2.5 Data Structures and Their Operations: 1-hour Pack

OCR H446 · Calculators not allowed · about 60 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A customer-service call centre in Cardiff models its call-holding system as a circular queue, implemented using an array of size 5 (indices 0 to 4), pointers front and rear, and a count of the number of calls currently held. Initially front = 0, rear = 0, count = 0. To enqueue: if count = 5 the queue is full; otherwise array[rear] <- item, rear <- (rear + 1) MOD 5, count <- count + 1. To dequeue: if count = 0 the queue is empty; otherwise item <- array[front], front <- (front + 1) MOD 5, count <- count - 1.

The following calls are held, in order: enqueue(P), enqueue(Q), enqueue(R), dequeue(), dequeue(), enqueue(S), enqueue(T), enqueue(U)

(a) Complete a trace table recording front, rear and count after each of the eight operations. **(8)**

(b) State the front-to-rear order of the calls remaining in the queue after all eight operations. **(1)**

(c) Explain why the array slot at index 0 can safely be reused for enqueue(U), even though it previously held the value P. **(2)**

(Total for Question 1 is 11 marks)

© Revision Library | revisionlibrary.co.uk

2 A file-directory viewer developed by a software company in Oxford represents a small folder structure as a binary tree, where each node stores a single-letter identifier. The root node is M. M's left child is H and right child is T. H's left child is D and right child is L. T's left child is S and right child is W. Nodes D, L, S and W each have no children.

(a) Write out the result of a pre-order traversal of the tree. **(2)**

(b) Write out the result of an in-order traversal of the tree. **(2)**

(c) Write out the result of a post-order traversal of the tree. **(2)**

(d) Write out the result of a breadth-first (level-order) traversal of the tree, and state the abstract data type that is used to implement this traversal, giving a reason for its use. **(3)**

(Total for Question 2 is 9 marks)

© Revision Library | revisionlibrary.co.uk

3 A social-network analysis tool built by a data-science team in Sheffield models the friendship connections between five users, Aisha, Ben, Chidi, Priya and Tom, as an undirected graph, where an edge represents a mutual friendship. The friendships are: Aisha-Ben, Aisha-Chidi, Ben-Chidi, Ben-Priya, Chidi-Priya, Priya-Tom.

(a) Complete the adjacency matrix for this graph, using 1 to indicate a friendship and 0 to indicate no friendship, with rows and columns ordered alphabetically: Aisha, Ben, Chidi, Priya, Tom. **(4)**

(b) Write the adjacency list representation of the same graph. **(3)**

(c) State one advantage of using an adjacency list, rather than an adjacency matrix, to represent a sparse graph such as this one (a graph with relatively few edges compared with the number of possible edges). **(2)**

(Total for Question 3 is 9 marks)

---

4 A software engineer at a customer-records company based in Glasgow is designing a system that must store a large, frequently changing collection of customer records and needs to support fast insertion, deletion and look-up of a record by its unique customer ID.

(a) Discuss the suitability of a hash table, a binary search tree and a linked list for this purpose, comparing their typical time complexities for insertion, deletion and look-up, and recommend which structure the engineer should choose, justifying your recommendation. **(9)**

(Total for Question 4 is 9 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.5 Data Structures and Their Operations: 1-hour Pack

Question 1

(a) enqueue(P): array[0] = P, rear = 1, count = 1 **B1**

(a) enqueue(Q): array[1] = Q, rear = 2, count = 2 **B1**

(a) enqueue(R): array[2] = R, rear = 3, count = 3 **B1**

(a) dequeue(): removes P, front = 1, count = 2 **B1**

(a) dequeue(): removes Q, front = 2, count = 1 **B1**

(a) enqueue(S): array[3] = S, rear = 4, count = 2 **B1**

(a) enqueue(T): array[4] = T, rear = (4+1) MOD 5 = 0, count = 3 **B1**

(a) enqueue(U): array[0] = U (overwriting the freed slot that held P), rear = 1, count = 4 **B1**

(a) Answer: After all eight operations: front = 2, rear = 1, count = 4, with the queue (front to rear) holding R, S, T, U.

(b) R, S, T, U ft from (a) **B1**

(b) Answer: R, S, T, U

(c) P was removed by an earlier dequeue operation, so it is no longer part of the logical queue and is no longer referenced by front or count (oe) **B1**

(c) a circular queue reuses slots via modulo arithmetic on the rear pointer once they fall outside the current front-to-rear range represented by count, avoiding the wasted-space problem of a linear queue (oe) **B1**

(c) Answer: P was already removed by the earlier dequeue() call, so index 0 is no longer part of the logical queue; because the queue is circular, rear can wrap back to index 0 using modulo arithmetic and safely overwrite it with U.

