GRAPH TRAVERSAL

QUESTION - 1

Inconsistent Subset

There are N variables $x_1, x_2, \ldots, x_N$ and M relations of the form $x_i < x_j$ where i != j. A subset S of relations is called inconsistent if there does not exist any assignment of variables that satisfies all the relations in S. e.g, $\{x_1 < x_2, x_2 < x_3, x_3 < x_1\}$ is inconsistent. You need to find if there is an inconsistent subset of M.

Inconsistent Subset (Contd..)

Hint 1

Think of creating a directed graph

Inconsistent Subset (Contd..)

Hint 1

Think of creating a directed graph

- $V = x_1, x_2, \ldots, x_N$
- $E = \{(x_i,\ x_j)$ iff $(x_i < \ x_j)$ in $S\}$

Inconsistent Subset (Contd..)

Hint 2

How can you detect inconsistency in the graph?

Inconsistent Subset (Contd..)

Hint 2

How can you detect inconsistency in the graph?

- Detect cycle in the graph

Inconsistent Subset (Contd..)

Solution

Use DFS to detect cycle

- if we encounter a vertex which is already on the stack, we found a loop

```
all vertices are unvisited
create a stack S
push v onto S
while S is non-empty
    peek at the top u of S
    if u has neighbour which in S
        there is a cycle
    else if u has a unvisited neighbour w
        push w onto S
    else mark u as finished and pop S
```

QUESTION -2

Two Coloring

Suppose we have an undirected graph and we want to color all the vertices with two colors ***red*** and ***blue*** such that no two neighbors have the same color. Design an **O(V+ E)** time algorithm which finds such a coloring if possible or determines that there is no such coloring.

Two Coloring (Contd..)

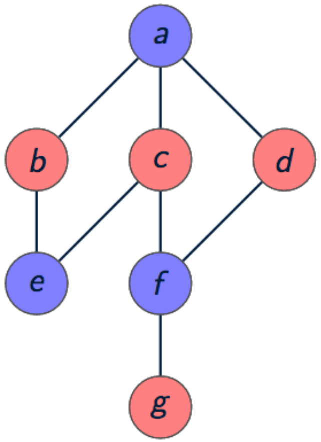
Hint 1

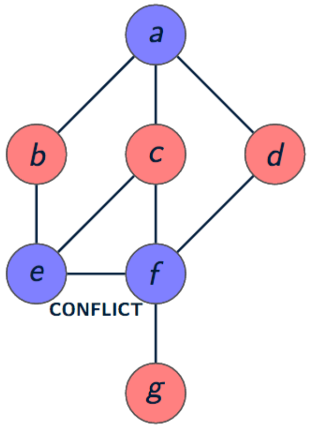
Can you use BFS or DFS?

Two Coloring (Contd..)

Solution

Let’s use BFS





Two Coloring (Contd..)

Solution

```
Start with an arbitrary vertex v, color it, and run BFS from there.
v <- remove(queue)
for each neighbor w of v
    if w is uncolored
        give it color opposite(color(v)) and put w into queue
    else if w is colored
        compare to color of v
        if different
            then move on to next w.
        if same
            halt with failure.
```

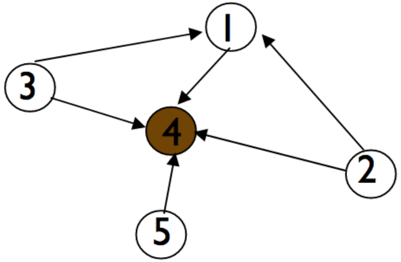
QUESTION -3

Universal Sink

When an adjacency-matrix representation is used, most graph algorithms require time $\Omega(V^2)$, but there are some exceptions.

Given an adjacency matrix for a directed graph G, determine whether G contains a ***universal sink*** *(a vertex with in-degree* $|V| - 1$ *and out-degree* 0*)* in time $\mathbf{O(V)}$.

Universal Sink (Contd..)



	1	2	3	4	5
1	0	0	0	1	0
2	1	0	0	1	0
3	1	0	0	1	0
4	0	0	0	0	0
5	0	0	0	1	0

Universal Sink (Contd..)

Hint 1

How can we determine whether a given vertex u is a universal sink?

Universal Sink (Contd..)

Hint 1

How can we determine whether a given vertex u is a universal sink?

