Stacks and Queues (Ref: CLRS 10)

- Data Structures
- Stacks and Queues
- Linked lists

These notes are heavily inspired by notes from Philip Bille and Inge Li Gørtz for the course Algorithms and Data Structures at DTU, http://www2.compute.dtu.dk/courses/02105+02326/2015/#generelinfo

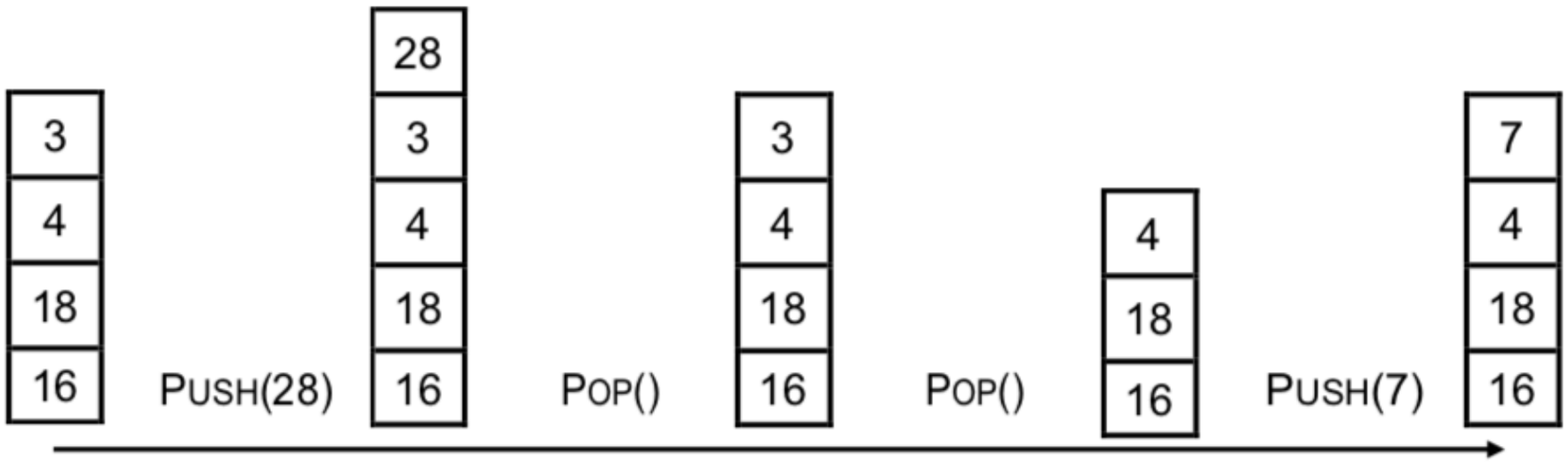
Introduction to Data Structures

- Data Structure: A method to organize data for efficient searching, access, and manipulation.
- Goal: Fast and compact.
- Terminology: Dynamic vs. Static data structures.

Stacks

Goal: Maintain a dynamic sequence (the stack) S of elements under the following operations.

