CSCI 115 Lab

Week 5- Binary Heaps and Priority Queues

- Professor: Dr. Matin Pirouz
  Email: mpirouz@csufresno.edu

- TA: Shreeja Miyyar
  Email:shreejarao12@mail.fresnostate.edu

Table of Contents

Priority Queues

- It is an abstract data type which can be implemented using several data structures.
- However it can be efficiently implemented using Binary Heaps.
- It supports insert, delete and extractmax, extractmin and modify operations in O(logn) time.

Binary Heaps

- A binary heap is an array representation of a complete binary tree. In a complete binary tree all the levels are filled except possibly the last level.

```
        10
      /     \
    20       100
   /
  30
```

Figure 1

```
         10
       /     \
    15         30
   /  \       /  \
 40    50   100   40
```

Figure 2

- This property of the binary heap allows it to be stored in an array.
- Binary heaps are categorized as Min Heap and Max Heap.

Example:

In Figure 1, the complete binary tree can be represented in an array as [10, 20, 100, 30]. Figure 2 can be represented as [10, 15, 30, 40, 50, 100, 40]

Binary heap representation

- The root element will be Array[0] i.e. the first element of the array.
- Below shows the index of other nodes for any given i^{th} node, i.e. Array[i]. (Assuming the index of the array starts with 0)

Arr[(i-1)/2]	Returns the parent node
Arr[(2*i)+1]	Returns the left child node
Arr[(2*i)+2]	Returns the right child node

Example

For this complete binary tree, the array representation is [10, 20, 100, 30].

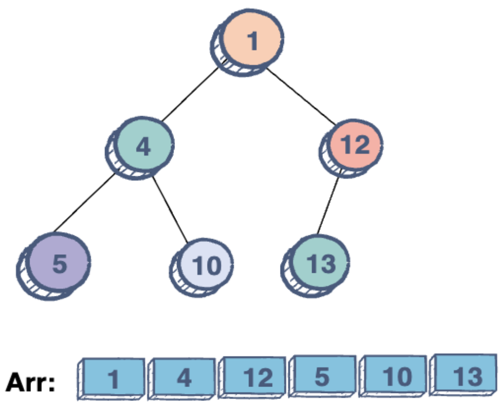
- The root element is Array[0] = 10
- For node with index i = 0, the left child is Array[(2*i)+1] = Array[1] = 20
- For node with index i = 0, the right child is Array[(2*i)+2] = Array[2] = 100
- For node with index i = 3, the parent node is Array[(i-1)/2] = Array[1] = 20

```
        10
      /    \
    20      100
   /
 30
```

Min Heap and Max Heap

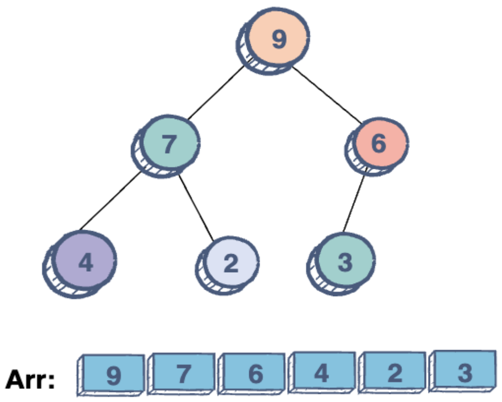
Min-Heap

- The root node has the **minimum** value.
- The value of each node is equal to or greater than the value of its parent node.
- A complete binary tree.



Max-Heap

- The root node has the **maximum** value.
- The value of each node is equal to or less than the value of its parent node.
- A complete binary tree.



