CS375 Project Presentation
Maximum Flow Problem

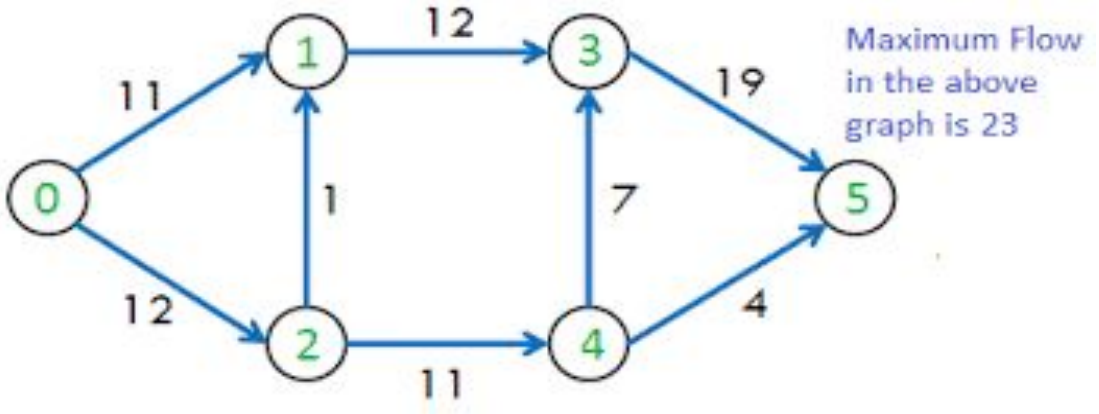
By: Rick Kabuto and Ethan Bown

What is the Maximum Flow Problem

Goal - To find the greatest possible “flow” of a commodity through a network from a **source node** to a **sink node**

Where is it mostly used?

- **Network optimization**
  - Telecommunications(Data Transfer)
- **Transportation**
  - Traffic Flow
- **Supply Chain**
  - Transfer of goods across a network



What Algorithms Solve This?

- Ford-Fulkersons
- Push-Relabel
- Edmonds-Karp Algorithm
- Capacity Scaling
- Shortest Augmenting Path Algorithm
- Dinic's Algorithm

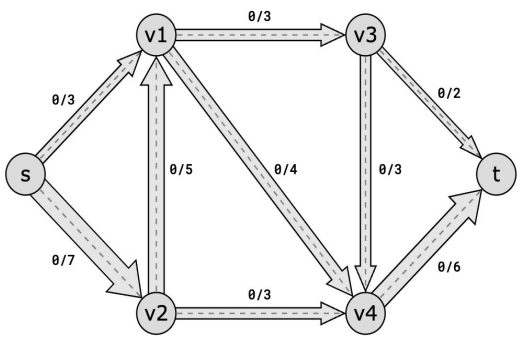
Project Goal

Edmonds-Karp Algorithm vs Dinic Algorithm

(Both guarantees to find the maximum flow)

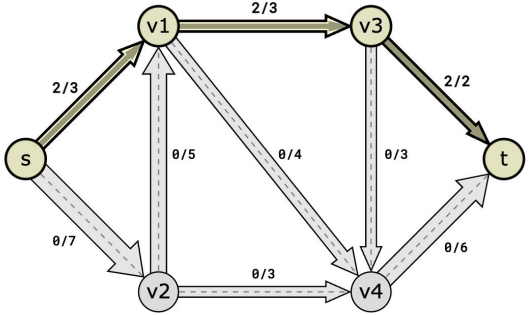
Edmonds-Karp Algorithm

No flow in the graph to start with.



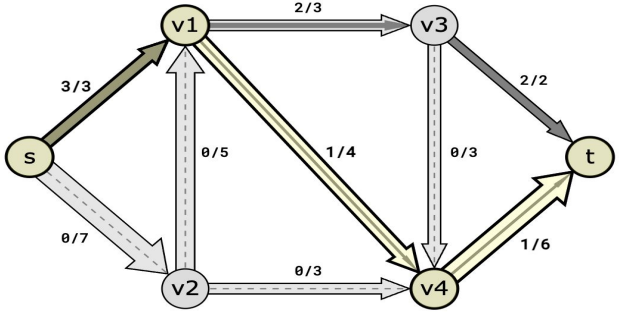
->

Breadth-First Search to find an augmented path where flow can be increased. S->V1->V3->T.

