CPSC 500 Scribe for Nov. 6th lecture (Instructor: Prof. Will Evans)

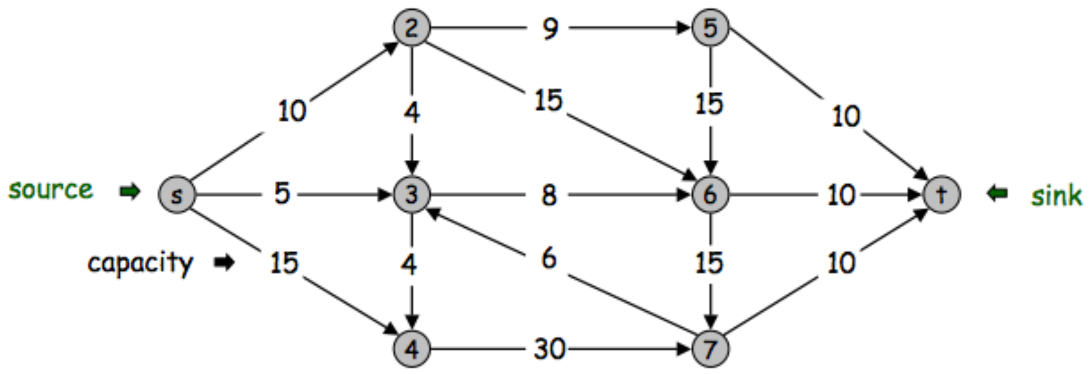
Scribe: Shuochen Su (80459134)

Overview

In this lecture, we continued on our discussion about *linear programming*, specifically *flow network* problem. After reviewing the *Ford-Fulkerson Algorithm*, which is a simple algorithm for computing *max flow* in a *flow network*, the drawback of this algorithm was mentioned. For proving its correctness, we introduced the definition of *cut*, and several lemmas and theorems. Furthermore, it can be proved that the value of *max flow* equals capacity of *min cut*, which is a special case of *duality theorem* for linear programs. In the end, two examples were introduced just to give us some illustrations on how certain problems that seemed complicated before could be solved now easily by converting them to max flow problems.

Review of Max Flow problem

Flow network is a directed graph $G=(V,E)$ in which each edge $(u,v)\in E$ has a positive capacity $c(u,v)$. There is a source vertex $s\in V$ and a sink vertex $t\in V$.



A *flow* is an assignment f of real numbers to edges of G, with the following constraints:

Capacity constraint: $f(u,v)\le c(u,v)$

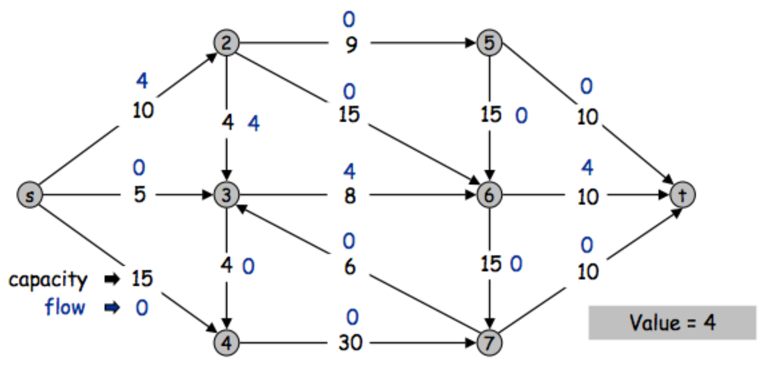
Flow conservation: $\sum_{(u,v)\in E, u\in V} f(u,v) = \sum_{(v,w)\in E, w\in V} f(v,w)$ for all $v\in V, v\neq s,t$

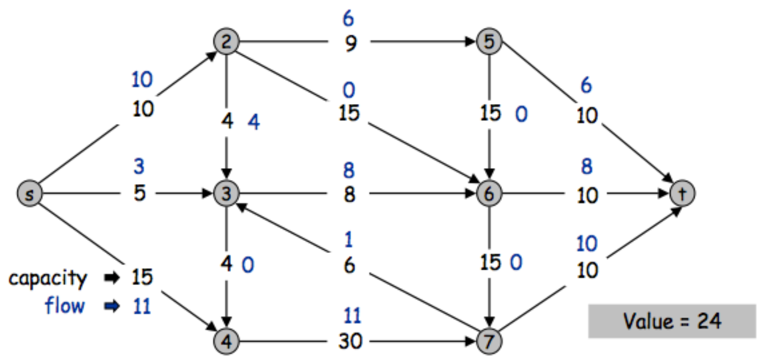
Like all *linear programming* problem, we also have an *objective function*:

