Hanoi University

Faculty of Information Technology

Department of Computer Science

HOMEWORK

DISCRETE MATHEMATICS

REPORT WEEK 11 – GRAPH PART 3

Name: Luong Thi Uy Thieu

ID: 2001040194

Class: DMA – B05

Semester: Fall 2021

In data structures and algorithms, graphs are a basic form of data structure and have very wide applications. Many complex systems can be simulated as a graph, such as transport systems, network systems, power transmission systems. In which, two algorithms for traversing the graph according to breadth (BFS) and depth (DFS) are the two most basic algorithms of the graph. These algorithms help us to "visit" all the edges and vertices of the graph in minimal time.

1.Depth-First-Search (DFS) Algorithm:

- DFS is a general technique for traversing a graph.
- A DFS traversal of a graph G
- Visit all the vertices and edges of G.
- Determines whether G is connected.
- Computes the connected components of G.
- Computes a spanning forest of G.

• DFS on a graph with n vertices and m edges take O(n+m) time.

• DFS can be further extended to solve other graph problems

- Find and report a path between two given vertices.
- Find a cycle.

• DFS Algorithm: Properties

1. DFS(v) visits all the vertices and edges in the connected component of v.
2. The discovery edges labeled by DFS(v) form a spanning tree of the connected component of v.

• DFS has many applications. Eg…

- Determine the connected components of the graph.
- Topological arrangement for graphs.
- Determine the strongly connected components of a directed graph.
- Check if a graph is planar or not.

DFS is commonly used in game simulations (and real-world game-like situations). In a casual game, you can choose one of many possible actions. Each choice leads to another, each leads to another, and so on into an ever-expanding tree of possibilities.

2.Breadth-First-Search (BFS) Algorithm

- BFS is a general technique for traversing a graph.
- A BFS traversal of a graph G
- Visit all the vertices and edges of G.
- Determines whether G is connected.
- Computes the connected components of G.
- Computes a spanning forest of G.

• BFS on a graph with n vertices and m edges take O(n+m) time.

• BFS can be further extended to solve other graph problems

- Find and report a paht with the minimum number of edges between two given vertices.
- Find a simple cycle, if there is one.

BFS has an interesting property: It first finds all vertices that are one distance from the starting point of an edge, then all vertices that are located two edges away, and so on. This is useful if you are trying to find the shortest path from a starting vertex to a given vertex. You start BFS and when you find the specified vertex you will know the path you have followed so far is the shortest path to the node. If there was a shorter path, BFS would have found it already.

BFS can be used to find neighboring nodes in peer-to-peer networks like BitTorrent, GPS systems to find nearby places, social networking sites to find people at a specific distance, and the like.

3.Application of DFS and BFS algorithms.

- Traverse all the vertices of the graph.
- Browse all connected components of the graph.
- Find the path from vertex to vertex on the graph.

- Check the strong connectivity of the graph.
- Browse the cylindrical vertices of the graph.
- Traverse the spherical edges of the graph.
- Check if a graph is dimensioned or not.

* Applications of Depth First Search:

- Detecting cycle in a graph

A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.

- Path Finding

We can specialize the DFS algorithm to find a path between two given vertices u and z.
i) Call DFS(G, u) with u as the start vertex.
ii) Use a stack S to keep track of the path between the start vertex and the current vertex.
iii) As soon as destination vertex z is encountered, return the path as the
contents of the stack

- Topological Sorting

Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when recomputing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in makefiles, data serialization, and resolving symbol dependencies in linkers.

- To test if a graph is bipartite

We can augment either BFS or DFS when we first discover a new vertex, color it opposited its parents, and for each other edge, check it doesn't link two vertices of the same color. The first vertex in any connected component can be red or black! See this for details.

- Finding Strongly Connected Components of a graph A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.

- Solving puzzles with only one solution, such as mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.)

* Applications of Breadth First Traversal:

- Shortest Path and Minimum Spanning Tree for unweighted graph In an unweighted graph, the shortest path is the path with least number of edges. With Breadth First, we always reach a vertex from given source using the minimum number of edges. Also, in case of unweighted graphs, any spanning tree is Minimum Spanning Tree and we can use either Depth or Breadth first traversal for finding a spanning tree.
- Peer to Peer Networks. In Peer to Peer Networks like BitTorrent, Breadth First Search is used to find all neighbor nodes.
- Crawlers in Search Engines: Crawlers build index using Breadth First. The idea is to start from source page and follow all links from source and keep doing same. Depth First Traversal can also be used for crawlers, but the advantage with Breadth First Traversal is, depth or levels of the built tree can be limited.
- Social Networking Websites: In social networks, we can find people within a given distance 'k' from a person using Breadth First Search till 'k' levels.
- GPS Navigation systems: Breadth First Search is used to find all neighboring locations.
- Broadcasting in Network: In networks, a broadcasted packet follows Breadth First Search to reach all nodes.
- In Garbage Collection: Breadth First Search is used in copying garbage collection using Cheney's algorithm. Refer this and for details. Breadth First Search is preferred over Depth First Search because of better locality of reference:
- Cycle detection in undirected graph: In undirected graphs, either Breadth First Search or Depth First Search can be used to detect cycle. We can use BFS to detect cycle in a directed graph also.

