Handbook USM

World Finals

Team: Team Name

University: UTFSM

Contents

General

▸ Templates

Template # fa7b2c

Description: Just the starting template code

```cpp
#include<bits/stdc++.h>

using namespace std;

using ll = long long;
using ld = long double;

const ll mod = 1e9 + 7;
const ll inf = 1e12;
const ld pi = acos(-1);

int main() {
  ios::sync_with_stdio(0); cin.tie(0);
  cout << fixed << setprecision(9);

  return 0;
}
```

Strings

▸ Pattern Matching

Rolling Hashing # df98cc

Description: Rolling hash for fast substring comparison, single hash function
Status: Tested

```cpp
template<class T>
struct rolling_hashing {
  int base, mod;
  vector<long long> p, H;
  int n;
  rolling_hashing(const T &s, int b, int m): base(b), mod(m), n(s.size()) {
    p.assign(n+1, 1);
    H.assign(n+1, 0);
    for (int i = 0; i < n; ++i) {
      H[i+1] = (H[i] * base + s[i]) % mod;
      p[i+1] = (p[i] * base) % mod;
    }
  }
  int get(int l, int r) {
    int res = (H[r+1] - H[l]*p[r-l+1]) % mod;
    if (res < 0) res += mod;
    return res;
  }
};
```

KMP # 90eb22

Description: Find occurrences of a pattern within given text, runs in $O(n+m)$, where n and m are the lengths of the text and pattern respectively.
Status: Tested on CSES

```cpp
// Memory-efficient string matching version
template<class T> struct KMP {
  T pattern; vector<int> lps;
  KMP(T &pat): pattern(pat) {
    lps.resize(pat.size(), 0);
    int len = 0, i = 1;
    while (i < pattern.size()) {
      if (pattern[i] == pattern[len])
        lps[i++] = ++len;
      else {
        if (len != 0) len = lps[len - 1];
        else lps[i++] = 0;
      }
    }
  }
  vector<int> search(T &text) {
    vector<int> matches;
    int i = 0, j = 0;
    while (i < text.size()) {
      if (pattern[j] == text[i]) {
        i++, j++;
        if (j == pattern.size()) {
          matches.push_back(i - j);
          j = lps[j - 1];
        }
      } else {
        if (j != 0) j = lps[j - 1];
        else i++;
      }
    }
    return matches;
  }
};

// Simple version
template<class T>
vector<int> prefix(T &S) {
  vector<int> P(S.size());
  P[0] = 0;
  for(int i = 1; i < S.size(); ++ i) {
    P[i] = P[i - 1];
    while(P[i] > 0 && S[P[i]] != S[i])
      P[i] = P[P[i] - 1];
    if(S[P[i]] == S[i])
      ++ P[i];
  }
  return P;
}
```

Z # b71846

Description: Z-algorithm for string matching, finds all occurrences in $O(n+m)$
Status: Tested

```cpp
struct Z {
  int n, m;
  vector<int> z;
  Z(string s) {
    n = s.size();
    z.assign(n, 0);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
      if (i <= r)
        z[i] = min(r - i + 1, z[i - l]);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]])
        ++z[i];
      if (i + z[i] - 1 > r)
        l = i, r = i + z[i] - 1;
    }
  }
  Z(string p, string t) {
    string s = p + "#" + t;
    n = p.size();
    m = t.size();
    z.assign(n + m + 1, 0);
    int l = 0, r = 0;
    for (int i = 1; i < n + m + 1; i++) {
      if (i <= r)
        z[i] = min(r - i + 1, z[i - l]);
      while (i + z[i] < n + m + 1 && s[z[i]] == s[i + z[i]])
        ++z[i];
      if (i + z[i] - 1 > r)
        l = i, r = i + z[i] - 1;
    }
  }
  void p_in_t(vector<int>& ans) {
    for (int i = n + 1; i < n + m + 1; i++) {
      if (z[i] == n)
        ans.push_back(i - n - 1);
    }
  }
};
```

Aho Corasick # 779b08

Description: Aho-Corasick algorithm for multiple pattern matching in text
Status: Tested

```cpp
struct AhoCorasick {
  enum {
    alpha = 26, first = 'a'
  }; // change this!
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end = -1, nmatches = 0;
    Node(int v) {
      memset(next, v, sizeof(next));
    }
  };
  vector < Node > N;
  vi backp;
  void insert(string & s, int j) {
    assert(!s.empty());
    int n = 0;
    for (char c: s) {
      int & m = N[n].next[c - first];
      if (m == -1) {
        n = m = sz(N);
        N.emplace_back(-1);
      } else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector < string > & pat): N(1, -1) {
    rep(i, 0, sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

    queue < int > q;
    for (q.push(0); !q.empty(); q.pop()) {
      int n = q.front(), prev = N[n].back;
      rep(i, 0, alpha) {
        int & ed = N[n].next[i], y = N[prev].next[i];
        if (ed == -1) ed = y;
        else {
          N[ed].back = y;
          (N[ed].end == -1 ? N[ed].end : backp[N[ed].start]) = N[y].end;
          N[ed].nmatches += N[y].nmatches;
```

```
          q.push(ed);
        }
      }
    }
  }
  vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    for (char c: word) {
      n = N[n].next[c - first];
      res.push_back(N[n].end);
      // count += N[n].nmatches;
    }
    return res;
  }
  vector < vi > findAll(vector < string > & pat, string word) {
    vi r = find(word);
    vector < vi > res(sz(word));
    rep(i, 0, sz(word)) {
      int ind = r[i];
      while (ind != -1) {
        res[i - sz(pat[ind]) + 1].push_back(ind);
        ind = backp[ind];
      }
    }
    return res;
  }
};
```

▸ **Advanced**

Suffix Array

 # 960aaf

Description: Suffix array construction with LCP array using radix sort, $O(n \log n)$
Status: Tested

```
using ll = long long;
struct SA {
  int n;
  vector<int> C, R, R_, sa, sa_, lcp;
  inline int gr(int i) { return i < n ? R[i] : 0; }
  void csort(int maxv, int k) {
    C.assign(maxv + 1, 0);
    for (int i = 0; i < n; i++) C[gr(i + k)]++;
    for (int i = 1; i < maxv + 1; i++) C[i] += C[i - 1];
    for (int i = n - 1; i >= 0; i--) sa_[--C[gr(sa[i] + k)]] = sa[i];
    sa.swap(sa_);
  }
  void getSA(vector<int>& s) {
    R = R_ = sa = sa_ = vector<int>(n);
    for (ll i = 0; i < n; i++) sa[i] = i;
    sort(sa.begin(), sa.end(), [&s](int i, int j) { return s[i] < s[j]; });
    int r = R[sa[0]] = 1;
    for (ll i = 1; i < n; i++) R[sa[i]] = (s[sa[i]] != s[sa[i - 1]]) ? ++r : r;
    for (int h = 1; h < n && r < n; h <<= 1) {
      csort(r, h);
      csort(r, 0);
      r = R_[sa[0]] = 1;
      for (int i = 1; i < n; i++) {
        if (R[sa[i]] != R[sa[i - 1]] || gr(sa[i] + h) != gr(sa[i - 1] + h)) r+
+;
        R_[sa[i]] = r;
      }
      R.swap(R_);
    }
  }
  void getLCP(vector<int> &s) {
    lcp.assign(n, 0);
    int k = 0;
    for (ll i = 0; i < n; i++) {
      int r = R[i] - 1;
      if (r == n - 1) {
        k = 0;
        continue;
      }
      int j = sa[r + 1];
      while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
      lcp[r] = k;
      if (k) k--;
    }
  }
  SA(vector<int> &s) {
    n = s.size();
    getSA(s);
    getLCP(s);
  }
};
```

Suffix Automaton

 # d735bb

Description: Suffix automaton (DAWG) for string processing, $O(n)$ construction
Status: Tested

```
struct SuffixAutomaton {
  struct state {
    int len, link;
    int next[26];
    state(int _len = 0, int _link = -1) : len(_len), link(_link) {
      memset(next, -1, sizeof(next));
    }
  };
  vector<state> st;
  int last;
  SuffixAutomaton() {}
  SuffixAutomaton(const string &s) { init(s); }
  inline int State(int len = 0, int link = -1) {
    st.emplace_back(len, link);
    return st.size() - 1;
  }
  void init(const string &s) {
    st.reserve(2 * s.size());
    last = State();
    for (char c : s)
      extend(c);
  }
  void extend(char _c) {
    int c = _c - 'a', cur = State(st[last].len + 1), P = last;
    while ((P != -1) && (st[P].next[c] == -1)) {
      st[P].next[c] = cur;
      P = st[P].link;
    }
    if (P == -1)
      st[cur].link = 0;
    else {
      int Q = st[P].next[c];
      if (st[P].len + 1 == st[Q].len)
        st[cur].link = Q;
      else {
        int C = State(st[P].len + 1, st[Q].link);
        copy(st[Q].next, st[Q].next + 26, st[C].next);
        while ((P != -1) && (st[P].next[c] == Q)) {
          st[P].next[c] = C;
          P = st[P].link;
        }
        st[Q].link = st[cur].link = C;
      }
    }
    last = cur;
  }
};
```

Manacher

 # 6abe01

Description: Find palindromes centered at i in $O(n)$
Status: Tested on CSES

```
template<class T>
struct manacher {
  vector<int> odd, even;
  T s; int n;
  manacher(T &s): s(s), n(s.size()) {
    odd.resize(n);
    even.resize(n);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
      int k = (i > r) ? 1 : min(odd[l+r-i], r-i+1);
      while (0 <= i-k and i+k < n and s[i-k] == s[i+k]) k++;
      odd[i] = k--;
      if (i+k > r) l = i-k, r = i+k;
    }
    for (int i = 0, l = 0, r = -1; i < n; i++) {
      int k = (i > r) ? 0 : min(even[l + r - i + 1], r - i + 1);
      while (0 <= i - k - 1 and i + k < n && s[i - k - 1] == s[i + k]) k++;
      even[i] = k--;
      if (i + k > r) l = i - k - 1, r = i + k;
    }
  }
  // Returns the longest palindrome centered at i
  pair<int, int> get(int i) {
    int o = 2 * odd[i] - 1;   // Normally centered (Is odd size)
    int e = 2 * even[i];      // Centered to the right
    if (o >= e)
      return {i - odd[i] + 1, i + odd[i] - 1};
    return {i - even[i], i + even[i] - 1};
  }
};
```

Trie

 # c8f186

Description: Trie (prefix tree) for string storage and prefix queries
Status: Tested

```
struct trie {
  vector <vector<ll>> tree;
  vector<ll> counts;
  trie() {
    tree.push_back(vector<ll>(26, -1));
    counts.push_back(0);
  };
  void insert(string &s) {
    ll i = 0, u = 0;
    while(i < s.size()){
        char c = s[i];
        if (tree[u][c - 'a'] == -1){
            ll pos = tree.size();
            tree.push_back(vector<ll>(26, -1));
            counts.push_back(0);
            tree[u][c - 'a'] = pos;
        }
        i++;
        u = tree[u][c - 'a'];
    }
    counts[u]++; // count final character
  }
  ll search(string &s){
```

```
    ll i = 0,u = 0;
    ll count_ends = 0;
    while(i < s.size()){
        char c = s[i];
        if (tree[u][c - 'a'] != -1){
            //If you need to know how many trailing characters it goes through,
uncomment.
            //count_ends += counts[u];
            u = tree[u][c - 'a'];
            i++;
        }else break;
    }
    if(i == s.size()) count_ends += counts[u];
    return count_ends;
  }
};
```

Min Rotation

184c43

Description: Finds the lexicographically smallest rotation of a string, runs on $O(n)$
Status: Tested on CSES

```
string minRotation(string &s) {
  int a = 0, N = s.size();
  string res = s; s += s;
  for (int b = 0; b < N; b++) {
    for (int k = 0; k < N; k++) {
      if (a + k == b || s[a + k] < s[b + k]) {
        b += max(0, k - 1); break;
      }
      if (s[a + k] > s[b + k]) {
        a = b; break;
      }
    }
  }
  rotate(res.begin(), res.begin() + a, res.end());
  return res;
}
```

Data Structures

▸ Sequences

Min Queue

4dc8b0

Description: Get the minumum element of the queue, runs on $O(1)$ per operation amortized. Can be modified to get the maximum just changing the function
Status: Tested on Maximum Subarray Sum II CSES

```
template <typename T>
struct min_queue {
  min_stack<T> in, out;
  void push(T x) { in.push(x); }
  bool empty() { return in.empty() and out.empty(); }
  int size() { return in.size() + out.size(); }
  void pop() {
    if (out.empty())
    for (;in.size(); in.pop()) out.push(in.top());
    out.pop();
  }
  T front() {
    if (!out.empty()) return out.top();
    for (;in.size(); in.pop()) out.push(in.top());
    return out.top();
  }
  T getmin() {
    if (in.empty()) return out.getmin();
    if (out.empty()) return in.getmin();
    return min(in.getmin(), out.getmin());
  }
};
```

Min Deque

36a398

Description: Get the minumum element of the deque, runs on $O(1)$ per operation amortized. Can be modified to get the maximum just changing the function
Status: Not tested

```
template<typename T>
struct min_deque {
  min_stack<T> l, r, t;
  void rebalance() {
    bool f = false;
    if (r.empty()) {f = true; l.swap(r);}
    int sz = r.size() / 2;
    while (sz--) {t.push(r.top()); r.pop();}
    while (!r.empty()) {l.push(r.top()); r.pop();}
    while (!t.empty()) {r.push(t.top()); t.pop();}
    if (f) l.swap(r);
  }
  int getmin() {
    if (l.empty()) return r.getmin();
    if (r.empty()) return l.getmin();
    return min(l.getmin(), r.getmin());
  }
  bool empty() {return l.empty() and r.empty();}
  int size() {return l.size() + r.size();}
  void push_front(int x) {l.push(x);}
  void push_back(int x) {r.push(x);}
  void pop_front() {if (l.empty()) rebalance(); l.pop();}
  void pop_back() {if (r.empty()) rebalance(); r.pop();}
  int front() {if (l.empty()) rebalance(); return l.top();}
  int back() {if (r.empty()) rebalance(); return r.top();}
  void swap(min_deque &x) {l.swap(x.l); r.swap(x.r);}
};
```

