (V[i].s - dq[s - 1].s) > 0) dq[s - 1] =
        V[i];
      }
      else return vector<line>();
    }
    else dq[s++] = V[i];

    while (s - r >= 3 && bad(dq[s - 2], dq[s - 1], dq[r])) --s;
    if (s - r >= 2 && dq[s - 2].d / dq[s - 1].d <= 0) return
    vector<line>();
  }
  if (s - r < 3) return vector<line>();
  return vector<line>(dq + r, dq + s);
}
```

6.6 Bulldozer

```
// 0-based
// n = number of points
// points must be sorted by increasing (x[i], y[i])
// two points with same coordinates NOT allowed
// three points collinear is OK
int main() {
  int n;
  cin >> n;
  int x[n], y[n];  // 중복점 절대 안됨
  for (int i = 0; i < n; i++) cin >> x[i] >> y[i];
  int ord[n], rnk[n];
  iota(ord, ord + n, 0);
  sort(ord, ord + n, [&](int i, int j) { return x[i] == x[j] ?
  y[i] < y[j] : x[i] < x[j]; });
  for (int i = 0; i < n; i++) rnk[ord[i]] = i;
  vector<pii> bdz;
  for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++) bdz.push_back({ord[i],
    ord[j]});
  auto det = [&](pii l, pii r) {
    return 1ll * (x[l.ss] - x[l.ff]) * (y[r.ss] - y[r.ff]) -
    1ll * (y[l.ss] - y[l.ff]) * (x[r.ss] - x[r.ff]);
  };
  sort(bdz.begin(), bdz.end(), [&](pii l, pii r) {
    ll x = det(l, r);
    return x == 0 ? rnk[l.ff] == rnk[r.ff] ? rnk[l.ss] <
    rnk[r.ss] : rnk[l.ff] < rnk[r.ff] : x > 0;
  });
  for (int i = 0, p = 0; i < bdz.size(); i = p) {
    while (p < bdz.size() && det(bdz[i], bdz[p]) == 0) ++p;
    for (int j = i; j < p; j++) {
      auto [u, v] = bdz[j];
      assert(rnk[v] == rnk[u] + 1);
      // your code goes here
      swap(rnk[u], rnk[v]);
      swap(ord[rnk[u]], ord[rnk[v]]);
    }
  }
}
```

6.7 Shamos-Hoey (1754B)

```
/* include 4 basic operators, ccw, point_on_line, intersect
from int template here */

// call calc() to check if there exists an intersection
// O(n log n)
int sweep;
struct line {
  point s, e;
  bool operator<(const line &ot) const {
    // compare s.y + (e - s).y / (e - s).x * (sweep - s.x)
    return (__int128)(ot.e.x - ot.s.x) * (1ll * (e.y - s.y) *
    (sweep - s.x) + 1ll * s.y * (e.x - s.x)) <
           (__int128)(e.x - s.x) * (1ll * (ot.e.y - ot.s.y) *
           (sweep - ot.s.x) + 1ll * ot.s.y * (ot.e.x -
           ot.s.x));
  }
};
bool operator<(const line &l, const int &y) { return ccw(l.s,
l.e, point{sweep, y}) > 0; }
bool operator<(const int &y, const line &l) { return ccw(l.s,
l.e, point{sweep, y}) < 0; }
bool isect(const line &l, const line &r) { return
line_intersect(l.s, l.e, r.s, r.e); }

struct event {
  int x, type, idx;
  bool operator<(const event &ot) const { return x == ot.x ?
  type < ot.type : x < ot.x; }
};
vector<line> L;
vector<event> E;
void add_line(point s, point e) {
  if (s.x > e.x || (s.x == e.x && s.y > e.y)) swap(s, e);
  int idx = (int)L.size();
  L.push_back({s, e});
  if (s.x != e.x) {
    E.push_back({s.x, 0, idx});
    E.push_back({e.x, 2, idx});
  }
  else E.push_back({s.x, 1, idx});
}
bool calc() {
  sort(E.begin(), E.end());
  multiset<line, less<>> ST;
  for (auto &e : E) {
    sweep = e.x;
    line l = L[e.idx];
    if (e.type == 0) {
      auto it = ST.insert(l);
      if (it != ST.begin() && isect(*it, *prev(it))) return
      true;
```