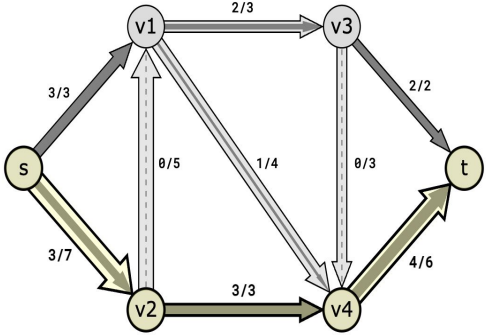


->

Next iteration we find a new augmented path. Find how much the flow in that path can be increased. The flow can only be increased by 1 with, S->V1->V4->T

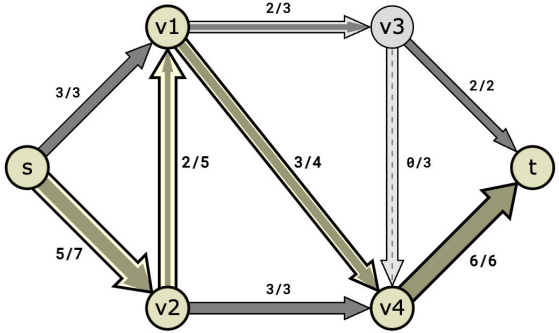


Find next augmented path. S->V1->V4->T can increase the flow by 3



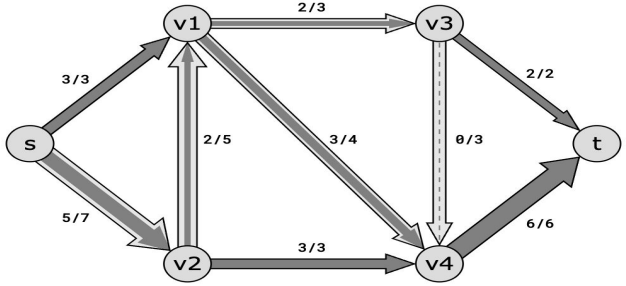
->

Last augmented path is S->V2->V1->V4->T and can increase the flow by 2.



->

Finally, it is not possible to find a path where more flow can be sent through from s to t which means the max flow has been found, Thus the maximum flow is 8.



Max flow: 8

Edmonds-Karp Pseudocode Code

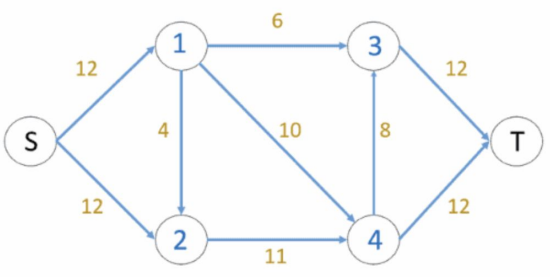
Create a function that takes three arguments(a source, a sink, and a residual graph)

- Build Level Graph
- Perform BFS on residual Graph starting from source
  - Track parent of each visited node to reconstruct the path.
  - Stop BFS when sink is reached or no path exists.
  - If no path is found return that there exist no augmenting path
- Construct the augmenting path from source to sink
  - Use the parent array from BFS to trace the path
  - Determine bottleneck capacity (min residual capacity along the path)
  - Update flow along the path:
    - For each edge in the path:
      - Decrease forward edge capacity by bottleneck.
      - Increase reverse edge capacity by bottleneck.
  - Update the residual graph.
- Return updated residual graph and bottleneck value.

Time Complexity - **$O(VE^2)$**.

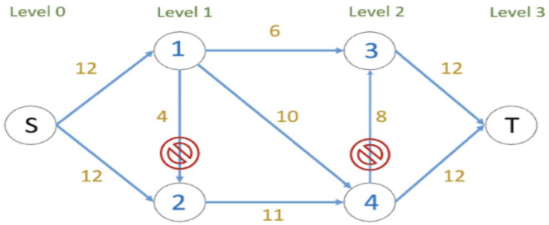
Dinic's Algorithm`

Initial Residual Graph



->

Use BFS to assign levels to all nodes. Then, we investigate whether an additional flow is possible or if there is a S->T path in the residual graph

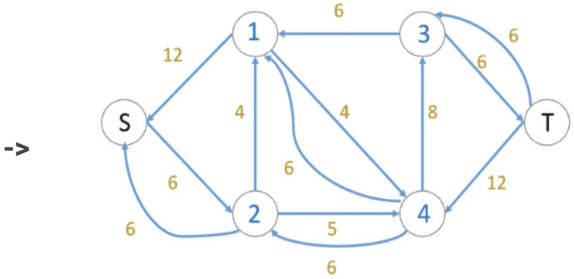


->

Note: Edges labeled with a block symbol indicate they cannot be used to transfer flow. Blocking flow is determined based on levels, where every flow path must follow sequential levels such as 0, 1, 2, 3. Unlike the Edmonds-Karp algorithm, which sends one flow at a time, this method sends multiple flows simultaneously.

