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)

```java
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)

\[
\text{depthFirst\_Graph}(v)\{
  v.\text{visited} = \text{true}
  \text{for each } w \text{ such that } (v,w) \text{ is in } E \quad // \ w \text{ in } v.\text{adjList}
  \quad \text{if } \! (w.\text{visited})\\
  \quad \quad \text{\small depthFirst\_Graph}(w)
\}
\]

// Here “visiting” just means “reaching”
Call Stack for depthFirst(a)
Call Stack for depthFirst(a)
Call Stack for depthFirst(a)

```
Call Stack:

a
b
c
d
f
e
b
h
```

Diagram:

```
a -> c -> f -> e
```

```c
d
```

```g
g -> h
```
Call Stack for `depthFirst(a)`

```
  a
 /\  "/  
 c   f
 |   /  
 d -> c
     |   |
     |   |
     v   v
     b
     |   |
     |   |
     v   v
     g
     |   |
     |   |
     v   v
     h
  b
 /\  "/  
 f   f
 |   |
 c   c
 |   |
 |   |
 a   a   a   a   a
```
Call Stack for depthFirst(a)
Call Tree

root

a  

b  c  f  e

f  f  f  f  f  f  f

c  c  c  c  c  c  c  c

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

```
traversalUsingStack(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....

```java
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...

```java
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)
Example: graphTraversalUsingStack(a)
Example: `graphTraversalUsingStack(a)`

A graph with nodes `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i` and edges connecting them.

The traversal order is `a → b → c → d → e → g → h → i → f`.

In the simplified version, only `a → b` and `d → e` edges are shown.
Example: graphTraversalUsingStack(a)
Example: graphTraversalUsingStack(a)

```
g h i

d e e e
b b b b
a a a a a
```
Example: graphTraversalUsingStack(a)
Example: `graphTraversalUsingStack(a)`
Example: `graphTraversalUsingStack(a)`

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

```java
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

![Graph Diagram](image)
Example

`graphTraversalUsingQueue(c)`

queue

c
f

![Graph Diagram]

Nodes: a, c, d, e, f, b

Edges: a to c, c to d, c to f, f to e, e to b, b to c, c to f, d to c, f to c, f to d
Example

graphTraversalUsingQueue(c)

queue
c
f
be
Example

`graphTraversalUsingQueue(c)`

Queue:
- c
- f
- be
- e
Example

`graphTraversalUsingQueue(c)`

It defines a tree.
Example: graphTraversalUsingQueue(a)
Example: $\text{graphTraversalsUsingQueue}(a)$

1. a
2. b
3. c
4. d
5. e
6. f
7. g
8. h
9. i

Output: a, bd
Example: graphTraversalUsingQueue(a)

1. a
2. b c
3. d e f g h i

Sequence: a bd dce
Example: graphTraversalUsingQueue(a)

a -> b -> c
a <- d <- e <- f
a <- g <- h <- i

1 <- 2 <- 4
3
Example: `graphTraversalUsingQueue(a)`

```
gdahbifc
dceegf
```

```
1                     2                    4
3                       5
```

```
a
bd
dce
ceg
egf
```
Example: `graphTraversalUsingQueue(a)`

```
        1                     2                    4
    a ------ b ------ c
        |      |      |
     3------3      |      5------5
    d ------ e ------ f
        |      |      |
     6------6      |      6------6
    g ------ h ------ i
```

```
a
bd
dce
ceg
egf
gfh
```
Example: graphTraversalUsingQueue(a)

1. a
2. b
3. d
4. c
5. e
6. g
7. f
8. h
9. i

Sequence: a, bd, dce, ceg, egf, gfh, fh
Example: graphTraversalUsingQueue(a)
Example: graphTraversalUsingQueue(a)
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.