CS 321: Introduction

Skills You'll Learn (1)

- ***Become a better programmer***
  - learn several fast algorithms for processing data and several useful data structures for organizing data that can be deployed directly in your own programs
- ***Sharpen your analytically skills***
  - get lots of practice describing and reasoning about algorithms and data structures
- ***Think in terms of algorithms***
  - After learning about algorithms it's hard not to see them everywhere, whether you're riding an elevator, watching a flock of birds, or walking in a city

Skills You'll Learn (2)

- ***Literacy with computer science's greatest hits***
  - No longer will you feel excluded at that computer science networking party….
- ***Ace your technical interviews***
  - Mastering the concepts in this class has enabled students to ace technical interview questions as the subject of this class is the core topic that companies often ask about

Programs = Algorithms + Data structures

Data Structures

- *A **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently.*
  - General data structure types include arrays, files, linked lists, stacks, queues, trees, graphs and so on.
- Depending on the organization of the elements, data structures are classified into two types:
  1. ***Linear data structures***: Elements are accessed in a sequential order but it is not compulsory to store all elements sequentially. Examples: Arrays, Linked Lists, Stacks and Queues.
  2. ***Nonlinear data structures***: Elements of this data structure are stored/accessed in a nonlinear order. Examples: Trees and Graphs

Data Types

- A *data type* in a programming language is a set of data with predefined values. *Examples of data types are: int, long, float, double, character, string, etc.*
- *At the top level, there are two types of data types:*
  - *System-defined data types (aka primitive data types)*
  - *User-defined data types*

More on Data Types

- By default, all primitive data types (int, float, etc.) support basic operations:
  - Addition and subtraction.
  - The system provides the implementations for the primitive data types.
- User-defined data types needs operations:
  - The implementation for these operations is done based on how we want to actually use them
  - User defined data types are defined along with their operations.

Abstract Data Types (1)

- An Abstract Data Type (ADT) is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.
- An Abstract Data Type is a data type, where only the behavior is defined but not the implementation.
- A Concrete Data Type contains an implementation of an ADT. We often refer to a concrete data type as simply a data structure.

Abstract Data Types (2)

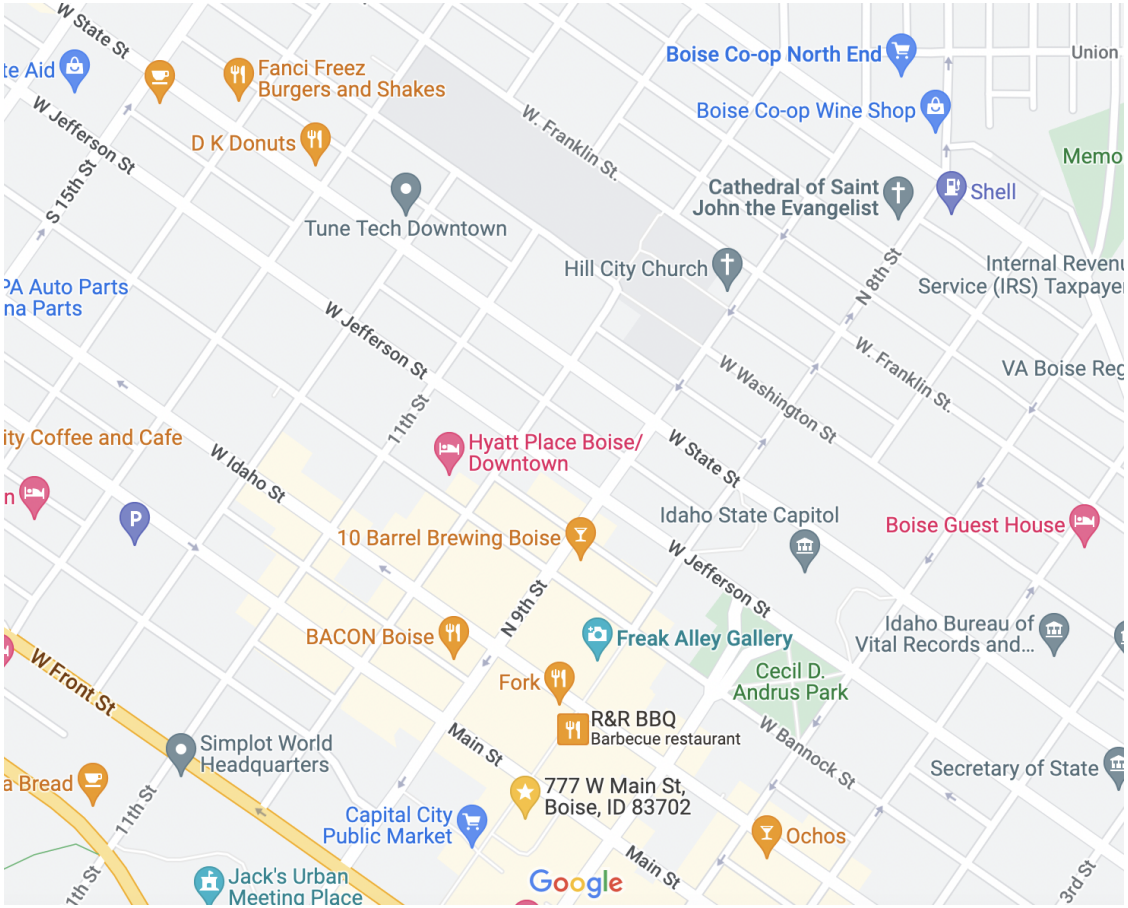
- An ADT consists of two parts:
  1. Declaration of data
  2. Declaration of operations
- *Commonly used ADTs include:*
  - Lists, Stacks, Queues, Priority Queues, Dictionaries, Sets, Maps, Trees, Graphs, and many others.
- *Example: common operations for a Stack ADT are:*
  - creating the stack, push an element onto the stack, pop an element from the stack, finding the current top of the stack, finding the number of elements in the stack, etc.

