NIYOTWIZERA VALENS

Regnumber:224014946

BIT

Part I:

A stack is a non-primitive linear data structure. It is an ordered list in which addition of new data item and deletion of already existing data item is done from only one end, known as Top of Stack (TOS). As all the deletion and insertion in a stack is done from top of the stack, the last added element will be the first to be removed from the stack. Due to this reason, the stack is also called Last-In-First-Out (LIFO) type of list.

A. Basics

1. Operation: Push/Pop (LIFO)

The MTN Momo app example shows the LIFO (Last-In-First-Out) nature of stacks because when you press back, the last step you added (pushed) is removed (popped) first. This demonstrates that the most recent item added to the stack is the first one to be removed.

The LIFO nature is demonstrated by the fact that pressing back removes the last step, which is the most recent item added to the stack.

2. Operation: Pop (Undo)

The UR Canvas example is like popping from a stack because pressing back undoes the last step, which is equivalent to removing the top item from the stack.

This action is like popping from a stack because it removes the most recent item (last step) from the stack, allowing you to revert to the previous state.

B. Application

3. Operation: Push (Add to stack)

A stack can enable the undo function in BK Mobile Banking's transaction history by storing each transaction as a separate item in the stack. When a user wants to undo a transaction, the top item (most recent transaction) can be popped from the stack, effectively reversing the transaction.

A stack can enable the undo function by storing each transaction as a separate item and allowing the user to pop the top item to reverse the most recent transaction.

4. Operation: Balanced Parentheses Check

Stacks can ensure forms are correctly balanced in Irembo registration forms by pushing opening brackets and popping when matching closing brackets are found. If the stack is empty at the end, the brackets are balanced. If there are unmatched opening or closing brackets, the stack will not be empty, indicating an error.

Stacks can ensure forms are correctly balanced by using a push-pop mechanism to match opening and closing brackets.

C. Logical

Q5. Operation: Push and Pop sequence
Let's analyze the sequence:
. Push ("CBE notes")
. Push ("Math revision")
. Push("Debate")
. Pop () -> removes "Debate"
. Push ("Group assignment")
The current state of the stack is:
["CBE notes", "Math revision", "Group assignment"]
The next task (top of stack) is "Group assignment".

Q6. Operation: Undo with multiple Pops
Let's assume the stack initially contains:
["Answer 1", "Answer 2", "Answer 3", "Answer 4"]
After undoing 3 recent actions (popping 3 times), the stack will contain:
["Answer 1"]
The answer that remains in the stack after undoing is "Answer 1".

Advanced Thinking

Q7. Let’s explore how each queue type maps to real Rwandan life:

- Linear queue: A linear queue is like people waiting in line at a wedding buffet in Rwanda. People arrive, join the end of the line, and are served in the order they arrive. Once served, they leave the line.

- Circular queue: A circular queue is like buses looping at HUYE bus terminal in Rwanda. Buses arrive, drop off passengers, and then continue their route, making room for new passengers. The bus terminal operates in a cyclical manner, with buses constantly arriving and departing.

- Deque: A deque is like boarding a bus from either the front or rear door in Rwanda. Passengers can enter or exit the bus from either door, allowing for more flexibility in the boarding process.

Each queue type maps to real Rwandan life by modeling different scenarios where people or objects need to be processed in a particular order.

Q8. Step 1: Push words onto the stack
A stack is a *Last-In, First-Out (LIFO)* structure.
We push each word one by one:
Push "Umwana" → Stack: [Umwana]
Push "ni" → Stack: [Umwana, ni]
Push "umutware" → Stack: [Umwana, ni, umutware]
Pop words from the stack
Now we pop (remove from top):
Pop → "umutware" → Output: umutware
Stack: [Umwana, ni]
Pop → "ni" → Output: umutware ni
Stack: [Umwana]
Pop → "Umwana" → Output: umutware ni Umwana
Stack: []
After popping up all words, the reversed proverb is:
umutware ni Umwana

Q9. A stack suits Depth-First Search (DFS) better than a queue because DFS requires backtracking to explore previously visited branches when a path is exhausted, a behavior that the stack's last in and first out (LIFO) principle naturally supports. In the library search analogy, a stack allows the student to go as deep as possible down one shelf (branch), and when they hit a dead end or a completed shelf, they can use the stack to return to the last decision point and try a different approach.

Q10. A suggested feature using stacks for transaction navigation is a "milt -level transaction details" where users can tap on a transaction to see its details, and then tap on an associated transfer or payment to view those details, with each view being pushed onto the stack. The user can then use the back button (pop) to return through the layers of information, such as from transfer details back to transaction details, and then back to the main transaction list.

How it works:

Initial View (Transaction History List): The user starts with a list of transactions.

Push to Transaction Details: When the user taps a transaction, its detail screen is pushed onto the stack, becoming the visible screen.

Push to Related Detail: If a transaction involves a transfer or payment that also has its own details, the user can tap that to see its details.

PART II

Queue is a non-primitive linear data structure that permits insertion of an element at one end and deletion of an element at the other end. The end at which the deletion of an element takes place is called front, and the end at which insertion of a new element can take place is called rear.

C. Logical

5. Operation: Sequence of Enqueue/Dequeue
The sequence is:
6. Enqueue("Alice")
7. Enqueue("Eric")
8. Enqueue("Chantal")
9. Dequeue() -> removes "Alice"
10. Enqueue("Jean")

The current state of the queue is:
["Eric", "Chantal", "Jean"]

The person at the front of the queue is "Eric".

6. Operation: FIFO message handling

A queue ensures fairness in handling RSSB pension applications by processing them in the order they arrive (First-In-First-Out). This means that applications are handled in a fair and orderly manner, without any bias or prioritization.

D. Advanced Thinking

7. Operation: Different queue types

Let's explore how each queue type maps to real Rwandan life:

- Linear queue: A linear queue is like people waiting in line at a wedding buffet in Rwanda. People arrive, join the end of the line, and are served in the order they arrived.
- Circular queue: A circular queue is like buses looping at Nyabugogo bus terminal in Rwanda. Buses arrive, drop off passengers, and then continue their route, making room for new passengers.
- Deque: A deque is like boarding a bus from either the front or rear door in Rwanda. Passengers can enter or exit the bus from either door, allowing for more flexibility in the boarding process.

8. Operation: Enqueue orders, Dequeue when ready

Queues can model the process of customers ordering food at a Kigali restaurant by:

9. Enqueueing customer orders as they are received.

10. Dequeuing orders when the food is ready, allowing the customer to be notified and collect their order.

11. Operation: Priority queue

This is a priority queue because emergencies are given priority over non-emergency cases. In a normal queue, patients would be treated in the order they arrived. However, in a priority queue, emergencies are treated first, regardless of their arrival time.