COL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Network Flow

Algorithm

`Ford-Fulkerson`

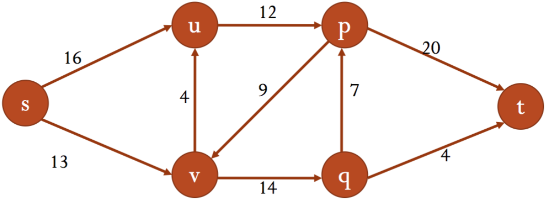
- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)

- What is the running time of the above algorithm? $O(m \cdot C)$
  - Claim 2: $v(f') > v(f)$.
  - Claim 3: The while loop runs for $C = \sum_{e \text{ out of } s} c(e)$ iterations.
  - Claim 4: Finding augmenting path and augmenting flow along this path takes $O(m)$ time.

Algorithm

`Ford-Fulkerson`

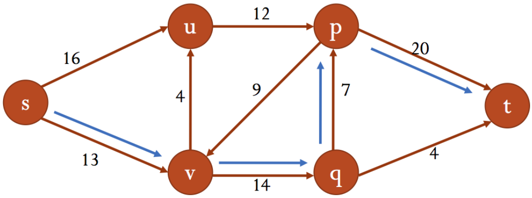
- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)



Algorithm

`Ford-Fulkerson`

- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)



Algorithm

`Ford-Fulkerson`

- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)

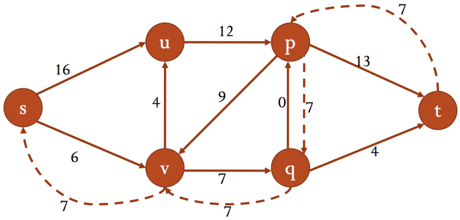
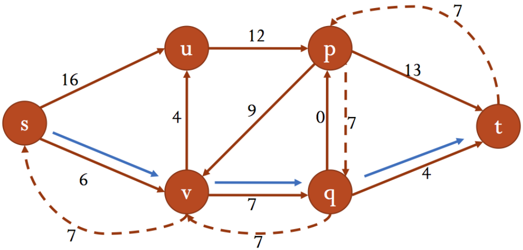


Figure: Graph G_f, where $f(s, u) = 0, f(s, v) = 7, f(v, u) = 0, f(v, q) = 7, f(u, p) = 0, f(p, v) = 0, f(p, t) = 7, f(q, p) = 7, f(q, t) = 0$

Algorithm

`Ford-Fulkerson`

- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)



Algorithm

`Ford-Fulkerson`

- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)

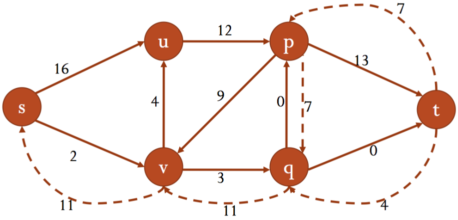
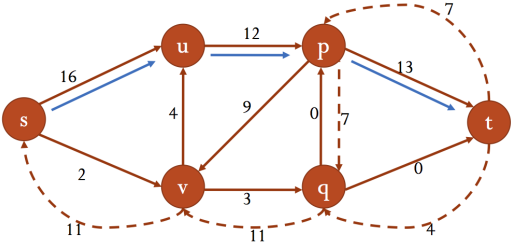


Figure: Graph G_f, where $f(s, u) = 0, f(s, v) = 11, f(v, u) = 0, f(v, q) = 11, f(u, p) = 0, f(p, v) = 0, f(p, t) = 7, f(q, p) = 7, f(q, t) = 4$

Algorithm

`Ford-Fulkerson`

- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)



Algorithm

`Ford-Fulkerson`

- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)

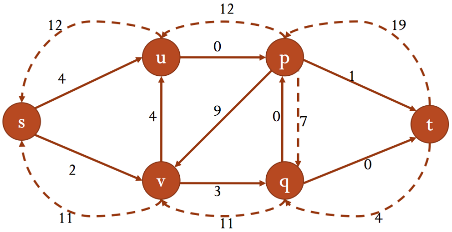


Figure: Graph G_f, where $f(s,u) = 12, f(s,v) = 11, f(v,u) = 0, f(v,q) = 11, f(u,p) = 12, f(p,v) = 0, f(p,t) = 19, f(q,p) = 7, f(q,t) = 4$

Algorithm

`Ford-Fulkerson`

- Start with a flow f such that $f(e) = 0$
- While there is an $s - t$ path P in G_f
  - Augment flow along an $s - t$ path and let f' be resulting flow
  - Update f to f' and G_f to $G_{f'}$
- return(f)

- How do we prove that the flow returned by the `Ford-Fulkerson` algorithm is the maximum flow?

