Fuck Logic ICPC Team Notebook

Contents

1 Intro

1.1 Strategies

```
Approaching any Problem:
- Abstraction and Modularity (also called top down approach).
- Questions and Perspective (keep looking and attacking it from diff contexts).
- What do you know? (every information given counts and so does partial progress).

Approaching DP Problems:
- Think with modularity (top-down or bottom up as it helps).
- Remember you need to encode all the information you have into the DP. This will help you deicde what the
      states of the DP should be. The answer always needs to be the global optimum, so you need to figure
      that out too.
- Make sure you update the DP, only when required You update the DP only when you have a decision to make.
- While dealing with bitwise operations, try to imagine of them in terms of Venn Diagrams.
```

1.2 Template

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

#define int long long
#define st first
#define nd second
#define mp make_pair
#define pq priority_queue
#define cl(x, v) memset((x), (v), sizeof(x))
#define gcd(x, y) __gcd((x), (y))
#define pb push_back

// Comment the following when flushing.
#define endl "\n"

#define all(c) (c).begin(), (c).end()
#define range(c, r) (c).begin(), (c).begin() + (r)
#define present(c, x) ((c).find(x) != (c).end())
#define cpresent(c, x) (find(all(c), x) != (c).end())
#define run(x, c) for ((x) = (c).begin(); (x) != (c).end(); (x)++)
#define rep(i, n) for (i = 0; i < n; i++)
#define REP(i, k, n) for (i = k; i <= n; i++)
#define REPR(i, k, n) for (i = k; i >= n; i--)

// Ordered set adds two new functions to set -
// (set).find_by_order([kth element based on zero indexing])
// and, order_of_key()
// order_of_key returns number of elements less that parameter.
// If element exists, that order is its index.
#define ordered_set                                                          \
  tree<ll, null_type, less<ll>, rb_tree_tag, tree_order_statistics_node_update>

#ifndef ONLINE_JUDGE
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<
#else
#define db(x) ((void)0)
#define dbs(x) ((void)0)
#endif

typedef long long ll;
typedef long double ld;

typedef pair<int, int> pii;
typedef pair<int, pii> piii;
typedef pair<ll, ll> ii;
typedef vector<ll> vi;
typedef vector<vi> vvi;
typedef vector<pii> vii;
typedef vector<vii> vvii;
```

```cpp
const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9 + 7;

const int N = 1e5 + 5;

/*
  Uncomment the following before submission.
*/
//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>
// using namespace __gnu_pbds;

signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  // freopen("in", "r", stdin);
  // freopen("out", "w", stdout);

  int t;
  cin >> t;
  while (t--) {
    // Fuck Logic.
  }
  return 0;
}
```

2 Data Structures

2.1 Linked List

```cpp
template<typename T>
struct node
{
    T data;
    struct node* next;
    struct node* prev;
};

template<typename T>
class LinkedList
{
    public:
    node<T> *head, *tail;

    // Constructor
    LinkedList()
    {
        head = NULL;
        tail = NULL;
    }

    // Print
    void display(node<T> *head)
    {
        if (head == NULL)
        {
            cout << endl;
        }
        else
        {
            cout << head->data << " ";
            display(head->next);
        }
    }

    // Push Back
    void push_back(T n)
    {
        node<T> *temp = new node<T>;
        temp->data = n;
        temp->next = NULL;
        temp->prev = NULL;

        if (head == NULL)
        {
            head = temp;
            tail = temp;
        }
        else
        {
            tail->next = temp;
            temp->prev = tail;
            tail = tail->next;
        }
    }

    // Push Front
    void push_front(T n)
    {
        node<T> *temp = new node<T>;
        temp->data = n;
        temp->next = NULL;
        temp->prev = NULL;

        if (head == NULL)
        {
            head = temp;
            tail = temp;
        }
        else
        {
            head->prev = temp;
            temp->next = head;
            head = head->prev;
        }
    }

    // Pop Front
    void pop_front()
    {
        if (head == NULL)
        {
            cout << "Empty list\n";
        }
        else
        {
            head = head->next;
            node<T> *temp = head->prev;
            delete temp;
        }
    }

    // Pop Back
    void pop_back()
    {
        if (head == NULL)
        {
            cout << "Empty list\n";
        }
        else
        {
            tail = tail->prev;
            node<T> *temp = tail->next;
            delete temp;
        }
    }
};
```

2.2 Minimum Queue

```cpp
// O(1) complexity for all operations, except for clear,
// which could be done by creating another deque and using swap

struct MinQueue {
  int plus = 0;
  int sz = 0;
  deque<pair<int, int>> dq;

  bool empty() { return dq.empty(); }
  void clear() { plus = 0; sz = 0; dq.clear(); }
  void add(int x) { plus += x; } // Adds x to every element in the queue
  int min() { return dq.front().first + plus; } // Returns the minimum element in the queue
  int size() { return sz; }

  void push(int x) {
    x -= plus;
    int amt = 1;
    while (dq.size() and dq.back().first >= x)
      amt += dq.back().second, dq.pop_back();
    dq.push_back({ x, amt });
    sz++;
  }

  void pop() {
    dq.front().second--, sz--;
    if (!dq.front().second) dq.pop_front();
  }
};
```

2.3 Ordered Set

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

typedef tree<int,null_type,less<int>,rb_tree_tag,
tree_order_statistics_node_update> ordered_set;

ordered_set s;
s.insert(2);
s.insert(3);
s.insert(7);
s.insert(9);

//find_by_order returns an iterator to the element at a given position
auto x = s.find_by_order(2);
cout << *x << "\n"; // 7

//order_of_key returns the position of a given element
cout << s.order_of_key(7) << "\n"; // 2

//If the element does not appear in the set, we get the position that the element would have in the set
cout << s.order_of_key(6) << "\n"; // 2
cout << s.order_of_key(8) << "\n"; // 3
```

2.4 Segment Tree

```
// Segment Tree (Range Query and Range Update)
// Update and Query - O(log n)

int n, v[N], lz[4*N], st[4*N];

void build(int p = 1, int l = 1, int r = n) {
  if (l == r) { st[p] = v[l]; return; }
  build(2*p, l, (l+r)/2);
  build(2*p+1, (l+r)/2+1, r);
  st[p] = min(st[2*p], st[2*p+1]); // RMQ -> min/max, RSQ -> +
}

void push(int p, int l, int r) {
  if (lz[p]) {
    st[p] = lz[p];
    // RMQ -> update: = lz[p],          increment: += lz[p]
    // RSQ -> update: = (r-l+1)*lz[p], increment: += (r-l+1)*lz[p]
    if(l!=r) lz[2*p] = lz[2*p+1] = lz[p]; // update: =, increment +=
    lz[p] = 0;
  }
}

int query(int i, int j, int p = 1, int l = 1, int r = n) {
  push(p, l, r);
  if (l > j or r < i) return INF; // RMQ -> INF, RSQ -> 0
  if (l >= i and j >= r) return st[p];
  return min(query(i, j, 2*p, l, (l+r)/2),
             query(i, j, 2*p+1, (l+r)/2+1, r));
  // RMQ -> min/max, RSQ -> +
}

void update(int i, int j, int v, int p = 1, int l = 1, int r = n) {
  push(p, l, r);
  if (l > j or r < i) return;
  if (l >= i and j >= r) { lz[p] = v; push(p, l, r); return; }
  update(i, j, v, 2*p, l, (l+r)/2);
  update(i, j, v, 2*p+1, (l+r)/2+1, r);
  st[p] = min(st[2*p], st[2*p+1]); // RMQ -> min/max, RSQ -> +
}
```

