Introduction to Graphs and BFS.

May 16, 2024

Contents

1 Graphs

1.1 Messed up trees

Trees have a sense of structure with a root and different nodes like leaves. With a graph is more of a messed up tree where there is no root node and no notion of depth apart from distance. There is also no idea of a parent or child in a graph. All the nodes in a graph are sort-of equivalent in terms of rank. One difference between a tree and a graph is that graphs can have cycles of nodes.

With graphs, nodes can be connected anywhere and can even be alone.

1.2 Formal definition

Graphs are trees that have no special root node, can contain cycles and any node can be connected to any other node. These nodes are called vertices and the connections between them are called edges.

1.3 Types

There are two types of graphs to keep track of: undirected and directed graphs. In an undirected graph, the nodes have no sense of direction between

nodes whereas in a directed graph, nodes point to each other in specific directions. One consequence of a directed graph is there can be nodes that point but are not pointed to.[1]

1.4 Path

A path is a set of nodes that can be followed to reach a specific node from a starting node.

1.5 Implementation

The following are the required functions for a graph:

- build([list of vertices and edge])
- insert_vertex(v): $O(n^2)$ for an adjacency matrix, $O(n)$ for an adjacency list.
- insert_edge(v_1, v_2): O(1) for an adjacency matrix, $O(1_a)$ for an adjacency list.
- remove_vertex(v): $O(n^2)$ for an adjacency matrix[2],
- remove_edge(v_1, v_2): $O(1)$ for an adjacency matrix, $O(1)$ for an adjacency list.
- path_exists(v_1, v_2): Is there a path between v_1 and v_2?
- shortest_path(v_1, v_2): How long is the shortest path between v_1 and v_2?
- single_source_shortest_paths(v): Return the lengths of the shortest paths between v and all other nodes.

Graphs can be implemented by a 2 dimensional adjacency matrix or an adjacency list using a direct access array or hash table.

2 Breadth-First Search.

One idea used to find the shortest path between two vertices is called Breadth-First Search (BFS). By going through an adjacency list, we find the adjacent values of the value we are looking for and then we spread out to the subarrays / sublists of these adjacent vertices. We repeat until we can find the value.

[1] Nodes can point to themselves too but this is not a focus of the class

[2] This depends, this can be $O(1)$ if you simply just ignore the removed vertex in the matrix