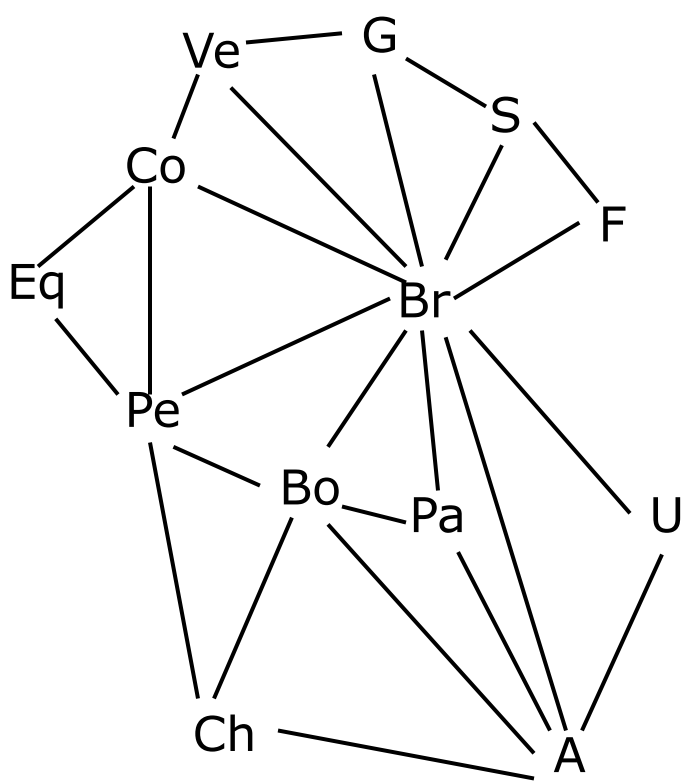
Section 1.4: Graphs and Trees

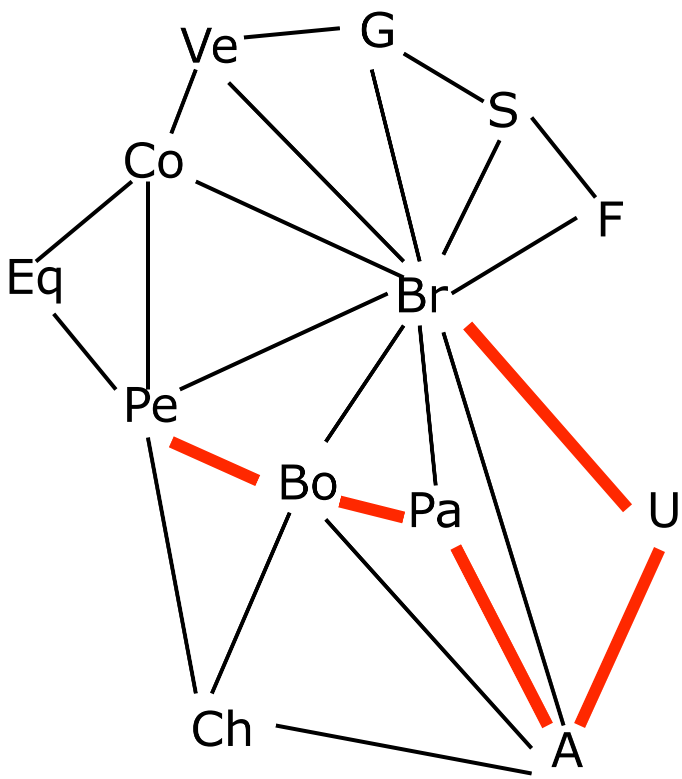
A **graph** is a set of objects (called **vertices** or **nodes**) and **edges** between pairs of nodes.



Vertices = {Ve, G, S, F, Br, Co, Eq, Pe, Bo,Pa, Ch, A, U}
Edges = { {Ve,G}, {Ve,Br}, … }

A **path** from vertex x_0 to x_n is a sequence of edges $x_0, x_1, \ldots, x_n$, where there is an edge from x_{i-1} to x_i for $1 \le i \le n$.

The **length** of a path is the number of edges in it.



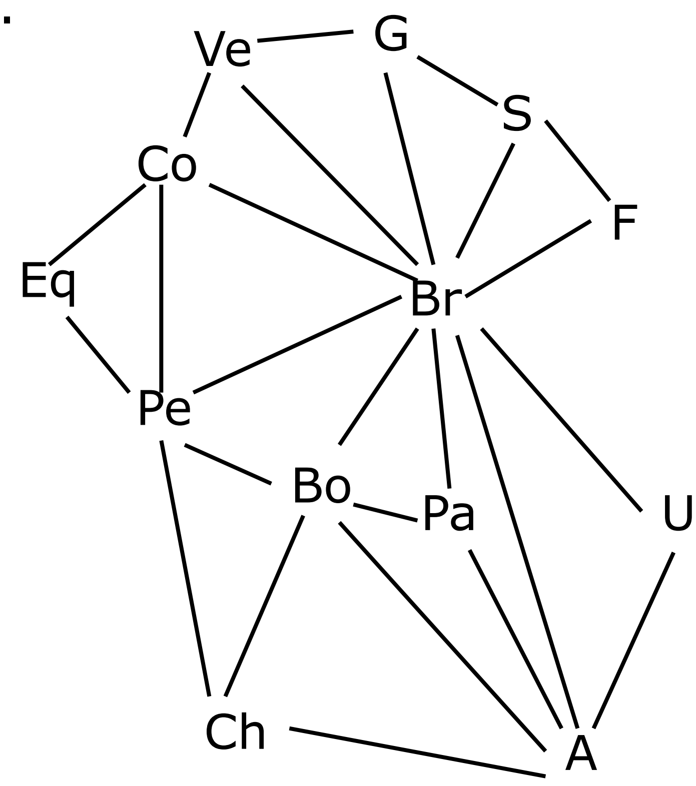
A path from ***Pe*** *to* ***Br***

A **cycle** is a path that begins and ends at the same vertex and has no repeated edges.

The sequence **Co**,**Br**,**G**,**Ve**,**Co** is a cycle.

The sequence **S**,**F**,**S** is not a cycle,
since edge {**S**,**F**} occurs twice.

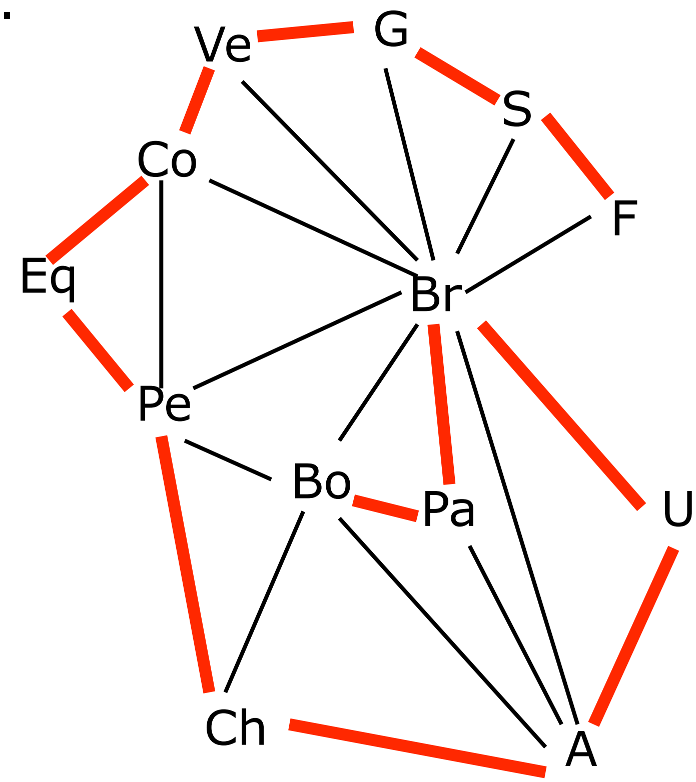
In-class quiz: What is the longest
path from **Bo** to **F**
with distinct edges and no cylces?



