BFS

June 8, 2020

1 Paths and Distances

Path Length: > Path Lenght $L(P)$ is the number of edges in the path

Paths

$L (B - C - F - E - D) = 4$

$L (B - A - F - D) = 3$

Distance: > The distance between two nodes is the lenght of the shortest path between them

Distance between B & D is 3

$d (B, D) = 3$

If the graph is directed, the distances/paths change

Distance from $A - C$ is a loop and thus infinity

1.1 Distance Layer

What if we wanted to show case the distance from one node to all other nodes?

We can just make it into a tree and get the height

What happens if we add a new node? Well you can add a node to C, D, E but you could not add it to B. Because if you added it to B then it would not have a distance of 3 but 2.

In general, only the edges which go between nodes inside somewhere or the edges which go from a layer to the adjacent layer are possible

For Directed Graphs, things change slightly. For example an edge from G to F can exist. It could not exist in an undirected graph. B to G cannot be possible because it makes distance closer

- In directed, layer going deeper then 1 unit are not possible, but going back up is possible

2 Breadth-First Search

Breadth-First Search basically just creates a distance layer tree. T_T

- The reason we use a Queue is due to computers non-parallel nature. A computer runs a program one by one, thus you need to keep track of the order

We take the node at the top of the queue and begin to process it.

2.1 Big-O

The running time of breadth-first search is $ O (|E| + |V|)$

Algorithm runs porportional to the size of the Edges and vertices

Proof: - Each vertex is enqueued at most once - Each edge is examined either once (for directed) or twice (for undirected)

2.2 Properties of BFS

Reachability: > Node $ u $ is reachable from node $ A $ if there is a path from $ A $ to $ u $

- Reachable nodes are discovered at some point, so they get a finite distance estimate from the source. Unreachable nodes are not discovered at any point, and the distance to them stays infinite

Proof: A reachable node is discovered - $ u $ is reahable undiscovered closest to $ A $ - $ A - v_1 - … - v_k - u - $ shortest path - $ u $ is discovered while processing $ v_k $

By the time node $ u $ at distance $ d $ from $ A $ is dequeued, all the nodes at distance at most $ d $ have already been discovered (enqueued). i.e, nodes cannot be processed in the wrong order

proof:

Queue Property: > At any moment, if the first node in the queue is at distance $ d $ from $ A $, then all the nodes in the queue are either at distance $ d $ from $ A $ or at distance $ d + 1 $ from $ A $. All the nodes in the queue at distance $ d $ go before (if any) all the nodes at distance $ d + 1 $

Proof: - All nodes at distance $ d $ were enqueued before first such node is dequeued, so they go before nodes at distance $ d + 1 $ - Nodes at $ d - 1 $ were enqueued before nodes at $ d $, so they are not in the queue anymore - Nodes at distance $ > d + 1 $ will be discovered when all $ d $ are gone

3 Correctness of Distances

Lemma: > When node $ u $ is discovered (enqueued), $ dist[u] $ is assigned exactly $ d(A, u) $

Proof: - Use mathematical induction - Base: when $ A $ is discovered, $ dist[A] $ is assigned $ 0 = d(A, A) $ - Inductive step: suppose proved for all nodes at distance $ <= k $ from $ A - $ prove for nodes at distance $ k + 1 $ - Take a node $ v $ at distance $ k + 1 $ from $ A $ - $ v $ was discovered while processing $ u $ - $ d(A, v) <= d(A, u) + 1 <– d(A, u) >= k $ - $ v $ is discovered after $ u $ is dequeued, so $ d(A, u) < d(A, v) = k + 1 $ - so $ d(A, u) = k $ and $ dist[v] –> dist[u] + 1 = k + 1 $

4 Shortest-Path Tree

The shortest-path is connected and a tree, and to prove its a tree, we just need to show it has no cycles

Lemma: > Shortest-path tree is indeed a tree, i.e, it doesn't contain cycles (it is a connected component by construction)

Proof: - only one outgoing edge from each node - distance decreases after going by edge

This is a contridiction, with an assumption there is a cycle

4.1 Algorithm

Constructing shortest-path tree

Reconstructing Shortest Path