Examination ID1020 Algorithms and Data Structures

2017-01-11

Examiners: Johan Karlander, Jim Dowling
Help Material: none
Examination questions do not need to be returned at the end of the exam.

Every submitted sheet should include the following information:

1. Name and personal number
2. Number for the attempted question
3. Page number

The exam consists of two parts. Part I contains questions to obtain grade E. Part II, which is not obligatory, contains four questions for grades A-D. Maximum writing time is four hours.

Every solution will be given a grade from 0-10 marks (points).

The requirement for a passing grade (E) is at least 50 marks in Part I. You have to receive a passing grade in Part I to qualify for a higher grade. You marks from Part I are not transferred to Part II.

On the condition that you have received a passing grade for Part I, the following are the grading ranges for Part II:

- Grade D: 5-10 marks
- Grade C: 11-20 marks
- Grade B: 21-30 marks
- Grade A: 31-40 marks

Solutions will appear on the courses homepage.

Part I Questions

The following questions are for grade E. You are required to give complete solutions and motivate your answers for all questions.

1. Give both the best-case and worst-case time complexity as a function of N for the following code fragments. Use the tilde notation. N is a large positive integer. All elements of the array a are distinct.

   a)

```
for (int i=0; i<N; i++)
  for (int j=N-1; j>0; j--)
    if (a[i]+a[j]== 0)
      cnt++;
```

   b)

```
for (int i=0; i<N; i++)
  if (a[i]==0)
    cnt++;
for (int j=N-1; j>0; j--)
  if (a[i]==1)
    cnt++;
```

   c)

```
// Assume that the N elements in the array 'a[]' are sorted in ascending order.

int i=0; int j=N-1; int k;
while(i<j) {
  k=(i+j)/2;
  if (a[k]==s)
    break;
  if (a[k] < s)
    j=k;
  else
    i=k;
}
```

   d)

```
for (int i =1; i<N; i=i*2)
  for (int j=1; j<N*N ; j++)
    cnt++;
```

2. The union find algorithms are used to solve dynamic connectivity problems. Assume we have a number of objects 0 to 9. Our initial state is that we have an array of size 10

that is initialized as [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

a) After a certain number of union operations, the underlying array in quick find looks as follows: [6, 3, 6, 3, 4, 3, 6, 4, 3, 6].
How many components are there? (1 pt)
How does the array look like after the operation union(2,3). (2 pts)
Alternatively, how does it look like after union(2,8). (1 pt)

b) After a certain number of union operations, the underlying array in weighted quick union looks as follows:[3, 6, 2, 3, 2, 3, 6, 4, 6, 2].
How many components are there? (1 pt)
How does the array look like after the operation union(7,1). (2 pts)

c) After a certain number of union operations, the underlying array in weighted quick union with path compression looks as follows: [3, 6, 2, 3, 2, 3, 6, 4, 6, 2].
How does the array look like after the operation union(7,1). (3 pts)

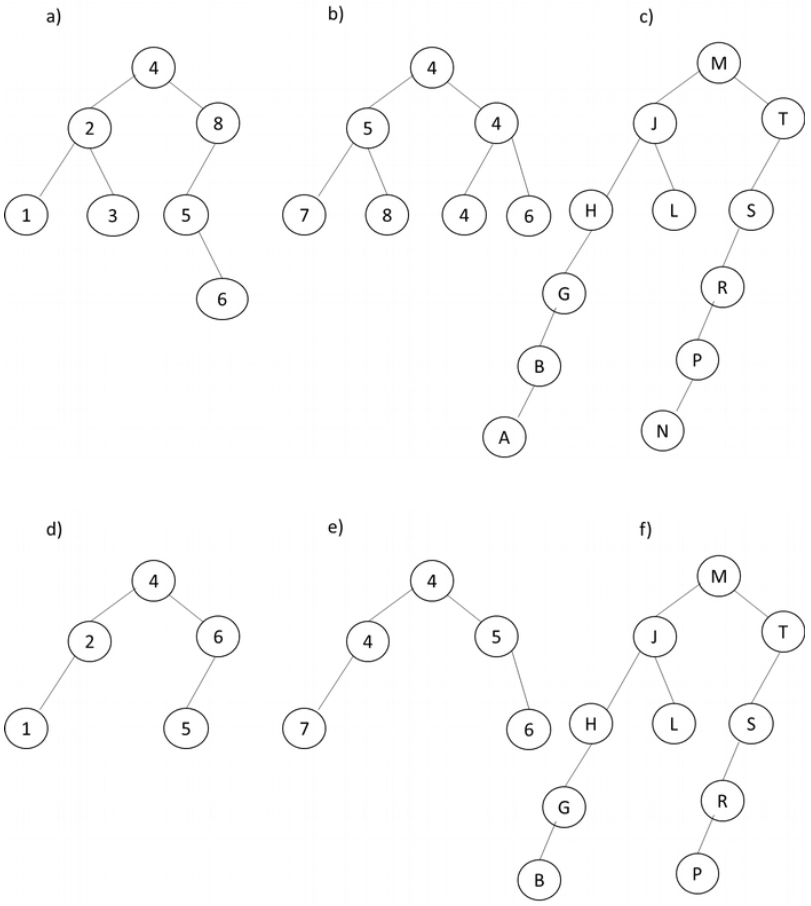
3. Assume that we start with the array [B1,D1,A1,A2,C1,B2,D2,A3] where the index is used only to keep track of occurences (that is A1, A2, and A3 have the same key and are duplicate elements).

