Hanoi university of Industry

# Wise Owls

Vuong Nguyen, Anh Nguyen Duy Tuan, Hoang Nguyen

2024-10-26

# Contest (1)

## template.cpp

22 lines

```
#include <bits/stdc++.h>
using namespace std;
#define FOR(i, a, b) for (int i = (a), _b = (b); i <= _b; ++i)
#define REP(i, n) for (int i = 0, _n = (n); i < _n; ++i)
#define PR(a,n) { cerr << #a << " = "; FOR(_,1,n) cerr << a[_]
    << ' '; cerr << endl; }
#define PR0(a,n) { cerr << #a << " = "; FOR(_, 0, n - 1) cerr
    << a[_] << ' '; cerr << endl; }
#define debug(x) cerr << #x << " = " << x << endl
#define TIME (1.0 * clock() / CLOCKS_PER_SEC)
#define __builtin_popcount __builtin_popcountll
typedef unsigned long long ull;
typedef pair<int, int> pii;

signed main()
{
    #ifdef LOCAL
        freopen("input.txt", "r", stdin);
        freopen("output.txt", "w", stdout);
        freopen("log.txt", "w", stderr);
    #endif
    ios_base::sync_with_stdio(0);
  cin.tie(0);
}
```

## task.json

27 lines

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "CF",
            "type": "shell",
            "command": "g++ ",
            "args": [
                "-DLOCAL",
                "-O2",
                "-Wall",
                "'-Wl,--stack,268435456'",
                "${fileBasename}",
                "-o",
                "${fileBasenameNoExtension}",
                ";",
                "./${fileBasenameNoExtension}"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

## troubleshoot.txt

53 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could find anything overflow, segmentation fault?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
Are you forget to type fast IO, if input or output is large?
Avoid vector, map. (use arrays/unordered_map)
Avoid dynamic allocation (in trie, persistent segment tree, etc
    )
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Avoid dynamic allocation (in trie, persistent segment tree, etc
    )
Are you clearing all data structures between test cases?
```

# Mathematics (2)

## 2.1 Equations

In general, given an equation $Ax = b$, the solution to a variable $x_i$ is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where $A_i'$ is $A$ with the $i$'th column replaced by $b$.

## 2.2 Sums

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

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

## 2.3 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

## 2.4 Probability theory

Let $X$ be a discrete random variable with probability $p_X(x)$ of assuming the value $x$. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where $\sigma$ is the standard deviation. If $X$ is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent $X$ and $Y$,

$$V(aX + bY) = a^2 V(X) + b^2 V(Y).$$

### 2.4.1 Discrete distributions

#### Binomial distribution

The number of successes in $n$ independent yes/no experiments, each which yields success with probability $p$ is $\mathrm{Bin}(n,p)$, $n=1,2,\dots,$ $0\le p\le 1$.

$$p(k)=\binom{n}{k}p^k(1-p)^{n-k}$$

$$\mu=np,\ \sigma^2=np(1-p)$$

$\mathrm{Bin}(n,p)$ is approximately $\mathrm{Po}(np)$ for small $p$.

#### First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability $p$ is $\mathrm{Fs}(p)$, $0\le p\le 1$.

$$p(k)=p(1-p)^{k-1},\ k=1,2,\dots$$

$$\mu=\frac{1}{p},\ \sigma^2=\frac{1-p}{p^2}$$

#### Poisson distribution

The number of events occurring in a fixed period of time $t$ if these events occur with a known average rate $\kappa$ and independently of the time since the last event is $\mathrm{Po}(\lambda)$, $\lambda=t\kappa$.

$$p(k)=e^{-\lambda}\frac{\lambda^k}{k!},k=0,1,2,\dots$$

$$\mu=\lambda,\ \sigma^2=\lambda$$

### 2.4.2 Continuous distributions

#### Uniform distribution

If the probability density function is constant between $a$ and $b$ and 0 elsewhere it is $\mathrm{U}(a,b)$, $a<b$.

$$f(x)=\begin{cases}\frac{1}{b-a} & a<x<b\\ 0 & \text{otherwise}\end{cases}$$

$$\mu=\frac{a+b}{2},\ \sigma^2=\frac{(b-a)^2}{12}$$

#### Exponential distribution

The time between events in a Poisson process is $\mathrm{Exp}(\lambda)$, $\lambda>0$.

$$f(x)=\begin{cases}\lambda e^{-\lambda x} & x\ge 0\\ 0 & x<0\end{cases}$$

$$\mu=\frac{1}{\lambda},\ \sigma^2=\frac{1}{\lambda^2}$$

#### Normal distribution

Most real random values with mean $\mu$ and variance $\sigma^2$ are well described by $\mathcal{N}(\mu,\sigma^2)$, $\sigma>0$.

$$f(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1\sim\mathcal{N}(\mu_1,\sigma_1^2)$ and $X_2\sim\mathcal{N}(\mu_2,\sigma_2^2)$ then

$$aX_1+bX_2+c\sim\mathcal{N}(\mu_1+\mu_2+c,a^2\sigma_1^2+b^2\sigma_2^2)$$

## 2.5 Linear algebra

### Gauss.h

**Description:** function to solve $Ax=b$, with last column of a is vector b, return number of solution and a solution. 0 index base
**Time:** $\mathcal{O}\left(n^2\times m\right)$ 56 lines

```
const int INF = 1e9;
const double EPS = 1e-9;
int gauss (vector <vector<double> > a, vector<double> & 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 (abs (a[sel][col]) < EPS)
            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) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    // If we need any solution (in case INF solutions), we
        should be
    // ok at this point.
    // If need to solve partially (get which values are fixed/
        INF value):
//    for (int i=0; i<m; ++i)
//        if (where[i] != -1) {
//            REP(j,n) if (j != i && fabs(a[where[i]][j]) > EPS
    ) {
//                where[i] = -1;
//                break;
//            }
//        }
    // Then the variables which has where[i] == -1 --> INF
        values

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

### GaussBinary.h

**Description:** function to solve $Ax=b$ in modulo 2, with last column of a is vector b, return number of solution and a solution. 0 index base Usage gauss(a, n, m, ans) with n is number of equations, m is number of vars.
**Time:** $\mathcal{O}\left(n^2\times m\right)$ 48 lines

```
const int N = 502, INF = 1e9;
int gauss (vector < bitset<N> > a, int n, int m, bitset<N> &
    ans)
{
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col)
    {
        for (int i=row; i<n; ++i)
            if (a[i][col])
            {
                swap(a[i], a[row]);
                break;
            }
        if (!a[row][col])
            continue;
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row && a[i][col])
                a[i] ^= a[row];
        ++row;
    }
    for (int i = 0; i < m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i = 0; i < n; ++i)
    {
        int sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs(sum - a[i][m]) > 0)
            return 0;
    }
    // If we need any solution (in case INF solutions), we
        should be
    // ok at this point.
    // If need to solve partially (get which values are fixed/
        INF value):
//    for (int i=0; i<m; ++i)
//        if (where[i] != -1) {
//            REP(j,n) if (j != i && fabs(a[where[i]][j]) > EPS
    ) {
//                where[i] = -1;
//                break;
//            }
//        }
    // Then the variables which has where[i] == -1 --> INF
        values
    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
```

```
    return 1;
}
```

