CSI 2120 – Lab Task 5

Abdorrahim Bahrami

Data Structures in Scheme

Your task in this lab is to get yourself familiarized with creating data structures in Scheme. Make sure you have DrRacket installed, and you are familiar with the interpreter and the coding parts. If you need help, ask your TA to help you with this.

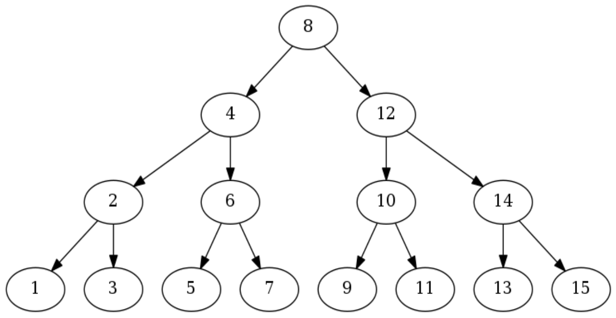
Then, you should do the following programming tasks. Each programming task in the lab is a design based on the subjects you learned during lectures. There are test samples that you can use to test your program. If you have questions, ask your TAs.

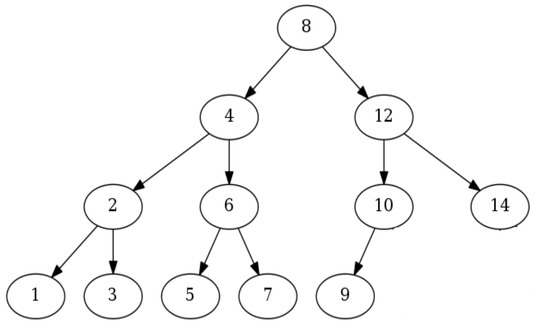
Important Note: Using ChatGPT during the lab time is strictly forbidden, if your TA catch you using it, you will receive 0 for all lab tasks immediately. During lab time, you can only use online references mentioned in class.

Heap is a data structure that has many applications in computer science. We have min-heap and max-heap. Max-heap is a binary tree, in which the data in every node is greater than both of its children. Note that there is no particular order between children of a node in a heap. You can use vector or list to implement a heap.

A perfect tree is a tree that has all the possible nodes in all the levels of the tree.

A complete tree is a tree that all its missing nodes are at the right most part of the last level of the tree.





Perfect Tree

Complete Tree

A heap is a complete tree and it always stays a complete tree.

Use the following link to see a simulation of inserting and removing an element to a tree.

https://www.cs.usfca.edu/~galles/visualization/Heap.html

This shows a min-heap but max-heap follows the same logic.

Task 1:

In this task, you need to write a function that takes a heap and an element and inserts that element into the heap. Note that, we need to keep the heap a complete tree. Therefore, we always insert the new element at the place of the first missing node in the last level of the tree. Then, we need to make a heap again. So, the inserted node is compared with its parent and if it is greater than its parent, they will be swapped. We keep doing this until the current node is not less than its parent or we reach the root. See the visualization for clarification.

```
(define (insert heap value)
)
```

Test cases are not provided for this task because it depends how you implement your data structure. You need to provide proper test cases for the method you implement the heap.

Task 2:

Heap is not a data structure made for searching. So, we don't have the search operation for the heap. Also, removing from a heap does not take a value to remove because we don't have the search operation to search for it. Don't get confused. Heap is a data structure for other purposes such as buffering. We do have a remove operation, which always removes the root of the heap. In this task, write a function that takes a heap and removes the root from the heap. For removing from heap, we need to replace the last node in the last level of the tree with the root. This keeps the heap a complete tree. Then, we need to bring the tree back to a heap. To do that, we compare the root with its children and if one of the children is greater than the root, we replace the root with that child and we move in that direction, we keep doing this task until we reach the last level of the tree. Once again, check the visualization for clarification.

```
(define (remove heap)
)
```

Test cases are not provided for this task because it depends how you implement your data structure. You need to provide proper test cases for the way method you implement the heap.