Team Reference Document

Far Eastern FU: BosonHiggsPrice

Максим Сказкин, Алексей Луговой, Дмитрий Горбатенко

2024

Contents

1 Contest **1**
- 1.1 Template . . . 1

2 Structures **2**
- 2.1 Disjoint Set Union . . . 2
- 2.2 PBDS Tree . . . 2
- 2.3 Rope . . . 2
- 2.4 Segment Tree . . . 2
- 2.5 Lazy Segment Tree . . . 2
- 2.6 Fenwick Tree . . . 3
- 2.7 2D Fenwick Tree . . . 3
- 2.8 RMQ . . . 3

3 Sorting **4**
- 3.1 Mergesort . . . 4
- 3.2 Radixsort . . . 4
- 3.3 Quickselect . . . 4
- 3.4 Binary search . . . 4

4 Strings **4**
- 4.1 Prefix function . . . 4
- 4.2 Z function . . . 5
- 4.3 Manacker algorithm . . . 5
- 4.4 Hashing . . . 5
- 4.5 Prefix Tree . . . 5

5 Graphs **6**
- 5.1 Dijkstra Algorithm . . . 6
- 5.2 Shortest Path Faster . . . 6
- 5.3 Floyd-Warshall . . . 6
- 5.4 Euler Path . . . 6
- 5.5 Hamilton Path . . . 7
- 5.6 Topological Sort . . . 7
- 5.7 Stronly Connected Components . . . 7
- 5.8 Bridge Edges & Cut Vertices . . . 7

6 Math **8**
- 6.1 Equations . . . 8
- 6.2 LU Decomposition . . . 8
- 6.3 Polynomials . . . 8
- 6.4 Extremums . . . 9
- 6.5 Sums . . . 9
- 6.6 Trigonometry . . . 9
- 6.7 Number theory . . . 9
- 6.8 Combinatorics . . . 10

7 Geometry **11**
- 7.1 Points . . . 11
- 7.2 Lines . . . 11
- 7.3 Circles . . . 12
- 7.4 Polygons . . . 13
- 7.5 Hull . . . 13
- 7.6 Some Formulas . . . 13

8 Dynamic Programming **14**
- 8.1 Longest Increasing Subsequence . . . 14

9 Various **14**
- 9.1 Bit Manipulations . . . 14
- 9.2 Cycle Detection . . . 14

1 Contest

1.1 Template

```
#pragma GCC optimize("O2,unroll-loops")
#pragma GCC target("avx2,lzcnt,popcnt")

#include <bits/stdc++.h>
using namespace std;

using ll = long long;
using ull = unsigned long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
using vi = vector<int>;
using vvi = vector<vector<int>>;
using vpii = vector<pair<int, int>>;
#define hashset unordered_set
#define hashmap unordered_map
#define PB push_back
#define MP make_pair
#define MT make_tuple
#define F first
#define S second
#define SP ' '
#define NL '\n'
#define all(c) (c).begin(), (c).end()

ios::sync_with_stdio(false);
cin.tie(0); cout.tie(0);
cin.exceptions(cin.failbit);
```

Link streams with files

```
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
```

Optimization

- ⋆ `target("O3")` – auto-vectorizing, inlining, unrolling loops

- ⋆ `target("Ofast")` – `O3` + some **unsafe** math (beware of it)
- ⋆ `optimize("trapv")` – debug int overflows

Use `__attribute__((target("..."), optimize("...")))` to enable for the function only

Comparing doubles

```
a == b or abs(a - b) < max(abs(a), abs(b)) * numeric_limits<double>::epsilon()
```

2 Structures

2.1 Disjoint Set Union

Stores disjoint subsets. **TC:** $O(1)$ **SC:** $O(n)$

```
struct DSU {
  vi parents;
  mt19937 rng;

  DSU(int n) {
    parents.resize(n);
    for (int i = 0; i < n; ++i) parents[i] = i;
    rng = mt19937(chrono::steady_clock::now().time_since_epoch().count());
  }
  int find(int idx) {
    if (parents[idx] == idx) return idx;
    return parents[idx] = find(parents[idx]);
  }
  void join(int lhs, int rhs) {
    if (rng() % 2) parents[find(lhs)] = find(rhs);
    else parents[find(rhs)] = find(lhs);
  }
};
```

2.2 PBDS Tree

Set/map with two additional ops. **TC:** $O(\log n)$ **SC:** $O(n)$

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

// Value = null_type for set
typedef __gnu_pbds::tree<Key, Value, less<Key>, __gnu_pbds::rb_tree_tag,
                         __gnu_pbds::tree_order_statistics_node_update> Ordtree;
```

- ⋆ `iterator find_by_order(size_t ord)`
- ⋆ `size_t order_of_key(T elem)`

2.3 Rope

`rope` – vector with fast slicing, insertion, but slow accessing. **TC:** $O(\log n)$ **SC:** $O(n)$

```
#include <ext/rope>
using namespace __gnu_cxx;
```

- ⋆ `void push_back(T elem)`
- ⋆ `T pop_back()`
- ⋆ `void insert(int pos, rope targ)`
- ⋆ `void erase(int pos, int count)`
- ⋆ `rope substr(int pos, int count)`
- ⋆ `void replace(int pos, int count, rope targ)`
- ⋆ `rope operator+(rope lhs, rope rhs)`

2.4 Segment Tree

Zero-indexed. Segments exclude end: $[a, b)$. `tree[0]` isn't used. **TC:** $O(\log n)$ **SC:** $O(n)$

```
struct Segtree {
  typedef int T;
  static const T unit = INT32_MIN;
  T f(T a, T b) { return max(a, b); };
  vector<T> tree;
  int size;

  Segtree(int sz, T def = unit) : size(__bit_ceil(sz)), tree(2 * sz, def) {}
  Segtree(vector<T> &data) {
    size = __bit_ceil(data.size());
    tree.resize(2 * size);
    copy(data.begin(), data.end(), tree.begin() + size);
    for (int i = size - 1; i > 0; --i) tree[i] = f(tree[2 * i], tree[2 * i + 1]);
  }
  T get(int i) { return tree[size + i]; }
  T qry(int lb, int rb) {
    T lf = unit, rf = unit;
    for (lb += size, rb += size; lb < rb; lb /= 2, rb /= 2) {
      if (lb % 2) lf = f(lf, tree[lb++]);
      if (rb % 2) rf = f(tree[--rb], rf);
    }
    return f(lf, rf);
  }
  void upd(int i, T val) {
    tree[i + size] = f(tree[i + size], val);
    for (i += size; i /= 2;) tree[i] = f(tree[2 * i], tree[2 * i + 1]);
  }
  void set(int i, T val) {
    for (tree[i += size] = val; i /= 2;) tree[i] = f(tree[2 * i], tree[2 * i + 1]);
  }
};
```

2.5 Lazy Segment Tree

The same, but also can apply `f` to the whole segment. **TC:** $O(\log n)$ **SC:** $O(n)$

```
struct LazySegtree {
  typedef int T;
  static const T unit = 0;
  vector<T> tree, lazy;
  int size;

  LazySegtree(int sz)
  : size(__bit_ceil(sz)), tree(2 * __bit_ceil(sz)), lazy(2 * __bit_ceil(sz)) {}
  T qry(int ql, int qr, int node = 1, int lb = 0, int rb = -1) {
    if (rb == -1) rb = size;
    if (qr <= lb or rb <= ql) return 0;
    if (ql <= lb && rb <= qr) return tree[node];
    push(node, lb, rb);
```