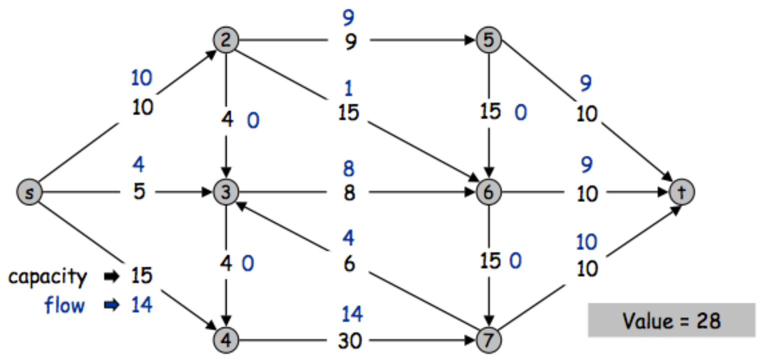
$$size(f)=\sum_{(s,v)\in E} f(s,v)-\sum_{(v,s)\in E} f(v,s)$$

which is essentially the total amount of flow that exits the source or the total amount of flow that enters the sink, according to flow conservation.

Below shows three example flows and their size:







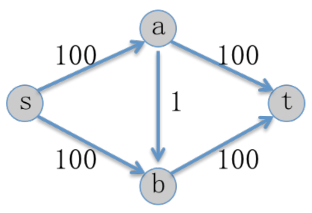
The **goal** of max flow problem is to find flow with maximum size.

Review of Ford-Fulkerson Algorithm

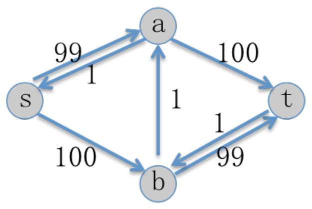
1. Start with zero flow
2. Choose "augmenting path" from *s* to *t* in "residual network"
3. Increase flow along edges of this path up to capacity of path
4. Repeat 2 and 3 until no path exist from *s* to *t*

Drawback of Ford-Fulkerson Algorithm

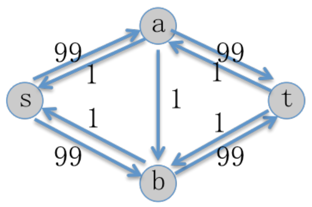
Consider the following network:



If we choose path s->a->b->t, we get



Then if we choose path s->b->a->t, we get



If "unfortunately" we keep choosing these two "bad" paths, 200 steps ($O(10^3)$, 3 is the number of digits in max capacity 100) are needed for finding the max flow, making Ford-Fulkerson an exponential algorithm. If we choose the shortest augmenting path then the Ford-Fulkerson algorithm runs in time $O(VE^2)$.

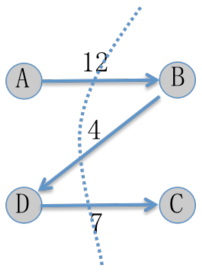
Correctness of Ford-Fulkerson Algorithm

For proving the correctness, first we need to introduce cut and its capacity.

Definition of cut: A cut is a partition (*S*, *T*) of *V* in a flow network such that $s \in S$ and $t \in T$. In other word, a cut separates *s* from *t*.

The capacity of cut (*S*, *T*) is $cap(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$

For example, the capacity of the following cut is 12+7=19



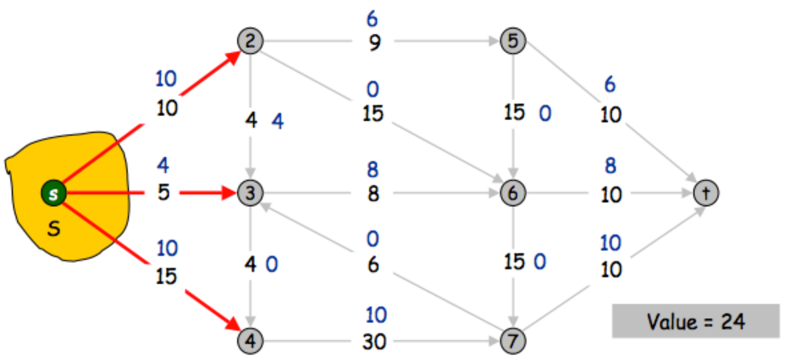
The importance of cut is that each cut bound the amount of flow from *S* to *T*. The following lemma formally states this observation:

