Data Structures

Introduction and Motivation

Pooja Singh
<pooja.singh@snu.edu.in>

Assistant Professor
Department of Computer Science and Engineering
School of Engineering

January, 2025

Shiv Nadar University

Outline

1. Introduction

Outline

Outline

Outline

1. **Introduction**
2. **Abstract Data Type**
3. **Collections**
4. **Structures in Modeling**

Outline

Outline

Outline

Introduction

- "Data Structures" is the study of representation of data and operations on data — most significantly on a collection of data
- In this course, we will explore some of the most important collections such as lists, trees, graphs, and sets
- Abstract Data Types (ADTs) provide mathematical abstractions of such structures
- These structures can easily be decomposed or composed — properties can be proved using structural induction — operations can be implemented using recursion (divide and conquer)
- Structures such as graphs are extremely useful as modeling tools
- They are useful in analysis and design too

Shiv Nadar University

- **Programming languages provide support for basic data types and operations on them. For example, C provides 'int', 'long', 'float', 'double', 'char', etc.**

Data and Data Types

- **Programming languages provide support for basic data types and operations on them. For example, C provides 'int', 'long', 'float', 'double', 'char', etc.**
- **Do we care how a floating point number is actually represented?**

Data and Data Types

- **Programming languages provide support for basic data types and operations on them. For example, C provides 'int', 'long', 'float', 'double', 'char', etc.**
- **Do we care how a floating point number is actually represented?**
- **We use <math.h> to perform several operations on floating point numbers. But, do we know, for example, how square root of a floating point number is computed?**

Data and Data Types

- Programming languages provide support for basic data types and operations on them. For example, C provides 'int', 'long', 'float', 'double', 'char', etc.
- Do we care how a floating point number is actually represented?
- We use <math.h> to perform several operations on floating point numbers. But, do we know, for example, how square root of a floating point number is computed?
- Libraries, such as <math.h> hide the data representation and implementation details — provide an abstraction and encapsulation of data + operations

Data and Data Types

- **Programming languages provide support for basic data types and operations on them. For example, C provides 'int', 'long', 'float', 'double', 'char', etc.**
- **Do we care how a floating point number is actually represented?**
- **We use <math.h> to perform several operations on floating point numbers. But, do we know, for example, how square root of a floating point number is computed?**
- **Libraries, such as <math.h> hide the data representation and implementation details — provide an abstraction and encapsulation of data + operations**
- **Will our application be affected if <math.h> is modified to use a different algorithm to compute square root?**

Building additional data types

- **When we develop applications in C, are we limited to the data types available in C?**

Building additional data types

- When we develop applications in C, are we limited to the data types available in C?
- For example, can we declare variables such as
  - Book b1;
  - Person p1;
  - Date d1, d2;
  - Rational r1, r2, r3;
  - Complex c1, c2;

Building additional data types

- When we develop applications in C, are we limited to the data types available in C?
- For example, can we declare variables such as
  - Book b1;
  - Person p1;
  - Date d1, d2;
  - Rational r1, r2, r3;
  - Complex c1, c2;
- And perform operations on them?
  - Rational r3 = r1 + r2;
  - get_aadhaar(p1);
  - if (isbn(b1) == mykey) return b1;

SHIV NADAR UNIVERSITY

Outline

- **We may use the structure and union aggregators in C to provide our own representations for such data**

```
struct person{
    char *fname;
    char *lname;
    Date dob;
    char *aadhaar;
    . . .
}
```

- **Suppose we develop applications using this structure. How much of the code needs to be changed when this structure is modified?**

Encapsulation of data representation and operations

- **We may develop a library comprising data representation and operations on them together**
- **For example, we may develop a module <Rational.h> to provide a representation for rational numbers, define and implement operations on them**

Encapsulation of data representation and operations

- **We may develop a library comprising data representation and operations on them together**
- **For example, we may develop a module <Rational.h> to provide a representation for rational numbers, define and implement operations on them**

```
typedef struct rational {
    int numerator ;
    int denominator ;
} Rational;
```

Shiv Nadar University

Encapsulation of data representation and operations

- **We may develop a library comprising data representation and operations on them together**
- **For example, we may develop a module <Rational.h> to provide a representation for rational numbers, define and implement operations on them**

```
typedef struct rational{
      int numerator ;
      int denominator ;
} Rational;

Rational Radd (Rational r1, Rational r2);
Rational Rsub (Rational r1, Rational r2);
Rational Rmult(Rational r1, Rational r2);
Rational Rdivide(Rational r1, Rational r2);
Rational Rinverse(Rational r1);
Rational Rreduce (Rational r1) ;
```

