CS180 Discussion 1B Week 1

Ling Ding

Email: lingding@cs.ucla.edu

Many slides courtesy of former CS 180 TA Zhehui Zhang, Jinghao Zhao

CS180 – Discussion 1B

- **Time: Fridays 2PM ~ 3:50PM**
- **Format:** Zoom Meeting online
- **TA**: *Ling Ding*
  - PhD student in ScAi Lab, UCLA CS Dept.
  - Email: lingding@cs.ucla.edu
  - Office Hours: after discussions (F 4PM ~ 5PM)

Why attending discussion sections?

- Customized Learning
  - Interaction, Ask questions, …
- Practice problem solving **TOGETHER**
  - Extra Exercises
  - Discuss with your fellow classmates/TA
- Share learning experiences
- Help & Support one another
- Have fun together ☺
- …

Outline

- Why Study Algorithms
- About The Course
- Proof by Induction
- Data Structures
- Stable Matching

What is an algorithm?

What is an Algorithm?

- In English (i.e. in a recipe book) **Algorithm:**

  A finite set of well-defined instructions for accomplishing some task.

What is an Algorithm?

- Algorithms predate computer science! Some examples:
  - Directions from one location to another.
  - Recipes for cooking.
  - Mathematical algorithms (multiplication, division, etc.).
  - And of course, computer programs...

What is an Algorithm?

- Algorithms can be represented in many ways:
  - In a programming language (C, Java, Cobol, etc).
  - As finite state automata.
  - On a circuit board or silicon chip.

Why Study Algorithms?

Why Study Algorithms?

- important for all other branches of computer science
- plays a key role in modern technological innovation

  – "Everyone knows Moore's Law – a prediction made in 1965 by Intel co-founder Gordon Moore that the density of transistors in integrated circuits would continue to double every 1 to 2 years….in many areas, performance gains due to improvements in algorithms have vastly exceeded even the dramatic performance gains due to increased processor speed."

  - Excerpt from *Report to the President and Congress: Designing a Digital Future*, December 2010 (page 71).

Slide courtesy of Tim Roughgarden

Why Study Algorithms?

- important for all other branches of computer science
- plays a key role in modern technological innovation
- provides novel "lens" on processes outside of computer science and technology
  - quantum mechanics, economic markets, evolution

Slide courtesy of Tim Roughgarden

Why Study Algorithms?

- important for all other branches of computer science
- plays a key role in modern technological innovation
- provides novel "lens" on processes outside of computer science and technology
- challenging (i.e., good for the brain!)
- fun

Slide courtesy of Tim Roughgarden

About The Course

Subject of this Class

This class has been named “Introduction to Algorithms.” However, we have all seen algorithms before. The real subject of this class, is *design* and *analysis* of algorithms. The major questions are:

- Given a problem and an algorithm, can we prove that the algorithm actually solves the problem?
- Given an algorithm, will it terminate in a reasonable amount of time? What is a reasonable amount of time anyway?
- Given a problem, how do we come up with an algorithm to solve it?
- Are there problems which no efficient algorithm can solve? How do we identify them?

Course Topics

- Vocabulary for design and analysis of algorithms
  - E.g., “Big-Oh” notation
  - “sweet spot” for high-level reasoning about algorithms
- Algorithm design paradigms
  - **Divide and conquer** algorithm design paradigm
  - **Greedy algorithm** design paradigm
  - **Dynamic programming** algorithm design paradigm
- Primitives for reasoning about graphs
- Use and implementations of data structures
- NP-complete problems and what to do about them

Syllabus

Some of the tools to answer these questions are already familiar; others we will learn about in this class. The course itself is structured around techniques for designing algorithms. The major techniques we will cover include:

1. Basic mathematical concepts and constructions
2. Principles of algorithm design and data structures
3. Complexity, efficiency and tractability of algorithms
4. Basic constructions and properties of graphs and algorithms on graphs
5. Greedy algorithms
6. Divide-and-conquer
7. Dynamic programming
8. Network flow
9. Complexity classes
10. Randomized algorithms*
11. Algorithms that run forever*

* pending time limitations

Skills You'll Learn

- Become a better programmer
- Sharpen your mathematical and analytical skills
- Start "thinking algorithmically"
- Literacy with computer science's "greatest hits"
- Ace your technical interviews

Prerequisites

- Ideally, you know some programming.
- Doesn't matter which language(s) you know.
- Some (perhaps rusty) mathematical experience.
  - Basic discrete math, proofs by induction, etc.

