CMSC 420: Spring 2022

CMSC 420 (0101) - Final Exam

This exam is closed-book and closed-notes. You may use three sheets of notes (front and back). Write all answers on the exam paper. You may use any algorithms or results given in class. If you have a question, either raise your hand or come to the front of class. Total point value is 120 points. Good luck!

Problem 1. (30 points) Short answer questions. Unless requested, explanations are not required, but may be given to help with partial credit.

(a) (4 points) Early in the semester we saw that a 2-dimensional matrix could be represented using a *multilist structure*. What are the main advantages of the multilist over a standard array-based representation? (Select all that apply.)

(1) Matrix entries can be *modified faster* with the multilist

(2) Matrix operations (e.g., multiplication) are *generally faster* with the multilist

(3) If the matrix is sparse (few non-zero elements), the multilist *saves space*

(b) (6 points) When we delete an entry from a simple (unbalanced) binary search tree, we sometimes need to find a replacement key. Suppose that `p` is the node containing the deleted key. Which of the following statements are true? (Select all that apply.)

(1) A replacement is needed whenever `p` is the root

(2) A replacement is needed whenever `p` is a leaf

(3) A replacement is needed whenever `p` has two non-null children

(4) It is best to take the replacement exclusively from `p`'s right subtree

(5) At most one replacement is needed for each deletion operation

(c) (2 points) The AA-tree data structure has the following constraint: "*Each red node can arise only as the right child of a black node.*" Which of the two restructuring operations (`skew` and `split`) enforces this condition?

(d) (4 points) Suppose that a subtree of height h in a Quake Heap has been constructed by applying some number of `link` operations (but no `cut`'s). As a function of h, how many leaves does this subtree have? (Select one.)

(1) Exactly 2^h

(2) At most 2^h, but possibly fewer

(3) At least 2^h, but possibly more

(4) We cannot put an exact upper or lower bound, but it will be $O(2^h)$

(5) None of the above

(e) (3 points) A node in a B-tree has too many children. Suppose that it is possible to resolve this either by *splitting* or *key-rotation* (adoption). Which is preferred and why?

(f) (4 points) Hashing is widely regarded as the fastest of all data structures for basic dictionary operations (insert, delete, find). Give an example of an operation that a tree-based search structure can perform *more efficiently* than a hashing-based data structure, and explain briefly.

(g) (3 points) In the (unstructured) memory management system discussed in class, each available block of memory stored the size of the block both at the beginning of the block (which we called `size`) and at the end of the block (which we called `size2`). Why did we store the block size at both ends?

(h) (4 points) In our implementation of Prim's EMST algorithm, which of the following is true throughout the execution of the algorithm? (Select all that apply)

(1) The number of entries in the spatial index (kd-tree) is at least as large as the number of points in the current spanning tree

(2) The number of entries in the spatial index (kd-tree) is at least as large as the number of points that are *not* in the current spanning tree

(3) The number of entries in the priority queue (heap) is at least as large as the number of points in the current spanning tree

(4) The number of entries in the priority queue (heap) is at least as large as the number of points that are *not* in the current spanning tree

Problem 2. (10 points) Show the result of executing the operation `splay(5)` on the tree in Fig. 1. (Intermediate results may be given for partial credit.)

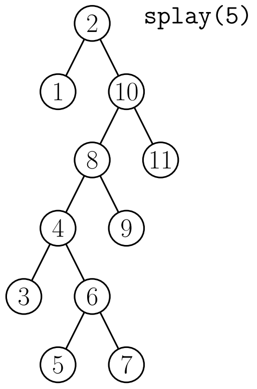

Figure 1: Splaying.

Problem 3. (15 points) Perform the following operations on a hash table. In each case, the operations are performed *as a sequence* and list the *number of probes.*

(a) Show the results of inserting the keys "`X`" then "`Y`" then "`Z`" into the hash table shown in Fig. 2(a) assuming *double hashing*, where `g()` is the jump size.

(b) Show the result of insertions and deletions in Fig. 2(b), assuming *linear probing.* (When deleting, indicate what your placeholding symbol is.)

Problem 4. (15 points) In this problem we will build a suffix tree for the text $S =$ `"babaaba$"`.

(a) (4 points) List the substring identifiers for the 8 suffixes of S. For the sake of uniformity, list them in order (either back to front or front to back).

(a) Double hashing

```
insert("X")  h("X") = 1; g("X") = 4
insert("Y")  h("Y") = 7; g("Y") = 2
insert("Z")  h("Z") = 1; g("Z") = 5
```

0	1	2	3	4	5	6	7	8	9
▨	▨	▨	▨	M	▨	V	L	▨	P

(b) Linear probing/Deletion

```
insert("X")  h("X") = 4
delete("B")  h("B") = 2
insert("Z")  h("Z") = 3
```

0	1	2	3	4	5	6	7	8	9
W	▨	F	M	B	J	V	▨	▨	P

Figure 2: Hashing.

(b) (8 points) Draw S's suffix tree. List children alphabetically (`"a" < "b" < "$"`). Label each leaf with its suffix index (from 0 to 7). Also, label each internal node with its number of descendent leaves.

(c) (3 points) We want to know how many occurrences of the substring `"ba"` occur in S. Which node(s) of the suffix tree provide the answer to this query?

Problem 5. (20 points: 2–6 points for each part) In this problem, we are given a set L of n horizontal line segments $\overline{s_i t_i}$ in the plane, where $s_i = (x_i^-, y_i)$ and $t_i = (x_i^+, y_i)$ (Fig. 3(a–b)). We want to preprocess them to answer the following queries efficiently:

Segment stabbing queries: Consider a vertical query line segment with x-coordinate q_x, whose lower endpoint has the y-coordinate q_y^-, and whose upper endpoint has y-coordinate q_y^+. *How many of the segments of L does this segment intersect?* (For example, the vertical segment in Fig. 3(c) intersects 5 segments of L.)

