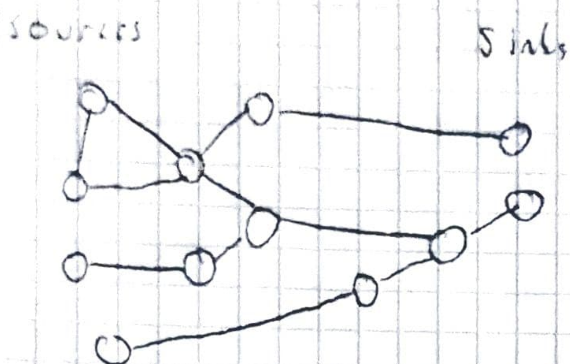
Ex 7-1. ✓✓ 10/10

65/100

Ex 7-2.

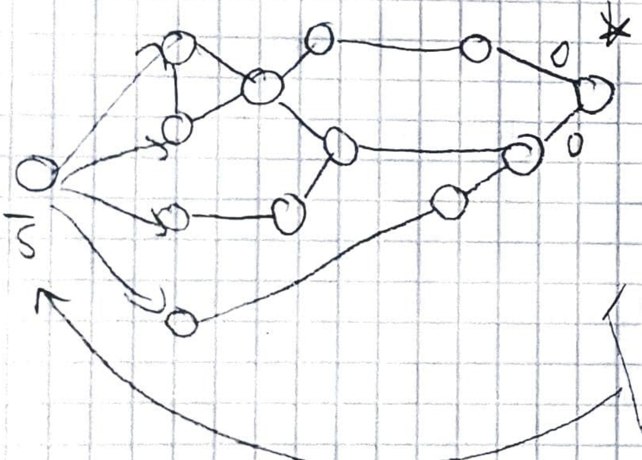
8/10

Say we have,



This holds flow properties such as, graph is directed,
We can also tell that as a property of flow there is no edge in reverse direction which is satisfied here.

* Each vertex has a path to sink which we can satisfy with adding a super sink, graph, flow becomes,



Now all vertices are connected and for each v there is path to sink.

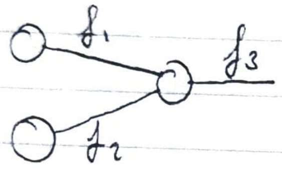
Non negativity constraints for edges are trivial.

For a single source we can add super source $\bar{s}$ which solves the issue of single source.

Ex 7-3.

If set S_n is set of flows from 1 to n.
For it to be convex set we need
$conv(\sum_n S_n) = \sum_n conv(S_n)$

Since flows are convex we can say that for f_1, f_2 and f_3 for



this holds true since we can add $f_1 + f_2$ to have f_3 which is property of conv.

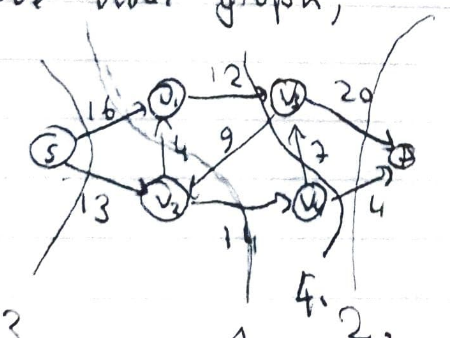
Then logically we can say we can multiply max flow of edges as we want and, multiply f_1 with α and f_2 with $(1-\alpha)$ than set is still convex since $\alpha f_1 + (1-\alpha) f_2$ is still flow since coming and leaving flows are unchanged since with the property of flow in = out.

4/10

Ex 7-4.

8/10

We have graph,



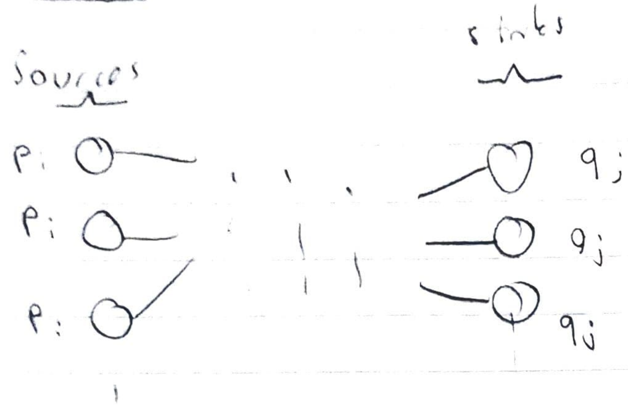
For maximum cut we draw
1. takes $16+4+9+14$ but we don't count 9 because when we cut 16 and 4 no water reaches V_1 and since 14 is cut no water reach V_3. We have 34.

