Maximum
Flow

Programming / Problem Solving
Session - 1

[AnilKishore]
IIIT – H, India

I learnt from…

- TopCoder Tutorial
  http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=maxFlow

- Solving a few problems on SPOJ, CodeChef, TopCoder

- Being active in TopCoder Forums
  (you should too !)

Hope you know…

- What is a **Graph** #DiscreteMath
- How to traverse a graph using
  - Depth First Search (**DFS**)
  - Breadth First Search (**BFS**)

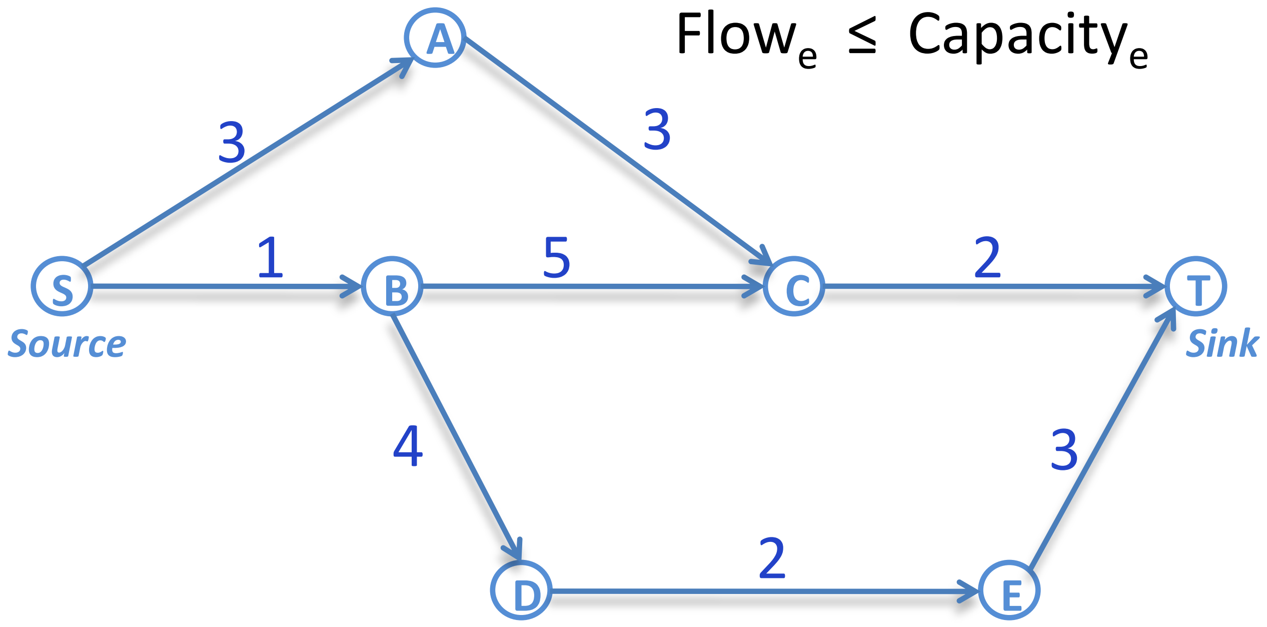
Thats it, you have got all that is needed to learn MaxFlow !

Some terms…

- A *network* of water pipes and junctions

  - Maximum amount of water a pipe allows is its **Capacity**
  - Actual amount of water in a particular instance is its **flow**

$\text{Flow}_e \leq \text{Capacity}_e$



Augment : to make or become greater in number, amount, strength, etc.; increase

so what is MaxFlow ?

- *Given a network of water pipes (some fat, some slim etc.,) find the maximum amount of water that can flow from S to T*

- Source produces
- Sink consumes
- Others neither produce nor consume

Source :

in Flow = 0

Sink :

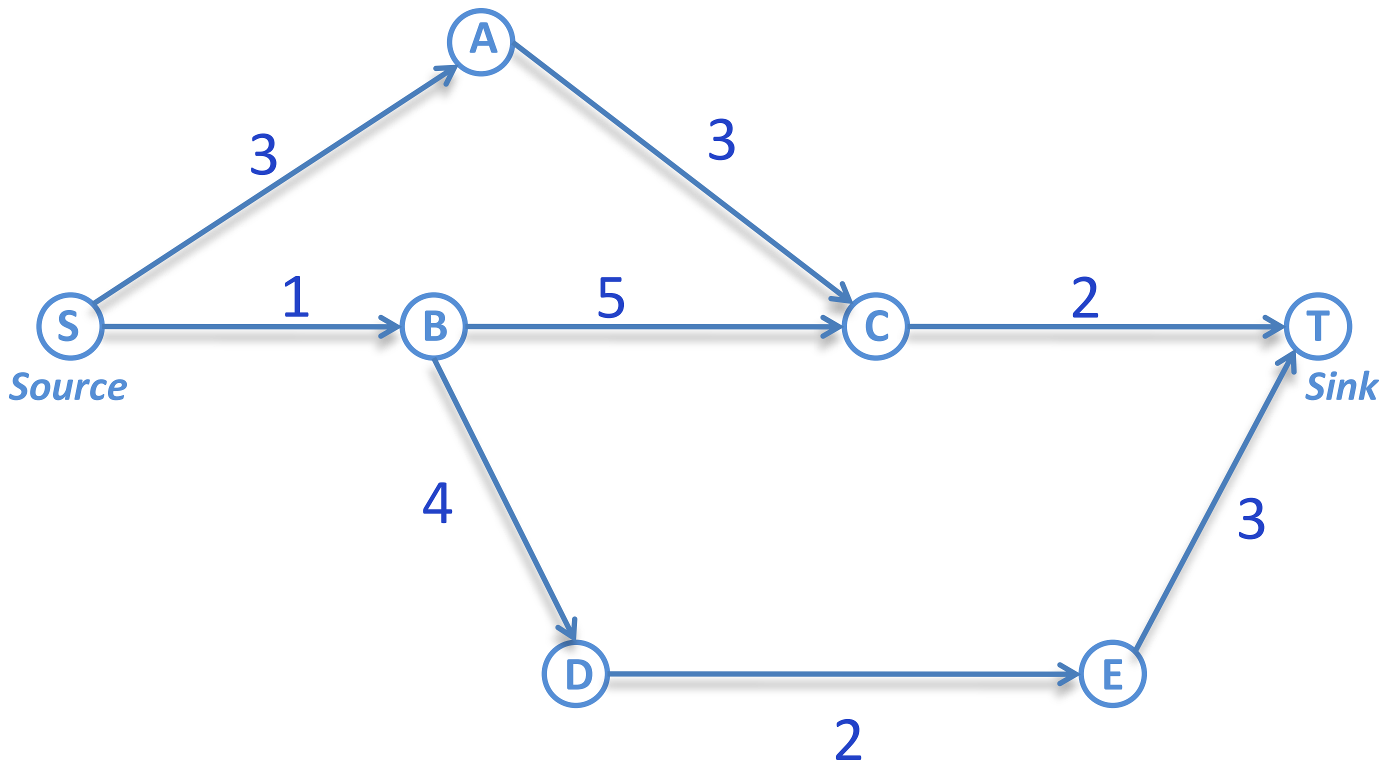
out Flow = 0

Others :

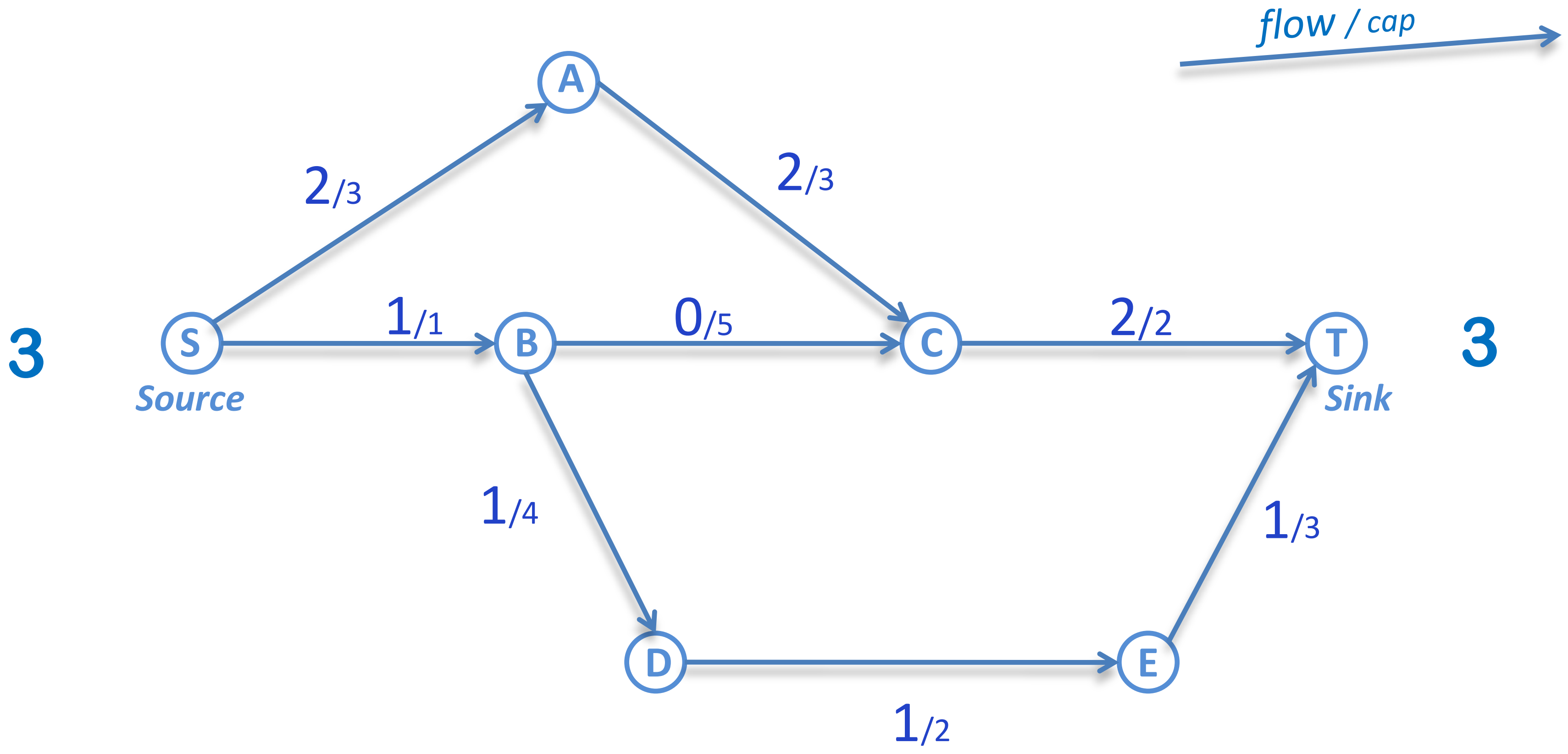
in Flow = out Flow

∴ Flow out of Source = Flow in to Sink

Can you find the MaxFlow for this ?



here is the MaxFlow



so how can we find the MaxFlow ?

