Queues

Lesson 4

Outline

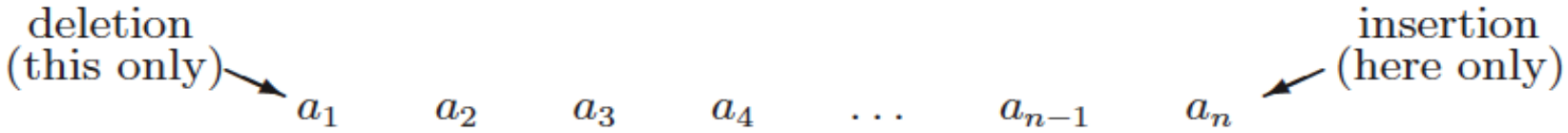
- What is a queue?
  - Straight queue
  - Circular Queue
- Sequential Implementation
- Linked Implementation
- Application: Topological sort
- Deques
- Final Notes

Outline

- **What is a queue?**
  - **Straight queue**
  - **Circular Queue**
- Sequential Implementation
- Linked Implementation
- Application: Topological sort
- Deques
- Final Notes

What is a queue?

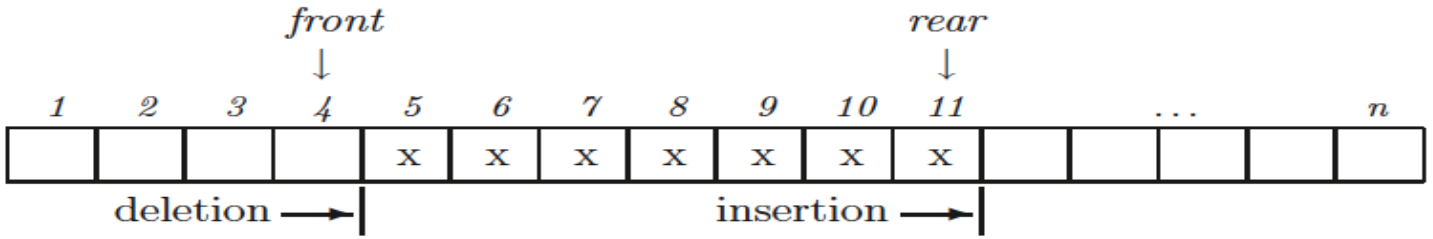
- ***"First in, first out"*** ADT



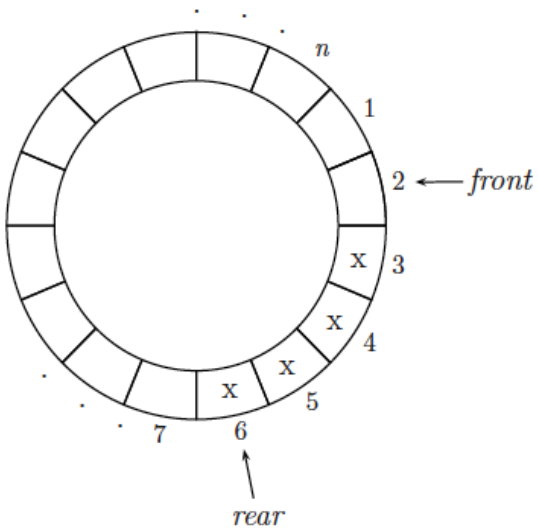
- Operations
  - Auxiliary (same as stacks)
  - Enqueue
  - Dequeue

Straight queue

- Straight queue



- Circular queue



Outline

- What is a queue?
  - Straight queue
  - Circular Queue
- **Sequential Implementation**
- Linked Implementation
- Application: Topological sort
- Deques
- Final Notes

Straight queue

- Enqueue

```
procedure ENQUEUE(Q, x)
if rear = n then call MOVE_QUEUE(Q)
rear ← rear + 1
Q(rear) ← x
end ENQUEUE

procedure MOVE_QUEUE(Q)
if front = 0 then [output 'No more available space'; stop]
             else [for i ← front + 1 to n do
                       Q(i − front) ← Q(i)
                   endfor
                   rear ← rear − front
                   front ← 0 ]
end MOVE_QUEUE
```

Straight queue

- Dequeue

```
procedure DEQUEUE(Q, x)
if front = rear then call QUEUEUNDERFLOW
                else [ front ← front + 1
                       x ← Q(front)
                       if front = rear then front ← rear ← 0 ]
end DEQUEUE
```

