Introduction to Data Structures

- Data Structures
- Stacks and Queues
- Linked Lists
- Dynamic Arrays

Philip Bille

Introduction to Data Structures

- Data Structures
- Stacks and Queues
- Linked Lists
- Dynamic Arrays

Data Structures

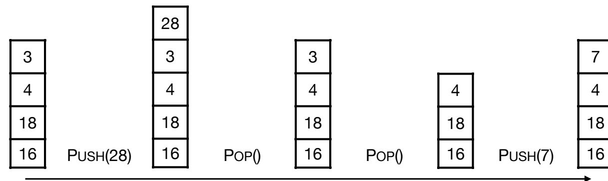
- Data structure. Method for organizing data for efficient access, searching, manipulation, etc.
- Goal.
  - Fast.
  - Compact
- Terminology.
  - Abstract vs. concrete data structure.
  - Dynamic vs. static data structure.

Introduction to Data Structures

- Data Structures
- Stacks and Queues
- Linked Lists
- Dynamic Arrays

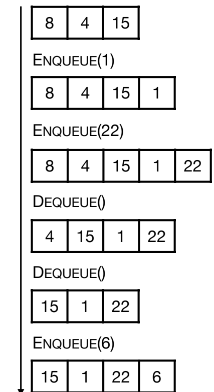
Stack

- Stack. Maintain dynamic sequence (stack) S supporting the following operations:
  - PUSH(x): add x to S.
  - POP(): remove and return the most recently added element in S.
  - ISEMPTY(): return true if S is empty.



Queue

- Queue. Maintain dynamic sequence (queue) Q supporting the following operations:
  - ENQUEUE(x): add x to Q.
  - DEQUEUE(): remove and return the earliest added element in Q.
  - ISEMPTY(): return true if Q is empty.

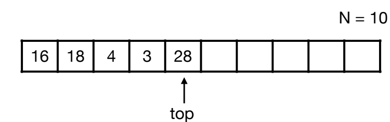


Applications

- Stacks.
  - Virtual machines
  - Parsing
  - Function calls
  - Backtracking
- Queues.
  - Scheduling processes
  - Buffering
  - Breadth-first searching

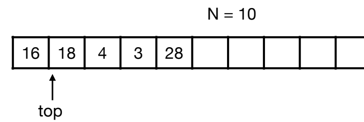
Stack Implementation

- Stack. Stack with capacity N
- Data structure.
  - Array S[0..N-1]
  - Index top. Initially top = -1
- Operations.
  - PUSH(x): Add x at S[top+1], top = top + 1
  - POP(): return S[top], top = top - 1
  - ISEMPTY(): return true if top = -1.
  - Check for overflow and underflow in PUSH and POP.



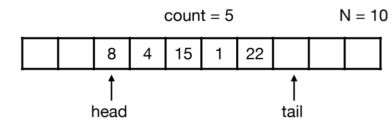
Stack Implementation

- Time
  - PUSH in O(1) time.
  - POP in O(1) time.
  - ISEMPTY in O(1) time.
- Space.
  - O(N) space.
- Limitations.
  - Capacity must be known.
  - Wasting space.



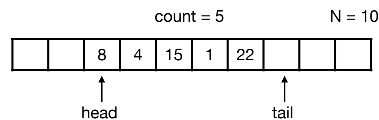
Queue Implementation

- Queue. Queue with capacity N.
- Data structure.
  - Array Q[0..N-1]
  - Indices head and tail and a counter.
- Operations.
  - ENQUEUE(x): add x at Q[tail], update count and tail cyclically.
  - DEQUEUE(): return Q[head], update count and head cyclically.
  - ISEMPTY(): return true if count = 0.
  - Check for overflow and underflow in DEQUEUE and ENQUEUE.



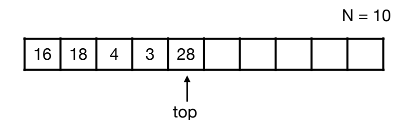
Queue Implementation

- Time.
  - ENQUEUE in O(1) time.
  - DEQUEUE in O(1) time.
  - ISEMPTY in O(1) time.
- Space.
  - O(N) space.
- Limitations.
  - Capacity must be known.
  - Wasting space.

