Homework 5

Andrew Gray
John Piermatteo
Wesley Morlock

Prof. Hamid
CSCI311
11/28/17

Introduction

The Ford-Fulkerson algorithm is an algorithm that tackles the max-flow min-cut problem. Given a network with vertices and edges between those vertices that have certain weights, how much "flow" can the network process at a time. Flow can mean anything, but typically it means data through a computer network. It was discovered in 1956 by Ford and Fulkerson. The Ford-Fulkerson algorithm assumes that the input will be a graph, G, along with a source vertex, s, and a sink vertex, t. The graph is any representation of a weighted graph where vertices are connected by edges of specified weights. There must also be a source vertex and sink vertex to understand the beginning and end of the flow network. Explanations of each class and method can be found below, with the necessary ones containing pseudocode and running time analysis. Some methods are explained through words rather than traditional pseudocode because of the complex nature of the algorithm; if we were to write pseudocode, it would look like normal code.

Discussion:

write_dot_file:
This function generates a .dot flow network file and a visual .pdf file of the flow network.

random_graph_creator:
This function creates a list of connections i.e. “[(1, 2, 4), (2, 3, 9), (3, 2, 4), (3, 3, 1)]” where (start_vertex, end_vertex, capacity) is one connection. The max vertex value is randomly generated between 7 and 15.

class Vertex:
The vertex only has a name, which we use for edge creation (and debugging), and whether or not the vertex is a source or a sink.

class Edge:
This class has a starting vertex, an ending vertex, a capacity, a flow, and a pointer to the return edge which is used in the residual graph. The capacity of an edge is the total weight it can carry whereas an edge's flow is the weight it is currently carrying.

class FlowNetwork:
This class has two attributes. The vertices attribute is a list of all of the instances of vertices in the graph. This list holds entire objects, linking a vertex's name to its source and sink attributes. The network dictionary is a data structure that maps every vertex's name to all of the edges coming out of the corresponding vertex. By simply having the vertex's name act as the dictionary key, we can easily switch back and forth between 'vertices' and 'network' to suit the needs of the algorithm.

FlowNetwork Methods:
This class also contains various helper functions that make later functions easier and help reduce code duplication.

addVertex:
Adds a vertex to the graph after it checks various error cases to make sure the vertex can be added. It adds the entire vertex to the vertices list and adds the vertex name to an empty list of edges.

addEdge:
First checks that both the starting and ending vertex are both in the graph and that they are not the same vertex. Then, it creates the new edge and a corresponding returning edge with capacity 0. It then adds the new edge to the network map and adds the return edge to the same map.

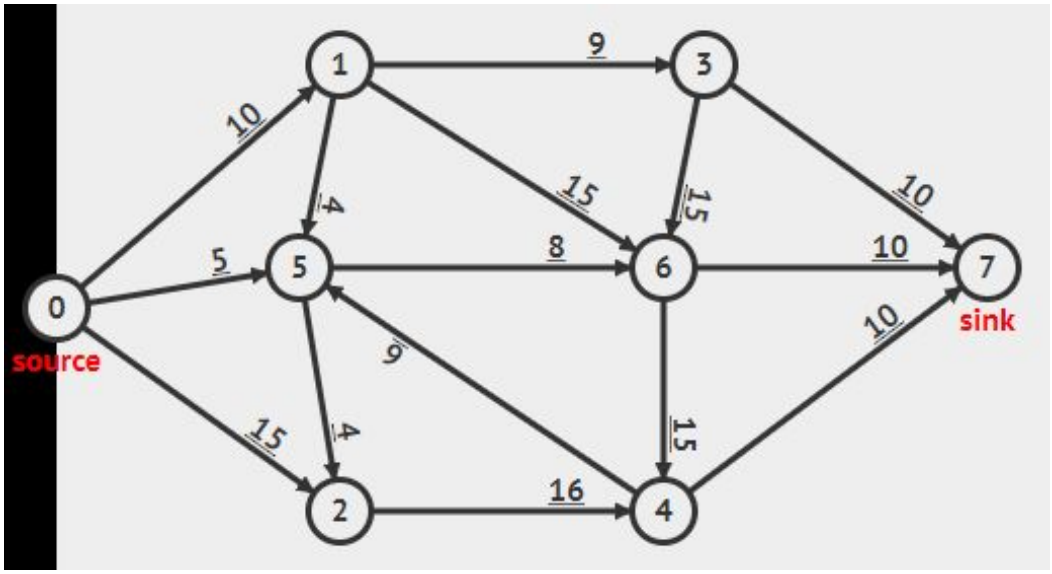
getPath:
A recursive function that walks through the flow network starting at a given vertex and calculates the residual capacity for each edge. This residual capacity is used to decide how much flow to send along a given path in the next function. The path is grown until it reaches the end of the flow network and at that point the base case for the recursion is called and the function ends.

