The Ford-Fulkerson algorithm to find the Maximum Flow in a Flow network

Author: Abdullah Al Mamun Oronno
12th Batch, CSEDU.

Revised by: …….

Introduction

The maximum flow problem is to find a feasible flow through a single-source, single-sink flow network that is maximum. The simplest form that the statement could take would be something along the lines of: "A list of pipes is given, with different flow-capacities. These pipes are connected at their endpoints. What is the maximum amount of water that you can route from a given starting point to a given ending point?" or equivalently "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?"

Given a directed graph G = (V, E) with integer capacities, c : E, and a source node s and a sink node t in V. Find a flow function on the arcs subject to 'conservation constraints' and 'capacity constraints'. The conservation constraint for a node is that the sum of the flow on incoming arcs is equal to the sum on outgoing arcs for all nodes other than the source and the sink. The capacity constraint for an arc is that the flow is no larger than the capacity. One wishes to maximize the flow out of the source (into the sink).

A flow network G = (V, E) is a directed graph in which each edge $(u,v) \in E$ has a nonnegative capacity $c(u,v) \geq 0$. A flow in G is a real-valued function $f: V \times V \rightarrow R$ that satisfies the following three properties:

Capacity constraint: For all $u, v \in V$, we require $f(u, v) \leq c(u, v)$.

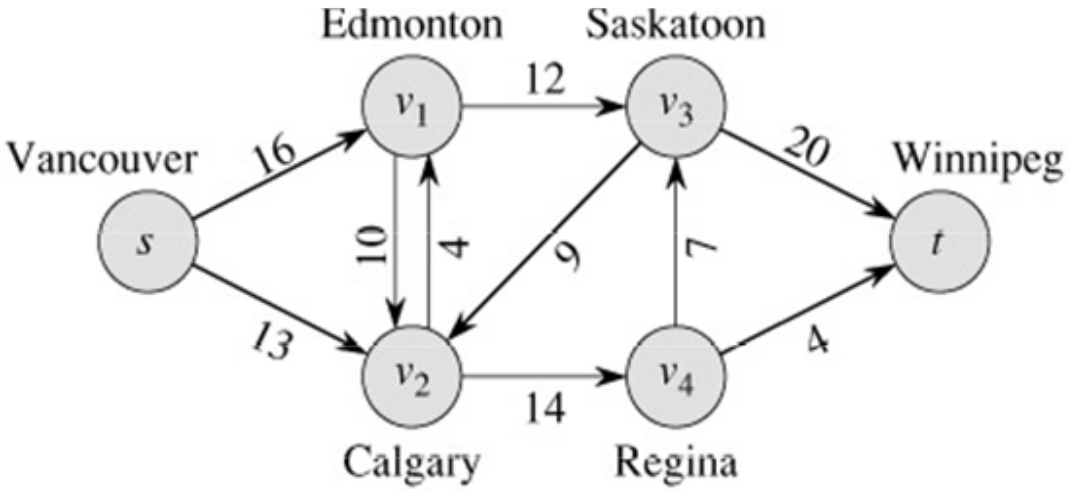
Skew symmetry: For all $u, v \in V$, we require $f(u, v) = -f(v, u)$.

Flow conservation: For all $u \in V - \{s, t\}$, we require

$$\sum_{v \in V} f(u, v) = 0 .$$

The quantity $f(u, v)$, which can be positive, zero, or negative, is called the ***flow*** from vertex u to vertex v.

Let consider the following Graph.



Here, Each edge (u,v) has a nonnegative capacity c(u,v). We have a source s, and a sink t.
A flow in the network is an integer-valued function **f** defined on the edges such that **f(u,v) <= c(u,v)**.

