Network Flow Graphs
Single Source Single Sink (S^4) **flow** algorithm
Multiple Source Multiple Sink $(MS)^2$ **flow** algorithm

Examples
Water pipes can be used for flow of water.
Gas lines can be used transfer gasoline between gas stations.
Transmission lines can be used to transmit messages.
Transport lines can be used to transport shipments.
Network power flow, fault tolerance.

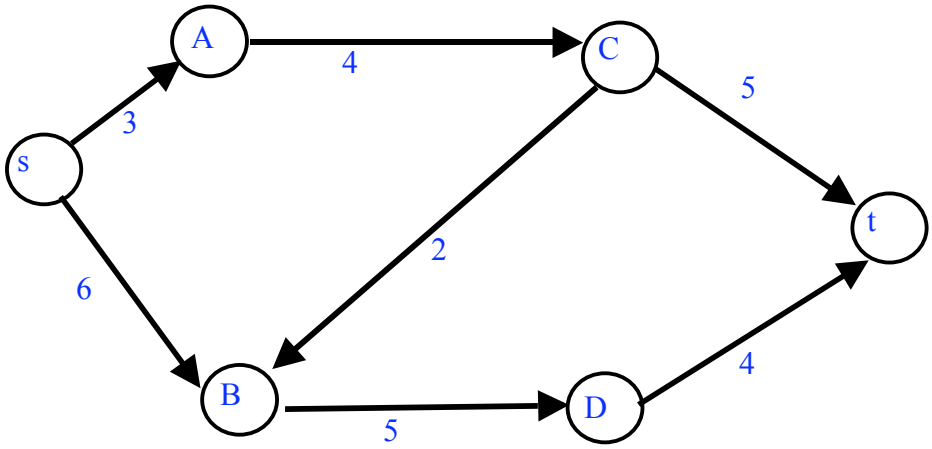
Definition.
Let G(V,E) be a **directed** graph. It has **two** special vertices s, t $\in$ V such that s is the **source** (start) with zero indegree, t is the **target**(terminal, destination, sink) with zero outdegree (one source, one sink).

Paths are paths in the undirected form of the graph. That is, an edge (u, v) is on the path if (u,v) is in E or (v,u) is in E.
The **length** of a path is the number of edges on the path. The number of **hops** to go from one vertex to another vertex.
The **capacity** of an edge e $\in$ E is denoted by c(e) $\geq$0.
The **flow** through an edge e = (u,v) $\in$E is denoted by f(e)$\geq$0 .
The **flow** through an edge e = (u,v), (v,u) $\in$E is denoted by f(e)$\leq$0 .
A **path** from s to t consists of edges (in undirected form)from E such that they form a sequence of edges connecting s to t.
A **Cut** set of edge when deleted there is no path from s to t
A **MinCut** is a cut with minimum capacity.
The network **flow function f** is a function that satisfies three properties
(1) For each e $\in$**E,** 0$\leq$f(e)$\leq$c(e), (**feasibility condition**)
(2) for each e = **(u,v)** $\in$**E**, f(u,v) = - f(v,u)
(3) for each vertex v $\in$ V ~{s,t}, $\sum_{u \in V, (u,v) \in E} f(u,v) = \sum_{w \in V, (v,w) \in E} f(v,w)$ (**conservation condition**)

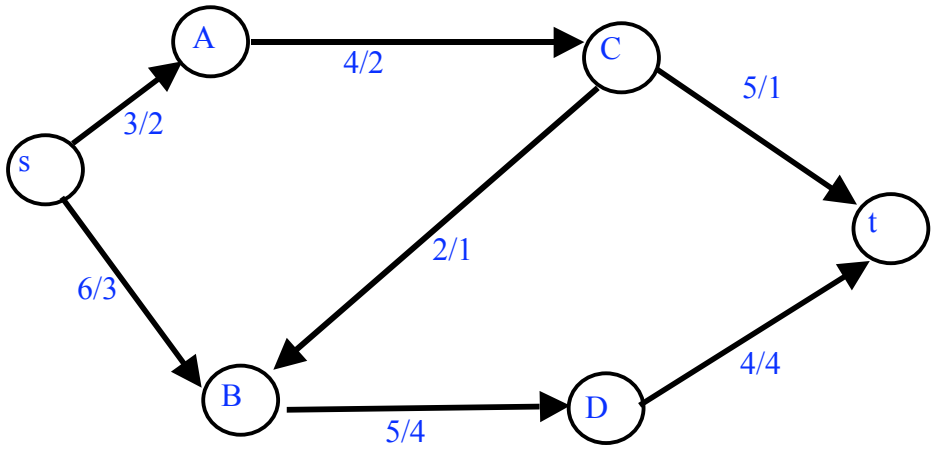
Properties (1) and (3) imply that $\sum_{u \in V} f(s,u) = \sum_{u \in V,} f(u,t)$ for edges in E,