Excellent free reference: "Mathematics for Computer Science", by Eric Lehman and Tom Leighton. (Easy to find on the Web.)

Supporting Materials

- A few of the many good textbooks:
  - Kleinberg/Tardos, *Algorithm Design*, 2005.
  - Dasgupta/Papadimitriou/Vazirani, *Algorithms*, 2006.
  - Cormen/Leiserson/Rivest/Stein, *Introduction to Algorithms*, 2009 (3rd edition).
  - Mehlhorn/Sanders, *Data Structures and Algorithms: The Basic Toolbox*, 2008.

Proof by Induction

Proof by Induction

- Mathematical induction infers that a statement involving a natural number n holds for all values of n. The proof consists of two steps:
  - Base case
  - Step case (Inductive Step)

Base case

- Prove that the statement holds for the first natural number n0.
  - Usually, n0 = 0 or n0 = 1;

Step case (Inductive Step)

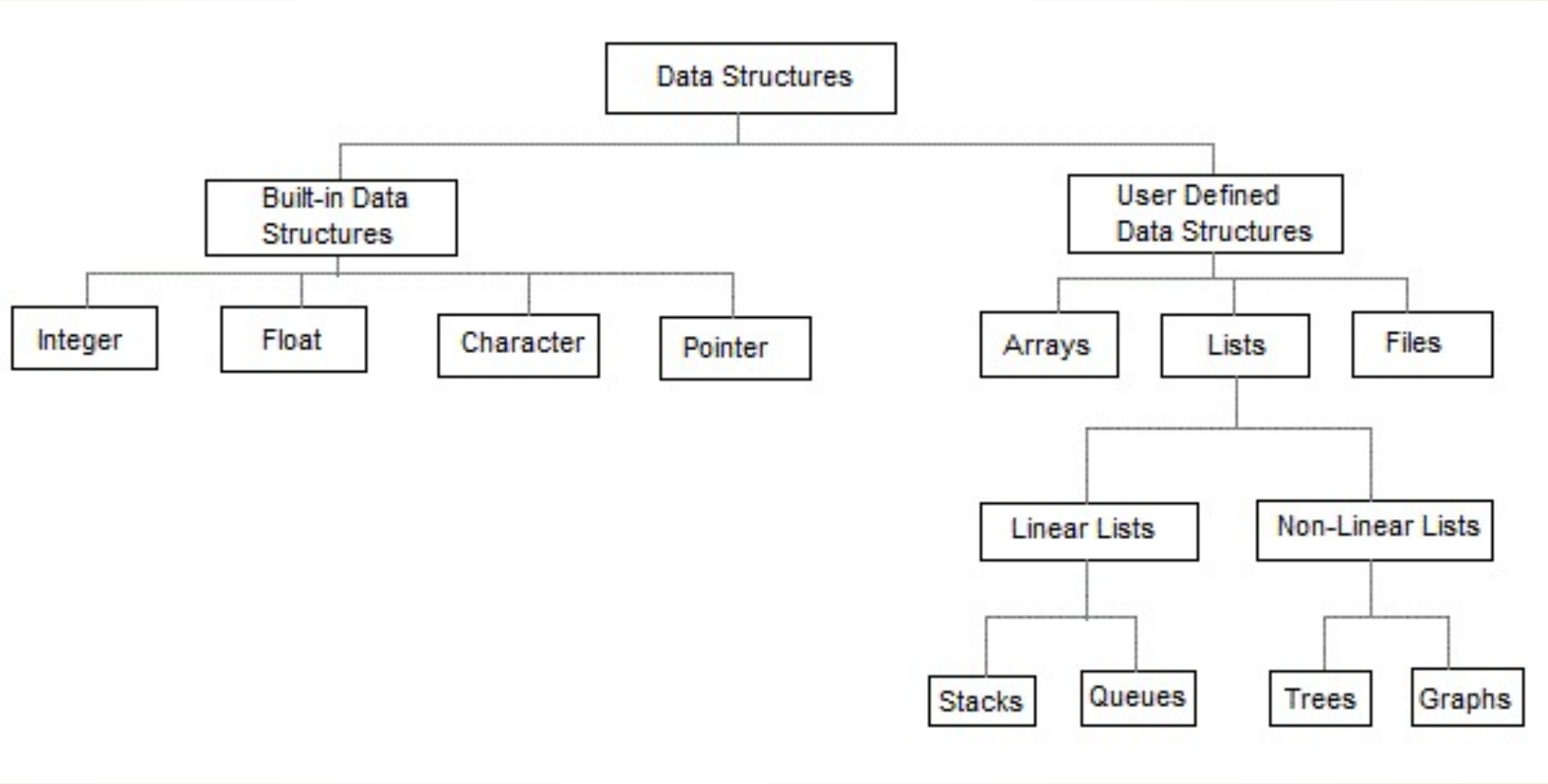
- Prove that for every n ≥ n0, if the statement holds for n, then it holds for n + 1.
  - In other words, assume the statement holds for some arbitrary natural number n ≥ n0, and prove that then the statement holds for n + 1.
- ***Induction/Inductive hypothesis***: the statement holds for some n
- To prove the inductive step, one assumes the induction hypothesis and then uses this assumption, involving n, to prove the statement for n + 1.

