CS2010 Semester 1 2012/2013

Data Structures and Algorithms II

Tutorial 04 - Graph DS & Graph Traversal 1

For Week 06 (17 September - 21 September 2012)

Released: Wednesday, September 12, 2012

Document is last modified on: September 19, 2012

1 Introduction and Objective

This tutorial marks the second part of CS2010: Graph. This tutorial covers basic graph data structures and traversal algorithms as discussed in Lecture 05 (Week05+06).

As we will have Quiz 1 that tests the first part of CS2010 is this coming Saturday, we have made the questions in this tutorial slightly shorter so that students can discuss Quiz 1 related materials with the tutor if needed.

Note: Use `http://www.comp.nus.edu.sg/~stevenha/visualization/representation.html` and `http://www.comp.nus.edu.sg/~stevenha/visualization/dfsbfs.html` to *verify* the answers of some questions in this tutorial. However during written tests, you have to be able to do this by yourself.

2 Tutorial 04 Questions

Graph Properties

Q1. Does the following graph contain a non-simple cycle? If so indicate that cycle.
A cycle is defined as simple if it does not contain repeated vertices except for the starting and ending vertex.

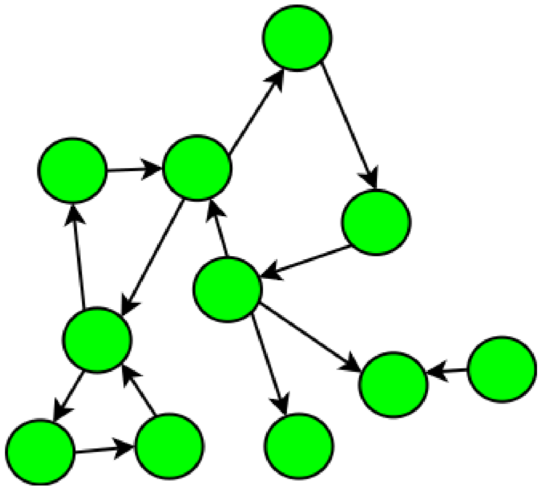

Figure 1:

Ans: **Yes**.
The cycle is indicated by the blue and red nodes. The red nodes represent the repeated vertices in the cycle. Since not only the start and end vertices are repeated, the given cycle is not simple.

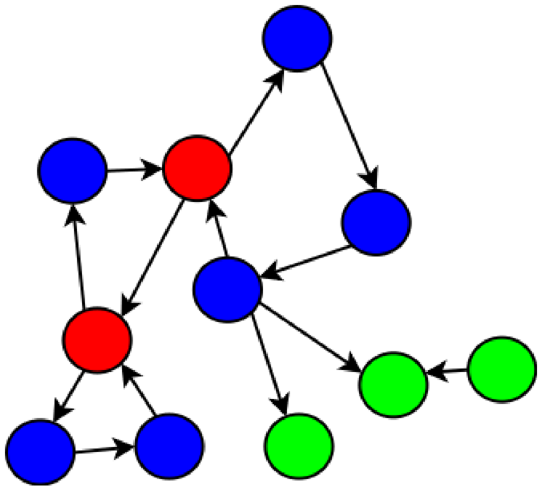

Figure 2:

Q2. How many edges are there in a complete bipartite graph of m nodes and n nodes respectively in each set?

A bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V; but there is no edge between vertices in U and also no edge between vertices in V.

A complete bipartite graph is a bipartite graph where every node in one set U is connected to every other node in the other set V.

Ans: $m \times n$ edges. Each node in the set with m nodes is connected to n nodes in the other set.

Q3. Give an algorithm to determine whether a graph is bipartite.

Ans:
Note that for any given node in a bipartite graph, if it were to be put into one set, then every node it is linked to by an edge must be put into the other set.
Thus we can perform a modified BFS/DFS on the given graph such that as we perform the search, we set the current node being visited to the opposing parity as the node visited before it. After setting its parity, we check whether any of the nodes it is linked too has the same parity. If they do, then the graph cannot be bipartite, else continue the search. The algorithm shown below is the modified DFS.

Algorithm 1 Bipartite(G = (V,E))

```
visited[1..n] ← 0
flag ← true
for i = 1 → n do                                  ▷ |V| = n
    if visited[i] == 0 then
        call dfs(i,1)
    end if
end for
return flag
```

Algorithm 2 dfs(v,c)

```
if flag == false then
    return                                        ▷ Just terminate recursion
end if
visited[v] ← c
for each node w such that (v, w) in E do
    if visited[w] == 0 then
        dfs(w, −1 ∗ c)                            ▷ alternate c from 1 to -1 to 1...
    else if visited[w] == c then
        flag ← false
        return                                    ▷ just terminate recursion
    end if
end for
```

Q4. List the nodes in the queue at each step when performing a BFS of the graph below starting from source vertex 0. At the end, give the spanning tree induced by the BFS. (Assume that neighbors are visited in increasing order). Compare your answer with the animation produced by the algorithm visualization tool!

