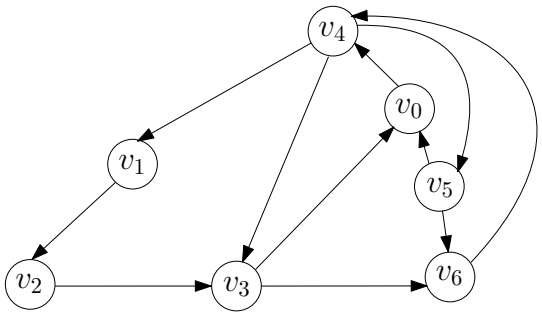
Homework 5 Solution (COMPSCI 223)

Q1: Starting from the node v_3, illustrate the BFS algorithm on the following graph.

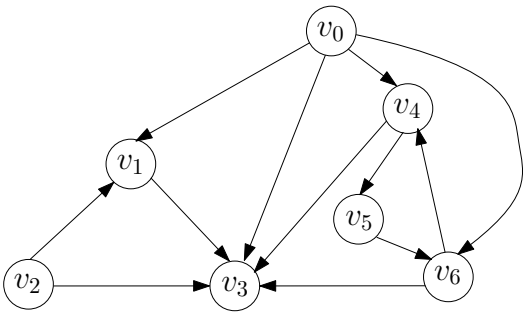


- Show the queue, and the level array at each stage.
- List the edges of the final BFS tree. Thus, if there is an edge from v_1 to v_6, you write (v_1, v_6) as one of the edges.

	level array							**queue**
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	
At Start:	∞	∞	∞	0	∞	∞	∞	$[v_3]$
Relax v_3:	1	∞	∞	0	∞	∞	1	$[v_0, v_6]$
Relax v_0:	1	∞	∞	0	2	∞	1	$[v_6, v_4]$
Relax v_6:	1	∞	∞	0	2	∞	1	$[v_4]$
Relax v_4:	1	3	∞	0	2	3	1	$[v_1, v_5]$
Relax v_1:	1	3	4	0	2	3	1	$[v_5, v_2]$
Relax v_5:	1	3	4	0	2	3	1	$[v_2]$
Relax v_2:	1	3	4	0	2	3	1	[ ]

Edges are: $(v_3, v_0); (v_3, v_6); (v_0, v_4); (v_4, v_1); (v_4, v_5); (v_1, v_2)$

Q2: Starting from the node v_0, illustrate the DFS algorithm on the following graph.



Show the stack, the closed set and the level array at each stage. When you relax a vertex, push adjacent vertices into the stack in increasing value of their indexes. Thus if v_0 has outgoing edges to v_2, v_5, and v_6, then first push v_2, then v_5, then v_6.

	closed	**stack**	**level array**						
			v_0	v_1	v_2	v_3	v_4	v_5	v_6
At Start	$\varnothing$	$[v_0]$	0	∞	∞	∞	∞	∞	∞
Pop (Relax v_0)	v_0	$[v_1, v_3, v_4, v_6]$	0	1	∞	1	1	∞	1
Pop (Relax v_6)	v_0, v_6	$[v_1, v_3, v_4, v_3, v_4]$	0	1	∞	2	2	∞	1
Pop (Relax v_4)	v_0, v_6, v_4	$[v_1, v_3, v_4, v_3, v_3, v_5]$	0	1	∞	3	2	3	1
Pop (Relax v_5)	v_0, v_6, v_4, v_5	$[v_1, v_3, v_4, v_3, v_3]$	0	1	∞	3	2	3	1
Pop (Relax v_3)	v_0, v_6, v_4, v_5, v_3	$[v_1, v_3, v_4, v_3]$	0	1	∞	3	2	3	1
Pop (Relax v_3)	v_0, v_6, v_4, v_5, v_3	$[v_1, v_3, v_4]$	0	1	∞	3	2	3	1
Pop (Relax v_4)	v_0, v_6, v_4, v_5, v_3	$[v_1, v_3]$	0	1	∞	3	2	3	1
Pop (Relax v_3)	v_0, v_6, v_4, v_5, v_3	$[v_1]$	0	1	∞	3	2	3	1
Pop (Relax v_1)	$v_0, v_6, v_4, v_5, v_3, v_1$	[ ]	0	1	∞	3	2	3	1

Q3: Answer true or false for the following.

- The complexity of BFS on a binary tree having N nodes is $O(N)$

  True. A binary tree having N nodes has $(N-1)$ edges.

- The complexity of DFS on a directed graph with V vertices and E edges is $O(V+E)$

  True.

