Data Organization: Trees and Graphs

Announcements

- The first lab exam is next Week.
  We posted exercises for practice on the course web site.

- Please check your grades in Autolab.
  If you are missing any grades that should have been entered, alert your TA and the instructors.

Last Lesson

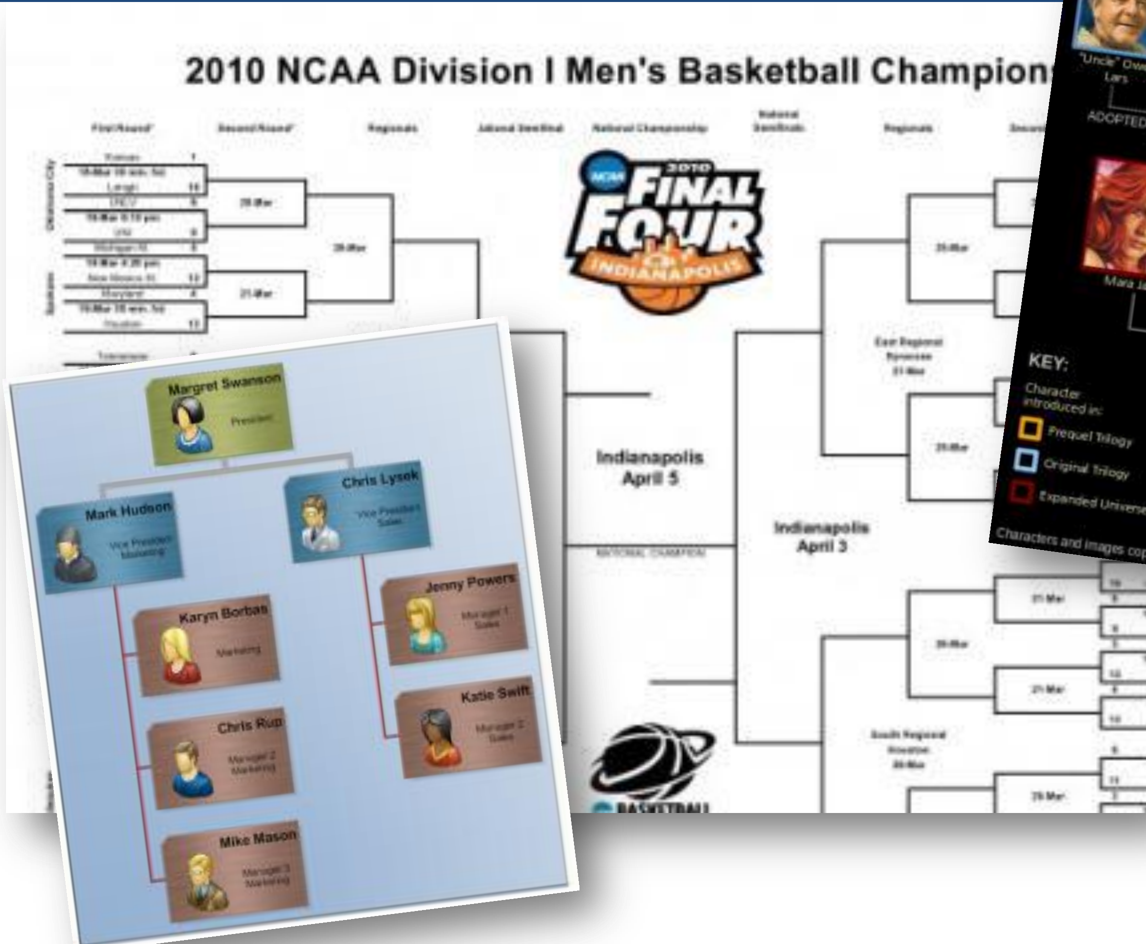
- Arrays
- Linked lists
- Hash tables

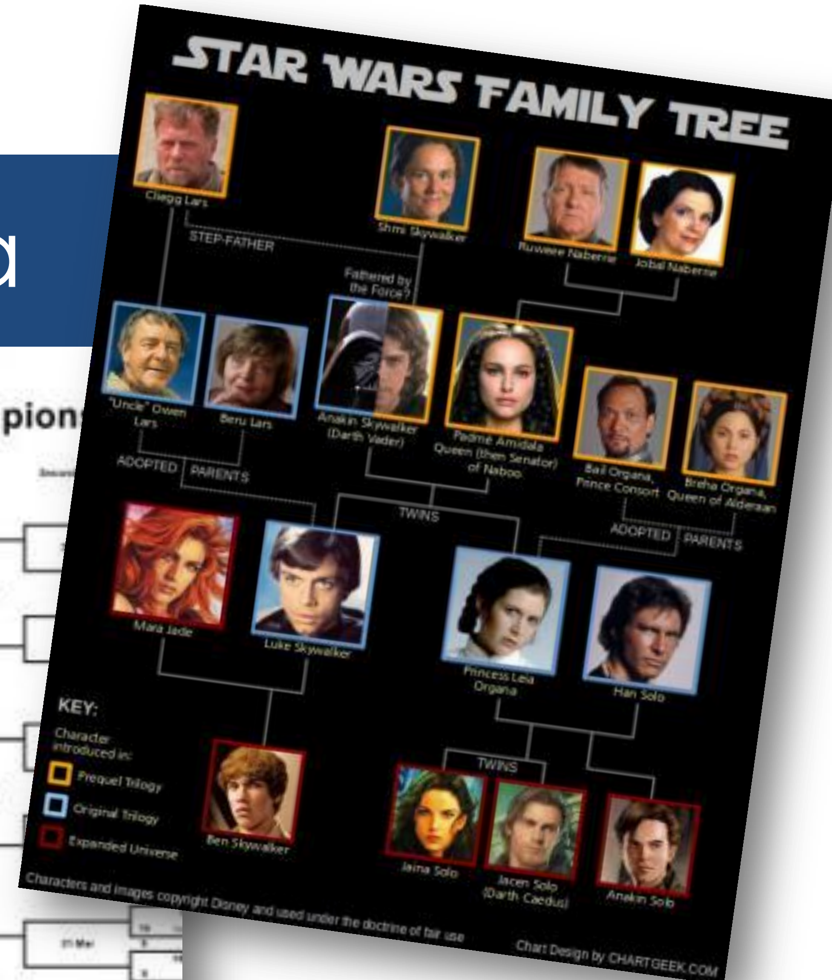
No hierarchy or relationship between data items, other than their order in the sequence in the case of arrays and linked lists

Today

- Data structures for hierarchical data

Hierarchical Data

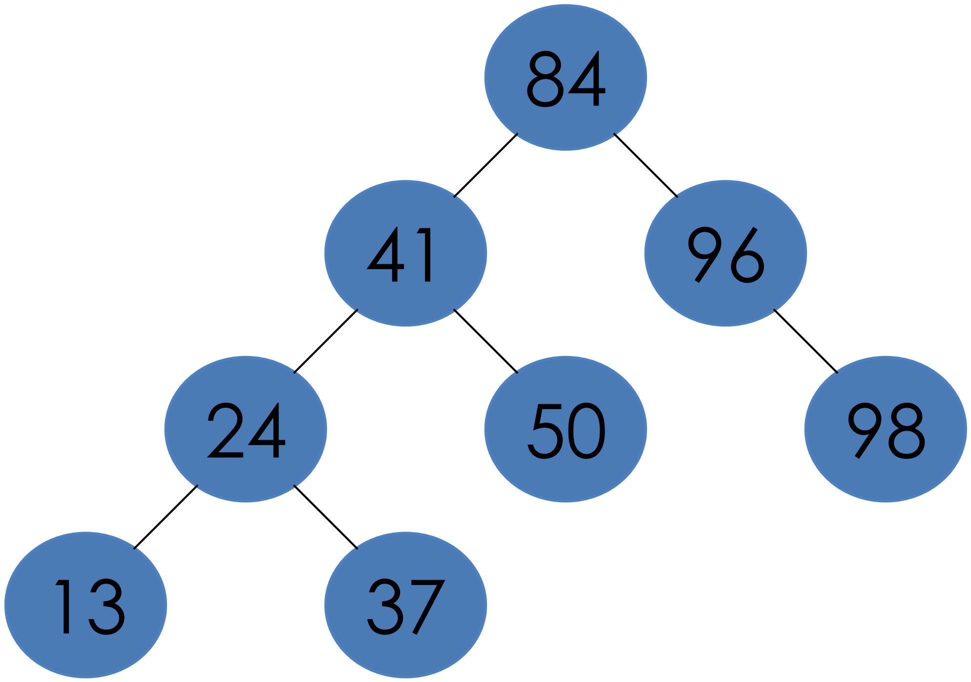




Trees

- A **tree** is a hierarchical data structure.
  - Every tree has a **node** called the **root**.
  - Each node can have 1 or more nodes as **children**.
  - A node that has no children is called a **leaf**.
- A common tree in computing is a **binary tree**.
  - A binary tree consists of nodes that have at most 2 children.
- **Applications:** data compression, file storage, game trees