- <u>Theorem 1</u>: Let f be the flow returned by the `Ford-Fulkerson` algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

Definition (f^{in} and f^{out})

Let S be a subset of vertices and f be a flow. Then

$$f^{in}(S) = \sum_{e \text{ into } S} f(e) \quad \text{and} \quad f^{out}(S) = \sum_{e \text{ out of } S} f(e)$$

- Theorem 1: Let f be the flow returned by the `Ford-Fulkerson` algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

Definition (f^{in} and f^{out})

Let S be a subset of vertices and f be a flow. Then

$$f^{in}(S) = \sum_{e \text{ into } S} f(e) \quad \text{and} \quad f^{out}(S) = \sum_{e \text{ out of } S} f(e)$$

Definition ($s - t$ cut)

A partition of vertices (A, B) is called an $s - t$ cut iff A contains s and B contains t.

Definition (Capacity of $s - t$ cut)

The capacity of an $s - t$ cut (A, B) is defined as $C(A, B) = \sum_{e \text{ out of } A} c(e)$.

- Theorem 1: Let f be the flow returned by the `Ford-Fulkerson` algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

Proof

- Claim 1.1: For any $s-t$ cut (A, B) and any $s-t$ flow f, $v(f) = f^{out}(A) - f^{in}(A)$.

- Theorem 1: Let f be the flow returned by the `Ford-Fulkerson` algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

Proof

- Claim 1.1: For any $s-t$ cut (A, B) and any $s-t$ flow f, $v(f) = f^{out}(A) - f^{in}(A)$.

Proof of claim 1.1.

$v(f) = f^{out}(\{s\}) - f^{in}(\{s\})$ and for all other nodes $v \in A$, $f^{out}(\{v\}) - f^{in}(\{v\}) = 0$. So,

$$v(f) = \sum_{v \in A} (f^{out}(\{v\}) - f^{in}(\{v\})) = f^{out}(A) - f^{in}(A).$$

$\square$

- Theorem 1: Let f be the flow returned by the `Ford-Fulkerson` algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

Proof

- Claim 1.1: For any s-t cut (A, B) and any s-t flow f, $v(f) = f^{out}(A) - f^{in}(A)$.
- Claim 1.2: Let f be any s-t flow and (A, B) be any s-t cut. Then $v(f) \leq C(A, B)$.

- Theorem 1: Let f be the flow returned by the `Ford-Fulkerson` algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

Proof

- Claim 1.1: For any s-t cut (A, B) and any s-t flow f, $v(f) = f^{out}(A) - f^{in}(A)$.
- Claim 1.2: Let f be any s-t flow and (A, B) be any s-t cut. Then $v(f) \leq C(A, B)$.

Proof of claim 1.2.

$v(f) = f^{out}(A) - f^{in}(A) \leq f^{out}(A) \leq C(A, B)$. □

- Theorem 1: Let f be the flow returned by the `Ford-Fulkerson` algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

Proof

- Claim 1.1: For any s-t cut (A, B) and any s-t flow f, $v(f) = f^{out}(A) - f^{in}(A)$.
- Claim 1.2: Let f be any s-t flow and (A, B) be any s-t cut. Then $v(f) \leq C(A, B)$.
- Claim 1.3: Let f be an s-t flow such that there is no s-t path in G_f. Then there is an s-t cut (A^*, B^*) such that $v(f) = C(A^*, B^*)$. Furthermore, f is a flow with maximum value and (A^*, B^*) is an s-t cut with minimum capacity.