Circular queue

- Enqueue

```
procedure ENQUEUE(Q, x)
if front = rear mod n + 1 then [output 'No more available cells'; stop]
                          else [rear ← rear mod n + 1;  Q(rear) ← x]
end ENQUEUE
```

- Dequeue

```
procedure DEQUEUE(Q, x)
if front = rear then call QUEUEUNDERFLOW
                else [front ← front mod n + 1;  x ← Q(front) ]
end DEQUEUE
```

Outline

- What is a queue?
  - Straight queue
  - Circular Queue
- Sequential Implementation
- **Linked Implementation**
- Application: Topological sort
- Deques
- Final Notes

Straight queue

- Enqueue

```
procedure ENQUEUE(Q, x)
call GETNODE(α)
INFO(α) ← x
LINK(α) ← Λ
if front = Λ then front ← rear ← α
             else [LINK(rear) ← α; rear ← α]
end ENQUEUE
```

Straight queue

- Dequeue

```
procedure DEQUEUE(Q, x)
if front = Λ then call QUEUEUNDERFLOW
             else [x ← INFO(front)
                   α ← front
                   front ← LINK(front)
                   call RETNODE(α)]
end DEQUEUE
```

Outline

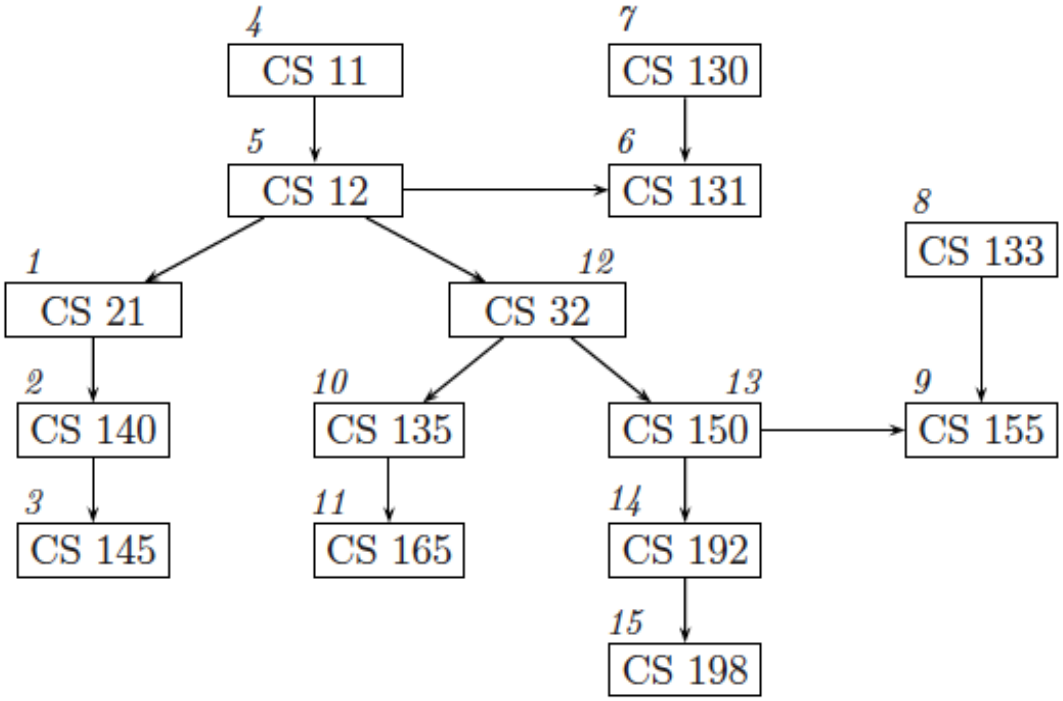
- What is a queue?
  - Straight queue
  - Circular Queue
- Sequential Implementation
- Linked Implementation
- **Application: Topological sort**
- Deques
- Final Notes

Topological sort

- Arrange objects in the sequence such that no objects appear before its (direct) predecessors
  - Notion of (partial) ***ordering of a set***
  - ***e.g. integer numbers arranged according to increasing magnitude***
    - ***1, 2, 6, 7, 5, 4, 3, 2 (not sorted)***
    - ***1, 2, 2, 3, 4, 5, 6, 7 (sorted)***

