Programming (Econometrics)

Lecture 5: Linear data structures

Tommi Tervonen

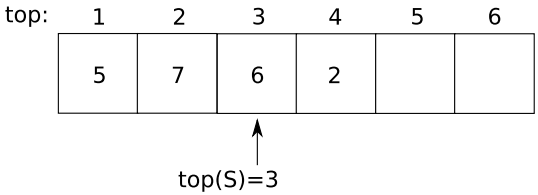
Econometric Institute, Erasmus University Rotterdam

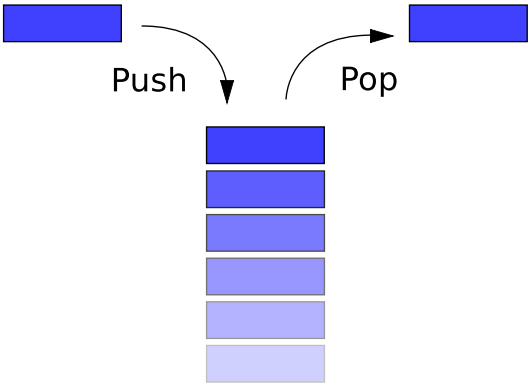
Array

- *Data structures* allow to store a set of elements and guarantee certain complexity for elementary operations (access, insert, delete, search x, search min/max)
- Arrays are the most elementary data structures
- Random access: O(1)
- All other operations: O(n)

Stack

- Last-In First-Out access semantics (LIFO)
- Can be implemented using an array and index of top element



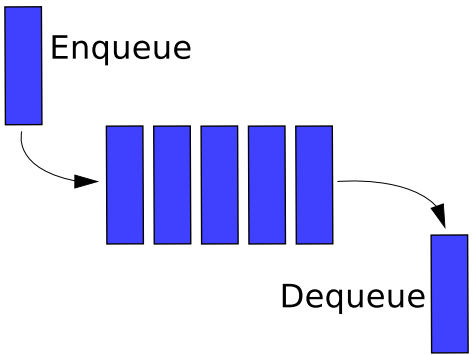


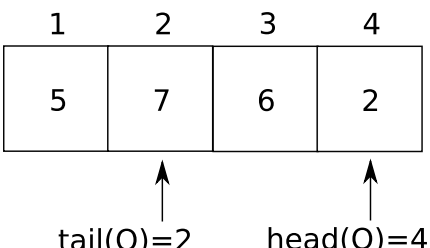
Use of stacks: call stack

- When you make a method call, the new frame of execution (local variables) is pushed to the stack
- When the method exits, the local variables are simply popped from the stack

Queue

- First-In First-Out access semantics (FIFO)
- Can be implemented using an array and indices of first (head) and last (tail) elements
- Empty queue is denoted with head $= 0$, tail $= 1$





4 queues (3, 7, 6, 2), 3 dequeues (delete 3, 7 and 6), and 1 enqueue (5): contents of Q are [2, 5].

Use of queues: simulation with (multiple) queues

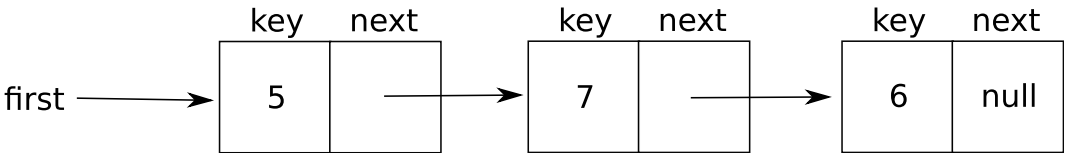
- Many practical problems can be modeled with queues
- E.g. factory arriving material to be processed, process step 1 storage, process step 2 storage, ...

Dynamic data structures

- Until now all the data structures we considered have been static
- When elements are constantly inserted / deleted, static structures are slow (O(n))
- Need for node-based dynamic structures

Linked list

- Linked list is a list where each element has its own node, that contains the key and a reference to the next node
- Can be used to implement a stack



Pass by reference in Matlab with OO

- OO-extensions and their use in passing by reference

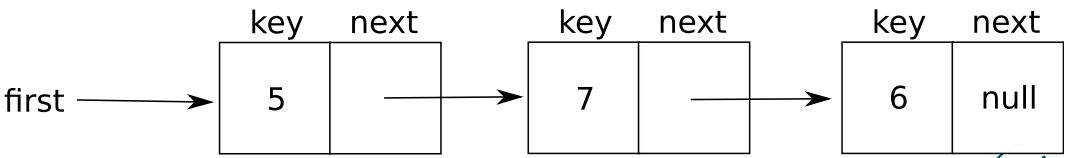
```
classdef node < handle
  properties
    key
    next
  end
end
```