2.5 Dynamic Segment Tree (Lazy Update)

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

/* tested:
    https://www.spoj.com/problems/BGSHOOT/
   ref:
    https://maratona.ic.unicamp.br/MaratonaVerao2022/slides/AulaSummer-SegmentTree-Aula2.pdf
*/
vector<int> e, d, mx, lazy;
//begin creating node 0, then start your segment tree creating node 1
int create(){
    mx.push_back(0);
    lazy.push_back(0);
    e.push_back(0);
    d.push_back(0);
    return mx.size() - 1;
}

void push(int pos, int ini, int fim){
    if(pos == 0) return;
    if (lazy[pos]) {
        mx[pos] += lazy[pos];
        // RMQ (max/min)    -> update: = lazy[p],          incr: += lazy[p]
        // RSQ (sum)        -> update: = (r-l+1)*lazy[p], incr: += (r-l+1)*lazy[p]
        // Count lights on -> flip:    = (r-l+1)-st[p];
        if (ini != fim) {
            if(e[pos] == 0){
                int aux = create();
                e[pos] = aux;
            }
            if(d[pos] == 0){
                int aux = create();
                d[pos] = aux;
            }
            lazy[e[pos]] += lazy[pos];
            lazy[d[pos]] += lazy[pos];
            // update:    lazy[2*p] = lazy[p],  lazy[2*p+1] = lazy[p];
            // increment: lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
            // flip:      lazy[2*p] ^= 1,       lazy[2*p+1] ^= 1;
        }
        lazy[pos] = 0;
    }
}

void update(int pos, int ini, int fim, int p, int q, int val){
    if(pos == 0) return;

    push(pos, ini, fim);

    if(q < ini || p > fim) return;

    if(p <= ini and fim <= q){
        lazy[pos] += val;
        // update:    lazy[p] = k;
        // increment: lazy[p] += k;
        // flip:      lazy[p] = 1;
        push(pos, ini, fim);
        return;
    }

    int m = (ini + fim) >> 1;
    if(e[pos] == 0){
        int aux = create();
        e[pos] = aux;
    }
    update(e[pos], ini, m, p, q, val);
    if(d[pos] == 0){
        int aux = create();
        d[pos] = aux;
    }
    update(d[pos], m + 1, fim, p, q, val);
    mx[pos] = max(mx[e[pos]], mx[d[pos]]);
}

int query(int pos, int ini, int fim, int p, int q){
    if(pos == 0) return 0;

    push(pos, ini, fim);

    if(q < ini || p > fim) return 0;

    if(p <= ini and fim <= q) return mx[pos];

    int m = (ini + fim) >> 1;
    return max(query(e[pos], ini, m, p, q) , query(d[pos], m + 1, fim, p, q));
}
```

2.6 Segment Tree 2D

```
// Segment Tree 2D - O(nlog(n)log(n)) of Memory and Runtime
const int N = 1e8+5, M = 2e5+5;
int n, k=1, st[N], lc[N], rc[N];

void addx(int x, int l, int r, int u) {
  if (x < l or r < x) return;

  st[u]++;
  if (l == r) return;

  if(!rc[u]) rc[u] = ++k, lc[u] = ++k;
  addx(x, l, (l+r)/2, lc[u]);
  addx(x, (l+r)/2+1, r, rc[u]);
}

// Adds a point (x, y) to the grid.
void add(int x, int y, int l, int r, int u) {
  if (y < l or r < y) return;

  if (!st[u]) st[u] = ++k;
  addx(x, 1, n, st[u]);

  if (l == r) return;

  if(!rc[u]) rc[u] = ++k, lc[u] = ++k;
  add(x, y, l, (l+r)/2, lc[u]);
  add(x, y, (l+r)/2+1, r, rc[u]);
}

int countx(int x, int l, int r, int u) {
  if (!u or x < l) return 0;
  if (r <= x) return st[u];

  return countx(x, l, (l+r)/2, lc[u]) +
         countx(x, (l+r)/2+1, r, rc[u]);
}

// Counts number of points dominated by (x, y)
// Should be called with l = 1, r = n and u = 1
int count(int x, int y, int l, int r, int u) {
  if (!u or y < l) return 0;
  if (r <= y) return countx(x, 1, n, st[u]);

  return count(x, y, l, (l+r)/2, lc[u]) +
         count(x, y, (l+r)/2+1, r, rc[u]);
}
```

2.7 Trie

```
// Trie <O(|S|), O(|S|)>
int trie[N][26], trien = 1;

int add(int u, char c){
  c-='a';
  if (trie[u][c]) return trie[u][c];
  return trie[u][c] = ++trien;
}

//to add a string s in the trie
int u = 1;
for(char c : s) u = add(u, c);
```

2.8 Union Find (Rollback)

```
/*******************************************************************************
* DSU (DISJOINT SET UNION / UNION-FIND)                                        *
* Time complexity:  Unite - O(alpha n)                                         *
*                   Rollback - O(1)                                            *
*                   Find - O(alpha n)                                          *
* Usage: find(node), unite(node1, node2), sz[find(node)]                       *
* Notation: par: vector of parents                                             *
*           sz:  vector of subsets sizes, i.e. size of the subset a node is in *
*           sp:  stack containing node and par from last op                    *
*           ss:  stack containing node and size from last op                   *
*******************************************************************************/

int par[N], sz[N];
stack <pii> sp, ss;

int find (int a) { return par[a] == a ? a : find(par[a]); }

void unite (int a, int b) {
    if ((a = find(a)) == (b = find(b))) return;
    if (sz[a] < sz[b]) swap(a, b);
    ss.push({a, sz[a]});
    sp.push({b, par[b]});
    sz[a] += sz[b];
    par[b] = a;
}

void rollback() {
    par[sp.top().st] = sp.top().nd; sp.pop();
    sz[ss.top().st]  = ss.top().nd; ss.pop();
}

int main(){
    for (int i = 0; i < N; i++) par[i] = i, sz[i] = 1;
    return 0;
}
```

3 Dynamic Programming

3.1 Bitmasking

```
/*
    Optimal selection
*/

// We are given the prices of k products over n days,
// and we want to buy each product exactly once. However,
// we are allowed to buy at most one product in a day.
// What is the minimum total price?

// f[mask][i] = denote the minimum total price for buying
// a subset mask of products by day i
for (int j = 0; j < k; j++) {
    f[1<<j][0] = price[j][0];
}

for (int i = 1; i < n; i++) {
    for (int mask = 0; mask < (1<<k); mask++) {
        f[mask][i] = f[mask][i - 1];
        for (int j = 0; j < k; j++) {
            if (mask & (1<<j)) {
                f[mask][i] = min(f[mask][i],
                f[mask ^ (1<<j)][i - 1]+price[j][i]);
            }
        }
    }
}

/*
    From permutations to subsets
*/

// Take input and initialize stuff with default values.
// Then, we have the following.