- BFS gives us shortest paths in directed graphs, whereas DFS gives us longest paths in directed graphs.

  False. BFS computes shortest paths in unweighted graphs, whereas DFS does not compute either shortest or longest path.

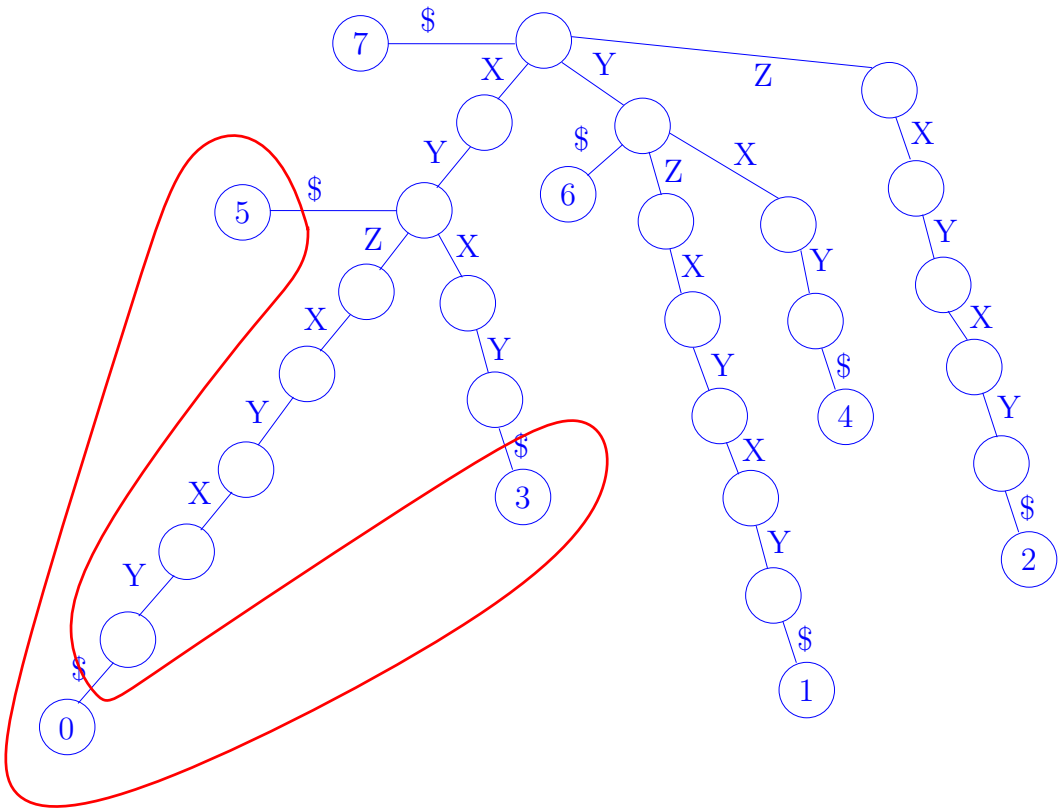
- BFS and DFS algorithms are faster (Big-O complexity wise) on directed graphs than in undirected graphs, where both the graphs have the same number of vertices and edges.

  False. They have the same complexity irrespective of the graph type.

- Starting BFS on two different vertices in a graph may lead to different level-values.

  True.

Q4: Draw the suffix-trie for the string: $XYZXYXY\$$. **Mark the leaves** corresponding to the occurrences of the pattern XY.



Q5: In an unweighted undirected connected graph having V vertices and E edges, define the average hop distance to be: $\frac{X}{Y}$, where

- X is the sum of the shortest path distances from every vertex to every other vertex
- $Y = V * (V - 1)$ is the number of pair of vertices.

Since the graph is connected, we have $E \geq V - 1$. Describe an $O(VE)$ time algorithm to compute the average hop distance.

Answer:

1. Let $X = 0$
2. for $(i = 0; i < V; i++)$
   - Run BFS from vertex i.
   - for $(j = 0; j < V; j++)$
     - $X = X + level[j]$
3. $Y = V * (V - 1)$
4. Avg. hop distance $= X/Y$

Each BFS execution needs $O(V+E)$ time. Updating X for each BFS execution needs $O(V)$ time. There are V many BFS executions. Hence, the complexity is $O(V*(V+E)) = O(VE)$ time.

