Trees

Section 11 in the textbook

1

Trees

Definition: A **tree** is a connected undirected graph with no simple circuits.

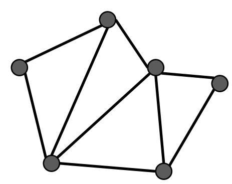
Since a tree cannot have a simple circuit, a tree cannot contain multiple edges or loops. Therefore, any tree must be a **simple graph**.

Theorem: An undirected graph is a tree if and only if there is a **unique simple path** between any of its vertices.

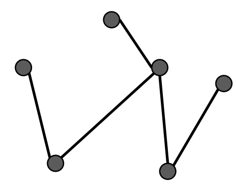
2

Trees

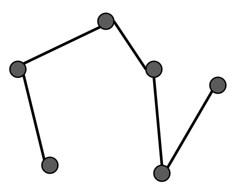
Example: Are the following graphs trees?



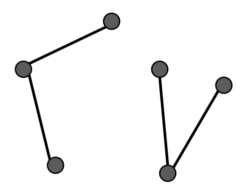
No.



Yes.



Yes.



No.

Trees

Definition: An undirected graph that does not contain simple circuits and is not necessarily connected is called a **forest**.

In general, we use trees to represent **hierarchical structures**.

We often designate a particular vertex of a tree as the **root**. Since there is a unique path from the root to each vertex of the graph, we direct each edge away from the root.

Thus, a tree together with its root produces a **directed graph** called a **rooted tree**.

Characteristic of trees

If T is a graph with n vertices, the following are equivalent:

a) T is a tree
b) T is connected and acyclic (having no cycles)
c) T is connected and has n-1 edges
d) T is acyclic and has n-1 edges

Tree Terminology

If v is a vertex in a rooted tree other than the root, the **parent** of v is the unique vertex u such that there is a directed edge from u to v.

When u is the parent of v, v is called the **child** of u.

Vertices with the same parent are called **siblings**.

The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.

Tree Terminology

The **descendants** of a vertex v are those vertices that have v as an ancestor.

A vertex of a tree is called a **leaf** if it has no children.

Vertices that have children are called **internal vertices**.

If a is a vertex in a tree, then the **subtree** with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

Tree Terminology

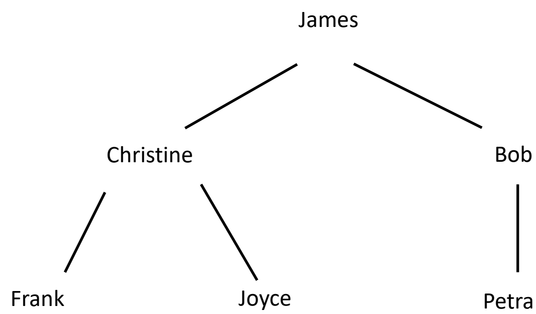
The **level** of a vertex v in a rooted tree is the length of the unique path from the root to this vertex.

The level of the root is defined to be zero.

The **height** of a rooted tree is the maximum of the levels of vertices.

Trees

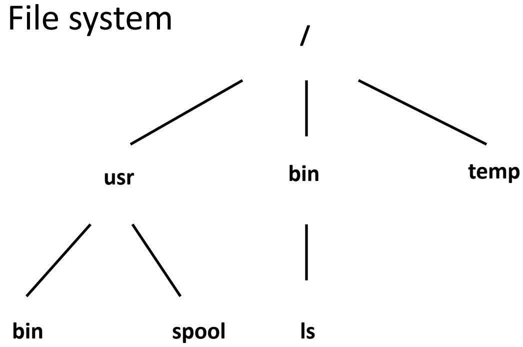
Example I: Family tree



9

Trees

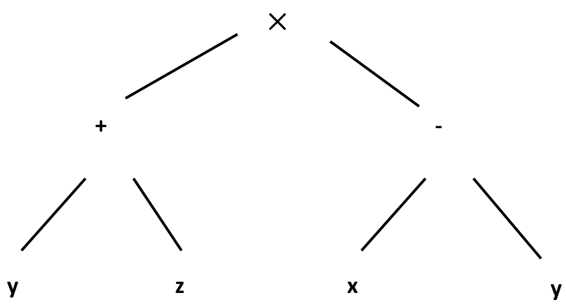
Example II: File system



10

Trees

Example III: Arithmetic expressions



This tree represents the expression $(y + z)\times(x - y)$.