Shiv Nadar University

Representation Efficiency

- **Note that the interface includes a function to reduce a rational number**

$$\frac{15}{51} = \frac{5}{17}$$

- **The idea here is that all the operations on rational numbers should internally use this reduce function to keep the internal representation in a compact form**
- **Otherwise, the numbers may eventually go out of range leading to internal representation error**

- **Functions declared in <Rational.h> may be implemented in a separate library module “Rational.c”**

```
. . .
#include "Rational.h"
. . .
Rational Rinverse(Rational r1)
{
   Rational r ;

   r.numerator = r1.denominator;
   r.denominator = r1.numerator;
   return Rreduce ( r );
}
```

Shiv Nadar University

Mathematical abstraction

- **We can now develop applications and use <Rational.h> to provide mathematical abstractions of rational numbers, and link with "Rational.c"**

```
. . .
#include "Rational.h"
. . .
Rational r1, r2, r 3 ;
. . .
r1 = Radd ( r2, Rinverse(r3)) ;
. . .
```

- **The intention here is that we don't need to modify our application even if implementations of operations are modified in <Rational.c> (as long as the interface is not modified)**

Shiv Nadar University

Abstract Data Types

- **Abstract Data Type (ADT) — way of representing data along with effective operations on them**

Abstract Data Types

- **Abstract Data Type (ADT) — way of representing data along with effective operations on them**
- **Encapsulation — data + operations are designed together and available as a single module (or class)**
- **Abstraction — Exposes only the interface (definitions of operations) and application developers need not worry about how they are implemented**

Abstract Data Types

- **Abstract Data Type (ADT) — way of representing data along with effective operations on them**
- **Encapsulation — data + operations are designed together and available as a single module (or class)**
- **Abstraction — Exposes only the interface (definitions of operations) and application developers need not worry about how they are implemented**
- **"Data Structures" is the study of ADTs and effective implementations of them in a chosen programming language**
- **Efficiency of representation and operations is extremely important!**

Outline

Collections and Dynamic Structures

- **Sets**
- **Tuples**
- **Arrays**
- **Sequences (also known as lists)**
- **Trees**
- **Graphs**

Sets and Tuples

- **Set is an unordered collection of objects**
  - **Usually homogeneous**
  - **No order among the objects** – {a, b} = {b, a}
  - **No multiple memberships** – {a, a, a} = {a, a} = {a}
  - **Programming languages like Python have direct support for sets**

Shiv Nadar University

Sets and Tuples

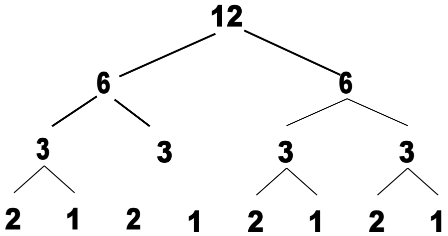
- **Set is an unordered collection of objects**
  - **Usually homogeneous**
  - **No order among the objects** – {a, b} = {b, a}
  - **No multiple memberships** – {a, a, a} = {a, a} = {a}
  - **Programming languages like Python have direct support for sets**
- **Tuple is an ordered but static collection of objects**
  - **Each member of** $S_1 \times S_2 \times \cdots \times S_n$ **is a tuple**
  - **For example, the tuple** $(1, 3.14)$ **is of type** $Z \times R$ **(in terms of C, it is something like (int, float))**
  - **Can be heterogeneous**
  - **Can not insert or delete dynamically**
  - **Programming languages like Python support tuples**

Shiv Nadar University

Sequences

- **Sequences are ordered collection of objects — $(1, 5, 9)$,** $(InsertCard, EnterPin, EnterAmount, CollectMoney, TakeOutCard)$
- **Unlike arrays, sequences are dynamic — objects can be inserted and deleted**
- **There is a concept of position (similar to array index) and each position has a next (except for the last) and a previous (except for the first) positions**
- **Sequences are usually traversed starting from the first object and then following next positions until the last object is reached**
- **Sequences can be decomposed into sub-sequences**
- **Can also be decomposed into an object and the rest of the sequence — for example, $(1, 5, 9)$ can be decomposed into 1, and $(5, 9)$**

- A tree is a collection of objects with special “parent-child” relation between objects
- There is a special node, that does not have a parent, called “root” of the tree
- Nodes without any children are called as “leafs” and others are called as “internal” nodes
- A tree may be decomposed into a node and sub-trees
- Note: A sequence is a special kind of tree where each internal node has exactly one child



