IFT2015-A25: TP4

November 20, 2025

General Instructions

- **Due date: Wednesday, December 3 at 11:59 PM.**
- This assignment must be completed individually or in teams of two.
- No plagiarism will be accepted.
- Clarifications or modifications to the instructions may be sent by email and posted on Studium.
- Each day of delay in submission will result in a 5-point penalty.
- All code must be written in Java.
- You must submit the required Java files (`Q1.java`, `Q2.java`, and `Q3.java`).
- **If you work in a team:** one person submits the necessary files and the other submits only a `.txt` file with their partner's name.
- Code evaluation may be automated; make sure to have **the same function signatures**.
- You may use `java.util.*` for all problems.
- You may create as many helper functions as you wish.
- **Don't forget to add comments to your implementations. Code without documentation will be penalized.**
- For any questions regarding TP4, please post them on the "TP4 Q&A" forum on Studium.
- Good luck!

1 Problem 1 (5pts) - Dependency Analysis and Cycles in a Graph

Context

You are developing a dependency analysis system for a software package manager. Packages and their dependencies form a directed graph where each node represents a package and each edge (u, v) means that package u depends on package v.

The graph is represented by an adjacency list: `graph[i]` contains the list of packages that package `i` depends on.

Tasks to implement in `Q1.java`

1. `boolean hasCycle(int[][] graph)`
   Determine if the directed graph contains at least one cycle using a depth-first search (DFS).

   Return: `true` if a cycle exists, `false` otherwise.

2. `List<List<Integer» findAllCycles(int[][] graph)`
   Find all simple cycles (without node repetition) in the graph. Each cycle must be represented starting with its smallest node and in ascending order between cycles.

   Example: For a cycle 1→2→3→1, return [1,2,3].

3. `List<List<Integer» stronglyConnectedComponents(int[][] graph)`
   Find all strongly connected components of the graph. Return components sorted by descending size, then by smallest element in case of tie.

4. `int countReachableNodes(int[][] graph, int start)`
   Count the total number of reachable nodes from node `start` (including `start` itself) using DFS.

5. `List<Integer> findBridgeNodes(int[][] graph)`
   A node is a "bridge node" if its removal increase the number of strongly connected components. Return all bridge nodes in ascending order.

6. `Map<Integer, Integer> getFinishingTimes(int[][] graph)`
   Perform a complete DFS and return a Map associating each node with its finishing time. Times start at 1.

   Note: If multiple starting nodes are possible, begin with the smallest unvisited node.

7. `boolean canInstallAll(int[][] graph, List<Integer> broken)`
   Determine if it is possible to install all non-broken packages knowing that certain packages in the `broken` list are unavailable. A package can be installed if all its dependencies (direct and transitive) are available and not broken.

   Return: `true` if all non-broken packages can be installed.

8. `List<Integer> findMinimalDependencySet(int[][] graph, int target)`
   Find the minimal set of packages to install in order to be able to install the `target` package. Return the list in ascending order (do not include `target`).

9. `int longestDependencyChain(int[][] graph)`
   Find the length of the longest dependency chain in the graph. A chain is a simple path (without cycles) in the graph.

   Return: The number of edges in the longest chain, or 0 if the graph is empty.

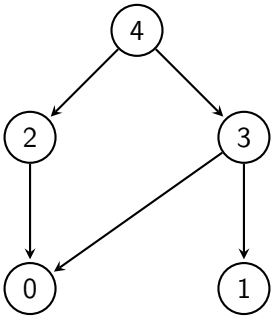
10. `List<Integer> findAllSourceNodes(int[][] graph)`
    Find all source nodes (nodes without incoming dependencies). Return in ascending order.

    Note: A source node is a node that does not appear in any adjacency list.

Example executions

Example 1 - Acyclic graph:

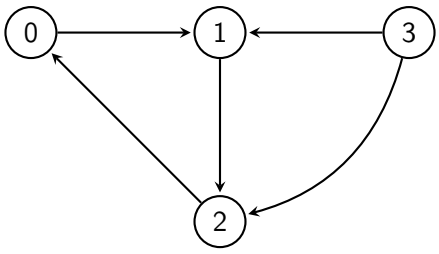
```
graph = [[],[],[0],[0,1],[2,3]]
```




```
hasCycle(graph) -> false

stronglyConnectedComponents(graph) -> [[4], [3], [2], [1], [0]]

countReachableNodes(graph, 4) -> 5

findBridgeNodes(graph) -> [1, 2, 3, 4]

longestDependencyChain(graph) -> 3

findMinimalDependencySet(graph, 4) -> [0, 1, 2, 3]

findAllSourceNodes(graph) -> [4]
```