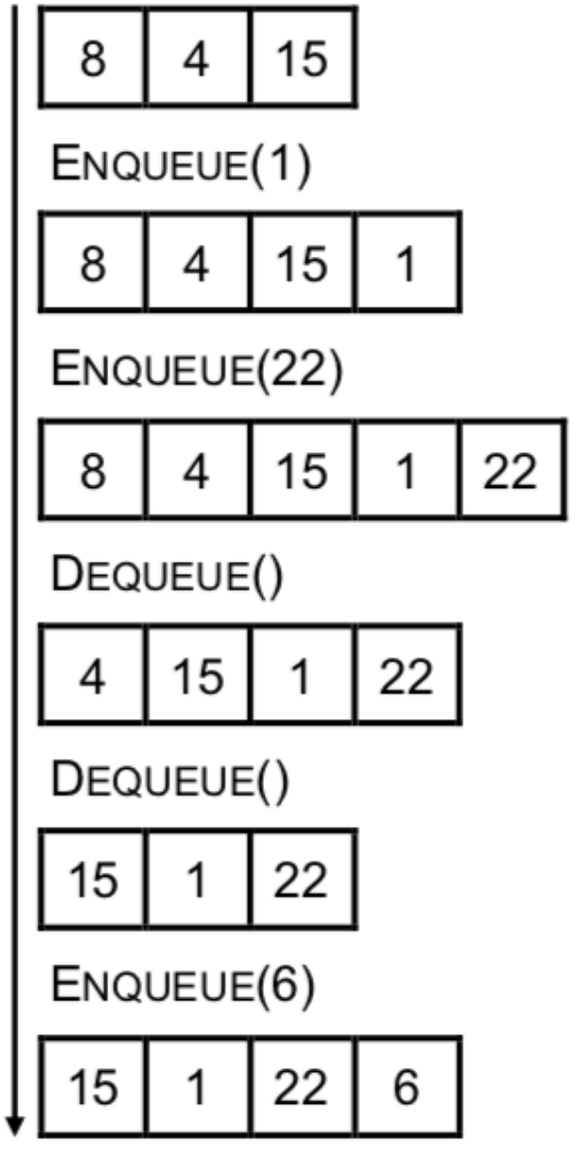
- PUSH(x): Add a new element x to S
- POP( ): Remove and return the most recently added element in S.
- ISEMPTY( ): Return true if S contains no elements.



Queues

Goal: Maintain a dynamic sequence (the queue) Q of elements under the following operations.

- ENQUEUE(x): Add a new element x to Q.
- DEQUEUE( ): Remove and return the earliest added element in Q.
- ISEMPTY( ): Return true if Q contains no elements.