```
    int md = lb + (rb - lb + 1) / 2;
    return max(qry(ql, qr, 2 * node, lb, md), qry(ql, qr, 2 * node + 1, md, rb));
  }
  void upd(int ql, int qr, T x, int node = 1, int lb = 0, int rb = -1) {
    if (rb == -1) rb = size;
    if (qr <= lb or rb <= ql) return;
    if (ql <= lb && rb <= qr) {
      lazy[node] = max(lazy[node], x);
      tree[node] = max(tree[node], x);
    } else {
      push(node, lb, rb);
      int md = lb + (rb - lb + 1) / 2;
      upd(ql, qr, x, 2 * node, lb, md);
      upd(ql, qr, x, 2 * node + 1, md, rb);
      tree[node] = max(tree[2 * node], tree[2 * node + 1]);
    }
  }
  void push(int node, int lb, int rb) {
    if (lazy[node]) {
      int md = lb + (rb - lb + 1) / 2;
      upd(lb, rb, lazy[node], 2 * node, lb, md);
      upd(lb, rb, lazy[node], 2 * node + 1, md, rb);
      lazy[node] = 0;
    }
  }
};
```

2.6 Fenwick Tree

Zero-indexed, segments include end: ($[a, b]$): `[i - (i & -i) + 1, i]` and `[i, i + (i & -i)]`
TC: $O(\log n)$ **SC:** $O(n)$

```
struct Fentree {
  typedef int T;
  vector<T> tree;
  int size;

  Fentree(int sz, T def = 0) : size(sz), tree(sz + 1) {
    for (int i = 1; i <= size; ++i) tree[i] = def * (i & -i);
  }
  Fentree(vector<T> &data) : size(data.size()), tree{0, data[0]} {
    tree.reserve(size + 1);
    vector<T> pref;
    pref.reserve(size); pref.push_back(data[0]);
    for (int i = 2; i <= data.size(); ++i) {
      pref.push_back(pref.back() + data[i - 1]);
      tree.push_back(pref.back());
      if (i != __bit_floor(i)) tree.back() -= pref[i - (i & -i) - 1];
    }
  }
  T get(int i) { return qry(i, i); }
  T qry(int rb) {
    int res = 0;
    for (rb += 1; rb > 0; rb -= (rb & -rb)) res += tree[rb];
    return res;
  }
  T qry(int lb, int rb) { return qry(rb) - qry(lb - 1); }
  void upd(int i, T val) {
    for (i += 1; i <= size; i += (i & -i)) tree[i] += val;
  }
  void set(int i, T val) { upd(i, val - get(i)); }
  void append(T val) {
    tree.push_back(val);
    int rb = tree.size() - 1;
    int lb = rb - (rb & -rb) + 1;
    if (1 <= lb and lb <= rb - 1) tree.back() += qry(lb, rb - 1);
  }
  int upper_bound(T x) {
    int i = 0;
    for (int pw = __bit_floor(tree.size()); pw >= 1; pw >>= 1) {
      if (i + pw < tree.size() and tree[i + pw] <= x) {
        i += pw;
        x -= tree[i];
      }
    }
    return i;
  }
};
```

2.7 2D Fenwick Tree

The same, but on the plane. **TC:** $O(\log n \log m)$ **SC:** $O(nm)$

```
struct Fenwick2D {
  typedef int T;
  vector<vector<T>> tree;
  int y_size, x_size;

  Fenwick2D(int y_sz, int x_sz)
  : y_size(y_sz), x_size(x_sz), tree(y_sz + 1, vector<T>(x_sz + 1, 0)) {}
  T get(int y, int x) { return qry(y, y, x, x); }
  T qry(int yr, int xr) {
    if (yr < 0 or xr < 0) return 0;
    T res = 0;
    for (int iy = yr + 1; iy > 0; iy -= iy & -iy)
      for (int ix = xr + 1; ix > 0; ix -= ix & -ix) res += tree[iy][ix];
    return res;
  }
  T qry(int yl, int yr, int xl, int xr) {
    return qry(yr, xr) - qry(yl - 1, xr) - qry(yr, xl - 1) + qry(yl - 1, xl - 1);
  }
  void upd(int y, int x, T val) {
    for (int iy = y + 1; iy <= y_size; iy += iy & -iy)
      for (int ix = x + 1; ix <= x_size; ix += ix & -ix) tree[iy][ix] += val;
  }
  void set(int y, int x, T val) { upd(y, x, val - get(y, x)); }
};
```

2.8 RMQ

Query minimum on any range in constant time. Preproc: **TC:** $O(n \log n)$ **SC:** $O(n \log n)$

```
struct RMQ {
  typedef int T;
  vector<vector<T>> seg;
  RMQ(vector<T> &data) : seg(1, data) {
    for (int pw = 1, k = 1; 2 * pw <= data.size(); pw *= 2, ++k) {
```

```
      seg.emplace_back(data.size() - 2 * pw + 1);
      for (int i = 0; i < seg.back().size(); ++i)
        seg[k][i] = min(seg[k - 1][i], seg[k - 1][i + pw]);
    }
  }
  T qry(int lb, int rb) {
    int pw = 31 - __builtin_clz(rb - lb);
    return min(seg[pw][lb], seg[pw][rb - (1 << pw)]);
  }
};
```

3 Sorting

3.1 Mergesort

Merge sort. **TC:** $O(n \log n)$ **SC:** $O(n)$

```
void mergesort(vi &nums, vi &buff, int lb, int rb) {
  if (rb - lb <= 1) return;
  int md = lb + (rb - lb) / 2;
  mergesort(nums, buff, lb, md);
  mergesort(nums, buff, md, rb);
  copy(nums.BEG + lb, nums.BEG + md, buff.BEG + lb);
  auto lhs = buff.BEG + lb, lhs_end = buff.BEG + md;
  auto rhs = nums.BEG + md, rhs_end = nums.BEG + rb;
  auto target = nums.BEG + lb;
  while (lhs != lhs_end or rhs != rhs_end) {
    if (lhs == lhs_end) *target = *(rhs++);
    else if (rhs == rhs_end) *target = *(lhs++);
    else if (*lhs <= *rhs) *target = *(lhs++);
    else *target = *(rhs++);
    target++;
  }
}
```

3.2 Radixsort

Decimal radix sort. **TC:** $O(n)$ **SC:** $O(n)$

```
void radixsort(vi &nums) {
  if (nums.size() <= 1) return;
  int max_rank = 1;
  for (int num : nums) max_rank = max(max_rank, (int)ceil(log10(abs(num) + 1)));
  vi sorted(nums.size());
  for (int rank = 0, power = 1; rank < max_rank; ++rank, power *= 10) {
    vi digit_count(10, 0);
    for (int num : nums) {
      int digit = (num / power) % 10; digit_count[digit] += 1;
    }
    for (int i = 0, digit_count_accum = 0; i < 10; ++i) {
      int tmp = digit_count[i];
      digit_count[i] = digit_count_accum;
      digit_count_accum += tmp;
    }
    for (int num : nums) {
      int digit = (num / power) % 10;
      sorted[digit_count[digit]] = num;
      digit_count[digit]++;
    }
    nums = sorted;
  }
}
```

3.3 Quickselect

Select k order statistic. **TC:** $O(n)$ **SC:** $O(1)$

