Max Flow Min Cut problem: Ford–Fulkerson algorithm

Corso: Combinatoria

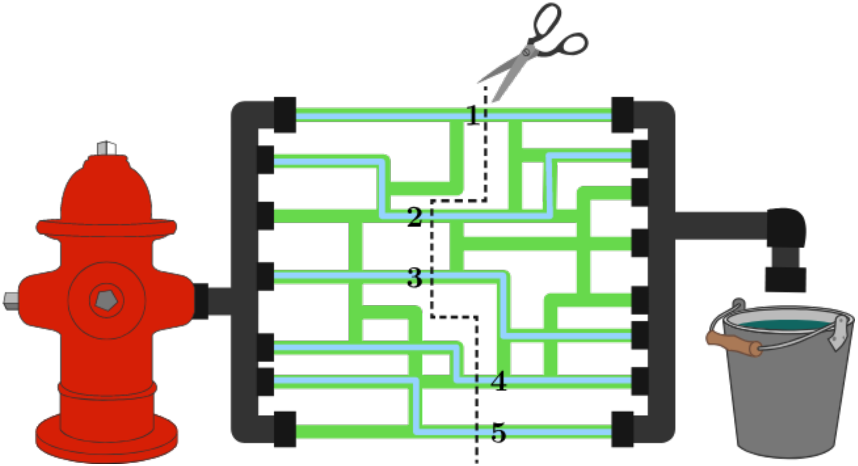
Facoltà di Scienze Matematiche Fisiche e Naturali
Sapienza Università di Roma

Candidato:
Andrea Mantuano

Matricola:
1739874

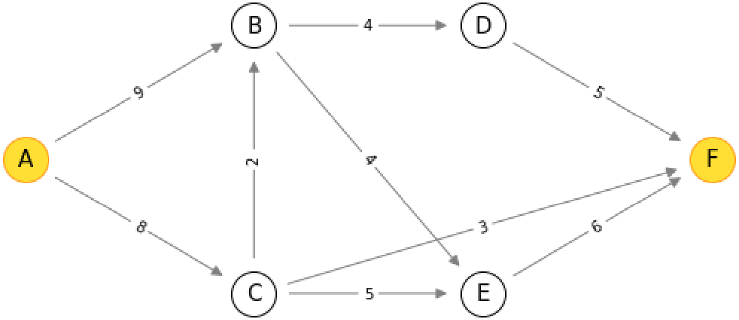
Table of contents

The problem



Definition

A **network** is a directed graph $G = (V, E)$, on whose edges is defined a function called **capacity** that associates to each edge uv a real number $c(uv) > 0$ called capacity of the edge.



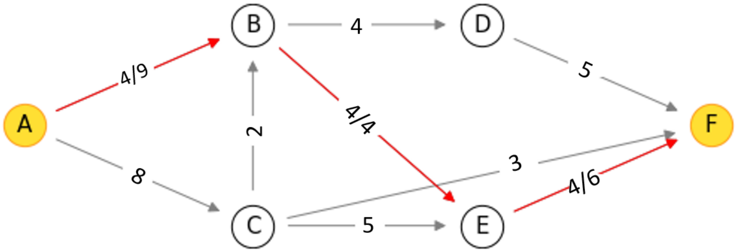
Two distinguished vertices exist in G:

- **Source** (denoted by s): the in-degree of this vertex is 0;
- **Sink** (denoted by t): the out-degree of this vertex is 0.

Definition

Given a network it is possible to define a **flow** running from the source to the sink,that is, it is possible to define a function $f : E \longrightarrow \mathbb{R}^+$ that associates to each edge of the network a non-negative real value such that:

- $f(uv) \leq c(uv)$
- $\sum_u f(uv) = \sum_w f(vw)$ for $v \neq s, t$



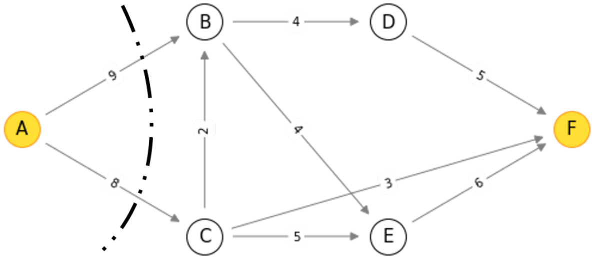
- For simplicity we call **value of flow** the amount of flow leaving the source and the amount entering the sink. We note that this quantity can never exceed the sum of the capacities of the outgoing edges from s:

$$v(f) \leq \sum_{u} c(su)$$

Definition

A **cut** in a network is a partition of vertices into two subsets X and $\bar{X}$, such that $s \in X$ and $t \in \bar{X}$. In addition we define:

- **cut edges**: the edges u, v such that $u \in X$ and $v \in \bar{X}$
- **cut capacity**: is the sum of the capacities of the cut edges



Definition

Given a flow network G, the **residual network** G_r is a network such that for every edge in G with flow less than that edge's capacity, there will be a corresponding edge in G_r that has a residual capacity of the original edge's capacity minus the flow on that edge.