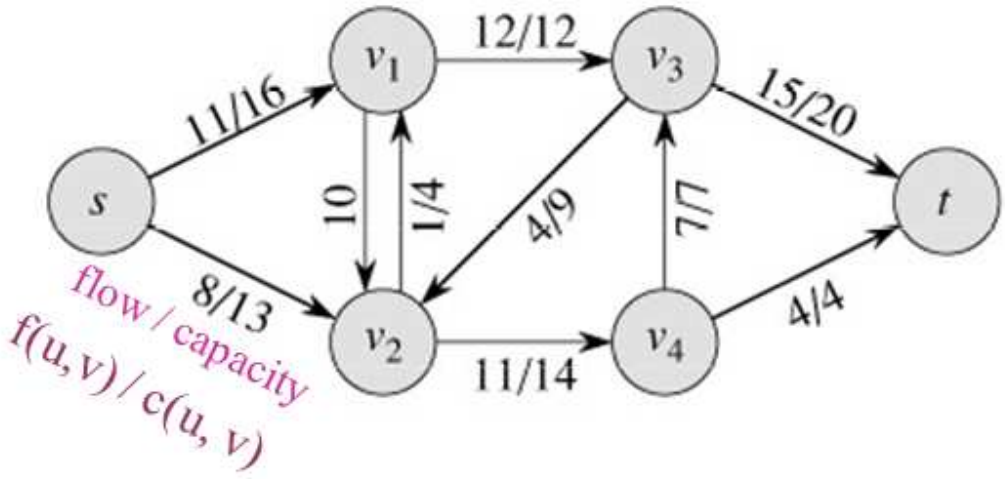


Fig: An example of a flow network with a maximum flow. The source is s, and the sink t. The numbers denote flow and capacity. We get total the amount 19 as the maximum flow.

Ford-Fulkerson method

- The Ford-Fulkerson method computes the maximum flow in a flow network. It was published in 1955 by L. R. Ford, Jr. and D. R. Fulkerson.
- We can call it a “method” rather than an “algorithm” because it encompasses several implementations with differing running times. The Ford-Fulkerson method depends on three important ideas that transcend the method and are relevant to many flow algorithms and problems: **residual networks**, **augmenting paths**, and **cuts**.

Residual Networks

Suppose that we have a flow network G = (V, E) with source s and sink t. Let f be a flow in G, and consider a pair of vertices u,v ∈ V. The amount of additional flow we can push from u to v before exceeding the capacity c(u,v) is the **residual capacity** of (u,v), given by:

$$c_f(u,v) = c(u,v) - f(u,v)$$

For example, if c(u,v) = 16 and f (u,v) = 11, then we can increase f (u,v) by c_f (u,v) = 5 units before we exceed the capacity constraint on edge (u,v).

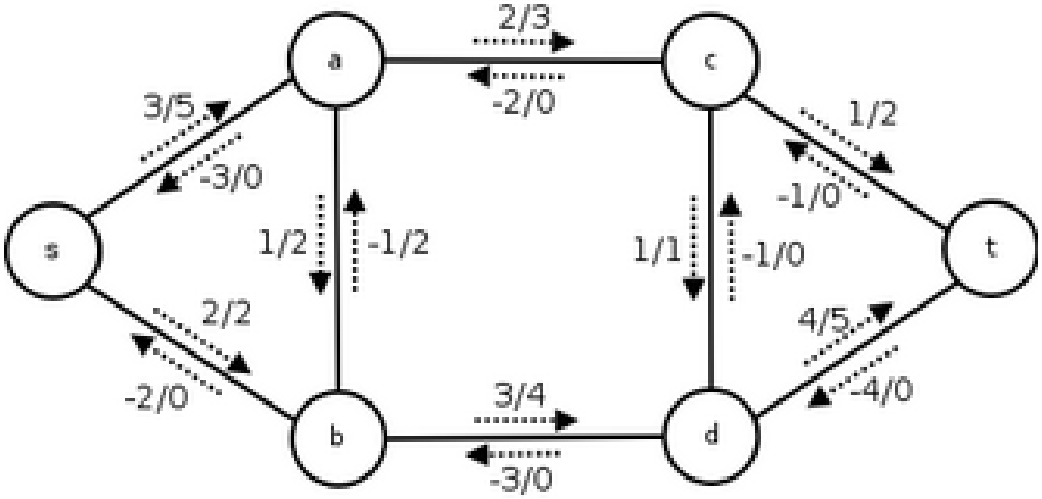


Fig: A flow network showing flow and capacity.

