AAD- Assignment -4

Problemset - 4

NAGAMANOHAR
2021101128

(Q1) Given,

G is a flow network with
source - s
Sink - t

edge capacity of edge (e) - c_e $e \in E$

(A,B) - minimum s-t cut

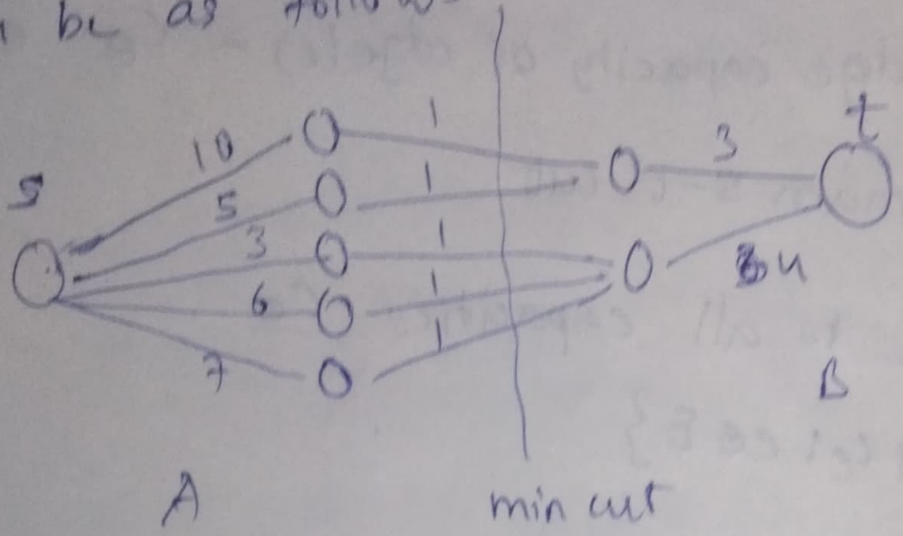
Now add 1 to all capacities c_e
$\Rightarrow \{1 + c_e ; e \in E\}$

Now we ~~pr~~ disprove the statement
(A,B) still is a minimum s-t cut. $\{1+c_e ; e \in E\}$

Proof:
we prove this by giving a Counter Example

let

G be as follows

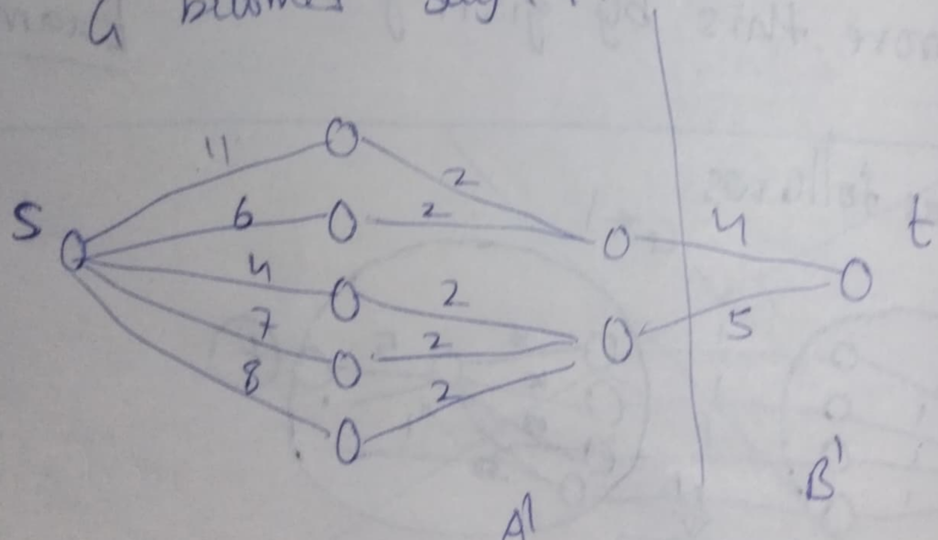


~~mincut~~ mincut(A,B) value $= 1+1+1+1+1 = 5$

Now make $\boxed{c_e = 1 + c_e \quad \forall e \in E}$

then

G becomes say (G')



New mincut $\Rightarrow (A', B') \Rightarrow$ mincut(A', B') value $= 9$

but

cut$(A, B) = 10$

$\therefore$ G, G' doesn't have same mincut

$\Rightarrow$ The statement is wrong

~~In (A,B)~~ in G' (A, B) need not be still

a min cut

Hence, proved.

Q2

we use the Max-flow algorithm to solve this problem.

