Student Name: Ganesh Nathan
Student ID: 2021BCS0104

ICS225 Data Strucutres-III
Lab - 4

---

1 Problem statement

Implement the binomial heap data structure with following operations
(1) Insertion
(2) FindMin
(3) Union
(4) Extract min

2 Solution Description

A Binomial Heap is a data structure that consists of a collection of Binomial Trees. Each Binomial Tree follows the heap property, where the key of a node is greater than or equal to the keys of its children. Binomial Heaps are based on the binomial tree structure, which is a set of binomial trees with specific properties.
The operations supported by a Binomial Heap include:
(1) **Insertion**: To add an element to the heap, it is first created as a single-node Binomial Heap. Then, it is merged with the existing Binomial Heap using a merge operation. This merge operation combines two Binomial Heaps of the same order into a new Binomial Heap of the next order.
(2) **Deletion**: To remove an element from the heap, the node to be deleted is located and its key is decreased to a minimum value. This operation transforms the Binomial Heap into two separate Binomial Heaps. Then, the two heaps are merged together to form a new Binomial Heap.
(3) **Extract Minimum**: The minimum value in the Binomial Heap is always found in the root of one of the Binomial Trees. To extract this minimum value, the root with the minimum key is located and removed from its Binomial Tree. Then, the remaining Binomial Trees are merged together to form a new Binomial Heap.
(4) **Union**: The union operation combines two Binomial Heaps into a single Binomial Heap. It involves merging the Binomial Trees of the same order from both heaps and maintaining the heap property.
(5) **Decrease Key**: The decrease key operation decreases the key of a node in the Binomial Heap. It is performed by locating the node, decreasing its key, and then applying a series of swaps to maintain the heap property.
(6) **Delete**: The delete operation removes a specific node from the Binomial Heap. It involves decreasing the key of the node to the minimum value and then performing the extract minimum operation to remove it from the heap.

3 Algorithm

3.0.1 Operations

Insertion

- Create a new node with the given value.
- Create a new binomial heap with the new node.
- Merge the new heap with the current heap.

FindMin

- If the heap is empty, return an error.
- Return the value of the root node in the heap.

Union

- Merge two binomial heaps by comparing and linking their trees based on their degrees.

Extract Min

- Find the minimum node in the heap.
- Remove the minimum node from the heap.
- Create a new binomial heap with the children of the minimum node.
- Merge the new heap with the current heap.

4 Implementation

```cpp
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

// Structure for a binomial tree node
struct BinomialTreeNode {
    int value;
    int degree;
    BinomialTreeNode* parent;
    BinomialTreeNode* child;
    BinomialTreeNode* sibling;
};

// Custom comparison function for the priority_queue
struct CompareDegree {
    bool operator()(const BinomialTreeNode* a, const BinomialTreeNode* b) const {
        return a->degree < b->degree;
    }
};

class BinomialHeap {
private:
    priority_queue<BinomialTreeNode*, vector<BinomialTreeNode*>, CompareDegree> heap;