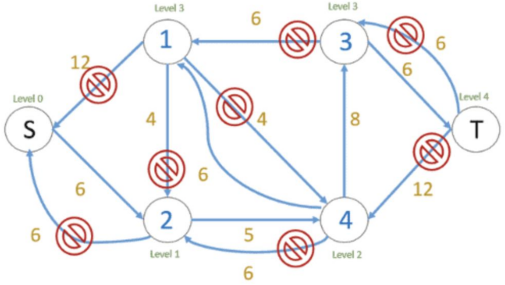
S->1->3->T = 6
S->1->4->T = 6
S->2->4->T = 6

Total Flow = 6+6+6 =18



->

Next we assign new levels to all nodes and investigate whether an additional flow is possible from S->T

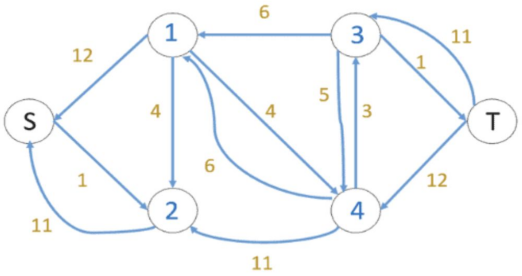


->

Edges marked with a block symbol indicate that they cannot carry any flow. Only one flow can be sent at a time.

$S \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow T$

Total flow = previous + 5 = 23.



->

Lastly, we run BFS and draw a level graph and determine whether more flow is possible. In this graph example, There is no path so the algorithm ends

Dinic's Pseudo Code

Create a function with 3 arguments(a source, a sink, and a residual Graph)

Step 1) Build Level Graph

- BFS from the source:
  - Designate each node with a level by its shortest distance from the source
  - If an edge has a zero residual capacity - do not include that edge
  - If sink is not reachable, return that there exist no augmenting path

Step 2) Find Blocking Flow

- Create a flow counter and set it to zero
- DFS to push flow from source node to the sink node
  - For each edge in the level
    - Path's minimum residual capacity determines the flow, which is set equal.
    - Continue to update the forward and reverse edge capacities
    - Add up the flow pushed into the counter we created.
- Stop when no more augmenting paths exist in the level graph.

Time Complexity - **$O(V^2E)$**.