   a) What does the array look like after sorting with selection sort?
   b) What does the array look like after sorting with mergesort?
   c) What does the array look like after sorting with standard quicksort (that uses the first elements as the partitioning element)?
   d) What does the array look like after sorting with insertion sort?

4. Show, step by step, how the following array is modified by sorting it using heapsort: [3,6,1,4,8,7,2].

5. Below, a number of binary trees are shown. In the subquestions below, you should list all of the trees that could possibly be the type that is looked for.

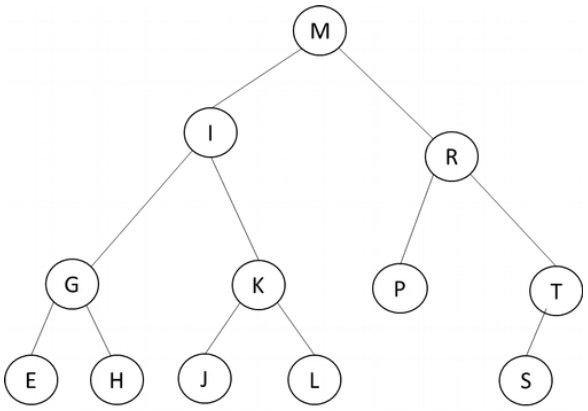
   a) Which tree could represent a minimum-oriented priority queue?
   b) Which could be a binary search tree?
   c) Which could represent a red-black binary search tree? (The colors of the links are not shown, so if you redraw some of the links red and the result is a valid red-black tree, the answer is yes).
   d) Which tree could represent the implicit tree in quick-union?



6. a) We investigate the behavior of a Java program that we have found on the Internet. The code is complex, so instead of extracting its algorithm from the code and analyzing it, we investigate it experimentally. The execution times for different input sizes are shown below. What is the approximate time complexity of the program

Input size	**Execution time**
1000	0.3 s
2000	0.6 s
4000	2.5 s
8000	13.5 s
16000	104 s
32000	829 s

b) Describe a method to remove an element from a binary search tree (1-2 sentences will suffice). Then, show how the tree would look like after removing the element in the tree below.



7. The standard recipe for resizing the backing array for a linear probing hash table is to keep the number of stored elements within the bounds of N/2 (upper bound) and N/8 (lower bound), where N is the size of the array.

   a) Consider the case where one reaches the upper bound (N/2). How are the elements distributed in the array in the best case? How many array accesses are needed on average for a search miss (that is, one searches for an element X that can be hashed to any possible value)?

   b) Consider the case where one reaches the upper bound (N/2). How are the elements distributed in the array in the worst case? How many array accesses are needed on average for a search miss (that is, one searches for an element X that can be hashed to any possible value)?

   c) Consider the case where one reaches the lower bound (N/8). How are the elements distributed in the array in the best case? How many array accesses are needed on average for a search miss (that is, one searches for an element X that can be hashed to any possible value)?

   d) Consider the case where one reaches the lower bound (N/8). How are the elements distributed in the array in the worst case? How many array accesses are needed on average for a search miss (that is, one searches for an element X that can be hashed to any possible value)?