- Ford–Fulkerson algorithm In Ford-Fulkerson algorithm, we can either use Breadth First or Depth First Traversal to find the maximum flow. Breadth First Traversal is preferred as it reduces worst case time complexity to O(VE2).
- To test if a graph is Bipartite We can either use Breadth First or Depth First Traversal.
- Path Finding We can either use Breadth First or Depth First Traversal to find if there is a path between two vertices.
- Finding all nodes within one connected component: We can either use Breadth First or Depth First Traversal to find all nodes reachable from a given node.

4.Actual when using DFS vs BFS algorithm.

That depends a lot on the structure of the search tree and the number and location of solutions (aka search entries).

- If you know a solution not far from the stump, a first search (BFS) might be better.
- If the tree is very deep and the solutions are rare, the first search (DFS) can take a very long time, but the BFS can be faster.
- If the tree is very large, BFS may need too much memory, so it may be completely impractical.
- If the solutions are frequent but deep in the tree, BFS may not be practical.
- If the search tree is very deep, you will need to constrain the search depth to depth-first search (DFS), anyway (with depth of repetition, for example).

5.Difference between DFS and BFS

Comparing BFS and DFS, the big advantage of DFS is that it has much lower memory requirements than BFS, since it is not necessary to store all the subpointers at each level. Depending on the data and what you are looking for, DFS or BFS can be advantageous.

For example, given a family tree if a person is looking for someone in the tree that is still alive, it is safe to assume that person will be under the tree. This means that a BFS will take a very long time to reach that final level. A DFS, however, will find the target faster. But, if one is looking for a family member who died a long time ago, that person will be

closer to the top of the tree. Then BFS will usually be faster than DFS. So the advantage of either depends on the data and what you are looking for.

Another example is Facebook; Suggestions about friends of friends. We needed friends right away to suggest where we could use BFS. Can find shortest path or detect cycle (using recursion) we can use DFS.

Breadth First Search is often the best approach when the depth of the tree is variable and you only need to search part of the tree to find a solution. For example, finding the shortest path from the start value to the end value is a good place to use BFS.

Depth First Search is often used when you need to search an entire tree. It's easier to implement (using recursion) than BFS and requires less state: While BFS requires you to store the entire 'border', DFS only requires you to store a list of the section's parent nodes. current death.

Simply put:

The Breadth First Search (BFS) algorithm, from its name "Breadth", discovers all the neighbors of a node through the outer edges of the node, then it discovers the unnoticed neighbors of the nodes. previously mentioned neighbors over their edges and until all the nodes reachable from the origin are visited (we can go ahead and get another origin if there are still unwanted nodes want and so forth). That's why it can be used to find the shortest path (if any) from one node (origin) to another if the weights of the edges are uniform.

The Depth First Search (DFS) algorithm, from the name "Depth", discovers unnoticed neighbors of the most recently discovered node x through its outer edges. If there are no unnoticed neighbors from node x, the algorithm goes back to discovering the node's unnoticed neighbors (through its outer edges) from which node x was discovered, and so on, until all the nodes reachable from the origin are accessed (we can go ahead and get another origin if there are still unwanted nodes and so forth).

Both BFS and DFS can be incomplete. For example, if the branching factor of a node is infinite or very large for the resources (memory) to support (e.g. when storing the next discovered nodes), the BFS is not complete even though The search key can be at the distance of some edge from the origin. This infinite branching factor can be attributed to

the infinite choices (neighborhood nodes) from a given node to explore. If the depth is infinite or very large for resources (memory) to support (e.g. when storing next discovered nodes), then DFS is not complete even though the search key may be a neighbor third of origin. This infinite depth is likely due to the situation where there is, for each node that the algorithm discovers, at least one new choice (neighborhood node) that was previously unnoticed.

Therefore, we can conclude when to use BFS and DFS. Let's say we're dealing with a manageable limit branching factor and a manageable limit depth. If the search node is shallow, i.e. accessible after some edges from the origin, then it is better to use BFS. On the other hand, if the node is deeply searched i.e. reachable after a lot of edges from the origin, then it is better to use DFS.

For example, in a social network if we want to search for people with similar interests to a particular person, we can apply BFS from this person as an origin, because most of these people will be friends his immediate friends or friends, for example one or two distal edges. On the other hand, if we want to search for people with completely different interests of a particular person, we can apply DFS from this person as an origin, because most of these people will be far away. him, ie friend of your friend....ie too much edge.

The applications of BFS and DFS may also differ due to the search mechanism in each. For example, we can use BFS (assuming the branching factor is manageable) or DFS (assuming the depth is manageable) when we just want to check accessibility from this node to another node without any information that the node may be present. In addition, both can solve similar tasks such as topological sorting of graphs (if any). BFS can be used to find the shortest path, with unit weight edges, from a node (origin) to another path. Meanwhile, DFS can be used to exhaust all choices because of its nature of going deep, like discovering the longest path between two nodes in an acyclic graph. Also DFS, can be used to detect cycles in graphs.

Finally, if we have infinite depth and infinite branching coefficient, we can use Iterative Depth Search (IDS).