CPCFI

Lecture: GRAPH TRAVERSAL I

Unit: 6 - GRAPHS 1

Instructor: WESTON

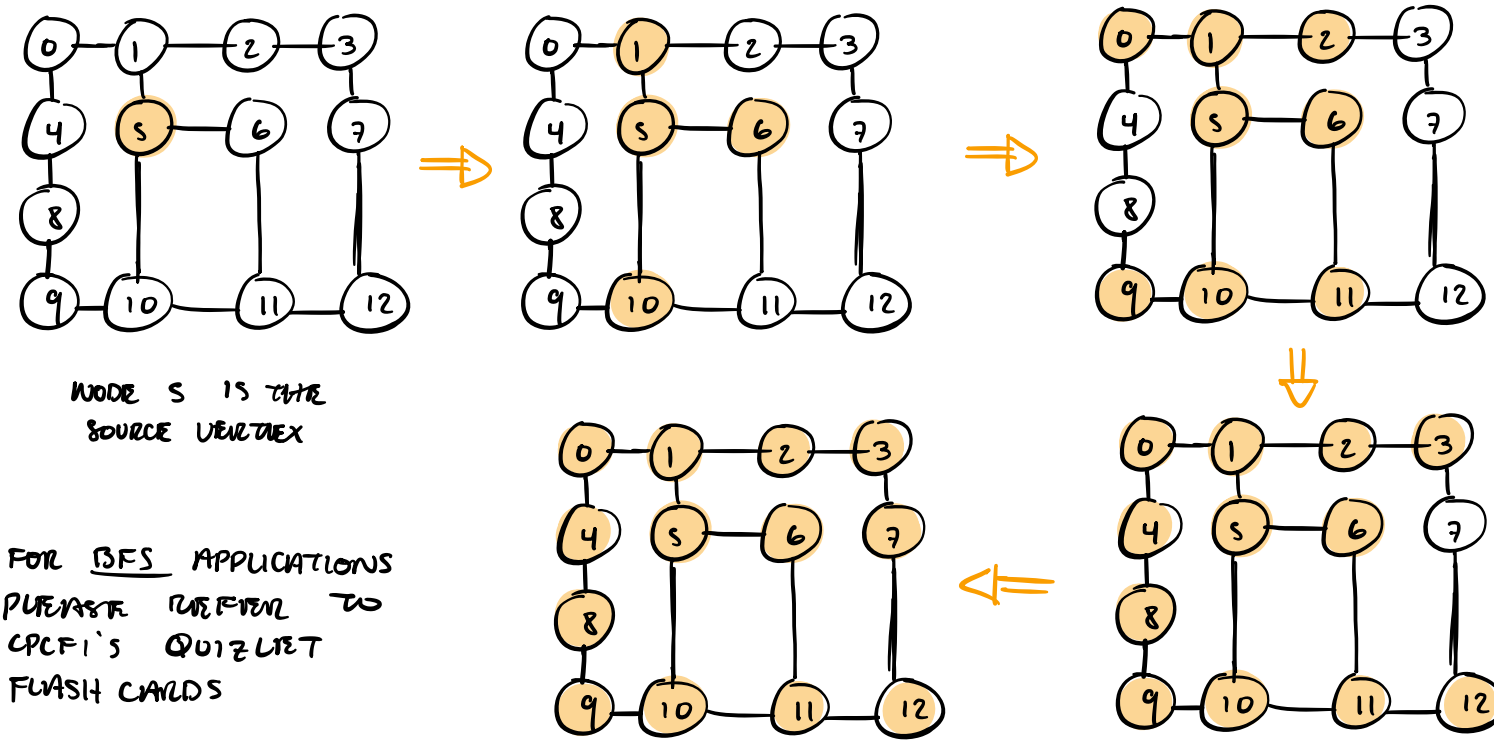
1 BREADTH - FIRST SEARCH

- ESSENTIAL SEARCHING ALGORITHM ON GRAPHS
- BFS PATH TO ANY NODE IS THE SHORTEST PATH
  - ↳ PATH THAT CONTAINS THE SMALLEST # OF EDGES
- TIME COMPLEXITY: $O(n+m)$

  n: # VERTICES
  m: # EDGES

DESCRIPTION

- INPUT: ① UNWEIGHTED GRAPH (DIRECTED OR UNDIRECTED)
  ② ID OF THE SOURCE VERTEX
- "FIRE SPREADING ALGORITHM"
  - BREADTH STEP: ONLY SOURCE VERTEX IS ON FIRE
  - FIRST STEP: FIRE SPREADS TO ALL OF ITS NEIGHBORS
  - ON EACH STEP THE RING OF FIRE EXPANDS BY ONE UNIT



