Chapter 1: Data Structures and Algorithms

Comp Sci 1575 Data Structures

Missouri S&T | Computer Science

Programming

If you give someone a program, you will frustrate them for a day; if you teach them to program, you will frustrate them for a lifetime.

Outline

Goals for today include

- Providing inspiration and a perspective to remember as you're solving problems and coding solutions throughout the coming weeks.
- Some points today are by analogy, so it doesn't have to feel concrete yet; next week it will.
- Remember to stop me if you have questions

Outline

What is to come in this course

- Present the commonly used data structures.
- Discuss tradeoffs and reinforce the concept that there are costs and benefits associated with every data structure.
- Measure the effectiveness of a data structure or algorithm. The techniques presented also allow you to judge the merits of new data structures that you or others might invent.

Outline

Typical job of computing

- Representing information is fundamental to computer science.
- The primary purpose of most computer programs is not to perform calculations, but to store and retrieve information, usually as fast as possible.
- Data structures and the algorithms that manipulate them is at the heart of computer science.
- Data structures helps you to understand how to structure information to support efficient processing.
- Using a good data structure can make the difference between a program running in a few seconds and one requiring many days, or completing at all.

Outline

Outline

What is efficiency

- A point in the range of the ratio of resources to return
- Examples of resource constraints include the total space available to store the data, and the time allowed to perform computation.

Outline

What is a data structure?

- Commonly, an organization or structuring for a collection of data items.
- Most generally, any data representation **and its associated operations.**

Basic definitions

- **Type** defined as a collection of values: e.g., bool, int, char
- **Aggregate/composite types**: e.g., user-defined structs, classes
- **Data item** is a member of a type
- **Data type** includes a type together with a set of its operations, e.g.,
  integer, char, and bool are data types with operations associated with them

Outline

Abstract data type (ADT)

- is an realization of a data type as a set of typed objects together with a set of operations.
- employs objects used to represent collections of objects, such as: sets, sequences, trees, and graphs, and their operations
- does not specify how the data type is implemented
  e.g., list is one of the most universal user interfaces with many different types of implementation

Outline

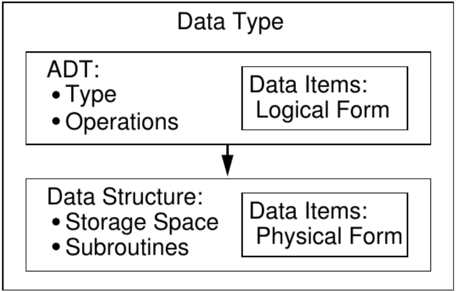
Data structures

- Data structure is an implementation of an ADT.
- In OOP C ++ , a data structure can take the form of a class with member data variables and functions
- Variables to store data items defined as data members
- Operations for the ADT are implemented by member functions or methods

Outline

Outline

Conceptual versus physical



- Logical concept of a data type versus physical implementation in a program.
- Data types have both a logical and a physical form.
- ADT defines the logical form of a data type.
- Implemented ADT (data structures) are the physical form of the data type.
- Using an ADT elsewhere in your program relies on the type's logical form.

Outline

Abstraction and modularity

- ADT is an invariant abstraction, which can be implemented in many ways.
- ADT encourages multiple layers of abstraction
- Decompose the problem into small modules, and use information hiding (abstraction)
- In design, there are trade-offs for where you embed information or functions in:
  -general modules which can process many objects, or
  -local modules themselves.

Kahoot!

Outline

Problem solving

Sometimes conflicting goals in computational problem solving:

1. Designs that are easy to understand, code, and debug.
2. Designs that are efficient for computer resources (this class!)

Occasionally an elegant solution captures both (we'll cover many).

Outline

Problems, algorithms, and programs

- **Problems** can be considered functions which map inputs to outputs
- Problems also include resource constraints.
- **Algorithms** are a correct recipe of finite length for solving a problem with concrete, unambiguous steps, which must terminate for all inputs.
- Algorithms must provide sufficient detail that they can be converted into a program when needed
- **Programs** are instantiations of algorithms in a programming language.

Does every problem have an algorithm?
Does every algorithm have a C++ program?
Is every program an algorithm?

Problems

- Problem as a function is a matching between inputs (the domain) and outputs (the range).
- Input to a function might be a single value or a collection of information.
- Values making up an input are called the parameters of the function.
- Selection of values for the parameters is called an instance of the problem; for example, the input parameter to a sorting function might be an array of integers with a given size and specific values for each position in the array
- Different instances might generate the same output, but any problem instance must always result in the same output every time the function is computed using that particular input.

Algorithms

- An algorithm is a method or a process followed to solve a problem.
- If the problem is viewed as a function, then an algorithm is an implementation for the function that transforms an input to the corresponding output.
- A problem can be solved by many different algorithms.
- Requirements
  - It must be correct, computing the desired function, converting each input to the correct output.
  - It is composed of a series of concrete steps; understood and doable by the person or machine that must perform the algorithm, in a finite amount of time.
  - There can be no ambiguity as to which step will be performed next.
  - It must be composed of a finite number of steps.
  - It must terminate.

Programs

- Computer program as an instance, or concrete representation, of an algorithm in some programming language.

Outline

Problem solving versus programming

Ordering your design work:

1. Specify input and output
2. Design data structures and algorithms
3. Translate into C++
4. Test and debug

Unpacked further next slide:

Problem solving versus programming

1. Formalize the problem:
   Abstract all but essential characteristics
   Generate a mathematical model
2. Create a high-level algorithm based on the model:
   Describe it using clear English.
   Decide on an ADT
3. Refine your pseudocode algorithm:
   Determine the most important operations
   Design the needed Data Structures accordingly.
4. Only lastly, implement the data structure

Kahoot!

Outline

Outline

How to select data structure?

1. Which operations must be supported?
   Inserting a data item,
   deleting a data item,
   finding an data item?
2. Quantify the resource constraints for each operation.
3. Select the data structure that best meets these requirements.

What types of insert are there?
What types of search are there?
Sort versus search tradeoff?

Outline

Consider when selecting a data structure

General features:

1. Data features and the operations to be performed on them
2. Representation for those data
3. Implementation of that representation.
4. Resource constraints (time, space) for important operations

Unpacked on the next slide

Example considerations

- Data items inserted into the data structure at initialization (simpler), or ongoing (more complicated)?
- Can data items be deleted? (more complicated)
- Items processed/accessed in defined order (simpler), or random access (more complicated)?
- Is search for specific data items allowed?
- Is search exact or range based?

Outline

Resource constraints

- Space for each data item stored
- Time to perform a single basic operation
- Programming effort

Costs and Benefits

- Each data structure has associated costs and benefits, and some data structures which perform badly in some situations excel in others.

Outline

ADTs in C++

- Building blocks of data structures such as structs, classes, arrays, and pointers enable compound/aggregate/composite types.
- C++ "class" can implement ADTs, hiding unnecessary details.
- Objects are instances of a class, stored during a particular execution
- You can perform operations on the data structure by calling the appropriate method.
- If implementation details need to be changed, just modify member methods, which doesn't have to interfere with the rest of the program