```
int quickselect(vi &nums, int k, int lb, int rb) {
  if (rb - lb == 1) return nums[lb];
  int pivot_idx = lb + rng() % (rb - lb);
  int pivot = nums[pivot_idx];
  swap(nums[lb], nums[pivot_idx]);
  int eq = lb, gr = lb + 1;
  for (int i = lb + 1; i < rb; ++i) {
    int ii = i;
    if (nums[ii] <= pivot) {
      swap(nums[gr], nums[ii]); ii = gr++;
    }
    if (nums[ii] < pivot) swap(nums[eq++], nums[ii]);
  }
  if (k < eq - lb) return quickselect(nums, k, lb, eq);
  if (k < gr - lb) return pivot;
  return quickselect(nums, k - (gr - lb), gr, rb);
}
```

3.4 Binary search

First index to insert element to keep array sorted. **TC:** $O(\log n)$ **SC:** $O(1)$

```
int lower_bound(vi &nums, int target) {
  int lb = 0, rb = nums.size();
  while (lb < rb) {
    int md = lb + (rb - lb) / 2;
    if (nums[md] < target) lb = md + 1;  // <= for upper_bound
    else rb = md;
  }
  return lb;
}
```

4 Strings

4.1 Prefix function

Given string `s`, $\forall$ `i` finds max `n` s.t. `s[:n] == s[i-n+1 : i+1]`. **TC:** $O(n)$ **SC:** $O(n)$

```
vi pref_func(string &s) {
  vi pref(s.size(), 0);
  for (int i = 1; i < s.size(); ++i) {
    int c = pref[i - 1];
    while (c > 0 and s[i] != s[c]) c = pref[c - 1];
    if (s[i] == s[c]) pref[i] = c + 1;
  }
  return pref;
}
vi pref_func(string &s, string &t) {
  vi t_pref(t.size(), 0), pref(s.size(), 0);
  for (int i = 1; i < t.size(); ++i) {
```

```
    int c = t_pref[i - 1];
    while (c > 0 and t[i] != t[c]) c = t_pref[c - 1];
    if (t[i] == t[c]) t_pref[i] = c + 1;
  }
  if (!s.empty() and !t.empty()) pref[0] = (s[0] == t[0]) ? 1 : 0;
  for (int i = 1; i < s.size(); ++i) {
    int c = pref[i - 1];
    while (c > 0 and s[i] != t[c] or c == t.size()) c = t_pref[c - 1];
    if (s[i] == t[c]) pref[i] = c + 1;
  }
  return pref;
}
```

4.2 Z function

Given string `s`, $\forall$ `i` finds max `n` s.t. `s[:n] == s[i : i + n]`. **TC:** $O(n)$ **SC:** $O(n)$

```
vi z_func(string &s) {
  int lb = 0, rb = 0;
  vi z(s.size(), 0);
  for (int i = 1; i < s.size(); ++i) {
    if (i <= rb) z[i] = min(z[i - lb], rb - i + 1);
    while (i + z[i] < s.size() and s[i + z[i]] == s[z[i]]) z[i]++;
    if (i + z[i] - 1 > rb) lb = i, rb = i + z[i] - 1;
  }
  return z;
}
vi z_func(string &s, string &t) {
  int lb = 0, rb = 0;
  vi tz(t.size(), 0), z(s.size(), 0);
  for (int i = 1; i < t.size(); ++i) {
    if (i <= rb) tz[i] = min(tz[i - lb], rb - i + 1);
    while (i + tz[i] < t.size() and t[i + tz[i]] == t[tz[i]]) tz[i]++;
    if (i + tz[i] - 1 > rb) lb = i, rb = i + tz[i] - 1;
  }
  while (z[0] < s.size() and z[0] < t.size() and t[z[0]] == s[z[0]]) z[0]++;
  lb = 0, rb = z[0] - 1;
  for (int i = 1; i < s.size(); ++i) {
    if (i <= rb) z[i] = min(tz[i - lb], rb - i + 1);
    while (i + z[i] < s.size() and z[i] < t.size() and s[i + z[i]] == t[z[i]]) z[i]++;
    if (i + z[i] - 1 > rb) lb = i, rb = i + z[i] - 1;
  }
  return z;
}
```

4.3 Manacker algorithm

Given string `s`, $\forall$ `i` finds max `n` s.t. `s[:n] == s[i-n+1 : i+1]`. **TC:** $O(n)$ **SC:** $O(n)$

```
vi manacker_odd(string &s) {
  vi rad(s.size(), 1);
  int lb = 0, rb = 0;
  for (int i = 1; i < s.size(); ++i) {
    if (i <= rb) rad[i] = min(rad[lb + rb - i], rb - i + 1);
    while (0 <= i - rad[i] and i + rad[i] < s.size() and
           s[i - rad[i]] == s[i + rad[i]]) rad[i]++;
    if (rb < i + rad[i] - 1) lb = i - rad[i] + 1, rb = i + rad[i] - 1;
  }
  return rad;
}
vi manacker_even(string &s) {
  vi rad(s.size() - 1, 0);
  int lb = -1, rb = -1;
  for (int i = 0; i < s.size(); ++i) {
    if (i <= rb) rad[i] = min(rad[lb + rb - i - 1], rb - i);
    while (0 <= i - rad[i] and i + rad[i] - 1 < s.size() and
           s[i - rad[i]] == s[i + rad[i] - 1]) rad[i]++;
    if (rb < i + rad[i]) lb = i - rad[i] + 1, rb = i + rad[i];
  }
  return rad;
}
```

4.4 Hashing

Try using 31-bit 970’592’641, 45-bit 31’443’539’979’727 or 52-bit 3’006’703’054’056’749

```
int LIM;
struct Power {
  ll B, M;
  vector<ll> val;
  Power(ll B, ll M) : B(B), M(M), val(LIM + 1) {
    val[0] = 1;
    for (int i = 1; i <= LIM; ++i) val[i] = (val[i - 1] * B) % M;
  }
  ll operator()(size_t i) { return val[i]; };
};
struct Hasher {
  Power &pw1, &pw2;
  vector<ll> ha1, ha2;
  Hasher(Power &pw1, Power &pw2, string &s)
  : pw1(pw1), pw2(pw2), ha1(s.size() + 1), ha2(s.size() + 1) {
    ha1[0] = ha2[0] = 0;
    for (int i = 1; i <= s.size(); ++i) {
      ha1[i] = (ha1[i - 1] + (s[i - 1] - 'a' + 1) * pw1(i)) % pw1.M;
      ha2[i] = (ha2[i - 1] + (s[i - 1] - 'a' + 1) * pw2(i)) % pw2.M;
    }
  }
  pll operator()(int l, int r) {
    return {
      ((ha1[r + 1] + pw1.M - ha1[l]) % pw1.M) * pw1(LIM - r) % pw1.M,
      ((ha2[r + 1] + pw2.M - ha2[l]) % pw2.M) * pw2(LIM - r) % pw2.M,
    };
  }
};
```

4.5 Prefix Tree

Efficiently stores strings as tree. **TC:** $O(\max(n_i))$ **SC:** $O(\sum n_i)$

```
static char buf[450 << 20];  // 450mb
void *operator new(size_t s) {
  static size_t i = sizeof buf;
  return (void *)&buf[i -= s];
}
void operator delete(void *) {}

const int K = 26;
struct Node {
```

```
  Node* suc[K] = {0};
  int cnt = 0;
};
struct Trie {
  Node root;