- The u-row must contain 0’s only
- The u-column must contain 1’s only
- A[u][u]=0

Universal Sink (Contd..)

Hint 1

```
Is-Sink(A, k)
    let A be |V|×|V|
    for j = 1 to |V|
        if a_kj == 1
            return FALSE
    for i = 1 to |V|
        if a_ik == 1 and i ≠ k
            return FALSE
    return TRUE
```

Universal Sink (Contd..)

Hint 1

```
Is-Sink(A, k)
    let A be |V|×|V|
    for j = 1 to |V|
        if a_kj == 1
            return FALSE
    for i = 1 to |V|
        if a_ik == 1 and i ≠ k
            return FALSE
    return TRUE
```

How long would it take to determine whether a given graph contains a universal sink if you were to check every single vertex in the graph?

Universal Sink (Contd..)

Hint 1

```
Is-Sink(A, k)
    let A be |V|×|V|
    for j = 1 to |V|
        if a_kj == 1
            return FALSE
    for i = 1 to |V|
        if a_ik == 1 and i ≠ k
            return FALSE
    return TRUE
```

How long would it take to determine whether a given graph contains a universal sink if you were to check every single vertex in the graph?

$O(V^2)$

Universal Sink (Contd..)

Hint 1

```
Is-Sink(A, k)
    let A be |V|×|V|
    for j = 1 to |V|
        if a_kj == 1
            return FALSE
    for i = 1 to |V|
        if a_ik == 1 and i ≠ k
            return FALSE
    return TRUE
```

How long would it take to determine whether a given graph contains a universal sink if you were to check every single vertex in the graph?

$O(V^2)$

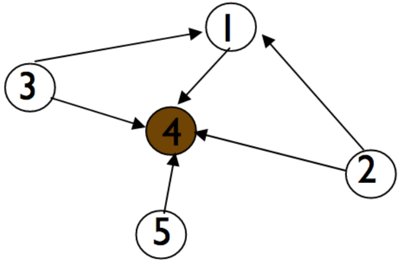
Can you suggest an algorithm in O(V) ?

Universal Sink (Contd..)

Hint 2

- If A[u][v]=1, then u cannot be a universal sink
- If A[u][v]=0, then v cannot be a universal sink

Universal Sink (Contd..)



	1	2	3	4	5
1	0	0	0	1	0
2	1	0	0	1	0
3	1	0	0	1	0
4	0	0	0	0	0
5	0	0	0	1	0

Universal Sink (Contd..)

Solution

```
Universal-Sink(A)
    let A be |V|×|V|
    i = j = 1
    while i≤|V| and j≤|V|
        if a_ij == 1
            i = i + 1
        else
            j = j + 1
    if i > |V|
        return “no universal sink”
    elsif Is-Sink(A,i) == FALSE
        return “no universal sink”
    else
        return i “is a universal sink”
```

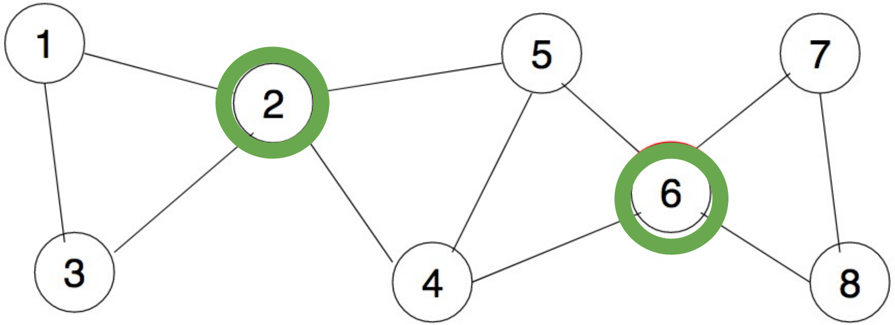
QUESTION - 4

Articulation Point

Let G = (V, E) be an undirected graph. A vertex $v \in V$ is called a ***cut vertex*** or an ***articulation point*** if the removal of v (and all edges incident upon v) increases the number of connected components in G.

Find all cut vertices in G in O(V+E)

Articulation Point (Contd..)



Articulation Point (Contd..)