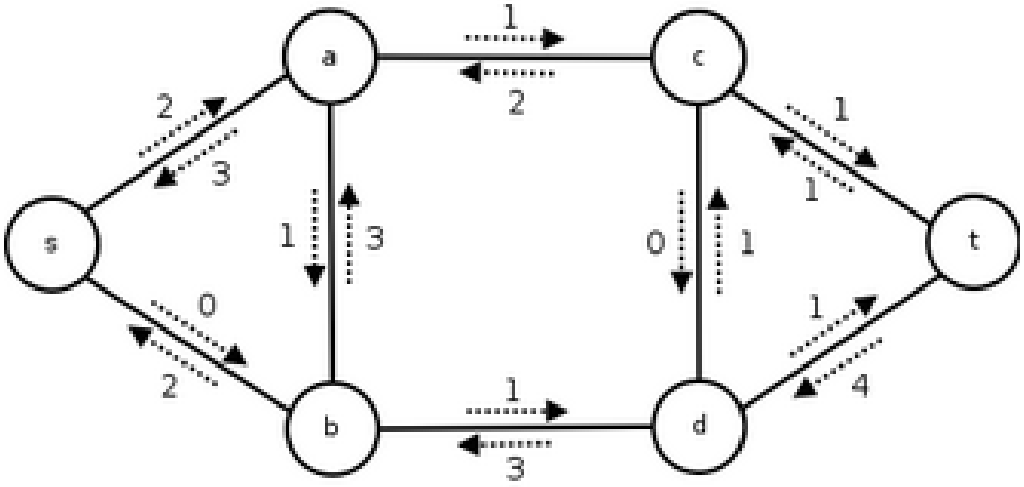
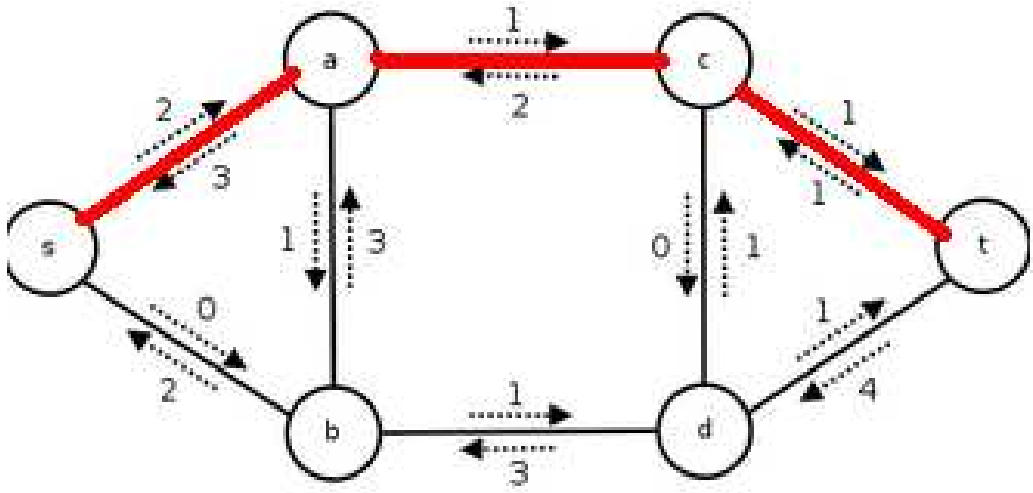


Fig: Residual network for the above flow network showing residual capacities.

Augmenting Path

Given a flow network G = (V, E) and a flow f , an augmenting path **p** is a simple path from s to t in the residual network G_f .



The red shaded path in the figure shows an augmenting path in a residual network.

There is available capacity along the paths (s,a,c,t), (s,a,b,d,t) and (s,a,b,d,c,t), which are then the augmenting paths.

The residual capacity of the first path is;