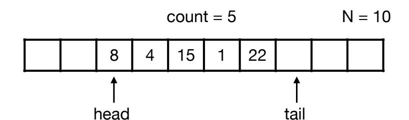


Stacks and Queues

- Stack.
  - Time. PUSH, POP, ISEMPTY in O(1) time.
  - Space. O(N)



- Queue.
  - Time. ENQUEUE, Dequeue, ISEMPTY in O(1) time.
  - Space. O(N)



- Challenge. Can we get linear space and constant time?

Introduction to Data Structures

- Data Structures
- Stacks and Queues
- Linked Lists
- Dynamic Arrays

Linked Lists

- Linked lists.
  - Data structure to maintain a dynamic sequence of items.
  - Recursive data structure. A linked list is either:
    - Empty
    - A reference to a node that has a reference to a linked list.
- Node.
  - An object that stores the item (or reference to the item) and the reference to a linked list.

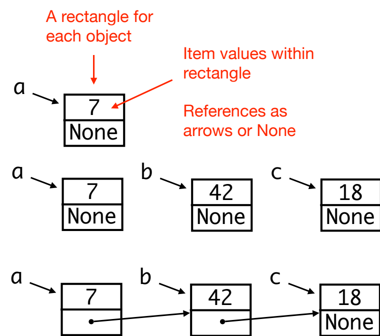
Linked Lists

```
class Node:
    def __init__(self, item, next):
        self.item = item
        self.next = next
```

```
a = Node(7, None)
```

```
b = Node(42, None)
c = Node(18, None)
```

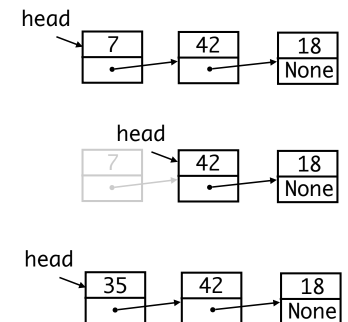
```
a.next = b
b.next = c
```



Linked Lists

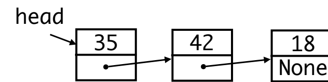
```
head = head.next
```

```
tmp = head
head = Node(35, tmp)
```



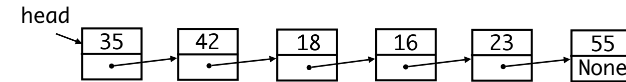
Linked Lists

```
x = head
while x != None:
    print(x.item)
    x = x.next
```



Linked Lists

- Linked lists.
  - Data structure to maintain a dynamic sequence of items.



- Space.
  - A list of n items uses O(n) space.
- Time.
  - Insert at the head of the list in O(1) time.
  - Delete at the head of the list in O(1) time.
  - Traverse the list in O(n) time.

Linked Lists

- Exercise. Consider how to implement stack and queue with linked lists efficiently.

- Stack. Maintain dynamic sequence (stack) S supporting the following operations:
  - PUSH(x): add x to S.
  - POP(): remove and return the most recently added element in S.
  - ISEMPTY(): return true if S is empty.

- Queue. Maintain dynamic sequence (queue) Q supporting the following operations:
  - ENQUEUE(x): add x to Q.
  - DEQUEUE(): remove and return the earliest added element in Q.
  - ISEMPTY(): return true if S is empty.

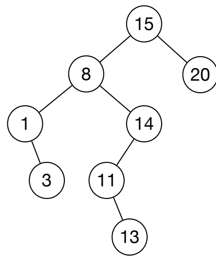
Linked Lists

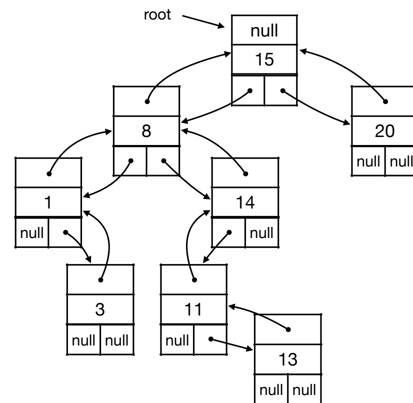
- Stacks and queues using linked lists
- Stack.
  - Time. PUSH, POP, ISEMPTY in O(1) time.
  - Space. O(n)