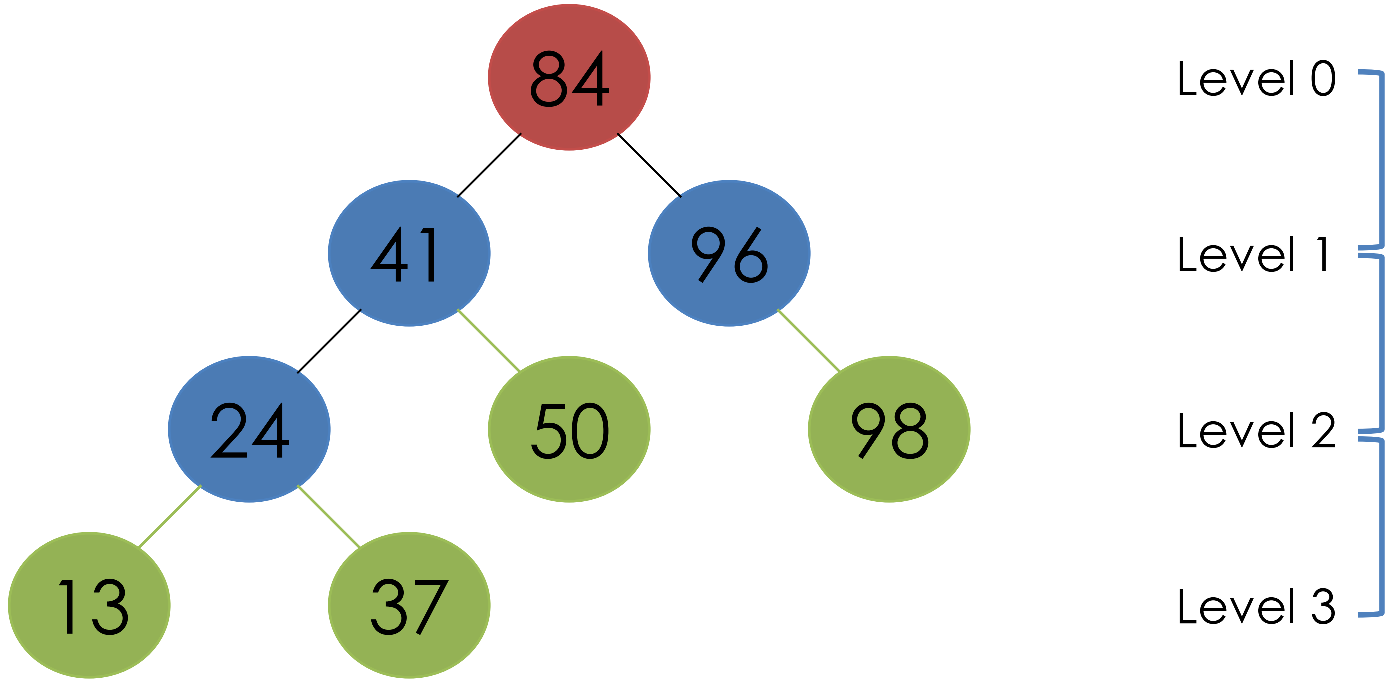
Binary Tree



In order to illustrate main ideas we label the tree nodes with the keys only.
In fact, every node would also store the rest of the data associated with that key. Assume that our tree contains integers keys.

Which one is the root?
Which ones are the leaves (external nodes)?
Which ones are internal nodes?
What is the height of this tree?

Binary Tree



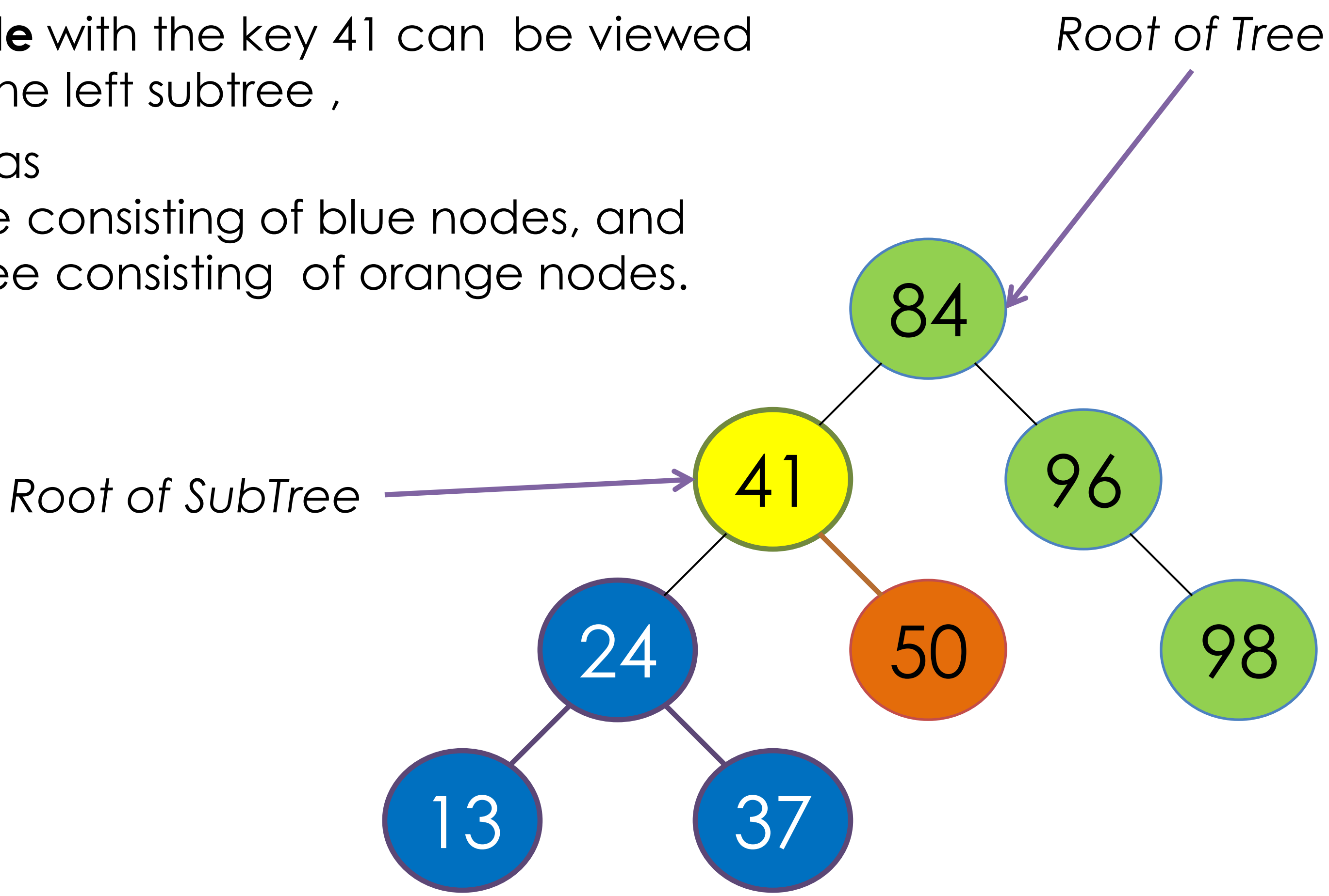
The **root** contains the data value **84**.
There are **4 leaves** in this binary tree: nodes containing **13, 37**, **50**, **98**.
There are **3 internal nodes** in this binary tree: nodes containing **41**, **96**, **24**
This binary tree has **height 3** – considering root is at level 0,
the **maximum level** among all nodes is **3**

Binary Tree

<u>Note the **recursive** structure</u>

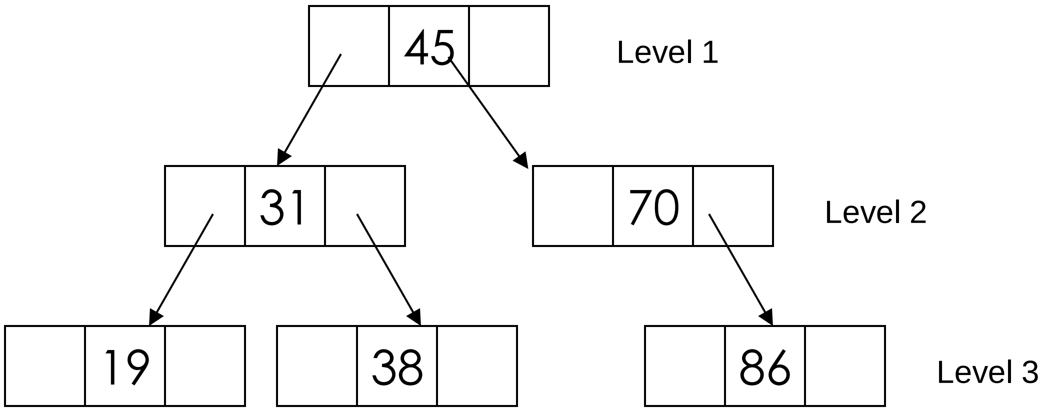
The **yellow node** with the key 41 can be viewed as the root of the left subtree ,

which in turn has
  a left subtree consisting of blue nodes, and
  a right subtree consisting of orange nodes.



