Mini Programming Homework #2

Due Time: 2021/12/02 14:20
Contact TAs: `ada-ta@csie.ntu.edu.tw`

Instructions and Announcements

- There is only **one single programming problem**.
- **Submission.** The judge system is located at https://ada-judge.csie.ntu.edu.tw. Please login and submit your code by the due time. **NO LATE SUBMISSION IS ALLOWED.**
- **Collaboration Policy.** Discussions with others are strongly encouraged. However, you should write your program **on your own**. In addition, you have to specify the references (e.g., the URL of the website you consulted or the people you discussed with) by commenting in the piece of code which you submit. Otherwise, you may get zero points due to the lack of references.
- **Tips for programming problems.** Since the input files for some programming problems may be large, please add
  - `std::ios_base::sync_with_stdio(false);`
  - `std::cin.tie(nullptr);`

  to the beginning of the main function if you are using `std::cin`.

To DFS, or to BFS, that is the question

Problem Description

TL; DR: Given a connected graph with N vertices and M bidirectional edges, please output the lexicographically smallest DFS sequence and the lexicographically smallest BFS sequence.

BB Travel Inc., a traveling agency in the *ADA Kingdom*, would like to launch a tourist plan that visit every city in the kingdom. The *ADA Kingdom* consists of N cities number from 1 to N, where the city numbered 1 is the capital. There are also M **bidirectional** road connecting pairs of cities. You can reach any city from any city using some of the roads.

The tourist plan should be a valid **Depth-First Search (DFS)** or **Breadth-First Search (BFS)** sequence of the kingdom starting from the capital (city 1). Thus, it should be a sequence of cities $s_1, s_2, \ldots, s_N$, where $s_1 = 1$, and each city occurs exactly once. Additionally, to attract more customers, the plan should be as *attractive* as possible. We say plan s is more *attractive* than plan s', if s is **lexicographically smaller** than s'.

Since *BB Travel Inc.* doesn't know whether their customers prefer BFS order or DFS order yet, they ask you to help them find out the plan for both scenarios (i.e. one plan for DFS, and one plan for BFS).

Glossary

Please see the course slides for the definition of *Depth-First Search* and *Breadth-First Search*. Note that for the case of *BFS*, the order is valid as long as we choose an unvisited vertex closest to the starting vertex every time.

A valid DFS/BFS sequence is a sequence s, such that there exists a DFS/BFS search order, where s_i is the i-th vertex added into the DFS/BFS tree.

A sequence s is *lexicographically smaller* than s' if there exists a index t, such that $\forall i < t, s_i = s'_i$ and $s_t < s'_t$.

Input

The first line of the input contains two integers N and M, denoting the number of cities and the number of roads in the *ADA Kingdom*.

The next M lines describes the roads connecting the cities. The i-th line contains two integers u_i and v_i, denoting a bidirectional road connecting city u_i and v_i.

- $1 \leq N \leq 10^5$
- $1 \leq M \leq 2 \times 10^5$
- $1 \leq u_i, v_i \leq N, \forall 1 \leq i \leq M$
- $u_i \neq v_i, \forall 1 \leq i \leq M$
- You can reach any city from any city using some of the roads.
- Every pair of cities is directly connected by at most one edge.

Test Group 0 (0 %)

- Sample Input

Test Group 1 (25 %)

- $N \leq 1000$
- $M \leq 2000$

Test Group 2 (75 %)

- No Additional Constraint

Output

Please output two lines. The first line contains the most *attractive* **DFS** trip plan starting from the capital. The second line contains the most *attractive* **BFS** trip plan starting from the capital.

Sample Input 1

```
5 4
1 2
1 4
4 3
2 5
```

Sample Output 1

```
1 2 5 4 3
1 2 4 3 5
```

Sample Input 2

```
5 7
5 4
3 5
2 3
5 1
1 4
3 4
4 2
```

Sample Output 2

```
1 4 2 3 5
1 4 5 2 3
```

Hint

1. bidirectional = undirected $\neq$ unidirectional
2. Make sure you understand the sample input/outputs before coding.
3. STL is your good firend. For example:
   (a) You can use `std::vector` to store the graph (adjacency list).
   (b) You can use `std::sort` to sort an array/vector/etc.