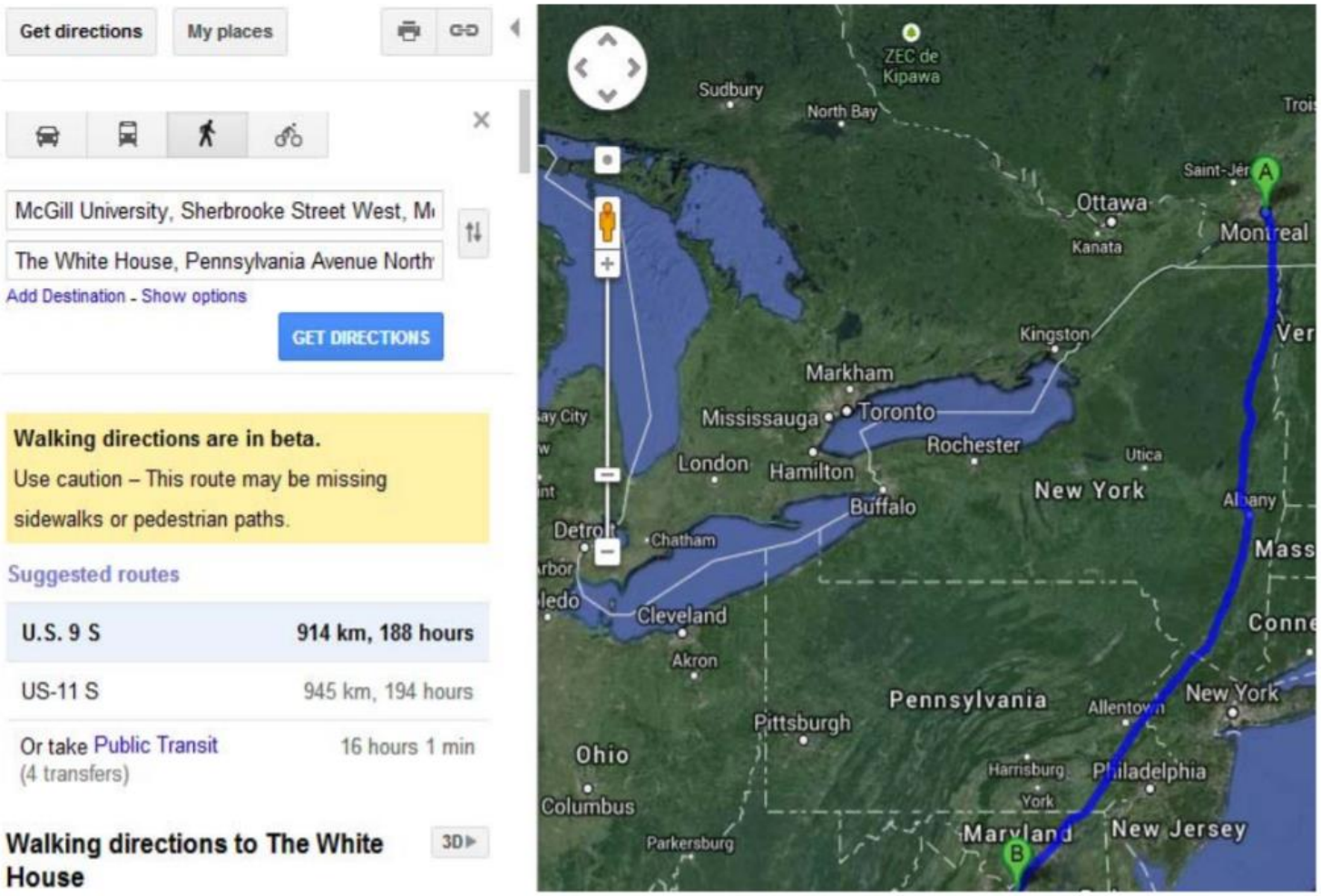
COMP 250

Lecture 28

graphs 2 (traversal)

Nov. 16, 2016



In COMP 251, you will learn Djikstra's algorithm for shortest path between two vertices of a (weighted) graph.

Today

- Recursive graph traversal
  - depth first
- Non-recursive graph traversal
  - depth first
  - breadth first

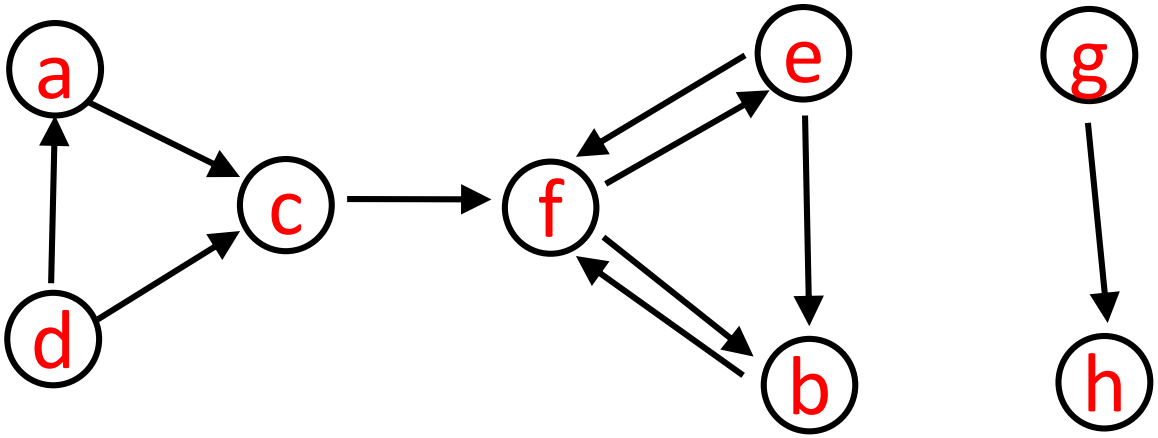
Recall: tree traversal (recursive)

```
depthfirst__Tree (root){
   if (root is not empty){
       visit root                          //      "preorder"
       for each child of root
            depthfirst__Tree( child )
   }
}
```

Graph traversal (recursive)

Need to specify a starting vertex.

Visit all nodes that are “reachable” by a path from a starting vertex.

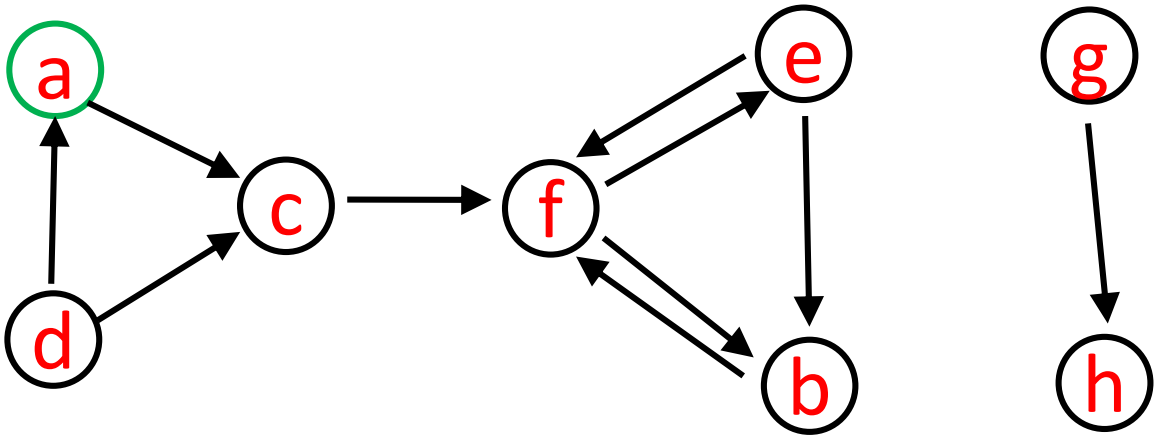


Graph traversal (recursive)

```
depthFirst_Graph(v){
  v.visited = true
  for each w such that (v,w) is in E   //  w in v.adjList
    if   ! (w.visited)                 //  avoids cycles
      depthFirst_Graph(w)
}
```

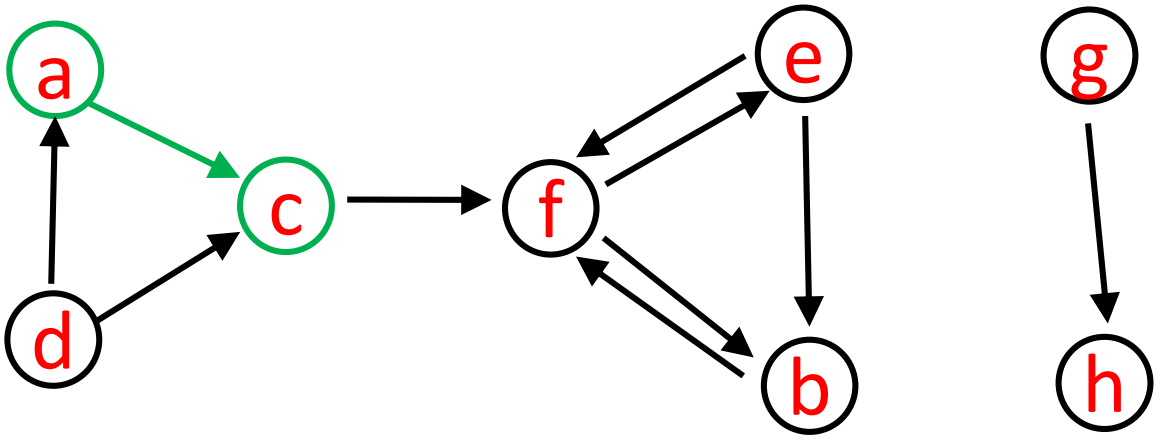
// Here “visiting” just means “reaching”

