Data Structures
Homework #4

Due: Dec 23, 2019

1. (20 pts) A *d-**ary heap*** is like a binary heap except that non-leaf nodes have d children instead of 2 children. Please answer the following questions.

   (a) How to represent a d-ary heap in an array?

   (b) What is the height of a d-ary heap of n elements in terms of n and d?

   (c) Describe how to *insert* an entry with key k into a d-ary max-heap and show that one insertion costs $O(\frac{\log n}{\log d})$ time.

   (d) Describe how the *removeMax* operation works in a d-ary max-heap. What is the time bound for this operation in terms of n and d?

2. (10 pts) Given a min-heap H and a key k, give an algorithm to compute all the entries in H with key less than or equal to k. The reported result is not necessarily sorted in some order. Your algorithm should run in time proportional to the number of entries returned.

3. (20 pts) Consider a 13-entry hash table which uses the hash function, $h(i) = (3i + 5) \bmod 13$, to hash the keys 10, 22, 31, 4, 15, 28, 17, 88, 59, 36, and 5. Please draw the resulting hash table using each one of the following strategies to handle collisions.

   (a) chaining

   (b) linear probing

   (c) double hashing using the secondary hash function $h'(k) = 7 - (k \bmod 7)$

4. (20 pts, **extra points**) Consider a heap of n keys, with x_k being the key in position k (in the continuous representation) for $0 \leq k \leq n-1$. Prove that the height of the subtree rooted at x_k is the greatest integer not exceeding $\log \frac{n}{k}$, for all k satisfying $0 \leq k \leq n-1$.

5. (50 pts) (**Programming Exercise**)
   This exercise is about to implement a ***skip list*** using ***linked structure*** with Python. A sample definition for the node and heap class is given on the course website. Recall the discussion in class. Each node has an entry with two attributes, key and name, where key is an integer and name is a string, as well as four links: `prev`, `next`, `up`, `down`. In order to manage multiple lists used in the skip list, we use the data structure, ***list***, of Python. The methods in the skip list include:

   `addEmptyList()`: This is to pad the skip list when the number of copies of the inserted node is more than the height (number of lists used) of the current skip list when inserting nodes;

   `search(v)` This method searches the skip list with the given node v using the key;

   `delete(v)`: This method will delete the given node v from the skip list;

   `insert(v)`: This is to insert the given node v to the skip list.

The Python program starts with function `create_SkipLists()` which reads the input file, `inFile.txt`. In the input file, each line contains only one entry: key and string and as follows:

```
10 mary
25 john
35 mars
50 lowe
```

An example of input file is also provided on the course website. The execution should be as below and corresponding list operations is posted on the website.

```
>>> create_SkipLists()
(-99999,)(8,kids)(27,luis)(40,kite)(99999,)
(-99999,)(8,kids)(27,luis)(40,kite)(99999,)
(-99999,)(27,luis)(99999,)
(-99999,)(99999,)
Insert (88, luke)
(-99999,)(8,kids)(27,luis)(40,kite)(88,luke)(99999,)
(-99999,)(8,kids)(27,luis)(40,kite)(99999,)
(-99999,)(27,luis)(99999,)
(-99999,)(99999,)
delete (40, kite)
(-99999,)(8,kids)(27,luis)(88,luke)(99999,)
(-99999,)(8,kids)(27,luis)(99999,)
(-99999,)(27,luis)(99999,)
(-99999,)(99999,)
Insert (27, eric)
Key found in the skip lists and will not inert the new node
(-99999,)(8,kids)(27,luis)(88,luke)(99999,)
(-99999,)(8,kids)(27,luis)(99999,)
(-99999,)(27,luis)(99999,)
(-99999,)(99999,)
delete (45, lisa)
Key not found in the skip lists and will not perform the deletion
(-99999,)(8,kids)(27,luis)(88,luke)(99999,)
(-99999,)(8,kids)(27,luis)(99999,)
(-99999,)(27,luis)(99999,)
(-99999,)(99999,)
delete (27, luis)
(-99999,)(8,kids)(88,luke)(99999,)
(-99999,)(8,kids)(99999,)
(-99999,)(99999,)
delete (8, kids)
(-99999,)(88,luke)(99999,)
(-99999,)(99999,)
delete (88, luke)
(-99999,)(99999,)
```