// f[mask][i] = denotes the case where for all possible permutations,
// wrt the subset mask, i is the last element of the permutation.
for (int mask = 1; mask < (1 << n); mask++) {
    for (int i = 0; i < n; i++) {
        if (mask & (1 << i)) {
            // You might be required to initialize stuff here.
            for (int j: adj[i]) {
            // or, f(int j = 0; j < n; j++) whatever is more suitable
                if (mask & (1 << j) and i != j)
                    f[mask][i] = min(f[mask][i], f[mask ^ (1<<i)][j] + cost[i][j]);
            }
        }
    }
}

// Answer might be f[(1<<n)-1][n] or something like min(f[(1<<n)-1][i]) for all i from 0 to n.
```

3.2 Divide and Conquer Optimization

```
// Divide and Conquer DP Optimization - O(k*n^2) => O(k*n*logn)
//
// dp[i][j] = min k<i { dp[k][j-1] + C[k][i] }
//
// Condition: A[i][j] <= A[i+1][j]
// A[i][j] is the smallest k that gives an optimal answer to dp[i][j]
//
// reference (pt-br):
// https://algorithmmarch.wordpress.com/2016/08/12/a-otimizacao-de-pds-e-o-garcom-da-maratona/

int n, maxj;
int dp[N][J], a[N][J];

// declare the cost function
int cost(int i, int j) {
  // ...
}

void calc(int l, int r, int j, int kmin, int kmax) {
  int m = (l + r) / 2;
  dp[m][j] = LINF;

  for (int k = kmin; k <= kmax; ++k) {
    ll v = dp[k][j - 1] + cost(k, m);

    // store the minimum answer for d[m][j]
    // in case of maximum, use v > dp[m][j]
    if (v < dp[m][j])
      a[m][j] = k, dp[m][j] = v;
  }

  if (l < r) {
    calc(l, m, j, kmin, a[m][k]);
    calc(m + 1, r, j, a[m][k], kmax);
  }
}

// run for every j
for (int j = 2; j <= maxj; ++j)
  calc(1, n, j, 1, n);
```

3.3 Knapsack

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

// 0-1 Knapsack Code
// Time Complexity: O(nW)
// Space Complexity: O(W)

// initialize your dp
int knapsack(int W, int n, int wt[], int val[]) {
    int dp[n+1][W+1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= W; j++) {
            if (i == 0 || j == 0) {
                dp[i][j] = 0;
            } else if (wt[i-1] <= j) {
                dp[i][j] = max(dp[i-1][j], dp[i-1][j-wt[i-1]] + val[i-1]);
            } else {
                dp[i][j] = dp[i-1][j];
            }
        }
    }
    return dp[n][W];
}

// Fractional Knapsack
// Time Complexity: O(nW)
// Space Complexity: O(W)
int fractionalKnapsack(int n, int W, int w[], int v[]) {
    int dp[n+1][W+1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= W; j++) {
            if (i == 0 || j == 0) {
                dp[i][j] = 0;
            } else if (j < w[i-1]) {
                dp[i][j] = dp[i-1][j];
            } else {
                dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i-1]] + v[i-1]*w[i-1]/j);
            }
        }
    }
    return dp[n][W];
}
```

3.4 Longest XYZ Subsequence

```
// Longest Increasing Subsequence - O(nlogn)
//
// dp(i) = max j<i { dp(j) | a[j] < a[i] } + 1
//
int longestIncreasingSubsequence(vector<int>& a) {
    int n = a.size();
    vector<int> dp(n);

    for (int i = 0; i < n; i++) {
        int mx = 0;
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i])
                mx = max(mx, dp[j]);
        }
        dp[i] = mx + 1;
    }
    return *max_element(dp.begin(), dp.end());
}

// Longest Common Subsequence - O(mn)
//
// dp(i, j) = max { dp(i-1, j-1) | a[i] == b[j] } + 1
//
int longestCommonSubsequence(vector<int>& a, vector<int>& b) {
    int n = a.size();
    int m = b.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i-1] == b[j-1])
                dp[i][j] = dp[i-1][j-1] + 1;
            else
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }
    return dp[n][m];
}
```

3.5 Sum over Subets

```
/*
        SOS DP
*/
// O(N * 2^N)
// A[i] = initial values
// Calculate F[i] = Sum of A[j] for j subset of i

//iterative version
for (int mask = 0; mask < (1<<N); ++mask) {
        dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
        for (int i = 0;i < N; ++i) {
                if (mask & (1<<i))
                        dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
                else
                        dp[mask][i] = dp[mask][i-1];
        }
        F[mask] = dp[mask][N-1];
}

//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
        F[i] = A[i];
for(int i = 0;i < N; ++i)
  for(int mask = 0; mask < (1<<N); ++mask)  {
          if(mask & (1<<i))
                  F[mask] += F[mask^(1<<i)];
}
```

4 Graphs

4.1 DFS

```
/***********************************************************************************
* DFS (DEPTH-FIRST SEARCH)                                                         *
* Time complexity: O(V+E)                                                          *
                                                             *
* Notation: adj[x]: adjacency list for node x                                      *
*           vis[i]: visited state for node i (0 or 1)                              *
***********************************************************************************/

const int N = 1e5+10;
int vis[N];
vector<int> adj[N];

void dfs(int u) {
  vis[u] = 1;
  for (int v : adj[u]){
    if (!vis[v]) {
        dfs(v);
    }
  }
    // vis[u] = 0;
    // Uncomment the line above if you need to
    // traverse only one path at a time (backtracking)
}
```

4.2 BFS

```
/***********************************************************************************
* BFS (BREADTH-FIRST SEARCH)                                                       *
* Time complexity: O(V+E)                                                          *
* Usage: bfs(node)                                                                 *
* Notation: s: starting node                                                       *
*           adj[i]: adjacency list for node i                                      *
*           vis[i]: visited state for node i (0 or 1)                              *
***********************************************************************************/

const int N = 1e5+10; // Maximum number of nodes
int dist[N], par[N];
vector <int> adj[N];
queue <int> q;

void bfs (int s) {
    memset(dist, 63, sizeof(dist));
    dist[s] = 0;
    q.push(s);

    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (auto v : adj[u]) if (dist[v] > dist[u] + 1) {
            par[v] = u;
            dist[v] = dist[u] + 1;
            q.push(v);
        }
    }
}
```

4.3 Toposort

```
/***********************************************************************************
* KAHN’S ALGORITHM (TOPOLOGICAL SORTING)                                           *
                                                                 *
* Time complexity: O(V+E)                                                          *
* Notation: adj[i]:  adjacency matrix for node i                                   *
*           n:       number of vertices                                            *
*           e:       number of edges                                               *
*           a, b:    edge between a and b                                          *
*           inc:     number of incoming arcs/edges                                 *
*           q:       queue with the independent vertices                           *
*           tsort:   final topo sort, i.e. possible order to traverse graph        *
***********************************************************************************/
```

```
vector <int> adj[N];
int inc[N]; // number of incoming arcs/edges

// undirected graph: inc[v] <= 1
// directed graph:   inc[v] == 0

queue<int> q;
for (int i = 1; i <= n; ++i) if (inc[i] <= 1) q.push(i);

while (!q.empty()) {
  int u = q.front(); q.pop();
  for (int v : adj[u])
    if (inc[v] > 1 and --inc[v] <= 1)
      q.push(v);
}
```

