CSCI 470 Practice Exam 02: Solutions

Date: April 3, 2024

Full Name:

Student ID:

DIRECTIONS:

- Write your full name, and student ID above.
- Write your answers on the exam paper.
- You are allowed one cheat sheet.
- If you need extra space, please use the back of a page.
- You have 80 minutes to complete the exam.
- Please do not turn the exam over until you are instructed to do so.
- Good Luck!

1. For each of the following problems, answer **True** or **False** and BRIEFLY JUSTIFY your answer.

   (a) The smallest possible depth of a leaf in a decision tree for a comparison sort is $n - 1$, where n is the number of elements in an array to sorted. *(You may use insertion sort as an example.)*

   Ans: **True**. In case of an insertion sort, the inner loop runs the least with when the input is already a sorted array in non-decreasing order. In this case, we only make $n - 1$ comparisons. Every time we compare the key with the elements left to it, $A[i] < key$, which prevents the inner loop from running.

   (b) A binary-search tree will always have its height $O(\lg n)$.

   Ans: **False**. A binary-search tree may have depth of $n - 1$ if the smallest value is at the root, and the rest of the values are inserted to the right of the root in an increasing order.

   (c) Searching an element in a linked-list with n objects takes, $\Theta(n \lg n)$.

   Ans: **False**. At most we need to iterate over all the nodes, which will take $\Theta(n)$ to search an element in a linked list.

(d) The total running time of the BFS procedure is $O(V + E)$.

Ans: **True**. The operations of enqueuing and dequeuing take $O(1)$ time, and so the total time devoted to queue operations is $O(V)$. Because the procedure scans the adjacency list of each vertex only when the vertex is dequeued, it scans each adjacency list at most once. Since the sum of the lengths of all the adjancecy lists is $\Theta(E)$, the total time spent in scanning adjacency lists is $O(E)$. The overhead for initialization is $O(V)$, and thus total running time of the BFS procedure is $O(V + E)$.

(e) The following procedure returns the successor of node x in a binary-search tree.

TREE-SUCCESSOR(x)

```
1  if x.right ≠ NIL
2       return TREE-MAXIMUM(x.right)
3  y = x.p
4  while y ≠ NIL and x == y.right
5       x = y
6       y = y.p
7  return y
```

Ans: **False**. Line 1-2 returns the maximum value in the right subtree of node rooted at x, which can be the successor of x as the maximum value is not always next largest value to $x.key$.

2. Here is the pseudocode of COUNTING-SORT:

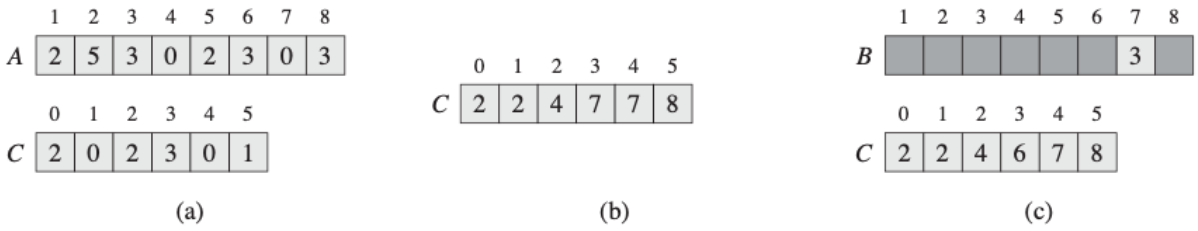
```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Suppose that we were to rewrite the **for** loop header in line 10 of the COUNTING-SORT as