Data Structures

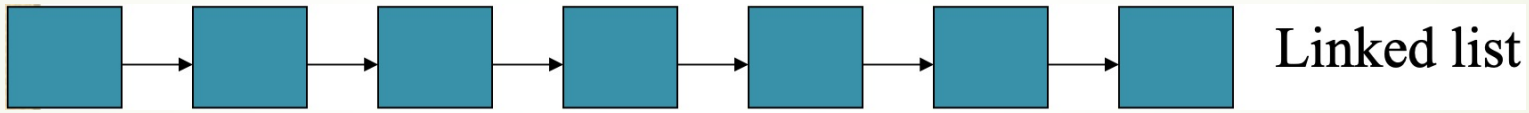
Data Structures

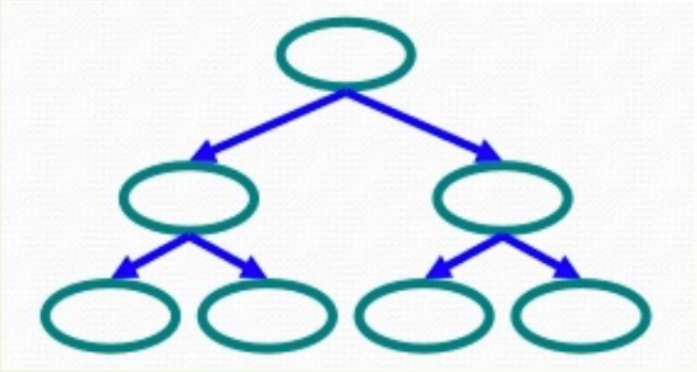
- A representation of data and the operations allowed on that data.
- A way to store and organize data in order to facilitate the access and modifications of data.
- The method of representing logical relationships between individual data elements related to the solution of a given problem.

