23.3 Summary

Graph Traversals Overview

- The same traversals that we used on trees can be generalized to graphs. Given a source vertex, we can visit vertices in:
  - DFS preorder: the order in which DFS is called on each vertex.
  - DFS postorder: the order in which DFS returns from each vertex.
  - BFS: the order of distance from the source node (this is level-order in trees).

BFS

- Unlike DFS, BFS has a natural solution that is iterative, not recursive. BFS visits a source vertex `s`, then every vertex at distance `1` from `s`, then every vertex at distance `2` from `s`, and so on.
- BFS uses a *fringe* of vertices that are next to be explored. In BFS, this fringe is a queue. We enqueue new vertices at the end, and dequeue vertices to visit from the front.
- BFS can be used to solve the shortest paths problem, given that we want to minimize the number of edges from source to each other vertex. If we want to recover the shortest path from BFS, we need to track the `edgeTo` each vertex during our traversal.

Graph Implementation

- The choice of API for a graph determines how clients must write their code. Certain APIs make some tasks easier and other tasks harder. The choice of API can also affect runtime and memory.
- Choice of graph implementations include adjacency matrices, lists of edges, and adjacency lists. An adjacency matrix is a 2D boolean array indicating whether any pair of vertices are adjacent. A list of edges is a collection of all edges in the graph.
-

- The most common approach to graph representation is an adjacency list. In this representation, we maintain a array of lists indexed by vertex number; each index stores all vertices connected to the given vertex.

Previous
23.2 Representing Graphs

Next
23.4 Exercises

Last updated 1 year ago