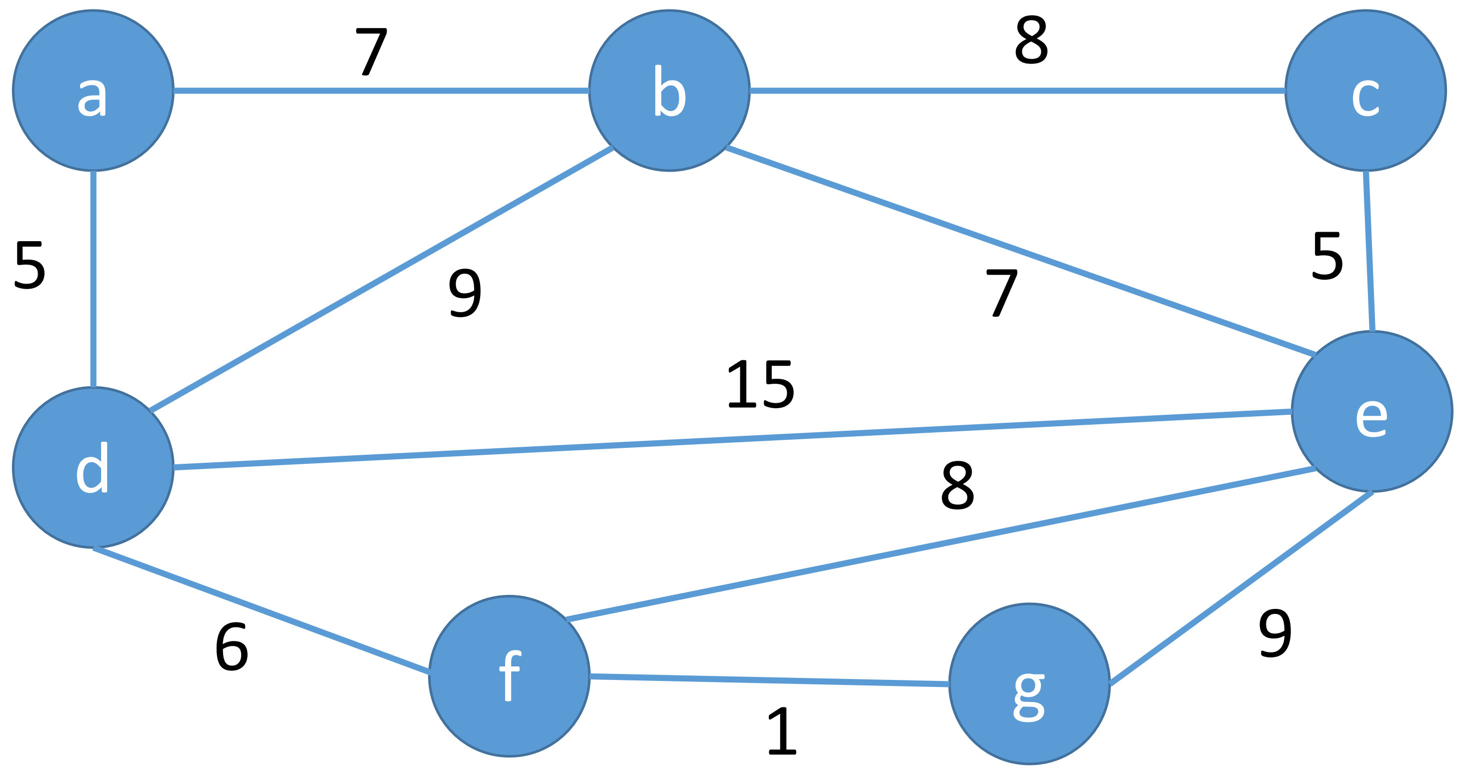
CSIT113
Problem Solving

Workshop - Week 10

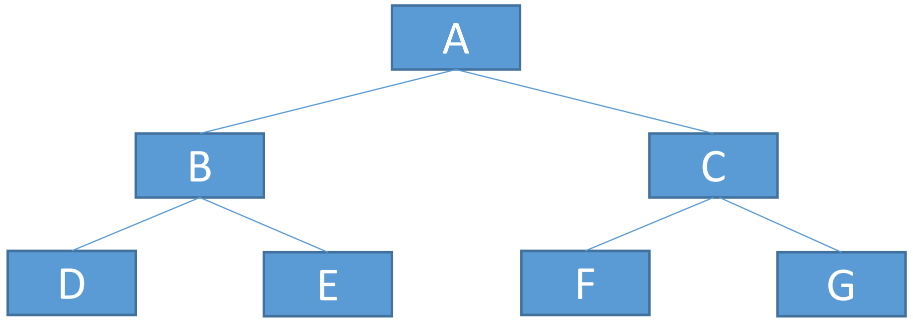
Minimal spanning Trees

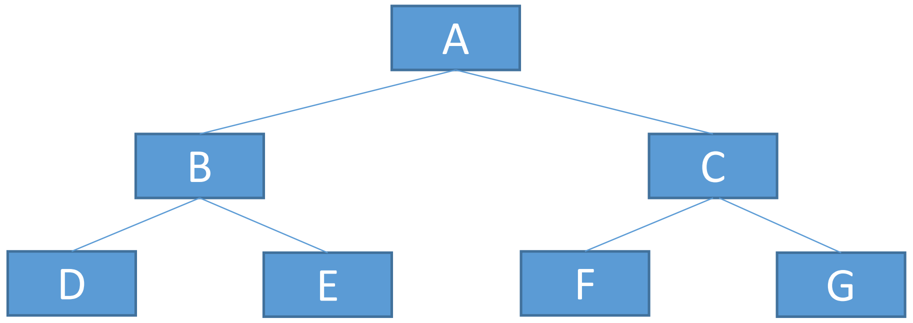
- Find the minimal spanning tree of the following graph



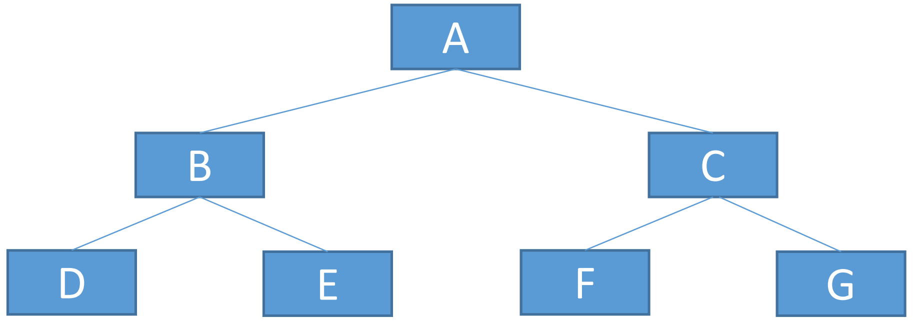
Listing Trees

- Sometimes we need to do something to every item in a tree.
- Trees are not ideally arranged for this process.
  - They are better suited for taking a single path from root to leaf.