Properties (1) and (2) imply that total flow at a vertex is zero:
$\sum_{u \in V} f(u,v) = 0$

Example of Capacities. The arrows indicate the direction of flow. The numbers on the edges represent the capacity of the edge for flow.

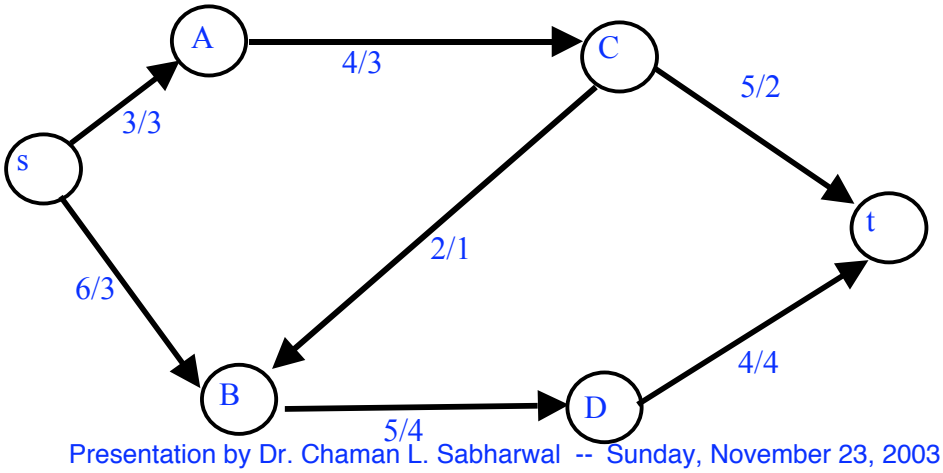


Notation capacities and flow.
Let us denote the edges with pairs (capacity, flow), in short, **capacity/flow: c/f.**
Example of Conservation of Flow. The following figure shows the **conservation of flow**.



Here the flow s-->3+2=5= 1+4-->t.
Maximum flow is a largest possible value of flow.

Problem: Maximize the amount of flow from source to sink, $\sum_{u \in V,(u,t) \in E} f(u,t)$.
The next figure shows that it is possible to increase the flow to 6.



How do we know we have reached a maximum or not?
Notation for Augmented paths, Min of slacks in the (edges hence) nodes on the path!!

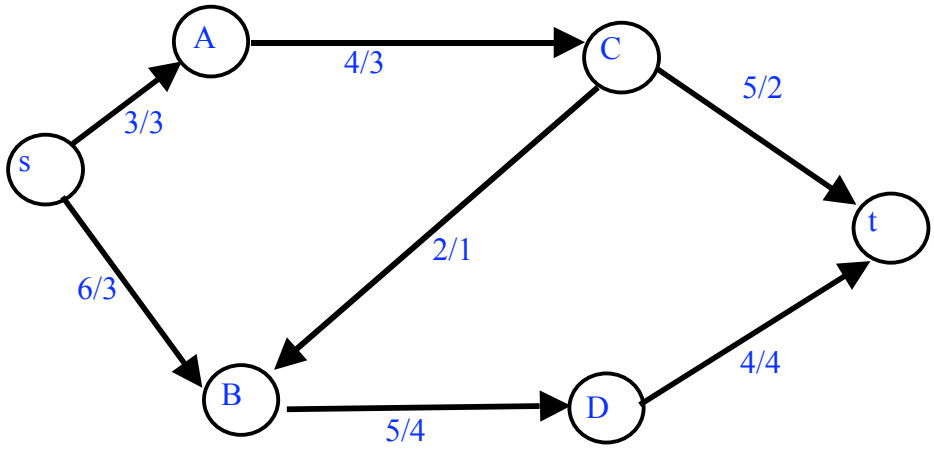
Augmented path ; **Residual flow—slack in each edge.**

A **path** for the **flow** is a path (in **undirected** form of graph) from s to t. An **augmented** path is a path that has **NO edge which is used to its full capacity**. *The path consists of edges from E without regard to the directions in E.*
An edge e is on an augmented path if
(1) if f(e) < c(e) for e □E **(forward edge on the path)** . This means there is possibility to **increase** the flow on this edge. The **slack of forward** edge is given by c(e)- f(e).
(2) if f(e) > 0 for e=(u,v) □E ((v,u) **backward edge on the path)** . This means there is possibility to **decrease** the flow on this edge. □
Rule: Whenever there is an augmented path, it is possible to increase the value of flow along this path. See the example.

