Trees, Heaps, and Graphs : Exercises

Prof. Dr. Thomas Schuster

Winter Semester 2024/25

Please solve the following exercises. Use Python for coding tasks and submit the solutions in Moodle. Collaborate in small groups for group exercises and ensure your submission reflects your group's effort.

1. (10 points) **Binary Search Tree (BST) Implementation**

   Implement a class `BinarySearchTree` in Python that supports the following operations:

   - `insert(data)`: Inserts a node with the specified data.
   - `search(key)`: Searches for a node with the given key.
   - `delete(key)`: Deletes the node with the given key.
   - `in_order_traversal()`: Returns the in-order traversal of the tree.

   Provide a test case using pytest to verify your implementation.

2. (10 points) **Min-Heap Implementation with Python's `heapq` Module**

   Using Python's `heapq` module, implement a min-heap that supports the following operations:

   - `heappush(item)`: Inserts an item into the heap.
   - `heappop()`: Removes and returns the smallest item from the heap.
   - `change_priority(item)`: Change the priority of an item in the heap.

   Provide a Python code snippet that demonstrates the functionality of the min-heap.

3. (15 points) **Implement BFS and DFS on a Graph**

   Implement both Breadth-First Search (BFS) and Depth-First Search (DFS) on an undirected graph using Python. Demonstrate their differences by applying them to the following graph:

   Graph: A–B–C–D, A–E–F, F–G

   Provide the Python code and the output of both algorithms when starting from node ‘A‘.

4. (20 points) **Implement Dijkstra's Algorithm for Shortest Path**

   Implement Dijkstra's algorithm in Python to find the shortest path from a starting node to all other nodes in a weighted graph. Apply it to the following graph:

   Graph: A–(2)–B–(5)–C, A–(1)–D–(2)–C

   Provide the Python code and output.