# Data Structure and Algorithm, Spring 2024
# Mini-Homework I

TA E-mail: dsa_ta@csie.ntu.edu.tw

**Due: 13:00:00, Tuesday, June 11, 2024**

**PLEASE NOTE THE DIFFERENT LATENESS POLICY IN RED BELOW**

## Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources (including chatGPT) can be consulted, but not copied from.
- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.
- In each mini-homework, we will have one programming problem that is closely connected to what we have taught in the class. The purpose of the problem is for you to *practice* what you have learned, before conquering the more challenging problems in other homework sets.
- Like other programming problems, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

  https://dsa2024.csie.org/.

  Each day, you can submit up to 5 times for each problem. The judge system will compile your code with

  ```
  gcc main.c -static -O2 -std=c11
  ```

- For debugging in the programming part, we strongly suggest you produce your own test-cases with programs and/or use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling.

  Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:

  - `gcc main.c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
  - `gcc main.c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$Late\ Score = \max\left(\left(\frac{1 \times 86400 - Delay\ Time(sec.)}{1 \times 86400}\right) \times Original\ Score, 0\right)$$

- If you have questions about this mini-homework, please go to the the Discord channel and discuss the issues with your classmates. This is *strongly preferred* because it will provide everyone with a more interactive learning experience. If you really need an email answer, please follow the rules outlined below to get a fast response:

  - The subject should contain one tag, `"[hwe]"`, specifying the problem where you have questions. For example, `"[hwe] Will there be a repeated number?"`. Adding these tags allows the TAs to track the status of each email and to provide faster responses to you.
  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

## Programming Part

# Graph (20 pts)

## Problem Description

In this problem, we ask you to implement the standard depth first search (DFS) algorithm to traverse a given directed graph $G = \{V, E\}$. Then, according to the results, classify each edge in $E$ to be a tree edge, a back edge, a forward edge, or a cross edge.

The pseudo code of the DFS algorithm is given below, which is identical to the one in Chapter 20.3 of the textbook.

```
DFS(G)
1  for each vertex u ∈ G.V
2       u.color = WHITE
3       u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6       if u.color == WHITE
7            DFS-Visit(G, u)

DFS-Visit(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5       if v.color == WHITE
 6            v.π = u
 7            DFS-Visit(G,v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

In the given graph $G = \{V, E\}$, each vertex in $V$ is identified with an integer between 1 and $|V|$. We denote the vertex with identification number $k$ as vertex $k$.

One additional assumption is that in line 5 of DFS and in line 4 of DFS-Visit, the for loops should iterate over the vertices with their identification numbers in ascending order. For example, in line 5 of DFS, the for loop would iterate with this order: vertex 1, vertex 2, …, vertex $|V|$.

## Input

The first line specifies $|V|$, the number of vertices in $G$.

The next $|V|$ lines specify the adjacency lists of $G$. For the $k$-th line, $1 \leq k \leq |V|$, it starts with an integer specifying the out degree of vertex $k$, $d_k$, followed by a sequence of identifcation numbers of the vertices, with consecutive numbers separated by a single space character, $n_1\ n_2\ \ldots\ n_{d_k}$. This specifies that there exist edges $(k, n_1), (k, n_2), \ldots, (k, n_{d_k})$ in $G$. if the out degree $d_k = 0$, then the line only contains $d_k$ and no other number. You can assume that $n_1\ n_2\ \ldots\ n_{d_k}$ are sorted in ascending order.

## Output

The output should have $|V|$ lines. For the $k$-th line, $1 \leq k \leq |V|$, it starts with the identification number of the vertex, $k$, a space character, followed by a string of length $d_k$ specifying the classification of the edges coming out from vertex $k$. That is, if the $k$-th line of the adjacency list part in the input is $d_k\ n_1\ n_2\ \ldots\ n_{d_k}$, then in the $k$-th line of the output we have $k\ c_1c_2\ldots c_{d_k}$, with the $j$-th character $c_j$ of the string representing the classification of the edge $(k, n_j)$, $c_j \in \{T, B, F, C\}$. Here, $T$ represents a tree edge, $B$ represents a back edge, $F$ represents a forward edge, and $C$ represents a cross edge.

## Constraint

- $1 \leq |V| \leq 1000$
- There are neither self edges nor multiple edges in $G$. In other words, you can assume that $0 \leq |E| \leq |V| \times (|V| - 1)$.

## Sample Testcases

**Sample Input 1**

```
5
2 2 4
1 1
3 2 4 5
2 3 5
0
```

**Sample Output 1**

```
1 TT
2 B
3 CBT
4 TF
5
```

**Sample Input 2**

```
3
2 2 3
2 1 3
2 1 2
```

**Sample Output 2**

```
1 TF
2 BT
3 BB
```