Topological sort

- Practical application: creating study plan for a program of study



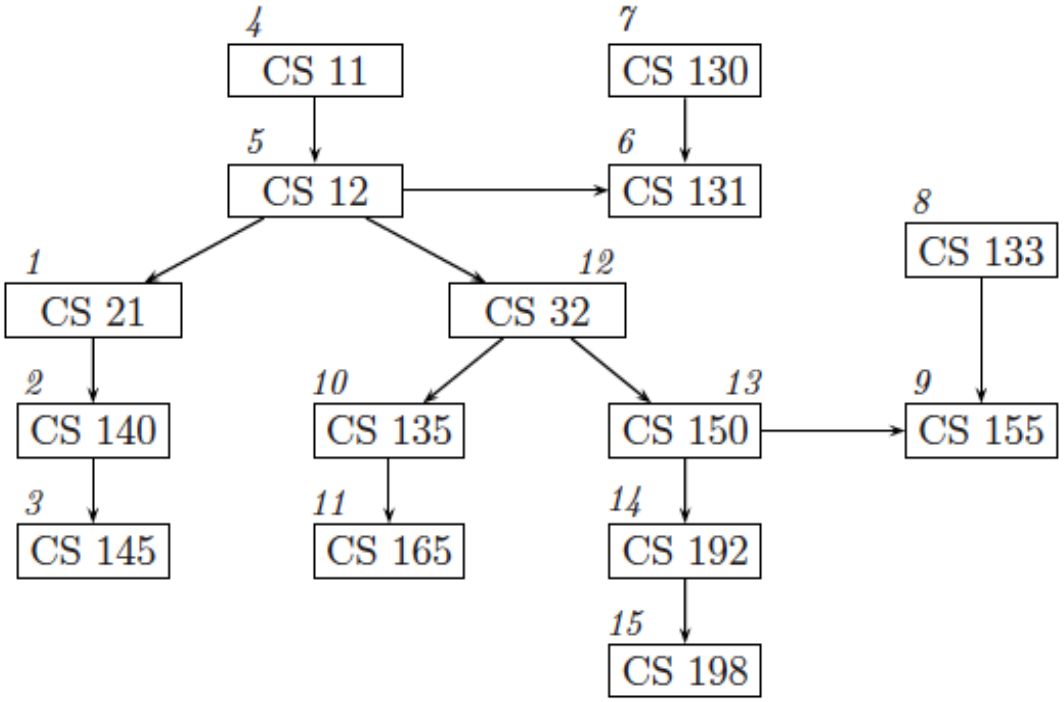
Topological sort

- Really a problem involving ***graphs,*** but nonetheless uses a queue in one of the implementations of topological sort
- Idea for topological sort
  - Determine partial ordering of the elements by creating predecessor-successor pairs
  - Keep count of the number of direct predecessors and "tag" successors for all objects
    - Can use linked lists here
  - Initialize a queue and dequeue it to produce output

