Dylan Mackey

CS251 - Homework 2: Hashing/sorting

Out: February 05, 2016 @ 9:00 pm
Due: February 12, 2016 @ 9:00 pm

Important: Each question has only one correct answer. Additionally, you must provide an explanation on each question. Any choice without an explanation, even though it is correct, will be graded with 0 points.

1. You have a hash table of size N (N = 65). You have total 10,000 elements (N << 10,000). The keys are all unique, and are all 64 bit positive integers. You have to find a suitable hash function h(k) for key 'k'. Which of the following hash functions will you NOT choose?

   a) h(k) = k mod N
   b) h(k) = (number of bits which are '1') mod N
   c) h(k) = $\lfloor \log_2 (k) \rfloor$
   d) h(k) = $\lfloor \log_2 (k^2) \rfloor$

   The beginning of the hash table would be extremely full as a low number of '1''s would be common.

2. Consider the same setting as question 1. The only change is, now the keys are all (non-extended) ASCII strings of length 32, instead of integers. Which of the following hash function you will NOT choose?

   a) h(k) = (sum of the ASCII values of all the characters) mod N
   b) h(k) = (Σ_i k[i] * 128^i) mod N , where $0 \le i \le 31$, and k[i] represent the i^{th} character in string k.
   c) h(k) = (ASCII value of k[i]) /2 , where 'i' is randomly chosen, $0 \le i \le 31$. And k[i] represent the i^{th} character in string k.
   d) None of the above are valid.
   e) All of the above are valid.

   If "i" is randomly chosen each time, then you will not know where the item is hashed to.

3. A hash table uses open addressing with quadratic probing. Which of the following scenarios leads to linear running time to search for a random key?

   a) All keys hash to same index
   b) All keys hash to different indices
   c) All keys hash to different even-numbered indices
   d) None of the above

   If all keys are hashed to the same index, it will take at most n probes to find the element.

   Assuming the quadratic probing circles back around to the beginning of the hash table

10
12 4

4. A hash table of length 12 uses open addressing with hash function h(k)=k mod 12, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below:

0	1	2	3	4	5	6	7	8	9	10	11
					17	18	67	6	41	22	

18 - insert
17 - insert
22 - insert
67 - insert
6 - probe, probe, insert
41 - probe, probe, probe, insert

What can be a possible order of insertion in the table?
a) 18, 17, 22, 67, 6, 41
~~b)~~ 17, 6, 18, 22, 41, 67
~~c)~~ 17, 18, 41, 67, 6, 22
~~d)~~ 22, 41, 18, 17, 67, 6

5. You are given a hash table which allows collision. So, each slot of the hash table actually keeps the head and tail pointer of a linked list, and whenever a collision occurs the new element is added to the tail of the list. Suppose, you have a hash table of size N containing M elements (N < M). Assume the hash function generates hash values i=h(k), which looks almost random. What is the average (or expected) running time (i.e., not the worst case running time) for an insert operation?
a) O(log M)
b) O(log N)
c) O(N)
d) O(1)

Average is O(1), but can be O(N) in worst case. Inserting to a hash table with linked lists is constant time.

6. In the same setting as question 5, which of the following options is closest to the average number of linked list nodes that will need to be traversed during a search operation?
a) log M
b) log N
c) M/N
d) 1

If every hash was even, then each hash slot will have $\frac{m}{n}$ elements, or nodes, that need to be traversed

7. You perform a sort on a list of 1000 items using an optimal ($O(n*\log_2 n)$) algorithm. You time the process and note that it took 20s to run. Some time later you run the sort again on a new list. This time, the process takes 96s to complete. About how many elements did you sort?
   a. 3000
   b. 4000
   c. 5000
   d. 10,000

$20 = 1000 \log_2(1000)$

$1\text{ sec} = 50 \log_2(1000)$

$(96)(50)\log_2(1000) = n \cdot \cdot$

$n \approx 47835$

8. Imagine that you have a flexible implementation of a heap data structure. You want to sort a list of (key, data) pairs by putting the list into the heap and then printing it out in decreasing sorted key order. The pseudocode for this is below. Could the below pseudocode work and what is the runtime of this function?

```
List mysort(List L)
{
      Heap heap;
      for every element e in L
            put e in heap;

      let N be length of L;
      List SL;
      for N times
            remove largest key and put in SL;

      return SL;
}
```

Assuming by returning sL we print the items as well, this can work. Each element will be inserted and then up-heaped. Then, the largest key will be removed and added to sL. To insert, it takes nlog(n), as well as to remove time.

So, 2n log(n), or O(nlogn)

a) This can work, and runs in $O(n^2)$.
b) This can work, and runs in $O(n*\log(n))$
c) This can not work, but runs in $O(n^2)$.
d) This can not work, but runs in $O(n*\log(n))$

9. You run BubbleSort on a list of 10,000 items that are listed in reverse order. Exactly how many swaps (not comparisons) will happen when running this program?

   a. 49,995,000
   b. 50,000,000
   c. 50,005,000
   d. 100,000,000

$\frac{n^2}{2} - \frac{n}{2}$. In a worst case scenario, a bubble sort will perform $\frac{n^2}{2} - \frac{n}{2}$ swaps. In this case, 49,995,000 when n is 10,000. The Big-Oh is still $O(n^2)$.

10. Now consider a Selection Sort (as could be implemented with a sequence based priority queue) with the same list of 10,000 items from problem 9. Exactly how many swaps (not comparisons) will happen when running this program?

   a. 10,001
   b. 10,000
   c. 4999
   d. 5000

There are $\frac{n}{2}$ swaps in this worst-case.

6 5 4 3 2 1
1 5 4 3 2 6
1 2 4 3 5 6
1 2 3 4 5 6

11. Generalize the result from question 10. Given a reversed list of N objects, exactly how many swaps will selection sort do on the list?

   a. N + 1
   b. N
   c. ceiling(N / 2)
   d. floor(N / 2)

Since the middle element will not need to be swapped, it is the floor of $(N/2)$.

5 4 3 2 1
1 4 3 2 5
1 2 3 4 5

12. Suppose that you have a pre-sorted singly linked list and no other auxiliary data structure. You want to insert a new item into the structure in a way that keeps the list in order. What is the runtime of this operation?

   a. O(log(n))
   b. O(n)
   c. O(1)
   d. O(n*log(n))

Inserting is $O(1)$, however finding the node to insert is $O(n)$, therefore $O(n)$.