8. Describe an algorithm that, given an undirected graph G, decides if the graph contains at least one cycle. The algorithm has time complexity $O(|V|)$ where V is the number of nodes. (Note that the main difficulty in this question is proving the time complexity).

9. We study a sketch of a method for finding a topological ordering of an acyclic directed graph. The algorithms looks as follows: we first find a node with indegree 0. We assign that node the number 1. We remove the node from the graph. We now find a node with indegree 0 in the new graph, remove the node from the graph, etc. We continue until all the nodes are numbered.

   - How do do we know that at each step we can find a node with indegree 0?
   - How do we know that the method gives a topological ordering?

- If the algorithm is implemented in the most efficient way, what time complexity does it have?

10. Let G be an undirected graph with edge weights. Let T be a minimal spanning tree of G and e an edge in T. Let us now assume that edge weights $w(e)$ are changed to $w(e) - 7$. Let G' be exactly like G apart from the changes to the weights of e. Is T a minimal spanning tree of G as well? Give a proof or a counter-example.

Part II Questions

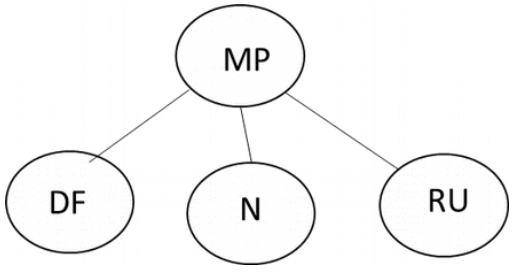
The following questions are for grade A-D. You are required to give complete solutions and motivate your answers for all questions.

1. Assume an array A of size N that contains strings as elements. Write an algorithm (pseudocode or Java) that solves the subproblems below. For this question, you should only use algorithms that have been presented in the course. Be clear on which algorithm(s) you are using (and any variation thereof) as well as the methods you are using. You should also give both the time and memory complexity of your solution in terms of N.

   a) Print out the number of distinct strings.
   b) Print out the number of different strings that appear at least 2 times in A.
   c) Print out the the M elements that appear most frequently. Print them out as a sequence of pairs (string, number of appearances). Order the pairs by the number of appearances, with the most appearances coming first.

2. A 2-3 tree is shown below. Draw the corresponding red-black binary search-tree. Then, show, step by step, all rotations and color flips that are needed what you add a key T. Then, draw the 2-3 tree that corresponds to the resultant red-black binary search-tree.

   Nedan visas ett 2-3 träd. Rita motsvarande röd-svart binärt sökträd. Sedan visa steg för steg alla ev. rotationer och färgflippning som behövs när man lägger till nyckeln T. Slutligen visa det slutliga motsvarande 2-3 träd.



3. For this question, you shall use the algorithms that have been presented in the course. Be clear on which algorithm(s) you are using (and any variation thereof) as well as the methods you are using (for example, you could write Quicksort (a) and mention that you are using the variant that uses insertion sort when the length of the list is less than 10.

a) Write a program (Java or pseudocode) that, given a text file, creates a concordance register (by using a data structure from the course). Such a register keeps track of all places where a word appears in the text. When the table is called with $get(string)$, it should return, in order, all the places in the text where the word appears. For example, if the text file is to be or not to be that is the question", then $get(\text{“}to'')$ would return $[1, 5]$, since to"is both the first and fifth word in the text. Give the time complexity in terms of N (the length of the text file in number of words) and also U, the number of unique words in the document.

b) Write a program (Java or pseudocode) that, given the concordance register from part (a), prints out the original text (in order). Also, give the time complexity in terms of N (the length of the text file in number of words) and also U, the number of unique words in the document.

4. Let G be a directed acyclic graph. Such a graph is naturally not strongly connected. But sometimes, it can happen that we can make it strongly connected by adding just one directed edge. A simple example is if the graph is a directed path that goes from a to b. We can then add the directed edge (b, a). Construct an algorithm that, given G, decides if G has this property. You can assume that G is acyclic, and you algorithm does not need to check if G is acyclic.