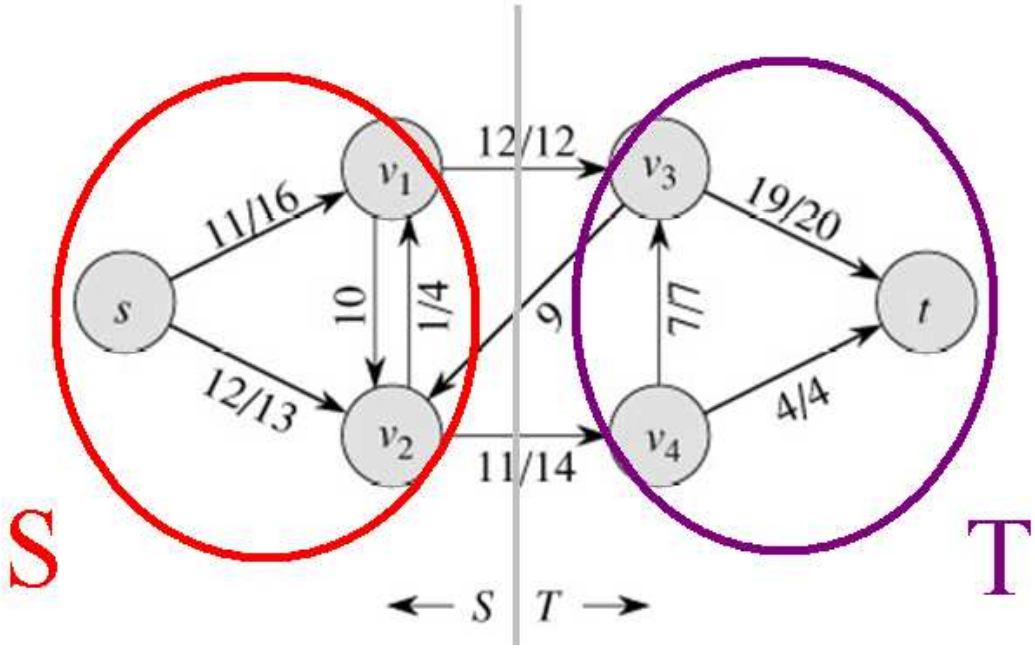
$$\min(c(s,a) - f(s,a), c(a,c) - f(a,c), c(c,t) - f(c,t))$$
$$= \min(5 - 3, 3 - 2, 2 - 1)$$
$$= \min(2, 1, 1)$$
$$= 1$$

Cut:

In graph theory, a cut is a partition of the vertices of a graph into two sets. More formally, let G(V, E) denote a graph. A cut is a partition of the vertices V into two sets S and T. Any edge (u,v) ∈ E with u ∈ S and v ∈ T (or u ∈ T and v ∈ S, in case of a directed graph) is said to be crossing the cut and is a cut edge.

In network flow, the size of a cut is defined to be the sum of weights of the edges crossing the cut from the source side to the sink side (but not the ones that go the other way).

Here we can divide the graph in the following partition so that:



$(s, v_1, v_2) \in S$

$(v_3, v_4, t) \in T$

The net flow through a cut (S,T) is define as:

$f(S,T) = \sum f(u,v) - \sum f(v,u)$, where $u \in S$ and $v \in T$

The Capacity of Cut(S,T) is define as:

$c(S,T) = \sum c(u,v)$, where $u \in S$ and $v \in T$

Naive Algorithm

```
FORD-FULKERSON-METHOD(G, s, t)
1    initialize flow f to 0
2        while there exists an augmenting path p
3            do augment flow f along p
4    return f
```

The basic idea of Ford-Fulkerson method is to find repeatedly augmented path from flow network until there exists no path.

When we have an edge u → v and have minimum flow f(u,v), the residual capacity is defined as:

$$c_f(u,v) = c(u,v) - f(u,v)$$

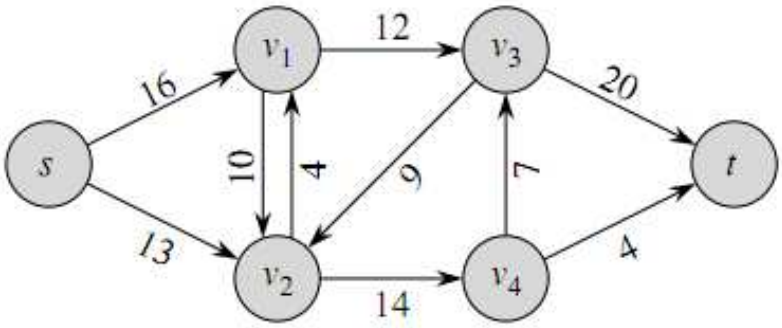
For every flow, we define a residual capacity in opposite direction (v → u) as:

$$c_f(v,u) = c(v,u) - f(u,v)$$

$$\Rightarrow c_f(v,u) = c(v,u) + f(v,u)$$ (as f(u,v)= - f(v,u) from screw symmetry property)