Call Stack for depthFirst(a)



a

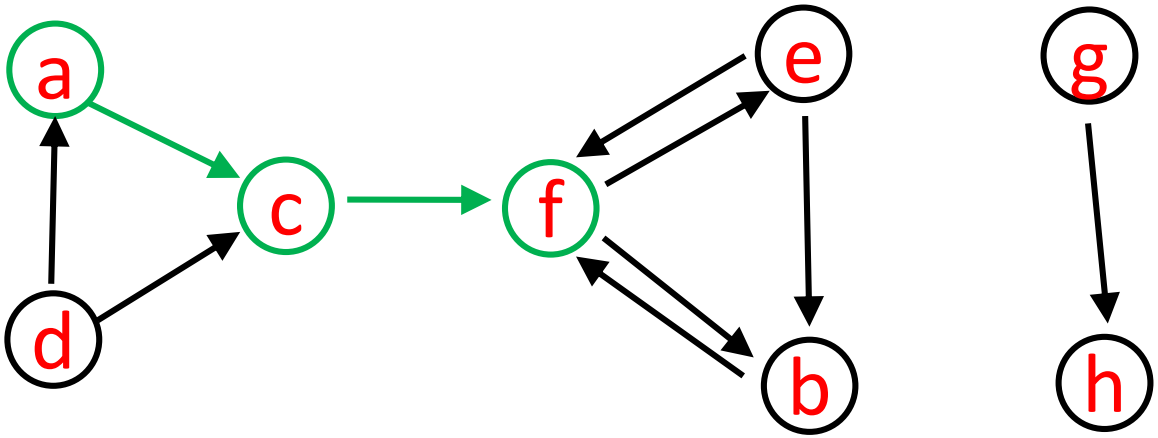
Call Stack for depthFirst(a)



c

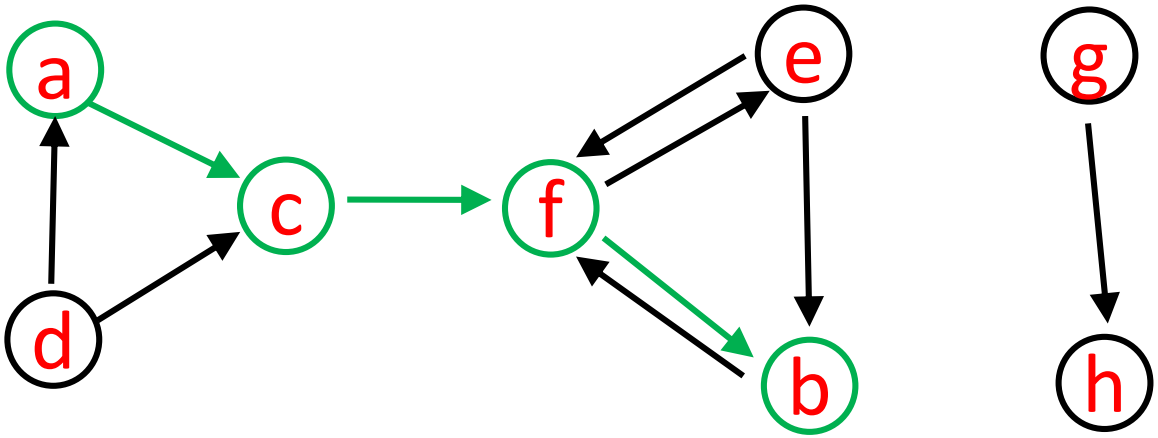
a a

Call Stack for depthFirst(a)



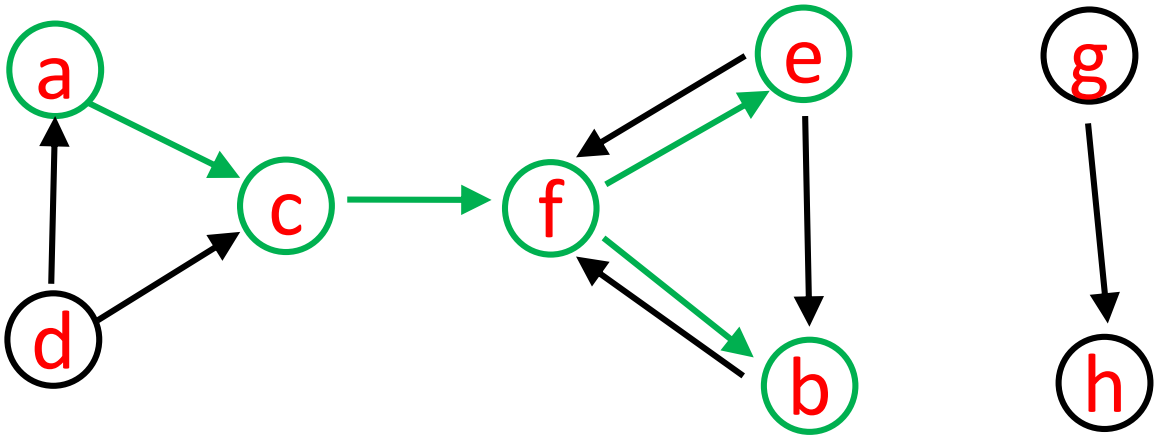
f
c c
a a a

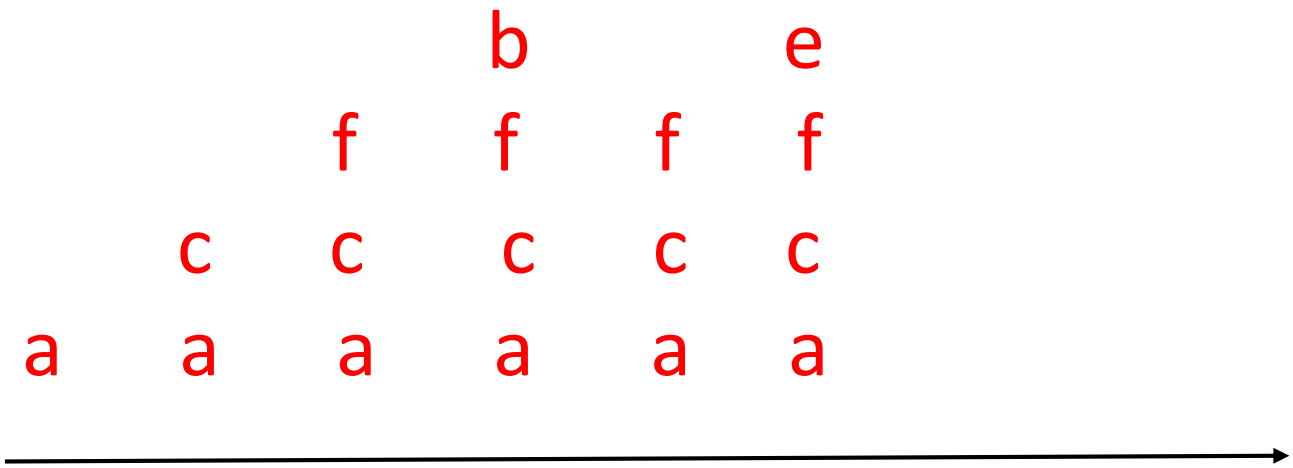
Call Stack for depthFirst(a)



b
f f
c c c
a a a a

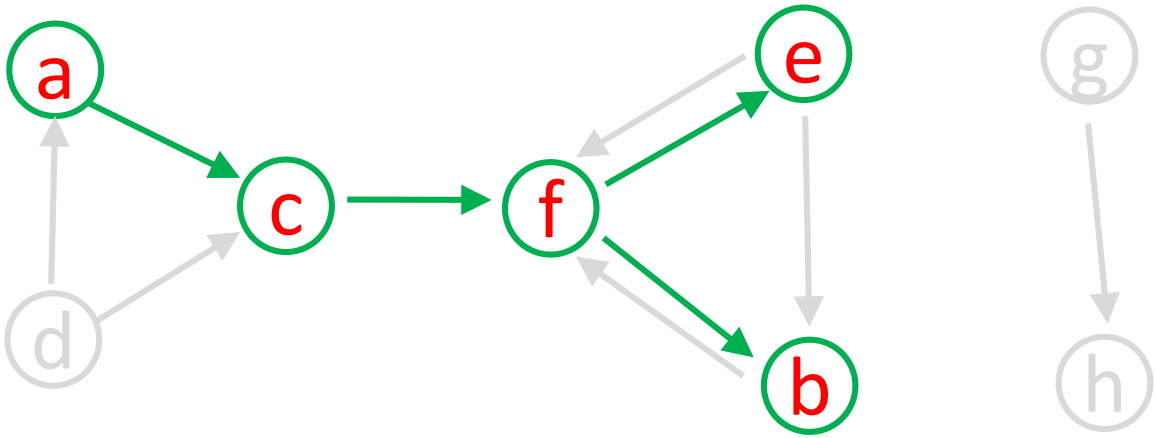
Call Stack for depthFirst(a)