Hint 1

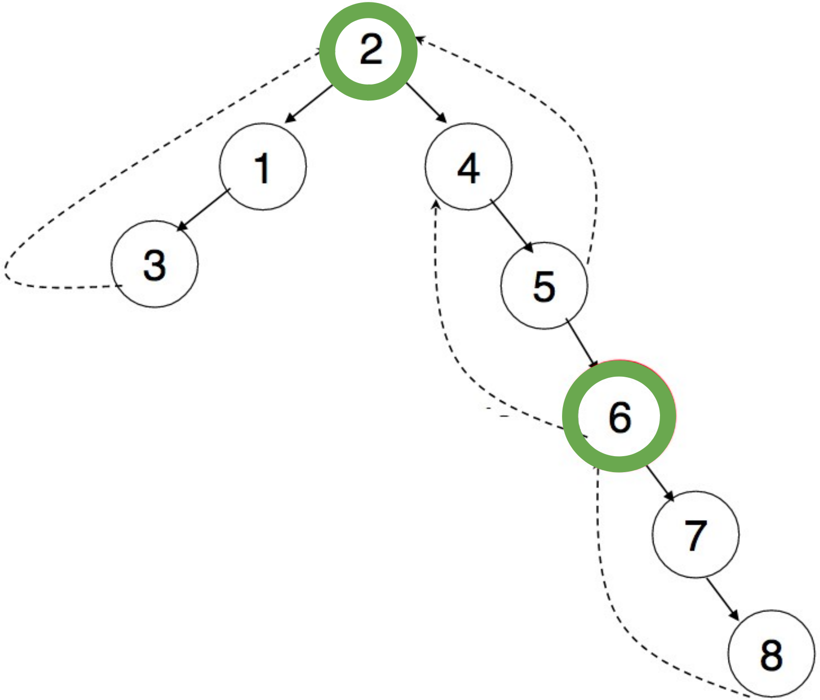
Use DFS tree and back edge

Articulation Point (Contd..)

Hint 2

- The ***root*** of the DFS tree is an articulation point if and only if it has ***at least two children (subtree)***
- A ***non-root vertex*** v of a DFS-tree is an articulation point of G if and only if has a child s such that there is ***no back edge*** from s or any descendant of s to a proper ancestor of v
- ***Leaves*** of a DFS-tree are ***never*** articulation points

Articulation Point (Contd..)



Articulation Point (Contd..)

- How do you know a subtree has a back edge climbing to an upper part of the tree?

  low[v] = min { discover[v]; discover[w] : (u,w) is a back edge for some descendant *u* of *v* }

- *u* is articulation point if it has a descendant *v* with

  low(v)>=discover [u]

Articulation Point (Contd..)

Solution

```
GetArticulationPoints(i, d)
    visited[i] = true
    discover[i] = d
    low[i] = d
    childCount = 0
    isArticulation = false
    for each ni in adj[i]
        if not visited[ni]
            parent[ni] = i
            GetArticulationPoints(ni, d + 1)
            childCount = childCount + 1
            if low[ni] >= discover[i]
                isArticulation = true
            low[i] = Min(low[i], low[ni])
        else if ni != parent[i]
            low[i] = Min(low[i], discover[ni])
    if (parent[i] != null and isArticulation) or (parent[i] == null and childCount > 1)
        Output i as articulation point
```

QUESTION - 5

Longest Path

You are given an ***undirected acyclic graph*** G(V,E). You need to find a pair of vertices (i,j) such that the ***length of the path between i and j is maximum*** among all such pairs. The length of a path is the number of edges on the path.

Longest Path (Contd..)

Hint 1

There is a trivial $O(V.(V + E))$ algorithm to solve this

Longest Path (Contd..)

Hint 1 sol

```
Run BFS V times starting from each vertex
Find max
```

O(V(V+E))

Longest Path (Contd..)

Hint 1 sol

```
Run BFS V times starting from each vertex.
Find max
```

$O(V(V+E))$

Q. Can you give an $O(V + E)$ algorithm?

Longest Path (Contd..)

Hint 2

Need to do BFS twice. How?

Longest Path (Contd..)

Hint 2 sol

```
Start BFS from any node x and find a node with the longest
distance from x
Run another BFS from this endpoint to find the actual longest
path.
```

Longest Path (Contd..)

Q. Prove that the end point found after the first BFS must be an end point of the longest path.

END