The sequence **Co**,**Br**,**G**,**Ve**,**Co** is a cycle.

The sequence **S**,**F**,**S** is not a cycle,
since edge {**S**,**F**} occurs twice.

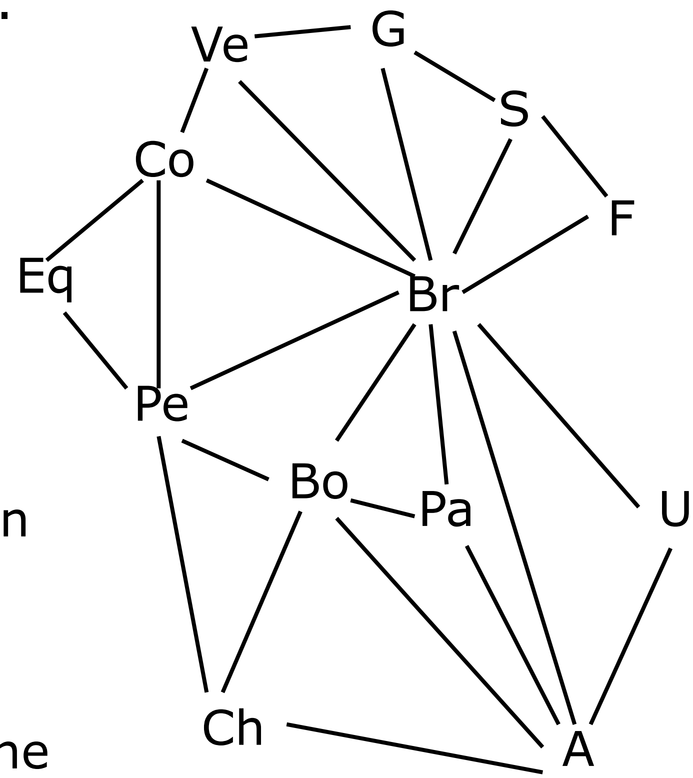
In-class quiz: What is the longest
path from **Bo** to **F**
with distinct edges and no cylces?



The sequence **Co**,**Br**,**G**,**Ve**,**Co** is a cycle.

The sequence **S**,**F**,**S** is not a cycle,
since edge {**S**,**F**} occurs twice.

In-class quiz: What is the longest path from **Bo** to **F** with distinct edges and no cylces?



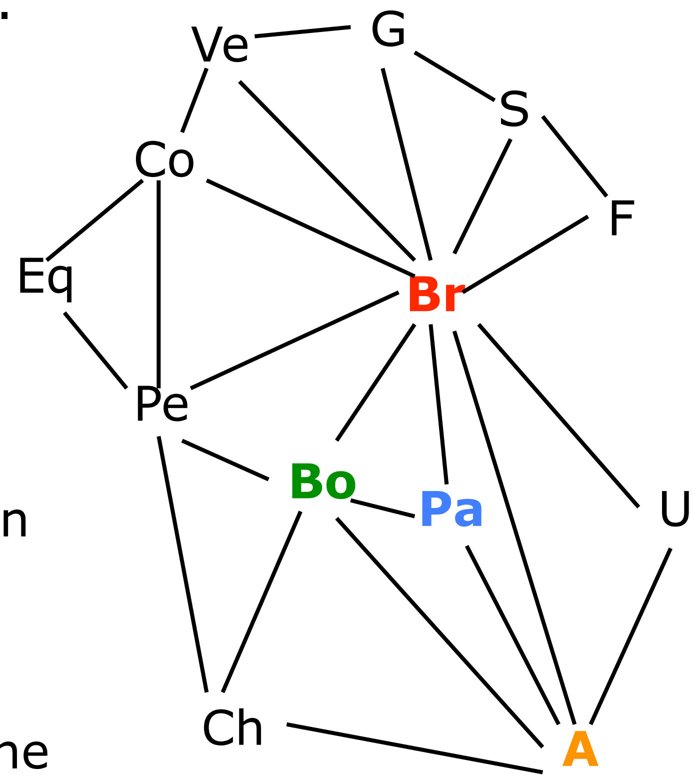
A graph is **n-colorable** if its vertices can be colored using n different colors such that adjacent vertices have different colors.
The **chromatic number** of a graph is the smallest such n.

In-class quiz: What is the chromatic color of this graph?
i.e., how many colors does it take to color this graph?

The sequence **Co**,**Br**,**G**,**Ve**,**Co** is a cycle.

The sequence **S**,**F**,**S** is not a cycle,
since edge {**S**,**F**} occurs twice.

In-class quiz: What is the longest
path from **Bo** to **F**
with distinct edges and no cylces?



A graph is **n-colorable** if its vertices can
be colored using n different colors
such that adjacent vertices have
different colors.
The **chromatic number** of a graph is the
smallest such n.

In-class quiz: What is the chromatic color of this graph?
i.e., how many colors does it take to color this graph?

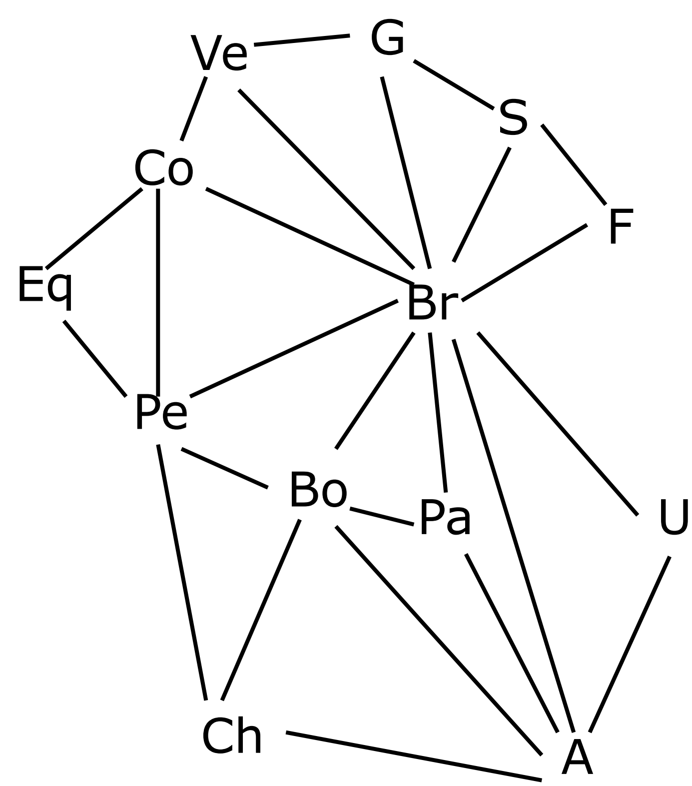
A planar graph can be drawn on a 2-D plane without edges crossing.
Theorem: All planar graphs can be colored with 4 (or fewer) colors.