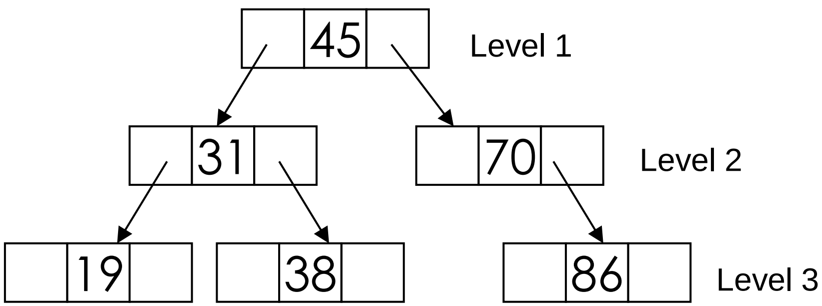
Binary Trees: Implementation

- One common implementation of binary trees uses nodes like a linked list does.
  - Instead of having a "**next**" pointer, each node has a "**left**" pointer and a "**right**" pointer.



Using Nested Lists

- Languages like Python do not let programmers manipulate pointers explicitly.
- We could use Python lists to implement binary trees. For example:



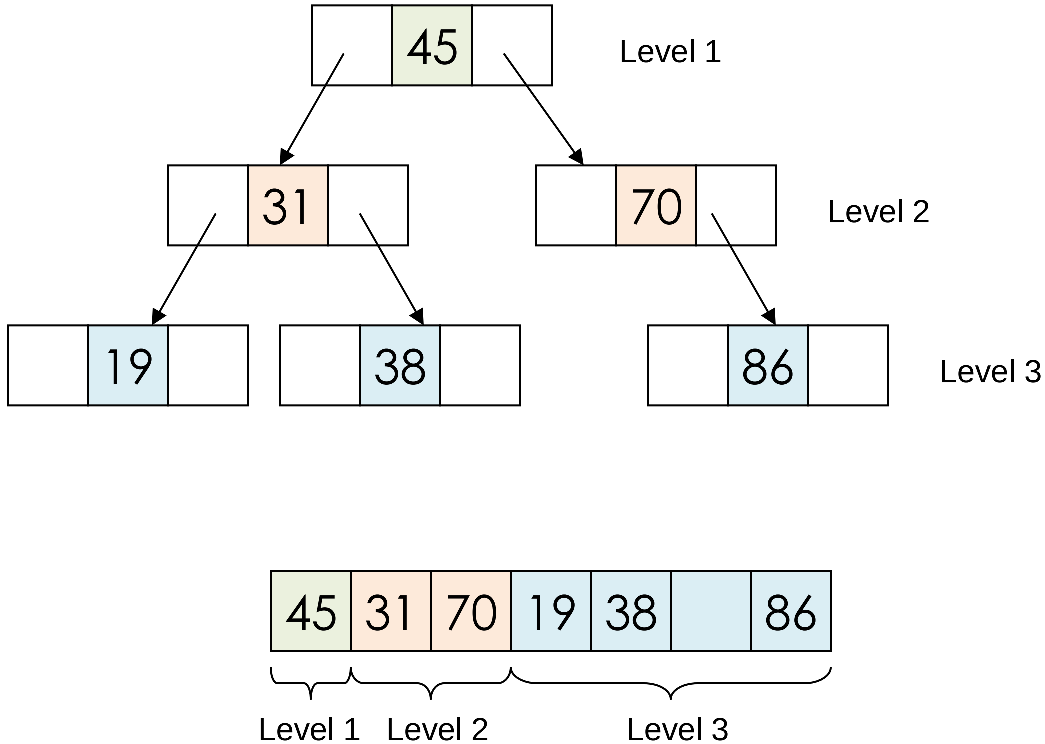

```
[45, left, right]

[45,[31,left,right],[70,left,right]]

[45, [31, [19,[],[]], [38, [], []]],
     [70, [],         [86, [], []]]
]
```

[] stands for an empty tree

Arrows point to subtrees

Using One Dimensional Lists

- We could also use a flat (one-dimensional list).



Dynamic Date Set Operations

- Insert
- Delete
- Search
- Find min/max
- ...

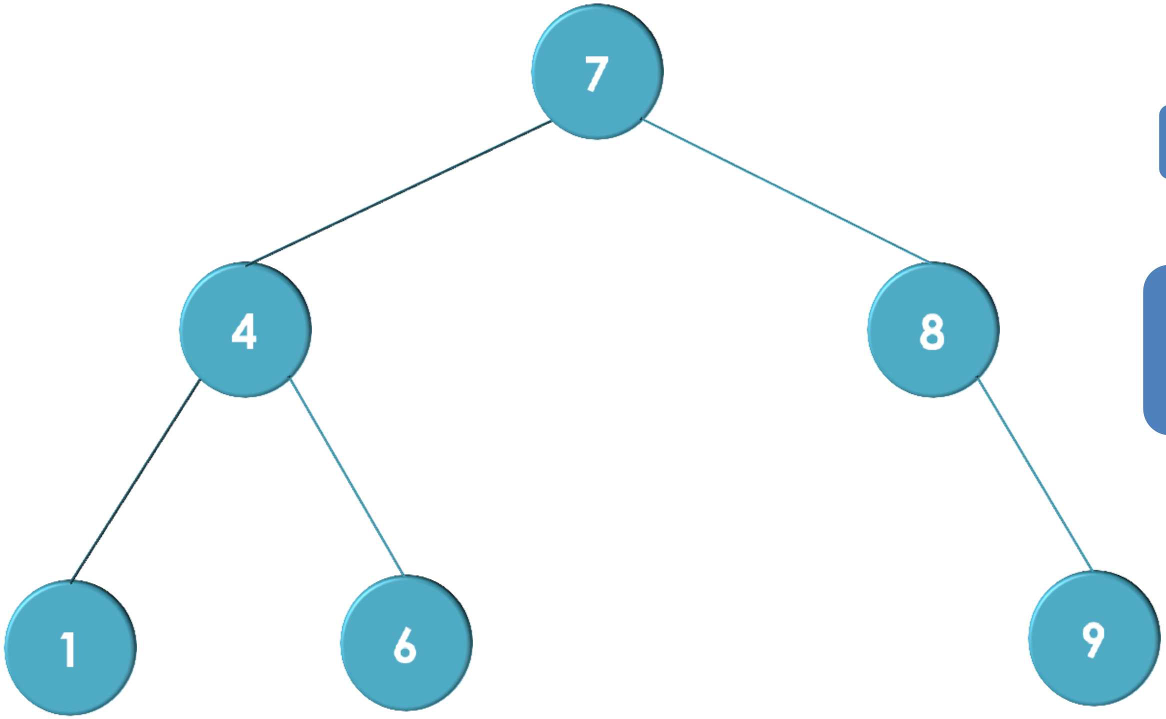
Choosing a specific data structure has consequences on which operations can be performed faster.

