Exam: TDDD86

Data Structures, Algorithms and Programming Paradigms

2024-03-13 kl: 08-12

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

Specific instructions for the computer exams:

- In summary: you log in 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. More details in the "EXAM_README.pdf" under "given_files".

- 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) 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 a question, write down your interpretation and assumptions.

- The exam is divided into two parts:
  - Part A with a maximum of 34 pts.
  - Part B with a maximum of 20 pts.

- Grading:
  - Grade 3 requires at least 20 pts exclusively from Part A.
  - Grade 4 requires grade 3 is secured and at least 8 pts from Part B.
  - Grade 5 requires grade 3 is secured and at least 12 pts from Part B.

Part A

Problem A.1: Asymptotic execution time (minimum 0pts, maximum 10 pts)

Consider the five methods `f1, f2, f3, f4, f5` and the nine complexity classes (A)-(I) depicted below. Assume the manipulated arrays are large enough in the four methods. You can always assume the size "n" of the problem to be a power of 2.

```
// Size of the problem is n (not x).
// You can assume n = 2^p for some integer p.
// The array "a" is sorted
int f2(int a[], int n, int x){
   int i= -1;
   for (i = 0; i < n ; i++){
      if(a[i] >= x){
         return i;
      }
   }
   return i;
}
```

```
// Size of the problem is n.
// You can assume n = 2^p for some integer p.
int f3(int a[], int n){
   int rslt = 0;
   for (i = 0; i + 10 < n; i++){
      int j = 0;
      int is_plateau = 1;
      for (j = i + 1; j < i + 10; j++){
         if(a[0] != a[j]){
            is_plateau = 0;
         }
      }
      if(j == i + 10 && is_plateau == 1 ){
         rslt = rslt + 1;
      }
   }
   return rslt;
}
```

```
// Size of the problem is n = hi - lo.
//You can assume n = 2^p for some integer p.
int f4(int a[], int lo, int hi, int x){
   if(lo >= hi){
      return -1;
   }
   int m = lo + (hi - lo)/2;
   if(a[m] < x){
      return f4(a, lo, m, x);
   }else{
      return f4(a, m, hi, x);
   }
}
```

```
// Size of the problem is n.
// You can assume n = 2^p for some integer p.
int f1(int a[], int n){
   int rslt = 0;
   int i = 0;
   int j = 0;
   while(j < n){
      if(i == n){
         j = j + 1;
         i = 0;
      }
      if(a[i] == a[j]){
         rslt = rslt + 1;
      }
      i = i + 1;
   }
   return rslt;
}
```

```
// Size of the problem is n = hi - lo.
// You can assume n = 2^p for some integer p.
int f5(int a[], int lo, int hi, int x){
   if(lo >= hi){
      return 0;
   }
   int m = lo + (hi - lo)/2;
   int a = f5(a, lo, m, x);
   int b = f5(a, m+1, hi, x);
   if(a[m] == 0){
      return a + b + 1;
   }else{
      return a + b;
   }
}
```

Complexity classes:

(A) $\Theta(1)$	(D) $\Theta(n \log n)$	(G) $\Theta(2^n)$
(B) $\Theta(\log n)$	(E) $\Theta(n^2)$	(H) $\Theta(3^n)$
(C) $\Theta(n)$	(F) $\Theta(n^3)$	(I) $\Theta(n!)$

1. For each one of the five methods above, give (without justification!) the complexity class among the classes (A-I) that best matches its asymptotic **worst-case** execution time. For each method, 1pts if correct, 0 if not answered, -1pts if incorrect.

2. For each one of the five methods above, give (without justification!) the complexity class among the classes (A-I) that best matches its asymptotic **best-case** execution time. For each method, 1pts if correct, 0 if not answered, -1pts if incorrect.

Problem A.2: Heaps (minimum 0pts, maximum 10 pts)

Consider the tree in Figure 1. It represents a heap with 7 elements.

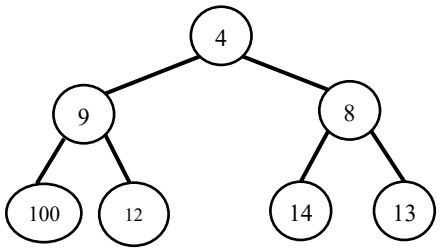

Figure 1. Heap corresponding to a priority queue.

3. The tree in Figure 1 is an ordered binary tree representing a heap. Is it a complete tree, a perfect tree, or a full tree? Answer without justification. A wrong answer counts negative within the contribution of problem A.2. (2pts if correct, 0 if not answered, -2pts if incorrect).

4. Is the priority queue captured by the tree in Figure 1 a min priority queue or a max priority queue? Answer without justification. A wrong answer counts negative within the contribution of problem A.2. (2pts if correct, 0 if not answered, -2pts if incorrect).

