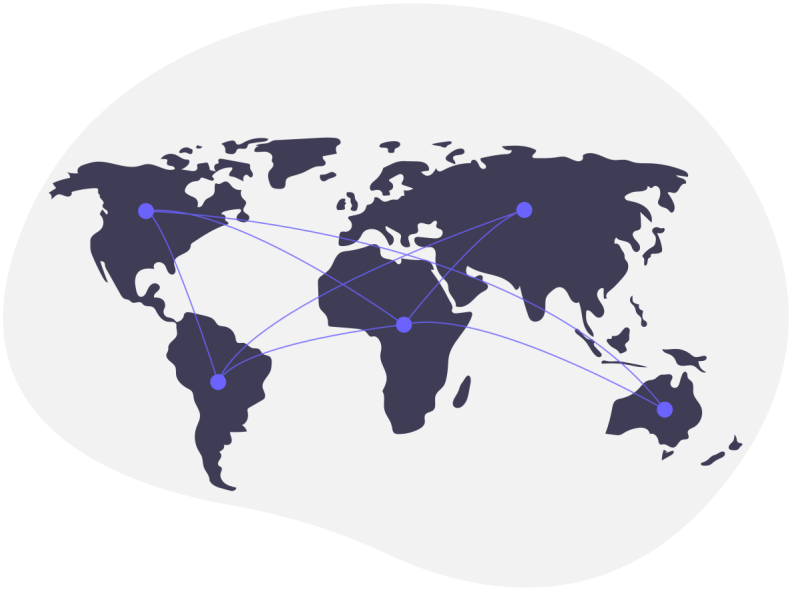
CITS2200 Project Report

University of Western Australia



[1]

Kristof Kovacs (22869854)
Alexandria Bennett (22969368)

May 21, 2021

All Devices Connected

Explanation

The `allDevicesConnected()` method is used to check if all of the devices in a directed network are connected to all of the other devices in the network. This is done by using a breadth-first search (BFS) twice; once on the given adjacency list and once on the transposed adjacency list. The first call to BFS checks if an arbitrary node (e.g. 0) is connected to all of the other nodes because if it is not, it it immediately returns false. Transposing the adjacency list allows us to check if every node is connected to the original node (0). By running a BFS on the transposed adjacency list, it creates a minimum spanning tree to the original node which we already know is already connected to all other nodes. If any node isn't connected back to the 0 node, it will return false since it is not connected to every other node.

Correctness

A BFS is the correct choice because makes a minimum spanning tree that is connected to every node and then we make a second min spanning tree to see if every node is connected to the original node, as required.

Complexity

The time complexity of a BFS is $O(D + L)$ because it processes each device in the network once and goes down each link at most twice in a worst case scenario. Since $N = D + L$ the overall complexity of the BFS used in `allDevicesConnected()` is $O(N)$. The BFS is run twice giving a time complexity of $O(2N) = O(N)$.

Number of Paths

Explanation

The `numPaths()` method calculates the number of paths from a source to a destination device. It runs a standard BFS traversal where by if the adjacent device to `current` is the destination then we know there exists a path. Otherwise, if the same adjacent device has not yet been seen, add it to the queue and mark it as visited. Then repeat where the head of the queue is assigned to `current` - until the queue is empty.

Correctness

Since this BFS explores all neighboring devices until moving onward, we can be sure that all paths are explored. If the destination device happens to be along the way of the path being traversed then we know there exists a link (or a series of links) from the source to the destination.

Complexity

Again, since this is a standard implementation of BFS the time complexity is $O(N)$ where $N = D + L$.

Closest In Subnet

Explanation

closestInSubnet() is a method to determine the shortest path from a source to a device that is in a sub-network - determined by a given query. Initially, the shortest paths to all devices from the source node is computed using BFS. Then for every query, we check every device to see if its address is in the sub-network. Once the devices have been organised all we have to do is just check which device in the given query has the shortest path.

Correctness

Using BFS traversal we obtain the shortest path to every device. By doing a trivial comparison we can check which device is in given query, we can then just find the node with the shortest distance in linear time.

Complexity

The time complexity of the BFS as seen before is $O(N)$. Added to the complexity checking each query once is $O(Q)$. For each query, we check each device in the subnet leading to $O(DQ)$ time. In total, the time complexity of `closestInSubnet()` in the worst case is $O(N) + O(DQ)$ which is equivalent to $O(N + DQ)$.