Trees

Definition: A rooted tree is called an **m-ary tree** if every internal vertex has no more than m children.

The tree is called a **full m-ary tree** if every internal vertex has exactly m children.

An m-ary tree with m = 2 is called a **binary tree**.

Theorem: A tree with n vertices has (n – 1) edges.

Theorem: A full m-ary tree with i internal vertices contains n = mi + 1 vertices.

Binary Search Trees

If we want to perform a large number of searches in a particular list of items, it can be worthwhile to arrange these items in a **binary search tree** to facilitate the subsequent searches.

A binary search tree is a binary tree in which each child of a vertex is designated as a **right or left child**, and each vertex is labeled with a **key**, which is one of the items.

When we construct the tree, vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.

13

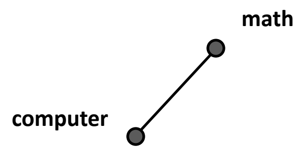
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.

14

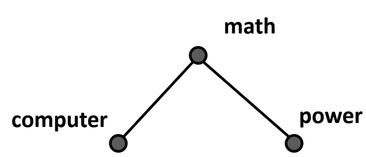
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



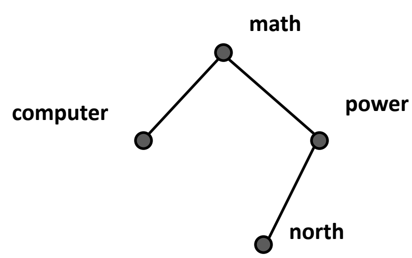
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



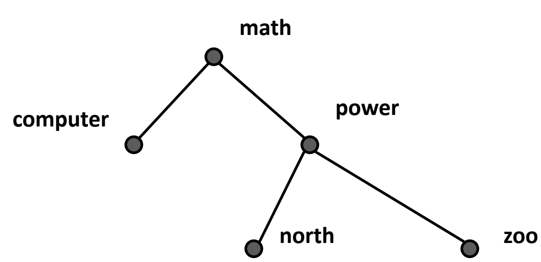
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



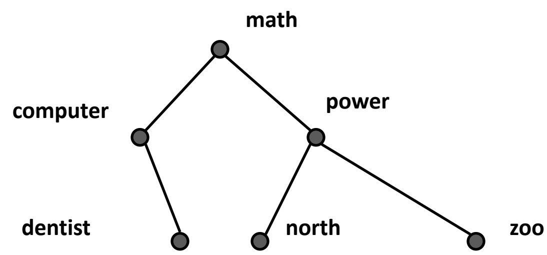
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



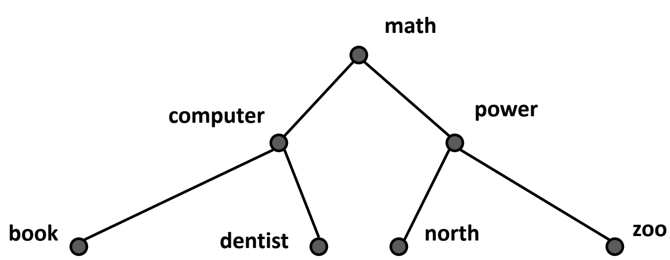
Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



Binary Search Trees

Example: Construct a binary search tree for the strings **math, computer, power, north, zoo, dentist, book**.



Binary Search Trees

To perform a search in such a tree for an item x, we can start at the root and compare its key to x. If x is **less** than the key, we proceed to the **left** child of the current vertex, and if x is **greater** than the key, we proceed to the **right** one.

This procedure is repeated until we either found the item we were looking for, or we cannot proceed any further.

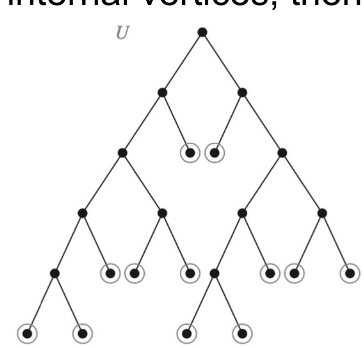
In a balanced tree representing a list of n items, search can be performed with a maximum of $\lceil \log(n + 1) \rceil$ steps (compare with binary search).