Wavelet Tree

96d72f

Description: Succint data structure to store strings in compressed space. It generalizes rank and select operations. Runs on $O(\log \sigma)$, where σ is the size of the alphabet
Status: Tested

```
template<typename T, int MAXLOG>
struct wavelet_tree {
  struct node {
    succinct_indexable_dictionary sid;
    node *ch[2];
    node() = default;
    node(size_t len) : sid(len + 1), ch{nullptr} {}
  };
  node *root;
  node *build(vector<T> &v, vector<T> &rbuff, int bit, int l, int r) {
    if (l >= r or bit == -1) return nullptr;
    node *_node = new node(r-l);
    ll left = 0, right = 0;
    for (ll k = l; k < r; k++) {
      if ((v[k] >> bit) & 1)
        rbuff[right++] = v[k], _node->sid.set(k - l);
      else v[l + left++] = v[k];
    }
    for (int k = 0; k < right; k++) v[l + left + k] = rbuff[k];
    _node->sid.build();
    _node->ch[0] = build(v, rbuff, bit - 1, l, l + left);
    _node->ch[1] = build(v, rbuff, bit - 1, l + left, r);
    return _node;
  }
  wavelet_tree() = default;
  wavelet_tree(vector<T> &v) {
    vector<T> rbuff(v.size());
    root = build(v, rbuff, MAXLOG - 1, 0, v.size());
  }
  int rank(node *t, int l, int r, const T&x, int level) {
    if (l >= r or t == nullptr) return 0;
    if (level == -1) return r-l;
    bool f = (x >> level) & 1;
    l = t->sid.rank(f, l), r = t->sid.rank(f, r);
    return rank(t->ch[f], l, r, x, level-1);
  }
  int rank(const T &x, int r) { return rank(root, 0, r, x, MAXLOG-1); }
  T kth(node *t, int l, int r, int k, int level) {
    if (l >= r || t == nullptr) return 0;
    int cnt = t->sid.rank(false, r) - t->sid.rank(false, l);
    bool f = cnt <= k;
    l = t->sid.rank(f, l), r = t->sid.rank(f, r);
    if (f) return kth(t->ch[f], l, r, k - cnt, level - 1) | (T(1) << level);
    return kth(t->ch[f], l, r, k, level - 1);
  }
  // k-th(0-indexed) smallest number in v[l,r]
  T kth_smallest(int l, int r, int k) {
    return kth(root, l, r+1, k, MAXLOG - 1);
  }
  // k-th(0-indexed) largest number in v[l,r]
  T kth_largest(int l, int r, int k) {
    return kth(l, r, r - l - k);
  }
  int range_freq(node *t, int l, int r, T upper, int level) {
    if (t == nullptr || l >= r) return 0;
    bool f = ((upper >> level) & 1);
    int ret = 0;
    if (f) ret += t->sid.rank(false, r) - t->sid.rank(false, l);
    l = t->sid.rank(f, l), r = t->sid.rank(f, r);
    return range_freq(t->ch[f], l, r, upper, level - 1) + ret;
  }
  // count i s.t. (l <= i < r) && (v[i] < upper)
  int range_freq(int l, int r, T upper) {
    return range_freq(root, l, r, upper, MAXLOG - 1);
  }
};
```

▸ Union Find

Union Find

8e1991

Description: Finds sets and merges elements, complexity $O(\alpha(n))$. $\alpha(10^{600}) \approx 4$
Status: Highly tested

```
struct union_find {
  vector<int> e;
  union_find(int n) { e.assign(n, -1); }
  int findSet (int x) {
    return (e[x] < 0 ? x : e[x] = findSet(e[x]));
  }
  bool sameSet (int x, int y) { return findSet(x) == findSet(y); }
  int size (int x) { return -e[findSet(x)]; }
  bool unionSet (int x, int y) {
    x = findSet(x), y = findSet(y);
    if (x == y) return 0;
    if (e[x] > e[y]) swap(x, y);
    e[x] += e[y], e[y] = x;
```

```
    return 1;
  }
};
```

Union Find Rollback

47ec45

Description: Same utility of normal union-find, but you can rollback the last union. Runs on $O(\alpha(n))$
Status: Highly tested

```
struct op{
  int v,u;
  int v_value,u_value;
  op(int _v,int _v_value,int _u,int _u_value):
v(_v),v_value(_v_value),u(_u),u_value(_u_value) {}
};
struct union_find_rb {
  vector<int> e;
  stack<op> ops;
  int comps;
  union_find_rb(){}
  union_find_rb(int n): comps(n) {e.assign(n, -1);}
  int findSet (int x) {
    return (e[x] < 0 ? x : findSet(e[x]));
  }
  bool sameSet (int x, int y) { return findSet(x) == findSet(y); }
  int size (int x) { return -e[findSet(x)]; }
  bool unionSet (int x, int y) {
    x = findSet(x), y = findSet(y);
    if (x == y) return 0;
    if (e[x] > e[y]) swap(x, y);
    ops.push(op(x,e[x],y,e[y])); comps--;
    e[x] += e[y], e[y] = x;
    return 1;
  }
  void rb(){
    if(ops.empty()) return;
    op last = ops.top(); ops.pop();
    e[last.v] = last.v_value;
    e[last.u] = last.u_value;
    comps++;
  }
};
```

▸ Range Queries

Sparse Table

015caa

Description: Range queries, precomputed $O(n \log n)$, query $O(1)$.
- Only supports queries of idempotent functions (min, max, gcd)
- Does not allow updates
Status: Highly tested

```
template <typename T, T m_(T, T)>
struct sparse_table {
  int n;
  vector<vector<T>> table;
  sparse_table() {}
  sparse_table(vector<T> &arr) {
    n = arr.size();
    int k = log2_floor(n) + 1;
    table.assign(n, vector<T>(k));
    for (int i = 0; i < n; i++)
      table[i][0] = arr[i];
    for (int j = 1; j < k; j++)
      for (int i = 0; i + (1 << j) <= n; i++)
        table[i][j] = m_(table[i][j-1], table[i+(1<<(j-1))][j-1]);
  }
  T query(int l, int r) {
    int k = log2_floor(r - l + 1);
    return m_(table[l][k], table[r-(1 << k)+1][k]);
  }
  int log2_floor(int n) { return n ? __builtin_clzll(1) - __builtin_clzll(n) :
-1; }
};
```

Segment Tree

913fb5

Description: Range queries, build $O(n)$, query and update $O(\log n)$, positions [0, n - 1]
Status: Highly tested

```
template<class T, T m_(T, T)> struct segment_tree{
  int n; vector<T> ST;
  segment_tree(){}
  segment_tree(vector<T> &a){
    n = a.size(); ST.resize(n << 1);
    for (int i=n;i<(n<<1);i++)ST[i]=a[i-n];
    for (int i=n-1;i>0;i--)ST[i]=m_(ST[i<<1],ST[i<<1|1]);
  }
  void update(int pos, T val){ // replace with val
    ST[pos += n] = val;
    for (pos >>= 1; pos > 0; pos >>= 1)
      ST[pos] = m_(ST[pos<<1], ST[pos<<1|1]);
  }
  T query(int l, int r){ // [l, r]
    T ansL, ansR; bool hasL = 0, hasR = 0;
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
      if (l & 1)
        ansL=(hasL?m_(ansL,ST[l++]):ST[l++]),hasL=1;
      if (r & 1)
        ansR=(hasR?m_(ST[--r],ansR):ST[--r]),hasR=1;
    }
    if (!hasL) return ansR; if (!hasR) return ansL;
    return m_(ansL, ansR);
  }
};
```

Segment Tree Lazy

0fe589

Description: Segment tree but with range updates, build $O(n)$, query and update $O(\log n)$
Status: Highly tested

```
template<
  class T1, // answer value stored on nodes
  class T2, // lazy update value stored on nodes
  T1 merge(T1, T1),
  void pushUpd(T2& /*parent*/, T2& /*child*/, int, int, int, int), // push
update value from a node to another. parent -> child
  void applyUpd(T2&, T1&, int, int)           // apply the update value of a
node to its answer value. upd -> ans
>
struct segment_tree_lazy{
  vector<T1> ST; vector<T2> lazy; vector<bool> upd;
  int n;
  void build(int i, int l, int r, vector<T1>&values){
    if (l == r){
      ST[i] = values[l];
      return;
    }
    build(i << 1, l, (l + r) >> 1, values);
    build(i << 1 | 1, (l + r) / 2 + 1, r, values);
    ST[i] = merge(ST[i << 1], ST[i << 1 | 1]);
  }
  segment_tree_lazy() {}
  segment_tree_lazy(vector<T1>&values){
    n = values.size(); ST.resize(n << 2 | 3);
    lazy.resize(n << 2 | 3); upd.resize(n << 2 | 3, false);
    build(1, 0, n - 1, values);
  }
  void push(int i, int l, int r){
    if (upd[i]){
      applyUpd(lazy[i], ST[i], l, r);
      if (l != r){
        pushUpd(lazy[i], lazy[i << 1], l, r, l, (l + r) / 2);
        pushUpd(lazy[i], lazy[i << 1 | 1], l, r, (l + r) / 2 + 1, r);
        upd[i << 1] = 1;
        upd[i << 1 | 1] = 1;
      }
      upd[i] = false;
      lazy[i] = T2();
    }
  }
  void update(int i, int l, int r, int a, int b, T2 &u){
    if (l >= a and r <= b){
      pushUpd(u, lazy[i], a, b, l, r);
      upd[i] = true;
    }
    push(i, l, r);
    if (l > b or r < a) return;
    if (l >= a and r <= b) return;
    update(i << 1, l, (l + r) >> 1, a, b, u);
    update(i << 1 | 1, (l + r) / 2 + 1, r, a, b, u);
    ST[i] = merge(ST[i << 1], ST[i << 1 | 1]);
  }
  void update(int a, int b, T2 u){
    if (a > b){
      update(0, b, u);
      update(a, n - 1, u);
      return ;
    }
    update(1, 0, n - 1, a, b, u);
  }
  T1 query(int i, int l, int r, int a, int b){
    push(i, l, r);
    if (a <= l and r <= b)
      return ST[i];
    int mid = (l + r) >> 1;
    if (mid < a)
      return query(i << 1 | 1, mid + 1, r, a, b);
    if (mid >= b)
      return query(i << 1, l, mid, a, b);
    return merge(query(i << 1, l, mid, a, b), query(i << 1 | 1, mid + 1, r, a,
b));
  }
  T1 query(int a, int b){
    if (a > b) return merge(query(a, n - 1), query(0, b));
    return query(1, 0, n - 1, a, b);
  }
};
```

Persistent Segment Tree

887787

Description: Segment tree but saves a new version of the tree for each update, build $O(n)$, query $O(\log n)$, new nodes for each update $O(\log n)$
Status: Tested on CSES

```
template<class T, T _m(T, T)>
struct persistent_segment_tree {
```

```
  vector<T> ST;
  vector<int> L, R;
  int n, rt;
  persistent_segment_tree(int n): ST(1, T()), L(1, 0), R(1, 0), n(n), rt(0) {}
  int new_node(T v, int l = 0, int r = 0) {
    int ks = ST.size();
    ST.push_back(v); L.push_back(l); R.push_back(r);
    return ks;
  }
  int update(int k, int l, int r, int p, T v) {
    int ks = new_node(ST[k], L[k], R[k]);
    if (l == r) {
      ST[ks] = v; return ks;
    }
    int m = (l + r) / 2, ps;
    if (p <= m) {
      ps = update(L[ks], l, m, p, v);
      L[ks] = ps;
    } else {
      ps = update(R[ks], m + 1, r, p, v);
      R[ks] = ps;
    }
    ST[ks] = _m(ST[L[ks]], ST[R[ks]]);
    return ks;
  }
  T query(int k, int l, int r, int a, int b) {
    if (l >= a and r <= b)
      return ST[k];
    int m = (l + r) / 2;
    if (b <= m)
      return query(L[k], l, m, a, b);
    if (a > m)
      return query(R[k], m + 1, r, a, b);
    return _m(query(L[k], l, m, a, b), query(R[k], m + 1, r, a, b));
  }
  int update(int k, int p, T v) {
    return rt = update(k, 0, n - 1, p, v);
  }
  int update(int p, T v) {
    return update(rt, p, v);
  }
  T query(int k, int a, int b) {
    return query(k, 0, n - 1, a, b);
  }
};
```

BIT

b586d7

Description: Range queries and updates, precomputed $O(n \log n)$, query/update $O(\log n)$
- Sum operation can be changed to xor

Status: Tested

```
struct fenwick_tree {
  vector<ll> bit; int n;
  fenwick_tree(int n): n(n) { bit.assign(n, 0); }
  fenwick_tree(vector<ll> &a): fenwick_tree(a.size()) {
    for (size_t i = 0; i < a.size(); i++)
      add(i, a[i]);
  }
  ll sum(int r) {
    ll ret = 0;
    for (; r >= 0; r = (r & (r + 1)) - 1)
      ret += bit[r];
    return ret;
  }
  ll sum(int l, int r) {
    return sum(r) - sum(l - 1);
  }
  void add(int idx, ll delta) {
    for (; idx < n; idx = idx | (idx + 1))
      bit[idx] += delta;
  }
};
```

Bit 2D

a9206c

Description: 2D Range queries, Runs on $O(\log n \cdot \log m)$ per operation

Status: Tested

```
struct fenwick_tree_2d {
  int N, M;
  vector<vector<ll>> BIT;
  fenwick_tree_2d(int N, int M): N(N), M(M) {
    BIT.assign(N + 1, vector<ll>(M + 1, 0));
  }
  void update(int x, int y, ll v) {
    for (int i = x; i <= N; i += (i & -i))
      for (int j = y; j <= M; j += (j & -j))
        BIT[i][j] += v;
  }
  ll sum(int x, int y) {
    ll s = 0;
    for (int i = x; i > 0; i -= (i & -i))
      for (int j = y; j > 0; j -= (j & -j))
        s += BIT[i][j];
    return s;
  }
  ll query(int x1, int y1, int x2, int y2) {
    return sum(x2, y2) - sum(x2, y1 - 1) - sum(x1 - 1, y2) + sum(x1 - 1, y1 -
1);
  }
};
```

Merge Sort Tree

4d309e

Description: Iterative merge sort tree. Each node stores sorted elements in its range.
- Query: Count elements > k in range [i, j]. O(log^2 n) per operation.
- Build: O(n log n)

Status: Distinct Values Queries (CSES)

```
template <typename T>
struct merge_sort_tree {
  int N; vector<vector<T>> ST;
  merge_sort_tree(){}
  merge_sort_tree(vector<T> &vs) {
    N = vs.size(); ST.resize(2 * N);
    for (int i = 0; i < N; i++) ST[i + N] = {vs[i]};
    for (int i = N - 1; i > 0; i--)
      merge(ST[i<<1].begin(), ST[i<<1].end(), ST[i<<1|1].begin(), ST[i<<1|
1].end(), back_inserter(ST[i]));
  }
  int query(int i, int j, int k) {
    int res = 0;
    for (i += N, j += N + 1; i < j; i >>= 1, j >>= 1) {
      if (i & 1) res += ST[i].size() - (upper_bound(ST[i].begin(), ST[i].end(),
k) - ST[i].begin()), i++;
      if (j & 1) --j, res += ST[j].size() - (upper_bound(ST[j].begin(),
ST[j].end(), k) - ST[j].begin());
    }
    return res;
  }
};
```

