Network Flow

- Maximum Flow: Problem Formulation
- Maximum Flow: Upper Bound
- Maximum Flow: Adding flow along paths
- Residual Network and Augmenting Path
- Ford-Fulkerson Algorithm – Max-Flow-Min-Cut Theorem
- Edmond-Karp Algorithm
- Maximum Flow: Variants and Applications

Imdad ullah Khan

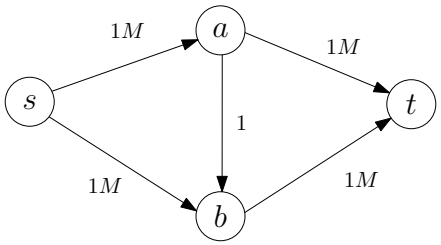
The Ford-Fulkerson Algorithm - Slow Example

The Ford-Fulkerson algorithm can be implemented in $O(mC_s)$ time, where $C_s = c([\{s\}, \overline{\{s\}}])$

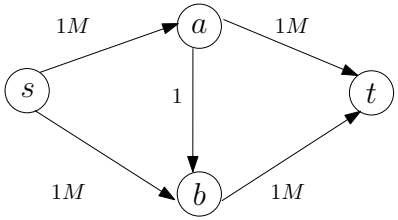
A tighter upper bound on runtime is $O(m \cdot C^*) = O(m \cdot size(f))$ where C^* is capacity of min cut

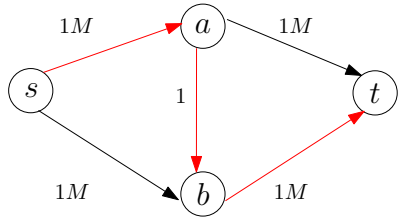
- The size of max flow f can be arbitrarily large
- not size of the input
- May be this upper bound is very loose upper (never met)

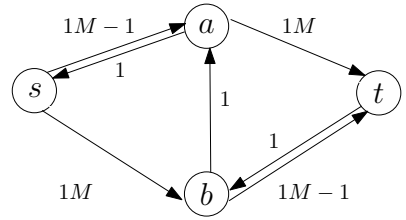
The Ford-Fulkerson Algorithm - Slow Example

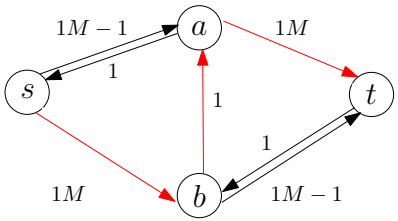


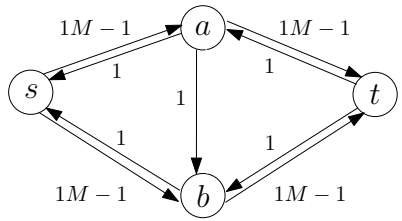
Flow Network

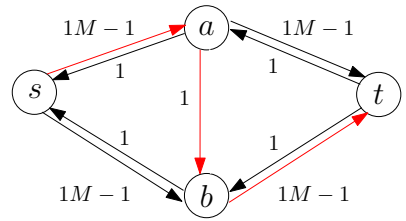


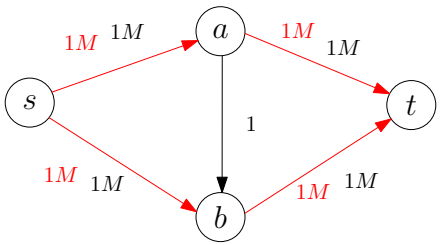


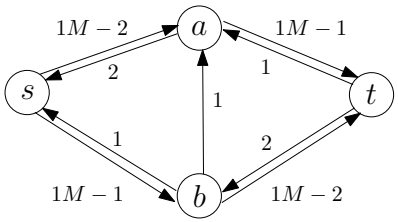


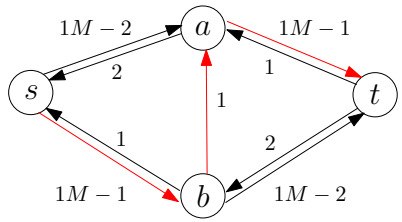


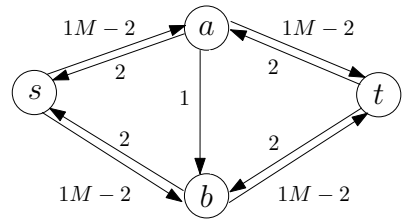












Max Flow of size $2M$

- Could take $2M$ iterations, augmenting 1 unit of flow
- No restriction on the choice of augmenting path
- Choose the augmenting path wisely to fix the problem

