

National Autonomous University of México
School of Engineering
Division of Electrical Engineering



ICPC 2026 Reference

CPCFI UNAM

>:)

RacsoFractal, zum, edwardsal17

Contents

1 Template	2	5.3.3 Bellman–Ford Algorithm	13
2 C++ Sintaxis	2	5.4 Minimum Spanning Tree (MST)	14
2.1 Compilation sentences	2	5.4.1 *Prim's Algorithm	14
2.2 Custom comparators	2	5.4.2 *Kruskal's Algorithm	14
2.3 *STL DS Usage	3	5.5 *Bipartite Checking	14
2.4 *Strings methods	3	5.6 *Negative Cycles	14
2.5 Pragmas	3	5.7 Topological Sort	14
2.6 Bit Manipulation Cheat Sheet	3	5.8 *Disjoint Set Union (DSU)	15
2.7 Comparing Floats	4	5.9 *Condensation Graph	15
2.8 Ceil	4	5.10 *Strongly Connected Components (SCC)	15
3 Miscellaneous	4	5.11 2-SAT	15
3.1 Binary Search	4	5.12 *Bridges and point articulation	18
3.1.1 *Parallel Binary Search	6	5.13 Flood Fill	18
3.1.2 *Ternary Search	6	5.14 Lava Flow (Multi-source BFS)	18
3.2 Kadane's Algorithm	6	5.15 MaxFlow	20
4 Queries	6	5.15.1 Dinic's Algorihtm	20
4.1 Prefix Sum 2D	6	5.15.2 *Ford-Fulkerson Algorithm	28
4.2 *Sparse Table	6	5.15.3 *Goldber-Tarjan Algoritm	28
4.3 *Sqrt Decomposition	6		
4.4 *Fenwick Tree	6		
4.5 *Fenwick Tree 2D	6		
4.6 Segment Tree	6		
4.6.1 *2D Segment Tree	8	6 Trees	28
4.6.2 *Persistent Segment Tree	8	6.1 Counting Childrens	28
4.6.3 *Lazy Propagation	8	6.2 *Tree Diameter	28
4.7 Ordered Set	8	6.3 *Centroid Decomposition	28
4.7.1 Multi-Ordered Set	9	6.4 *Euler Tour	28
4.8 *Treap	9	6.5 *Lowest Common Ancestor (LCA)	28
4.9 *Trie	9	6.6 *Heavy-Light Decomposition (HLD)	28
5 Graph Theory	9		
5.1 Breadth-First Search (BFS)	9	7 Strings	28
5.2 Deep-First Search (DFS)	11	7.1 Knuth-Morris-Pratt Algorithm (KMP)	28
5.3 Shortest Path	12	7.2 *Suffix Array	30
5.3.1 Dijkstra's Algorithm	12	7.3 *Rolling Hashing	30
5.3.2 *Floyd-Warshall's Algorithm	13	7.4 *Z Function	30
		7.5 *Aho-Corasick Algorithm	30
8 Dynamic Programming	30		
8.1 *Coins	30		
8.2 *Longest Increasing Subsequence (LIS)	30		
8.3 *Edit Distance	30		
8.4 *Knapsack	30		
8.5 *SOS DP	30		

8.6 *Digit DP	30
8.7 *Bitmask DP	30
9 Mathematics	30
9.1 Number Theory	30
9.1.1 Greatest Common Divisor (GCD)	30
9.1.2 Gauss Sum	30
9.1.3 *Modular Theory	30
9.1.4 *Modulo Inverse	30
9.1.5 *Fermat's Little Theorem	30
9.1.6 *Chinese Remainder Theorem	30
9.1.7 Binpow	30
9.1.8 Matrix Exponentiation (Linear Recurrency)	31
9.1.9 Prime checking	32
9.1.10 Prime factorization	32
9.1.11 Sieve of Eratosthenes	32
9.1.12 Sum of Divisors	33
9.2 Combinatorics	33
9.2.1 Binomial Coefficients	33
9.2.2 Common combinatorics formulas	34
9.3 *Stars and Bars	34
9.4 Probability	34
9.5 Computational Geometry	34
9.5.1 *Cross Product	34
9.5.2 *Convex Hull	34
9.6 *Fast Fourier Transform (FFT)	34
10 Appendix	34
10.1 What to do against WA?	34
10.2 Primitive sizes	35
10.3 Printable ASCII characters	35
10.4 Numbers bit representation	36
10.5 How a <code>vector<vector<pair<int,int>>></code> looks like	36
10.6 How all neighbours of a grid looks like	36

1 Template

```

1 // Racso programmed here
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 typedef long double ld;
6 typedef __int128 sll;
7 typedef pair<int,int> ii;
8 typedef pair<ll,ll> pll;
9 typedef vector<int> vi;
10 typedef vector<ll> vll;
11 typedef vector<ii> vii;
12 typedef vector<vi> vvi;
13 typedef vector<vii> vvii;
14 typedef vector<pll> vpll;
15 typedef unsigned int uint;
16 typedef unsigned long long ull;
17 #define fi first
18 #define se second
19 #define pb push_back
20 #define all(v) v.begin(),v.end()
21 #define rall(v) v.rbegin(),v.rend()
22 #define sz(a) (int)(a.size())
23 #define fori(i,a,n) for(int i = a; i < n; i++)
24 #define endl '\n'
25 const int MOD = 1e9+7;
26 const int INFTY = INT_MAX;

```

```

27 const long long LLINF = LLONG_MAX;
28 const double EPS = DBL_EPSILON;
29 void printVector( auto& v ){ fori(i,0,sz(v)) cout << v[i] << " "; cout << endl; }
30 void fastIO() { ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0); }

```

Listing 1: Racso's template.

2 C++ Sintaxis

2.1 Compilation sentences

To compile and execute in C++:

```

g++ -Wall -o solucion.exe solucion.cpp
g++ -std=c++20 -Wall -o main a.cpp ./solucion.exe < input.txt > output.txt

```

To compile and execute in Java:

```

javac -Xlint Solucion.java
java Solucion < input.txt > output.txt

```

To execute in Python (two options):

```

python3 solucion.py < input.txt > output.txt
pypy3 solucion.py < input.txt > output.txt

```

2.2 Custom comparators

```

1 // Using function
2 bool cmpFunction(const pair<int,int> &a, const pair<
3   int,int> &b) {
4   return a.second < b.second;
}

```

```

4 }
5 sort(all(v), cmpFunction);
6 // Using functor
7 struct CmpFunctor {
8     bool operator()(const pair<int,int> &a, const pair
9         <int,int> &b) const {
10        return a.second < b.second;
11    }
12 };
13 sort(all(v), CmpFunctor());
14 // Using lambda function
15 sort(all(v), [](const pair<int,int> &a, const pair<
16        int,int> &b) {
17        return a.second < b.second;
18 });

```

Listing 2: Template for custom sorting, using as example ordering a vii ascending by second element.

2.3 *STL DS Usage

2.4 *Strings methods

2.5 Pragmas

```

1 #pragma GCC optimize("O3")
2 #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
3 #pragma GCC optimize("unroll-loops")

```

Listing 3: Common pragmas.

2.6 Bit Manipulation Cheat Sheet

Bitwise operators:

- $\&$ (AND): Sets each bit to 1 if both bits are 1.
- $|$ (OR): Sets each bit to 1 if at least one bit is 1.
- \wedge (XOR): Sets each bit to 1 if bits are different.
- \sim (NOT): Inverts all bits.
- $<<$ (Left shift): Shifts bits left, fills with 0.
- $>>$ (Right shift): Shifts bits right.

Basic Bit Tasks:

- Get bit: $(n \& (1 << i)) != 0$
- Set bit: $n | (1 << i)$
- Clear bit: $n \& \sim(1 << i)$
- Toggle bit: $n \wedge (1 << i)$
- Clear LSB: $n \& (n - 1)$
- Get LSB: $n \& -n$

Set Operations:

- Subset check: $(A \& B) == B$
- Set union: $A | B$
- Set intersection: $A \& B$
- Set difference: $A \& \sim B$
- Toggle subset: $A \hat{B}$

Equations:

Properties of bitwise:

- $a | b = a \oplus b + a \& b$

- $a \oplus (a \& b) = (a \mid b) \oplus b$
- $b \oplus (a \& b) = (a \mid b) \oplus a$
- $(a \& b) \oplus (a \mid b) = a \oplus b$

In addition and subtraction:

- $a + b = (a \mid b) + (a \& b)$
- $a + b = a \oplus b + 2(a \& b)$
- $a - b = (a \oplus (a \& b)) - ((a \mid b) \oplus a)$
- $a - b = ((a \mid b) \oplus b) - ((a \mid b) \oplus a)$
- $a - b = (a \oplus (a \& b)) - (b \oplus (a \& b))$
- $a - b = ((a \mid b) \oplus b) - (b \oplus (a \& b))$

Gray code: $G = B \oplus (B \gg 1)$

C++ built in functions:

- `__builtin_popcount(x)` - Count the number of set bits in x.
- `__builtin_clz(x)` - Count the number of leading zeros in x.
- `__builtin_ctz(x)` - Count the number of trailing zeros in x.

2.7 Comparing Floats

```

1 long double a, b, EPS = 1e-9;
2 if( abs(a - b) < EPS ) {
3     // 'a' equals 'b'
4 }
```

Listing 4: Check if two real numbers are equal using an epsilon scope.

2.8 Ceil

$$\lceil \frac{a}{b} \rceil = \frac{a+b-1}{b}$$

(Originally I thought $\lceil \frac{a}{b} \rceil = \frac{a-1}{b} + 1$, but this calculates the wrong way in the cases where $a = 0$)

```

1 int myCeil(long long a, long long b) {
2     return (a + b - 1)/b;
3 }
```

Listing 5: A way to do ceil operation between integers without making explicit conversions. `a` and `b` are `int`, but the operation `a+b-1` can cause an overflow, so they must be casted into `long long` to avoid this. The result must be `int` anyway.

3 Miscellaneous

3.1 Binary Search

```

1 int binary_search( vector<int>& list, int n, int
2                     target ) {
3
4     int x0 = 0, x1 = n-1, mid;
5
6     while( x0 <= x1 ) {
7
8         mid = (x0 + x1) / 2; // (x1 - x0) / 2 + x0;
9
10        if( list[mid] == target )
11            return mid;
12
13        list[mid] < target ? x0 = mid + 1 : x1 = mid
14            - 1;
15    }
16
17    return -1;
18 }
```

Listing 6: Classic (Vanilla) implementation of Binary Search. Returns the index where `target` was found. Binary Search works on $O(\log_2(n))$, let n be the size of the container.

```

1 int binary_search( vector<int>& list, int n, int
2     target ) {
3
4     int ans = 0;
5
6     function<bool(int)> check = [&](int idx)->bool {
7
8         return idx < n && list[idx] <= target;
9     };
10
11    for( int i = 31; i >= 0; i-- ) {
12        if( check( ans + (1 << i) ) )
13            ans += 1 << i;
14    }
15
16    return list[ans] == target ? ans : -1;
17 }
```

Listing 7: Logarithmic jumps implementation of Binary Search. Returns the index where `target` was found. Binary Search works on $O(\log_2(n))$, let n be the size of the container.

```

1 int binary_search( vector<int>& list, int n, int
2     target ) {
3
4     int left = 1, right = n + 1, mid;
5
6     while(right - left >= 1) {
7
8         mid = left + (right - left) / 2;
9
10        if( list[mid] >= target && target > list[mid -
11]) {
```

```

6         //----- TODO logic here -----//
7         break;
8     }
9     else
10    if( list[mid] < target )
11        left = mid + 1;
12    else
13        right = mid;
14 }
15 }
```

Listing 8: Binary Search implementation for searching an element in the interval $(numbers_{i-1}, numbers_i]$. Originally used on Codeforces problem 474 - B (Worms). Returns the index where `target` was found. Binary Search works on $O(\log_2(n))$, let n be the size of the container.

```

1 function<bool(ll)> check = [&](ll t) -> bool {
2
3     ll products = 0;
4
5     for(ll machine : v) {
6
7         products += t / machine;
8
9         if( products >= target )
10            return true;
11     }
12
13     return products >= target;
14 }
15
16 for(int i = 0; i < 70; i++) {
17
18     mid = (x0 + x1) / 2;
19
20     check(mid) ? x1 = mid : x0 = mid + 1;
21 }
```

Listing 9: Implementation of Binary Search in the Answer. Originally used on CSES problem *Factory Machines*. Binary Search works on $O(\log_2(n))$, let n be the size of the container.