```
    if (it != prev(ST.end()) && isect(*it, *next(it))) return
    true;
  }
 else if (e.type == 1) {
    if (ST.lower_bound(l.s.y) != ST.upper_bound(l.e.y))
    return true;
  }
 else {
    auto it = ST.lower_bound(l);
    if (it != ST.begin() && it != prev(ST.end())) {
      if (isect(*prev(it), *next(it))) return true;
    }
    ST.erase(it);
  }
 }
 return false;
}
```

6.8 Smallest Enclosing Circle (802B)

```
/* include 5 basic operators, ccw, intersect from double
template */
point rot(point p) { return {-p.y, p.x}; }
double dst(point p, point q) { return sqrt((p - q) * (p - q));
}
point center(point p, point q, point r) {
 point x = 0.5 * (p + q);
 point y = 0.5 * (q + r);
 return intersect(x, rot(q - p), y, rot(r - q));
}
// O(n) randomized
pair<point, double> min_enclosing_circle(vector<point> P) {
 mt19937 rnd(1557);
 shuffle(P.begin(), P.end(), rnd);
 point p = P[0];
 double r = 0;
 for (int i = 1; i < P.size(); i++) if (dst(p, P[i]) > r) {
  p = P[i]; r = 0;
  for (int j = 0; j < i; j++) if (dst(p, P[j]) > r) {
    p = 0.5 * (P[i] + P[j]);
    r = dst(p, P[j]);
    for (int k = 0; k < j; k++) if (dst(p, P[k]) > r) {
      p = center(P[i], P[j], P[k]);
      r = dst(p, P[k]);
    }
  }
 }
 return {p, r};
}
```

6.9 Voronoi Diagram (5768B)

```
/* include 5 basic operators, intersect from double template */
const double eps = 1e-9;
point rot(point p) { return {-p.y, p.x}; }
namespace voronoi {
 int dcmp(double x) { return x < -eps ? -1 : x > eps ? 1 : 0;
 }
 double pb_int(point l, point r, double sweepline) {
  if (dcmp(l.y - r.y) == 0) return (l.x + r.x) / 2.0;
  int sign = l.y < r.y ? -1 : 1;
  point v = intersect(l, r - l, point{0, sweepline}, point{1,
  0});
  point m = 0.5 * (l + r);
  double d1 = (m - v) * (m - v);
  double d2 = (m - l) * (m - l);
  return v.x + sign * sqrt(max(0.0, d1 - d2));
 }
 point circumcenter(point p, point q, point r) { return
 intersect(0.5 * (p + q), rot(p - q), 0.5 * (q + r), rot(q -
 r));  }
 struct Beachline {
  struct node {
    node() {}
    node(point p, int idx) : p(p), idx(idx), end(0), link{0,
    0}, par(0), prv(0), nxt(0) {}
    point p; int idx, 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) {
        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;
    node *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;
    point u = rot(cur->p - cur->prv->p);
    point v = rot(cur->nxt->p - cur->p);
    if (dcmp(u/v) != 1) return false;
    point p = circumcenter(cur->p, cur->prv->p, cur->nxt->p);
    double d = sqrt((p - cur->p) * (p - cur->p));
    next_sweep = p.y + d;
    return true;
  }
  node *lower_bound(double x) {
    node *cur = root;
    while (cur) {
      double l = cur->prv ? pb_int(cur->prv->p, cur->p,
      sweepline) : -1e30;
      double r = cur->nxt ? pb_int(cur->p, cur->nxt->p,
      sweepline) : 1e30;
      if (l - eps < x && x < r + eps) {
        splay(cur); return cur;
      }
      cur = cur->link[x > r];
    }
  }
 };
 using Bnode = Beachline::node;
 vector<Bnode*> arr;
 Bnode* new_node(point p, int idx) {
  arr.push_back(new Bnode(p, idx));
  return arr.back();
 }
 struct event {
  event(double sweep, int idx) : type(0), sweep(sweep),
  idx(idx) {}
  event(double sweep, Bnode *cur): type(1), sweep(sweep),
  prv(cur->prv->idx), cur(cur), nxt(cur->nxt->idx) {}
  int type, idx, prv, nxt;
  Bnode* cur;
  double sweep;
  bool operator>(const event &l) const {
    return sweep > l.sweep;
  }
 };
 void VoronoiDiagram(vector<point> &P, vector<point> &V,
 vector<pii> &E, vector<pii> &D) {
  Beachline bl = Beachline();
  priority_queue<event, vector<event>, greater<event>>
  events;
  auto add_edge = [&](int u, int v, int a, int b, Bnode* c1,
  Bnode *c2) {
    if (c1) c1->end = E.size() * 2;
    if (c2) c2->end = E.size() * 2 + 1;
    E.push_back({u, v});
```