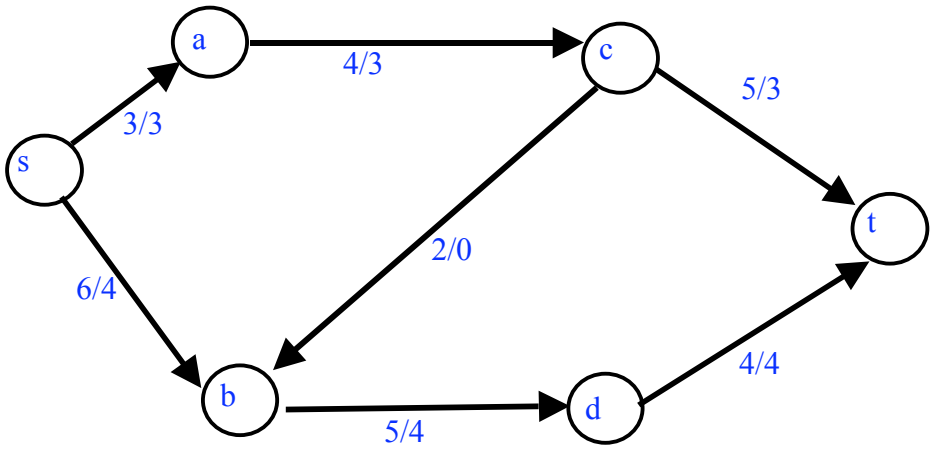
Phillosophy
1. initialize flows to zero
1. while augmenting paths p exist
    a. augment flow along p
    a. return p.

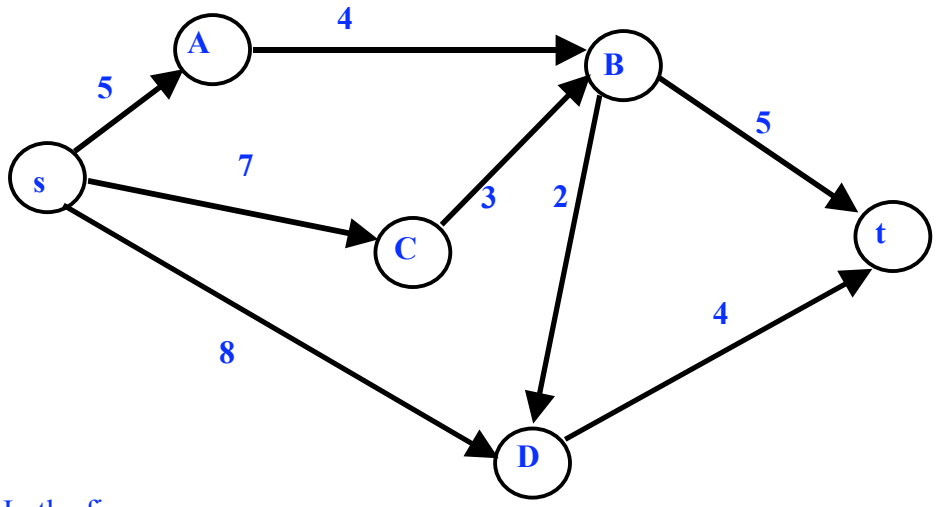
In this figure, it is clear that flow through the edges **cannot exceed 9** at t.

Can we do better than this? How do we do it?
But **what is the maximum flow through the network?**



The edge cb is not helpful, decrease the flow to 0. Next figure shows that total flow can be increased to 7.





Example. In the figure,
the arrows indicate the direction of flow,
the numbers show the capacities of the flow through edges.

In this example again, it is clear that flow through the edges **cannot exceed 9 at t**.
Note. Even though maximum input to C is 7, but it cannot transfer more than 3.
Question? What is the maximum flow through the network?