3.1.1 *Parallel Binary Search

3.1.2 *Ternary Search

3.2 Kadane's Algorithm

```

1 arns = v[0], maxSum = 0;
2 for(i,0,n)
{
    maxSum += v[i];
    arns = max(arns, maxSum);
    maxSum = max(0LL, maxSum);
}

```

Listing 10: Uses Kadane's Algorithm to find maximum subarray sum in $O(n)$.

4 Queries

4.1 Prefix Sum 2D

```

1 for(int i = 1; i <= n; i++)
2     for(int j = 1; j <= n; j++)
3     {
4         prefix[i][j] = prefix[i][j-1] + prefix[i-1][j] -
5             prefix[i-1][j-1];
6         prefix[i][j] += forest[i-1][j-1] == '*' ? 1 : 0;
7     }

```

```

6 }
7 for(int i = 0; i < q; i++)
8 {
9     pair<int,int> p1, p2;
10    cin >> p1.fi >> p1.se >> p2.fi >> p2.se;
11    int arns = prefix[p2.fi][p2.se];
12    arns -= prefix[p2.fi][p1.se-1] + prefix[p1.fi-1][
13        p2.se];
14    arns += prefix[p1.fi-1][p1.se-1];
15    cout << arns << endl;
}

```

Listing 11: Construction and querie of how many 1's are there in a matrix. Originally used on *Forest Queries* from CSES.

4.2 *Sparse Table

4.3 *Sqrt Decomposition

4.4 *Fenwick Tree

4.5 *Fenwick Tree 2D

4.6 Segment Tree

```

1 typedef long long ll;
2 typedef vector<ll> vll;
3 typedef vector<int> vi;
4 const int INF = INT_MAX;
5 class Segment_tree {
6     public: vll t;
7     Segment_tree( int n = 1e5+10 ) {

```

```

8     t.assign(n*4,INF);
9 }
10 void update(int node, int index, int tl, int tr,
11   int val) {
12   if( tr < index || tl > index ) return;
13   if( tr == tl ) t[node] = val;
14   else {
15     int mid = tl + ((tr-tl)>>1);
16     int lft = node << 1;
17     int rght = lft + 1;
18     update(lft,index,tl,mid,val);
19     update(rght,index,mid+1,tr,val);
20     t[node] = min(t[lft],t[rght]);
21   }
22 }
23 ll query(int node, int l, int r, int tl, int tr)
24 {
25   if( tl > r || tr < l ) return INF;
26   if( tl >= l and tr <= r ) return t[node];
27   else {
28     int mid = tl + ((tr-tl)>>1);
29     int lft = node << 1;
30     int rght = lft + 1;
31     ll q1 = query(lft,l,r,tl,mid);
32     ll q2 = query(rght,l,r,mid+1,tr);
33     return min(q1,q2);
34   }
35 }
36 void build(vi &v, int node, int tl, int tr) {
37   if( tl == tr ) t[node] = v[tl];
38   else {
39     int mid = tl + ((tr-tl)>>1);
40     int lft = node << 1;
41     int rght = lft + 1;
42     build(v,lft,tl,mid);
43     build(v,rght,mid+1,tr);
44     t[node] = min(t[lft],t[rght]);
45   }
46 }
47 Segment_tree st(n);
48 st.build(v,1,0,n-1);
49 st.update(1,a-1,0,n-1,b);
50 st.query(1,a-1,b-1,0,n-1));

```

Listing 12: Segment Tree for Dynamic Range **Minimum** Queries. Racso's Implementation.

```

1 vector<long long> v, sex;
2 int n;
3 void build(int node, int l, int r){
4   if(l == r) sex[node] = v[l];
5   else{
6     int mid = (l+r)/2;
7     build(2*node, l, mid);
8     build(2*node + 1, mid+1, r);
9     sex[node] = sex[2*node] + sex[2*node + 1];

```

```

10    }
11 }
12 void update(int node, int l, int r, int idx, int val
13 ) {
14     if(l == r) {
15         v[idx] = val;
16         sex[node] = val;
17     }
18     else{
19         int mid = (l+r)/2;
20         if(l <= idx && idx <= mid) update(2*node, l,
21         mid, idx, val);
22         else update(2*node +1, mid+1, r, idx, val);
23         sex[node] = sex[2*node] + sex[2*node + 1];
24     }
25 }
26 int query(int node, int tl, int tr, int l, int r){
27     if(r < tl || tr < l) return 0;
28     if(l <= tl && tr <= r) return sex[node];
29     int tm = (tl+tr)/2;
30     return query(2*node, tl, tm, l, r) + query(2*
31     node +1, tm+1, tr, l, r);
32 }
33 v.resize(n);
34 sex.resize(4 * n);
35 build(1, 0, n - 1);
36 query(1, 0, n-1, l - 1, r - 1)

```

Listing 13: Segment Tree for Dynamic Range **Sum** Queries. Zum's Implementation.

4.6.1 *2D Segment Tree

4.6.2 *Persistent Segment Tree

4.6.3 *Lazy Propagation

4.7 Ordered Set

```

1 //<-- Header.
2 #include <bits/stdc++.h>
3 #include <ext/pb_ds/assoc_container.hpp>
4 #include <ext/pb_ds/tree_policy.hpp>
5 using namespace std;
6 using namespace __gnu_pbds;
7 template<typename T, typename Cmp = less<T>>
8 using ordered_set = tree<T,null_type,Cmp,rb_tree_tag
9 ,tree_order_statistics_node_update>;
10 //<-- Declaration.
11 ordered_set<int> oset;
12 //<-- Methods usage.
13 // K-th element in a set (counting from zero).
14 ordered_set<int>::iterator it = oset.find_by_order(k
15 );
16 // Number of items strictly smaller than k.
17 ordered_set<int>::iterator it = oset.order_of_key(k
18 );
19 // Every other std::set method.

```

Listing 14: Ordered set necessary includes in header, declaration of the object, and usage of its new methods.

4.7.1 Multi-Ordered Set

```

1 //<-- Header.
2 #include <bits/stdc++.h>
3 #include <ext/pb_ds/assoc_container.hpp>
4 #include <ext/pb_ds/tree_policy.hpp>
5 using namespace std;
6 using namespace __gnu_pbds;
7 template<typename T, typename Cmp = less<T>>
8 using ordered_set = tree<T,null_type,Cmp,rb_tree_tag
    ,tree_order_statistics_node_update>;
9 //<-- Use in main.
10 ordered_set<pair<int,int>> multi_oiset;
11 map<int,int> cuenta;
12 function<void(int)> insertar = [&](int val) -> void
    {
13     multi_oiset.insert({val,++cuenta[val]});
14 };
15 function<void(int)> eliminar = [&](int val) -> void
    {
16     multi_oiset.erase({val,cuenta[val]--});
17 };

```

Listing 15: Declaration of multi-oiset structure.

4.8 *Treap

4.9 *Trie

5 Graph Theory

5.1 Breadth-First Search (BFS)

```

1 vector<vector<int>> graph;
2 vector<bool> visited;
3 graph.assign(n, vector<int>()); // <-- main
4 visited.assign(n, false); // <-- main
5
6 void bfs( int s ) {
7     queue<int> q;
8     q.push( s );
9     visited[ s ] = true;
10    while( ! q.empty() ) {
11        int u = q.front();
12        q.pop();
13        for( auto v : graph[ u ] ) {
14            if( ! visited[ u ] ) {
15                visited[ u ] = true;
16                q.push( v );
17                // --- ToDo logic here ---
18            }
19        }
20    }
21    return;
22 }

```

Listing 16: Iterative implementation of BFS graph traversal over a graph represented as an AdjacencyList on vector of vectors. BFS runs in $O(|V| + |E|)$.

```

1 int n, m;
2 string arns = "";
3 cin >> n >> m;
4 vector<vector<bool>> visited(n, vector<bool>(m, false));
5 vector<string> path(n, string(m, '0'));
6 vector<string> grid(n);
7 vii dirs = {{0,1},{0,-1},{1,0}, {-1,0}};
8 string commands = "LRUD";
9 queue<ii> q;
10 ii start, end, curr;
11 function<bool(int,int)> valid = [&](int i, int j) ->
12     bool {
13     return ( i >= 0 && i < n && j >= 0 && j < m &&
14     grid[i][j] != '#' && ! visited[i][j] );
15 };
16 fori(i,0,n) cin >> grid[i];
17 fori(i,0,n) {
18     fori(j,0,m)
19         if( grid[i][j] == 'A' ) {
20             visited[i][j] = true;
21             q.push( {i,j} );
22         }
23     }

```

```

22     while( ! q.empty() ) {
23         curr = q.front();
24         q.pop();
25         int i = curr.fi;
26         int j = curr.se;
27         if( grid[i][j] == 'B' ) {
28             end.fi = i;
29             end.se = j;
30             break;
31         }
32         int newI, newJ;
33         fori(I,0,4) {
34             newI = i + dirs[I].fi;
35             newJ = j + dirs[I].se;
36             if( valid(newI,newJ) ) {
37                 visited[newI][newJ] = true;
38                 q.push( {newI,newJ} );
39                 path[newI][newJ] = commands[I];
40             }
41         }
42     }
43     while( path[ end.fi ][ end.se ] != '0' ) {
44         fori(i,0,4) {
45             if( path[ end.fi ][ end.se ] == commands[i] )
46             {
47                 arns += i & 1 ? commands[i-1] : commands[i+1];
48                 end.fi -= dirs[i].fi;
49             }
50         }
51     }
52 }

```

```

48         end.se == dirs[i].se;
49     }
50 }
51 }
52 reverse(all(arnts));
53 if( arnts == "" ) cout << "NO" << endl;
54 else cout << "YES" << endl << arnts.size() << endl <<
    arnts << endl;

```

Listing 17: BFS on Grid to find shortest path from an starting point *A* to an end *B*. Once the path is found, it reconstruct it with movements *LRUD*. Works in $O(n \cdot m)$. Originally used on problem *Labyrinth* from CSES.