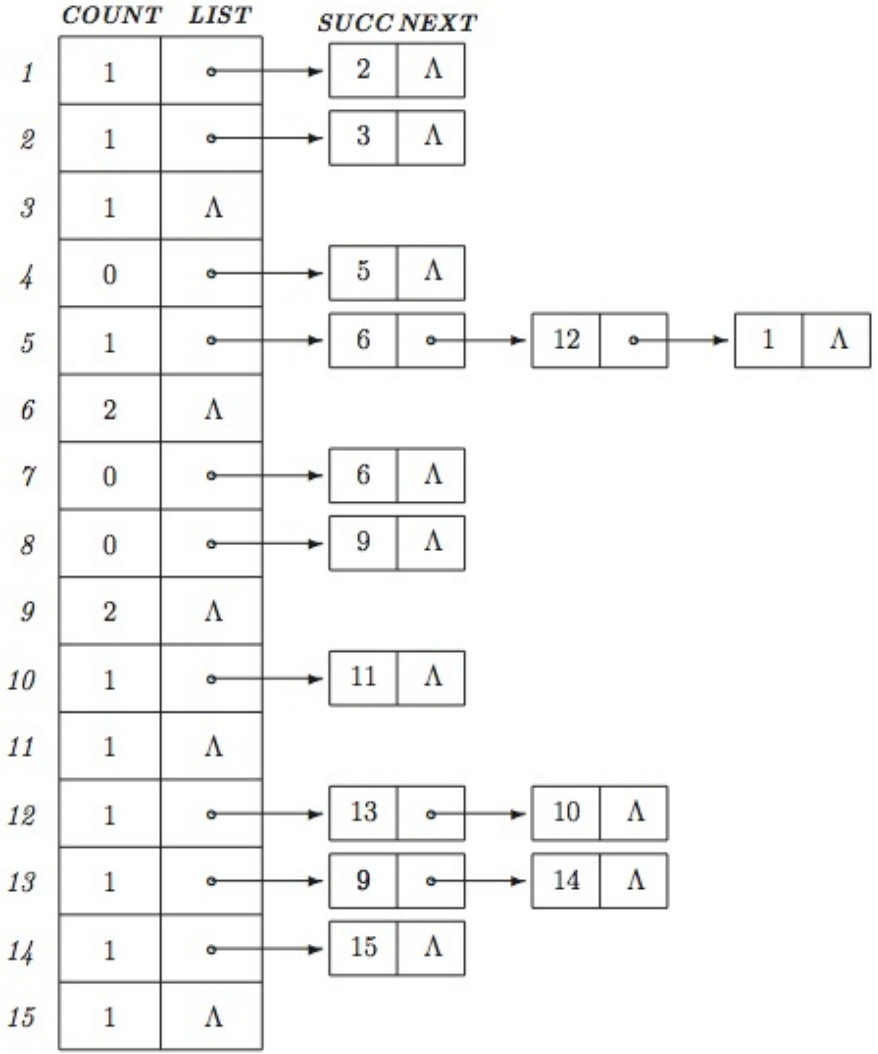
Involving queues in TS

- Makes use of a ***linked queue***
  - Initially, all elements are objects with no direct predecessors
  - A count of all the predecessors of the objects is kept
    - Each time count of direct predecessors drops to zero, object is inserted into queue ready for output
  - A linked list of all the direct successors of an object is kept
    - Optimization: overstore queue in count array by reusing the count field as a link field when the count drops to zero

TS: An Example

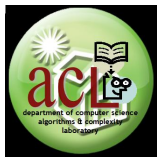


TS: An Example



Involving queues in TS

- Initialize the queue
  - Scan the table for all inputs. If an object is found which has no direct predecessors, then enqueue it
    - Use the count field as a link field (optimization)
    - Initially of course, front points to nothing, and once an initial object is enqueued, that object becomes front and rear altogether



Involving queues in TS

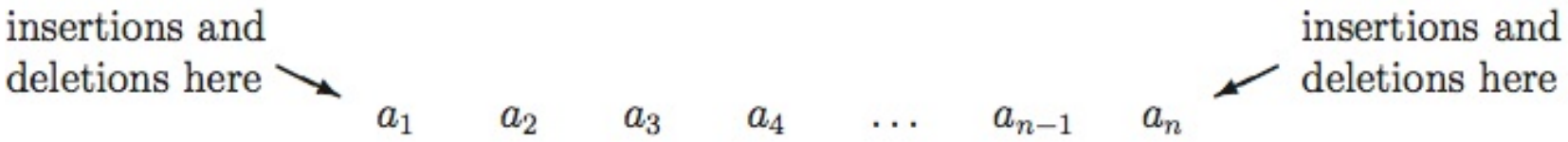
- Dequeueing the output
  - Output front of the queue and all that link to it (i.e. the successors).
    - For each of the successors that is output, decrement the count of its direct successors
      - If the count of a direct successor reaches zero, enqueue it
  - Dequeue the front of the queue and proceed processing of the next item in line
  - Repeat process until queue is empty

Outline

- What is a queue?
  - Straight queue
  - Circular Queue
- Sequential Implementation
- Linked Implementation
- Application: Topological sort
- **Deques**
- Final Notes

