

# Everyone's Connected



## Contents

<b>1</b>	<b>Template</b>	<b>2</b>
<b>2</b>	<b>Data structures</b>	<b>2</b>
2.1	STL Algorithms	2
2.2	Binary Search	3
2.3	Simplified DSU (Stolen from GGDem)	4
2.4	Disjoint Set Union	4
2.5	Segment Tree	4
2.6	Segment Tree Lazy	4
2.7	Trie	4
<b>3</b>	<b>Graphs</b>	<b>4</b>
3.1	Graph Transversal	4
3.1.1	BFS	4
3.1.2	DFS	4
3.2	Topological Sort	5
3.3	APSP: Floyd Warshall	5
3.4	SSSP	5
3.4.1	Lazy Dijkstra	5
3.4.2	Bellman-Ford	5
3.5	Strongly Connected Components: Kosaraju	5
3.6	Articulation Points and Bridges: ModTarjan	5

<b>4</b>	<b>Math</b>	<b>5</b>
4.1	Identities	5
4.2	Binary Exponentiation and modArith	6
4.3	Modular Inverse (dividir mod)	6
4.4	Modular Binomial Coefficient and Permutations	6
4.5	Non-Mod Binomial Coefficient and Permutations	6
4.6	Modular Catalan Numbers	6
4.7	Ceil Fraccionario	6
4.8	Numeros de Fibonacci	6
4.9	Sieve Of Eratosthenes	6
4.10	Sieve-based Factorization	6
4.11	Cycle Finding	6
4.12	Berlekamp Massey	6
4.13	Modular Berlekamp Massey	6
4.14	Matrix exponentiation	6
4.15	Ecuaciones Diofantinas	6
4.16	Pollard-Rho, Stolen from GGDem	6
4.17	FFT, Stolen from GGDem	6
4.18	Euler Totient Function	6
<b>5</b>	<b>Geometry</b>	<b>6</b>
<b>6</b>	<b>Strings</b>	<b>6</b>
6.1	Explode by token	6
6.2	Multiple Hashings DS	7
6.3	Permute chars of string	7
6.4	Longest common subsequence	7
6.5	KMP	7
6.6	Suffix Array	7
6.7	STL Suffix Array	7
<b>7</b>	<b>Classics</b>	<b>7</b>
7.1	Job scheduling	7
7.1.1	One machine, linear penalty	7
7.1.2	One machine, deadlines	7
7.1.3	One machine, profit	7
7.1.4	Two machines, min time	7
<b>8</b>	<b>Flow</b>	<b>7</b>
8.1	Dinic, thx GGDem	7
<b>9</b>	<b>Miscellaneous</b>	<b>7</b>
9.1	pbds	7

9.2 Bit Manipulation . . . . . 7

10 Testing 7

10.1 Gen and AutoRun testcases . . . . . 7

10.1.1 Gen.cpp . . . . . 7

10.1.2 Stress testing . . . . . 7

10.1.3 Autorun . . . . . 7

10.2 Highly Composite Numbers . . . . . 7

1 Template

```
1 #include <bits/stdc++.h>
2 #define endl '\n'
3 #define ll long long int
4 #define ull unsigned long long int
5 #define MOD7 1000000007
6 #define MOD9 1000000009
7 #define MAX 1000001
8
9 using namespace std;
10
11 /*-----SOLBEGIN-----*/
12
13 void solve() {
14     return;
15 }
16
17 int main() {
18     ios_base::sync_with_stdio(0);
19     cin.tie(0);
20
21     int t = 1; cin >> t;
22     while (t--) solve();
23
24     return 0;
25 }
```

2 Data structures

2.1 STL Algorithms

STL stands for Standard Template Library. It is a library that provides several generic classes and functions, allowing programmers to manipulate data structures in an easy and efficient way. The STL provides a range of algorithms which can be used to manipulate data stored in containers. The following list shows some of the algorithms provided by the STL and its functions:

Non-Manipulating Algorithms

- **sort(first\_iterator, last\_iterator)** - Sorts the elements in the range [first, last) in ascending order.
- **sort(frst\_iterator, last\_iterator, greater<int>())** - Sorts elements inside the vector, in descending order.