Line Container

c89a68

Description: Line container for convex hull trick optimization in DP

Status: Tested

```
const ll INF = 1e15;
struct Line {
  mutable ll a, b, c;
  bool operator<(Line r) const {
    return a < r.a;
  }
  bool operator<(ll x) const {
    return c < x;
  }
};
// dynamically insert `a*x + b` lines and query for maximum
// at any x all operations have complexity O(log N)
struct LineContainer: multiset < Line, less < >> {
  ll div(ll a, ll b) {
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x -> c = INF, 0;
    if (x -> a == y -> a) x -> c = x -> b > y -> b ? INF : -INF;
    else x -> c = div(y -> b - x -> b, x -> a - y -> a);
    return x -> c >= y -> c;
  }

  void add(ll a, ll b) {
    // a *= -1, b *= -1 // for min
    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) -> c >= y -> c) isect(x, erase(y));
  }

  ll query(ll x) {
    if (empty()) return -INF; // INF for min
    auto l = * lower_bound(x);
    return l.a * x + l.b;
    // return -l.a * x - l.b; // for min
  }
};
```

MO

2c642f

Description: Allow us to answer queries offline with sqrt decomposition

Status: Tested

```
struct query {
  ll l, r, i;
};
template<class T, typename U, class T2>
struct mo_algorithm {
  vector<U> ans;
  template<typename... Args>
  mo_algorithm(vector<T> &v, vector<query> &queries, Args... args) {
    T2 ds(args...);
    ll block_sz = sqrtl(v.size());
    ans.assign(queries.size(), -1);
    sort(queries.begin(), queries.end(), [&](auto &a, auto &b) {
      return a.l/block_sz != b.l/block_sz ? a.l/block_sz < b.l/block_sz : a.r <
b.r;
    });
    ll l = 0, r = -1;
```

```
    for (query q : queries) {
      while (l > q.l) ds.add(v[--l]);
      while (r < q.r) ds.add(v[++r]);
      while (l < q.l) ds.remove(v[l++]);
      while (r > q.r) ds.remove(v[r--]);
      ans[q.i] = ds.answer(q);
    }
  }
};
```

▸ Balanced Trees

Ordered Set

862f9a

Description: Built-in version of C++ of select and rank operations in sets
Status: Tested

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using oset = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
```

Treap

c949ed

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Status: Tested

```
const ll INF = 1e15;
struct treap {
  static mt19937_64 MT;
  struct node {
    node *left, *right; ll key, priority, value, max_value;
    node(ll k, ll v = 0) {
      left = right = NULL; key = k; priority = MT();
      max_value = value = v;
    }
  };
  ll value(node* T) { return T ? T->value : -INF; }
  ll max_value(node* T) { return T ? T->max_value : -INF; }
  void update(node* T) {
    T->max_value = max({T->value, max_value(T->left), max_value(T->right)});}
  node *root;
  void merge(node* &T, node* T1, node* T2) {
    if(T1 == NULL) { T = T2; return; }
    if(T2 == NULL) { T = T1; return; }
    if(T1->priority > T2->priority)
      merge(T1->right, T1->right, T2), T = T1;
    else merge(T2->left, T1, T2->left), T = T2;
    return update(T);
  }
  void merge(node* &T, node* T1, node* T2, node* T3) {
    merge(T, T1, T2); merge(T, T, T3);
  }
  void split(node* T, ll x, node* &T1, node* &T2) {
    if(T == NULL) { T1 = T2 = NULL; return; }
    if(T->key <= x) { split(T->right, x, T->right, T2); T1 = T; }
    else { split(T->left, x, T1, T->left); T2 = T; }
    return update(T);
  }
  void split(node* T, ll x, ll y, node* &T1, node* &T2, node* &T3) {
    split(T, x-1, T1, T2);  split(T2, y, T2, T3);
  }
  bool search(node* T, ll x) {
    if(T == NULL) return false; if(T->key == x) return true;
    if(x < T->key) return search(T->left, x);
    return search(T->right, x);
  }
  void insert(node* &T, node* n) {
    if(T == NULL) { T = n; return; }
    if(n->priority > T->priority) {
      split(T, n->key, n->left, n->right); T = n;
    } else if(n->key < T->key) insert(T->left, n);
    else insert(T->right, n);
    return update(T);
  }
  void erase(node* &T, ll x) {
    if(T == NULL) return;
    if(T->key == x) { merge(T, T->left, T->right); }
    else if(x < T->key) erase(T->left, x);
    else erase(T->right, x);
    return update(T);
  }
  bool set(node* T, ll k, ll v) {
    if(T == NULL) return false;
    bool found;
    if(T->key == k) T->value = k, found = true;
    else if(k < T->key) found = set(T->left, k, v);
    else found = set(T->right, k, v);
    if(found) update(T); return found;
  }
  node* find(node* T, ll k) {
    if(T == NULL) return NULL;
    if(T->key == k) return T;
    if(k < T->key) return find(T->left, k);
    return find(T->right, k);
  }
  treap() {root = NULL;}
  treap(ll x) {root = new node(x);}
  treap &merge(treap &O) {merge(root, root, O.root); return *this; }
  treap split(ll x) {treap ans; split(root, x, root, ans.root); return ans; }
  bool search(ll x) {return search(root, x); }
  void insert(ll x) {if(search(root, x)) return; return insert(root, new
node(x));}
  void erase(ll x) {return erase(root, x); }
  void set(ll k, ll v) {if(set(root, k, v)) return; insert(root, new node(k,
v));}
  ll operator[](ll k) {
    node* n = find(root, k);
    if(n == NULL) n = new node(k), insert(root, n); return n->value;
  }
  ll query(ll a, ll b) {
    node *T1, *T2, *T3; split(root, a, b, T1, T2, T3);
    ll ans = max_value(T2); merge(root, T1, T2, T3);
    return ans;
  }
};
mt19937_64 treap::MT(chrono::system_clock::now().time_since_epoch().count());
```

Implicit Treap

c24c46

Description: Acts like an array, allowing us to reverse, sum, remove, slice, order of, and remove in $O(\log n)$
Status: Tested

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct implicit_treap {
  static mt19937_64 MT;
  struct node{
    node *l = nullptr, *r = nullptr, *par = nullptr;
    ll sz = 1, p = MT(), v, sumv, lzsum = 0;
    bool lzflip = false;
    node(ll x=0) { sumv=v=x; }
  } *root = nullptr;
  gp_hash_table<int, node*> mp; // Mapping values to nodes for order_of
  implicit_treap(){root=nullptr;}
  implicit_treap(ll x){root=new node(x);}
  ll sz(node* T) { return T?T->sz:0; }
  ll sumv(node* T) { return T?T->sumv:0;};
  void upd(node* T){
    if (T->l) T->l->par = T;
    if (T->r) T->r->par = T;
    T->sumv=T->v+sumv(T->l)+sumv(T->r);
    T->sz=1+sz(T->l)+sz(T->r);
  }
  void push(node* T){
    if(T->lzsum){
      T->v+=T->lzsum;
      T->sumv+=T->sz*T->lzsum;
      if(T->l)T->l->lzsum+=T->lzsum;
      if(T->r)T->r->lzsum+=T->lzsum;
      T->lzsum=0;
    }
    if(T->lzflip){
      swap(T->l,T->r);
      if(T->l)T->l->lzflip^=1;
      if(T->r)T->r->lzflip^=1;
      T->lzflip=0;
    }
  }
  void merge(node*& T,node* L,node* R){
    if(!L||!R){T=L?L:R;return;}
    push(L);push(R);
    if(L->p>R->p) merge(L->r,L->r,R), T=L;
    else merge(R->l,L,R->l), T=R;
    upd(T);
  }
  void split(node* T,int k,node*& L,node*& R){
    if(!T){L=R=nullptr;return;}
    push(T);
    int key=sz(T->l);
    if(key<=k) split(T->r,k-key-1,T->r,R), L=T;
    else split(T->l,k,L,T->l), R=T;
    upd(T);
  }
  void set(node* T,int k,ll x){
    push(T);
    int key=sz(T->l);
    if(key==k)T->v=x;
    else if(k<key)set(T->l,k,x);
    else set(T->r,k-key-1,x);
    upd(T);
  }
  node* find(node* T,int k){
    push(T);
    int key=sz(T->l);
    if(key==k) return T;
    if(k<key) return find(T->l,k);
    return find(T->r,k-key-1);
  }
  int size(){return sz(root);}
  implicit_treap& merge(implicit_treap& O){
    merge(root,root,O.root);
    return *this;
  }
  implicit_treap split(int k){
    implicit_treap ans;
    split(root,k,root,ans.root);
    return ans;
  }
  void erase(int i,int j){
    node *T1,*T2,*T3;
    split(root,i-1,T1,T2), split(T2,j-i,T2,T3);
    merge(root,T1,T3);
  }
  void erase(int k){erase(k,k);}
  void set(int k,ll x){set(root,k,x);}
  ll operator[](int k){return find(root,k)->v;}
  ll query(int i,int j){
    node *T1,*T2,*T3;
    split(root,i-1,T1,T2), split(T2,j-i,T2,T3);
    ll ans=sumv(T2);
    merge(T2,T2,T3), merge(root,T1,T2);
    return ans;
  }
  void update(int i,int j,ll x){
    node *T1,*T2,*T3;
    split(root,i-1,T1,T2), split(T2,j-i,T2,T3);
    T2->lzsum+=x;
    merge(T2,T2,T3), merge(root,T1,T2);
  }
  void flip(int i,int j){
    node *T1,*T2,*T3;
    split(root,i-1,T1,T2), split(T2,j-i,T2,T3);
    T2->lzflip^=1;
    merge(T2,T2,T3), merge(root,T1,T2);
  }
  void insert(int i,ll x){
    node *T1,*T2;
    split(root,i-1,T1,T2);
    merge(T1,T1,new node(x)), merge(root,T1,T2);
  }
```

```
  void push_back(ll x){auto nn = new node(x); merge(root,root, nn); mp[x] = nn;}
  int order_of(int x) {
    node* t = mp[x];
    vector<node*> path;
    for (node* cur = t; cur; cur = cur->par)
      path.push_back(cur);
    reverse(path.begin(), path.end());
    for (node* n : path) push(n), upd(n);
    int idx = sz(t->l);
    node* cur = t;
    while (cur->par) {
      if (cur == cur->par->r)
        idx += sz(cur->par->l) + 1;
      cur = cur->par;
    }
    return idx;
  }
};
mt19937_64 implicit_treap::MT(chrono::system_clock::now().time_since_epoch().count());
```

LCT

e6d667

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree. Runs on $O(\log n)$ per operation.
Status: Tested

```
struct Node { // Splay tree. Root's pp contains tree's parent.
  Node *p = 0, *pp = 0, *c[2];
  bool flip = 0;
  Node() { c[0] = c[1] = 0; fix(); }
  void fix() {
    if (c[0]) c[0]->p = this;
    if (c[1]) c[1]->p = this;
    // (+ update sum of subtree elements etc. if wanted)
  }
  void pushFlip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
  }
  int up() { return p ? p->c[1] == this : -1; }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
      x->c[h] = y->c[h ^ 1];
      z->c[h ^ 1] = b ? x : this;
    }
    y->c[i ^ 1] = b ? this : x;
    fix(); x->fix(); y->fix();
    if (p) p->fix(); swap(pp, y->pp);
  }
  void splay() { /// Splay this up to the root. Always finishes without flip
set.
    for (pushFlip(); p;) {
      if (p->p) p->p->pushFlip();
      p->pushFlip(); pushFlip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node *first() { /// Return the min element of the subtree rooted at this,
splayed to the top.
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
  }
};
struct link_cut {
  vector<Node> node;
  link_cut(int N) : node(N) {}
  void link(int u, int v) { // add an edge (u, v)
    makeRoot(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top);
    x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) {
    Node *nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void makeRoot(Node *u) {
    access(u); u->splay();
    if (u->c[0]) {
      u->c[0]->p = 0; u->c[0]->flip ^= 1;
      u->c[0]->pp = u; u->c[0] = 0;
      u->fix();
    }
  }
  Node *access(Node *u) {
    u->splay();
    while (Node *pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0;
        pp->c[1]->pp = pp;
      }
      pp->c[1] = u;
      pp->fix(); u = pp;
    }
    return u;
  }
};
```

Maths

Number Theory

Extended Gcd

493f93

Description: Extended euclidean algorithm
Status: Not stress-tested, but it gave AC in problems

```
struct GCD_type { ll x, y, d; };
GCD_type ex_GCD(ll a, ll b){
  if (b == 0) return {1, 0, a};
  GCD_type pom = ex_GCD(b, a % b);
  return {pom.y, pom.x - a / b * pom.y, pom.d};
}
```

CRT

725c32

Description: Chinese remainder theorem
Status: Stress-tested

```
using lll = __int128_t;
ll crt(vector<ll> a, vector<ll> m) {
  int n = a.size();
  for (int i = 0; i < n; i++) {
    a[i] %= m[i];
    a[i] = a[i] < 0 ? a[i] + m[i] : a[i];
  }
  lll ans = a[0];
  lll M = m[0];
  for (int i = 1; i < n; i++) {
    auto pom = ex_GCD(M, m[i]);
    lll x1 = pom.x;
    lll d = pom.d;
    if ((a[i] - ans) % d != 0)
      return -1;
    ans = ans + x1 * (a[i] - ans) / d % (m[i] / d) * M;
    M = M * m[i] / d;
    ans %= M;
    ans = ans < 0 ? ans + M : ans;
    M = M / (lll)__gcd((ll)M, (ll)m[i]) * m[i];
  }
  return ans;
}
```

Prime Factors

d58001

Description: Get all prime factors of n, runs on $O(\sqrt{n})$
Status: Highly tested

```
vector<ll> primeFactors(ll n) {
  vector<ll> factors;
  for (ll i = 2; (i*i) <= n; i++) {
    while (n % i == 0) {
      factors.push_back(i);
      n /= i;
    }
  }
  if (n > 1) factors.push_back(n);
  return factors;
}
```

Erathostenes Sieve

3efaeb

Description: Get all prime numbers up to n, runs on $O(n\log(\log(n)))$
Status: Highly tested

```
struct eratosthenes_sieve {
  vector<ll> primes;
  vector<bool> isPrime;
  eratosthenes_sieve(ll n) {
    isPrime.resize(n + 1, true);
    isPrime[0] = isPrime[1] = false;
    for (ll i = 2; i <= n; i++) {
      if (isPrime[i]) {
        primes.push_back(i);
        for (ll j = i*i; j <= n; j += i)
          isPrime[j] = false;
      }
    }
  }
};
```

Euler Phi

ac936a

Description: Return $\varphi(n)$, runs on $O(\sqrt{n})$
Status: Not tested

```
ll phi(ll n) {
  ll result = n;
  for (ll i = 2; i*i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) n /= i;
        result -= result / i;
    }
  }
  if (n > 1)
    result -= result / n;
  return result;
}
```