Max Flow : The Edmond-Karp Algorithm

Fix to Ford-Fulkerson algorithm: Choose shortest augmenting paths

Given a flow network G with source s and t

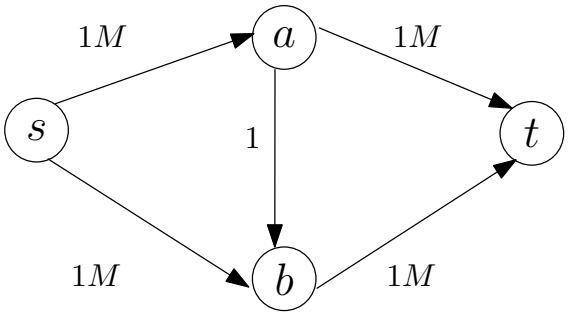
Algorithm Edmond-Karp Algorithm

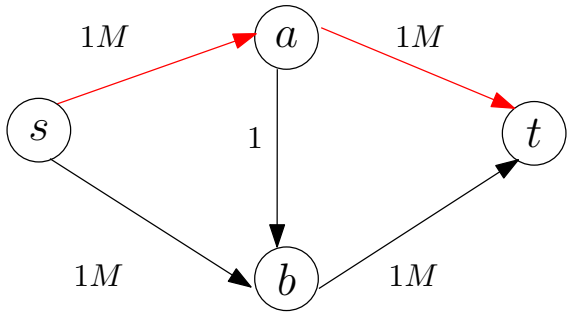
```
f ← 0                          ▷ Initialize to a (valid) flow of size 0 (on every edge)
while TRUE do
    Compute G_f
    Find a shortest s − t path P in G_f                          ▷ Using BFS
    if no such path then
        return f
    else
        f ← AUGMENT(P, f)
```

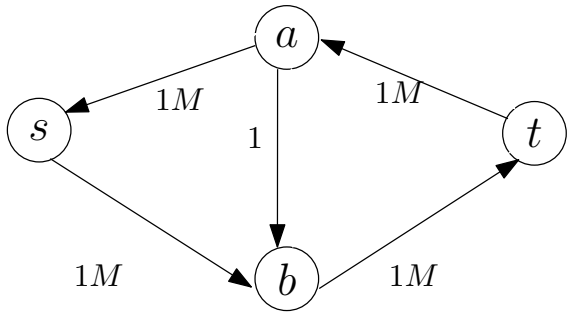
- Correctness and optimality follow immediately from the above proof, as that works for any augmenting path
- Need to analyze running time

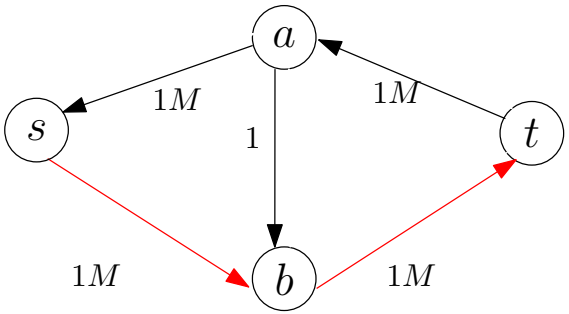
Max Flow : The Edmond-Karp Algorithm

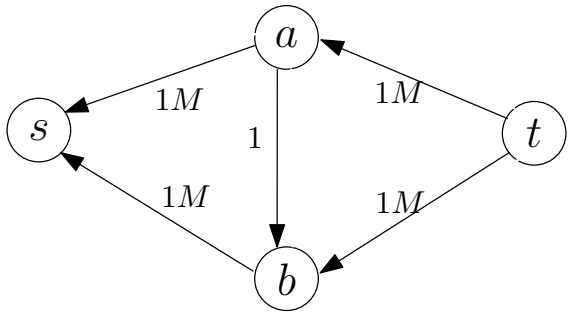
Running the Edmond-Karp algorithm on the pathalogical example

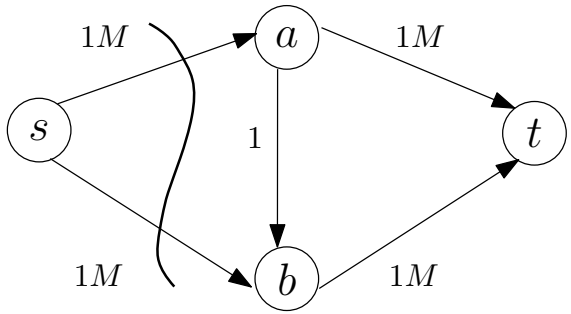




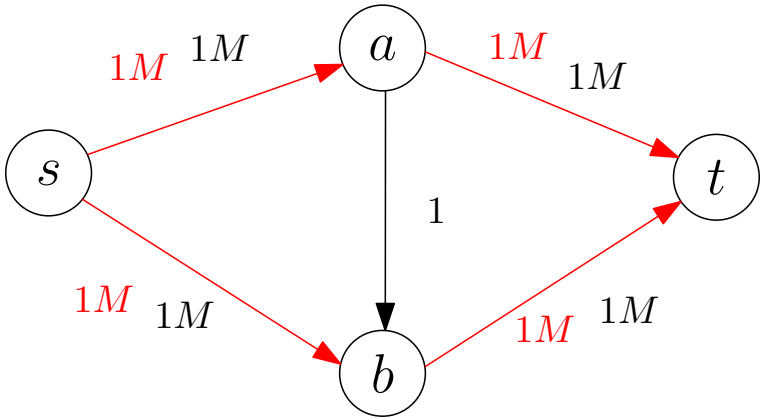








The Edmond-Karp Algorithm: Analysis



- Each $\textsc{augment}(P, f)$ saturates at least one edge on P
- The saturated edge is unavailable for the next iteration (possibly many iterations), at least in that direction

The key is to prove that an edge is not saturated too many times

▷ Main problem in the pathological example, the edge ab was saturated too many times

The Edmond-Karp Algorithm: Increasing Distances

After each iteration of the Edmonds-Karp algorithm, $d_{G_f}(s, v)$ (hop-length in G_f from s to v) does not decrease for all vertices v.