Graph Traversals

A graph **traversal** starts at some vertex v and visits all vertices without visiting any vertex more than once.
(We assume connectedness: all vertices are reachable from v.)

Breadth-First Traversal

- First visit v.
- Then visit all vertices reachable from v with a path length of 1.
- Then visit all vertices reachable from v with a path length of 2. (… not already visited earlier)
- And so on.

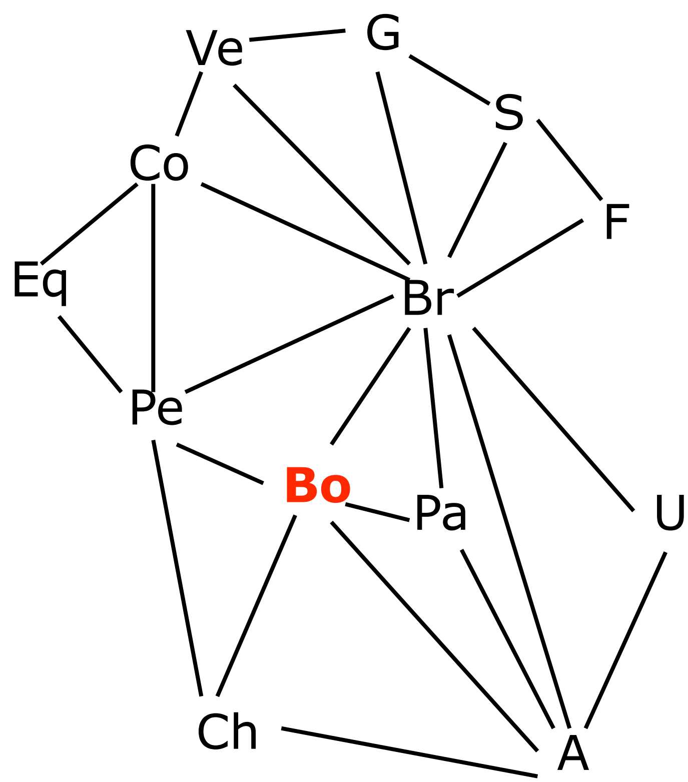


Graph Traversals

A graph **traversal** starts at some vertex v and visits all vertices without visiting any vertex more than once.
(We assume connectedness: all vertices are reachable from v.)

Breadth-First Traversal

- First visit v.
- Then visit all vertices reachable from v with a path length of 1.
- Then visit all vertices reachable from v with a path length of 2. (… not already visited earlier)
- And so on.



Example: v=Bo

Bo

Graph Traversals

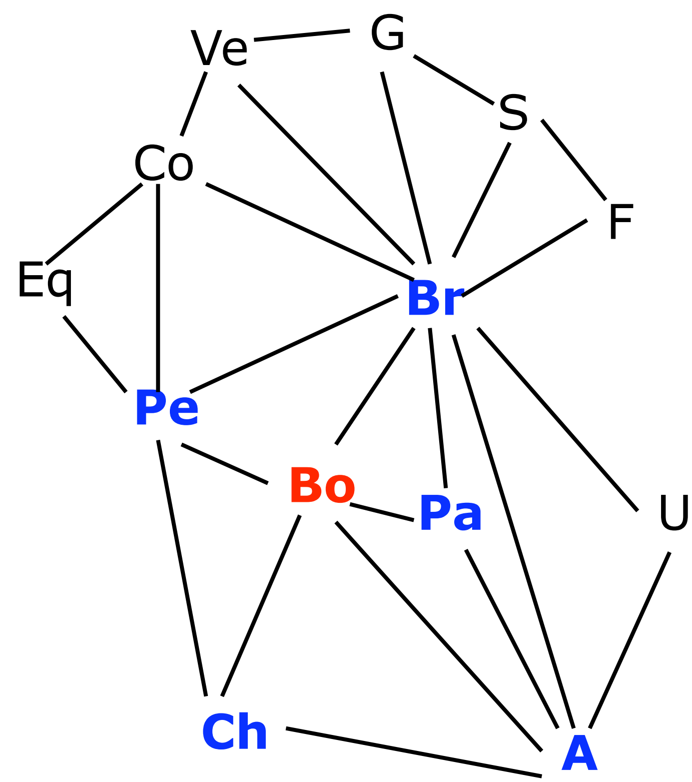
A graph **traversal** starts at some vertex v and visits all vertices without visiting any vertex more than once.
(We assume connectedness: all vertices are reachable from v.)

Breadth-First Traversal

- First visit v.
- Then visit all vertices reachable from v with a path length of 1.
- Then visit all vertices reachable from v with a path length of 2. (… not already visited earlier)
- And so on.

Example: v=Bo

Bo,Pe,Br,Pa,A,Ch

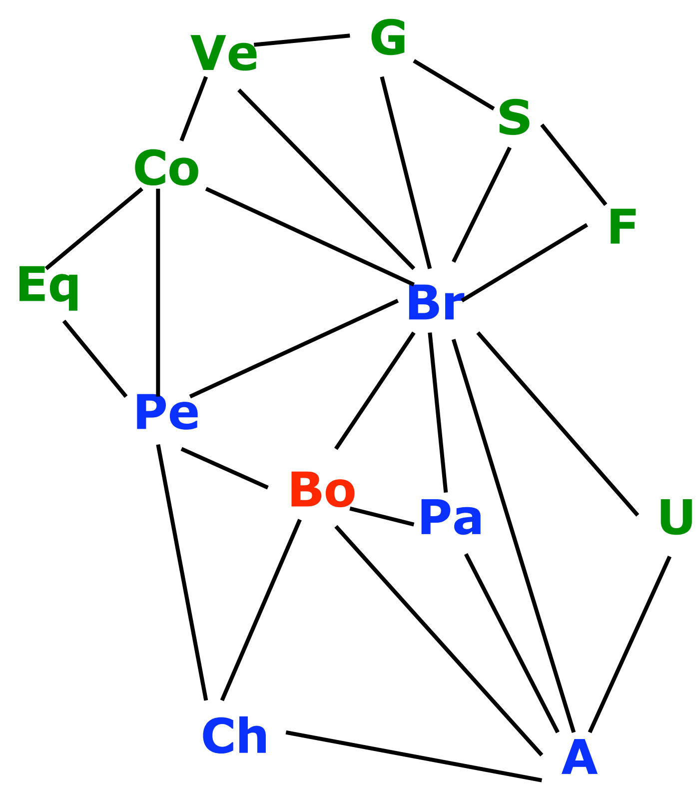


Graph Traversals

A graph **traversal** starts at some vertex v and visits all vertices without visiting any vertex more than once.
(We assume connectedness: all vertices are reachable from v.)