Example 2 - Graph with cycle:

```
graph = [[1],[2],[0],[1,2]]
```




```
hasCycle(graph) -> true

findAllCycles(graph) -> [[0,1,2]]

stronglyConnectedComponents(graph) -> [[0,1,2], [3]]

countReachableNodes(graph, 3) -> 4
```

```
getFinishingTimes(graph) -> {0:6, 1:5, 2:4, 3:8}

canInstallAll(graph, [1]) -> false

findAllSourceNodes(graph) -> [3]
```

Example 3 - Complex components:

```
graph = [[1],[2],[0],[4],[3,5],[6],[5]]
```


```
stronglyConnectedComponents(graph) -> [[0,1,2], [5,6], [3,4]]

findBridgeNodes(graph) -> []

longestDependencyChain(graph) -> 4

findAllSourceNodes(graph) -> []
```

2 Problem 2 (5pts) - Red-Black Tree for Intelligent Cache

Context

You are developing an intelligent cache system for a database. The cache stores key-value pairs and must maintain order to allow efficient range queries. To guarantee optimal performance even in the worst case, you use a **red-black tree**.

Each node contains:

- `int key`: The key (unique integer)
- `String value`: The associated value
- `boolean isRed`: `true` if red, `false` if black
- `int accessCount`: Number of accesses to this entry
- `long lastAccessTime`: Timestamp of last access

Tasks to implement in `Q2.java`

1. `void insert(int key, String value, long timestamp)`
   Insert a new key-value pair. Initialize `accessCount` to 1 and `lastAccessTime` to the provided timestamp.

2. `String get(int key, long timestamp)`
   Search for and return the value. If found: increment `accessCount` and update `lastAccessTime`. Return `null` if not found.

3. `boolean delete(int key)`
   Delete a node while respecting RB properties. Return `true` if deleted.

4. `List<String> getRangeValues(int minKey, int maxKey)`
   Return all values in [minKey, maxKey] in ascending order.

5. `int getBlackHeight()`
   Calculate and return the black height of the tree (black nodes on a root-to-leaf path).

6. `boolean verifyProperties()`
   Verify that the tree respects all RB properties.

7. `List<Integer> getMostAccessedKeys(int k)`
   Return the k most accessed keys in descending order.

   Example : If accessCount: {15:2, 25:3, 10:1, 20:1, 30:1}, top 3 = [25, 15, 10]

8. `void evictOldEntries(long currentTime, long maxAge)`
   Delete all entries where `currentTime - lastAccessTime > maxAge`.

9. `int countRedNodes()`
   Count the total number of red nodes.

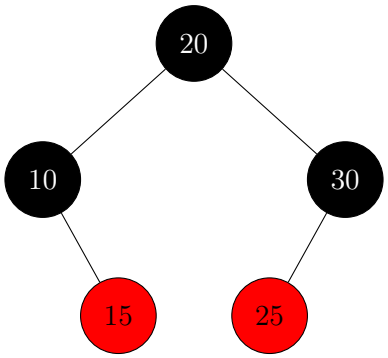
10. `Map<String, Integer> getColorStatisticsByLevel()`
    Return a Map: level → number of red nodes at this level.

Example execution

Initial insertions (t=0):

```
insert(10, "data10", 0)
insert(20, "data20", 0)
insert(30, "data30", 0)
insert(15, "data15", 0)
insert(25, "data25", 0)
```

Resulting tree (R=red, B=black):



Some operations:

```
get(15, 100) -> "data15"  (accessCount: 1->2, lastAccess: 0->100)

getRangeValues(12, 27) -> ["data15", "data20", "data25"]

getBlackHeight() -> 2

verifyProperties() -> true

countRedNodes() -> 2  (nodes 15 and 25)

getColorStatisticsByLevel() -> {"0": 0, "1": 0, "2": 2}

evictOldEntries(300, 150)  // Deletes everything except 25 (lastAccess=200)

delete(20) -> true  (the root is deleted and replaced)
```

3 Problem 3 (5pts) - Cat and Mouse Game

Context

You are developing an analysis engine for a two-player game on an undirected graph. The Mouse and the Cat move alternately on the nodes of the graph. The game ends when one of the players wins or there is a draw.

Game rules:

- The Mouse starts at node 1 and plays first
- The Cat starts at node 2 and plays second
- Node 0 is the refuge (forbidden to the Cat)

- Each turn, the active player must move to an adjacent node
- **Cat wins** if it occupies the same node as the Mouse
- **Mouse wins** if it reaches node 0 (refuge)
- **Draw** if the game exceeds a limit number of moves
- Both players play optimally (maximize their chances of winning)

The graph is represented by an adjacency list: `graph[i]` contains the list of accessible nodes from node `i`.

Tasks to implement in `Q3.java`

1. `int gameResult(int[][] graph)`
   Determine the game result with optimal play from both sides.

   Return: 1 if the Mouse wins, 2 if the Cat wins, 0 if draw.