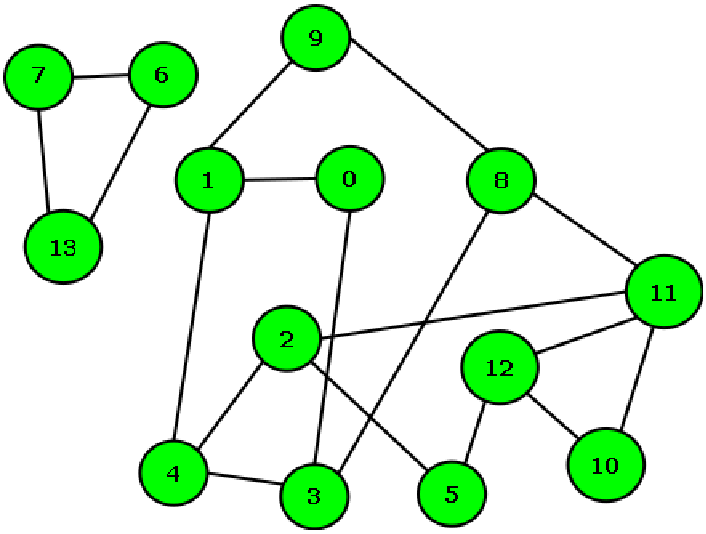

Figure 3:

Ans:

Q={0}
Q={1,3}
Q={3,4,9}
Q={4,9,8}
Q={9,8,2}
Q={8,2}
Q={2,11}
Q={11,5}
Q={5,10,12}
Q={10,12}
Q={12}
Q={}

Note that vertices 6, 7, 13 are NOT reachable from source vertex 0, so they are not touched yet.

BFS Spanning Tree (vertices 6, 7, 13 are excluded).

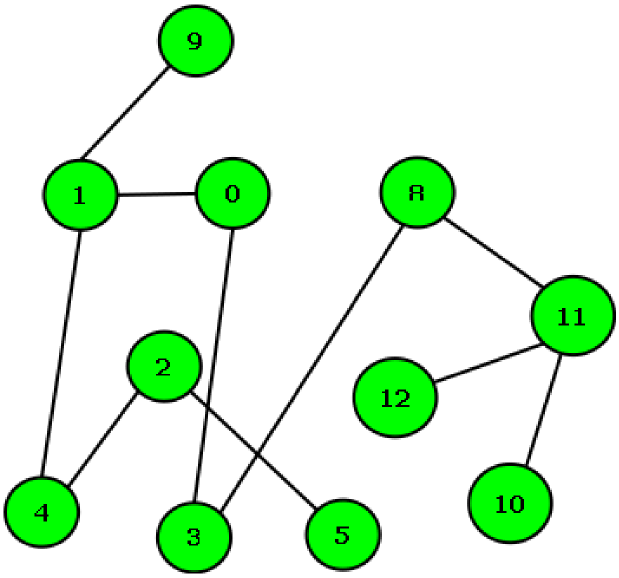

Figure 4:

Q5. Given a map of cities represented as a graph where cities are nodes and edges are routes between cities, give an algorithm to generate any valid path for a vehicle to go from a city a to another city b (you can assume that there is always some way to get from any city to every other city).

Ans:

As we just need any valid path and the graph is connected, then if we perform a graph traversal (either DFS or BFS) using city a as the source, we can obtain a spanning tree of the graph (this information is stored in the predecessor/parent array p). Then, use this predecessor information to re-produce the path from city b back to city a. This has been discussed in Lecture 05.
Note that any other graph algorithm that produces spanning tree from a starting vertex to all other vertices in that graph (component) is also valid, like Prim's, Bellman Ford's, Dijkstra's, etc.

Problem Set 3

Q6. Discussion of PS3 Subtask 1 (a quick one)

Ans:

Finding 'important rooms' in this problem is actually finding the 'cut vertices' or 'articulation points' of an undirected graph: A vertex that will disconnect the graph if removed. In a tree, each non leaf vertex is a cut vertex, because removing such vertex will disconnect the tree. It is easy to determine if a vertex is a leaf vertex or not. A non leaf vertex will have degree > 1. The solution for this Subtask 1 is thus very simple: Iterate through each vertex and count its degree. If it is > 1, check if it has the lowest rating score so far. Note that the output -1 can only occur if the tree has $\leq$ 2 vertices (do you understand why?).

Subtask 1 is essentially just for testing students understanding on Adjacency List data structure.

Quiz 1

Free and Easy time to discuss past year Quiz 1 problems/solutions.