Question 2

(a) correct method applied: node, then left subtree, then right subtree, at every level **M1**

(a) M, H, D, L, T, S, W cao **A1**

(a) Answer: M, H, D, L, T, S, W

(b) correct method applied: left subtree, then node, then right subtree, at every level **M1**

(b) D, H, L, M, S, T, W cao **A1**

(b) Answer: D, H, L, M, S, T, W

(c) correct method applied: left subtree, then right subtree, then node, at every level **M1**

(c) D, L, H, S, W, T, M cao **A1**

(c) Answer: D, L, H, S, W, T, M

(d) M, H, T, D, L, S, W cao **B1**

(d) a queue **B1**

(d) nodes are added (enqueued) to the queue in the order they are discovered, level by level, and removed (dequeued) in the same order, so the FIFO behaviour of a queue ensures each level is fully visited before moving to the next (oe) **B1**

(d) Answer: M, H, T, D, L, S, W. A queue is used, because its FIFO behaviour ensures nodes are visited level by level, in the same order they were discovered.

Question 3

(a) Aisha's row/column correct: connected to Ben and Chidi only **B1**

(a) Ben's row/column correct: connected to Aisha, Chidi and Priya **B1**

(a) Chidi's and Priya's rows/columns correct: Chidi connected to Aisha, Ben, Priya; Priya connected to Ben, Chidi, Tom **B1**

(a) Tom's row/column correct (connected to Priya only), matrix symmetric about the diagonal, and all diagonal entries 0 (no self-friendships) **B1**

(a) Answer: Aisha Ben Chidi Priya Tom
Aisha 0 1 1 0 0
Ben 1 0 1 1 0
Chidi 1 1 0 1 0
Priya 0 1 1 0 1
Tom 0 0 0 1 0

(b) Aisha: [Ben, Chidi] **B1**

(b) Ben: [Aisha, Chidi, Priya]; Chidi: [Aisha, Ben, Priya] **B1**

(b) Priya: [Ben, Chidi, Tom]; Tom: [Priya] **B1**

(b) Answer: Aisha: [Ben, Chidi]; Ben: [Aisha, Chidi, Priya]; Chidi: [Aisha, Ben, Priya]; Priya: [Ben, Chidi, Tom]; Tom: [Priya]

(c) an adjacency list uses less memory for a sparse graph, since it only stores the edges that actually exist (oe) **B1**

(c) an adjacency matrix always requires space proportional to the square of the number of vertices (n^2), regardless of how few edges are actually present, which wastes space for a sparse graph (oe) **B1**

(c) Answer: An adjacency list uses less memory, because it only stores the edges that actually exist, whereas an adjacency matrix always needs n^2 storage regardless of how sparse the graph is.

© Revision Library | revisionlibrary.co.uk

Question 4

(a) **L1** Limited or generic statements are made about one or more of the structures, with little or no reference to time complexity, or an unjustified/unsupported recommendation is given. **1-3**

(a) **L2** A generally sound discussion compares at least two of the three structures, with mostly correct time complexities and some relevant trade-offs, and gives a recommendation with partial or limited justification. **4-6**

(a) **L3** A thorough, well-structured discussion accurately compares all three structures, using correct Big O notation for insertion, deletion and look-up, evaluates relevant trade-offs (e.g. hash-table collisions/worst case, BST balance, linked-list linear search), and gives a clearly justified recommendation tied to the stated requirements. **7-9**

(a) Indicative content:

- Hash table: average-case O(1) for insertion, deletion and look-up, given a good hash function and a reasonably low load factor; worst-case performance can degrade towards O(n) if there are many collisions (e.g. a poor hash function, or a high load factor causing clustering); a hash table does not keep records in any particular order.
- Binary search tree: average-case O(log n) for insertion, deletion and look-up if the tree remains reasonably balanced; worst case degrades to O(n) if the tree becomes unbalanced (for example if customer IDs are inserted in already-sorted order), unless a self-balancing variant is used; a BST does maintain records in sorted order, which is useful if ordered/range-based access is also needed.
- Linked list: O(1) insertion at a known position such as the head, but O(n) look-up and O(n) deletion, since the list must first be traversed from the head to locate the record, as there is no direct/random access to a given position.
- For this scenario, a hash table is generally the most suitable choice, because fast insertion, deletion and look-up by a unique ID are all explicitly required, the customer ID is well suited to being used as (or converted into) a hash key, and there is no stated requirement to keep records in a particular order.
- A valid alternative recommendation, such as a self-balancing binary search tree, can be credited if it is justified with accurate reasoning, for example if ordered traversal of customers by ID were also a requirement of the system.

(a) Answer: A hash table is the most suitable choice, since it offers average-case O(1) insertion, deletion and look-up by unique ID (versus O(log n) for a balanced BST and O(n) for look-up/deletion in a linked list), and no ordering requirement is stated that would favour a BST.

© Revision Library | revisionlibrary.co.uk