```
    D.push_back({a, b});
  };
  auto write_edge = [&](int idx, int v) { (idx % 2 == 0 ?
  E[idx / 2].x : E[idx / 2].y) = v; };
  auto add_event = [&](Bnode *cur) {
    double nxt;
    if (bl.get_event(cur, nxt)) events.push({nxt, cur});
  };
  int n = P.size();
  sort(P.begin(), P.end(), [&](point p, point q) { return p.y
  == q.y ? p.x < q.x : p.y < q.y; });
  Bnode *prv = new_node(P[0], 0);
  bl.root = prv;
  for (int i = 1; i < n; i++) {
    if (dcmp(P[i].y - P[0].y) == 0) {
      add_edge(-1, -1, i - 1, i, 0, prv);
      Bnode *cur = new_node(P[i], i);
      bl.insert(cur, prv, 1);
      prv = cur;
    }
    else events.push({P[i].y, i});
  }
  while (!events.empty()) {
    event q = events.top(); events.pop();
    Bnode *prv, *cur, *nxt, *site;
    int v = V.size(), idx = q.idx;
    bl.sweepline = q.sweep;
    if (q.type == 0) {
      point p = P[idx];
      cur = bl.lower_bound(p.x);
      bl.insert(site = new_node(p, idx), cur, 0);
      bl.insert(prv = new_node(cur->p, 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;
      V.push_back(circumcenter(prv->p, nxt->p, cur->p));
      write_edge(prv->end, v);
      write_edge(cur->end, v);
      add_edge(v, -1, prv->idx, nxt->idx, 0, prv);
      bl.erase(cur);
      add_event(prv);
      add_event(nxt);
    }
  }
  arr.clear();
 }
}
```

7 Note

7.1 Unimodular Matrix

- Unimodular Matrix : 임의의 square submatrix의 determinant가 1, 0, -1 중 하나인 matrix
- A의 모든 row에 nonzero element는 +1이 최대 1개, -1이 최대 1개만 있다면 A는 Unimodular matrix이다. (역은 성립하지 않음)
- A가 Unimodular Matrix라면, $Ax \le b$ 꼴의 LP는 항상 정수해를 가진다. (역은 성립하지 않음)
- $\begin{pmatrix} A \\ d^T \end{pmatrix}$가 Unimodular Matrix라면, $Ax \le b$꼴의 LP에서 $d^T x = k$일때 $max c^T x$를 $f(k)$라 할 때, f가 convex하다. (역은 성립하지 않음)

7.2 Lyndon Decomposition

- S가 lyndon word (simple word)라는 것은 S가 S의 모든 proper suffix보다 사전순으로 작다는 의미이다. 즉, S의 suffix array를 구했을 때, 첫 번째 칸이 1(S 전체)이다.
- Example : "a", "ab", "aab", "abb", "abcd", "abac"
- S의 어떤 prefix와 suffix도 동일하지 않다.
- S는 S의 모든 cyclic shift보다 사전순으로 작다. (같을 수도 없다.)
- L_1, L_2가 lyndon word라면 $L_1 < L_2 \Leftrightarrow L_1 + L_2 < L_2 + L_1 \Leftrightarrow L_1 + L_2$ is lyndon word