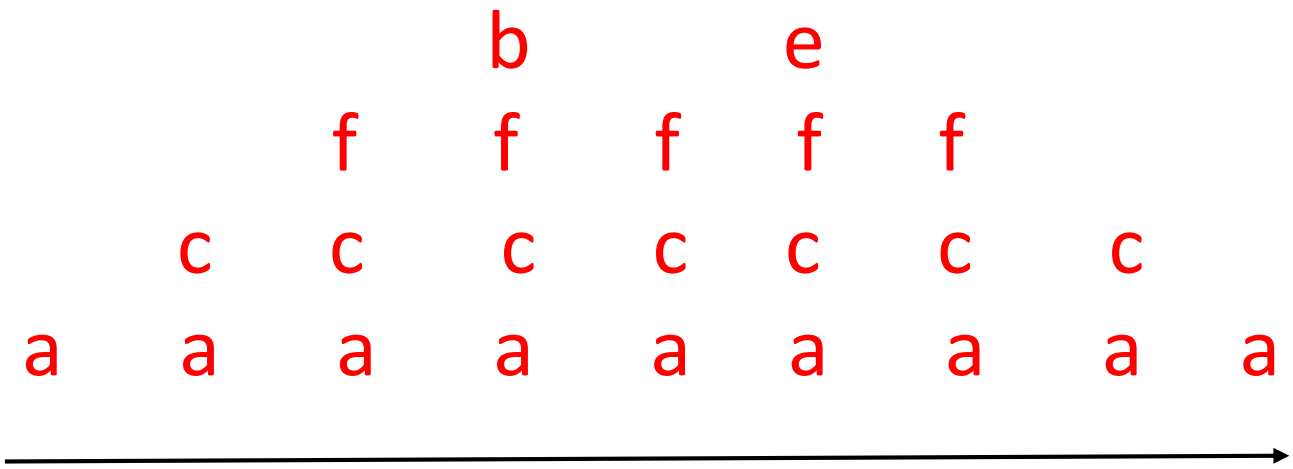




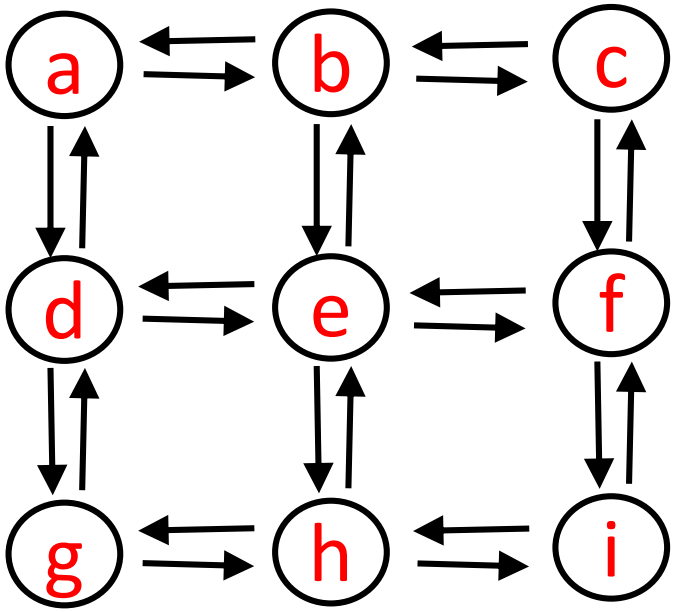
Call Tree

root





Example 2

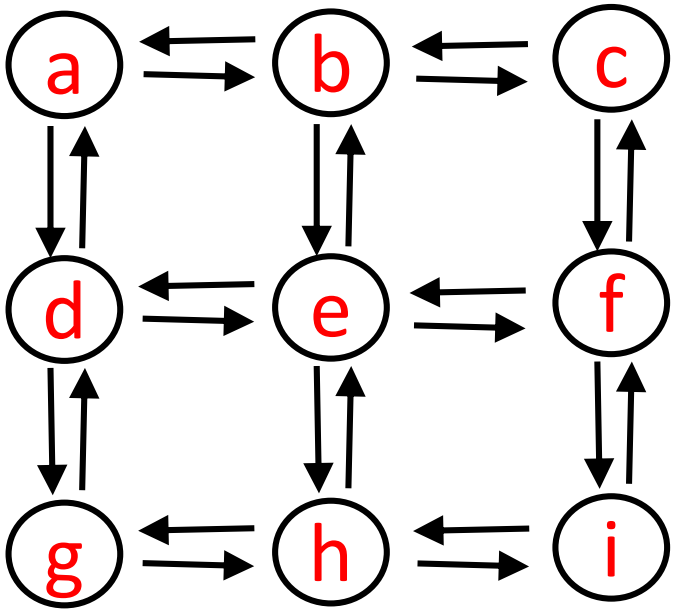


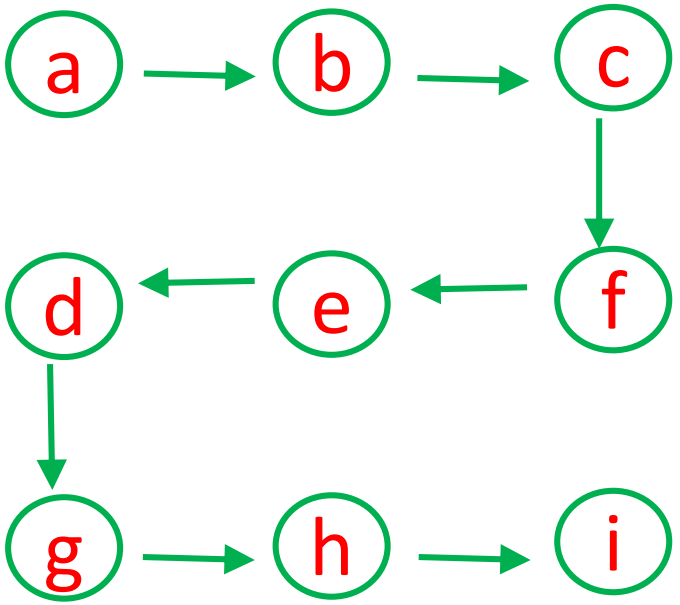
What is the call tree for depthFirst(a) *?*

Adjacency List

a - (b,d)
b - (a,c,e)
c - (b,f)
d - (a,e,g)
e - (b,d,f,h)
f - (c,e,i)
g - (d,h)
h - (e,g,i)
i - (f,h)

Example 2





Call Tree for depthFirst(a)

HEADS UP ! Prior to traversal, ….

```
for each w in V
    w.visited = false
```

How to implement this ?

```
class  Graph<T>  {
    HashMap< String, Vertex<T> >    vertexMap;
       :
    public void resetVisited() {
        Set<String>  vertexKeySet =  vertexMap.keySet();
        for ( String key : vertexKeySet ){
            vertexMap.get(key).visited = false;
        }
    }
}
```

Q: Non-recursive graph traversal ?

A: Similar to tree traversal: Use a stack (or a queue)

Recall: depth first tree traversal

```
treeTraversalUsingStack(root){
    initialize empty stack s
    s.push(root)
    while s is not empty {
        cur = s.pop()
        visit cur
        for each child of cur
            s.push(child)
    }
}
```

Visit a node *after popping* it from the stack.

Every node in the tree gets pushed, visited, and then popped.

Preorder because we visit a node before visiting children. However "visit" is not the same as "reach" in this case.

Slight variation….

```
treeTraversalUsingStack(root){
    initialize empty stack s
    visit root
    s.push(root)
    while s is not empty {
        cur = s.pop()
        for each child of cur
            visit child
            s.push(child)
    }
}
```

Visit a node *before pushing* it onto the stack.