…by keep sending some positive flow from Source to Sink

$$\begin{tikzcd} S \arrow[r, "4"] & B \arrow[r, "7"] & C \arrow[r, "3"] & T \end{tikzcd}$$

If we send flow = 1

$$\begin{tikzcd} S \arrow[r, "1/4"] & B \arrow[r, "1/7"] & C \arrow[r, "1/3"] & T \end{tikzcd}$$

instead, look at capacity left

$$\begin{tikzcd} S \arrow[r, "3"] & B \arrow[r, "6"] & C \arrow[r, "2"] & T \end{tikzcd}$$

How much more can we send ?
Minimum capacity of an edge along the path

$$\begin{tikzcd} S \arrow[r, "1"] & B \arrow[r, "4"] & C \arrow[r, "0"] & T \end{tikzcd}$$

keep sending more and more…

So all we need to do is,

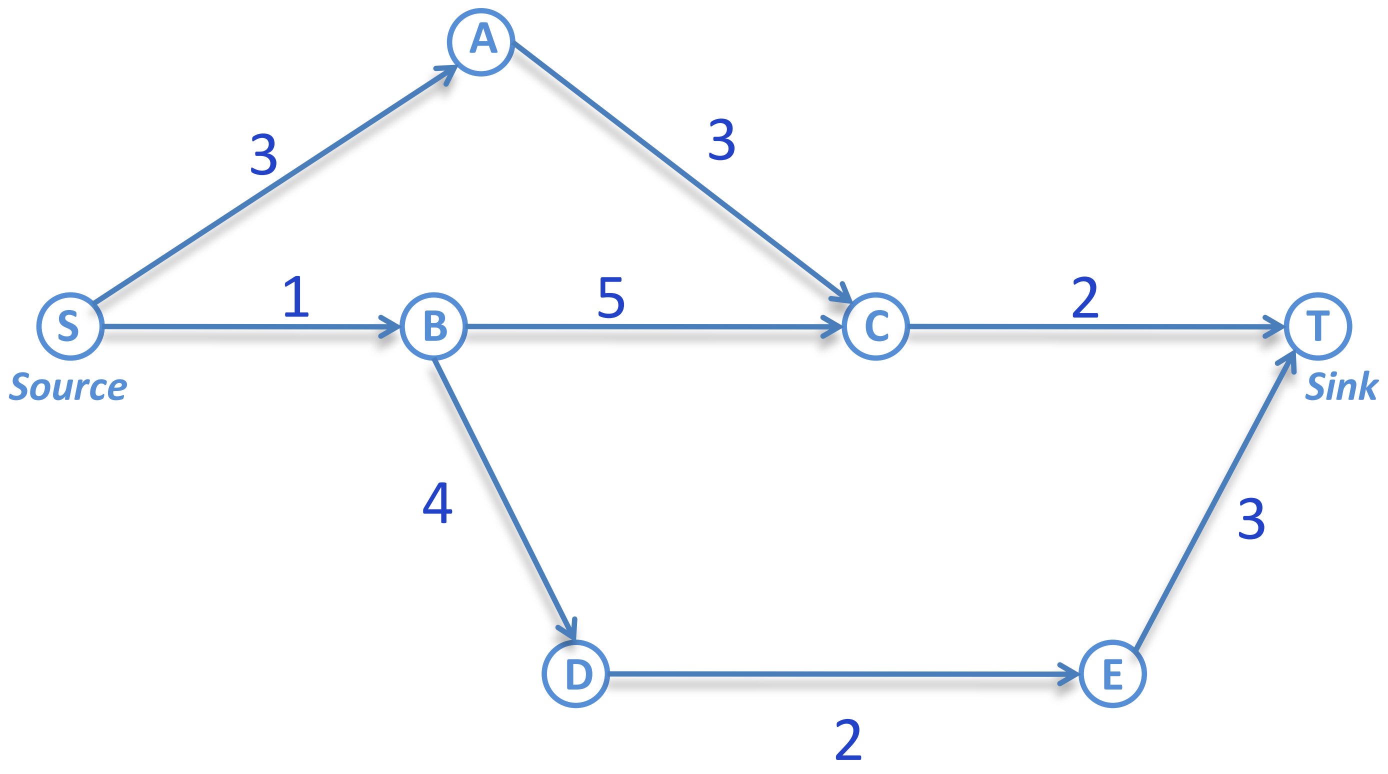
$$\left.\begin{array}{l}\textit{“ Find a path from Source to Sink} \\ \textit{having edge with minimum capacity left > 0 ”}\end{array}\right\} \textbf{Augmenting Path}$$

ALGORITHM

```
While ( there is an augmenting path P )
{
        Increase flow along P and adjust capacities left
}
```

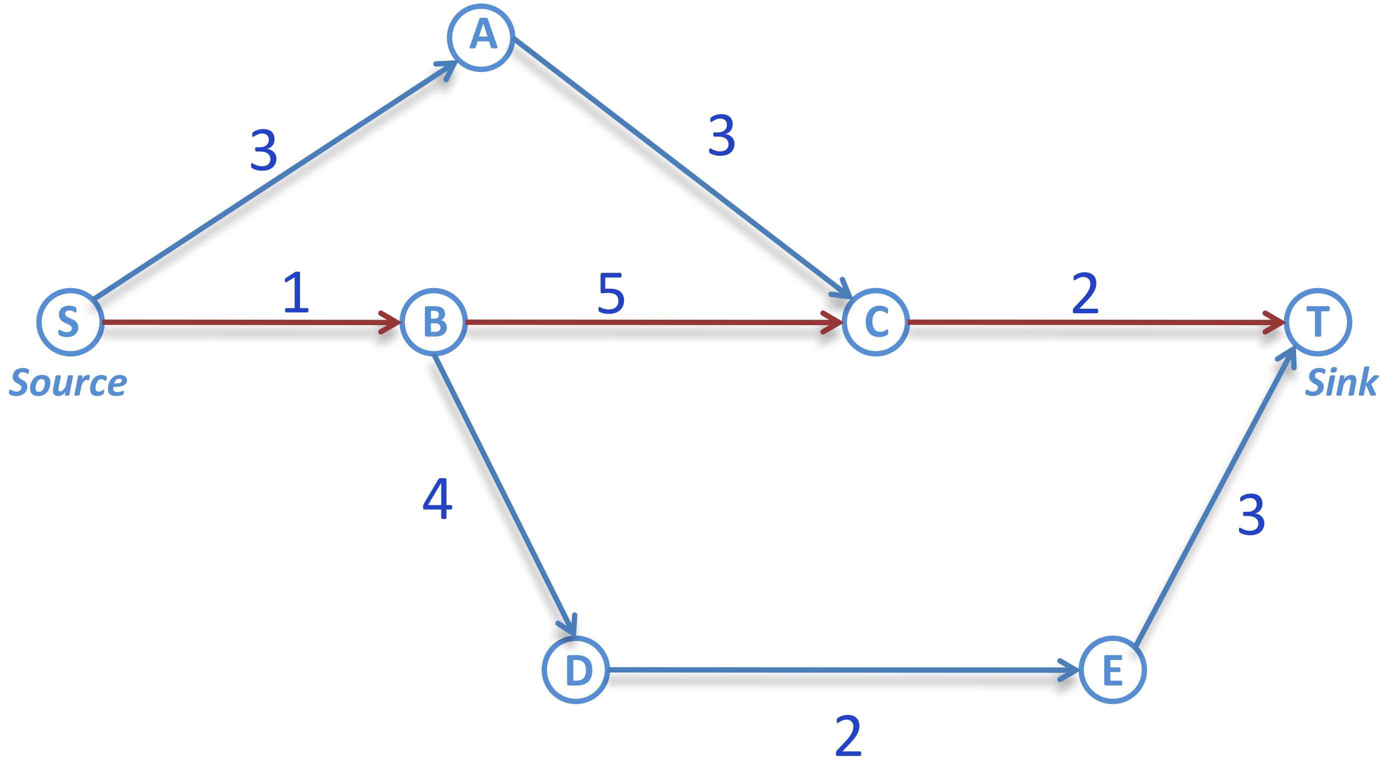
we are almost done.. just one more observation

