

## 1 Overview

In this lecture, we will talk about graph algorithms. We will first learn the basic knowledge about a graph, and then discuss a graph traversal algorithm called Depth First Search (DFS).

## 2 Preliminaries

### 2.1 Definitions of a Graph

A graph is a data structure made of nodes and edges. We define a graph  $G = (V, E)$  where  $V$  represents the set of vertices and  $E$  represents the set of edges. We also refer to vertices as nodes. By default, we use  $n$  to denote the number of vertices in a graph, and use  $m$  to denote the number of edges in a graph.

The set of edges can either be undirected or directed. Given an edge  $e = (u, v) \in E$ . If  $G$  were undirected,  $e$  could be traversed from  $u$  to  $v$  or from  $v$  to  $u$ . However, if  $G$  were directed, then only one of these would be possible. Usually, in a directed graph, if an edge is expressed by  $(u, v)$ , it means the direction is from  $u$  and pointing to  $v$ . The edge with the reverse direction would be expressed as  $(v, u)$ .

### 2.2 Relationship between # Vertices and # Edges

As mentioned above, we use  $n$  to denote the number of vertices in the graph, and use  $m$  to denote the number of edges.

Often, in a graph, the same edge only appears at most once. In this case, there is at most one edge for each pair of vertices. Here the pair is ordered, i.e.,  $(u, v)$  and  $(v, u)$  are considered different pairs. Since there are  $n^2$  pairs of vertices in total, we know that

$$m \leq n^2.$$

Moreover, in most cases, the vertices in a graph are connected. If we want to connect  $n$  vertices together, the minimum number of edges needed is  $n - 1$ . Thus, in this case,

$$m \geq n - 1.$$

In particular, for connected graph we also have

$$\Theta(\log m) = \Theta(\log n).$$

### 2.3 Representing Graphs

There are multiple ways to represent a graph, two of which are adjacency array and adjacency list.

### 2.3.1 Adjacency Array

An adjacency array is a two-dimension array defined as

$$a[i][j] = \begin{cases} 1 & \text{if there is an edge } (i, j) \\ 0 & \text{if there is no edge } (i, j) \end{cases}$$

This array takes  $O(n^2)$  space. It can check whether  $(i, j)$  is an edge in  $O(1)$  time, but enumerating edges adjacent to a vertex takes  $O(n)$  time. This representation is better for dense graphs when  $m = \Theta(n^2)$  because in this case each vertex has approximately  $\Omega(n)$  neighbors and  $O(n)$  time is necessary.

### 2.3.2 Adjacency List

An adjacency list maintains a linked list for each vertex storing its neighbors. This representation takes  $O(n + m)$  space because storing all the nodes takes  $O(n)$  space and the number of elements in the linked lists are  $O(m)$ . To analyze the time complexity, we need to define *degree* first.

**Definition 1.** In an undirected graph, the degree of a vertex  $degree(i)$  is the number of edges that is adjacent to  $i$ .

**Definition 2.** In a directed graph, the in-degree of a vertex  $in-deg(i)$  is the number of incoming edges of  $i$ , and the out-degree of a vertex  $out-deg(i)$  is the number of outgoing edges of  $i$ .

After those definitions, we know that for an undirected graph, checking if  $(i, j)$  is an edge takes  $O(degree(i))$  time, and enumerating all edges adjacent to  $i$  also takes  $O(degree(i))$  time. Those are because the length of the linked list for node  $i$  is  $degree(i)$ . For directed graphs, the time complexity becomes  $O(out-deg(i))$ .

We also have the following claim:

**Claim 1.** For an undirected graph,  $\sum_{i \in V} degree(i) = 2m$ , where  $V$  is the set of vertices and  $m$  is the number of edges.

*Proof.* Let's consider the contribution of each edge in the graph to both sides of this equation. Since each edge is connecting two vertices, the contribution for an edge to the left hand side will be 2. The contribution for an edge to the right hand side is also 2. Thus, this equality holds.  $\square$

If the graph is sparse, i.e.,  $m = \Theta(n)$ , then the average degree

$$\sum_{i \in V} \frac{degree(i)}{n} = \frac{2m}{n} = \Theta(1).$$

Thus, adjacency list is better for sparse graphs.