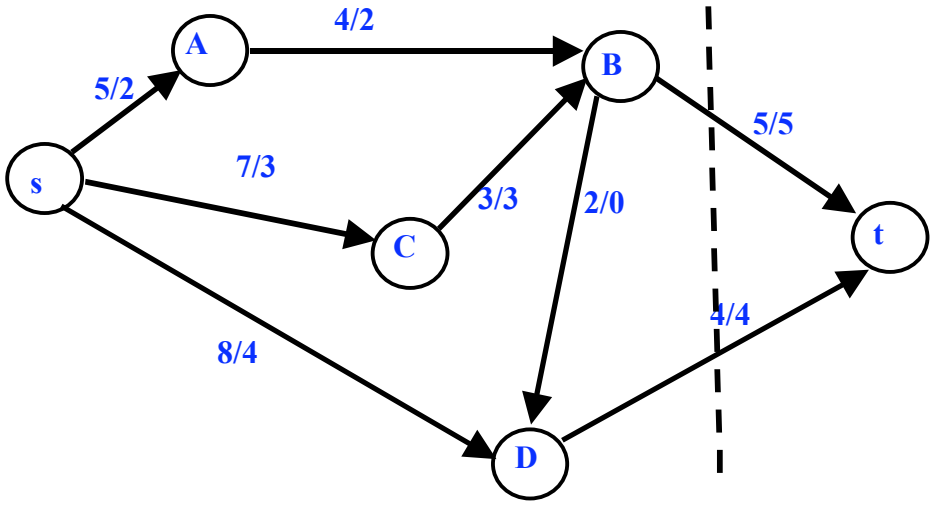
Remember: The flow paths are not unique, but maximum flow value is unique.Examples -- Conservation property

The flow is not maximum. In the following example flow is maximum but the paths are not unique.

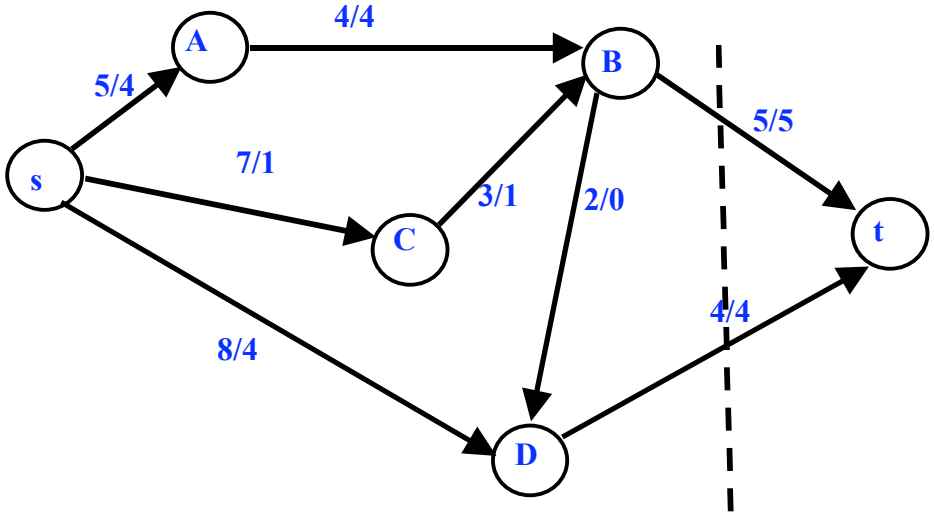
A **cut** is the set of edges such that if these edges are removed, there is no flow (path) from s to t.
Across a cut, edge is **forward** if flow is from S to T
Across a cut, edge is **backward** if flow is from T to S

