**ASSIGNMENT 3**

**SOLUTION**

**Question 1:**  
  
To provide a counterexample to Professor Sabatier's conjecture, consider the following scenario:

Let G be a connected, undirected graph with four vertices V = {A, B, C, D} and the following weight function w defined on the edges:

w(A-B) = 2

w(A-C) = 3

w(B-C) = 1

w(B-D) = 4

w(C-D) = 5

Now, consider the subset A = {A-B, B-C} which is included in some minimum spanning tree for G. Let (S, V - S) be a cut of G such that S = {A, B} and V - S = {C, D}.

In this case, (A-C) is a safe edge for A crossing the cut (S, V - S). However, (A-C) is not a light edge for this cut since w(A-C) = 3, which is greater than the weight of the other edge crossing the cut, w(B-C) = 1.

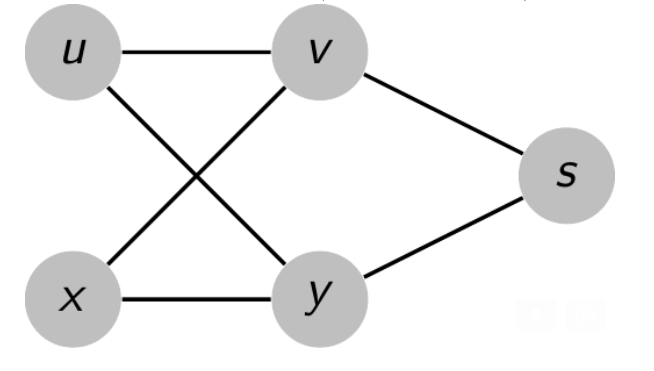
Therefore, the counterexample disproves Professor Sabatier's conjecture.

**Question 2:**

If we represent the input graph for BFS using an adjacency matrix, the running time of BFS would be O(V^2), where V is the number of vertices in the graph.

With an adjacency matrix representation, BFS requires examining each cell in the matrix to determine the adjacency between vertices. Since the matrix has V rows and V columns, a total of V^2 cells need to be checked.

**Question 3:**

Let *G* be the graph shown below (*s* be the source vertex.):  


To demonstrate that for each vertex v ∈ V, the unique simple path from s to v is a shortest path in G, we'll go through each vertex v and find its shortest path from s:

1. For vertex u: s - v - u (length: 2)

2. For vertex v: s - v (length: 1)

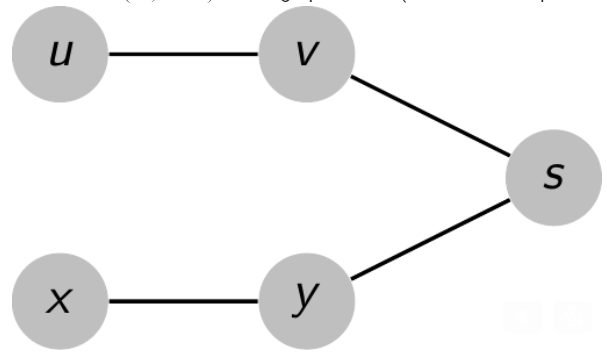
3. For vertex s: s (length: 0)

4. For vertex y: s - y (length: 1)

5. For vertex x: s - y - x (length: 2)

As we can see, the unique simple paths from s to all other vertices are indeed the shortest paths in G.

Let *Gπ* = (*V*,*E* *π* ) be the graph shown (it can never be produced by running BFS on G):



If we run BFS on the graph G starting from the source vertex s, if v is explored first then y-x edge is not produced and if y is explored first then y-x edge is not produced.

Therefore, we have an example of a graph where the unique simple paths from the source vertex s to all other vertices are the shortest paths, but the set of edges E\_n cannot be produced by running BFS on G, regardless of the vertex ordering in the adjacency lists.  
  