  Trie() = default;
  void insert(string& s) {
    auto node = &root;
    for (char c : s) {
      if (node->suc[c - 'a'] == 0) node->suc[c - 'a'] = new Node;
      node = node->suc[c - 'a'];
    }
    node->cnt++;
  }
  int count(string& s) {
    auto node = search(s);
    return node == 0 ? 0 : node->cnt;
  }
  Node* search(string& s) {
    auto node = &root;
    for (char c : s) {
      if (node->suc[c - 'a'] == 0) return 0;
      node = node->suc[c - 'a'];
    }
    return node;
  }
};
```

5 Graphs

5.1 Dijkstra Algorithm

Shortest path lengths to all vertices from `src` given all the edges are of positive cost
TC: $O(n \log m)$ **SC:** $O(max(n, m))$

```
vi dijkstra(vector<vector<pii>> &graph, int src) {
  int n = graph.size();
  vi dist(n, INT32_MAX);
  dist[src] = 0;
  priority_queue<pii, vpii, greater<>> pq;
  pq.emplace(0, src);
  while (!pq.empty()) {
    while (!pq.empty() and dist[pq.top().S] <= pq.top().F) pq.pop();
    if (pq.empty()) continue;
    auto [d, v] = pq.top();
    pq.pop();
    dist[v] = d;
    for (auto [to, len] : graph[v]) pq.emplace(d + len, to);
  }
  return dist;
}
```

5.2 Shortest Path Faster

Shortest path lengths to all vertices from `src` with negative weights allowed. If negative cycle exists, `false` returned. **TC:** $O(m)$ for random graphs, $O(nm)$ in worst cases

```
pair<vi, bool> spf(vector<vector<pii>> &graph, int src) {
  int n = graph.size();
  vi dist(n, INT32_MAX), cnt(n, 0); dist[src] = 0;
  vector<bool> inqueue(n, false); inqueue[src] = true;
  queue<int> q; q.push(src);

  while (!q.empty()) {
    int v = q.front(), d = dist[q.front()];
    q.pop();
    inqueue[v] = false;
    for (auto [to, len] : graph[v]) {
      if (d + len >= dist[to]) continue;
      dist[to] = d + len;
      if (inqueue[to]) continue;
      q.push(to);
      inqueue[to] = true;
      cnt[to]++;
      if (cnt[to] > n) return {dist, false};  // negative cycle
    }
  }
  return {dist, true};
}
```

5.3 Floyd-Warshall

Shortest paths between all pairs of vertices. The graph may have negative weights, but no negative weight cycles. **TC:** $O(n^3)$

```
vvi floyd_warshall(vvi &graph) {
  int n = graph.size();
  auto dist(graph);
  for (int i = 0; i < n; ++i) dist[i][i] = 0;
  for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        if (dist[i][k] != INT32_MAX and dist[k][j] != INT32_MAX and
            dist[i][j] < dist[i][k] + dist[k][j]) {
          dist[i][j] = dist[i][k] + dist[k][j];
          dist[j][i] = dist[i][j];
        }
      }
    }
  }
  return dist;
}
```

5.4 Euler Path

Path containing all the edges. The graph must be connected and degrees of $\geq n-2$ vertices must be even. Don't forget to reverse resulting vector. **TC:** $O(m)$

```
void euler(vector<hashset<int>> &graph, vi &path, int src = 0) {
  while (!graph[src].empty()) {
    int dst = *graph[src].begin();
    graph[src].erase(dst);
    graph[dst].erase(src);  // only for directed graph
    euler(graph, path, dst);
  }
```

```
  path.push_back(src);
}
```

5.5 Hamilton Path

Path containing all the vertices. That's NP-complete problem so n must be really small. **TC:** $O(2^n)$

```
int shortest_hamilton(int mask, int end, vvi &dp, vector<hashmap<int, int>> &graph) {
  if ((mask & (mask - 1)) == 0) return 0;
  if (dp[mask][end] != -1) return dp[mask][end];
  int len = INT32_MAX;
  for (int bit = 0; bit < 32; ++bit) {
    if ((mask & (1 << bit)) == 0) continue;
    if (graph[bit].count(end) == 0) continue;
    len = min(len, shortest_hamilton(mask & ~(1 << end), bit, dp, graph) +
               graph[bit].find(end)->second);
  }
  return dp[mask][end] = len;
}
int count_hamiltons(int mask, int end, vvi &dp, vector<hashset<int>> &graph) {
  if ((mask & (mask - 1)) == 0) return 1;
  if (dp[mask][end] != -1) return dp[mask][end];
  int cnt = 0;
  for (int bit = 0; bit < 32; ++bit) {
    if ((mask & (1 << bit)) == 0) continue;
    if (graph[bit].count(end) == 0) continue;
    cnt += count_hamiltons(mask & ~(1 << end), bit, dp, graph);
  }
  return dp[mask][end] = cnt;
}
int check_hamilton(int mask, vi &dp, vi &graph) {
  if ((mask & (mask - 1)) == 0) return mask;
  if (dp[mask] != -1) return dp[mask];
  int ans = 0;
  for (int bit = 0; bit < 32; ++bit) {
    if ((mask & (1 << bit)) == 0) continue;
    if ((check_hamilton(mask & ~(1 << bit), dp, graph) & graph[bit]) != 0)
      ans += 1 << bit;
  }
  return dp[mask] = ans;
}
```

5.6 Topological Sort

Orders vertices s.t. all edges go from left to right

```
vi toposort(vvi &graph) {
  vi order; order.reserve(graph.size());
  vector<bool> seen(graph.size(), false);
  for (int v = 0; v < graph.size(); ++v) if (!seen[v]) dfs(graph, seen, order, v);
  reverse(all(order));
  return order;
}
void dfs(vvi &graph, vector<bool> &seen, vi &order, int v) {
  seen[v] = true;
  for (int to: graph[v]) if (!seen[to]) dfs(graph, seen, order, to);
  order.push_back(v);
}
```

5.7 Stronly Connected Components

Splits graph into set of components, s.t. within one components there are paths in both directions between any two vertices. Compressing every component into single vertex makes graph acyclic. **TC:** $O(n)$

```
pair<vi, vi> scc(vvi &graph) {
  int n = graph.size();
  vvi trans(n);
  vi order; order.reserve(n);
  vector<bool> seen(n, false);
  for (int v = 0; v < n; ++v) {
    for (int to: graph[v]) trans[to].push_back(v);
    if (!seen[v]) dfs(graph, seen, order, v);
  }
  reverse(all(order));
  vi component(n, -1);
  int n_components = 0;
  for (int v: order)
    if (component[v] == -1) tdfs(trans, component, n_components, v), n_components++;
  return {component, order};
}
void dfs(vvi &graph, vector<bool> &seen, vi &order, int v) {
  seen[v] = true;
  for (int to: graph[v]) if (!seen[to]) dfs(graph, seen, order, to);
  order.push_back(v);
}
void tdfs(vvi &trans, vi &component, int n_components, int v) {
  component[v] = n_components;
  for (int to: trans[v])
    if (component[to] == -1) tdfs(trans, component, n_components, to);
}
```

5.8 Bridge Edges & Cut Vertices

Given undirected graph, finds edges s.t. removing each of them increases number of connected components. **TC:** $O(O(n + m))$

