Data Structures and Algorithms (COMP-2540)
Final Exam., 23 Aug., 2021
Total Marks : 40
Time: 3 hours

Instructions:

- This exam is closed-notes, closed-book; use of any material in electronic form is also not allowed.
- There are 20 questions each worth 2 points. Answer all 20 questions.
- For a multiple-choice question *clearly* write down the single choice that you believe is the most accurate. For a TRUE/FAlSE question, *clearly* write down TRUE or FALSE.
- You must sign and date the declaration below.

SID: 105213585 **Name:** Tyler Hodges Johnston

Declaration: I have not received any unauthorized help from any external source in completing this test.

Signature and Date:
Tyler Hodges Johnston Aug. 23, 2021

1. Consider the following algorithm to sort an array of n integers $a[0], a[1], \ldots, a[n-1]$ in *non-decreasing* order.

```
Algorithm mysterySort(int a[], int n)
{
        for(int j = n-2; j >= 0; j--)
           for (int i = 0; i <= j ; i++)
               if (a[i+1] < a[i]) exchange a[i] and a[i+1];
}
```

Assertion: Independent of the initial input array, the number of comparisons made by the above sorting algorithm is always $n(n-1)/2$.

Is the above assertion TRUE or FALSE ?

Ans: True

2. In Lab 3, you used a stack to implement an algorithm to check if a parenthesis sequence is balanced or not (for example, the parenthesis sequence (())() is balanced, while ((() is not).

It is possible to use a simple counting algorithm to check if a parentheses sequence is balanced. Initialize a variable *count* to 0 (this is because an empty parenthesis sequence is assumed to be balanced). Scan the input sequence from left to right. Increment count by 1 when an opening parenthesis is encountered and decrement by 1 when a closing parenthesis is encountered. The algorithm returns the value of *count* upon reading the input sequence completely.

The input parenthesis sequence is balanced if the final value of *count* is 0, otherwise not.

Assertion: There exists a balanced parenthesis sequence of length $2n$, where n is a positive integer, for which the variable *count* assumes the value 1(one) n times before the algorithm finally returns a value of 0.

Is the above assertion TRUE or FALSE ?

Ans: True

```
// declarations
{
  if (knownF[i] != 0) return knownF[i];
  int t = i ;
  if (i < 0) return 0;
  if (i > 1) t = F(i-1) + F(i-2);
  return knownF[i] = t;
}
```

3. In Lab 4, you implemented an algorithm, based on dynamic programming (see above for a description), for computing an element F_n ($= F(n)$ in the subroutine above) of a Fibonacci sequence for some $n \geq 0$.

   Observe that once you compute F_n using the above subroutine, all the Fibonnaci numbers up to and including F_n are available in the array $knownF[]$. Use this to answer the following question.

   Lucas numbers L_n can defined by setting $L_0 = 2$, $L_1 = 1$ and $L_n = F_{n+1} + F_{n-1}$ for $n > 1$.

   Answer if the following claim is TRUE or FALSE: "Enough information is available to compute L_n for $n > 1$ by one call to the above subroutine for computing F_{n+1}".

   Ans: True

4. In Lab 5, you implemented a divide-and-conquer algorithm for finding a maximum contiguous sub-sequence of a sequence of n integers. The time-complexity $T(n)$ of this algorithm is:

   (A) $O(n)$
   (B) $O(n \log n)$
   (C) $O(n^2)$
   (D) None of the above.

   Ans: B

5. In Lab 6, you implemented an algorithm based on dynamic programming to determine the minimum number of coins, $min(k)$, of denominations $C_1, C_2, \ldots, C_n$ needed to make change for an amount k.

   If we apply a greedy algorithm to this problem, in each step we always choose the coin of the largest denomination that is not greater than the residual amount.

Assume that only coins of denominations 1, 10 and 25 are available.

Assertion: Applying the greedy algorithm to make change for the amount 84 uses 12 coins, whereas applying the dynamic programming algorithm uses 9 coins.

Is the above assertion TRUE or FALSE ?

Ans: True

6. In Lab 7, you implemented the *quickSort* algorithm, following the partition procedure, as explained in class and in the courseware. As you well-know, in the worst-case this algorithm takes $O(n^2)$ time.

   Assertion: A worst-case input for this implementation of *quickSort* is when the input list is $n, n-1, \ldots, 1$, for some $n \geq 2$, and the list must be sorted in increasing order.