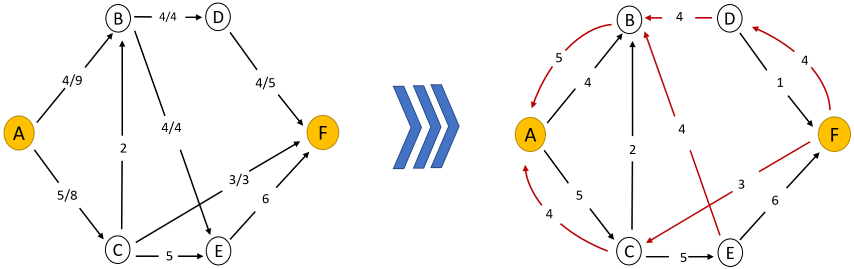


Table of contents

Lemma

If the value of a flow f is equal to the capacity c of a cut, then such a flow f is maximum and the capacity c of the cut is minimum

- If there exist a flow f' such that $v(f') > v(f) \Rightarrow v(f') > c$ which is absurd
- so f is maximum.
- If there exist a cut c' such that $c' < c \Rightarrow c' < v(f)$
- so the flow f would be greater than the capacity of the cut $c' \Rightarrow c$ is minimal.

$\blacksquare$

Theorem (Ford-Fulkerson)

On a network, the maximum value that a flow can assume is equal to the minimum value of the capacities of the possible cuts

- we'll show that, starting from a flow f of maximum value, it is possible to construct a cut $(X, \bar{X})$ of capacity c equal to the value of f.
- let $s \in X$ and add nodes to X with the following rule

$$u \in X \text{ and } \{u, v\} \in E(G) \Rightarrow v \in X \iff u \longrightarrow v \text{ is not saturated}$$

or

$$u \in X \text{ and } \{u, v\} \in E(G) \Rightarrow v \in X \iff u \longleftarrow v \text{ the flow is not null}$$

- if $t \in X \Rightarrow$ there exists a path from s to t

$$s = u_1, \ldots, u_n = t$$

such that:

$$\begin{array}{ll} u_i \longrightarrow u_{i+1} & \text{is not satured} \\ u_i \longleftarrow u_{i+1} & \text{the flow is not null} \end{array}$$

- this would be an augmenting path, but this is impossible $\Rightarrow t \notin X$
- so we have a partition of the nodes into X and $\bar{X}$ and thus a cut $(X, \bar{X})$

We show now that the value of the flow f is equal to the capacity c of the cut we just found:

- if $u \in X$ and $v \in \bar{X}$, with u and v adjacent :
  - $f(\{u \to v\}) = c(\{u \to v\})$
  - $f(\{v \leftarrow u\}) = 0$
- then:

$$\begin{aligned} v(f) &= \sum_{u \in X, v \in \bar{X}} f(uv) - \sum_{u \in X, v \in \bar{X}} f(vu) \\ &= \sum_{u \in X, v \in \bar{X}} c(uv) - 0 \\ &= c(X, \bar{X}) \end{aligned}$$

■

Corollary

A flow f is maximum if and only if there are no augmenting paths

- $\Rightarrow$: if f is maximum, it cannot be augmented, and therefore there are no augmenting paths.
- $\Leftarrow$: If no augmenting paths exist $\Rightarrow t \notin X$. So X and $\bar{X}$ are a cut for s, t and $c = v(f)$. Therefor f is maximum.

