PAGE NO.
DATE / /

AOA week 9 Notes

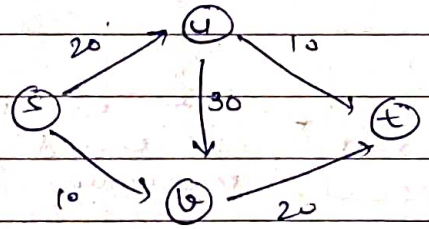
Network Flow:

Many types of problems can be solved from network flow problem.

Flow network is a directed graph $G = (V, E)$ with following features:

- Associated with each edge 'e' is a capacity, which is a nonnegative number that we denote c_e
- There is a single source $s \in V$
- There is a single sink $t \in V$

Nodes other than s and t are internal nodes.



Flow: s-t flow is a function f that maps each edge e to a nonnegative real number

$$f: E \longrightarrow R^+$$

$f(e)$: represents the ~~value~~ amount of flow carried by edge e.

f satisfies two properties:

(i) Capacity condition

For each $e \in E$, we have $0 \leq f(e) \leq c_e$

(ii) Conservation condition.

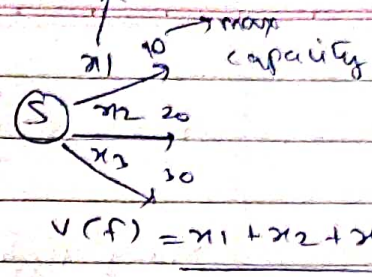
For each node v other than s and t, we have

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Carrying Flow.

value of Flow $v(f)$

$$v(f) = \sum_{e \text{ out of } s} f(e)$$



For making eqn's compact.

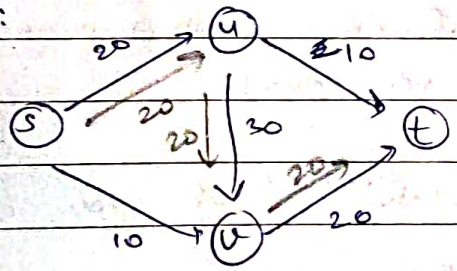
$$f^{out}(v) = \sum_{e \text{ out of } v} f(e)$$

$$f^{in}(v) = \sum_{e \text{ in } v} f(e)$$

Given a flow ntwk, find a flow of maximum possible value.

Maximum flow minimum cut. — Cut of the graph puts a bound on maximum flow value.

Eg:



Flow = 20

we cannot pass anymore flow

But by looking at the figure we can say max flow that can be achieved is 30.

So there must be a way to undo some flow in case if we get stuck to move forward in the algorithm.

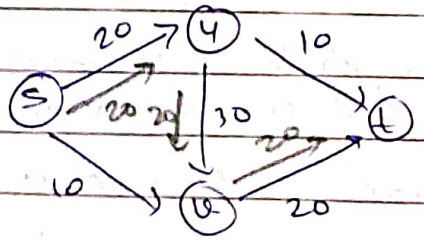
So we undo 10 units of flow on (v, u)
finally pushing 10 units from (v, t)
Hence increasing the flow to 30.

(*) we can push Forward on the edges with leftover capacity
we can push backward on the edges that are already carrying flow, to divert it in a different direction.

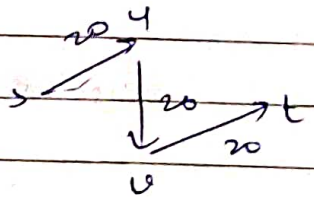
(*) Residual Graph.
Has same no. of nodes as that of G
For a given flow f on G

{ edge $e = (u, v)$ of G on which $f(e) < c_e$ there are $c_e - f(e)$ "leftover" units which we could try pushing forward. }

{ There are $f(e)$ units of flow that can be undone, by pushing flow backward. So we include $e' = (v, u)$ in G_f with capacity $f(e)$ }

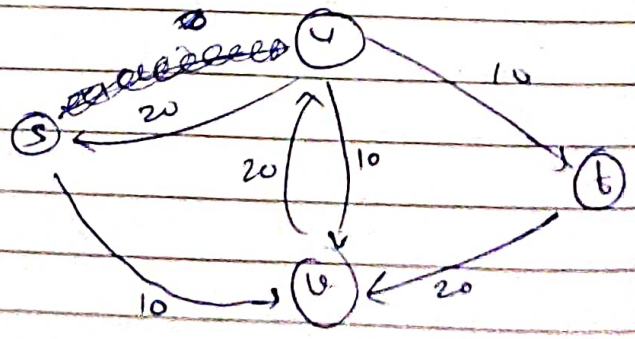


(G)



(f)

(G_f)



PAGE NO.
DATE / /

Thus G_f can have atmost twice as many edges as G.

