ECE430.217 Data Structures

Containers, Relations, and Abstract Data Types

Weiss Book Chapter 3.1

Byoungyoung Lee
https://compsec.snu.ac.kr
byoungyoung@snu.ac.kr

Lecture slides were prepared based on Douglas W. Harder's notes (**dwharder@alumni.uwaterloo.ca).**

Outline

This topic will describe

- The storage of objects in *containers*
- We will focus on linear orderings:
  - Implicitly defined linear orderings (sorted lists)
  - Explicitly defined linear orderings
- We will summarize this information
- We will also look briefly at:
  - Hierarchical orderings
  - Partial orderings
  - Equivalence relations
  - Adjacency relations

Outline

Any form of information processing or communication requires that **data** must be **stored** in and **accessed** from either main or secondary **memory**

This topic will cover Abstract Data Types:

- Models of the storage and access of information

The next topic will cover data structures and algorithms:

- The concrete methods for organizing and accessing data in the computer

Containers

The most general Abstract Data Type (ADT) is that of a *container*

- The Container ADT

A container describes structures that store and give access to objects

The queries and operations of interest may be defined on:

- The container as an entity, or
- The objects stored within a container

Operations on a Container

The operations we may wish to perform on a container are:

- Create a new container
- Copy or destroy an existing container
- Empty a container
- Query how many objects are in a container
- Query what is the maximum number of objects a container can hold
- Given two containers:
  - Find the union (merge), or
  - Find the intersection

Operations on a Container

Many of these operations on containers are in the Standard Template Library

Constructor	`Container()`
Copy Constructor	`Container( Container const & )`
Destructor	`~Container()`
Empty it	`void clear()`
How many objects are in it?	`int size() const`
Is it empty?	`bool empty() const`
How many objects can it hold?	`int max_size() const`
Merge with another container	`void insert( Container const & )`

Check more: https://en.cppreference.com/w/cpp/container

Operations on Objects Stored in a Container

Given a container, we may wish to:

- Insert an object into a container
- Access or modify an object in a container
- Remove an object from the container
- Query if an object is in the container
  - If applicable, count how many copies of an object are in a container
- Iterate (step through) the objects in a container

Operations on Objects Stored in a Container

Many of these operations are also common to the Standard Template Library

Insert an object	`void insert( Type const & )`
Erase an object	`void erase( Type const & )`
Find or access an object	`iterator find( Type const & )`
Count the number of copies	`int count( Type const & )`
Iterate through the objects in a container	`iterator begin() const`

Simple and Associative Containers

We may split containers into two general classifications:

Simple Containers

Containers that store individual objects

Associative Containers

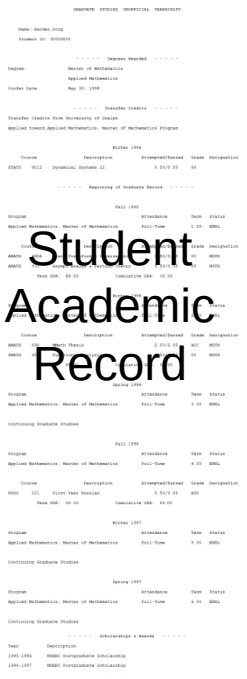
Containers that store keys but also store records associated with those keys

Temperature values

Circular Array

SNU ID Number

mysnu Server



Student Academic Record

Unique or Duplicate Objects

Another design requirement may be to either:

- Require that all objects in the container are unique, or
- Allow duplicate objects

Many of the containers we will look at will assume uniqueness unless otherwise stated

- Dealing with duplicate objects is often just additional, and sometimes subtle, code

Standard Template Library

We will begin by introducing four containers from the C++ Standard Template Library (STL)

Unique Objects/Keys	Duplicate Objects/Keys
`set<Type>`	`multiset<Type>`
`map<Key_type, Type>`	`multimap<Key_type, Type>`