Binary Search Tree (BST)

- A binary search tree (BST) is a binary tree that satisfies the binary **search tree ordering invariant** stated on the next slide

Example: Binary Search Tree

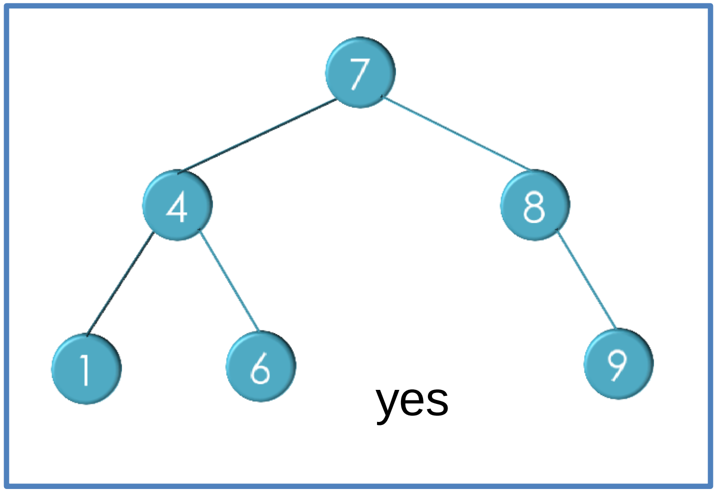
BST ordering invariant: At any node with key *k,* all keys of elements in the left subtree are strictly less than *k* and all keys of elements in the rightsubtree are strictly greater than *k* (assume that there are no duplicates in the tree)

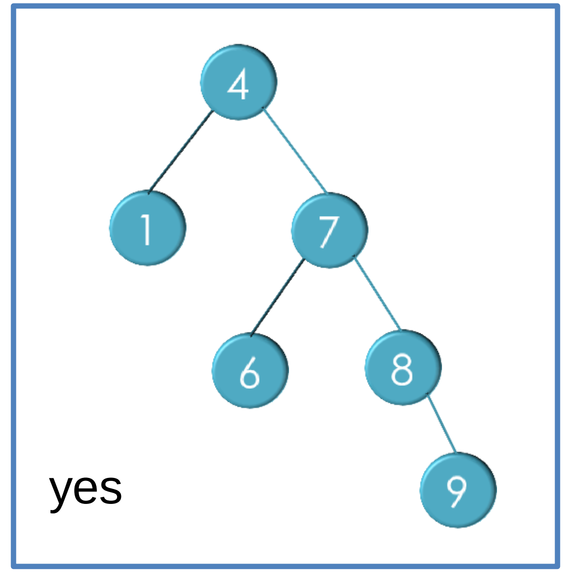


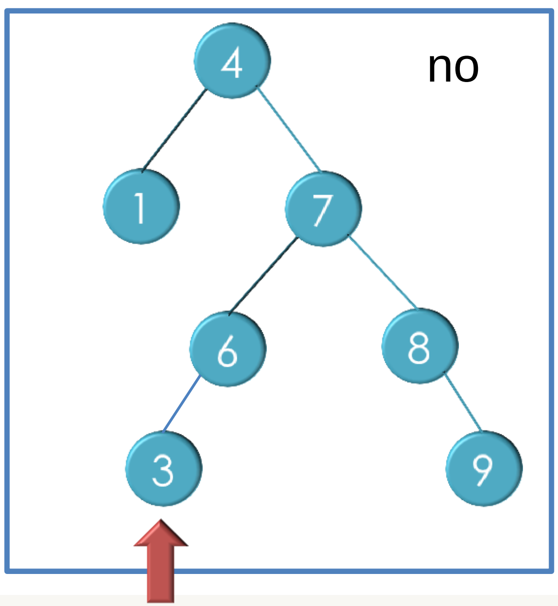
Binary tree

Satisfies the ordering invariant

Test: Is this a BST?







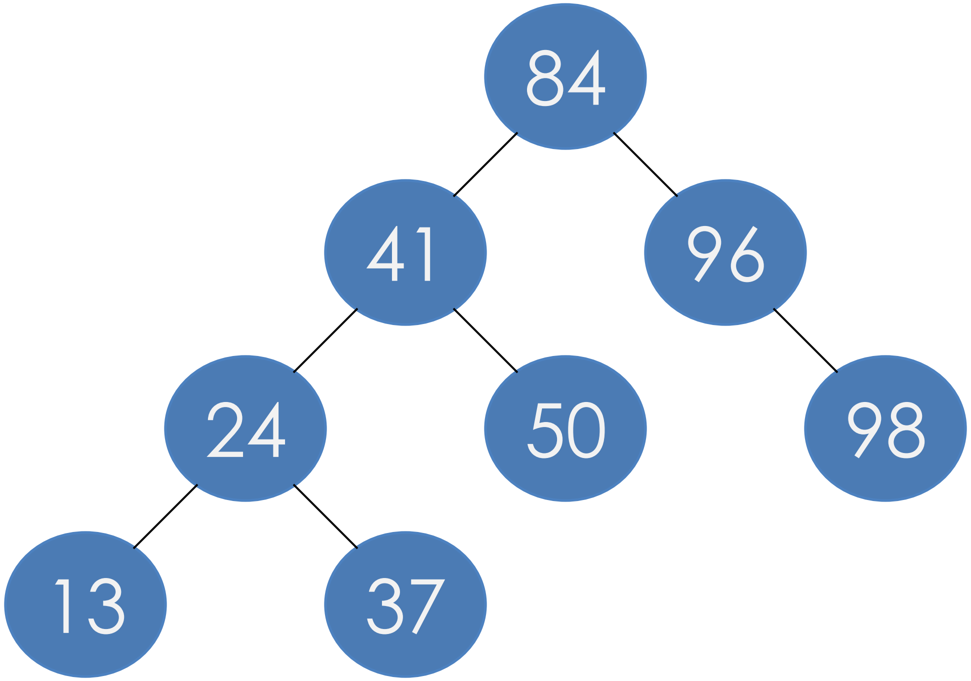
Inserting into a BST

For each data value that you wish to insert into the binary search tree:

- Start at the root and compare the new data value with the root.
- If it is less, move down left. If it is greater, move down right.
- Repeat on the child of the root until you end up in a position that has no node.
- Insert a new node at this empty position.

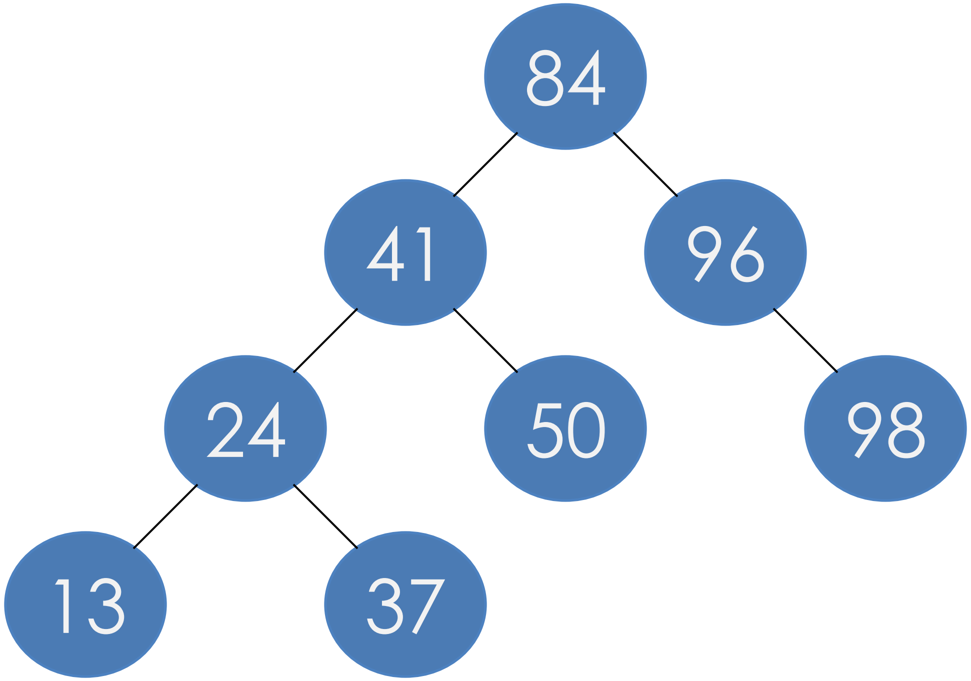
Example

- Insert: 84, 41, 96, 24, 37, 50, 13, 98



Using a BST

- How would you search for an element in a BST?