### MatrixInverseBinary.h

**Description:** return the $A^-1$ if $|A| \neq 0$ Usage inv(a, n) with n is size of matrix
**Time:** $\mathcal{O}\left(n^3\right)$ 25 lines

```
const int N = 1003;
vector <bitset<N>> inv(vector <bitset<N>> a, int n)
{
    for (int i = 0; i < n; ++i)
        a[i][n + i] = 1;
    vector<int> where (n, -1);
    for (int col=0, row=0; col< n && row<n; ++col) {
        for (int i=row; i<n; ++i)
            if (a[i][col]) {
                swap (a[i], a[row]);
                break;
            }
        if (!a[row][col])
            continue;
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row && a[i][col])
                a[i] ^= a[row];
        ++row;
    }
    for (int i = 0; i < n; ++i)
        a[i] >>= n;
    return a;
}
```

# Data structures (3)

## 3.1 STL extended data structures

### OrderStatisticTree.h

**Description:** A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null_type.
**Time:** $\mathcal{O}(\log N)$
<ext/pb_ds/assoc_container.hpp> 16 lines

```
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

### HashMap.h

**Description:** Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).
<ext/pb_ds/assoc_container.hpp> 7 lines

```
using ll = long long;
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

## 3.2 Tree data structures

### FenwickTree2D.h

**Description:** Computes prefix sum of regtangle $(1, 1)$ to $(x, y)$ and updates single elements a[x][y] by taking the difference between the old and new value. Note that index is from 1.
**Time:** Both operations are $\mathcal{O}\left(\log^2 n\right)$. 29 lines

```
using ll = long long;
struct BIT2D
{
    vector <vector <ll>> a;
    int n, m;
    BIT2D(int _n, int _m)
    {
        n = _n, m = _m;
        a.resize(n + 1, vector <ll>(m + 1, 0));
    }
    void add(int x, int y, long long val)
    {
        for (int i = x; i <= n; i += i & -i)
        {
            for (int j = y; j <= m; j += j & -j)
                a[i][j] += val;
        }
    }
    ll get(int x, int y)
    {
        ll res = 0;
        for (int i = x; i; i -= i & -i)
        {
            for (int j = y; j; j -= j & -j)
                res += a[i][j];
        }
        return res;
    }
};
```

### Trie.h

**Description:** A data structure for dealing with prefix of string
**Memory:** $\mathcal{O}(|s| \times k)$
**Time:** $\mathcal{O}(|s|)$ for each method 55 lines

```
struct Trie{
    struct Node{
        Node* child[26];
        int exist, cnt;
        Node() {
            for (int i = 0; i < 26; i++) child[i] = nullptr;
            exist = cnt = 0;
        }
    };
    int cur;
    Node* root;
    Trie() : cur(0) {
        root = new Node();
    };
    void add_string(string s) {
        Node* p = root;
        for (auto f : s) {
            int c = f - 'a';
            if (p->child[c] == nullptr) p->child[c] = new Node
                ();
            p = p->child[c];
            p->cnt++;
        }
        p->exist++;
    }
    bool delete_string_recursive(Node* p, string& s, int i) {
        if (i != (int)s.size()) {
            int c = s[i] - 'a';
            bool isChildDeleted = delete_string_recursive(p->
                child[c], s, i + 1);
            if (isChildDeleted) p->child[c] = nullptr;
        }
        else p->exist--;
        if (p != root) {
            p->cnt--;
            if (p->cnt == 0) {
                delete(p);
                return true;
            }
        }
        return false;
    }
    void delete_string(string s) {
        if (find_string(s) == false) return;
        delete_string_recursive(root, s, 0);
    }
    bool find_string(string s) {
        Node* p = root;
        for (auto f : s) {
            int c = f - 'a';
            if (p->child[c] == nullptr) return false;
            p = p->child[c];
        }
        return (p->exist != 0);
    }
};
```

## 3.3 Mo algorithms

Sort queries by $(l/\sqrt{n}, r)$ –¿ Answer queries in O($q\sqrt{n}$) $\times$ (times for extend one element of the segment).

On tree: - Standard method of linearizing does not work, because nodes can be $O(n)$ apart - Slightly modify the euler tour on tree, by adding root of subtree on the tour, when finishing visits all vertices of this subtree. This ensures that nodes right next to each other in the traversal are at most 3 nodes apart in the actual tree

### MoOnTree.h

**Description:** Example solve for https://www.spoj.com/problems/COT2/ using mo algorithm on tree
**Time:** $\mathcal{O}\left(q\sqrt{n}\right)$ 82 lines

```
const int M = 1e5 + 5, SZ = 282;
int st[M], en[M], t, n, q;
bool flag[M];
int a[M], dep[M];
int up[M][20];
int cnt[M], res[M], cur = 0;
vector <int> adj[M];
vector <int> tour;
void add(int x)
{
  x = tour[x];
  // modify base on node id x
}
```

```
struct Query
{
  int l, r, id, lca;
};
void compress(){}
int find_kth_ancestor(int x, int k){}
int find_lca(int u, int v){}
void dfs(int x)
{
  flag[x] = 1;
  tour.push_back(x);
  st[x] = t++;
  for (auto it: adj[x])
  {
    if (!flag[it])
    {
      up[it][0] = x;
      dep[it] = dep[x] + 1;
      dfs(it);
    }
  }
  tour.push_back(x);
  en[x] = t++;
}

void moving(int l1, int r1, int l2, int r2)
{
  if (l2 < l1)
  {
    for (int i = l2; i < l1; ++i)
      add(i);
  }
  if (l2 > l1)
  {
    for (int i = l1; i < l2; ++i)
      add(i);
  }
  if (r2 > r1)
  {
    for (int i = r1 + 1; i <= r2; ++i)
      add(i);
  }
  if (r2 < r1)
  {
    for (int i = r2 + 1; i <= r1; ++i)
      add(i);
  }
}
signed main()
{
  // get input and compress data

  // find euler tour and init for lca
  vector <Query> m(q);
  for (int i = 0; i < q; ++i)
  {
    int u, v;
    cin >> u >> v;
    if (st[u] > st[v])
      swap(u, v);
    int l = find_lca(u, v);
    if (l == u || l == v)
        m[i] = {st[u], st[v], i, 0};
    else
      m[i] = {en[u], st[v], i, l};
  }
  // implentment standard mo algorithm
}
```

# Number theory (4)

## 4.1 Modular arithmetic

### ModInt.h

**Description:** namespace for modulo basic operations.
".

template.h" 34 lines

```
namespace ModInt
{
    const int MOD = 1e9 + 7;
    int add(int x, int y)
    {
        return (x + y) % MOD;
    }
    int sub(int x, int y)
    {
        return (x - y + MOD) % MOD;
    }
    int mul(int x, int y)
    {
        return 1ll * x * y % MOD;
    }
    int pow_mod(int x, int y)
    {
        if (y == 0)
            return 1;
        int tg = pow_mod(x, y / 2);
        if (y % 2 == 0)
            return mul(tg, tg);
        return mul(x, mul(tg, tg));
    }
    int inv(int x)
    {
        return pow_mod(x, MOD - 2);
    }
    int div_mod(int x, int y)
    {
        return mul(x, inv(y));
    }
}
using namespace ModInt;
```

### ModLog.h

**Description:** funtion return min $n \geq 0$ such that $a^n = b$ (mod m), return -1 if it doesn't have any integers.
**Time:** $\mathcal{O}\left(\sqrt{m}\right)$

"ModInt.h" 20 lines

```
int log_mod(int a, int b)
{
    if (a % MOD == 0)
        return b == 0? 0: -1;
    int m = sqrt(MOD);
    unordered_map<int, int> st;
    for (int cur = 0, a_m = pow_mod(a, m), val = 1; cur < MOD -
        1; cur += m)
    {
        st[val] = st.find(val) != st.end()? min(cur, st[val]):
            cur;
        val = mul(val, a_m);
    }
    int res = MOD;
    for (int y = 0, invA = inv(a), curY = 1; y < m; ++y, curY =
        mul(invA, curY))
    {
        int tmp = mul(b, curY);
        if (st.find(tmp) != st.end())
            res = min(res, st[tmp] + y);
    }
    return res == MOD? -1: res;
}
```

