UNIT 6C
Organizing Data: Trees and Graphs

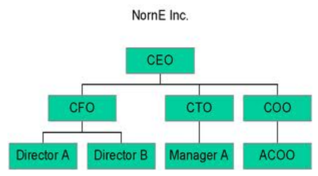
Last Lecture

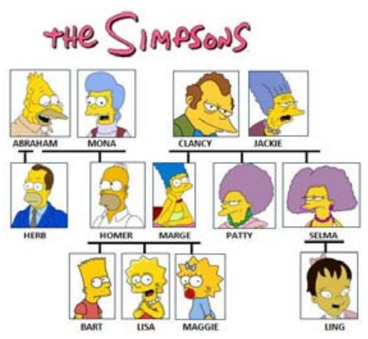
- Hash tables
  - Using hash function to map keys of different data types to array indices
  - Constant search time if the load factor is small
- Associative arrays in Ruby

This Lecture

- Non-linear data structures
  - Trees, specifically, binary trees
  - Graphs

Trees

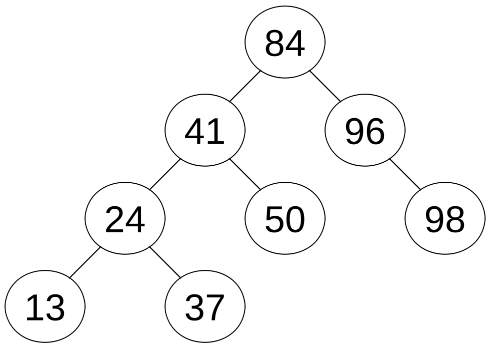




Trees

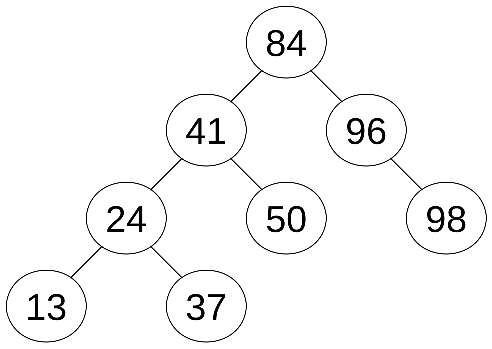
- A **tree** is a hierarchical data structure.
  - Every tree has a **node** called the **root**.
  - Each node can have 1 or more nodes as **children**.
  - A node that has no children is called a **leaf**.
- A common tree in computing is a **binary tree**.
  - A binary tree consists of nodes that have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

Binary Tree



Which one is the root?
Which ones are the leaves?
Is this a complete binary tree?
What is the height of this tree?

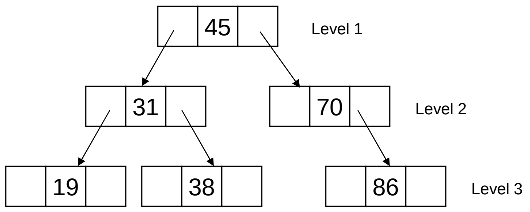
Binary Tree



The root has the data value 84.
There are 4 leaves in this binary tree: 13, 37, 50, 98.
This binary tree is not complete.
This binary tree has height 3.

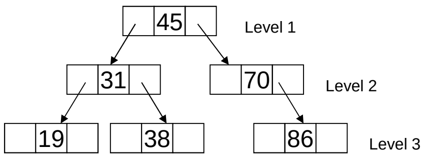
Binary Trees: Implementation

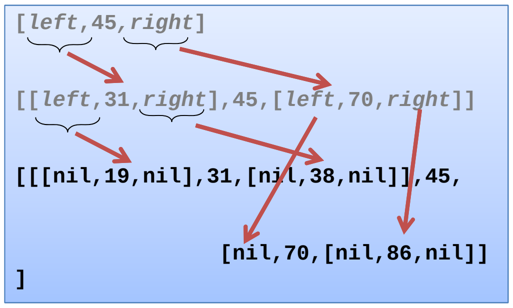
- One common implementation of binary trees uses nodes like a linked list does.
  - Instead of having a "next" pointer, each node has a "left" pointer and a "right" pointer.



Using Nested Arrays

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





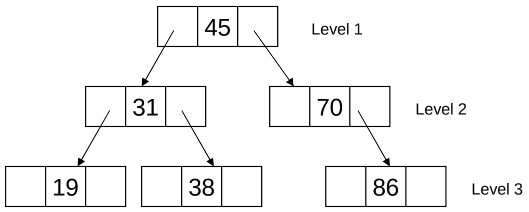
```
[left,45,right]

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

[[[nil,19,nil],31,[nil,38,nil]],45,

                [nil,70,[nil,86,nil]]
]
```

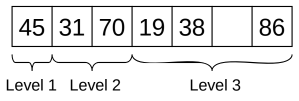
`nil` stands for an empty tree

Arrows point to subtrees

Using One Dimensional Arrays

- We could also use a flat array.





Dynamic Set Operations

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

Binary Search Tree (BST)

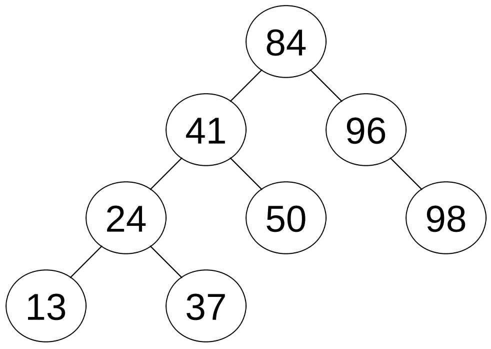
- A binary search tree (BST) is a binary tree such that
  - All nodes to the left of any node have data values less than that node
  - All nodes to the right of any node have data values greater than that node

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.

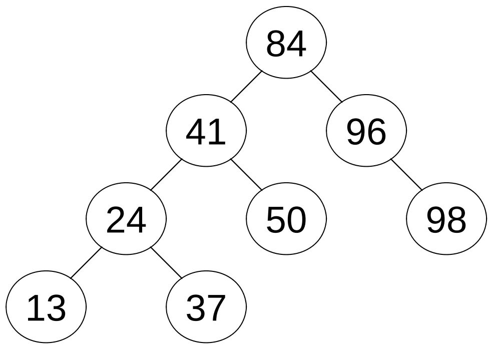
Example

- 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 (data value) 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.

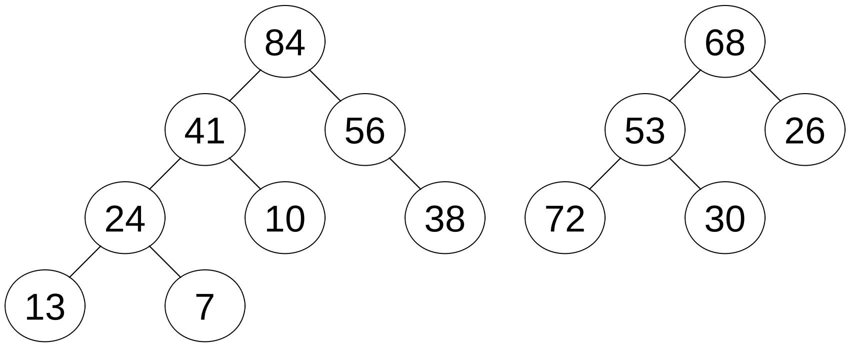
Exercises

- 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?

Max-Heaps

- A max-heap is a binary tree such that
  - The largest data value is in the root
  - For every node in the max-heap, its children contain smaller data.
  - The max-heap is an almost-complete binary tree.
    - An almost-complete binary tree is a binary tree such that every level of the tree has the maximum number of nodes possible except possibly the last level, where its nodes are attached as far left as possible.

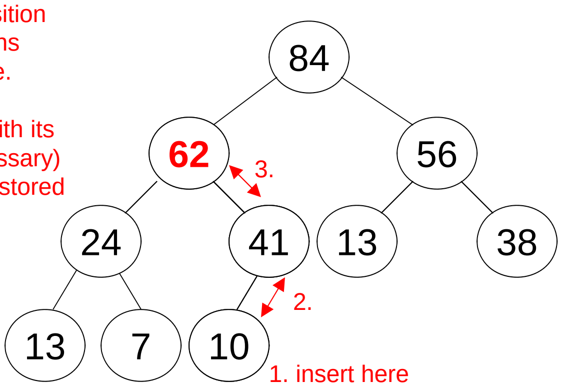
These are not heaps! Why?

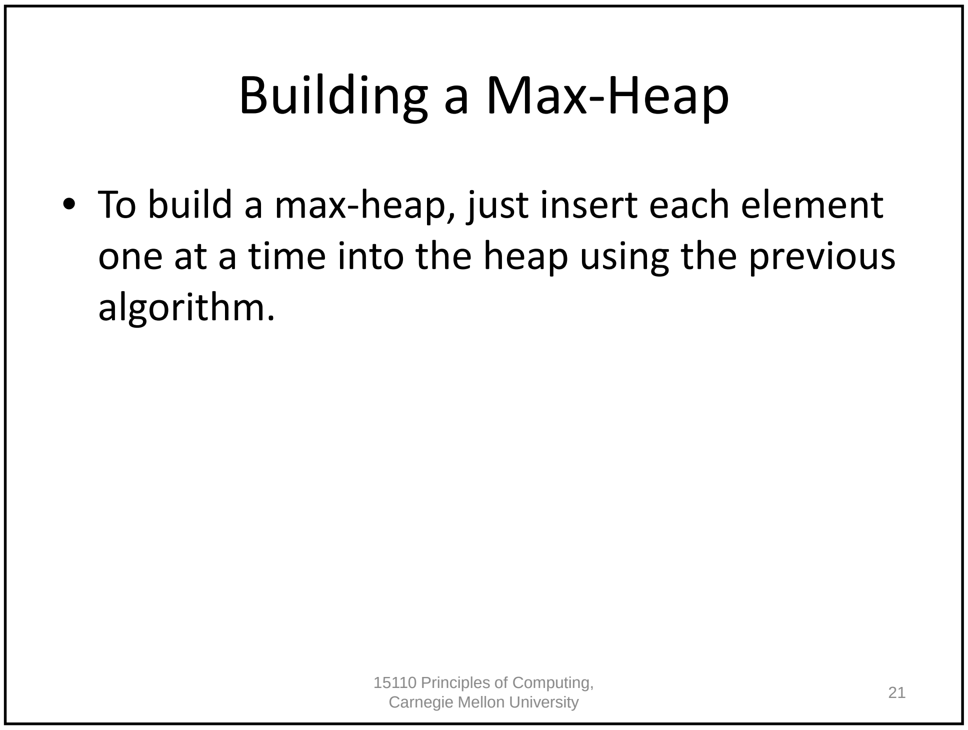


Adding data to a Max-Heap

Insert new data into next available tree position so the tree remains (almost) complete.