   Hint: Use a recursive approach exploring all possible moves. Suggested move limit: $4 \times n + 200$ *where* n *is the number of nodes.*

2. `List<Integer> getMouseWinningMoves(int[][] graph)`
   If the Mouse can win from the initial state, return all first moves that lead to victory (in scending order). Otherwise, return an empty list.

3. `int minMovesToWin(int[][] graph, int player)`
   If the specified player (1=Mouse, 2=Cat) can force a win, return the minimum number of moves needed to win with optimal play. Return -1 if impossible.

   Note: A "move" = one player's displacement.

4. `int gameResultFrom(int[][] graph, int mousePos, int catPos, boolean mouseTurn)`
   Determine the game result from an arbitrary position. The rules remain the same but starting positions change.

5. `List<String> getAllDrawPositions(int[][] graph)`
   Return all positions (m, c, t) that lead to a draw, formatted as "m-c-t" where $t = 0$ (mouse turn) or 1 (cat turn), in lexicographic order.

6. `Map<String, Integer> analyzeAllPositions(int[][] graph)`
   For each valid position (m, c, t) where $m \neq c$, $c \neq 0$, calculate the optimal result. Return a Map: "m-c-t" $\rightarrow$ result (0/1/2).

7. `int countWinningPositions(int[][] graph, int player)`
   Count the total number of positions (m, c, t) where the specified player has a guaranteed winning strategy.

8. `List<Integer> findSafeNodes(int[][] graph)`
   Return all nodes (except 0) where if the Mouse starts its turn at this node and the Cat is at any other valid position, the Mouse can always force a win. Return in ascending order.

Theoretical questions (answer in comments):

a) Explain why a limit on the number of moves is necessary to detect draws. What would happen without this limit?

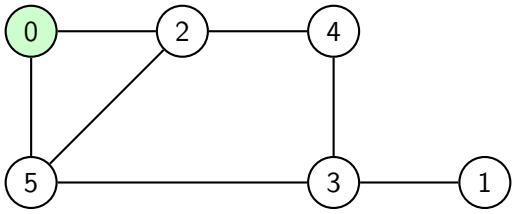
b) What is the time complexity of your `gameResult` algorithm as a function of n (number of nodes) and m (number of edges)? Justify.

c) In which case is the game always won by the Mouse, regardless of the graph structure? Prove your answer.

Example executions

Example 1:

```
graph = [[2,5],[3],[0,4,5],[1,4,5],[2,3],[0,2,3]]
```

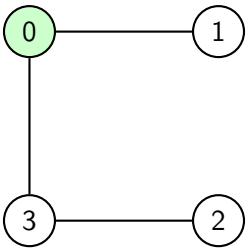


Note: Node 0 (in green) is the Mouse's refuge

```
gameResult(graph) -> 0 (draw)

getMouseWinningMoves(graph) -> [] (mouse cannot win)

minMovesToWin(graph, 1) -> -1
minMovesToWin(graph, 2) -> -1

findSafeNodes(graph) -> [] (no always safe node)
```

Example 2:

```
graph = [[1,3],[0],[3],[0,2]]
```



Note: Node 0 (in green) is the Mouse's refuge

```
gameResult(graph) -> 1 (mouse wins)

getMouseWinningMoves(graph) -> [0] (go directly to refuge)

minMovesToWin(graph, 1) -> 1 (only one move to win)

gameResultFrom(graph, 1, 3, true) -> 1 (mouse wins)
gameResultFrom(graph, 3, 2, false) -> 2 (cat captures in 1 move)

findSafeNodes(graph) -> [1] (from node 1, mouse always wins)
```

Academic Integrity Attestation

Mandatory Declaration

For each problem solved in this assignment, you must include in your code comments an attestation regarding the use of generative artificial intelligence tools.

IMPORTANT: You are not allowed to have code generated and simply submit it, this is **PLAGIARISM**, the use of AI is only authorized to help you understand either the logic, make comments...

Required Format

At the beginning of each file (`Q2.java` and `Q3.java`), add the following comment:

```
/*
* ACADEMIC INTEGRITY ATTESTATION
*
* [ ] I certify that I have not used any generative AI tools
*     to solve this problem.
*
* [ ] I have used one or more generative AI tools.
*     Details below:
*
*     Tool(s) used: _______________________________
*
*     Reason(s) for use:
*     ____________________________________________________
*     ____________________________________________________
*
*     Affected code sections:
*     ____________________________________________________
*     ____________________________________________________
*/
```

Instructions

- Check the appropriate box by replacing `[ ]` with `[X]`.
- If you used an AI tool, you must fill in all fields.
- AI tools include (but are not limited to): ChatGPT, Claude, GitHub Copilot, Bard, etc.
- **Failure to comply with this requirement or any false declaration will be considered a violation of the academic honor code.**
- The use of AI must be justified and code sections generated must be clearly identified.