5.2 Deep-First Search (DFS)

```

1 vector<vector<int>> graph;
2 vector<bool> visited;
3 graph.assign(n, vector<int>()); // <--- main
4 visited.assign(n, false); // <--- main
5
6 void dfs( int s ) {
7     if( visited[s] == true ) return;
8     visited[s] = true;
9     vector<int>::iterator i;
10    for( i = graph[s].begin(); i < graph[s].end();
11        ++i ) {
12        if( ! visited[*i] ) {
13            // --- ToDo logic here ---
14            dfs(*i);
15        }
16    }
17}

```

```

14     }
15 }
16 }

```

Listing 18: Recursive implementation of DFS graph traversal over a graph represented as a AdjacencyList on vector of vectors. DFS runs in $O(|V| + |E|)$.

```

1 vector<vector<int>> graph;
2 vector<bool> visited;
3 void dfs( int s ) {
4     stack<int> stk;
5     stk.push(s);
6     while ( !stk.empty() ) {
7         int u = stk.top();
8         stk.pop();
9         if ( visited[u] ) continue;
10        visited[u] = true;
11        // --- ToDo logic here ---
12        for( auto it = graph[u].rbegin(); it != graph
13            [u].rend(); ++it)
14            if ( !visited[*it])
15                stk.push(*it);
16    }
17}

```

Listing 19: Iterative implementation of DFS graph traversal over a graph represented as a AdjacencyList on vector of vectors. DFS runs in $O(|V| + |E|)$.

```

1 typedef long long ll;
2 vector<vector<ll>> adj;

```

```
3 vector<bool> visited;
4 function<void(int)> dfs = [&](int u) -> void {
5     if( visited[u] ) return;
6     visited[u] = true;
7     for( int v : adj[u] )
8         dfs(v);
9 };
10 dfs(n);
```

Listing 20: DFS implementation with a lambda function (adjacency list and visited don't need to be passed thorough argument). DFS runs in $O(|V| + |E|)$.

```
1  typedef long long ll;
2  typedef vector<ll> vll;
3  map<ll,vll> adj;
4  set<ll> visited;
5  function<void(ll)> dfs = [&](ll u) -> void {
6      if( visited.count(u) ) return;
7      visited.insert(u);
8      for( ll v : adj[u] )
9          dfs(v);
10 };
11 dfs(n);
```

Listing 21: DFS implementation with a lambda function implemented with a map instead of vector of vectors, and a set to track visited nodes. DFS runs in $O(|V| + |E|)$.

5.3 Shortest Path

5.3.1 Dijkstra's Algorithm

```
1  typedef long long ll;
2  typedef pair<ll, ll> pll;
3
4  vector<vector<ll>> graph;
5  vector<ll> visited;
6  graph.assign(n, vector<ll>() );
7  visited.assign(n, false);
8
9  vector<ll> dijkstra( int n, int source, vector<
10   vector<pll>> &graph ) {
11    vector<ll> dist( n, INFTY );
12    priority_queue<pll, vector<pll>, greater<pll>> pq;
13    dist[ source ] = 0;
14    pq.push( {0, source} );
15    while( ! pq.empty() ) {
16      ll d = pq.top().first;
17      ll u = pq.top().second;
18      pq.pop();
19      if( d > dist[ u ] ) continue;
20      for( auto &edge : graph[ u ] ) {
21        ll v = edge.first;
22        ll weight = edge.second;
23        if( dist[ u ] + weight < dist[ v ] ) {
24          dist[ v ] = dist[ u ] + weight;
25          pq.push( {dist[ v ], v} );
26        }
27      }
28    }
29  }
```

```

25     }
26 }
27
28 return dist;
29 }
```

Listing 22: Iterative implementation of Dijkstra's Algorithm for shortest path over a graph represented as a AdjacencyList on vector of vectors. Returns a vector with the shortest path to every other vertex in the graph. $O(|E| \times \log_2(|V|))$. doesn't work with negative weights.

```

1 vvp1l graph(n+1,vpll());
2 vector<bool> visited(n+1, false);
3 function<vll(int)> dijkstra = [&](int source) -> vll
4 {
5     vll dist(n+1, INF);
6     priority_queue<pll,vpll,greater<pll>> pq;
7     dist[source] = 0;
8     pq.push({0,source});
9     while( ! pq.empty() ) {
10         ll d = pq.top().fi;
11         ll u = pq.top().se;
12         pq.pop();
13         if( d > dist[u] ) continue;
14         for(pll edge : graph[u]) {
15             ll v = edge.fi;
16             ll w = edge.se;
17             if( dist[u] + w < dist[v] ) {
18                 dist[v] = dist[u] + w;
19                 pq.push({dist[v],v});
20             }
21         }
22     }
23     return dist;
24 };
```

```

18         pq.push({dist[v],v});
19     }
20 }
21
22 return dist;
23 };
```

Listing 23: Iterative implementation of Dijkstra's Algorithm as a Lambda Function for shortest path over a graph represented as a AdjacencyList on vector of vectors. Returns a vector with the shortest path to every other vertex in the graph. $O(|E| \times \log_2(|V|))$. Doesn't work with negative weights.

5.3.2 *Floyd-Warshall's Algorithm

5.3.3 Bellman–Ford Algorithm

```

1 int V, E;
2 cin >> V >> E;
3 vvi edges(E,vi(3,0));
4
5 for(int i = 1; i <= E; i++)
6     cin >> edges[i][0] >> edges[i][1] >> edges[i][2];
7
8 function<vi(int)> bellman_ford = [&](int src) -> vi
9 {
10     vi dist(V,INF);
11     dist[src] = 0;
12     for(int i = 0; i < V; i++)
13     {
14         for( vi edge : edges )
15             if( dist[edge[0]] + edge[2] < dist[edge[1]] )
16                 dist[edge[1]] = dist[edge[0]] + edge[2];
17     }
18 }
```

```

15     {
16         int u = edge[0];
17         int v = edge[1];
18         int w = edge[2];
19         if( dist[u] != INF and dist[u] + w < dist[v]
20     )
21     {
22         if( i == V-1 )
23             return {-1};
24         dist[v] = dist[u] + w;
25     }
26 }
27 return dist;
28 };

```

Listing 24: Finds the shortest route from a source vertex, to every other one in the graph. Works over a list of edges. Runs in $O(|V| \times |E|)$. Can be used to find negative cycles.

```

2 vvi graph(V+1,vi());
3 vector<bool> visited(V+1,false);
4 function<void(int)> dfs = [&](int u) -> void {
5     visited[u] = true;
6     for(int v : graph[u])
7         if( ! visited[v] )
8             dfs(v);
9     topo.pb(u);
10 }
11 function<void()> topological_sort = [&]() -> void {
12     for(int i = 1; i <= V; i++)
13         if( ! visited[i] )
14             dfs(i);
15     reverse(all(topo));
16 }
17 topological_sort();
18 for(int i = 0; i < V; i++) cout << topo[i] << " ";

```

Listing 25: Recursive toposort implementation for unweighted DAG through vvi with DFS with inverted postorder. Runs in $O(|V| \times |E|)$.

5.4 Minimum Spanning Tree (MST)

- 5.4.1 *Prim's Algorithm
- 5.4.2 *Kruskal's Algorithm

5.5 *Bipartite Checking

5.6 *Negative Cycles

5.7 Topological Sort

```
1 vi topo;
```

```

1 vi indegree(V+1,0);
2 vvi graph(V+1,vi());
3 vector<bool> visited(V+1,false);
4 fori(i,0,E) {
5     graph[u].pb(v);
6     indegree[v]++;
7 }
8 function<vi()> topological_sort = [&]() -> vi {

```

```

9  vi order, deg = indegree; // copy
10 queue<int> q;
11 for(int i = 1; i <= V; i++) {
12     if( deg[i] == 0 )
13         q.push(i);
14     while( ! q.empty() ) {
15         int u = q.front(); q.pop();
16         order.pb(u);
17         for(int v : graph[u]) {
18             deg[v]--;
19             if(deg[v] == 0)
20                 q.push(v);
21         }
22     }
23     return order;
24 };
25 vi topo = topological_sort();
26 if( (int)(topo.size()) != V ) cout << "IMPOSSIBLE"
    << endl;

```

Listing 26: Kahn's Algorithm for Topological Sorting using BFS and indegree vertex analysis (nodes in a cycle will never have indegree zero). Works over unweighted directed graphs containing cycles through vvi. Runs in $O(|V| \times |E|)$.

5.8 *Disjoint Set Union (DSU)

5.9 *Condensation Graph

5.10 *Strongly Connected Components (SCC)