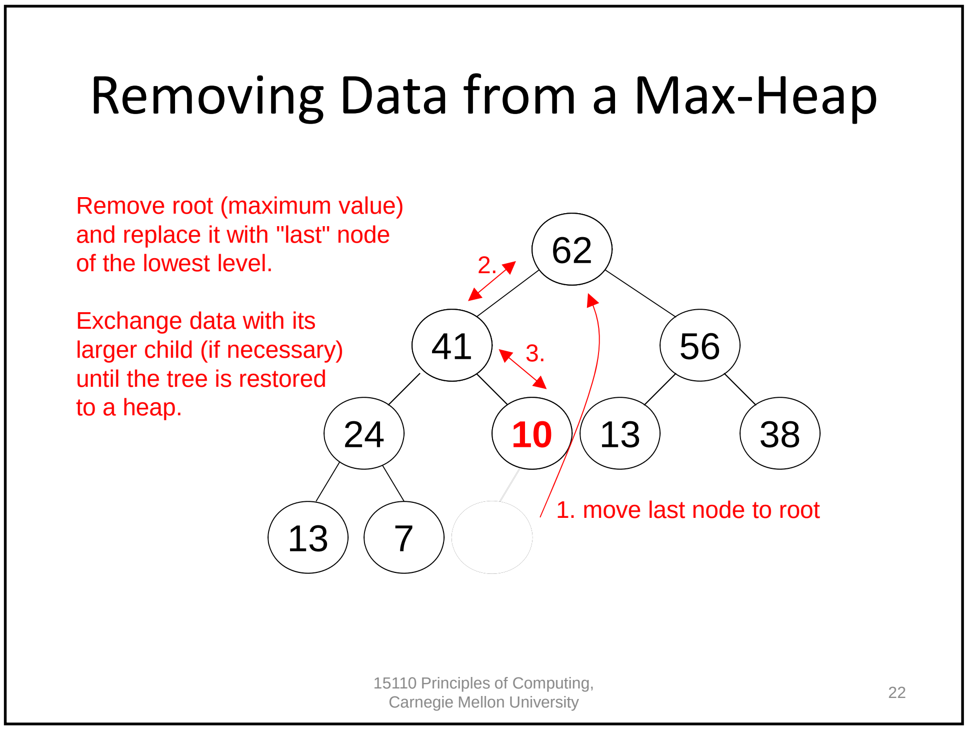
Exchange data with its parent(s) (if necessary) until the tree is restored to a heap.





Building a Max-Heap

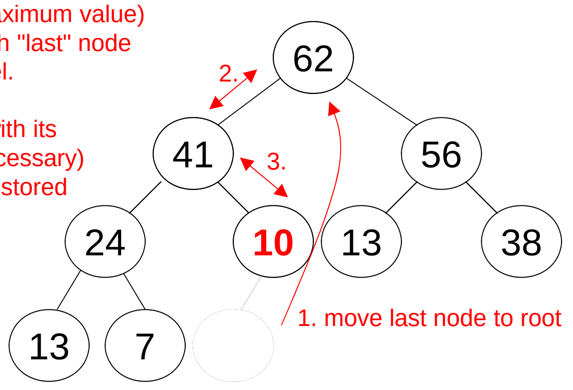
- To build a max-heap, just insert each element one at a time into the heap using the previous algorithm.



Removing Data from a Max-Heap

Remove root (maximum value) and replace it with "last" node of the lowest level.

Exchange data with its larger child (if necessary) until the tree is restored to a heap.



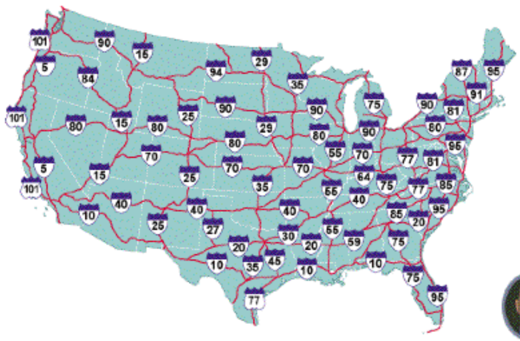
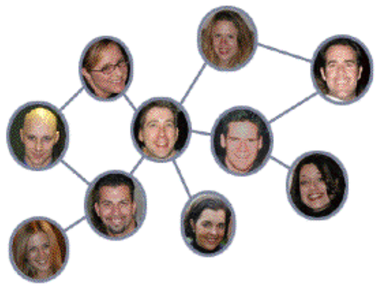
BSTs vs. Max-Heaps

- Which tree is designed for easier searching?
- Which tree is designed for retrieving the maximum value quickly?
- A heap is guaranteed to be "balanced" (complete or almost-complete).
  What about a BST?

