CS 341 — Lecture 12

Bartosz Antczak *Instructor: Bin Ma* *October 19, 2017*

12.1 More on Graph Algorithms

12.1.1 Bipartite Graphs

Recall that a **bipartite graph** is a graph that can be subdivided into two sets, where every vertex in both sets are not adjacent with vertices within the same set.
How can we prove that a graph is bipartite? We refer to a lemma.

Lemma 1

A bipartite graph can't have an odd cycle

We can use a breath-first search to help us: Today we'll learn about the depth-first search, which is another

```
Build a BFS
Put odd layers to s_1, even layers to s_2
if no edge exists in each layer for s_1 and s_2 then
    return true
return false
```

approach to scanning a graph.

12.1.2 Depth-first Search

This search determines the start and end times of when this node is reached in this traversal. The algorithm that implements the search requires a helper method:

Algorithm 1 explore(u)

```
for each neighbour v of u do
    if visited[v] is false then
        visited[v] ← true
        explore(v)
```

Given a graph $G = (V, E)$ and a source vertex $s \in V$, the main algorithm is:

Algorithm 2 DFS(s)

```
visited[v] ← false for all v ∈ V
visited[s] ← true
explore(s)
```

An implementation of DFS involving a stack is also shown:

```
s ← empty stack
visited[v] ← false for all nodes v ∈ V
visited[s] ← true
push(S, s)
while S is not empty do
  u ← S.pop()
  for each neighbour v of u do
    if !visited[v] then
      visited[v] ← true
      push(S, v)
```

Notice that this is the same algorithm structure as the BFS algorithm, except the data structure used is a stack rather than a queue.

Properties of DFS Trees

We define a **back-edge** as an edge (u, v) is u is the ancestor of v and $(u, v) \notin$ DFS tree. We outline two lemmas:

All non-tree edges on a DFS tree are back edges

For any u, v, the two intervals [start(v), finish(v)], [start(u), finish(u)] are either disjoint or one contains the other

A **cut vertex** is a vertex that, if removed, causes the graph to be disconnected.
A **cut edge** is an edge that, if removed, causes the graph to be disconnected.