- **reverse(first\_iterator, last\_iterator)** - Reverses elements inside a vector.
- **\*max\_element(first\_iterator, last\_iterator)** - Finds the maximum element of a vector.
- **\*min\_element(first\_iterator, last\_iterator)** - Finds the minimum element of a vector.
- **accumulate(first\_iterator, last\_iterator, initial value of sum)** - Summates all the vector elements.
- **count(first\_iterator, last\_iterator, x)** - Counts all occurrences 'x' inside a vector.
- **find(first\_iterator, last\_iterator, x)** - Returns an iterator to the first occurrence of 'x' in vector and points to last address if the element is not present.
- **binary\_search(first\_iterator, last\_iterator, x)** - Tests if 'x' exists in sorted vector or not.
- **lower\_bound(first\_iterator, last\_iterator, x)** - Returns an element pointing to the first element in range [first, last), which has a value less than 'x'.
- **upper\_bound(first\_iterator, last\_iterator, x)** - Returns an element pointing to the first element in range [first, last), which has a value greater than 'x'.

### Manipulating Algorithms

- **arr.erase(position to delete)** - Erases selected element in vector and shifts and resizes it accordingly.
- **arr.erase(unique(arr.begin(), arr.end()), arr.end())** - Erases the duplicate occurrences in sorted vector in a single line.
- **next\_permutation(first\_iterator, last\_iterator)** - Modifies the vector to its next permutation.
- **prev\_permutation(first\_iterator, last\_iterator)** - Modifies the vector to its previous permutation.
- **distance(first\_iterator, desired\_iterator)** - Returns the distance of the desired position from the first iterator to a desired one.

## 2.2 Binary Search

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  vector<int> vec;
5
6  int binary_search_first_occurrence(const vector<int>& vec, int value) {
7      // Binary search algorithm finds the first occurrence of a value in
8      // a sorted vector
9      // Declare left and right pointers
10     int left = 0;
11     int right = vec.size() - 1;
12     int result = -1;
13     // While left and right pointers do not cross, keep searching
14     while (left <= right) {
15         // Calculate the middle element of the vector
16         int mid = left + (right - left) / 2;
17         // If the middle element is the value we are looking for, return
18         // its index
19         if (vec[mid] == value) {
20             result = mid;
21             // left = mid + 1; // Continue searching in the right half
22             // (for last occurrence)
23             right = mid - 1; // Continue searching in the left half
24             // If the middle element is smaller than the value we are
25             // looking for, search in the right half
26         } else if (vec[mid] < value) {
27             left = mid + 1;
28             // If the middle element is greater than the value we are
29             // looking for, search in the left half
30         } else {
31             right = mid - 1;
32         }
33     }
34     return result; // Returns -1 if value is not found
35 }
36
37 int main() {
38     // Assign the variable value to the value you want to search
39     int elements, value = 0;
40     cin >> elements;
41     // Read the elements of the vector

```

```

37     for (int i = 0; i < elements; i++) {
38         int x;
39         cin >> x;
40         vec.push_back(x);
41     }
42     cout << binary_search_first_occurrence(vec, value);
43
44     return 0;
45 }

```

## 2.3 Simplified DSU (Stolen from GGDem)

### 2.4 Disjoint Set Union

### 2.5 Segment Tree

### 2.6 Segment Tree Lazy

### 2.7 Trie

## 3 Graphs

### 3.1 Graph Transversal

#### 3.1.1 BFS

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  vector<bool> visited;
5  vector<vector<int>>> adj;
6
7  void breadth_first_search(int node) {
8      // BFS requires queue data structure, starting from a given initial
      // node
9      queue<int> q;
10     q.push(node);
11     visited[node] = true;
12     // While queue is not empty, pop the first element and push its
      // children
13     while (!q.empty()) {
14         int v = q.front();
15         cout << v << "□";
16         q.pop();
17         // Push all children of v

