Heaps & Shortest Path

Performance Analysis of Binary Heap, Fibonacci Heap, and Hollow Heap in Dynamic Shortest Path Problems

---

Overview

This project is designed as a **comparative case study** to analyze and evaluate the performance of three advanced heap data structures — **Binary Heap**, **Fibonacci Heap**, and **Hollow Heap** — when used as priority queues within **Dijkstra's for all pairs shortest path algorithm**. Students will implement all three heap structures from scratch, integrate them into Dijkstra's algorithm, and analyze runtime, operation performance, and structural properties (e.g., tree height, number of nodes, consolidation behavior).

The study also includes a large-scale performance test with graph datasets of varying densities and sizes.

Learning Objectives

- Implement three priority queue data structures: Binary Heap, Fibonacci Heap, and Hollow Heap.
- Understand and analyze differences in theoretical and empirical performance.
- Integrate these heaps into Dijkstra's algorithm and simulate real-world dynamic routing conditions.
- Evaluate performance for insert, extract-min, and decrease-key operations.
- Record and visualize heap properties: height, number of trees, and consolidation patterns.
- Produce analytical insights comparing the three implementations.

Case Study Background

A **smart logistics company** maintains an automated routing system that computes shortest delivery paths for thousands of vehicles. Edge weights (travel times) change dynamically due to real-time traffic conditions, requiring frequent updates to the shortest paths.

Since Dijkstra's algorithm relies heavily on priority queue operations — especially decrease-key when edge weights are reduced — selecting the most efficient heap structure can drastically affect performance. The company's engineers want to determine which heap provides the best trade-off between speed and memory efficiency under different workloads.

Tasks:

1. Implement Binary, Fibonacci, and Hollow heaps from scratch.
2. Use each heap in Dijkstra's algorithm for shortest path computation.
3. Run the algorithm on multiple graph datasets.
4. Record operation times and heap structural statistics.
5. Compare results and provide an analytical report.

Implementation Requirements

Heap Implementations

Each heap must support the following operations:

- insert(key, value)
- find_min()
- extract_min()
- decrease_key(node, new_key)
- delete(node) (optional for Binary Heap)

Integration

- Implement Dijkstra's algorithm using a generic PriorityQueue interface.
- Plug in Binary, Fibonacci, and Hollow heap implementations without changing algorithm logic.

Performance Measurement

Measure and report execution times for:

- Total Dijkstra runs time per heap.
- Average time for each heap operation (insert, extract_min, decrease_key).

Additionally, for Fibonacci and Hollow Heaps, record:

- Heap height after all insertions.
- Number of trees in the root list (for Fibonacci and Hollow heaps).
- Number of cascading cuts (for Fibonacci heap).

Metrics to Record

Metric	Binary Heap	Fibonacci Heap	Hollow Heap
Insert Time (avg)			
Extract-Min Time (avg)			
Decrease-Key Time (avg)			
Total Runtime (ms)			
Heap Height			
Number of Trees			
Memory Usage (MB)			

Sample Experiments

1. **Experiment A — Static Routing:** Run Dijkstra once on each graph using all three heaps; record total runtime.
2. **Experiment B — Operation Profiling:** Randomly generate 100,000 priority queue operations and measure time per operation.

Example Output (Sample Table)

Heap Type	Nodes	Edges	Insert Time (μs)	Extract-Min Time (μs)	Decrease-Key Time (μs)	Total Runtime (s)	Height - N	#Trees	Memory Usage
Binary	10000	190900	1.2	3.1	4.0	0.002	100	1000	10 MB
Fibonacci	10000	190900	1.0	2.1	0.9	0.001	19	800	9 MB
Hollow	10000	190900	1.3	2.5	0.8	0.0015	12	659	7 MB

Code Requirement:

You are provided with three different graph datasets. When the program runs, it should prompt the user to select one of the dataset files. Once the user selects a file, the system should process the graph data and generate the output in the required format. Additionally, the system must create a text file that contains the same output table.

Deliverables

1. Source code for all three heap implementations.
2. A test suite with correctness and performance tests.
3. Performance report with tables and plots.
4. A final analysis comparing performance across graph sizes and heap types.

Report Outline

1. Theoretical Overview of Heaps
    - Binary Heap
    - Fibonacci Heap
    - Hollow Heap
2. Implementation Details
3. Experimental Setup & Test Data
4. Results & Analysis
    - Operation Timing Comparison
    - Heap Structure Statistics
    - Trade-off Discussion
5. Conclusion & Recommendations

Bonus Challenge

- Implement parallelized Dijkstra (multi-source or multi-threaded) and compare heap scalability.
- Add visualization of heap evolution (tree height, number of roots) during Dijkstra runs.