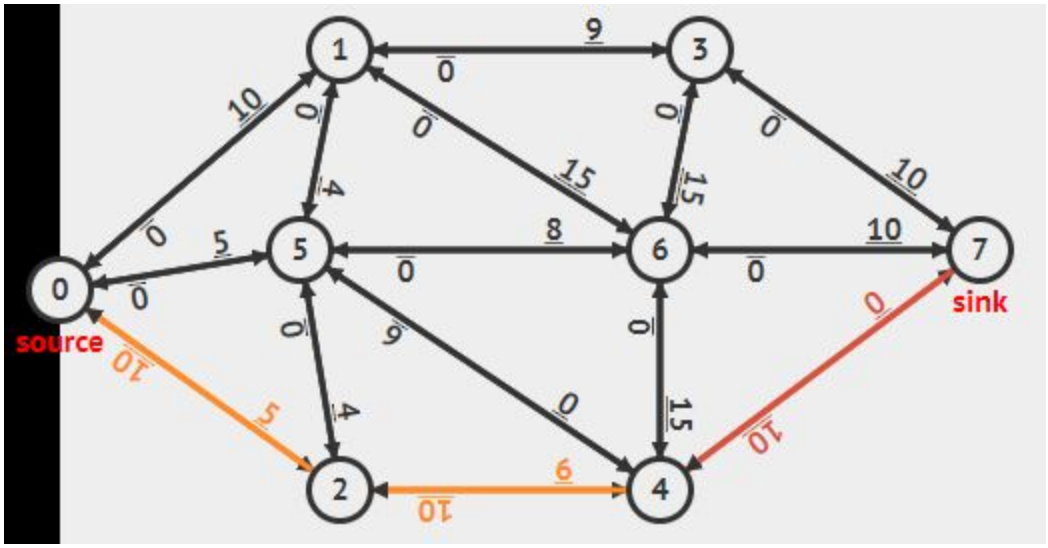
calculateMaxFlow:
Calls getPath with the correct parameters and adds up the max flow. It first finds the source and sink of the network. Then, it calculates an initial augmenting path. Then, we start the Ford-Fulkerson method and continue to calculate flow while there still is a path. While there is a path, an augmenting capacity is in each edge of the path, so a flow can be calculated. That flow is added to the forward edges and subtracted from the reverse edges. Then, another path is calculated and the process resumes. Finally, we only need to calculate the total flow coming out of the source because this is the exact amount of flow through the entire network (since we only have one source).

```
flow = 0
for each edge (u, v) in G:
    flow(u, v) = 0
while there is a path, p, from s -> t in residual network G_f:
    residual_capacity(p) = min(residual_capacity(u, v) : for (u, v) in p)
    flow = flow + residual_capacity(p)
    for each edge (u, v) in p:
        if (u, v) is a forward edge:
            flow(u, v) = flow(u, v) + residual_capacity(p)
        else:
            flow(u, v) = flow(u, v) - residual_capacity(p)
return flow
```

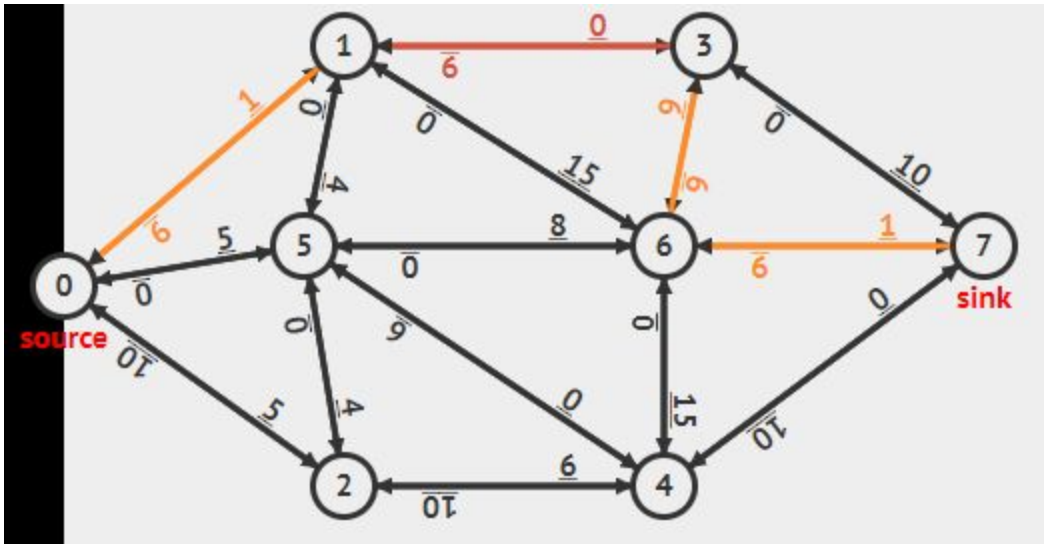
Running Time Analysis:
By adding the flow augmenting path to the flow already established in the graph, the maximum flow will be reached when no more flow augmenting paths can be found in the graph. However, there is no certainty that this situation will ever be reached, so the best that can be guaranteed is that the answer will be correct if the algorithm terminates. In the case that the algorithm runs forever, the flow might not even converge towards the maximum flow. However, this situation only occurs with irrational flow values. When the capacities are integers, the runtime of Ford–Fulkerson is bounded by O(E*f), where E is the number of edges in the graph and f is the maximum flow in the graph. This is because each augmenting path can be found in O(E) time and increases the flow by an integer amount of at least 1, with the upper bound f.

Input/Output Analysis (explain one test case in detail for the test network generator and the max flow generator class, show each intermediate step of the solution)

