Cs445 — Homework #2. Due Tue 10/6/2015 Before Class

Instructions.

1. Solution could **not** be submitted by students in pairs.
2. You could submit a pdf of the homework, either printed or hand-written and scanned, as long as it is **easily** readable.
3. If your solution is not clearly written, it might not be graded.
4. Prove the correctness of your answer. A correct answer without a proof might not be accepted.
5. If you have discussed the solution with other students, mentioned their names clearly on the homework. These discussions are not forbidden and are actually **encouraged**. However, you must write your whole solution yourself.
6. All questions have same weight.
7. You could refer to a data structure studied in class, and just mention briefly their guaranties. For example "*It is known that a Red-Black tree could support the insert/delete/find operations on a set of n elements in time $O(\log n)$.*
8. If your answer uses one of the data structures or algorithms that were studied in class, you could refer to it without having to repeat details studied in class. If you answer requires only modifications of one of the algorithm, it is enough to mention the required modifications, and what's the effect (if any) on the running time and on other operations that the algorithm performs.
9. In general, a complete answer should contain the following parts:
   (a) High level description of the data structures (if needed). *E.g. We use a binary balanced search tree. Each node contains, a key and pointers to its children. We augment the tree so each node also contains a field...*
   (b) High level description of the algorithms
   (c) Proof of correctness (why your algorithm provides what is required).
   (d) A claim about the running time, and a proof showing this claim.
10. When talking about tress, do not confuse the *children* of a node with the *decedents* of the node.

1. Explain which field(s) would you add to each element in a SkipList, so you could perform all operations of the dynamic order statistic `OS-SELECT`(i, S) and `OS-RANK`(x, S) for a set S of n elements stored in the SkipList. Specify exactly what each field contains, and how you update them during insertions and deletions of keys. The expected time for `OS-SELECT`(i, S), `OS-RANK`(x, S), and for insertion, deletion or finding a key should all be $O(\log n)$.

   For your convenience, here are the definitions of these operations: `OS-SELECT`(i, S) returns the i-th smallest key in the dynamic set S, while `OS-RANK`(x, S) returns the number of keys in S smaller or equal to x.

2. Let $A[1..n]$ be a given array, not necessarily sorted. and let k be a given integer such that $k = 2^\ell$ for some integer ℓ. Suggest an algorithm with expected running time $O(\ell n)$ that finds the set $S = \{x_1, x_2 \ldots x_{k-1}\}$ of keys from A such that the rank of x_i in A is $\lfloor n\frac{i}{k} \rfloor$.

   As usual, rank(x) is the number of keys of A smaller than x. For example, if $k = 2$ then S contains only the median of A. If $\ell = \log_2 n$ then S contains every key from A.

3. Your friend (who happened to be an instructor in a survival course) has dropped you at a point p at the desert. You wish to walk to the nearest road, from which you could hitchhike back home. However, you have no idea which direction should you head to. Explain how you could walk a distance of $c \cdot d$. Here d is the distance from p to the nearest point on any of the roads, and c is a constant (does not depend on d) that you have to compute.

   Comments:

   (a) Assume that the desert is flat, so every path is possible. Also assume that you are equipped with an accurate GPS, but have no access to maps.

   (b) For simplicity, assume that you need to be physically on the road to see it. So ignore range-of-visibility issues.

   (c) Assume that the roads create a connected graph. Meaning that from any point on one road you could reach any point on any other road while being on roads along your trail.

   (d) Note that you do not have to reach the closest point to p.

4. Assume an open addressing hash table, with a load balance α given. What is the expected number of probes needed when searching a key x that is *in* the table ? Note that the development shown in class was for the case that x *is not* in the table (unsuccessful search).

5. Assume a hash table $T[0..15]$ (that is, $m = 16$), and a double addressing hashing where

   $$h(x, i) = \{x + i \cdot (x \textit{ mod } 8)\} \textit{ mod } 16.$$

   Show an example of a set of keys 5 keys $\{k_1, k_2, k_3, k_4, k_5\}$ that you could not insert into the table, but $k_j \textit{ mod } 8 > 0$ for $j = 1, 2, 3, 4, 5$.