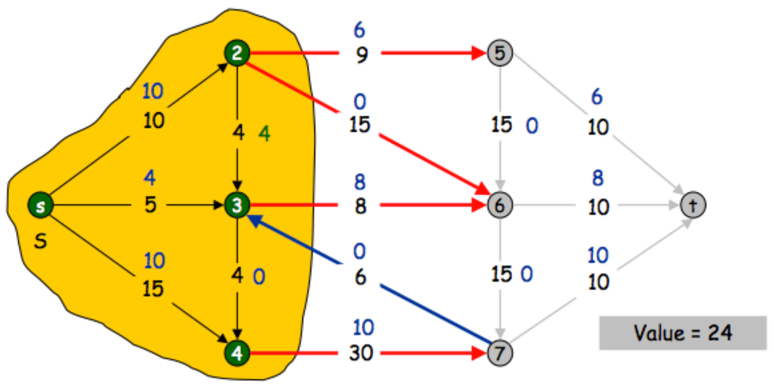
Lemma: For any flow *f* and any cut (*S*, *T*), $size(f) \le cap(S,T)$

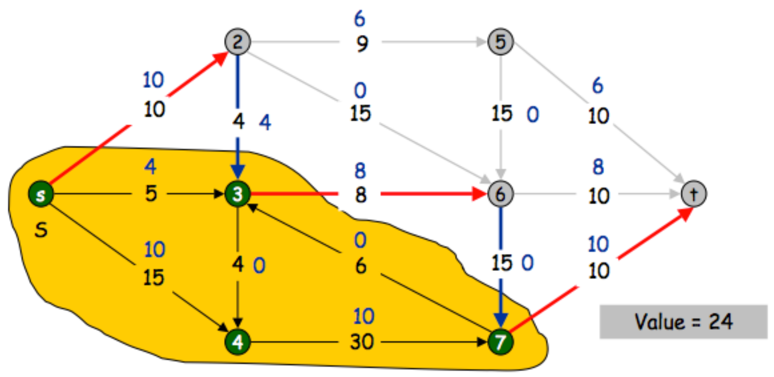
This can be proved by introducing flow across cut (or net flow across cut):

Definition of flow across cut: $\sum_{a \in S} \sum_{b \in T} [f(a,b) - f(b,a)]$

Observation 1: Let *f* be a flow, and let (*S*, *T*) be **any** cut (*S*, *T*). Then, the net flow sent across the cut is equal to the amount reaching *t*, which is the size of flow.







and this can be proved using flow conservation easily (consider after adding another vertex into *S*, which edges are introduced and which edges are removed when calculating the flow across the new cut).

Observation 2: Let *f* be a flow, and let (*S*, *T*) be **any** cut (*S*, *T*). Then, the net flow sent across the cut is at most the capacity of the cut.

This is easy to derive, because when calculating the capacity of cut we only consider those "forward" edges from *S* to *T*, while for calculating the net flow, we also need to minus those "backward" edges.

Based on the above two observations, the Lemma stands.

After this preparation, we can formally prove the correctness of Ford-Fulkerson algorithm, which is the following theorem.

Theorem: If residual network G^f has no augmenting path then *f* is max size flow.

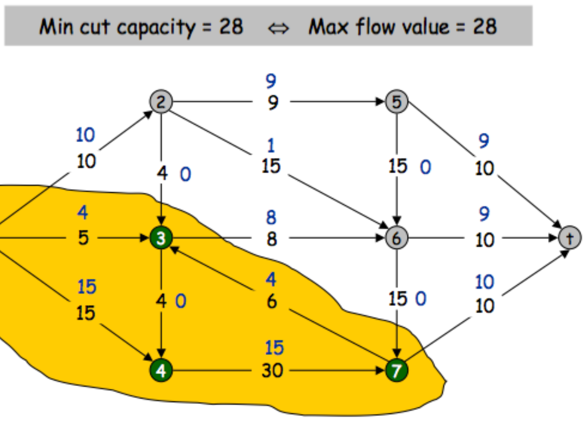
Proof:

Let *S* represents the set of nodes that can reach from *s* in G^f

It is obvious that $t \notin S$ because G^f has no augmenting path
Let $T = V$-S, so that (S, T) is cut.
The size of flow = $f(S, T) = c(S, T)$ because $f(u, v) = c(u, v)$ for all $u \in S$ and $v \in T$.
According to the Lemma, we have
The size of any flow <= $c(S, T)$ = size(f)
With that, we proved that f is max size flow. So Ford-Fulkerson Algorithm is correct.