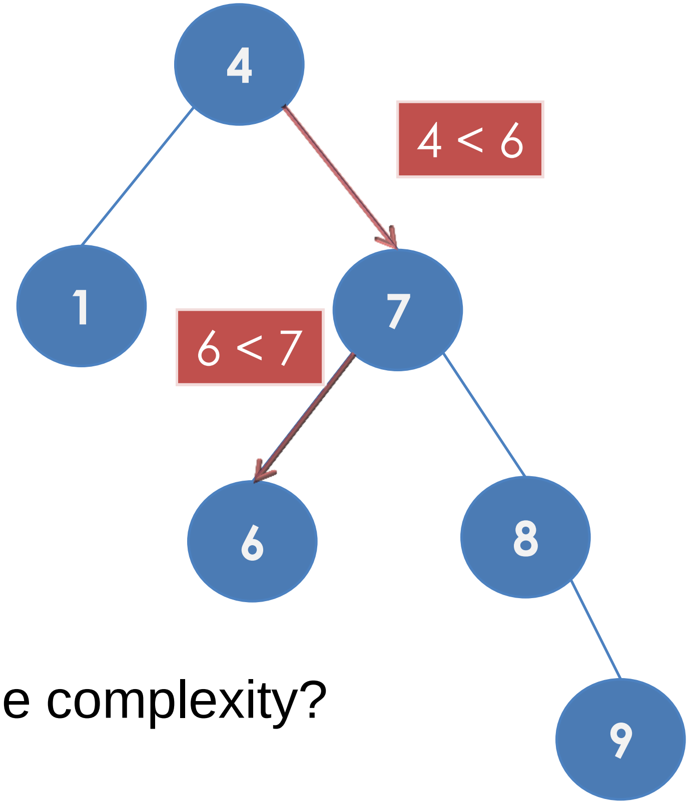


Searching a BST

- For the key that you wish to search
  - Start at the root and compare the key with the root. If equal, key found.
  - Otherwise
    - If it is less, move down left. If it is greater, move down right. Repeat search on the child of the root.
    - If there is no non-empty subtree to move to, then key not found.

Searching the tree

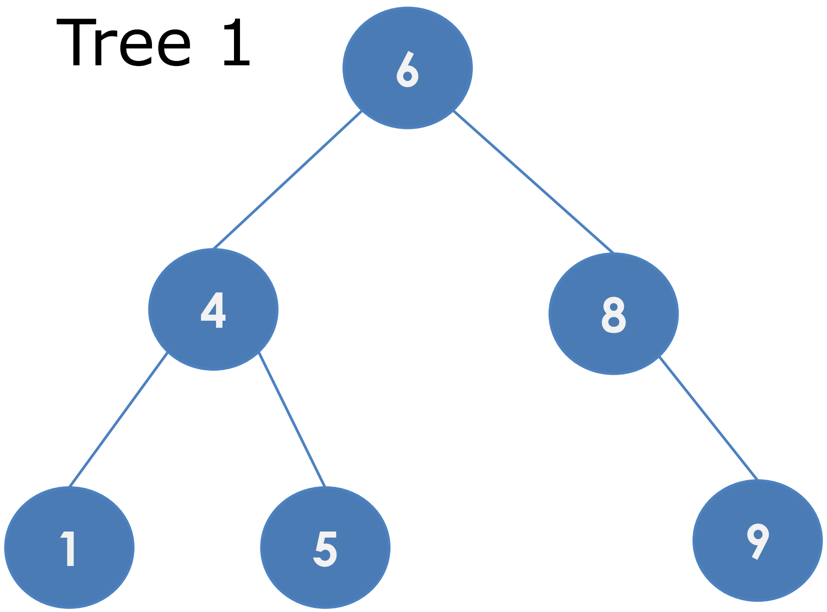
Example: searching for 6



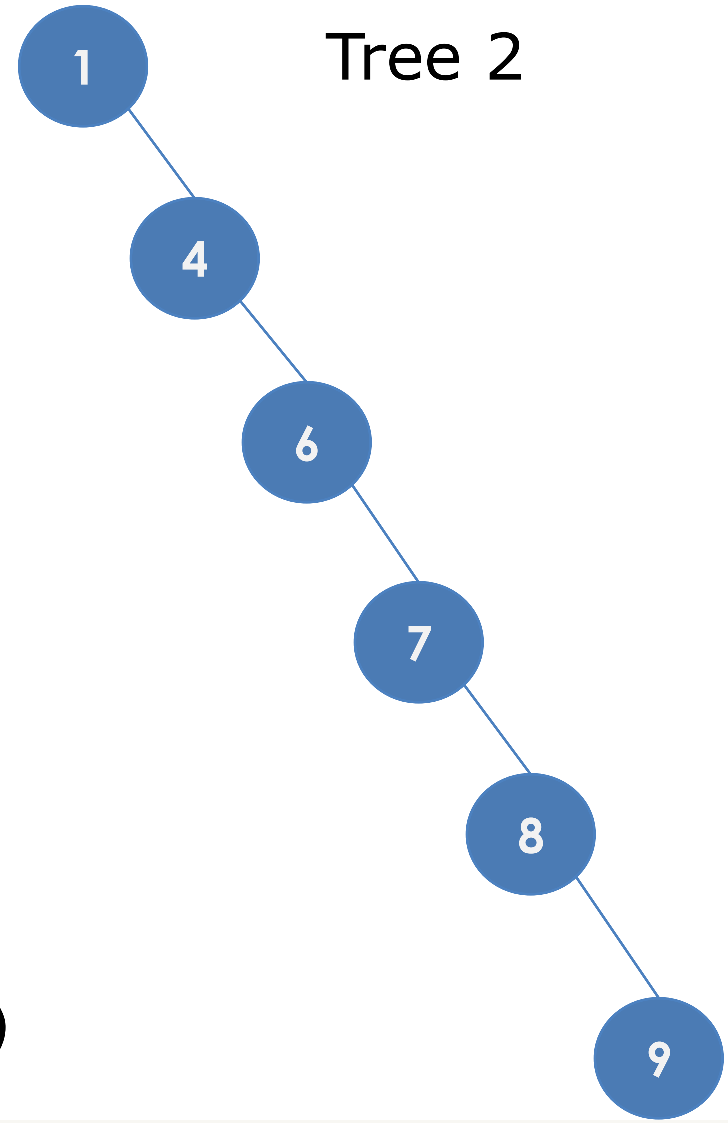
Can we form a conjecture about worst case complexity?