Miller Rabin

d3ad25

Description: Detect if a number is prime or not in $O(\log^2(n))$, needs 128 bits binary pow
Status: Highly tested

```
bool miller_rabin(uint64_t n) {
  if (n <= 1) return false;
  auto check = [](uint64_t n, uint64_t a, uint64_t d, uint64_t s) {
    uint64_t x = binpow(a, d, n); // Usar binpow de 128bits
    if (x == 1 or x == n-1) return false;
    for (uint64_t r = 1; r < s; r++) {
      x = (__uint128_t)x*x % n;
      if (x == n-1) return false;
    }
    return true;
  };
  uint64_t r = 0, d = n - 1;
  while ((d & 1) == 0) d >>= 1, r++;
  for (int x : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
    if (x == n) return true;
    if (check(n, x, d, r)) return false;
  }
  return true;
}
```

Pollard Rho

90cc17

Description: Finds a non-trivial factor of a composite number n using Pollard's Rho algorithm.
Status: Partially tested

```
ll pollard_rho(ll n) {
  if((n&1)^1) return 2;
  ll x = 2, y=2, d=1;
  ll c = rand()%n+1;
  while(d==1) {
    x=(mulmod(x,x,n)+c)%n;
    y=(mulmod(y,y,n)+c)%n;
    y=(mulmod(y,y,n)+c)%n;
    if(x>=y)d=__gcd(x-y,n);
    else d=__gcd(y-x,n);
  }
  return d==n? pollard_rho(n):d;
}
```

Modular

Mulmod

c29f90

Description: Multiply two number fast, secure for number lesser than 2^{57}
Status: Partially tested

```
ll mulmod(ll a, ll b, ll m) {
  ll r=a*b-(ll)((ld)a*b/m+.5)*m;
  return r<0?r+m:r;
}
```

Binary Pow

6c139d

Description: Exponentiation by squares, computes $a^b \bmod m$ in $O(\log b)$
Status: Highly tested

```
ll binpow(ll a, ll b, ll m) {
  a %= m;
  ll res = 1;
  while (b > 0) {
    if (b & 1)
      res = (res * a) % m;
    a = (a * a) % m;
    b >>= 1;
  }
  return res;
}
```

Linear Algebra

Gauss

576480

Description: Matrix elimination, runs on $O(n^3)$
Status: Not tested

```
const double EPS = 1e-18;
const int INF = 2; // it doesn't actually have to be infinity or a big number
const int MOD = 1e9+7;
int gauss(vector<vector<ll>> a, vector<ll> & ans) {
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;

  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m && row < n; ++col) {
    int sel = row;
    for (int i = row; i < n; ++i)
      if (abs(a[i][col]) > abs(a[sel][col]))
        sel = i;
    if (!a[sel][col])
      continue;
    for (int i = col; i <= m; ++i)
      swap(a[sel][i], a[row][i]);
    where[col] = row;

    for (int i = 0; i < n; ++i)
      if (i != row) {
        ll c = (a[i][col] * binpow(a[row][col], MOD - 2, MOD)) % MOD;
        for (int j = col; j <= m; ++j)
          a[i][j] = ((a[i][j] - a[row][j] * c) % MOD + MOD) % MOD;
      }
    ++row;
  }

  ans.assign(m, 0);
  for (int i = 0; i < m; ++i)
    if (where[i] != -1)
      ans[i] = (a[where[i]][m] * binpow(a[where[i]][i], MOD - 2, MOD)) % MOD;
  for (int i = 0; i < m; ++i)
    if (where[i] == -1)
      return INF;
  return 1;
}
```

Xor Basis

bc3ab1

Description: Calculate the xor basis of a set of numbers.
Complexity:
- Insertion: O(d)
- Query if a number can be formed: O(d)
- Get maximum xor subset: O(d)
- Merging two bases: O(d^2)
Status: Partially tested

```
struct xor_basis {
  int d, sz;
  vector<long long> basis;
  xor_basis(int _d): d(_d), sz(0), basis(_d, 0) {}
  bool insert(long long mask) {
    for (int i = d - 1; i >= 0; --i) {
      if ((mask & (1LL << i)) == 0) continue;
      if (!basis[i]) {
        basis[i] = mask;
        ++sz;
        return true;
      }
      mask ^= basis[i];
    }
    return false;
  }
  bool can_make(long long x){
    for (int i = d - 1; i >= 0; --i) {
      if ((x & (1LL << i)) == 0) continue;
      if (!basis[i]) return false;
      x ^= basis[i];
    }
    return x == 0;
  }
  long long get_max(){
    long long res = 0;
    for (int i = d - 1; i >= 0; --i)
      if ((res ^ basis[i]) > res) res ^= basis[i];
    return res;
  }
  void merge(const xor_basis &o) {
    for (long long v : o.basis)
      if (v) insert(v);
  }
};
```

Matrix

a95ea4

Author: Carlos Lagos
Description: Matrix operations including multiplication and binary exponentiation
Status: Tested

```
template<class T>
vector<vector<T>> multWithoutMOD(vector<vector<T>> &a, vector<vector<T>> &b){
    int n = a.size(),m = b[0].size(),l = a[0].size();
    vector<vector<T>> ans(n,vector<T>(m,0));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            for(int k = 0; k < l; k++){
```

```
                ans[i][j] += a[i][k]*b[k][j];
            }
        }
    }
    return ans;
}

template<class T>
vector<vector<T>> mult(vector<vector<T>> a, vector<vector<T>> b, ll mod){
    int n = a.size(),m = b[0].size(),l = a[0].size();
    vector<vector<T>> ans(n,vector<T>(m,0));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            for(int k = 0; k < l; k++){
                T temp = ((ll)a[i][k]*b[k][j]) % mod;
                ans[i][j] = (ans[i][j] + temp) % mod;
            }
        }
    }
    return ans;
}

vector<vector<ll>> binpow(vector<vector<ll>> v,ll n,long long mod){
    ll dim = v.size(); vector<vector<ll>> ans(dim,vector<ll>(dim,0));
    for(ll i = 0; i < dim; i++) ans[i][i] = 1;
    while(n){
        if(n & 1) ans = mult(ans,v,mod);
        v = mult(v,v,mod);
        n = n >> 1;
    }
    return ans;
}
```

Simplex

Description: Optimizes linear function, based of linear restrictions, in $O(n^2)$
Status: Not tested

9f5446

```
template<class T> struct Simplex {
    T ans;
    vector<vector<T>> a;
    vector<T> b,c,d;
    void pivot(int ii, int jj){
        d[ii] = c[jj];
        T s1 = a[ii][jj];
        for (int i = 0; i < a[0].size(); i++)
            a[ii][i] /= s1;
        b[ii] /= s1;
        for (int i = 0; i < d.size(); i++){
            if (i == ii || a[i][jj] == 0) continue;
            T s2 = a[i][jj];
            for (int j = 0; j < a[0].size(); j++)
                a[i][j] -= s2*a[ii][j];
            b[i] -= s2*b[ii];
        }
    }
    bool next_point(){
        int idx = -1; T mx;
        for (int i = 0; i < (int)a[0].size(); i++){
            T z = 0;
            for (int j = 0; j < (int)d.size(); j++)
                z += a[j][i]*d[j];
            if (idx == -1 || mx < c[i]-z)
                mx = c[i]-z, idx = i;
        }
        if (mx > 0){
            int idx2 = -1; T mn;
            for (int i = 0; i < (int)b.size(); i++){
                if (a[i][idx] == 0 || b[i]/a[i][idx] <= 0) continue;
                if (idx2 == -1 || mn > b[i]/a[i][idx])
                    mn = b[i]/a[i][idx], idx2 = i;
            }
            if (idx2 == -1) return 0; // unbounded
            pivot(idx2,idx);
            return 1;
        }
        return 0;
    }
    Simplex(vector<vector<T>> & _a, vector<T> & _b, vector<T> & _c) :
a(_a),b(_b),c(_c){
        d.resize(b.size(),0);
        while (next_point());
        ans = 0;
        for (int i = 0; i < b.size(); i++)
            ans += b[i]*d[i];
    }
};
```

Polynomials

FFT

Description: FFT(a) computes $\widehat{f(k)} = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2, runs on $O(N \log N)$, where $N = |A| + |B|$
- For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back.
- Rounding is safe if $(\sum a_i^2 + \sum b_i^2) \log_2(N) < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs).
Status: Highly tested (josupo.jp)

00a05a

```
struct FFT {
  const long double PI = acos(-1);
  typedef long double d; // to double if too slow
  void fft(vector<complex<d>> &a) {
    int n = a.size(), L = 31 - __builtin_clz(n);
    vector<complex<d>> R(2, 1), rt(2, 1);
    for (int k = 2; k < n; k *= 2) {
      R.resize(n); rt.resize(n);
      auto x = polar(1.0L, PI / k);
      for(int i = k; i < 2*k; ++i) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i /
2];
    }
    vector<int> rev(n);
    for(int i = 0; i < n; ++i) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    for(int i = 0; i < n; ++i) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k)
    for(int j = 0; j < k; ++j) {
      auto x = (d*)&rt[j + k], y = (d*)&a[i + j + k];
      complex<d> z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
      a[i + j + k] = a[i + j] - z, a[i + j] += z;
    }
  }
  vector<int> conv(vector<d> &a, vector<d> &b) {
    if (a.empty() || b.empty()) return {};
    vector<d> res(a.size() + b.size() - 1);
    int B = 32 - __builtin_clz(res.size()), n = 1 << B;
    vector<complex<d>> in(n), out(n);
    copy(a.begin(), a.end(), in.begin());
    for(int i = 0; i < b.size(); ++i) in[i].imag(b[i]);
    fft(in); for (auto &x : in) x *= x;
    for(int i = 0; i < n; ++i) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out); for(int i = 0; i < res.size(); ++i) res[i] = imag(out[i]) / (4 *
n);
    vector<int> resint(res.size());
    for (int i = 0; i < res.size(); i++) resint[i] = round(res[i]);
    return resint;
  }
```

```
  vector<ll> convMod(vector<ll> &a, vector<ll> &b, ll mod) {
    if (a.empty() || b.empty()) return {};
    vector<d> res(a.size() + b.size() - 1);
    int B = 32 - __builtin_clz(res.size()), n = 1 << B;
    ll cut = (ll)sqrt(mod);
    vector<complex<d>> L(n), R(n), outs(n), outl(n);
    for (int i = 0; i < (int)a.size(); i++) L[i] = complex<d>(a[i]/cut,
a[i]%cut);
    for (int i = 0; i < (int)b.size(); i++) R[i] = complex<d>(b[i]/cut,
b[i]%cut);
    fft(L), fft(R);
    for (int i = 0; i < n; i++) {
      int j = -i & (n-1);
      outl[j] = (L[i] + conj(L[j])) * R[i] / ((d)2.0 * n);
      outs[j] = (L[i] - conj(L[j])) * R[i] / ((d)2.0 * n) / complex<d>(0, 1);
    }
    fft(outl), fft(outs);
    for (int i = 0; i < (int)res.size(); i++) {
      ll av = (ll)(real(outl[i])+.5), cv = (ll)(imag(outs[i])+.5);
      ll bv = (ll)(imag(outl[i])+.5) + (ll)(real(outs[i])+.5);
      res[i] = ((av % mod * cut + bv) % mod * cut + cv) % mod;
    }
    vector<ll> resll(res.size());
    for (int i = 0; i < (int)res.size(); i++) resll[i] = (ll)round(res[i]);
    return resll;
  }
};
```

NTT

Description: Same utility as FFT but with some magic primes, runs in $O(n \log n)$ with better constant
Posible primes and their roots:
- 998244353, 3
- 9223372036737335297, 3
Status: Tested only using 998244353 as mod

fed936

```
template<int mod, int root>
struct NTT {
  void ntt(ll* x, ll* temp, ll* roots, int N, int skip) {
    if (N == 1) return;
    int n2 = N/2;
    ntt(x, temp, roots, n2, skip*2);
    ntt(x+skip, temp, roots, n2, skip*2);
    for (int i = 0; i < N; i++) temp[i] = x[i*skip];
    for (int i = 0; i < n2; i++) {
      ll s = temp[2*i], t = temp[2*i+1] * roots[skip*i] % mod;
      x[skip*i] = (s + t) % mod;
      x[skip*(i+n2)] = ((s - t) % mod + mod) % mod;
    }
  }
  void ntt(vector<ll>& x, bool inv = false) {
    ll e = binpow(root, (mod-1)/((int)x.size()), mod);
    if (inv) e = binpow(e, mod-2, mod);
    vector<ll> roots(x.size(), 1), temp = roots;
    for (int i = 1; i < (int)x.size(); i++) roots[i] = roots[i-1] * e % mod;
    ntt(&x[0], &temp[0], &roots[0], x.size(), 1);
  }
  vector<ll> conv(vector<ll> a, vector<ll> b) {
    int s = a.size()+b.size()-1;
    if (s <= 0) return {};
    int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1 << L;
    a.resize(n); ntt(a);
    b.resize(n); ntt(b);
    vector<ll> c(n); ll d = binpow(n, mod-2, mod);
    for (int i = 0; i < n; i++) c[i] = a[i] * b[i] % mod * d % mod;
    ntt(c, true); c.resize(s);
    for (int i = 0; i < n; i++) if(c[i] < 0) c[i] += mod;
```

```
    return c;
  }
};
```

Lagrange Point

0ebfc4

Description: Using points evaluated at 0,1,..n from f(n), interpolates f(x).
Time complexity O(k), where k is the amount of points
Status: Tested on codeforces

```
ll lagrange_point(vector<ll> & p, ll x, ll mod){
    int n = p.size();
    if (x < n) return p[x];
    vector<ll> pref(n+1,1), suff(n+1,1);
    for (int i = 1; i <= n; i++) pref[i] = (pref[i-1]*(x-(i-1)))%mod;
    for (int i = n-1; i >= 0; i--) suff[i] = (suff[i+1]*(x-i))%mod;
    vector<ll> ifact(n);
    ll fact = 1;
    for (ll i = 1; i < n; i++) fact = (fact*i)%mod;
    ifact[n-1] = binpow(fact,mod-2);
    for (ll i = n-2; i >= 0; i--) ifact[i] = (ifact[i+1]*(i+1))%mod;
    ll res = 0;
    for (int i = 1; i <= n; i++){
        ll val = (pref[i-1]*suff[i])%mod;
        ll temp = (((n-i)&1)?mod-1:1);
        val = (val*temp)%mod;
        val = (val*ifact[i-1])%mod;
        val = (val*ifact[n-i])%mod;
        val = (val*p[i-1])%mod;
        res = (res+val)%mod;
    }
    if (res < 0) res += mod;
    return res;
}
```

