CSCI-UA 480.4: APS
Algorithmic Problem Solving

Linear Data Structures

Instructor: Joanna Klukowska

created based on materials for this class by
Bowen Yu and materials shared by the authors of
the textbook Steven and Felix Halim

Questions

- homework 1 questions ?

Basic Linear Data Structures

- list
  - array
  - linked
- stack
- queue

Basic Linear Data Structures

- list
  - array
  - linked
- stack
- queue

Operations that determine performance of a data structure:

- insertion
- deletion
- query / find
- update / modify

Basic Linear Data Structures

- list
  - array
  - linked
- stack
- queue

Operations that determine performance of a data structure:

- insertion
- deletion
- query / find
- update / modify

Data structures are separate from their implementations. For most there are many different ways to implement them.

Basic Linear Data Structures

- list
  - array
  - linked
- stack
- queue

Operations that determine performance of a data structure:

- insertion
- deletion
- query / find
- update / modify

Data structures are separate from their implementations. For most there are many different ways to implement them.

Most of the linear data structures are implemented in built-in libraries.

Lists

List as a Static Array

static array = fixed size, no need to resize it

- natively supported by both C/C++ and Java
- can be declared with appropriate size up-front if the problem specifies the maximum input size (HINT: use extra buffer for safety to avoid going out of bounds)
- can be multi-dimensional: 1D, 2D, 3D, ...

List as a Static Array

static array = fixed size, no need to resize it

- natively supported by both C/C++ and Java
- can be declared with appropriate size up-front if the problem specifies the maximum input size (HINT: use extra buffer for safety to avoid going out of bounds)
- can be multi-dimensional: 1D, 2D, 3D, ...
- allocated in memory as consecutive memory locations (important for fast accesses)
- assume that indexes in use are all in the front (starting at low indexes) and there are no gaps

List as a Static Array

static array = fixed size, no need to resize it

- natively supported by both C/C++ and Java
- can be declared with appropriate size up-front if the problem specifies the maximum input size (HINT: use extra buffer for safety to avoid going out of bounds)
- can be multi-dimensional: 1D, 2D, 3D, ...
- allocated in memory as consecutive memory locations (important for fast accesses)
- assume that indexes in use are all in the front (starting at low indexes) and there are no gaps

- performance of operations
  - insert/delete in the back O(1)
  - insert/delete in the front or *middle* O(N)
  - update/modify O(???)
  - find O(???)

List as a Static Array

static array = fixed size, no need to resize it

- natively supported by both C/C++ and Java
- can be declared with appropriate size up-front if the problem specifies the maximum input size (HINT: use extra buffer for safety to avoid going out of bounds)
- can be multi-dimensional: 1D, 2D, 3D, ...
- allocated in memory as consecutive memory locations (important for fast accesses)
- assume that indexes in use are all in the front (starting at low indexes) and there are no gaps

- performance of operations
  - insert/delete in the back O(1)
  - insert/delete in the front or *middle* O(N)
  - update/modify: using index O(1), if need to find, then see below
  - find O(N) if not sorted, O(logN) if sorted

List as a Static Array

Java 1D and 2D static array

C/C++

List as a Dynamic/Resizable Array

- implementation provided by buil-in classes:
  - in C++ STL
  - or in Java (is faster because it is unsynchronized)
- used when required size is not known at compile time

List as a Dynamic/Resizable Array

- implementation provided by buil-in classes:
  - in C++ STL
  - or in Java (is faster because it is unsynchronized)
- used when required size is not known at compile time

performance of operations

use **amortized analysis**: average/amortized performance of a sequence of operations rather than each single operation
reason ???

List as a Dynamic/Resizable Array

- implementation provided by buil-in classes:
  - in C++ STL
  - or in Java (is faster because it is unsynchronized)
- used when required size is not known at compile time

performance of operations

use **amortized analysis**: average/amortized performance of a sequence of operations rather than each single operation
reason: because this way the cost of a resize and copy of data is averaged out
(works only if resizing is done by multiplicative factor)

List as a Dynamic/Resizable Array

- implementation provided by buil-in classes:
  - in C++ STL
  - or in Java (is faster because it is unsynchronized)
- used when required size is not known at compile time

performance of operations

use **amortized analysis**: average/amortized performance of a sequence of operations rather than each single operation
reason: because this way the cost of a resize and copy of data is averaged out
(works only if resizing is done by multiplicative factor)

- insert/delete in the back O(1)
- insert/delete in the front or *middle* O(N)
- update/modify: using index O(1), if need to find, then see below
- find O(N) if not sorted, O(logN) if sorted

List as a Linked Structure (a Linked List)

- implementation provided by buil-in classes:
  - in C++ STL
  - in Java
- rarely used due to poor performance of accessing elements
- (good exercise: implement your own in both Java and C++ to practice reference/pointer operations)
- performance of operations
  - insert/delete in the back/front O(1) (assume a doubly linked)
  - insert/delete in the *middle* O(N)
  - update/modify: O(N) (except for in the front/back)
  - find O(N)

