COMP 250
Lecture 32
graph traversal
March. 28, 2022
Today

• Recursive graph traversal
  • depth first

• Non-recursive graph traversal
  • depth first
  • breadth first
Graph traversal (recursive)

Specify a starting vertex.

Visit all nodes that are “reachable” by a path from a starting vertex. Today we will say “reaching” is the same as “visiting”.

![Graph diagram](image.png)
Recall: Tree traversal (recursive)

```plaintext
depthfirst_Tree (root){
    root.visited = true       // “preorder”
    for each child of root
        depthfirst_Tree( child )
}
}
```
Graph traversal (recursive)

def depthfirst_Graph(v):
    v.visited = true
    for each w such that (v,w) is in E  // w in v.adjList
        ______?______________

}
Graph traversal (recursive)

def 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)
}
Call Stack for depthFirst_Graph(a)

depthFirst_Graph(v){
    v.visited = true
    for each w such that (v,w) is in E
        if ! w.visited
            depthFirst_Graph(w)
}
Call Stack for depthFirst_Graph(a)

depthFirst_Graph(v)
{
    v.visited = true
    for each w such that (v,w) is in E
        if ! (w.visited)
            depthFirst_Graph(w)
}
Call Stack for `depthFirst_Graph(a)`

```
depthFirst_Graph(v){
    v.visited = true
    for each w such that (v,w) is in E
        if ! (w.visited)
            depthFirst_Graph(w)
}
```
Call Stack for depthFirst_Graph(a)

```
depthFirst_Graph(v) {
    v.visited = true
    for each w such that (v,w) is in E
        if ! w.visited
            depthFirst_Graph(w)
}
```
Call Stack for `depthFirst_Graph(a)`

```
depthFirst_Graph(v) {
    v.visited = true
    for each w such that (v,w) is in E
        if ! w.visited
            depthFirst_Graph(w)
}
```
Call Stack for depthFirst_Graph(a)

depthFirst_Graph(v) {
    v.visited = true
    for each w such that (v, w) is in E
        if ! (w.visited)
            depthFirst_Graph(w)
}
“Call Tree” for `depthFirst_Graph(a)`

In a running program, the call stack actually exists but the call tree does not exist. The call tree is only a way to visualize what the recursive calls are.
Graph Connectivity

Unlike tree traversal for rooted tree, a graph traversal started from some arbitrary vertex does not necessarily reach all other vertices.

Knowing which vertices can be reached by a path from some starting vertex is itself an important problem. You will learn about such graph `connectivity' problems in COMP 251.
Example 2

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

What is the call tree for depthFirst_Graph( a )?

(Do it in your head.)
Example 2

call tree for depthFirst_Graph (a)
Heads up -- Initialization

\[\text{depthfirstWithReset}(v)\{
\text{for each vertex } w \text{ in graph} \quad \text{// reset vertices}
\quad w.\text{visited} = \text{false}
\text{depthfirst_Graph} (v)\}
\]

\[\text{depthfirst_Graph}(v)\{ \quad \text{// helper method}
\quad v.\text{visited} = \text{true}
\quad \text{for each } w \text{ such that } (v,w) \text{ is in } E
\quad \quad \text{if } \quad ! (w.\text{visited})
\quad \quad \quad \text{depthfirst_Graph}(w)
\quad \}
\]
class Graph<T> {  
    HashMap<String, Vertex<T>> vertexMap;

class Vertex<T> {  
    ArrayList<Vertex> adjList;  
    T element;  
    boolean visited;

    void resetVisited() {  
        for (Vertex<T> v : vertexMap.values()) {  
            v.visited = false;  
        }

        // Implementation of pseudocode on previous slide
    }
}
ASIDE: Graph Traversal Example
A3 part 2

Recursive depth first graph traversal and visiting can have many forms, e.g.

solveMazeUtil(char maze[][], boolean found, int x, int y) {
    if (solveMazeUtil(maze, found, x + 1, y)) {
        return true;
    } else if (solveMazeUtil(maze, found, x - 1, y)) {
        return true;
    } else if (solveMazeUtil(maze, found, x, y + 1)) {
        return true;
    } else if (solveMazeUtil(maze, found, x, y - 1)) {
        return true;
    } else {  // backtrack
        return false;
    }
}
Today

• Recursive graph traversal
  • depth first

• Non-recursive graph traversal
  • depth first  (using stack)
  • breadth first  (using queue)
Recall: depth first **tree** traversal (non-recursive, using stack)