   Assume the table is empty before the first insertion. Specify in which order they are inserted, and what is the cause that no further insertions are possible.

6. As you remember, in open-addressing hashing tables we try to keep the load balanced $\alpha = n/m$ to be ≤ 0.5. An efficient hash implementation requires that the table is an array stored in a contiguous block of memory. This creates a problem if n is not known in advanced, and as new elements are inserted, the array might be too small (that is, $\alpha > 0.5$). Consider the following approach for handling this issue:

   (a) Start with a relatively small array, say $T[1..100]$.

   (b) Repeat inserting elements into T. (Ignore deletions in this exercise). If after an element is inserted $\alpha > 0.5$, do

      i. Set $m' = 1.1m$ (increase m be 10%.)

      ii. Allocate a new array T' containing m' cells. Note that takes $\Theta(m')$ time.

      iii. Find a hash function that is appropriate for m'.

      iv. Scan t and copy all keys from T to T', using the new hash function. Gravestones used to indicate deleted elements are ignored.

      v. Discard T, relabel $T = T'$ and $m = m'$.

   Questions:

   (a) What is the value of α right after the expansion ?

   (b) What is the maximum time required for a single insertion (expressed as a function of n) ?

   (c) What is the total time needed for performing n insertions, for n much larger than 100.

   (d) Discuss pro and cons of this approach, vs. increasing the array by a multiplicative factor of 3. (so $m' = 3m$ rather than $m' = 1.1m$).

   (e) Discuss pro and cons of this approach, vs. increasing the array by adding 100 empty cells. (so $m' = m + 100$ rather than $m' = 1.1m$).

7. (Tricky) Consider the interval $[0, 1]$ of all points x such that $0 \leq x \leq 1$. Assume that each point of the interval can be colored either white or black, and initially all the points of the intervals are colored white. Suggest a data structure that supports the following operations

   `Reverse`(a, b) — reverse the color of each point between a and b where $0 \leq a < b \leq 1$. (that is, every point in (a, b).) So a point y where $a < y < b$ that was white is becoming black and vice versa. The color of the other points are not effected.

   `Report`(x) — What is the color of the point x.

   Each operation should take $O(\log n)$ where n is the number of `Reverse` operations. The space required is $O(n)$. Try to minimize the actual space required.

   Note that you could not discritize the interval $[0, 1]$. That is, there is no **finite** set of points $D \subseteq [0, 1]$ such that the values of a, b or x in the operations above is always a point of D.

8. Consider an open addressing where $h(x, i) = (h(x)+i)\ mod\ m$. Here m is the size of the table, and $h(x)$ is some hash function known to you (it is not important to the question what it is exactly). Show that there is a set $S = \{k_1 \ldots k_m\} \subseteq U$ such that inserting them into the table T would take $\Theta(m^2)$ time.

You do not need to discuss the time for **finding** these keys. Only to show their existence. Assume $|U| > m^2$.

9. You need to implement Quicksort algorithm, for sorting an array $A[1..n]$. Assume all keys in A are number between 1 and 10 (not necessarily integers). Which methods would for picking the pivot would you recommend perform well for picking the pivot ? (see slides if you are not sure about the definitions).

   (a) Use $A[1]$ as the pivot.

   (b) Pick a random number p between 1 and 10, and use it as a pivot.

   (c) Pick a random integer $1 \leq i \leq n$ (where each index i has the same probability to be picked) and use $A[i]$ as a pivot.

10. Explain how you would use hash functions to find if your computer contains two identical files (possibly under different names). Give peudocode of your solution. Treat each file as an ascii file consists of a string of characters. Specify which and how your hash functions are used. Do not use values provided by the file system.

    Assume all files have the same length.