# CS 367 Homework 6

Mingren Shen ( mshen32@wisc.edu )

## Question 1

Below is an *incomplete* definition of the `Graphnode` class.

```
class Graphnode<T> {

    private boolean visitMark;
    private List<Graphnode<T>> successors;

    public boolean getVisitMark() {
        return visitMark;
    }

    public void setVisitMark(boolean mark) {
        visitMark = mark;
    }

    public List<Graphnode<T>> getSuccessors() {
        return successors;
    }
}
```

**Write the hasSelfCycle method whose header is provided below.** This method takes a `Graphnode node` and returns true iff there exists a path from `node` to itself (i.e., if there is a cycle in the graph that starts and ends at `node`). Your implementation must be based on the depth-first search algorithm (i.e., modify DFS to implement `hasSelfCycle`) and must exit early if possible.

`public boolean hasSelfCycle( Graphnode<T> node )`

You may assume that all the nodes in the graph have been marked unvisited prior to the `hasSelfCycle` method being called and that there are no self edges (i.e., there are no edges from a node to itself). You may not modify the `Graphnode` class. You may find it helpful to write an auxiliary method.

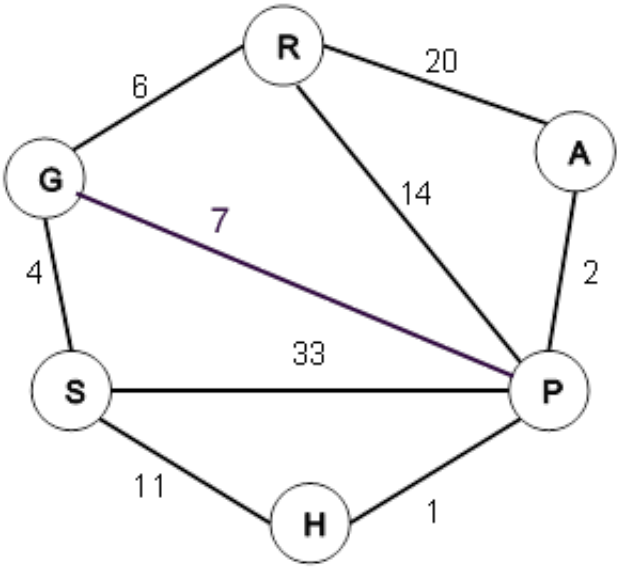
### Answer

```
public boolean hasSelfCycle( Graphnode<T> node )
    {
        // call helper method
        return hasSelfCycleAux( node, node );
    }

    private boolean hasSelfCycleAux( Graphnode<T> n, Graphnode<T> reference )
    {
        // set current node true
        n.setVisitMark( true );
        // loop all the children
        for ( Graphnode<T> m : n.getSuccessors() )
        {
            // if found child is reference node, return true
            if ( m == reference )
            {
                return true;
            }
            else
            {
                // if the child is not visited, visit
                if ( !m.getVisitMark() )
                {
                    // depth first search
                    if ( hasSelfCycleAux( m, reference ) )
                    {
                        return true;
                    }
                }
            }
        }
        return false;
    }
```

## Question 2

Given the following graph with nodes having character labels and edges having non-negative integer weights:



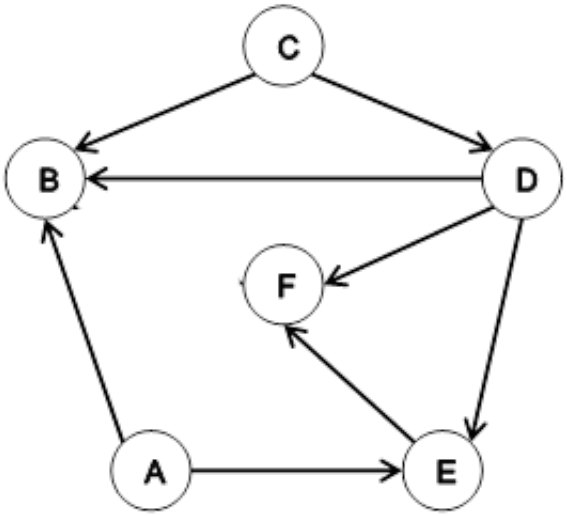

**Trace Dijkstra's algorithm, starting at node S,** by filling in the remaining rows in the table below. Each row in the table represents one iteration of the of the algorithm, so use as many or as few rows as needed for the algorithm to complete.

### Answer

| visited nodes and their shortest distances from start | dist values for nodes in U (only finite values, listed in increasing order) |
|---|---|
| - | (0,S) |
| S:0 | (4,G), (11,H), (33,P) |
| S:0, G:4 | (10,R), (11,H), (11,P) |
| S:0, G:4, R:10 | (11,H), (11,P), (30,A) |
| S:0, G:4, R:10, H:11 | (11,P), (30,A) |
| S:0, G:4, R:10, H:11, P:11 | (13,A) |
| S:0, G:4, R:10, H:11, P:11, A:13 | - |

# Question 3

Give **three distinct topological orderings** of the following directed acyclic graph:




## Answer

(1) C -> D -> A -> B -> E -> F

(2) C -> D -> A -> E -> F -> B

(3) A -> C -> D -> E -> F -> B