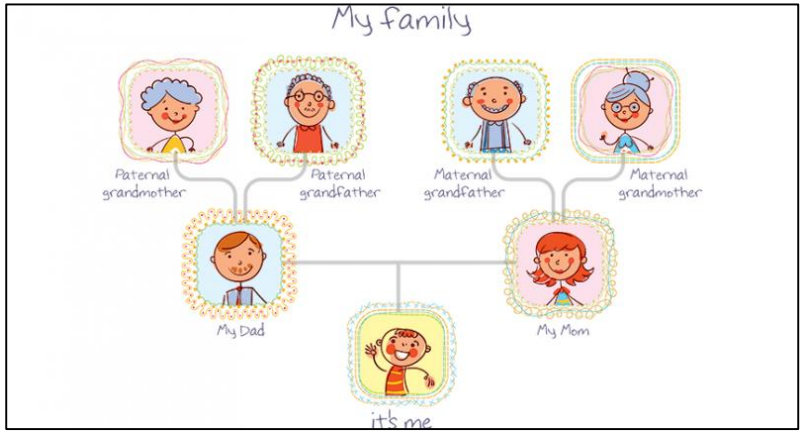
The STL set Container

```
#include <iostream>
#include <set>

int main() {
     std::set<int> ints;

    for ( int i = -100; i <= 100; ++i ) {
        ints.insert( i*i );         // Ignores duplicates:  (-3)*(-3) == 3*3
    }                               // Inserts 101 values: 0, 1, 4, 9, ..., 10000

    std::cout << "Size of 'is': " << ints.size() << std::endl;  // Prints 101

    ints.erase( 50 );                  // Does nothing
    ints.erase( 9 );                   // Removes 9
    std::cout << "Size of 'is': " << ints.size() << std::endl;  // Prints 100

    return 0;
}
```

Operations with Relationships

We may want to store not only objects, but **relationships between the objects**

- Consequently, we may have additional operations based on the relationships
- Consider a genealogical database
  - We don't only want to store the people, but we want to also make queries about the relationships between the people



https://www.sciencenewsforstudents.org/wp-content/uploads/2019/11/860_SS_Genealogy-800x428.png

Operations w/o Relationships

If we are **not storing relationships**, there is a data structure that is always the same speed no matter how many objects are stored

- **A *hash table*** roughly takes the same time to find an object whether there are 10 or one billion objects in the container
- Example:
  - Assume a department has 12 staffs that are frequently changing
  - Rather than having a mailbox for each person, have 24 mailboxes and place mail into the *bin* corresponding to the person's last name

A	**E**	**I**	**M**	**R**	**V**
B	**F**	**J**	**M**	**S**	**W**
C	**G**	**K**	**N**	**T**	**XY**
D	**H**	**L**	**PQ**	**U**	**Z**

  - This works fine as long as there is not too much collisions

Relationships

We will look at four relationships:

- Linear orderings

- Hierarchical orderings

- Partial orderings

- Adjacency relations

Relationships

Relationships are often Boolean-valued binary operations

Example: given two integers:

- Are they equal? $x = y$
- Is one less than the other? $x < y$
- Does the first divide the second? $x \mid y$
- Do they have the same remainder modulo 16? $x \underset{16}{\equiv} y$

Linear Orderings

A linear ordering is any relationship where any two objects x and y that can be compared, exactly one of:

$$x < y \text{ , } x = y \text{, or } y < x$$

is true and where the relation is transitive

- Such a relation is therefore anti-symmetric
- Any collection can therefore be *sorted* according to this relation

Examples of sets which are linearly ordered include:

- Integers 1, 2, 3, 4, 5, 6, ...
- Real numbers 1.2, 1.2001, 1.24, 1.35, 2.78, ...
- The alphabet a, b, c, d, e, f, ..., x, y, z
- Memory address **`0x00000000, 0x00000001, ..., 0xFFFFFFFF`**

We could store linearly ordered sets using arrays or linked lists

Operations on Linear Orderings

Queries that may be asked about linear orderings:

- What are the first and last objects (the *front* and the *back*)?
- What is the k^{th} object?
- What are all objects in a given interval $[a, b]$
- Given a reference to one object in the container:
  - What are the previous and next objects?

Operations that may be performed as a result:

- Sort a collection of objects
- Insert an object into a sorted list
- Insert an object at either the front, the back, or into the k^{th} position

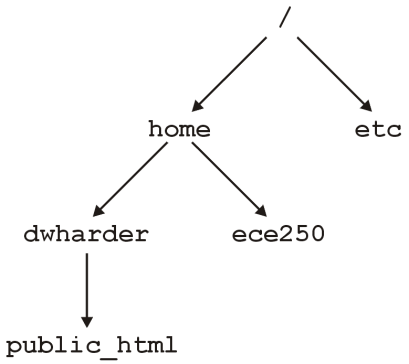
- You will learn how to do this throughout the semester!

Hierarchical Orderings

Consider directories in a file system:

$x \prec y$ if x contains y within one of its subdirectories

– In Unix, there is a single root director /

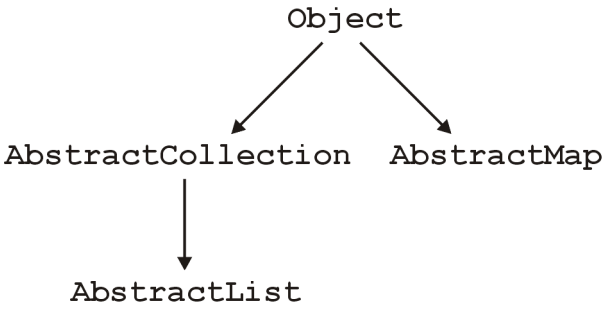


Such structures allow us to organize information

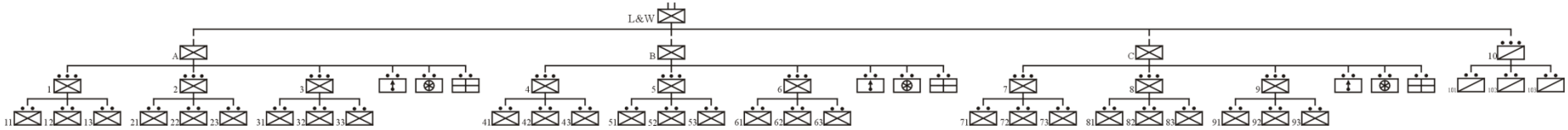
Hierarchical Orderings

Other examples:

Classes in Java and C#




```
int f() {
    int a;
    {
        int b;
        {
            int c;
        }
    }

    {
        int d;
    }
    return a;
}
```



Operations on Hierarchical Orders

If the hierarchical order is explicitly defined (the usual case), given two objects in the container, we may ask:

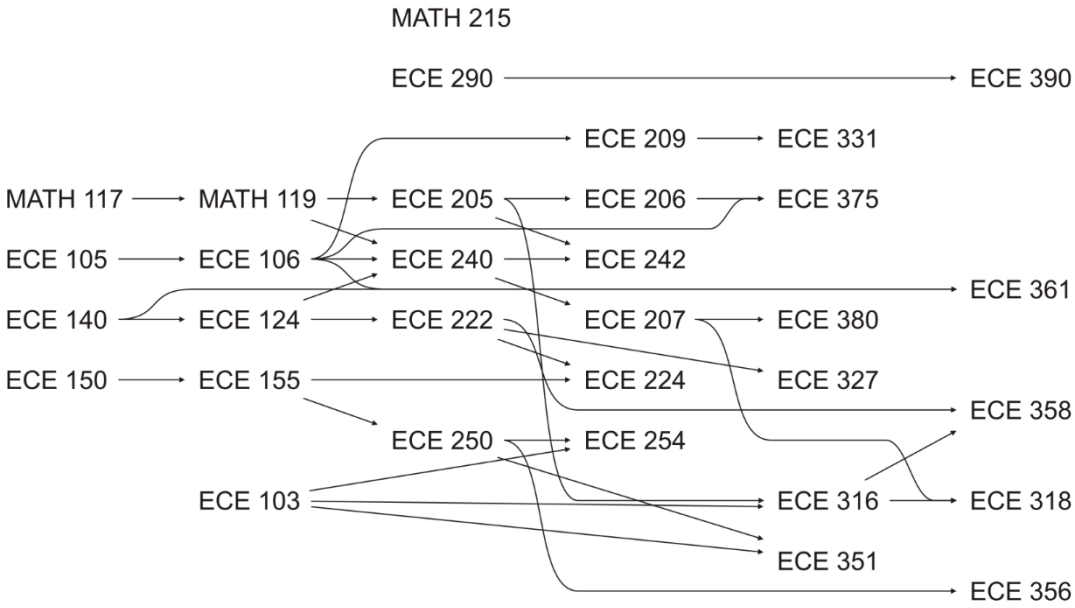
- Does one object contains the other?
- Are both objects at the same depth?
- What is the nearest common predecessor?

