Sabanci University
Faculty of Engineering and Natural Sciences

CS 300 Data Structures
Homework 3: HEAPS

Assigned: April 15, 2025
Due: April 30, 2025

Introduction

SwiftLogiTech, a global logistics company, manages the dispatch of autonomous drones, vehicles, and warehouse bots. These systems handle millions of dynamically prioritized tasks such as deliveries, refueling, rerouting, and emergency support.
In this assignment, your job is to build a modular task scheduling system using two different heap-based priority queue algorithms. You will simulate task management, priority updates, and merging of task queues using deterministic input and output formats suitable for automated testing. The algorithms you'll use are:

- Binomial Heap
- D-ary Heap

Binomial Heap

A **Binomial Heap** is a data structure composed of a forest of binomial trees, where each tree follows the *min-heap property*. It is structured such that a tree of order k has 2^k nodes, and no two trees in the heap share the same order. This structure allows the heap to support an efficient merge operation in $O(\log n)$ time.
Binomial Heaps are well-suited for applications where *combining task queues from different sources* happens frequently. For example, when two logistics hubs need to synchronize their scheduling queues, Binomial Heaps allow this to be done efficiently.
Think of a Binomial Heap as a group of well-organized teams (trees), each responsible for a different size of task group. When two such groups need to join, they can quickly align and merge their structures without starting from scratch.

Operations:

- `void insert(int task_id, int priority);`
  Inserts a new task with the given ID and priority.

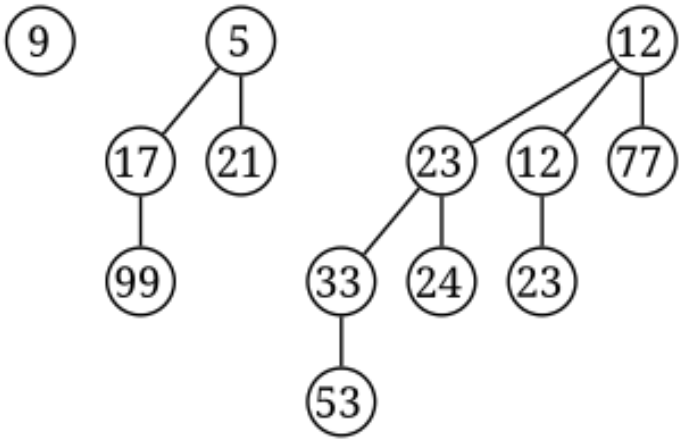

Figure 1: Example of a binomial heap containing 13 nodes with distinct keys. The heap consists of three binomial trees with orders 0, 2, and 3.

- `int extract_min();`
  Removes and returns the task_id with the lowest priority. If the heap is empty, return `-1`.

- `void decrease_key(int task_id, int new_priority);`
  Decreases the priority of the task with the given ID. If task_id is not found, do nothing.

- `void merge_with(BinomialHeap& other);`
  Merges the current heap with another binomial heap. After the call, `other` may be empty or unchanged.

- `void print_heap() const;`
  Prints all tasks in the heap in a readable format, including task IDs and priorities.

D-ary Heap

A **D-ary Heap** generalizes the Binary Heap by allowing each node to have d children instead of just 2. It is implemented as an array-based min-heap and performs well in systems where tasks are inserted or updated frequently.

Increasing d reduces tree height (faster insert/decrease), but may increase the cost of finding the minimum during extraction.

Think of it like a manager who supervises more team members. The team works faster for new tasks or changes in priority, but reviewing everyone for the best performer takes slightly more time.

Operations:

- `DaryHeap(int d);`
  Constructor that initializes the heap with the specified number of children per node. The value of `d` must be ≥ 2.

- `void insert(int task_id, int priority);`
  Inserts a new task with the given ID and priority into the heap.
- `int extract_min();`
  Removes and returns the task ID with the lowest priority value. If the heap is empty, return `-1`.
- `void decrease_key(int task_id, int new_priority);`
  Decreases the priority of the specified task. If `task_id` is not found, the function should do nothing.
- `void merge_with(DaryHeap& other);`
  Merges another DaryHeap into the current heap. You may implement this by inserting all tasks from `other` into `this` heap.
- `void print_heap() const;`
  Prints the task IDs and priorities in the heap in array order or any clearly readable structure.

