Exam: TDDD86

Data Structures, Algorithms and Programming Paradigms

2021-12-15 kl: 08-12

On-call (jour): Ahmed Rezine (tel: 1938)

Specific instructions for the computer exams:

- A more detailed description can be found at the end of this document.
- In summary: you log on with your LiuID and your private password. You can only save files in the desktop. We might leave files for you in the read-only "given_files" folder. You will use the "student chat client", or "student client" to receive information during the exam, to ask questions and to submit your solution.
- Your "student client" should start automatically, if you close it and need to start it again, double click on the "fish icon" on your desktop.
- Submit one file with all your answers. We will only look at the last submitted file.
- You can access OpenDSA using chromium. The start page will list available links.

General instructions:

- You may answer in either English or Swedish.
- **Submit a single document (text file or pdf) with your answers. The document should only contain text** (for instance, no pictures, no drawings).
- The questions are formulated so that you can answer with any text editor (e.g., vim, emacs, gedit, etc) or an office program (e.g., Open/Libre Office).
- If in doubt about the question, write down your interpretation and assumptions.
- The exam is divided into two parts:
  - Part A with a maximum of 30 pts.
  - Part B with a maximum of 17 pts.
- Grading:
  - Grade 3 requires at least 20 pts exclusively from Part A.
  - Grade 4 requires grade 3 is secured and at least 6 pts from Part B.
  - Grade 5 requires grade 3 is secured and at least 12 pts from Part B.

Preliminaries

- Recall that general binary trees can be represented sequentially. We adopt the approach described in 8.3.1 in OpenDSA. For instance, the binary tree in Figure 2 can be represented using the sequence: "AB/D//CEG///FH//I//", where the symbol "/" is used to represent a "null" child. Do not draw your trees! Use this approach instead.
- You can write Theta for Θ and Omega for Ω.

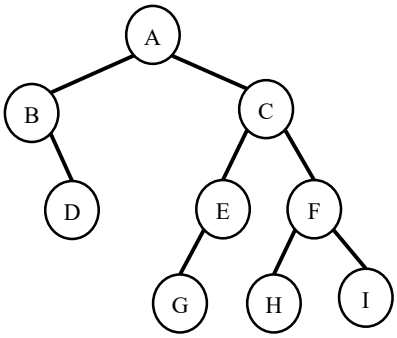

Figure 1

Part A

Binary search trees and their traversal (8 pts)

Consider the sequence of 15 elements:

S = 613, 349, 930, 574, 192, 688, 154, 398, 628, 519, 225, 527, 292, 928, 147

1. Give a sequential representation (see the preliminaries) of the final binary search tree obtained by starting from an empty tree and inserting all the integers of the sequence S one after the other (i.e., first insert 613, then 349, then 930 …). Do not try to balance the tree. Call this tree T_1. (2pts).

2. List the integer values encountered in a **post-order** traversal of T_1. (2pts).

3. Give a sequential representation of the tree obtained by removing the root node from tree T_1. Write your assumptions in case you make choices. (2pts).

4. Give a sequence S' that is a reordering of the S so that the binary search tree obtained by starting from an empty tree and inserting the integers in S' one after the other is of a minimum height. (2pts).

Hashing and conflict resolution (6 pts)

Consider the sequence hashSeq defined as:

hashSeq: 613, 349, 930, 574, 192, 688, 154, 628, 527, 292, 928, 147

Observe this sequence is different from the previously considered sequences!

We will assume closed hashing and use the hash function $h(k) = k \% 20$ to compute the "home" (or "default") slot of key k. We will consider in the following three collision resolution approaches.

For each approach, we will insert all elements of hashSeq (one after the other) starting from an empty hash table "table". Indices of the table range from 0 to 19 (i.e., the table is of capacity 20). Initially all cells table[0], … table[19] are marked empty.