- How can we efficiently list all the elements in a tree?
- E.g.

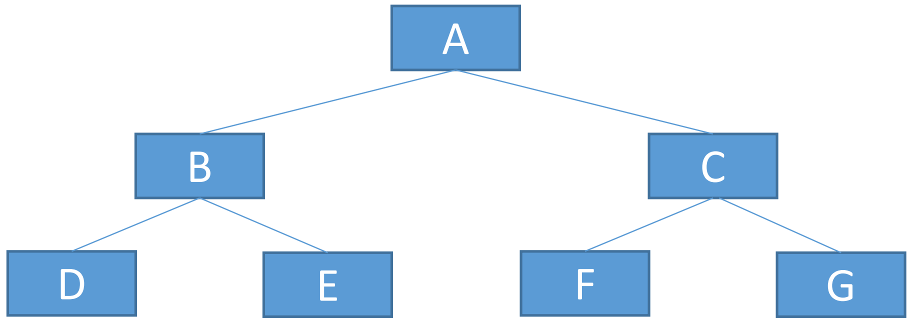




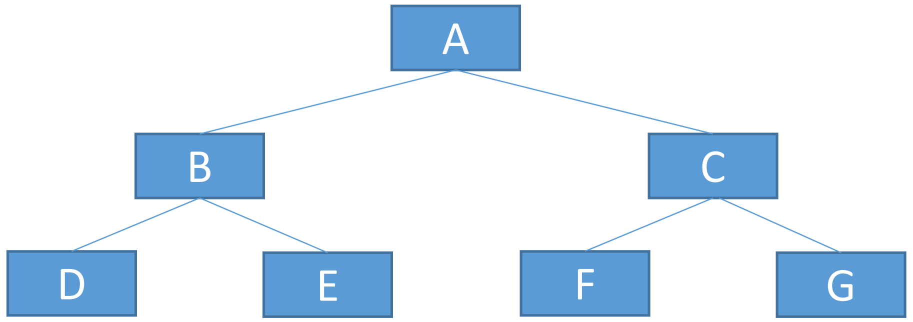
- The obvious list that this tree suggests is [A, B, C, D, E, F, G]
- How do we get this?
- Easy! List all of the nodes, left to right, at each level.
- This called a ***breadth-first*** traversal
- Actually, Not easy! The tree can only be navigated via the edges.
- This means that we are constantly travelling up and down branches and have to remember all the places we have been so far.