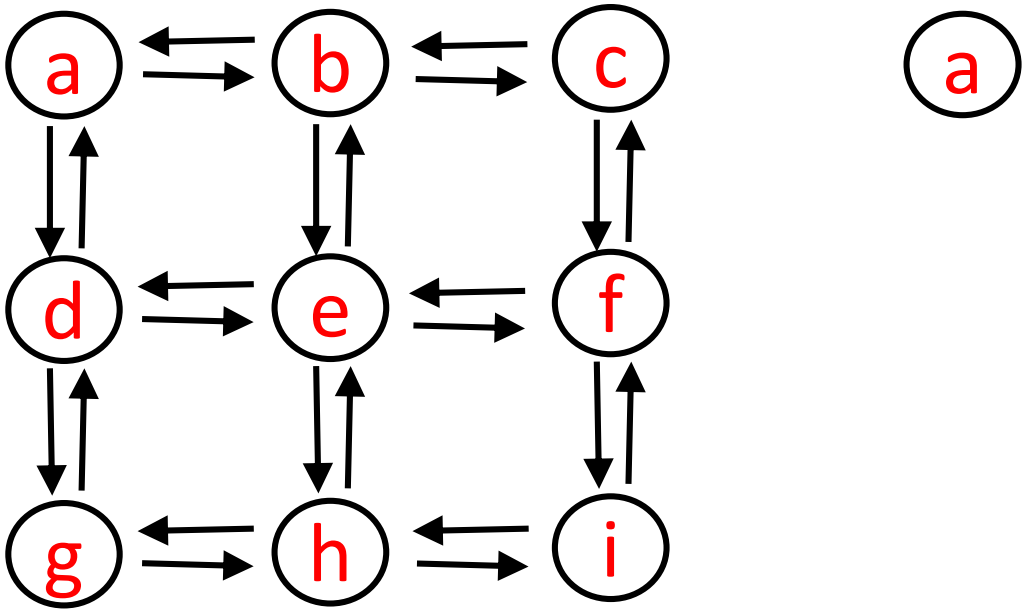
Every node in the tree gets visited, pushed, and then popped.

Preorder because we visit a node before visiting children.
Here “visit” = “reach”.

Generalize to graphs…

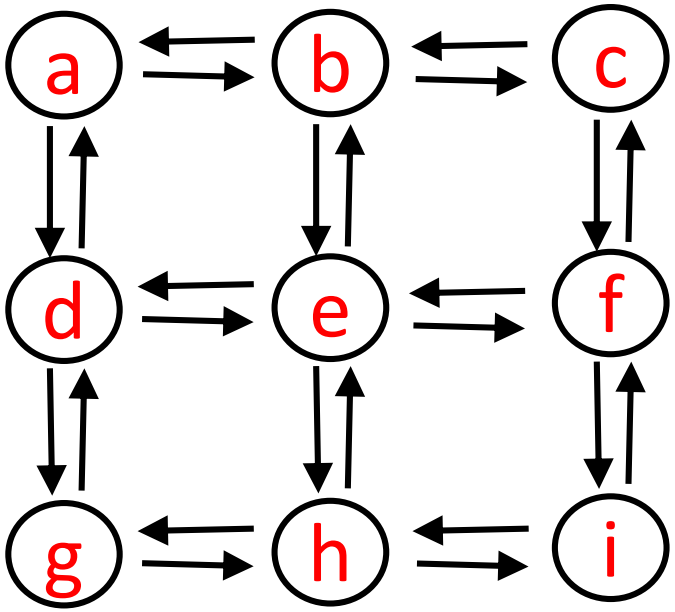
```
graphTraversalUsingStack(v){
    initialize empty stack s
    v.visited = true
    s.push(v)
    while (!s.empty) {
        u =  s.pop()
        for each w in u.adjList{
            if (!w.visited){                    //  the only new part
                w.visited = true
                s.push(w)
            }
        }
    }
}
```

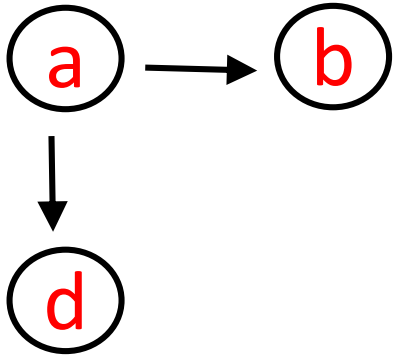
Example: graphTraversalUsingStack(a)



a

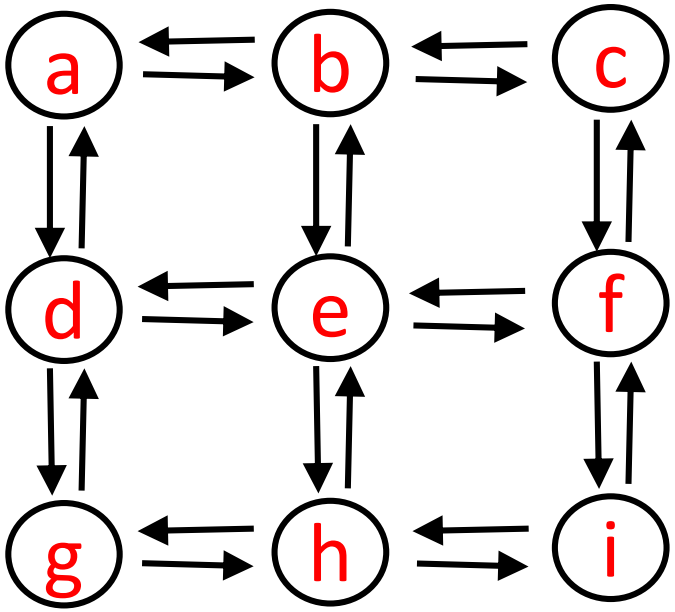
Example: graphTraversalUsingStack(a)

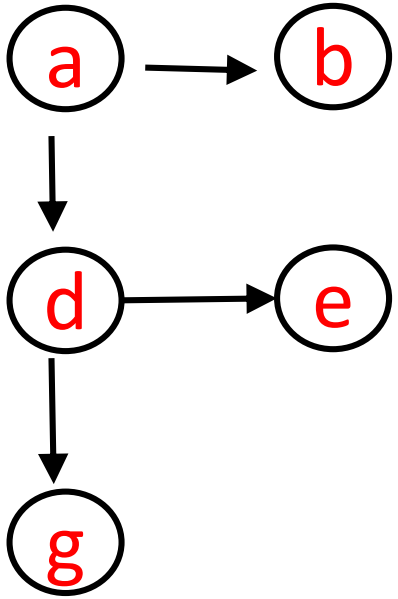




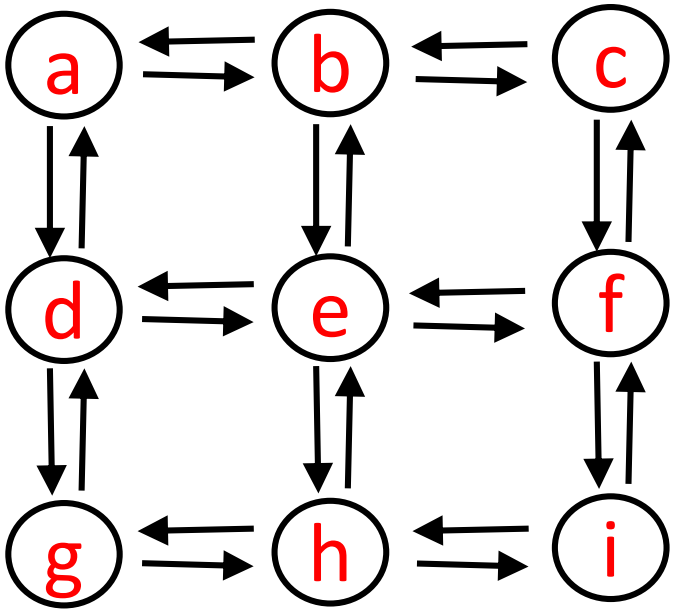
d
b
a a

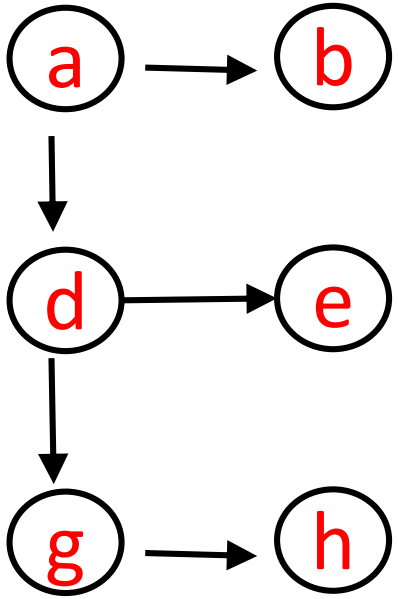
Example: graphTraversalUsingStack(a)