Max Heap algorithm

<u>Construction of max heap from an array of N integers:</u>

MaxHeapify(A, i)

- Find left and right children of Node at index i using 2*i + 1 and 2*i + 2 respectively.
- LargestNode = Node at index i
- If Index of left child is less that array size and left child is greater than Node at index i;
  - LargestNode = Left child
- If Index of right child is less that array size and right child is greater largestNode;
  - LargestNode = right child
- If LargestNode is not equal to Node i;
  - Swap Node at index i and LargestNode
  - MaxHeapify (A, LargestNode)

BuildMaxHeap(A)

- Find index of the last non-leaf node using (N/2) – 1
- Loop variable I from (N/2) - 1 to zero
  - MaxHeapify(A, i)

Min Heap algorithm

<u>Construction of min heap from an array of N integers:</u>

MinHeapify(A, i)

- Find left and right children of Node at index i using 2*i + 1 and 2*i + 2 respectively.
- SmallestNode = Node at index i
- If Index of left child is less that array size and left child is less than Node at index i;
  - SmallestNode = Left child
- If Index of right child is less that array size and right child is less than SmallestNode;
  - SmallestNode = right child
- If SmallestNode is not equal to Node i;
  - Swap Node at index i and SmallestNode
  - MinHeapify (A, SmallestNode)

BuildMinHeap(A)

- Find index of the last non-leaf node using (N/2) – 1
- Loop variable I from (N/2) - 1 to zero
  - MinHeapify(A, i)

Heap Sort algorithm - Ascending order

HeapSortAscending(A)

- BuildMaxHeap(A)
- Loop i from N-1 to 0
  - Swap Node at index i with the root node.
  - MaxHeapify (A, i)

Heap Sort algorithm - Descending order

HeapSortDescending(A)

- BuildMinHeap(A)
- Loop i from N-1 to 0
  - Swap Node at index i with the root node.
  - MinHeapify (A, i)

Algorithm to Insert a node in Max Heap

- Create a new node at the end of the array.
- Increment the heapsize by 1
- Loop variable i from N-1 to 0
  - If the value of the inserted node is greater than the parent node, then swap them.
  - i = Parent of i

Algorithm to Insert a node in Min Heap

- Create a new node at the end of the array.
- Increment the heapsize by 1
- Loop variable i from N-1 to 0
  - If the value of the inserted node is lesser than the parent node, then swap them.
  - i = Parent of i

Algorithm to modify the value of a node in Max Heap

Modify(i, newValue)

- Array[i] = newValue
- Loop variable i till 0
  - If the value of the modified node is greater than the parent node, then swap them.
  - i = Parent of i

Algorithm to modify the value of a node in Min Heap

Modify(i, newValue)

- Array[i] = newValue
- Loop variable i till 0
  - If the value of the modified node is lesser than the parent node, then swap them.
  - i = Parent of i

Algorithm to extract the maximum element in Max Heap

ExtractMax()

- Assign the root of the heap to a temporary variable.
- The last element of the heap becomes the root of the heap.
- Decrement the heapsize by 1
- MaxHeapify(0)

Algorithm to extract the minimum element in Min Heap

ExtractMin()

- Assign the root of the heap to a temporary variable.
- The last element of the heap becomes the root of the heap.
- Decrement the heapsize by 1
- MinHeapify(0)

Lab Assignment

Write a program that takes a list and does the following operations:

1. Asks for user input for ascending or descending order

2. If ascending order then build max heap from the list. Else if descending order then build min heap from the list.

Hint: Use the Max Heap algorithm and Min Heap algorithm mentioned in the slide no. 6 and 7 respectively to write the program.

3. Insert an element into the heap.

Hint: Use the the algorithm mentioned in the slide no. 10 to write the program.

4. Modify the current element in the heap.

Hint: Use the the algorithm mentioned in the slide no. 9 to write the program.

5. Extract the maximum and minimum element from the max heap and min heap respectively.

Hint: Use the the algorithm mentioned in the slide no. 11 to write the program.

6. Print the maximum and minimum element of the max heap and min heap respectively.

Hint: Just return the first element of the heap. Do not remove it.

7. Perform heap sort (ascending or descending based on the user input)

Hint: Use the the algorithm mentioned in the slide no. 8 to write the program.

8. Fill out the report sheet.

Hint: Write a detailed report as per the template.

Coding guidelines

- In the main function provide the input array.
- Create a function for each of the operations.
- Use temporary variables when deemed necessary.

Questions?