University of Zurich

Department of Informatics
Database Technology Group
Prof. Dr. Michael Böhlen

Informatics II
Exercise 7

Spring Semester 2022
Week 8
Stacks, Queues

Task 1. Short Questions

(a) Consider the operations $push(S, x)$ which inserts item x into the stack S and the operation $pop(S)$ which removes the item at the top of the stack S and returns it. Further consider the operations $enqueue(Q, x)$ which inserts item x at the tail of queue Q and the operation $dequeue(Q)$ which removes the item from the head of Q and returns it.
The following sequence of operations are performed on two initially empty stacks S_1 and S_2 and two initially empty queues Q_1 and Q_2:

(1) $push(S_1, 5)$
(2) $push(S_2, 7)$
(3) $push(S_1, 3)$
(4) $enqueue(Q_1, 2)$
(5) $enqueue(Q_1, 4)$
(6) $enqueue(Q_2, 1)$
(7) $push(S_1, dequeue(Q_1))$
(8) $enqueue(Q_2, pop(S_2))$
(9) $push(S_2, 6)$
(10) $push(S_1, pop(S_1) + pop(S_2))$
(11) $enqueue(Q_2, pop(S_1) + dequeue(Q_2))$
(12) $push(S_2, dequeue(Q_2))$
(13) $dequeue(Q_2)$

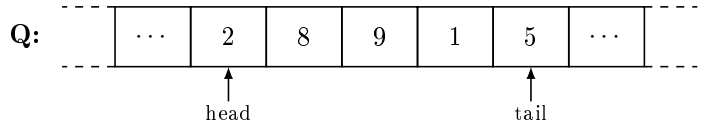
What will be the item returned by the last *dequeue* operation?

Answer:

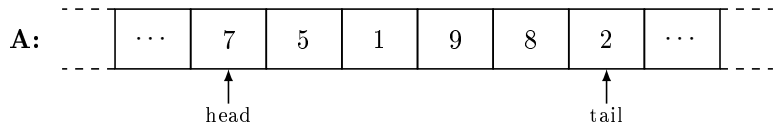
What items do the stacks S_1, S_2 and the queues Q_1 and Q_2 contain after the last *dequeue* operation?

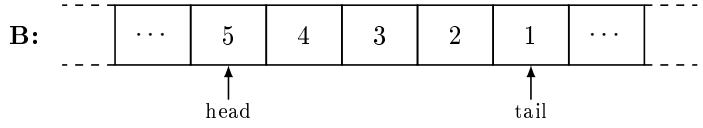
Answer:

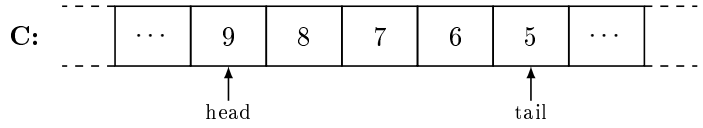
(b) Consider the following queue Q:

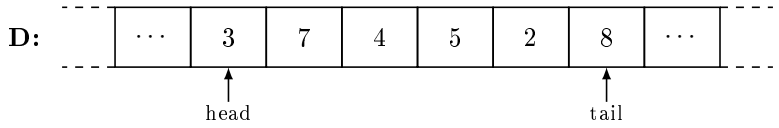


Which of the following queues could be transformed into queue Q *after less than ten* enqueue and / or dequeue operations?









Answer:

(c) Consider the following state of a stack S at some point in time (where the leftmost item is the top of the stack): $S = [7, 3, 5, 1]$
Which of the following are possible sequences of operations which were performed *immediately before* the situation shown above has arisen?

A: $pop(), pop(), push(2), pop(), push(7)$

B: $pop(), pop(), push(4), push(1), pop(), push(3), push(7)$

C: $push(7), pop(), push(5), pop(), push(3), pop()$

D: $push(5), push(7), push(3), pop(), pop()$

E: $push(pop()), push(pop()), push(pop()), push(pop())$

Answer:

(d) Consider a queue Q which currently contains n distinct integers as items. Assume you want to remove the all items from Q which are divisible by 3. The other elements in the queue shall be in the exact same order as before after the removal of these elements and you may not use another storage for the items other than Q and a single helper variable.
How many enqueue and dequeue operations are minimally required in the best case and worst case to achieve this task?

Answer:

(e) Consider a sequence of $m > 0$ stacks $(S_1, \ldots, S_m)$ and a sequence of $n > 0$ queues $(Q_1, \ldots, Q_n)$. All of the stacks and all of the queues have an identical size (capacity) of c. An input sequence of k mutually distinct integers is processed as items by these sets of m stacks and n queues in the following way:

(i) At first, any item from the input sequence is pushed to stack S_1.