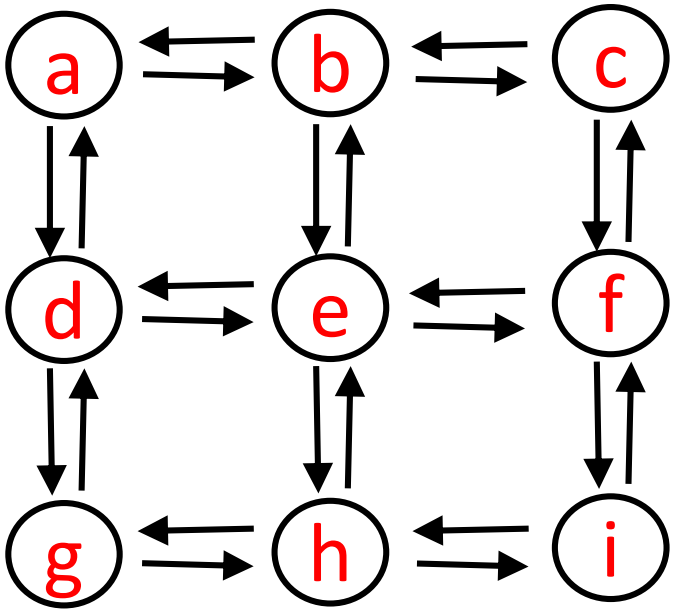
Example: graphTraversalUsingStack(a)

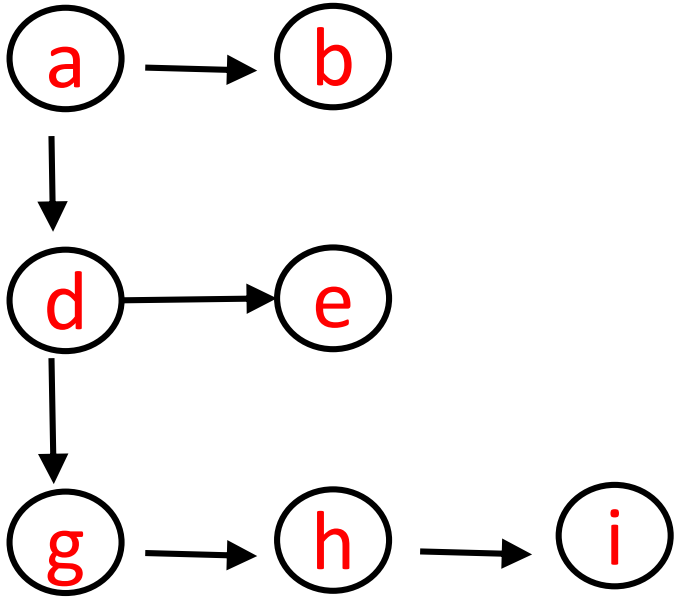




		g	h
	d	e	e
	b	b	b
a	a	a	a

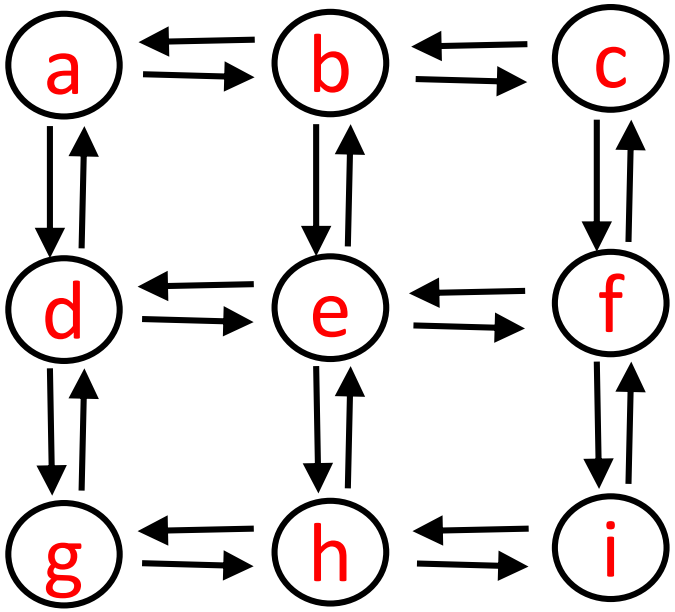
Example: graphTraversalUsingStack(a)

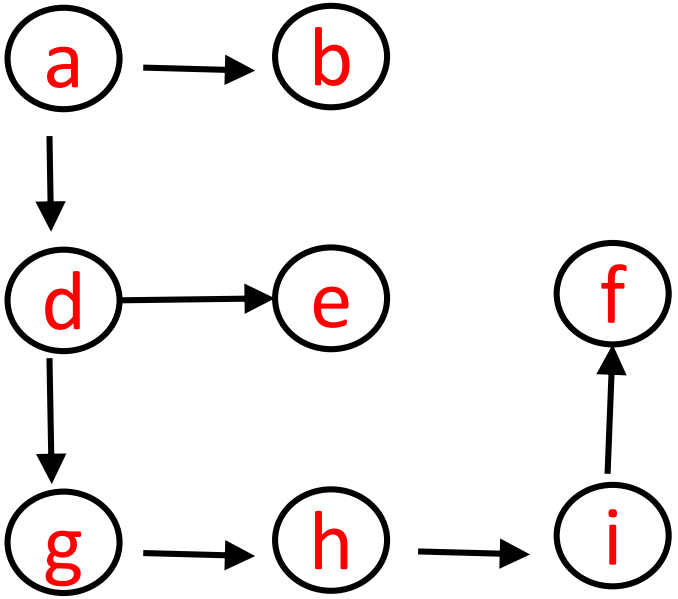




		g	h	i
	d	e	e	e
	b	b	b	b
a	a	a	a	a

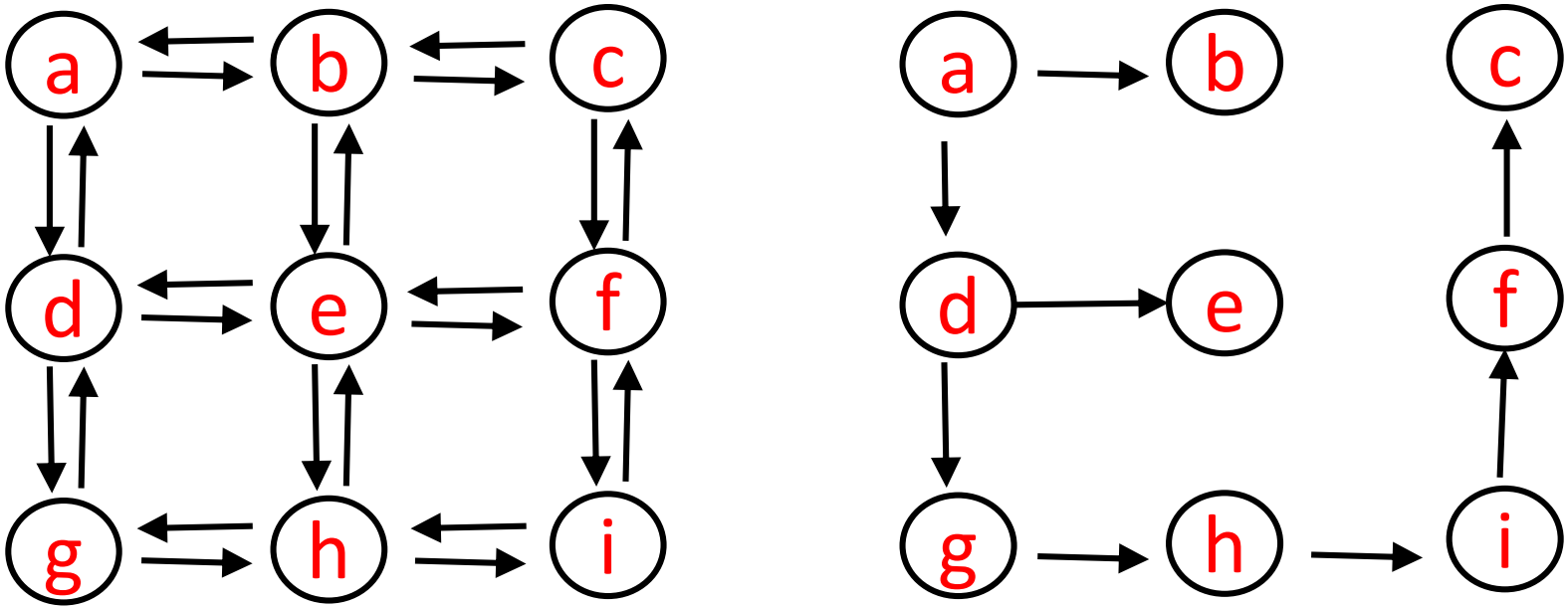
Example: graphTraversalUsingStack(a)