■

Observation

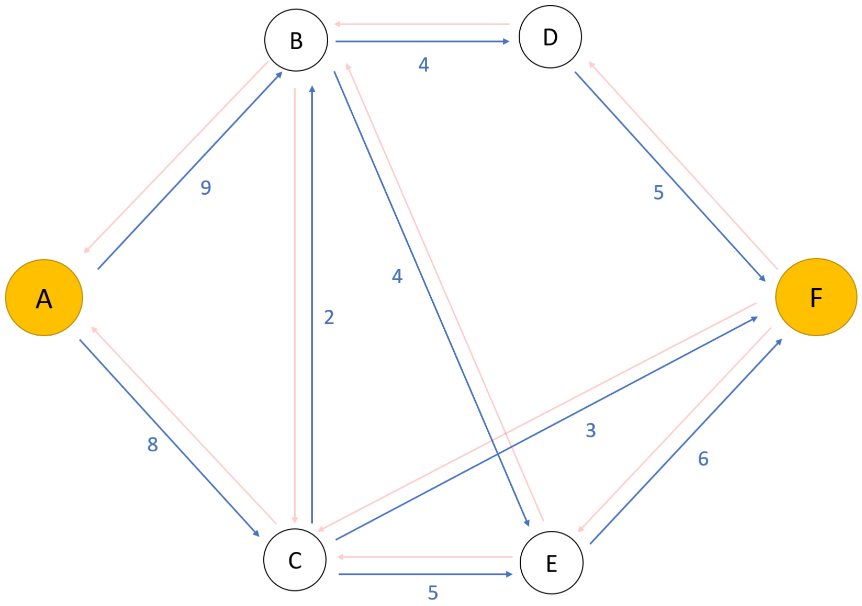
Issues concerning bipartite graphs can be stated in terms of flows by considering networks of the following type.
Let G be a bipartite graph, $V = V_{12}$ and let G' be the network obtained by adding to G the nodes s, t and the edges oriented from s to V_1 and from t to V_2.

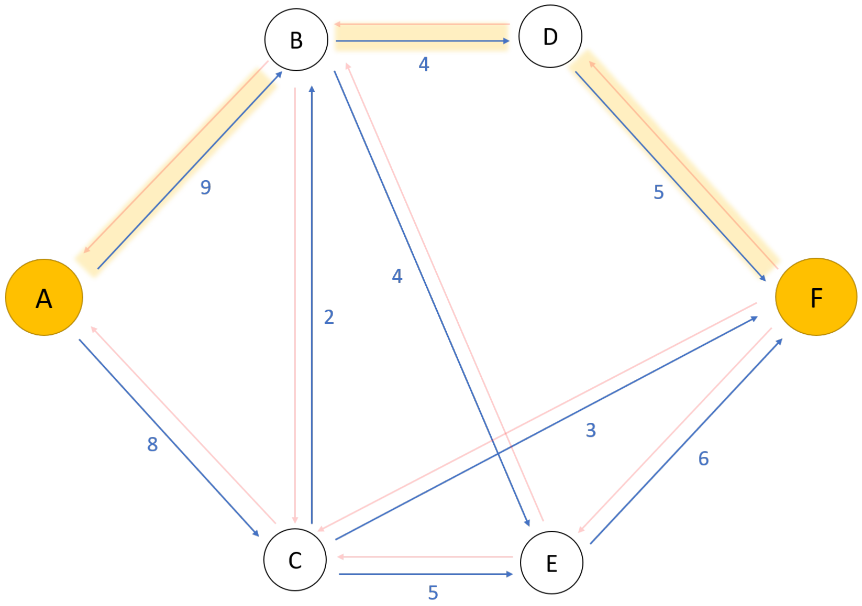
Then it is possible to show that:

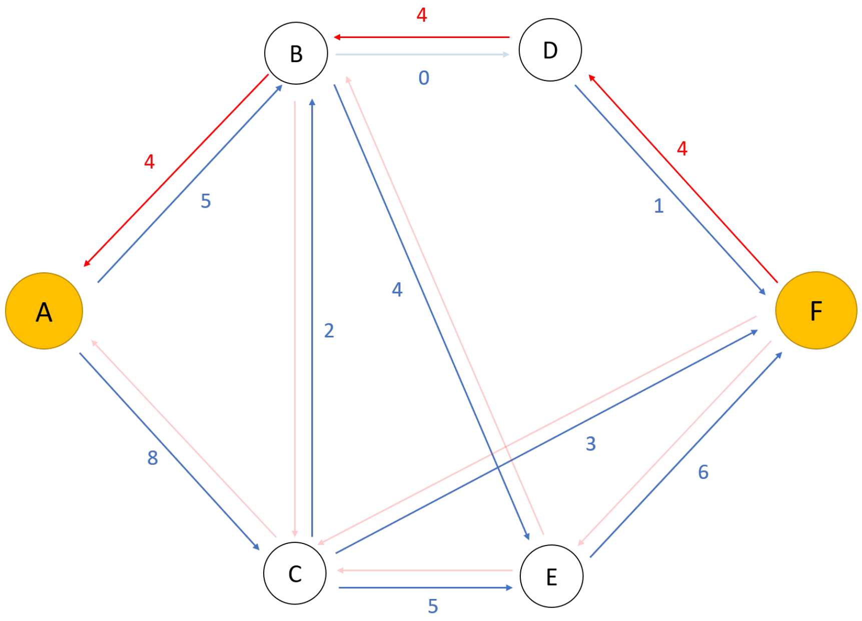
- König-Egerv*á*ry theorem
- Hall theorem