1. we first construct the Graph
2. then Impement the algorithm on it.

Counts

1. consider
(P_i) patients to be vertices $P_i \in V$ - n
(H_i) Hospitals to be vertices $H_i \in V$ - K

Now connect every patient $\underline{P_i}$ to the set of Hospitals {H_i s/t P_i can be taken to H_i} in time}

Assume that the data about what all hospitals a patient can ~~go~~ reach is already given.

connect P_i to {H_i} => add a directed edge from P_i to {H_i} of edge weight ($c_e = 1$)

Now to form a network-flow take 2 nodes
s - source
t - sink

- Add directed edges from s to all patients (P_i) with capacity $c_e = 1$ $\boxed{s \to \{P_i\}}$ $\boxed{\cancel{P_i} \to S}$
- Add directed edges from all Hospitals (H_i) to sink t $\boxed{\{H_i\} \to t}$

with capacity $\lceil \frac{n}{k} \rceil$ = ceil of $\left(\frac{n}{k}\right)$

$\{H_i\} \rightarrow t$ $\boxed{c_e = \lceil \frac{n}{k} \rceil}$ $e \in E_{H \to t}$ (edges of $H \to t$)

- From source s Apply Max-flow algorithm
  (Edmonds-Karp network flow algorithm)

if (Max-flow == n)
then ~~we~~ it is possible to take all patients to Hospitals.

else if (Max-flow < n)
No evacuation Procedure possible

Runtime :-

① Building the Graph takes :-
$O(nk)$ - $\left(\text{each patient (total } n) \text{ can go to atmost } k \text{ hospitals}\right)$
$+$
$O(n)$ — source-s to all patients
$+$
$O(k)$ — $\{H_i\} \rightarrow$ sink t

② Maxflow algo take's :- $O(ve^2)$

v = No. of vertices = $n+k+2$ (s,t) $= n+k+2$

e = $nk+n+k$

$\Rightarrow O\left((n+k+2)(nk+n+k)^2\right) + O(nk) + O(k) + O(n)$

$\Rightarrow \boxed{O\left((n+k)(n^2k^2)\right)}$

① Here a patient can go to only one Hospital even if he has many options in the final-flow because - every time we consider an augemented path through that patient $\boxed{S \to P_i}$ is either 0/1 so once we chose a path to one H_j then to change we have to undo that path so that $\boxed{S \to P_i}$ become 1 again thus allowing to go to only one hospital at a time

- As $S \to P_i$ and $P_i \to H_j$ capacities are 1 ($C_e = 1$) we can consider max-flow to be

No. of people flowing to hospitals (reaching)

- Thus the algorithm works.

(Q3)

Given $G(V,E)$ with max flow $f \Rightarrow v(f)$ = value of flow

G' is formed by reducing capacity of an edge $e' \in E$ in G.

Here we first write the algorithm and then prove that G' has maximum-flow value between $[v(f)-1$ and $v(f)]$

Algorithm:

① If $e' \notin$ max-flow path (f) (or) e' is does not contribute to max-flow then even if we reduce e' capacity by the max-flow remains the same $\Rightarrow v(f)$

Now consider e' is a part of the flow and $e' = (u,v)$ $u,v \in V$

- Now we reduce the capacity of e' ~~as follows~~: s/t

① construct a path from $\underline{v}$ to $\underline{t}$ all edges carry flow and reduce the flow on each edge by one unit.

Now to restore the overall flow condition construct a path from $\underline{u}$ back to $\underline{s}$ s/t all edges carry flow and reduce the flow on each edge by one unit.

- Graph is G' now

- Now let the flow be f'. $\boxed{v(f') = v(f) - 1}$

So we check if f' is max-flow.

Now we try to find an augmenting path from s to t in residual graph $G'_{f'}$ and

if we get one such path (true)

then max-flow can be higher but as we reduced it by 1 it can only be

$v(f) - 1 + 1 = v(f)$

else (No augmenting path)

then

max-flow is $v(f') = v(f) - 1$

thus ~~flows~~ the can be atleast $\underline{v(f) - 1}$ or

at most $\underline{v(f)}$

Now we prove that G' has atleast $v(f)-1$ max-flow and atmost $v(f)$

for G max-flow $= v(f)$

and by (Max-flow = Min-cut)

we have a min-cut (A, B) in G with capacity $= v(f)$

there are 2-cases :

① $e' \in (A, B)$
② $e' \notin (A, B)$

① $e' \in (A, B) \Rightarrow$ then in G' the min-cut is still (A, B) but

capacity of mincut $= v(f)-1$

$\Rightarrow$ max-flow in G' is $v(f)-1$

② $e' \notin (A, B) \Rightarrow$ then again the mincut is (A, B) (need not be unique)