ST=V, S∩T=∅

In the following example cut is not mincut, flow is max. capacity = 9, flow=9

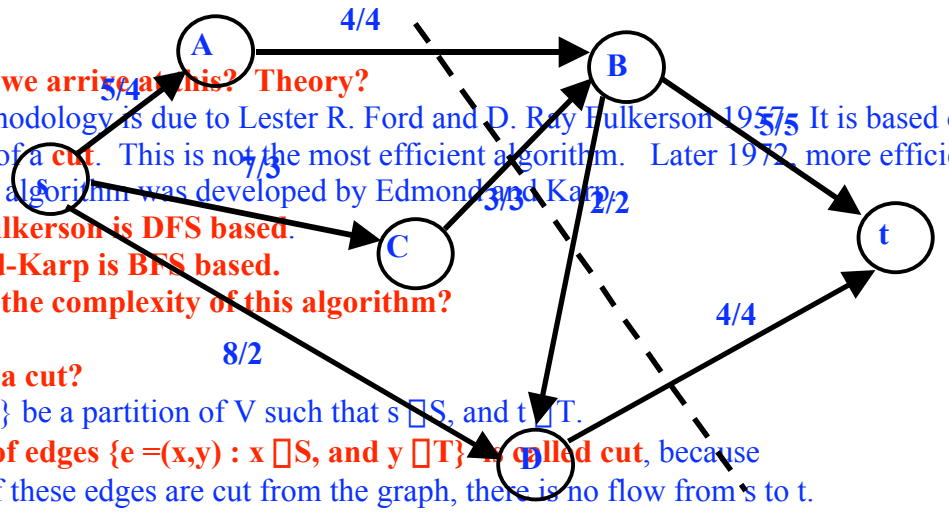


or



Or

In the following example cut is not mincut, flow is max. capacity = 11, flow=9



How do we arrive at this? Theory?
The methodology is due to Lester R. Ford and D. Ray Fulkerson 1957. It is based on the concept of a **cut**. This is not the most efficient algorithm. Later 1972, more efficient and complex algorithm was developed by Edmond and Karp.
Ford-Fulkerson is DFS based.
Edmund-Karp is BFS based.
What is the complexity of this algorithm?

What is a cut?
Let {S,T} be a partition of V such that s ∈ S, and t ∈ T.
The **set of edges {e =(x,y) : x ∈ S, and y ∈ T} is called cut**, because

1. if these edges are cut from the graph, there is no flow from s to t.
1. every such edges is used to its full capacity.

In short, such an ordered **pair (S, T) is also called a cut**.
The flow through a cut is $\sum_{u \in S, v \in T} f(u,v)$.
The capacity of a cut is $\sum_{u \in S, v \in T, (u,v) \in E} c(u,v)$ = **capacity(S,T). It includes only forward edges from S to T.**

A minimum value **cut** is called a **min-cut**. A min-cut provides for max flow. This is called **max-flow min-cut**.

Note. Whenever there is an augmented path, the flow is not maximum. We are looking for cut so that there is no augmented path?

Theorem The value of any flow is less than or equal to the capacity of any cut.

Note. If flow admits no augmented path, the cut is min-cut.

The Integral-Flow theorem
If all the edges in the network have integer capacities, then there is maximum flow with integer value.
Note. The capacities are integers, the max flow is finite. The Flow values are initialized to zero. The augmented paths are eliminated by adding slack values to the path flow. The slack flow values are integers. When no augmented path is left, the maximum flow is the capacity of the min cut which is an integer.
Algorithm
Method:
initialize the flow to zero along every edge, that is each edge is labeled as c/0
repeat

1. find an augmented path p starting at s and ending at t. – use stack
1. calculate the minimum of the slacks along this path p
   m = min{slack(e); e on the path p};m>0
1. reset or update the flow along each edge on the path p as follows
   a. **increase** the flow along forward direction edge

f(e) +=m

a. **decrease** the flow along the backward direction edge

until no more augmented paths e∈E

f(e) -=m

or

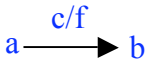
f(~e) +=m

1. Now collect all the augmented paths starting at s (and not reaching t). S is the set of all vertices on these paths. T is the set of remaining vertices.
1. find the cut, verify that **maximum** flow =capacity of this cut, called the **Min-Cut**

Note. flow across the edges will depend on the order the paths is traversed, **flow is not unique**.
The **value of flow along the min-cut is unique**.
The Ford-Fulkerson methodology **does not tell how to detect augmented paths**.
Therefore the **complexity** of this algorithm **does not depend on the number** of edges or vertices.

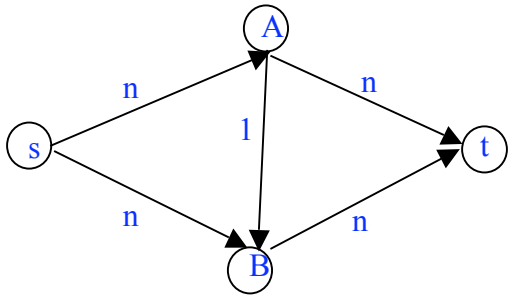
Does the complexity depend on the number of edges?
Does the complexity depend on the number of paths?
Does the complexity depend on the number of augmented paths?
How do we determine the augmented paths?