Partial Orderings

Partial orderings is denoted as

$x \prec y$ if x is a prerequisite of y

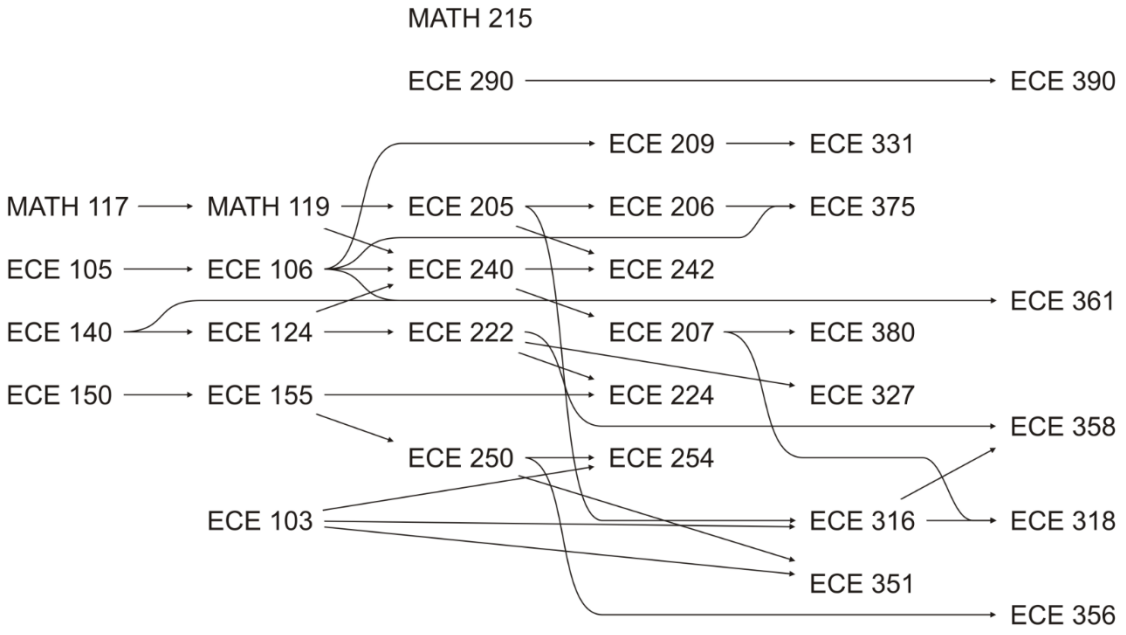
This is not a hierarchy, as there are **multiple starting points** and one class may have **multiple prerequisites**



Partial Orderings

Arrows are necessary to indicate the direction:

- Having completed ECE 140, you can now take ECE 124 and ECE 361



Operations on Partial Orderings

Partial orders are similar to hierarchical orders; consequently, some operations are similar:

- Given two objects, does one precede the other?
- Which objects have no predecessors?
- Which objects are immediately preceded by an object?
  - A hierarchical order has only one immediate predecessor
- Which objects immediately succeed an object?