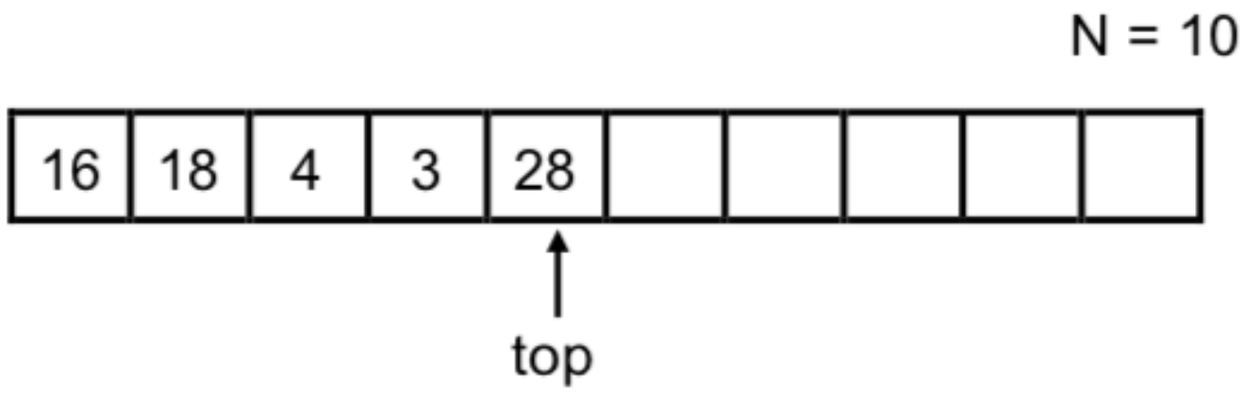
Applications

- Stacks
  - Virtual Machines
  - Parsing
  - Function calls
  - Backtracking

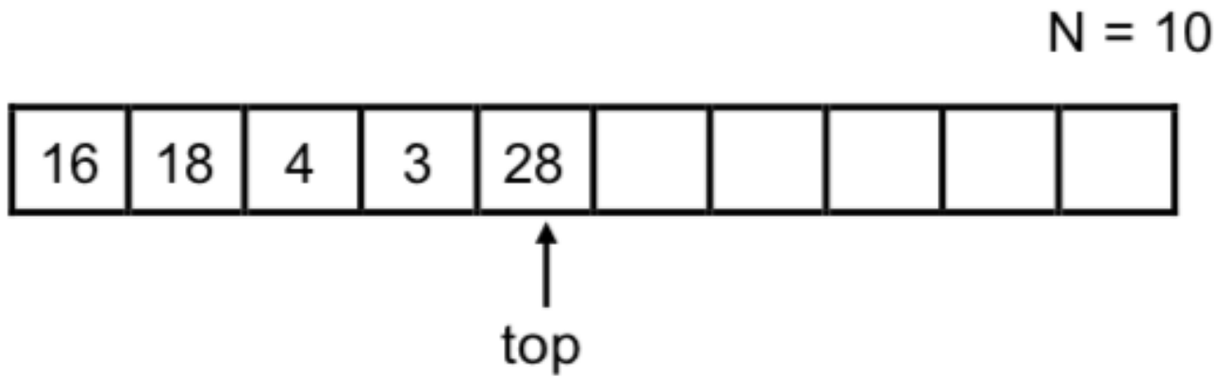
- Queues
  - Process Scheduling
  - Buffering
  - Breadth-First Search

Implementation of Stack with Arrays

- Stack with capacity N using an array
- Data Structure
  - Array S[1..N]
  - Index top in S
- Operations
  - PUSH(x): Place x in S[top+1], and set top = top + 1.
  - POP( ): Return S[top], set top = top -1.
  - ISEMPTY( ): Return true if and only if top = 0.
  - Check for overflow and underflow in PUSH and POP.