Running MaxFlow



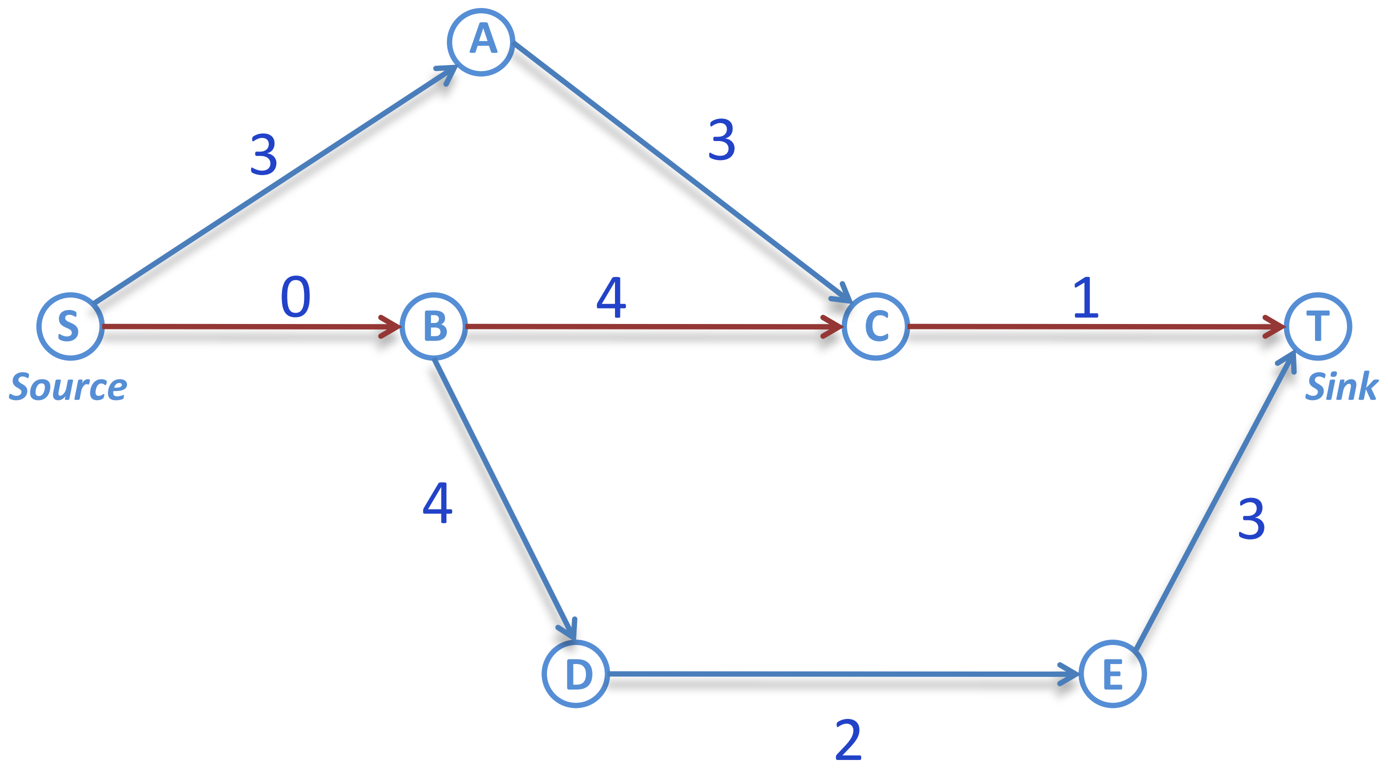
flow = 0

Running MaxFlow



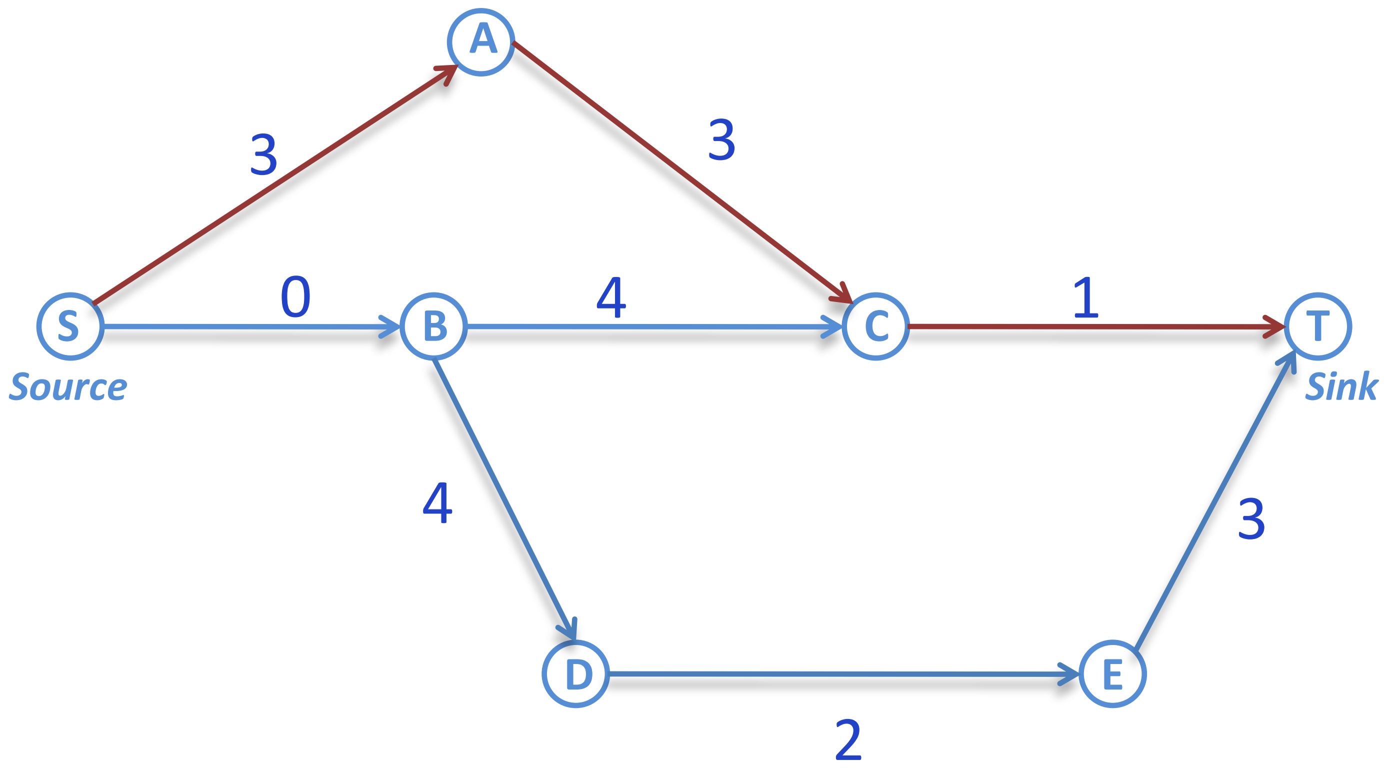
flow = 0

Running MaxFlow



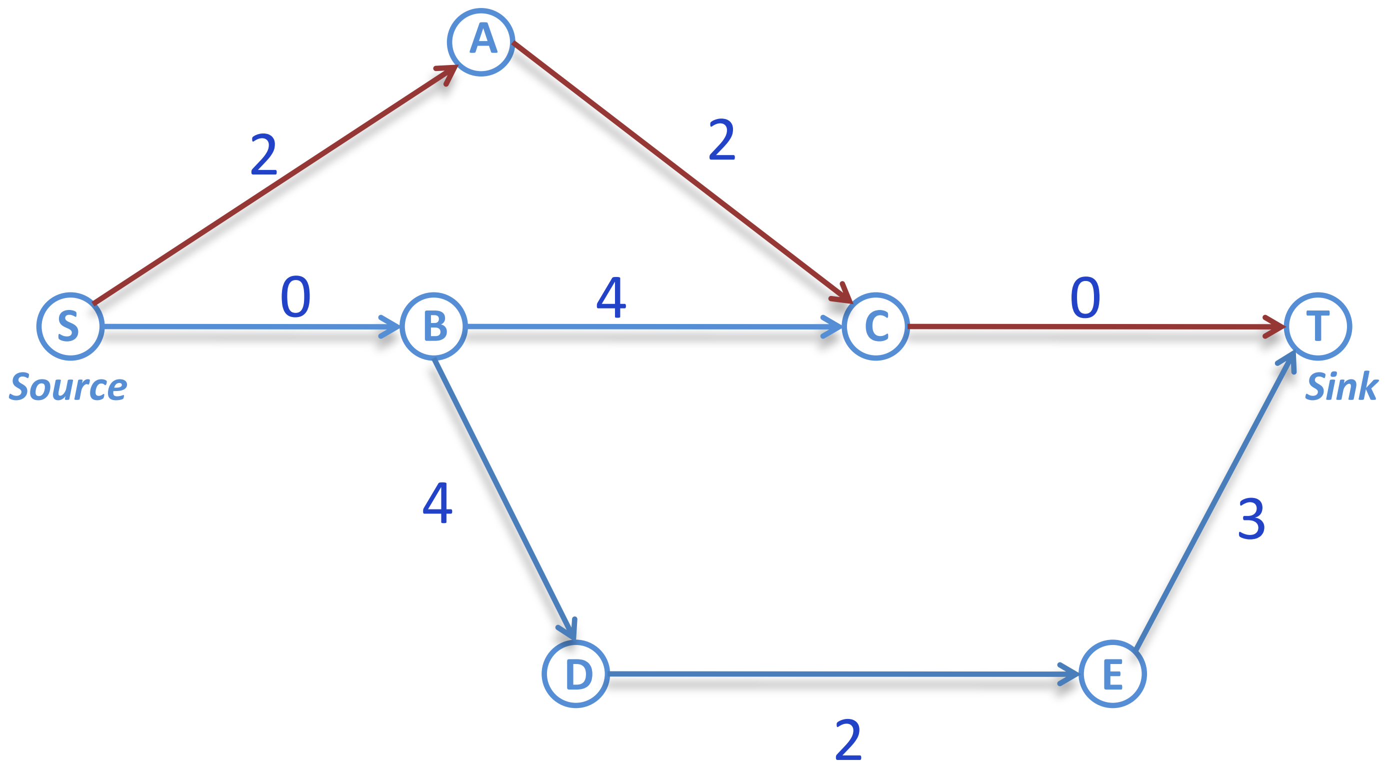
flow = 1

Running MaxFlow



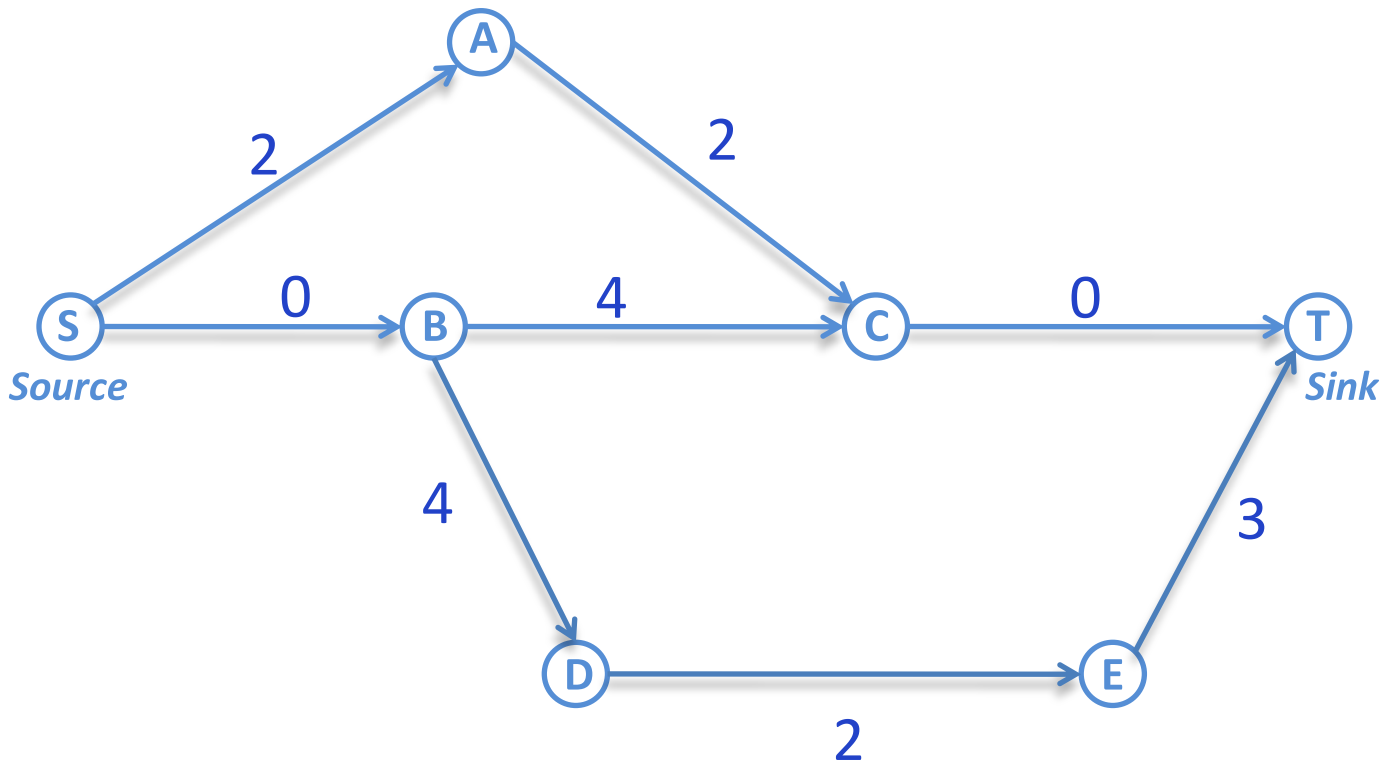
flow = 1

Running MaxFlow



flow = 2

Running MaxFlow

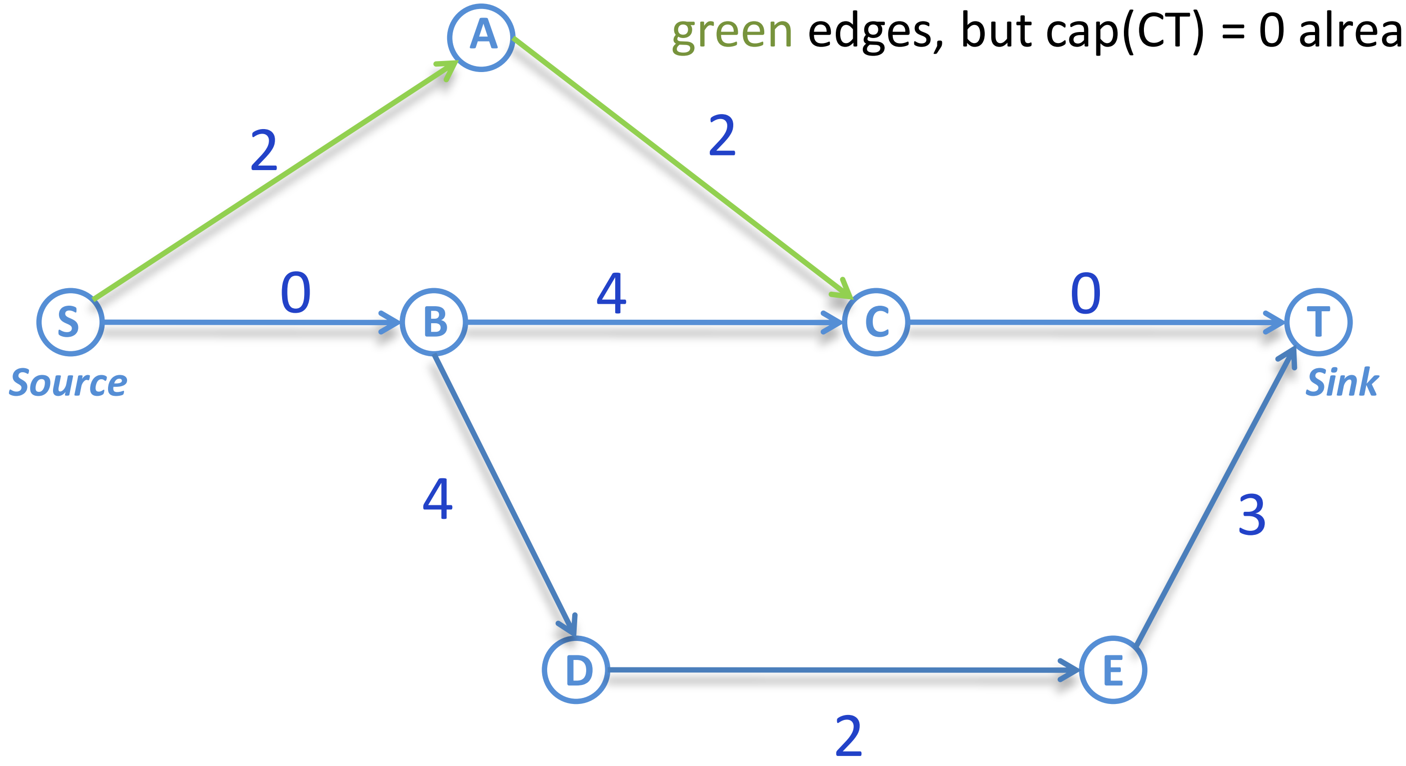