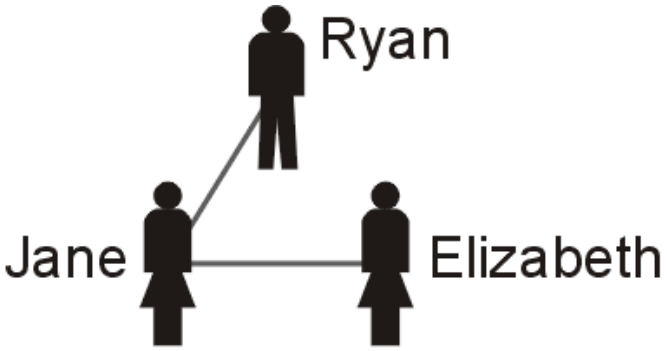
Adjacency Relations

Adjacency relations can be represented as:

$x \leftrightarrow y$ if x and y are friends

Like a tree, we will display such a relationship by displaying a line connecting two individuals if they are friends (a *graph*)

E.g., Jane and Ryan are friends, Elizabeth and Jane are friends, but Elizabeth thinks Ryan is a little odd...

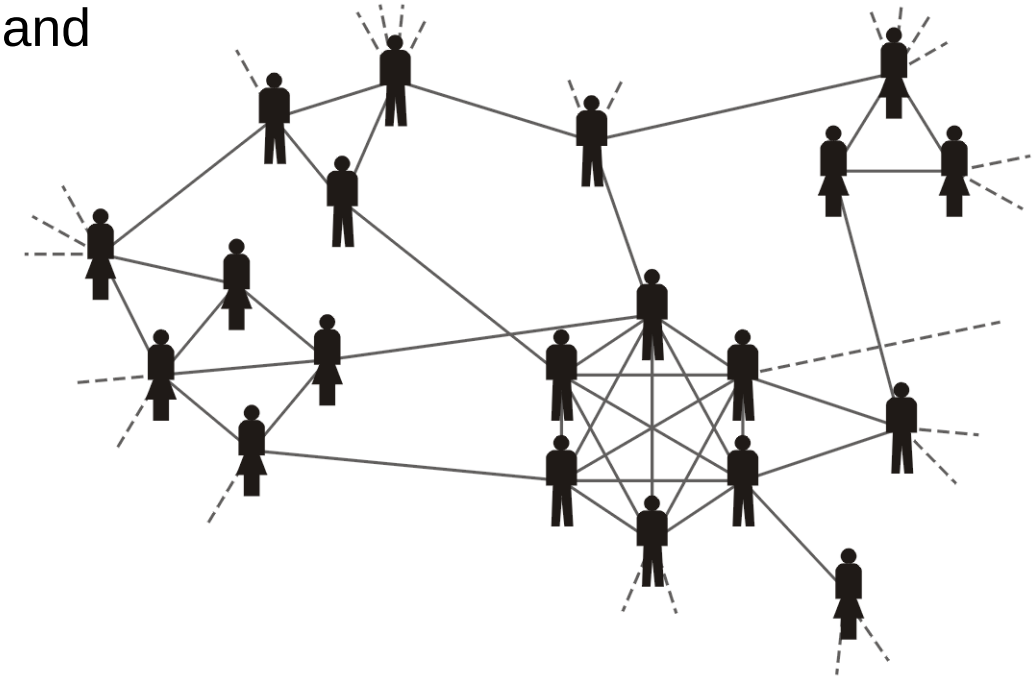


Adjacency Relations

Such a relationship is termed an *adjacency relationship*

