# Homework 4

[2024-2] 데이터사이언스 응용을 위한 컴퓨팅 (001)

**Due: 2024년 12월 8일 23:59**

## 1. Validate Red-Black Tree [50pts]

A Red-Black Tree is a type of binary search tree (BST) that maintains balance and performance guarantees through specific properties. A Red-Black Tree consists of nodes, where each node has the following structure:

```
struct Node {
    int key; // Key value of the node
    bool isRed;// true if RED, false if BLACK
    Node* left; // Pointer to the left child node
    Node* right; // Pointer to the right child node
    Node* parent; // Pointer to the parent node (if applicable)
};
```

Implement the `validateRedBlackTree` function in `prob1.cpp` to verify if a given tree is a valid Red-Black Tree. In a Red-Black Tree, all leaves (represented as `nullptr`) are considered BLACK nodes by definition.

**1.1 Input**:

- `Node* root`: A pointer to the root node of the tree.

**1.2 Output**:

- Returns `true` if the tree is a valid Red-Black Tree, otherwise returns `false`.

**1.3 Example**

```
Node* root = new Node(5, false, nullptr, nullptr, nullptr);
validateRedBlackTree(root);

>> true
```

# 2. Height Order [50pts]

There are n students, each with a unique height. The heights are given in a relationship format where a < b means that student a is shorter than student b. If the height of a is shorter than b, it is represented in the graph as a->b.

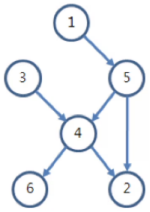


Figure 1

Take a look at Figure 1, which shows a graph illustrating how the heights of six students relate to each other. Student 1 is shorter than Student 5, and Student 5 is shorter than Student 4. Therefore, Student 1 is also shorter than Student 4. By a similar comparison, it can be observed that Students 1, 3, and 5 are all shorter than Student 4, while Students 2 and 6 are taller than Student 4. With the given comparisons, Student 4 is the only one who can clearly identify who is taller and who is shorter than Student 4. All other students, except for Student 4, cannot precisely identify their rank.

Given n students and m height comparisons, find how many students can have their absolute position in the height order known relative to everyone else. Implement the `findKnownOrderStudents` function in `prob2.cpp`.

### 2.1 Input

- `int n`: The number of students, where $2 \leq n \leq 500$.
- `vector<pair<int, int>>& comparisons`: A reference to a vector containing pairs of integers. Each pair `{a, b}` indicates that student a is shorter than student b. This list represents the height comparisons between students.

### 2.2 Output

- Returns an integer representing the number of students whose relative height order can be determined with respect to all other students.

### 2.3 Constraints

- $2 \leq n \leq 500$

- $0 \leq m \leq n(n-1)/2$, where $m$ is the number of comparisons
- $1 \leq a, b \leq n$
- There will be no self-comparisons

### 2.4 Example

```
int n = 6;
vector<pair<int, int>> comparisons = {{1, 5}, {3, 4}, {5, 4}, {4, 2}, {4,
6}, {5, 2}};
findKnownOrderStudents(n, comparisons);
>> 1
```

## Submission Requirements

- For each problem, write your implementation for Problem 1 in `prob1.cpp` and for Problem 2 in `prob2.cpp`. Only `prob1.cpp` and `prob2.cpp` will be graded when you submit to Gradescope. You may modify other files for testing purposes, but keep in mind that only `prob1.cpp` and `prob2.cpp` will be evaluated, using the original header files.
- Grading will be conducted using the C++11 standard. The `main` function in the autograder, which includes test cases, may differ from the `main` function released to students. Compilation will follow this process:
    - `$ g++ -std=c++11 -c main.cpp`
    - `$ g++ -std=c++11 -c prob{no.}.cpp` (using your `prob{no.}.cpp`)
    - `$ g++ -std=c++11 main.o prob{no.}.o -o main`
- Before submitting, remove all debugging code, including any `stdout` print statements, from `prob{no.}.cpp`, except those explicitly required by the problem. Grading will rely solely on `stdout`, and any unrelated output may impact your score.
- Do not share your code with other students. Similarly, students must not use AI tools to generate code directly. High similarity levels in code, as detected by our similarity detection tools, may lead to serious penalties.
- You have 5 grace days for homework submissions throughout the semester, counted in 24-hour increments from the original due time. For instance, if an assignment is due on 12/8 at 11:59 PM, submitting it 30 minutes late will use one grace day. Submitting on 12/10 at 9:00 PM will use two grace days. Late submissions are only accepted if grace days are available.