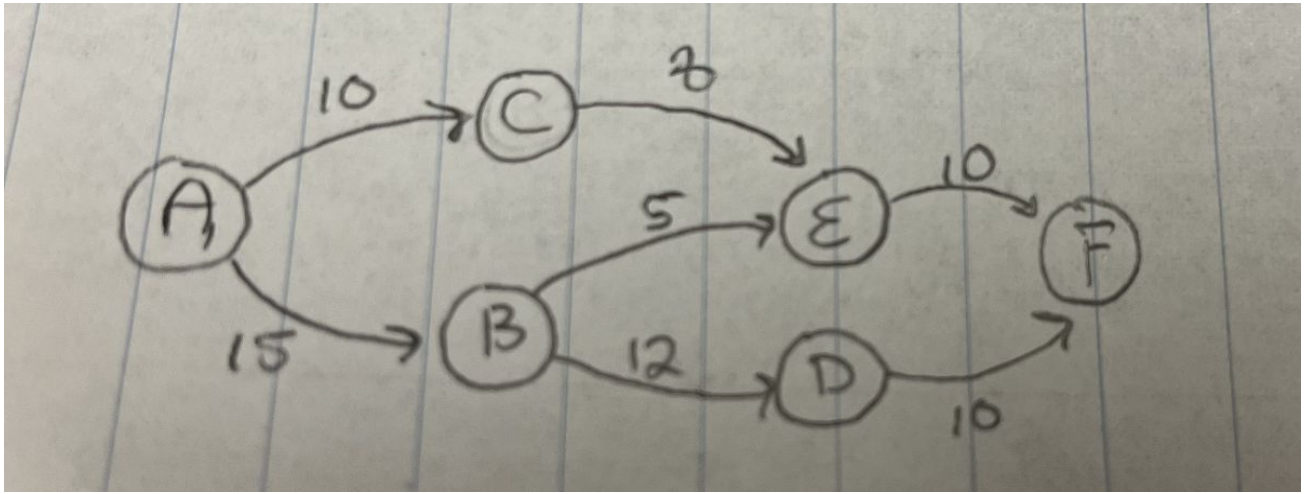
Step 3) Update Residual Graph

- Update the capacities along the paths where flow was transferred in step 2

Step 4) Return the updated Graph and total.

Real World Example

Real World Question - In a city, a network of roads connects intersections, and each road has a specific maximum capacity, representing the number of cars it can handle per unit time. Traffic Engineers need to determine the maximum number of cars that can travel from a central parking area to a stadium during an event without causing congestion. The roads between intersections follow the graph below. **Find the maximum flow of cars from the starting node A(the central parking area) to the end node F(the stadium).**



Results Comparison

Edmonds-Karp Algorithm and Dinic's Algorithm

```
GeeksforGeeks Test Case
--------------------------------
Expected Output: 23
Actual Output: Maximum Flow: 23
Execution Time: 13 microseconds


Real-World Scenario Word Problem
--------------------------------
Expected Output: 20 Cars
Actual Output: Maximum Flow: 20 Cars
Execution Time: 10 microseconds
```

GeeksforGeeks Base Case

```
vector<vector<int>> geeksforgeeksGraph = {
    {0, 16, 13, 0, 0, 0},
    {0, 0, 10, 12, 0, 0},
    {0, 4, 0, 0, 14, 0},
    {0, 0, 9, 0, 0, 20},
    {0, 0, 0, 7, 0, 4},
    {0, 0, 0, 0, 0, 0}
};
int source = 0;
int sink = 5;
```

Expected Output: 23

Conclusion

Edmonds-Karp Algorithm

- Complexity: **$O(VE^2)$**.
- ***Edge Heavy Graphs*** - Where the number of edges significantly exceeds the number of vertices. Also known as Sparse Graphs.
    - ***Real World Example*** - A road network graph with cities as nodes and roads as edges.
        - Fewer cities and many interconnecting roads, Thus, Edmonds-Karp is more efficient.
        - Calculating maximum car flow between two major hubs in a densely connected highway network.

Dinic's Algorithm

- Complexity: **$O(V^2E)$**.
- ***Vertex Heavy Graphs*** - where the number of vertices is large compared to the number of edges. Also known as Dense Graphs
    - ***Real World Example*** - A social network graph with users as nodes and occasional direct connections between friends. as the edges between friends.
        - Will have many vertices but fewer edges
        - Calculating maximum message flow in a peer-to-peer social platform.

Sources

Bouguezzi, S. (2023, May 31). Network flow: Dinic's algorithm. Baeldung. Retrieved November 23, 2024, from https://www.baeldung.com/cs/dinics

Erickson, J. (2019). *Applications of flows and cuts*. University of Illinois at Urbana-Champaign. https://jeffe.cs.illinois.edu/teaching/algorithms/book/11-maxflowapps.pdf

GeeksforGeeks. (2023, June 1). *Ford-Fulkerson algorithm for maximum flow problem*. GeeksforGeeks. https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/

Udacity. (2015, February 23). *Analysis of Edmonds-Karp - Georgia Tech - Computability, Complexity, Theory: Algorithms* [Video]. YouTube. https://www.youtube.com/watch?v=FlIB73vSl4s

Wikipedia contributors. (n.d.). *Dinic's algorithm*. In Wikipedia, The Free Encyclopedia. Retrieved November 23, 2024, from https://en.wikipedia.org/wiki/Dinic%27s_algorithm

WilliamFiset. (2018, November 7). *Dinic's Algorithm | Network Flow | Graph Theory* [Video]. YouTube. https://www.youtube.com/watch?v=M6cm8UeeziI

W3Schools. (n.d.). *Edmonds-Karp algorithm*. W3Schools. Retrieved November 23, 2024, from https://www.w3schools.com/dsa/dsa_algo_graphs_edmondskarp.php

Questions?