Max Flow/Min Cut Theorem

This theorem can be described as:
size of max flow = capacity of min cut



The proof is as follows:
Proof:
According to Lemma,

size of $f^* \le c(S^*, T^*)$,

in which f^* is the max flow, while (S^*, T^*) is the min capacity cut.

Let (S, T) be the cut in the previous proof of theorem for telling the correctness of Ford-Fulkerson Algorithm. We have

size of $f^* = c(S, T)$,

We also have

$c(S, T) \ge c(S^*, T^*)$

because (S^*, T^*) is min capacity cut.

In conclusion, we get

size of $f^* = c(S^*, T^*)$

Examples

1. Bipartite Matching

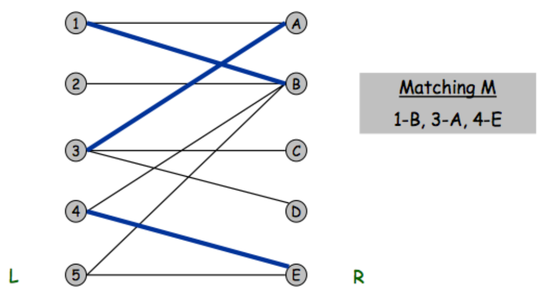
Bipartite graph is the graph that has partition of vertices into two sets such that all edges go between sets.

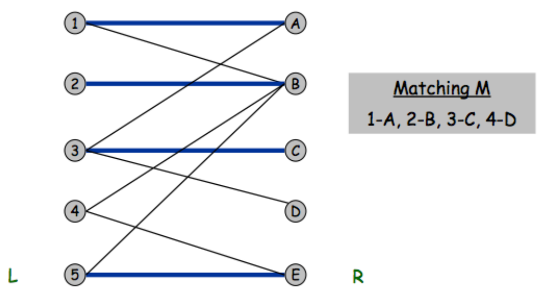
The Bipartite matching problem can be described as:

Given: Undirected Bipartite graph $G = (V, E)$ with partitions L and R

Find: Maximum matching in G, that is the maximum number of subset edges such that no two edges in subset share endpoint.

Here are two examples of Bipartite matching:





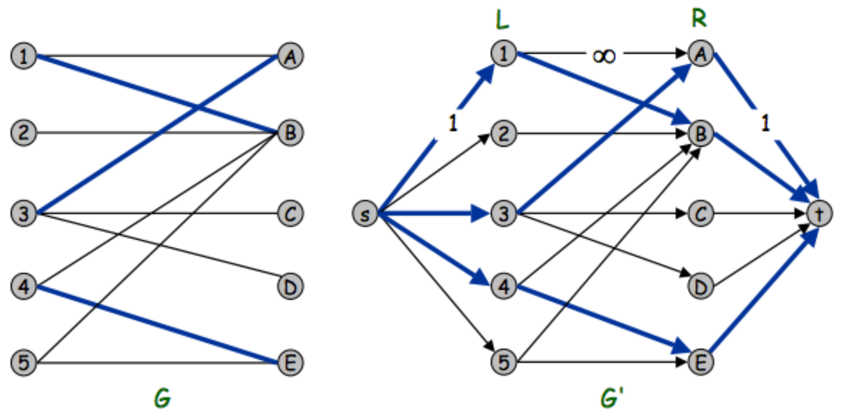
For solving this, we can reduce it to a max flow problem by adding source and sink and assign capacities. The steps are as follows:

1) Create a directed graph $G^{'}$

2) Direct all edges from L to R, and give infinite capacity

3) Add source s, and unit capacity edges from s to each vertex in L

4) Add sink t, and unit capacity edges from each vertex from R to t



Now the original Bipartite matching problem reduces to max flow problem and can be solved using Ford-Fulkerson algorithm.

2. Pennant Race Problem

Given:

a team A,

a list of other teams T1T2…Tn,

win/loss record for each team, and

list of remaining games to play

Determine:

Is it possible for team A to win at least as many games as each other team by the end of the season
We will show during next lecture that this problem can also be reduced to max flow problem.

Summary
In this lecture, max flow/min cut problem was introduced, which is a classic topic of CS algorithm courses. We will continue with the remaining two examples during the next lecture.

References

Wikipedia page for max flow algorithm:
http://en.wikipedia.org/wiki/Maximum_flow_problem

Wikipedia page for Ford-Fulkerson algorithm:
http://en.wikipedia.org/wiki/Ford–Fulkerson_algorithm

Princeton University COS226 (Spring '04) online lecture slides:
http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf