July 2023 CSE 208

Offline

Fibonacci Heap

In this assignment, you will implement a max priority queue using a Fibonacci Heap. You will have to implement the operations specified below. All operations should have amortized time complexities as specified.

You are required to implement the following functions:

1. **make_heap()**: This function initializes an empty Fibonacci Heap and returns a reference. It should run in O(1) amortized time complexity.
2. **is_empty(heap)**: This function takes the Fibonacci Heap as input and returns True if the heap is empty, otherwise False. It should run in O(1) amortized time complexity.
3. **insert(heap, key, value)**: This function inserts a new element with the given key and value into the Fibonacci Heap. It should run in O(1) amortized time complexity.
4. **extract_max(heap)**: This function extracts and returns the element with the maximum key from the Fibonacci Heap. It should run in O(log n) amortized time complexity.
5. **increase_key(heap, value, new_key)**: This function increases the key of the specified value to the new key. It should run in O(1) amortized time complexity.
6. **delete(heap, value):** This function deletes the specified node of the given value from the Fibonacci Heap. It should run in O(log n) amortized time complexity.
7. **meld(heap1, heap2):** This function melds two Fibonacci Heaps into one and returns it. It should run in O(1) amortized time complexity.
8. **find_max(heap):** This function returns the maximum element in the Fibonacci Heap without extracting it. It should run in O(1) amortized time complexity.
9. **print(heap):** This function will print the given heap. The output format is:

    Tree 1: (55, 3) -> (45, 5), (32, 4)

    (45, 5) -> (7, 8)

    Tree 2: (24, 7) -> (5, 10)

    Tree 3: ……..

    Output explanation: The heap contains two trees. The first tree is rooted at (55, 3) (key, value pair) which has two children (45, 5) and (32, 4). The (45, 5) node has one child (7, 8). The second tree is rooted at (24, 7) which has one child (5, 10)

10. **test():** Test all the functionalities (mentioned above) of your implemented priority queue for all possible scenarios. Carefully think of all possible scenarios for each operation of the priority queue. Test your implementation with various scenarios, including inserting elements, extracting maximum elements, increasing keys, deleting elements, melding heaps, finding the maximum element, etc. Design test cases to validate the correctness of your implementation. Simulate those test cases by hand and using your code too. This function will print "passed" if all the test cases are passed (the expected output and the output from your called functions are the same), otherwise print "not passed".

You are encouraged to use object-oriented programming principles. There should be a class for the Fibonacci heap implementation and a class for the priority queue implementation. The priority queue class should have the test public functions/methods mentioned above and the Fibonacci heap class should have all the required logic to implement the Fibonacci heap.

Constraints:

- -100000 <= key <=100000
- 0 <= value <= 100000
- All the values are unique in the heap

Submission Guidelines:

➔ Place all your source files inside a folder named ***<your_id>*** (***2105999***).

➔ Zip the folder, name it **<your_id>.zip (2105999.zip)**, and upload the zipped file in Moodle.

➔ **Deadline: Monday, 5 February 2024 11:00 PM.**

➔ **Do not copy from your classmate. In this case, both of you will get -100%. Also, do not copy code from any website or ChatGPT. You will get -200% in this case.**