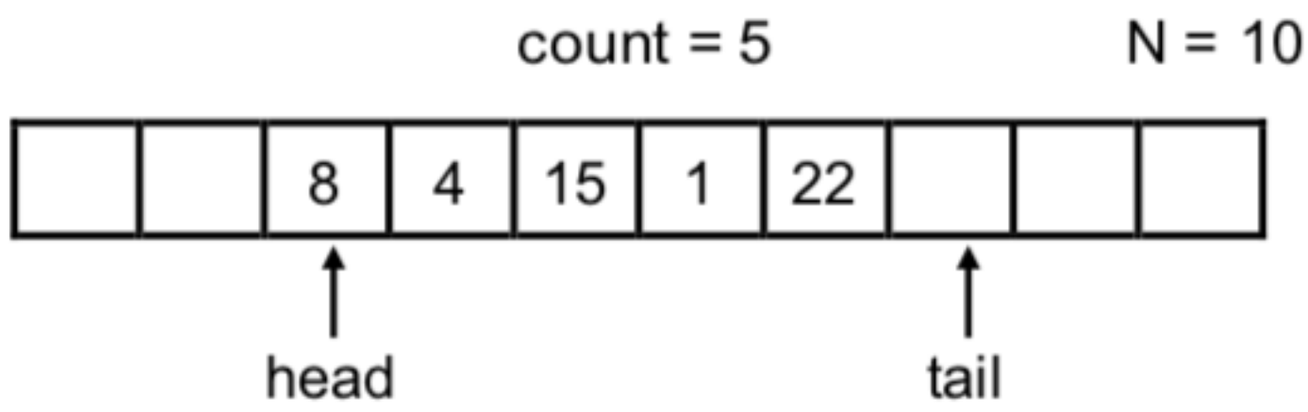
Implementation of Stack with Arrays



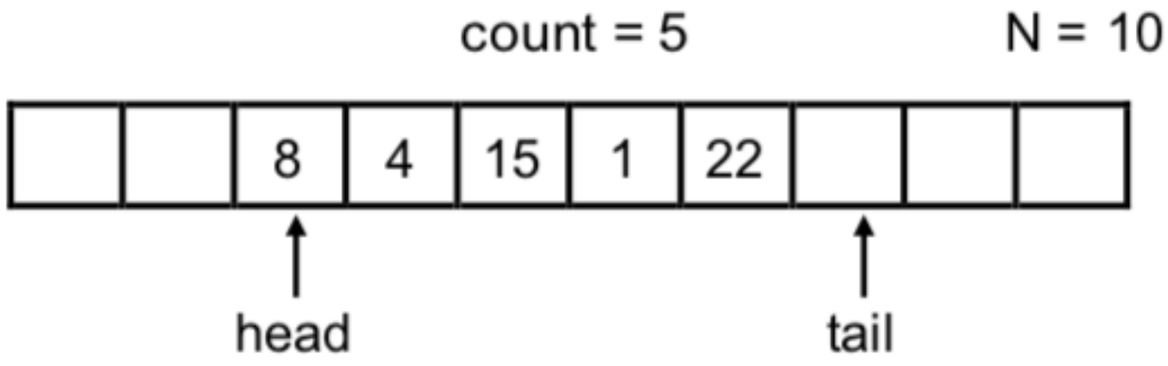
- Running time
  - PUSH: Takes $\Theta(1)$ time.
  - POP: Takes $\Theta(1)$ time.
  - ISEMPTY: Takes $\Theta(1)$ time.
- Space: $\Theta(N)$
- Problems
  - Need to know N from the beginning.
  - Waste space if the actual number of elements is << N.

Implementation of Queues with Arrays

- Queue with capacity N using a table.
- Data Structure:
  - Table S[1..N]
  - Indices head (earliest inserted element) and tail (next available element) in S, and counter count (number of elements in the queue).
- Operations:
  - ENQUEUE(x): Add x to S[tail], update count and tail cyclically.
  - DEQUEUE( ): Return S[head], update count and head cyclically.
  - ISEMPTY( ): Return true if and only if count = 0.
- Check for overflow and underflow in ENQUEUE and DEQUEUE.



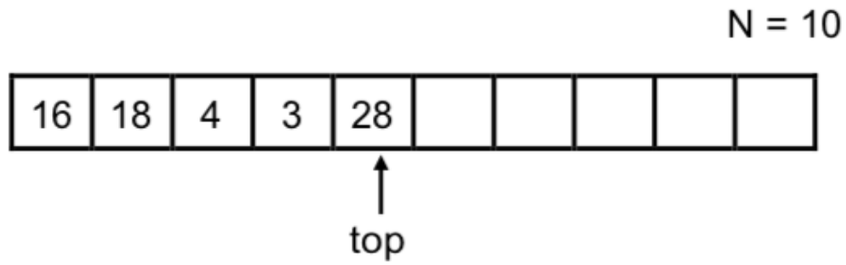
Implementation of Queue with Arrays



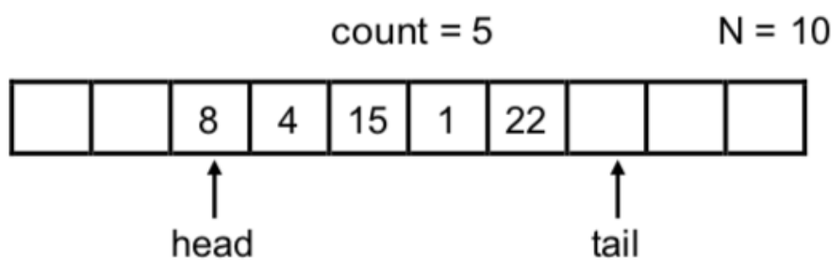
- Running time
  - ENQUEUE: Takes $\Theta(1)$ time.
  - DEQUEUE: Takes $\Theta(1)$ time.
  - ISEMPTY: Takes $\Theta(1)$ time.
- Space: $\Theta(N)$
- Problems
  - Need to know N from the beginning.
  - Waste space if the actual number of elements is << N.

