2.5 Data Structures and Their Operations: 30-minute Pack

Total Marks

OCR H446 · Calculators not allowed · about 30 minutes

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A calculator app developed by a startup in Manchester evaluates arithmetic expressions that have already been converted into postfix (Reverse Polish) notation, using a stack. Tokens are scanned left to right: when a number is read, it is pushed onto the stack; when an operator is read, the top two values are popped (the second-popped value, a, is the left operand and the first-popped value, b, is the right operand), the operation a operator b is applied, and the result is pushed back onto the stack.

Consider the postfix expression: 6 2 3 + * 4 -

(a) Complete a trace table showing the contents of the stack after each token in the expression has been processed. **(6)**

(b) State the final value of the expression. **(1)**

(c) Explain why a stack, rather than a queue, is the appropriate data structure for evaluating postfix expressions in this way. **(2)**

(Total for Question 1 is 9 marks)

© Revision Library | revisionlibrary.co.uk

2 A route-planning tool used by a delivery company in Leicester models six delivery hubs, Amir, Beth, Carlos, Dana, Elif and Farah, and the direct roads connecting them, as an undirected graph. The roads are: Amir-Beth, Amir-Carlos, Beth-Dana, Carlos-Dana, Carlos-Elif, Dana-Farah, Elif-Farah. When visiting the neighbours of a hub, they are always considered in alphabetical order.

(a) Perform a breadth-first search (BFS) of the graph starting at Amir, using a queue. Show the changing contents of the queue and state the order in which the hubs are visited. **(4)**

(b) Perform a depth-first search (DFS) of the graph starting at Amir, using a stack (or recursion), always exploring the first unvisited neighbour in alphabetical order before backtracking. State the order in which the hubs are visited. **(4)**

(c) Explain, with reference to the traversals performed in (a) and (b), one situation in which BFS would be preferred over DFS. **(2)**

(Total for Question 2 is 10 marks)

3 A library index system used in a school in Preston stores book reference numbers in a binary search tree (BST), so that they can be searched efficiently. Starting from an empty tree, the following values are inserted, one at a time, always starting comparisons at the root: 50, 30, 70, 20, 40, 60, 80, 35. The standard BST insertion rule is used: at each node, if the new value is less than the node's value, move left; otherwise, move right; the value is inserted at the first empty position reached.

(a) Determine the position of 35 in the completed tree, by comparing it against the root and each node subsequently reached, stating at each step whether the comparison sends it left or right. Hence state the parent of 35 and whether it becomes the left or right child of that parent. **(3)**

(b) State the value stored at the root of the completed tree. **(1)**

(c) Write out the pre-order traversal of the completed tree. **(3)**

(d) Write out the in-order traversal of the completed tree, and state what property of a binary search tree this result demonstrates. **(3)**

(Total for Question 3 is 10 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.5 Data Structures and Their Operations: 30-minute Pack

Question 1

(a) after 6, 2, 3 are read: stack = [6, 2, 3] (three pushes) **B1**

(a) '+' : pops 3 (b) and 2 (a), computes 2 + 3 = 5, pushes 5: stack = [6, 5] **B1**

(a) '*' : pops 5 (b) and 6 (a), computes 6 * 5 = 30, pushes 30: stack = [30] **B1**

(a) after 4 is read: stack = [30, 4] **B1**

(a) '-' : pops 4 (b) and 30 (a), computes 30 - 4 = 26, pushes 26: stack = [26] **B1**

(a) final stack correctly shown as [26] with all intermediate values consistent **B1**

(a) Answer: Stack after each token: [6] -> [6,2] -> [6,2,3] -> [6,5] -> [30] -> [30,4] -> [26]

(b) 26 ft from (a) **B1**

(b) Answer: 26

(c) an operator always needs to combine with the operands nearest to it (most recently pushed), which matches a stack's LIFO behaviour (oe) **B1**

(c) a queue would return items in the order they were first inserted (FIFO), which would not give the operator the correct, most recent operands needed for evaluation (oe) **B1**

(c) Answer: An operator always needs to act on the two most recently seen operands, which is exactly what a stack's LIFO behaviour provides; a queue's FIFO behaviour would instead return the oldest values first, giving the wrong operands.

Question 2

(a) visits Amir, then enqueues its unvisited neighbours in alphabetical order, Beth then Carlos: queue = [Beth, Carlos] **M1**

(a) dequeues and visits Beth before Carlos (FIFO order), correctly enqueuing each hub's new unvisited neighbours as they are found (Dana when processing Beth; Elif when processing Carlos; Farah when processing Dana) **M1**

(a) final visit order: Amir, Beth, Carlos, Dana, Elif, Farah cao **A1**

(a) correctly states that the algorithm terminates once the queue becomes empty, with every hub visited exactly once **B1**

(a) Answer: Amir, Beth, Carlos, Dana, Elif, Farah

(b) visits Amir, then immediately visits Beth (its first unvisited neighbour), rather than visiting all of Amir's neighbours first **M1**

(b) correctly continues deep from Beth to Dana, then from Dana to Carlos (Beth already visited), then from Carlos to Elif, then from Elif to Farah, backtracking only once no unvisited neighbours remain **M1**

(b) final visit order: Amir, Beth, Dana, Carlos, Elif, Farah cao **A1**

(b) correctly identifies that DFS uses a stack (explicitly, or implicitly via the recursive call stack), in contrast to the queue used by BFS **B1**

(b) Answer: Amir, Beth, Dana, Carlos, Elif, Farah

(c) identifies a suitable use case, e.g. finding the shortest route (fewest roads) between two hubs in this unweighted graph (oe) **B1**

(c) justifies this: BFS explores all hubs at the current distance from the start before moving further away, so the first time the target hub is reached is guaranteed to be via a shortest path, whereas DFS explores one branch fully before backtracking and does not guarantee this (oe) **B1**

(c) Answer: BFS would be preferred for finding the shortest route (fewest roads) between two hubs, because it visits all hubs at each distance from the start before moving further out, so the first time it reaches the target hub is guaranteed to be via a shortest path, which DFS does not guarantee.

© Revision Library | revisionlibrary.co.uk

Question 3

(a) compares 35 with the root, 50: since 35 < 50, moves left (to the subtree rooted at 30) **M1**

(a) compares 35 with 30: since 35 ≥ 30, moves right (to the subtree rooted at 40) **M1**

(a) compares 35 with 40: since 35 < 40, and 40 has no left child, 35 becomes the left child of 40 cao **A1**

(a) Answer: 35 becomes the left child of 40.

(b) 50 **B1**

(b) Answer: 50

(c) correctly traverses the left subtree of 50 (rooted at 30) in pre-order: 30, 20, 40, 35 **M1**

(c) correctly traverses the right subtree of 50 (rooted at 70) in pre-order: 70, 60, 80 **M1**

(c) 50, 30, 20, 40, 35, 70, 60, 80 cao **A1**

(c) Answer: 50, 30, 20, 40, 35, 70, 60, 80

(d) correctly traverses the left subtree of 50 in-order: 20, 30, 35, 40 **M1**

(d) correctly traverses the right subtree of 50 in-order: 60, 70, 80, giving the full sequence 20, 30, 35, 40, 50, 60, 70, 80 cao **M1**

(d) states that an in-order traversal of a BST always produces the stored values in ascending (sorted) order (oe) **B1**

(d) Answer: 20, 30, 35, 40, 50, 60, 70, 80. This demonstrates that an in-order traversal of a binary search tree always visits the values in ascending (sorted) order.

© Revision Library | revisionlibrary.co.uk