Augmenting paths in a Residual Graph.

let P be a simple s-t path.

↳ no cycles
• visits any node only once

bottleneck(P, f) = minimum residual capacity of any edge of P, with respect to flow f.

augment (f, P) → yield new flow f' in G

```
augment(f, P)
    b = bottleneck(P, f)
    For each edge (u, v) ∈ P:
        if e = (u, v) is Forward edge :
            increase f(e) in G by b.

        else (u, v) is backward edge,
            and let e = (v, u) :
            decrease f(e) in G by b.

    return (f).
```

PAGE No.
DATE / /

(7.1) TRUE

IF F is legal flow F' is also legal flow.?

F' should follow 2 conditions

(i) Capacity condition.

If $e = (u, v)$ is fwd edge with residual capacity $(e - f(e))$

$$0 \le f(e) \le f'(e)$$

$$f'(e) = f(e) + \text{bottleneck}(P, f) \quad \text{known.}$$

$$0 \le f(e) \le f(e) + \text{bottleneck}(P, f)$$

$$\text{bottleneck}(P, f) = c(e) - f(e)$$

$$0 \le f(e) \le f(e) + c(e) - f(e)$$

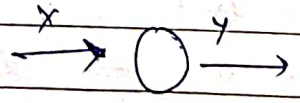
$$0 \le f(e) \le c(e) \quad \text{always True}$$

same can be proved for backward edge.

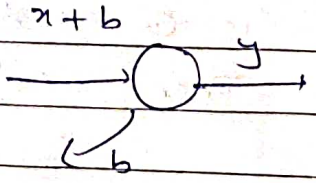
(ii) Conservation condition.

If f followed conservation condition

then in f' we increase fwd edge by b and dec back edge by b



How F
if $x = y$ (given)



F'

$x + b = y + b$

$x = y$ (given)

Hence F' is also conserved.

PAGE No.
DATE / /

(*) FORD FULKERSON ALGO

Max Flow.

Initially $f(e) = 0$ For all e in G

While there is an s-t path in residual graph G_f:

Let P be simple s-t path in G_f

$f' =$ augmentation (f, P)

update f to f'

Updat residual graph G_f to $G_{f'}$

Endwhile

return f

(7.2) At every intermediate stage of the Ford Fulkerson algo, the Flow values $\{f(e)\}$ and residual capacities in G_f are integers.

Obvious

Starting with int

$b \Rightarrow$ int

int $\pm$ b $\Rightarrow$ int.

(7.3) Let f be a Flow in G, and let P be a simple s-t path in G_f. Then $v(f') = v(f) + \text{bottleneck}(P, f)$

But bottleneck $(P, f) > 0$

$\therefore$ $v(f') > v(f)$

PAGE No.
DATE / /

(7.4) Suppose that all capacities in the flow ntwk G are integers. Then the Ford Fulkerson algo terminates in atmost C iterations of the while loop.

$$C = \sum_{e \text{ out of } s} c_e$$

TRUE
G_f cannot have value greater than C.

initially Flow = 0
in worst case Ford Fulkerson will increase Flow by 1
Hence atmost C iterations

(7.5) $O(m+n) \Rightarrow O(m)$
Since all the nodes have atleast 1 incident edge.
suppose from above all capacities in the flow network G are integers.
Ford Fulkerson runtime $O(C \cdot m)$

C → while there is a s-t path in residual graph.

Path found using BFS or DFS $O(m+n) = O(m)$ ← Let P be simple s-t path in G_f

f = augmentation $(f, P) \rightarrow O(n)$ path P will have atmost n-1 edges
update f to f'
update G_f to $G_{f'}$ } $\rightarrow O(m)$

pseudo polynomial

Now we have to show Ford Fullerson returned Max flow.

From (7.4) we know

$$\text{Max-flow} \leq C \rightarrow \sum_{e \text{ out of } s} c_e$$

Sometimes this upperbound is very weak.

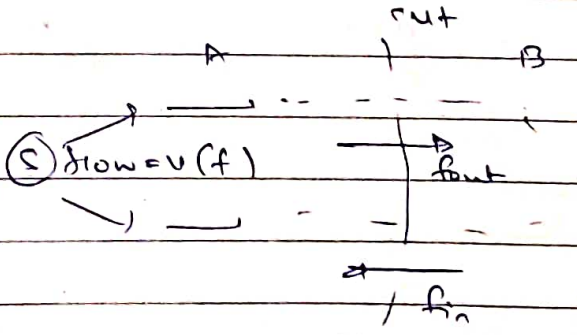
(*) A cut (A, B) has capacity $= c(A, B)$

$$c(A, B) = \sum_{e \text{ out of } A} c_e$$

Cut divides graph into two partitions A and B.