Breadth-First Traversal

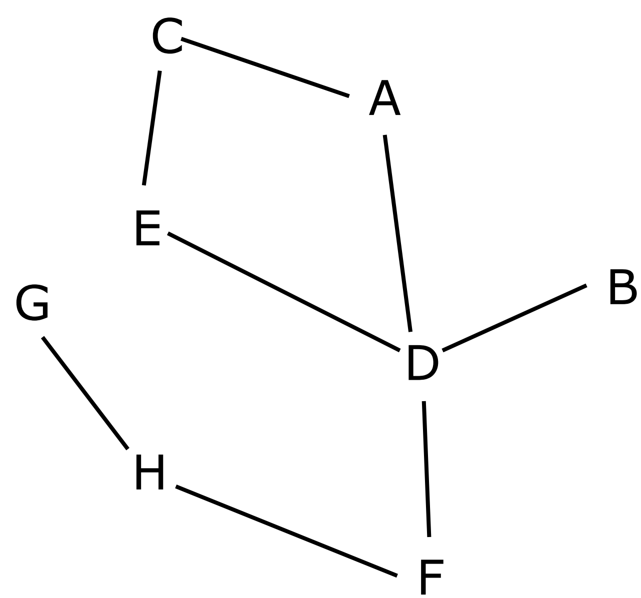
- First visit v.
- Then visit all vertices reachable from v with a path length of 1.
- Then visit all vertices reachable from v with a path length of 2. (… not already visited earlier)
- And so on.



Example: v=Bo

Bo,Pe,Br,Pa,A,Ch,U,Eq,Ve,S,G,F,Co

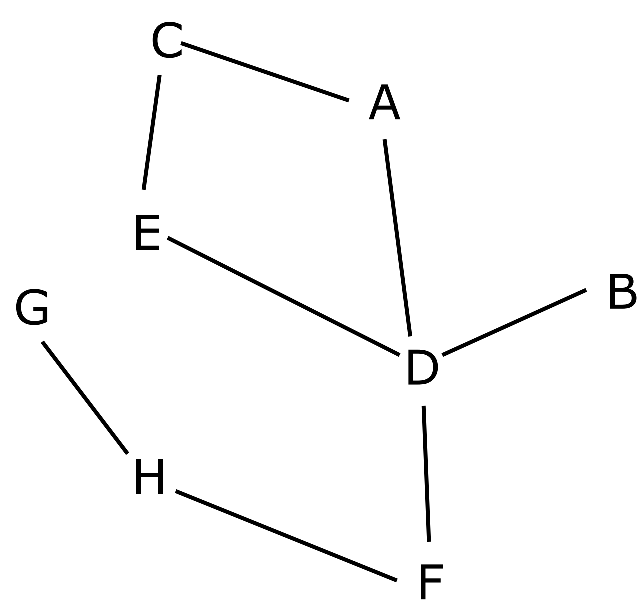
In-Class Quiz: Find a breadth-first traversal starting with F.



In-Class Quiz: Find a breadth-first traversal starting with F.

One answer: F,H,D,G,B,A,E,C

In-Class Quiz: Find a breadth-first traversal starting with C.

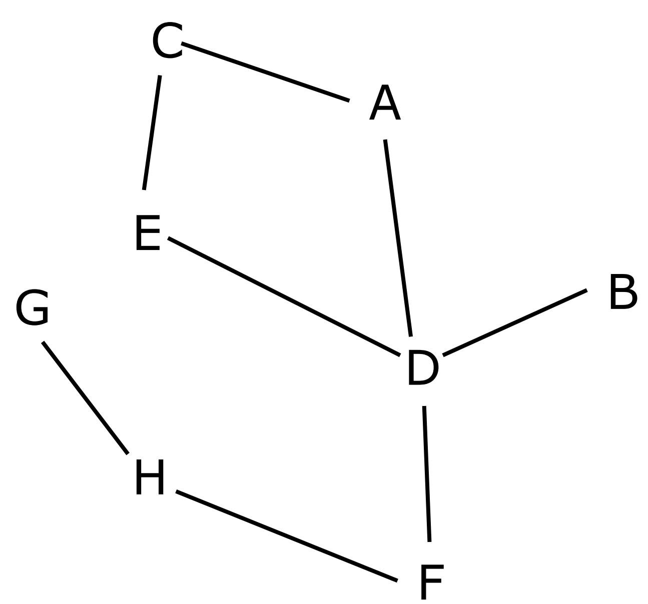


In-Class Quiz: Find a breadth-first traversal starting with F.

One answer: F,H,D,G,B,A,E,C

In-Class Quiz: Find a breadth-first traversal starting with C.

One answer: C,A,E,D,F,B,H,G

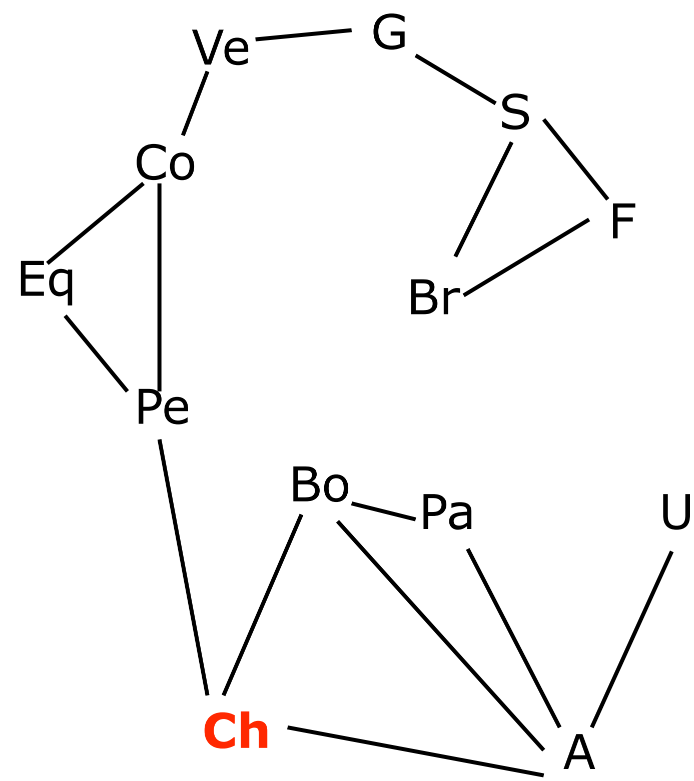


Depth-First Traversal

Start with a vertex v and visit all reachable vertices.
Start by going as far as you can.
Then backup a little and go down another path as far as possible.
Only backup as far as necessary, then try the next path.

Example: Start at Ch.

Ch

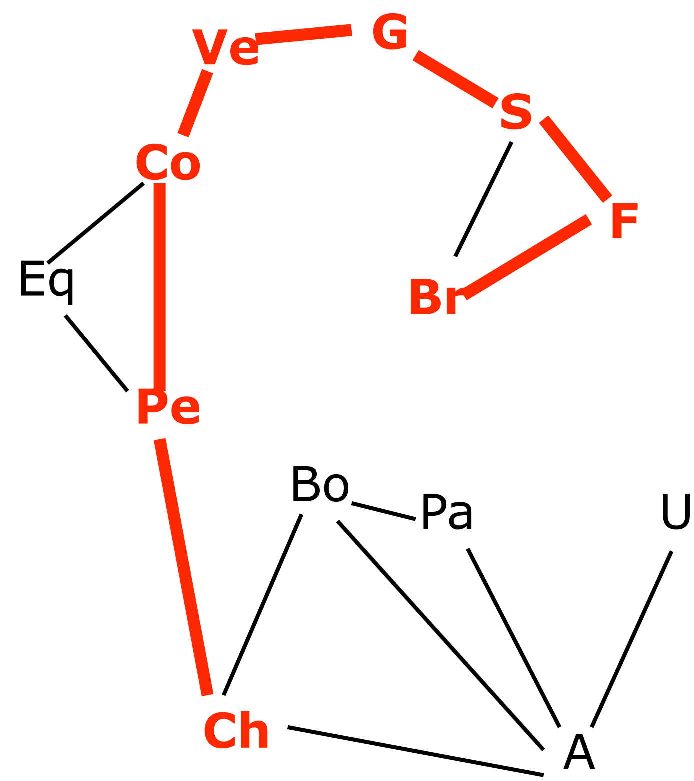


Depth-First Traversal

Start with a vertex v and visit all reachable vertices.
Start by going as far as you can.
Then backup a little and go down another path as far as possible.
Only backup as far as necessary, then try the next path.