- Queue.
  - Time. ENQUEUE, DEQUEUE, ISEMPTY in O(1) time.
  - Space. O(n)

Linked Lists

- Linked list. Flexible data structure to maintiain sequence of elements.
- Other linked data structures: cyclic lists, trees, graphs, …





Introduction to Data Structures

- Data Structures
- Stacks and Queues
- Linked Lists
- Dynamic Arrays

Stack Implementation with Array

- Challenge. Can we implement a stack efficiently with arrays?
  - Do we need a fixed capacity?
  - Can we get linear space and constant time?

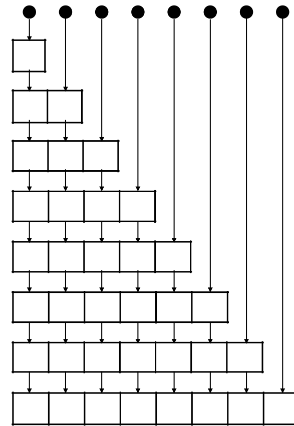
Dynamic Arrays

- Goal.
  - Implement a stack using arrays in O(n) space for n elements.
  - As fast as possible.
  - Focus on PUSH. Ignore POP and ISEMPTY for now.

- Solution 1
  - Start with array of size 1.
- PUSH(x):
  - Allocate new array of size + 1.
  - Move all elements to new array.
  - Delete old array.

Dynamic Arrays

- PUSH(x):
  - Allocate new array of size + 1.
  - Move all elements to new array.
  - Delete old array.
- Time. Time for n PUSH operations?
  - ith PUSH takes O(i) time.
  - ⇒ total time is $1 + 2 + 3 + 4 + \ldots + n = O(n^2)$
- Space. O(n)
- Challenge. Can we do better?

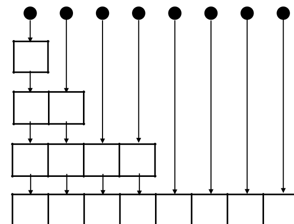


Dynamic Arrays

- Idea. Only copy elements some times
- Solution 2.
  - Start with array of size 1.
- PUSH(x):
  - If array is full:
    - Allocate new array of twice the size.
    - Move all elements to new array.
    - Delete old array.

Dynamic Arrays

- PUSH(x):
  - If array is full:
    - Allocate new array of twice the size.
    - Move all elements to new array.
    - Delete old array.
- Time. Time for n PUSH operations?
  - PUSH 2^k takes $O(2^k)$ time.
  - All other PUSH operations take O(1) time.
  - ⇒ total time $< 1 + 2 + 4 + 8 + 16 + \ldots + 2^{\log n} + n = O(n)$
- Space. O(n)



Dynamic Arrays

- Stack with dynamic array.
  - n PUSH operations in O(n) time and space.
  - Extends to n PUSH, POP og ISEMPTY operations in O(n) time.
- Time is amortized O(1) per operation.
- With more clever tricks we can deamortize to get O(1) worst-case time per operation.
- Queue with dynamic array.
  - Similar results as stack.
- Global rebuilding.
  - Dynamic array is an example of global rebuilding.
  - Technique to make static data structures dynamic.

Stack and Queues

Data structure	PUSH	POP	ISEMPTY	Space
Array with capacity N	O(1)	O(1)	O(1)	O(N)
Linked List	O(1)	O(1)	O(1)	O(n)
Dynamic Array 1	O(n)	O(1)†	O(1)	O(n)
Dynamic Array 2	O(1)†	O(1)†	O(1)	O(n)
Dynamic Array 3	O(1)	O(1)	O(1)	O(n)

† = amortized

Introduction to Data Structures

- Data Structures
- Stacks and Queues
- Linked Lists
- Dynamic Arrays