Combinatorics

Factorial

7e9435

Description: Calculate factorials from 1 to n and their factorial, runs on $O(n)$
Status: Highly tested

```
struct Factorial {
  vector<ll> f, finv, inv; ll mod;
  Factorial(ll n, ll mod): mod(mod) {
    f.assign(n+1, 1); inv.assign(n+1, 1); finv.assign(n+1, 1);
    for(ll i = 2; i <= n; ++i)
      inv[i] = mod - (mod/i) * inv[mod%i] % mod;
    for (ll i = 1; i <= n; ++i) {
      f[i] = (f[i-1] * i) % mod;
      finv[i] = (finv[i-1] * inv[i]) % mod;
    }
  }
};
```

Graphs

MST

Kruskal

caa8da

Description: Minimum spanning tree in $O(E \log E)$
Status: Tested

```
struct Edge {
  int a; int b; ll w;
  Edge(int a_, int b_, ll w_) : a(a_), b(b_), w(w_) {}
};
bool c_edge(Edge &a, Edge &b) { return a.w < b.w; }
ll Kruskal() {
  int n = G.size();
  union_find sets(n);
  vector<Edge> edges;
  for(int i = 0; i < n; i++) {
    for(auto eg : G[i]) {
      // node i to node eg.first with cost eg.second
      edges.emplace_back(i, eg.first, eg.second);
    }
  }
  sort(edges.begin(), edges.end(), c_edge);
  ll min_cost = 0;
  for(Edge e : edges) {
    if(sets.sameSet(e.a, e.b) != true) {
      tree.emplace_back(e.a, e.b, e.w);
      min_cost += e.w;
      sets.unionSet(e.a, e.b);
    }
  }
  return min_cost;
}
```

Flow Matching

Dinic

b2efb3

Author: Pablo Messina
Source: https://github.com/PabloMessina/Competitive-Programming-Material
Description: Flow algorithm with complexity $O(|E| \cdot |V|^2)$
Status: Not tested

```
struct Dinic {
  struct Edge { ll to, rev; ll f, c; };
  ll n, t_; vector<vector<Edge>> G;
  vector<ll> D, q, W;
  bool bfs(ll s, ll t) {
    W.assign(n, 0); D.assign(n, -1); D[s] = 0;
    ll f = 0, l = 0; q[l++] = s;
    while (f < l) {
      ll u = q[f++];
      for (const Edge &e : G[u]) if (D[e.to] == -1 && e.f < e.c)
        D[e.to] = D[u] + 1, q[l++] = e.to;
    }
    return D[t] != -1;
  }
  ll dfs(ll u, ll f) {
    if (u == t_) return f;
    for (ll &i = W[u]; i < (ll)G[u].size(); ++i) {
      Edge &e = G[u][i]; ll v = e.to;
      if (e.c <= e.f || D[v] != D[u] + 1) continue;
      ll df = dfs(v, min(f, e.c - e.f));
      if (df > 0) { e.f += df, G[v][e.rev].f -= df; return df; }
    }
    return 0;
  }
  Dinic(ll N) : n(N), G(N), D(N), q(N) {}
  void add_edge(ll u, ll v, ll cap) {
    G[u].push_back({v, (ll)G[v].size(), 0, cap});
    G[v].push_back({u, (ll)G[u].size() - 1, 0, 0}); // Use cap instead of 0 if
bidirectional
  }
  ll max_flow(ll s, ll t) {
    t_ = t; ll ans = 0;
    while (bfs(s, t)) while (ll dl = dfs(s, LLONG_MAX)) ans += dl;
    return ans;
  }
};
```

Hopcroft Karp

2e26b3

Author: Abner Vidal
Description: Computes maximal matching in $O\left(|E| \cdot \sqrt{|V|}\right)$, faster than Dinic.
Status: Tested on CSES

```
struct hopcroft_karp {
  const int INF = 1e9;
  int n; vector<int> l, r, d, ptr, g_edges, g_start, q;
  int q_h, q_t;
  hopcroft_karp(int _n, const vector<vector<int>>& adj) : n(_n) {
    l.assign(2*n+1, 0); r.assign(2*n+1, 0); d.assign(2*n+1, 0);
ptr.assign(2*n+1, 0);
    g_start.resize(n+2); q.resize(n+10); q_h = q_t = 0;
    for (int u = 1; u <= n; u++) {
      g_start[u] = g_edges.size();
      for (int b : adj[u]) g_edges.push_back(b+n);
    }
    g_start[n+1] = g_edges.size();
  }
  bool bfs() {
    q_h = q_t = 0;
    for (int u = 1; u <= n; u++) {
      if (!l[u]) { d[u] = 0; q[q_t++] = u; }
      else d[u] = INF;
    }
    d[0] = INF;
    while (q_h < q_t) {
      int u = q[q_h++];
      for (int i = g_start[u]; i < g_start[u+1]; i++) {
        int v = g_edges[i], nxt = r[v];
        if (d[nxt] == INF) {
          d[nxt] = d[u]+1;
          if (nxt) q[q_t++] = nxt;
          else d[0] = d[u]+1;
        }
      }
    }
    return d[0] != INF;
  }
  bool dfs(int u) {
    vector<pair<int,int>> st;
    int i = ptr[u];
    int u_start = g_start[u];
    int u_end = g_start[u+1];
    while (true) {
      if (i < u_end-u_start) {
        int v = g_edges[u_start+i];
        int nxt = r[v];
        if (!nxt) {
          l[u] = v; r[v] = u; ptr[u] = i+1;
          while (!st.empty()) {
            auto [pu, pi] = st.back(); st.pop_back();
            int pv = g_edges[g_start[pu]+pi];
            l[pu] = pv; r[pv] = pu; ptr[pu] = pi+1;
          }
          return true;
        } else if (d[nxt] == d[u]+1) {
          st.push_back({u, i}); u = nxt; i = ptr[u];
          u_start = g_start[u]; u_end = g_start[u+1];
        } else i++;
```

```
      } else {
        d[u] = INF;
        if (st.empty()) return false;
        auto [pu, pi] = st.back(); st.pop_back();
        u = pu; i = pi+1; ptr[u] = i;
        u_start = g_start[u]; u_end = g_start[u+1];
      }
    }
  }
  int maximum_matching(int want) {
    int match = 0;
    for (int u = 1; u <= n && match < want; u++) {
      if (!l[u]) {
        for (int i = g_start[u]; i < g_start[u+1]; i++) {
          int v = g_edges[i];
          if (!r[v]) { l[u] = v; r[v] = u; match++; break; }
        }
      }
    }
    while (match < want && bfs()) {
      for (int u = 1; u <= n; u++) ptr[u] = 0;
      for (int u = 1; u <= n && match < want; u++)
      if (!l[u] && dfs(u)) match++;
    }
    return match;
  }
};
```

Hungarian

64d965

Description: Solves assignament problem in $O(n^3)$. If the matrix is rectangular in $O(n^2m)$, where $n \leq m$
Status: Tested

```
void Hungarian(vector<vector<ll>> &A, vector<pair<int, int>> &result, ll &C,
const ll INF = 1e15) {
  int n = A.size() - 1, m = A[0].size() - 1;
  vector<ll> minv(m + 1), u(n + 1), v(m + 1);
  vector<int> p(m + 1), way(m + 1);
  vector<bool> used(m + 1);
  for (int i = 1; i <= n; ++i) {
    p[0] = i; int j0 = 0;
    for (int j = 0; j <= m; ++j)
      minv[j] = INF;
    for (int j = 0; j <= m; ++j)
      used[j] = false;
    do {
      used[j0] = true;
      int i0 = p[j0], j1;
      ll delta = INF;
      for (int j = 1; j <= m; ++j)
        if (!used[j]) {
          ll cur = A[i0][j] - u[i0] - v[j];
          if (cur < minv[j]) minv[j] = cur, way[j] = j0;
          if (minv[j] < delta) delta = minv[j], j1 = j;
        }
      for (int j = 0; j <= m; ++j) {
        if (used[j]) u[p[j]] += delta, v[j] -= delta;
        else minv[j] -= delta;
      }
      j0 = j1;
    } while (p[j0] != 0);
    do {
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    } while(j0);
  }
  for (int i = 1; i <= m; ++i)
    result.push_back(make_pair(p[i], i));
  C = -v[0];
}
```

Min Cost Flow

ebc841

Author: Abner Vidal
Description: Minimun cost flow with complexity $O(F(|E||V|\log(|E|)))$
Status: Tested on CSES

```
template<class T1, class T2>
struct min_cost_flow {
  struct edge {int v; T1 c,f; T2 w; };
  vector<vector<int>> g; vector<T2> dist,pot; vector<T1> fl;
  vector<bool> vis; vector<int> par; vector<edge> e;
  min_cost_flow(int n):g(n),dist(n),pot(n),fl(n),vis(n),par(n){}
  void add_edge(int u, int v, T1 c, T2 w){
    int k = e.size();
    e.push_back({v,c,0,w}); e.push_back({u,c,c,-w});
    g[u].push_back(k); g[v].push_back(k^1);
  }
  using pli = pair<T2,int>;
  void dijk(int s){
    fill(dist.begin(),dist.end(),numeric_limits<T2>::max());
    fill(vis.begin(),vis.end(),0); fl[s] = numeric_limits<T1>::max();
    priority_queue<pli,vector<pli>,greater<pli>> q;
    q.push({0,s}); dist[s] = 0;
    while (!q.empty()){
      pli p = q.top(); q.pop(); int x = p.second;
      if (vis[x]) continue; vis[x] = 1;
      for (int to : g[x]){
        T2 d2 = p.first+e[to].w+pot[x]-pot[e[to].v];
        if (!vis[e[to].v] && e[to].f < e[to].c && d2 < dist[e[to].v]){
          dist[e[to].v] = d2; fl[e[to].v] = min(fl[x],e[to].c-e[to].f);
          par[e[to].v] = to; q.push({d2,e[to].v});
        }
      }
    }
  }
  T2 calc_flow(int s, int t, T1 _f){
    T2 cost = 0;
    while (1){
      dijk(s);
      if (!vis[t]) return -1;
      for (int i = 0; i < g.size(); i++)
        dist[i] += pot[i]-pot[s];
      T1 mnf = min(_f,fl[t]);
      cost += dist[t]*mnf;
      int cur = t;
      while (cur != s){
        e[par[cur]].f += mnf;
        e[par[cur]^1].f -= mnf;
        cur = e[par[cur]^1].v;
      }
      dist.swap(pot);
      _f -= mnf;
      if (!_f) break;
    }
    return cost;
  }
};
```

▸ Trees

LCA

27c09a

Description: Computes lowest common ancestor, precomputed in $O(V \log V)$, $O(\log V)$ per query, uses binary lifting and works for directed and undirected graphs.
Status: Tested

```
struct LCA {
  vector<vector<int>> T, parent;
  vector<int> depth, vis;
  int LOGN, V;
  // If WA, bigger logn?
  LCA(vector<vector<int>> &T, int logn = 20): T(T), LOGN(logn), vis(T.size()) {
    parent.assign(T.size()+1, vector<int>(LOGN, 0));
    depth.assign(T.size()+1, 0);
    // If is forest
    //for (int u = 0; u < T.size(); u++)
    //  if (!vis[u]) dfs(u);
  }
  void dfs(int u = 0, int p = -1) {
    vis[u] = true;
    for (int v : T[u]) {
      if (p != v) {
        depth[v] = depth[u] + 1;
        parent[v][0] = u;
        for (int j = 1; j < LOGN; j++)
          parent[v][j] = parent[parent[v][j-1]][j-1];
        dfs(v, u);
      }
    }
  }
  int query(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    int k = depth[u]-depth[v];
    for (int j = LOGN - 1; j >= 0; j--)
      if (k & (1 << j))
        u = parent[u][j];
    if (u == v)
      return u;
    for (int j = LOGN - 1; j >= 0; j--) {
      if (parent[u][j] != parent[v][j]) {
        u = parent[u][j];
        v = parent[v][j];
      }
    }
    return parent[u][0];
  }
};
```

HLD

a65b7f

Author: Javier Oliva
Description: Answer queries in $O(\log(V) \cdot A \cdot B)$, where A is the complexity of merging two chains and B is the complexity of the data structure query.
Status: Partially tested

```
template <class DS, class T, T merge(T, T), int IN_EDGES>
struct heavy_light {
  vector <int> parent, depth, heavy, head, pos_down;
  int n, cur_pos_down; DS ds_down;
  int dfs(int v, vector < vector < int >>
    const & adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c: adj[v])
      if (c != parent[v]) {
        parent[c] = v, depth[c] = depth[v] + 1;
        int c_size = dfs(c, adj);
        size += c_size;
        if (c_size > max_c_size)
          max_c_size = c_size, heavy[v] = c;
```

```
    }
    return size;
  }
  void decompose(int v, int h, vector<vector<int>>
    const & adj, vector <T> & a_down, vector <T> & values) {
    head[v] = h, pos_down[v] = cur_pos_down++;
    a_down[pos_down[v]] = values[v];
    if (heavy[v] != -1)
      decompose(heavy[v], h, adj, a_down, values);
    for (int c: adj[v]) {
      if (c != parent[v] && c != heavy[v])
        decompose(c, c, adj, a_down, values);
    }
  }
  heavy_light(vector <vector<int>> &adj, vector <T> & values) {
    n = adj.size();
    parent.resize(n);
    depth.resize(n);
    heavy.resize(n, -1);
    head.resize(n);
    pos_down.resize(n);
    vector < T > a_down(n);
    cur_pos_down = 0;
    dfs(0, adj);
    decompose(0, 0, adj, a_down, values);
    ds_down = DS(a_down);
  }
  void update(int a, int b, T x) {
    while (head[a] != head[b]) {
      if (depth[head[a]] < depth[head[b]])
        swap(a, b);
      ds_down.update(pos_down[head[a]], pos_down[a], x);
      a = parent[head[a]];
    }
    if (depth[a] < depth[b])
      swap(a, b);
    if (pos_down[b] + IN_EDGES > pos_down[a])
      return;
    ds_down.update(pos_down[b] + IN_EDGES, pos_down[a], x);
  }
  void update(int a, T x) { ds_down.update(pos_down[a], x); }
  T query(int a, int b) {
    T ans; bool has = 0;
    while (head[a] != head[b]) {
      if (depth[head[a]] < depth[head[b]])
        swap(a, b);
      ans = has ? merge(ans, ds_down.query(pos_down[head[a]], pos_down[a])) :
ds_down.query(pos_down[head[a]], pos_down[a]);
      has = 1;
      a = parent[head[a]];
    }
    if (depth[a] < depth[b])
      swap(a, b);
    if (pos_down[b] + IN_EDGES > pos_down[a])
      return ans;
    return has ? merge(ans, ds_down.query(pos_down[b] + IN_EDGES,
pos_down[a])) : ds_down.query(pos_down[b] + IN_EDGES, pos_down[a]);
  }
};
```