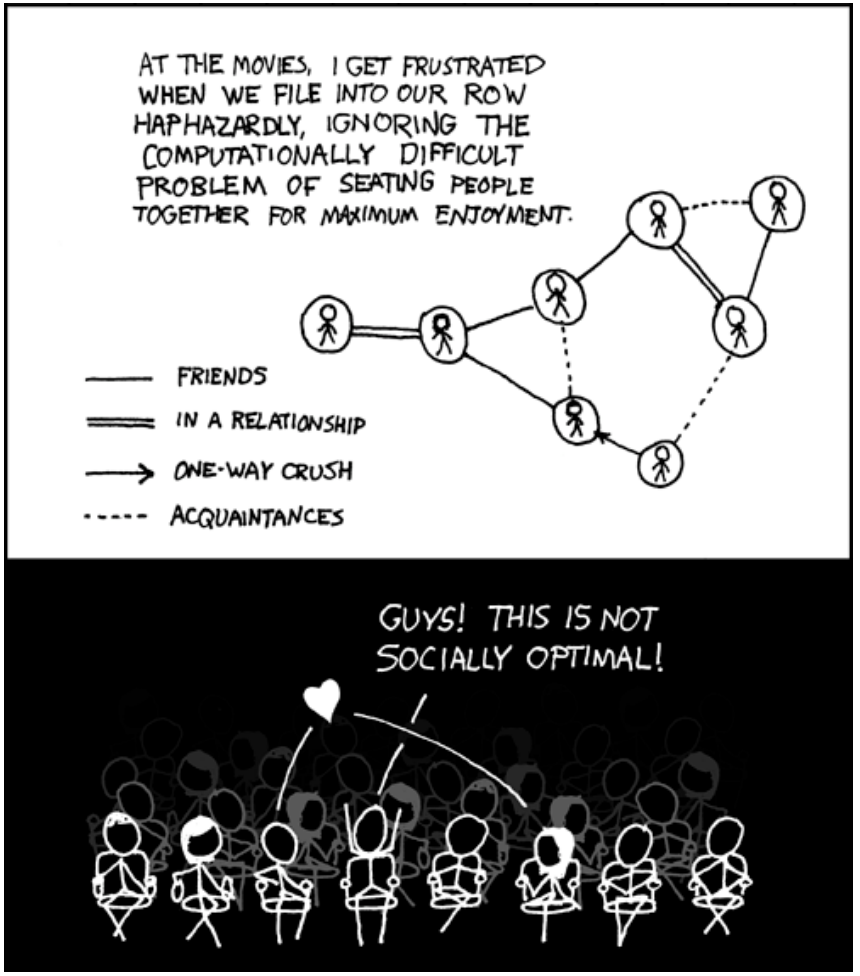
- Two individuals who are related are also said to be *adjacent* to each other

Here we see a hockey team and some of their friends



Adjacency Relations

Alternatively, the graph may be more complex



http://xkcd.com/173/

Adjacency Relations

In some cases, you do not have global relationships, but rather, you are simply aware of neighbouring, or adjacent, nodes

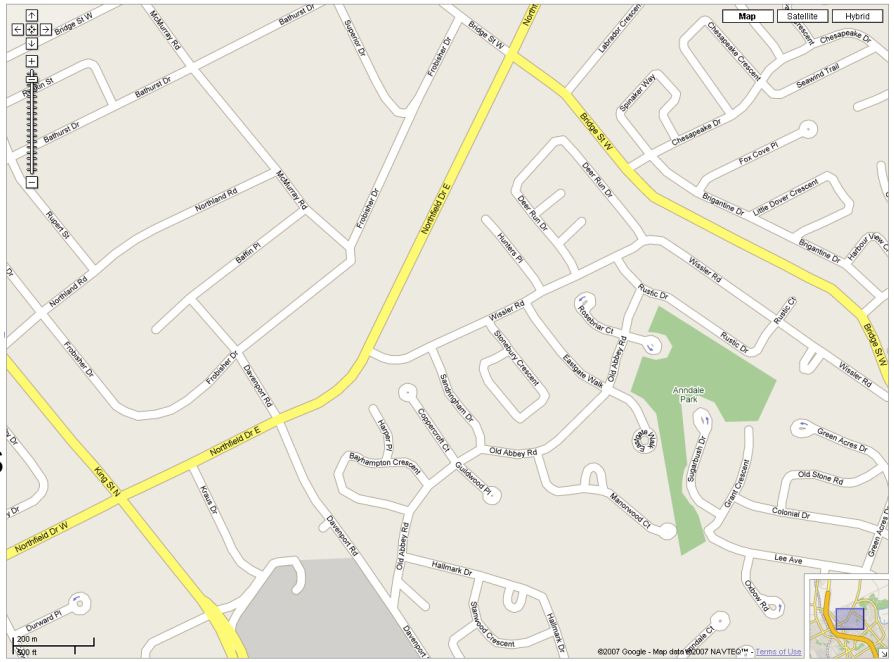
Such a relationship defines a graph, where:

