Nafis Abeer EC 330 HW 7

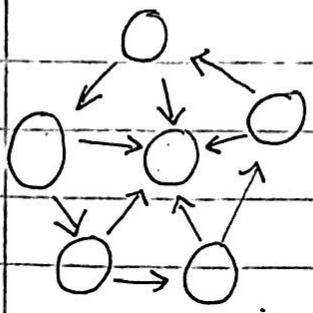
1) In order for a graph to be bipartite, the graph $G(V,E)$ has its vertex (V) set divided into $V = L \cup R$ where L and R are disjoint and all Edges (E) go between L and R. Any element of L does not have an edge connecting to another element in L. Pseudo code to check for a bipartite:

- will be using a similare idea as breadth first search
- will color every vertex white to indicate that they have not been visited.
- $G.adj[u]$ $u \in V$, is used to indicate all adjacent vertices of vertex u.
- color Red refers to Partition 1
- color Blue refers to partition 2
- Run through every vertex and if it contains neighbors that has already been assigned the same color as itself, return false.
- If end of function is reached, return true because no adjacent vertecies of any given vertex was the same color as itself.
- using an adjacency list will give a run time of $\Theta(V+E)$.

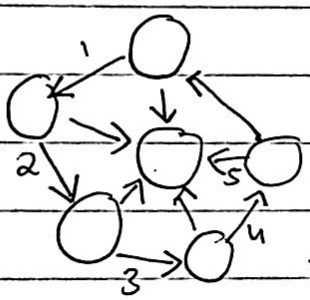
```
bool Bipartite(G, s){// s is starting node
    for(each vertex u ∈ G.V)
        u.Color = white;
    S.color = RED; // source is red as in partition 1.
    Q = ∅; Enqueue(Q, s);
    while(Q ≠ ∅){
        u = Dequeue(Q); // this complies queue overtime
        if (any G.adj[u] == u) // if there is a self loop.
            return false;
        for(each v ∈ G.adj[u]){// loop through every neighbor
            if (v.color == white && u.color == Red){
                v.color = blue;    // assign the opposite
                Enqueue(Q, v); }   // Color
            else if (v.Color == white && u.color == Blue){
                v.color = RED;     // assign the opp. color
                Enqueue(Q, v); }   // store in queue
            else if (v.color == u.color) // same partition
                return false;
        }
    }
    return true;
}
// in the case v is already the opposite color, the
// for loop just continues
```

Scanned with Mobile Scanner

2)



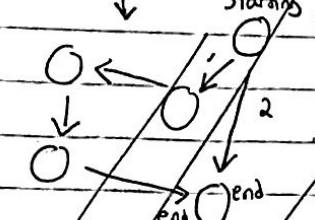
A Eulerian Cycle is impossible as there are many edges (one for every vertex) that leads to the celebrity, but in order to visit every edge one would have to reach each one of those edges. This is clearly impossible since you can't leave the celebrity vertex once you reach it. In a cycle you must be able to come back to the starting node once you leave it down one edge so its a closed loop traced without "lifting off your pencil." You'd have to "lift your pencil" every time you reach celebrity.



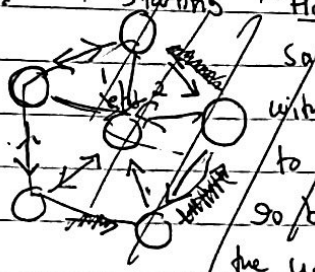
A hamiltonian Path would be possible by following the numbered edges, but a cycle is a "closed loop" that comes back to the starting node which is clearly impossible because no edge leads out of a celebrity. A hamiltonian Cycle for this reason is impossible.

Scanned with Mobile Scanner

Visiting every edge will require 1 to trace back on every edge in a given path

2) 

Eulerian cycle (impossible w/ celebrity)
In order to have a eulerian cycle, the starting node must also be the end node. In the case where a celebrity is present, the celebrity node is always the end node since no edge leads out from it. For this reason it will be impossible to visit every edge because say we go down path 1 from starting node in the case above. We stop at the celebrity and we never get to visit path 2. The same is true for the path 2, as we never visit any edge in path 2.



Hamiltonian cycle (impossible w/ celebrity)
Same as the eulerian case, in order to visite every node without "lifting the pen off the paper" would require one to reach the celebrity (end) node from the starting node then go back and visit certain nodes again to reach the rest of the unvisited nodes

3) weight $w(p)$ of path (p) = sum of weights of its constituents

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Shortest path weight $\delta(u,v)$ from u to v is

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

```
Dijkstra(G, w, s)
Initialize_single_source(G, s)
S = ∅
Q = G.V                                   O(V) inside O(V) gives O(V^2) + O(EV)  ← this would be determining lower bound
While Q ≠ ∅ // O(V)                       so ~~O(EV)~~ over all
    u = extract-min(Q) → O(V) // once per vertex // for linked list   | Θ(V^2) |
    S = S ∪ {u}
    for each vertex v ∈ G.adj[u] // E times
        Relax(u, v, w)   O(1)
```

Scanned with Mobile Scanner