4.4 Articulation points and bridges

```
// Articulation points and Bridges O(V+E)
int par[N], art[N], low[N], num[N], ch[N], cnt;

void articulation(int u) {
  low[u] = num[u] = ++cnt;
  for (int v : adj[u]) {
    if (!num[v]) {
      par[v] = u; ch[u]++;
      articulation(v);
      if (low[v] >= num[u]) art[u] = 1;
      if (low[v] >  num[u]) { /* u-v bridge */ }
      low[u] = min(low[u], low[v]);
    }
    else if (v != par[u]) low[u] = min(low[u], num[v]);
  }
}

for (int i = 0; i < n; ++i) if (!num[i])
  articulation(i), art[i] = ch[i]>1;
```

4.5 Shortest Path (Bellman-Ford)

```
/***********************************************************************************
* BELLMAN-FORD ALGORITHM (SHORTEST PATH TO A VERTEX - WITH NEGATIVE COST)          *
* Time complexity: O(VE)                                                           *
* Usage: dist[node]                                                                *
* Notation: m:             number of edges                                         *
*           n:             number of vertices                                      *
*           (a, b, w):     edge between a and b with weight w                      *
*           s:             starting node                                           *
***********************************************************************************/
const int N = 1e4+10; // Maximum number of nodes
vector<int> adj[N], adjw[N];
int dist[N], v, w;

memset(dist, 63, sizeof(dist));
dist[0] = 0;
for (int i = 0; i < n-1; ++i)
  for (int u = 0; u < n; ++u)
    for (int j = 0; j < adj[u].size(); ++j)
      v = adj[u][j], w = adjw[u][j],
      dist[v] = min(dist[v], dist[u]+w);
```

4.6 Shortest Path (Dijkstra)

```
/***********************************************************************************
* DIJKSTRA’S ALGORITHM (SHORTEST PATH TO A VERTEX)                                 *
* Time complexity: O((V+E)logE)                                                    *
* Usage: dist[node]                                                                *
* Notation: m:             number of edges                                         *
*           (a, b, w):     edge between a and b with weight w                      *
*           s:             starting node                                           *
*           par[v]:        parent node of u, used to rebuild the shortest path     *
***********************************************************************************/

vector<int> adj[N], adjw[N];
```

```
int dist[N];

memset(dist, 63, sizeof(dist));
priority_queue<pii> pq;
pq.push(mp(0,0));

while (!pq.empty()) {
  int u = pq.top().nd;
  int d = -pq.top().st;
  pq.pop();

  if (d > dist[u]) continue;
  for (int i = 0; i < adj[u].size(); ++i) {
    int v = adj[u][i];
    int w = adjw[u][i];
    if (dist[u] + w < dist[v])
      dist[v] = dist[u]+w, pq.push(mp(-dist[v], v));
  }
}
```

4.7 Floyd Warshall

```
/*********************************************************************************
* FLOYD-WARSHALL ALGORITHM (SHORTEST PATH TO ANY VERTEX)                         *
* Time complexity: O(V^3)                                                        *
* Usage: dist[from][to]                                                          *
* Notation: m:           number of edges                                         *
*           n:           number of vertices                                      *
*           (a, b, w):   edge between a and b with weight w                      *
*********************************************************************************/

int adj[N][N]; // no-edge = INF

for (int k = 0; k < n; ++k)
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      adj[i][j] = min(adj[i][j], adj[i][k]+adj[k][j]);
```

4.8 Zero One BFS

```
// 0-1 BFS - O(V+E)

const int N = 1e5 + 5;

int dist[N];
vector<pii> adj[N];
deque<pii> dq;

void zero_one_bfs (int x){
    cl(dist, 63);
    dist[x] = 0;
    dq.push_back({x, 0});
    while(!dq.empty()){
        int u = dq.front().st;
        int ud = dq.front().nd;
        dq.pop_front();
        if(dist[u] < ud) continue;
        for(auto x : adj[u]){
            int v = x.st;
            int w = x.nd;
            if(dist[u] + w < dist[v]){
                dist[v] = dist[u] + w;
                if(w) dq.push_back({v, dist[v]});
                else dq.push_front({v, dist[v]});
            }
        }
    }
}
```

4.9 MST (Kruskal)

```
/*********************************************************************************
* KRUSKAL'S ALGORITHM (MINIMAL SPANNING TREE - INCREASING EDGE SIZE)             *
* Time complexity: O(ElogE)                                                      *
* Usage: cost, sz[find(node)]                                                    *
* Notation: cost: sum of all edges which belong to such MST                      *
*           sz:   vector of subsets sizes, i.e. size of the subset a node is in  *
*********************************************************************************/

// + Union-find

int cost = 0;
vector <pair<int, pair<int, int>>> edges; //mp(dist, mp(node1, node2))

int main () {
    // ...
    sort(edges.begin(), edges.end());
    for (auto e : edges)
        if (find(e.nd.st) != find(e.nd.nd))
            unite(e.nd.st, e.nd.nd), cost += e.st;

    return 0;
}
```

4.10 Lowest Common Ancestor

```
// Lowest Common Ancestor <O(nlogn), O(logn)>
const int N = 1e6, M = 25;
int anc[M][N], h[N], rt;

// TODO: Calculate h[u] and set anc[0][u] = parent of node u for each u

// build (sparse table)
anc[0][rt] = rt; // set parent of the root to itself
for (int i = 1; i < M; ++i)
  for (int j = 1; j <= n; ++j)
    anc[i][j] = anc[i-1][anc[i-1][j]];

// query
int lca(int u, int v) {
  if (h[u] < h[v]) swap(u, v);
  for (int i = M-1; i >= 0; --i) if (h[u]-(1<<i) >= h[v])
    u = anc[i][u];

  if (u == v) return u;

  for (int i = M-1; i >= 0; --i) if (anc[i][u] != anc[i][v])
    u = anc[i][u], v = anc[i][v];
  return anc[0][u];
}
```

4.11 Strongly Connected Components

```
// Kosaraju - SCC O(V+E)
// For undirected graph uncomment lines below

vi adj[N], adjt[N];
int n, ordn, cnt, vis[N], ord[N], cmp[N];
//int par[N];

void dfs(int u) {
  vis[u] = 1;
  for (auto v : adj[u]) if (!vis[v]) dfs(v);
  // for (auto v : adj[u]) if(!vis[v]) par[v] = u, dfs(v);
  ord[ordn++] = u;
}

void dfst(int u) {
  cmp[u] = cnt, vis[u] = 0;
  for (auto v : adjt[u]) if (vis[v]) dfst(v);
  // for (auto v : adj[u]) if(vis[v] and u != par[v]) dfst(v);
}

// in main
for (int i = 1; i <= n; ++i) if (!vis[i]) dfs(i);
for (int i = ordn-1; i >= 0; --i) if (vis[ord[i]]) cnt++, dfst(ord[i]);
```

5 Strings

5.1 Knuth-Morris-Pratt

```
// Knuth-Morris-Pratt - String Matching O(n+m)
char s[N], p[N];
int b[N], n, m; // n = strlen(s), m = strlen(p);

void kmppre() {
  b[0] = -1;
  for (int i = 0, j = -1; i < m; b[++i] = ++j)
    while (j >= 0 and p[i] != p[j])
      j = b[j];
}

void kmp() {
  for (int i = 0, j = 0; i < n;) {
    while (j >= 0 and s[i] != p[j]) j=b[j];
    i++, j++;
    if (j == m) {
      // match position i-j
      j = b[j];
    }
  }
}
```