5.11 2-SAT

```

1 struct TwoSatSolver {
2     int n_vars;
3     int n_vertices;
4     vector<vector<int>> adj, adj_t;
5     vector<bool> used;
6     vector<int> order, comp;
7     vector<bool> assignment;
8
9     TwoSatSolver(int _n_vars) : n_vars(_n_vars),
10      n_vertices(2 * n_vars), adj(n_vertices), adj_t(
11      n_vertices), used(n_vertices), order(), comp(
12      n_vertices, -1), assignment(n_vars) {
13         order.reserve(n_vertices);
14     }
15     void dfs1(int v) {
16         used[v] = true;
17         for (int u : adj[v]) {
18             if (!used[u])
19                 dfs1(u);
20         }
21         order.push_back(v);
22     }
23     void dfs2(int v, int cl) {
24         comp[v] = cl;
25         for (int u : adj_t[v]) {
26             if (comp[u] == -1)
27                 dfs2(u, cl);
28         }
29     }
30
31     void solve() {
32         for (int v : n_vertices) {
33             if (!used[v])
34                 dfs1(v);
35         }
36         for (int v : n_vertices) {
37             if (comp[v] == -1)
38                 dfs2(v, 0);
39         }
40         for (int v : n_vertices) {
41             if (assignment[v] == false)
42                 assignment[v] = true;
43             else
44                 assignment[v] = false;
45         }
46     }
47
48     int get_assignment(int v) {
49         return assignment[v];
50     }
51
52     int get_comp(int v) {
53         return comp[v];
54     }
55
56     void print_order() {
57         for (int v : order) {
58             cout << v << " ";
59         }
60         cout << endl;
61     }
62
63     void print_assignment() {
64         for (int v : n_vertices) {
65             cout << v << ": " << assignment[v] << endl;
66         }
67     }
68
69     void print_comp() {
70         for (int v : n_vertices) {
71             cout << v << ": " << comp[v] << endl;
72         }
73     }
74
75     void print_all() {
76         print_order();
77         cout << endl;
78         print_assignment();
79         cout << endl;
80         print_comp();
81     }
82
83     void print_all() {
84         print_order();
85         cout << endl;
86         print_assignment();
87         cout << endl;
88         print_comp();
89     }
90
91     void print_all() {
92         print_order();
93         cout << endl;
94         print_assignment();
95         cout << endl;
96         print_comp();
97     }
98
99     void print_all() {
100        print_order();
101        cout << endl;
102        print_assignment();
103        cout << endl;
104        print_comp();
105    }
106
107    void print_all() {
108        print_order();
109        cout << endl;
110        print_assignment();
111        cout << endl;
112        print_comp();
113    }
114
115    void print_all() {
116        print_order();
117        cout << endl;
118        print_assignment();
119        cout << endl;
120        print_comp();
121    }
122
123    void print_all() {
124        print_order();
125        cout << endl;
126        print_assignment();
127        cout << endl;
128        print_comp();
129    }
130
131    void print_all() {
132        print_order();
133        cout << endl;
134        print_assignment();
135        cout << endl;
136        print_comp();
137    }
138
139    void print_all() {
140        print_order();
141        cout << endl;
142        print_assignment();
143        cout << endl;
144        print_comp();
145    }
146
147    void print_all() {
148        print_order();
149        cout << endl;
150        print_assignment();
151        cout << endl;
152        print_comp();
153    }
154
155    void print_all() {
156        print_order();
157        cout << endl;
158        print_assignment();
159        cout << endl;
160        print_comp();
161    }
162
163    void print_all() {
164        print_order();
165        cout << endl;
166        print_assignment();
167        cout << endl;
168        print_comp();
169    }
170
171    void print_all() {
172        print_order();
173        cout << endl;
174        print_assignment();
175        cout << endl;
176        print_comp();
177    }
178
179    void print_all() {
180        print_order();
181        cout << endl;
182        print_assignment();
183        cout << endl;
184        print_comp();
185    }
186
187    void print_all() {
188        print_order();
189        cout << endl;
190        print_assignment();
191        cout << endl;
192        print_comp();
193    }
194
195    void print_all() {
196        print_order();
197        cout << endl;
198        print_assignment();
199        cout << endl;
200        print_comp();
201    }
202
203    void print_all() {
204        print_order();
205        cout << endl;
206        print_assignment();
207        cout << endl;
208        print_comp();
209    }
210
211    void print_all() {
212        print_order();
213        cout << endl;
214        print_assignment();
215        cout << endl;
216        print_comp();
217    }
218
219    void print_all() {
220        print_order();
221        cout << endl;
222        print_assignment();
223        cout << endl;
224        print_comp();
225    }
226
227    void print_all() {
228        print_order();
229        cout << endl;
230        print_assignment();
231        cout << endl;
232        print_comp();
233    }
234
235    void print_all() {
236        print_order();
237        cout << endl;
238        print_assignment();
239        cout << endl;
240        print_comp();
241    }
242
243    void print_all() {
244        print_order();
245        cout << endl;
246        print_assignment();
247        cout << endl;
248        print_comp();
249    }
250
251    void print_all() {
252        print_order();
253        cout << endl;
254        print_assignment();
255        cout << endl;
256        print_comp();
257    }
258
259    void print_all() {
260        print_order();
261        cout << endl;
262        print_assignment();
263        cout << endl;
264        print_comp();
265    }
266
267    void print_all() {
268        print_order();
269        cout << endl;
270        print_assignment();
271        cout << endl;
272        print_comp();
273    }
274
275    void print_all() {
276        print_order();
277        cout << endl;
278        print_assignment();
279        cout << endl;
280        print_comp();
281    }
282
283    void print_all() {
284        print_order();
285        cout << endl;
286        print_assignment();
287        cout << endl;
288        print_comp();
289    }
290
291    void print_all() {
292        print_order();
293        cout << endl;
294        print_assignment();
295        cout << endl;
296        print_comp();
297    }
298
299    void print_all() {
300        print_order();
301        cout << endl;
302        print_assignment();
303        cout << endl;
304        print_comp();
305    }
306
307    void print_all() {
308        print_order();
309        cout << endl;
310        print_assignment();
311        cout << endl;
312        print_comp();
313    }
314
315    void print_all() {
316        print_order();
317        cout << endl;
318        print_assignment();
319        cout << endl;
320        print_comp();
321    }
322
323    void print_all() {
324        print_order();
325        cout << endl;
326        print_assignment();
327        cout << endl;
328        print_comp();
329    }
330
331    void print_all() {
332        print_order();
333        cout << endl;
334        print_assignment();
335        cout << endl;
336        print_comp();
337    }
338
339    void print_all() {
340        print_order();
341        cout << endl;
342        print_assignment();
343        cout << endl;
344        print_comp();
345    }
346
347    void print_all() {
348        print_order();
349        cout << endl;
350        print_assignment();
351        cout << endl;
352        print_comp();
353    }
354
355    void print_all() {
356        print_order();
357        cout << endl;
358        print_assignment();
359        cout << endl;
360        print_comp();
361    }
362
363    void print_all() {
364        print_order();
365        cout << endl;
366        print_assignment();
367        cout << endl;
368        print_comp();
369    }
370
371    void print_all() {
372        print_order();
373        cout << endl;
374        print_assignment();
375        cout << endl;
376        print_comp();
377    }
378
379    void print_all() {
380        print_order();
381        cout << endl;
382        print_assignment();
383        cout << endl;
384        print_comp();
385    }
386
387    void print_all() {
388        print_order();
389        cout << endl;
390        print_assignment();
391        cout << endl;
392        print_comp();
393    }
394
395    void print_all() {
396        print_order();
397        cout << endl;
398        print_assignment();
399        cout << endl;
400        print_comp();
401    }
402
403    void print_all() {
404        print_order();
405        cout << endl;
406        print_assignment();
407        cout << endl;
408        print_comp();
409    }
410
411    void print_all() {
412        print_order();
413        cout << endl;
414        print_assignment();
415        cout << endl;
416        print_comp();
417    }
418
419    void print_all() {
420        print_order();
421        cout << endl;
422        print_assignment();
423        cout << endl;
424        print_comp();
425    }
426
427    void print_all() {
428        print_order();
429        cout << endl;
430        print_assignment();
431        cout << endl;
432        print_comp();
433    }
434
435    void print_all() {
436        print_order();
437        cout << endl;
438        print_assignment();
439        cout << endl;
440        print_comp();
441    }
442
443    void print_all() {
444        print_order();
445        cout << endl;
446        print_assignment();
447        cout << endl;
448        print_comp();
449    }
450
451    void print_all() {
452        print_order();
453        cout << endl;
454        print_assignment();
455        cout << endl;
456        print_comp();
457    }
458
459    void print_all() {
460        print_order();
461        cout << endl;
462        print_assignment();
463        cout << endl;
464        print_comp();
465    }
466
467    void print_all() {
468        print_order();
469        cout << endl;
470        print_assignment();
471        cout << endl;
472        print_comp();
473    }
474
475    void print_all() {
476        print_order();
477        cout << endl;
478        print_assignment();
479        cout << endl;
480        print_comp();
481    }
482
483    void print_all() {
484        print_order();
485        cout << endl;
486        print_assignment();
487        cout << endl;
488        print_comp();
489    }
490
491    void print_all() {
492        print_order();
493        cout << endl;
494        print_assignment();
495        cout << endl;
496        print_comp();
497    }
498
499    void print_all() {
500        print_order();
501        cout << endl;
502        print_assignment();
503        cout << endl;
504        print_comp();
505    }
506
507    void print_all() {
508        print_order();
509        cout << endl;
510        print_assignment();
511        cout << endl;
512        print_comp();
513    }
514
515    void print_all() {
516        print_order();
517        cout << endl;
518        print_assignment();
519        cout << endl;
520        print_comp();
521    }
522
523    void print_all() {
524        print_order();
525        cout << endl;
526        print_assignment();
527        cout << endl;
528        print_comp();
529    }
530
531    void print_all() {
532        print_order();
533        cout << endl;
534        print_assignment();
535        cout << endl;
536        print_comp();
537    }
538
539    void print_all() {
540        print_order();
541        cout << endl;
542        print_assignment();
543        cout << endl;
544        print_comp();
545    }
546
547    void print_all() {
548        print_order();
549        cout << endl;
550        print_assignment();
551        cout << endl;
552        print_comp();
553    }
554
555    void print_all() {
556        print_order();
557        cout << endl;
558        print_assignment();
559        cout << endl;
560        print_comp();
561    }
562
563    void print_all() {
564        print_order();
565        cout << endl;
566        print_assignment();
567        cout << endl;
568        print_comp();
569    }
570
571    void print_all() {
572        print_order();
573        cout << endl;
574        print_assignment();
575        cout << endl;
576        print_comp();
577    }
578
579    void print_all() {
580        print_order();
581        cout << endl;
582        print_assignment();
583        cout << endl;
584        print_comp();
585    }
586
587    void print_all() {
588        print_order();
589        cout << endl;
590        print_assignment();
591        cout << endl;
592        print_comp();
593    }
594
595    void print_all() {
596        print_order();
597        cout << endl;
598        print_assignment();
599        cout << endl;
600        print_comp();
601    }
602
603    void print_all() {
604        print_order();
605        cout << endl;
606        print_assignment();
607        cout << endl;
608        print_comp();
609    }
610
611    void print_all() {
612        print_order();
613        cout << endl;
614        print_assignment();
615        cout << endl;
616        print_comp();
617    }
618
619    void print_all() {
620        print_order();
621        cout << endl;
622        print_assignment();
623        cout << endl;
624        print_comp();
625    }
626
627    void print_all() {
628        print_order();
629        cout << endl;
630        print_assignment();
631        cout << endl;
632        print_comp();
633    }
634
635    void print_all() {
636        print_order();
637        cout << endl;
638        print_assignment();
639        cout << endl;
640        print_comp();
641    }
642
643    void print_all() {
644        print_order();
645        cout << endl;
646        print_assignment();
647        cout << endl;
648        print_comp();
649    }
650
651    void print_all() {
652        print_order();
653        cout << endl;
654        print_assignment();
655        cout << endl;
656        print_comp();
657    }
658
659    void print_all() {
660        print_order();
661        cout << endl;
662        print_assignment();
663        cout << endl;
664        print_comp();
665    }
666
667    void print_all() {
668        print_order();
669        cout << endl;
670        print_assignment();
671        cout << endl;
672        print_comp();
673    }
674
675    void print_all() {
676        print_order();
677        cout << endl;
678        print_assignment();
679        cout << endl;
680        print_comp();
681    }
682
683    void print_all() {
684        print_order();
685        cout << endl;
686        print_assignment();
687        cout << endl;
688        print_comp();
689    }
690
691    void print_all() {
692        print_order();
693        cout << endl;
694        print_assignment();
695        cout << endl;
696        print_comp();
697    }
698
699    void print_all() {
700        print_order();
701        cout << endl;
702        print_assignment();
703        cout << endl;
704        print_comp();
705    }
706
707    void print_all() {
708        print_order();
709        cout << endl;
710        print_assignment();
711        cout << endl;
712        print_comp();
713    }
714
715    void print_all() {
716        print_order();
717        cout << endl;
718        print_assignment();
719        cout << endl;
720        print_comp();
721    }
722
723    void print_all() {
724        print_order();
725        cout << endl;
726        print_assignment();
727        cout << endl;
728        print_comp();
729    }
730
731    void print_all() {
732        print_order();
733        cout << endl;
734        print_assignment();
735        cout << endl;
736        print_comp();
737    }
738
739    void print_all() {
740        print_order();
741        cout << endl;
742        print_assignment();
743        cout << endl;
744        print_comp();
745    }
746
747    void print_all() {
748        print_order();
749        cout << endl;
750        print_assignment();
751        cout << endl;
752        print_comp();
753    }
754
755    void print_all() {
756        print_order();
757        cout << endl;
758        print_assignment();
759        cout << endl;
760        print_comp();
761    }
762
763    void print_all() {
764        print_order();
765        cout << endl;
766        print_assignment();
767        cout << endl;
768        print_comp();
769    }
770
771    void print_all() {
772        print_order();
773        cout << endl;
774        print_assignment();
775        cout << endl;
776        print_comp();
777    }
778
779    void print_all() {
780        print_order();
781        cout << endl;
782        print_assignment();
783        cout << endl;
784        print_comp();
785    }
786
787    void print_all() {
788        print_order();
789        cout << endl;
790        print_assignment();
791        cout << endl;
792        print_comp();
793    }
794
795    void print_all() {
796        print_order();
797        cout << endl;
798        print_assignment();
799        cout << endl;
800        print_comp();
801    }
802
803    void print_all() {
804        print_order();
805        cout << endl;
806        print_assignment();
807        cout << endl;
808        print_comp();
809    }
810
811    void print_all() {
812        print_order();
813        cout << endl;
814        print_assignment();
815        cout << endl;
816        print_comp();
817    }
818
819    void print_all() {
820        print_order();
821        cout << endl;
822        print_assignment();
823        cout << endl;
824        print_comp();
825    }
826
827    void print_all() {
828        print_order();
829        cout << endl;
830        print_assignment();
831        cout << endl;
832        print_comp();
833    }
834
835    void print_all() {
836        print_order();
837        cout << endl;
838        print_assignment();
839        cout << endl;
840        print_comp();
841    }
842
843    void print_all() {
844        print_order();
845        cout << endl;
846        print_assignment();
847        cout << endl;
848        print_comp();
849    }
850
851    void print_all() {
852        print_order();
853        cout << endl;
854        print_assignment();
855        cout << endl;
856        print_comp();
857    }
858
859    void print_all() {
860        print_order();
861        cout << endl;
862        print_assignment();
863        cout << endl;
864        print_comp();
865    }
866
867    void print_all() {
868        print_order();
869        cout << endl;
870        print_assignment();
871        cout << endl;
872        print_comp();
873    }
874
875    void print_all() {
876        print_order();
877        cout << endl;
878        print_assignment();
879        cout << endl;
880        print_comp();
881    }
882
883    void print_all() {
884        print_order();
885        cout << endl;
886        print_assignment();
887        cout << endl;
888        print_comp();
889    }
890
891    void print_all() {
892        print_order();
893        cout << endl;
894        print_assignment();
895        cout << endl;
896        print_comp();
897    }
898
899    void print_all() {
900        print_order();
901        cout << endl;
902        print_assignment();
903        cout << endl;
904        print_comp();
905    }
906
907    void print_all() {
908        print_order();
909        cout << endl;
910        print_assignment();
911        cout << endl;
912        print_comp();
913    }
914
915    void print_all() {
916        print_order();
917        cout << endl;
918        print_assignment();
919        cout << endl;
920        print_comp();
921    }
922
923    void print_all() {
924        print_order();
925        cout << endl;
926        print_assignment();
927        cout << endl;
928        print_comp();
929    }
930
931    void print_all() {
932        print_order();
933        cout << endl;
934        print_assignment();
935        cout << endl;
936        print_comp();
937    }
938
939    void print_all() {
940        print_order();
941        cout << endl;
942        print_assignment();
943        cout << endl;
944        print_comp();
945    }
946
947    void print_all() {
948        print_order();
949        cout << endl;
950        print_assignment();
951        cout << endl;
952        print_comp();
953    }
954
955    void print_all() {
956        print_order();
957        cout << endl;
958        print_assignment();
959        cout << endl;
960        print_comp();
961    }
962
963    void print_all() {
964        print_order();
965        cout << endl;
966        print_assignment();
967        cout << endl;
968        print_comp();
969    }
970
971    void print_all() {
972        print_order();
973        cout << endl;
974        print_assignment();
975        cout << endl;
976        print_comp();
977    }
978
979    void print_all() {
980        print_order();
981        cout << endl;
982        print_assignment();
983        cout << endl;
984        print_comp();
985    }
986
987    void print_all() {
988        print_order();
989        cout << endl;
990        print_assignment();
991        cout << endl;
992        print_comp();
993    }
994
995    void print_all() {
996        print_order();
997        cout << endl;
998        print_assignment();
999        cout << endl;
1000       print_comp();
1001   }