flow = 2

There seems to be no augmenting path now, but MaxFlow was 3   :-/

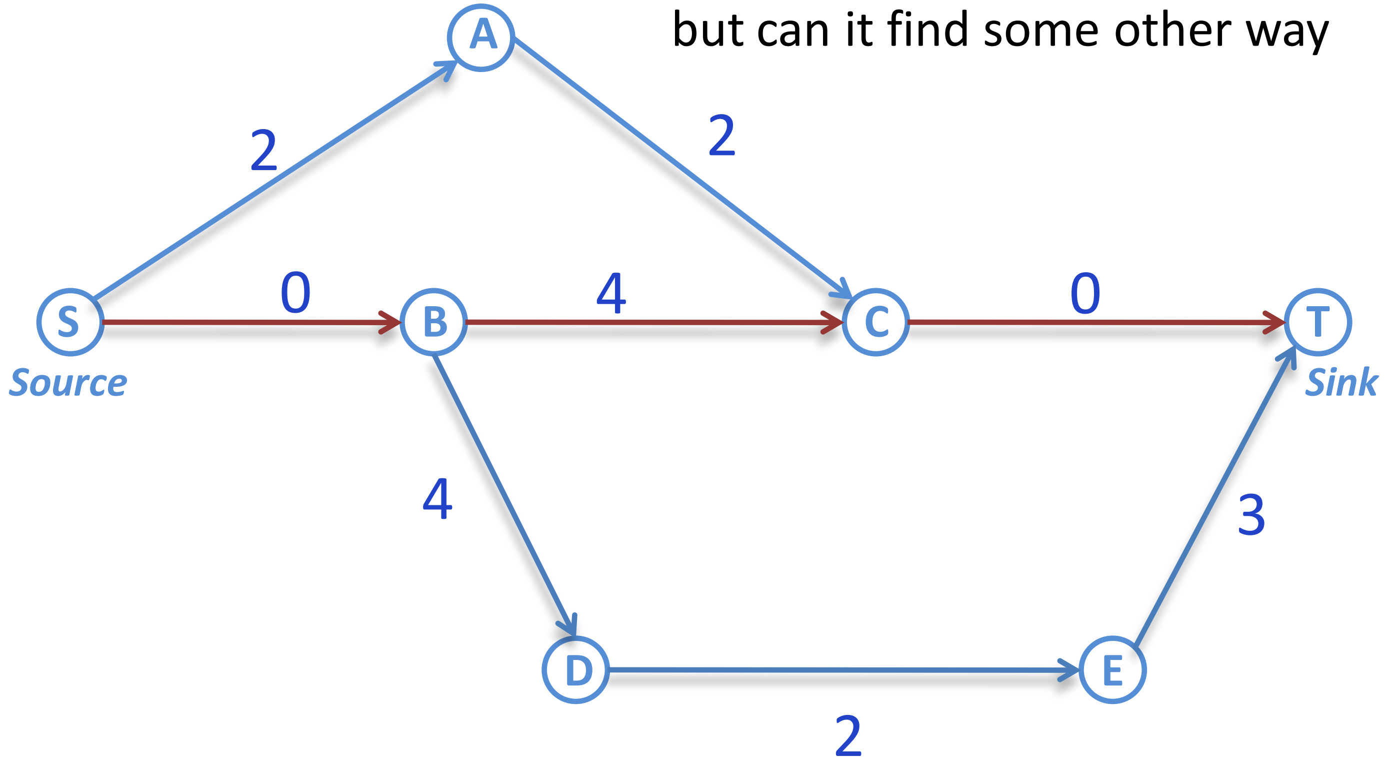
Running MaxFlow

We can try to increase flow along green edges, but cap(CT) = 0 already ☹



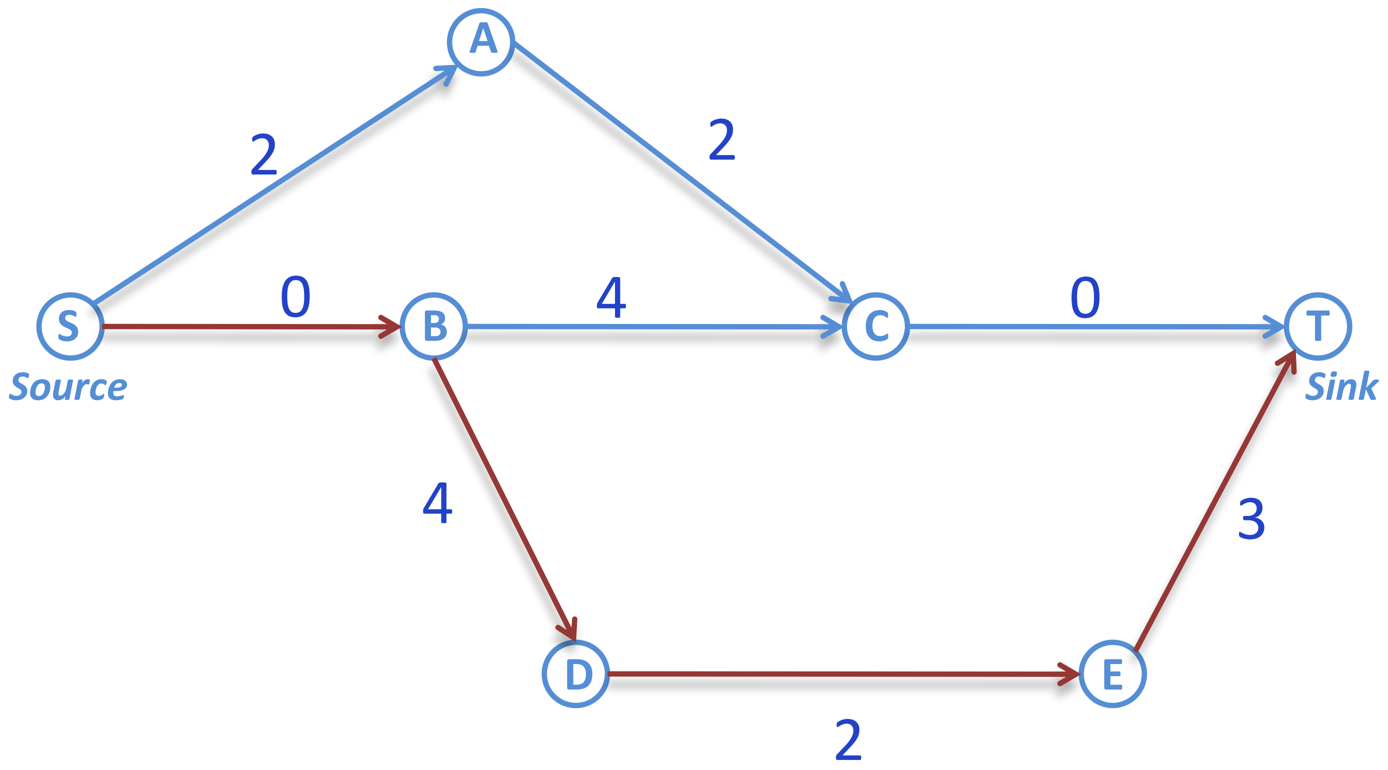
Running MaxFlow

There is flow = 1 along red path already, but can it find some other way