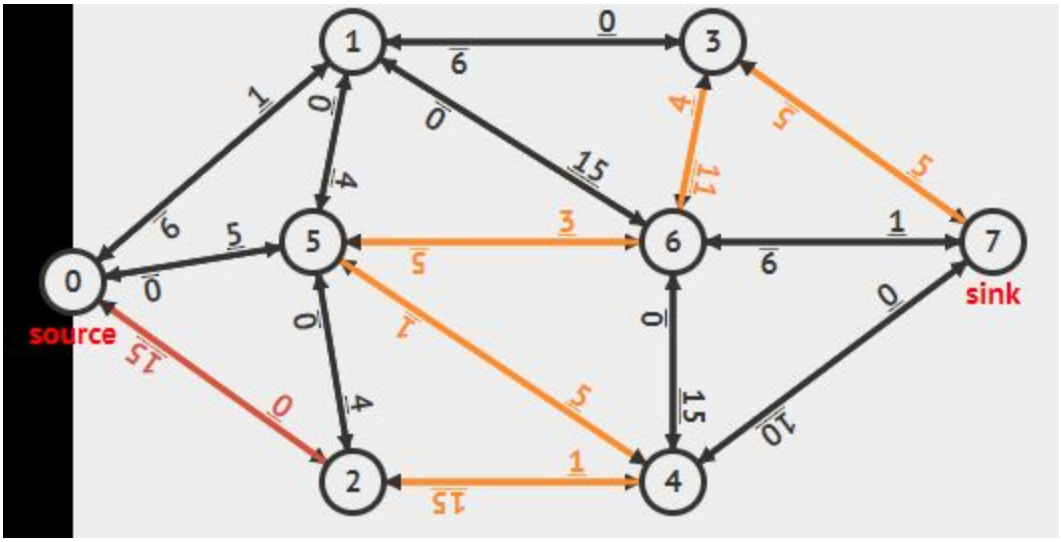




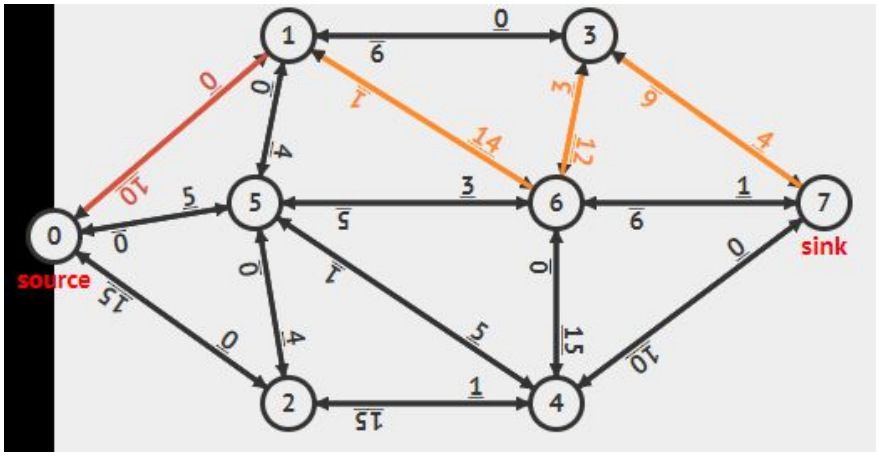
Max flow from 0 to 7 is currently 10



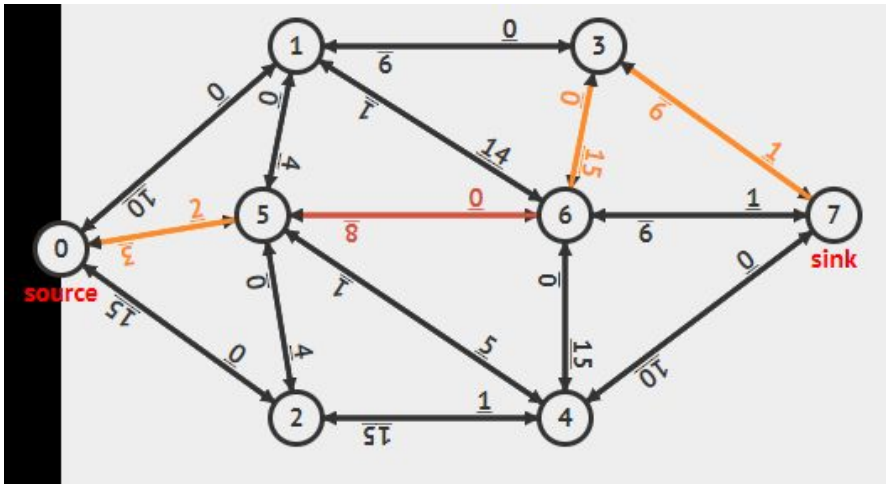
Max flow from 0 to 7 is currently 19 (increased by 9)



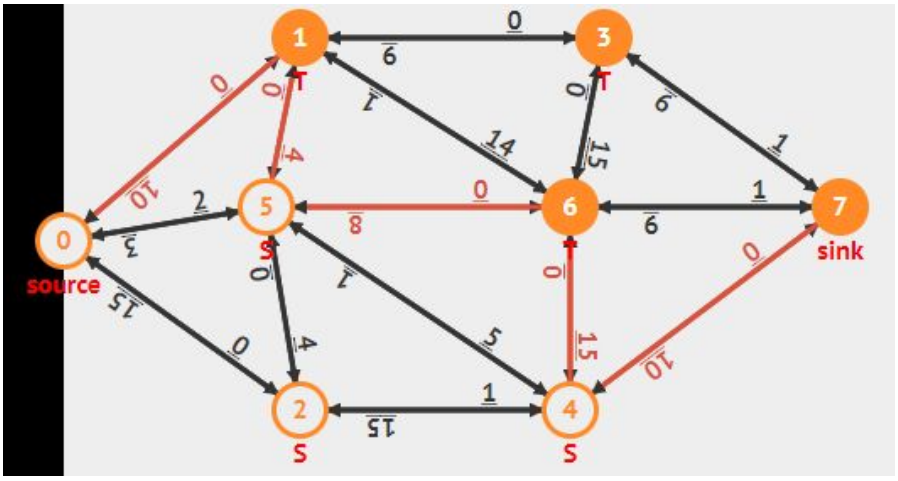
Max flow from 0 to 7 is currently 24 (increased by 5)



Max flow from 0 to 7 is currently 25 (increased by 1)



Max flow from 0 to 7 is currently 28 (increased by 3)



Max flow = min cut from 0 to 7 = 28

Result Analysis

Seen in Appendix A, a test was run with varying sizes of graphs. The number of vertices tested were 7, 9, 11, 13, and 15. The number of edges also varied, due to the code which randomizes a number of edges between |V| and 4|V|. Using the command line, these tests were executed using the "time" command, which results in the time values shown at the end of each snippet of output code. Because each of the executions resulted in very little time, these values do not show much of a trend. This is largely due to the fact that most of the execution time is dependent on how long it takes to calculate the max flow.

However, there is a small trend seen from these results. As the number of edges increases, the run time also increases. Due to the design of the program, as the number of vertices increases, the number of edges associated with that graph usually increases as well. However, through the earlier calculation of the time complexity, it is the number of edges and the max flow that will alter the time it takes to run. This is because creating the graph should not have much of an effect, but it is the edges and the max flow that are large contributors to the run time.

A second test was run, keeping the number of vertices the same (|V| = 7) while the number of edges increased. Here, the results seen in Appendix B, show a clear increase in run time as the number of edges increase. There were three different values of |E| tested so, whereas values much larger with 7 vertices began to take too long to execute the program. From the results though, we do see the runtime of the computer increase, showing the relationship of |E| and the runtime.

Conclusion

The analysis of Ford-Fulkerson depends heavily on how the augmenting paths are found. The typical method is to use breadth-first search to find the path. If this method is used, Ford-Fulkerson runs in polynomial time. If all flows are integers, then the while loop of Ford-Fulkerson is run at most f times, where f is the maximum flow. This is because the flow is increased, at worst, by 1 in each iteration. Finding the augmenting path inside the while loop takes O(V+E) where E is the set of edges in the residual graph. This can be simplified to O(E). And so, the runtime of Ford-Fulkerson is O(E*f). We found that changing the amount of vertices in the network did not have a large effect on how quickly the max flow was calculated. This could be because our algorithm was executing so quickly that it was hard to see a real difference in the running time, but nonetheless, we could not find a substantial effect.