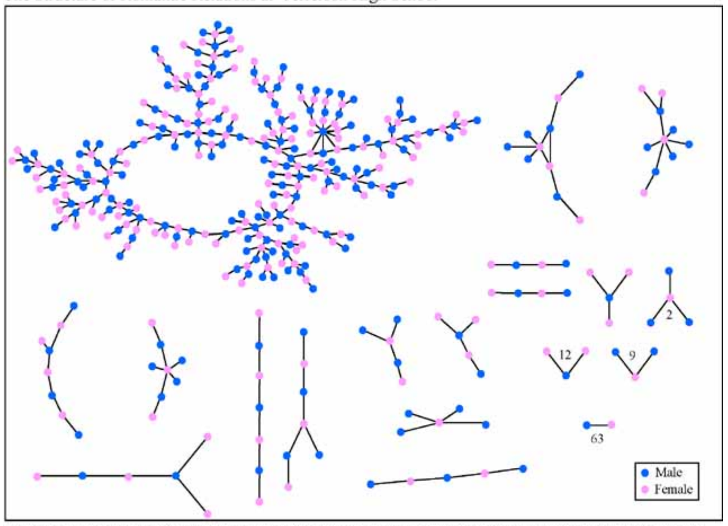
BSTs vs Max-Heaps

- BST with n elements
  - Insert and Search:
    - worst case O(log n) if tree is "balanced"
    - worst case O(n) in general since tree could have one node per level
- Max-Heap with n elements
  - Insert and Remove-Max
    - worst case O(log n) since tree is always "balanced"
  - Find-Max
    - worst case O(1) since max is always at the root

Graphs

The Structure of Romantic Relations at "Jefferson High School"

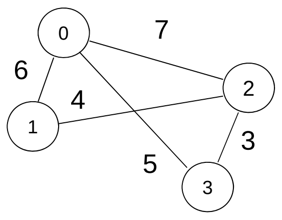


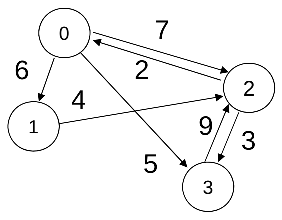
Each circle represents a student and lines connecting students represent romantic relations occuring within the 6 months preceding the interview. Numbers under the figure count the number of times that pattern was observed (i.e. we found 63 pairs unconnected to anyone else).

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





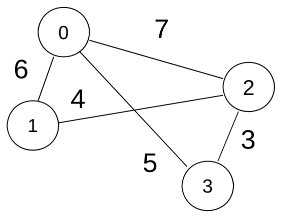
to / from

	0	1	2	3
0	0	6	7	5
1	6	0	4	∞
2	7	4	0	3
3	5	∞	3	0

to / from

	0	1	2	3
0	0	6	7	5
1	∞	0	4	∞
2	2	∞	0	3
3	∞	∞	9	0

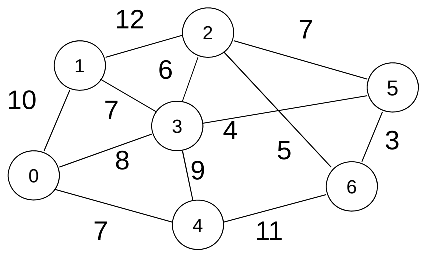
Graphs in Ruby



from \ to	0	1	2	3
0	0	6	7	5
1	6	0	4	∞
2	7	4	0	3
3	5	∞	3	0

```
inf = 1.0/0.0

graph =
[ [ 0, 6, 7, 5 ],
  [ 6, 0, 4, inf ],
  [ 7, 4, 0, 3],
  [ 5, inf, 3, 0] ]
```