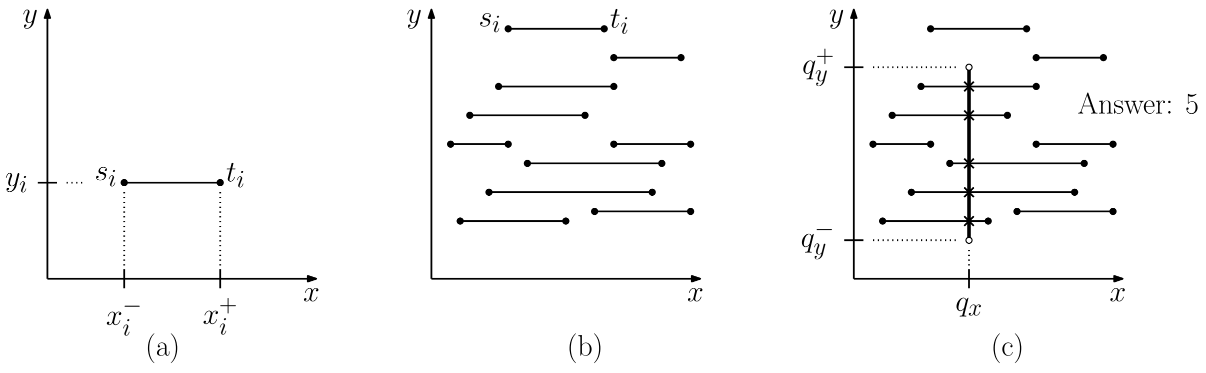

Figure 3: Segment stabbing queries.

Answer the following for the query vertical query (q_x, q_y^-, q_y^+) and horizontal segment $\overline{s_i t_i}$. (**Hint:** To simplify your answer, you may assume that all the coordinates are distinct, so the endpoint of one segment will never lie in the interior of another.)

(a) What conditions must a horizontal segment's y-coordinate (y_i) satisfy to intersect the query segment?

(b) What conditions must a horizontal segment's left endpoint (x_i^-) satisfy to intersect the query segment?

(c) What conditions must a horizontal segment's right endpoint (x_i^+) satisfy to intersect the query segment?

(d) Based on parts (a)–(c), briefly explain the structure of range tree to answer segment stabbing queries. (Query processing comes later.)

(e) Briefly explain how to apply your structure from (d) to answer segment stabbing queries.

(f) Given that there are n segments, what is the space and query time of your data structure?

Problem 6. (15 points) You are designing an expandable hash table using open addressing. Let m denote the current table size. Initially $m = 4$. Let us make the ideal assumption that each hash operation takes exactly 1 time unit. After each insertion, if the number of entries in the table is greater than or equal to $3m/4$, we expand the table as follows. We allocate a new table of size $4m$, create a new hash function, and rehash all of the elements from the current table into the new table. The time to do this expansion is $3m/4$.

(a) (10 points) Derive the amortized time to perform an insertion in this hash table (assuming that m is very large). State your amortized running time and explain how you derived it. (For fullest credit, your running time should as tight as possible.)

Hint: The amortized time need not be an integer.

(b) (5 points) One approach to decrease the amortized time is to modify the table expansion factor, which in this case is 4. In order to reduce the amortized time, should we *increase* or *decrease* this factor? If you make this adjustment, what negative side effect (if any) might you observe regarding the space and time performance of the data structure? **Explain briefly. (Don't give a formal analysis)**

Problem 7. (15 points) In this problem you will write a program to check the validity of an AVL tree. The node structure is given below. All members are public.

```
class AVLNode {
    public int key                // key
    public int height             // height of this subtree
    public AVLNode left, right    // left and right children
}
```

In order to be valid, every node `p` of the tree must satisfy the following conditions:

- `p.height` is correct given the heights of its children. (Recall: `height(null) == -1`.)
- The absolute height difference between `p`'s left and right subtrees is at most 1
- An inorder traversal of the tree encounters keys in *strictly* ascending order

Present pseudocode for a function `boolean validAVL(AVLNode root)`, which returns `true` if the tree structure at the given `root` node is a valid AVL tree and `false` otherwise. Explain how your function works. For full credit, your function should run in time $O(n)$, where n is the number of nodes in the tree. You may *not* assume the existence of complex utility functions (e.g., for sorting, performing tree traversals, or computing tree heights).

Hint: Use recursion. You may write additional helper functions.

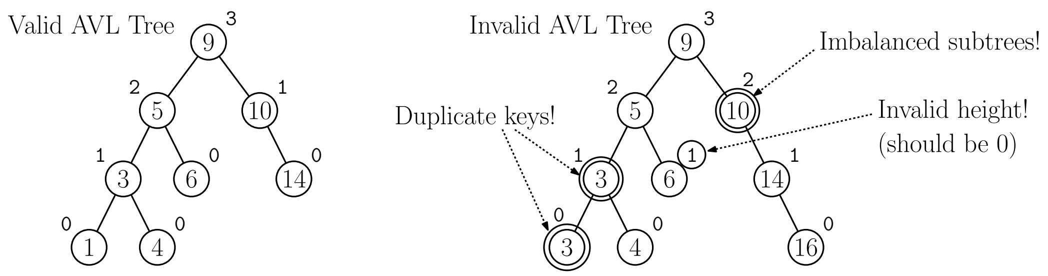

Figure 4: Valid and invalid AVL trees.