```

```
25         dfs2(u, cl);
26     }
27 }
28
29 bool solve_2SAT() {
30     order.clear();
31     used.assign(n_vertices, false);
32     for (int i = 0; i < n_vertices; ++i) {
33         if (!used[i])
34             dfs1(i);
35     }
36
37     comp.assign(n_vertices, -1);
38     for (int i = 0, j = 0; i < n_vertices; ++i)
39     {
40         int v = order[n_vertices - i - 1];
41         if (comp[v] == -1)
42             dfs2(v, j++);
43     }
44
45     assignment.assign(n_vars, false);
46     for (int i = 0; i < n_vertices; i += 2) {
47         if (comp[i] == comp[i + 1])
48             return false;
49         assignment[i / 2] = comp[i] > comp[i + 1];
50     }
51     return true;
52 }
53
54 void add_disjunction(int a, bool na, int b, bool nb) {
55     // na and nb signify whether a and b are to
56     // be negated
57     a = 2 * a ^ na;
58     b = 2 * b ^ nb;
59     int neg_a = a ^ 1;
60     int neg_b = b ^ 1;
61     adj[neg_a].push_back(b);
62     adj[neg_b].push_back(a);
63     adj_t[b].push_back(neg_a);
64     adj_t[a].push_back(neg_b);
65 }
66
67 static void example_usage() {
68     TwoSatSolver solver(3); // a, b, c
69     solver.add_disjunction(0, false, 1, true);
70     //    a v not b
71     solver.add_disjunction(0, true, 1, true);
72     // not a v not b
73     solver.add_disjunction(1, false, 2, false);
74     //    b v c
75     solver.add_disjunction(0, false, 0, false);
76     //    a v a
77     assert(solver.solve_2SAT() == true);
78 }
```

```
72         auto expected = vector<bool>(True, False);
73         True);
74         assert(solver.assignment == expected);
75     }
```

Listing 27: 2-SAT implementation from cp-algorithms.com. Each component added is a expression of the form $a \vee b$, which is equivalent to $\neg a \Rightarrow b \wedge \neg b \Rightarrow a$ (if one of the variables is false, then the other one must be true). A directed graph is constructed based on these implication: For each x , there's two vertices v_x and $v_{\neg x}$. If there is an edge $a \Rightarrow b$, then there also is an edge $\neg b \Rightarrow \neg a$. For any x , if x is reachable from $\neg x$ and $\neg x$ is reachable from x , the problem has no solution. This means, each variable must be in a different SCC than their negative. This is verified by the method `solve_2SAT()`, which returns a boolean: `True` if it has a solution and `False` if it doesn't.

Giant Pizza How does a particular 2-SAT problem look like? Following is the statement for the problem CSES 1684 (Giant Pizza):

Uolevi's family is going to order a large pizza and eat it together. A total of n family members will join the order, and there are m possible toppings. The pizza may have any number of toppings. Each family member gives two wishes concerning the toppings of the pizza. The wishes are of the form "topping x is good/bad". Your task is to choose the toppings so that at least one wish from everybody becomes true (a good topping is included in the pizza or a bad topping is not included).

Input

The first input line has two integers n and m : the number of family members and toppings. The toppings are numbered $1, 2, \dots, m$. After this, there are n lines describing the wishes. Each line has two wishes of the form " $+x$ " (topping x is good) or " $-x$ " (topping x is bad).

Output

Print a line with m symbols: for each topping "+" if it is included and "-" if it is not included. You can print any valid solution. If there are no valid solutions, print "IMPOSSIBLE".

```
1 int main(){
2     fastIO();
3     int n = nxt(), m = nxt();
4     TwoSatSolver TwoSat(m);
5
6     for(i, 0, n){
7         char type1, type2;
8         int top1, top2;
9         cin >> type1 >> top1 >> type2 >> top2;
10
11        top1--; top2--;
12        TwoSat.add_disjunction(top1, type1 == '—',
13 top2, type2 == '—');
14    }
15
16    if(TwoSat.solve_2SAT()){
17        for(i, 0, m){
18            if(TwoSat.assignment[i])
19                cout << "+ ";
20            else
21                cout << "- ";
22        }
23    }
24    else cout << "IMPOSSIBLE";
25 }
```

```

24     return 0;
25 }
```

Listing 28: Main method for solving CSES 1684 Giant Pizza using 2-SAT template.

5.12 *Bridges and point articulation

5.13 Flood Fill

```

1 vector<string> grid(n);
2 vii dirs = {{0,1},{0,-1},{1,0}, {-1,0}};
3 ii start;
4 int arns = 0;
5 function<void(int,int)> traverse = [&](int i, int j)
6     -> void {
7     if( grid[i][j] == '#' ) return;
8     int newI, newJ;
9     if( grid[i][j] != '.' ) arns += grid[i][j] - '0';
10    grid[i][j] = '#';
11    for( ii move : dirs ) {
12        newI = i + move.fi; newJ = j + move.se;
13        if( newI >= 0 && newI < n && newJ >= 0 && newJ <
14            m && grid[newI][newJ] == 'T' )
15            return;
16    for( ii move : dirs ) {
17        newI = i + move.fi; newJ = j + move.se;
18        if( newI >= 0 && newI < n && newJ >= 0 && newJ <
19            m )
20            traverse(newI, newJ);
```

```

19    }
20 };
21 fori(i,0,n)
22     cin >> grid[i];
23 fori(i,0,n) {
24     fori(j,0,m) {
25         if( grid[i][j] == 'S' ) {
26             grid[i][j] = '.';
27             start.fi = i;
28             start.se = j;
29         }
30     }
31 }
32 traverse(start.fi, start.se);
33 cout << arns << endl;
```

Listing 29: Traverse a matrix of 'n' x 'm' on grid representation. The matrix is composed of '.' for a valid space (empty), '#' for a wall, 'T' for a trap, and a number for a treasure. This implementation takes the sum of every treasure in the maze. The condition for moving to the next location is that there are no Traps nearby (up, down, left, right), so the player will never be killed while traversing. It also implements a way to read numerous test cases, but without knowing beforehand how many there are. Runs in $O(n \cdot m)$. Originally used on the problem *Treasures* from 2024-2025 ICPC Bolivia Pre-National Contest.

5.14 Lava Flow (Multi-source BFS)

```

1 typedef array<int,3> iii;
2
```

```
3 vii dirs = {{1,0},{0,1}, {-1,0}, {0,-1}};
4 map<int, string> path = {{0, "D"}, {1, "R"}, {2, "U"}, {3, "L"}};
5 int n, m;
6 string arns = "";
7 bool escaped = false;
8 cin >> n >> m;
9 vector<string> grid(n);
10 vvi times(n, vi(m, INF)), prev(n, vi(m, -1));
11 vector<vector<bool>> visited(n, vector<bool>(m, false));
12 queue<iiii> q;
13 ii start, end;
14 for(int i = 0; i < n; i++) {
15     cin >> grid[i];
16     for(int j = 0; j < m; j++) {
17         if( grid[i][j] == 'M' ) {
18             q.push({i,j,0});
19             times[i][j] = 0;
20         }
21         else if( grid[i][j] == 'A' )
22             start = {i,j};
23     }
24 }
25 function<bool(int, int)> valid = [&](int I, int J) ->
26     bool {
27         return (I >= 0) and (I < n) and (J >= 0) and (J < m) and (grid[I][J] != '#') and (times[I][J] ==
28             INF);
29     };
30 function<bool(int, int)> valid_player = [&](int I,
31     int J) -> bool {
32     return (I >= 0) and (I < n) and (J >= 0) and (J < m) and (!visited[I][J]) and (grid[I][J] != '#');
33 };
34 function<bool(int, int)> is_border = [&](int I, int J) -> bool {
35     return I == 0 || I == n-1 || J == 0 || J == m-1;
36 };
37 // Corner cases
38 if( is_border(start.fi, start.se) ) {
39     cout << "YES" << endl << "0" << endl;
40     return 0;
41 }
42 // Multi-Source BFS
43 while( ! q.empty() ) {
44     iii u = q.front();
45     q.pop();
46     for(ii dir : dirs) {
47         int newI = u[0] + dir.fi;
48         int newJ = u[1] + dir.se;
49         int w = u[2] + 1;
50         if( valid(newI, newJ) ) {
51             times[newI][newJ] = w;
52             q.push({newI, newJ, w});
53         }
54     }
55 }
```

```

51     }
52 }
53 // Player BFS
54 q.push({start.fi,start.se,0});
55 visited[start.fi][start.se] = true;
56 while( ! q.empty() and !escaped ) {
57     iii u = q.front();
58     q.pop();
59     for(int i = 0; i < 4; i++) {
60         int newI = u[0] + dirs[i].fi;
61         int newJ = u[1] + dirs[i].se;
62         int w = u[2] + 1;
63         if( valid_player(newI,newJ) and w < times[newI][newJ] ) {
64             visited[newI][newJ] = true;
65             prev[newI][newJ] = i;
66             q.push({newI,newJ,w});
67             if( is_border(newI,newJ) ) {
68                 end.fi = newI;
69                 end.se = newJ;
70                 escaped = true;
71                 break;
72             }
73         }
74     }
75 }
76 if( !escaped ) {
77     cout << "NO" << endl;
78     return 0;
79 }
80 // Path reconstruction
81 cout << "YES" << endl;
82 int i = end.fi;
83 int j = end.se;
84 while( prev[i][j] != -1 ) {
85     int oldI = i;
86     arns += path[ prev[i][j] ];
87     i -= dirs[prev[i][j]].fi;
88     j -= dirs[prev[oldI][j]].se;
89 }
90 reverse(all(arns));
91 cout << sz(arns) << endl << arns << endl;

```

Listing 30: Classic Lava Flow problem implementation, where the timer from the starting point A needs to be less than every other in the MS-BFS starting in M places. Once one edge is reached, the path is reconstructed from the output. Runs in BFS complexity $O(|V| + |E|)$. Originally used in the CSES problem *Monsters*.