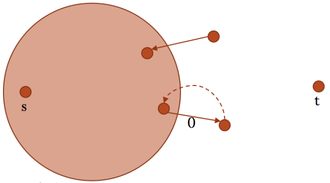
$\square$

- Claim 1.3: Let f be an s-t flow such that there is no s-t path in G_f. Then there is an s-t cut (A^*, B^*) such that $v(f) = C(A^*, B^*)$. Furthermore, f is a flow with maximum value and (A^*, B^*) is an s-t cut with minimum capacity.

Proof of claim 1.3

- Let A^* be all vertices reachable from s in the graph G_f (see figure below). Then we have:

$$\begin{aligned} v(f) &= f^{out}(A^*) - f^{in}(A^*) \\ &= f^{out}(A^*) - 0 \\ &= C(A^*, B^*) \end{aligned}$$



A^* (all vertices reachable from s in G_f)

Theorem (Max-flow-min-cut theorem)

In every flow network, the maximum value of s-t flow is equal to the minimum capacity of s-t cut.

- Summary:
  - `Ford-Fulkerson` Algorithm:
    - Given network with integer capacities, find a source-to-sink path and push as much flow along the path as possible.
    - Update the residual capacity of edges in the residual graph.
    - Repeat.
  - Proof of correctness:
    - The algorithm terminates (since the capacities are integers).
    - Max-flow-min-cut theorem: In every flow network, the maximum value of s-t flow is equal to the minimum capacity of s-t cut.

Applications of Network Flow

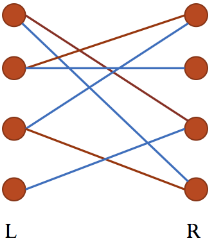
Definition (Matching in bipartite graphs)

A subset M of edges such that each node appears in at most one edge in M.

Problem

Given a bipartite graph $G = (L, R, E)$, design an algorithm to give a maximum matching in the graph.

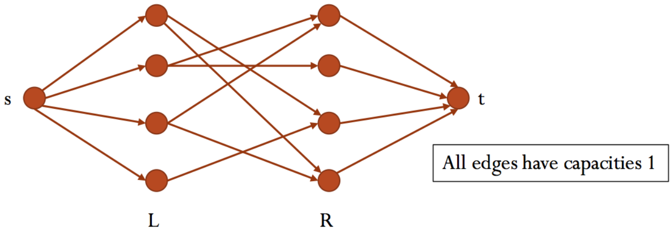
- Example:



Problem

Given a bipartite graph $G = (L, R, E)$, design an algorithm to give a maximum matching in the graph.

- Consider the network graph below constructed from the bipartite graph.

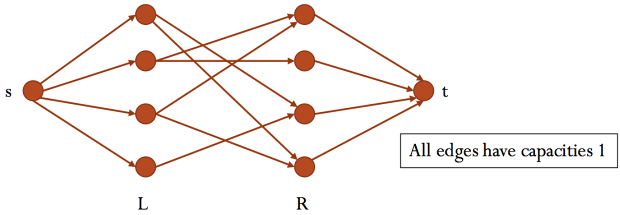


- Claim 1: Suppose there is an integer flow of value k in the network graph. Then the bipartite graph has a matching of size k.

Problem

Given a bipartite graph $G = (L, R, E)$, design an algorithm to give a maximum matching in the graph.

- Consider the network graph below constructed from the bipartite graph.



- Claim 1: Suppose there is an integer flow of value k in the network graph. Then the bipartite graph has a matching of size k.
  Claim 2: Suppose the bipartite graph has a matching of size k. Then there is an integer flow of value k in the network graph.

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 10
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - Daredevils Vs Sunrisers, Sunrisers Vs Lions, Daredevils Vs Lions, Sunrisers Vs Daredevils

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 10
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - Daredevils Vs Sunrisers, Sunrisers Vs Lions, Daredevils Vs Lions, Sunrisers Vs Daredevils
- A team is said to be eliminated if it cannot end with maximum number of wins.
- Can we say that Supergiants have been eliminated give the current scenario?

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 9
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - 4 games between Daredevils and Sunrisers.
- Can we say that Supergiants have been eliminated give the current scenario?

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

End