2nd Project ASA 2019/2020 Report

Group: tp039

Students: Pavle Arandjelovic (93745) & António Marques Murteira (90706)

Description of the Problem:

The project in question required us to create a program, where given:

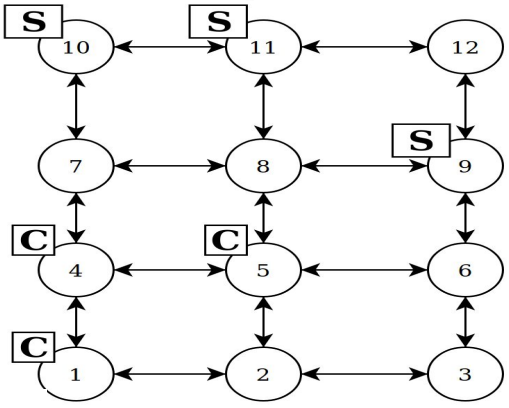
- The number of avenues and streets (M and N respectively)
- The number of shops and citizens (S and C respectively)
- The locations of every shop (S shops)
- The locations of every citizen's house (C citizen houses)

would return us the maximum amount of people allowed to leave their houses for shopping, based on their position in relation to other citizen's locations and the shop's locations.

Analysis of the Problem at Hand:

To fully understand the problem at hand, we first constructed a graph with the second example given in the project brief. Each intersection represented **one** node in our graph and each node had a vector of other nodes it was able to travel to (each edge had a capacity of 1). We noticed that it would be difficult to limit flow to a certain node without having inefficient and convoluted code in this manner (see Fig. 1 for further clarification).

Fig. 1



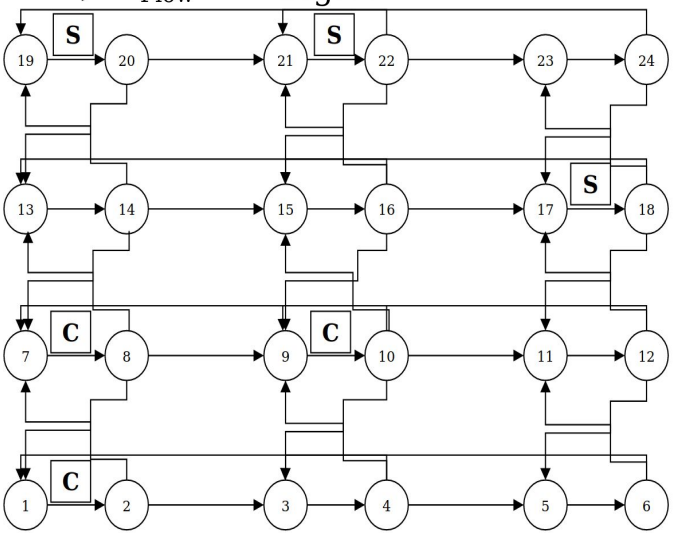
Solution of the Problem:

Due to the difficulty in limiting flow from one node to another (as mentioned in the previous section), we constructed another graph where each intersection represents **two** nodes (input -> output) which solves the problem (see Fig. 2 for further clarification).

Although not drawn in Fig. 2, we had a source node (from which the flow would run out of and into all the citizen nodes) and a target node (which we had all the flow running out from each shop node and into it), this way we could calculate the max amount of citizens allowed to leave their house, by discovering the max flow in our graph.

Key:
"C" = Citizen
"S" = Shop
"->" = Flow

Fig. 2



We decided that the best-fit algorithm to help us solve this problem would be the Edmonds Karp algorithm which would help us identify the max flow in our graph from our source to our designated target using BFS as an auxiliary search mechanism. Our program was coded in C++ and contains portions of code from GeeksForGeeks [(1)].

(1) Ford-Fulkerson Algorithm for Maximum Flow Problem

2nd Project ASA 2019/2020 Report

Group: tp039

Students: Pavle Arandjelovic (93745) & António Marques Murteira (90706)

We initially started by constructing an adjacency matrix to represent all of the relationships between each node, however this idea was quickly scrapped due to it consuming a lot of memory (V^2 per node introduced into the matrix) as a lot of the elements in each row would comprise of 0's that will never be needed. We replaced our adjacency matrix with an adjacency list, reducing our memory consumption and accepted the slight speed loss in comparison to an adjacency matrix (lookup time of O(1)).