### ModSqrt.h

**Description:** funtion return an integer number $x \geq 0$ such that $x^2 = a$ (mod p), return -1 if it doesn't have any integers.
**Time:** $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}\left(\log p\right)$ for most $p$

33 lines

```
using ll = long long;
ll modpow(ll b, ll e, ll mod) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}

ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  if (modpow(a, (p-1)/2, p) != 1)
      return -1;
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

## 4.2 Primality

### MillerRabin.h

**Description:** funtion for checking a non negative integer number is a prime or not. It work with all number less than $2^{64}$
**Usage:** is_prime(*n*)
**Time:** 12 times the complexity of $a^b \mod c$.
".

template.h" 53 lines

```
// Rabin miller {{{
inline uint64_t mod_mult64(uint64_t a, uint64_t b, uint64_t m)
    {
    return __int128_t(a) * b % m;
}
uint64_t mod_pow64(uint64_t a, uint64_t b, uint64_t m) {
    uint64_t ret = (m > 1);
    for (;;) {
        if (b & 1) ret = mod_mult64(ret, a, m);
        if (!(b >>= 1)) return ret;
        a = mod_mult64(a, a, m);
    }
}

// Works for all primes p < 2^64
bool is_prime(uint64_t n) {
    if (n <= 3) return (n >= 2);
```

```
    static const uint64_t small[] = {
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
            53, 59, 61, 67,
        71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127,
            131, 137, 139,
        149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
            199,
    };
    for (size_t i = 0; i < sizeof(small) / sizeof(uint64_t); ++i) {
        if (n % small[i] == 0) return n == small[i];
    }

    // Makes use of the known bounds for Miller-Rabin pseudoprimes.
    static const uint64_t millerrabin[] = {
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
    };
    static const uint64_t A014233[] = {  // From OEIS.
        2047LL, 1373653LL, 25326001LL, 3215031751LL,
            2152302898747LL,
        3474749660383LL, 341550071728321LL, 341550071728321LL,
        3825123056546413051LL, 3825123056546413051LL,
            3825123056546413051LL, 0,
    };
    uint64_t s = n-1, r = 0;
    while (s % 2 == 0) {
        s /= 2;
        r++;
    }
    for (size_t i = 0, j; i < sizeof(millerrabin) / sizeof(uint64_t); i++) {
        uint64_t md = mod_pow64(millerrabin[i], s, n);
        if (md != 1) {
            for (j = 1; j < r; j++) {
                if (md == n-1) break;
                md = mod_mult64(md, md, n);
            }
            if (md != n-1) return false;
        }
        if (n < A014233[i]) return true;
    }
    return true;
}
// }}}
```

### PrimePi.h

**Description:** return number of primes $\leq n$.
**Usage:** `prime_pi(n)`
**Time:** $\mathcal{O}\left(n^{0.75}\right)$
"template.h" 53 lines

```
using ll = long long;
int isqrt(ll n) {
    return sqrtl(n);
}
ll prime_pi(const ll N) {
    if (N <= 1) return 0;
    if (N == 2) return 1;
    const int v = isqrt(N);
    int s = (v + 1) / 2;
    vector<int> smalls(s);
    for (int i = 1; i < s; i++) smalls[i] = i;
    vector<int> roughs(s);
    for (int i = 0; i < s; i++) roughs[i] = 2 * i + 1;
    vector<ll> larges(s);
    for (int i = 0; i < s; i++) larges[i] = (N / (2 * i + 1) - 1) / 2;
    vector<bool> skip(v + 1);
    const auto divide = [](ll n, ll d) -> int { return (double) n / d;};
    const auto half = [](int n) -> int { return (n - 1) >> 1;};
    int pc = 0;
    for (int p = 3; p <= v; p += 2) if (!skip[p]) {
        int q = p * p;
        if ((ll)q * q > N) break;
        skip[p] = true;
        for (int i = q; i <= v; i += 2 * p) skip[i] = true;
        int ns = 0;
        for (int k = 0; k < s; k++) {
            int i = roughs[k];
            if (skip[i]) continue;
            ll d = (ll)i * p;
            larges[ns] = larges[k] - (d <= v ? larges[smalls[d >> 1] - pc] : smalls[half(divide(N, d))]) + pc;
            roughs[ns++] = i;
        }
        s = ns;
        for (int i = half(v), j = ((v / p) - 1) | 1; j >= p; j -= 2) {
            int c = smalls[j >> 1] - pc;
            for (int e = (j * p) >> 1; i >= e; i--) smalls[i] -= c;
        }
        pc++;
    }
    larges[0] += (ll)(s + 2 * (pc - 1)) * (s - 1) / 2;
    for (int k = 1; k < s; k++) larges[0] -= larges[k];
    for (int l = 1; l < s; l++) {
        ll q = roughs[l];
        ll M = N / q;
        int e = smalls[half(M / q)] - pc;
        if (e < l + 1) break;
        ll t = 0;
        for (int k = l + 1; k <= e; k++)
            t += smalls[half(divide(M, roughs[k]))];
        larges[0] += t - (ll)(e - l) * (pc + l - 1);
    }
    return larges[0] + 1;
}
```

## 4.3 Divisibility

### ExtendedEuclide.h

**Description:** function to find a solution of the equation ax + by = gcd(a, b).
**Time:** $\mathcal{O}\left(\log(max(a,b))\right)$
6 lines