(7.6) Let f be any s-t flow, and (A, B) be any s-t cut.

Then $v(f) = f^{out}(A) - f^{in}(A)$



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

PAGE NO.
DATE / /

(7.7) Let f be any s-t flow, and (A, B) any s-t cut. Then $v(f) = f^{in}(B) - f^{out}(B)$

(7.8) Let f be any s-t flow, and (A, B) be st cut.

$$v(f) = f^{in}(B) - f^{out}(B)$$

$$c(A, B) = f^{out}(A)$$

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

$$v(f) \leq f^{out}(A)$$

$$\boxed{v(f) \leq c(A, B)}$$

value of every flow is upper-bounded by capacity of every cut.

(*) Let $\bar{f}$ be flow returned by Ford Fulkerson. We want $\bar{f}$ to have max possible value

We exhibit a cut (A^*, B^*)

$$v(f') = c(A^*, B^*)$$

This immediately establishes that $\bar{f}$ has the maximum value of any flow, and that (A^*, B^*) has the minimum capacity of any s-t cut.

PAGE NO.
DATE / /

(7.9) If F is ~~any~~ an st flow that there is no s-t path in the residual graph G_f, then there is an s-t cut (A^*, B^*) in G for which $v(f) = c(A^*, B^*)$.
Consequently f has the maximum value of any flow in G, and (A^*, B^*) has minimum capacity of any s-t cut in G.

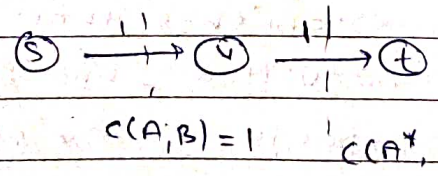
(7.10) Flow returned by Ford Fulkerson is max flow.

(7.11) Given a flow f of maximum value, we can compute an s-t cut of minimum capacity in $O(m)$ time.

Run BFS/DFS from s to find nodes reachable from s
they belong to A^*
$B^* = V - A^*$

$O(m+n) \Rightarrow O(m)$

Note: There may be many min capacity cuts



Our method returns the cut closest to S.

PAGE No.
DATE / /

This must be max flow

(7.12) In every flow network, there is a flow f and a cut (A, B) so that

$$v(f) = c(A, B)$$

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

(7.14) If all capacities in flow ntwk are integers, there there is a maximum flow f for which every flow value f(e) is an integer.

In every iteration

$$v(f') = v(f) + \text{bottleneck}(p, f)$$

As all numbers are int → ≥ 1

∴ while loop terminates in atmost C iteration.

But consider values in decimal.

$$v(f'') = v(f) + \text{bottleneck}(p, f)$$

→ 0.000001... !

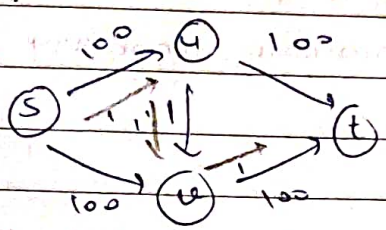
This loop wont end in finite time

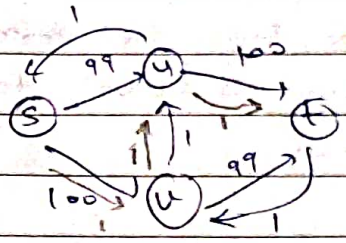
PAGE No.
DATE / /

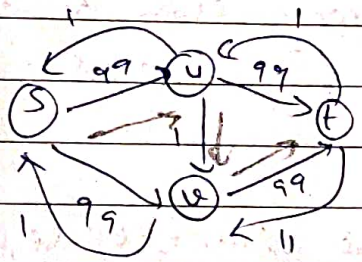
(A) Choosing good augmenting paths

As seen we just choose any path and augment it.

eg:







As seen if we keep picking this path it may take us 200 iterations

But we can do it in less number of iterations.

Idea: Recall that augmentation increase the value of the maximum flow by the bottleneck capacity of the selected path; so if we choose paths with large bottleneck capacity, we will be making a lot of progress.

Having to find such paths can slow down each individual iteration by quite a bit. we will also avoid this slowdown by not worrying about selecting the path that has exactly the largest bottleneck value.