Graphs

- **Graphs represent arbitrary relations between objects**
- **Nodes represent objects and edges represent relations (in general, referred to as neighborhood relation) between objects**
- **Graphs may be decomposed into sub-graphs**
- **Note: Tree is a special kind of graph where each node can have only one parent**

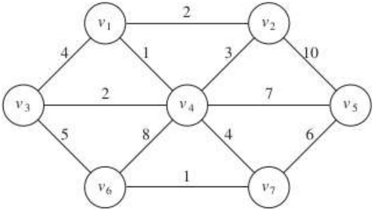


Figure: Adopted from book by Mark Allen Weiss

Outline

Google Page Ranking

- **Websites are represented by nodes and anchors are represented by edges**
- **Problem is to find "weights" for all the nodes**

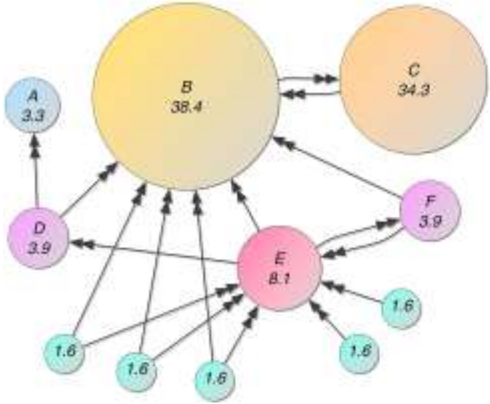


Figure: Adopted from Wikipedia

Shiv Nadar University

Google Maps

- **Nodes represent locations and edges represent "cost" of driving from one location to the other**

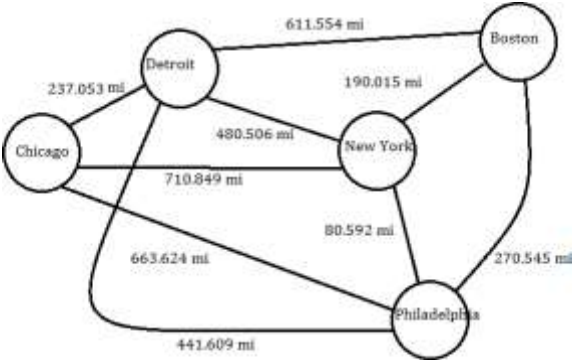


Figure: Adopted from blogs.cornell.edu

Protocol Design

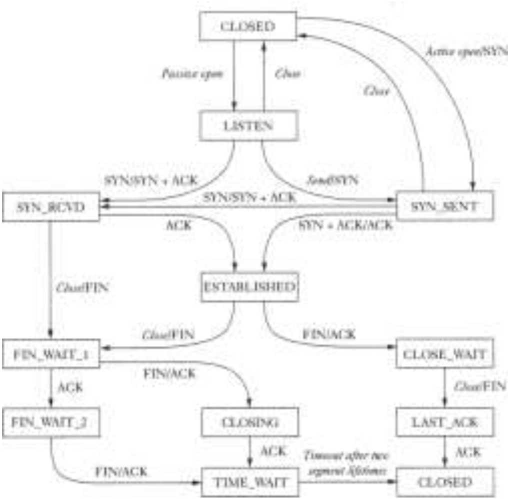


Figure: Adopted from ssfnet.org

Laying OFC in the campus

- Nodes represent buildings and edges represent cost of laying OFC
- Problem is to decide on choosing minimal number of edges (OFC routes) so that cost is minimal and all the buildings are covered

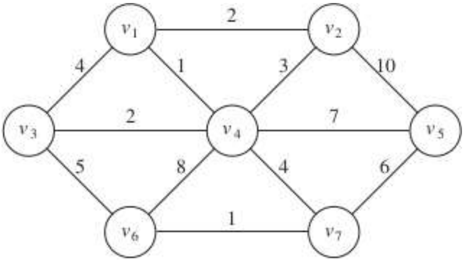


Figure: Adopted from book by Mark Allen Weiss

Minimum Spanning Tree

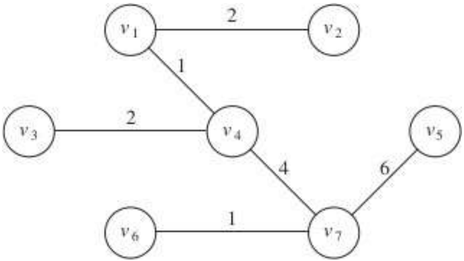


Figure: **Adopted from book by Mark Allen Weiss**

Outline

Structures in Analysis and Design

- Consider a collection of sensor data values — for the sake of simplicity, we will assume an array of temperature values
- Problem is to find both the minimum and maximum values

Structures in Analysis and Design

- Consider a collection of sensor data values — for the sake of simplicity, we will assume an array of temperature values
- Problem is to find both the minimum and maximum values

```
min = a[0]; max = a[0]
for (int i=1; i<n; i++){
    if ( a[i] < min ) min = a[i];
    if ( a[i] > max ) max = a[i];
}
```

Shiv Nadar University

- Consider a collection of sensor data values — for the sake of simplicity, we will assume an array of temperature values
- Problem is to find both the minimum and maximum values

```
min = a[0]; max = a [0]
for (int i=1; i <n; i++) {
    if ( a[i]< min ) min = a[i];
    if ( a[i]> max ) max = a[i ];
}
```

- How many comparisons are made?