   Is the above assertion TRUE or FALSE ?

   Ans: True

7. This refers to Lab 8. Assume that the keys in a binary search tree are distinct and in the range from 1 to 10 whose root has key value 3. Suppose we initiate a search for the key 6 in this tree, assumed to be present in the tree. Which one of the following sequences could not be the sequence of keys examined on the path from the root to the node containing the key 6 ?

   (A) 3, 1, 2, 6
   (B) 3, 7, 5, 6
   (C) 3, 8, 7, 6
   (D) 3, 5, 10, 9, 6
   (E) None of the above.

   Remember: A binary search tree has the property that the key value at each node is greater than the key values in the left subtree and less than the key values in the right subtree. This assumes that all key values are distinct.

   Ans: A

8. This refers to Lab 9. To implement *heapSort*, we used an essentially complete binary tree, with the key values in the tree nodes satisfying the max-heap property, namely that the key value at any node is greater than or equal to the key values of its children, if any.

If we represent the list 5, 1, 4, 9, 8, 2, 7 as an essentially complete binary tree, we see that the max-heap property is not satisfied. Now, if we invoke the *buildheap* procedure on this list, where heapification is done bottom-up, the resulting max-heap is described by one of the lists (A), (B), (C) or none of these. Choose the correct answer.

(A) 9, 8, 7, 5, 4, 2, 1
(B) 9, 8, 7, 1, 5, 2, 4
(C) 8, 9, 5, 7, 2, 4, 1
(D) None of the above

Hint: Tree drawings would be helpful.

Ans: B

9. The Binary Search Tree of Figure 1 is not AVL-balanced. The imbalance was caused by the insertion of the key 50. AVL-balance can now be restored by an:

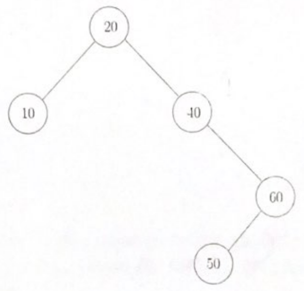


Figure 1: *Figure for Qn. 9, Part I*

(A) LR rotation
(B) RL rotation
(C) LL rotation
(D) RR rotation

Hint: Identify the node at which the tree has become imbalanced.

Ans: B

10. Let S be a set of n keys. Consider implementing dictionary operations on S: that is, search for, insert into and delete a key from S efficiently.

Assertion: Using a hashtable we can implement these dictionary operations that on average take $O(1)$ time, under the assumption of uniform hashing.

Is the above assertion TRUE or FALSE ?

Ans: True

11. This is based on your Lab 10 work. Suppose we implement an open-address hashing scheme with *linear probing* to insert keys into a hashtable of size 29. To insert a key k into the hashtable, the probe sequence, $h_0(k), h_1(k), h_2(k), \ldots$ is generated using the formula $h_i(k) = (h_0(k) + i) \bmod 29$, for $i = 1, 2, \ldots$, with $h_0(k) = h(k)$ defined as below:

$$h(k) \quad = \quad \lfloor n(kA \ mod \ 1) \rfloor, \tag{1}$$

where $n = 29$ is the size of the hashtable and $A = 0.62$ is an empirically chosen constant.

If an element with key value $k = 59$ is to be inserted into the hashtable and the first three locations successively probed are found to be already occupied, then the next location in the hashtable that will be probed is:

(A) 11
(B) 19
(C) 20
(D) None of the above

Ans: B

12. In a height-balanced AVL tree T, the balance factor at each node in the tree is defined to be the value of the difference, $h_L - h_R$, where h_L and h_R are, respectively, the heights of the left and right subtrees of the node.

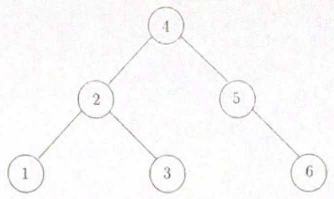
Assertion: T is AVL-balanced if the balance factor at each node of T is +1, -1 or 0.

Is the above assertion TRUE or FALSE ?

Ans: True

13. Starting with an empty binary search tree, we insert the key values 1, 2, 3, 4, 5, 6 into this tree successively, maintaining AVL-balance after each insertion.