Appendix A:

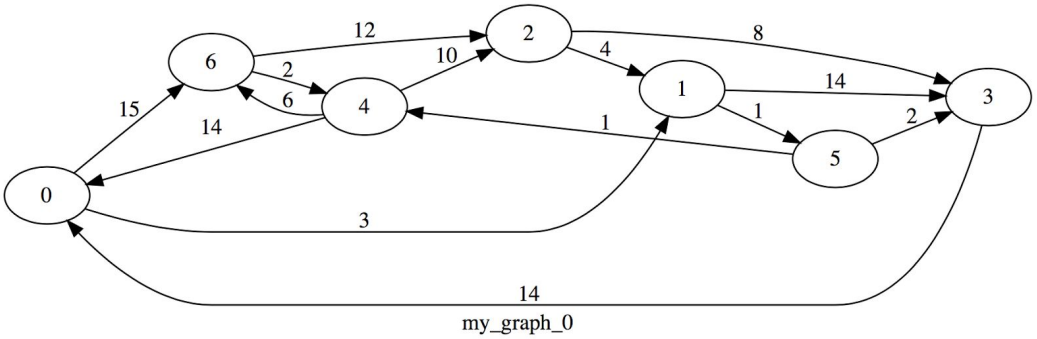


Figure 1: Graph resulting from 7 Vertices

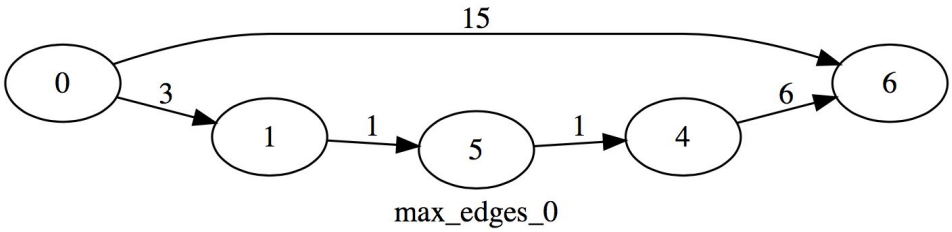


Figure 2: Max Graph resulting from 7 Vertices

No. of VERTICES: 7
No. of EDGES: 14

vertices: ['0', '6', '4', '2', '1', '5', '3']

edges: ['1 -> 2; 0/0', '1 -> 5; 0/1', '1 -> 0; 0/0', '1 -> 3; 0/14', '0 -> 6; 0/15', '0 -> 4; 0/0', '0 -> 1; 0/3', '0 -> 3; 0/0', '3 -> 1; 0/0', '3 -> 2; 0/0', '3 -> 0; 0/14', '3 -> 5; 0/0', '2 -> 6; 0/0', '2 -> 1; 0/4', '2 -> 3; 0/8', '2 -> 4; 0/0', '5 -> 1; 0/0', '5 -> 4; 0/1', '5 -> 3; 0/2', '4 -> 6; 0/0', '4 -> 6; 0/6', '4 -> 5; 0/0', '4 -> 0; 0/14', '4 -> 2; 0/10', '6 -> 0; 0/0', '6 -> 4; 0/2', '6 -> 4; 0/0', '6 -> 2; 0/12']

max flow: 16
max edges: [['0', '6', 15], ['0', '1', 3], ['1', '5', 1], ['5', '4', 1], ['4', '6', 6]]

Time:
real 0m0.168s
user 0m0.058s
sys 0m0.053s

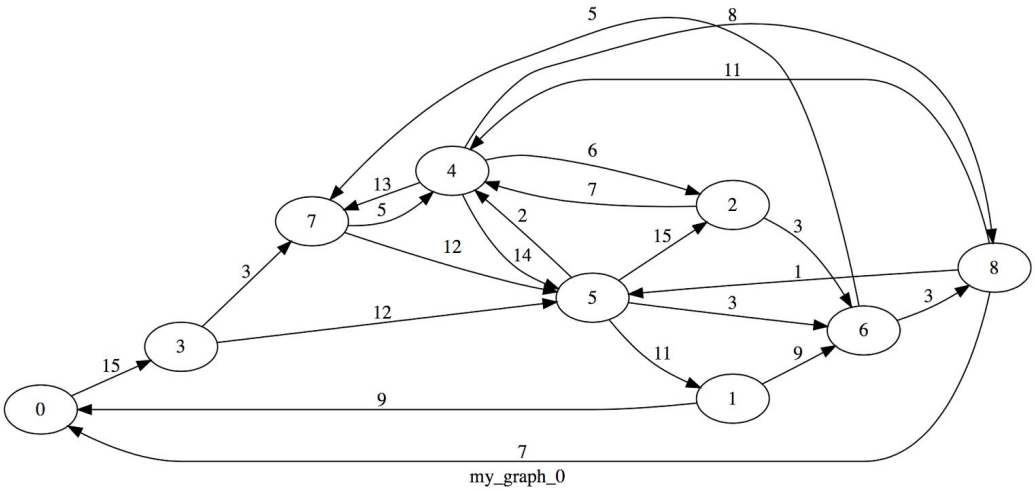


Figure 3: Graph resulting from 9 vertices

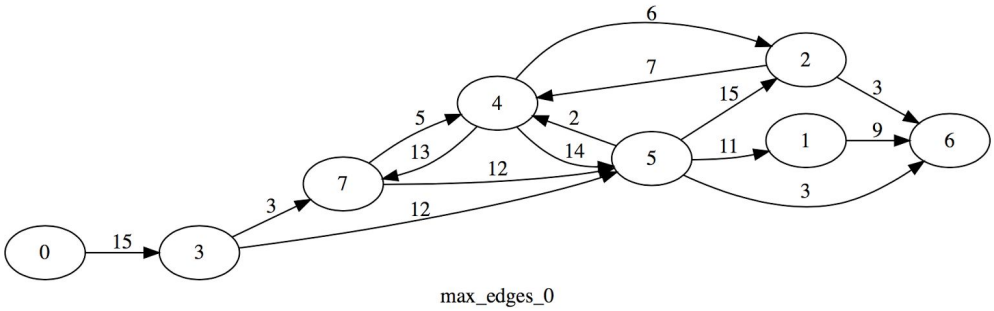


Figure 4: Max Graph resulting from 9 vertices

No. of VERTICES: 9
No. of EDGES: 22

vertices: ['0', '6', '3', '7', '4', '2', '8', '5', '1']