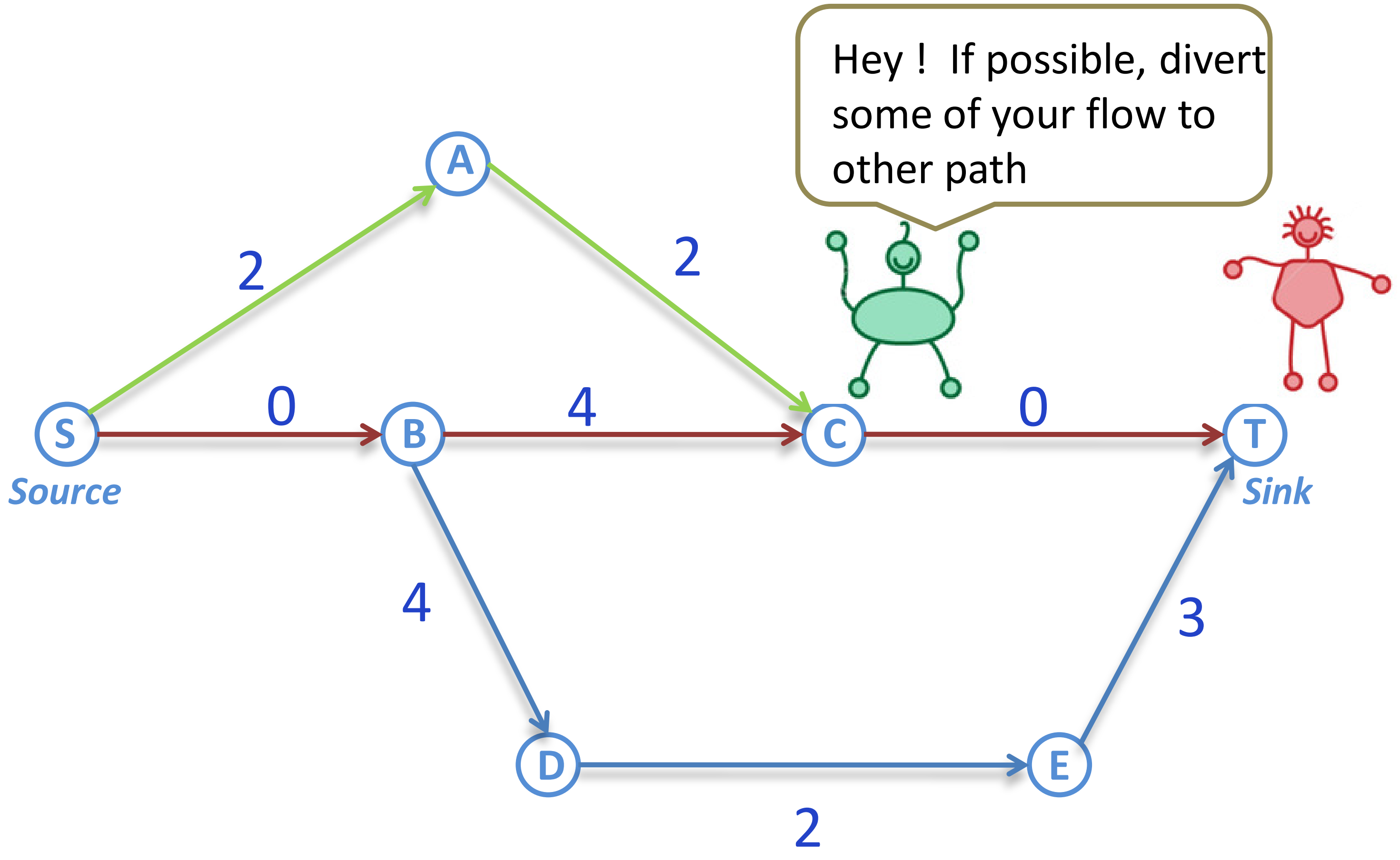


Running MaxFlow

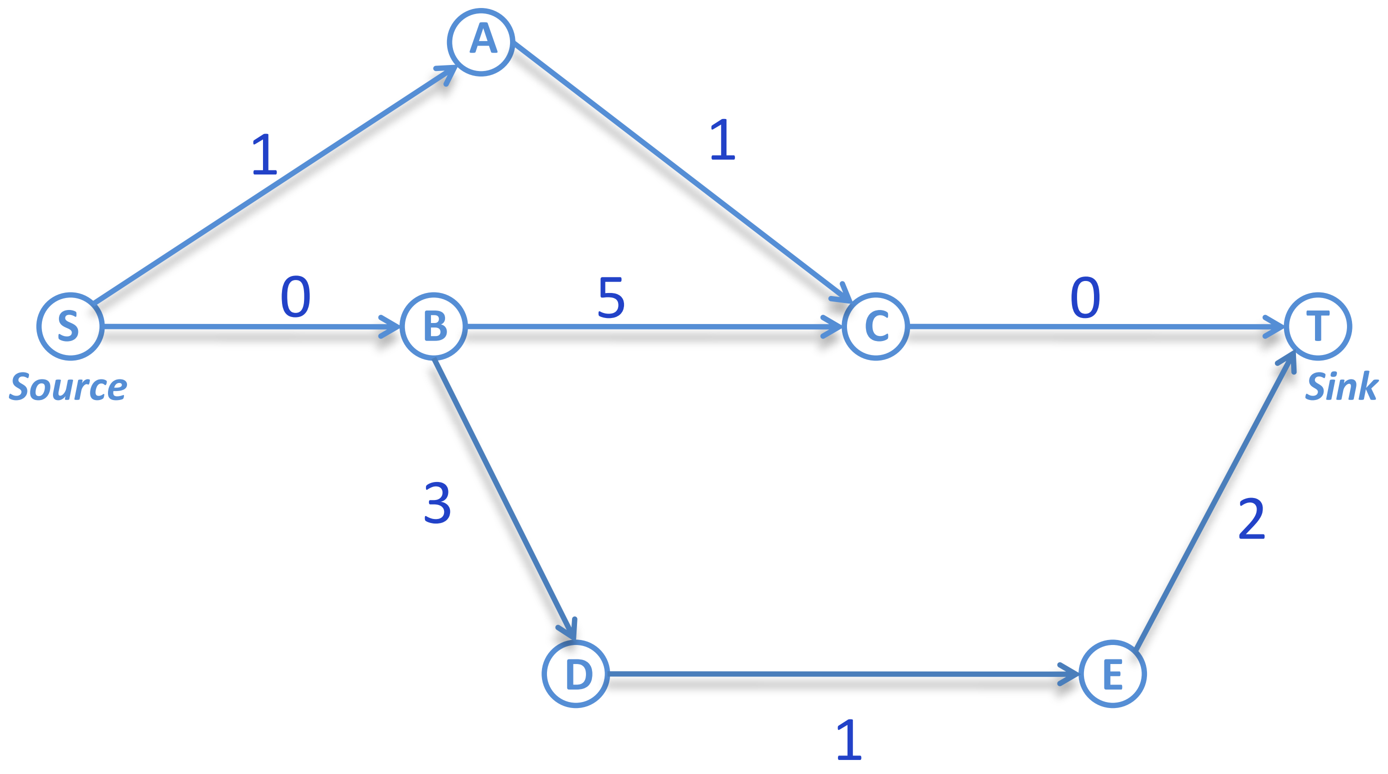
Yes !



Running MaxFlow



Running MaxFlow

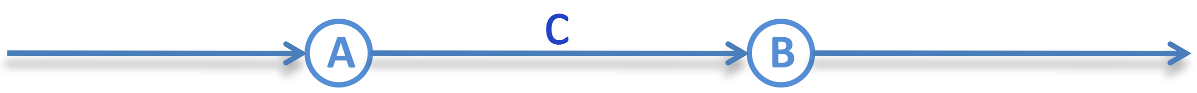


After diversion of some existing flow, we can increase overall flow to 3

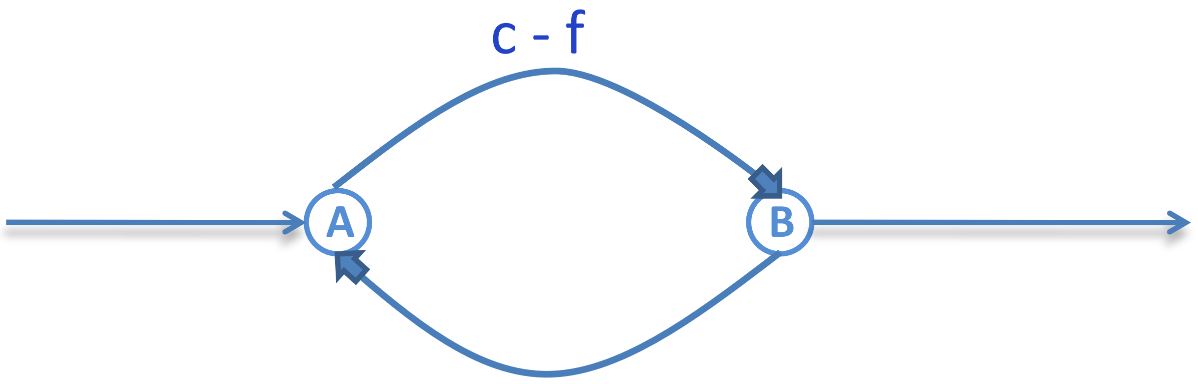
Diversion looks complex.. lets simplify it by adding additional edges