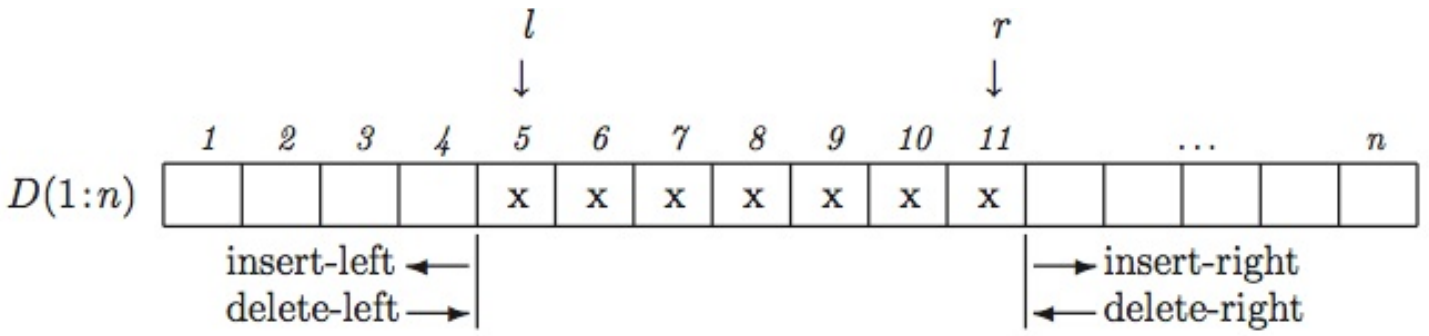
Deques

- **Deque** = **D**ouble **E**nded **Que**ue



  - Apart from the usual input parameters in enqueue and dequeue, one must also **specify where insertion or deletion will be done**.
  - Also, instead of front and rear, **ends of a deque are labeled l(eft) and r(ight)**

Deques – Sequential Implementation



- Initializing a sequential deque

$$r \leftarrow \lfloor n/2 \rfloor$$
$$l \leftarrow r + 1$$

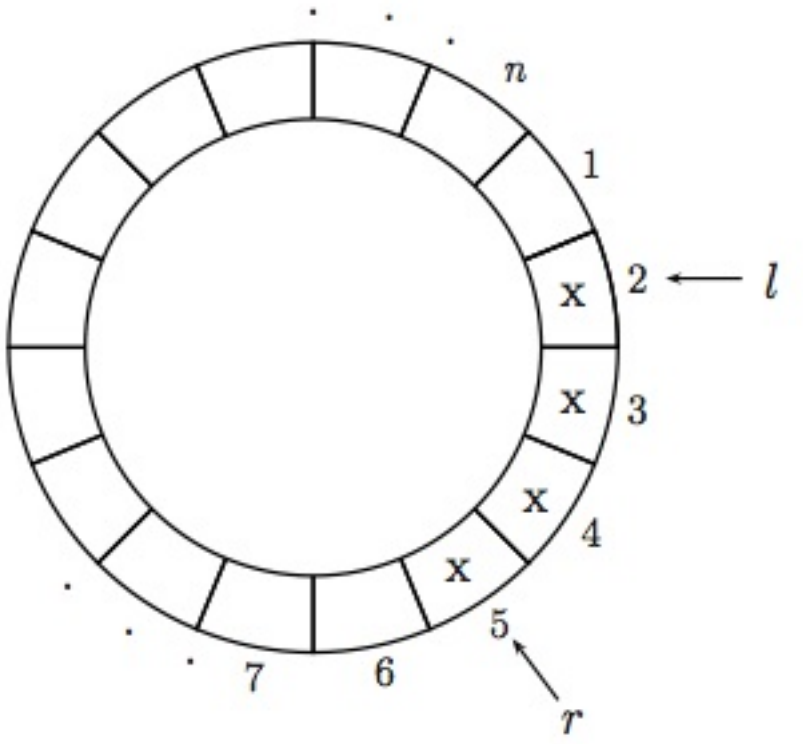
  - **WHY?!**

Deques – Sequential Implementation

```
procedure INSERT_DEQUE(𝔻, left, x)
if (left and l = 1) or (not left and r = n)  then call MOVE_DEQUE(𝔻, left)
if left then [ l ← l − 1; D(l) ← x ]
        else [ r ← r + 1; D(r) ← x ]
end INSERT_DEQUE

procedure MOVE_DEQUE(𝔻, left)
if l = 1 and r = n then [ output 'No more available cells'; stop ]
if left then [ s ← ⌈(n − r)/2⌉
               for i ← r to l by −1 do
                  D(i + s) ← D(i)
               endfor
               r ← r + s; l ← l + s ]
        else [ s ← ⌈(l − 1)/2⌉
               for i ← l to r do
                  D(i − s) ← D(i)
               endfor
               r ← r − s; l ← l − s ]
end MOVE_DEQUE
```

