

lecture 33

graph traversal

Recall: tree traversal (depth first)

```
preOrder(root) {
  visit root
  for each child of root
    preOrder(root.child)
}
```

Graph traversal

Given - a graph $G = (V, E)$
 - a node $v \in V$

Task

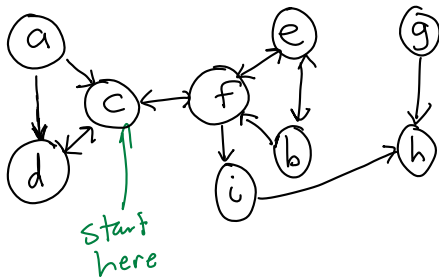
Visit all nodes that can be reached from v by a path

(a path is a sequence $v_1 \sim v_m$ such that $(v_i, v_{i+1}) \in E$)

Depth first traversal (also known as Depth first search)

```
dft(v) {
  v.visited = true
  for each w in adjList(v) {
    if !w.visited // avoids cycles (infinite loops)
      dft(w)
  }
}
```

Example

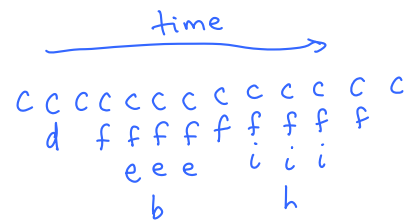


- a - c, d
- b - e, f
- c - d, f
- d - c
- e - b, f
- f - c, e, i
- g - h
- h - i
- i - h

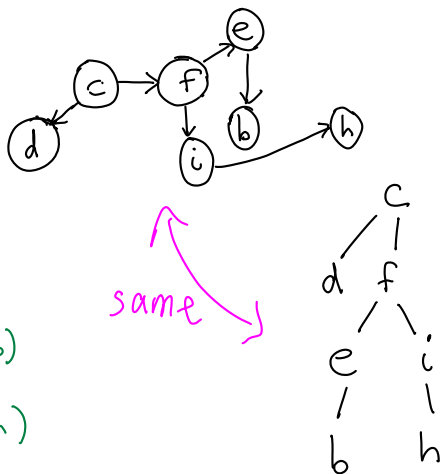
```
dft(c)
  dft(d)
  dft(f)
    dft(e)
    dft(b)
    dft(i)
      dft(h)
```

"Call Stack" (Recursion)

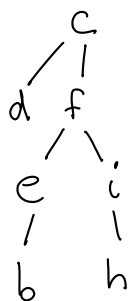
```
dft(c)
  dft(d)
  dft(f)
    dft(e)
    dft(b)
    dft(i)
      dft(h)
```



"Call tree" (we only visit once)



dft(c)
dft(d)
dft(f)
dft(e)
dft(b)
dft(i)
dft(h)



Pre. vs Post Order?

```

dft(v) {
  v.visited = true
  for each w in adjList(v) {
    if !w.visited
      dft(w)
  }
}
  
```



```

dft(v) {
  v.visited = true
  v.reached = true
  for each w in adjList(v) {
    if !w.visited & !w.reached
      dft(w)
  }
}
  
```

post order

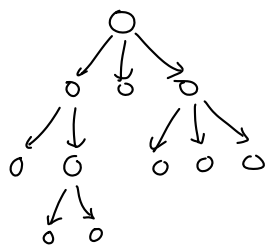
Breadth First Traversal

Given v , find all vertices that can be reached from v by examining paths of length $i = 1, 2, 3, 4, \dots$

RECALL: Breadth-first tree traversal

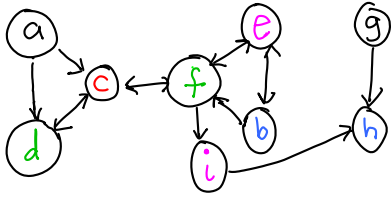
```

BFT(root) {
  q ← empty queue
  q.enqueue(root)
  while (!q.empty) {
    cur ← q.dequeue()
    visit cur
    for each child of cur
      q.enqueue(child)
  }
}
  
```



```

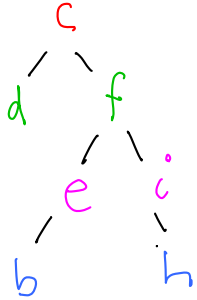
bft(v) {
  q = empty queue
  v.visited = true
  q.enqueue(v)
  while !q.empty {
    u = q.dequeue()
    for each w in u.adjList
      if !w.visited {
        w.visited = true
        q.enqueue(w)
      }
    }
  }
}
  
```



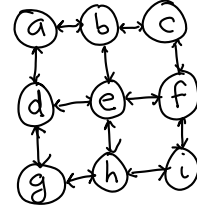
queue state

c
d f
f e i
e i b h
i b h
h

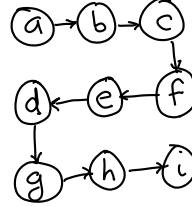
time



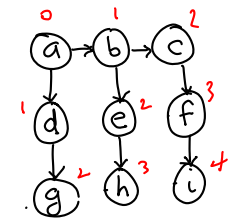
Example



dfs(a)



bfs(a)



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