Stacks and Queues (Summary)

- Stacks
  - Time: PUSH, POP, ISEMPTY in $\Theta(1)$ time.
  - Space: $\Theta(N)$



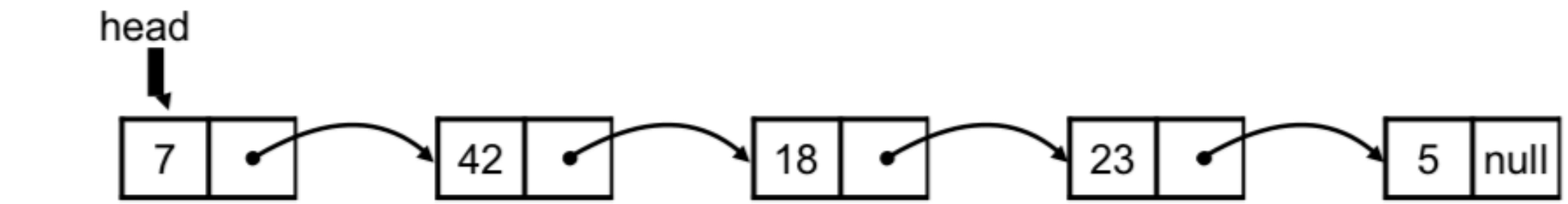
- Queues
  - Time: ENQUEUE, DEQUEUE, ISEMPTY in $\Theta(1)$ time.
  - Space: $\Theta(N)$

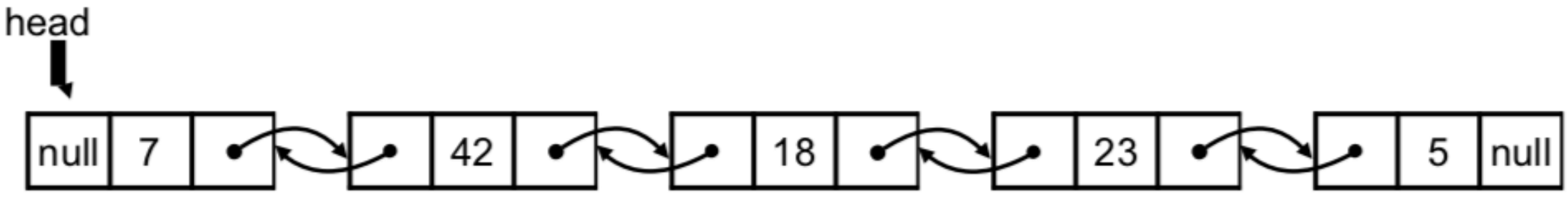


Challenge: Can we achieve linear space with the same time complexity without knowing N in advance?

Linked Lists

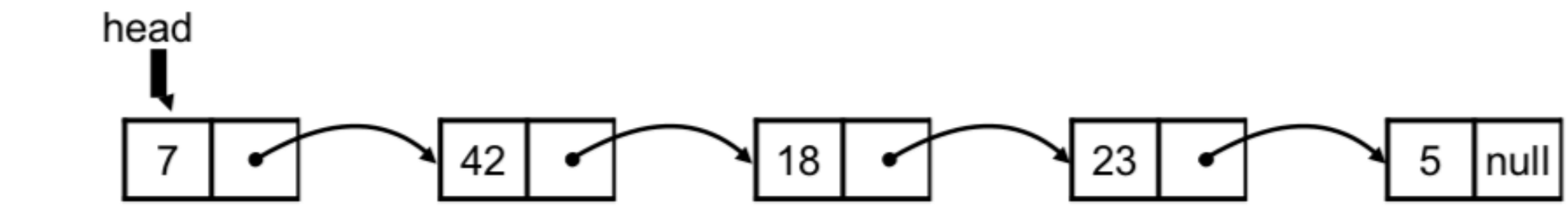
- Data structure to maintain a dynamic sequence of elements in linear space.
- The order of elements is determined by references/pointers called links.
- Efficient for inserting and removing elements or contiguous parts of elements.
- Doubly-linked vs. singly-linked.