Residual Graph

Edges with capacity left

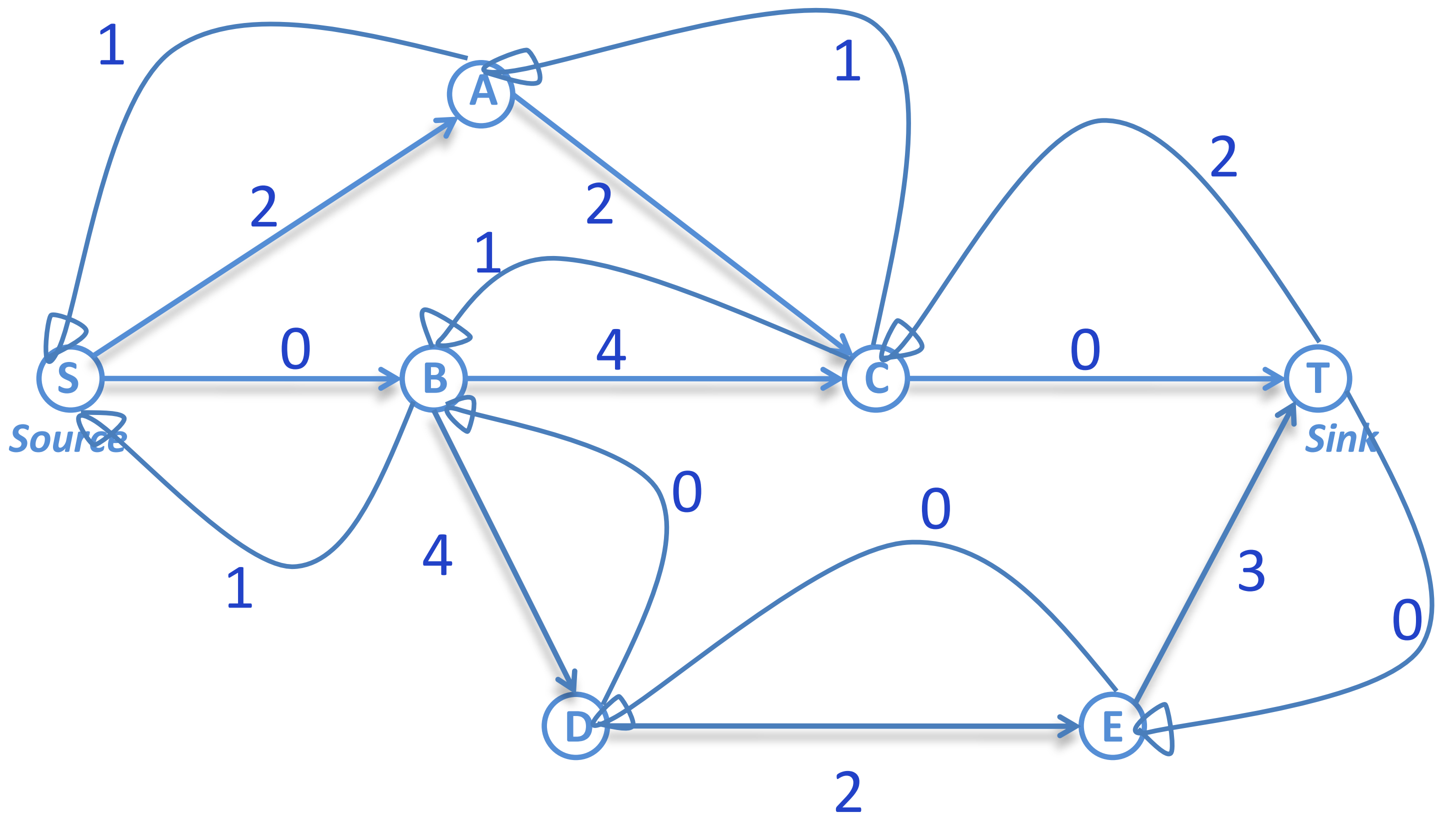


after sending f flow



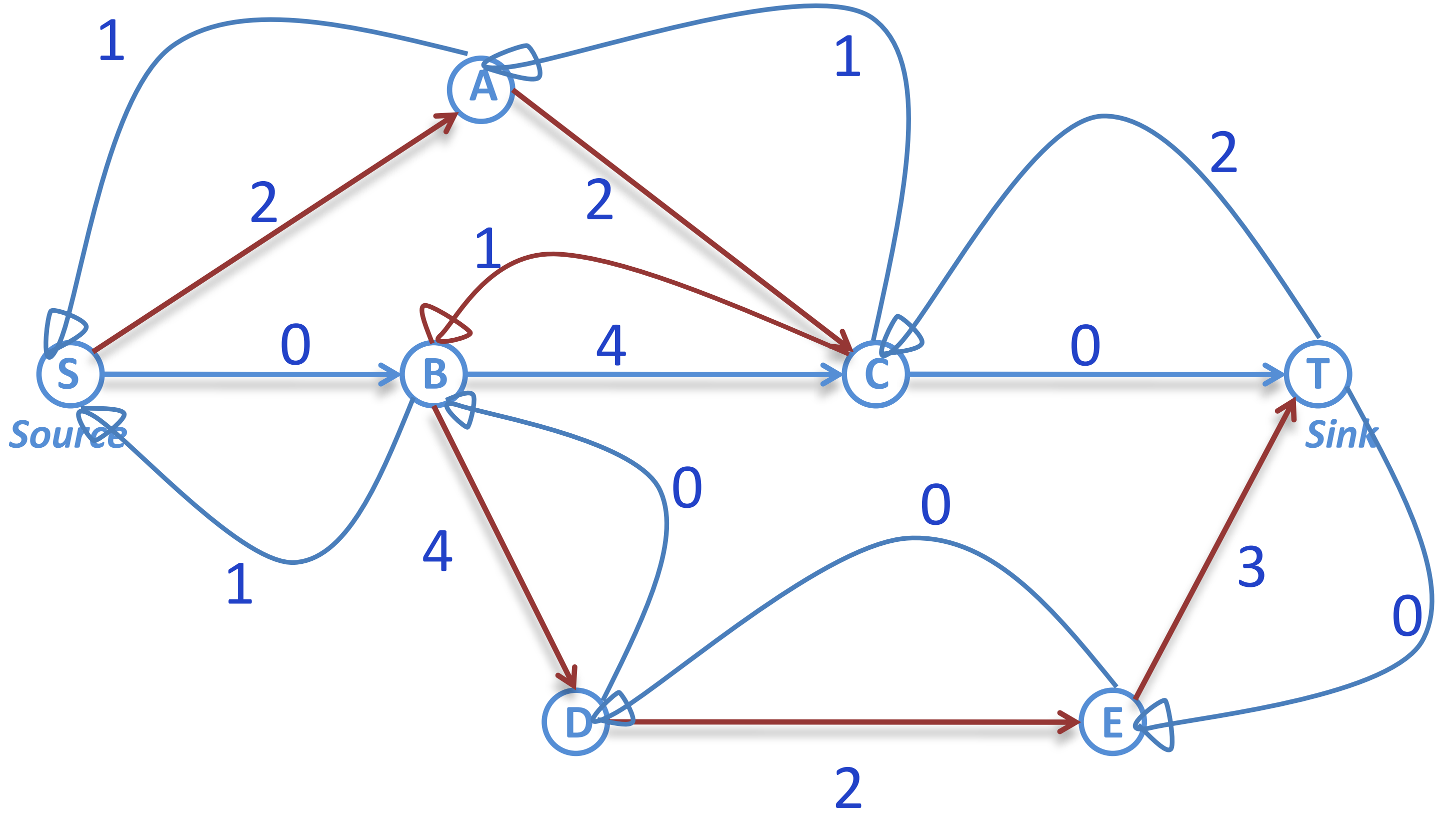
f maximum you can send back if needed

after flow = 2, the residual graph



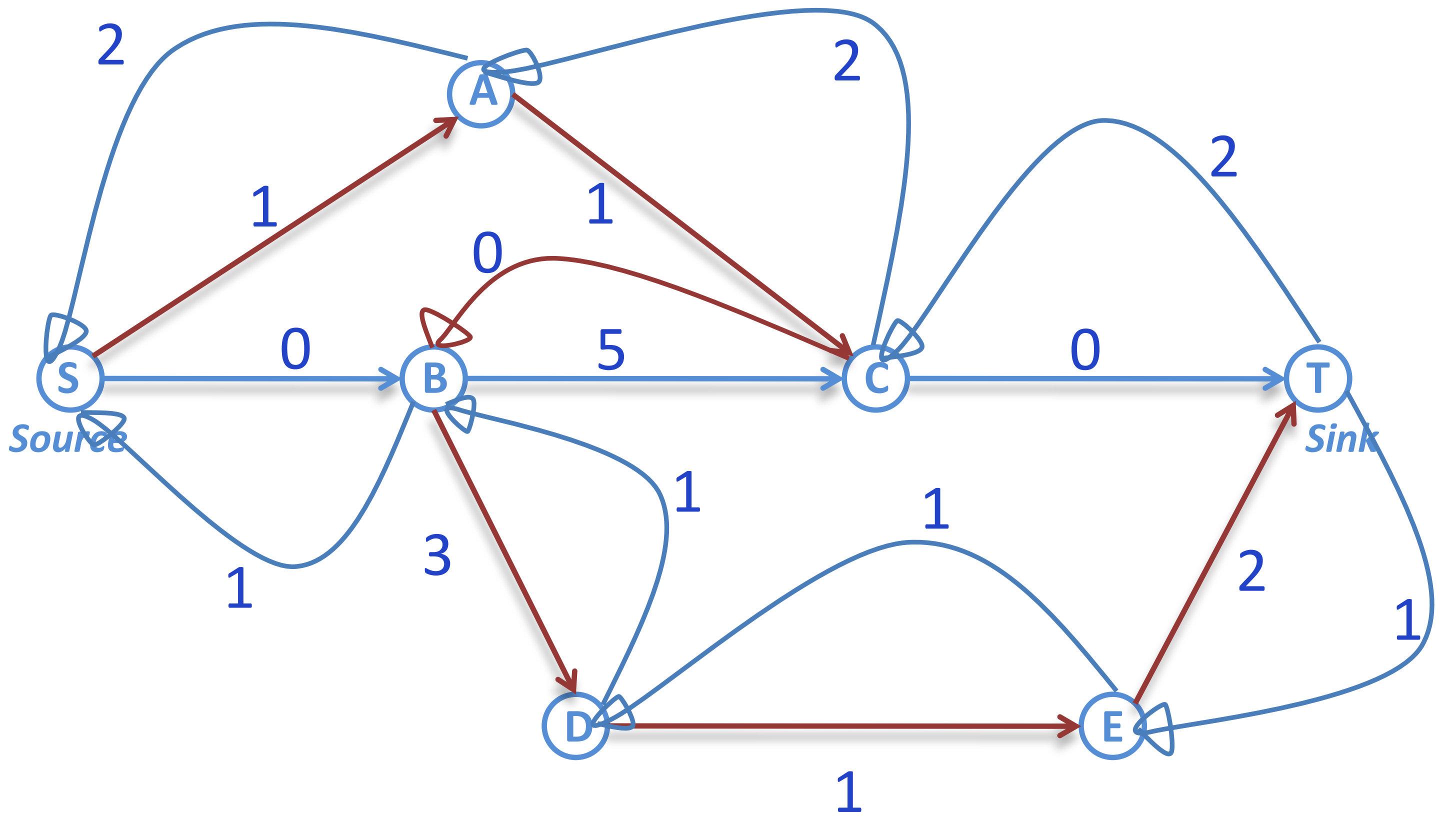
flow = 2

now here is one more augmenting path



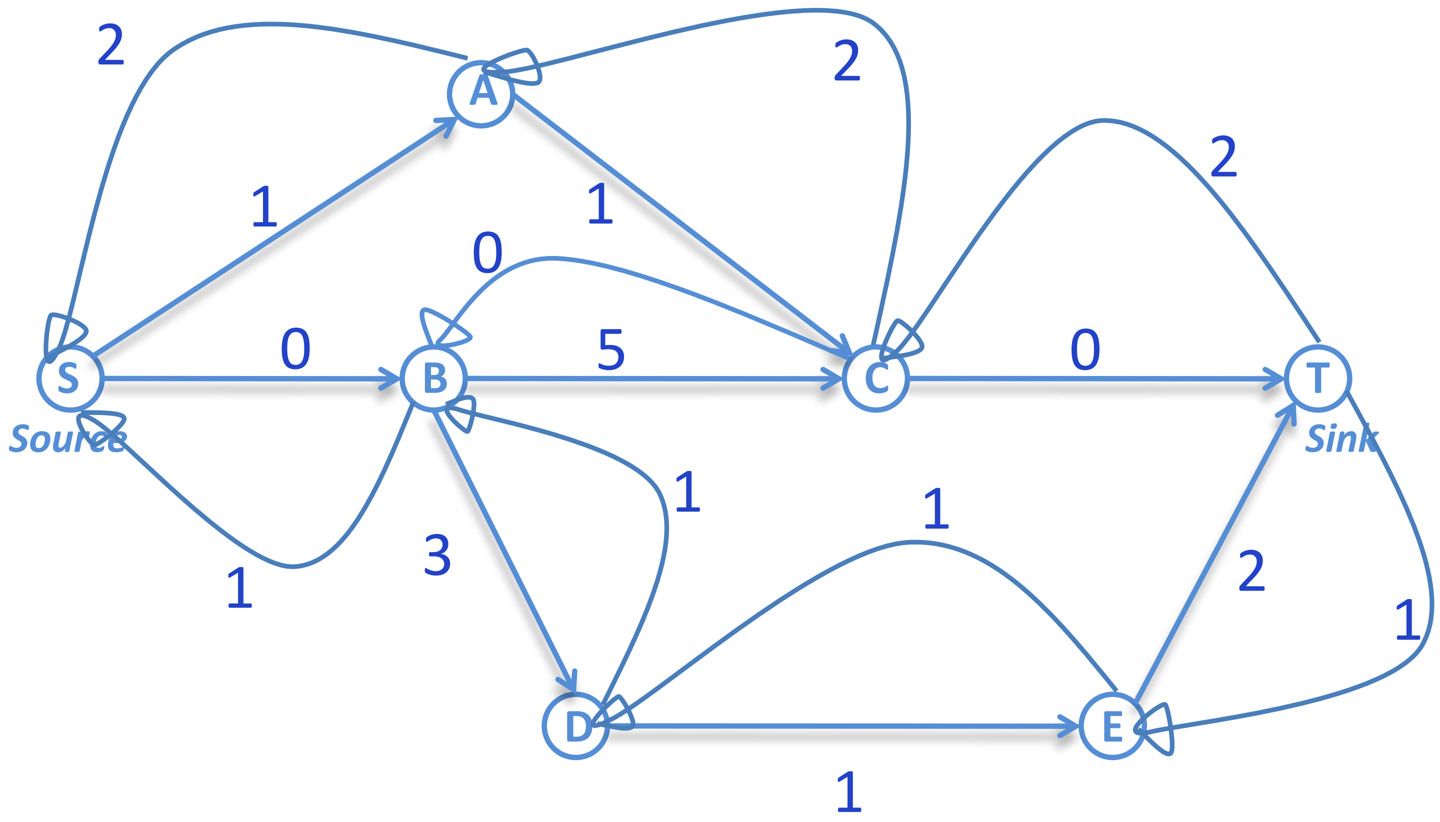
flow = 2

after augmenting



flow = 3

No more augmenting paths



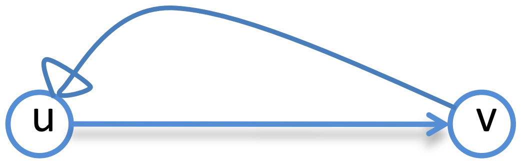
flow = 3

its actually very simple finally !