Give the content of table[17] after adding all elements of hashSeq according to the above description:

5. Case 1: use linear probing with probe function $p(k, i) = i$. (2pts).

6. Case 2: use linear probing with probe function $p(k, i) = 3i$. (2pts).

7. Case 3: use quadratic probing with probe function $p(k, i) = i^2$. (2pts).

Sorting (8pts)

Consider the array `arr` containing the 7 integers: `613,349,930,574,192,688,154`; with `arr[0]=613, arr[1]=349, … arr[6]=154.`

8. Consider the radix sort algorithm with digits in `0..9`. When sorting the content of array `arr` in ascending order, the radix sort algorithm proceeds in rounds, one round for each digit. Give the sequence of arrays obtained at the end of each round. There will be 4 arrays counting the initial array. You can represent an array with the sequence of its content. For instance, the initial array `arr` contains the sequence 613, 349, 930, 574, 192, 688, 154. (4pts).

9. Consider the quicksort algorithm and suppose we use it to sort the array `arr` above by calling `quicksort(arr,0,6)`. Observe that a recursive call on a remaining sequence in the described quicksort algorithm of chapter 11.11 of OpenDSA is only made if the remaining sequence has 2 or more elements.

    a. **Give a sequence of pivot elements** that results, for the array `arr,` in a **largest number of calls** to `quicksort`. Briefly explain without going in the details of the partition steps. (2pts).

    b. **Give a sequence of pivot elements** that results, for the array `arr,` in a **smallest number of calls** to `quicksort`. Briefly explain without going in the details of the partition steps. (2pts).

Graphs and shortest paths (8pts)

Consider the graph in Figure 2. It represents cities around Vättern. Edges are undirected and weighted. We will identify the edges with the cities at their extremities. For instance, the Linköping-Norrköping edge has weight 43, while the Karlborg-Jönköping edge has weight 99. The weights correspond to road-distances in kilometers.

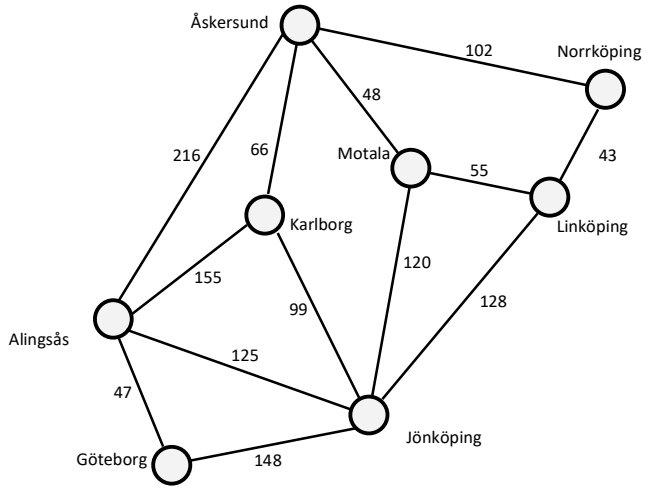

Figure 2. Road distances around Vättern in kilometers

10. List the **cities** in the order they are discovered by a **depth first traversal** of the graph in Figure 2. The traversal should start from Linköping. Use the weights to break ties when choosing among the neighboring cities of a given node. For instance, when choosing among the unvisited cities neighboring Linköping, the algorithm should first consider Norrköping, then Motala, then Jönköping. (2pts).

11. List the **cities** in the order they are encountered by a **breadth first traversal** of the graph in Figure 2. The traversal should start from Linköping. Use the weights to break ties when choosing among the neighboring cities of a given node. For instance, when choosing among the unvisited cities neighboring Linköping, the algorithm should first consider Norrköping, then Motala, then Jönköping. (2pts).