- Nodes are termed vertices
- Edges denote adjacencies

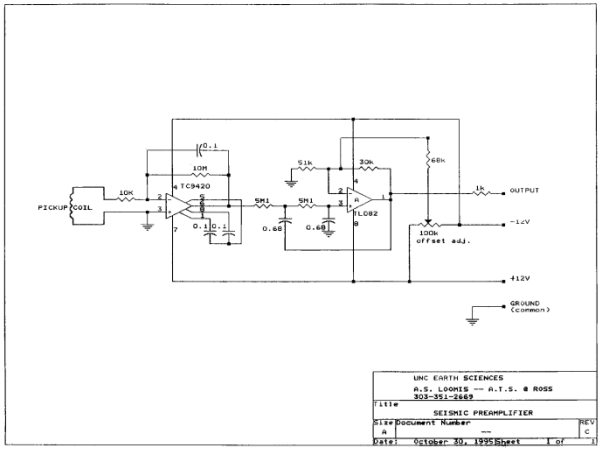
Adjacency Relations

Two examples:

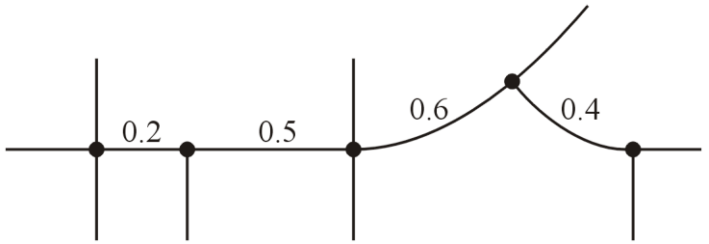
- City streets
  - intersections are vertices
  - streets are edges
- Circuits
  - circuit elements are vertices
  - connections are edges



http://maps.google.ca/



http://esci.unco.edu/resource/circuit.htm



Operations on Adjacency Relations

Given an adjacency relation:

- Are two objects adjacent?
- Iterate through all objects adjacent to one object

Summary of Relations

We have now seen four relationships:

- Linear orderings
- Hierarchical orderings
- Partial orderings
- Adjacency relations

All of these are relationships that exist on the objects we may wish to store, access, and query

Abstract Data Types

In engineering, we tend to see certain patterns that occur over and over in applications

In these circumstances, we first name these patterns and then proceed to define certain standard solutions or implementations

In software in storing objects and relationships in containers, there are reoccurring containers of objects and associated relationships where the actual queries and operations are restricted

- We model such containers by *Abstract Data Types* or ADTs

What's next?

We have discussed containers, relationships, and ADTs

- What is it we want to store and access
- What queries and operations are we interested in

The next question is, how do we implement these efficiently on a computer?

The next step is to look at *data structures*

Summary

In this topic, we have covered:

- The Container ADT as a basic model of organizing data
  - Queries and operations on containers
  - Simple and associative containers
  - Unique or duplicate objects
- Relationships between data
  - Linear ordering
  - Hierarchical ordering
  - Partial ordering
  - Adjacency relation
- In each case, we considered relationship-specific queries and operations
- Abstract Data Types as a model for organizing information

References

Wikipedia, http://en.wikipedia.org/wiki/Container_(abstract_data_type)
http://en.wikipedia.org/wiki/Binary_relation