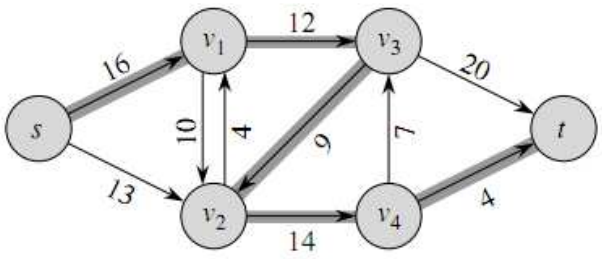
The following example with figure will clear all:

Let, we want to find the maximum flow on the following graph:

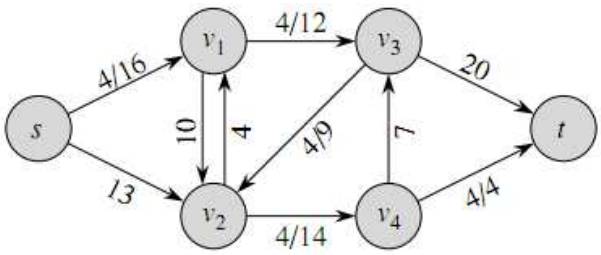


1st Step: the augmenting path **s→v1→v3→v2→v4→t** and the min-Cost from all edges in path is **min**(16,12,9,14,4)=4, thus current max-flow = 4.

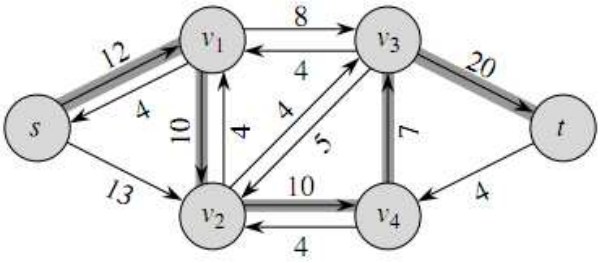
Flow Network

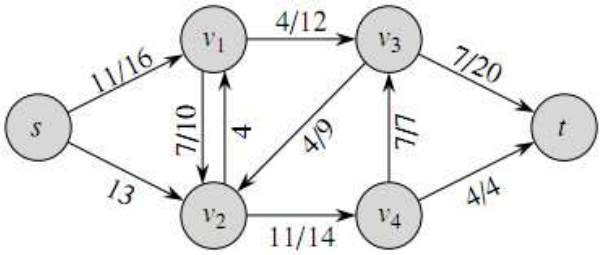


Residual Network

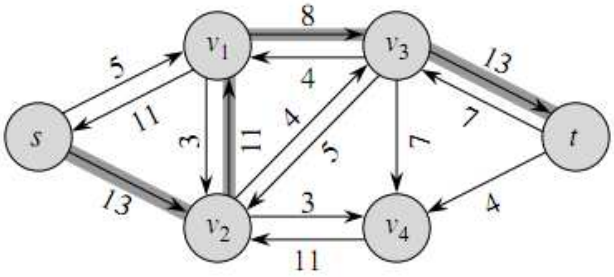


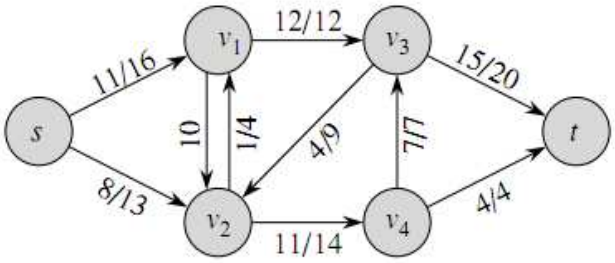
2nd Step: the augmenting path **s→v1→v2→v4→v3→t** from the residual network and have the max-flow = 4 + 7 = 11