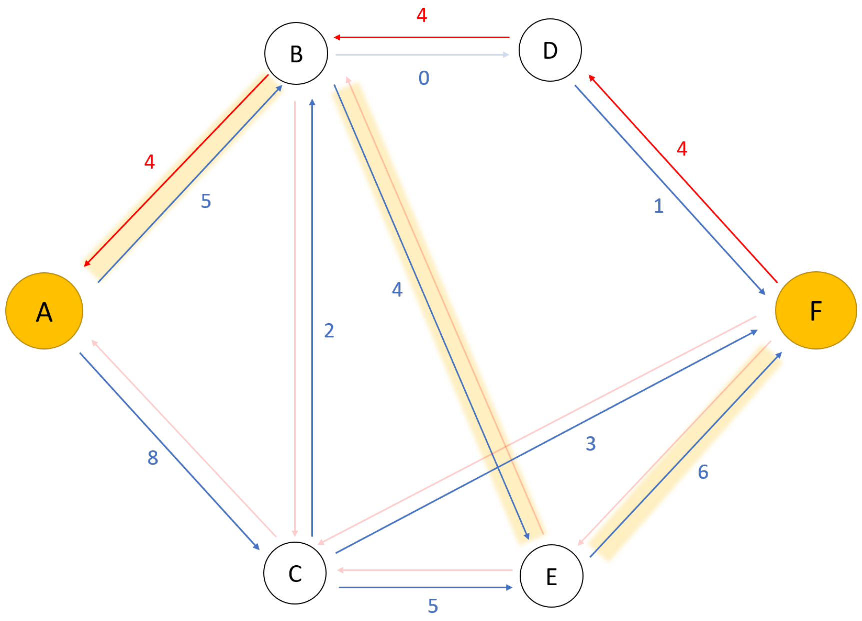
in the form for bipartite graphs, follow from the max flow-min cut theorem.