Full Binary tree

Definition: A *full* binary tree is a binary tree in which each vertex has two or no children.

Theorem: If T is a *full binary tree* with k internal vertices, then:

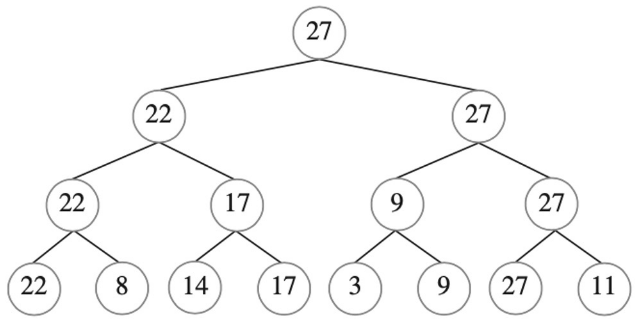
- T has k+1 leaves
- The total vertices is 2k+1



Full Binary tree

Theorem: If a binary tree of height h has t leaves, then $t \leq 2^h$

If all leaves t of a full binary tree T have the same level h = height of T, then $t = 2^h$



Spanning Trees

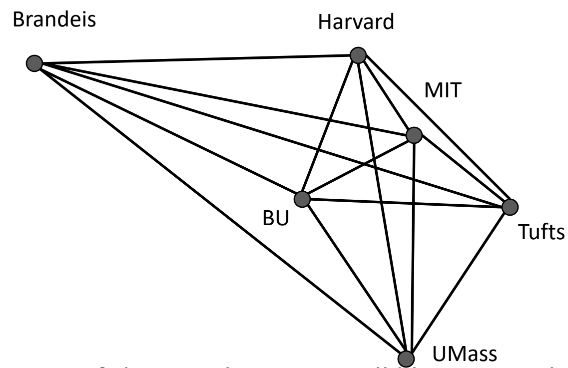
Definition: Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.

Note: A spanning tree of G = (V, E) is a connected graph on V with a minimum number of edges (|V| - 1).

Example: Since winters in Boston can be very cold, six universities in the Boston area decide to build a tunnel system that connects their libraries.

Spanning Trees

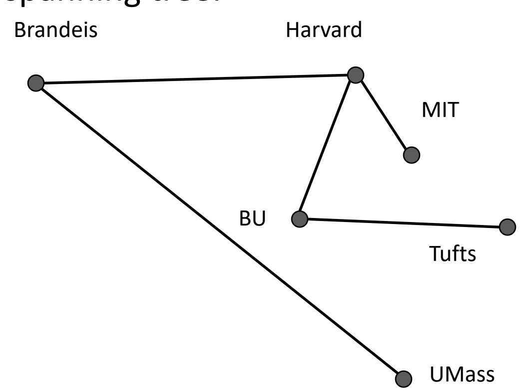
The complete graph including all possible tunnels:



The spanning trees of this graph connect all libraries with a minimum number of tunnels.

Spanning Trees

Example for a spanning tree:



Since there are 6 libraries, 5 tunnels are sufficient to connect all of them.

Spanning Trees

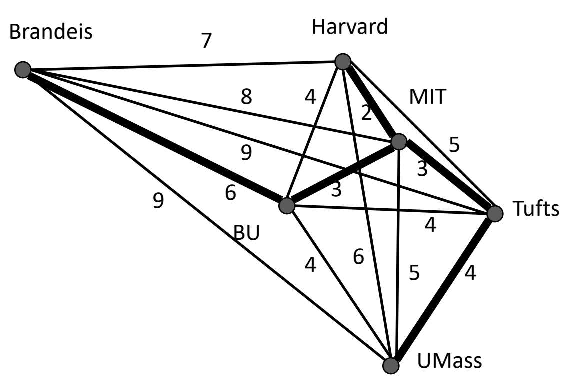
Now imagine that you are in charge of the tunnel project. How can you determine a tunnel system of **minimal cost** that connects all libraries?