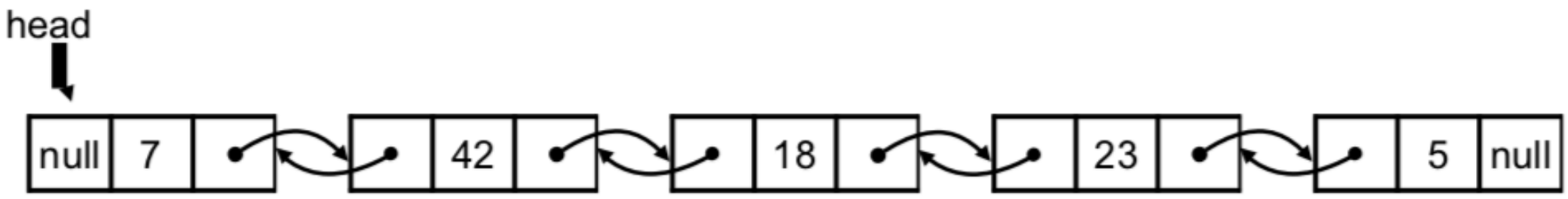




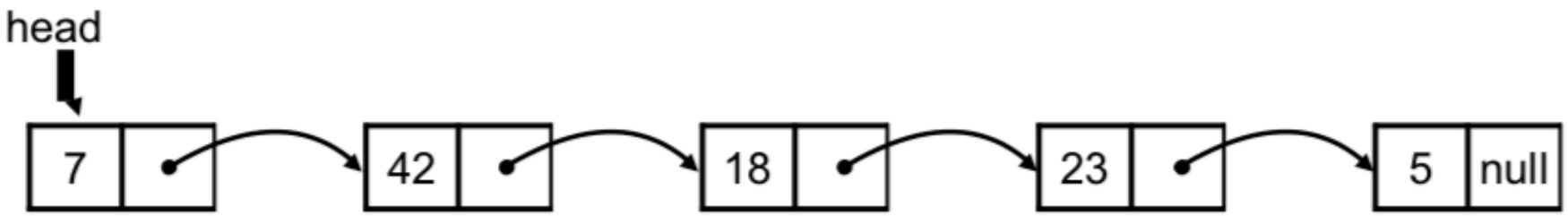
Linked lists

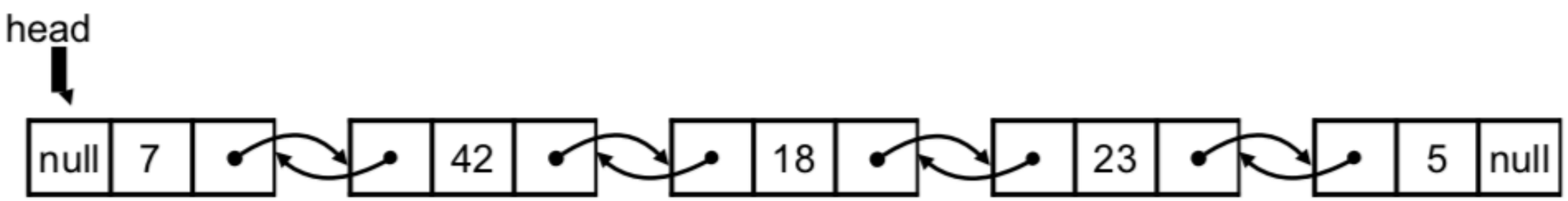
- Simple Operations
  - SEARCH(head, x): Return the node with value x in the list. Return null if not found.
  - INSERT(head, x): Insert node x at the beginning of the list. Return new head.
  - INSERT(y, x): Insert node x after node y (requires having y in advance).
  - DELETE(head, x): Remove node x from the list.