```
template<typename T>
T extgcd(T a, T b, T &x, T &y) {
    T g = a; x = 1; y = 0;
    if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
    return g;
}
```

### 4.3.1 Bézout's identity

For $a \neq, b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If $(x, y)$ is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

*Euler's* $\phi$ function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with $n$.

$\phi(1) = 1$, $p$ prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, $m, n$ coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$.

If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1-1} ... (p_r - 1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.

$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$

**Euler's thm**: $a, n$ coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.

**Fermat's little thm**: $p$ prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p}\ \forall a$.

## 4.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either $m$ or $n$ even.

## 4.5 Fact about primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power $p^a$, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

## 4.6 Divisors

$\sum_{d|n} d = O(n \log \log n)$. $\sum_{d=1}^{n} \frac{n}{d} = O(n \log n)$. The maximun number of divisors $d(n)$ with $n$ is about $O(n^{\frac{1}{3}})$. Here is the exact $d(n)$ for some value of $n$.

| $n$ | $5 \times 10^4$ | $5 \times 10^5$ | $10^7$ | $10^{10}$ | $10^{19}$ |
|---|---|---|---|---|---|
| $d(n)$ | 100 | 200 | 448 | 2304 | 161 280 |

# Combinatorial (5)

## 5.1 Permutations

### 5.1.1 Factorial

| $n$ | 1 2 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|
| $n!$ | 1 2 6 | 24 | 120 | 720 | 5040 | 40320 | 362880 | 3628800 |

| $n$ | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|
| $n!$ | 4.0e7 | 4.8e8 | 6.2e9 | 8.7e10 | 1.3e12 | 2.1e13 | 3.6e14 |

### 5.1.2 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

### 5.1.3 Burnside's lemma

Given a group $G$ of symmetries and a set $X$, the number of elements of $X$ *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where $X^g$ are the elements fixed by $g$ ($g.x = x$).

## 5.2 Partitions and subsets

### 5.2.1 Partition function

Number of ways of writing $n$ as a sum of positive integers, disregarding the order of the summands.

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

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

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

### 5.2.2 Lucas' Theorem

Let $n,m$ be non-negative integers and $p$ a prime. Write $n=n_kp^k+...+n_1p+n_0$ and $m=m_kp^k+...+m_1p+m_0$. Then $\binom{n}{m}\equiv\prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$.

## 5.3 General purpose numbers

### 5.3.1 Stirling numbers of the first kind

Number of permutations on $n$ items with $k$ cycles.

$$c(n,k)=c(n-1,k-1)+(n-1)c(n-1,k),\ c(0,0)=1$$
$$\textstyle\sum_{k=0}^{n}c(n,k)x^k=x(x+1)\dots(x+n-1)$$

$c(8,k)=8,0,5040,13068,13132,6769,1960,322,28,1$
$c(n,2)=0,0,1,3,11,50,274,1764,13068,109584,\dots$

### 5.3.2 Stirling numbers of the second kind

Partitions of $n$ distinct elements into exactly $k$ groups.

$$S(n,k)=S(n-1,k-1)+kS(n-1,k)$$
$$S(n,1)=S(n,n)=1$$
$$S(n,k)=\frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

### 5.3.3 Bell numbers

Total number of partitions of $n$ distinct elements. $B(n)=1,1,2,5,15,52,203,877,4140,21147,\dots$. For $p$ prime,

$$B(p^m+n)\equiv mB(n)+B(n+1)\pmod p$$

### 5.3.4 Labeled unrooted trees

# on $n$ vertices: $n^{n-2}$
# on $k$ existing trees of size $n_i$: $n_1n_2\cdots n_kn^{k-2}$

### 5.3.5 Catalan numbers

$$C_n=\frac{1}{n+1}\binom{2n}{n}=\binom{2n}{n}-\binom{2n}{n+1}=\frac{(2n)!}{(n+1)!n!}$$

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

$C_n=1,1,2,5,14,42,132,429,1430,4862,16796,58786,\dots$

- sub-diagonal monotone paths in an $n\times n$ grid.
- Number of correct bracket sequence consisting of $n$ opening and $n$ closing brackets
- binary trees with with $n+1$ leaves (0 or 2 children).
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

# Graph (6)

## 6.1 Network flow

Dinic.h

**Description:** class for finding max flow of a network (index start from 0)
**Usage:** MaxFlow flow(n)
For each edge: flow.addEdge(u, v, c)
flow.getMaxFlow(s, t)
**Time:** $\mathcal{O}\left(n\times m^2\right)$ or $\mathcal{O}\left((n+m)\times f\right)$ with f is the max flow value.

<vector> 78 lines

```
using namespace std;
const int INF = 1000000000;

struct Edge {
    int a, b, cap, flow;
};

struct MaxFlow {
    int n, s, t;
    vector<int> d, ptr, q;
    vector< Edge > e;
    vector< vector<int> > g;

    MaxFlow(int _n) : n(_n), d(_n), ptr(_n), q(_n), g(_n) {
        e.clear();
        for (int i = 0; i < n; i++) {
            g[i].clear();
            ptr[i] = 0;
        }
    }

    void addEdge(int a, int b, int cap) {
        Edge e1 = { a, b, cap, 0 };
        Edge e2 = { b, a, 0, 0 };
        g[a].push_back( (int) e.size() );
        e.push_back(e1);
        g[b].push_back( (int) e.size() );
        e.push_back(e2);
    }
    int getMaxFlow(int _s, int _t) {
        s = _s; t = _t;
        int flow = 0;
        for (;;) {
            if (!bfs()) break;
            std::fill(ptr.begin(), ptr.end(), 0);
            while (int pushed = dfs(s, INF))
                flow += pushed;
        }
        return flow;
    }

private:
    bool bfs() {
        int qh = 0, qt = 0;
        q[qt++] = s;
        std::fill(d.begin(), d.end(), -1);
        d[s] = 0;

        while (qh < qt && d[t] == -1) {
            int v = q[qh++];
            for (int i = 0; i < (int) g[v].size(); i++) {
                int id = g[v][i], to = e[id].b;
                if (d[to] == -1 && e[id].flow < e[id].cap) {
                    q[qt++] = to;
                    d[to] = d[v] + 1;
                }
            }
        }
        return d[t] != -1;
    }

    int dfs (int v, int flow) {
        if (!flow) return 0;
        if (v == t) return flow;
        for (; ptr[v] < (int)g[v].size(); ++ptr[v]) {
            int id = g[v][ptr[v]],
                to = e[id].b;
            if (d[to] != d[v] + 1) continue;
            int pushed = dfs(to, min(flow, e[id].cap - e[id].
                 flow));
            if (pushed) {
                e[id].flow += pushed;
                e[id^1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
};
```

## 6.2 Matching

DFSMatching.h

**Description:** class for finding max matching of a bipartite graph (index start from 0)
**Usage:** Matching mat(max(n, m))
For each edge: mat.addEdge(u, v, c)
call mat.match() return max match size.
then matchL[i] is the id of right node matching with i left node
(-1 means i left node is not match)
**Time:** $\mathcal{O}\left(nm\right)$ , 0.5s with n = 50000 and m = 150000.

73 lines

```
struct Matching
{
    vector <vector <int>> adj;
    vector <int> matchL, matchR, t;
    int n, curT;
    Matching(int _n)
    {
        n = _n, curT = 0;
        adj.resize(n), matchL.resize(n, -1), matchR.resize(n,
             -1), t.resize(n, 0);
    }
    int init()
```

```
    {
        int initSz = 0;
        for (int l = 0; l < n; ++l)
        {
            for (auto r: adj[l])
            {
                if (matchR[r] == -1)
                {
                    assign(l, r);
                    initSz++;
                    break;
                }
            }
        }
        return initSz;
    }
    void add_edge(int u, int v)
    {
        adj[u].push_back(v);
    }
    void assign(int u, int v)
    {
        matchL[u] = v, matchR[v] = u;
    }
    bool dfs(int l)
    {
        t[l] = curT;
        for (auto r: adj[l])
        {
            if (matchR[r] == -1)
            {
                assign(l, r);
                return true;
            }
        }
        for (auto r: adj[l])
        {
            if (t[matchR[r]] < curT && dfs(matchR[r]))
            {
                assign(l, r);
                return true;
            }
        }
        return false;
    }
    int match()
    {
        int res = init();
        while (1)
        {
            curT++;
            bool havNewMatch = 0;
            for (int i = 0; i < n; ++i)
            {
                if (matchL[i] == -1 && dfs(i))
                    havNewMatch = 1, res++;
            }
            if (!havNewMatch)
                return res;
        }
    }
};
```

### WeightedMatching.h

**Description:** class for finding a prefect match with min cost flow Return mincost, match from left Index from 0, n is number of left nodes and m is the number of right nodes.

**Usage:** `init()`
For each edge: `add_edge(u, v, cost)`
`Hungarian (n, m, c)`
**Time:** $\mathcal{O}\left(n^3\right)$ 1s for n = 1000.

66 lines