Definition: A **minimum spanning tree** in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

How can we find a minimum spanning tree?

Spanning Trees

The complete graph with cost labels (in billion $):



The least expensive tunnel system costs $20 billion.

Prim's algorithm

Step 0: Pick any vertex as a starting vertex (call it *a*). T = {a}.

Step 1: Find the edge with smallest weight incident to *a*. Add it to T Also include in T the next vertex and call it *b*.

Step 2: Find the edge of smallest weight incident to either *a* or *b.* Include in T that edge and the next incident vertex. Call that vertex *c*.

Step 3: Repeat Step 2, choosing the edge of smallest weight that does not form a cycle until all vertices are in T. The resulting subgraph T is a minimum spanning tree.

29

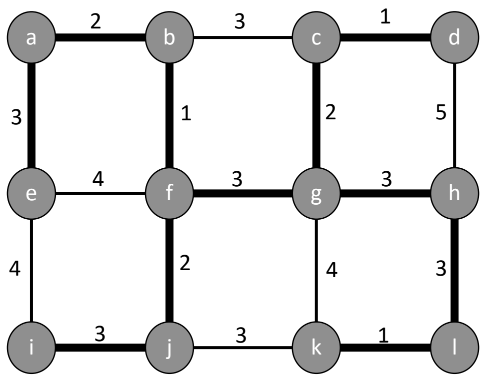
Prim's algorithm

ALGORITHM 1 **Prim's Algorithm.**

procedure *Prim*(G: weighted connected undirected graph with n vertices)
T := a minimum-weight edge
for $i := 1$ **to** $n - 2$
  e := an edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T
  $T := T$ with e added
return T {T is a minimum spanning tree of G}

30

Prim's algorithm



Choice	Edge	Weight
1	{b, f}	1
2	{a, b}	2
3	{f, j}	2
4	{a, e}	3
5	{i, j}	3
6	{f, g}	3
7	{c, g}	2
8	{c, d}	1
9	{g, h}	3
10	{h, l}	3
11	{k, l}	1
		24

UMass Boston

31

Kruskal's algorithm

Step 1: Find the edge in the graph with smallest weight (if there is more than one, pick one at random). Mark it with any given color, say red.

Step 2: Find the next edge in the graph with smallest weight that doesn't close a cycle. Color that edge and the next incident vertex.

Step 3: Repeat Step 2 until you reach out to every vertex of the graph. The chosen edges form the desired minimum spanning tree.

UMass Boston

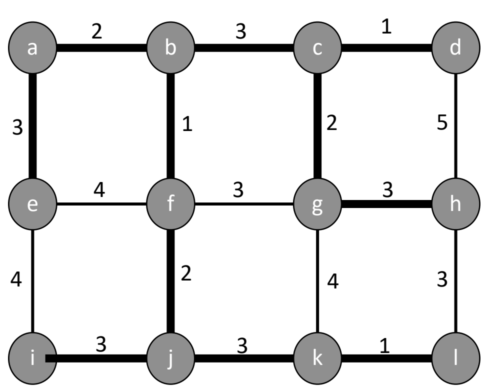
32

Kruskal's algorithm

ALGORITHM 2 Kruskal's Algorithm.

procedure *Kruskal*(G: weighted connected undirected graph with n vertices)
T := empty graph
for i := 1 **to** $n - 1$
  e := any edge in G with smallest weight that does not form a simple circuit
    when added to T
  T := T with e added
return T {T is a minimum spanning tree of G}

Kruskal's algorithm



Choice	Edge	Weight
1	{c, d}	1
2	{b, f}	1
3	{k, l}	1
4	{c, g}	2
5	{a, b}	2
6	{f, j}	2
7	{b, c}	3
8	{j, k}	3
9	{g, h}	3
10	{i, j}	3
11	{a, e}	3
		24

Spanning Trees

Prim vs. Kruskal:

- Both algorithms are **guaranteed** to produce a minimum spanning tree of a connected weighted graph.
- The two algorithms differ in the way they can be implemented and their efficiency under different conditions.
- As a rule of thumb, **Prim's algorithm** is more efficient when initially there are many **more edges** than vertices.
- For graphs with initially only **few edges** in comparison to the number of vertices, **Kruskal's algorithm** typically performs more efficiently.