Basic Data Structures

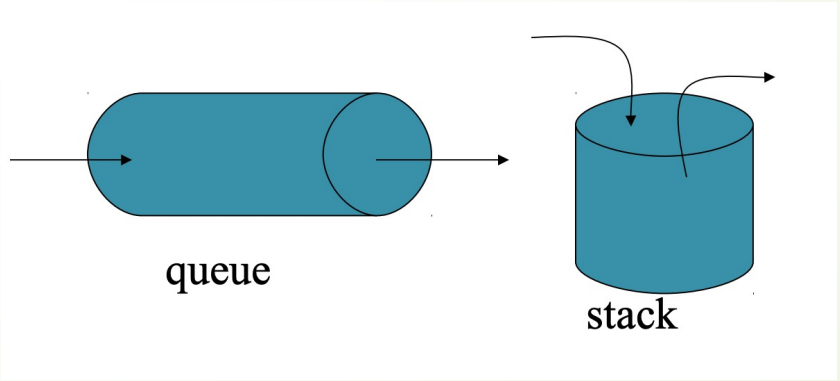


Basic Data Structures





Tree



Selection of Data Structure

The choice of a particular data model depends on two consideration:

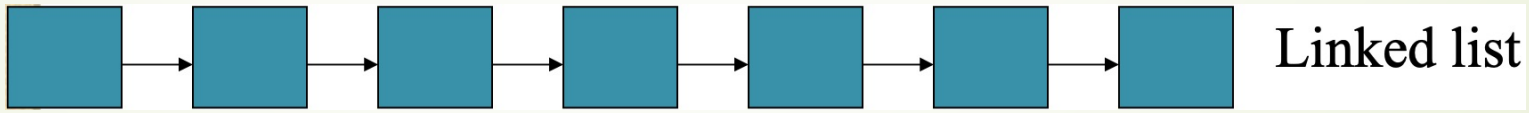
- It must be rich enough in structure to represent the relationship between data elements
- The structure should be simple enough that one can effectively process the data when necessary

List

A flexible structure, because a list can grow and shrink on demand.

- Elements of a list can be:
  - Inserted
  - Accessed
  - Deleted

  At any position



Stacks

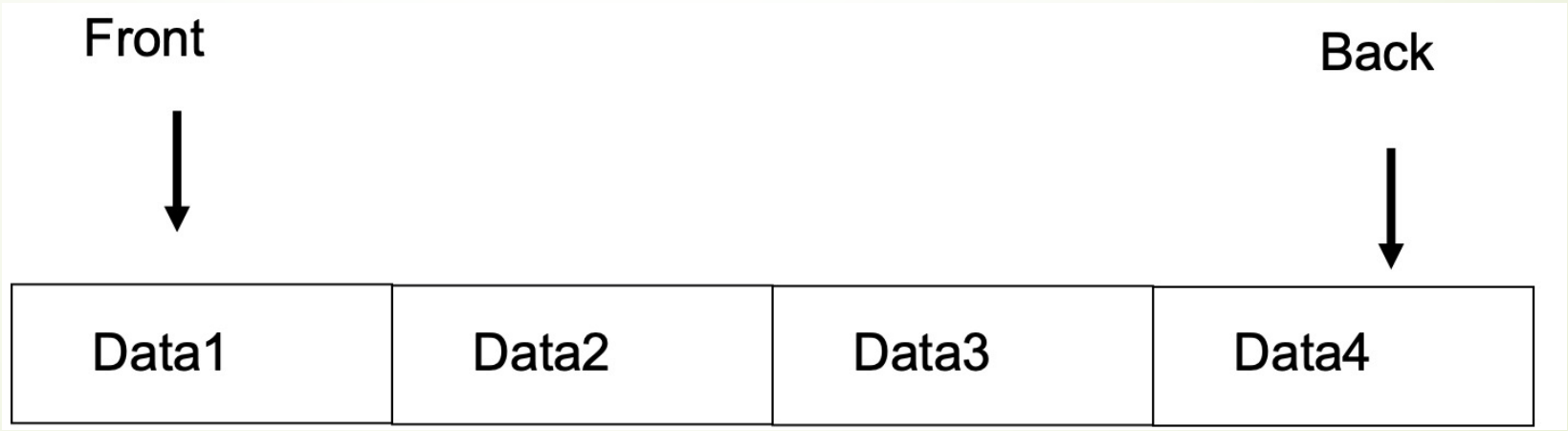
A data collection with access only to the last element inserted: **Last In First Out (LIFO)**

