Algorithm design and analysis

9/9/24

What is an algorithm?

- From Wikipedia:

  An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function. (http://en.wikipedia.org/wiki/Algorithm)

- Solution to computational problem **NOT** tied to a specific programming language
  - often expressed in pseudocode with similar structure but looser syntax than “real” programming languages

This course

- Text: “Introduction to Algorithms” (CLRS 4th ed)
- Builds on CS 142, but focus on proofs rather than programming
- Most students take it late in their time at Knox

- Peer Instruction, plus working on problems
- Reading before class (sometimes w/ reading-quiz)
- HWs and exams (2 midterms and a final)

Collaboration

(One slide version; read the syllabus)

- You are **encouraged** to work on the problem sets in small groups

- Just be sure that you:
  - Credit everyone that you collaborate with and every outside source you consult
  - Write up your solution on your own and completely understand it yourself

Administrivia

- Join Google Classroom (email me if not invited)
- Review binary search tree operations; expect RQ
- Review Induction

Arrays with fast initialization

Based on Lewis and Denenberg, “Data Structures & Their Algorithms”, Harper Collins Press, pp. 136-138, 1991.

Recall: Abstract Data Types (ADTs)

- Set of operations that are supported

- Does not specify implementation and may have different “good” implementations
  - Ex: List ADT (add to end, add to front, read value at given index, remove from end, ...) can be implemented using an array or a (singly- or doubly-) linked list

How many of the following operations are asymptotically faster in an array-based list than in a doubly-linked list with a tail pointer?

I. addFront II. add III. get(i) IV. removeLast

A. 0
B. 1
C. 2
D. 3
E. 4

How many of the following operations are asymptotically faster in an array-based list than in a doubly-linked list with a tail pointer?

I. addFront II. add III. get(i) IV. removeLast

A. 0
B. 1 (just III)
C. 2
D. 3
E. 4

Array ADT

- Constructor
- Way to read i^{th} cell

  T access(int index)
- Way to set i^{th} cell

  void assign(int index, T value)
- Way to set all cells to given value

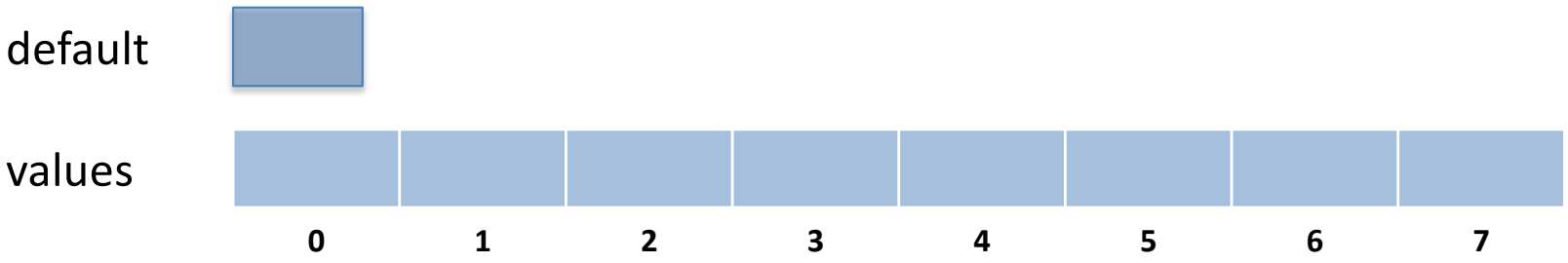
  void initialize(T value)

“Standard” implementation

- In C
  - Contiguous memory, with address arithmetic to access/assign cells
  - Use loop for initialization
- In Java
  - Constructor includes initialization
  - Adds bounds checking

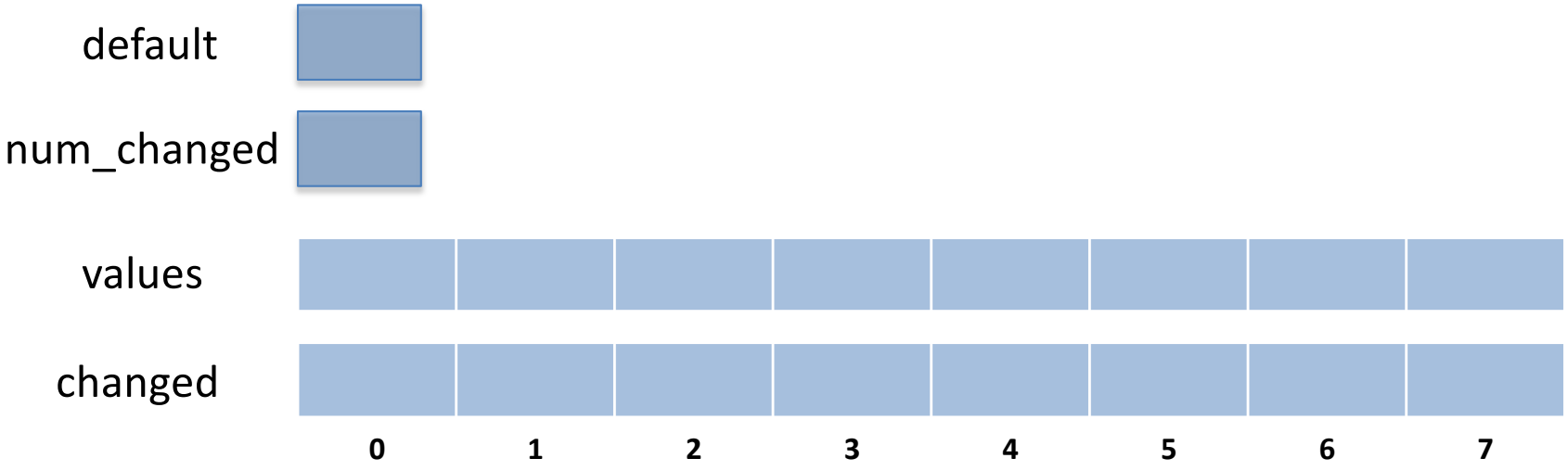
Idea 1: Default value

- Store separate “default” value
  - value of cells not changed since last initialization



Now, access “just” checks whether an index has been updated since the last initialization

Idea 2: Keep list of changes



Idea 3: Store index locations

- Stores where in list each index occurs

(hint how to find the index)