ALGORITHM

```
While ( there is an augmenting path P in residual network )
{
        Increase flow along P and adjust capacities left
}
```

Adjust :



Sending *f* flow from *u* to *v*

Cap[u][v] - = f
Cap[v][u] + = f

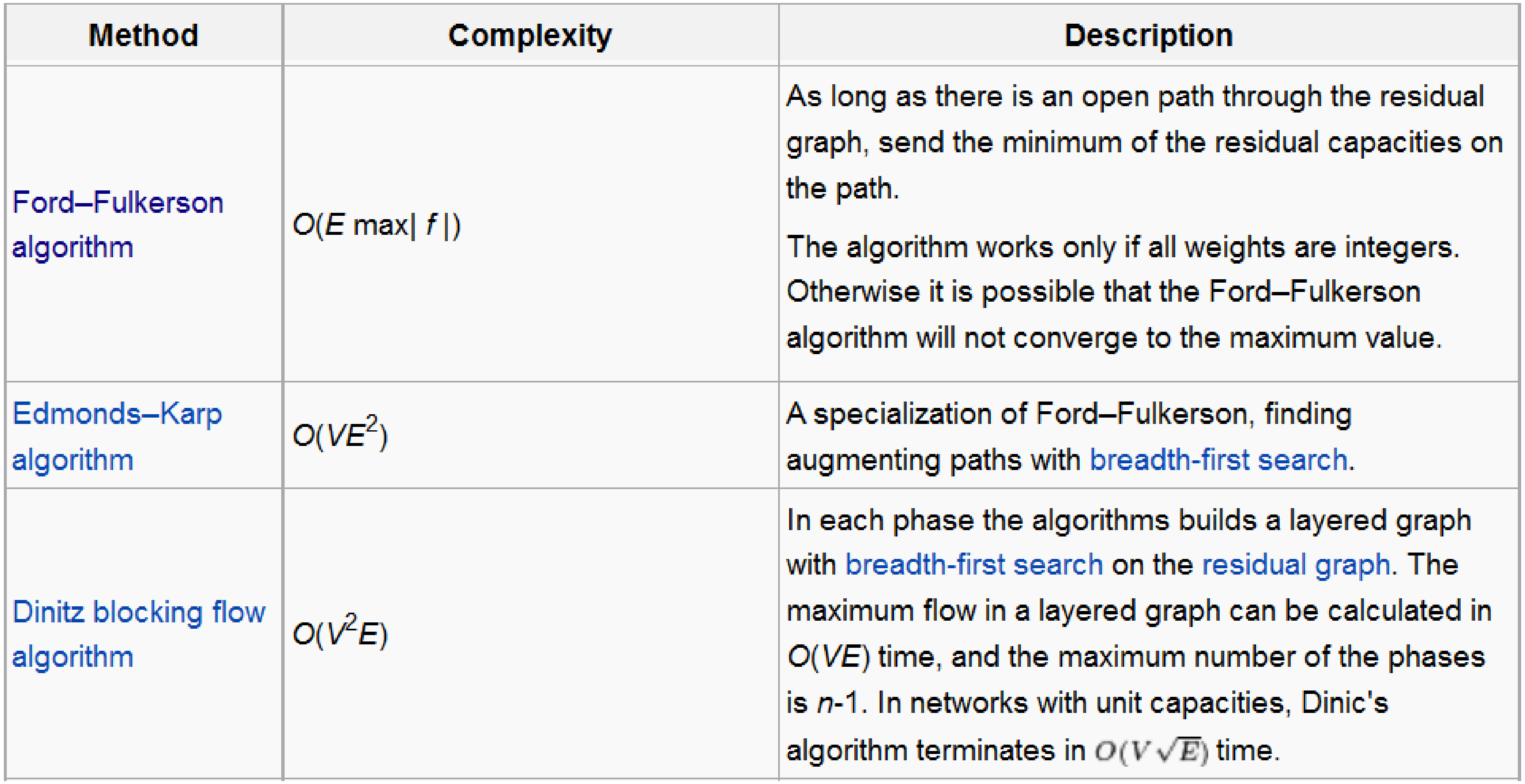
a simple application

(***its not always water that flows***)

A company owns a factory located in city X where products are manufactured that need to be transported to the distribution center in city Y.

You are given the one-way roads that connect pairs of cities in the country, and the maximum number of trucks that can drive along each road. What is the maximum number of trucks that the company can send to the distribution center

some commonly used algorithms



Method	Complexity	Description
Ford–Fulkerson algorithm	$O(E \max\| f \|)$	As long as there is an open path through the residual graph, send the minimum of the residual capacities on the path. The algorithm works only if all weights are integers. Otherwise it is possible that the Ford–Fulkerson algorithm will not converge to the maximum value.
Edmonds–Karp algorithm	$O(VE^2)$	A specialization of Ford–Fulkerson, finding augmenting paths with breadth-first search.
Dinitz blocking flow algorithm	$O(V^2E)$	In each phase the algorithms builds a layered graph with breadth-first search on the residual graph. The maximum flow in a layered graph can be calculated in $O(VE)$ time, and the maximum number of the phases is n-1. In networks with unit capacities, Dinic's algorithm terminates in $O(V\sqrt{E})$ time.

source : Wikipedia

All that matters is which augmenting path to choose now !

lets solve one quickly from scratch

- http://www.spoj.pl/problems/POTHOLE/
  - By finding augmenting path using
    - DFS
    - BFS

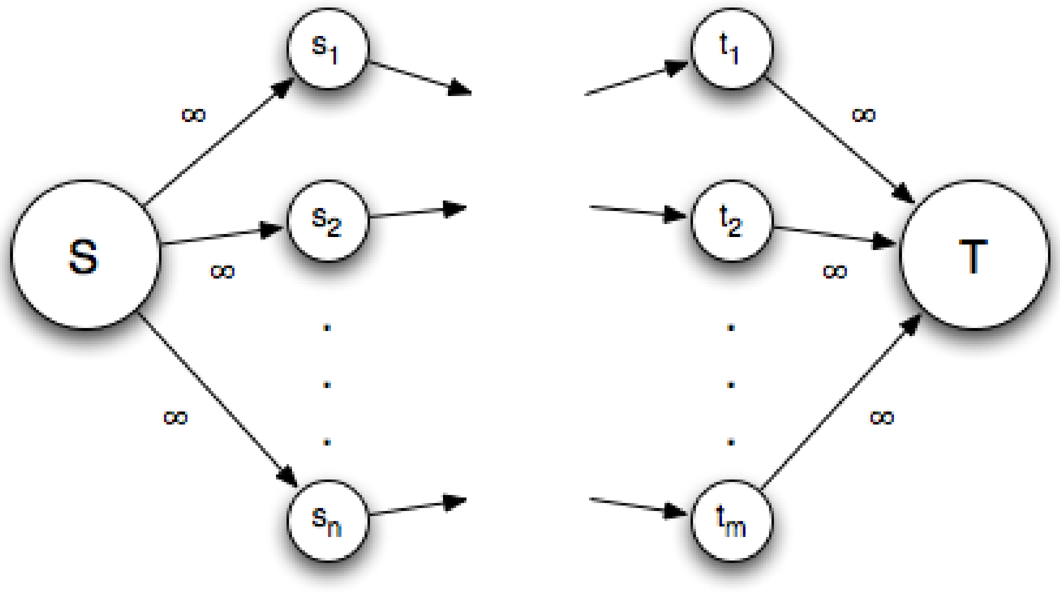
- Coding is pretty standard. The fun part is how to visualize it as a maxflow problem, what are the nodes, edges and how to add the constraints given in the problem as capacities

now some standard Variations, Applications and Theorems on MaxFlow

Multiple Sources / Sinks

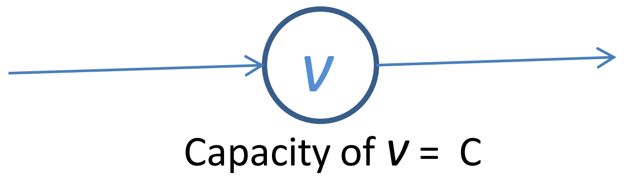
• Add a Super Source and a Super Sink and corresponding edges between them

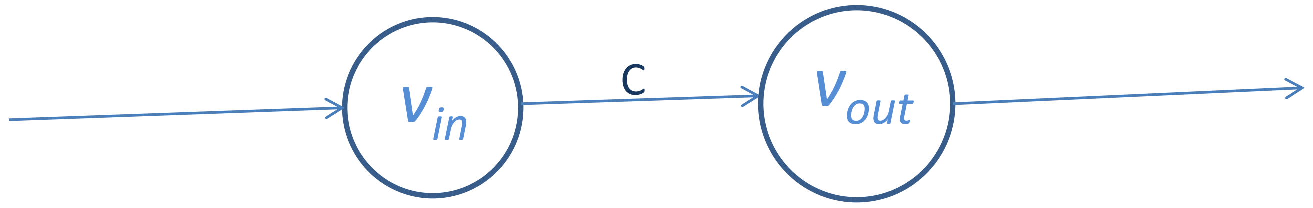
• Restriction on maximum amount a Source can produce / maximum amount a Sink can consume, can be added as corresponding edge capacities



