COMPSCI 330: Design and Analysis of Algorithms 09/04/2014

Lecture 4

Lecturer: Debmalya Panigrahi *Scribe: Ang Li*

1 Overview

This lecture covers basic definitions and algorithms concerning graph theories. Content covered includes different types of graph, definitions of a path and a cycle, graph representations in adjacency matrix or adjacency list, applications of graph theories, and the Depth First Search (DFS) algorithm.

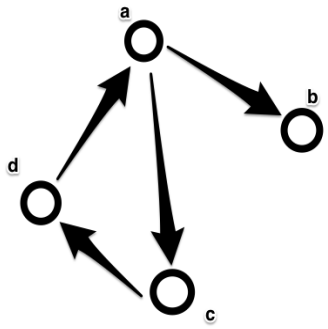
2 Definitions

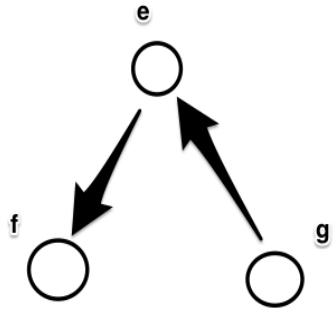
2.1 Definition of a graph

A graph can be viewed as a collection of the set of vertices and the set of edges. Mathematically,

$$G = (V,E),\ E = \{(u,v) : u \in V, v \in V\},\ |V| = n,\ |E| = m \tag{1}$$

where V is the set of vertices, E is the set of edges, n is the number of vertices, and m is the number of edges.





2.2 Different types of graph

1. Directed graph. Edges in this type of graph are directed, i.e. pairs of vertices are ordered.
2. Graph with multiple parallel edges, also known as multi-graph.
3. Graph whose edges have weights, costs, or capacities.
4. Graph that contains self-loops, i.e. edges from a vertex to itself.

2.3 Path

A path is a sequences of edges

$$(u_1, v_1),\ (u_2, v_2),\ ...,\ (u_k, v_k)\ \textit{such that}\ v_1 = u_2,\ v_2 = u_3,\ ...,\ v_{k-1} = u_k \tag{2}$$

For a path, it is important that you remember your start vertex, since starting from a different vertex results in a different path.

2.4 cycle

A cycle is a path that satisfies an additional constraint

$$v_k = u_1 \tag{3}$$

in the path definition above.

For a cycle, it is not important where you start since starting from any vertex in the cycle results in the same list of vertices.

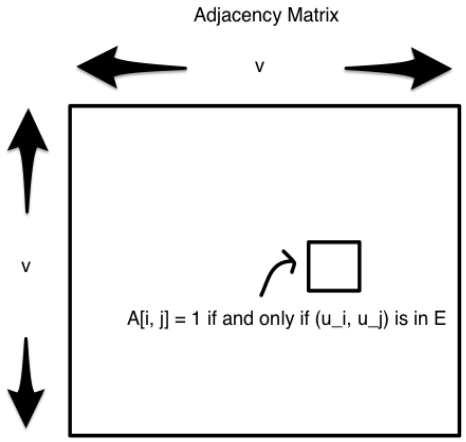
2.5 Connectivity between vertices

A vertex u is connected to a vertex v if there is a path starting at u and ending at v. Note that the order is important. A path from u to v does not have the same meaning as a path from v to u. Some vertices can become connected when a directed graph is changed to an undirected graph.

3 Graph representations

3.1 Adjacency Matrix

Adjacency matrix uses a $n*n$ size two-dimensional array to store information about the graph. Size or space requirement for an adjacency matrix is $O(n^2)$. The lookup time for adjacency matrix representation is $O(1)$.



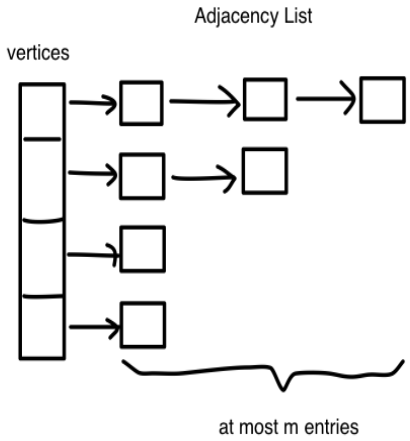
The possible number of edges for undirected and directed graph,

$$undirected\ graph = \binom{n}{2} = n(n-1)/2 = O(n^2) \tag{4}$$

$$directed\ graph = n*(n-1) = O(n^2) \tag{5}$$

3.2 Adjacency List

Adjacency list uses linked lists to store neighbors of the vertices. The lookup time for adjacency list representation is $O(m)$ since the list could be as long as m in length.



