Detecting Cycles in graphs

- One can detect a cycle in a graph using a small modification of DFS
- A graph contains a cycle iff it contains a back edge
- An edge (u, v) is said to be a back edge if v is an ancestor of u.
- v is an ancestor of u iff v has a gray color.

Detecting cycles in graphs

```
function DFS-VISIT(adj, u)
    u.color ← GRAY
    time ← time + 1
    u.d ← time
    foreach v ∈ adj[u] do
        if v.color = WHITE then
            v.p ← u
            DFS-VISIT(adj, v)
        else if v.color = GRAY ∧ v ≠ u.p then
            cycle ← true
    u.color ← BLACK
    times ← time + 1
    u.f ← time
```

Island

Given a 2-d array, *int*[][] *grid*, whose values are either 1 (land) or 0 (water), write a function to return the number of islands in the grid. An island is a land surrounded by water and is formed by connecting adjacent lands horizontally or vertically (not diagonally). Assume that the grid itself is surrounded by water. You may use auxillary functions.

Example1 :

```
1   int numIslands(int[][] grid) {
2     // write your code here
3   }
4   int[][] grid={
5     {1,1,0,0,0},
6     {1,1,0,0,0},
7     {0,0,1,0,0},
8     {0,0,0,1,1},
9     };
10  numIslands(grid); //should return 3
```

Example 2:

Given a directed graph $G = \langle V, E \rangle$, a source vertex $s \in V$, destination vertex $d \in V$, write the Java method:

`int totalPaths(Graph g, int s, int d, int m)`

That returns the number of paths from s to d having exactly m edges. Example, in the graph shown below `totalPaths(g,0,3,4)` should return 3 since there are 3 paths from 0 to 3 with length 4:

a