5.15 MaxFlow

5.15.1 Dinic's Algoirthm

```

1 const ll INF = 1e17;
2 /**
3  * @brief Represents a directed edge in a flow
4  * network.
5  * @details Stores the edge's source, destination,
6  * capacity, and current flow.

```

```
5 *           Used in max-flow algorithms like Dinic
6 or Ford-Fulkerson. */
7 struct flowEdge {
8     int u; // Source node
9     int v; // Destination node
10    ll cap; // Maximum flow capacity of the edge
11    ll flow = 0; // Current flow through the edge (
12        initially 0)
13    flowEdge( int u, int v, ll cap ) : u(u), v(v), cap
14        (cap) {};
15 };
16 /**
17 * @brief Implementation of Dinic's max-flow
18 algorithm.
19 * @details Manages a flow network with BFS (Level
20 Graph) and DFS (Blocking Flow) optimizations. */
21 struct Dinic {
22     vector<flowEdge> edges; // All edges in the flow
23         network (including reverse edges)
24     vector<vi> adj;
25     int n; // Total number of nodes in the graph
26     int s; // Source node
27     int t; // Sink node (destination of flow)
28     int id = 0; // Counter for edge indexing
29     vi level; // Stores the level (distance from 's')
30         of each node during BFS
31     vi next; // Optimization for DFS: tracks the next
32         edge to explore for each node
33
34     queue<int> q; // Queue for BFS traversal
35     /**
36      * @brief Constructs a Dinic solver for a flow
37      network.
38      * @param n Number of nodes.
39      * @param s Source node.
40      * @param t Sink node. */
41     Dinic( int n, int s, int t ) : n(n), s(s), t(t) {
42         adj.resize(n); // Initialize adjacency list for
43         'n' nodes.
44         level.resize(n); // Prepare level array for BFS.
45         next.resize(n); // Prepare next-edge array for
46         DFS.
47         fill(all(level),-1); // Mark all levels as
48         unvisited (-1).
49         level[s] = 0; // The source has level 0.
50         q.push(s); // Start BFS from the source.
51     }
52     /**
53      * @brief Adds a directed edge and its residual
54      reverse edge to the flow network. */
55     void addEdge( int u, int v, ll cap ) {
56         edges.emplace_back(u,v,cap); // Original edge: u
57             -> v
58         edges.emplace_back(v,u,0); // Residual edge: v
59             -> u
60         adj[u].pb(id++);
61         adj[v].pb(id++);
62     }
63 }
```

```
46 }  
47 /**  
48 * @brief Performs BFS to construct the level  
graph (Layered Network) from source 's' to sink '  
t'.  
49 * @details Assigns levels (minimum distances  
from 's') to all nodes and checks if 't' is  
reachable.  
50 *          Levels are used to guide the DFS  
phase in Dinic's algorithm.  
51 * @return bool True if the sink 't' is reachable  
(i.e., there exists an augmenting path), false  
otherwise. */  
52 bool bfs() {  
53     while( ! q.empty() ) {  
54         int curr = q.front();  
55         q.pop();  
56         for( auto e : adj[curr] ) {  
57             if( edges[e].cap - edges[e].flow < 1 ) //  
Skip saturated edges (no residual capacity).  
58                 continue;  
59             if( level[ edges[e].v ] != -1 ) // Skip  
already visited nodes (level assigned).  
60                 continue;  
61             // Assign level to the neighbor node.  
62             level[ edges[e].v ] = level[ edges[e].u ] +  
1; // Next level = current + 1.  
63         q.push( edges[e].v ); // Add neighbor to the  
queue for further BFS.  
64     }  
65 }  
66 return level[t] != -1; // Return whether the  
sink 't' was reached (level[t] != -1).  
67 }  
68 /**  
69 * @brief Finds a blocking flow using DFS in the  
level graph constructed by BFS.  
70 * @param u Current node being processed.  
71 * @param flow Maximum flow that can be sent from  
'u' to the sink 't'.  
72 * @return ll The amount of flow successfully  
sent to 't'. */  
73 ll dfs( int u, ll flow ) {  
74     if( flow == 0 ) // No remaining flow to send.  
75         return 0;  
76     if( u == t ) // Reached the sink; return  
accumulated flow.  
77     return flow;  
78     // Explore edges from 'u' using 'next[u]' to  
avoid revisiting processed edges.  
79     for( int& cid = next[u]; cid < sz(adj[u]); cid++  
) {  
80         int e = adj[u][cid]; // Index of the edge in '  
edges'.
```

```
81     int v = edges[e].v; // Destination node of
the edge.
82     // Skip invalid edges:
83     // 1. Not in the level graph (level[u] + 1 != level[v]). Just edges in exactly one level ahead (ensures shortest paths).
84     // 2. No residual capacity (cap - flow < 1).
85     if( level[edges[e].u] + 1 != level[v] || edges[e].cap - edges[e].flow < 1 )
86         continue;
87     ll f = dfs( v, min(flow, edges[e].cap - edges[e].flow) ); // Recursively send flow to 'v'.
88     if( f == 0 ) // No flow could be sent via this edge.
89         continue;
90     // Update residual capacities:
91     edges[e].flow += f; // Increase flow in the original edge.
92     edges[e ^ 1].flow -= f; // Decrease flow in the reverse edge. (All reverse edges have distinct parity)
93     return f; // Return the flow sent.
94 }
95 return 0; // No augmenting path found from 'u'.
96 }
97 /**
98 * @brief Computes the maximum flow from source 's' to sink 't' using Dinic's algorithm.
99 * @details Iterates through BFS and DFS phases to find the maximum flow.
100 Accumulates flow while there exists augmentation paths in the residual graph.
101 Restart auxiliary structures for every new phase.
102 * @return ll The maximum flow value. */
103 ll maxFlow() {
104     ll flow = 0; // Tracks the total flow sent.
105     while( bfs() ) { // While there are augmenting paths:
106         fill(all(next), 0); // Reset 'next' for DFS.
107         for( ll f = dfs(s, INF); f != 0LL; f = dfs(s,
108             INF) ) // Send blocking flow in the level graph:
109             flow += f;
110         // Reset for next BFS phase:
111         fill(all(level), -1);
112         level[s] = 0;
113         q.push(s);
114     }
115     return flow;
116 }
117 /**
118 * @brief Finds edges belonging to the minimum cut after maxFlow().
119 */
```

```
118     * @details First, it marks all the reachable
119     nodes from 's' with an augmentation path after
120     obtained the max flow
121
122     and all the saturated edges coming out from
123     any of the nodes who belong to the min-cut.
124
125     For 'minCut()' to work, 'maxFlow()' must be
126     first executed to get the min-cut.
127
128     If only is needed the value, is enough
129     returning the value of 'maxFlow()'.
```

* @return vii List of edges (u, v) in the min-cut. Its size is the minimum number of 'roads' to close. */

```
123 vii minCut() {
124     vii ans;
125     fill(all(level), -1); // Reset levels.
126     level[s] = 0; // Mark source as reachable
127
128     q.push(s);
129     while( ! q.empty() ) { // BFS to mark nodes
130         reachable from 's' in the residual graph.
131         int curr = q.front();
132         q.pop();
133         for( int id = 0; id < sz(adj[curr]); id++ ) {
134             // For every edge going out from 'curr'.
135             int e = adj[curr][id];
136             // If 'v' is has not been visited yet, and
137             // the edge have residual capacity.
```

```
134             if( level[edges[e].v] == -1 && edges[e].cap
135             - edges[e].flow > 0 ) {
136                 q.push(edges[e].v);
137                 level[edges[e].v] = level[edges[e].u] + 1;
138             }
139         }
140         for( int i = 0; i < sz(level); i++ ) {
141             if( level[i] != -1 ) {
142                 for( int id = 0; id < sz(adj[i]); id++ ) {
143                     int e = adj[i][id];
144                     if( level[edges[e].v] == -1 && edges[e].
145                     cap - edges[e].flow == 0 )
146                         ans.emplace_back(edges[e].u, edges[e].v);
147                 }
148             }
149         }
150     }
151     /**
152      * @brief Reconstructs the maximum bipartite
153      matching after running 'maxFlow()'.
```

* @details Every edge that belong to the original graph and have flow greater than zero, belongs to the matching.

```
153
154     For 'maximumMatching()' to work, 'maxFlow()'
```

```

155     * @return vii List of matched pairs (boy, girl).
156     */
157
158     vii maximumMatching() {
159         vii ans;
160         fill(all(level), -1); // Reset levels.
161         level[s] = 0; // Mark source as reachable
162         .
163         q.push(s);
164         while( ! q.empty() ) { // BFS to mark nodes
165             reachable via saturated edges with flow greater
166             than zero.
167             int curr = q.front();
168             q.pop();
169             for( int id = 0; id < sz(adj[curr]); id++ ) {
170                 int e = adj[curr][id];
171                 // If 'v' has not been visited yet, the edge
172                 is saturated and have flow greater than zero.
173                 if( level[edges[e].v] == -1 && edges[e].cap -
174                     edges[e].flow == 0 && edges[e].flow != 0ll ) {
175                     q.push(edges[e].v);
176                     level[edges[e].v] = level[edges[e].u] + 1;
177                 }
178             }
179             for( int i = 0; i < sz(level); i++ ) { //
180                 Collect original edges (boy -> girl) that are
181                 saturated and have flow > 0.
182                 if( level[i] != -1 ) {
183                     for( int id = 0; id < sz(adj[i]); id++ ) {
184                         int e = adj[i][id];
185                         if( edges[e].u != s && edges[e].v != t
186                             && edges[e].cap - edges[e].flow == 0 && edges[e].
187                             flow != 0ll )
188                             ans.emplace_back(edges[e].u,edges[e].v
189 );
190                     }
191                 }
192             }
193         }
194         return ans;
195     }
196 }
```

Listing 31: Commented template for solving MaxFlow problems with Dinic's algorithm. Works in complexity $O(|V|^2 \times |E|)$. In bipartite graphs and graphs with unitary max capacity the complexity turns $O(|E| \times \sqrt{|V|})$.

Download Speed How does a particular flow problem looks like? Following is the statement for the problem CSES 1694 (Download Speed):

Consider a network consisting of n computers and m connections. Each connection specifies how fast a computer can send data to another computer.

Kotivalo wants to download some data from a server. What is the maximum speed he can do this, using the connections in the network?

Input

The first input line has two integers n and m : the number of computers and connections. The computers are numbered $1, 2, \dots, n$. Computer 1 is the server and computer n is Kotivalo's computer.

After this, there are m lines describing the connections. Each line has three

integers a , b , and c : computer a can send data to computer b at speed c .

Output

Print one integer: the maximum speed Kotivalo can download data.

```

1 int main()
2 {
3     fastIO();
4
5     int n, m, u, v, w;
6
7     cin >> n >> m;
8
9     Dinic flow(n+1,1,n); // size n+1 to fix 0-
10    indexed indexes, 1 is the source (server), 'n' is
11    the sink (Kotivalo)
12
13    for(i,0,m)
14    {
15        cin >> u >> v >> w;
16        flow.addEdge(u,v,w);
17    }
18
19    cout << flow.maxFlow() << endl;
20
21    return 0;
22 }
```

Listing 32: Main method for solving CSES 1697 Download Speed using MaxFlow template.

Police Chase Max Flow-Min Cut Theorem: $\text{MaxFlow} = \text{MinCut}$.

Following is the statement for the problem CSES 1695 (Police Chase):

Kaaleppi has just robbed a bank and is now heading to the harbor. However, the police wants to stop him by closing some streets of the city.

What is the minimum number of streets that should be closed so that there is no route between the bank and the harbor?

Input

The first input line has two integers n and m : the number of crossings and streets. The crossings are numbered $1, 2, \dots, n$. The bank is located at crossing 1, and the harbor is located at crossing n .

After this, there are m lines that describing the streets. Each line has two integers a and b : there is a street between crossings a and b . All streets are two-way streets, and there is at most one street between two crossings.

Output

First print an integer k : the minimum number of streets that should be closed. After this, print k lines describing the streets. You can print any valid solution.

```

1 int main()
2 {
3     fastIO();
4
5     int n, m, u, v;
6     vii minCut;
7
8     cin >> n >> m;
9
10    Dinic flow(n+1,1,n); // size n+1 to fix 0-
11    indexed indexes, 1 is the source (bank), 'n' is
12    the sink (harbor)
13
14    cout << minCut.size() << endl;
15
16    for(i,0,minCut.size())
17    {
18        cout << minCut[i].first << " " << minCut[i].second << endl;
19    }
20
21    return 0;
22 }
```

```

12     for(i,0,m)
13     {
14         cin >> u >> v;
15         flow.addEdge(u,v,1);
16         flow.addEdge(v,u,1);
17     }
18
19     flow.maxFlow();
20
21     minCut = flow.minCut();
22
23     cout << (sz(minCut)/2) << endl;
24     for(int i = 0; i < sz(minCut); i += 2)
25         cout << minCut[i].fi << " " << minCut[i].se
26     << endl;
27
28     return 0;
29 }
```

Listing 33: Main method for solving CSES 1695 Police Chase using MaxFlow template.

School Dance MaxFlow = MinCut = MaxMatching.

Following is the statement for the problem CSES 1696 (School Dance):

There are n boys and m girls in a school. Next week a school dance will be organized. A dance pair consists of a boy and a girl, and there are k potential pairs.

Your task is to find out the maximum number of dance pairs and show how this number can be achieved.

Input

The first input line has three integers n , m and k : the number of boys, girls, and potential pairs. The boys are numbered $1, 2, \dots, n$, and the girls are numbered $1, 2, \dots, m$.

After this, there are k lines describing the potential pairs. Each line has two integers a and b : boy a and girl b are willing to dance together.

Output

First print one integer r : the maximum number of dance pairs. After this, print r lines describing the pairs. You can print any valid solution.

```

1 int main()
2 {
3     fastIO();
4
5     int n, m, k, a, b;
6     ll maxPairs;
7     vii pairs;
8
9     cin >> n >> m >> k;
10
11    Dinic flow(n+m+2,0,n+m+1);
12
13    fori(boy,0,n+1)
14        flow.addEdge(0,boy,1);
15
16    fori(girl,n+1,n+m+1)
17        flow.addEdge(girl,n+m+1,1);
18
19    fori(i,0,k)
20    {
```

```

21     cin >> a >> b;
22     flow.addEdge(a, n+b, 1);
23 }
24
25 maxPairs = flow.maxFlow();
26 pairs = flow.maximumMatching();
27
28 cout << maxPairs << endl;
29 for(i, 0, sz(pairs))
30     cout << pairs[i].fi << " " << (pairs[i].se -
n) << endl;;
31
32 return 0;
33 }
```

Listing 34: Main method for solving CSES 1696 School Dancing using MaxFlow template.

5.15.2 *Ford-Fulkerson Algorithm

5.15.3 *Goldber-Tarjan Algorithm

6 Trees

6.1 Counting Childrens

```

1 vi childrens(n+1, 0);
2 vvi graph(n+1);
3 vector<bool> visited(n+1, false);
4 for(i, 2, n+1) {
5     cin >> tmp;
```

```

6     graph[tmp].pb(i);
7     graph[i].pb(tmp);
8 }
9 function<int(int)> dfs = [&](int u) -> int {
10     visited[u] = true;
11     for(int v : graph[u]) {
12         if( !visited[v] )
13             childrens[u] += dfs(v);
14     }
15     return childrens[u] + 1;
16 };
17 dfs(1);
```

Listing 35: Algorithm that counts how many childrens does every node have, from 2..n in a rooted tree (root = 1).

6.2 *Tree Diameter

6.3 *Centroid Decomposition

6.4 *Euler Tour

6.5 *Lowest Common Ancestor (LCA)

6.6 *Heavy-Light Decomposition (HLD)

7 Strings

7.1 Knuth-Morris-Pratt Algorithm (KMP)

```

1 // Longest Prefix-Suffix
2 vi compute_LPS(string s) {
3     size_t len = 0, i = 1, sz = s.size();
```

```
4     vi lps(sz,0);
5     while( i < sz ) {
6         if( s[i] == s[len] )
7             lps[i++] = ++len;
8         else
9             if( len != 0 )
10                 len = lps[len-1];
11             else
12                 lps[i++] = 0;
13     }
14     return lps;
15 }
16 // Get number of occurrences of a pattern p in a
17 // string s
18 int kmp(string s, string p) {
19     vi lps = compute_LPS(p);
20     size_t n = s.size(), m = p.size(), i = 0, j = 0;
21     int cnt = 0;
22     while( i < n ) {
23         if( p[j] == s[i] ) {
24             j++; i++;
25         }
26         if( j == m ) { // Full match
27             cnt++;
28             j = lps[j-1];
29         }
30         else if( i < n and p[j] != s[i] ) { // Mismatch
31             if( j != 0 )
32                 j = lps[j-1];
33             else
34                 i++;
35         }
36     }
37     return cnt;
38 }
```

Listing 36: KMP algorithm for counting how many times a pattern appear into a string. Runs in $O(n + m)$.

7.2 *Suffix Array**7.3 *Rolling Hashing****7.4 *Z Function****7.5 *Aho-Corasick Algorithm****8 Dynamic Programming****8.1 *Coins****8.2 *Longest Increasing Subsequence (LIS)****8.3 *Edit Distance****8.4 *Knapsack****8.5 *SOS DP****8.6 *Digit DP****8.7 *Bitmask DP****9 Mathematics****9.1 Number Theory****9.1.1 Greatest Common Divisor (GCD)**

```

1 int gcd(int a, int b) {
2     if (a == 0) return b;
3     if (b == 0) return a;
4     if (a == b) return a;
5     if (a > b)
6         return gcd(a - b, b);
7     return gcd(a, b - a);
8 }
```

Listing 37: Implementation of handmade GCD, because using `gcd()` runs slow with long long, also `--gcd()`.

9.1.2 Gauss Sum

The sum of the first n natural numbers in $O(1)$.

$$S = \frac{n(n+1)}{2} \quad (1)$$

$$n = \sqrt{2S + \frac{1}{4}} - \frac{1}{2} \quad (2)$$

```

1 int S = (1LL * n * (1LL * n + 1LL))/2;
2 int n = (int)( sqrt( 2 * S + 0.25 ) - 0.5 )
```

Listing 38: Implementation of the Gauss Sum.

9.1.3 *Modular Theory**9.1.4 *Modulo Inverse****9.1.5 *Fermat's Little Theorem****9.1.6 *Chinese Remainder Theorem****9.1.7 Binpow**

```

1 const int MOD = 1e9+7;
2 int binpow( long long a, long long b ) { // a^b
3     long long sol = 1;
4     a %= MOD;
5     while( b > 0 ) {
6         if( b & 1 )
```

```
7         sol = ( 1LL * sol * a ) % MOD;
8
9         a = ( 1LL * a * a ) % MOD;
10
11        b >= 1;
12
13    }
14
15    return sol % MOD;
16
17 }
```

Listing 39: Applying binary exponentiation to a problem requiring $a^b \bmod(10^9 + 7)$ in $O(\log_2(b))$.

```

15 Matrix(int n, int m) {
16     mat.resize(n);
17     for(int i = 0; i < n; i++) {mat[i].resize(m);
18 }
19 }
20 int rows() const { return mat.size(); }
21 int cols() const { return mat[0].size(); }
22 void makeIdent() {
23     for(int i = 0; i < rows(); i++)
24         for(int j = 0; j < cols(); j++)
25             mat[i][j] = (i == j ? 1 : 0);
26 }
27 Matrix operator*=(const Matrix &b) {
28     matmul(mat, b.mat);
29     return *this;
30 }
31 void print() {
32     for(int i = 0; i < rows(); i++) {
33         for(int j = 0; j < cols(); j++)
34             cout << mat[i][j] << " ";
35         cout << endl;
36     }
37 }
38 Matrix operator*(const Matrix &b) { return Matrix
39     (*this) *= b; }
40 };
41 int main() {
42     Matrix<11> A( {{1,1},{1,0}} );

```

9.1.8 Matrix Exponentiation (Linear Recurrency)

```
1 template <typename T> void matmul(vector<vector<T>>
2     &a, const vector<vector<T>>& b) {
3     size_t n = a.size(), m = a[0].size(), p = b[0].
4     size();
5     assert(m == b.size());
6     vector<vector<T>> c(n, vector<T>(p));
7     for(size_t i = 0; i < n; i++)
8         for(size_t j = 0; j < p; j++)
9             for(size_t k = 0; k < m; k++)
10                 c[i][j] = (c[i][j] + a[i][k] * b[k][j])
11                 % MOD;
12     a = c;
13 }
14
15 template <typename T> struct Matrix {
16     vector<vector<T>> mat;
17     Matrix() {}
18     Matrix(vector<vector<T>> a) { mat = a; }
```

```

41 Matrix<ll> ini(2,1);
42 ini.mat[0][0] = 0;
43 ini.mat[1][0] = 1;
44 Matrix<ll> iden(2,2);
45 iden.makeIden();
46 ll n;
47 cin >> n;
48 while(n > 0) {
49     if( n & 1 ) iden *= A;
50     A *= A;
51     n >>= 1;
52 }
53 Matrix<ll> res = iden * ini;
54 cout << res.mat[0][0] << endl;
55 return 0;
56 }
```

Listing 40: Template to pow a matrix of size n to a certain exponent with logarithmic time (using binpow), and multiply it to another matrix, with modulo operation, as well as how to use it. Full implementation for calculating n -th Fibonacci term with linear recurrency.

9.1.9 Prime checking

```

1 bool prime( int n ){
2     if( n == 2 )
3         return true;
4     if( n % 2 == 0 || n <= 1 )
5         return false;
```

```

6     for( int i = 3; i * i <= n; i += 2 )
7         if( ( n % i ) == 0 )
8             return false;
9     return true;
10 }
```

Listing 41: Returns if n is a prime number in $O(\sqrt{n})$. Avoids overflow $\forall n \leq 10^6$ ($\approx INT_MAX$).