A worst case Example
Notation.
s→A means edge from s to A
s→ (c/f) →A denotes the capacity and flow through the edge from s to A.



Example there are four vertices and four possible paths. Think of s→A→t, s=>B=>t we get the answer by traversing two paths. The cut is S ={s}, T = {A,B,t}. The flow value is 2n.

If we follow s→A→B→t, then s→B→A→t, the flow is 1, but it is repeated 2n times to arrive at min-cut.

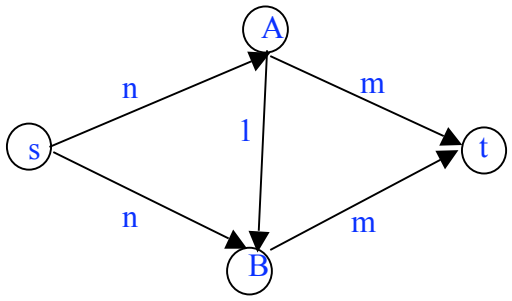


Initial flow is s→ (n/0) →A, s→ (n/0) → B, A→ (1/0) →B, A→ (n/0) →t, B→ (n/0) →t
Consider the path s→A→B→t,
the slacks in the nodes are s(∞)→A(n) →B(1) →t(1)

the augmented path flow is s→ (n/1) →A→ (1/1) →B→ (n/1) →t
Now follow the path s→B→A→t
the slacks in the nodes are s(∞)→B(n) →A(1) →t(1)
the augmented path flow is s→ (n/1) →B→ (1/0) →A→ (n/1) →t
After eliminating two augment paths we have the flow
s→ (n/1) →A, s→ (n/1) →B, A→ (1/0) →B, A→ (n/1) →t, B→ (n/1) →t
By repeating eliminating two augment paths n times we will have the flow
s→ (n/n) →A, s→ (n/n) →B, A→ (1/0) →B, A→ (n/n) →t, B→ (n/n) →t
The min cut becomes {s,A} and {B,t}.

For any n, there will be **2*n** augmented paths before reaching a min-cut.

Another version. Ford-Fulkerson is **DFS** based. The number of augmented paths is 2*min(m,n) + 1



Complexity of the algorithm
The complexity of the algorithm does not depend on the number of edges and vertices.
O(|c| |E|) capacity of an edge is increases one step at a time, in the worst case O(|E|) edges end up at a target.

We have used the DFS approach to detect augmented paths. This is not most efficient. Later 1972, more efficient and complex algorithm was developed by Edmond and Karp.
Ford-Fulkerson is **DFS** based.
Edmund-Karp is **BFS** based.

Note. Algorithms work evn for cyclic graphs. We use only simple paths.
EdmondKarp algorithm
Reapeat

find augmented path of min length.
augment the path.
Until not more augmented paths.

Each node n is labeled as (parent, slack(n)) slack(n) maximum additional flow that can be transferred from the parent to n.

Edmond-Karp BFS algorithm
Labeling algorithm

1. initialize the flow to zero with every edge, that is each edge is labeled as Capacity/0
2. initialize the source s as (nullParent, ∞ amountThatCanBeShipped), each vertex will be labeled as pair (parent, maxAdditionalThatCanBeShipped))

Repeat -- O(E) since each edge can be an augmented path, each iteration remove on augmented edge from the path.

{Find an augmented path of min length}
x=s
insert x in Q
Repeat --O(V)
Remove x from Q
For each edge (x,y) -- O(V)
if e=(x,y) is forward, (x,y) ∈E, **y is not labeled**, f(e) < c(e), label(x) = (parent, slack_x),
then label(y) = (x, min(sack_x, c(e)-f(e)))
if e=(x,y) is backward, (y,x) ∈E, **y is not labeled**, f(e) >0, label(x) = (parent, slack_x),
then label(y) = (x, min(slack_x, f(e)))

a) if the sink is **labeled (p,v). Reset the flows along this path** and **unlabel the vertices**. Note v>0 and is ≤ the slack in the **edge(p,t)** , the flow can be increased along (p,t) by v min of the slacks on all edges along the path? backtrack to , P, parent of p. Now (P,p) is forward edge or back edge, for forward edge, increase the flow by v, for backedge, decrease the flow by v. Along the entire path, unlabel all the vertices and restart with s.
b) else insert y in Q

until sink is reached or not possible to label further—QueueIsEmpty
until not possible to label further--QueueIsEmpty

Complexity of EdmondKarp algorithm
$O(V^2E)$