edges: ['1 -> 5; 0/0', '1 -> 6; 0/9', '1 -> 0; 0/9', '0 -> 3; 0/15', '0 -> 1; 0/0', '0 -> 8; 0/0', '3 -> 0; 0/0', '3 -> 7; 0/3', '3 -> 5; 0/12', '2 -> 4; 0/0', '2 -> 6; 0/3', '2 -> 5; 0/0', '2 -> 4; 0/7', '5 -> 7; 0/0', '5 -> 1; 0/11', '5 -> 8; 0/0', '5 -> 4; 0/2', '5 -> 4; 0/0', '5 -> 2; 0/15', '5 -> 3; 0/0', '5 -> 6; 0/3', '4 -> 7; 0/0', '4 -> 2; 0/6', '4 -> 8; 0/0', '4 -> 7; 0/13', '4 -> 5; 0/0', '4 -> 5; 0/14', '4 -> 2; 0/0', '4 -> 8; 0/8', '7 -> 3; 0/0', '7 -> 4; 0/5', '7 -> 4; 0/0', '7 -> 5; 0/12', '7 -> 6; 0/0', '6 -> 2; 0/0', '6 -> 8; 0/3', '6 -> 1; 0/0', '6 -> 5; 0/0', '6 -> 7; 0/5', '8 -> 6; 0/0', '8 -> 4; 0/11', '8 -> 5; 0/1', '8 -> 4; 0/0', '8 -> 0; 0/7']

max flow: 15
max edges: [['0', '3', 15], ['3', '7', 3], ['7', '4', 5], ['4', '2', 6], ['2', '6', 3], ['3', '5', 12], ['5', '1', 11], ['1', '6', 9], ['5', '4', 2], ['4', '7', 13], ['7', '5', 12], ['5', '2', 15], ['2', '4', 7], ['4', '5', 14], ['5', '6', 3]]

Time
real *0m0.181s*
user *0m0.063s*
sys *0m0.055s*

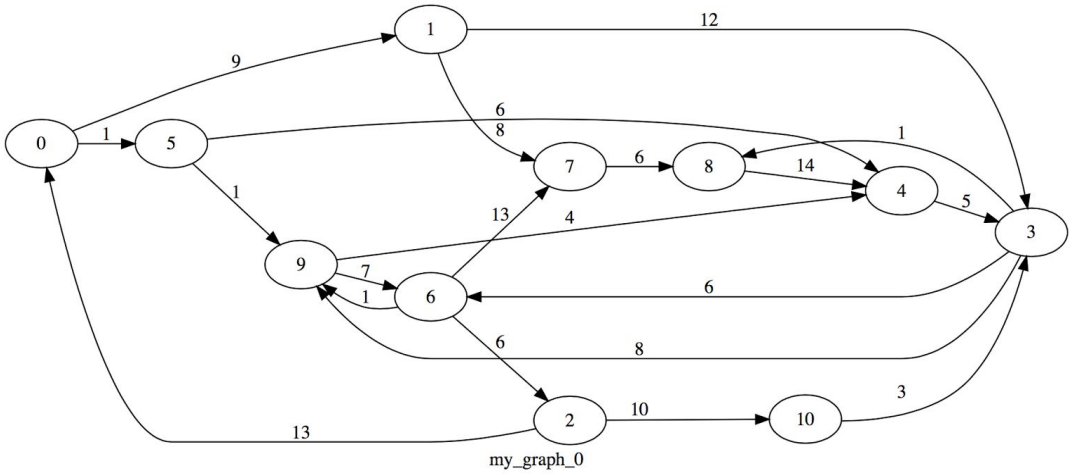


Figure 5: Graph resulting from 11 vertices

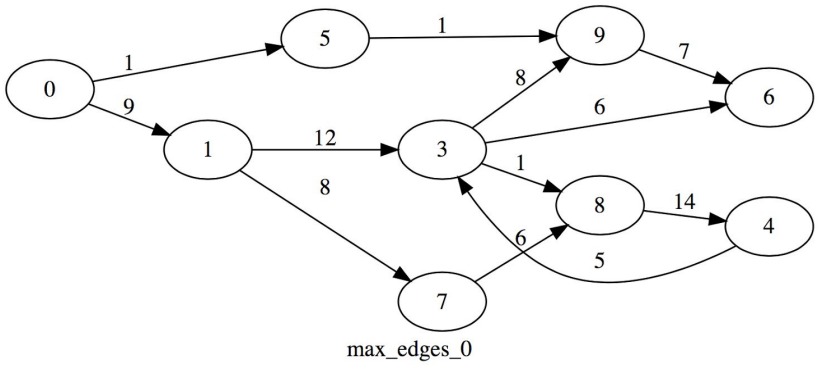


Figure 6: Max Graph resulting from 11 vertices

No. of VERTICES: 11
No. of EDGES: 20

vertices: ['0', '6', '5', '9', '2', '1', '3', '8', '4', '7', '10']

edges: ['10 -> 2; 0/0', '10 -> 3; 0/3', '1 -> 0; 0/0', '1 -> 3; 0/12', '1 -> 7; 0/8', '0 -> 5; 0/1', '0 -> 2; 0/0', '0 -> 1; 0/9', '3 -> 1; 0/0', '3 -> 8; 0/1', '3 -> 4; 0/0', '3 -> 6; 0/6', '3 -> 9; 0/8', '3 -> 10; 0/0', '2 -> 6; 0/0', '2 -> 0; 0/13', '2 -> 10; 0/10', '5 -> 0; 0/0', '5 -> 9; 0/1', '5 -> 4; 0/6', '4 -> 8; 0/0', '4 -> 3; 0/5', '4 -> 5; 0/0', '4 -> 9; 0/0', '7 -> 6; 0/0', '7 -> 1; 0/0', '7 -> 8; 0/6', '6 -> 9; 0/0', '6 -> 2; 0/6', '6 -> 3; 0/0', '6 -> 7; 0/13', '6 -> 9; 0/1', '9 -> 5; 0/0', '9 -> 6; 0/7', '9 -> 3; 0/0', '9 -> 6; 0/0', '9 -> 4; 0/4', '8 -> 3; 0/0', '8 -> 4; 0/14', '8 -> 7; 0/0']

max flow: 10
max edges: [['0', '5', 1], ['5', '9', 1], ['9', '6', 7], ['0', '1', 9], ['1', '3', 12], ['3', '8', 1], ['8', '4', 14], ['4', '3', 5], ['3', '6', 6], ['1', '7', 8], ['7', '8', 6], ['3', '9', 8]]