```
vvi graph; vector<bool> seen; vi tin, low; int timer = 0;

void find_bridges() {
  seen.assign(graph.size(), false);
  tin.assign(graph.size(), -1), low.assign(graph.size(), -1);
  for (int v = 0; v < graph.size(); ++v)
    if (!seen[v]) dfs(v);
}
void dfs(int v, int p = -1) {
  seen[v] = true;
  tin[v] = low[v] = timer++;
  for (int to: graph[v]) {
    if (to == p) continue;
    if (seen[to]) {
      low[v] = min(low[v], tin[to]);
    } else {
      dfs(to, v);
      low[v] = min(low[v], low[to]);
```

```
      if (low[to] > tin[v]) ITS_BRIDGE(v, to);
    }
  }
}
```

Small change in `dfs` needed to find cut vertices s.t. removing each of them along with their edges increases number of connected components:

```
void dfs(int v, int p = -1) {
  seen[v] = true;
  tin[v] = low[v] = timer++;
  int children = 0;
  for (int to: graph[v]) {
    if (to == p) continue;
    if (seen[to]) {
      low[v] = min(low[v], tin[to]);
    } else {
      dfs(to, v);
      low[v] = min(low[v], low[to]);
      if (low[to] > tin[v] and p != -1) ITS_CUTPOINT(v);
      children++;
    }
  }
  if (p == -1 and children > 1) ITS_CUTPOINT(v);
}
```

6 Math

6.1 Equations

$$ax^2 + bx + c = 0 \implies x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},\ x_{extr} = -\frac{b}{2a},\ y_{extr} = -\frac{b^2}{4a} + c$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Given an equation $Ax = b$, find $x_i = \frac{\det A'_i}{\det A}$, where A'_i is A with i'th column replaced by b

6.2 LU Decomposition

Matrix as product of lower triangular matrix and upper triangular matrix. **TC:** $O(n^3)$

```
pair<vii, vii> LU(vii &A) {
  int n = A.size();
  vii L(n, vi(n, 0)), U(n, vi(n, 0));
  for (int i = 0; i < n; ++i) L[i][i] = 1;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      if (i <= j) {
        U[i][j] = A[i][j];
        for (int k = 0; k < i; ++k) U[i][j] -= L[i][k] * U[k][j];
      } else {
        L[i][j] = A[i][j];
        for (int k = 0; k < j; ++k) L[i][j] -= L[i][k] * U[k][j];
        L[i][j] /= U[j][j];
      }
    }
  }
  return {L, U};
}
```

6.3 Polynomials

Instantiate polynomial from it's coefficients or from given point set using interpolation. Find it's root but beware that it may give more roots to the left or to the right

```
struct Poly {
  typedef double D;
  typedef vector<D> vd;
  vd cf;

  Poly(vd &_cf) : cf(_cf) {}
  Poly(vd &x, vd &y) {
    int n = x.size();
    cf.resize(n);
    vd temp(n);
    for (int k = 0; k < n - 1; ++k)
      for (int i = k + 1; i < n; ++i) y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0;
    temp[0] = 1;
    for (int k = 0; k < n; ++k) {
      for (int i = 0; i < n; ++i) {
        cf[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
      }
    }
  }
  double eval(double x) {
    double y = 0;
    for (int i = cf.size() - 1; i >= 0; --i) (y *= x) += cf[i];
    return y;
  }
  void diff() {
    for (int i = 1; i < cf.size(); ++i) cf[i - 1] = i * cf[i];
    cf.pop_back();
  }
  // may give more roots to left or to the right
  vd roots(double xmin, double xmax) {
    if (cf.size() == 2) return {-cf[0] / cf[1]};
    vd roots;
    Poly deriv(*this);
    deriv.diff();
    vd dr = deriv.roots(xmin, xmax);
    dr.push_back(xmin - 1);
    dr.push_back(xmax + 1);
    sort(dr.begin(), dr.end());
    for (int i = 0; i < dr.size() - 1; ++i) {
      double lb = dr[i], rb = dr[i + 1];
      bool sign = eval(lb) > 0;
      if (sign ^ (eval(rb) > 0)) {
        for (int it = 0; it < 60; ++it) {
          double md = (lb + rb) / 2;
          if ((eval(md) <= 0) ^ sign) lb = md;
          else rb = md;
```

```
        }
        roots.push_back((lb + rb) / 2);
      }
    }
    return roots;
  }
};
```

6.4 Extremums

Finds minimum of f on $[a, b]$ where f has one local minima **TC:** $O((b-a)/\varepsilon)$

```
double gss(double a, double b, double (*f)(double)) {
  double r = (sqrt(5) - 1) / 2, eps = 1e-6;
  double x1 = b - r * (b - a), x2 = a + r * (b - a);
  double f1 = f(x1), f2 = f(x2);
  while (b - a > eps) {
    if (f1 < f2) {  // > for maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - r * (b - a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + r * (b - a); f2 = f(x2);
    }
  }
  return a;
}
```

6.5 Sums

$$c^a + c^{a+1} + \ldots + c^b = \frac{c^{b+1} - c^a}{c - 1},\ c \neq 1$$

$$1 + \ldots + n = \frac{n(n+1)}{2} \qquad 1^2 + \ldots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + \ldots + n^3 = \frac{n^2(n+1)^2}{4} \qquad 1^4 + \ldots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

6.6 Trigonometry

$$\sin(a \pm b) = \sin a \cos b \pm \sin b \cos a \quad \cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$$

$$\tan(a \pm b) = \frac{\tan a \pm \tan b}{1 \mp \tan a \tan b} \quad \cot(a \pm b) = \frac{\cot a \cot b \mp 1}{\cot a \pm \cot b}$$

$$\sin a \pm \sin b = 2 \sin \frac{a \pm b}{2} \cos \frac{a \mp b}{2} \quad \cos a + \cos b = 2 \cos \frac{a+b}{2} \cos \frac{a-b}{2}$$

$$(A+B)\frac{\tan(a-b)}{2} = (A-B)\frac{\tan(a+b)}{2}, \text{ where } A, B - \text{sides opposite to } a, b$$

$$(x, y) + \angle\varphi = (x\cos\varphi - y\sin\varphi, x\sin\varphi + y\cos\varphi)$$

6.7 Number theory

Bézout's identity

$$a, b \in \mathbb{Z} \implies \exists x, y \in \mathbb{Z} : xa + yb = \gcd(a, b)$$

$$a, b \in \mathbb{N} \implies \exists x, y \in \mathbb{N} : xa - yb = \gcd(a, b)$$

$$\text{if } (x, y) - \text{ solution, then all the solutions: } \left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), k \in \mathbb{Z}$$

Euclid Extended

Finds gcd and Bézout coefficients

```
int euclid(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = euclid(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

Chinese Remainder Theorem

Finds $x : \ x \equiv a \pmod m,\ x \equiv b \pmod n$. If $|a| < m,\ |b| < n \implies 0 \le x < \operatorname{lcm}(m, n)$. Assumes $mn < 2^{62}$. **TC:** $O(\log n)$