- **Consider a collection of sensor data values — for the sake of simplicity, we will assume an array of temperature values**
- **Problem is to find both the minimum and maximum values**

```
min = a[0]; max = a [0]
for (int i=1; i<n; i++) {
    if ( a[i] < min ) min = a[i];
    if ( a[i] > max ) max = a[i ];
}
```

- **How many comparisons are made?** $2n - 2$

Divide and Conquer

- **Divide the collection of n objects into two halves — (Divide)**
- **Find (max1, min1) from the first half ($\frac{n}{2}$ objects) and (max2, min2) from the second half ($\frac{n}{2}$ objects)— (Conquer)**
- **max = maximum(max1, max2); min = minimum(min1, min2); — (Merge)**

SHIV NADAR UNIVERSITY

Divide and Conquer

- **Divide the collection of n objects into two halves — (Divide)**
- **Find (max1, min1) from the first half ($\frac{n}{2}$ objects) and (max2, min2) from the second half ($\frac{n}{2}$objects)— (Conquer)**
- **max = maximum(max1, max2); min = minimum(min1, min2); — (Merge)**
- **How do we find (max1, min1) from the first half?**

Shiv Nadar University

- **Divide the collection of n objects into two halves — (Divide)**
- **Find (max1, min1) from the first half ($\frac{n}{2}$ objects) and (max2, min2) from the second half ($\frac{n}{2}$ objects)—(Conquer)**
- **max = maximum(max1, max2); min = minimum(min1, min2);— (Merge)**
- **How do we find (max1, min1) from the first half?**
- **Recursively use the same strategy!**

- **Divide the collection of n objects into two halves — (Divide)**
- **Find (max1, min1) from the first half ($\frac{n}{2}$ objects) and (max2, min2) from the second half ($\frac{n}{2}$ objects)— (Conquer)**
- **max = maximum(max1, max2); min = minimum(min1, min2); — (Merge)**
- **How do we find (max1, min1) from the first half?**
- **Recursively use the same strategy!**
- **When do we stop?**

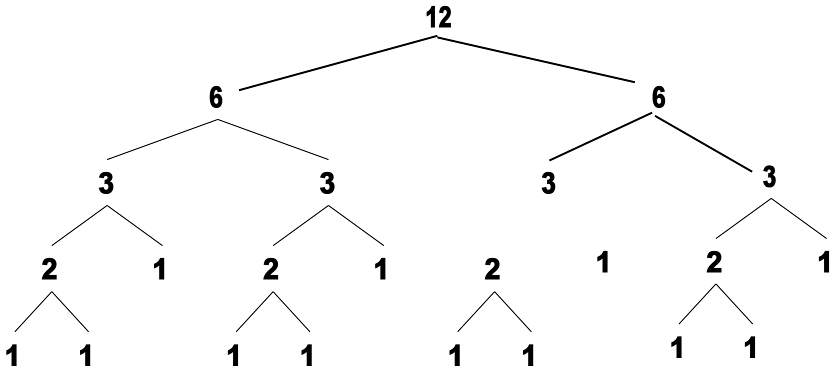
- **Divide the collection of n objects into two halves — (Divide)**
- **Find (max1, min1) from the first half ($\frac{n}{2}$ objects) and (max2, min2) from the second half ($\frac{n}{2}$ objects)— (Conquer)**
- **max = maximum(max1, max2); min = minimum(min1, min2); — (Merge)**
- **How do we find (max1, min1) from the first half?**
- **Recursively use the same strategy!**
- **When do we stop?**
- **if (n == 1) max = min = a[0]; (the only object in the collection)**

How many comparisons?

- **How many comparisons are made in our recursive function? (when we start with n objects)**

Shiv Nadar University

- **How many comparisons are made in our recursive function? (when we start with *n* objects)**



How many comparisons?

- Decomposition of the array can be visualized as a binary tree
- All the leaf nodes correspond to the base cases — no comparisons made
- All the other nodes (internal nodes) correspond to the recursive cases — two comparisons made
- How many leaf nodes will be there when we start with n objects?