Max Download Speed

Explanation / Correctness

The `maxDownloadSpeed()` uses the Edmonds-Karp maximum flow algorithm. It is a modification of the Ford-Fulerson algorithm where the search order for augmenting paths is done via a BFS instead of a depth-first search (DFS). They both employ the max-flow min cut theorem which states "that the maximum flow through any network from a given source to a given sink is exactly the sum of the edge weights that, if removed, would totally disconnect the source from the sink" [2], where the "sink" is equivalent to the destination. From the source node of a weighted and directed graph, the weight is the amount of 'flow' or capacity which it can hold at a maximum. From the source from there may exist other paths that can restrict the flow as shown in Figure 1.

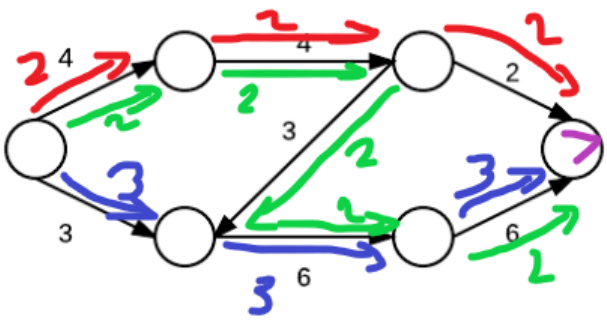
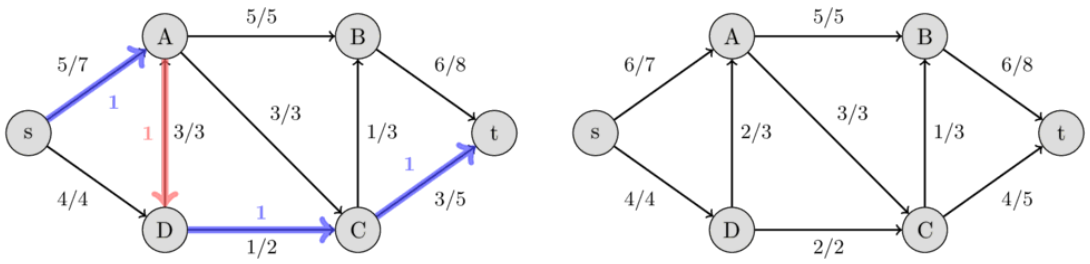


Figure 1: Flow through links

Using shortest paths, we follow them to the sink node, however, sometimes the augmenting path takes uses a reversed edge which may not possible in the given graph so we use the idea of a reverse graph. Any time the shortest path goes down a reversed path, we subtract one from a reverse flow and add it to a forward flow as shown in 2.



(a) Step 1 (b) Step 2

Figure 2: Flow augmentation [3]

In our implementation we use array 2-d array called `flow` which holds the current best speed for each device. We must "convert" the given 2-d speeds array to one that is alike a square matrix. This is so that we can easily compute comparisons between the current best speed and the max speed of each link for each device. Our implementation emerged from the ideas presented in [4] and [5].

Complexity

The Edmonds-Karp algorithm has $O(DL^2)$ because every time we find a new augmenting path, one link becomes saturated and in the worst case scenario, we backtrack (via the 'residual graph') and if we do this for every device, that is $O(L^2)$, $O(D)$ times, hence our asymptotic times complexity becomes $O(DL^2)$.

Due to our 2-d `flow` array, this solution will require at most $O(D^2)$ space.

References

[1] K. Limpitsouni. (2021). "Undraw," [Online]. Available: https://undraw.co/. (accessed May. 18, 2021).

[2] A. Chumbley, Z. Vinger, and E. Ross. (2021). "Max-flow min-cut algorithm," [Online]. Available: https://brilliant.org/wiki/max-flow-min-cut-algorithm/. (accessed May. 12, 2021).

[3] J. Kogler. (2021). "Maximum flow - ford-fulkerson and edmonds-karp," [Online]. Available: https://cp-algorithms.com/graph/edmonds_karp.html. (accessed May. 12, 2021).

[4] W. Fiset. (2018). "Edmonds karp algorithm — network flow — graph theory," [Online]. Available: https://www.youtube.com/watch?v=RppuJYwlcI8. (accessed May. 7, 2021).

[5] W. Fiset. (2018). "Edmonds karp algorithm — source code," [Online]. Available: https://www.youtube.com/watch?v=OViaWp9Q-Oc. (accessed May. 7, 2021).