- 문자열 S에 대하여 S의 lyndon decomposition은 S를 $w_1 w_2 \cdots w_k$로 쪼개, $w_1, w_2, \cdots, w_k$가 모두 lyndon word이며, $w_1 \ge w_2 \ge \cdots \ge w_k$를 만족하게 하는 분할이다.

7.3 Dilworth Theorem

Theorem. poset의 maximum antichain의 크기는 minimum chain cover와 같다.

- DAG의 모든 정점을 정점-disjoint한 최소 개수 path로 덮기 ⇒ 정점 분할 후 $|V|-$(이분 매칭의 크기)
- DAG에서 k개의 정점-disjoint한 path로 덮을 수 있는 최대 정점 개수 ⇒ 정점 분할 후 MCMF
- DAG의 모든 정점을 최소 개수 path로 덮기 ⇒ Transitive closure에서 정점 분할 후 $|V|-$ (이분 매칭의 크기) ⇒ Dilworth theorem에 의해 maximum antichain의 크기
- poset의 maximum antichain 찾기 ⇒ 정점 분할한 이분그래프의 vertex cover 찾고, in과 out 정점이 모두 vertex cover에 포함 안된 정점들 선택
- DAG의 모든 간선을 간선-disjoint한 최소 개수 path로 덮기 ⇒ $\sum_{v \in V} \max(d_{out}(v) - d_{in}(v), 0)$

7.4 Konig's Theorem

Theorem. 이분그래프에서 최대 매칭의 크기는 최소 정점 커버의 크기와 같다.

- 매칭 크기 = vertex cover 크기 = $|V|-$ independent set 크기
- Vertex cover 복원하기
  - 최대 매칭을 찾고, 매칭에 포함되지 않은 정점들의 nbr를 전부 선택
  - 이 과정에서 매칭에 포함된 정점들만 선택되고, 어떤 매칭 간선의 양쪽 끝점이 모두 선택되는 일이 없음
  - 매칭 간선 중 양쪽 끝점이 다 선택되지 않은 게 있다면 둘 중 아무거나 선택

7.5 Menger's theorem

- k- vertex connected graph : $k-1$개의 정점을 지워도 그래프가 connected.
- k- edge connected graph : $k-1$개의 간선을 지워도 그래프가 connected.
- A graph is k- edge connected iff every pair of vertices has k edge-disjoint paths in between.
- A graph is k- vertex connected iff every pair of vertices has k internally vertex-disjoint paths in between.
- Biconnected graph (2-vertex connected graph)에서 임의의 두 간선을 포함하는 단순 사이클이 존재한다.

7.6 Hall's Theorem, Tutte's Theorem

- **Hall's Theorem.** 이분 그래프 $G = (A \cup B, E)$에 완전 매칭이 존재할 필요충분조건은 $|A| = |B|$이고 임의의 $A' \subseteq A$에 대해, $|A'| \le |N(A')|$을 만족하는 것이다. 이때, $N(A')$은 A'의 어떤 정점과 인접한 B의 모든 정점을 나타낸다.
- **Tutte's Theorem.** 어떤 그래프 $G = (V, E)$에 완전 매칭이 존재하는 필요충분조건은 임의의 정점의 부분집합 $S \subseteq V$에 대해 G에서 S를 제외한 그래프에서 정점의 개수가 홀수인 연결 성분의 개수가 S의 크기를 넘지않는 것이다.

7.7 Erdos-Gallai Theorem

A sequence of non-negative integers $d_1 \ge d_2 \ge \cdots \ge d_n$ an be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \cdots + d_n$ is even and for $\forall 1 \le k \le n$

$$\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

7.8 Tutte Matrix

그래프 $G = (V, E)$가 주어졌을 때, 랜덤한 소수 p를 고르고 다음과 같은 $|V| \times |V|$ 행렬 T를 만들자.