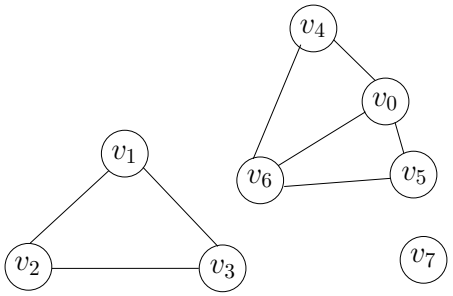
Q6: Let G be a directed graph having V vertices and E edges. A transitive closure is a matrix M such that $M[i][j]$ is true if there is a path from vertex v_i to vertex v_j, else $M[i][j]$ is false. How will you use DFS to compute the matrix M in $O(V^2+VE)$ time?

Answer:

1. Create a two-dimensional array M of dimensions $V \times V$
2. for $(i = 0; i < V; i++)$
   - Run DFS from vertex i.
   - for $(j = 0; j < V; j++)$
     - if ($level[j]$ equals ∞), then set $M[i][j] = false$, else set $M[i][j] = true$

Each DFS execution needs $O(V+E)$ time. Updating M for each DFS execution needs $O(V)$ time. There are V many DFS executions. Hence, complexity is $O(V*(V+E)) = O(V^2+VE)$ time.

Q7: A component in an undirected graph is a subset of vertices (and edges) such that there is a path between every pair of vertices. For example, the 3 components of the following graph are $\{v_0, v_4, v_5, v_6\}$; $\{v_1, v_2, v_3\}$; and $\{v7\}$.



Using Breadth-First Search (don't use DFS) as the main idea, give an $O(V+E)$ detailed pseudo-code to find the number of components of an undirected graph with V vertices and E edges. Assume the following:

- The vertices are labeled $\{0, 1, 2, ..., V-1\}$
- The graph is given in an adjacency-list form

Answer:

1. Set $numComponents = 0$. Initialize all entries of $level[\,]$ array to ∞
2. for $(i = 0; i < V; i++)$,
   - if ($level[i]$ equals ∞), then
     - set $level[i] = 0$ and enqueue i
     - increment $numComponents$ by one

- while (queue has at least one vertex)
  - $u = dequeue()$
  - for every outgoing edge (u, v) of u
    - if ($level[v]$ equals ∞) enqueue v and set $level[v] = level[u] + 1$

Note that once you execute BFS from a vertex, then any vertex which is reachable from the vertex (and thus in the same component) will have its level value set to some finite integer; so, we never call BFS from a vertex whose level value is finite. Thus, every vertex and every edge is relaxed a constant number of times. Hence, the complexity is $O(V + E)$.

Q8: A repeating substring of a string X is a substring that occurs at least twice in X. For e.g., if $X = abcdbce$, then bc is a repeating substring in X, whereas ab is not.

A longest repeating substring is a repeating substring which is also the longest (there may be more than one). For e.g., if $X = cabbcabca$, then cab and bca are both longest repeating substrings.

Your task is to find such a longest repeating substring (or detect there is none). Suppose you are given a suffix trie of X . Assume the following:

- At any node in the trie, you can traverse to a child node or parent node in $O(1)$ times.
- For any node, you can find the number of leaves in the subtree of that node in $O(1)$ time.
- At each node, you can read the label of the edge to its parent in $O(1)$ time.
- The number of nodes in the trie is $m > n$.

Describe an $O(m)$ time algorithm to compute a longest repeating substring using the suffix trie.

Answer: Note that if a node in the trie has more than one leaf in its subtree, then the corresponding prefix (formed by the string from root to that node) occurs at least twice. A repeating substring is any such prefix; thus, one of the deepest ones is a longest repeating substring.

Using BFS, find the deepest node (ties broken arbitrarily) which has ≥ 2 leaves in its substree. The answer is the string formed by concatenating the edge labels from the root to this node.

Using BFS, the deepest node is found in $O(m)$ time. Now use one of the suffixes (corresponding to a leaf) under this node to traverse the trie from the root and reach this node in $O(n)$ time; concatenate the edge labels along the way. Overall complexity is, therefore, $O(m)$.

More specifically,

- Initially, $answerNode = null$ and $longestRepeatLength = 0$
- Start at the root node and execute a BFS
- While executing BFS, when an edge from u to v is relaxed, do the following:
  - $level[v] = level[u] + 1$ and enqueue v
  - if (v has more than one children in its subtree AND $level[v] > longestRepeatLength$)
    - $longestRepeatLength = level[v]$
    - $answerNode = v$
  - Initialize $LRS =$ “”
  - Start at v and traverse up to the root node. Concatenate the edge characters to LRS
  - Reverse LRS, which gives the the longest repeating substring