Linked list construction

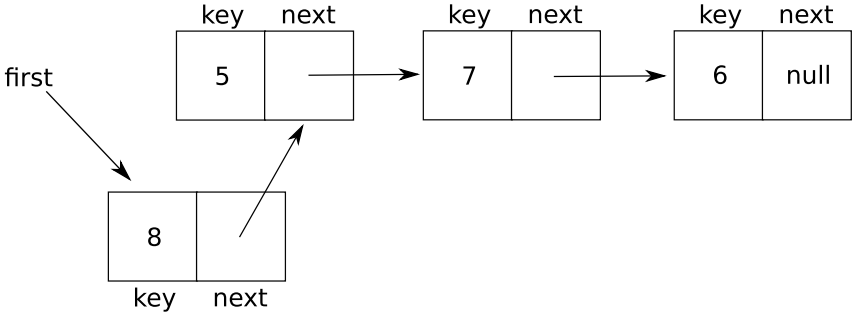
```
classdef linkedlist < handle
  properties
    first
  end
end

function L = initLinkedList(value)
  L = linkedList();
  fNode = node();
  fNode.key = value;
  fNode.next = [];
  L.first = fNode;
end
```

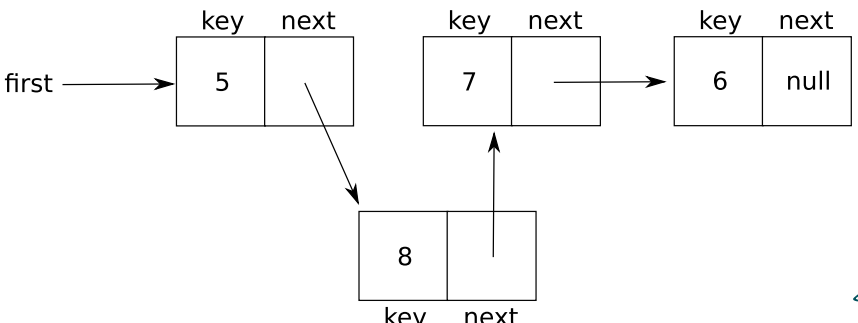
Linked list traversal

```
function node = findKey(L, key)
  curNode = L.first;
  while (~isempty(curNode))
    if (curNode.key == key)
      node = curNode;
      break;
    end
    curNode = curNode.next;
  end
end
```

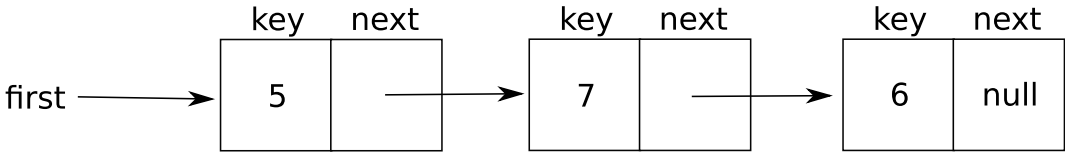


Linked list insert element



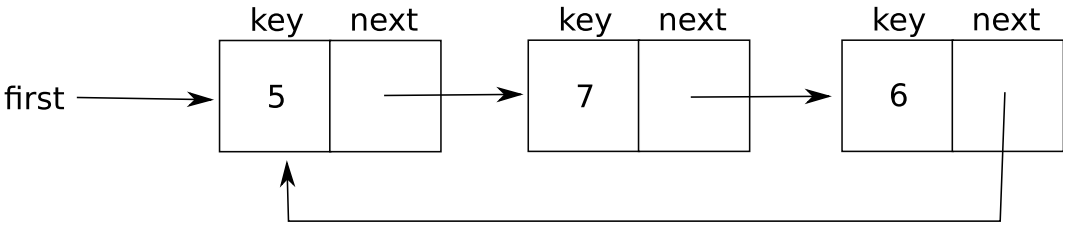


Linked list delete element



deleteNodeAfter(nodeOf5)

Circular linked list



- Uses: round-robin scheduling of processes in multi-tasking environments (e.g. your computer)

Complexity of linked list operations

- Insert/delete element in beginning: O(1)
- Insert/delete element at current iteration location: O(1)
- Random access: O(n)
- Search: O(n)