12. Dijkstra's algorithm can be used to find shortest paths from a single source. Such paths consist of edges as those in Figure 2. The algorithm chooses edges that will be part of resulting shortest paths from the single source. Assume the single source is Linköping. Enumerate the edges that will be part of the shortest paths in the order they are discovered by the algorithm. You can stop as soon as you find a shortest path from Linköping to Göteborg. Observe that (i) you should only list the edges chosen by the algorithm to be part of shortest paths from Linköping to other cities, not the edges that are encountered by the algorithm but that are not

chosen to be part of shortest paths, and (ii) you should list these "shortest-paths-edges" in the order in which the algorithm decides they are indeed part of a shortest path from Linköping to some city. (4pts).

Part B:

Problem 1 (4pts):

A* search algorithm. Consider again Figure 2 with the road distances around Vättern in kilometers. Suppose you are given estimations of the distances among all cities (these are straight line distances and are therefore under-approximations of the road distances). The A* search algorithm can be used to find a shortest path from a source (here Linköping) to a destination (here Göteborg). Unlike Dijkstra's algorithm, it uses a heuristic to prioritize its search when "picking" cities from its priority queue. These cities that are picked from A*'s priority queue will not necessarily be part of a shortest path from Linköping to Göteborg.

13. Enumerate the cities picked by by A*. You can stop as soon as you encounter Göteborg. (4pts).

Linköping	Norrköping	Motala	Åskersund	Karlsborg	Alingsås	Jönköping	Göteborg	
0	38	36	66	65	189	113	224	Linköping
	0	67	80	97	227	150	267	Norrköping
		0	39	31	160	96	200	Motala
			0	51	170	127	213	Åskersund
				0	135	82	173	Karlsborg
					0	97	41	Alingsås
						0	130	Jönköping
							0	Göteborg

Figure 3. Under-approximations of distances used as heuristics for Part B. Problem 1.

Problem 2 (4pts):

Splay trees. Recall the tree T_1 you obtained as answer to question 1. From now on, we will treat T_1 as a splay tree (it is exactly the same tree you obtained from question 1, only the following operations will involve splay operations). The following two questions will perform find operations on it T_1. These might modify T1 since it is now assumed to be a splay tree.

14. Give the sequential representation of the tree T_2 resulting from performing a find(192) in T_1. (2pts).

15. Give the sequential representation of the tree T_3 resulting from performing a find(688) in T_1 (observe again this is the original T_1 from question 1, not T_2 obtained after question 14 above). (2pts).

Problem 3 (9 pts):

We will work with matrix representations of multisets of totally ordered keys. The considered representation is known as "Young tableaux" or "tableaux" for short. A tableau is an $M \times N$ matrix with $i: 0 \leq i < M$ rows and $j: 0 \leq j < N$ columns. We will use it to store integers (representing keys totally sorted with "<"). Duplicates are possible. Initially, a tableau stores `MAX_INT` at all positions and represents an empty tableau. A tableau is said to be sorted if: (i) each row $i: 0 \leq i < M$ is sorted in ascending order from left (column index $j = 0$) to right (column index $j = N - 1$), **and** (ii) each column $j: 0 \leq j < N$ is sorted in ascending order from top (row index $i = 0$) to bottom (row index $i = M - 1$).

1. Give two different 4x4 sorted tableaux containing the same elements:
   $$\{613,349,930,574,192,688,154,398,628,519,225,527\}$$
   It is up to you how you place the elements as long as the resulting tableau is sorted. You can enumerate the rows to represent the tableaux in your answer. (2pts). For example, the 4x3 tableau:

1	2	MAX_INT
3	MAX_INT	MAX_INT
MAX_INT	MAX_INT	MAX_INT
MAX_INT	MAX_INT	MAX_INT

   Can be represented with the four rows (with _ standing for MAX_INT):
   Row 0: 1, 2, _
   Row 1: 3, _,_
   Row 2: _,_,_
   Row 3: _,_,_