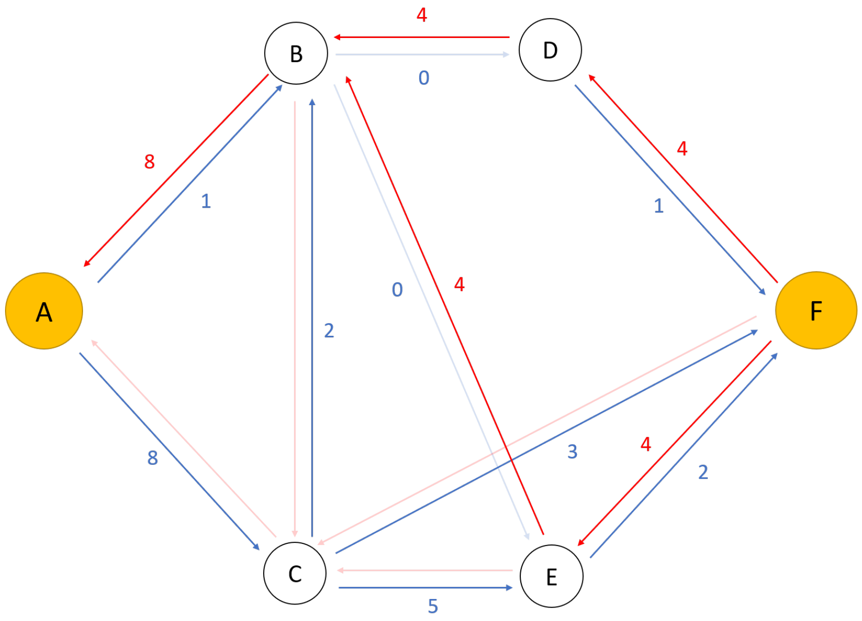
Table of contents


A
B
C
D
E
F
9
8
2
4
4
5
3
6
5

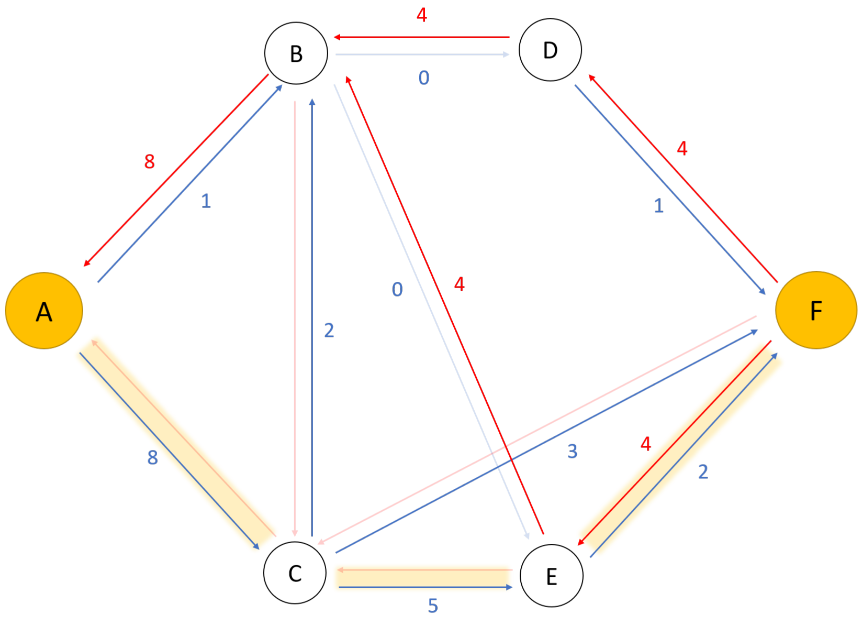

B
D
A
F
C
E
4
9
5
4
2
8
3
6
5


A
B
C
D
E
F
4
0
4
5
4
1
2
4
8
3
6
5

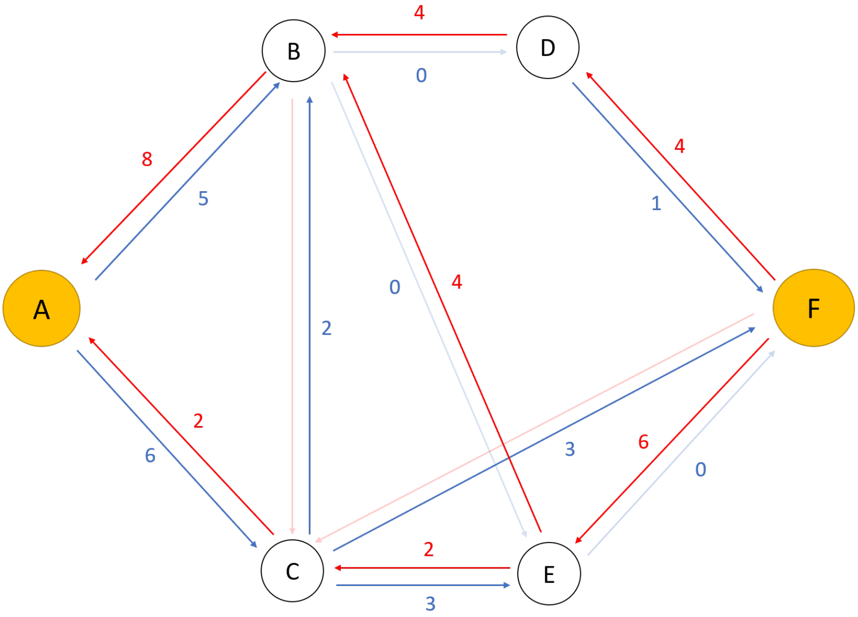

4
B
D
0
4
5
4
1
A
4
F
2
8
3
6
C
E
5

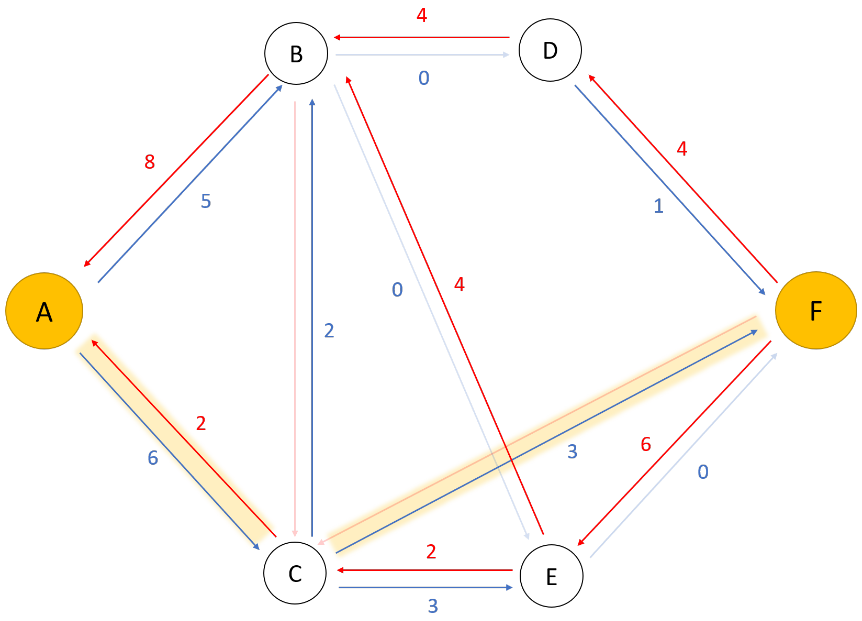


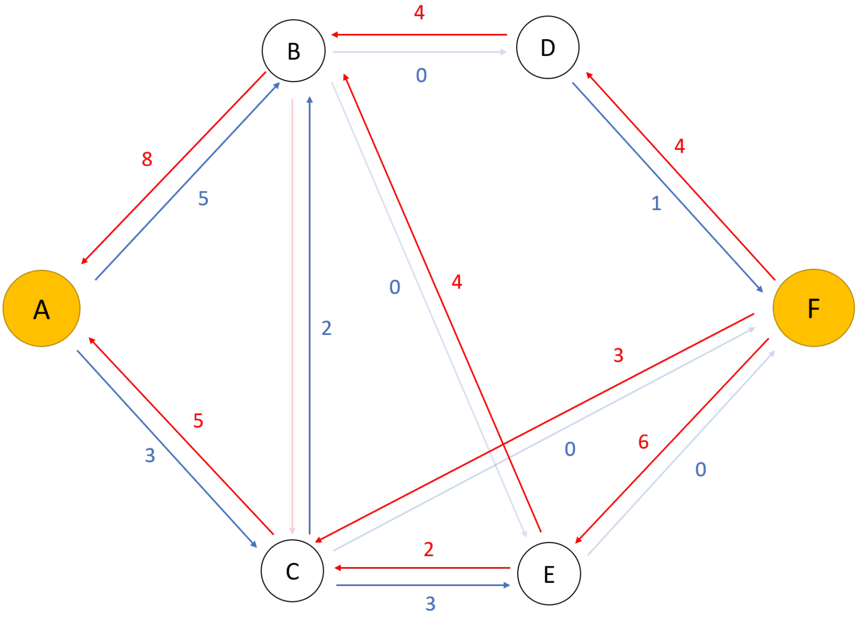
A
B
C
D
E
F
4
0
8
1
2
0
4
4
1
8
3
4
2
5

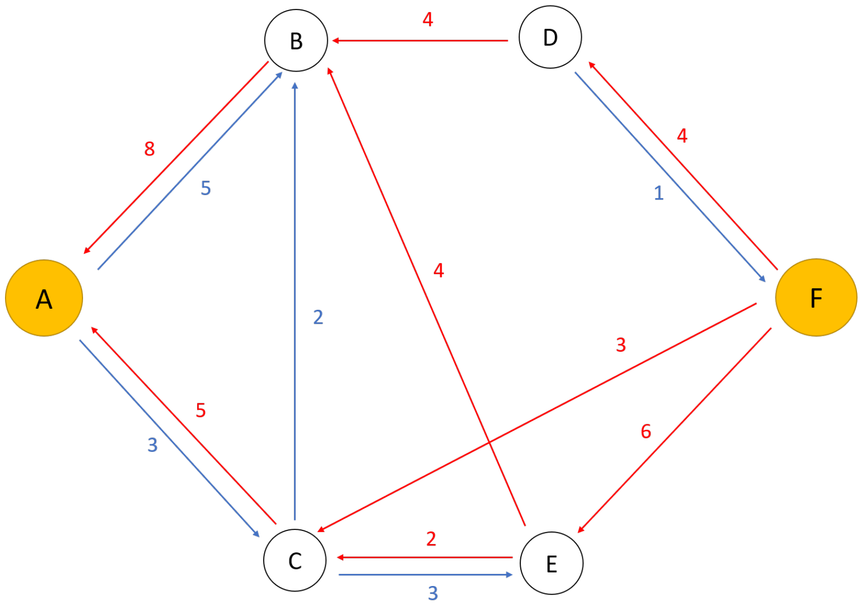


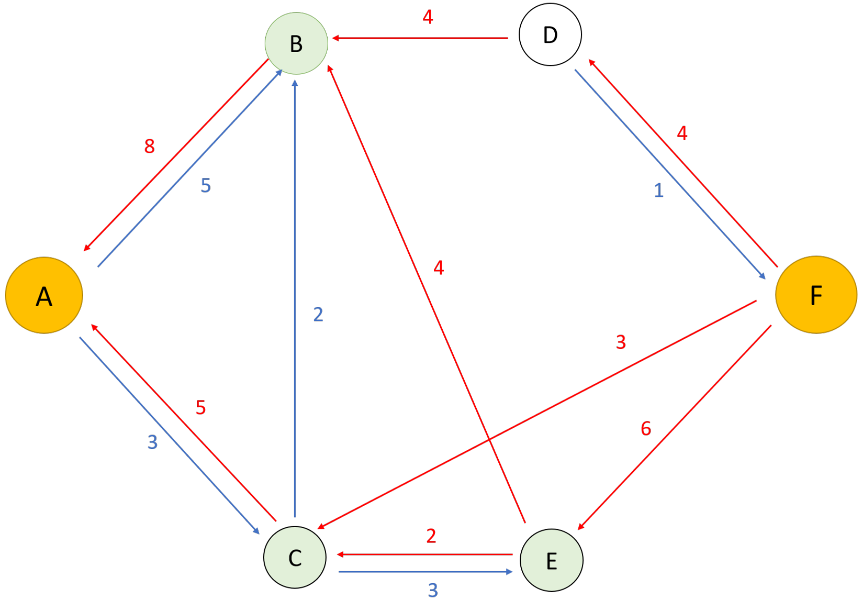
A
B
C
D
E
F
4
0
8
1
4
1
0
4
2
8
3
4
2
5

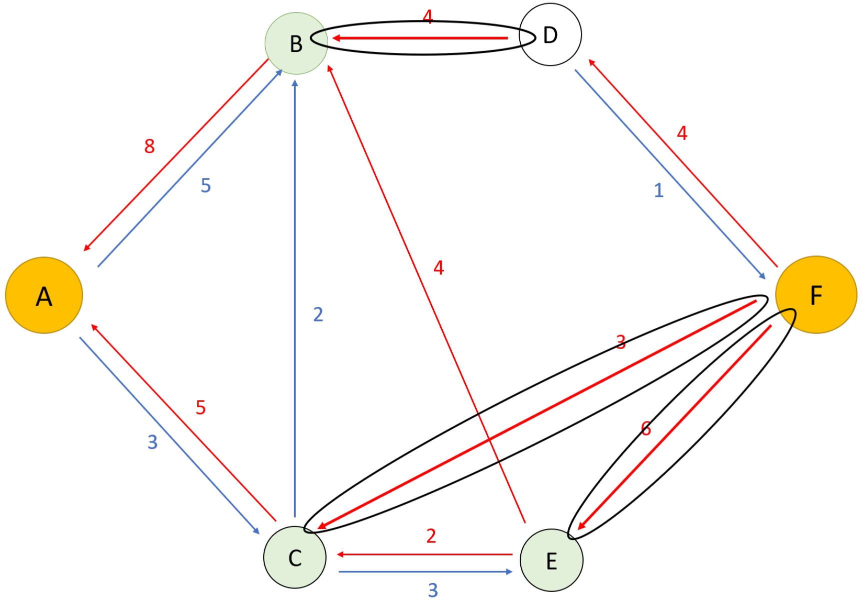

A
B
C
D
E
F
4
0
8
5
4
1
2
0
4
2
6
3
6
0
2
3

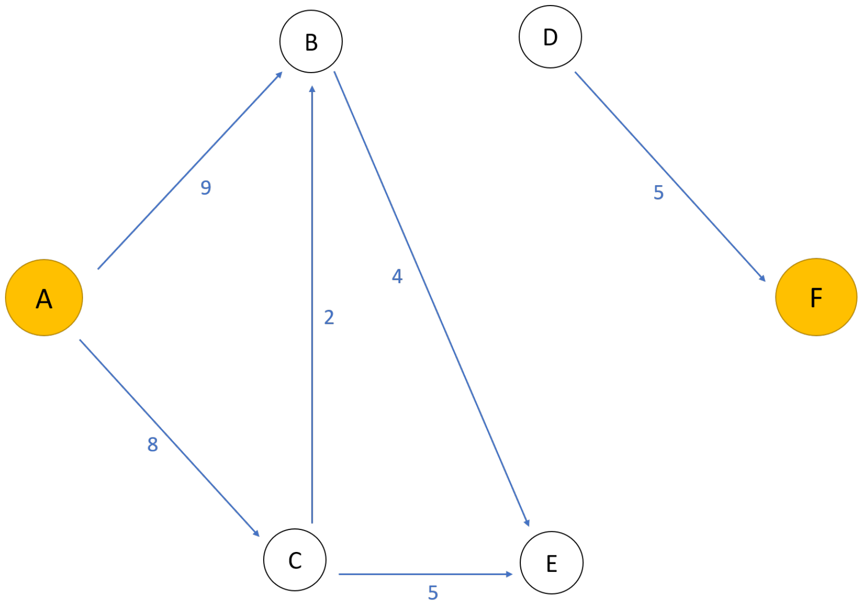

4
0
B
D
8
5
4
1
A
F
0
4
2
2
6
3
6
0
2
C
E
3


A
B
C
D
E
F
8
5
3
5
4
0
2
0
4
1
4
3
0
6
0
2
3


B
4
D
8
5
4
1
4
A
2
F
3
5
6
3
2
C
E
3


B
D
A
F
C
E
4
8
5
4
1
4
2
3
5
3
6
3
2
3


A
B
C