Example: Start at Ch.

Ch,Pe,Co,Ve,G,S,F,Br

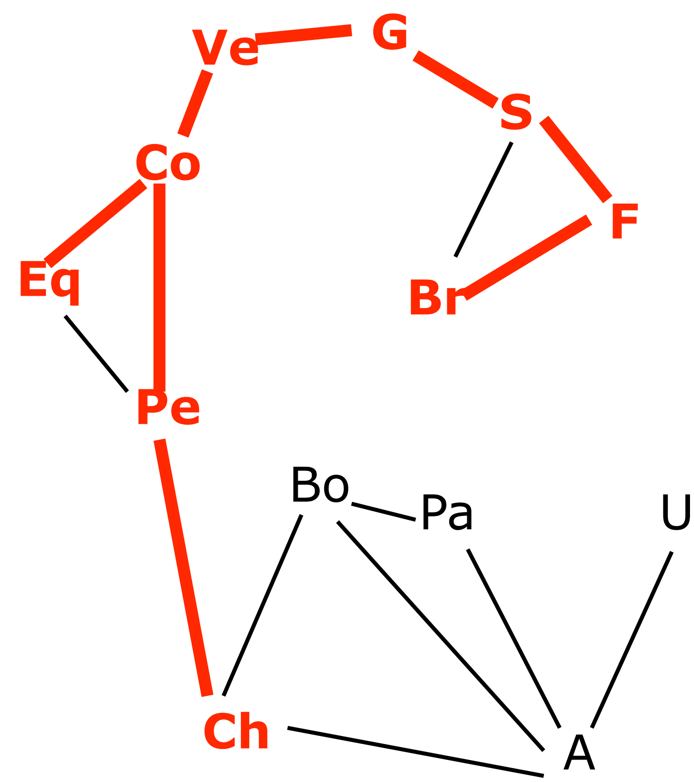


Depth-First Traversal

Start with a vertex v and visit all reachable vertices.
Start by going as far as you can.
Then backup a little and go down another path as far as possible.
Only backup as far as necessary, then try the next path.

Example: Start at Ch.

Ch,Pe,Co,Ve,G,S,F,Br,Eq

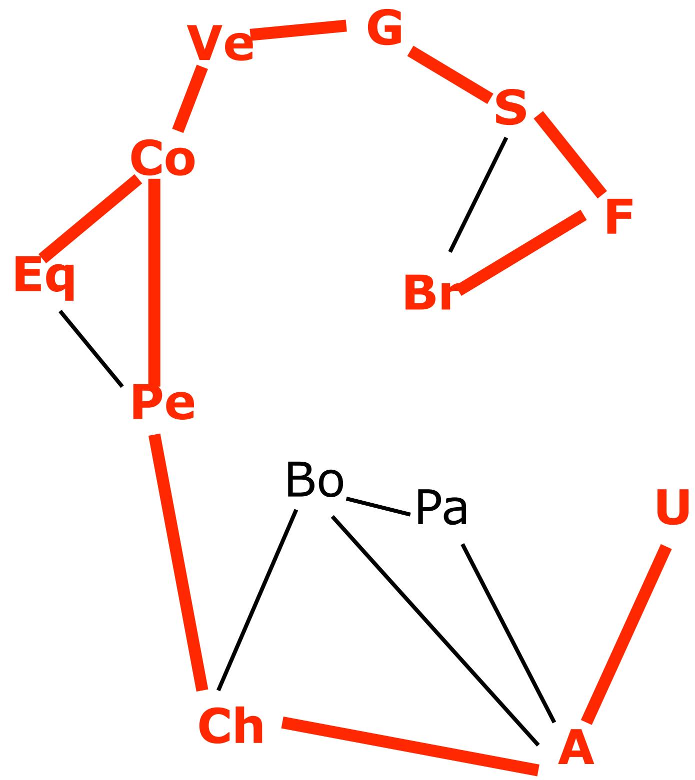


Depth-First Traversal

Start with a vertex v and visit all reachable vertices.
Start by going as far as you can.
Then backup a little and go down another path as far as possible.
Only backup as far as necessary, then try the next path.

Example: Start at Ch.

Ch,Pe,Co,Ve,G,S,F,Br,Eq,A,U

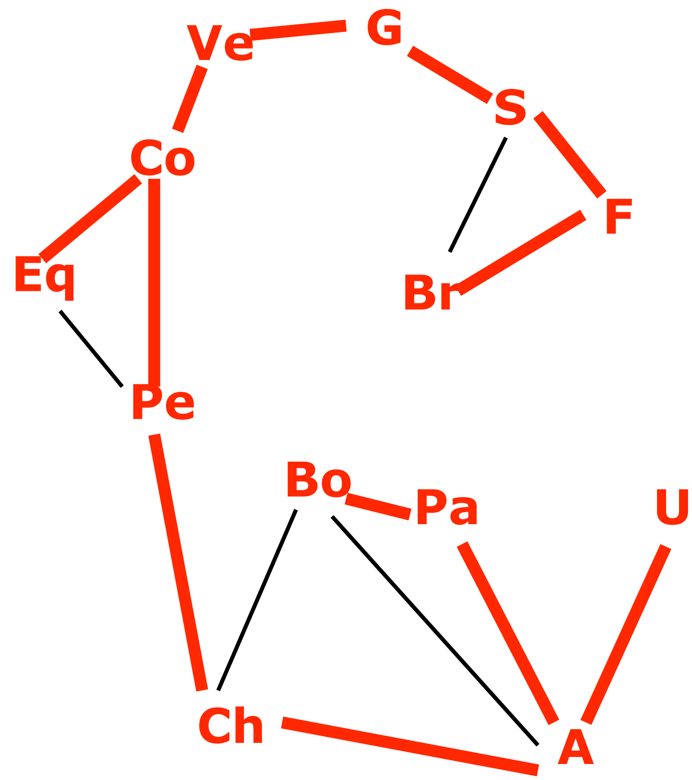


Depth-First Traversal

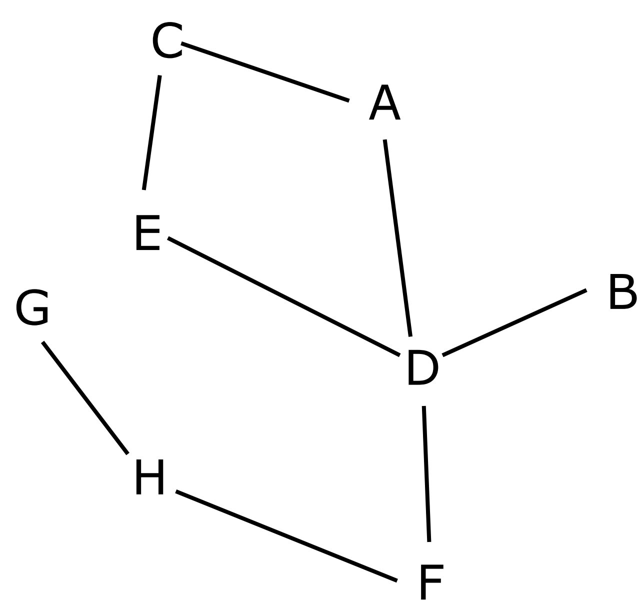
Start with a vertex v and visit all reachable vertices.
Start by going as far as you can.
Then backup a little and go down another path as far as possible.
Only backup as far as necessary, then try the next path.