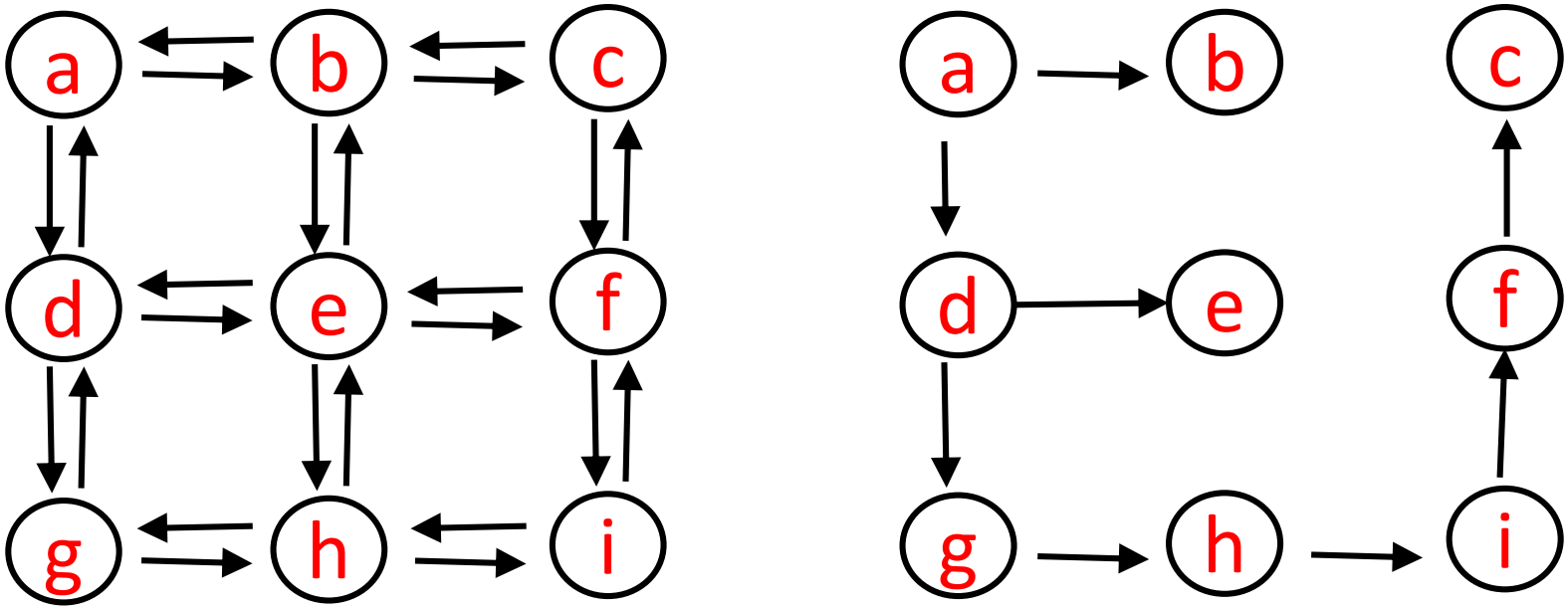
		g	h	i	f
	d	e	e	e	e
	b	b	b	b	b
a	a	a	a	a	a

Example: graphTraversalUsingStack(a)



		g	h	i	f	c
	d	e	e	e	e	e
	b	b	b	b	b	b
a	a	a	a	a	a	a

Example: graphTraversalUsingStack(a)



g h i f c
d e e e e e e
b b b b b b b b
a a a a a a a a a a

Breadth first graph traversal

Given an input vertex, find all vertices that can be reached by paths of length 1, 2, 3, 4, ….

i.e. find the shortest path (number of edges) that can be reached from the input vertex.

Breadth first graph traversal

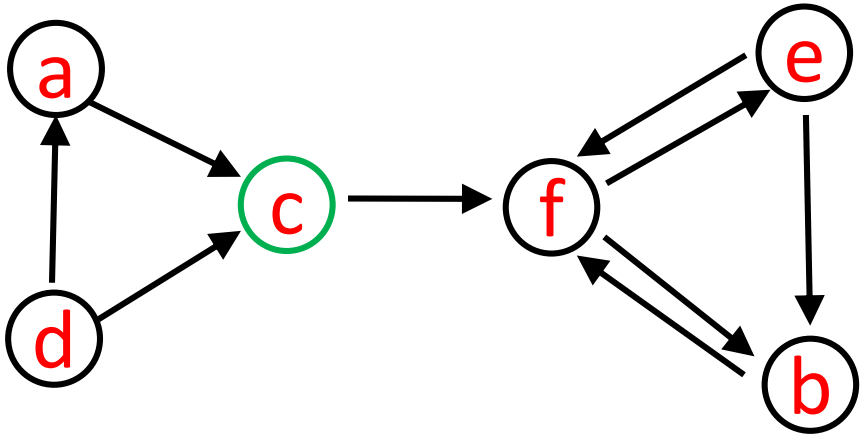
```
graphTraversalUsingQueue(v){     //  see lecture 18 slides 24-31
   initialize empty queue q
   v.visited = true
   q.enqueue(v)
   while (! q.empty) {
      u =  q.dequeue()
      for each w in u.adjList{
         if (!w.visited){
            w.visited = true
            q.enqueue(w)
         }
      }
   }
}
```

Example

graphTraversalUsingQueue(c)

queue

c



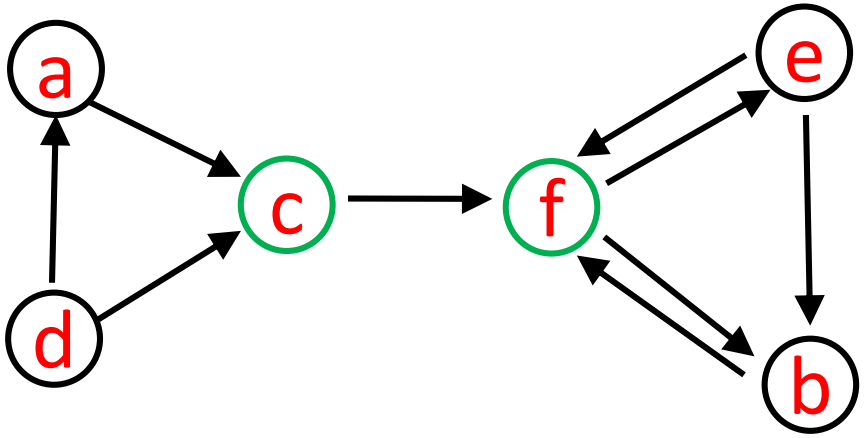
Example

graphTraversalUsingQueue(c)

queue

c

f



Example

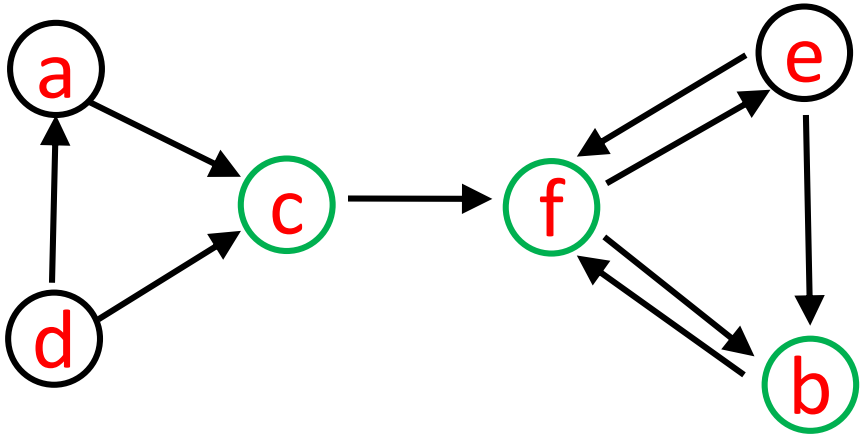
graphTraversalUsingQueue(c)

queue

c

f

be



Example

graphTraversalUsingQueue(c)