5. Priority queues can be efficiently implemented using arrays. Any heap "h" like the one in Figure 1 can be represented using an array (where |h| is the number of elements in the priority queue):

   $$\mathbf{arrayOf}(h) = [a_0, a_1, \dots, a_{|h|-1}]$$

   The array $\mathbf{arrayOf}(h)$ captures the binary tree corresponding to the heap h. Let h_0 be the heap captured in Figure 1. Give the sequence of elements appearing in $\mathbf{arrayOf}(h_0)$. (2pts).

6. Give the sequence of elements appearing in $\mathbf{arrayOf}(h_1)$ where h_1 is the heap obtained by inserting the element 10 to the priority queue captured by h_0. (2pts).

7. Assume the heap h_1 you obtained in the previous question (i.e., after adding 10 to the priority queue captured by h_0). Give the sequence of elements appearing in $\mathbf{arrayOf}(h_2)$ where h_2 is the heap obtained by performing a "pop" or "remove" operation from h_1. (2pts).

Problem A.3: Binary search trees (minimum 0pts, maximum 8 pts)

Recall that binary trees can be represented sequentially. In this problem (i.e., problem A.3), we adopt the approach described in 8.3.1 in OpenDSA. For instance, the binary tree in Figure 2 to the right can be represented using the sequence: "A B / D // C E G /// F H // I //". The symbol "/" is used to represent a "null" child. Do not draw your trees! Use this approach instead to answer the following questions.

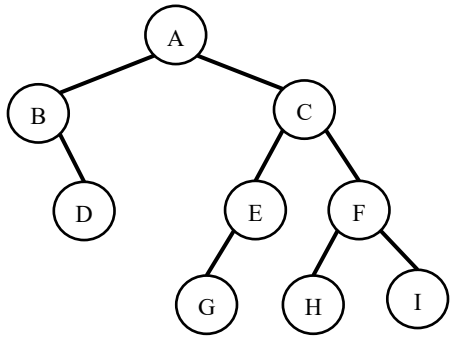

Figure 2. Only relevant for Problem A.3

Consider this sequence of 15 elements:

S: 12, 13, 14, 3, 11, 7, 9, 10, 5, 1, 15, 2, 6, 4, 8

8. Give a sequential representation (see the description of sequential representations of binary trees at the beginning of this problem) 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 12, then 13, then 14 …). **Do not try to balance the tree. Do not perform any splay operation in this question.** Call this tree T_1. (2pts).

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

10. Give a sequential representation of the tree T_2 obtained by removing the node containing key 8 from the tree T_1. **Do not try to balance the tree. Do not perform any splay operation in this question.** (2pts).

11. Consider now that the tree T_1 you obtained in question 8 (observe this question 11 is about the tree T_1 obtained in question 8 without any splay operations and without removing any node). Give a sequential representation of the tree T_3 resulting from performing a splay operation on the node containing key 1 in T_1. (2pts).

Problem A.4: Graphs and shortest paths (minimum 0pts, maximum 6pts)

Consider the graph in Figure 2.

12. Give, without justification, a topological sort of the nodes of the directed graph depicted in Figure 3. (2pt).

13. List the letters in the order they are processed in a **depth first traversal** that starts from node A in the graph of Figure 3. Observe the traversal needs not pass all nodes! Use the alphabetical ordering to break ties if any. (2pts)

14. List the letters in the order they are processed in a **breadth first traversal** that starts from node A in the graph of Figure 3. Observe the traversal needs not pass all nodes! Use the alphabetical ordering to break ties if any. (2pts)

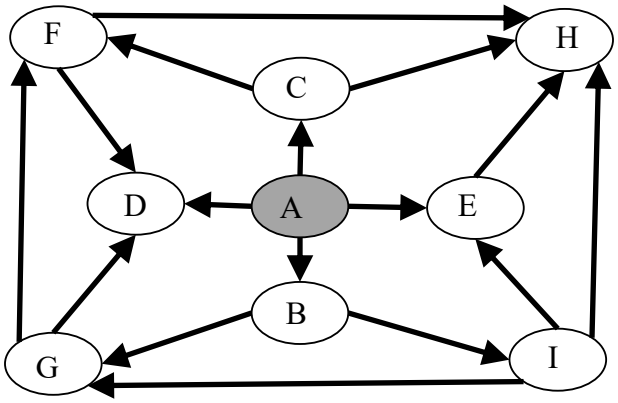

Figure 3. Directed graph for questions 12, 13 and 14.

Part B:

Problem B.1 (minimum 0pts, maximum 6pts)

Answer with yes or no (no need for justification). A wrong answer counts negative. All algorithms discussed in this question take an array of size n as input. The algorithms can handle arrays of any size. We are interested in the asymptotic time complexity of the algorithms, i.e., in the asymptotic number of steps required as the size of the inputs increases. A step here is assumed to take a constant amount of time.