```
const int N = 202, INF = 1e6;
int n, m;
long long c[N][N];
void init()
{
    for (int i = 0; i < n; ++i) for (int j = 0; j < m; ++j)
        c[i][j] = INF;
}

void add_edge(int u, int v, long long cost)
{
    c[u][v] = min(c[u][v], cost);
}
template<typename T>
pair<T, vector<int>> Hungarian (int n, int m, T c[][N]) {
    vector<T> v(m), dist(m);
    vector<int> L(n, -1), R(m, -1);
    vector<int> index(m), prev(m);
    auto getc = [&] (int i, int j) {return c[i][j] - v[j];};

    iota(index.begin(), index.end(), 0);
    for (int f = 0; f < n; ++f) {
        for (int j = 0; j < m; ++j) {
            dist[j] = getc(f, j), prev[j] = f;
        }
        T w = 0; int j, l = 0, s = 0, t = 0;
        while (true) {
            if (s == t) {
                l = s, w = dist[index[t++]];
                for (int k = t; k < m; ++k) {
                    j = index[k]; T h = dist[j];
                    if (h <= w) {
                        if (h < w) t = s, w = h;
                        index[k] = index[t], index[t++] = j;
                    }
                }
                for (int k = s; k < t; ++k) {
                    j = index[k];
                    if (R[j] < 0) goto augment;
                }
            }
            int q = index[s++], i = R[q];
            for (int k = t; k < m; ++k) {
                j = index[k];
                T h = getc(i, j) - getc(i, q) + w;
                if (h < dist[j]) {
                    dist[j] = h, prev[j] = i;
                    if (h == w) {
                        if (R[j] < 0) goto augment;
                        index[k] = index[t], index[t++] = j;
                    }
                }
            }
        }
    augment:
        for (int k = 0; k < l; ++k) v[index[k]] += dist[index[k
            ]] - w;
        int i;
        do {
            i = R[j] = prev[j];
            swap(j, L[i]);
        } while (i != f);
    }
    T ret = 0;
    for (int i = 0; i < n; ++i) ret += c[i][L[i]];
    return {ret, L};
}
```

## 6.3 DFS algorithms

### SCC.h

**Description:** Index from 0, find scc of directed graph, and compress scc to a dag, reverse(tree.scc) is topo sorted
**Usage:** `DirectedDfs tree(g)`
**Time:** $\mathcal{O}(n+m)$

50 lines

```
struct DirectedDfs {
    vector<vector<int>> g;
    int n;
    vector<int> num, low, current, S;
    int counter;
    vector<int> comp_ids;
    vector< vector<int> > scc;

    DirectedDfs(const vector<vector<int>>& _g) : g(_g), n(g.
        size()),
            num(n, -1), low(n, 0), current(n, 0), counter(0),
                comp_ids(n, -1) {
        for (int i = 0; i < n; i++) {
            if (num[i] == -1) dfs(i);
        }
    }

    void dfs(int u) {
        low[u] = num[u] = counter++;
        S.push_back(u);
        current[u] = 1;
        for (auto v : g[u]) {
            if (num[v] == -1) dfs(v);
            if (current[v]) low[u] = min(low[u], low[v]);
        }
        if (low[u] == num[u]) {
            scc.push_back(vector<int>());
            while (1) {
                int v = S.back(); S.pop_back(); current[v] = 0;
                scc.back().push_back(v);
                comp_ids[v] = ((int) scc.size()) - 1;
                if (u == v) break;
            }
        }
    }

    // build DAG of strongly connected components
    // Returns: adjacency list of DAG
    std::vector<std::vector<int>> build_scc_dag() {
        std::vector<std::vector<int>> dag(scc.size());
        for (int u = 0; u < n; u++) {
            int x = comp_ids[u];
            for (int v : g[u]) {
                int y = comp_ids[v];
                if (x != y) {
                    dag[x].push_back(y);
                }
            }
        }
        return dag;
    }
};
```

2sat.h

**Description:** class to find a solution for 2-SAT problem For lexicographical min result can solve with - For each variable: check if it can be set to False (by adding constraint i -> !i) - If solver.solve() -> keep constraint i -> !i - Otherwise, remove constraint i -> !i, and add !i -> i to force it to True

**Time:** $\mathcal{O}(n+m)$

"SCC.h" 36 lines

```
struct TwoSatSolver {
  // number of variables
    int n_vars;
    // vertex 0 -> n_vars - 1: Ai is true
    // vertex n_vars -> 2*n_vars - 1: Ai is false
    vector<vector<int>> g;
    TwoSatSolver(int _n_vars) : n_vars(_n_vars), g(2*n_vars) {}

    void x_or_y_constraint(bool is_x_true, int x, bool
         is_y_true, int y) {
        assert(x >= 0 && x < n_vars);
        assert(y >= 0 && y < n_vars);
        if (!is_x_true) x += n_vars;
        if (!is_y_true) y += n_vars;
        // x || y
        // !x -> y
        // !y -> x
        g[(x + n_vars) % (2*n_vars)].push_back(y);
        g[(y + n_vars) % (2*n_vars)].push_back(x);
    }
    // Returns:
    // If no solution -> returns {false, {}}
    // If has solution -> returns {true, solution}
    //    where |solution| = n_vars, solution = true / false
    pair<bool, vector<bool>> solve() {
        DirectedDfs tree(g);
        vector<bool> solution(n_vars);
        for (int i = 0; i < n_vars; i++) {
            if (tree.comp_ids[i] == tree.comp_ids[i + n_vars])
                 {
                return {false, {}};
            }
            // Note that reverse(tree.scc) is topo sorted
            solution[i] = tree.comp_ids[i] < tree.comp_ids[i +
                 n_vars];
        }
        return {true, solution};
    }
};
```

## 6.4 Trees

HLD.h

**Description:** Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most $\log n$ light edges.

**Usage:** implenting segment tree.
after push edge in adj[], call dfs(root) and hld(root).
For query path u -> v call query(u, v) (jump to chain containing lca(u, v))
For update node u update position pos[u] in segment tree

**Time:** $\mathcal{O}\left(\log^2 n\right)$ for each path queries.

78 lines

```
const int M = 1e5 + 5;
int head[M], pos[M], par[M], sz[M], dep[M];
vector <int> adj[M];
int curId, n;
// implenting segment tree
int get(int l, int r)
{
    return 0;
}
void update(int l, int r, int val)
{

}
// end implentment segment tree

void dfs(int x, int pre = -1)
{
    par[x] = pre;
    sz[x] = 1;
    for (auto it: adj[x])
    {
        if (it != pre)
        {
            dep[it] = dep[x] + 1;
            dfs(it, x);
            sz[x] += sz[it];
        }
    }
}

void hld(int x, int chainHead = 1, int pre = -1)
{
    head[x] = chainHead, pos[x] = curId++;
    int nxt = -1;
    for (auto it: adj[x])
    {
        if (it == pre)
            continue;
        if (nxt == -1 || sz[nxt] < sz[it])
            nxt = it;
    }
    if (nxt == -1)
        return;
    hld(nxt, chainHead, x);
    for (auto it: adj[x])
    {
        if (it != pre && it != nxt)
            hld(it, it, x);
    }
}

int query(int u, int v)
{
    int res = 0;
    while (head[u] != head[v])
    {
        if (sz[head[u]] > sz[head[v]])
            swap(u, v);
        res += get(pos[head[u]], pos[u]);
        u = par[head[u]];
    }
    if (dep[u] > dep[v])
        swap(u, v);
    res += get(pos[u], pos[v]);
    return res;
}
// useful funtion for query path
int lca(int u, int v)
{
    while (head[u] != head[v])
    {
        if (dep[head[u]] < dep[head[v]])
            swap(u, v);
        u = par[head[u]];
    }
    return (dep[u] < dep[v]? u: v);
}
```

## 6.5 Math

### 6.5.1 Number of Spanning Trees

Create an $N \times N$ matrix mat, and for each edge $a \to b \in G$, do mat[a][b]--, mat[b][b]++ (and mat[b][a]--, mat[a][a]++ if $G$ is undirected). Remove the $i$th row and column and take the determinant; this yields the number of directed spanning trees rooted at $i$ (if $G$ is undirected, remove any row/column).

### 6.5.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

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

# Strings (7)

KMP.h

**Description:** pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.

**Time:** $\mathcal{O}(n)$

".

template.h" 15 lines

```
vector<int> prefix_function(string s)
{
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++)
    {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j])
            j = pi[j - 1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

SuffixArray.h

**Description:** Builds suffix array for a string. sa[i] is the starting index of the suffix which is $i$'th in the sorted suffix array. ("BANANA@" -> 6, 5, 3, 1, 0, 4, 2) The returned vector is of size $n$, and sa[0] = n. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0. ("BANANA@" -> 0 0 1 3 0 0 2) The input string must not contain any zero bytes. Push back a dummny charater before call the funtions.