Linked lists

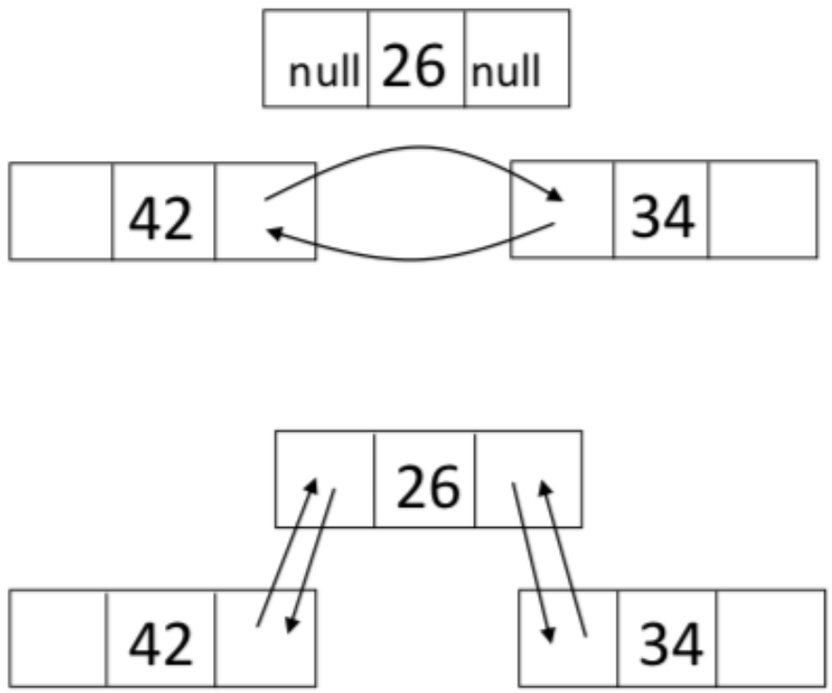




- Running Time
  - SEARCH in $\Theta(n)$ time.
  - INSERT and DELETE in $\Theta(1)$ time.
- Space: $\Theta(n)$ (n = number of elements stored)

Linked lists

Implementation of INSERT: Change pointers to ensure that the new node is between the two existing ones.



DELETE: Do the opposite.

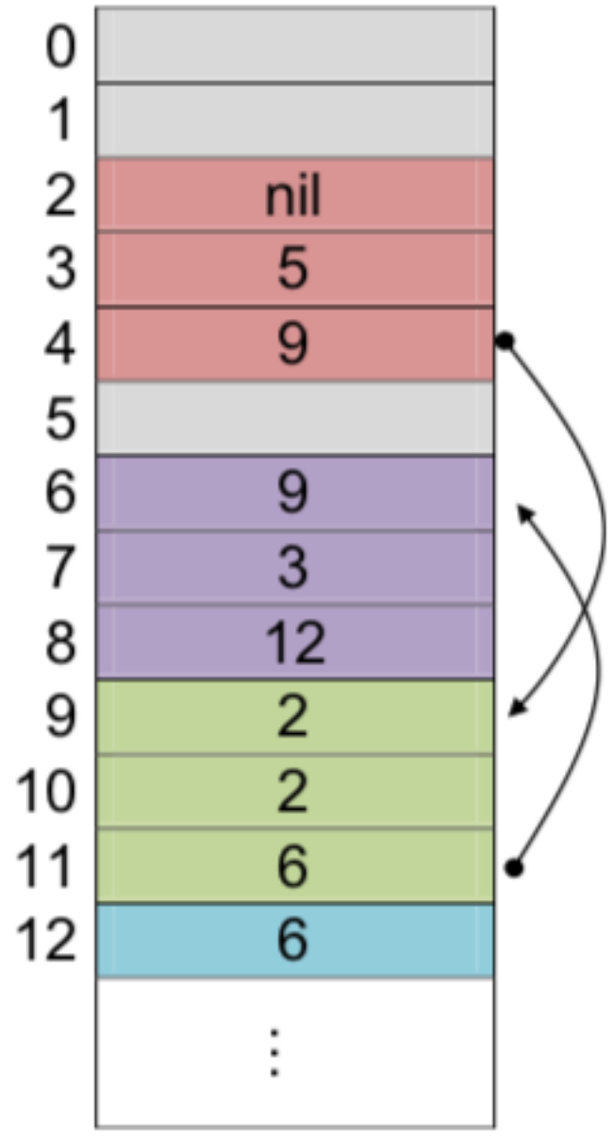
Linked lists in the RAM model

- The pointer to a node points to its first location in the RAM
- Each node takes up 3 words in the RAM

See example below: The first node in the list is placed in locations 2 to 4 in the RAM.

The colours to the right correspond to the nodes in the linked list below. The nodes that contain the numbers 1 and 8 are not visible.

Note how the nodes in the linked list do not necessarily follow one another in the RAM (but all the memory allocated to just a single node does). There can also be unused space between the nodes.