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y; ll g = euclid(m, n, x, y);
  if ((a - b) % g != 0) return -1;  // no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m * n / g : x;
}
```

Modular Arithmetics

- $(a + b) \% m = ((a \% m) + (b \% m)) \% m$
- $(a - b) \% m = ((a \% m) - (b \% m) + m) \% m$
- $(a \cdot b) \% m = ((a \% m) \cdot (b \% m)) \% m$
- $(b/a) \% m = ((b \% m) \cdot (a^{-1} \pmod m)) \% m$

Long Modular Mul, Pow

For `ull`: $0 \le a, b \le m \le 7.2 \cdot 10^{18}$

```
ull modmul(ull a, ull b, ull mod) {
  ll ret = a * b - mod * ull(1.L / mod * a * b);
  return ret + mod * (ret < 0) - mod * (ret >= (ll)mod);
}
```

```
ull modpow(ull a, ull b, ull mod) {
  ull ret = 1;
  for (; b; a = modmul(a, a, mod), b /= 2)
    if (b & 1) ret = modmul(ret, a, mod);
  return ret;
}
```

Inverting By Modulo

Computes $a^{-1} \pmod{m}$. **Note that m must be prime or coprime with a:**

```
ll invert(ll a) {
  ll x, y; ll d = euclid(a, MOD, x, y);
  return (x + MOD) % MOD;
}
```

Primes

Miller-Rabin primality test for numbers $< 7 \cdot 10^{18}$. **TC:** $O(1)$

```
ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
bool is_prime(ull num) {
  if (num < 2 or num % 6 % 4 != 1) return (num | 1) == 3;
  ull s = __builtin_ctzll(num - 1);
  ull d = num >> s;
  for (ull a: A) {
    ull p = modpow(a % num, d, num), i = s;
    while (p != 1 and p != num - 1 and a % num and i--)
      p = modmul(p, p, num);
    if (p != num - 1 and i != s) return false;
  }
  return true;
}
```

Generating primes

Eratosthenes sieve. Note that `S = (int)round(sqrt(LIM))`

```
const int LIM = 1e6, S = 1e3;
bitset<LIM> is_prime;
vi eratosthenes() {
  const int R = LIM / 2;
  vi pr = {2}, sieve(S + 1);
  pr.reserve(int(LIM / log(LIM) * 1.1));
  vector<pii> cp;
  for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
    cp.push_back({i, i * i / 2});
    for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
  }
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto &[p, idx] : cp)
      for (int i = idx; i < S + L; idx = (i += p))
        block[i - L] = 1;
    for (int i = 0; i < min(S, R - L); ++i)
      if (!block[i]) pr.push_back((L + i) * 2 + 1);
  }
  for (int i : pr) is_prime[i] = 1;
  return pr;
}
```

Factorization

ρ-Pollard factorization, factors are returned in arbitrary order. **TC:** $O(\sqrt[4]{n})$

```
ull pollard(ull num) {
  auto f = [num](ull x) { return modmul(x, x, num) + 1; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 or __gcd(prd, num) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x, y) - min(x, y), num))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, num);
}
vector<ull> factor(ull num) {
  if (num == 1) return {};
  if (is_prime(num)) return {num};
  ull x = pollard(num);
  auto L = factor(x), R = factor(num / x);
  L.insert(L.end(), R.begin(), R.end());
  return L;
}
```

Divisibility

- $n \,\vdots\, 3 \text{ (or } 9) \iff \sum\{\text{digits}\} \,\vdots\, 3 \text{ (or } 9)$
- $n \,\vdots\, 7 \iff (\sum\{\text{even digits}\} - \sum\{\text{odd digits}\}) \,\vdots\, 7$
- $n \,\vdots\, 11 \iff (\sum\{\text{even digits}\} - \sum\{\text{odd digits}\}) \,\vdots\, 11 \text{ or } = 0$

Pythagorean triples

Given two numbers m, n: $m > n > 0$, m and n are coprime, $m - n$ is odd, generate triple: $a = m^2 - n^2, b = 2mn, c = m^2 + n^2$. In such way triples are unique. When multipliying triples by $k > 0$, it's no longer guaranteed.

6.8 Combinatorics

Combinations

$$C_n^k = \frac{n!}{k!(n-k)!} \qquad C_n^k = \frac{n}{k} C_{n-1}^{k-1} \qquad \sum_{r=k}^{n} C_r^k = C_{n+1}^{k+1}$$

$$\overline{C_n^k} = C_{n+k-1}^k$$

$$C_n^0 = C_n^n = 1 \qquad \sum_{k=0}^{n} C_n^k = 2^n \qquad \sum_{k=0}^{n} \left(C_n^k\right)^2 = C_{2n}^n$$

$$C_n^k = C_n^{n-k}$$

$$C_{n-1}^{k-1} + C_{n-1}^k = C_n^k \qquad \sum_{r=0}^{k} C_m^r \cdot C_n^{k-r} = C_{n+m}^k \qquad \sum_{k=0}^{m} (-1)^k \cdot C_n^k = (-1)^m \cdot C_{n-1}^m$$

Computes C_n^k modulo m. Precomputes factorials and reversed factorials. **TC:** $O(n)$

```
vector<ll> fac(LIM), rev(LIM);
fac[0] = 1;
for (int i = 1; i < LIM; i++) fac[i] = i * fac[i - 1] % MOD;
```

```
rev.back() = inv(fac.back())
for (int i = LIM - 1; i >= 1; i--) rev[i-1] = rev[i] * i % MOD;

ll C(int n, int k) { return fac[n] * rev[k] % MOD * rev[n - k] % MOD; }
```

Catalan numbers

$$C_0 = 1,\ C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1},\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n = \sum C_i C_{n-i}$$

- ⋆ Valid parentheses string of length $2n$
- ⋆ Binary trees with n internal nodes ($n+1$ leaves)
- ⋆ Paths from $(0,0)$ to $(2n,0)$ with steps $(1,1)$, $(1,-1)$ above X-axis
- ⋆ Triangulations of polygon with $n+2$ vertices
- ⋆ Connections of $2n$ points on circle into n non-crossing chords
- ⋆ Decomposition of n into sum of positives (order matters)
- ⋆ Monopaths in grid $n \times n$ from left-top to right-bottom not crossing diagonal
- ⋆ Tuples $(x_1 \dots x_n) : x_1 = 1,\ x_i \le x_{i-1} + 1$

7 Geometry

7.1 Points

Point in the plane. For better performance use `ll` instead of `double`, but remember about rounds and possible overflows. When comparing `double`s remember about `EPS`.

```
typedef double D;
struct P {
  D x, y;
  P(D _x = 0, D _y = 0) : x(_x), y(_y) {}
  bool operator==(P p) { return tie(x, y) == tie(p.x, p.y); }
  P operator+(P p) const { return P(x + p.x, y + p.y); }
  P operator-(P p) const { return P(x - p.x, y - p.y); }
  P operator*(D d) const { return P(x * d, y * d); }
  P operator/(D d) const { return P(x / d, y / d); }
  D dot(P p) { return x * p.x + y * p.y; }
  D cross(P p) { return x * p.y - y * p.x; }
  D cross(P a, P b) { return (a - *this).cross(b - *this); }
  D dist2() { return x * x + y * y; }
  D dist() { return sqrt(dist2()); }
  double angle() { return atan2(y, x); }
  P unit() { return *this / dist(); }
  P perp() { return P(-y, x); }
  P normal() { return perp().unit(); }
  P rotate(double a) {
    return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
  }
};

int sgn(D x) { return (x > 0) - (x < 0); }
static const D EPS = numeric_limits<D>::epsilon();
using VP = vector<P>;
```

Closest Points

The closest pair of points, i.e. with minimum distance.

