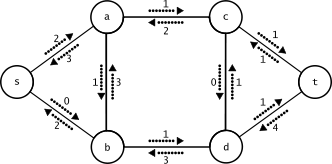
****

**Maximum Flow in A Network Flow Problem**

**Report by:**

**Hasan Haider**

**Zainab Ali Vyajkora**

**Laiba Fatima Khan**

**Abira Jeoffery**

*Compiled as part of the course* ***Algorithms: Design and Analysis*** *at Habib University, November 30, 2020*

# **Problem Description:**

**Network flow problems** are a class of computational problems in which the input is a [flow network](https://en.wikipedia.org/wiki/Flow_network) (a graph with numerical capacities on its edges), and the goal is to construct a [flow](https://en.wikipedia.org/wiki/Flow_network#Flows), numerical values on each edge that respect the capacity constraints and that have incoming flow equal to outgoing flow at all vertices except for certain designated terminals.

The [maximum flow problem](https://en.wikipedia.org/wiki/Maximum_flow_problem): the goal is to maximize the total amount of flow out of the source terminals and into the sink terminals.

## Practical Applications:

1. **Computer Networks:** Routing as many packets as possible on a given network.
2. **Transportation:** Sending as many trucks as possible, where roads have limits on the number of trucks per unit time.
3. **Bridges:** Destroying some bridges to disconnect s from t, while minimizing the cost of destroying the bridges.
4. **Bipartite matching:** Finding a maximum cardinality matching.

## Algorithms used:

The following algorithms have been used as approaches to solve the maximum flow problem stated above:

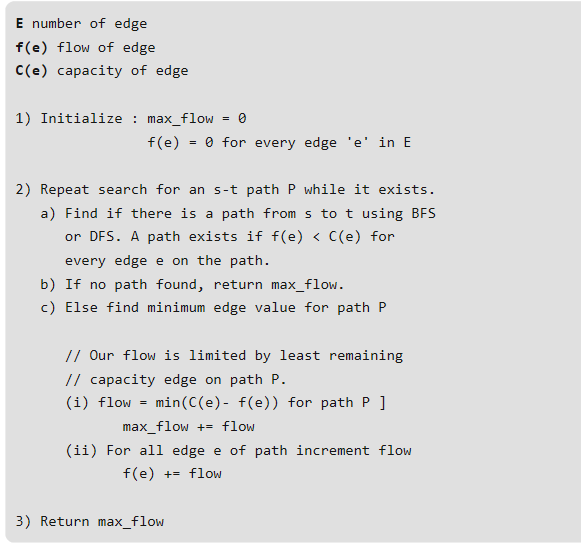
1. Naïve Algorithm

2. Ford- Fulkerson Algorithm

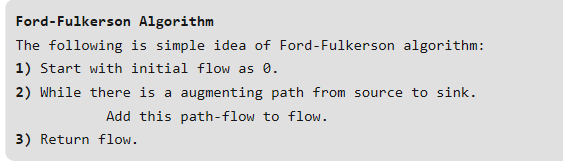
3. Edmond-Karp Algorithm

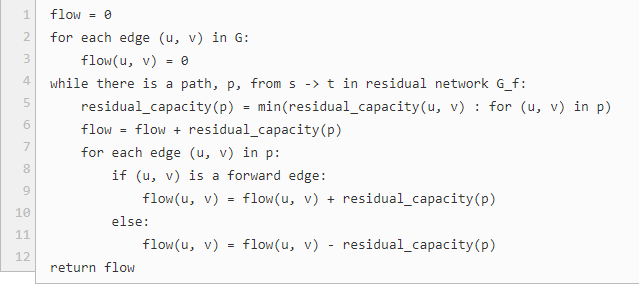
4. Dinic’s Algorithm

1. **Naive Approach:**



1. **Ford-Fulkerson Method:**

****

****

1. **Edmonds-Karp Approach:**

**Input**: Graph (graph[v] should be the list of edges coming out of vertex v in the original graph and their corresponding constructed reverse edges which are used for push-back flow. Each edge should have a capacity, flow, source and sink as parameters, as well as a pointer to the reverse edge.) s (Source vertex) t (Sink vertex)

**Output**: flow (Value of maximum flow) flow := 0 (Initialize flow to zero)

repeat (Run a bfs to find the shortest s-t path. We use 'pred' to store the edge taken to get to each vertex, so we can recover the path afterwards)

q := queue() q.push(s) pred := array(graph.length)

while not empty(q) cur := q.pull()

for Edge e in graph[cur] if pred[e.t] = null and e.t ≠ s and e.cap > e.flow pred[e.t] := e q.push(e.t)

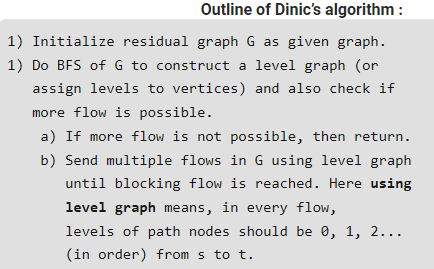
if not (pred[t] = null) (We found an augmenting path. See how much flow we can send) df := ∞

for (e := pred[t]; e ≠ null; e := pred[e.s]) df := min(df, e.cap - e.flow) (And update edges by that amount) for (e := pred[t]; e ≠ null; e := pred[e.s]) e.flow := e.flow + df e.rev.flow := e.rev.flow - df flow := flow + df

until pred[t] = null (i.e., until no augmenting path was found)

return flow

1. **Dinic's Method:**



## Theoretical Time Complexities:

|  |  |  |
| --- | --- | --- |
| S. No | Algorithm | Theoretical Time Complexity |
| 1. | Naive Algorithm | O(E max| f |) |
| 2. | Ford-Fulkerson Algorithm | O(E max| f |) |
| 3. | Edmonds-Karp Algorithm | O(V E2) |
| 4. | Dinic's Algorithm | O(V2 E) |

## Empirical Complexities - Graphs and Analysis:

**Conclusion:**

Hence, we can observe through empirical analysis that over vary large graphs, Dinic’s Algorithm is the fastest followed by Naïve, Ford- Fulkerson and Edmond-Karp algorithm (in that order).

## Limitations And Specifications

1. The analysis was performed for at most a graph with 2000 nodes. The behavior might change if the size is increased.
2. It is assumed that the entered graph would always have a source and a sink node. Incorrect graphs have not been taken into account.
3. The algorithms were coded in python.
4. Majority of the generated graphs are dense. Things may vary for sparse graphs.
5. The graphs were stored as adjacency lists. The complexities may vary if they were stored in some other data structure.
6. Machine specifications:

- Processor: 2.4 GHz

- Memory: 16 GB

**References:**

1. https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/

2. https://www.cs.princeton.edu/~wayne/cs423/lectures/max-flow-applications

3. https://web.stanford.edu/class/cs97si/08-network-flow-problems.pdf

4. https://visualgo.net/en/maxflow

5. https://brilliant.org/wiki/ford-fulkerson-algorithm/