Time complexity of search

Tree 1



Tree 2



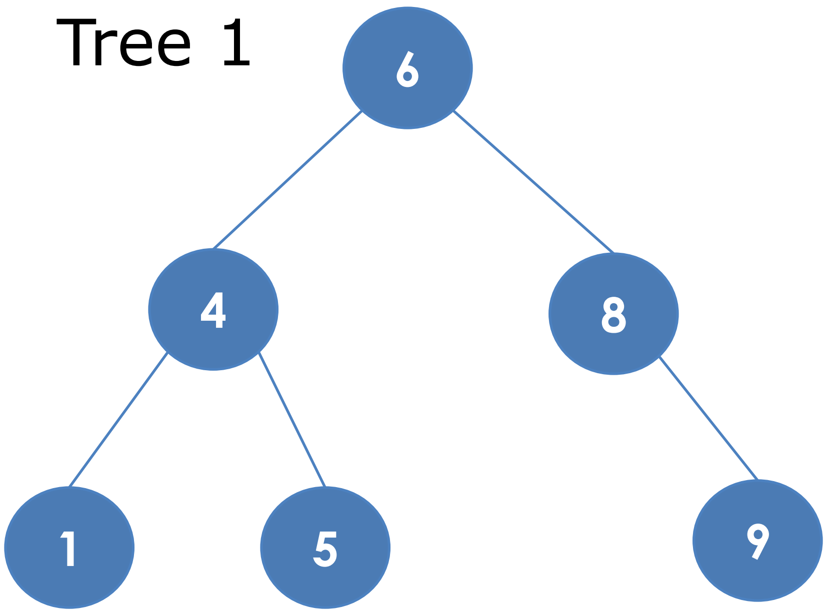
Number of nodes: *n*

Worst case: *O(height)*

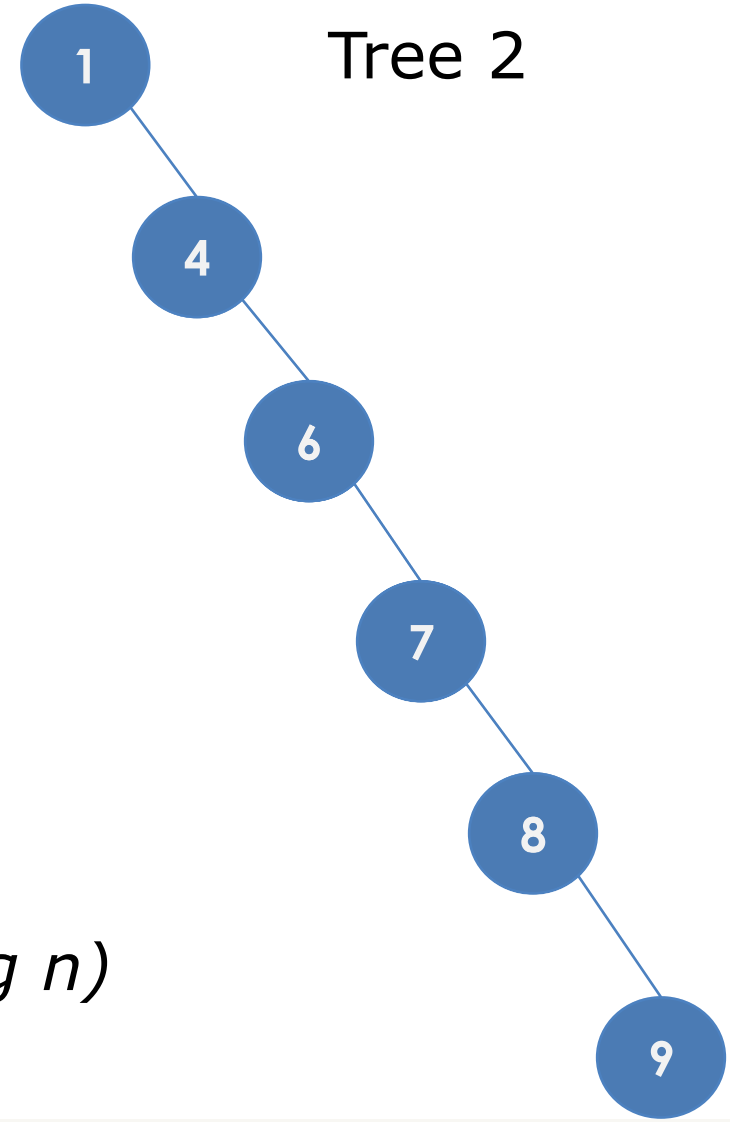
Worst *height*: *n* → *O(n)*

Time complexity of search

Tree 1



Tree 2



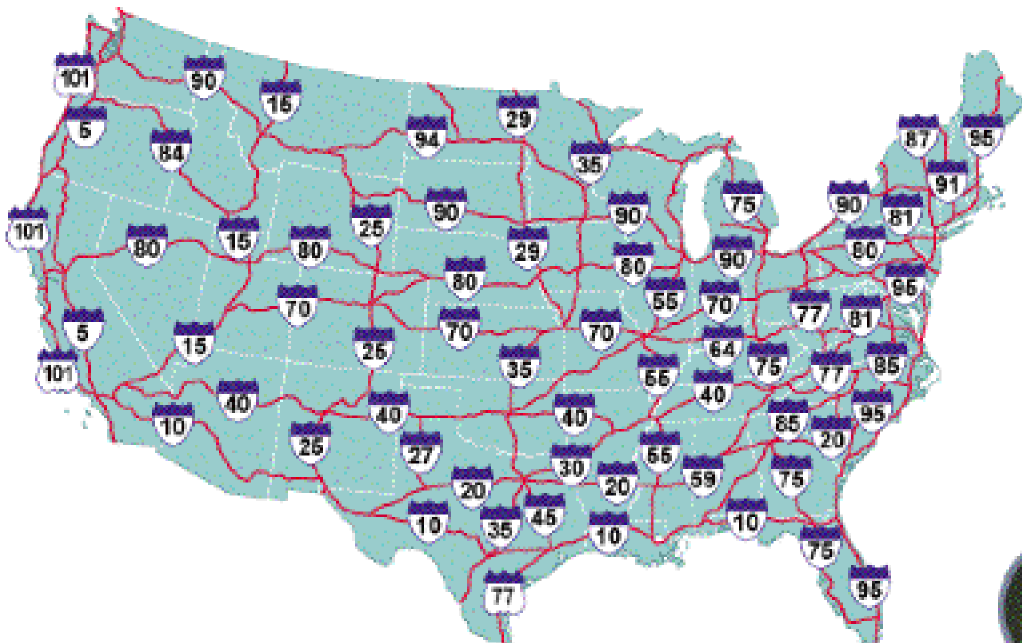
Number of nodes: *n*

What if we could always have balanced trees? → *O(log n)*

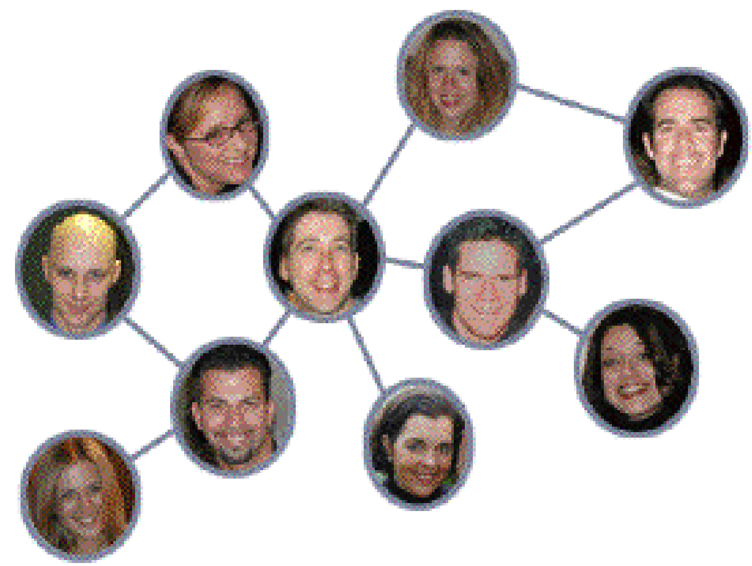
Exercises

- How you would find the minimum and maximum elements in a BST?
- What would be output if we walked the tree in left-node-right order?

Graphs



101
90
15
5
84
29
94
35
87
95
91
101
80
15
80
25
90
29
90
75
90
81
80
80
90
95
70
55
70
77
81
5
26
70
35
70
64
75
77
85
15
101
40
10
40
25
27
40
30
20
66
59
65
20
95
85
75
20
10
35
45
10
10
75
77
95



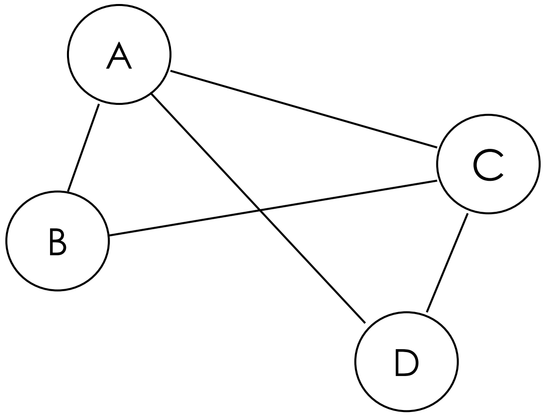
Graphs

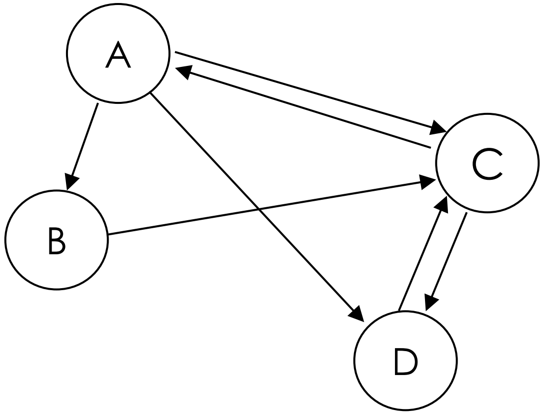
A **graph** is a data structure that consists of a set of vertices and a set of edges connecting pairs of the vertices.

- A graph doesn't have a **root**, per se.
- A vertex can be connected to any number of other vertices using edges.
- An edge may be bidirectional or directed (one-way).
- An edge may have a weight on it that indicates a cost for traveling over that edge in the graph.