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

".

template.h" 49 lines

```
vector <int> build_suffix_array(string &s, int lim = 256)
{
    int n = s.size();
    vector <int> sa(n);
    vector <int> prevSA(n), prevLabel(n), label(n), cnt(max(lim
         + 1, n), 0);
    REP (i, n) cnt[s[i]]++;
    FOR (i, 1, lim - 1) cnt[i] += cnt[i - 1];
    for (int i = n - 1; i >= 0; --i)
```

```
        sa[--cnt[s[i]]] = i;
    label[sa[0]] = 0;
    FOR (i, 1, n - 1)
        label[sa[i]] = (s[sa[i]] == s[sa[i - 1]]? label[sa[i -
             1]]: label[sa[i - 1]] + 1);
    for (int len = 1;; len *= 2)
    {
        REP (i, n) prevSA[i] = (sa[i] - len + n) % n;
        REP (i, n) prevLabel[i] = label[prevSA[i]];
        cnt.assign(n, 0);
        REP (i, n) cnt[prevLabel[i]]++;
        FOR (i, 1, n - 1) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; --i)
            sa[--cnt[prevLabel[i]]] = prevSA[i];
        swap(prevLabel, label);
        label[sa[0]] = 0;
        FOR (i, 1, n - 1)
        {
            pii cur = {prevLabel[sa[i]], prevLabel[(sa[i] + len
                 ) % n]};
            pii p = {prevLabel[sa[i - 1]], prevLabel[(sa[i - 1]
                  + len) % n]};
            label[sa[i]] = (cur == p? label[sa[i - 1]]: label[
                 sa[i - 1]] + 1);
        }
        if (label[sa[n - 1]] == n - 1)
            break;
    }
    return sa;
}

vector <int> build_lcp(string &s, vector <int> &sa)
{
    int n = s.size(), rank[n];
    vector <int> lcp(n);
    REP (i, n) rank[sa[i]] = i;
    for (int i = 1, q = 0; i < n; ++i)
    {
        int pre = sa[rank[i] - 1];
        while (s[pre + q] == s[i + q]) ++q;
        lcp[rank[i]] = q;
        if (q > 0) --q;
    }
    return lcp;
}
```

# Geometry (8)

## 8.1 Geometric primitives

### Point.h

**Description:** Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)
".

template.h" 47 lines

```
const double EPS = 1e-6;
const double PI = acos(-1.0);
template <class T1, class T2> int cmp(T1 x, T2 y)
{
    if constexpr (is_floating_point<T1>() || is_floating_point<
         T2>())
        return x < y - EPS? -1: (x > y + EPS? 1: 0);
    else
        return x < y? -1: (x == y? 0: 1);
}
template<typename T>
struct P {
    T x, y;
    P() { x = y = T(0); }
    P(T _x, T _y) : x(_x), y(_y) {}

    P operator + (const P& a) const { return P(x+a.x, y+a.y); }
    P operator - (const P& a) const { return P(x-a.x, y-a.y); }
    P operator * (T k) const { return P(x*k, y*k); }
    P<double> operator / (double k) const { return P(x/k, y/k);
         }

    T operator * (const P& a) const { return x*a.x + y*a.y; }
         // dot product
    T operator % (const P& a) const { return x*a.y - y*a.x; }
         // cross product

    int cmp(const P<T>& q) const { if (int t = ::cmp(x,q.x))
         return t; return ::cmp(y,q.y); }

    #define Comp(x) bool operator x (const P& q) const { return
          cmp(q) x 0; }
    Comp(>) Comp(<) Comp(==) Comp(>=) Comp(<=) Comp(!=)
    #undef Comp

    T norm() { return x*x + y*y; }

    // Note: There are 2 ways for implementing len():
    // 1. sqrt(norm()) --> fast, but inaccurate (produce some
         values that are of order X^2)
    // 2. hypot(x, y) --> slow, but much more accurate
    double len() { return hypot(x, y); }

    P<double> rotate(double alpha) {
        double cosa = cos(alpha), sina = sin(alpha);
        return P(x * cosa - y * sina, x * sina + y * cosa);
    }
};
using Point = P<double>;

template<typename T>
int ccw(P<T> a, P<T> b, P<T> c) {
    return cmp((b-a)%(c-a), T(0));
}
```

### Angle.h

**Description:** Return angle and direct angle for AOB. Only work with P<double> and P<long double>

"Point.h" 11 lines

```
double angle(Point a, Point o, Point b) { // min of directed
      angle AOB & BOA
    a = a - o; b = b - o;
    return acos((a * b) / sqrt(a.norm()) / sqrt(b.norm()));
}

double directed_angle(Point a, Point o, Point b) { // angle AOB
     , in range [0, 2*PI)
    double t = -atan2(a.y - o.y, a.x - o.x)
            + atan2(b.y - o.y, b.x - o.x);
    while (t < 0) t += 2*PI;
    return t;
}
```

### Line.h

**Description:** Line with some constructor and funtion about postion of 2 lines. NOTE: WILL NOT WORK WHEN a = b = 0. Point A, B is NOT ENSURED that these are valid

"Point.h" 49 lines

```
struct Line {
    double a, b, c;  // ax + by + c = 0
    Point A, B;  // Added for polygon intersect line.
    Line(double _a, double _b, double _c) : a(_a), b(_b), c(_c)
         {}
    Line(Point _A, Point _B) : A(_A), B(_B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = - (a * A.x + b * A.y);
    }
    Line(Point P, double m) {
        a = -m; b = 1;
        c = -((a * P.x) + (b * P.y));
    }
    double f(Point p) {
        return a*p.x + b*p.y + c;
    }
};

bool areParallel(Line l1, Line l2) {
    return cmp(l1.a*l2.b, l1.b*l2.a) == 0;
}

bool areSame(Line l1, Line l2) {
    return areParallel(l1 ,l2) && cmp(l1.c*l2.a, l2.c*l1.a) ==
         0
                && cmp(l1.c*l2.b, l1.b*l2.c) == 0;
}

bool areIntersect(Line l1, Line l2, Point &p) {
    if (areParallel(l1, l2)) return false;
    double dx = l1.b*l2.c - l2.b*l1.c;
    double dy = l1.c*l2.a - l2.c*l1.a;
    double d  = l1.a*l2.b - l2.a*l1.b;
    p = Point(dx/d, dy/d);
    return true;
}

// closest point from p in line l.
void closestPoint(Line l, Point p, Point &ans) {
    if (fabs(l.b) < EPS) {
        ans.x = -(l.c) / l.a; ans.y = p.y;
        return;
    }
    if (fabs(l.a) < EPS) {
        ans.x = p.x; ans.y = -(l.c) / l.b;
        return;
    }
    Line perp(l.b, -l.a, - (l.b*p.x - l.a*p.y));
    areIntersect(l, perp, ans);
}
```

