Final Project for CS 372

Kevin Chamberlain

Algorithm, Application, Language Choice

- Algorithm choice: Max Flow Edmonds-Karp
- Application choice: Network mapping
- Language choice: Python

Where It Is Used

The Edmonds-Karp, a derivative of Ford-Fulkerson, is used to solve many different types of problems. Chief among them being maximum flow problems, such as: with infinite input through the source, how much flow can be pushed through the network given that each edge has a certain capacity. Examples of these types of problems would be:

- Water going through a set of pipes
- Electricity going through breakers and wires
- How to handle the flow of traffic.

Other applications

Edmonds-Karp can also be used to solve a host of positioning problems such as where something is located in relation to another point. Examples of these problems would be:

- Billboards in reference to a freeways
- Lights in reference to what they should be shining on
- Line of sight radios, being able to “see” each other

Alternative algorithms

- Ford-Fulkerson
- Dinic’s Algorithm

Reason for choice

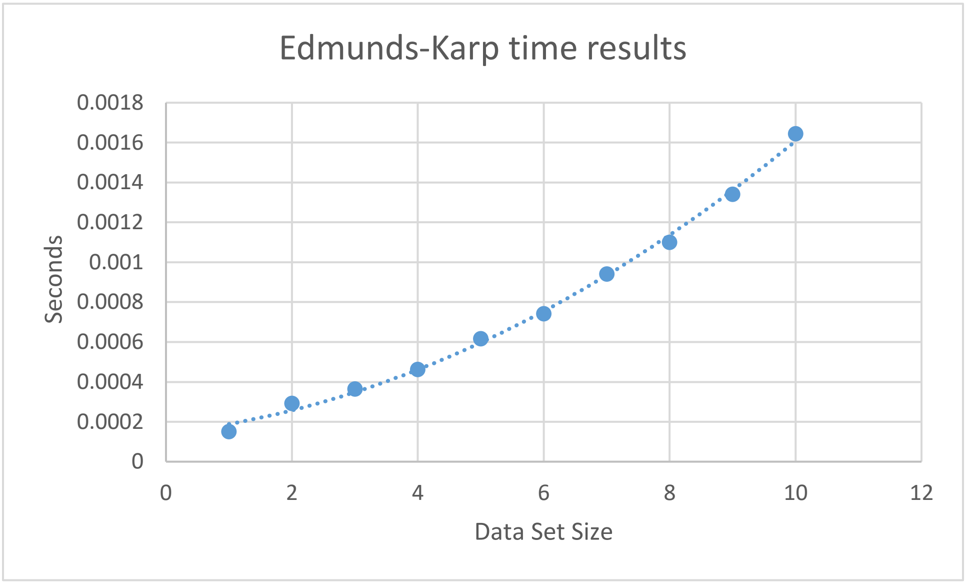
The reason I chose this algorithm was because I originally saw that it could be used for network mapping and lately, though I don’t have any knowledge about technical networking it has interested me, and I am taking networking next year. I researched the algorithm and thought the concept was interesting and useful though as I started researching the networking side of the problem, I realized I was a bit out of my depth for a project. After this realization I thought back to the only networking experience I had from the Marine Corps, and that relied on line of sight radios. Running with my radio experience I delved into the algorithm and found that the Edmunds-Karp algorithm was useful for positional mapping and that mapping out the radio network would be a useful solution that I wish I had had access to in the service. Being able to apply this algorithm to a situation that had troubled me for years was essentially why I picked this combination as my algorithm choice.

How Your Project Works

My program is definitely focused more on the problem and solution than it is on being intuitive or user friendly. With this being said there is no input from the user and instead relies on hardcoded values above the function calls for its data. However with the modularity of the program an input function could easily be written to read in files with the appropriate data. The data I refer to is three lists of coordinates, two lists to hold the positions of radios, and a list of obstructions. For the purpose of keeping them straight the radio lists are labeled Receivers and Transmitters though in reality each radio has both, but if the receivers can see and talk to the transmitters then the opposite is also true. For this situation, the start of the program checks to see if the number of transmitters and receivers are the same, and if so a function(maxFlowGraph) is called to create the graph. msxFlowGraph build out the adjacency matrix though it implements a function(lineOfSight) to determine if the radios can "see" each other, or if there are obstructions in the way that would prevent the radios from talking to each other.

After the graph is populated, the max flow is found using the Edmonds_Karp algorithm. The algorithm utilizes a breadth first search on the graph to ensure that there is a path. Sidenote: If a depth first search was used it would make this a Ford-Fulkerson algorithm. While there is a path, as found by the BFS, from source(the beginning of the graph) to sink(the end of the graph) the algorithm will continue to run. The function will find the bottleneck of the path, in this case our lowest value, and the will be set as our path flow variable. This is done by running again from source to sink, our source and sink will be reset so that we can use these variables again. Now that our variables are reset we run again from source to sink this time subtracting the flow from the path for our residual arches and adding the flow to our backward edges. With every iteration through our path flow will be added to our max flow. The max flow is what this function will return. It is the max flow that will be checked against the number of radios, if the values are the same then we know that our radios can see each other.

Run time



Code Correctness Tests

Test 1; test1()

The test1() tests my isPaired() function to ensure that the amount of Receivers and Transmitters are the same, so that the rest of my program can perform appropriately.

Input

The first input: (1, 4), (2, 4), (7, 4)

The second input: (4, 4), (6, 2)

Expected Output

False

Actual Output

False

Test 2; test2()

The test2() tests my lineOfSight() function to determine if the transmitters and receivers would be able to see each other from there respective positions. Since this is ultimately what my programs job is to do, it is imperative that this function is correct.

Input

The first input: (2, 4), (2, 2), (5, 4)

The second input: (6, 2),(7,4),(6,3)

Expected Output

4 seen, 5 obstructed

Actual Output

4 seen, 5 obstructed

Test 3; test3()

The test3() tests my overall program to ensure that as a whole it is correct in it's solution and output.

Input

The first input: (2, 4), (2, 2), (5, 4)
The second input: (4, 4),(6,3),(6,2)

Expected Output	Actual Output
All troops are communicating	All troops are communicating

There are 11 tests in total including the time test available for review in the TestSuite file of the project

References

Williem Fiset
https://www.youtube.com/watch?v=LdOnanfc5TM&t=236s

Introduction to Algorithms Third Edition