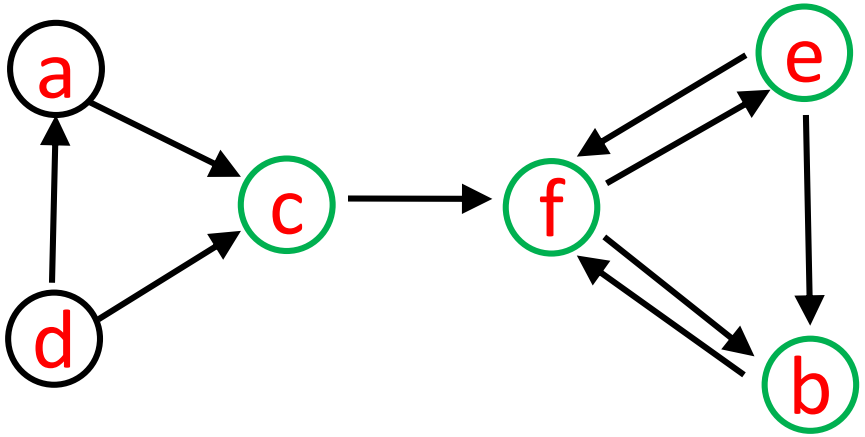
<u>queue</u>

c

f

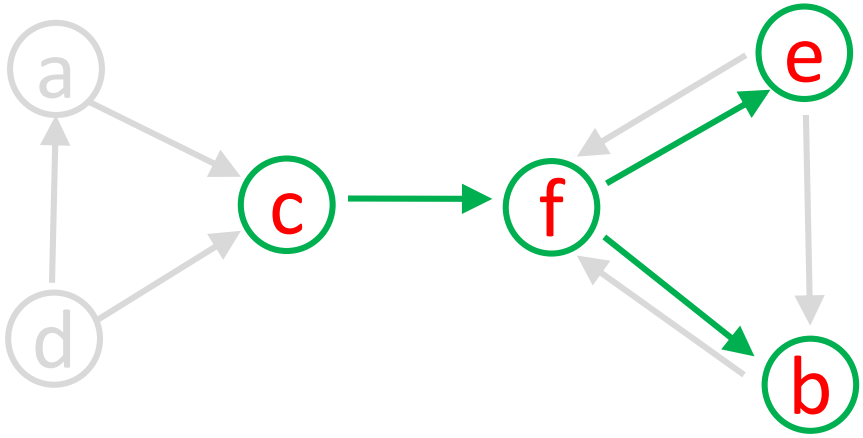
be

e



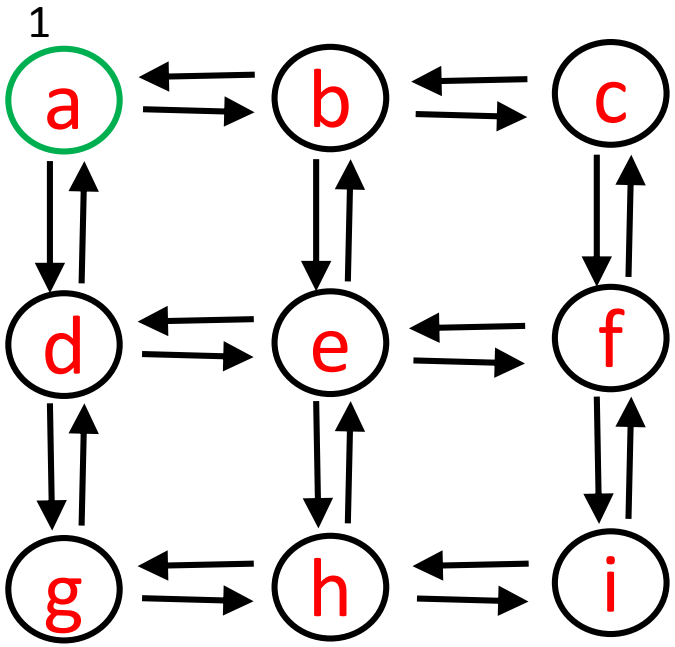
Example

graphTraversalUsingQueue(c)



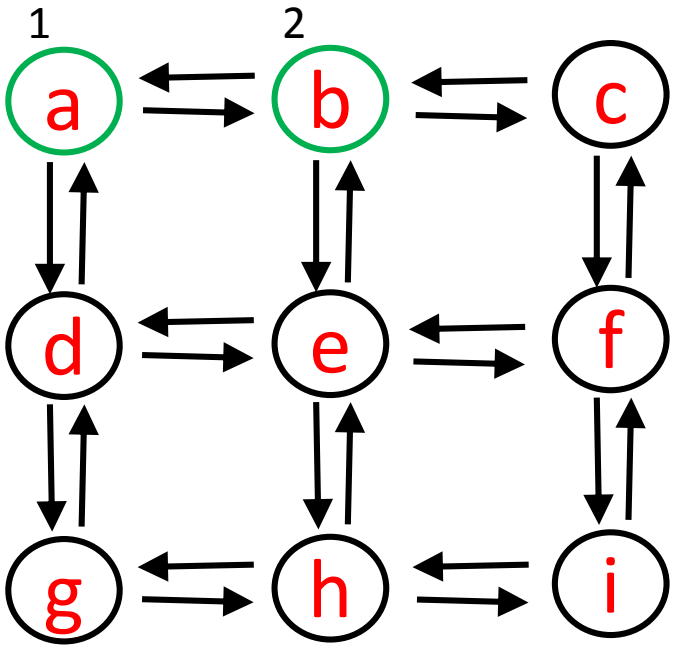
It defines a tree.

Example: graphTraversalUsingQueue(a)



a

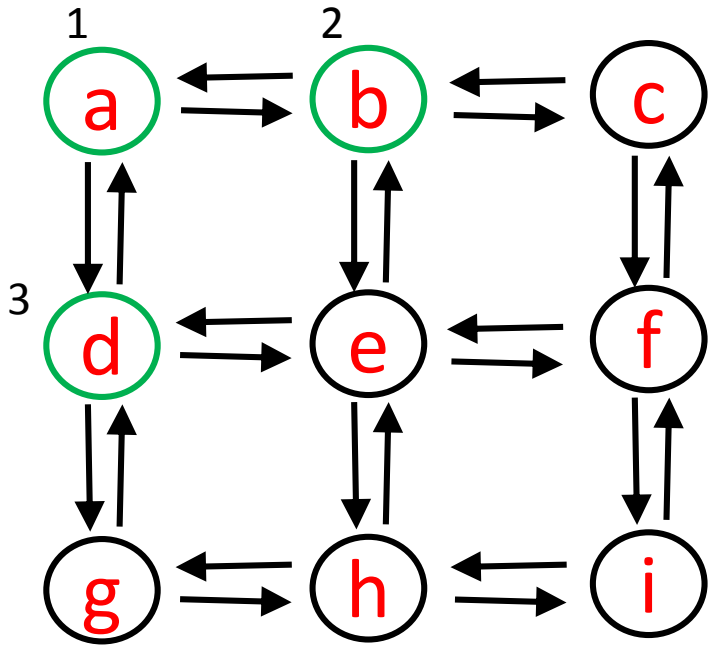
Example: graphTraversalUsingQueue(a)



a

bd

Example: graphTraversalUsingQueue(a)

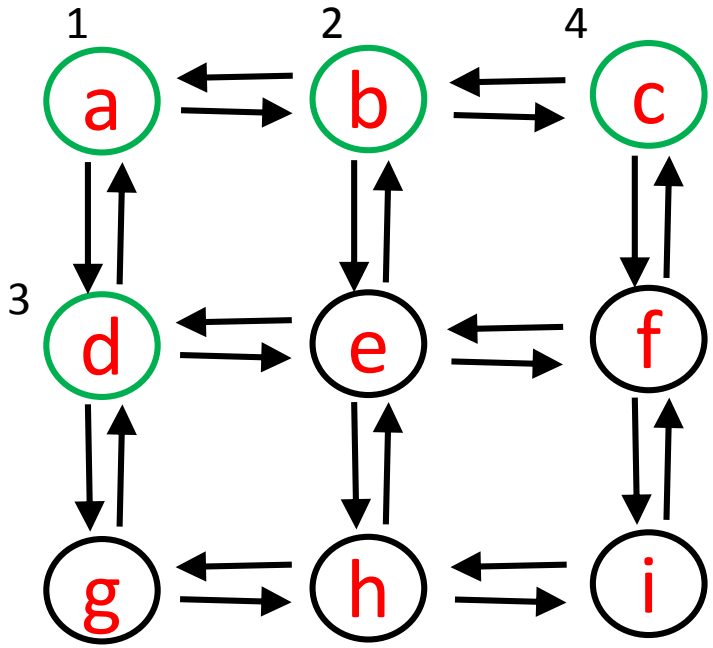


a

bd

dce

Example: graphTraversalUsingQueue(a)



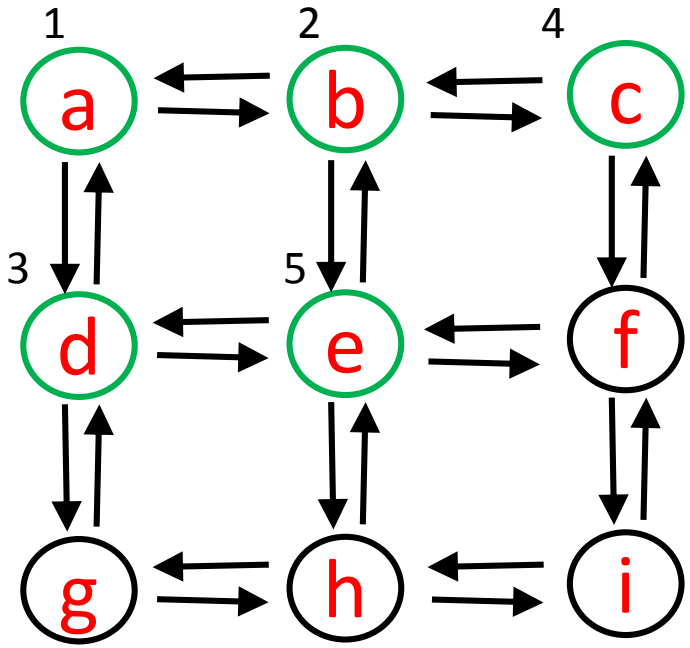
a

bd

dce

ceg

Example: graphTraversalUsingQueue(a)