Tree Traversal

Let T be an ordered rooted tree with root r.

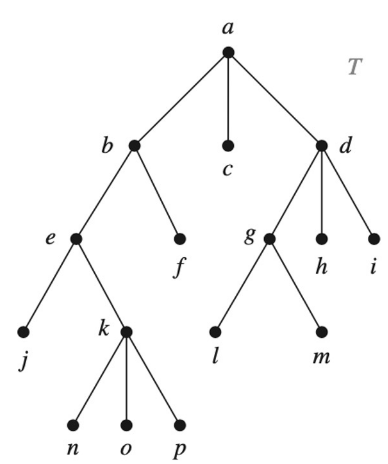
Definition 1: If T consists only of r, then r is the ***preorder traversal*** of T . Otherwise, suppose that $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right in T . The *preorder traversal* begins by visiting r. It continues by traversing T_1 in preorder, then T_2 in preorder, and so on, until T_n is traversed in preorder.

Defintion 2: If T consists only of r, then r is the ***inorder traversal*** of T . Otherwise, suppose that $T_1, T_2 \ldots, T_n$ are the subtrees at r from left to right. The *inorder traversal* begins by traversing T_1 in inorder, then visiting r. It continues by traversing T_2 in inorder, and so on until T_n is traversed in inorder.

Tree Traversal

Definition 3: Let T be an ordered rooted tree with root r. If T consists only of r, then r is the ***postorder traversal*** of T . Otherwise, suppose that $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right. The *postorder traversal* begins by traversing T_1 in postorder, then T_2 in postorder, . . . , then T_n in postorder, and ends by visiting r.

Pre-Order Traversal



Preorder traversal:

a-b-e-j-k-n-o-p-f-c-d-g-l-m-h-I

Inorder traversal:

j-e-n-k-o-p-b-f-a-c-l-g-m-d-h-i

Postorder traversal:

j-n-o-p-k-e-f-b-c-l-m-g-h-i-d-a

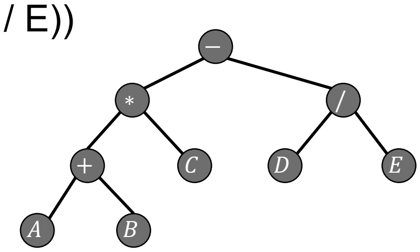
Arithmetic expressions

Standard: *infix* form

(A+B) ∗ C – D/ E

Fully parenthesized form (in-order & parenthesis):

(((A + B) ∗ C) – (D / E))



39

Arithmetic expressions

Prefix form (Polish notation):

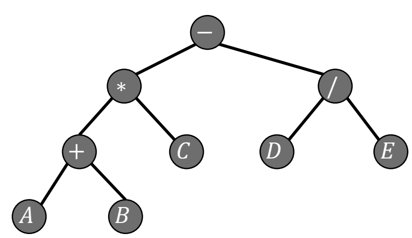
- ∗ + A B C / D E

→ Preorder traversal

Postfix form (reverse Polish notation):

A B + C ∗ D E / -

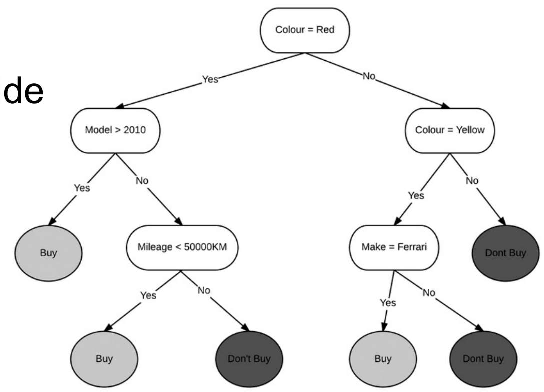
→ Postorder traversal



40

Decision Tree

A *decision tree* is a binary tree containing an algorithm to decide which course of action to take.



41

END

42

Final Exam policy

- **Camera on!** Webcams will be used for proctoring
- Showing Student ID at the start of the exam.
- Please login 15 minutes early (9:45am)
- Private message in the chatbox
- **Exactly 10 minutes** to submit the solution (Gradescope)