```

```
    // Merge two binomial trees of the same degree
    BinomialTreeNode* mergeTrees(BinomialTreeNode* tree1, BinomialTreeNode* tree2) {
        if (tree1->value > tree2->value)
            swap(tree1, tree2);

        tree2->parent = tree1;
        tree2->sibling = tree1->child;
        tree1->child = tree2;
        tree1->degree++;

        return tree1;
    }

    // Merge two binomial heaps
    void mergeHeaps(BinomialHeap& other) {
        priority_queue<BinomialTreeNode*, vector<BinomialTreeNode*>, CompareDegree> mergedHeap;

        BinomialTreeNode* tree1 = nullptr;
        BinomialTreeNode* tree2 = nullptr;
        BinomialTreeNode* tree3 = nullptr;
        BinomialTreeNode* carry = nullptr;

        while (!heap.empty() || !other.heap.empty() || carry != nullptr) {
            if (carry != nullptr) {
                mergedHeap.push(carry);
                carry = nullptr;
            }

            if (!heap.empty()) {
                tree1 = heap.top();
                heap.pop();
            }
            else {
                tree1 = nullptr;
            }

            if (!other.heap.empty()) {
                tree2 = other.heap.top();
                other.heap.pop();
            }
            else {
                tree2 = nullptr;
            }

            if (tree1 != nullptr && tree2 != nullptr) {
                if (tree1->degree < tree2->degree) {
                    tree3 = tree1;
                    tree1 = tree1->sibling;
                }
                else if (tree1->degree > tree2->degree) {
                    tree3 = tree2;
                    tree2 = tree2->sibling;
                }
```

```cpp
                else {
                    carry = mergeTrees(tree1, tree2);
                    tree1 = nullptr;
                    tree2 = nullptr;
                }
            }
            else if (tree1 != nullptr) {
                tree3 = tree1;
                tree1 = tree1->sibling;
            }
            else {
                tree3 = tree2;
                tree2 = tree2->sibling;
            }

            if (tree3 != nullptr)
                mergedHeap.push(tree3);
        }

        heap = mergedHeap;
    }

    // Extract the minimum node from the binomial heap
    BinomialTreeNode* extractMinNode() {
        if (heap.empty())
            return nullptr;

        BinomialTreeNode* minNode = heap.top();
        heap.pop();

        BinomialTreeNode* child = minNode->child;
        BinomialTreeNode* prev = nullptr;
        BinomialTreeNode* next = nullptr;

        while (child != nullptr) {
            next = child->sibling;
            child->sibling = prev;
            child->parent = nullptr;
            prev = child;
            child = next;
        }

        BinomialHeap childHeap;
        childHeap.heap = priority_queue<BinomialTreeNode*, vector<BinomialTreeNode*>,
            CompareDegree>(prev, nullptr);

        mergeHeaps(childHeap);

        return minNode;
    }

public:
    // Insert a new element into the binomial heap
    void insert(int value) {
```

```cpp
        BinomialTreeNode* newNode = new BinomialTreeNode();
        newNode->value = value;
        newNode->degree = 0;
        newNode->parent = nullptr;
        newNode->child = nullptr;
        newNode->sibling = nullptr;

        BinomialHeap newHeap;
        newHeap.heap = priority_queue<BinomialTreeNode*, vector<BinomialTreeNode*>,
            CompareDegree>({newNode});

        mergeHeaps(newHeap);
    }

    // Find the minimum element in the binomial heap
    int findMin() {
        if (heap.empty())
            return -1;

        return heap.top()->value;
    }

    // Union two binomial heaps
    void unionWith(BinomialHeap& other) {
        mergeHeaps(other);
    }

    // Extract the minimum element from the binomial heap
    int extractMin() {
        BinomialTreeNode* minNode = extractMinNode();

        if (minNode == nullptr)
            return -1;

        int minValue = minNode->value;
        delete minNode;

        return minValue;
    }
};

int main() {
    BinomialHeap binomialHeap;

    binomialHeap.insert(5);
    binomialHeap.insert(7);
    binomialHeap.insert(3);

    cout << "Minimum element: " << binomialHeap.findMin() << endl;

    BinomialHeap otherHeap;
    otherHeap.insert(2);
    otherHeap.insert(9);

```

```cpp
    binomialHeap.unionWith(otherHeap);

    cout << "Minimum element after union: " << binomialHeap.findMin() << endl;

    int extractedMin = binomialHeap.extractMin();
    cout << "Extracted minimum element: " << extractedMin << endl;
    cout << "Minimum element after extraction: " << binomialHeap.findMin() << endl;

    return 0;
}
```

5 Algorithm Analysis

If n represents the number of nodes in the Binomial Heap, the time and space complexity of the operations are as follows:

- Insertion: Time = $O(\log n)$, Space = $O(1)$
- Deletion: Time = $O(\log n)$, Space = $O(1)$
- Extract Minimum: Time = $O(\log n)$, Space = $O(1)$
- Union: Time = $O(\log n)$, Space = $O(1)$
- Decrease Key: Time = $O(\log n)$, Space = $O(1)$
- Delete: Time = $O(\log n)$, Space = $O(1)$

6 Reference

- Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
- Binomial Heaps lecture notes from MIT.