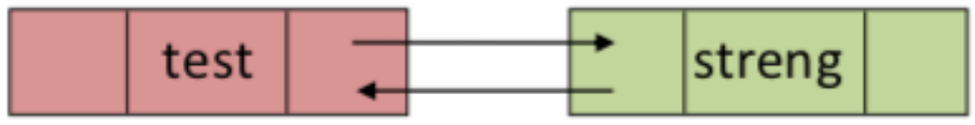


Linked lists in the RAM model

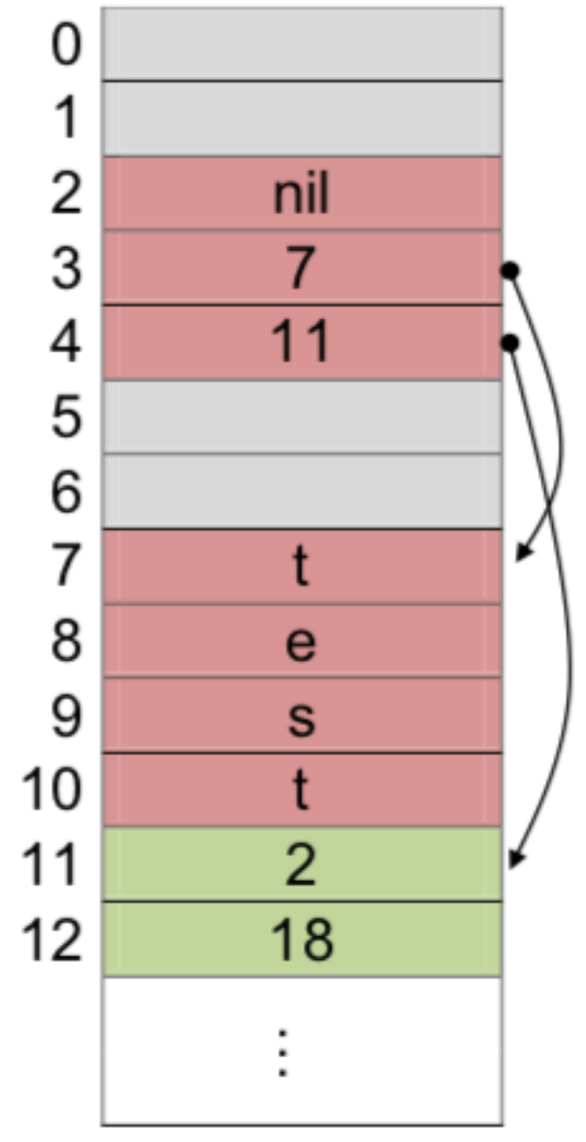
What if a node contains more than one number?

Same idea, except that the node now contains a pointer to where the data is stored.

See example below.



Note: Each letter occupies one word of the RAM. In practice, there would also be a word after the letters of “test” to indicate that we have reached the end of the string, but we have omitted that here.



Implementing stacks and queues with linked lists

Stack: Maintain a dynamic sequence (the stack) S of elements under the following operations.

- PUSH(x): Add a new element x to S
- POP( ): Remove and return the most recently added element in S.
- ISEMPTY( ): Return true if S contains no elements.

Queue: Maintain a dynamic sequence (the queue) Q of elements under the following operations.

- ENQUEUE(x): Add a new element x to Q.
- DEQUEUE( ): Remove and return the earliest added element in Q.
- ISEMPTY( ): Return true if Q contains no elements.

Think about how you can implement stacks and queues with linked lists.

Implementing stacks and queues with linked lists

- Stacks
  - PUSH, POP, ISEMPTY in $\Theta(1)$ time.
  - Space: $\Theta(n)$ (n = number of elements stored)
- Queues
  - ENQUEUE, DEQUEUE, ISEMPTY in $\Theta(1)$ time.
  - Space: $\Theta(n)$

Linked lists

Linked lists: Flexible data structure to maintain a sequence of elements in linear space.

Other linked data structures: Circular linked lists, Trees, Graphs..