5.2 Knuth-Morris-Pratt (Automaton)

```
// KMP Automaton - <O(26*pattern), O(text)>

// max size pattern
const int N = 1e5 + 5;

int cnt, nxt[N+1][26];

void prekmp(string &p) {
  nxt[0][p[0] - 'a'] = 1;
  for(int i = 1, j = 0; i <= p.size(); i++) {
    for(int c = 0; c < 26; c++) nxt[i][c] = nxt[j][c];
    if(i == p.size()) continue;
    nxt[i][p[i] - 'a'] = i+1;
    j = nxt[j][p[i] - 'a'];
  }
}

void kmp(string &s, string &p) {
  for(int i = 0, j = 0; i < s.size(); i++) {
    j = nxt[j][s[i] - 'a'];
    if(j == p.size()) cnt++; //match i - j + 1
  }
}
```

5.3 Rabin-Karp

```
// Rabin-Karp - String Matching + Hashing O(n+m)
const int B = 31;
char s[N], p[N];
int n, m; // n = strlen(s), m = strlen(p)

void rabin() {
  if (n<m) return;

  ull hp = 0, hs = 0, E = 1;
  for (int i = 0; i < m; ++i)
    hp = ((hp*B)%MOD + p[i])%MOD,
    hs = ((hs*B)%MOD + s[i])%MOD,
    E = (E*B)%MOD;

  if (hs == hp) { /* matching position 0 */ }
  for (int i = m; i < n; ++i) {
    hs = ((hs*B)%MOD + s[i])%MOD;
    hhs = (hs - s[i-m]*E%MOD + MOD)%MOD;
    if (hs == hp) { /* matching position i-m+1 */ }
  }
}
```

5.4 String Hashing

```
// String Hashing
// Rabin Karp - O(n + m)

// max size txt + 1
const int N = 1e6 + 5;

// lowercase letters p = 31 (remember to do s[i] - 'a' + 1)
// uppercase and lowercase letters p = 53 (remember to do s[i] - 'a' + 1)
// any character p = 313

const int MOD = 1e9+9;
ull h[N], p[N];
ull pr = 313;

int cnt;


void build(string &s) {
  p[0] = 1, p[1] = pr;
  for(int i = 1; i <= s.size(); i++) {
    h[i] = ((p[1]*h[i-1]) % MOD + s[i-1]) % MOD;
    p[i] = (p[1]*p[i-1]) % MOD;
  }
}

// 1-indexed
ull fhash(int l, int r) {
  return (h[r] - ((h[l-1]*p[r-l+1]) % MOD) + MOD) % MOD;
}

ull shash(string &pt) {
  ull h = 0;
  for(int i = 0; i < pt.size(); i++)
    h = ((h*pr) % MOD + pt[i]) % MOD;
  return h;
}

void rabin_karp(string &s, string &pt) {
  build(s);
  ull hp = shash(pt);
  for(int i = 0, m = pt.size(); i + m <= s.size(); i++) {
    if(fhash(i+1, i+m) == hp) {
      // match at i
      cnt++;
    }
  }
}
```

5.5 String Multihashing

```
// String Hashing
// Rabin Karp - O(n + m)
template <int N = 3>
struct Hash {
  int hs[N];
  static vector<int> mods;

  static int add(int a, int b, int mod) { return a >= mod - b ? a + b - mod : a + b; }
  static int sub(int a, int b, int mod) { return a - b < 0 ? a - b + mod : a - b; }
  static int mul(int a, int b, int mod) { return 1ll * a * b % mod; }

  Hash(int x = 0) { fill(hs, hs + N, x); }

  bool operator<(const Hash& b) const {
    for (int i = 0; i < N; i++) {
      if (hs[i] < b.hs[i]) return true;
      if (hs[i] > b.hs[i]) return false;
    }
    return false;
  }

  Hash operator+(const Hash& b) const {
    Hash ans;
    for (int i = 0; i < N; i++) ans.hs[i] = add(hs[i], b.hs[i], mods[i]);
    return ans;
  }

  Hash operator-(const Hash& b) const {
```

```cpp
    Hash ans;
    for (int i = 0; i < N; i++) ans.hs[i] = sub(hs[i], b.hs[i], mods[i]);
    return ans;
  }

  Hash operator*(const Hash& b) const {
    Hash ans;
    for (int i = 0; i < N; i++) ans.hs[i] = mul(hs[i], b.hs[i], mods[i]);
    return ans;
  }

  Hash operator+(int b) const {
    Hash ans;
    for (int i = 0; i < N; i++) ans.hs[i] = add(hs[i], b, mods[i]);
    return ans;
  }

  Hash operator*(int b) const {
    Hash ans;
    for (int i = 0; i < N; i++) ans.hs[i] = mul(hs[i], b, mods[i]);
    return ans;
  }

  friend Hash operator*(int a, const Hash& b) {
    Hash ans;
    for (int i = 0; i < N; i++) ans.hs[i] = mul(b.hs[i], a, b.mods[i]);
    return ans;
  }

  friend ostream& operator<<(ostream& os, const Hash& b) {
    for (int i = 0; i < N; i++) os << b.hs[i] << " \n"[i == N - 1];
    return os;
  }
};

template <int N> vector<int> Hash<N>::mods = { (int) 1e9 + 9, (int) 1e9 + 33, (int) 1e9 + 87 };

// In case you need to generate the MODs, uncomment this:
// Obs: you may need this on your template
// mt19937_64 llrand((int) chrono::steady_clock::now().time_since_epoch().count());
// In main: gen<>();
/*
template <int N> vector<int> Hash<N>::mods;
template<int N = 3>
void gen() {
  while (Hash<N>::mods.size() < N) {
    int mod;
    bool is_prime;
    do {
      mod = (int) 1e8 + (int) (llrand() % (int) 9e8);
      is_prime = true;
      for (int i = 2; i * i <= mod; i++) {
        if (mod % i == 0) {
          is_prime = false;
          break;
        }
      }
    } while (!is_prime);
    Hash<N>::mods.push_back(mod);
  }
}
*/

template <int N = 3>
struct PolyHash {
  vector<Hash<N>> h, p;

  PolyHash(string& s, int pr = 313) {
    int sz = (int)s.size();
    p.resize(sz + 1);
    h.resize(sz + 1);

    p[0] = 1, h[0] = s[0];
    for (int i = 1; i < sz; i++) {
      h[i] = pr * h[i - 1] + s[i];
      p[i] = pr * p[i - 1];
    }
  }

  Hash<N> fhash(int l, int r) {
    if (!l) return h[r];
    return h[r] - h[l - 1] * p[r - l + 1];
  }

  static Hash<N> shash(string& s, int pr = 313) {
    Hash<N> ans;
    for (int i = 0; i < (int)s.size(); i++) ans = pr * ans + s[i];
    return ans;
  }

  friend int rabin_karp(string& s, string& pt) {
    PolyHash hs = PolyHash(s);
    Hash<N> hp = hs.shash(pt);
    int cnt = 0;
    for (int i = 0, m = (int)pt.size(); i + m <= (int)s.size(); i++) {
      if (hs.fhash(i, i + m - 1) == hp) {
        // match at i
        cnt++;
      }
    }

    return cnt;
  }
};
```

