EX Sheet: 09
Data Structure Design for Managing Two Checkout Counters

Prepared by: Ashik Emon

Problem Statement

Design a data structure that manages customers waiting for service at one of two checkout counters. Each customer is represented by an object reference. Let n be the total number of customers currently waiting.

Data Structure

The following components are maintained:

- Two doubly linked lists L_1 and L_2 representing queues at counters 1 and 2.
- A hash table H mapping each customer reference x to:
  - the list (L_1 or L_2) containing x,
  - a pointer to the node in that list.
- Two Boolean variables `open`[1] and `open`[2] indicating whether counters are open.

Doubly linked lists allow constant-time insertions and deletions when a node reference is known, and the hash table provides direct access to any customer.

Method Descriptions

1. `open(c)`

Description: If checkout counter $c \in \{1, 2\}$ is closed, mark it as open. If customers were previously waiting in its line, they return to the same line in original order.

Implementation:

- Set `open`$[c]$ = `true`.
- Restore customers to their original line if necessary.

Running Time: Worst case $O(n)$, as restoring customers may require iterating over all customers previously assigned to this counter.

2. `add(x)`

Description: If at least one checkout counter is open, add customer x to the shorter of the currently open queues.

Implementation:

- Compare sizes of L_1 and L_2 (only among open counters).
- Append x to the tail of the shorter list.
- Store a reference to x in hash table H.

Running Time: $O(1)$ for size comparison, tail insertion, and hash table update.

3. `process(c)`

Description: Remove the customer at the front of checkout counter c.

Implementation:

- Remove the head node of list L_c.
- Delete the customer's entry from hash table H.

Running Time: $O(1)$ for head removal and hash deletion.

4. `leave(x)`

Description: Remove customer x from the data structure using an object reference.

Implementation:

- Use hash table H to locate the node and list containing x.
- Remove the node from the doubly linked list.
- Delete x from hash table H.

Running Time: $O(1)$ for lookup and removal.

5. `close_counter(c)`

Description: If both counters are open, close counter c and reorganize all customers into the other queue in arrival order.

Implementation:

- Let the open counter be $c' \neq c$.
- Merge customers from both lists into a single list $L_{c'}$, preserving arrival order.
- Set $\texttt{open}[c] = \texttt{false}$.

Running Time: $O(n)$, as each customer is moved once.

Correctness and Runtime Summary

Operation	Worst-Case Time
`open(c)`	$O(n)$
`add(x)`	$O(1)$
`process(c)`	$O(1)$
`leave(x)`	$O(1)$
`close_counter(c)`	$O(n)$

All required asymptotic worst-case running times are satisfied.

Intuitive Pseudocode Version

Data Structure

Algorithm 1 Data Structure

L_1, L_2 : queues for checkout counters 1 and 2
$open[1], open[2]$: booleans indicating whether counters are open
H : hash table mapping customer $\rightarrow$ (queue, node)

`open(c)`

Algorithm 2 `open(c)`

if $open[c] = true$ **then**
  return
end if
$open[c] \leftarrow true$
Restore customers in their original order

`add(x)`

`process(c)`

`leave(x)`

`close_counter(c)`

Algorithm 3 `add(x)`

```
if open[1] and open[2] then
  if |L_1| ≤ |L_2| then
    append x to L_1
    H[x] ← (L_1, node)
  else
    append x to L_2
    H[x] ← (L_2, node)
  end if
else if open[1] then
  append x to L_1
  H[x] ← (L_1, node)
else if open[2] then
  append x to L_2
  H[x] ← (L_2, node)
end if
```

Algorithm 4 `process(c)`

```
if L_c is empty then
  return
end if
x ← first element of L_c
remove x from L_c
remove x from H
```

Algorithm 5 `leave(x)`

```
(L, node) ← H[x]
remove node from L
remove x from H
```

Algorithm 6 `close_counter(c)`

```
if not (open[1] and open[2]) then
  return
end if
c' ← the other counter
for each customer x in arrival order do
  append x to L_c'
  update H[x]
end for
clear L_c
open[c] ← false
```

Challenge: Can All Operations Be Done in $O(1)$?

Achieving $O(1)$ for all operations is not possible. Operations such as `open(c)` and `close_counter(c)` require reorganizing up to n customers while preserving arrival order. Any data structure must inspect or move each customer at least once, resulting in a lower bound of $\Omega(n)$. Therefore, the given solution is asymptotically optimal.