This gives rise to the following final AVL-tree:



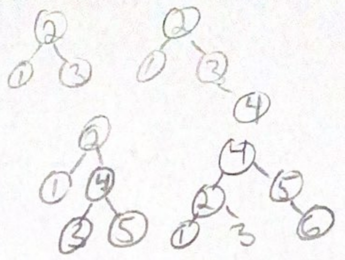


Figure 2: *Figure for Qn. 11*

requiring the following sequence of intermediate rebalancing rotations:

(A) RL, LR, LL
(B) RR, RR, RR
(C) LR, RL, RR
(D) LL, LL, LL
(E) None of the above

Ans: D

14. We use Huffman's algorithm to obtain an encoding of the alphabet $\{a, b, c\}$ with frequencies $\{f_a, f_b, f_c\}$.

Assertion: No matter what frequencies are assigned to f_a, f_b, f_c, it is impossible to obtain the codeswords, 10, 00, 01, for a, b and c respectively.

Is the above assertion TRUE or FALSE ?

Ans: True

15. Recall that Huffman's algorithm generates prefix-free codewords for letters in a text, T, given the frequencies of their occurrences in T.

Assertion: It is a greedy algorithm that constructs a full-binary tree (from which codewords are generated) by merging at each step two trees in the current forest with the *highest* frequencies.

Is the above assertion TRUE or FALSE ?

Ans: False

16. Let $G = (V, E)$ be an undirected, connected graph whose vertex set is V and edge set is E. We perform Breadth First Search on G from a given source vertex S.

Assertion: The time-complexity of constructing a Breadth First Search Tree, which gives the shortest paths (measured by the number of edges) from S to all other vertices, is $O(|V| * |E|)$, where $|V|$ and $|E|$ are respectively the number of vertices and edges of G.

Is the above assertion TRUE or FALSE ?

Ans: False

17. Refer to the graph of Figure 3. We use the *breadth-first search* algorithm to find the shortest path from the node labeled D to all the other nodes. Note that the shortest path from a node x to a node y is measured by the number of edges on that path.

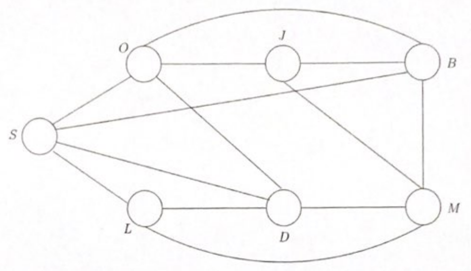


Figure 3: *Figure for Qn. 3, Part II*

Assertion: Are the distances from D to the other vertices shown (inside the circles, next to the corresponding vertex label) in the Breadth-First Search (BFS) Tree below correct ?

Answer TRUE or FALSE.

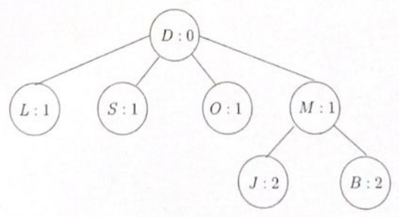


Figure 4: *BFS Tree*

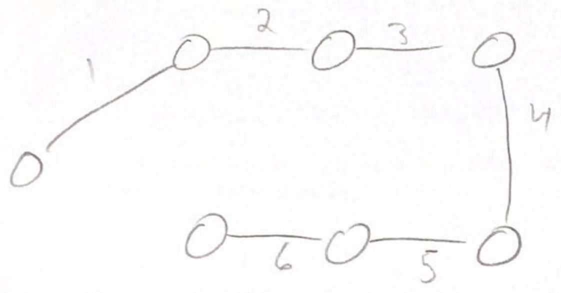
Ans: True

18. Once again, let $G = (V, E)$ be an undirected, connected graph whose vertex set is V and edge set is E. A spanning tree of G is a graph T on the same set of vertices as G but has only a subset of edges of G. T is minimally connected (removing an edge disconnects T) but maximally acyclic (adding an edge creates a cycle).

Assertion: The graph of Fig. 5 has a spanning tree consisting of 6 edges (ignore the weights of the edges).

Is the above assertion TRUE or FALSE ?

Ans: True



19. Given the weighted, undirected graph of Figure 5, we find a Minimum-Weight Spanning Tree (MST for short) using Prim's algorithm, starting from the vertex labeled S.