SHIV NADAR UNIVERSITY

How many comparisons?

- **Decomposition of the array can be visualized as a binary tree**
- **All the leaf nodes correspond to the base cases — no comparisons made**
- **All the other nodes (internal nodes) correspond to the recursive cases — two comparisons made**
- **How many leaf nodes will be there when we start with n objects?** n
- **How many internal nodes will be there?**

Shiv Nadar University

How many comparisons?

- **Decomposition of the array can be visualized as a binary tree**
- **All the leaf nodes correspond to the base cases — no comparisons made**
- **All the other nodes (internal nodes) correspond to the recursive cases — two comparisons made**
- **How many leaf nodes will be there when we start with n objects?** n
- **How many internal nodes will be there?** $n - 1$ **(this can be proved using the properties of binary trees!)**
- **So, how many comparisons are made?**

How many comparisons?

- **Decomposition of the array can be visualized as a binary tree**
- **All the leaf nodes correspond to the base cases — no comparisons made**
- **All the other nodes (internal nodes) correspond to the recursive cases — two comparisons made**
- **How many leaf nodes will be there when we start with n objects? n**
- **How many internal nodes will be there? $n - 1$ (this can be proved using the properties of binary trees!)**
- **So, how many comparisons are made? $2n - 2$**

Recurrence Equations

$$T(1) = 0$$

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 2$$

$$T(1) = 0$$

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 2$$

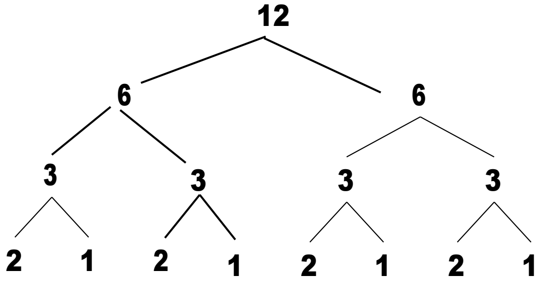
Number of comparisons made: $T(n) = 2n - 2$

Base case?

- **When n = 2, we don't need two comparisons!**
- **when (n == 2) if (a[0] < a[1]) { min = a[0]; max = a[1]; } else { min = a[1]; max = a[0]; }**

Base case?

- When $n = 2$, we don't need two comparisons!
- when (n == 2) if (a[0] < a[1]) { min = a[0]; max = a[1]; } else { min = a[1]; max = a[0]; }



Shiv Nadar University

How many comparisons?

$$T(1) = 0$$

$$T(2) = 1$$

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 2$$

How many comparisons?

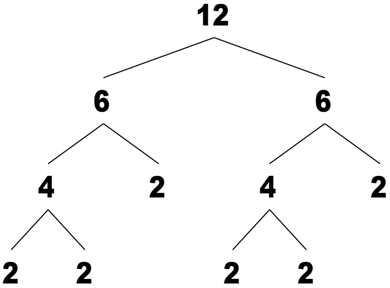
$$T(1) = 0$$

$$T(2) = 1$$

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 2$$

Number of comparisons made: $T(n) = \frac{5}{3}n - 2$

Alternate Decomposition



How many comparisons?

$$T(1) = 0$$

$$T(2) = 1$$

$$T(4k) = T(2k) + T(2k) + 2$$

$$T(4k+1) = T(2k) + T(2k+1) + 2$$

$$T(4k+2) = T(2k) + T(2k+2) + 2$$

$$T(4k+3) = T(2k+1) + T(2k+2) + 2$$

How many comparisons?

$$T(1) = 0$$

$$T(2) = 1$$

$$T(4k) = T(2k) + T(2k) + 2$$

$$T(4k+1) = T(2k) + T(2k+1) + 2$$

$$T(4k+2) = T(2k) + T(2k+2) + 2$$

$$T(4k+3) = T(2k+1) + T(2k+2) + 2$$

Number of comparisons made: $T(n) = 3n/2 - 2$

Outline

Summary

- Programming languages provide simple data representations and operations on them
- For developing various applications, we may need to build our own data structures
- Abstract Data Types — encapsulation, mathematical abstractions, modularity, reusability
- Abstractions for collections of data — Sequences, Trees, Graphs
- Structures in modeling
- Structures in analysis and design
- Efficient implementation of operations is a must!

What next?

- We will study the notations used to express time and space complexities of an algorithm
- We will learn how to solve simple recurrences to analyze recursive algorithms
- We will start learning about the linear collection structures
- We will eventually learn about non-linear structures namely trees and graphs
- We will implement most of the data structures and algorithms during lab sessions