Circular Deques



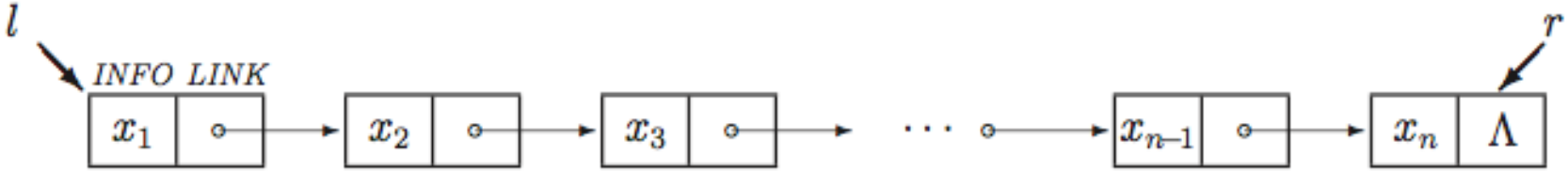
– Initialization (HOW?!!)

Circular Deques

procedure INSERT_DEQUE($\mathbb{D}$, *left*, x)
if $l = (r + 1) \bmod n + 1$ **then** [**output** 'No more available cells'; **stop**]
if *left* **then** [$l \leftarrow n - (1 - l) \bmod n$; $D(l) \leftarrow x$]
else [$r \leftarrow r \bmod n + 1$; $D(r) \leftarrow x$]
end INSERT_DEQUE

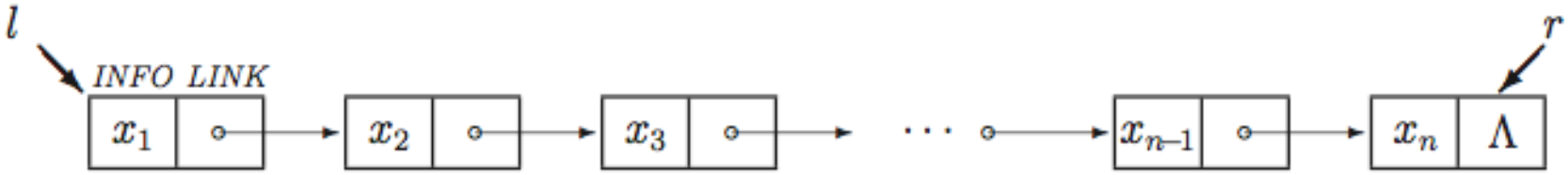
procedure DELETE_DEQUE($\mathbb{D}$, *left*, x)
if $l = r \bmod n + 1$ **then call** DEQUEUNDERFLOW
else [**if** *left* **then** [$x \leftarrow D(l)$; $l \leftarrow l \bmod n + 1$]
else [$x \leftarrow D(r)$; $r \leftarrow n - (1 - r) \bmod n$]
end DELETE_DEQUE

Deques – Linked Implementation



procedure INSERT_DEQUE($\mathbb{D}$, *left*, x)
call GETNODE(α)
$INFO(\alpha) \leftarrow x$
case
: $l = \Lambda$: [$l \leftarrow r \leftarrow \alpha$; $LINK(\alpha) \leftarrow \Lambda$]
: *left* : [$LINK(\alpha) \leftarrow l$; $l \leftarrow \alpha$]
: **else** : [$LINK(r) \leftarrow \alpha$; $r \leftarrow \alpha$; $LINK(r) \leftarrow \Lambda$]
endcase
end INSERT_DEQUE

Deques – Linked Implementation




```
procedure DELETE_DEQUE(𝔻, left, x)
if l = Λ then call DEQUEUNDERLOW
          else [ if left or l = r then [ α ← l; l ← LINK(l) ]
                                  else [ α ← l
                                         while LINK(α) ≠ r do
                                            α ← LINK(α)
                                         endwhile
                                         r ↔ α     ▷ swap pointers
                                         LINK(r) ← Λ ]
                 x ← INFO(α)
                 call RETNODE(α) ]
end DELETE_DEQUE
```

Outline

- What is a queue?
  - Straight queue
  - Circular Queue
- Sequential Implementation
- Linked Implementation
- Application: Topological sort
- Deques
- **Final Notes**

Final Notes

- Queues are usually implemented in problems that resemble first-come, first-served policies, as well as traversal algorithms
- Issue in TS: What if there are loops, i.e. A < B < C < A?
- Deques are not commonly used in practice; nonetheless, implementing the deque ADT is an interesting academic exercise.
  - But there are interesting practical applications, e.g. processor job scheduling

Thank you
for your attention.

Questions ...