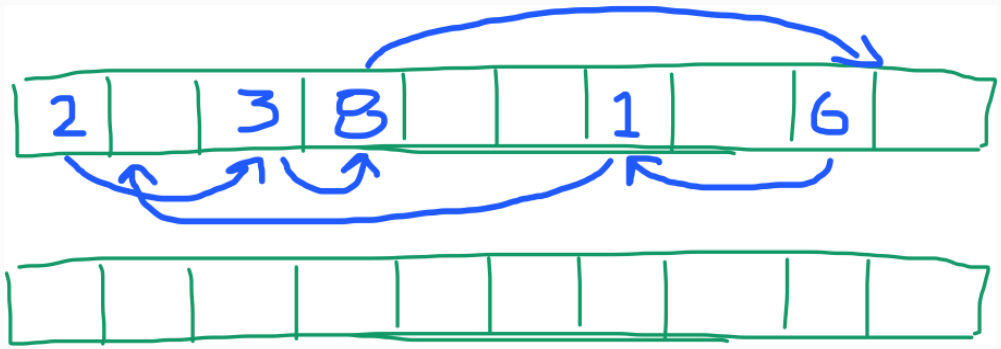
List *Hybrids*

- dynamically allocated list that consists of short fixed sized arrays that are connected into a linked list

List *Hybrids*

- dynamically allocated list that consists of short fixed sized arrays that are connected into a linked list

- linked list in which the "links" are provided by indexes in a separate array



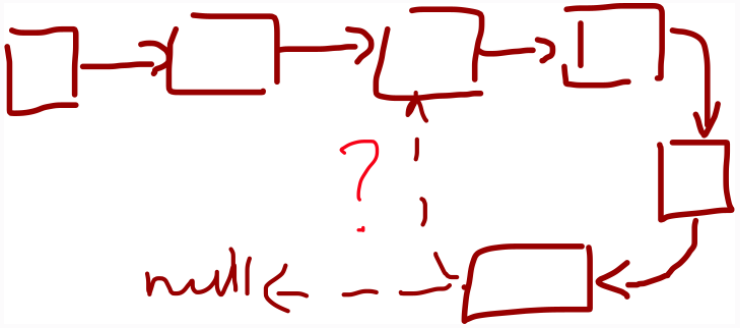
Challenge

Given a pointer/reference to a singly linked list, determine if it has a loop.

Determine the length of the loop if it exists.

Restrictions:

- Elements in the list are not unique.
- Do not use extra storage.



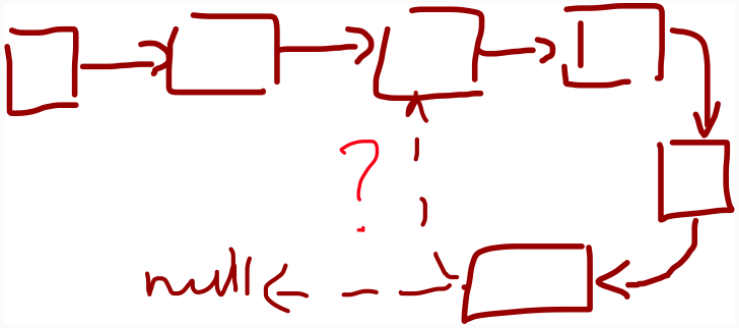
Challenge

Given a pointer/reference to a singly linked list, determine if it has a loop.

Determine the length of the loop if it exists.

Restrictions:

- Elements in the list are not unique.
- Do not use extra storage.



Solution Rabbit and Turtle (two pointers)

- start two pointers at head
- advance rabbit two steps per iteration
- advance turtle one step per iteration
- if they meet (i.e., rabbit does not find the end of the list) then there is a loop and a note at which they meet is somewhere on the loop
- trace through the loop with one of them to determine the number of nodes in the loop

Challenge

- Can a singly linked list be circular?
- Can a doubly linked list be circular?
- Can a doubly linked list have a loop?

Stacks

Stack (first in last out, FILO)

- implementation provided by buil-in classes:
  - in C++ STL
  - in Java

Stack (first in last out, FILO)

- implementation provided by buil-in classes:
  - in C++ STL
  - in Java
- operations performed
  - add/push O(1) - adds to the top
  - remove/pop O(1) - removes from the top
  - top O(1) - access the element on the top (optional)
  - empty O(1) - determine if the stack is empty (optional)

Stack (first in last out, FILO)

- implementation provided by buil-in classes:
  - in C++ STL
  - in Java
- operations performed
  - add/push O(1) - adds to the top
  - remove/pop O(1) - removes from the top
  - top O(1) - access the element on the top (optional)
  - empty O(1) - determine if the stack is empty (optional)
- used in many algorithms for solving problems
  - postfix, prefix calculations and conversions
  - graph algorithms

Challenge: Function Call Stack

What is the output of ?

Challenge: Function Call Stack

What is the output of ?

Output

Challenge: Function Call Stack

What is the output of ?

Challenge: Function Call Stack

What is the output of ?

Output

Challenge: Function Call Stack

What is the output of ?

Challenge: Function Call Stack

What is the output of ?

Output

Challenge: Function Call Stack

What is the output of ?

Output

How about these functions? (Try it on your own after the class.)

Challenge: Function Call Stack

What is the output of ?

Output

How about these functions? (Try it on your own after the class.)