## 3 Depth First Search

### 3.1 Algorithm

Depth First Search is a graph traversal algorithm that visits neighbor's neighbor first, which is the following algorithm:

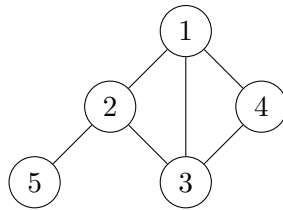


Figure 1: Example Graph

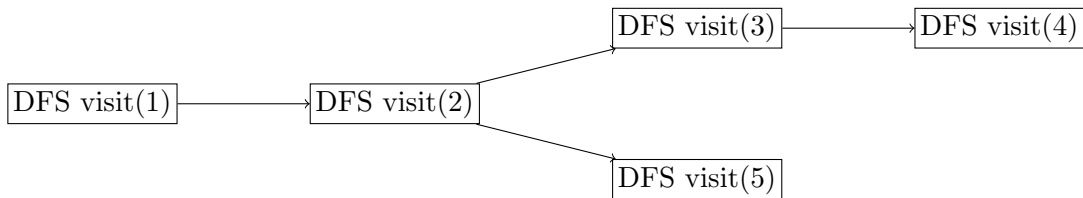


Figure 2: Example Run of DFS in Example Graph

```

DFS visit(u)
  Mark u as visited;
  FOR each edge (u,v) do
    IF v is not visited then
      DFS visit(v)
  
```

```

DFS
  FOR u = 1 to n do
    DFS visit(u)
  
```

Let's look at an example run of this algorithm: Consider the graph in Figure 1, and we start with node 1. Assume we are enumerating the children of any node in an increasing order, then the order of visiting each node will be the same as Figure 2. First we call the DFS visit at 1, then it first visit 2, and then 3 and 4. After that, it returns to 3 and further returns to 2 because there are no new children to be visited. Then the algorithm visited 5 and returns back to 1 and terminate. Figure 2 gives us the ordering of calling all those visit functions.

### 3.2 DFS Tree

A DFS Tree is constructed like this: If  $DFS\ visit(u)$  calls  $DFS\ visit(v)$ , add  $(u, v)$  to the tree. That is, only preserve an edge if it is used to discover a new vertex. It is a tree because in DFS

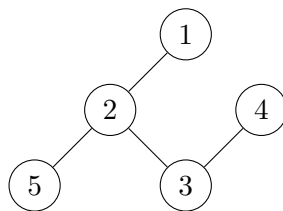


Figure 3: DFS Tree of Example Run

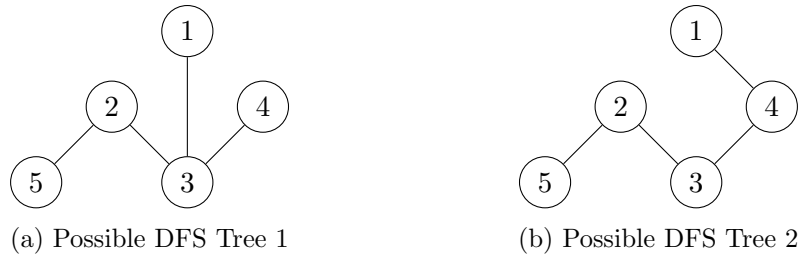


Figure 4: Possible DFS Tree

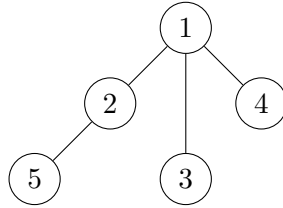


Figure 5: Impossible DFS Tree

we do not visit the previously visited nodes, which ensures there are no cycles. In the example run above, the DFS tree is the one in Figure 3, which can be derived from Figure 2.

DFS tree is not unique, it may change if you choose different starting points and/or the ordering of edges. For example, starting from node 1, if you decide to follow (1, 3) first, the DFS tree will be like Figure 4a. If you decide to follow (1, 4) first, the DFS tree will be like Figure 4b.

However, there are also some trees that cannot be a DFS tree. For instance, if you start the DFS from node 1, then the tree in Figure 5 cannot be a DFS tree because once you choose a child of 1 to visit, that child will visit the other two children before returning to 1.

### 3.3 Pre-Order and Post-Order

Pre-Order and Post-Order are the orderings of nodes in a graph. Pre-Order is defined as the order in which the vertices are visited (entered the stack), and Post-Order is the order in which the vertices are last touched (leaving the stack).

Take our example run in Figure 2 as an example, we call the DFS visit function in the order of 1, 2, 3, 4, 5. Thus, the pre-order of that run will be 1, 2, 3, 4, 5. Also, in that example run, the DFS visits are finished in the order of 4, 3, 5, 2, 1, which is the post-order.

Note that it is possible that the orderings are not unique. In Figure 4a, if we first visit vertex 2 after vertex 3, then the pre-order is 1, 3, 2, 5, 4 and post-order is 5, 2, 4, 3, 1. If we first visit vertex 4 after vertex 3, then the pre-order is 1, 3, 4, 2, 5 and post-order is 4, 5, 2, 3, 1.