$$T_{i,j} = \begin{cases} r_{i,j} & \text{if } (i,j) \in E \text{ and } i < j \\ -r_{j,i} & \text{if } (i,j) \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

$r_{i,j}$ 에 $[1, p-1]$ 사이의 랜덤 정수를 부여하고, Gaussian elimination을 통해서 (mod p 에서) $rank(T)$ 를 구하면, G 의 최대 매칭의 크기는 높은 확률로 $\frac{rank(T)}{2}$ 이다.

Theorem. G에 perfect matching이 있음과, $\det T \neq 0$임이 동치이다.

7.9 Kirchoff's Theorem

For a connected graph G,

$$L_{i,j} = \begin{cases} deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{is adjacent to} v_j \\ 0 & \text{otherwise} \end{cases}$$

Any cofactor (delete 1 row, 1 column from L, and take determinant) equals to the number of labeled spanning trees of this graph.

Let $\lambda_1, \lambda_2, \cdots \lambda_n$ be nonzero eigenvalues of L, $t(G) = \frac{1}{n}\lambda_1 \lambda_2 \cdots \lambda_n$.

7.10 LP Dual

Primal Problem	Dual Problem
Maximization	Minimization
Inequality constraint ($\le$)	Nonnegative variable ($\ge$)
Inequality constraint ($\ge$)	Nonpositive variable ($\le$)
Equality constraint ($=$)	Free variable
Nonnegative variable ($\ge$)	Inequality constraint ($\ge$)
Nonpositive variable ($\le$)	Inequality constraint ($\le$)
Free variable	Equality constraint ($=$)

$$\begin{aligned} \max_x \quad & c^\top x \quad \text{(maximization)} \\ \text{s.t.} \quad & Ax \le b \quad \text{(constraint } \le \text{)} \\ & x \ge 0 \quad \text{(variable } \ge \text{)} \end{aligned} \quad \overset{\text{dual}}{\Longleftrightarrow} \quad \begin{aligned} \min_\lambda \quad & b^\top \lambda \quad \text{(minimization)} \\ \text{s.t.} \quad & \lambda \ge 0 \quad \text{(variable } \ge \text{)} \\ & A^\top \lambda \ge c \quad \text{(constraint } \ge \text{)} \end{aligned}$$

$$\max_{x,y,z} \quad c^\top x + d^\top y + f^\top z$$
$$\text{s.t.} \quad Ax + By + Cz \le p$$
$$Dx + Ey + Fz \ge q$$
$$Gx + Hy + Jz = r$$
$$x \ge 0, \quad y \le 0, \quad z \text{ free}$$

$\overset{\text{dual}}{\iff}$

$$\min_{\lambda,\eta,\mu} \quad p^\top \lambda + q^\top \eta + r^\top \mu$$
$$\text{s.t.} \quad \lambda \ge 0, \quad \eta \le 0, \quad \mu \text{ free}$$
$$A^\top \lambda + D^\top \eta + G^\top \mu \ge c$$
$$B^\top \lambda + E^\top \eta + H^\top \mu \le d$$
$$C^\top \lambda + F^\top \eta + J^\top \mu = f$$

8 Misc

8.1 Enumerate quotients