D
E
F
8
5
4
4
1
2
4
3
5
3
6
2
3


A
B
C
D
E
F
9
8
2
4
5
5

Breadth-first search

```python
def BFS(graph, s, t, path):

    visited = set()
    queue = [s]
    visited.add(s)
    if s not in path: path[s] = ()
    while len(queue) > 0:
        source = queue.pop(0)
        neighbors = getNeighborsMinWeight(source, graph)
        for target in neighbors:
            if target not in visited and neighbors[target] > 0:
                queue.append(target)
                visited.add(target)
                path[target] = (source, neighbors[target])
    if t in visited:
        return True, path

    return False, path
```

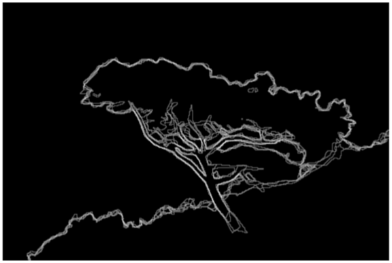
Ford-Fulkerson algorithm

```python
def Ford_Fulkerson(graph, source, target):

    # initialize the residual graph creating the backword edges
    res = createResidualG(graph)
    max_flow = 0
    path = dict()

    # update the residual graph untill exists a path between soruce
     ↪  and target in the residual graph
    while BFS(res, source, target, path)[0]:
        #find the min weight of edges in the current path
        bottleneck = getBottleneck(path, source, target)
        #update the max-flow
        max_flow += bottleneck
        updateResGraph(res, path, bottleneck, source, target)

    print("Max Flow is:", max_flow)
    #find all nodes reachable from source in the final residual graph
    reachable = reachFromS(res, source)
    edges = set()
    capacity = 0

    for v in reachable:
        neighbor = graph[v].get_out_relation
        for target in neighbor:
            relX = res[v].get_out_relation[target] # take the relation in the residual graph
            rely = graph[v].get_out_relation[target] # take the relation in the original graph
            # if we reach an extreme node in the residual graph and its out edge weight is 0 this has to been cut
            if relX.weight == 0 and relX.target not in reachable and (v, relX.target) not in edges:
                edges.add((v, relX.target))
                capacity += rely.weight

    print("Capacity is:", capacity)
    return edges
```

Table of contents

Image Segmentation

Input is in image.
Partition image in background and foreground.



Partition V in two disjoint sets F and B in order to maximize:

$$q(F,B) = \sum_{i \in F} f_i + \sum_{iinB} b_i - \sum_{\substack{\{i,j\} \in E(G) \\ |F \cap \{i,j\}| = 1}} p_{ij}$$

Thanks for your attention

You can also find all the codes at the following link:
`https://github.com/Mantuano-A/Min-Cut-Max-Flow-problem`