```

```

18     for (int u : adj[v]) {
19         // If not visited, push and mark as visited
20         if (!visited[u]) {
21             q.push(u);
22             visited[u] = true;
23         }
24     }
25 }
26
27
28 int main() {
29     int nodes, edges;
30     cin >> nodes >> edges;
31     // Initialize visited and adjacency list
32     visited.assign(nodes, false);
33     adj.assign(nodes, vector<int>());
34     int u, v;
35     // Values of nodes, given as pairs
36     for (int i = 0; i < edges; i++) {
37         cin >> u >> v;
38         adj[u].push_back(v);
39         adj[v].push_back(u); // <- Assuming undirected graph
40     }
41     breadth_first_search(0); // Start BFS from node x
42
43     return 0;
44 }

```

#### 3.1.2 DFS

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  vector<bool> visited;
5  vector<vector<int>>> adj;
6
7  void depth_first_search(int node) {
8      // DFS requires stack data structure, starting from a given initial
      // node
9      visited[node] = true;
10     cout << node << '□';
11     // For each child of node, if it hasn't been visited, call DFS
      // function

```

```

12     for(int i = 0; i < adj[node].size(); i++) {
13         int child = adj[node][i];
14         if(!visited[child]) {
15             depth_first_search(child);
16         }
17     }
18 }
19
20 int main() {
21     int nodes, edges;
22     cin >> nodes >> edges;
23     // Initialize visited and adjacency list
24     visited.assign(nodes, false);
25     adj.assign(nodes, vector<int>());
26     // Values of nodes, given as pairs
27     for(int i = 0; i < edges; i++) {
28         int u, v;
29         cin >> u >> v;
30         adj[u].push_back(v);
31         adj[v].push_back(u); // <- Assuming undirected graph
32     }
33     // For each node, if it hasn't been visited, call DFS function
34     for(int i = 0; i < nodes; i++) {
35         if(!visited[i]) {
36             depth_first_search(i);
37         }
38     }
39
40     return 0;
41 }

```

## 3.2 Topological Sort

## 3.3 APSP: Floyd Warshall

## 3.4 SSSP

### 3.4.1 Lazy Dijkstra

### 3.4.2 Bellman-Ford

## 3.5 Strongly Connected Components: Kosaraju

## 3.6 Articulation Points and Bridges: ModTarjan

# 4 Math

## 4.1 Identities

Coeficientes binomiales.

$$(a+b)^n = \sum_{k=0}^n \binom{n}{k} a^{n-k} b^k$$

$$\binom{n}{k} = \binom{n}{n-k}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$k \binom{n}{k} = n \binom{n-1}{k-1}$$

$$\sum_{k=0}^n n \binom{n}{k} = 2^n$$

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

$$\binom{n+m}{t} = \sum_{k=0}^t \binom{n}{k} \binom{m}{t-k}$$

$$\sum_{j=k}^n \binom{j}{k} = \binom{n+1}{k+1}$$

Numeros Catalan.

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

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

$$C_n \sim \frac{4^n}{n^{3/2} \sqrt{\pi}}$$

$$\Sigma(n) = O(\log(\log(n))) \text{ (number of divisors of } n)$$

$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$

$$\sum_{i=1}^n F_i = F_{n+2} - 1$$

$$F_{n+i} F_{n+j} - F_n F_{n+i+j} = (-1)^n F_i F_j$$

(Möbius Function)

0 if n is square-free

1 if n got even amount of distinct prime factors

0 if n got odd amount of distinct prime factors

(Möbius Inv. Formula)

Let  $g(n) = \sum_{d|n} f(d)$ , then  $f(n) = \sum_{d|n} d \mu\left(\frac{n}{d}\right)$ .

Permutaciones objetos repetidos

$$P(n, k) = \frac{P(n, k)}{n_1! n_2! \dots}$$

Separadores, Ecuaciones lineares  $a$  variables =  $b$

$$\binom{a}{b} = \binom{a+b-1}{b} = \binom{a+b-1}{a-1}$$

**Teorema chino**

sean  $\{n_1, n_2, \dots, n_k\}$  primos relativos

$$P = n_1 \cdot n_2 \cdot \dots \cdot n_k$$

$$P_i = \frac{P}{n_i}$$

$$x \cong a_1(n_1)$$

$$x \cong a_2(n_2) \dots x \cong a_k(n_k)$$

$P_1 S_1 \cong 1(n_1)$  Donde  $S$  soluciones.

$$x = P_1 S_1 a_1 + P_2 S_2 a_2 \dots P_k S_k a_k$$

## 4.2 Binary Exponentiation and modArith

### 4.3 Modular Inverse (dividir mod)

### 4.4 Modular Binomial Coeficient and Permutations

### 4.5 Non-Mod Binomial Coeficient and Permutations

### 4.6 Modular Catalan Numbers

### 4.7 Ceil Fraccionario