```
// enumerating all possible floor(n / x) in decreasing order
for (ll l = 1; l <= n; ) {
  ll q = n / l, r = n / q;
  // floor(n / x) = q for x in [l, r]
  l = r + 1;
}
```

8.2 FastIO

```
struct fastio {
  char buf[1 << 25]; // size > input size
  int idx;
  fastio() : idx(0) {
    fread(buf, sizeof(buf), 1, stdin);
  }
  inline int read() {
    int x = 0, s = 1;
    int c = buf[idx++];
    while (c <= 32) c = buf[idx++];
    if (c == '-') s = -1, c = buf[idx++];
    while (c > 32) x = (x << 3) + (x << 1) + (c - '0'), c =
    buf[idx++];
    if (s < 0) x = -x;
    return x;
  }
}fi;
```

8.3 Barrett Reduction

```
// speeding up modulo operation for non-const MOD

/* PUT THIS IN GLOBAL */
typedef __uint128_t L;
ull m;

/* PUT THIS IN MAIN AFTER INPUT MOD*/
m = ull((L(1) << 64) / MOD);

// replace operator* of mint with this
mint operator*(int ot) {
  ull a = (ull)x * ot;
  ull q = (ull)((L(m) * a) >> 64);
  ull r = a - q * MOD;
  return r >= MOD ? r - MOD : r;
}
```

8.4 Stress test (DM)

```
mt19937 rnd(1557);
ll rng(ll s, ll e) { return uniform_int_distribution<ll>(s,
e)(rnd); }

void generator() {
  ofstream ouf("in");
  // generator
  // ouf << 1 << x << y << '\n';
  ouf.flush();
}
void compile() {
  system("g++ -o main main.cpp");
  system("g++ -o naive naive.cpp");
}
bool compare() {
  auto start = chrono::steady_clock::now();
  system("./main < in > out");
  auto end = chrono::steady_clock::now();
  system("./naive < in > ans");

  cout << "time : " << (end - start).count() / 1e6 << "ms\n\n";

  system("diff out ans > diff");
  ifstream inf ("diff");
  string S; getline(inf, S);
  return S == "";
}
int main() {
  compile();

  int tc = 0;
  while (1) {
    cout << "test case " << ++tc << '\n';

    generator();
    if (!compare() || tc == 5000) break;
  }
}
```

8.5 Highly Composite Numbers, Large Prime

< 10^k	number	divisors	2 3 5 71113171923293137
9	735134400	1344	6 3 2 1 1 1 1
12	963761198400	6720	6 4 2 1 1 1 1 1 1
15	866421317361600	26880	6 4 2 1 1 1 1 1 1 1 1
18	897612484786617600	103680	8 4 2 2 1 1 1 1 1 1 1 1

prime	primitive root
10^9+7	402
10^9+9	1407
10^18+3	4001
10^18+9	5001

Rolling Hash safe prime: 9223372036854771239

8.6 Lexicographically next bit permutation

```
int t = v | (v - 1);
return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
```

8.7 Partition Numbers, Stirling Numbers

- Partition Number

  $p(n)$: Number of ways of writing n as a sum of positive integers, ignoring order.

$$p(0) = 1, \; p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

- Stirling numbers of the first kind

  $c(n,k)$: Number of permutations of length n with k cycles

$$c(0,0) = 1, c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k) \quad (1)$$
$$\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\cdots(x+n-1) \quad (2)$$

- Stirling numbers of the second kind

  $S(n,k)$: Number of ways to partition a n-element set into k sets

$$S(n,1) = S(n,n) = 1 \quad (3)$$
$$S(n,k) = S(n-1,k-1) + kS(n-1,k) \quad (4)$$

8.8 pbds

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
// not multiset, use pair<int, int> for multiset
using ordered_set = tree<int, null_type, less<int>,
    rb_tree_tag, tree_order_statistics_node_update>;
int main() {
    ordered_set X;
    X.insert(1); X.insert(3); X.insert(5); X.insert(7);
    X.insert(9);
    cout << X.order_of_key(4) << '\n';   // 2
    cout << X.order_of_key(5) << '\n';   // 2
    cout << X.order_of_key(10) << '\n';  // 5
    cout << *X.find_by_order(0) << '\n'; // 1
    X.erase(7);
    cout << *X.find_by_order(3) << '\n'; // 9
}
```

8.9 Formulas

- **Pick's Theorem**. $(Area) = (Inside) + (Boundary)/2 - 1$
- **Green's Theorem**. Let C is positive, smooth, simple curve. D is region bounded by C.
  $\oint_C (Ldx + Mdy) = \iint_D (\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y})$
- **Mobius inversion**. $g(n) = \sum_{d|n} f(d)$ $f(n) = \sum_{d|n} g(d)\mu(\frac{n}{d})$
- **Zeta transform**. $z(f(x)) = \sum_{s' \subseteq s} f(s')$
- **Mobius transform**. $\mu(f(x)) = \sum_{s' \subseteq s} (-1)^{|s \setminus s'|} f(s')$
  $\mu = \sigma z \sigma$, $\mu = z^{-1}$ where σ is odd negation