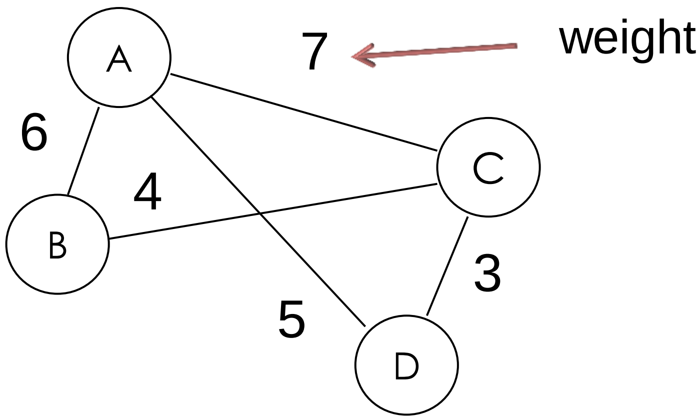
Applications: computer networks, transportation systems, social relationships

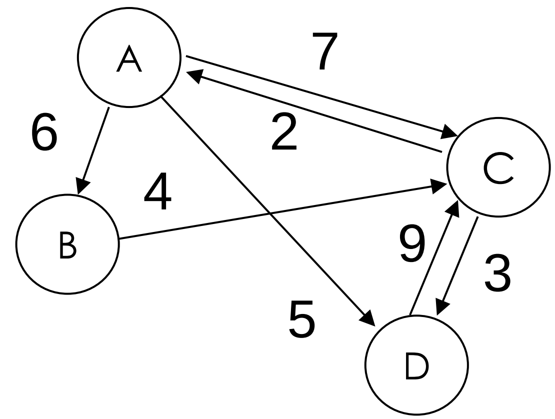
Undirected and Directed Graphs





Undirected and Directed Graphs

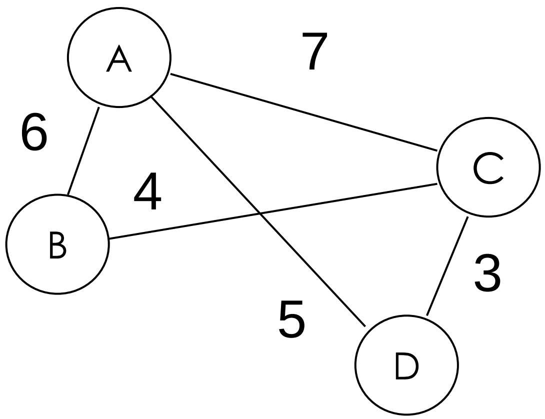




from \ to	A	B	C	D
A	0	6	7	5
B	6	0	4	∞
C	7	4	0	3
D	5	∞	3	0

from \ to	A	B	C	D
A	0	6	7	5
B	∞	0	4	∞
C	2	∞	0	3
D	∞	∞	9	0

Graphs in Python

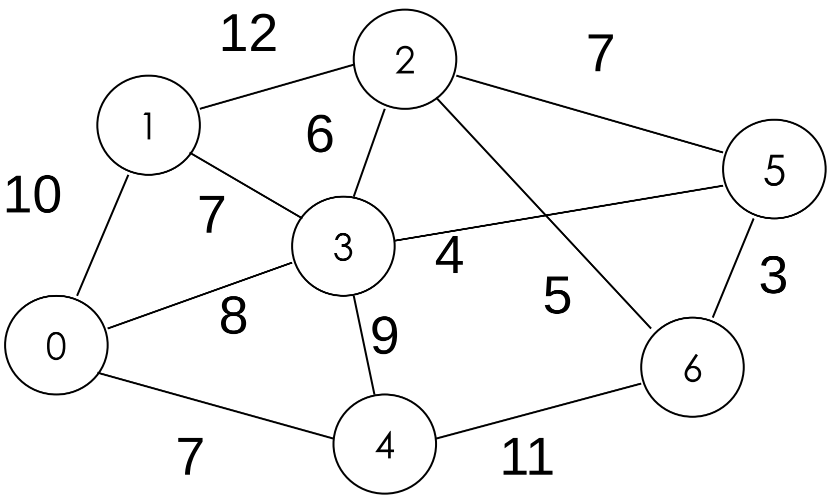


to / from

	A	B	C	D
A	0	6	7	5
B	6	0	4	∞
C	7	4	0	3
D	5	∞	3	0

```
graph =
[ [ 0, 6, 7, 5 ],
  [ 6, 0, 4, float('inf') ],
  [ 7, 4, 0, 3],
  [ 5, float('inf'), 3, 0] ]
```

An Undirected Weighted Graph



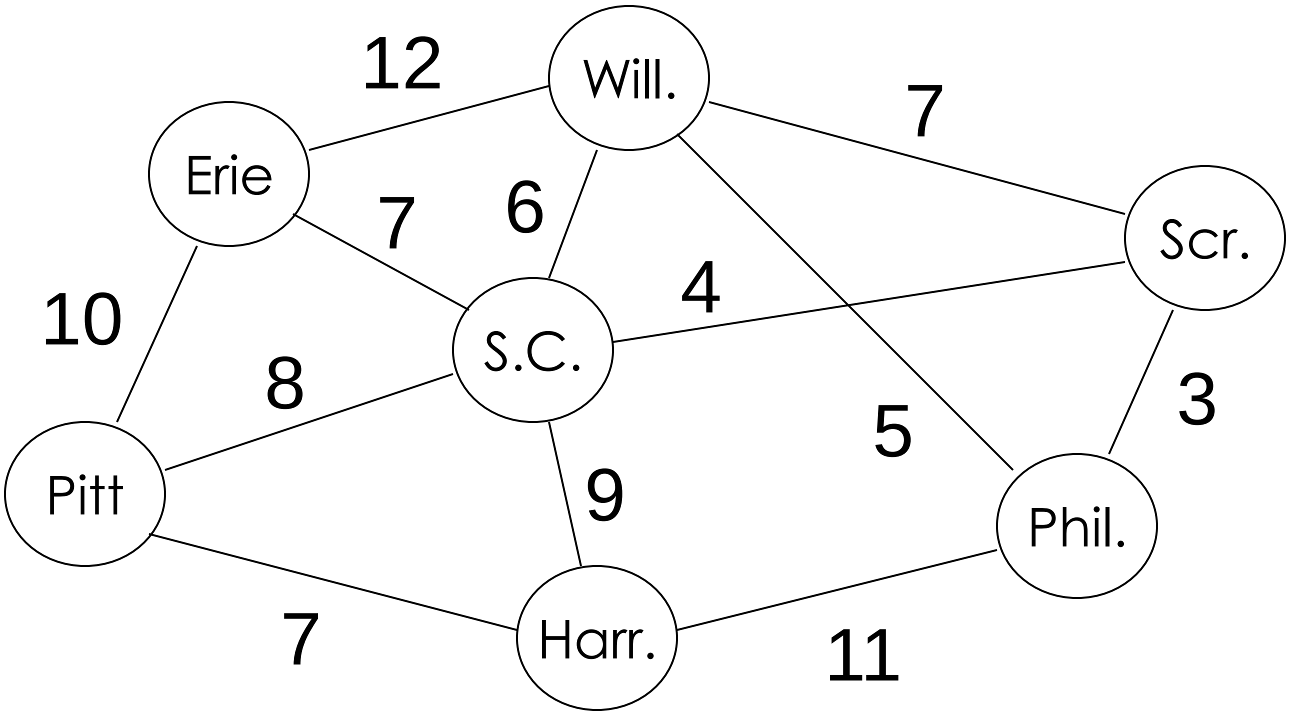
0	1	2	3	4	5	6
Pitt.	Erie	Will.	S.C.	Harr.	Scr.	Phil.

vertices

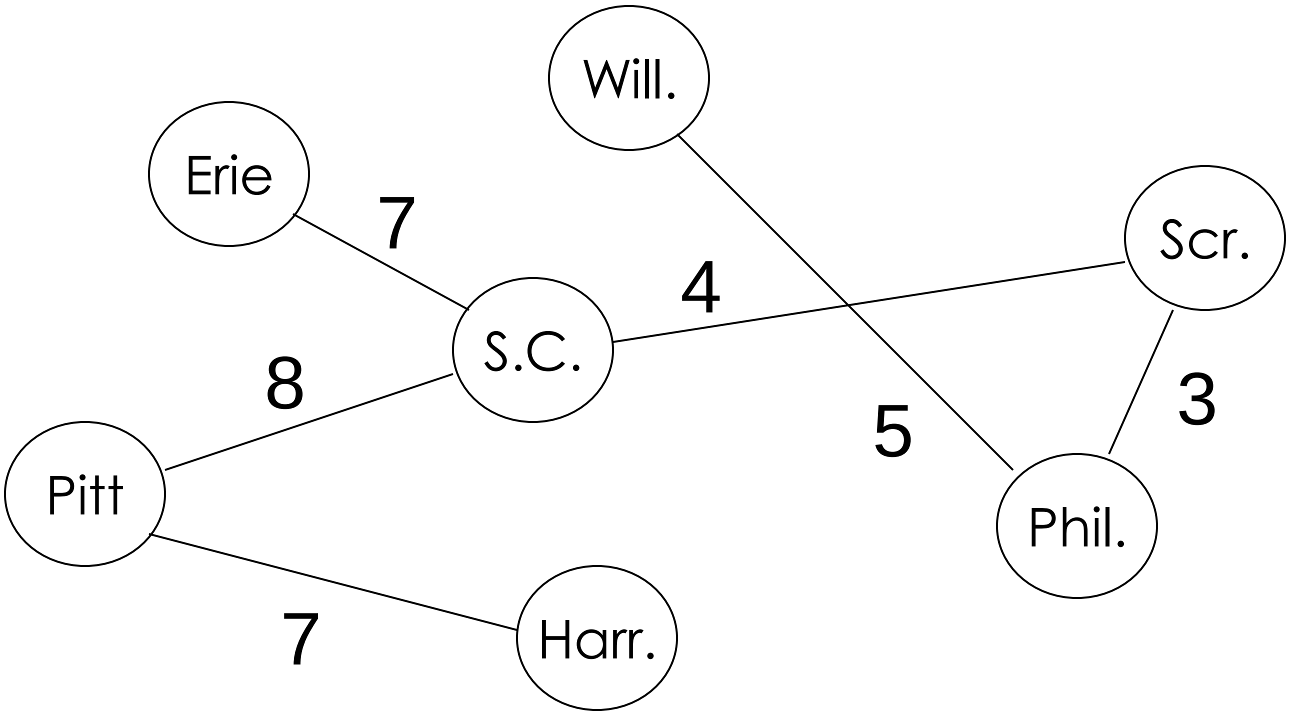
from \ to	0	1	2	3	4	5	6
0	0	10	∞	8	7	∞	∞
1	10	0	12	7	∞	∞	∞
2	∞	12	0	6	∞	7	5
3	8	7	6	0	9	4	∞
4	7	∞	∞	9	0	∞	11
5	∞	∞	7	4	∞	0	3
6	∞	∞	5	∞	11	3	0