```
pair<P, P> closest_points(VP ps) {
  assert(ps.size() > 1);
  set<P> S;
  sort(all(ps), [](P &a, P &b) { return a.y < b.y; });
  pair<D, pair<P, P>> clos{numeric_limits<D>::max(), {P(), P()}};
  int j = 0;
  for (auto &p : ps) {
    P d{1. + sqrt(clos.first), 0.};
    while (ps[j].y <= p.y - d.x) S.erase(ps[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo) clos = min(clos, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return clos.second;
}
```

7.2 Lines

Point – Line distance

Signed distance from point `p` to line containing points `s, e`. Distance >0 on the left side as seen from `s` towards `e`.

```
D line_dist(P s, P e, P p) {
  assert(s != e);
  return (e - s).cross(p - s) / (e - s).dist();
}
```

Point – Segment Distance

Shortest unsigned distance between point `p` and the line segment `[s, e]`.

```
D segment_dist(P s, P e, P p) {
  if (s == e) return (p - s).dist();
  D d = (e - s).dist2(), t = min(d, max(.0, (p - s).dot(e - s)));
  return ((p - s) * d - (e - s) * t).dist() / d;
}
```

Lines Intersection

If the lines `s1-e1` and `s2-e2` have single intersection point then `{1, point}` returned. If no intersection point exists `{0, (0, 0)}` returned. If infinitely many exist `{-1, (0, 0)}` returned.

```
pair<int, P> line_isect(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, s2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

Segments Intersection

If the line segments `[s1, e1]` and `[s2, e2]` have single intersection point then it's returned. If no intersection point exists an empty vector returned. If infinitely many exist a vector of 2 endpoints of common segment returned.

```
VP segment_isect(P s1, P e1, P s2, P e2) {
  D os1 = s2.cross(e2, s1), oe1 = s2.cross(e2, e1);
  D os2 = s1.cross(e1, s2), oe2 = s1.cross(e1, e2);
  if (sgn(os1) * sgn(oe1) < 0 and sgn(os2) * sgn(oe2) < 0)
    return {(s1 * oe1 - e1 * os1 / (oe1 - os1))};
  set<P> s;
  if (segment_dist(s2, e2, s1) <= EPS) s.insert(s1);
  if (segment_dist(s2, e2, e1) <= EPS) s.insert(e1);
  if (segment_dist(s1, e1, s2) <= EPS) s.insert(s2);
  if (segment_dist(s1, e1, e2) <= EPS) s.insert(e2);
  return {all(s)};
}
```

Side of Point

Where point `p` is as seen from `s` towards `e`: 1/0/-1 $\iff$ left/on line/right.

```
int side_of(P s, P e, P p) {
  D a = (e - s).cross(p - s);
  D q = (e - s).dist() * EPS;
  return (a > q) - (a < -q);
}
```

Linear Transformation

Apply the same linear transformation that takes line `s1-e1` to `s2-e2` to point `p`.

```
P transform(P s1, P e1, P s2, P e2, P p) {
  P d1 = e1 - s1, d2 = e2 - s2, num(d1.cross(d2), d1.dot(d2));
  return s2 + P((p - s1).cross(num), (p - s1).dot(num)) / d1.dist2();
}
```

7.3 Circles

Circles Intersection

The pair of points at which two circles intersect. Returns `false` in case of no intersection.

```
bool circle_isect(P c1, D r1, P c2, D r2, pair<P, P> *out) {
  if (c1 == c2) {
    assert(r1 != r2);
    return false;
  }
  P vec = c2 - c1;
  D d2 = vec.dist2(), sum = r1 + r2, dif = r1 - r2,
    p = (d2 + r1 * r1 - r2 * r2) / (d2 * 2), h2 = r1 * r1 - p * p * d2;
  if (sum * sum < d2 or dif * dif > d2) return false;
  P mid = c1 + vec * p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

Circles Tangents

The external tangents of two circles if `r2 >= 0` or internal if `r2 < 0`. Returns 0, 1 or 2 tangents.

```
vector<pair<P, P>> circle_tangents(P c1, D r1, P c2, D r2) {
  P d = c2 - c1;
  D dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 or h2 < 0) return {};
  vector<pair<P, P>> tan;
  for (D sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    tan.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) tan.pop_back();
  return tan;
}
```

Intersection of Circle with Polygon

The area of the intersection of a circle with CCW polygon. **TC:** $O(n)$ **SC:** $O(1)$

```
#define arg(p, q) atan2((p).cross(q), (p).dot(q))
D circle_poly_isect(P c, D a, VP &pg) {
  auto tri = [&](P p, P q) {
    auto r2 = a * a / 2;
    P d = q - p;
    auto a = d.dot(p) / d.dist2(), b = (p.dist2() - a * a) / d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a - sqrt(det)), t = min(1., -a + sqrt(det));
    if (t < 0 or 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p, u) * r2 + u.cross(v) / 2 + arg(v, q) * r2;
  };
  auto area = 0.;
  for (int i = 0; i < pg.size(); ++i)
    area += tri(pg[i] - c, pg[(i + 1) % pg.size()] - c);
  return area;
}
```

Circumcircle

The circle going intersecting all three points.

```
pair<P, D> circumcircle(P A, P B, P C) {
  P b = C - A, c = B - A, a = C - B;
  P cen = A + (b * c.dist2() - c * b.dist2()).perp() / b.cross(c) / 2;
  D rad = c.dist() * a.dist() * b.dist() / abs(c.cross(b)) / 2;
  return {cen, rad};
}
```

Minimum Enclosing Circle

The minimum circle that encloses a set of points. **TC:** $O(n)$ **SC:** $O(1)$

```
pair<P, D> enclosing_circle(VP &ps) {
  shuffle(all(ps), mt19937(time(0)));
  P cen = ps[0];
  D rad = 0, eps = 1 + 1e-8;
  for (int i = 0; i < ps.size(); ++i) {
    if ((cen - ps[i]).dist() > rad * eps) {
      cen = ps[i], rad = 0;
      for (int j = 0; j < i; ++j)
        if ((cen - ps[j]).dist() > rad * eps) {
          cen = (ps[i] + ps[j]) / 2;
          rad = (cen - ps[i]).dist();
          for (int k = 0; k < j; ++k)
            if ((cen - ps[k]).dist() > rad * eps) {
              cen = circumcircle(ps[i], ps[j], ps[k]).first;
```

```
            rad = (cen - ps[i]).dist();
          }
        }
      }
  }
  return {cen, rad};
}
```

7.4 Polygons

Point Inside Polygon

Checks if `p` lies within the polygon. If `strict=false` then boundary isn't included. **TC:** $O(n)$

```
bool inside_poly(VP &pg, P a, bool strict = true) {
  int cnt = 0, n = pg.size();
  for (int i = 0; i < n; ++i) {
    P &q = pg[(i + 1) % n];
    if (segment_dist(pg[i], q, a) <= EPS) return !strict;
    cnt ^= ((a.y < pg[i].y) - (a.y < q.y)) * a.cross(pg[i], q) > 0;
  }
  return cnt;
}
```

Area of Polygon

Twice the signed area of CCW polygon. CW enumeration gives negative area.