```
10  for j = 1 to A.length
```

Show that the algorithm still works properly.

Ans: The algorithm still works. Iterating from first element to the last element, we will still be using the running sum in C, which helps us drop the elements in a sorted order. Consider the example as shown in the book.



When we iterate starting from 1 to $A.length$, we encounter 2 first, which will be placed at index 4. Later the other 2 will be placed at index 3. More importantly, we will place two 2s in the array in index 3 and 4 despite iterating from $A.length$ to 1 or other way round. The same repeats for all the elements in the array.

(Please use this page if the previous page is not enough to write your answers.)

3. Hash tables

   (a) Briefly describe what is a collision in a hash table.

   Given a table T, and hash function h, when $T[h(x_i.key)] = T[h(x_j.key)]$ for any two x_i and x_j nodes, such situation is called collision in a hash table.

   (b) Argue why collision is inevitable even with a perfect hashing.

   Given $|U|$ is the set of keys to be stored in a table of size m. When $|U| > m$, even with a perfect hashing, collision is inevitable.

(c) Describe a method to resolve collision in a hash table.

Chaining can be used to resolve collision in a hash table. Each collided nodes hashed to the same location in the hash table can be chained together using a linked list.

(d) State the runtime of a successful search in a hash table in terms of the ***load factor***, α. If $n = O(m^3)$, where n is the number of keys, and m is the number of slots in a hash table, what would be the upper bound of searching an element in the hash table on average?

We know that on an average, the runtime for a successful search in a hash table is $\Theta(1+\alpha)$, where $\alpha = n/m$. If $n = O(m^3)$, then $\Theta(1 + \alpha) = \Theta(1 + n/m) = \Theta(1 + O(m^3)/m) = \Theta(1 + m^2) = \Theta(m^2)$. Therefore, the upper bound of searching an element in this table on average will be $O(m^2)$.

4. BFS

   (a) Write the pseudocode to find the shortest path from s to v, given we have already run a breadth-first search on the graph.

