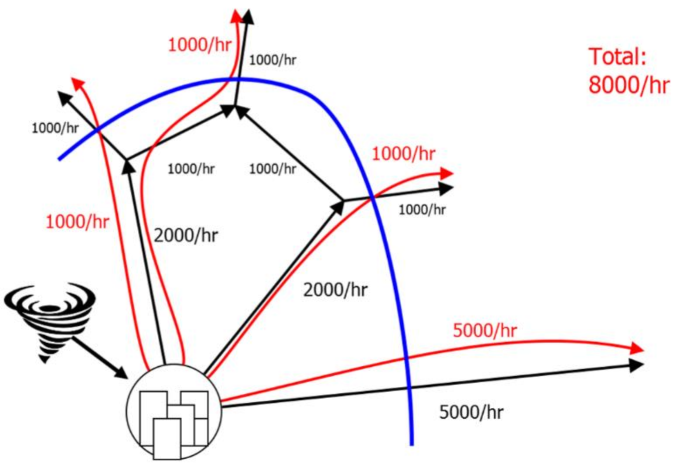
Maximum flow algorithms

1)Introduction:

so basically maximum flow algorithms are a special kind of algorithms which are used to find maxflow in a network. For example consider the case of evacuation of a city which is hit by a tornado, in such kind of situations we need to find a naive way to evacuate as many people as we can. We can do so by applying the concept of maximum flow algorithms which involves finding the path among all the possible networks that can carry maximum number of people.



2)Possible approaches:

Residual networks

Given network G, flow f. we construct a residual network Gf, representing places where flow can still be added to f, including places where existing flow can be cancelled. For each edge e of G, Gf has edges: e with capacity Ce −fe (unless fe = Ce). opposite e with capacity fe (unless fe = 0).

Ford Fulkerson:

Start with zero flow.

Repeatedly add flow.

Stop when you cannot add more.

Have flow f.

Compute residual Gf.

Any new flow f + g, where g a flow for Gf.

Need to find flow for Gf.

See if there's a source-sink path.

If there's no source-sink path in Gf:

Reachable vertices define cut of size 0.

No g of positive size. $|f + g| \leq |f|$.

f is a maxflow.

If there is a path, add flow along path

Find flow g for G_f with $|g| > 0$.

Replace f by $f + g$.

$|f + g| > |f|$.

$f \leftarrow 0$ repeat:

Compute G_f

Find $s - t$ path P in G_f

    if no path:

     return f

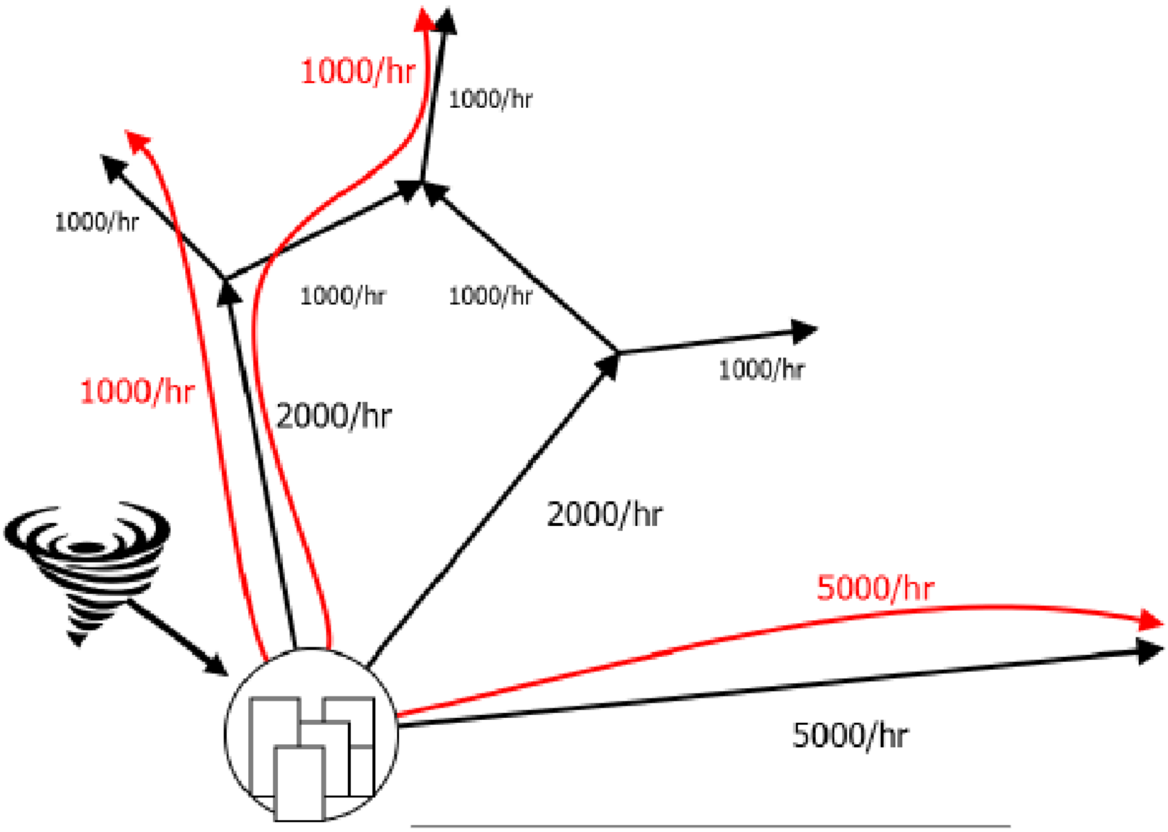
$X \leftarrow \min_{e \in P} C_e$

g flow with $g_e = X$ for $e \in P$

$f \leftarrow f + g$

a real life problem:

A tornado is approaching the city, and we need to evacuate the people quickly. There are several roads outgoing from the city to the nearest cities and other roads going further. The goal is to evacuate everybody from the city to the capital, as it is the only other city which is able to accomodate that many newcomers. We need to evacuate everybody as fast as possible, and your task is to find out what is the maximum number of people that can be evacuated each hour given the capacities of all the roads.



Solution:

import queue

class Edge:

    def __init__(self, u, v, capacity):
        self.u = u
        self.v = v

```python
        self.capacity = capacity
        self.flow = 0
class Flow_Graph:
    def __init__(self, n):
        self.edges = []
        self.graph = [[] for _ in range(n)]

    def add_edge(self, from_, to, capacity):
        forward_edge = Edge(from_, to, capacity)
        backward_edge = Edge(to, from_, 0)
        self.graph[from_].append(len(self.edges))
        self.edges.append(forward_edge)
        self.graph[to].append(len(self.edges))
        self.edges.append(backward_edge)

    def size(self):
        return len(self.graph)

    def get_ids(self, from_):
        return self.graph[from_]

    def get_edge(self, id):
        return self.edges[id]

    def add_flow(self, id, flow):
```

```
        self.edges[id].flow += flow
        self.edges[id ^ 1].flow -= flow
        self.edges[id].capacity -= flow
        self.edges[id ^ 1].capacity += flow


def read_data():
    vertex_count, edge_count = map(int, input().split())
    graph = Flow_Graph(vertex_count)
    for _ in range(edge_count):
        u, v, capacity = map(int, input().split())
        graph.add_edge(u - 1, v - 1, capacity)
    return graph


def max_flow(graph, from_, to):
    flow = 0
    while True:
        has_Path, path, X = bfs(graph, from_, to)
        if not has_Path:
            return flow
        for id in path:
            graph.add_flow(id, X)
        flow += X
    return flow
```

```
def bfs(graph, from_, to):
    X = float('inf')
    has_Path = False
    n = graph.size()
    dist = [float('inf')]*n
    path = []
    parent = [(None, None)]*n
    q = queue.Queue()
    dist[from_] = 0
    q.put(from_)
    while not q.empty():
        currFromNode = q.get()
        for id in graph.get_ids(currFromNode):
            currEdge = graph.get_edge(id)
            if float('inf') == dist[currEdge.v] and currEdge.capacity > 0:
                dist[currEdge.v] = dist[currFromNode] + 1
                parent[currEdge.v] = (currFromNode, id)
                q.put(currEdge.v)
                if currEdge.v == to:
                    while True:
                        path.insert(0, id)
                        currX = graph.get_edge(id).capacity
                        X = min(currX, X)
```

```
                if currFromNode == from_:
                    break
                currFromNode, id = parent[currFromNode]
            has_Path = True
            return has_Path, path, X
    return has_Path, path, X

if __name__ == '__main__':
    graph = read_data()
    print(max_flow(graph, 0, graph.size() - 1))
```

Content and Code:

Vashishth Gajjar (19BCE2286)

Editing and Styling:

Rohan Arora (19BCE2248)

Ideas and Suggestions:

Tejas Jonnadula (19BCE2259)

Saurav Sharma (19BCE2218)