Vertex Capacities

- In the truck example : in addition to “maximum number of trucks that can drive along each road” there can also be, “maximum number of trucks that can drive through a city”





split the vertex

Disjoint paths

- If all edges have unit capacity then

  MaxFlow = Maximum number of edge disjoint paths from S to T

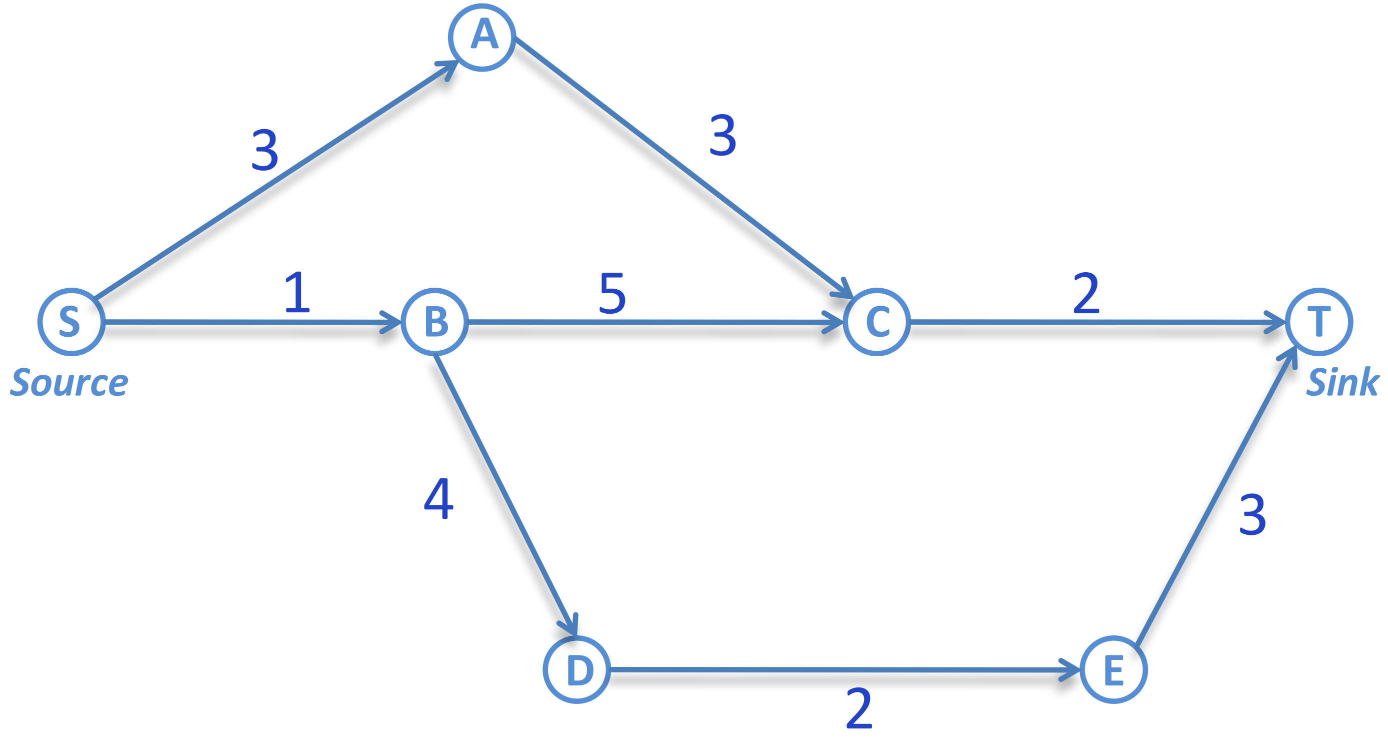
- Adding unit capacity to each vertex, similarly

  MaxFlow = Maximum number of vertex disjoint paths from S to T

Min-Cut

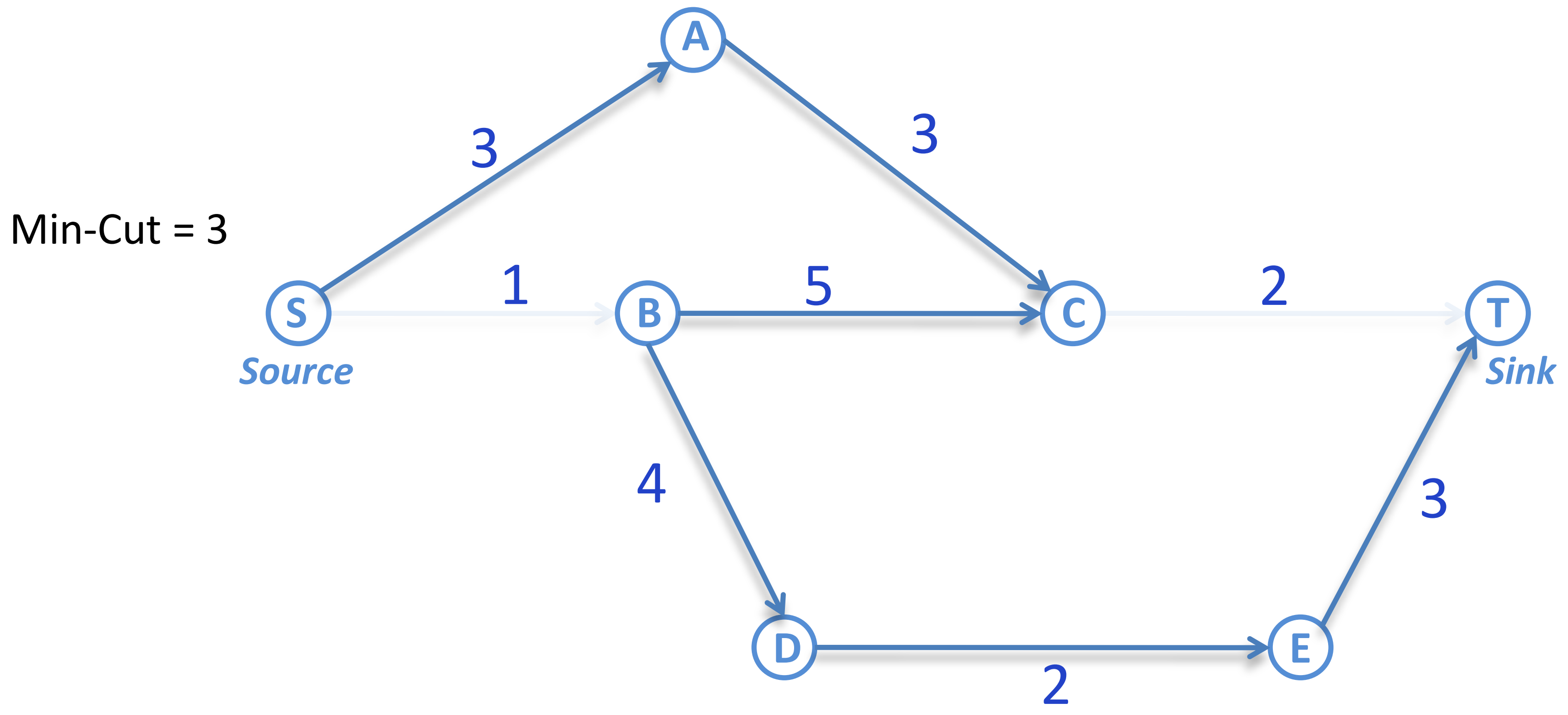
- Min-Cut : Given a weighted directed graph, remove a minimum-weighted set of edges in such a way that S is unreachable from T

Find a Min-Cut :



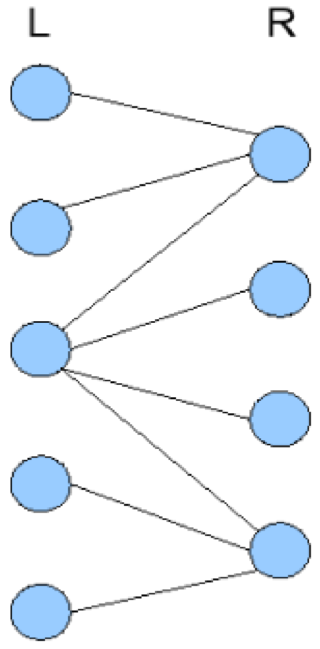
Max-Flow-Min-Cut Theorem

Maximum Flow = Minimum Cut



Maximum Bipartite Matching

- Find maximum cardinality matching (independent edges) in a bipartite graph

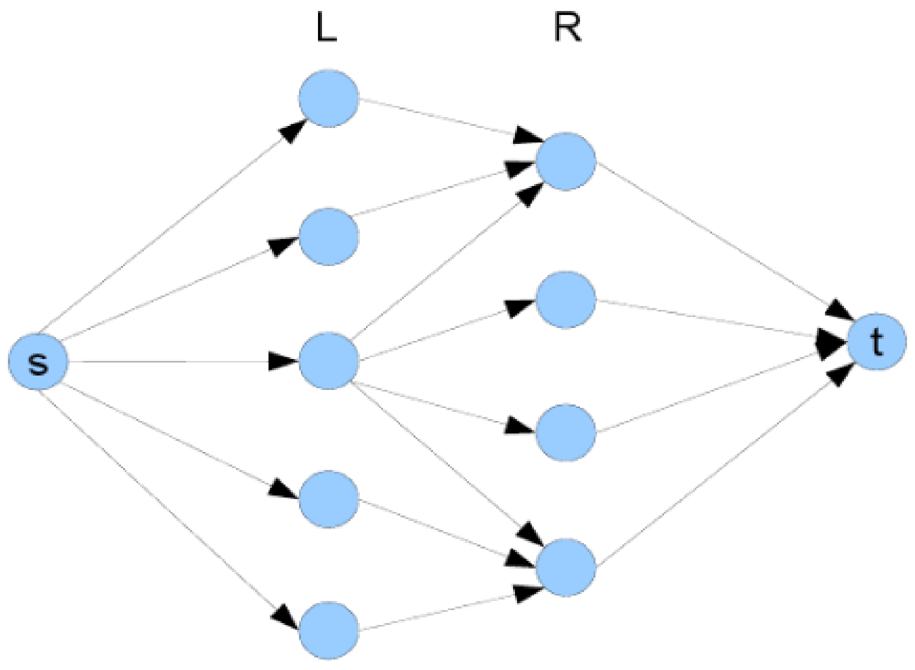


Bipartite Graph

Maximum Bipartite Matching

- Find maximum cardinality matching (independent edges) in a bipartite graph

Max Bipartite Matching
=
Max Flow from S to T



Flow Network

other related theorems

Do check at least the Wikipedia pages of them to get basic idea.
Very helpful while solving problems !

- Konig's Theorem : In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover.

- Hall's Theorem : For a set *W* of vertices of *G*, let $N_G(W)$ denote the neighborhood of *W* in *G*, i.e. the set of all vertices adjacent to some element of *W*. An *X*-saturating matching exists if and only if for every subset *W* of *X*

  $$|W| \le |N_G(W)|.$$

  In other words every subset *W* of *X* has enough adjacent vertices in *Y*.

- Dilworth's Theorem : In any finite partially ordered set, the maximum number of elements in any antichain equals the minimum number of chains in any partition of the set into chains.

  |Chain Cover Set| = |V| - |Max Bipartite Matching|

  http://www.codechef.com/problems/POSTERS

Problems Discussion

Frog Jumps

- A frog wants to jump from left bank of the river to the right bank, which are D units apart. There are some stones in the river where it can jump on.
  - X = 0, left bank
  - X = D, right bank
  - Given array X[i] of N stones in order

➢ Minimize the maximum distance jump the frog needs to make

➢ Minimize the maximum distance jump the frog needs to make, if it is allowed to make not more than K jumps

Frog Jumps

- A frog wants to jump from left bank of the river to the right bank, which are D units apart. There are some stones in the river where it can jump on.
  - X = 0, left bank
  - X = D, right bank
  - Given array X[i] of N stones in order

- The frog wants to start from left bank and collect food from right bank and want to return back to left bank. Some stones are so small that at most one jump can be made on each of them. Other stones are big and can jump any number of times.

➢ Minimize the maximum distance jump the frog needs to make

Dancing Party TC#SRM399

- You are organizing a dance party. The party will be attended by n boys and n girls. There will be several rounds of dancing. In each round, you divide the guests into n dancing pairs. Each guest must be in exactly one pair, and each pair must contain one boy and one girl. Each boy must dance with a different girl in every round. Some boys and girls like each other and some do not. During the party, each boy agrees to dance with at most **k** girls he doesn't like. Similarly, each girl agrees to dance with at most **k** boys she doesn't like.
- You are given a String[] **likes**. The j-th character of **likes**[i] is 'Y' if the i-th boy and the j-th girl like each other, and 'N' if they don't. Return the maximum number of rounds you can organize.

Practice

- Problems on SPOJ - TAXI, POTHOLE, IM, QUEST4, MUDDY, EN, CABLETV, STEAD, NETADMIN, COCONUTS, OPTM

- Many awesome problems on TopCoder

- Look at Baseball-Elimination problem, Cycle cover

End