```
PRINT-PATH(G, s, v)
1  if v ==s
2      print s
3  elseif v.π == NIL
4      print "no path from" s "to" v "exists"
5  else PRINT-PATH(G, s, v.π)
6      print v
```

(b) Find out if there is a shortest path from node P to U using a BFS procedure. You must use a predecessor graph to figure out if there is a shortest path from P to U or not.

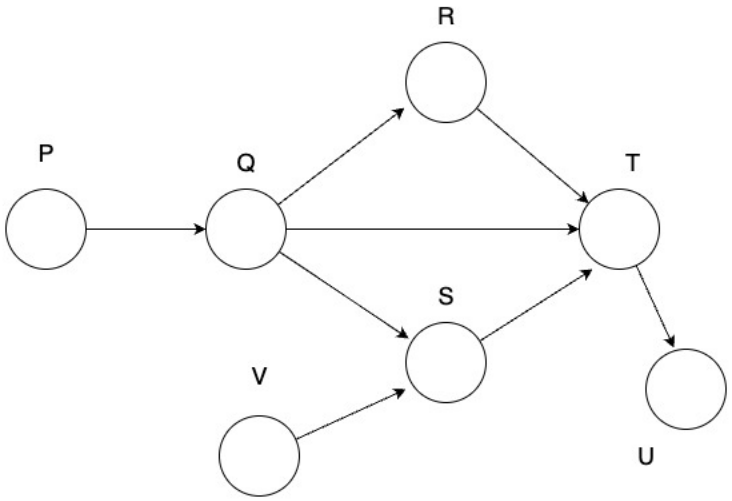


Figure 1

vertex	P	Q	R	S	T	U	V
d	NIL	P	Q	Q	Q	T	NIL
π	0	1	2	2	2	3	∞

Using the table, we can trace the shortest path from P to U.

$P \to Q \to T \to U$.

(Please use this page if the previous page is not enough to write your answers.)

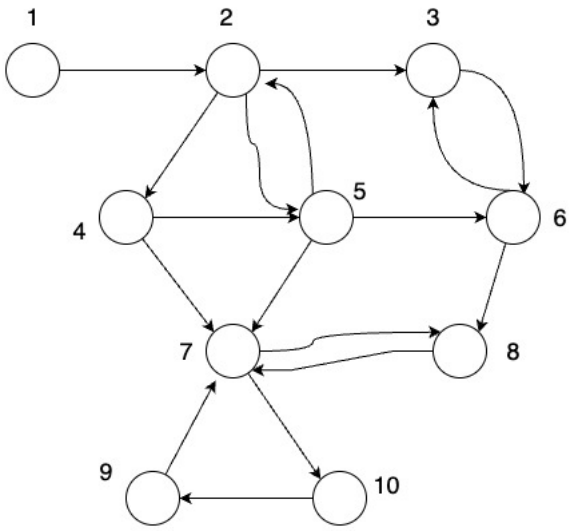


Figure 2

5. DFS

   (a) Run a DFS on the graph above. Assume that the iteration over the nodes is in an increasing order, and assume that each adjacency list is also ordered in an increasing order.

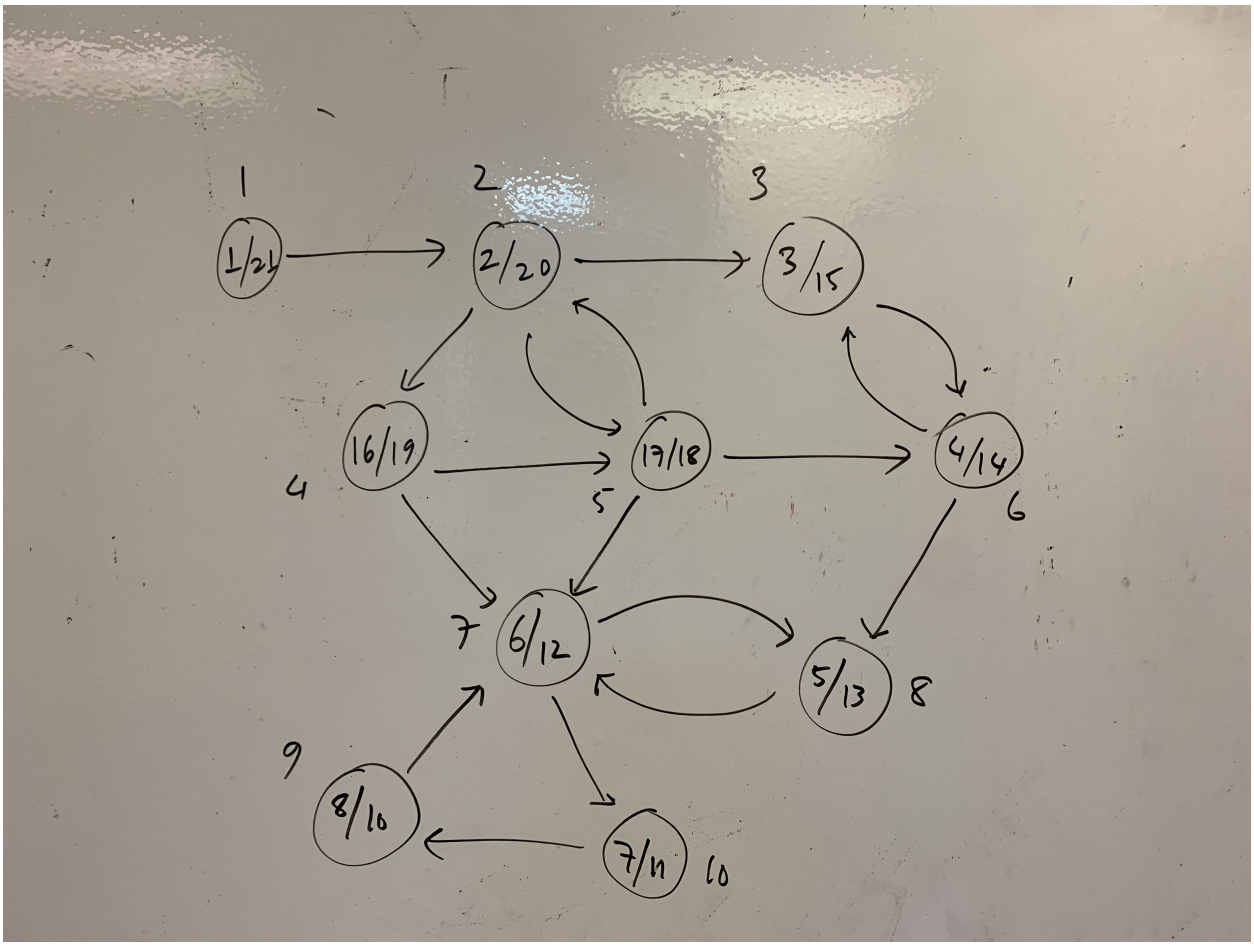


Figure 1: The graph populated with discovery and finishing times as we run DFS on the graph.

(b) Transpose the graph above.

We reverse the direction of all edges in the graph. It's shown in the next figure.