```
D poly_area2(VP &pg) {
  D area = pg.back().cross(pg[0]);
  for (int i = 0; i < pg.size() - 1; ++i) area += pg[i].cross(pg[i + 1]);
  return area;
}
```

Centroid of Polygon

The center of mass for a polygon.

```
P poly_center(VP &pg) {
  P cen(0, 0);
  D area = 0;
  for (int i = 0, j = pg.size() - 1; i < pg.size(); j = i++) {
    cen = cen + (pg[i] + pg[j]) * pg[j].cross(pg[i]);
    area += pg[j].cross(pg[i]);
  }
  return cen / area / 3;
}
```

7.5 Hull

CCW convex hull, points on the edges aren't considered. **TC:** $O(n \log n)$ **SC:** $O(n)$

```
VP convex_hull(VP ps) {
  if (ps.size() <= 1) return ps;
  sort(all(ps));
  VP hull(ps.size() + 1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(ps)))
    for (auto p : ps) {
      while (t >= s + 2 and hull[t - 2].cross(hull[t - 1], p) <= 0) t--;
      hull[t++] = p;
    }
  return {hull.begin(), hull.begin() + t - (t == 2 and hull[0] == hull[1])};
}
```

Convex Hull Diameter

The two points with max distance on a CCW convex hull.

```
pair<P, P> hull_diam(VP &hull) {
  int n = hull.size(), j = n < 2 ? 0 : 1;
  pair<D, pair<P, P>> diam({0, {hull[0], hull[0]}});
  for (int i = 0; i < j; ++i) {
    for (;; j = (j + 1) % n) {
      diam = max(diam, {(hull[i] - hull[j]).dist2(), {hull[i], hull[j]}});
      if ((hull[(j + 1) % n] - hull[j]).cross(hull[i + 1] - hull[i]) >= 0)
        break;
    }
  }
  return diam.second;
}
```

Point Inside Convex Hull

Checks if `p` lies within CCW convex hull. If `strict=false` then boundary isn't included. **TC:** $O(\log n)$

```
bool inside_hull(VP &hull, P p, bool strict = true) {
  int a = 1, b = hull.size() - 1, r = !strict;
  if (hull.size() < 3)
    return r and segment_dist(hull[0], hull.back(), p) <= EPS;
  if (side_of(hull[0], hull[a], hull[b]) > 0) swap(a, b);
  if (side_of(hull[0], hull[a], p) >= r or side_of(hull[0], hull[b], p) <= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (side_of(hull[0], hull[c], p) > 0 ? b : a) = c;
  }
  return sgn(hull[a].cross(hull[b], p)) < r;
}
```

7.6 Some Formulas

Triangles

Given side lengths a, b, c, vertices A, B, C, angles α, β, γ and semiperimeter $p = \frac{a+b+c}{2}$:

$$\text{Area } S = \sqrt{p(p-a)(p-b)(p-c)} \qquad \text{Area } S = \frac{1}{2}|(A-C)\times(B-C)|$$

$$\text{Circumradius } R = \frac{abc}{4S} \qquad \text{Inradius } r = \frac{S}{p}$$

$$\text{Median } m_a = \frac{1}{2}\sqrt{2b^2+2c^2-a^2} \qquad \text{Bisector } s_a = \sqrt{bc\left(1-(b+c)^{-2}\right)}$$

Law of sines $\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} = \frac{1}{2R}$ Law of cosines $a^2 = b^2 + c^2 - 2bc\cos\alpha$

Law of tangents $\frac{a+b}{a-b} = \frac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$ Triple tangent $\frac{a}{\tan\alpha} + \frac{b}{\tan\beta} + \frac{c}{\tan\gamma} = 0$

Quadrilaterals

Given side lengths a, b, c, d, diagonals e, f, diagonals angle θ, semiperimeter p and $F = b^2 + d^2 - a^2 - c^2$

$$4S = 2ef\sin\theta = F\tan\theta = \sqrt{4e^2f^2 - F^2}$$

For cyclic quadrilaterals $ef = ac + bd$, $S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$

Spherical coordinates

$$x = r\sin\theta\cos\varphi \qquad y = r\sin\theta\sin\varphi \qquad z = r\cos\theta$$

$$r = \sqrt{x^2 + y^2 + z^2} \qquad \theta = \operatorname{acos}(z/r) \qquad \varphi = \operatorname{atan2}(y, x)$$

8 Dynamic Programming

8.1 Longest Increasing Subsequence

```
int find_lis(vi &nums) {
  if (nums.size() <= 1) return nums.size();
  vi lis(1, nums.front());
  for (int i = 1; i < nums.size(); ++i) {
    auto ins = lower_bound(lis.begin(), lis.end(), nums[i]);
    if (ins == lis.end()) lis.push_back(nums[i]);
    else *ins = nums[i];
  }
  return lis.size();
}
```

9 Various

9.1 Bit Manipulations

Built-in binary operations for `int`'s. With `long long` use suffix `ll` (except `__lg`)

- ⋆ `__bit_floor` – floor to 2's power
- ⋆ `__bit_ceil` – ceil to 2's power
- ⋆ `__builtin_popcount` – number of 1's
- ⋆ `__builtin_parity` – parity of number of 1's
- ⋆ `__builtin_clz` – number of leading zeros
- ⋆ `__builtin_ctz` – number of trailing zeros
- ⋆ `__builtin_ffs` – index of rightmost 1 (1-indexed)
- ⋆ `__lg` – floor(log2) = index of leftmost 1 (0-indexed)

Optimize `popcount`, `clz` via `#pragma GCC target("popcnt,lzcnt")`

Bit Hacks

- ⋆ `x & -x` – the least bit of `x`
- ⋆ `for (int x = m; x;) { --x &= m; ... }` – loop over all subset masks of `m`
- ⋆ `c = x & -x; r = x + c; (((r ^ x) >> 2) / c) | r` – the next number after `x` with the same number of bits set

Vector Hashing

```
template <>
struct hash<vector<int>> {
  size_t operator()(const vector<int> &vec) const {
    size_t seed = vec.size();
    for(auto x : vec) {
      x = ((x >> 16) ^ x) * 0x45d9f3b;
      x = ((x >> 16) ^ x) * 0x45d9f3b;
      x = (x >> 16) ^ x;
      seed ^= x + 0x9e3779b9 + (seed << 6) + (seed >> 2);
    }
    return seed;
  }
};
```

Gray Code

Two successive numbers differ in only one bit

```
int to_graycode(int num) { return num ^ (num >> 1); }
```

9.2 Cycle Detection

Given the cyclic functional sequence finds cycle **TC:** $O(n)$ **SC:** $O(1)$

```
pii floyd_cycle(int x) {
  int tort = f(x), hare = f(f(x)), len = 1, pos = 0;
  while (tort != hare) tort = f(tort), hare = f(f(hare));
  tort = x;
  while (tort != hare) tort = f(tort), hare = f(hare),pos++;
  hare = f(hare);
  while (tort != hare) hare = f(hare), len++;
  return {len, pos}
}
```

```
pii brent_cycle(int x) {
  int tort = x, hare = f(x), pow = 1, len = 1, pos = 0;
  while (tort != hare) {
    if (pow == len) pow *= 2, len = 0, tort = hare;
    hare = f(hare), len++;
  }
  tort = hare = x;
  for (int i = 0; i < len; ++i) hare = f(hare);
  while (tort != hare) tort = f(tort), hare = f(hare), pos++;
  return {len, pos};
}
```