Instead, we will maintain a so called scaling parameter Δ, and we will look for paths that have bottleneck capacity of atleast Δ.

let $G_f(\Delta)$ be the subset of the residual graph consisting only of edges with residual capacity of atleast Δ.

we will work with values of Δ that are powers of 2.

scaling max flow $O(m^2 \log_2 C)$

initially $f(e) = 0$ for all e in G

Initially set Δ to be the largest power of 2 that is no larger than maximum capacity out of s

$(\Delta \leq \max_{e \text{ out of } s} c_e)$

$\log_2 C$ while $\Delta \geq 1$ Ford Fulkerson

$O(m)$ while there is an s-t path in graph $G_f(\Delta)$

let P be a simple s-t path in $G_f(\Delta)$ $O(m)$

$f' = \text{augment}(f, P)$ $O(n)$

update f to be f' and update $G_f(\Delta)$

End while

$\Delta = \Delta/2$

end while

return f

(7.15) If the capacities are integer valued, then throughout the scaling max-flow algorithm the flow and the residual capacities remain integer-valued. This implies that when $\Delta = 1$ $G_f(\Delta)$ is the same as G_f, and hence when the algorithm terminates the flow f, is of maximum value.

(7.16)

$\therefore$ Δ SHOULD BE LESS THAN 70 OR EQUAL TO

$\Delta = 64$

The number of iterations of the outer while loop is atmost $1 + \log_2 C$

(7.17) During the Δ-scaling phase, each augmentation increases the flow by atleast Δ.

as all edges have capacity $\geq \Delta$

$\therefore$ BOTTLENECK $\geq \Delta$

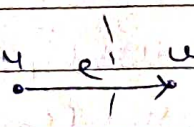
new flow = old flow + BOTTLENECK.

381 (7.18) let f be the flow at the end of the Δ-scaling phase. There is an st cut (A, B) in G_f for which $C(A,B) \leq v(f) + m\Delta$, where m is the number of edges in the graph. Consequently, the maximum flow in the network has value atmost $v(f) + m\Delta$

Proof

let there be A → nodes reachable from s

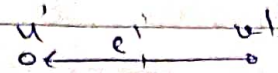
B → V − A.



$c_e < f(e) + \Delta$

or else v would also have been reachable and be in set A.

B A



$f(e') < \Delta$

IF $f(e') \geq \Delta$ then u' will belong to A

$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in } A} f(e)$$

$$\geq \sum_{e \text{ out of } A} (c_e - \Delta) - \sum_{e \text{ in } A} \Delta$$

$$= \sum_{e \text{ out of } A} c_e - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ into } A} \Delta$$

$$\boxed{v(f) \geq c(A,B) - m\Delta}$$

(7.19) The number of augmentations in a scaling phase is atmost $2m$

new flow $\leq$ old flow + m(old delta)

old delta $= 2$ new delta

new flow $\leq$ old flow + $2m$ (new delta)

↳ atmost $2m$ augmentations in scaling phase

PAGE No.
DATE / /

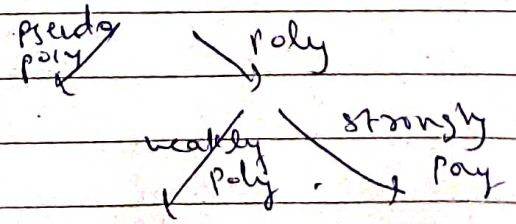
(7.20) The Scaling max-flow Algorithm in a graph with m edges and integer capacities finds a maximum flow in at most ~~$O(m^2 \log_2 ( )$~~ $2m(1 + \log_2 ( )$ augmentations.
It can be implemented to run in at most $O(m^2 \log_2 ( )$ time.

$O(m^2 \log_2 ( )$
efficient
dependent on no. of bits
weakly polynomial.

(*) strongly polynomial
(relevant if input consists of integers)

An algorithm runs in strongly polynomial time if the number of operations is bounded by a polynomial in the number of integers in the input

(*) weakly polynomial
An algorithm runs in weakly polynomial time if the no. of operations is bounded by a polynomial in the number of bits in the input, but not the number of integers in the input.



PAGE No.
DATE / /

Edmond's Karp

Same as Ford Fulkerson, except that each augmenting path must be a shortest path with available capacity

shortest path → least number of edge.

$O(nm^2)$

Ford Fulkerson	$O(C \cdot m)$	pseudo polynomial
Scaled Ford Fulkerson	$O(m^2 \log_2 C)$	weakly polynomial.
Edmonds Karp	$O(m^2 n)$	strongly polynomial.
orlin + KTR	$O(nm)$	

Recently developed methods to solve maxflow in close to linear time wrt m

These FMC approximations of max flow not exact maxflow.