(ii) Whenever a stack S_i is full, all its items are popped and then immediately pushed to the next stack S_{i+1} until the stack S_i is empty.

(iii) When the last stack in the sequence S_m is full, all its items are popped and then immediately enqueued to the first queue in the sequence Q_1 until the stack S_m is empty.

(iv) Whenever a queue Q_i is full, all its items are dequeued and then immediately enqueued to the next queue Q_{i+1} until the queue Q_i is empty.

(v) When the last queue in the sequence Q_n is full, all its items are dequeued and constitute, in the order that they were dequeued, the output sequence.

For which values of m, n, k and c will the output sequence be identical to the input sequence?

Answer:

Task 2. Array-based Implementation of a Stack
Consider the given code skeleton `task2_skeloton.c` which contains a starting point for the implementation of a stack in C. In particular, the skeleton contains a `struct` which shall be used for the implementation and is also shown below.

```
typedef struct Stack {
    unsigned int capacity;
    int* items;
    int top;
} Stack;
```

Expand the given skeleton and implement in C the following functions. Test your implementation with appropriate calls in the `main` function.

(a) `create` which allocates memory for a new stack

(b) `delete` which frees the memory allocated to the stack

(c) `is_empty` which returns 1 if the stack is empty and 0 otherwise

(d) `is_full` which returns 1 if the stack is full and 0 otherwise

(e) `get_capacity` which returns the capacity of the stack

(f) `num_items` which returns the number of items currently stored in the stack

(g) `push` which pushes a new value to the stack

(h) `pop` which pops the top item from the stack

(i) `peek` which returns the value currently at the top of the stack without removing it

(j) `print` which prints all items currently in the stack

(k) `is_equal` which checks whether two given stacks have identical contents

(l) `reverse` which reverses the order of the items in the stack

Task 3. Computing Spans
Consider an array $A[0..n-1]$ of n integers. The span $s(A,i)$ of array element $A[i]$ is the maximum number of consecutive elements $A[j]$ immediately preceding $A[i]$ and such that $A[j] \leq A[i]$. For example, array $A = [6,3,4,5,2]$ has the spans $s(A,0) = 1$, $s(A,1) = 1$, $s(A,2) = 2$, $s(A,3) = 3$ and $s(A,4) = 1$ which can be written as an array as follows: $s(A) = [1,1,2,3,1]$.

(a) Write a function `get_span(int data[], int num_data, int* spans)` in C which calculates the spans at each position of an integer array and fills the calculated value to the respective position of array `spans`.

(b) Determine the asymptotic time complexity of your algorithm. Is it possible to solve this problem in linear time?

Task 4. Are These Push and Pop Operations Possible?
Consider an initially empty stack S for which a number of n *push* operations and n *pop* operations is performed. Every time when an item is pushed to S this item is stored at the same time at the end of array In and every time when an item is popped from S this item is stored at the same time at the end of array Out.
The goal of this task is to devise and analyse an algorithm which takes two arrays $In[0..n-1]$ and $Out[0..n-1]$ as paramters and decides whether they could potentially be the result of such *push* and *pop* operations. For example, consider $In = [1, 2, 3, 4, 5]$ and $Out = [4, 5, 3, 2, 1]$. These two arrays can be the result of *push* and *pop* operations as follows:

(1) $push(1)$
(2) $push(2)$
(3) $push(3)$
(4) $push(4)$
(5) $pop() \rightarrow 4$
(6) $push(5)$
(7) $pop() \rightarrow 5$
(8) $pop() \rightarrow 3$
(9) $pop() \rightarrow 2$
(10) $pop() \rightarrow 1$

(a) Is $In = [1, 2, 3, 4, 5]$ and $Out = [4, 3, 1, 5, 2]$ a pair of arrays that could arise from *push* and *pop* operations as described? If so, show the respective sequence of operations to achieve these arrays. If not, explain why these arrays can not possibly result.

(b) Implement an algorithm in C that takes two arrays as parameters an checks whether they can be achieved by performing the *push* and *pop* operations as explained above. You may assume that

(i) both arrays contain the same number of elements,
(ii) the elements of In are a permutation of the elements in Out,
(iii) neither of the two arrays is empty and
(iv) neither of the two arrays contains duplicates.

(c) What is the average time complexity of the algorithm you devised in subtask (b) with respect to n where n is the number of elements in the arrays In and Out.