### OnSegment.h

**Description:** check a point p is on segment [a, b] Both endpoints (p == a or p == b) is also return true.

"Point.h" 6 lines

```
template<typename T>
bool onSegment(const P<T>& a, const P<T>& b, const P<T>& p) {
    return ccw(a, b, p) == 0
        && min(a.x, b.x) <= p.x && p.x <= max(a.x, b.x)
        && min(a.y, b.y) <= p.y && p.y <= max(a.y, b.y);
}
```

### SegmentDistance.h

**Description:** Some funtion to find closest point and distance from point P to line and segment AB Return distance and reference to closest Point –> c

"Point.h" 19 lines

```
double distToLine(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    c = a + (ab * u);
```

```
    return (p-c).len();
}
double distToLineSegment(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    if (u < 0.0) {
        c = Point(a.x, a.y);
        return (p - a).len();
    }
    if (u > 1.0) {
        c = Point(b.x, b.y);
        return (p - b).len();
    }
    return distToLine(p, a, b, c);
}
```

### SegmentIntersection.h

**Description:** Check 2 segment is intersect or not (including end points)

"OnSegment.h" 12 lines

```
template<typename T>
bool segmentIntersect(const P<T>& a, const P<T>& b, const P<T>&
     c, const P<T>& d) {
    if (onSegment(a, b, c)
            || onSegment(a, b, d)
            || onSegment(c, d, a)
            || onSegment(c, d, b)) {
        return true;
    }

    return ccw(a, b, c) * ccw(a, b, d) < 0
        && ccw(c, d, a) * ccw(c, d, b) < 0;
}
```

## 8.2 Polygons

### PointInPolygon.h

**Description:** check a point is in, or out, or on the boundary of a polygon. It works with any polygon and P<double>.
**Time:** $\mathcal{O}(n)$

"Point.h" 28 lines

```
typedef vector<Point> Polygon;
enum PolygonLocation { OUT, ON, IN };
PolygonLocation in_polygon(const Polygon &p, Point q) {
    if ((int)p.size() == 0) return PolygonLocation::OUT;

    // Check if point is on edge.
    int n = p.size();
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        Point u = p[i], v = p[j];

        if (u > v) swap(u, v);

        if (ccw(u, v, q) == 0 && u <= q && q <= v) return
             PolygonLocation::ON;
    }

    // Check if point is strictly inside.
    int c = 0;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        if (((p[i].y <= q.y && q.y < p[j].y)
                || (p[j].y <= q.y && q.y < p[i].y))
            && q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[
                 i].y) / (double) (p[j].y - p[i].y)) {
            c = !c;
        }
    }
    return c ? PolygonLocation::IN : PolygonLocation::OUT;
}
```

### ConvexHull.h

**Description:** Finds the the convex hull of $n$ point, destroy the initial points not belonging to the convex hull. Max point to keep colinear triple, and min point is not. NOTE: Max. point DOES NOT WORK when some points are the SAME.
**Usage:** `If minimum point --> define REMOVE_REDUNDANT`
**Time:** $\mathcal{O}(n \log n)$

"OnSegment.h" 40 lines

```
typedef vector< Point > Polygon;
 #define REMOVE_REDUNDANT
template<typename T>
T area2(P<T> a, P<T> b, P<T> c) { return a%b + b%c + c%a; }

template<typename T>
void ConvexHull(vector<P<T>> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<P<T>> up, dn;
    for (int i = 0; i < (int) pts.size(); i++) {
#ifdef REMOVE_REDUNDANT
        while (up.size() > 1 && area2(up[up.size()-2], up.back
             (), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back
             (), pts[i]) <= 0) dn.pop_back();
#else
        while (up.size() > 1 && area2(up[up.size()-2], up.back
             (), pts[i]) > 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back
             (), pts[i]) < 0) dn.pop_back();
#endif
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.
         push_back(up[i]);

#ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < (int) pts.size(); i++) {
        if (onSegment(dn[dn.size()-2], pts[i], dn.back())) dn.
             pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && onSegment(dn.back(), dn[1], dn[0])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
#endif
}
```

### PointInConvex.h

**Description:** Function for check point in convex polygon
**Time:** $\mathcal{O}(\log n)$

"OnSegment.h", "PointInPolygon.h" 16 lines

```
#define Det(a,b,c) ((double)(b.x-a.x)*(double)(c.y-a.y)-(double
     )(b.y-a.y)*(c.x-a.x))
PolygonLocation in_convex(vector<Point>& l, Point p){
    int a = 1, b = l.size()-1, c;
    if (Det(l[0], l[a], l[b]) > 0) swap(a,b);

    if (onSegment(l[0], l[a], p)) return ON;
    if (onSegment(l[0], l[b], p)) return ON;

    if (Det(l[0], l[a], p) > 0 || Det(l[0], l[b], p) < 0)
        return OUT;
    while(abs(a-b) > 1) {
        c = (a+b)/2;
        if (Det(l[0], l[c], p) > 0) b = c; else a = c;
    }
    int t = cmp(Det(l[a], l[b], p), 0);
    return (t == 0) ? ON : (t < 0) ? IN : OUT;
}
```

### Area.h

**Description:** Find the area of any polygon
**Time:** $\mathcal{O}(n)$

"PointInPolygon.h" 11 lines

```
template<typename T>
T signed_area2(vector<P<T>> p) {
    T area(0);
    for(int i = 0; i < (int) p.size(); i++) {
        area += p[i] % p[(i + 1) % p.size()];
    }
    return area;
}
double area(const Polygon &p) {
    return std::abs(signed_area2(p) / 2.0);
}
```

## 8.3 Circles

### Circle.h

**Description:** Basic method for Circle, theta is assumed in [0, 2 * PI] using radian.

"Point.h" 13 lines

```
struct Circle : Point {
    double r;
    Circle(double _x = 0, double _y = 0, double _r = 0) : Point
         (_x, _y), r(_r) {}
    Circle(Point p, double _r) : Point(p), r(_r) {}
    bool contains(Point p) { return (*this - p).len() <= r +
         EPS; }
    double area() const { return r*r*M_PI; }
    double sector_area(double theta) const {
        return 0.5 * r * r * theta;
    }
    double segment_area(double theta) const {
        return 0.5 * r * r * (theta - sin(theta));
    }
};
```

### SmallestEnclosingCircle.h

**Description:** Given N points. Find the smallest circle enclosing these points.
**Time:** Except $\mathcal{O}(N)$.

"Circle.h" 38 lines

```
struct SmallestEnclosingCircle {
    Circle getCircle(vector<Point> points) {
        assert(!points.empty());
        mt19937 rng(time(0));
        shuffle (points.begin(), points.end(), rng);
        Circle c(points[0], 0);
        int n = points.size();

        for (int i = 1; i < n; i++)
            if ((points[i] - c).len() > c.r + EPS)
            {
                c = Circle(points[i], 0);
                for (int j = 0; j < i; j++)
```

```
                if ((points[j] - c).len() > c.r + EPS)
                {
                    c = Circle((points[i] + points[j]) / 2,
                          (points[i] - points[j]).len() /
                         2);
                    for (int k = 0; k < j; k++)
                        if ((points[k] - c).len() > c.r +
                             EPS)
                            c = getCircumcircle(points[i],
                                 points[j], points[k]);
                }
            }

        return c;
    }
    // NOTE: This code work only when a, b, c are not collinear
         and no 2 points are same —> DO NOT
    // copy and use in other cases.
    Circle getCircumcircle(Point a, Point b, Point c) {
        assert(a != b && b != c && a != c);
        assert(ccw(a, b, c));

        double d = 2.0 * (a.x * (b.y - c.y) + b.x * (c.y - a.y)
             + c.x * (a.y - b.y));
        assert(fabs(d) > EPS);
        double x = (a.norm() * (b.y - c.y) + b.norm() * (c.y -
            a.y) + c.norm() * (a.y - b.y)) / d;
        double y = (a.norm() * (c.x - b.x) + b.norm() * (a.x -
            c.x) + c.norm() * (b.x - a.x)) / d;
        Point p(x, y);
        return Circle(p, (p - a).len());
    }
};
```

