2.5 Data Structures and Their Operations: Test Pack

OCR H446 · Calculators not allowed · about 25 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A ticket-checking app used at a small theatre in Leeds implements a stack, a data structure that operates on a Last In First Out (LIFO) basis, to keep track of tickets as they are scanned.

(a) State what is meant by the term 'Last In First Out (LIFO)'. **(1)**

(b) Give one real-world example of the use of a stack, other than the scenario described above. **(1)**

(c) An empty stack is implemented using an array of maximum size 3 (indices 0 to 2), with a pointer called top that stores the index of the top item. When the stack is empty, top = -1. The following operations are performed, in order:
push(5), push(9), push(2), pop(), push(7)

Complete a trace table showing the contents of the array and the value of top after each of the five operations. **(4)**

(d) Explain what would happen if a further operation, push(3), were attempted immediately after the operations in part (c). **(2)**

(Total for Question 1 is 8 marks)

© Revision Library | revisionlibrary.co.uk

2 An online ticket-booking system used by a leisure centre in Swansea stores booking reference numbers in a hash table with 7 slots (indices 0 to 6), using the hash function h(k) = k MOD 7. Collisions are resolved using linear probing: if the calculated slot is already occupied, the next slot is tried (wrapping from slot 6 back to slot 0), and so on, until an empty slot is found.

The following keys are inserted, in this order, into an initially empty hash table: 23, 12, 15, 8, 19, 30

(a) Calculate h(k) for each of the six keys. **(3)**

(b) Complete a trace table showing the slot at which each key is finally stored, including any probing required due to collisions. **(6)**

(c) State the final contents of the hash table (slot index: key) for all 7 slots. **(1)**

(d) Calculate the load factor of the hash table after all six keys have been inserted, giving your answer as a fraction in its simplest form. **(2)**

(e) Explain one disadvantage of linear probing as a collision-resolution technique, compared with separate chaining. **(2)**

(Total for Question 2 is 14 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.5 Data Structures and Their Operations: Test Pack

Question 1

(a) the most recently added item is the first one to be removed (oe) **B1**

(a) Answer: The item most recently added to the structure is the first one to be removed.

(b) any valid example, e.g. the undo function in a word processor, the back button in a web browser, or storing return addresses/local variables for function calls (call stack) (oe) **B1**

(b) Answer: The undo feature in a word processor, which reverses the most recent action first.

(c) push(5): array = [5, _, _], top = 0 **B1**

(c) push(9): array = [5, 9, _], top = 1 **B1**

(c) push(2): array = [5, 9, 2], top = 2 **B1**

(c) pop(): array = [5, 9, _], top = 1; then push(7): array = [5, 9, 7], top = 2 **B1**

(c) Answer: push(5): [5,_,_] top=0. push(9): [5,9,_] top=1. push(2): [5,9,2] top=2. pop(): [5,9,_] top=1 (2 removed). push(7): [5,9,7] top=2.

(d) a stack overflow error would occur ft from (c) **B1**

(d) because top (2) already equals the maximum valid index (size - 1 = 2), so the array is completely full and there is no free slot to store 3 (oe) **B1**

(d) Answer: A stack overflow occurs, because top is already at index 2, the maximum index of the array (size 3, indices 0 to 2), so there is no room to push another item.

Question 2

(a) h(23) = 2 and h(12) = 5 **B1**

(a) h(15) = 1 and h(8) = 1 **B1**

(a) h(19) = 5 and h(30) = 2 **B1**

(a) Answer: h(23)=2, h(12)=5, h(15)=1, h(8)=1, h(19)=5, h(30)=2

(b) 23: h=2, slot 2 empty, stored at slot 2 **B1**

(b) 12: h=5, slot 5 empty, stored at slot 5 **B1**

(b) 15: h=1, slot 1 empty, stored at slot 1 **B1**

(b) 8: h=1, slot 1 occupied (15); probes slot 2, occupied (23); probes slot 3, empty; stored at slot 3 **B1**

(b) 19: h=5, slot 5 occupied (12); probes slot 6, empty; stored at slot 6 **B1**

(b) 30: h=2, slot 2 occupied (23); probes slot 3, occupied (8); probes slot 4, empty; stored at slot 4 **B1**

(b) Answer: 23->slot2, 12->slot5, 15->slot1, 8->slot3 (after probing 1,2), 19->slot6 (after probing 5), 30->slot4 (after probing 2,3)

(c) 0: empty, 1: 15, 2: 23, 3: 8, 4: 30, 5: 12, 6: 19 ft from (b) **B1**

(c) Answer: 0: empty, 1: 15, 2: 23, 3: 8, 4: 30, 5: 12, 6: 19

(d) load factor = number of occupied slots / total number of slots = 6 / 7 **M1**

(d) 6/7 cao (accept awrt 0.86) **A1**

(d) Answer: 6/7

(e) linear probing can lead to clustering, where consecutive occupied slots build up, making future collisions (and therefore searches and insertions) increasingly likely and increasingly slow (oe) **B1**

(e) separate chaining instead allows each slot to hold multiple items (e.g. in a linked list), so performance degrades less severely, and a slot never has to overflow into a neighbouring one (oe) **B1**

(e) Answer: Linear probing can cause clustering, where a build-up of consecutive occupied slots makes further collisions, and therefore searches, progressively slower, as seen with the probing needed for 8 and 30 above; separate chaining avoids this, since each slot can hold multiple items in its own chain rather than spilling into neighbouring slots.

© Revision Library | revisionlibrary.co.uk