Centroid Decomposition

898938

Description: centroid decomposition algorithm
Status: Partially tested

```
struct centroid_decomp {
    int n;
    vector<bool> vis;
    vector<int> subtr, parent;
    vector<vector<int>> adj;
    // Optional
    // Here you can insert the functions for traversing the partition
    // ----------
    void dfs(int v,int p = -1){
        for(int u: adj[v]){
            if(u != p && !vis[u]) dfs(u,v);
        }
    }
    // ----------

    int calculate_subtree(int v, int p){
        subtr[v] = 1;
        for (int to : adj[v]){
            if (to == p || vis[to]) continue;
            subtr[v] += calculate_subtree(to,v);
        }
        return subtr[v];
    }
    int get_centroid(int v, int p, int sz){
        for (int to : adj[v]){
            if (to == p || vis[to]) continue;
            if (subtr[to]*2> sz) return get_centroid(to,v,sz);
        }
        return v;
    }
    int build(int v = 0){
        int centroid = get_centroid(v,v,calculate_subtree(v,v));
        // Optional
        // Here you can call functions for traversing the partition
        // ----------
        dfs(centroid);
        // ----------
        vis[centroid] = 1;
        for (int to : adj[centroid]){
            if (vis[to]) continue;
            build(to);
            // Optional
            // ----------
            // Save parent centroid (Optional)
            parent[build(to)] = centroid;
            // ----------
        }
        return centroid;
    }
    centroid_decomp(vector<vector<int>> &_adj){
        adj = _adj;
        n = adj.size();
        subtr.resize(n,0);
        parent.resize(n,-1);
        vis.resize(n,0);
        build();
    }
};
```

▸ Shortest Paths

Dijkstra

63c738

Author: Antti Laaksonen
Description: Computes shortest path in $O(V \log(V+E))$, does not allow negative weights

```
for (int i = 1; i <= n; i++) distance[i] = INF;
distance[x] = 0;
q.push({0,x});
while (!q.empty()) {
  int a = q.top().second; q.pop();
  if (processed[a]) continue;
  processed[a] = true;
  for (auto u : adj[a]) {
    int b = u.first, w = u.second;
    if (distance[a]+w < distance[b]) {
      distance[b] = distance[a]+w;
      q.push({-distance[b],b});
    }
  }
}
```

Bellman Ford

28ac61

Description: Calculates shortest paths from s in a graph that might have negative edge weights.
Status: Tested on CSES

```
struct BellmanFord {
  struct Edge { int from, to; ll weight; };
  int n, last_updated = -1; const ll INF = 1e18;
  vector<int> p; vector<ll> dist;
  BellmanFord(vector<Edge> &G, int s, int _n) {
    n = _n; dist.assign(n+1,INF);
    p.assign(n+1,-1); dist[s] = 0;
    for (int i = 1; i <= n; i++) {
      last_updated = -1;
      for (Edge &e : G)
        if (dist[e.from] + e.weight < dist[e.to]) {
          dist[e.to] = dist[e.from] + e.weight;
          p[e.to] = e.from; last_updated = e.to;
        }
    }
  }
  bool getCycle(vector<int> &cycle) {
    if (last_updated == -1) return false;
    for (int i = 0; i < n-1; i++)
      last_updated = p[last_updated];
    for (int x = last_updated ;; x=p[x]) {
      cycle.push_back(x);
      if (x == last_updated and cycle.size() > 1) break;
    }
    reverse(cycle.begin(), cycle.end());
    return true;
  }
};
```

Floyd Warshall

c94555

Description: All shortest path, runs on $O(V^3)$
Status: Tested

```
// Create distance matrix
for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= n; j++) {
    if (i == j) distance[i][j] = 0;
    else if (adj[i][j]) distance[i][j] = adj[i][j];
    else distance[i][j] = INF;
  }
}

// Floyd-Warshall
for (int k = 1; k <= n; k++) {
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      distance[i][j] = min(distance[i][j], distance[i][k]+distance[k][j]);
    }
  }
}
```

▸ Connectivity

DFS

2b411e

Description: Just traverse a graph in dfs order
Status: Tested

```
void dfs(int s) {
  if (visited[s]) return;
  visited[s] = true;
  // process node s
  for (auto u: adj[s]) {
    dfs(u);
  }
}
```

BFS

6d1e5b

Description: Just traverse a graph in bfs order
Status: Tested

```
visited[x] = true;
distance[x] = 0;
q.push(x);
while (!q.empty()) {
int s = q.front(); q.pop();
  // process node s
  for (auto u : adj[s]) {
    if (visited[u]) continue;
    visited[u] = true;
    distance[u] = distance[s]+1;
    q.push(u);
  }
}
```

Tarjan Scc

1c8aed

Description: Finds SCC on a graph
Complexity: $O(V + E)$
Status: Tested on CSES

```
struct tarjan {
  int n,timer;
  vector<bool> vis,ist;
  vector<int> tin,low,st,cmp;
  vector<vector<int>> scc;
  void dfs(int u, vector<vector<int>> & adj){
    vis[u] = ist[u] = 1;
    st.push_back(u); tin[u] = low[u] = timer++;
    for (auto v : adj[u]){
      if (!vis[v]){
        dfs(v,adj);
        low[u] = min(low[u],low[v]);
      } else if (ist[v]){
        low[u] = min(low[u],tin[v]);
      }
    }
    if (low[u] == tin[u]){
      scc.emplace_back(); int cur;
      do {
        cur = st.back(); st.pop_back();
        ist[cur] = 0; scc.back().push_back(cur);
        cmp[cur] = (int)scc.size()-1;
      } while (cur != u);
    }
  }
  tarjan(vector<vector<int>> & adj){
    n = adj.size(); tin.resize(n); timer = 0; cmp.resize(n);
    low.resize(n); vis.resize(n,0); ist.resize(n,0);
    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i,adj);
  }
};
```

Bridges

f76e48

Description: Get the bridges to given graph, runs on $O(V + E)$
Status: Tested

```
struct Bridges {
  int n, timer;
  vector <int> tin, low, vis;
  vector <pair<int, int>> bridges;
  Bridges(vector <vector<int>> &adj, int root = 0) {
    n = adj.size();
    timer = 0;
    vis.resize(n, false);
    tin.resize(n);
    low.resize(n);
    dfs(root, root, adj);
  }
  void dfs(int v, int p, vector <vector<int>> &adj) {
    vis[v] = 1; tin[v] = low[v] = timer++;
    for (int to: adj[v]) {
      if (to == p) continue;
      if (vis[to]) low[v] = min(low[v], tin[to]);
      else {
        dfs(to, v, adj);
        low[v] = min(low[v], low[to]);
        if (low[to] > tin[v]) bridges.push_back({v,to});
      }
    }
  }
};
```

Articulation Points

dfb13c

Description: Finds articulation points in a graph using DFS. Runs on $O(V + E)$
Status: Tested on CSES

```
struct ArticulationPoints {
  int n, timer;
  vector <int> tin, low, vis;
  set <int> points;
  void dfs(int v, int p, vector<vector<int>> &adj) {
    vis[v] = true; tin[v] = low[v] = timer++;
    int child = 0;
    for (int to: adj[v]) {
      if (to == p) continue;
      if (vis[to]) low[v] = min(low[v], tin[to]);
      else {
        dfs(to, v, adj);
        low[v] = min(low[v], low[to]);
        if ((low[to] >= tin[v]) && (p != -1))
          points.insert(v);
        child++;
      }
    }
    if ((p == -1) && (child > 1)) points.insert(v);
  }
  ArticulationPoints(vector < vector < int >> & adj, int root = 0) {
    n = adj.size();
    vis.resize(n, false);
    tin.resize(n);
    low.resize(n);
    dfs(root, -1, adj);
  }
};
```

Kosaraju

adc6ba

Description: SCC in $O(V + E)$
Status: Tested, but needs to be re-written, is too large

```
template<typename T>
struct SCC {
  vector<vector<int>> GT, G, SCC_G, SCC_GT, comp_nodes;
  vector<T> data, cdata;
  stack<int> order;
  vector<int> comp, dp;
  vector<bool> visited;
  T (*cfunc)(T, T);
  int comp_count = 0;
  void topsort(int u) {
    visited[u] = true;
    for (int v : G[u])
      if (!visited[v])
        topsort(v);
    order.push(u);
  }
  void build_component(int u) {
    visited[u] = true;
    for (int v : GT[u])
      if (!visited[v])
        build_component(v);
    comp[u] = comp_count;
    comp_nodes[comp_count].push_back(u);
  }
  void compress_graph() {
    for (int u = 0; u < G.size(); u++)
      cdata[comp[u]] = cfunc(cdata[comp[u]], data[u]);
      for (int u = 0; u < G.size(); u++)
        for (int v : G[u])
          if (comp[u] != comp[v]) {
            SCC_G[comp[u]].push_back(comp[v]);
            SCC_GT[comp[v]].push_back(comp[u]);
          }
  }
  T process(int cmp, T (*func)(T a, T b), T (*merge)(T a, T b)) {
    if (dp[cmp]) return dp[cmp];
    dp[cmp] = cdata[cmp];
    for (int u : SCC_G[cmp])
      dp[cmp] = merge(dp[cmp], func(process(u, func, merge), cdata[cmp]));
    return dp[cmp];
  }
  SCC(vector<vector<int>> &G, vector<T> &data, T (*cfunc)(T a, T b), T
comp_identity, T dp_identity): cfunc(cfunc), G(G), data(data) {
    GT.resize(G.size()); comp_nodes.resize(G.size());
    visited.assign(G.size(), 0);
    cdata.assign(G.size(), comp_identity);
    comp.assign(G.size(), 0);
    SCC_G.resize(G.size()); SCC_GT.resize(G.size());
    dp.assign(G.size(), dp_identity);
    for (int u = 0; u < G.size(); u++)
    for (int v : G[u])
      GT[v].push_back(u);
      for (int u = 0; u < G.size(); u++)
        if (!visited[u])
          topsort(u);
    visited.assign(G.size(), 0);
    while (!order.empty()) {
```

```
      int u = order.top();
      order.pop();
      if (visited[u]) continue;
      build_component(u);
      comp_count++;
    }
    compress_graph();
  }
};
```

Euler Directed Graph

2c50f4

Description: Find Eulerian path in directed graphs using Hierholzer's algorithm
Status: Tested

```
bool euler_directed_graph(vector<vector<int>> &adj, vector<int> &path){
  path.resize(0);
  int n = adj.size();
  vector<int> indeg(n), outdeg(n);
  int v1 = -1, v2 = -1;
  for (int i = 0; i < n; i ++){
    outdeg[i] = adj[i].size();
    for (int x : adj[i])
      indeg[x] ++;
  }
  for (int i = 0; i < n; i ++){
    int dif = outdeg[i] - indeg[i];
    if (dif == 1){
      if (v1 == -1) v1 = i;
      else return false;
    }
    else if (dif == -1){
      if (v2 == -1) v2 = i;
      else return false;
    }
    else if (dif != 0) return false;
  }
  if (v1 != v2 and (v1 == -1 or v2 == -1)) return false;
  int first = v1;
  if (v1 != -1) adj[v2].push_back(v1);
  else
    for (int i = 0; i < n; i ++)
      if (outdeg[i] > 0)
        first = i;
  stack<int> st;
  st.push(first);
  vector<int> res;
  while(!st.empty()){
    int v = st.top();
    if (outdeg[v] == 0){
      res.push_back(v);
      st.pop();
    }
    else{
      st.push(adj[v][0]);
      outdeg[v] --;
      swap(adj[v][0], adj[v][outdeg[v]]);
    }
  }
  for (int i = 0; i < n; i ++)
    if (outdeg[i] != 0)
      return false;
  reverse(res.begin(), res.end());
  if (v1 != -1){
    for (int i = 0; i + 1 < res.size(); i ++){
      if (res[i] == v2 && res[i + 1] == v1){
        vector<int> res2(res.begin() + i + 1, res.end());
        res2.insert(res2.end(), res.begin(), res.begin() + i + 1);
        res = res2;
        break;
      }
    }
  }
  path = res;
  return true;
}
```

Misc

Chu Liu

691a14

Description: Minimum spanning arborescence (directed MST), Edmonds/Chu-Liu algorithm
Complexity: $O(nm)$, returns -1 if not reachable
Status: Tested

```
template <class T> struct chu_liu {
  struct edge {
    int u, v, id;
    T len;
  };
  int n;
  vector<edge> e;
  vector<int> inc, dec, take, pre, num, id, vis;
  vector<T> inw;
  T INF;

  chu_liu(int n) : n(n), pre(n), num(n), id(n), vis(n), inw(n) {
    INF = numeric_limits<T>::max() / 2;
  }
  void add_edge(int x, int y, T w) {
    inc.push_back(0), dec.push_back(0), take.push_back(0);
    e.push_back({x, y, (int)e.size(), w});
  }
  T solve(int root) {
    auto e2 = e;
    T ans = 0;
    int eg = e.size() - 1, pos = eg;
    while (1) {
      for (int i = 0; i < n; i++)
        inw[i] = INF, id[i] = vis[i] = -1;
      for (auto &ed : e2)
        if (ed.len < inw[ed.v])
          inw[ed.v] = ed.len, pre[ed.v] = ed.u, num[ed.v] = ed.id;
      inw[root] = 0;
      for (int i = 0; i < n; i++)
        if (inw[i] == INF)
          return -1;
      int tot = -1;
      for (int i = 0; i < n; i++) {
        ans += inw[i];
        if (i != root)
          take[num[i]]++;
        int j = i;
        while (vis[j] != i && j != root && id[j] < 0)
          vis[j] = i, j = pre[j];
        if (j != root && id[j] < 0) {
          id[j] = ++tot;
          for (int k = pre[j]; k != j; k = pre[k])
            id[k] = tot;
        }
      }
      if (tot < 0)
        break;
      for (int i = 0; i < n; i++)
        if (id[i] < 0)
          id[i] = ++tot;
      n = tot + 1;
      int j = 0;
      for (int i = 0; i < (int)e2.size(); i++) {
        int v = e2[i].v;
        e2[j] = {id[e2[i].u], id[e2[i].v], e2[i].id, e2[i].len - inw[v]};
        if (e2[j].u != e2[j].v) {
          inc.push_back(e2[i].id);
          dec.push_back(num[v]);
          take.push_back(0);
          e2[j++].id = ++pos;
        }
      }
      e2.resize(j);
      root = id[root];
    }
    while (pos > eg) {
      if (take[pos])
        take[inc[pos]]++, take[dec[pos]]--;
      pos--;
    }
    return ans;
  }
};
```