Functional Requirements

In this assignment, you are required to implement the following components from scratch:

1. Abstract Base Class: `Heap`

Define an abstract base class named `Heap` that outlines the interface for all heap types. Both `BinomialHeap` and `DaryHeap` must inherit from this class and implement all methods.

```
class Heap {
public:
    virtual void insert(int task_id, int priority) = 0;
    virtual int extract_min() = 0;
    virtual void decrease_key(int task_id, int new_priority) = 0;
    virtual void merge_with(Heap& other) = 0;
    virtual void print_heap() const = 0;
    virtual ~Heap() = default;
};
```

2. Heap Implementations

a) Binomial Heap

Create a class `BinomialHeap` that inherits from `Heap`. It must include the following public member functions:

```
class BinomialHeap : public Heap {
public:
    void insert(int task_id, int priority) override;
    int extract_min() override;
    void decrease_key(int task_id, int new_priority) override;
    void merge_with(BinomialHeap& other);
    void print_heap() const override;
};
```

b) D-ary Heap

Create a class `DaryHeap` that also inherits from `Heap`, and accepts the branching factor `d` as a constructor parameter:

```
class DaryHeap : public Heap {
public:
    DaryHeap(int d);
    void insert(int task_id, int priority) override;
    int extract_min() override;
    void decrease_key(int task_id, int new_priority) override;
    void merge_with(DaryHeap& other);
    void print_heap() const override;
};
```

NOTE: The above mentioned methods are mandatory to be implemented. For the efficient execution of the data structure, if any other functions are required, you are allowed to add them. There is no restriction.

3. Task Class

Implement a class named `Task` to represent tasks in the heap. The class must include the following fields:

```
class Task {
public:
    int task_id;
    int priority;
    std::string location;
    std::string category;

    Task(int id, int prio, std::string loc = "", std::string cat = "");
};
```

Operator overloads and custom comparators may be added if required for your heap implementation.

4. Task ID Mapping for `decrease_key()`

Each heap must maintain a mapping from `task_id` to the corresponding node or index in the heap. This is necessary for efficient execution of the `decrease_key()` operation.
Use a hash map such as:

```
std::unordered_map<int, Node*> task_map;
```

If the given `task_id` is not found in the heap during a `decrease_key()` call, the function should return silently without an error.

5. Command Parser

Write a parser that reads task scheduling commands from a plain text input file and routes them to the heap instance. The following commands must be supported:

```
HEAP_TYPE BINOMIAL
HEAP_TYPE DARY <d>
INSERT <task_id> <priority>
DECREASE_KEY <task_id> <new_priority>
EXTRACT_MIN
MERGE <filename>
PRINT
```

The parser must handle all commands in order and support merging from an external input file using the same format.

Sample Input and Output

Input File: `sample_input_1.txt`

```
HEAP_TYPE BINOMIAL
INSERT 100 50
INSERT 101 20
INSERT 102 10
DECREASE_KEY 100 5
EXTRACT_MIN
PRINT
MERGE other_heap.txt
EXTRACT_MIN
```

Input File: `other_heap.txt`

```
HEAP_TYPE BINOMIAL
INSERT 200 15
INSERT 201 30
INSERT 202 25
```

Expected Output: `output_1.txt`

```
Extracted: 100
Heap:
TaskID: 102, Priority: 10
TaskID: 101, Priority: 20
Extracted: 102
```

For the implementation of the D-ary Heap, please modify the input files by replacing every instance of the word BINOMIAL with DARY (in all capital letters). All other content in the input and output files should remain unchanged. You do not need to create any additional sample files. To test the D-ary Heap implementation, you may use the modified versions of sample_input_1.txt and other_heap.txt.

Project Structure

```
heap_scheduler/
 src/
    main.cpp
    Task.h
    Heap.h
    BinomialHeap.h
    BinomialHeap.cpp
    DaryHeap.h
    DaryHeap.cpp
    InputParser.cpp
 test_inputs/
    sample_input_1.txt
    other_heap.txt
 expected_outputs/
    output_1.txt
 Makefile
```

Notes:

- All source files must be under `src/`.
- Your parser must support running with a command like:

```
./heap_scheduler test_inputs/sample_input_1.txt
```