Theoretical Analysis of our Solution:

Looking at the flow of our program, we broke it down into steps with time-complexity analyses:

- The process of reading and processing data given to us depended linearly on how many citizens and shops we had to process (S = Shops, C = Citizens).
  Time Complexity -> Θ(|S| + |C|)
- Creation of graph (residual + normal) with each node having ~8 edges (normal and reverse edges = 4 * 2) is done in linear time, as we are only associating a fixed number of edges per node added to our graph.
  Time Complexity -> O(V)
- Create graph object and initializations: Constant time as we are only allocating memory for a fixed amount of data structures.
  Time Complexity -> O(1)
- BFS algorithm using an adjacency list depends solely on our edge and node count.
  Time Complexity -> O(|V| + |E|)
- Altered Edmond Karp Algorithm: Equal to that of a normal Edmond Karp algorithm.
  Time Complexity -> $O(|V| \cdot |E|^2)$
- Representation of data to Stdout: This is done in constant time as we are merely printing out the result from our Edmond Karp algorithm.
  Time Complexity -> O(|1|)

TOTAL TIME COMPLEXITY OF SOLUTION -> $O(|V| \cdot |E|^2)$

2nd Project ASA 2019/2020 Report

Group: tp039

Students: Pavle Arandjelovic (93745) & António Marques Murteira (90706)

Rundown of our Experimental Results:

We created a large dataset and timed it; below are specifications used for our testing environment:

- CPU - Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz
- Enabled cores - 2
- Threads - 4
- RAM - 8GiB

To have only one variable in our test, our dataset was created by setting the number of avenues to be equal to the number of roads (#avenues = #roads), we shall call this value “N”. We also set the values for the number of shops and citizens to be 10% of N’s value (ex. N = 10, #Shops/Citizens = 1). N took values between [10, 1000] and was incremented by 10 for each test, essentially our cities were N * N, which meant that our created graph would have ~N^2 nodes. The results in Fig. 3 confirmed our theoretical analysis of our program: we experienced quadratic growth in our function.

Relation between graph size and time taken to find max flow

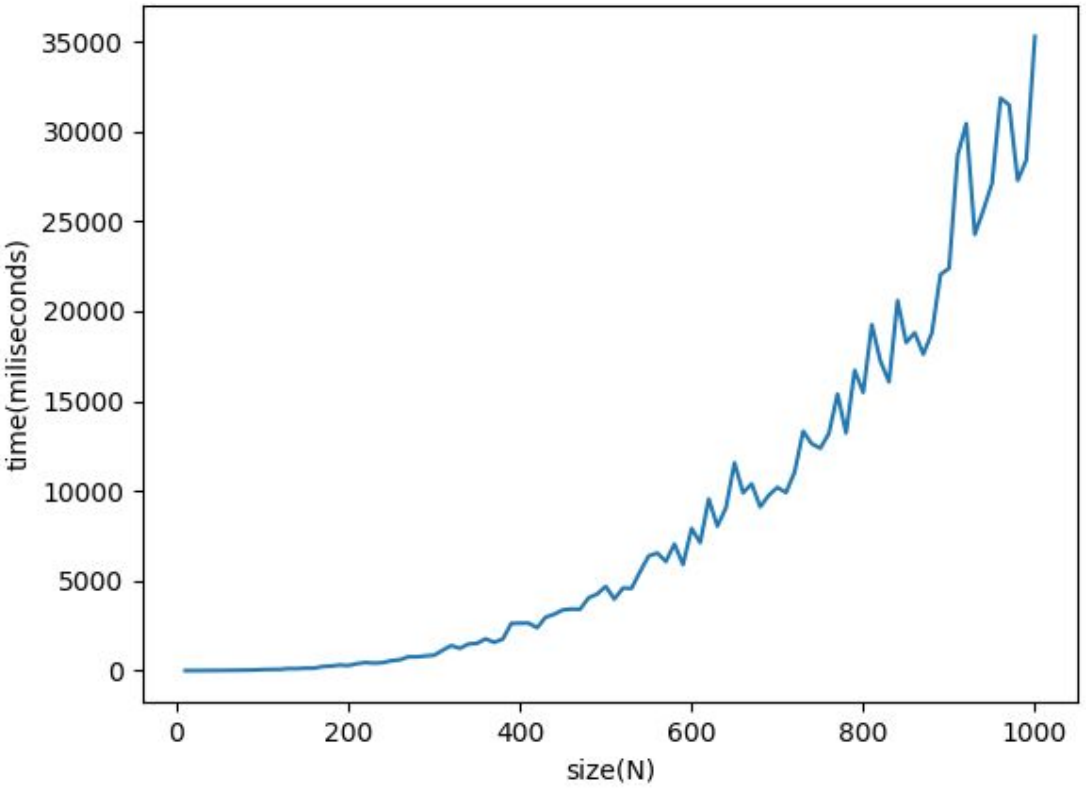


Fig. 3