In-class Activity 1 (1)

- **Cities**: What is the data? What type of data structure is used?

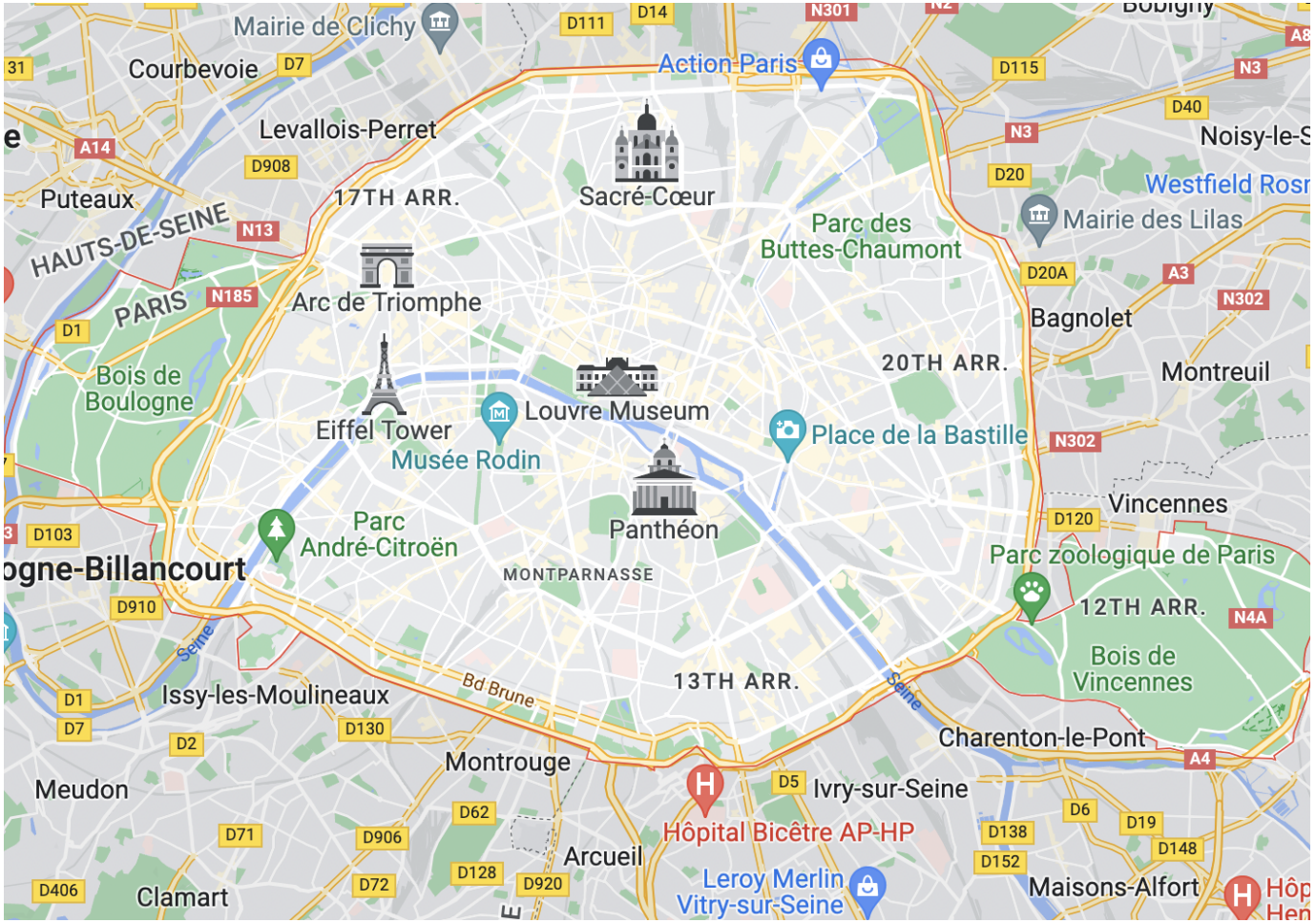
In-class Activity 1 (2)

- **Cities**:



In-class Activity 1 (3)

- **Cities**:



In-class Activity 1 (4)

- **Forests**:

In-class Activity 2

- How to organize your clothes in a closet?
- Assume each piece of clothing has a unique tag (like a number).
- How to structure your closet to make the access the fastest possible?
- What if the space was unlimited?