(c) Run another DFS on the graph based on the decreasing finishing times of the nodes computed in 5(a).

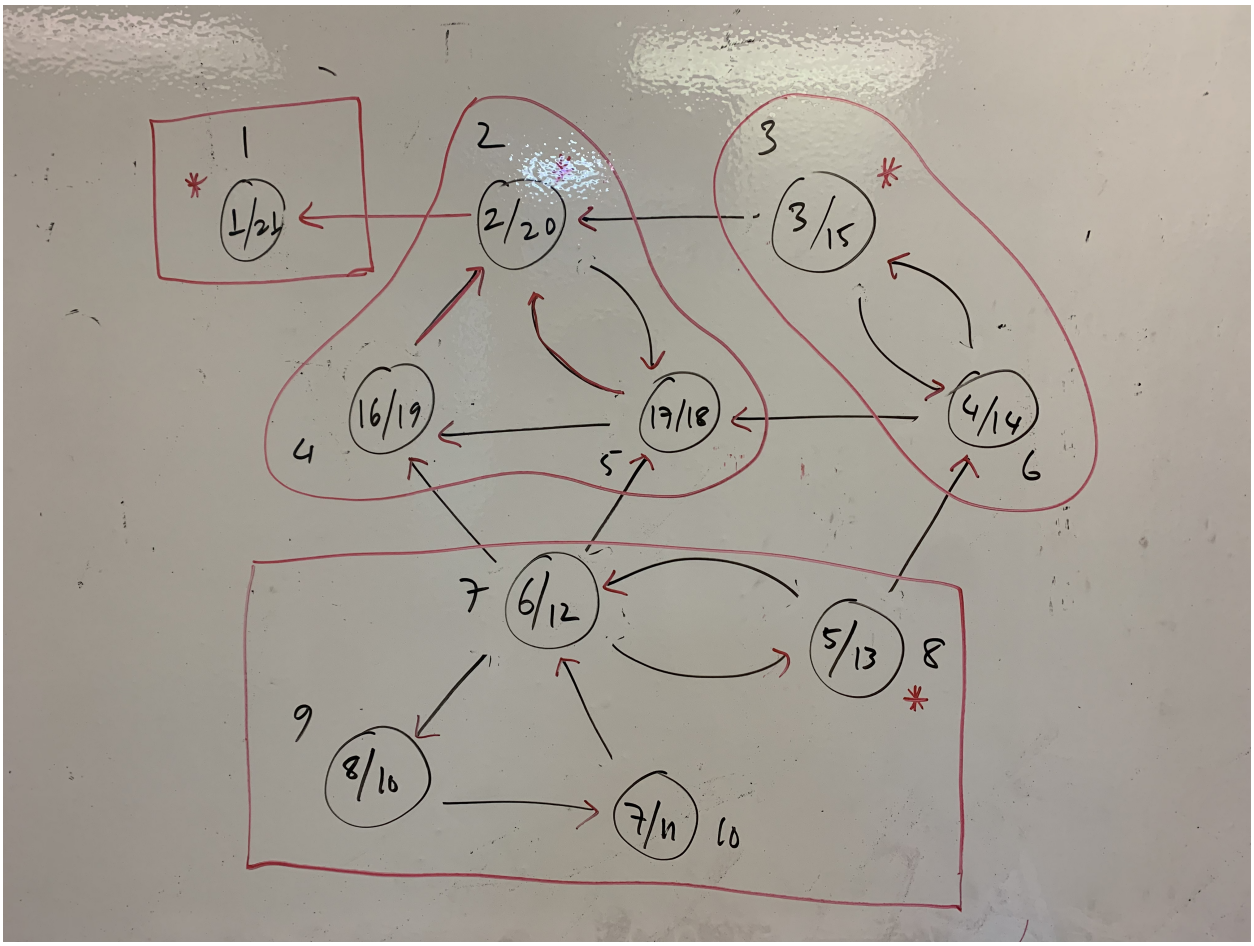


Figure 2: We start the DFS run on the transposed graph, where we start from the node that has the highest finishing time. With each DFS run based on the decreasing finishing time, we collect the strongly connected components. Here are the strongly connected components: $\{1\}, \{2, 5, 4\}, \{3, 6\}, \{8, 7, 9, 10\}$

(d) List the nodes of each tree in the depth-first search forest created by the DFS on G^T.

As we run the DFS run on G^T using the decreasing finishing times. Here are the strongly connected components: $\{1\}, \{2, 5, 4\}, \{3, 6\}, \{8, 7, 9, 10\}$.

(e) Write the ordering of vertices which are topologically sorted using the computation in 5(a), if there is a valid topological order.

There are no valid topological ordering in this graph because there are cycles in the graph. Topological ordering is only possible in directed acylic graphs (DAGs).

(Please use this page if the previous page is not enough to write your answers.)