- The alternative is ***depth-first*** traversal.
- With this approach we go down a branch to the leaf before we traverse the rest of the tree.
- We can easily achieve this with a recursive procedure; **Visit**.
- Visit (node)
    - Visit (left child)
    - Visit (right child)
- We can see this working on the next slide.



- Visit (A)
    - Visit (B)
        - Visit (D)
        - Visit (E)
    - Visit (C)
        - Visit (F)
        - Visit(G)
- This traverses the tree but we still have one problem.
- When do we list the contents of the node?



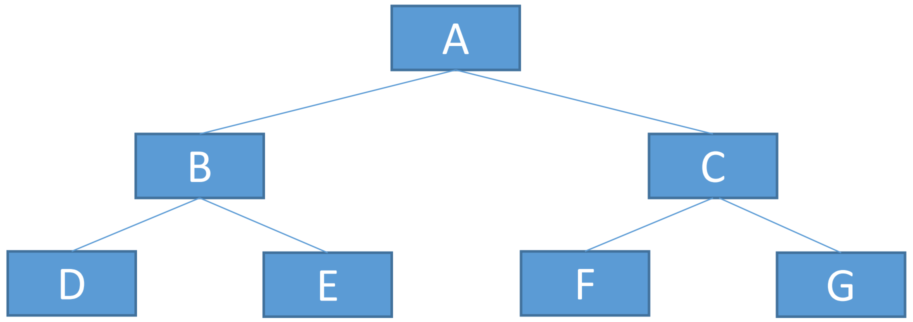
- We can modify Visit to list the tree by adding a Print (node) to it.
- We can do this in any one of three locations.

```
Visit_pre (node)
    Print (node)
    Visit_pre (left child)
    Visit_pre (right child)

Visit_in (node)
    Visit_in (left child)
    Print (node)
    Visit_in (right child)

Visit_post (node)
    Visit_post (left child)
    Visit_post (right child)
    Print (node)
```

- These are called ***pre-order***, ***in-order*** and ***post-order*** traversals respectively.
- Each gives us a list of the nodes in a different order.

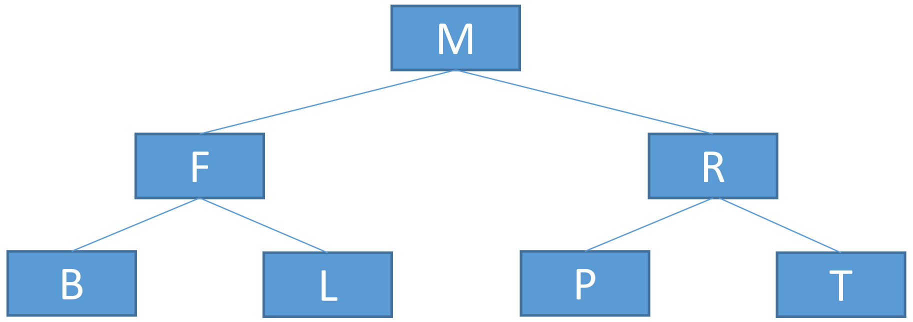


- What output do we get if we list the above tree:
  - Using pre-order traversal?
  - Using in-order traversal?
  - Using post-order traversal?

```
Visit_pre (node)
    Print (node)
    Visit_pre (left child)
    Visit_pre (right child)
```

```
Visit_in (node)
    Visit_in (left child)
    Print (node)
    Visit_in (right child)
```

```
Visit_post (node)
    Visit_post (left child)
    Visit_post (right child)
    Print (node)
```



- If we traverse a binary search tree like the one above;
  - Which of the three traversal strategies makes the most sense?

```
Visit_pre (node)
    Print (node)
    Visit_pre (left child)
    Visit_pre (right child)
```

```
Visit_in (node)
    Visit_in (left child)
    Print (node)
    Visit_in (right child)
```

```
Visit_post (node)
    Visit_post (left child)
    Visit_post (right child)
    Print (node)
```