Checking for cycles in a Graph

Wednesday, February 12, 2020 3:35 PM

Expected running time: O(m+n)

Algorithm Design

- DFS can be used to explore to tree, with some additional book keeping - we maintain the parents of every node when its explored, the parent of the starting node would be NULL
- Say we are starting from a vertex s, and exploring all the adjacent vertices leading from s
- When the adjacent vertex is not seen before, we proceed with recursive DFS
- If the adjacent vertex is seen before, and the adjacent vertex has no parent child relationship with the starting point of the DFS in the current iteration, then there is a cycle
- We break the loop there and backtrack the graph from the starting point of the current iteration, until we reach the adjacent vertex and print the same

Pseudo code

```
G=(V,E)
Visited=[]
Parent=[]
Stack=[]

v=For all nodes in V:
    If v is not Visited:
        If Stack is empty:
            Stack=dfs(graph,v,stack)
        If Stack is not empty:
            Stack=dfs(graph,last entry in stack, stack \ last entry)

Dfs(graph,start,stack):
    If stack is empty:
        Visited[start]=True
        Parent[start]=None
    Stack=stack+start
    For all v such that (start,v) in V:
        If v is visited and parent(start) != v and parent(v)!= start:
            Print('cycle detected')
            Cycle=[]
            Head = v
            Tail=start
            Cycle=[]+[Head,Tail]
            While parent[Tail]!=Head:
                Tail=Parent[Tail]
                Cycle=Cycle+[Tail]
            Exit recursion
        If v is not visited:
            Visited[v]=True
            Parent[Vertex]=Start
            Stack=dfs(graph,vertex,stack)
            Break
    Return(stack)
```

Note: This logic would work for both directed graph and also undirected graph, provided both way undirected connections are represented in the linked list. Eg: if there is an edge between node 3 and 4, 3 should be present in the linked list pertaini ng to node 4 and vice versa