CS 225 Spring 2019 :: TA Lecture Notes
4/15 Traversal

By Wenjie

Graph Traversal

- Objective: Visit every vertex and every edge exactly once
- Purpose: Search for interesting substructures in the graph
- We've done it on trees, but it was easier

Trees	Graphs
1. **Ordered** → we always go from parents to children. 2. **Obvious start** → we start at the root node. 3. **Notion of completeness** → we are done when we reach leaf nodes.	1. **Unordered** → no notion of children nodes, just neighbours. 2. **No** obvious start → we can start anywhere. 3. **No** notion of completeness → we need to know when we have visited all nodes.

- **BFS Algorithm:**
    1. Add the starting point
    2. While the queue is not empty
        a. Dequeue **v**
        b. For all of the unlabeled edges adjacent to **v**
            - If an adjacent edge "discovers" a new vertex **t**:
                - Label the edge a "discovery edge"
                - Enqueue **t**, update the information of **t** (distance = dist(**v**) + 1, predecessor = **v**)
            - If an adjacent edge is between two visited vertices
                - Label the edge a "cross edge"

- **Example**: see previous lecture notes / video

- **The code**

```
BFS(G):
    Input: Graph, G
    Output: A labeling of the edges on
              G as discovery and cross edges
```

CS 225 Spring 2019 :: TA Lecture Notes 4/15 Traversal

By Wenjie

```
5
6       foreach (Vertex v : G.vertices()):
7           setLabel(v, UNEXPLORED)
8       foreach (Edge e : G.edges()):
9           setLabel(e, UNEXPLORED)
10      foreach (Vertex v : G.vertices()):
11          if getLabel(v) == UNEXPLORED:
12              BFS(G, v)
13              //components++;
```

```
14  BFS(G, v):
15      Queue q
16      setLabel(v, VISITED)
17      q.enqueue(v)
18
19      while !q.empty():
20          v = q.dequeue()
21          foreach (Vertex w : G.adjacent(v)):
22              if getLabel(w) == UNEXPLORED:
23                  setLabel(v, w, DISCOVERY)
24                  setLabel(w, VISITED)
25                  q.enqueue(w)
26              elseif getLabel(v, w) == UNEXPLORED:
27                  setLabel(v, w, CROSS)
28                  // cycleExists = true;
```

- Use cases and functionality:
    - Does this code work on a disjoint graph (2 or more separate pieces)?
        - Yes, since line 10 goes through every vertex, regardless of connectedness
    - How do we use the traversal to count the number of components?
        - Every BFS run indicates a connected component, so we can add a counter after a call of BFS (line 13).
    - Can our implementation detect a cycle?
        - Yes, a cross edge indicates a cycle (line 28).

CS 225 Spring 2019 :: TA Lecture Notes
4/15 Traversal

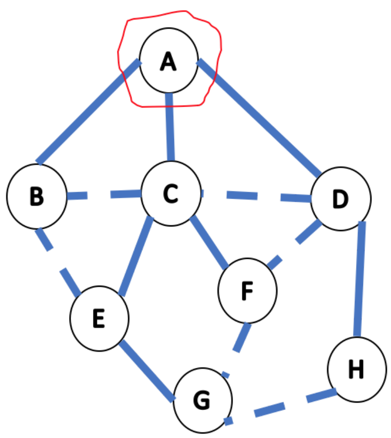
By Wenjie

- **Running time:**
  - Expect: visit every edge and vertex, so O(n+m)
  - Looking at specific parts of the code:
    - Second part of the code:
      - Line 19: $O(n)$
      - Line 21: $O(deg(v))$
      - Lines 22-27: $O(1)$
      - This whole chunk is $O(n \times deg(v))$
      - $deg(v)$ is not very informative, but we know that we will have $n \times deg(v) = \sum^{n} deg(v) = 2m \Rightarrow O(2m)$
    - First part of the code:
      - Lines 6-7: $O(n)$
      - Lines 8-9: $O(m)$
      - Lines 10-12: $O(n)$
  - Total running time is $O(n+m)$.
  - This is optimal running time because we know we have to visit every edge and vertex, therefore we cannot do better than $O(n+m)$.
- BFS doesn't give a unique solution, but the properties are guaranteed.

CS 225 Spring 2019 :: TA Lecture Notes
4/15 Traversal

By Wenjie

BFS Observations



key	visited	dist.	pred.	adj. vertices
A	✓	0	null	C B D
B	✓	1	A	A E C
C	✓	1	A	A B D E F
D	✓	1	A	A C F H
E	✓	2	C	B C G
F	✓	2	C	C D G
G	✓	3	E	E F H
H	✓	2	D	D G

- What is the shortest path from A to H?
    - path along discovery edges: A->D->H
- What is the shortest path from E to H?
    - Actual: E->G->H
    - Is not obvious in the BFS result
    - **BFS only finds the shortest paths from the start**
        - single source shortest path
- How does a cross edge related to **dist**?
    - Cross edge will never change the **dist** more than 1.
    - BFS keeps thins local: every edge increase/decrease distance by 1.
- What structure is made from discovery edges?
    - A tree (forest, if the graph is not connected) rooted at the start
    - Further, it's a spanning tree (forest): it connects all vertices in the graph

DFS

- Idea:
    - similar idea with BFS, but we can use either
        - A stack

        - A recursion
    - Recursive algorithm: we visit a vertex **v**
        1. Check all adjacent vertices of **v**
            a. if it's not been discovered, label the edge "discovery edge". Visit the new vertex
            b. label the edge that leads us to a already discovered vertex "back edge"
                i. since it *usually* brings us to a closer vertex
                ii. the distance difference is unbounded
    - Observations
        - The discovery edges make a spanning tree
        - **d** does not find the shortest path
        - the benefit is: it discovers new vertices very quickly

- The code: use system stack as our workstack (recursion)