Example: Start at Ch.

Ch,Pe,Co,Ve,G,S,F,Br,Eq,A,U,Pa,Bo



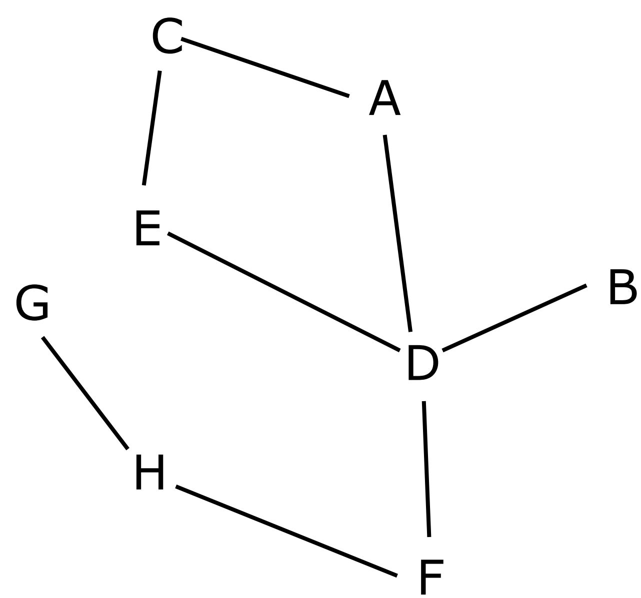
In-Class Quiz: Find a depth-first traversal starting a F.



In-Class Quiz: Find a depth-first traversal starting a F.

One Answer: F,H,G,D,B,A,C,E

In-Class Quiz: Find a depth-first traversal starting a E.

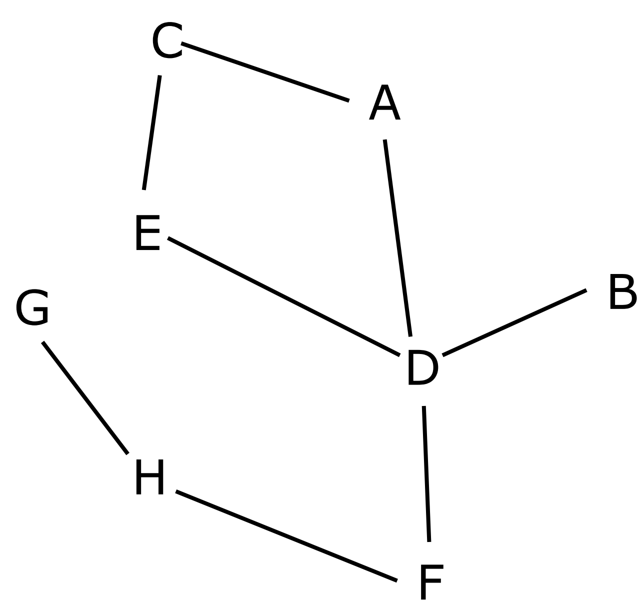


In-Class Quiz: Find a depth-first traversal starting a F.

One Answer: F,H,G,D,B,A,C,E

In-Class Quiz: Find a depth-first traversal starting a E.

One Answer: E,D,F,H,G,A,C,B



An algorithm to visit vertices in depth-first order

visit(v) – This function should be called when a vertex is first visited.

The function being defined is "D".
Recursive: D calls itself

```
D(v):
    if v has not been visited then
        visit(v)
        for each edge from v to x
            D(x)
        endFor
    endIf
```

Trees

A tree is a special kind of graph

- **Connected** – a path between any two nodes
- No cycles

Trees are drawn “upside down”

Root – the node at the top; Every tree has exactly one root.

Parent / **Children** – The parent is immediately above its children

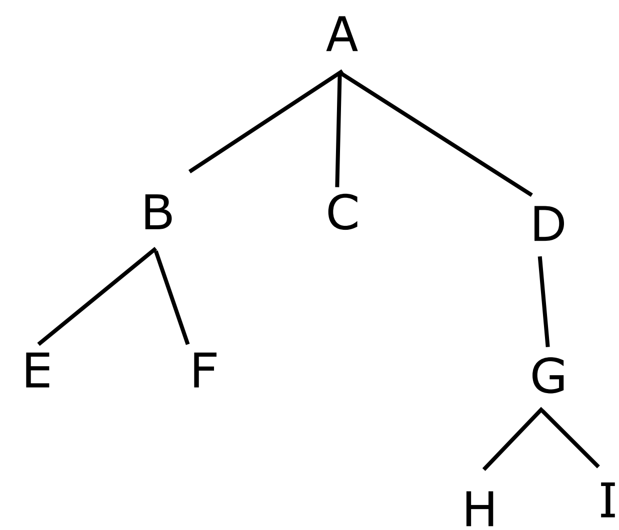
Leaves – Nodes without children

Height (or **depth**) of the tree

- – length of longest path from root to some leaf.

Example:

- Which node is the root?
- What are the children of A?
- Who is the parent of node G?
- Which nodes are leaves?
- What is the depth of this tree?



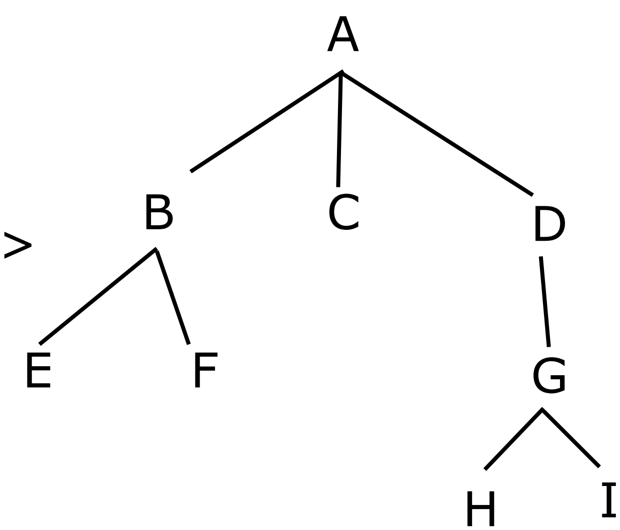
Subtrees

Any node in a tree is the root of a subtree.

Representing Trees with Lists

One way to represent a tree is as a list whose head is the root of the tree anad whose tail is a list of subtrees. Each subtree is represented the same way.