In-class Activity 3

- Trees: What is the data? What are the operations?

In-class Activity 4

- Where is my bushel of corn? US Freight Railroad Map

- Graphs: What is the data? What are the operations?

Efficiency

- An operation is said to be efficient if it solves the problem within its resource constraints.
  - Time
  - Space

- The cost of an operation is the amount of resources that its execution consumes.

Selecting a Data Structure

1. Analyze the problem to determine the resource constraints a solution must meet.
2. Determine the basic operations that must be supported. Quantify the resource constraints for each operation.
3. Select the data structure that best meets these requirements.

Some Questions to Ask

- Are all data inserted into the data structure at the beginning, or are insertions interspersed with other operations?
- Can data be deleted?
- Are all data processed in some well-defined order, or is random access allowed?

Data Structure Philosophy

Each data structure has costs and benefits.

Rarely is one data structure better than another in all situations.

A data structure requires:

- space for each data item it stores,
- time to perform each basic operation,
- programming effort.

Data Structure Skills

1. Learn the commonly used data structures.
   - These form a programmer's basic data structure “toolkit."

2. Understand how to measure the cost of a data structure.
   - These techniques also allow you to judge the merits of new data structures that you or others might invent or encounter.

What is an Algorithm?

- An algorithm is a step-by-step unambiguous sequence of instructions to solve a well-defined problem.
- Two main criteria for evaluating an algorithm:
  - ***correctness*** -- does the algorithm give solution to the problem in a finite number of steps?
  - ***efficiency*** -- how much resources (in terms of time and memory) does it take to execute?

Why the Analysis of Algorithms?

- In computer science, multiple algorithms are available for solving the same problem
  - Example: the sorting problem has many algorithms, like insertion sort, selection sort, merge sort, quick sort and many more.
- *Algorithm analysis helps us to determine which algorithm is most efficient in terms of resource usage such as time, space (or energy and others).*

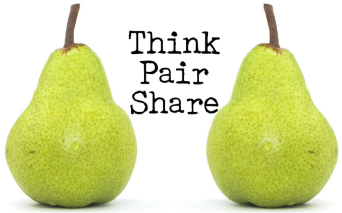
Goal of the Analysis of Algorithms

- The goal of the analysis of algorithms is to compare algorithms mainly in terms of ***running time*** but also in terms of other factors (e.g., memory, developer effort, etc.)

How to Compare Algorithms?

- *Execution times?*
  - Not a good measure as execution times are specific to a particular computer.
- *Number of statements executed*?
  - Not a good measure, since the number of statements varies with the programming language as well as the style of the individual programmer.
- *Ideal solution?*
  - We express the running time of a given algorithm as a function of the input size n (i.e., f(n)) and compare these different functions corresponding to running times.
  - This kind of comparison is independent of machine time, programming style, etc.

In-class Activity 4 (1)



- How to compute 2^n efficiently?
  - Naive (brute-force) approach: 2 x 2 x . . . x 2
  - Recursive doubling:

    $2 \times 2 = 2^2$

    $2^2 \times 2^2 = 2^4$

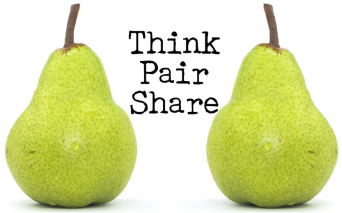
    $2^4 \times 2^4 = 2^8$

    $2^8 \times 2^8 = 2^{16}$

    . . .

  - What if *n* isn't a power of two? For example, what if *n* is 23.

In-class Activity 1-2



- See example code in class code repository folder examples/power: BigPower.java and BigPowerTest.java

- The recursive doubling implementation runs about 100 times faster than the naive approach!

Fundamental Computing Issue

The choice of the data structure and algorithm can make the difference between a program running in a few seconds or many days or even months or years or centuries!

Thought for the semester!

The society that scorns excellence in plumbing because plumbing is a humble activity and tolerates shoddiness in philosophy because it is an exalted activity will have neither good plumbing nor good philosophy. **Neither its pipes nor its theories will hold water**."

--John Gardner, "Excellence"

Further Reading

- *Hello World: Being a Human in the Age of Algorithms*, Hannah Fry
- *Automate This: How Algorithms Came to Rule the World*, Christopher Steiner