$\Rightarrow$ if there is a cut say X in G $\boxed{e' \in X}$

s.t capacity(X) = capacity(A, B) + 1

then in G'

capacity(X) = capacity(A, B)

but still as we take value not which cut

capacity of min-cut in $G' = v(f)$

$\Rightarrow$ ~~Min~~ capacity of min-cut in $G' = v(f)$

$\Rightarrow$ max-flow $= v(f)$

$\therefore$ from (1) & (2)

G' has ~~a~~ max-flow atleast $v(f) - 1$

and atmost $v(f)$.

Runtime: The Algorithm run's in $O(m+n)$ time only as we for constructing a path is DFS/BFS and ~~a~~ finding a single augmenting path is also $O(m+n)$

$\Rightarrow O(m+n)$

(Q4)

Given G has <u>unit capacity</u> edges

source $s \in V$ $\quad c_e = 1 \ \forall e \in E$

sink $t \in V$

given, a <u>k</u>

W.K.T

maximum s-t flow = Valute of minimum s-t cut

any k

Hence, $\boxed{val = k}$ as $c_e = 1$ | $k = $ | val = capacity sum of k edges

firstly if we remove any k edges

$\boxed{\sum_{e \in k} c_e}$

then for a particular cut the

$= \sum_{e \in k} c_e$

capacity may reduce by val (at most)

⇒ as min-cut is also a cut it's

capacity may also reduce by (at most val)

but

if we remove edges directly from

min-cut then

min-cut capacity will reduce by val (= k)

⇒ Max-flow reduces by val (= k)

(which is minimum possible flow)

Here we chose k s/t

if the min-cut has

less number of edges than k

then remove only those ⇒ $\boxed{\text{max flow} = 0}$

o/t ⇒ remove k edges.

Run Time :- As finding the Min-cut

is polynomial Time

⇒ The Algorithm runs in polynomial

Time

Q5

Given $G(V, E)$ source $- s \in V$
sink $- t \in V$
$c_e \Rightarrow$ edge capacities
$c_e \in +ve$
$\forall e \in E$

<u>Algorithm</u>

Firstly - Run mincut algorithm to find a min cut C. $|C|$ = capacity of min-cut C

Now we have to check if $\exists$ another cut D s/t $|C| = |D|$

we have the following :-

- we consider all possible cuts a particular set of cuts at ~~once~~ a time with their original capacities and find min-cut every time and check if it equals $|C|$.

thus

- we increase cut C capacity edge by edge
  $\Rightarrow$ few cuts capacities increase
  $\Rightarrow$ the other cut's that doesn't have this particular edge have original capacities

Thus we can get the New (X) min-cut in original graph other than C and if that min-cut's capacity (say $|X|$) $= |C|$) then minimum s-t cut is not distinct

o/w

it is distinct

let $e_1, e_2, \dots e_k$ be the edges in c

then

iteration 1: increase e_1 (say $(e_1 = e_1 + 1)$) and
find min-cut $\underline{X}$ and check
$(|c| == |X|)$?

~~Similarly~~ Here the cut's that don't
have e_1 in them are considered with
their original capacities (say $c_1, c_2, \dots c_{k_1}$)

The cut's that have e_1 anyhow don't form
a min-cut as we increased e_1

Similarly
let the cut's be ($c_{k_1+1}, c_{k_1+2}, \dots c_{k_2}$)
with their original values when we do
iteration 2: in G ⟹ increase e_2 this time
in G
and so on

only one e_i is changed ~~every time the other~~

for each i $i \in [1, k]$ $e_i = e_i + 1$ and all are unchanged
calculate mincut (x) & $|x| == |c|$ (check)

~~Keeee~~

at the end we have considered all possible as in G
cuts other than $\underline{c}$

min cut $(c_1, \dots c_{k_1}, c_{k_1+1}, \dots c_{k_2}, \dots c_{k_k})$

Thus we have calculated a mincut
out of original all cut's other than
$\underline{c}$

Hence,

if $|X| == |C|$ (in any particular iteration)

min cut s-t is NOT unique

else

mincut s-t is unique

Runtime: Mincut for k times

as Mincut run's in polynomial time say f then this algorithm run's in $O(f \cdot k)$ time

for $i \in [1, k]$:-

$c_i = c_i + 1$

Calculate min cut (X)

Check $|X| == |C|$? – ans = 0/1

$c_i = c_i - 1$ (change back)