NODE 5 IS THE SOURCE VERTEX

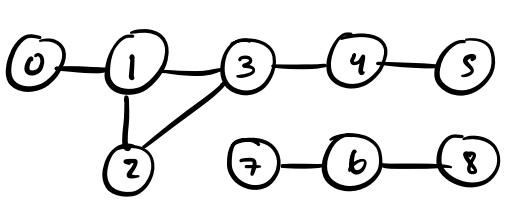
* FOR BFS APPLICATIONS PLEASE REFER TO CPCFI'S QUIZLET FLASH CARDS

II DEPTH-FIRST SEARCH

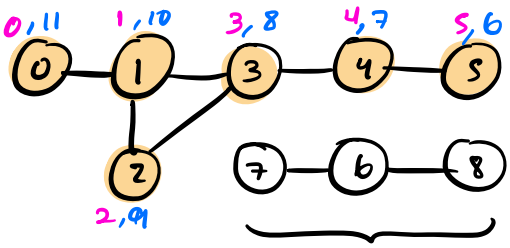
- FINDS THE LEXICOGRAPHICAL FIRST PATH IN THE GRAPH FROM SOURCE VERTEX s TO ANY VERTEX
- FINDS THE SHORTEST PATH IN A TREE (DOES NOT APPLY TO GENERAL GRAPHS)
- TIME COMPLEXITY $\boxed{O(n+m)}$

DESCRIPTION

- GO AS DEEP INTO THE GRAPH AS POSSIBLE AND BACKTRACK ONCE YOU'RE AT A VERTEX WITH UNVISITED ADJACENT VERTICES



DFS(0)



REMAIN UNVISITED BECAUSE THE GRAPH CONTAINS TWO UNCONNECTED COMPONENTS

TIME IN
TIME OUT
+ COLOR → 0: NOT VISITED
→ 1: VISITED
→ 2: EXITED

EDGE CLASSIFICATION (USES TIME IN, OUT, COLOR)

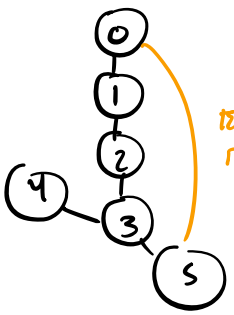
IF v NOT VISITED: FOR THE FIRST TIME

- TREE EDGE: IF v IS VISITED AFTER u → (u,v) IS A TREE EDGE

IF v IS VISITED BEFORE u:

BACK EDGE | FORWARD EDGE | CROSS EDGE

BACK EDGES:



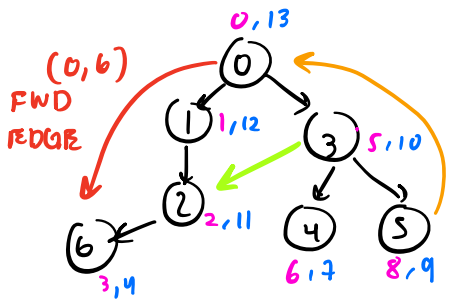
u, v
EDGE (0, 5) IS A BACK EDGE

IF WE REACH V BUT NOT EXITED YET

UNDIRECTED GRAPH

FORWARD EDGE

EXITED V AND TIMEIN[u] < TIMEIN[v]



DIRECTED GRAPH

CROSS EDGE

EXITED V AND TIMEIN[u] > TIMEIN[v]