## 8.4 Misc. Point Set Problems

### ClosestPair.h

**Description:** Finds the closet pair among $n$ points. Returns a pair with distance and 2 points. If need point ids -> add ID to struct P. If need exact square dist -> can compute from returned points

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

"Point.h" 49 lines

```
template<typename T>
std::pair<double, std::pair<P<T>, P<T>>> closest_pair(vector<P<
     T>> a) {
    int n = a.size();
    assert(n >= 2);
    double mindist = 1e20;
    std::pair<P<T>, P<T>> best_pair;
    std::vector<P<T>> t(n);
    sort(a.begin(), a.end());

    auto upd_ans = [&] (const P<T>& u, const P<T>& v) {
        double cur = (u - v).len();
        if (cur < mindist) {
            mindist = cur;
            best_pair = {u, v};
        }
    };

    std::function<void(int,int)> rec = [&] (int l, int r) {
        if (r - l <= 3) {
            for (int i = l; i < r; ++i) {
                for (int j = i + 1; j < r; ++j) {
                    upd_ans(a[i], a[j]);
                }
            }
            sort(a.begin() + l, a.begin() + r, cmpy<T>);
            return;
        }

        int m = (l + r) >> 1;
        T midx = a[m].x;
        rec(l, m);
        rec(m, r);

        std::merge(a.begin() + l, a.begin() + m, a.begin() + m,
             a.begin() + r, t.begin(), cmpy<T>);
        std::copy(t.begin(), t.begin() + r - l, a.begin() + l);

        int tsz = 0;
        for (int i = l; i < r; ++i) {
            if (abs(a[i].x - midx) < mindist) {
                for (int j = tsz - 1; j >= 0 && a[i].y - t[j].y
                     < mindist; --j)
                    upd_ans(a[i], t[j]);
                t[tsz++] = a[i];
            }
        }
    };
    rec(0, n);

    return {mindist, best_pair};
}
```

# Miscellaneous (9)

## 9.1 Polynomial

### FFT.h

**Description:** fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all $k$. N must be a power of 2. Useful for convolution: conv(a, b) = c, where $c[x] = \sum a[i]b[x-i]$. 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). Otherwise, use NTT/FFTMod.

**Time:** $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

88 lines

```
using ld = long double;
// Can use std::complex<ld> instead to make code shorter (but
     it will be slightly slower)
struct Complex {
    ld x[2];
    Complex() { x[0] = x[1] = 0.0; }
    Complex(ld a) { x[0] = a; }
    Complex(ld a, ld b) { x[0] = a; x[1] = b; }
    Complex(const std::complex<ld>& c) {
        x[0] = c.real();
        x[1] = c.imag();
    }
    Complex conj() const {
        return Complex(x[0], -x[1]);
    }
    Complex operator + (const Complex& c) const {
        return Complex {
            x[0] + c.x[0],
            x[1] + c.x[1],
        };
    }
    Complex operator - (const Complex& c) const {
        return Complex {
            x[0] - c.x[0],
            x[1] - c.x[1],
        };
    }
    Complex operator * (const Complex& c) const {
        return Complex(
            x[0] * c.x[0] - x[1] * c.x[1],
            x[0] * c.x[1] + x[1] * c.x[0]
        );
    }

    Complex& operator += (const Complex& c) { return *this = *
         this + c; }
    Complex& operator -= (const Complex& c) { return *this = *
         this - c; }
    Complex& operator *= (const Complex& c) { return *this = *
         this * c; }
};
void fft(vector<Complex>& a) {
    int n = a.size();
    int L = 31 - __builtin_clz(n);
    static vector<Complex> R(2, 1);
    static vector<Complex> rt(2, 1);
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = Complex(polar(1.0L, acos(-1.0L) / 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 = (ld*) &rt[j+k].x, y = (ld*) &a[i+j+k].
                     x;
                Complex 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<ld> multiply(const vector<ld>& a, const vector<ld>& b) {
    if (a.empty() || b.empty()) return {};
    vector<ld> res(a.size() + b.size() - 1);
    int L = 32 - __builtin_clz(res.size()), n = 1<<L;
    vector<Complex> in(n), out(n);

    for (size_t i = 0; i < a.size(); ++i) in[i].x[0] = a[i];
    for (size_t i = 0; i < b.size(); ++i) in[i].x[1] = b[i];

    fft(in);
    for (Complex& x : in) x *= x;

    for (int i = 0; i < n; ++i) out[i] = in[-i & (n-1)] - in[i
         ].conj();
    fft(out);

    for (size_t i = 0; i < res.size(); ++i) res[i] = out[i].x
         [1] / (4*n);
    return res;
}
long long my_round(ld x) {
    if (x < 0) return -my_round(-x);
    return (long long) (x + 1e-2);
}
// }}}
```

## Karatsuba.h

**Description:** short code to compute convolution of 2 sequence work with any mod. n is must power of 2 `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$.

**Time:** $\mathcal{O}\left(n^{\log_2 3}\right)$

20 lines

```
const int MOD = 1e9 + 7;
template<int n>
void Mul(int *a, int *b, int *ab) {
    if (n == 1) return ab[0] = 1ll * a[0] * b[0] % MOD, void();

    const int m = n >> 1;
    int _a[m], _b[m], _ab[n]{};
    for (int i = 0; i < m; ++i)
        _a[i] = (a[i] + a[i + m]) % MOD,
        _b[i] = (b[i] + b[i + m]) % MOD;

    Mul<m>(_a, _b, _ab);
    Mul<m>(a, b, ab);
    Mul<m>(a + m, b + m, ab + n);

    for (int i = 0; i < n; ++i)
        _ab[i] = (MOD + MOD + _ab[i] - ab[i] - ab[i + n]) % MOD
            ;
    for (int i = 0; i < n; ++i)
        ab[i + m] = (ab[i + m] + _ab[i]) % MOD;
}
```

## SOSDp.h

**Description:** 2 functions involve in sum over subset dp

**Time:** $\mathcal{O}\left(m \times 2^m\right)$ m is bit size.

25 lines

```
vector <int> sum_over_super_set(vector <int> a, int m)
{
    // a = {1, 4, 2, 3} then dp = {10, 7, 5, 3}.
    auto dp = a;
    for(int i = 0; i < m; i++) {
        for(int mask = (1 << m) - 1; mask >= 0; mask--) {
            if(mask >> i & 1)
                dp[mask ^ (1 << i)] += dp[mask];
        }
    }
    return dp;
}

vector <int> sum_over_subset(vector <int> a, int m)
{
    // a = {1, 4, 2, 3} then dp = {1, 5, 3, 10}.
    auto dp = a;
    for(int i = 0; i < m; i++) {
        for(int mask = 0; mask < (1 << m); ++mask) {
            if(mask >> i & 1)
                dp[mask] += dp[mask ^ (1 << i)];
        }
    }
    return dp;
}
```

# Techniques (A)

techniques.txt 159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```