**Question 4:**

**Directed:**  
DFS(G):

for each vertex u in G:

u.color = WHITE

u.parent = NULL

edge\_types = {} // Dictionary to store edge types

time = 0

for each vertex u in G:

if u.color == WHITE:

DFS-Visit(u, edge\_types)

DFS-Visit(u, edge\_types):

time = time + 1

u.d = time

u.color = GRAY

for each v in G.adj[u]:

if v.color == WHITE:

edge\_types[(u, v)] = "Tree Edge" // New tree edge discovered

v.parent = u

DFS-Visit(v, edge\_types)

else if v.color == GRAY:

edge\_types[(u, v)] = "Back Edge" // Back edge to a gray vertex

else if v.color == BLACK:

if u.d < v.d:

edge\_types[(u, v)] = "Forward Edge" // Forward edge to a black vertex

else:

edge\_types[(u, v)] = "Cross Edge" // Cross edge to a black vertex

u.color = BLACK

time = time + 1

u.f = time

// Print the edge and its type

for each (u, v) in edge\_types:

print("Edge:", u, "->", v, "Type:", edge\_types[(u, v)])

**Undirected:**  
  
DFS(G):

for each vertex u in G:

u.color = WHITE

u.parent = NULL

edge\_types = {} // Dictionary to store edge types

time = 0

for each vertex u in G:

if u.color == WHITE:

DFS-Visit(u, edge\_types)

DFS-Visit(u, edge\_types):

time = time + 1

u.d = time

u.color = GRAY

for each v in G.adj[u]:

if v.color == WHITE:

edge\_types[(u, v)] = "Tree Edge" // New tree edge discovered

v.parent = u

DFS-Visit(v, edge\_types)

else if v.color == GRAY and v.parent != u:

edge\_types[(u, v)] = "Back Edge" // Back edge to a gray vertex

~~else if v.color == BLACK:~~

~~if u.d < v.d:~~

~~edge\_types[(u, v)] = "Forward Edge" // Forward edge to a black vertex~~

~~else:~~

~~edge\_types[(u, v)] = "Cross Edge" // Cross edge to a black vertex~~

u.color = BLACK

time = time + 1

u.f = time

// Print the edge and its type

for each (u, v) in edge\_types:

print("Edge:", u, "-", v, "Type:", edge\_types[(u, v)])  
  
**Question 5:**  
  
DFS(G, start):

for each vertex u in G:

u.color = WHITE

u.parent = NULL

stack = [] // Stack to keep track of vertices

time = 0

stack.push(start)

while stack is not empty:

u = stack.pop()

if u.color == WHITE:

u.color = GRAY

time = time + 1

u.d = time

// Process vertex u

for each v in G.adj[u]:

if v.color == WHITE:

v.parent = u

stack.push(v)

u.color = BLACK

time = time + 1

u.f = time

**Question 6:**

Let's consider the following directed graph G:

s -> w

w -> v

w -> u

u -> w

Here, we have a path from u to v: u -> w -> v.

Now, let's perform a depth-first search (DFS) on G. Starting from s, we explore the graph as follows:

1. Set s.d = 1 (discovery time of s).

2. Explore the adjacent vertex w.

3. Set w.d = 2 (discovery time of w).

4. Explore the adjacent vertices v and u.

5. Set v.d = 3 (discovery time of v).

6. Backtrack to the previous vertex, w, as v has no outgoing edges.

7. Set u.d = 4 (discovery time of u).

8. Explore the adjacent vertex w.

9. Since w has already been discovered, we don't update its discovery time.

In this case, we can observe that u.d = 4, and v.d = 3. According to the conjecture, if u.d < v.d, then v should be a descendant of u in the depth-first forest. However, in this counterexample, v is not a descendant of u. They are on separate branches of the depth-first forest.

Therefore, this counterexample disproves the conjecture.

**Question 7:**

Consider a directed graph G with vertices w, u, and v and edges={w->u,u->w,w->v}. We will perform a depth-first search starting from vertex w and traverse the graph in the following order: w-u, u-w, w-v.

During this traversal, the discovery and finish times for the vertices w, u, and v are as follows:

1. Discover w (w.d = 1).

2. Discover u (u.d = 2).

3. Finish u (u.f = 3).

4. Discover v (v.d = 4).

5. Finish v (v.f = 5).

6. Finish w (w.f = 6).

In this traversal, we start at vertex w and follow the directed edge from w to u. Then, we backtrack from u to w and proceed to vertex v. The inequality v.d < u.f does not hold in this case because v.d = 4 and u.f = 3, contradicting the conjecture.