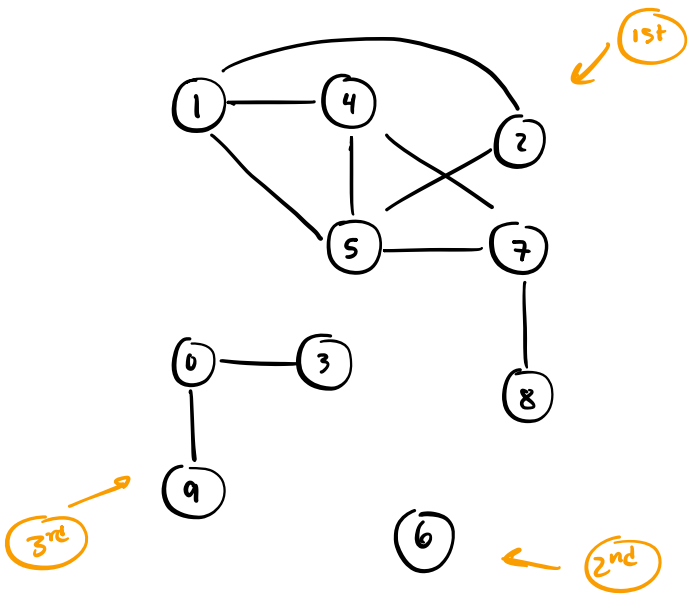
- CYCLES CAN BE DETECTED USING BACK EDGES

NOW TRIVIAL APPLICATIONS : (CHECK CPCFI QUIZLET FLASH CARDS)

1. CHECK IF A VERTEX IN A TREE IS AN ANCESTOR OF SOME OTHER VERTEX
2. TOPOLOGICAL SORTING
3. FIND STRONGLY CONNECTED COMPONENTS IN A DIRECTED GRAPH
4. FIND BRIDGES IN AN UNDIRECTED GRAPH

III CONNECTED COMPONENTS

↳ GROUPS OF VERTICES WITHIN AN UNDIRECTED GRAPH WHERE FOR EACH GROUP EVERY VERTEX CAN BE REACHED FROM ANY NODE AND NO PATH EXISTS BETWEEN DIFFERENT GROUPS



→ 3 CONNECTED COMPONENTS

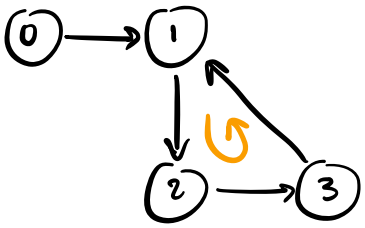
IV. COLORING CONNECTED COMPONENTS

↳ "FLOOD FILL"

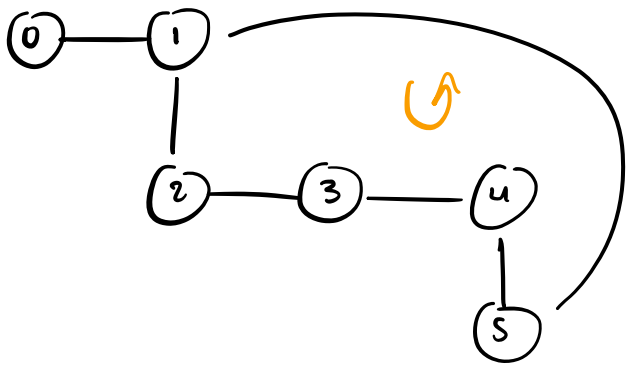
- UVA 469 - WETLANDS OF FLORIDA

V. TOPOLOGICAL SORT : BACKGROUND – ACYCLICITY

- USING DFS WE COLOR THE VERTICES OF A GRAPH
  - 0: NOT VISITED (DEFAULT FOR ALL VERTICES)
  - 1: VISITED
  - 2: VISITED AND EXITED
- WE FIND A CYCLE IF WE MOVE TO A 1 VERTEX



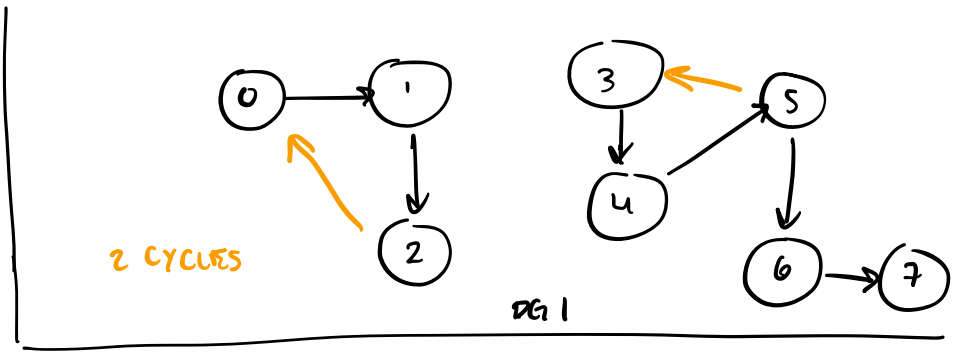
* NOTE : IN UNDIRECTED GRAPH, IF V GETS COLOURED 2, IT WILL NEVER BE VISITED AGAIN.