15. Assume the **best-case** time complexity of an algorithm is in $\Omega(n \log n)$. Does this exclude the existence of an infinite sequence of arrays $a_1, a_2, \ldots$ (where the size of each array a_n is n) for which the algorithm always terminates in less than 10 steps? (1pts if correct, 0 if not answered, -1pts if incorrect).

16. Assume the **best-case** time complexity of an algorithm is in $O(n \log n)$. Does this exclude the possibility that the algorithm always terminates in less than 10 steps no matter the size of the input? (1pts if correct, 0 if not answered, -1pts if incorrect).

17. Assume the **best-case** time complexity of an algorithm is in $\Theta(n \log n)$. Does this exclude the existence of an infinite sequence of arrays $a_1, a_2, \ldots$ (where the size

of each array a_n is n) for which the algorithm always terminates in less than 10 steps? (1pts if correct, 0 if not answered, -1pts if incorrect).

18. Assume the **worst-case** time complexity of an algorithm is in $\Theta(n \log n)$. Does this exclude the possibility that the algorithm always terminates in less than 10 steps no matter the size of the input? (1pts if correct, 0 if not answered, -1pts if incorrect).

19. Assume the **worst-case** time complexity of an algorithm is in $\Omega(n \log n)$. Does this this exclude the existence of an infinite sequence of arrays $a_1, a_2, \ldots$ (where the size of each array a_n is n) for which the algorithm always terminates in less than 10 steps? (1pts if correct, 0 if not answered, -1pts if incorrect).

20. Assume the **worst-case** time complexity of an algorithm is in $O(n \log n)$. Does this exclude the existence of an infinite sequence of arrays $a_1, a_2, \ldots$ (where the size of each array a_n is n) for which the algorithm always terminates in less than 10 steps? (1pts if correct, 0 if not answered, -1pts if incorrect).

Problem B.2 (minimum 0pts, maximum 6 pts):

Assume a data structure where you can insert and remove letters. Suppose the sequence X Y U V W can be read only once, one letter at a time from left to right. Each time a letter is read, it can be either inserted in the data structure or it can be printed without insertion. A letter can also be removed from the data structure (if it was inserted of course) and printed. You can mix read operations (involving printing or insertion) and remove operations, i.e., you have access to operations of the form:

- read and print: read next letter and print it without insertion.
- read and insert: read next letter and insert it in data structure without printing
- remove and print: remove letter from data structure and print it.

You can only store letters in the considered data structure. A sequence of operations "read and print", "read and insert" and "remove and print" allows you to print the same letters as the sequence but in a possibly different order.

21. Can a sequence of such operations result in printing the sequence X U W V Y if the data structure is a stack? If your answer is yes, give the sequence of involved operations, if your answer is no, explain why. (3pts)

22. Can a sequence of such operations result in printing the sequence Y U W X V if the data structure is a stack? If your answer is yes, give the sequence of involved operations, if your answer is no, explain why. (3pts)

Problem B.3 (minimum 0pts, maximum 8 pts):

A polynomial expression p of degree n is an expression of the form:

$$p(x) = a_0 + a_1x^1 + \ldots + a_nx^n$$

Where x is a variable and $a_0, a_1, \ldots, a_n$ are constants. Both variable and constants are meant to be rational numbers and are captured using doubles.

23. Consider the pseudo code below. The method `evaluate` takes a constant reference to the coefficients of the polynomial p and a value for the variable x. It returns the value of the polynomial for the variable value x. Write a simple "C++ pseudo code"[1] for the `evaluate` method. Your code should only use for loops and simple (i.e., constant time) operations such as declarations of integer and double variables, assignments to such variables, additions or multiplications, or access to vector elements. Analyze the time complexity of your solution as a function of the degree n of the polynomial. Assume, in your analysis that you need to compute x^i using multiplication (and not by calling some primitive operation for power computations). (3pts)

24. Using the same assumptions as in the previous question, give "C++ pseudo code" for a solution that always returns the result in time complexity Θ(n), where n is the degree of the polynomial. (3pts)

```
#include <iostream>
#include <vector>

using namespace std;

double evaluate(const vector<double>& p, double x){
   …
}

int main(){
  int n;
  double x, a;
  vector<double> p;

  cout << "Give the polynomial degree n: \n" << endl;
  cin >> n;
  for(int i=0; i <= n; i++){
    cout << "coefficient a" << i << "?\n" ;
    cin >> a;
    p.push_back(a);
  }

  cout << "give value for variable x: \n" << endl;
  cin >> x;
  cout << "The value of the polynomial for x="
       << x << " is " << evaluate(p,x) << endl;

  return 0;
}
```

[1] Like the code above, "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 are not a problem.