- Insert/push
- Remove/pop
- top
- make empty

Queues

A data collection with access only to the item that has been present the longest: **First In First Out (FIFO)** (or Last in last out)

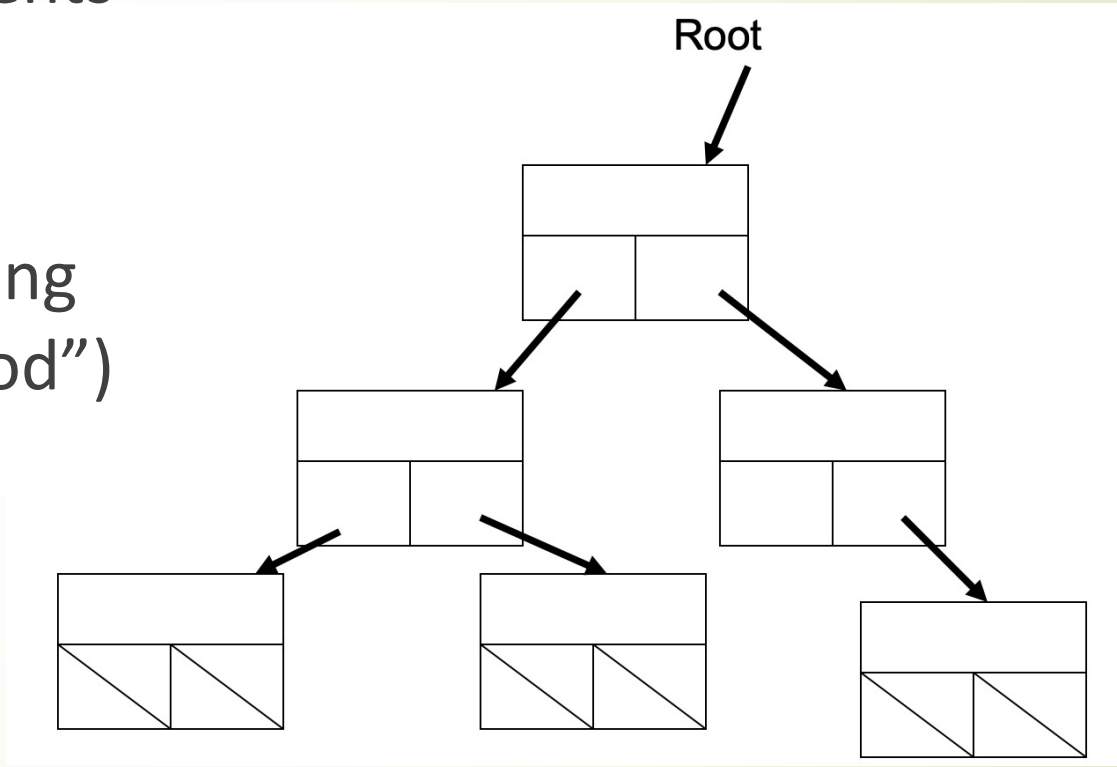
- enqueue, dequeue, front, back
- Priority queues (Heap) and Deque (double-ended queue)



Tree

A Tree is a collection of elements called nodes.

- One of the nodes is distinguished as a root, along with a relation (“parenthood”) that places a hierarchical structure on the nodes.



Supporting Materials on Data Structures

- UCLA CS 32
- Prof. Carey Nachenberg's CS 32 Slides:

http://careynachenberg.weebly.com/cs-slides.html

Stable Matching

(See separate slides)

Thank you!