Time
real 0m0.186s
user 0m0.064s
sys 0m0.057s

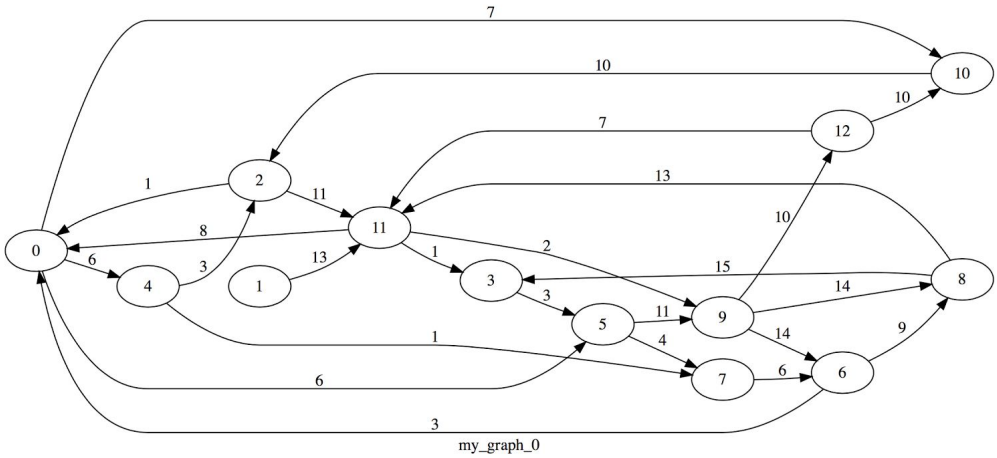


Figure 7: Graph resulting from 13 vertices

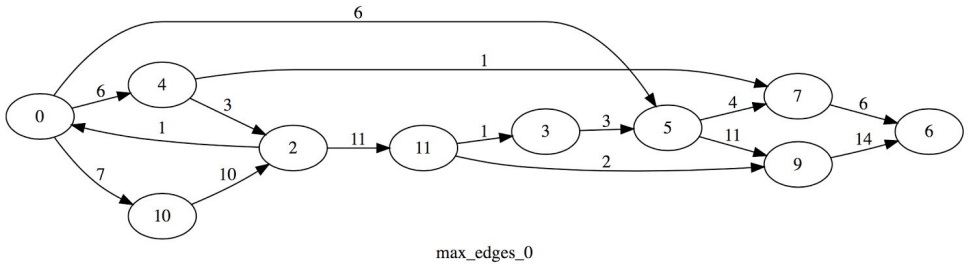


Figure 8: Max Graph resulting from 13 vertices

No. of VERTICES: 13
No. of EDGES: 25

vertices: ['0', '6', '4', '2', '5', '9', '8', '11', '12', '10', '1', '3', '7']

edges: ['11 -> 8; 0/0', '11 -> 9; 0/2', '11 -> 12; 0/0', '11 -> 0; 0/8', '11 -> 1; 0/0', '11 -> 2; 0/0', '11 -> 3; 0/1', '10 -> 0; 0/0', '10 -> 2; 0/10', '10 -> 12; 0/0', '12 -> 9; 0/0', '12 -> 11; 0/7', '12 -> 10; 0/10', '1 -> 11; 0/13', '0 -> 4; 0/6', '0 -> 2; 0/0', '0 -> 5; 0/6', '0 -> 11; 0/0', '0 -> 10; 0/7', '0 -> 6; 0/0', '3 -> 5; 0/3', '3 -> 8; 0/0', '3 -> 11; 0/0', '2 -> 4; 0/0', '2 -> 0; 0/1', '2 -> 11; 0/11', '2 -> 10; 0/0', '5 -> 0; 0/0', '5 -> 9; 0/11', '5 -> 3; 0/0', '5 -> 7; 0/4', '4 -> 0; 0/0', '4 -> 2; 0/3', '4 -> 7; 0/1', '7 -> 4; 0/0', '7 -> 5; 0/0', '7 -> 6; 0/6', '6 -> 9; 0/0', '6 -> 8; 0/9', '6 -> 0; 0/3', '6 -> 7; 0/0', '9 -> 5; 0/0', '9 -> 6; 0/14', '9 -> 11; 0/0', '9 -> 12; 0/10', '9 -> 8; 0/14', '8 -> 6; 0/0', '8 -> 11; 0/13', '8 -> 3; 0/15', '8 -> 9; 0/0']

max flow: 10
max edges: [['0', '4', 6], ['4', '2', 3], ['2', '0', 1], ['0', '5', 6], ['5', '9', 11], ['9', '6', 14], ['2', '11', 11], ['11', '9', 2], ['0', '10', 7], ['10', '2', 10], ['11', '3', 1], ['3', '5', 3], ['5', '7', 4], ['7', '6', 6], ['4', '7', 1]]