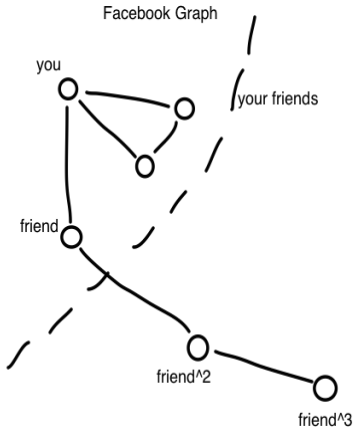
4 Examples and Applications

4.1 Map coloring

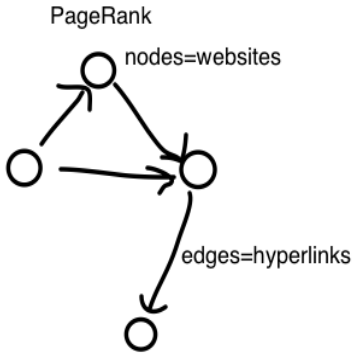
Map coloring is the problem of coloring the vertices of a graph such that no two neighboring vertices have the same color. The name stems from the old problem of coloring countries on a geographical map such that the countries' borders can be distinguished by using different colors for neighboring countries. In that problem, each vertex represents a country, and there is an edge between two countries whenever they share a border.

Another example of the application of the map coloring algorithm is the exam scheduling problem. In this problem, exams for classes need to be scheduled in such a way that if a student is taking multiple classes, his exams will not conflict with one another. A good way to reduce this problem to graph coloring is to use a vertex to represent a class, a edge between two vertices when theres at least one student taking both classes, and color to represent time slots.

4.2 Facebook Graph



4.3 PageRank



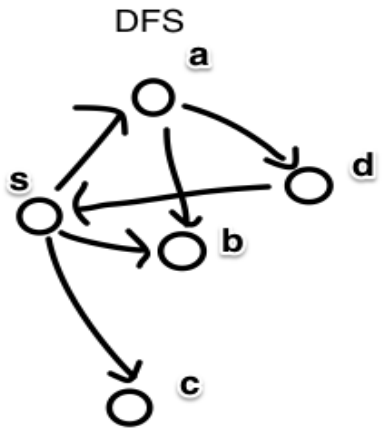
5 Depth first search (DFS) and DFS tree

5.1 Depth first search (DFS)

Pseudo-code:

```
1   DFS(s)
2           push(s) into S;
3           while s != empty set
4                   pop (v) from s;
5                   mark (v);
6                   for each neighbor u of v
7                           If u is unmarked
8                                   push (u) in S;
```

The running time of DFS is $O(n+m)$. Note that when the graph has no edges, it takes $O(n)$ because it runs DFS on every single vertex and returns immediately. In more common cases, it takes $O(m)$ time based on the *while loop* and *for loop* in the pseudo-code above.



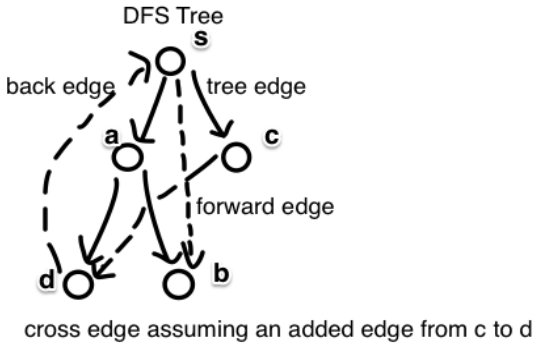
One possible order of visits for the example in the diagram:
s a d a b a s c s

5.2 Pre and post order

A *pre order* of a vertex is the first time a vertex is discovered in the ordered list of DFS visits. For example, $pre(a) = 2$ in the example above.

A *post order* of a vertex is the last time a vertex appears in the list of DFS visits. For example, $post(a) = 6$ in the example above.

5.3 DFS tree



In a DFS tree, if theres a path from u to v then u is an ancestor of v and v is a descendant of u.

There are four possible types of edges in a DFS tree:
tree edges: edges from vertices to their immediate descendants
back edges: edges between vertices and their ancestors
forward edges: edges between vertices and their descendants
cross edges: edges between vertices that are neither ancestors nor descendants

Properties of different types of edges:
for a tree edge (u,v) or forward edge (u,v),

$$pre(u) < pre(v) < post(v) < post(u); \tag{6}$$

for a back edge (u,v),

$$pre(v) < pre(u) < post(u) < post(v); \tag{7}$$

for a cross edge (u,v),

$$pre(u) < post(u); \tag{8}$$
$$pre(v) < post(v); \tag{9}$$

6 Summary

This lecture covers the basic definitions of a graph and related terminologies, some useful applications of graph theories, the details of a Depth First Search (DFS), and the properties of a DFS tree and its edges.