|  |
| --- |
| **Software Engineering Department - ITU** |
| **SE200L: Data Structures & Algorithms Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 20/11/2024** |
| **Teaching Assistant: Zainab Bashir & Ryan Naveed** | **Semester: Fall 2024** |
| **Lab Engineer: Sadia Ijaz** | **Batch: BSSE2023B** |

# **Lab 12. Fibonacci Heap**

|  |  |  |
| --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 35)** |
| Muhammad Mukarram Raza | BSSE-23029-B |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to practice problems related to implementation of Prim's Algorithm.

## **Equipment and Component**

|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **1.4 Theory and Background**

A **Fibonacci Heap** is an advanced data structure used for efficiently managing a priority queue. It is based on a collection of trees, which are maintained using a lazy approach to improve the amortized time complexity of certain operations. Fibonacci Heaps are particularly useful in graph algorithms like **Dijkstra’s shortest path** and **Prim’s minimum spanning tree**.

**Key Characteristics:**

1. **Structure**: A Fibonacci Heap consists of a collection of trees, each following the **min-heap property**, where the key of a parent is less than or equal to the keys of its children.
2. **Efficiency**: Fibonacci Heaps support:
   * **Insert**: O(1) amortized.
   * **Find Min**: O(1) amortized.
   * **Extract Min**: O(logn) amortized.
   * **Decrease Key**: O(1) amortized.
   * **Delete**: O(logn) amortized.
3. **Lazy Merging**: Nodes are only reorganized when absolutely necessary, such as during extractMin or decreaseKey.

**Applications:**

* Graph algorithms requiring efficient priority queue operations.
* Dynamic algorithms where frequent updates are needed.

.

## **Lab Tasks**

**Class Definitions:**

**1. FibonacciNode**

This class represents a single node in the Fibonacci Heap.

**Attributes**

* **int key:** The key value stored in the node.
* **int degree:** The number of children this node currently has.
* **FibonacciNode\* parent:** Pointer to the parent node (if any).
* **FibonacciNode\* child:** Pointer to one of the children of this node (if any).
* **FibonacciNode\* left:** Pointer to the left sibling in the circular doubly-linked list.
* **FibonacciNode\* right:** Pointer to the right sibling in the circular doubly-linked list.
* **bool mark:** Indicates whether this node has lost a child (true = lost, false = not lost).
* **char C:** Temporary marker for node search.

**Methods**

* **FibonacciNode(int key):** Constructor to initialize a FibonacciNode with the given key.
* **void link(FibonacciNode\* child):** Links a child node to the current node, updating degrees.
* **int getKey():** Returns the key value

**2. FibonacciHeap**

This class represents the Fibonacci Heap, which is a collection of FibonacciNodes.

**Attributes**

* **FibonacciNode\* minNode:** Pointer to the node with the minimum key in the heap.
* **int nodeCount:** The total number of nodes in the heap.
* **FibonacciNode\* rootList:** Pointer to the circular doubly-linked list containing the heap’s root nodes.

**Methods**

* **FibonacciHeap():** Constructor to initialize an empty Fibonacci Heap.
* **FibonacciNode\* insert(int key):** Inserts a new node with the given key into the heap and returns a pointer to it.
* **FibonacciNode\* findMin():** Returns a pointer to the node with the minimum key in the heap.
* **FibonacciNode\* extractMin():** Removes and returns a pointer to the node with the minimum key.
* **void decreaseKey(FibonacciNode\* node, int newKey):** Decreases the key value of a node and ensures the heap property is maintained.
* **void deleteNode(FibonacciNode\* node):** Deletes a node from the heap.
* **void unionHeap(FibonacciHeap\* otherHeap):** Merges the current heap with another Fibonacci Heap.
* **void consolidate():** Combines trees in the root list to ensure each tree has a unique degree.
* **void cut(FibonacciNode\* node, FibonacciNode\* parent):** Cuts a node from its parent and adds it to the root list.
* **void cascadingCut(FibonacciNode\* node):** Performs cascading cuts to restore the heap property after a decrease\_key operation.

*Please read the following instructions carefully:*

1. ***Do Not Modify test.cpp:*** *You are strictly prohibited from making any changes to the test.cpp file. This file is designed to test your implementation and any modifications will lead to the assignment being graded as zero.*
2. ***Class Definitions:*** *All class definitions and implementations must be provided solely within the files functions.h and functions.cpp. You are not allowed to create any additional files for your class definitions or implementations.*

*Any deviation from these rules, including creating additional files or modifying the test.cpp file, will result in your assignment receiving a grade of zero.*

**Assessment Rubric for Lab**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Performance metric** | **CLO** | **Able to complete the task over 80% (4-5)** | **Able to complete the task 50-80% (2-3)** | **Able to complete the task below 50% (0-1)** | **Marks** |
| 1. Realization of experiment | 1 | Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed. | Executes without errors, user prompts are understandable,minimum use of symbols or spacing in output. Some testing has been completed. | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed. |  |
| 2. Conducting experiment | 1 | Able to make changes and answer all questions. | Partially able to make changes and few incorrect answers. | Unable to make changes and answer all questions. |  |
| 3. Computer use | 2 | Document submission timely. | Document submission late. | Document submission not done. |  |
| 4. Teamwork | 3 | Actively engages and cooperates with other group member(s) in an effective manner. | Cooperates with other group member(s) in a reasonable manner but conduct can be improved. | Distracts or discourages other group members from conducting the experiment |  |
| 5. Laboratory safety and disciplinary rules | 3 | Code comments are added and do help the reader to understand the code. | Code comments are added and do not help the reader to understand the code. | Code comments are not added. |  |
| 6. Data collection | 3 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap. | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables. | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy. |  |
| 7. Data analysis | 4 | Solution is efficient, easy to understand, and maintain. | A logical solution that is easy to follow but it is not the most efficient. | A difficult and inefficient solution. |  |
| **Total (out of 35):** | | | | |  |