Fast Chu Liu

1ab942

Description: Minimum spanning arborescence (directed MST), optimized Edmonds/Chu-Liu algorithm using lazy skew heaps
Complexity: $O(E \log V)$, returns $(-1, \emptyset)$ if not reachable
Status: Tested (Fastest Speedrun)

```
template <class T> struct fast_chu_liu {
  struct edge {
    int u, v;
    T len;
  };
  struct node {
    edge key;
    node *l = nullptr, *r = nullptr;
    T delta = 0;
    node(edge e) : key(e) {}
    void prop() {
      key.len += delta;
      if (l) l->delta += delta;
      if (r) r->delta += delta;
      delta = 0;
    }
    edge top() {
      prop();
      return key;
    }
  };
  int n;
  vector<edge> e;
  fast_chu_liu(int n) : n(n) {}
  void add_edge(int x, int y, T w) { e.push_back({x, y, w}); }
  node *merge(node *a, node *b) {
    if (!a || !b)
      return a ? a : b;
    a->prop(), b->prop();
    if (a->key.len > b->key.len)
      swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
  }
  void pop(node *&a) {
    a->prop();
    a = merge(a->l, a->r);
  }
  pair<T, vector<int>> solve(int root) {
```

```
    union_find_rb uf(n);
    vector<node *> heap(n, nullptr);
    for (edge &ed : e)
      heap[ed.v] = merge(heap[ed.v], new node(ed));
    T ans = 0;
    vector<int> seen(n, -1), path(n), par(n);
    seen[root] = root;
    vector<edge> Q(n), in(n, {-1, -1, 0});
    deque<tuple<int, int, vector<edge>>> cycs;
    for (int s = 0; s < n; s++) {
      int u = s, qi = 0, w;
      while (seen[u] < 0) {
        if (!heap[u])
          return {-1, {}};
        edge ed = heap[u]->top();
        heap[u]->delta -= ed.len, pop(heap[u]);
        Q[qi] = ed, path[qi++] = u, seen[u] = s;
        ans += ed.len, u = uf.findSet(ed.u);
        if (seen[u] == s) {
          node *cyc = nullptr;
          int end = qi, t = (int)uf.ops.size();
          do
            cyc = merge(cyc, heap[w = path[--qi]]);
          while (uf.unionSet(u, w));
          u = uf.findSet(u), heap[u] = cyc, seen[u] = -1;
          cycs.push_front({u, t, {&Q[qi], &Q[end]}});
        }
      }
      for (int i = 0; i < qi; i++)
        in[uf.findSet(Q[i].v)] = Q[i];
    }
    for (auto &[u, t, comp] : cycs) {
      while ((int)uf.ops.size() > t)
        uf.rb();
      edge inEdge = in[u];
      for (auto &ed : comp)
        in[uf.findSet(ed.v)] = ed;
      in[uf.findSet(inEdge.v)] = inEdge;
    }
    for (int i = 0; i < n; i++)
      par[i] = in[i].u;
    return {ans, par};
  }
};
```

Geometry

Algorithms

Convex Hull

 # 5662dc

Description: Get the convex hull of the cloud of points, allow collinear points if is needed
Status: Highly tested

```
template<typename T>
vector<Point2D<T>> convexHull(vector<Point2D<T>> cloud, bool ac = 0) {
  int n = cloud.size(), k = 0;
  sort(cloud.begin(), cloud.end(), [](Point2D<T> &a, Point2D<T> &b) {
    return a.x < b.x or (a.x == b.x and a.y < b.y);
  });
  if (n <= 2) return cloud;
  bool allCollinear = true;
  for (int i = 2; i < n; ++i) {
    if (((cloud[1] - cloud[0]) ^ (cloud[i] - cloud[0])) != 0) {
      allCollinear = false; break;
    }
  }
  if (allCollinear) return ac ? cloud : vector<Point2D<T>>{cloud[0],
cloud.back()};
  vector<Point2D<T>> ch(2 * n);
  auto process = [&](int st, int end, int stp, int t, auto cmp) {
    for (int i = st; i != end; i += stp) {
      while (k >= t and cmp(ch[k - 1], ch[k - 2], cloud[i])) k--;
      ch[k++] = cloud[i];
    }
  };
  process(0, n, 1, 2, [&](auto a, auto b, auto c) {
    return ((a - b) ^ (c - b)) < (ac ? 0 : 1);
  });
  process(n - 2, -1, -1, k + 1, [&](auto a, auto b, auto c) {
    return ((a - b) ^ (c - b)) < (ac ? 0 : 1);
  });
  ch.resize(k - 1);
  return ch;
}
```

Halfplane Intersection

 # 82b7a7

Description: Halfplane intersection
Status: Partially tested

```
const ld inf = 1e18;
const ld eps = 1e-9;
vector<Point2D<ld>> hp_intersect(vector<Halfplane> &H) {
  Point2D<ld> box[4] = {{inf, inf}, {-inf, inf}, {-inf, -inf}, {inf, -inf}};
  for (int i = 0; i < 4; i++) {
    Halfplane aux(box[i], box[(i + 1) % 4]);
    H.push_back(aux);
  }
  sort(H.begin(), H.end());
  deque<Halfplane> dq;
  int len = 0;
  for (int i = 0; i < H.size(); i++) {
    while (len > 1 && H[i].out(inter(dq[len - 1], dq[len - 2])))
      dq.pop_back(), --len;
    while (len > 1 && H[i].out(inter(dq[0], dq[1])))
      dq.pop_front(), --len;
    if (len > 0 && fabsl((H[i].pq ^ dq[len - 1].pq)) < eps) {
      if ((H[i].pq | dq[len - 1].pq) < 0.0)
        return vector<Point2D<ld>>();
      if (H[i].out(dq[len - 1].p))
        dq.pop_back(), --len;
      else
        continue;
    }
    dq.push_back(H[i]), ++len;
  }
  while (len > 2 && dq[0].out(inter(dq[len - 1], dq[len - 2])))
    dq.pop_back(), --len;
  while (len > 2 && dq[len - 1].out(inter(dq[0], dq[1])))
    dq.pop_front(), --len;
  if (len < 3)
    return vector<Point2D<ld>>();
  vector<Point2D<ld>> ret(len);
  for (int i = 0; i + 1 < len; i++)
    ret[i] = inter(dq[i], dq[i + 1]);
  ret.back() = inter(dq[len - 1], dq[0]);
  return ret;
}
```

Point Inside Polygon

 # 12fb92

Description: Check if a point is inside, bounds or out of the polygon
- 0: Bounds
- 1: Inside
- −1: Out

Status: Tested

```
template<typename T>
int pointInsidePolygon(vector<Point2D<T>> &P, Point2D<T> q) {
  int N = P.size(), cnt = 0;
  for(int i = 0; i < N; i++) {
    int j = (i == N-1 ? 0 : i+1);
    Segment<T> s(P[i], P[j]);
    if(s.contain(q)) return 0;
    if(P[i].x <= q.x and q.x < P[j].x and q.cross(P[i], P[j]) < 0) cnt++;
    else if(P[j].x <= q.x and q.x < P[i].x and q.cross(P[j], P[i]) < 0) cnt++;
  }
  return cnt&1 ? 1 : -1;
}
```

Polygon Area

 # 2f7cd4

Description: Get the area of a polygon
- If you want the double of the area, just enable the flag

Status: Highly tested

```
template<typename T>
T polygonArea(vector<Point2D<T>> P, bool x2 = 0) {
  T area = 0;
  for(int i = 0; i < P.size()-1; ++i)
    area += P[i]^(P[i+1]);
  // Si el primer punto se repite, sacar:
  area += (P.back())^(P.front());
  return abs(area)/ (x2 ? 1 : 2);
}
```

Nearest 2 Points

 # fe88e0

Description: Using divide and conquer, allow us to know what are the two nearest point between them
Status: Tested on CSES

```
#define sq(x) ((x)*(x))
template <typename T>
pair<Point2D<T>, Point2D<T>> nearestPoints(vector<Point2D<T>> &P, int l, int r)
{
  const T INF = 1e10;
  if (r-l == 1) return {P[l], P[r]};
  if (l >= r) return {{INF, INF}, {-INF, -INF}};

  int m = (l+r)/2;
  pair<Point2D<T>, Point2D<T>> D1, D2, D;
  D1 = nearestPoints(P, l, m);
  D2 = nearestPoints(P, m+1, r);
  D = (D1.first.sqdist(D1.second) <= D2.first.sqdist(D2.second) ? D1 : D2);

  T d = D.first.sqdist(D.second), x_center = (P[m].x + P[m+1].x)/2;
  vector<Point2D<T>> Pk;
  for (int i = l; i <= r; i++)
    if (sq(P[i].x-x_center) <= d)
      Pk.push_back(P[i]);

  sort(Pk.begin(), Pk.end(), [](const Point2D<T> p1, const Point2D<T> p2) {
    return p1.y != p2.y ? p1.y < p2.y : p1.x < p2.x;
  });
```

```
  for(int i = 0; i < Pk.size(); ++i) {
    for(int j = i-1; j >= 0; --j) {
      if(sq(Pk[i].y-Pk[j].y) > d) break;
      if(Pk[i].sqdist(Pk[j]) <= D.first.sqdist(D.second))
        D = {Pk[i],Pk[j]};
    }
    for(int j = i+1; j < Pk.size(); ++j) {
      if(sq(Pk[i].y-Pk[j].y) > d) break;
      if(Pk[i].sqdist(Pk[j]) <= D.first.sqdist(D.second))
        D = {Pk[i],Pk[j]};
    }
  }

  return D;
}

template <typename T>
pair<Point2D<T>, Point2D<T>> nearestPoints(vector<Point2D<T>> &P) {
  sort(P.begin(), P.end(), [](const Point2D<T> &p1, const Point2D<T> &p2) {
    if (p1.x == p2.x) return p1.y < p2.y;
    return p1.x < p2.x;
  });

  return nearestPoints(P, 0, P.size()-1);
}
```

Minimum Enclosing Circle

57b558

Description: Minimum enclosing circle, Welzl algorithm $O(n)$ expected
Status: Tested

```
circle<ld> minimum_enclosing_circle(vector<Point2D<ld>> p) {
  mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
  shuffle(p.begin(), p.end(), rng);
  int n = p.size();
  if (n == 0) return circle<ld>();
  circle<ld> cur(p[0], 0);
  auto out = [&](int i) { return sign(sqrtl(cur.c.sqdist(p[i])) - cur.r) >
0; };
  for (int i = 1; i < n; i++) if (out(i)) {
    cur = circle<ld>(p[i], 0);
    for (int j = 0; j < i; j++  ) if (out(j)) {
      cur = circle<ld>((p[i] + p[j]) / 2, sqrtl(p[i].sqdist(p[j])) / 2);
      for (int k = 0; k < j; k++) if (out(k))
        cur = circle<ld>(p[i], p[j], p[k]);
    }
  }
  return cur;
}
```

Convex Hull 3D

c18592

Description: 3D Convex Hull using randomized incremental algorithm
Time: $O(n \log n)$ expected
Status: Tested

```
template <typename T> struct convex_hull_3d {
  using pt = point3d<T>;
  const ld EPS = 1e-9;
  struct edge;
  struct face {
    int a, b, c;
    pt pa, pb, pc, q;
    edge *e1, *e2, *e3;
    vector<int> points;
    int dead = 1e9;
    face(int a, int b, int c, pt pa, pt pb, pt pc)
        : a(a), b(b), c(c), pa(pa), pb(pb), pc(pc) {
      q = ((pb - pa)^(pc - pa));
      e1 = e2 = e3 = nullptr;
    }
  };

  struct edge {
    edge *rev; face *f;
  };

  vector<pt> p;
  vector<face *> f;

  static void glue(face *F1, face *F2, edge *&e1, edge *&e2) {
    e1 = new edge, e2 = new edge;
    e1->rev = e2, e2->rev = e1;
    e1->f = F2, e2->f = F1;
  }

  void prepare() {
    int n = (int)p.size();
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    shuffle(p.begin(), p.end(), rng);
    vector<int> ve = {0};
    for (int i = 1; i < n; i++) {
      if (ve.size() == 1) {
        if ((p[ve[0]] - p[i]).abs() > EPS) ve.push_back(i);
      } else if (ve.size() == 2) {
        if ((p[ve[1]] - p[ve[0]]).cross(p[i] - p[ve[0]]).abs() > EPS)
          ve.push_back(i);
      } else if (abs((p[i] - p[ve[0]])
                .dot((p[ve[1]] - p[ve[0]])
                .cross(p[ve[2]] - p[ve[0]]))) > EPS) {
        ve.push_back(i);
        break;
      }
    }
    assert((int)ve.size() == 4);
    vector<pt> ve2;
    for (int i : ve) ve2.push_back(p[i]);
    reverse(ve.begin(), ve.end());
    for (int i : ve) p.erase(p.begin() + i);
    p.insert(p.begin(), ve2.begin(), ve2.end());
  }

  convex_hull_3d(vector<pt> points) : p(move(points)) {
    int n = p.size();
    prepare();
    vector<face *> new_face(n, nullptr);
    vector<vector<face *>> conflict(n);

    auto add_face = [&](int a, int b, int c) {
      face *F = new face(a, b, c, p[a], p[b], p[c]);
      f.push_back(F);
      return F;
    };

    face *F1 = add_face(0, 1, 2);
    face *F2 = add_face(0, 2, 1);
    glue(F1, F2, F1->e1, F2->e3);
    glue(F1, F2, F1->e2, F2->e2);
    glue(F1, F2, F1->e3, F2->e1);

    for (int i = 3; i < n; i++) {
      for (face *F : {F1, F2}) {
        T Q = (p[i] - p[F->a]).dot(F->q);
        if (Q > EPS) conflict[i].push_back(F);
        if (Q >= -EPS) F->points.push_back(i);
      }
    }

    for (int i = 3; i < n; i++) {
      for (face *F : conflict[i])
        F->dead = min(F->dead, i);
      int v = -1;
      for (face *F : conflict[i]) {
        if (F->dead != i)
          continue;
        int parr[3] = {F->a, F->b, F->c};
        edge *earr[3] = {F->e1, F->e2, F->e3};
        for (int j = 0; j < 3; j++) {
          int j2 = (j + 1)%3;
          if (earr[j]->f->dead > i) {
            face *Fn = new_face[parr[j]] = add_face(parr[j], parr[j2], i);
            set_union(F->points.begin(), F->points.end(),
                      earr[j]->f->points.begin(), earr[j]->f->points.end(),
                      back_inserter(Fn->points));
            Fn->points.erase(
                stable_partition(Fn->points.begin(), Fn->points.end(),
                                 [&](int k) {
                                   return k > i and
                                          (p[k] - p[Fn->a]).dot(Fn->q) > EPS;
                                 }),
                Fn->points.end());
            for (int k : Fn->points) conflict[k].push_back(Fn);
            earr[j]->rev->f = Fn;
            Fn->e1 = earr[j];
            v = parr[j];
          }
        }
      }
      if (v == -1) continue;
      while (new_face[v]->e2 == nullptr) {
        int u = new_face[v]->b;
        glue(new_face[v], new_face[u], new_face[v]->e2, new_face[u]->e3);
        v = u;
      }
    }
    f.erase(remove_if(f.begin(), f.end(), [&](face *F) { return F->dead < n; }), f.end());
  }

  int num_faces() const { return f.size(); }
  T area() {
    T res = 0;
    for (face *F : f)
      res += F->q.abs() / 2.0;
    return res;
  }
  T volume() {
    T vol = 0;
    for (face *F : f)
      vol += F->pa.dot(F->q);
    return abs(vol) / 6.0;
  }
  vector<array<int, 3>> faces() {
    vector<array<int, 3>> res;
    for (face *F : f)
      res.push_back({F->a, F->b, F->c});
    return res;
  }
};
```