2. Consider the "`void insert(vector<vector<int>>& tableau, int i, int j)`" method in the code described in figure 4. This method is used to insert an element to a sorted (but not full) tableau. Before the call to insert, the new element is first placed at the bottom right position of the tableau (position `i=M-1` and `j=N-1)`. See the line "`tableau[M-1][N-1] = key;`" in the main method of Figure 4. The insert method is then called with "`insert(tableau, M-1, N-1);`" and is meant to move the new element in the tableau by performing some permuations in order to result in a sorted tableau with the same elements as those passed to `insert`. This method can be written so that "`insert(tableau, M-1, N-1)`" has a $\Theta(M+N)$ worst case time complexity.
   a. Propose "C++ like pseudo code"[1] for "`void insert(vector<vector<int>> &tableau, int i, int j)`". The method can be iterative or recursive. It

[1] Similar to the code in the main method, "C++ like pseudo code" should clarify the steps involved in your method. It should be possible to compile with minor changes. Syntax errors such as missing semi-colons not a problem.

should move the key currently at position `(i,j)` in `tableau` to a position where the resulting `tableau` is sorted and has the same elements as the input tableay. You can assume the input `tableau` is sorted except possibly for position `(i,j)`. A call to your method with "`insert(tableau, M-1, N-1)`" should have $\Theta(M+N)$ worst case time complexity. Hint: the element to be inserted is placed first at the bottom right extremity `tableau[M-1][N-1]` (assuming it is MAX_INT). The element is then moved up and left to a position where the tableau is sorted. This is done in a manner that is similar to the restoring the heap property after deleting an element. (2pts).

   b. Briefly argue why your solution is correct and explain why it has $\Theta(M+N)$ worst case time complexity. (2pts).

3. Consider the "`bool find (const vector<vector<int>>& tableau, int k)`" method in the code described in the next page. This method is meant to search for the key `k` in `tableau` and to return true if `k` is in `tableau` and false otherwise. This method can be written to have a $\Theta(M+N)$ worst case time complexity (assuming the tableau is sorted and has N rows and M columns):

   a. Propose "C++ like pseudo code" for "`void find (const vector<vector<int>>& tableau, int key)`". The method can be iterative. You can assume the input `tableau` is sorted. Your method should have $\Theta(N+M)$ worst case time complexity. (Hint: start comparing with the top right extremity of the tableau, `tableau[0][N-1]`, and proceed depending on the outcome). (2pts).

   b. Briefly argue why your solution is correct and explain why it has $\Theta(M+N)$ worst case time complexity. (2pts).

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <climits>
using namespace std;

// The method “insert” moves the element at position (i,j) in order to
// ultimately result in a sorted tableau with the same elements as the tableau
// in input. When called, the tableau is sorted except for position (i,j).
void insert(vector<vector<int>>& tableau, int i, int j)
{
                ...
}

// The method “find” searches the tableau for a key k. It returns true
// if the k is found, false otherwise. The tableau is assumed to be sorted.
bool find(const vector<vector<int>>& tableau, int key)
{
        ...
}

int main()
{
  vector<int> keys = {613, 349, 930, 574, 192, 688, 154, 398, 628, 519, 225, 527};

  // A 4 x 4 tableau will be enough
  int N = 4; // number of rows
  int M = 4; // number of columns

  // construct an M x N empty tableau
  vector<vector<int>> tableau(M, vector<int>(N, INT_MAX));
  for (int key: keys)
  {
    // is there room left?
    if (tableau[M-1][N-1] == INT_MAX) {
      tableau[M-1][N-1] = key;
      // move the key to its correct position in the tableau
      insert(tableau, M-1, N-1);
    }
  }

  vector<int> otherKeys = {613, 350, 93, 574, 19, 1688};
  for(int key: otherKeys){
    if(find(tableau, key)){
      cout << key << " found!" << endl;
    }else{
      cout << key << " not found!" << endl;
    }
  }

  return 0;
}
```

Figure 4. Code skeleton for tableau insertion and find