V. TOPOLOGICAL SORT

- DIRECTED ACYCLIC GRAPH (DAG)
- NUMBER THE VERTICES SO EVERY EDGE LEADS FROM THE VERTEX WITH THE SMALLEST NUMBER TO THE LARGEST

* WE CAN CHECK IF A GRAPH IS ACYCLIC BY COUNTING BACK EDGES



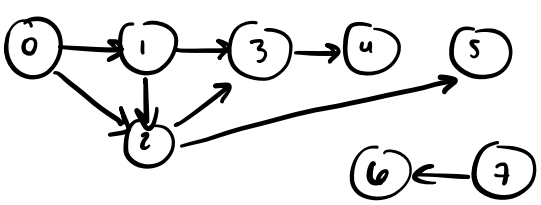
DG 1

TOPOLOGICAL ORDER

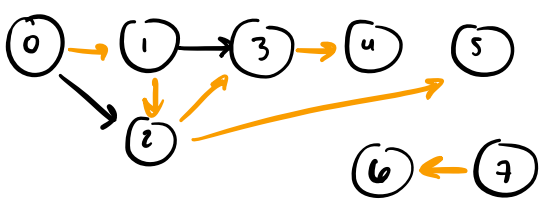
- NON-UNIQUE
- DOESN'T EXIST IF GRAPH CONTAINS CYCLES ($A \to B$, $B \to A$ ∇)

PERMUTATION OF THE VERTICES

- LINEAR ORDERING OF THE VERTICES IN THE DAG SO THAT VERTEX u COMES BEFORE v IF $u \mapsto v$ EXISTS ON THE DAG
- EVERY DAG HAS AT LEAST ONE POSSIBLE T.S.



DG2



7 → 6 → 0 → 1 → 2 → 5 → 3 → 4

T.S. OF DG2

JUST ONE POSSIBLE T.S. OF DG 2

ALGORITHM I : TARJAN's (DFS)

- APPEND SOME NODE u TO THE BACK OF SOME LIST ONLY AFTER STORING ALL SUBTREES BELOW u

ALGORITHM II : KAHN's (BFS)

A DAG HAS AT LEAST ONE VERTEX WITH IN-DEGREE 0 AND ONE VERTEX WITH OUT-DEGREE 0

Step-1: Compute in-degree (number of incoming edges) for each of the vertex present in the DAG and initialize the count of visited nodes as 0.
Step-2: Pick all the vertices with in-degree as 0 and add them into a queue (Enqueue operation)
Step-3: Remove a vertex from the queue (Dequeue operation) and then.

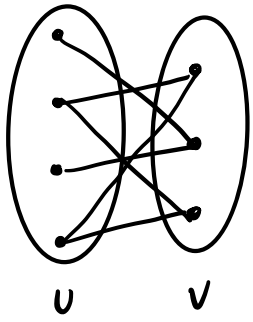
1. Increment count of visited nodes by 1.
2. Decrease in-degree by 1 for all its neighbouring nodes.
3. If in-degree of a neighbouring nodes is reduced to zero, then add it to the queue.

Step 4: Repeat Step 3 until the queue is empty.
Step 5: If count of visited nodes is **not** equal to the number of nodes in the graph then the topological sort is not possible for the given graph.

VI. BIPARTITE GRAPH CHECK

- **BIPARTITE GRAPH**

  ↳ " GRAPH WHOSE VERTICIES CAN BE DIVIDED INTO TWO DISJOINT SETS SO THAT EVERY VERTEX IN U CONNECTS TO ONE IN V "



- **CHECK IF BIPARTITE GRAPH :**

  (1) THEOREM : A GRAPH IS BIPARTITE IF AND ONLY IF ALL OF ITS CYCLES HAVE EVEN LENGTH

  (2) A GRAPH IS BIPARTITE IF ITS TWO - COLORABLE