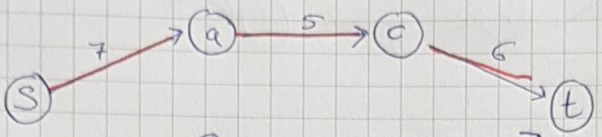
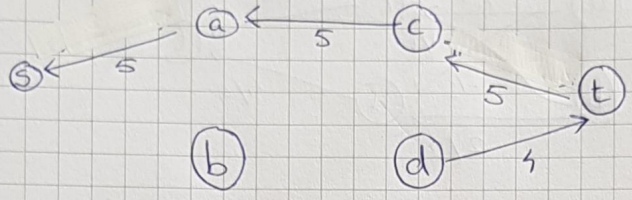
Exercise 1

We show Δ, $G_f(\Delta)$ at every iteration of the inner while loop:

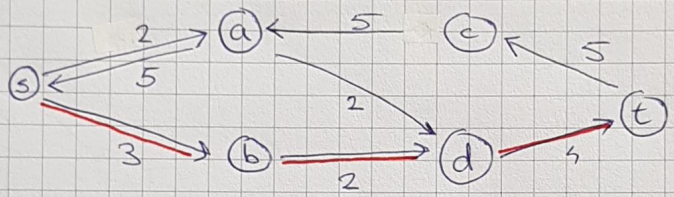
initially $\Delta = 2^2 = 4$ since $C = 7$