5.6 Manacher

```cpp
// Manacher (Longest Palindromic String) - O(n)
int lps[2*N+5];
char s[N];

int manacher() {
  int n = strlen(s);

  string p (2*n+3, '#');
  p[0] = '^';
  for (int i = 0; i < n; i++) p[2*(i+1)] = s[i];
  p[2*n+2] = '$';

  int k = 0, r = 0, m = 0;
  int l = p.length();
  for (int i = 1; i < l; i++) {
    int o = 2*k - i;
    lps[i] = (r > i) ? min(r-i, lps[o]) : 0;
    while (p[i + 1 + lps[i]] == p[i - 1 - lps[i]]) lps[i]++;
    if (i + lps[i] > r) k = i, r = i + lps[i];
    m = max(m, lps[i]);
  }
  return m;
}
```

6 Mathematics

6.1 Basics

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long int
#define ld long double
#define ull unsigned long long int
#define N (int)1e7
#define MOD 1000000007

// Greatest Common Divisor & Lowest Common Multiple
ll gcd(ll a, ll b) { return b ? gcd(b, a%b) : a; }
ll lcm(ll a, ll b) { return a/gcd(a, b)*b; }

// Fast exponentiation
ll power(ll a, ll b, ll m = MOD) {
  ll ans = 1;
  while (b) {
    if (b & 1)
      ans = (ans * a) % m;
    a = (a * a) % m;
    b >>= 1;
  }
  return ans;
}

// Multiply caring overflow
ll mulmod(ll a, ll b, ll m = MOD) {
  ll r=0;
  for (a %= m; b; b>>=1, a=(a*2)%m) if (b&1) r=(r+a)%m;
  return r;
}

// Another option for mulmod is using long double
```

```
ull mulmod(ull a, ull b, ull m = MOD) {
  ull q = (ld) a * (ld) b / (ld) m;
  ull r = a * b - q * m;
  return (r + m) % m;
}

// Matrix exponentiation
vector<vector<ll>> matmul(vector<vector<ll>> a, vector<vector<ll>> b) {
  int n = a.size();
  vector<vector<ll>> ans(n, vector<ll>(n));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++)
        ans[i][j] = (ans[i][j] + a[i][k] * b[k][j]) % MOD;
  return ans;
}

vector<vector<ll>> matpow(vector<vector<ll>> a, ll n) {
  if (n == 0) return vector<vector<ll>>(1, vector<ll>(1, 1));
  if (n == 1) return a;
  vector<vector<ll>> b = matpow(a, n/2);
  b = matmul(b, b);
  if (n % 2) b = matmul(b, a);
  return b;
}

// nCr % p using Fermat's little theorem.
ll f[N];
void factorial(int n) {
  f[0] = 1;
  for (int i = 1; i <= n; i++)
    f[i] = (f[i - 1] * i) % MOD;
}
ll modinv(ll n, int p) {
  return power(n, p - 2, p);
}
ll C(ll n, int r, int p) {
  if (n < r)
    return 0;
  if (r == 0)
    return 1;
  return (f[n] * modinv(f[r], p) % p * modinv(f[n - r], p) % p) % p;
}
```

6.2 Advanced

```
/* Line integral = integral(sqrt(1 + (dy/dx)^2)) dx */

/* Multiplicative Inverse over MOD for all 1..N - 1 < MOD in O(N)
 Only works for prime MOD. If all 1..MOD - 1 needed, use N = MOD */
ll inv[N];
inv[1] = 1;
for(int i = 2; i < N; ++i)
        inv[i] = MOD - (MOD / i) * inv[MOD % i] % MOD;

/* Catalan
 f(n) = sum(f(i) * f(n - i - 1)), i in [0, n - 1] = (2n)! / ((n+1)! * n!) = ...
 If you have any function f(n) (there are many) that follows this sequence (0-indexed):
 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440
 than it's the Catalan function */
ll cat[N];
cat[0] = 1;
for(int i = 1; i + 1 < N; i++) // needs inv[i + 1] till inv[N - 1]
        cat[i] = 2ll * (2ll * i - 1) * inv[i + 1] % MOD * cat[i - 1] % MOD;

/* Floor(n / i), i = [1, n], has <= 2 * sqrt(n) diff values.
 Proof: i = [1, sqrt(n)] has sqrt(n) diff values.
 For i = [sqrt(n), n] we have that 1 <= n / i <= sqrt(n)
 and thus has <= sqrt(n) diff values.
*/

/* l = first number that has floor(N / l) = x
 r = last number that has floor(N / r) = x
 N / r >= floor(N / l)
 r <= N / floor(N / l)*/
for(int l = 1, r; l <= n; l = r + 1){
        r = n / (n / l);
        // floor(n / i) has the same value for l <= i <= r
}

/*
Recurrence using matrix:
        h[i + 2] = a1 * h[i + 1] + a0 * h[i]
        [h[i] h[i-1]] = [h[1] h[0]] * [a1 1] ^ (i - 1)
                                                                  [a0 0]
```

```
*/

/* Mobius Inversion:
  Sum of gcd(i, j), 1 <= i, j <= N?
  sum(k->N) k * sum(i->N) sum(j->N) [gcd(i, j) == k], i = a * k, j = b * k
  = sum(k->N) k * sum(a->N/k) sum(b->N/k) [gcd(a, b) == 1]
  = sum(k->N) k * sum(a->N/k) sum(b->N/k) sum(d->N/k) [d | a] * [d | b] * mi(d)
  = sum(k->N) k * sum(d->N/k) mi(d) * floor(N / kd)^2, l = kd, l <= N, k | l, d = l / k
  = sum(l->N) floor(N / l)^2 * sum(k|l) k * mi(l / k)
  If f(n) = sum(x|n)(g(x) * h(x)) with g(x) and h(x) multiplicative, than f(n) is multiplicative
  Hence, g(l) = sum(k|l) k * mi(l / k) is multiplicative
  = sum(l->N) floor(N / l)^2 * g(l)
*/
```

6.3 Sieve of Eratosthenes

```
// Sieve of Erasthotenes
int p[N]; vi primes;

for (ll i = 2; i < N; ++i) if (!p[i]) {
  for (ll j = i*i; j < N; j+=i) p[j]=1;
  primes.pb(i);
}
```

6.4 Extended Euclidean and Chinese Remainder

```
// Extended Euclid:
void euclid(ll a, ll b, ll &x, ll &y) {
  if (b) euclid(b, a%b, y, x), y -= x*(a/b);
  else x = 1, y = 0;
}

// find (x, y) such that a*x + b*y = c or return false if it's not possible
// [x + k*b/gcd(a, b), y - k*a/gcd(a, b)] are also solutions
bool diof(ll a, ll b, ll c, ll &x, ll &y){
  euclid(abs(a), abs(b), x, y);
  ll g = abs(__gcd(a, b));
  if(c % g) return false;
  x *= c / g;
  y *= c / g;
  if(a < 0) x = -x;
  if(b < 0) y = -y;
  return true;
}

// auxiliar to find_all_solutions
void shift_solution (ll &x, ll &y, ll a, ll b, ll cnt) {
  x += cnt * b;
  y -= cnt * a;
}