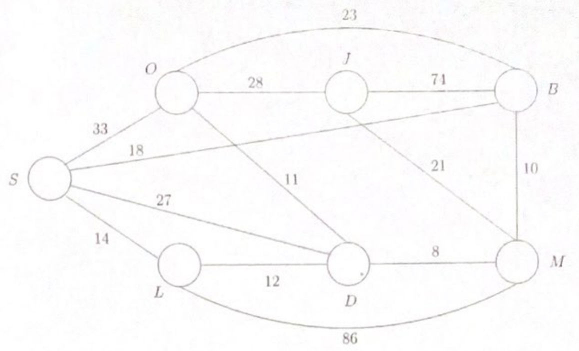


Figure 5: *Figure for Qn. 4, Part II*

For this, we used the priority table method. Shown in the figure below is the resulting priority table with some entries missing.

	S	B	D	J	L	M	O
$\{\}$	0/S	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
$\{S\}$	×	$18/S$	$27/S$	∞/nil	$14/S$	∞/nil	33
$\{S, L\}$	×	$18/S$	$12/L$	∞/nil	×	$86/L$	33
$\{S, L, D\}$	×	$18/S$	×	∞/nil	×	$8/D$	11
$\{S, L, D, M\}$	×	$10/M$	×	$21/M$	×	×	11
$\{S, L, D, M, B\}$	×	×	×	$21/M$	×	×	11
$\{S, L, D, M, B, O\}$	×	×	×	$21/M$	×	×	×
$\{S, L, D, M, B, O, J\}$	×	×	×	×	×	×	×

Figure 6: *Figure for Qn. 4, Part II*

The missing entries in the column labeled, O, in increasing row order are:

(A) ∞/nil, ∞/nil, $33/S$, $33/S$, $11/D$.
(B) ∞/nil, $33/S$, $33/S$, $33/S$, $11/D$.
(C) $33/S$, $33/S$, $11/D$, $11/D$, $11/D$.
(D) None of the above

Ans: C

20. The weighted, directed graph of Figure 7 shows (fictitious) distances in airmiles between some pairs of US airports. The priority table method was usied to compute the shortest paths from S (San Francisco) on the West coast to all other reachable airports, using Dijkstra's algorithm.

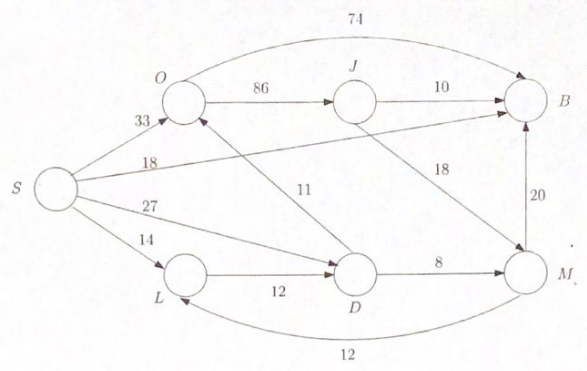


Figure 7: *Figure for Qn. 5, Part II*

In the priority table shown below entries of the row labeled $\{S, L\}$ are missing.

	S	B	D	J	L	M	O
$\{\}$	0/S	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
$\{S\}$	$\times$	$18/S$	$27/S$	∞/nil	$14/S$	∞/nil	$33/S$
$\{S, L\}$	X	18	26	$\times$	X	X	33
$\{S, L, B\}$	$\times$	$\times$	$26/L$	∞/nil	$\times$	∞/nil	$33/S$
$\{S, L, B, D\}$	$\times$	$\times$	$\times$	∞/nil	$\times$	$34/D$	$33/S$
$\{S, L, B, D, O\}$	$\times$	$\times$	$\times$	$119/O$	$\times$	$34/D$	$\times$
$\{S, L, B, D, O, M\}$	$\times$	$\times$	$\times$	$119/O$	$\times$	$\times$	$\times$
$\{S, L, B, D, O, M, J\}$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$

Figure 8: *Priority Table for Shortest Path Dijkstra*

Which one of the choices below correctly describes the missing entries of this row:

(A) $\times$, $18/S$, $27/S$, ∞/nil, $\times$, ∞/nil, $33/S$
(B) $\times$, $18/S$, $26/L$, ∞/nil, $\times$, ∞/nil, $33/S$
(C) $\times$, $18/L$, $26/L$, ∞/nil, $\times$, ∞/nil, $33/S$

(D) None of the above.

Ans: B