Q6

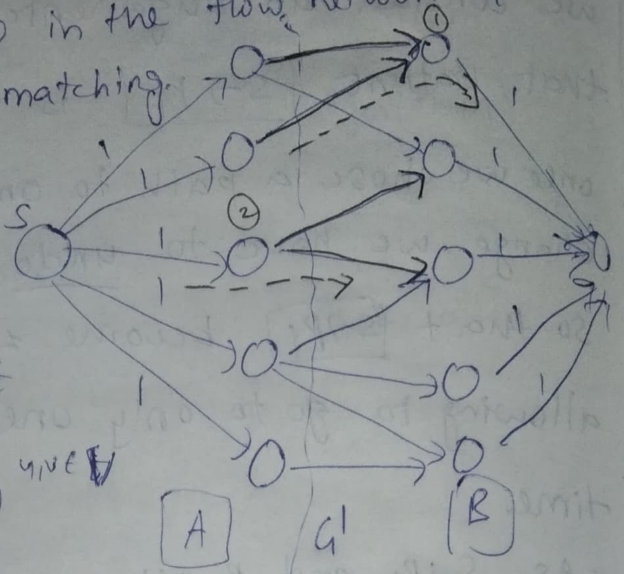
$|V_1| = |V_2| = n$ $\boxed{V_1 = A \quad V_2 = B}$

Given, $G(V_1, V_2, E)$

also G has a ~~d~~ perfect-matching in it.

RTP: Maximum flow in the flow network G'

gives perfect matching.



we get

edges in G'

by adding $1 \ \forall e \in E$

and $c_e = 1$ for $e(s,u)$ $u, v \in V$
$e(v,t)$

* we first prove that the edges in maximum flow will correspond to largest possible matching and as we have perfect matching the largest possible matching is perfect-matching itself

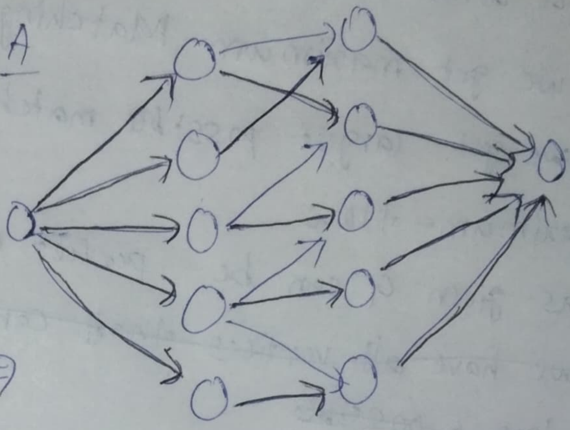
- as the capacities for a node in A from $s = 1$ and even though there are many edges out of each node in A, ~~we have~~ ~~only 2 options~~ there can only be one edge that is picked and it has to be their in the flow

Thus we
- use an edge completely (use & send flow through it)
- not use that edge at all

Now
From ① & ② in the diagram we see that ⓐ
as capacities of (s, x) (or) (x, t) are all one
- we can choose atmost 1 edge leaving any node in A and similarly we can choose only atmost one edge entering any node in B.

~~Let S be a set of edges from A to B (used for ~~in~~ flow)~~

Now
As inflow to ~~edges~~ nodes in A is atmost 1 the flow can be atmost 1 i.e., 1 or 0 only



Now from the diagram we see $\Rightarrow$ that

① If there is a matching of x edges then, there is a flow f of value x

② Also If there is a flow f of value x, then there is a matching with x edges.

① $\Rightarrow$ as atmost flow possible through each node is only 1 and each edge is 1
atmost flow into t from each node from $A \to B$
atmost flow ~~out~~ of each node is also 1

thus no 2 edges can reach same node in flow is followed

f $\Rightarrow$ matching in π B

~~Now~~ Similarly (2) is true

Now

we evaluate maximum flow f ~~...~~

and let it's value be $x = v(f)$

then by (2) it has x edges matching

Now if there were more edges matching than x

the $v(f) =$ max-flow value $> x$, which

is a contradiction

=) we get maximum Matching edges

=) we get largest possible matching with

maximum-flow

=) as given G can be perfect matching ~~as~~

~~=) we have all vertices that can be matched as largest possible~~

=) largest possible matching = perfect matching

will be returned

Thus,

maximum flow in network gives

perfect matching. (given G has a perfect matching)

Hence, proved

For

Q3 — I have referred to klienberg tardos Solved exercises

Q2, Q6 — I have referred to lecture slides of

some other universities.