Order By Angle

63609f

Description: Sort the point depending their angle
Status: Not tested

```
template <typename T>
int semiplane(Point2D<T> p) { return p.y > 0 or (p.y == 0 and p.x > 0); }

template <typename T>
void orderByAngle(vector<Point2D<T>> &P) {
  sort(P.begin(), P.end(), [](Point2D<T> &p1, Point2D<T> &p2) {
    int s1 = semiplane(p1), s2 = semiplane(p2);
    return s1 != s2 ? s1 > s2 : (p1^p2) > 0;
  });
}
```

▸ Primitives

Point 2D

421c69

Description: Just a 2D Point
Status: Tested

```
template <typename T> struct Point2D {
  T x, y;
  Point2D(){};
  Point2D(T x_, T y_) : x(x_), y(y_) {}
  Point2D<T> &operator=(Point2D<T> t) {
    x = t.x, y = t.y;
    return *this;
  }
  Point2D<T> &operator+=(Point2D<T> t) {
    x += t.x, y += t.y;
    return *this;
  }
  Point2D<T> operator-(Point2D<T> t) { return {x - t.x, y - t.y}; }
  Point2D<T> operator+(Point2D<T> t) { return {x + t.x, y + t.y}; }
  Point2D<T> operator*(T t) { return {x * t, y * t}; }
```

```
  Point2D<T> operator/(T t) { return {x / t, y / t}; }
  Point2D<T> &operator-=(Point2D<T> t) {
    x -= t.x, y -= t.y;
    return *this;
  }
  Point2D<T> &operator*=(Point2D<T> t) {
    x *= t.x, y *= t.y;
    return *this;
  }
  Point2D<T> &operator/=(Point2D<T> t) {
    x /= t.y, y /= t.y;
    return *this;
  }
  T operator|(Point2D<T> b) { return x * b.x + y * b.y; }
  T operator^(Point2D<T> b) { return x * b.y - y * b.x; }
  T cross(Point2D<T> a, Point2D<T> b) { return (a - *this) ^ (b - *this); }
  T norm() { return (*this) | (*this); }
  T sqdist(Point2D<T> b) { return ((*this) - b).norm(); }
  double abs() { return sqrt(norm()); }
  double proj(Point2D<T> b) { return (*this | b) / b.abs(); }
  double angle(Point2D<T> b) {
    return acos(((*this) | b) / this->abs() / b.abs());
  }
  Point2D<T> rotate(T a) {
    return {cosl(a) * x - sinl(a) * y, sinl(a) * x + cosl(a) * y};
  }
};
template <typename T> Point2D<T> operator*(T a, Point2D<T> b) { return b * a; }
```

Segment

02f807

Description: Segment implementation
Status: Partially tested

```
template<typename T >
struct Segment {
  Point2D< T > P;
  Point2D< T > Q;
  const T INF = numeric_limits<T>::max();
  Segment(Point2D<T> P, Point2D<T> Q): P(P), Q(Q) {}
  int sign(T x, T eps = 0) { return x > eps ? 1 : x < -eps ? -1 : 0; }
  bool contain(Point2D<T> p, T eps = 0) {
    return ((P - p)|(Q - p)) <= (T)0 and abs(((Q - P)^(p - P))) <= eps;
  }
  bool intersect(Segment<T> b) {
    if (this->contain(b.P) or this->contain(b.Q) or b.contain(P) or
b.contain(Q))
      return true;
    // change < 0 or <= depending the problem
    return sign((((b.P) - P))^((Q - P)))*sign(((b.Q - P)^(Q - P))) < 0 and
           sign(((P - b.Q)^(b.Q - b.P)))*sign(((Q- b.P)^(b.Q - b.P))) < 0;
  }
  // not tested
  Point2D<T> intersection(Segment<T> b) {
    if(this->intersect(b))
      return (((b.Q-b.P)^(Q-b.P))*P + ((P-b.P)^(b.Q-b.P))*Q)/((P-Q)^(b.Q-b.P));
    return {INF, INF};
  }
};
```

Halfplane

8f4c07

Description: Halfplane
Status: Partially tested

```
struct Halfplane {
  Point2D<ld> p, pq;
  ld angle;
  Halfplane() {}
  Halfplane(Point2D<ld> &a, Point2D<ld> &b) : p(a), pq(b - a) {
    angle = atan2l(pq.y, pq.x);
  }
  bool out(Point2D<ld> r) { return ((pq) ^ (r - p)) < -eps; }
  bool operator<(Halfplane &e) { return angle < e.angle; }
  friend Point2D<ld> inter(Halfplane &s, Halfplane &t) {
    ld alpha = ((t.p - s.p) ^ t.pq) / ((s.pq) ^ (t.pq));
    return s.p + (s.pq * alpha);
  }
};
```

Circle

234d03

Description: 2D circle, constructors: empty, (center, radius), (a, b, c)
circumcircle Status: Tested

```
template <typename T> struct circle {
  Point2D<T> c;
  T r;
  circle() : c(), r(0) {}
  circle(Point2D<T> c_, T r_) : c(c_), r(r_) {}
  circle(Point2D<T> a, Point2D<T> b, Point2D<T> p) {
    Point2D<T> ab = b - a, ap = p - a;
    T d = 2 * (ab ^ ap);
    if (sign(d) == 0) {
      c = Point2D<T>();
      r = 0;
      return;
    }
    T s = ab.norm(), t = ap.norm();
    c = a + Point2D<T>(ap.y * s - ab.y * t, ab.x * t - ap.x * s) / d;
    r = c.sqdist(a);
    r = sqrtl(r);
  }
};
```

Dynamic Programming

▸ Classic

LIS

c94f42

Description: Longest increasing subsequence in $O(n \log n)$, allow us to recover the sequence
Status: Highly tested

```
template <class I> vector<int> LIS(const vector<I> &S) {
  if (S.empty())
    return {};
  vector<int> prev(S.size());
  vector<pair<I, int>> res;
  for (int i = 0; i < S.size(); i++) {
    auto it = lower_bound(res.begin(), res.end(), pair<I, int>{S[i], i});
    if (it == res.end())
      res.emplace_back(), it = res.end() - 1;
    *it = {S[i], i};
    prev[i] = (it == res.begin() ? 0 : (it - 1)->second);
  }
  int L = res.size(), cur = res.back().second;
  vector<int> ans(L);
  while (L--)
    ans[L] = cur, cur = prev[cur];
  /* Recover the sequence
  for (int i = 0; i+1 < ans.size(); i++)
    ans[i] = S[ans[i]];
  */
  return ans;
}
```

▸ Optimization

D&C

05a4a8

Description: Divide and conquer DP optimization. $O(kn^2)$ to $O(kn \log n)$.
Status: Tested

```
const ll INF = 1e18;
int n;
vector<ll> dp, ndp;

ll cost(int l, int r); // define your cost function

void dac_solve(int l, int r, int optl, int optr) {
  if (l > r) return;
  int mid = (l + r) / 2, opt = optl;
  ndp[mid] = INF;
  for (int k = optl; k <= min(mid - 1, optr); k++) {
    ll val = dp[k] + cost(k + 1, mid);
    if (val < ndp[mid]) ndp[mid] = val, opt = k;
  }
  dac_solve(l, mid - 1, optl, opt);
  dac_solve(mid + 1, r, opt, optr);
}

ll dac_dp(int layers) {
  dp.assign(n + 1, INF); dp[0] = 0;
  for (int i = 0; i < layers; i++) {
    ndp.assign(n + 1, INF);
    dac_solve(1, n, 0, n - 1);
    swap(dp, ndp);
  }
  return dp[n];
}
```

Knuth

b96d2e

Description: Knuth DP optimization. $O(n^3)$ to $O(n^2)$.
Status: Tested

```
const ll INF = 1e18;
int n;
vector<vector<ll>> dp;
vector<vector<int>> opt;

ll cost(int i, int j); // define your cost function

ll knuth_dp() {
  dp.assign(n, vector<ll>(n, 0));
  opt.assign(n, vector<int>(n));
  for (int i = 0; i < n; i++) opt[i][i] = i;

  for (int len = 2; len <= n; len++) {
    for (int i = 0; i + len <= n; i++) {
      int j = i + len - 1;
      dp[i][j] = INF;
      for (int k = opt[i][j - 1]; k <= opt[min(i + 1, j)][j]; k++) {
        ll val = dp[i][k] + (k + 1 <= j ? dp[k + 1][j] : 0) + cost(i, j);
```

```
        if (val < dp[i][j]) dp[i][j] = val, opt[i][j] = k;
      }
    }
  }
  return dp[0][n - 1];
}
```

Convex Hull Trick

fea06b

Description: Given lines mantains a convex space to maximum queries, a is the slope and b the constant.
Important: Sort slopes before use
Status: Partially tested

```
struct convex_hull_trick {
  vector<ll> A, B;
  const ll inf = numeric_limits<ll>::max()/4;
  double cross(ll i, ll j, ll k) {
    return 1.0*(A[j] - A[i]) * (B[k] - B[i]) - 1.0*(A[k] - A[i]) * (B[j] -
B[i]);
  }
  void add(ll a, ll b) {
    A.push_back(a), B.push_back(b);
    while(A.size() > 2 and cross(A.size() - 3, A.size() - 2, A.size() - 1) <=
0)
      A.erase(A.end() - 2), B.erase(B.end() - 2);
  }
  ll query(ll x) {
    if(A.empty()) return inf;
    ll l = 0, r = A.size() - 1;
    while (l < r) {
      ll mid = (l + r)/2;
      ll f1 = A[mid] * x + B[mid];
      ll f2 = A[mid + 1] * x + B[mid + 1];
      if(f1 > f2) l = mid + 1;
      else r = mid;
    }
    return A[l] * x + B[l];
  }
};
```

Li Chao Tree

90cf3d

Description: Dynamically insert lines of the form y = a*x + b and query for the minimum value at any x in a fixed interval [L, R].
Status: Partially tested

```
struct Line{
  ll a,b; // ax + b
  Line(ll _a, ll _b) : a(_a), b(_b) {};
  ll eval(ll x){ return a*x + b;}
};

struct li_chao{
  const ll INF = 1e18;
  ll L,R;
  vector<Line> st;
  li_chao(ll l,ll r){
    L = l, R = r+1;
    st = vector<Line>(4*(r - l + 1),Line(0,INF));
  };
  void add(Line nw, ll v, ll l, ll r){
    ll m = (l+r)/2;
    bool lc = nw.eval(l) < st[v].eval(l);
    bool mc = nw.eval(m) < st[v].eval(m);
    if(mc) swap(nw,st[v]);
    if(r - l == 1) return;
    if(lc != mc) add(nw,2*v,l,m);
    else add(nw,2*v + 1,m,r);
  }
  ll get(ll x, ll v, ll l, ll r){
    ll m = (l+r)/2;
    if(r - l == 1) return st[v].eval(x);
    if(x < m) return min(st[v].eval(x),get(x,2*v,l,m));
    return min(st[v].eval(x),get(x,2*v+1,m,r));
  }

  void add(ll a, ll b){add(Line(a,b),1,L,R);};
  ll get(ll x){return get(x,1,L,R);};
};
```

Combinatorial

Weighted Matroid Intersection

834d3c

Description: Maximum weight base in matroid intersection $O(r^{2n(O1+O2)})$, where r is the rank of the basis, and n is the size of the ground set, O1 is the cost of Oracle query in the first matroid, and O2 the cost of Oracle query in second matroid.

Status: Not tested

```
template<
    typename W_t, // weight type
    typename T, // element of the ground set type
    typename Orc1, // Oracle of the first matroid
    typename Orc2 // Oracle of the second matroid
>
vector<vector<T>> weighted_matroid_intersection(const vector<T> & ground_set, const Orc1 & matroid1,
const Orc2 & matroid2){
    int n = ground_set.size();
    vector<vector<T>> res;
    vector<char> in_set(n), in_matroid1(n), in_matroid2(n);
    vector<pair<W_t,int>> dist(n);
    vector<pair<int,int>> edges;
    vector<int> par(n), l, r;
    l.reserve(n);
    r.reserve(n);
    while (1) {
        res.push_back({});
        for (int i = 0; i < n; i++)
            if (in_set[i])
                res.back().push_back(ground_set[i]);
        Orc1 m1 = matroid1;
        Orc2 m2 = matroid2;
        l.clear();
        r.clear();
        for (int i = 0; i < n; i++)
            if (in_set[i]) {
                m1.add(ground_set[i]);
                m2.add(ground_set[i]);
                l.push_back(i);
            } else {
                r.push_back(i);
            }
        fill(in_matroid1.begin(), in_matroid1.end(), 0);
        fill(in_matroid2.begin(), in_matroid2.end(), 0);
        for (int i : r) {
            in_matroid1[i] = m1.independed_with(ground_set[i]);
            in_matroid2[i] = m2.independed_with(ground_set[i]);
        }
        edges.clear();
        for (int i : l) {
            {
                Orc1 m = matroid1;
                for (int j : l)
                    if (i != j)
                        m.add(ground_set[j]);

                for (int j : r)
                    if (m.independed_with(ground_set[j]))
                        edges.emplace_back(i, j);
            }
            {
                Orc2 m = matroid2;
                for (int j : l)
                    if (i != j)
                        m.add(ground_set[j]);
                for (int j : r)
                    if (m.independed_with(ground_set[j]))
                        edges.emplace_back(j, i);
            }
        }
        static constexpr W_t INF = numeric_limits<W_t>::max();
        fill(dist.begin(), dist.end(), pair{INF, -1});
        fill(par.begin(), par.end(), -1);
        for (int i : r)
            if (in_matroid1[i])
                dist[i] = {-ground_set[i].weight, 0};

        while (1) {
            bool any = false;
            for (auto &[v, u] : edges) {
                if (dist[v].first == INF)
                    continue;

                int coeff = in_set[v] ? -1 : 1;
                if (dist[u] > pair{dist[v].first + coeff * ground_set[u].weight, dist[v].second + 1}) {
                    par[u] = v;
                    dist[u] = {dist[v].first + coeff * ground_set[u].weight, dist[v].second + 1};
                    any = 1;
                }
            }
            if (!any)
                break;
        }
        int finish = -1;
        for (int v : r)
            if (in_matroid2[v] && dist[v].first != INF && (finish == -1 || dist[finish] > dist[v]))
                finish = v;
        if (finish == -1)
            break;

        for (; finish != -1; finish = par[finish])
            in_set[finish] ^= 1;
    }
    return res;
}
```