<A, xxx, yyy, zzz>
    where
        xxx = <B, <E>, <F>>
        yyy = <C>
        zzz = <D,<G, <H>, <I>>>
<A,<B,<E>,<F>>,<C>,<D,<G,<H>,<I>>>>

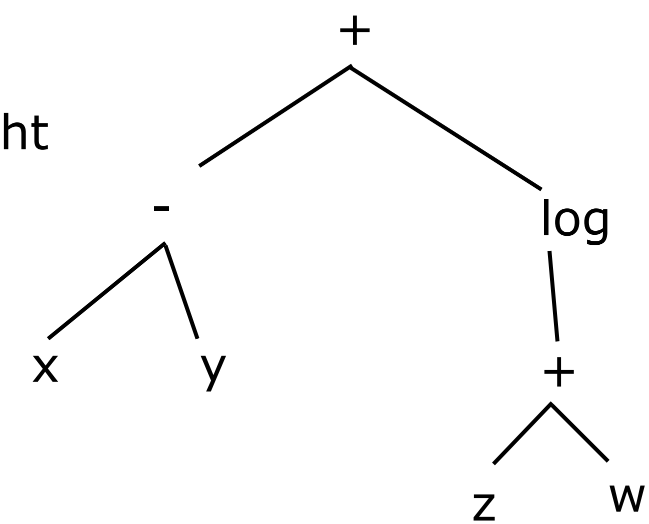


Representing Expressions with Trees

Any algebraic expression can be represented with a tree.

Example: (x-y) + log(z+w)

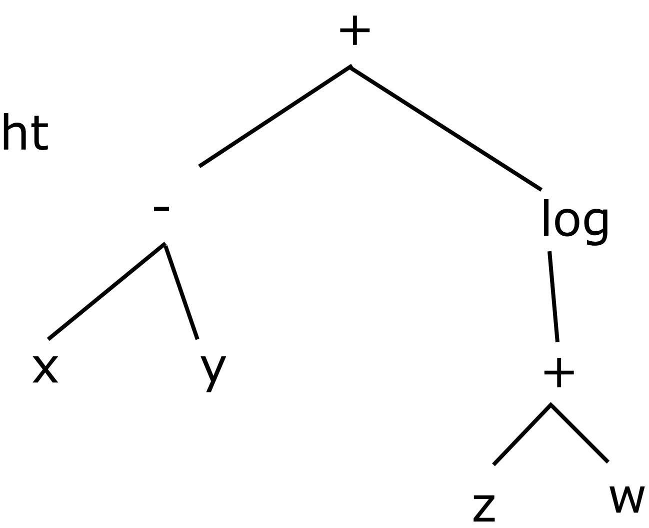
In-class quiz: Find a depth-first, left-to-right traversal of this tree.



Representing Expressions with Trees

Any algebraic expression can be represented with a tree.

Example: (x-y) + log(z+w)



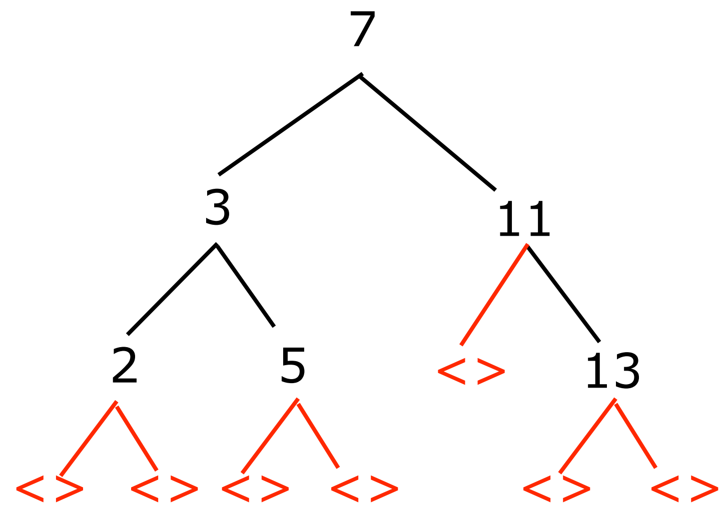
In-class quiz: Find a depth-first, left-to-right traversal of this tree.

+ - x y log + z w

This is the **prefix form** of the expression.

Note: Parentheses are never needed in a prefix-form expression.

Binary Trees



Each vertex either…

    is empty, denoted <>

    has two subtees that are binary trees.

        Left subtree, **right** subtree

Alternately: nodes have ≤2 children.

Representing binary trees with tuples

    empty: <>

    non-empty: <L,x,R>

        where x is the subtree’s root, L and R are the two subtrees.

    A node with no children (a leaf): <<>,2,<>>

    A node with two children: <<<>,2,<>>,3, <<>,5,<>>>

A **binary search tree** represents ordered information.

    The predecessors of x are in the left subtree of x.

    The successors of x are in the right subtree of x.

Example: This is a binary search tree for the first 6 prime numbers.

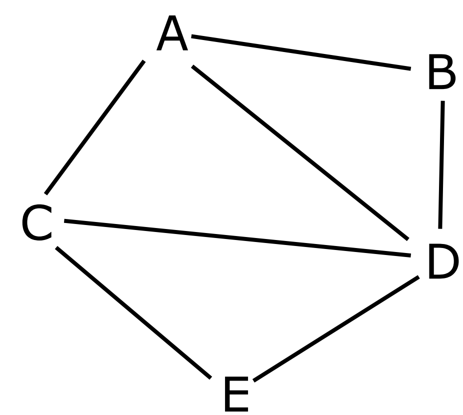
Spanning Trees

A spanning tree for a connected graph is a tree whose nodes are the nodes of the graph and whose edges are a subset of the edges of the graph.

A **weighted graph**: Each edge has an associated value, its weight.

A **minimal spanning tree** is a spanning tree that minimizes the weights on the edges in the tree.

Prim's Algorithm:



Let V be the set of vertices in the graph
Compute S = the set of edges in the
spanning tree
W = a variable, a set of vertices reached

Initialize S := Ø
Pick any v in V. Set W := {v}
<u>while</u> W≠V
Find a minimum weight edge {x,y}, where x∈W and y∈V-W
S := S ∪ {{x,y}}
W := W ∪ {y}
<u>endWhile</u>

Spanning Trees

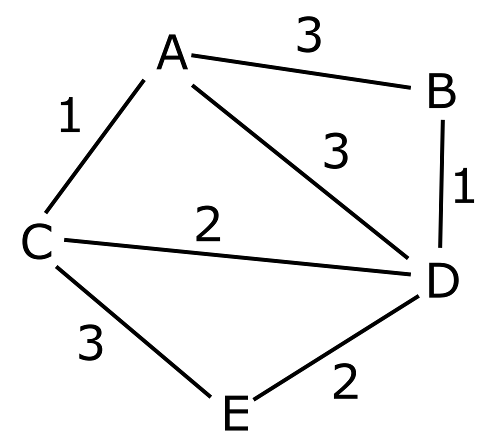
A spanning tree for a connected graph is a tree whose nodes are the nodes of the graph and whose edges are a subset of the edges of the graph.

A **weighted graph**: Each edge has an associated value, its weight.

A **minimal spanning tree** is a spanning tree that minimizes the weights on the edges in the tree.