An Undirected Weighted Graph



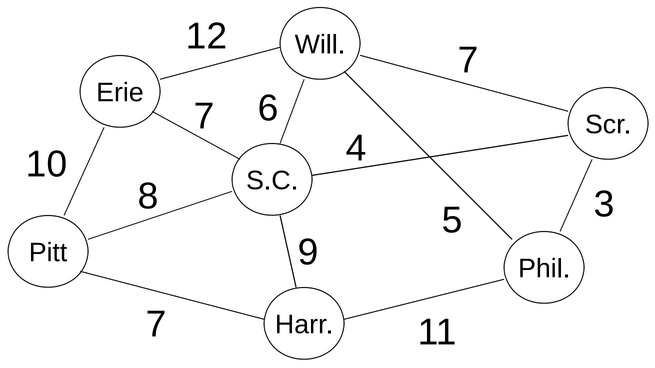
0	1	2	3	4	5	6
Pitt.	Erie	Will.	S.C.	Harr.	Scr.	Phil.

vertices

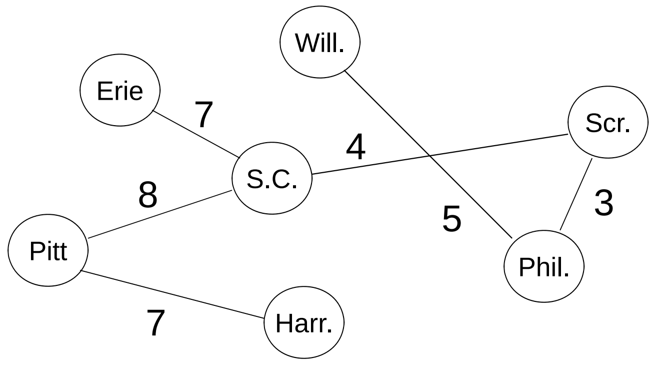
from \ to	0	1	2	3	4	5	6
0	0	10	∞	8	7	∞	∞
1	10	0	12	7	∞	∞	∞
2	∞	12	0	6	∞	7	5
3	8	7	6	0	9	4	∞
4	7	∞	∞	9	0	∞	11
5	∞	∞	7	4	∞	0	3
6	∞	∞	5	∞	11	3	0

edges

Original Graph

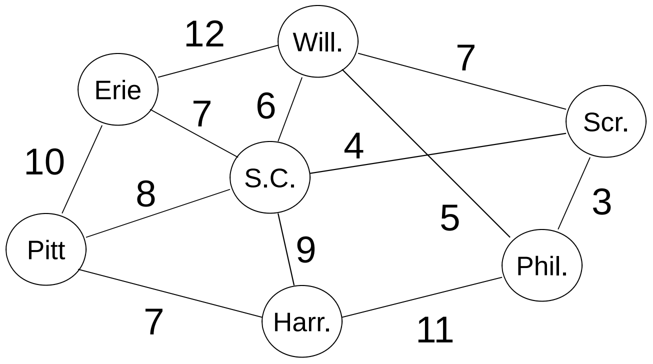


A Minimal Spanning Tree

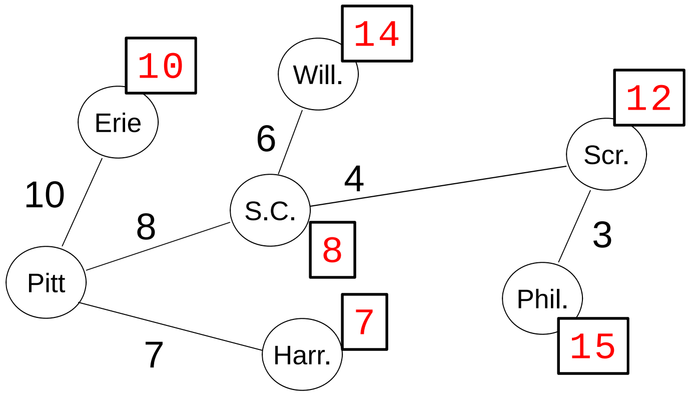


The minimum total cost to connect all vertices using edges from the original graph without using cycles. (minimum total cost = 34)

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.