// Find the amount of solutions of
// ax + by = c
// in given intervals for x and y
ll find_all_solutions (ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll maxy) {
  ll x, y, g = __gcd(a, b);
  if(!diof(a, b, c, x, y)) return 0;
  a /= g; b /= g;

  int sign_a = a>0 ? +1 : -1;
  int sign_b = b>0 ? +1 : -1;

  shift_solution (x, y, a, b, (minx - x) / b);
  if (x < minx)
    shift_solution (x, y, a, b, sign_b);
  if (x > maxx)
    return 0;
  int lx1 = x;

  shift_solution (x, y, a, b, (maxx - x) / b);
  if (x > maxx)
    shift_solution (x, y, a, b, -sign_b);
  int rx1 = x;

  shift_solution (x, y, a, b, - (miny - y) / a);
  if (y < miny)
    shift_solution (x, y, a, b, -sign_a);
  if (y > maxy)
    return 0;
  int lx2 = x;
  shift_solution (x, y, a, b, - (maxy - y) / a);
  if (y > maxy)
    shift_solution (x, y, a, b, sign_a);
  int rx2 = x;

  if (lx2 > rx2)
    swap (lx2, rx2);
  int lx = max (lx1, lx2);
  int rx = min (rx1, rx2);

  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}

bool crt_auxiliar(ll a, ll b, ll m1, ll m2, ll &ans){
  ll x, y;
  if(!diof(m1, m2, b - a, x, y)) return false;
  ll lcm = m1 / __gcd(m1, m2) * m2;
  ans = ((a + x % (lcm / m1) * m1) % lcm + lcm) % lcm;
  return true;
}

// find ans such that ans = a[i] mod b[i] for all 0 <= i < n or return false if not possible
// ans + k * lcm(b[i]) are also solutions
bool crt(int n, ll a[], ll b[], ll &ans){
  if(!b[0]) return false;
  ans = a[0] % b[0];
  ll l = b[0];
  for(int i = 1; i < n; i++){
   if(!b[i]) return false;
    if(!crt_auxiliar(ans, a[i] % b[i], l, b[i], ans)) return false;
    l *= (b[i] / __gcd(b[i], l));
  }
  return true;
}
```

6.5 Euler Phi

```
// Euler phi (totient)
int ind = 0, pf = primes[0], ans = n;
while (1ll*pf*pf <= n) {
  if (n%pf==0) ans -= ans/pf;
  while (n%pf==0) n /= pf;
  pf = primes[++ind];
}
if (n != 1) ans -= ans/n;

// IME2014
int phi[N];
void totient() {
  for (int i = 1; i < N; ++i)  phi[i]=i;
  for (int i = 2; i < N; i+=2) phi[i]>>=1;
  for (int j = 3; j < N; j+=2) if (phi[j]==j) {
    phi[j]--;
    for (int i = 2*j; i < N; i+=j) phi[i]=phi[i]/j*(j-1);
  }
}
```

6.6 Prime Factors

```
// Prime factors (up to 9*10^13. For greater see Pollard Rho)
vi factors;
int ind=0, pf = primes[0];
while (pf*pf <= n) {
  while (n%pf == 0) n /= pf, factors.pb(pf);
  pf = primes[++ind];
}
if (n != 1) factors.pb(n);
```

6.7 Miller-Rabin

```
bool isPrime(unsigned ll n) {
        if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
        unsigned ll A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
            s = __builtin_ctzll(n-1), d = n >> s;
        for (ull a : A) {   // ^ count trailing zeroes
                ull p = modpow(a%n, d, n), i = s;
                while (p != 1 && p != n - 1 && a % n && i--)
                        p = modmul(p, p, n);
                if (p != n-1 && i != s) return 0;
        }
        return 1;
}
```

6.8 Pollard-Rho

```
// factor(N, v) to get N factorized in vector v
// O(N ^ (1 / 4)) on average
// Miller-Rabin - Primarily Test O(|base|*(logn)^2)
ll addmod(ll a, ll b, ll m){
        if(a >= m - b) return a + b - m;
        return a + b;
}

ll mulmod(ll a, ll b, ll m){
        ll ans = 0;
        while(b){
                if(b & 1) ans = addmod(ans, a, m);
                a = addmod(a, a, m);
                b >>= 1;
        }
        return ans;
}

ll fexp(ll a, ll b, ll n){
        ll r = 1;
        while(b){
                if(b & 1) r = mulmod(r, a, n);
                a = mulmod(a, a, n);
                b >>= 1;
        }
        return r;
}

bool miller(ll a, ll n){
        if (a >= n) return true;
        ll s = 0, d = n - 1;
        while(d % 2 == 0) d >>= 1, s++;
        ll x = fexp(a, d, n);
        if (x == 1 || x == n - 1) return true;
        for (int r = 0; r < s; r++, x = mulmod(x,x,n)){
                if (x == 1) return false;
                if (x == n - 1) return true;
        }
        return false;
}

bool isprime(ll n){
        if(n == 1) return false;
        int base[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
        for (int i = 0; i < 12; ++i) if (!miller(base[i], n)) return false;
        return true;
}

ll pollard(ll n){
        ll x, y, d, c = 1;
        if (n % 2 == 0) return 2;
        while(true){
                y = x = 2;
                while(true){
                        x = addmod(mulmod(x,x,n), c, n);
                        y = addmod(mulmod(y,y,n), c, n);
                        y = addmod(mulmod(y,y,n), c, n);
                        if (x == y) break;
                        d = __gcd(abs(x-y), n);
                        if (d > 1) return d;
                }
                c++;
        }
}

vector<ll> factor(ll n){
        if (n == 1 || isprime(n)) return {n};
        ll f = pollard(n);
        vector<ll> l = factor(f), r = factor(n / f);
        l.insert(l.end(), r.begin(), r.end());
        sort(l.begin(), l.end());
        return l;
}
```

```
//n < 2,047 base = {2};
//n < 9,080,191 base = {31, 73};
//n < 2,152,302,898,747 base = {2, 3, 5, 7, 11};
//n < 318,665,857,834,031,151,167,461 base = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
//n < 3,317,044,064,679,887,385,961,981 base = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41};
```

6.9 Fast Fourier Transform

```
// Fast Fourier Transform - O(nlogn)

/*
// Use struct instead. Performance will be way better!
typedef complex<ld> T;
T a[N], b[N];
*/

struct T {
  ld x, y;
  T() : x(0), y(0) {}
  T(ld a, ld b=0) : x(a), y(b) {}

  T operator/=(ld k) { x/=k; y/=k; return (*this); }
  T operator*(T a) const { return T(x*a.x - y*a.y, x*a.y + y*a.x); }
  T operator+(T a) const { return T(x+a.x, y+a.y); }
  T operator-(T a) const { return T(x-a.x, y-a.y); }
} a[N], b[N];

// a: vector containing polynomial
// n: power of two greater or equal product size
/*
// Use iterative version!
void fft_recursive(T* a, int n, int s) {
  if (n == 1) return;
  T tmp[n];
  for (int i = 0; i < n/2; ++i)
    tmp[i] = a[2*i], tmp[i+n/2] = a[2*i+1];

  fft_recursive(&tmp[0], n/2, s);
  fft_recursive(&tmp[n/2], n/2, s);