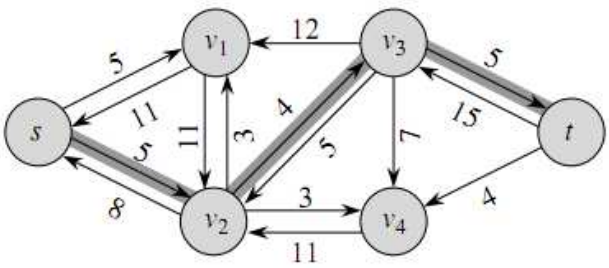


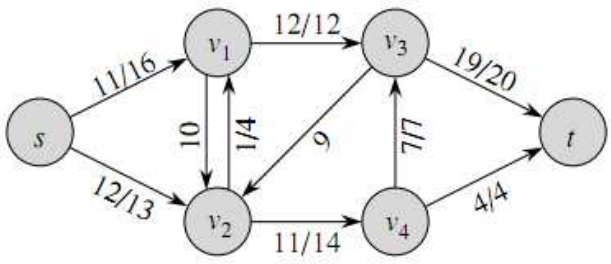
3rd Step: the augmenting path **s→v2→v1→v3→t** from the residual network and have the max-flow = 11+ 8 = 19



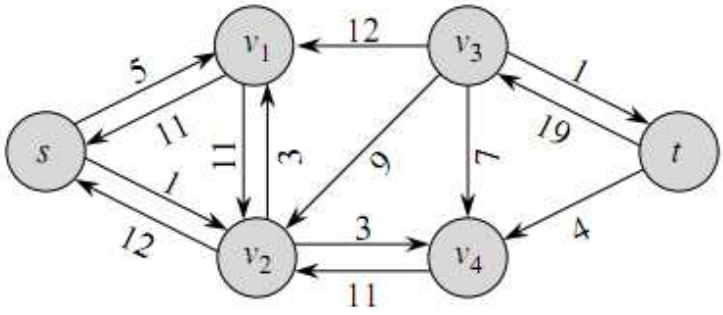


4th Step: the augmenting path **s→v2→ v3→t** from the residual network and have the max-flow = 19 + 4 = 23





This is the last situation where we can stop because there is no augmenting path in the following residual network:



So we get 23 as the maximum flow for this flow network.

The basic Ford-Fulkerson algorithm:

```
FORD-FULKERSON(G, s, t)
1  for each edge (u, v) ∈ E[G]
2       do f[u, v] ← 0
3          f[v, u] ← 0
4  while there exists a path p from s to t in the residual network G_f
5       do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
6          for each edge (u, v) in p
7               do f[u, v] ← f[u, v] + c_f(p)
8                  f[v, u] ← −f[u, v]
```

We can find path from source(s) to sink(t) by any known algorithm like, DFS, BFS etc.

A simple code using DFS for max-flow can be as follows:

// Code of Max-Flow | Using DFS

```
#define MIN(a,b)  ((a<b)?a:b)
#define INF  2147483647

int flow[101][101],prev[101],flowfound,no_of_Node;
void DFS(int source, int destination, int c)
{
	int i;
	if(source==destination)
		flowfound=c;

	if(flowfound!=0)
		return;

	for(i=1;i<=no_of_Node;i++)
	{
		if(prev[i]==0 && flow[source][i]!=0)
		{
			prev[i]=source;
			c=MIN(flow[source][i],c);	//#define MIN(a,b)  ((a<b)?a:b)
			DFS( i, destination, c );
		}
	}
}

int ford_fulkerson_using_DFS(int source, int destination)
{
	memset(prev,0,(no_of_Node+1)*sizeof(prev[0]));
	flowfound=0;
	prev[source]=-1;
	DFS( source, destination, INF );		//#define INF  2147483647
	return flowfound;
}

int maxflow(int source, int destination)
{
	int f,i,maxflow=0;

	while( (f=ford_fulkerson_using_DFS( source, destination )) != 0 )
	{
		i=destination;
		while(prev[i]!=-1)
		{
			flow[prev[i]][i] -= f;
			flow[i][prev[i]] += f;
			i=prev[i];
		}
		maxflow+=f;
	}
	return maxflow;
}

int main()
{
```