Time:
real 0m0.208s
user 0m0.091s
sys 0m0.060s

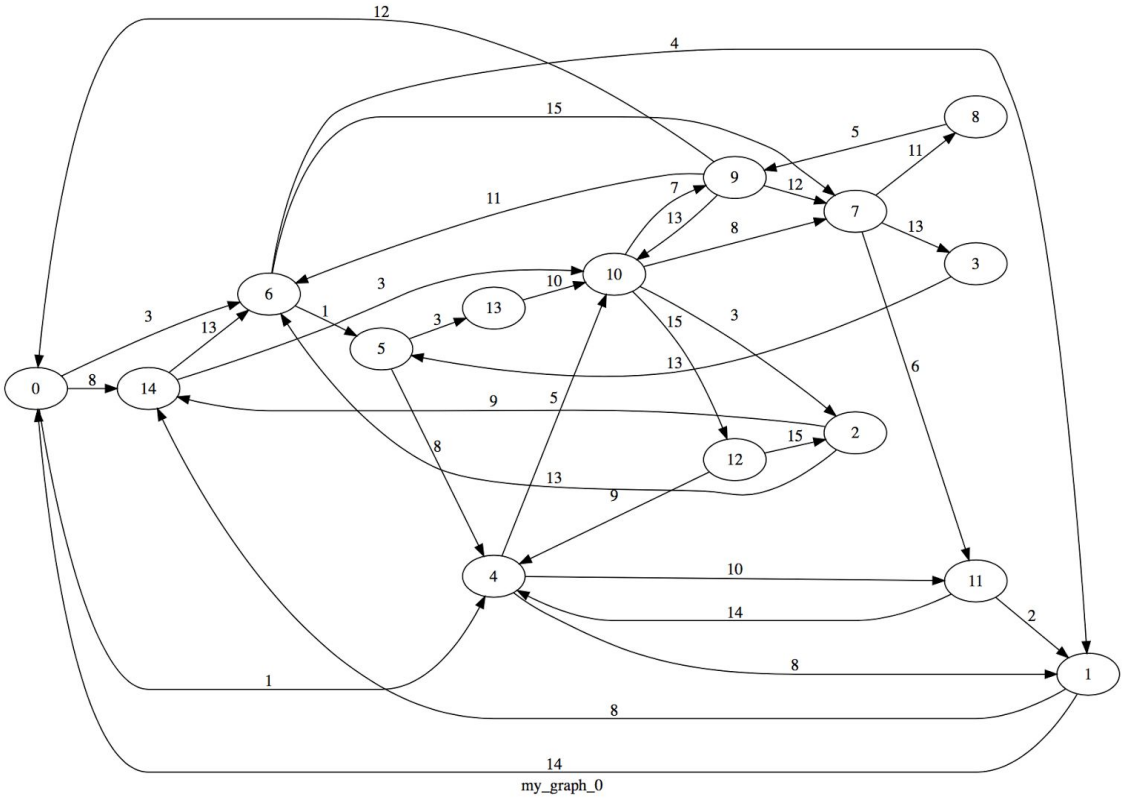


Figure 9: Graph resulting from 15 vertices

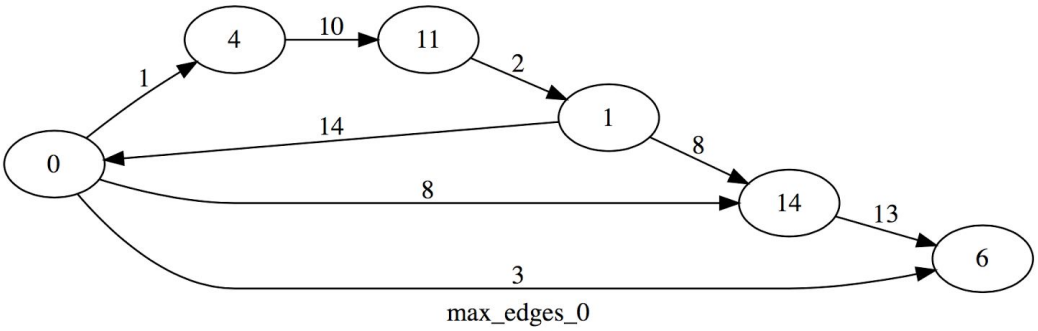


Figure 10: Graph resulting from 15 vertices

No. of VERTICES: 15
No. of EDGES: 35

vertices: ['0', '6', '14', '1', '4', '11', '10', '12', '2', '5', '9', '13', '7', '8', '3']

edges: ['11 -> 4; 0/0', '11 -> 1; 0/2', '11 -> 7; 0/0', '11 -> 4; 0/14', '10 -> 14; 0/0', '10 -> 12; 0/15', '10 -> 4; 0/0', '10 -> 9; 0/7', '10 -> 9; 0/0', '10 -> 13; 0/0', '10 -> 7; 0/8', '10 -> 2; 0/3', '13 -> 5; 0/0', '13 -> 10; 0/10', '12 -> 10; 0/0', '12 -> 2; 0/15', '12 -> 4; 0/9', '14 -> 0; 0/0', '14 -> 6; 0/13', '14 -> 1; 0/0', '14 -> 10; 0/3', '14 -> 2; 0/0', '1 -> 6; 0/0', '1 -> 0; 0/14', '1 -> 11; 0/0', '1 -> 14; 0/8', '1 -> 4; 0/0', '0 -> 14; 0/8', '0 -> 1; 0/0', '0 -> 4; 0/1', '0 -> 9; 0/0', '0 -> 6; 0/3', '3 -> 5; 0/13', '3 -> 7; 0/0', '2 -> 12; 0/0', '2 -> 6; 0/13', '2 -> 14; 0/9', '2 -> 10; 0/0', '5 -> 6;

0/0', '5 -> 4; 0/8', '5 -> 13; 0/3', '5 -> 3; 0/0', '4 -> 0; 0/0', '4 -> 11; 0/10', '4 -> 5; 0/0', '4 -> 10; 0/5', '4 -> 11; 0/0', '4 -> 1; 0/8', '4 -> 12; 0/0', '7 -> 10; 0/0', '7 -> 8; 0/11', '7 -> 6; 0/0', '7 -> 11; 0/6', '7 -> 3; 0/13', '7 -> 9; 0/0', '6 -> 14; 0/0', '6 -> 1; 0/4', '6 -> 2; 0/0', '6 -> 5; 0/1', '6 -> 0; 0/0', '6 -> 9; 0/0', '6 -> 7; 0/15', '9 -> 10; 0/0', '9 -> 10; 0/13', '9 -> 0; 0/12', '9 -> 8; 0/0', '9 -> 6; 0/11', '9 -> 7; 0/12', '8 -> 7; 0/0', '8 -> 9; 0/5']

max flow: 12
max edges: [['0', '14', 8], ['14', '6', 13], ['0', '4', 1], ['4', '11', 10], ['11', '1', 2], ['1', '0', 14], ['0', '6', 3], ['1', '14', 8]]