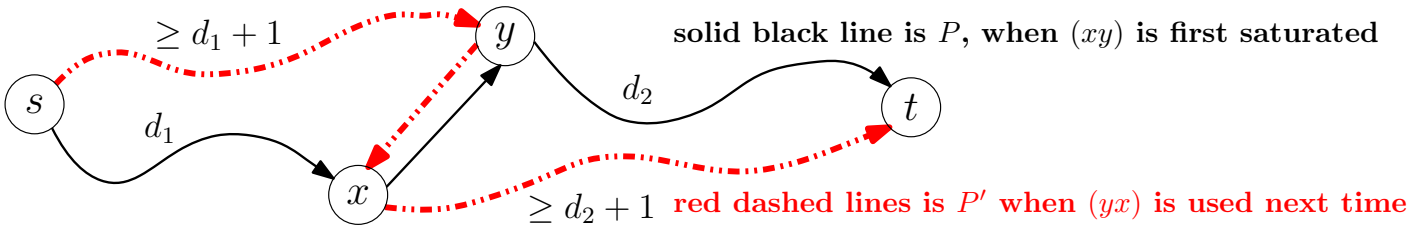
Same is true for $d_{G_f}(v, t)$ and hence $d_{G_f}(s, t)$.

Suppose the edge xy is saturated in G_f and f' is the augmented flow

- When xy is saturated we remove it from $G_{f'}$ ▷ edge yx is there
- $d_{G_f}(s, y)$ is greater than $d_{G_f}(s, x)$ ▷ sup-path optimality
  - if there was a shorter $s - y$ path we will use that to go to t
- In $G_{f'}$, the edge xy is not present, there cannot be a shorter path from s to v (a backward edge cannot decrease distance, as it is from a vertex farther away from s (e.g. y) to a vertex closer to s (e.g. x))
- All edges ever added are from vertices farther from s to vertices closer to s (we only introduce reverse edges along a shortest path)

The Edmond-Karp Algorithm: Edge Reuse

When edge $e = xy$ is saturated it is not reused until $d_{G_f}(s, t)$ strictly increases



- Suppose $e = xy$ is saturated when augmenting $P = s, \ldots, x, y, \ldots, t$
- P is a shortest $s - t$ path, $\ell(P) = d(s, t)$
- Let $d_1 = \ell(P[s - x])$ and $d_2 = \ell(P[y - t])$ $\triangleright$ $\ell(P) = d_1 + 1 + d_2$
- Note that in this G_f $d(s, y) = d_1 + 1$
- Suppose yx is used in some $G_{f'}$ (since xy doesn't exist any more, for xy to be used again, yx must be used before it)
- By the previous lemma $d_{G_{f'}}(s, y) \geq d_1 + 1$ and $d_{G_{f'}}(x, t) \geq d_2 + 1$
- If a path in $G_{f'}$ uses yx, since it is a shortest path, $d_{G_{f'}}(s, t) \geq d_1 + 1 + 1 + d_2 + 1 \geq d_1 + d_2 + 3 \geq \ell(P) + 2$

Hence to use xy, distance from s to t must have increased $\square$

The Edmond-Karp Algorithm: Edge Reuse

The Edmond-Karp algorithm takes $O(nm^2)$ time

- Distance from s to t can only increase $O(n)$ times ▷ why?
- For a fixed value $d(s,t)$, an edge xy can be saturated only once
- After xy is saturated once, if the distance does not increase, then the algorithm can saturate other edges but not xy
- Hence for this fixed value of distance there can be at most $O(m)$ iterations
  ▷ each iteration saturates some edge
- After which the distance must increase, and for this second fixed value again there could be $O(m)$ iterations after which the distance must increase
- Number of different values of $d(s,t)$ is $O(n)$ so there could be $O(nm)$ iterations
- Each iteration takes $O(n+m)$ (BFS). Hence total runtime is $O(nm^2)$
  ▷ could be $O(n^5)$ for dense graphs. But is polynomial in size of input

See what happens if all capacities are 0's or all are 1's

Max Flow : Better Algorithms

- There are many better algorithms (please read about them)
- Some algorithms are directly based on the Ford-Fulkerson algorithm
  - ▷ (like Edmond-Karp algorithm)
- Especially, read the push-relabel algorithm in your textbook
- You are also encouraged to read about the scaling algorithm. This is a very useful trick
- The state of the art algorithm for Max-Flow algorithm is $O(nm)$