```
1 long long int ceil(long long int numerator, long long int denominator) {
2     return (numerator + denominator - 1) / denominator;
3 }
```

### 4.8 Numeros de Fibonacci

### 4.9 Sieve Of Eratosthenes

```
1 #include <bits/stdc++.h>
2 #define MAX 1000001
3 using namespace std;
4
5 // Define both prime and pfix arrays
6 bool prime[MAX];
7 int pfix[MAX];
8
9 // Sieve of Eratosthenes
10 void sieve() {
11     // Set all numbers as prime
12     memset(prime, true, sizeof(prime));
13     // 0 and 1 are not prime
14     prime[0] = prime[1] = false;
```

```
15 // Iterate over all numbers
16 for (int p = 2; p * p < MAX; p++)
17     if (prime[p]) for (int i = p * p; i < MAX; i += p) prime[i] =
18         false;
19 // Calculate prefix sum of prime numbers
20 for (int i = 2; i < MAX; i++) {pfix[i] = pfix[i - 1] + prime[i];}
21 }
```

## 4.10 Sieve-based Factorization

### 4.11 Cycle Finding

### 4.12 Berlekamp Massey

### 4.13 Modular Berlekamp Massey

### 4.14 Matrix exponentiation

### 4.15 Ecuaciones Diofantinas

### 4.16 Pollard-Rho, Stolen from GGDem

### 4.17 FFT, Stolen from GGDem

### 4.18 Euler Totient Function

## 5 Geometry

## 6 Strings

### 6.1 Explode by token

```
1 vector<string> explode_by_token(string const& s, char delimiter) {
2     vector<string> result;
3     // Create a string stream from the string, allowing to perform input
4     // output operations on strings.
5     stringstream iss(s);
6     // Read the string stream, tokenizing it by the delimiter
7     for(string token; getline(iss, token, delimiter);) {
8         // Split the string by the delimiter and push it to the result
9         vector
10         result.push_back(move(token));
11     }
12     // Return the result vector
13     return result;
```

<sup>12</sup> |}

## 6.2 Multiple Hashings DS

### 6.3 Permute chars of string

### 6.4 Longest common subsequence

### 6.5 KMP

### 6.6 Suffix Array

### 6.7 STL Suffix Array

## 7 Classics

### 7.1 Job scheduling

#### 7.1.1 One machine, linear penalty

#### 7.1.2 One machine, deadlines

#### 7.1.3 One machine, profit

#### 7.1.4 Two machines, min time

## 8 Flow

### 8.1 Dinic, thx GGDem

## 9 Miscellaneous

### 9.1 pbds

### 9.2 Bit Manipulation

## 10 Testing

### 10.1 Gen and AutoRun testcases

#### 10.1.1 Gen.cpp

#### 10.1.2 Stress testing

#### 10.1.3 Autorun

### 10.2 Highly Composite Numbers

Particularly useful when testing number theoretical solutions.