a

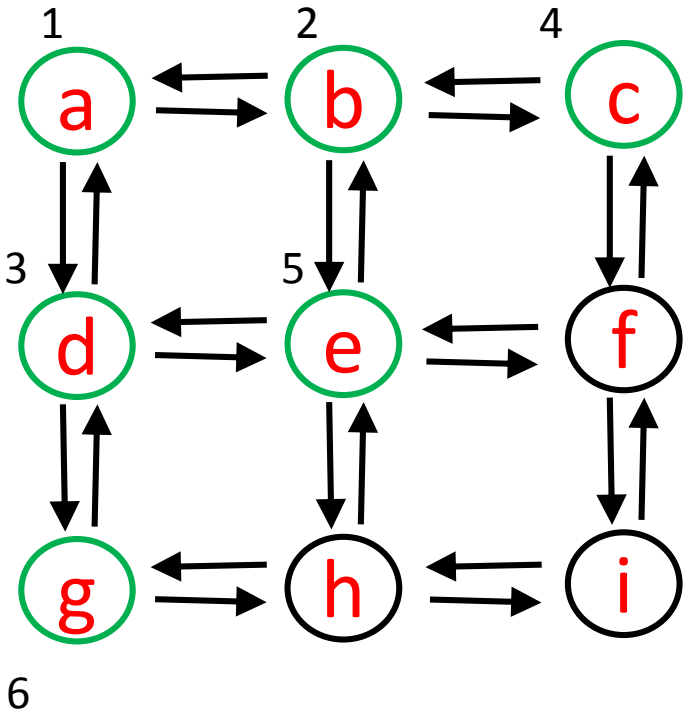
bd

dce

ceg

egf

Example: graphTraversalUsingQueue(a)



a

bd

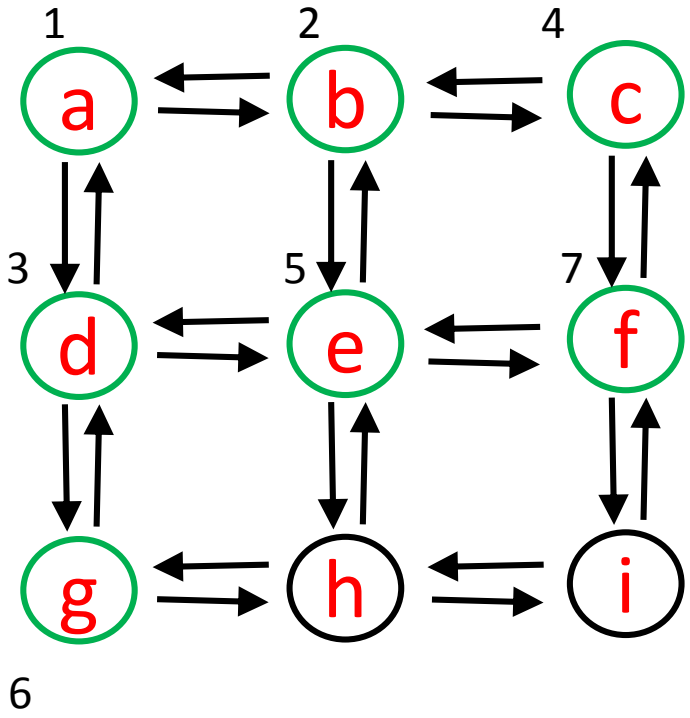
dce

ceg

egf

gfh

Example: graphTraversalUsingQueue(a)



a

bd

dce

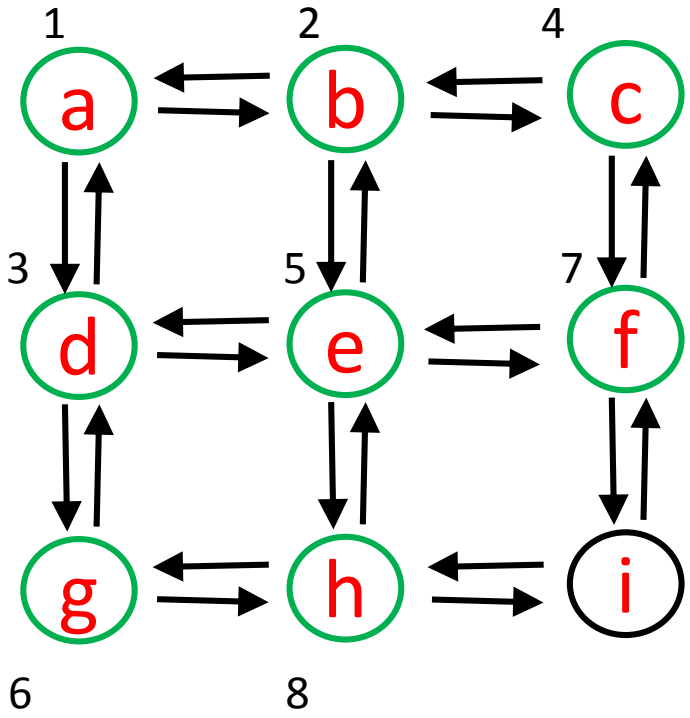
ceg

egf

gfh

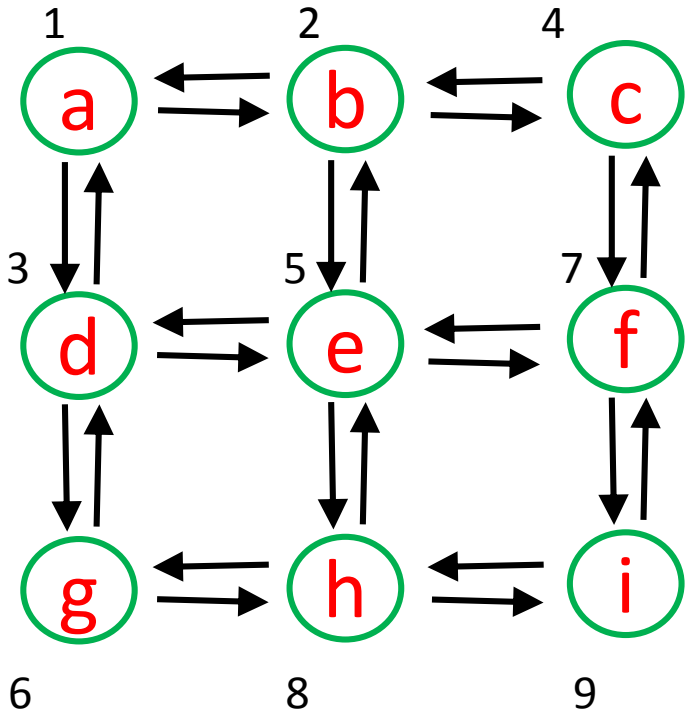
fh

Example: graphTraversalUsingQueue(a)



a
bd
dce
ceg
egf
gfh
fh
hi

Example: graphTraversalUsingQueue(a)



a

bd

dce

ceg

egf

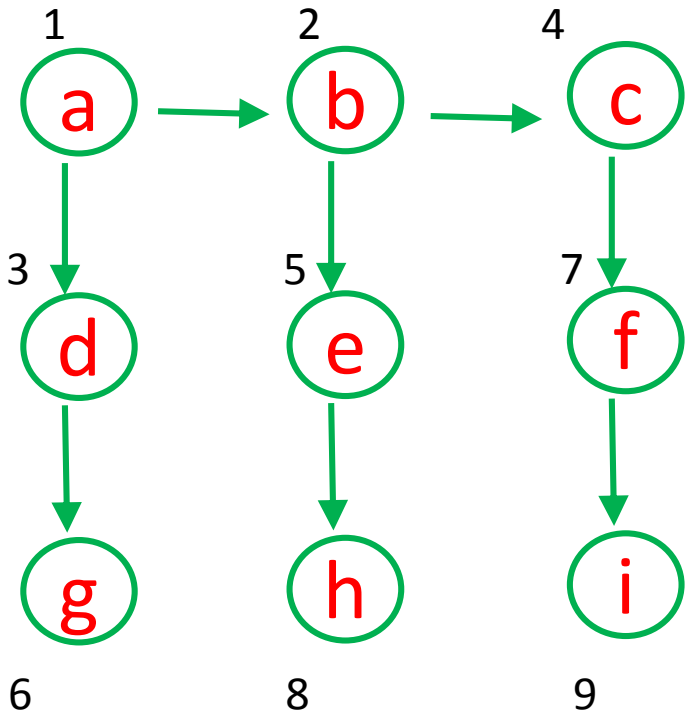
gfh

fh

hi

i

Example: graphTraversalUsingQueue(a)



Announcements

Assignment 4 tomorrow (hopefully)

- due in two weeks (Friday Dec. 2)
- Q1 Hashmaps
- Q2 Graphs and some object oriented design stuff

Lots of TA office hours: if you don't yet use debug mode, or you have trouble with packages, then get help from a TA.