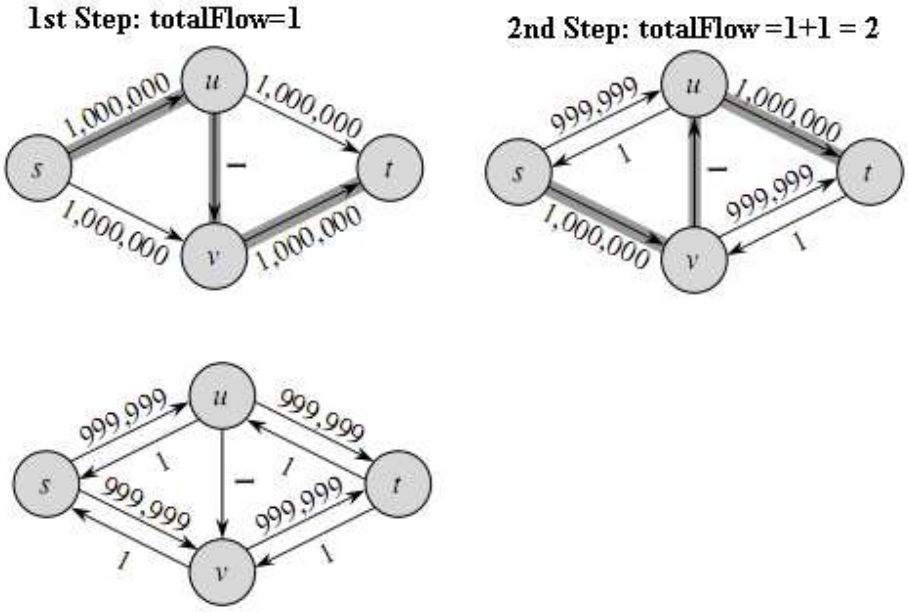
```
        //freopen("in.txt","rt",stdin);

        int source,destination,no_of_Edge,i,j,a,b,c;
                scanf("%d",&no_of_Node);
                scanf("%d %d %d",&source,&destination,&no_of_Edge);

                for(i=1;i<=no_of_Node;i++)
                        for(j=1;j<=no_of_Node;j++)
                                flow[i][j]=0;

                for(i=1;i<=no_of_Edge;i++)
                {
                        scanf("%d %d %d",&a,&b,&c);
                        flow[a][b]+=c;
                        flow[b][a]+=c;
                }
                printf("Maximum Flow is : %d\n",maxflow(source,destination) );
return 0;
}
```

The running time of FORD-FULKERSON depends on how the augmenting path p is determined. If it is chosen poorly, the algorithm might not even terminate: the value of the flow will increase with successive augmentations, but it need not even converge to the maximum flow value. See the example in following graph:



From this graph we can continue as do as 1st step and next to 2nd step. Thus we will find 2,000,000 paths(!) and total flow will = 2000000.

But if we simply choose the path **s→u→t** in 1st step and **s→v→t** in 2nd step, we get the maximum flow by 2 paths only!

Better Implementation:

The bound on FORD-FULKERSON can be improved if we implement the computation of the augmenting path p with a breadth-first search, that is, if the augmenting path is a shortest path from s to t in the residual network, where each edge has unit distance (weight). We call the Ford-Fulkerson method so implemented the Edmonds-Karp algorithm. This algorithm runs in polynomial time.

Code using BFS:

```
int bfs(int source,int destination)
{
	int u,v;
	queue<int>Q;
	memset(visited,0,sizeof(visited));
	visited[source]=1;
	pr[source]=0;
	Q.push(source);
	while(!Q.empty())
	{
		u=Q.front();
		Q.pop();
		if(u==destination) return 1;
		for(v=1;v<=V;v++)
		{
			if( (!visited[v]) && (capacity[u][v]-flow[u][v]>0) )
			{
				Q.push(v);
				pr[v]=u;
				visited[v]=1;
			}
		}
	}
	return 0;
}
int ford_fulkerson(int source,int destination)
{
	int u,increment,v,max_flow=0;
	while(bfs(source,destination))
	{
		increment=inf;
		for(v=destination;pr[v]>0;v=pr[v])
		{
			u=pr[v];
			increment=min(increment,capacity[u][v]-flow[u][v]);
		}
		for(v=destination;pr[v]>0;v=pr[v])
		{
			u=pr[v];
			flow[u][v]+=increment;
			flow[v][u]-=increment;
		}
		max_flow+=increment;
	}
	return max_flow;
}
```

Another Approach:

// this will be using priority-first search (PFS). Ref: topcoder tutorial