Time:
real *0m0.172s*
user *0m0.069s*
sys *0m0.053s*

Appendix B:

No. of VERTICES: 7
No. of EDGES: 16

vertices: ['0', '6', '1', '4', '3', '2', '5']

edges: ['1 -> 0; 0/0', '1 -> 4; 0/12', '1 -> 3; 0/0', '1 -> 6; 0/0', '1 -> 6; 0/9', '1 -> 5; 0/14', '0 -> 1; 0/15', '0 -> 3; 0/0', '0 -> 3; 0/8', '0 -> 6; 0/0', '0 -> 2; 0/15', '3 -> 4; 0/0', '3 -> 0; 0/7', '3 -> 0; 0/0', '3 -> 1; 0/14', '3 -> 2; 0/0', '2 -> 0; 0/0', '2 -> 5; 0/5', '2 -> 3; 0/10', '5 -> 2; 0/0', '5 -> 1; 0/0', '5 -> 6; 0/6', '5 -> 6; 0/0', '4 -> 1; 0/0', '4 -> 3; 0/6', '4 -> 6; 0/14', '6 -> 4; 0/0', '6 -> 1; 0/15', '6 -> 1; 0/0', '6 -> 0; 0/7', '6 -> 5; 0/0', '6 -> 5; 0/7']

max flow: 27
max edges: [['0', '1', 15], ['1', '4', 12], ['4', '3', 6], ['3', '0', 7], ['0', '3', 8], ['3', '1', 14], ['1', '6', 9], ['4', '6', 14], ['0', '2', 15], ['2', '5', 5], ['5', '6', 6], ['2', '3', 10], ['1', '5', 14]]

Time:
real 0m0.225s
user 0m0.144s
sys 0m0.046s

No. of VERTICES: 7
No. of EDGES: 21

vertices: ['0', '6', '4', '1', '2', '3', '5']

edges: ['1 -> 6; 0/0', '1 -> 2; 0/13', '1 -> 5; 0/0', '1 -> 6; 0/6', '0 -> 4; 0/12', '0 -> 4; 0/0', '0 -> 6; 0/9', '0 -> 3; 0/0', '3 -> 2; 0/0', '3 -> 2; 0/14', '3 -> 5; 0/3', '3 -> 6; 0/0', '3 -> 6; 0/8', '3 -> 0; 0/8', '2 -> 1; 0/0', '2 -> 3; 0/12', '2 -> 3; 0/0', '2 -> 4; 0/4', '2 -> 4; 0/0', '2 -> 5; 0/0', '2 -> 5; 0/12', '2 -> 6; 0/0', '5 -> 3; 0/0', '5 -> 1; 0/2', '5 -> 4; 0/0', '5 -> 2; 0/13', '5 -> 2; 0/0', '4 -> 0; 0/0', '4 -> 0; 0/1', '4 -> 2; 0/0', '4 -> 2; 0/2', '4 -> 6; 0/0', '4 -> 5; 0/2', '4 -> 6; 0/7', '6 -> 0; 0/0', '6 -> 1; 0/11', '6 -> 1; 0/0', '6 -> 3; 0/14', '6 -> 3; 0/0', '6 -> 4; 0/9', '6 -> 4; 0/0', '6 -> 2; 0/9']

max flow: 20
max edges: [['0', '4', 12], ['4', '0', 1], ['0', '6', 9], ['4', '2', 2], ['2', '3', 12], ['3', '2', 14], ['2', '4', 4], ['4', '5', 2], ['5', '1', 2], ['1', '6', 6], ['1', '2', 13], ['2', '5', 12], ['4', '6', 7], ['3', '6', 8], ['5', '2', 13]]

Time:
real 0m0.278s
user 0m0.161s
sys 0m0.054s

No. of VERTICES: 7
No. of EDGES: 25

vertices: ['0', '6', '2', '4', '5', '1', '3']

edges: ['1 -> 4; 0/0', '1 -> 4; 0/1', '1 -> 6; 0/0', '1 -> 2; 0/5', '1 -> 0; 0/0', '1 -> 0; 0/5', '1 -> 3; 0/0', '0 -> 2; 0/15', '0 -> 2; 0/0', '0 -> 5; 0/9', '0 -> 6; 0/1', '0 -> 6; 0/0', '0 -> 1; 0/5', '0 -> 1; 0/0', '3 -> 2; 0/0', '3 -> 2; 0/3', '3 -> 6; 0/12', '3 -> 1; 0/13', '3 -> 6; 0/0', '2 -> 0; 0/0', '2 -> 4;

0/9', '2 -> 4; 0/0', '2 -> 5; 0/4', '2 -> 5; 0/0', '2 -> 0; 0/12', '2 -> 6; 0/0', '2 -> 1; 0/0', '2 -> 3; 0/14', '2 -> 3; 0/0', '5 -> 4; 0/0', '5 -> 4; 0/9', '5 -> 2; 0/0', '5 -> 2; 0/3', '5 -> 0; 0/0', '4 -> 2; 0/0', '4 -> 5; 0/1', '4 -> 5; 0/0', '4 -> 2; 0/13', '4 -> 6; 0/0', '4 -> 1; 0/13', '4 -> 1; 0/0', '4 -> 6; 0/12', '6 -> 0; 0/0', '6 -> 2; 0/6', '6 -> 0; 0/5', '6 -> 4; 0/12', '6 -> 4; 0/0', '6 -> 1; 0/8', '6 -> 3; 0/0', '6 -> 3; 0/14']

max flow: 25
max edges: [['0', '2', 15], ['2', '4', 9], ['4', '5', 1], ['5', '4', 9], ['4', '2', 13], ['2', '5', 4], ['5', '2', 3], ['2', '0', 12], ['0', '6', 1], ['4', '1', 13], ['1', '4', 1], ['4', '6', 12], ['0', '5', 9], ['0', '1', 5], ['2', '3', 14], ['3', '6', 12], ['1', '2', 5]]

mkdir: input_graphs: File exists
mv: rename my_graph_* to input_graphs/my_graph_*: No such file or directory

Time:
real 0m0.385s
user 0m0.264s
sys 0m0.058s