2nd cut would be $20+4=24$.
3rd would be 29.

First one with 34 value would be thus maximum possible.

4th is $12+7+4=23$ which is least we can find and thus it is minimum cut. It cuts (V_1, V_3), (V_4, V_3), (V_4, t)

Ex 7-5.



create a source s.t. produced flow is $\sum_{0}^{s} p_i$ and for each source our super source is connected to each of them.

For the sink, our single supersink have capacity of $\sum_{1}^{t} q_j$ for flow.

We also need $\underbrace{\sum_{1}^{s} p_i = \sum_{1}^{t} q_j}_{\text{flow}}$ since flow need to have in = out.

10/10

P7-1.

3,5/4

a)

1- Increasing capacity of single edge would at most increase total max flow by one however, it is possible the vertex after it or so on might not be able to take these extra flow. So, if that edge is part of min cut then it increases.

2- This logic is same as first one and so if that edge is bottleneck (in min cut) then increasing it will increase flow but o.w. it won't change.

3- If an edge can transfer all of its capacity to later vertices and edges then again decreasing its capacity will decrease flow, it is also in min cut so we can see it that way.

4- Again this is almost same as third one since we are essentially doing same thing. If edge is in min cut it decreases.

→

S.S / 8

b) if ((u,v) ∈ mincut):
    while (path from s to t with capacity exists)
        ford-fulkerson (G, 1)
                                    value 1 iteratively so we don't exceed
    return G;

With iterating with one if max flow can change we securely do operations. Since ford-fulkerson runs in $O(E)$ and since max number of times we call ford-fulkerson is $(r - c(u,v))$ we have complexity of $O(E*(r-c(u,v))) \simeq O(r*E)$

c) Here, we check if the decreased edge was part of the min cut again and if so we run the following algorithm.
Else just return

3 / 8

algorithm;
// count=0
for each path s → t
    if path includes edge (u,v)
        while (each elt in path > 0 and count < (c(u,v) − r))
            each edge in path −−
            count ++
setlimit((u,v), r)
run ford_fulkerson (G)
\\

Now we have flow with properties.
We make sure flow property is satisfied by deducting from each edge in path until we hit r.
Then we run ford-fulkerson to maximize flow with constraint.
Complexity of second part → $O(E.r)$
                first part → k paths, $O(k.E)$
So it would be $\max(O(E.r), O(k.E))$

⇨

10/15

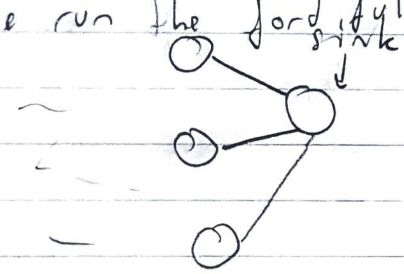
P 7-2.

This is a classic flow maximization problem. We set the limit for each edge to 1. Now we have the problem but for optimization we need to run ford-fulkerson algoritm.

For that algoritm we need single source and sink so, we add super source and sink just like the exercise.

Super source is connected to all original sources for companies. one by one.

Then we run the ford-fulkerson for number of companies.



However, there is an edge case, if there is a single edge to sink this would only work for a small company which is not ideal.

Runtime would be $O(E.(\text{num of companies}).V)$ since we run it for each company.

P 7-3.

3/15

Algoritm: complexity → $O(m.n)$

arr = sort all b_j values max to min
~~for~~ i in arr:
  for a_i: each a_n ; //each customer
    if(i in a_i.set):
      a_i.remove from a_n
      i -- //deduct one from foods

This would give optimal solution since first we are always taking most produced food and last least produced which would best for giving least coupons.