edges

Original Graph

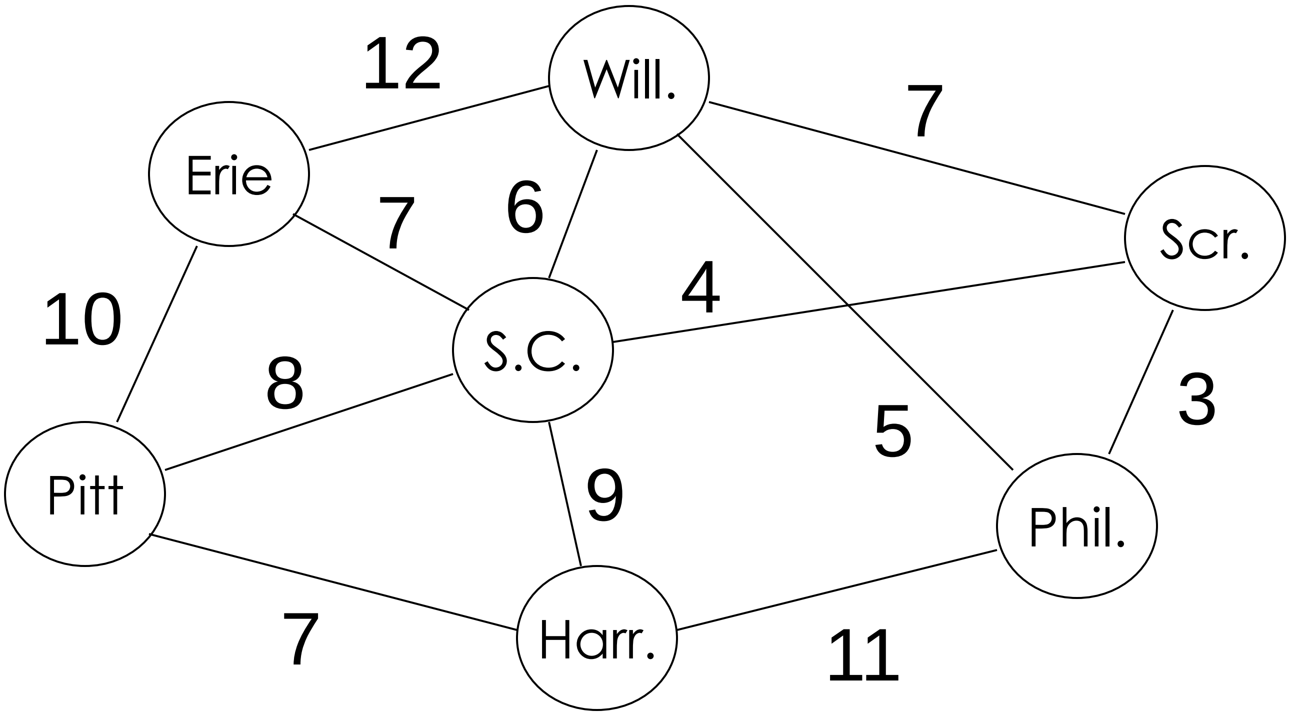


A Minimal Spanning Tree

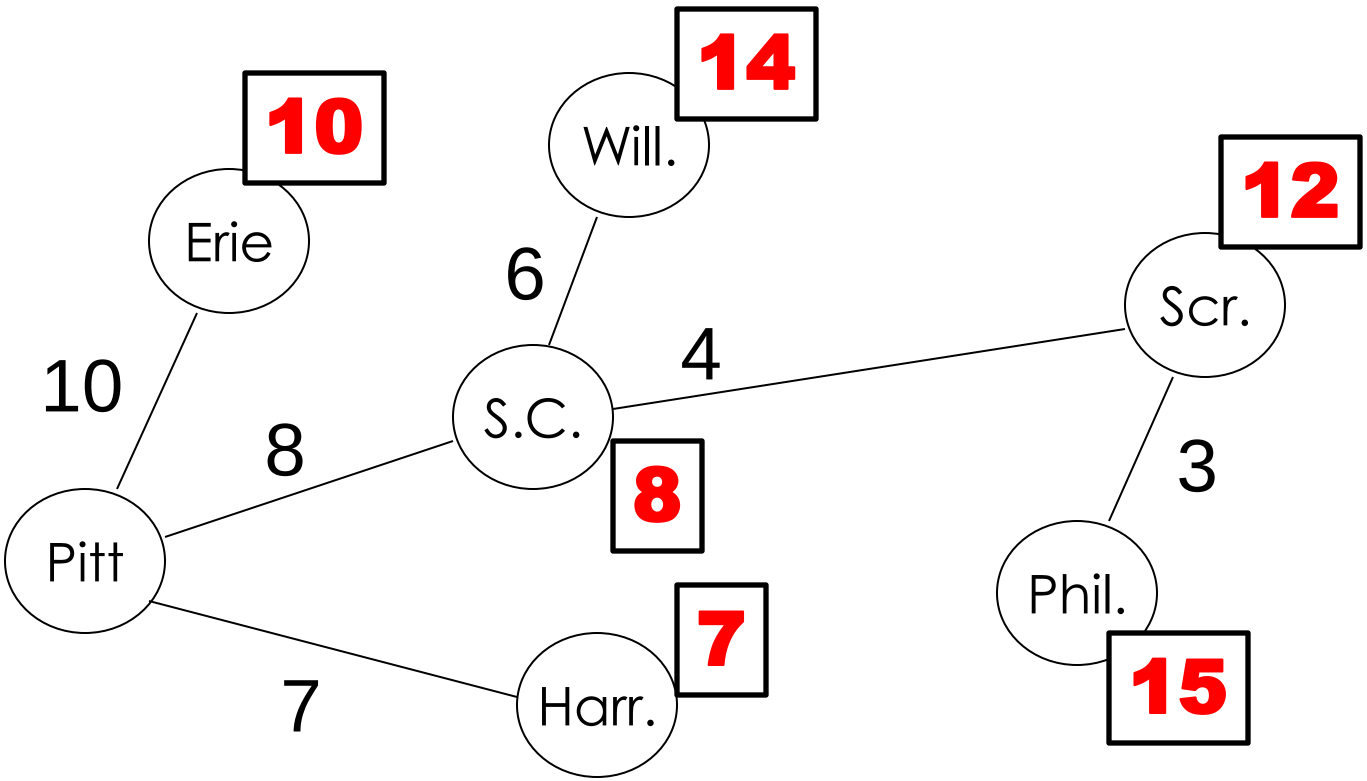


The minimum total cost to connect all vertices using edges from the original graph without using cycles. (minimum total cost = 34)

Original Graph



Shortest Paths from Pittsburgh



The total costs of the shortest path from Pittsburgh to every other location using only edges from the original graph.

Graph Algorithms

There are algorithms to compute the minimal spanning tree of a graph and the shortest paths for a graph.

There are algorithms for other graph operations:

- If a graph represents a set of pipes and the number represent the maximum flow through each pipe, then we can determine the maximum amount of water that can flow through the pipes assuming one vertex is a "source" (water coming into the system) and one vertex is a "sink" (water leaving the system)
- Many more graph algorithms... very useful to solve real life problems.

We did not focus on graph algorithms in this unit. We only covered how to represent them with lists.

Next Time

Data Representation