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)
    }
}
Slight variation on depth first **tree** traversal (using stack)

```python
treeTraversalUsingStack(root){
    visit root          // visit before push
    initialize empty stack s
    s.push(root)
    while s is not empty {
        cur = s.pop()
        for each child of cur
            visit child    // visit at ‘same time’ as push
                           s.push(child)
    }
}
```

We are still visiting each node before its children (but visit order is different).
Depth first graph traversal (using stack)

\texttt{graphTraversalUsingStack(v)}{
\begin{lstlisting}
visit v \hspace{1cm} // this can be done after push below
initialize empty stack s
s.push(v)
while (s is not empty) {
    u = s.pop()
    for each w in u.adjList{ \hspace{1cm} // new part
        if (!w.visited){
            visit w \hspace{1cm} // these two instruction can be done
            s.push(w) \hspace{1cm} // in either order ('same time')
        }
    }
}
\end{lstlisting}
\}

Updated after lecture:
see Exercises 12 (graphs) Question 6.
Example: `graphTraversalUsingStack(a)`

```
Example: graphTraversalUsingStack(a)

a   b   c
\downarrow \uparrow \downarrow \uparrow \downarrow \uparrow
\downarrow \uparrow \downarrow \uparrow \downarrow \uparrow
d   e   f
\downarrow \uparrow \downarrow \uparrow \downarrow \uparrow
g   h   i
\downarrow \uparrow \downarrow \uparrow \downarrow \uparrow
```

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

The traversal defines a rooted tree, but it is not a “call tree”. (The algorithm is not recursive.)

```
da    b  
an    e  
di    f  
g  h    i
```

′a′ was popped. ′b′ and ′d′ were pushed.
Example: graphTraversalUsingStack(a)

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

‘d’ was popped. ‘e’ and ‘g’ were pushed.
Example: `graphTraversalUsingStack(a)`

`g` was popped. `h` was pushed.
Example: `graphTraversalUsingStack(a)`

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

- 'h' was popped.
- 'i' was pushed.
Example: `graphTraversalUsingStack(a)`

`g h i f
 d e e e e e
da b b b b b`
Example: `graphTraversalUsingStack(a)`

```
g    d    e    f    g    h    i
  ^    ^    ^    v    ^    ^    ^
a    b    c    d    e    f    i
  |  |    |  |    |  |    |  |
  v  v    v  v    v  v    v  v
b  c    f  e    i  h    g  d
```

'f' was popped. 'c' was pushed.
Example: `graphTraversalUsingStack(a)`

Order of nodes visited (push order): `abdeghifc`
Recall: breadth first tree traversal

for each level i
visit all nodes at level i

treeTraversalUsingQueue(root){
    initialize empty queue q
    q.enqueue(root)
    while q is not empty {
        cur = q.dequeue()
        visit cur
        for each child of cur
            q.enqueue(child)
    }
}

// visit after dequeue
Breadth first graph traversal

```java
graphTraversalUsingQueue(v){
    visit v
    initialize empty queue q
    q.enqueue(v)
    while (q is not empty) {
        u = q.dequeue()
        for each w in u.adjList{
            if (!w.visited){
                visit w // visit at ‘same time’ as enqueue
                q.enqueue(w)
            }
        }
    }
}
```
Example

graphTraversalUsingQueue(c)

queue
  c

a → d → c → f

d → c

c

e → b
Example

graphTraversalUsingQueue(c)

queue
c
f
Example

graphTraversalUsingQueue(c)

Using alphabetical order for adjacency list.
Traversal defines a tree whose root is the starting vertex.
Example: `graphTraversalUsingQueue(a)`
Example: graphTraversalUsingQueue(a)
Example: `graphTraversalUsingQueue(a)`
Example: graphTraversalUsingQueue(a)
Example: `graphTraversalUsingQueue(a)`
Example: graphTraversalUsingQueue(a)
Example: `graphTraversalUsingQueue(a)`
Example: graphTraversalUsingQueue(a)
Example: graphTraversalUsingQueue(a)

Note order of nodes visited: abdcegfhi.

Traversal defines a tree whose root is the starting vertex.

One can show in general that this traversal first reaches nodes whose shortest path is length 0, then 1, then 2, etc. i.e. breadth first.
Coming up...

**Lectures**

Wed & Fri, March 30 & April 1
recurrences

Mon, Wed, Fri: April 4, 6, 8
big O, ...

**Assessments**

Quiz 5 is in Mon. April 4

Assignment 4 due Wed. April 6.