Prim's Algorithm:

Let V be the set of vertices in the graph
Compute S = the set of edges in the
spanning tree
W = a variable, a set of vertices reached
Initialize S := Ø
Pick any v in V. Set W := {v}
while W≠V
Find a minimum weight edge {x,y}, where x∈W and y∈V-W
S := S ∪ {{x,y}}
W := W ∪ {y}
endWhile



Spanning Trees

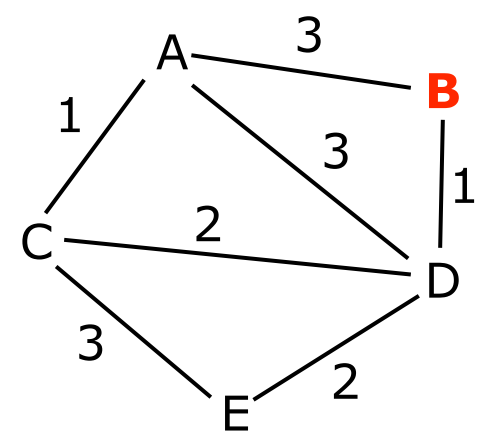
A spanning tree for a connected graph is a tree whose nodes are the nodes of the graph and whose edges are a subset of the edges of the graph.

A **weighted graph**: Each edge has an associated value, its weight.

A **minimal spanning tree** is a spanning tree that minimizes the weights on the edges in the tree.

Prim's Algorithm:

Let V be the set of vertices in the graph
Compute S = the set of edges in the
spanning tree
W = a variable, a set of vertices reached
Initialize S := Ø
Pick any v in V. Set W := {v}
while W≠V
Find a minimum weight edge {x,y}, where x∈W and y∈V-W
S := S ∪ {{x,y}}
W := W ∪ {y}
endWhile



W = { **B** }
S = { }

Spanning Trees

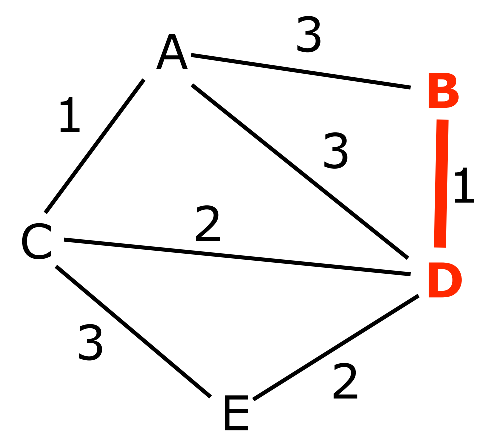
A spanning tree for a connected graph is a tree whose nodes are the nodes of the graph and whose edges are a subset of the edges of the graph.

A **weighted graph**: Each edge has an associated value, its weight.

A **minimal spanning tree** is a spanning tree that minimizes the weights on the edges in the tree.

Prim's Algorithm:

Let V be the set of vertices in the graph
Compute S = the set of edges in the
spanning tree
W = a variable, a set of vertices reached
Initialize S := Ø
Pick any v in V. Set W := {v}
while $W \neq V$
Find a minimum weight edge {x,y}, where $x \in W$ and $y \in V-W$
$S := S \cup \{\{x,y\}\}$
$W := W \cup \{y\}$
endWhile



W = { **B**, **D** }
S = { **BD** }

Spanning Trees

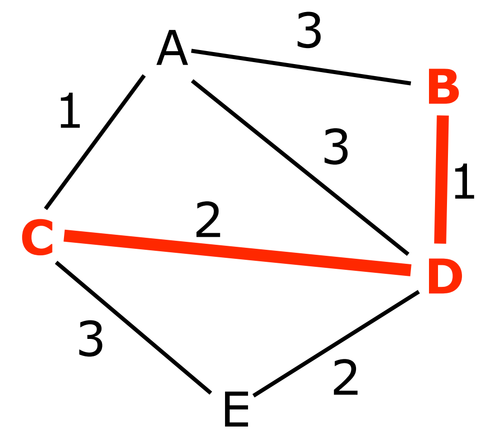
A spanning tree for a connected graph is a tree whose nodes are the nodes of the graph and whose edges are a subset of the edges of the graph.

A **weighted graph**: Each edge has an associated value, its weight.

A **minimal spanning tree** is a spanning tree that minimizes the weights on the edges in the tree.

Prim's Algorithm:

Let V be the set of vertices in the graph
Compute S = the set of edges in the spanning tree
W = a variable, a set of vertices reached
Initialize S := Ø
Pick any v in V. Set W := {v}
while W≠V
Find a minimum weight edge {x,y}, where x∈W and y∈V-W
S := S ∪ {{x,y}}
W := W ∪ {y}
endWhile



W = { **B**, **D**, **C** }
S = { **BD**, **CD** }

Spanning Trees

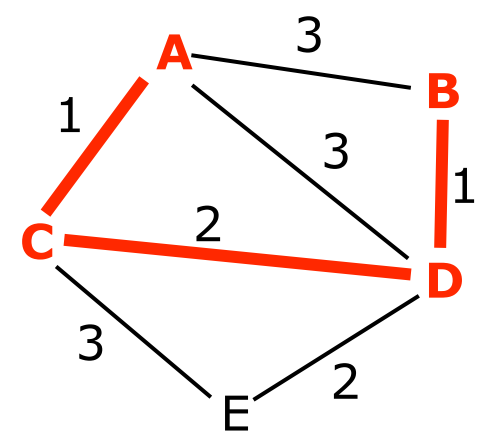
A spanning tree for a connected graph is a tree whose nodes are the nodes of the graph and whose edges are a subset of the edges of the graph.

A **weighted graph**: Each edge has an associated value, its weight.

A **minimal spanning tree** is a spanning tree that minimizes the weights on the edges in the tree.

Prim's Algorithm:

Let V be the set of vertices in the graph
Compute S = the set of edges in the spanning tree
W = a variable, a set of vertices reached



Initialize S := Ø
Pick any v in V. Set W := {v}
while W≠V
Find a minimum weight edge {x,y}, where x∈W and y∈V-W
S := S ∪ {{x,y}}
W := W ∪ {y}
endWhile

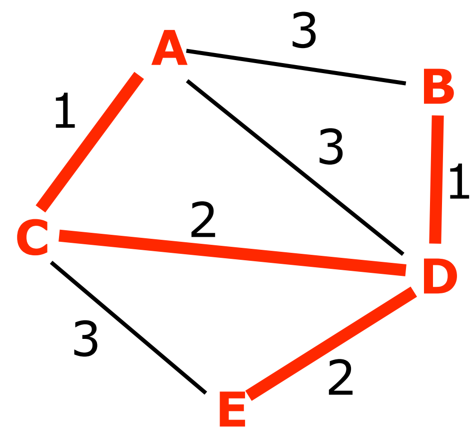
W = { **B**, **D**, **C**, **A** }
S = { **BD**, **CD**, **AC** }

Spanning Trees

A spanning tree for a connected graph is a tree whose nodes are the nodes of the graph and whose edges are a subset of the edges of the graph.

A **weighted graph**: Each edge has an associated value, its weight.

A **minimal spanning tree** is a spanning tree that minimizes the weights on the edges in the tree.



Prim's Algorithm:

    Let V be the set of vertices in the graph
    Compute S = the set of edges in the
        spanning tree
    W = a variable, a set of vertices reached
Initialize S := Ø
Pick any v in V. Set W := {v}
<u>while</u> W≠V
    Find a minimum weight edge {x,y}, where x∈W and y∈V-W
    S := S ∪ {{x,y}}
    W := W ∪ {y}
<u>endWhile</u>

W = { **B**, **D**, **C**, **A**, **E**}
S = { **BD**, **CD**, **AC**, **DE**}