9.1.10 Prime factorization

```

1 void prime_factorization(vll& factorization, ll n) {
2     for(long long d = 2; d*d <= n; d++) {
3         while(n % d == 0) {
4             factorization.push_back(d);
5             n /= d;
6         }
7     }
8     if( n > 1 )
9         factorization.push_back(n);
10 }
```

Listing 42: Returns prime factorization of the number n using *trial division*, simplest way. Runs in $O(\sqrt{n})$. e.g. for 12 the result is 2x2x3.

9.1.11 Sieve of Eratosthenes

```

1 void sieve_of_eratosthenes(vector<bool>& is_prime,
2                             int n) {
3     is_prime.assign(n+1,true);
```

```

3     is_prime[0] = is_prime[1] = false;
4     for(int i = 2; i <= n; i++) {
5         if( is_prime[i] && (long long)i * i <= n ) {
6             for(int j = i*i; j <= n; j += i)
7                 is_prime[j] = false;
8         }
9     }
10 }
```

Listing 43: Calculates every prime number up to n with sieve of eratosthenes in a boolean 1-indexed vector. Runs in $O(n \log \log n)$.

9.1.12 Sum of Divisors

```

1 ll sum_of_divisors(ll n) {
2     ll sum = 1;
3     for (long long i = 2; i * i <= n; i++) {
4         if(n % i == 0) {
5             int e = 0;
6             do {
7                 e++;
8                 n /= i;
9             } while (n % i == 0);
10            ll s = 0, pow = 1;
11            do {
12                s += pow;
13                pow *= i;
14            } while (e-- > 0);
15            sum *= s;
16        }
17    }
18    if(n > 1)
19        sum *= (1 + n);
20    return sum;
21 }
```

```

16     }
17 }
18 if(n > 1)
19     sum *= (1 + n);
20 return sum;
21 }
```

Listing 44: Calculates the sum of all divisors of number n . e.g.
 $\text{sum_of_divisors}(12) = 18$. Runs in $O(\sqrt{n})$.

```

1 void sum_of_divisors_sieve( vll& sigma, int n ) {
2     sigma.assign(n+1,0);
3     for(int i = 1; i <= n; i++)
4         for(int j = i; j <= n; j+=i)
5             sigma[j] += i;
6 }
```

Listing 45: Calculates the sum of all divisors of all numbers from 1 to n . Runs in $O(n \log(n))$.

9.2 Combinatorics

9.2.1 Binomial Coefficients

```

1 const int MAXN = 1e6+1;
2 vll fact(MAXN+1), inv(MAXN+1);
3 int binpow( ll a, ll b ) { // a^b
4     ll sol = 1;
5     a %= MOD;
6     while( b > 0 ) {
```

```

7     if( b & 1 )
8         sol = ( 1LL * sol * a ) % MOD;
9         a = ( 1LL * a * a ) % MOD;
10        b >>= 1;
11    }
12
13    return sol % MOD;
14}
15
16 void combi() {
17     fact[0] = inv[0] = 1;
18     for(i,1,MAXN+1) {
19         fact[i] = fact[i-1] * i % MOD;
20         inv[i] = binpow( fact[i], MOD - 2 );
21     }
22
23 nCr( ll n, ll r ) {
24     return fact[n] * inv[r] % MOD * inv[n-r] % MOD;
25 }
26 combi();
27 nCr(a,b);

```

Listing 46: Template for calculating binomial coefficients $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Precalculate *fact* and *inv* runs in $O(MAXN \cdot \log_2(MOD))$ ($\log_2(MOD) \approx 30$). So, in general case when $NMAX = 10^6$ and $MOD = 10^9 + 7$ can be generalized to $O(n \cdot \log(n))$, $n \leq 10^6$.

9.2.2 Common combinatorics formulas

$$\binom{n}{2} = \frac{n(n-1)}{2} \quad (3)$$

$$\sum_{k=0}^n \binom{n}{k} = 2^n \quad (4)$$

$$\sum_{k=0}^n \binom{n}{k} \binom{n}{n-k} = \binom{2n}{n} \quad (5)$$

$$\sum_{k=0}^n k \binom{n}{k} = n2^{n-1} \quad (6)$$

$$\sum_{k=0}^{\infty} \binom{2k}{k} \binom{2n-2k}{n-k} = 4^n \quad (7)$$

$$(8)$$

9.3 *Stars and Bars

9.4 Probability

9.5 Computational Geometry

9.5.1 *Cross Product

9.5.2 *Convex Hull

9.6 *Fast Fourier Transform (FFT)

10 Appendix

10.1 What to do against WA?

1. Have you done the correct complexity analysis?
2. Have you understood well the statement?
3. Have you corroborated yet the trivial test cases?
4. Have you checked all the corner cases?
5. Have you proposed a lot of non-trivial test cases?

6. Isn't there any possibility of overflow? (Multiplying two `int` needs to be fitted into a `long long`)
7. Have you done a desktop test?
8. Have you red all the variables? (`tc` variable on `main`)
9. Every part of your code works as it's meant to?

10.2 Primitive sizes

Data type	[B]	Minimum value it takes	Maximum value it takes
<code>bool</code>	1	0	1
<code>signed char</code>	1	0	255
<code>unsigned char</code>	1	-128	127
<code>signed int</code>	4	$-2,147,483,648 \approx -2 \times 10^9$	$2,147,483,647 \approx 2 \times 10^9$
<code>unsigned int</code>	4	0	$4,294,967,295 \approx 4 \times 10^9$
<code>signed short</code>	2	-32,768	32,767
<code>unsigned short</code>	2	0	65,535
<code>signed long long int</code>	8	$-9,223,372,036,854,775,808 \approx -9 \times 10^{18}$	$9,223,372,036,854,775,807 \approx 9 \times 10^{18}$
<code>unsigned long long int</code>	8	0	$18,446,744,073,709,551,615 \approx 18 \times 10^{18}$
<code>float</code>	4	1.1×10^{-38}	3.4×10^{38}
<code>double</code>	8	2.2×10^{-308}	1.7×10^{308}
<code>long double</code>	12	3.3×10^{-4932}	1.1×10^{4932}

Table 1: Capacity of primitive data types in C++.

32	whitespace	58	:	65	A	97	a
33	!	59	;	66	B	98	b
34	"	60	i	67	C	99	c
35	#	61	=	68	D	100	d
36	\$	62	¸	69	E	101	e
37	%	63	?	70	F	102	f
38	&	64	@	71	G	103	g
39	'	91	[72	H	104	h
40	(92	\	73	I	105	i
41)	93]	74	J	106	j
42	*	94	^	75	K	107	k
43	+	95	-	76	L	108	l
44	,	96	,	77	M	109	m
45	-	126	~	78	N	110	n
46	.			79	O	111	o
47	/			80	P	112	p
48	0			81	Q	113	q
49	1			82	R	114	r
50	2			83	S	115	s
51	3			84	T	116	t
52	4			85	U	117	u
53	5			86	V	118	v
54	6			87	W	119	w
55	7			88	X	120	x
56	8			89	Y	121	y
57	9			90	Z	122	z

Table 2: Code and symbol of printable ASCII characters.

10.4 Numbers bit representation

1	00000001	31	00011111	61	00111101	91	01011011	121	01111001
2	00000010	32	00100000	62	00111110	92	01011100	122	01111010
3	00000011	33	00100001	63	00111111	93	01011101	123	01111011
4	00000100	34	00100010	64	01000000	94	01011110	124	01111100
5	00000101	35	00100011	65	01000001	95	01011111	125	01111101
6	00000110	36	00100100	66	01000010	96	01100000	126	01111110
7	00000111	37	00100101	67	01000011	97	01100001	127	01111111
8	00001000	38	00100110	68	01000100	98	01100010	128	10000000
9	00001001	39	00100111	69	01000101	99	01100011	129	10000001
10	00001010	40	00101000	70	01000110	100	01100100	130	10000010
11	00001011	41	00101001	71	01000111	101	01100101	131	10000011
12	00001100	42	00101010	72	01001000	102	01100110	132	10000100
13	00001101	43	00101011	73	01001001	103	01100111	133	10000101
14	00001110	44	00101100	74	01001010	104	01101000	134	10000110
15	00001111	45	00101101	75	01001011	105	01101001	135	10000111
16	00010000	46	00101110	76	01001100	106	01101010	136	10001000
17	00010001	47	00101111	77	01001101	107	01101011	137	10001001
18	00010010	48	00110000	78	01001110	108	01101100	138	10001010
19	00010011	49	00110001	79	01001111	109	01101101	139	10001011
20	00010100	50	00110010	80	01010000	110	01101110	140	10001100
21	00010101	51	00110011	81	01010001	111	01101111	141	10001101
22	00010110	52	00110100	82	01010010	112	01110000	142	10001110
23	00010111	53	00110101	83	01010011	113	01110001	143	10001111
24	00011000	54	00110110	84	01010100	114	01110010	144	10010000
25	00011001	55	00110111	85	01010101	115	01110011	145	10010001
26	00011010	56	00111000	86	01010110	116	01110100	146	10010010
27	00011011	57	00111001	87	01010111	117	01110101	147	10010011
28	00011100	58	00111010	88	01011000	118	01110110	148	10010100
29	00011101	59	00111011	89	01011001	119	01110111	149	10010101
30	00011110	60	00111100	90	01011010	120	01111000	150	10010110

10.5 How a `vector<vector<pair<int,int>>` looks like

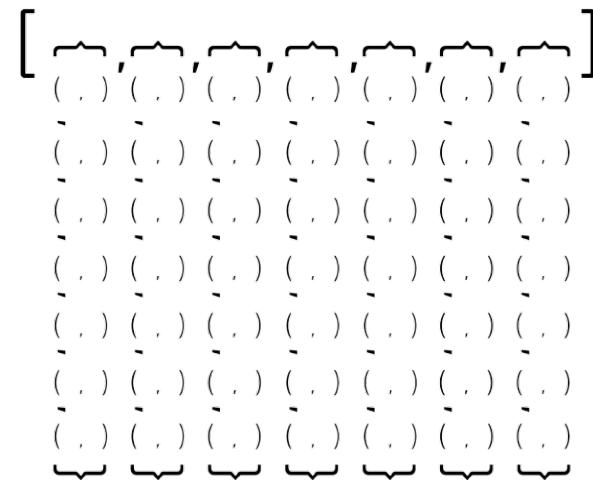


Figure 1: Visual representation of a vector of vector of pairs.

10.6 How all neighbours of a grid looks like

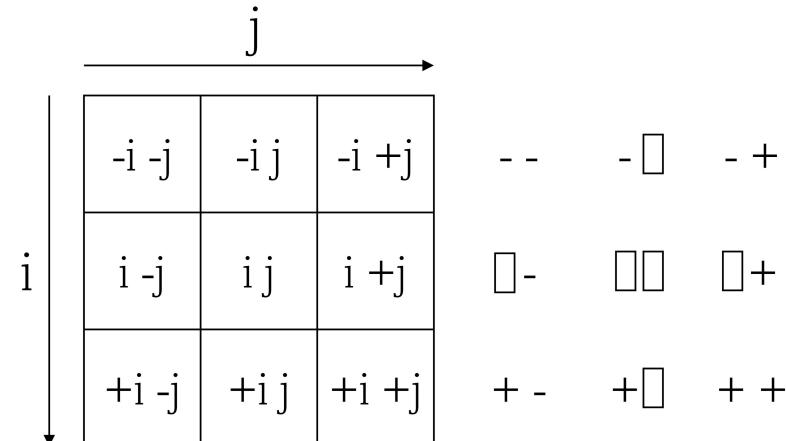


Figure 2: Visual representation of how all adjacent cells in a grid looks like.