  T wn = T(cos(s*2*PI/n), sin(s*2*PI/n)), w(1,0);
  for (int i = 0; i < n/2; i++, w=w*wn)
    a[i] = tmp[i] + w*tmp[i+n/2],
    a[i+n/2] = tmp[i] - w*tmp[i+n/2];
}
*/

void fft(T* a, int n, int s) {
  for (int i=0, j=0; i<n; i++) {
    if (i>j) swap(a[i], a[j]);
    for (int l=n/2; (j^=l) < l; l>>=1);
  }

  for(int i = 1; (1<<i) <= n; i++){
    int M = 1 << i;
    int K = M >> 1;
    T wn = T(cos(s*2*PI/M), sin(s*2*PI/M));
    for(int j = 0; j < n; j += M) {
      T w = T(1, 0);
      for(int l = j; l < K + j; ++l){
        T t = w*a[l + K];
        a[l + K] = a[l]-t;
        a[l] = a[l] + t;
        w = wn*w;
      }
    }
  }
}

// assert n is a power of two greater of equal product size
// n = na + nb; while (n&(n-1)) n++;
void multiply(T* a, T* b, int n) {
  fft(a,n,1);
  fft(b,n,1);
  for (int i = 0; i < n; i++) a[i] = a[i]*b[i];
  fft(a,n,-1);
  for (int i = 0; i < n; i++) a[i] /= n;
}

// Convert to integers after multiplying:
// (int)(a[i].x + 0.5);
```

7 Miscellaneous

7.1 Bitset

```
//Goes through the subsets of a set x :
int b = 0;
do {
// process subset b
} while (b=(b-x)&x);
```

7.2 builtin

```
__builtin_ctz(x) // trailing zeroes
__builtin_clz(x) // leading zeroes
__builtin_popcount(x) // # bits set
__builtin_ffs(x) // index(LSB) + 1 [0 if x==0]

// Add ll to the end for long long [__builtin_clzll(x)]
```

7.3 Max of all fixed length subarrays

```
/*
    Finding the maximum for all subarrays of fixed length
*/
std::deque<int> Q(k);
int i = 0;
for (; i < k; i++)
{
                while(!Q.empty() and arr[i] >= arr[Q.back()])
                                Q.pop_back();
}

for (; i < n; i++)
{
    cout << arr[Q.front()] << " ";
    while (!Q.empty() && Q.front() <= i - k)
        Q.pop_front();
    while ((!Q.empty()) && arr[i] >= arr[Q.back()])
        Q.pop_back();
    Q.push_back(i);
}
cout << arr[Q.front()];
```

7.4 Merge Sort (Inversion Count)

```
// Merge-sort with inversion count - O(nlog n)

int n, inv;
vector<int> v, ans;

void mergesort(int l, int r, vector<int> &v){
    if(l == r) return;
    int mid = (l+r)/2;
    mergesort(l, mid, v), mergesort(mid+1, r, v);
    int i = l, j = mid + 1, k = l;
    while(i <= mid or j <= r){
        if(i <= mid and (j > r or v[i] <= v[j])) ans[k++] = v[i++];
        else ans[k++] = v[j++], inv += j-k;
    }
    for(int i = l; i <= r; i++) v[i] = ans[i];
}

//in main
ans.resize(v.size());
```

8 Math Extra

8.1 Seldom used combinatorics

- Fibonacci in $O(\log(N))$ with memoization is:

$$f(0) = f(1) = 1$$
$$f(2k) = f(k)^2 + f(k-1)^2$$
$$f(2k+1) = f(k) \times (f(k) + 2 * f(k-1))$$

- Wilson's Theorem Extension:

  $B = b_1 b_2 \ldots b_m (\mod n) = \pm 1$, all $b_i \leq n$ such that $\gcd(b_i, n) = 1$.

  If $n \leq 4$ or $n = (\text{odd prime})^k$ or $n = 2(\text{odd prime})^k)$ then $B = -1$ for any k.

  Else $B = 1$.

- Stirling numbers of the second kind, denoted by $S(n,k)$ = number of ways to split n numbers into k non-empty sets.

$$S(n,1) = S(n,n) = 1$$
$$S(n,k) = kS(n-1,k) + S(n-1,k-1)$$

  $S(n-d+1, k-d+1) = S(n,k)$ where if indexes i, j belong to the same set, then $|i-j| \geq d$.

- Burnside's Lemma: $|\text{classes}| = \frac{1}{|G|} \sum_{\forall\ g \in G} (K^{C(g)})$, where:

  G = different permutations possible,

  $C(g)$ = number of cycles on the permutation g,

  and K = Number of states for each element

- Different ways to paint a necklace with N beads and K colors:

$$G = (1, 2, \ldots N), (2, 3, \ldots N, 1), \ldots (N, 1, \ldots N-1)$$

  $g_i = (i, i+1, \ldots i+N)$, (taking mod N to get it right) $i = 1 \ldots N$ with i per step, that is, $i \to 2i \to 3i \ldots$

  Cycles in g_i all have size $n / \gcd(i,n)$, so

$$C(g_i) = \gcd(i,n)$$
$$\text{ans} = \frac{1}{N} \sum_{i=1 \ldots N} (K^{\gcd(i,n)})$$
$$\text{ans} = \frac{1}{N} \sum_{\forall\ d|N} (\phi(\frac{N}{d}) K^d)$$

8.2 Combinatorial formulas

$$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$$
$$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$$
$$\sum_{k=0}^{n} k^4 = (6n^5 + 15n^4 + 10n^3 - n)/30$$
$$\sum_{k=0}^{n} k^5 = (2n^6 + 6n^5 + 5n^4 - n^2)/12$$
$$\sum_{k=0}^{n} x^k = (x^{n+1} - 1)/(x-1)$$
$$\sum_{k=0}^{n} kx^k = (x - (n+1)x^{n+1} + nx^{n+2})/(x-1)^2$$
$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$
$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$
$$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$$
$$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$$
$$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$$
$$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$$
$$\sum_{k=1}^{n} k\tbinom{n}{k} = n2^{n-1}$$
$$\sum_{k=1}^{n} k^2\tbinom{n}{k} = (n+n^2)2^{n-2}$$
$$\binom{m+n}{r} = \sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k}$$
$$\binom{n}{k} = \prod_{i=1}^{k} \frac{n-k+i}{i}$$

8.3 Number theory identities

Lucas' Theorem: For non-negative integers m and n and a prime p,

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod{p},$$

where

$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$$

is the base p representation of m, and similarly for n.

8.4 Stirling Numbers of the second kind

Number of ways to partition a set of n numbers into k non-empty subsets.

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{(k-j)} \binom{k}{j} j^n$$

Recurrence relation:

$$\left\{ {0 \atop 0} \right\} = 1$$

$$\left\{ {n \atop 0} \right\} = \left\{ {0 \atop n} \right\} = 1$$

$$\left\{ {n+1 \atop k} \right\} = k \left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$$

8.5 Burnside's Lemma

Let G be a finite group that acts on a set X. For each g in G let X^g denote the set of elements in X that are fixed by g, which means $X^g = \{x \in X | g(x) = x\}$. Burnside's lemma assers the following formula for the number of orbits, denoted $|X/G|$:

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

8.6 Numerical integration

RK4: to integrate $\dot{y} = f(t, y)$ with $y_0 = y(t_0)$, compute

$$\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1) \\
k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2) \\
k_4 &= f(t_n + h, y_n + h k_3) \\
y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$