$G_f(\Delta)$ = 

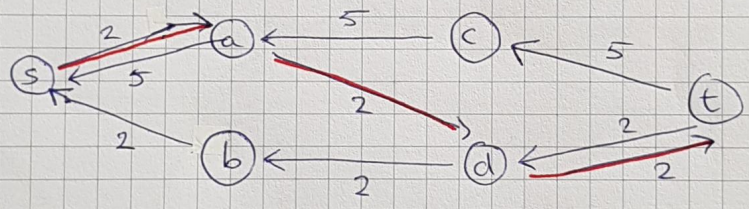
$\Delta = 4$

$G_f(\Delta)$ = 

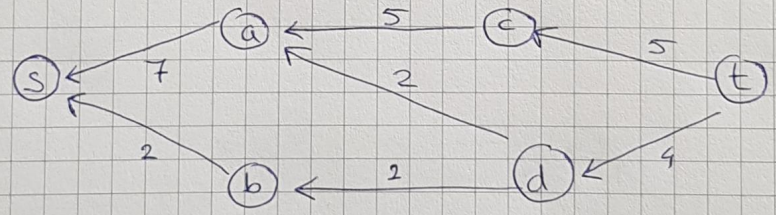
$\Delta = 4$

$G_f(\Delta)$ = 

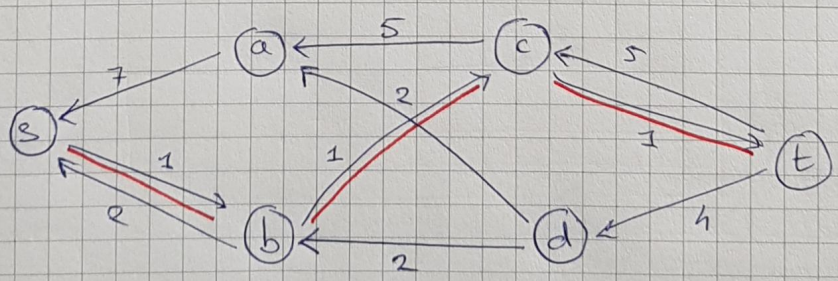
$\Delta = 2$

$G_f(\Delta)$ = 

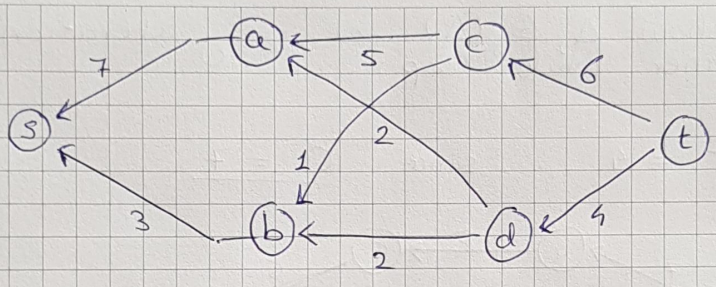
$\Delta = 2$

$G_f(\Delta)$ = 

$\Delta = 2$

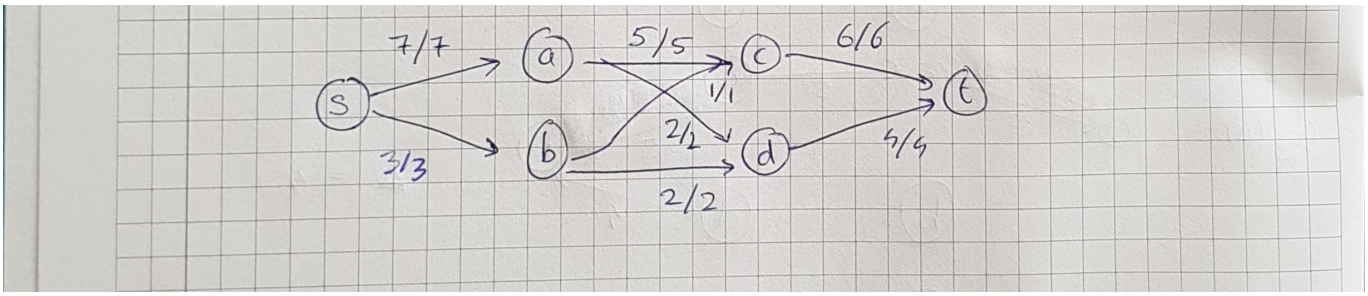
$G_f(\Delta)$ = 

$\Delta = 1$

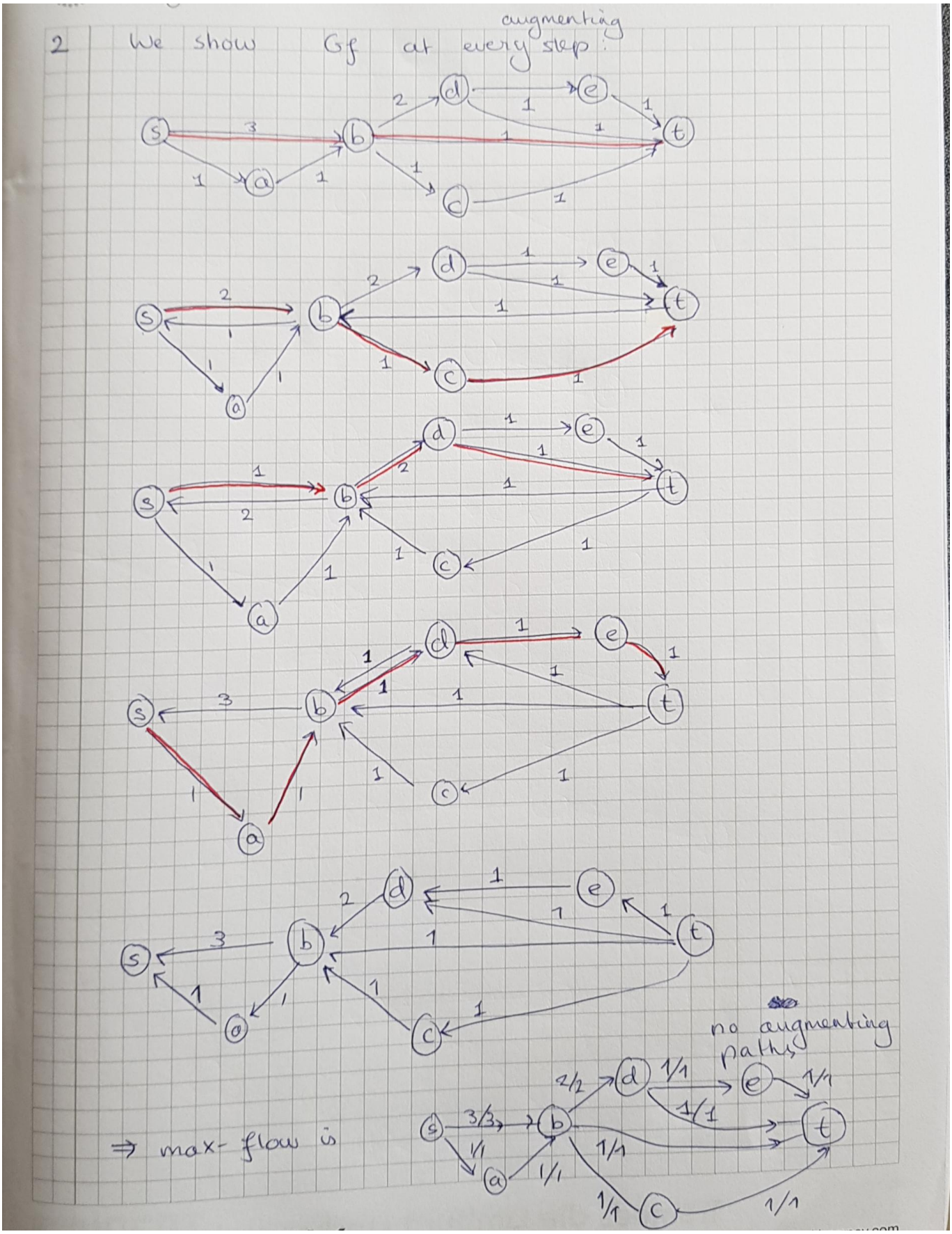
$G_f(\Delta)$ = 

$\Delta = 1$

finished. Max-flow:



Exercise 2



Exercise 3

We transform this into a flow problem. If the flow problem has a max-flow of value at least n, then this gives a possible way of connecting clients to base stations. If the max-flow value is lower than n, not all clients can be connected in the original problem statement.

Define a graph G = (V,E), where

- V is a set with for every client a node Cl_i, for every station a node St_i, and two additional nodes S and T.
- Edges E connect S -> Cl_i for every client Cl_i, St_i -> T for every station St_i, and there is an edge Cl_i -> St_j if and only if Cl_i is within range r of St_j.
- Set capacities on the edges: for each edge (Cl_i, St_j) that exists in E as per the previous point, set c(Cl_i,St_j) = 1. Set for all St_j c(St_j, T) = L, where L is the load parameter. Set c(S, Cl_i) = 1 for all Cl_i.
- Let S be the source node and T be the sink node for the associated max-flow problem

Since this max-flow problem has integer capacity, a max-flow can be assumed to have integer flows on all the edges (since the unit amount by which flows can be increased is 1, any variant of Ford-Fulkerson will yield an integer solution as well). This means that for every edge (Cl_i,St_j) we either have a flow of 0 or 1. Let Cl_i be connected to St_j iff f(Cl_i,St_j) = 1.

If the outflow of S is n, this means that every Cl_i has a flow of 1 passing into it and thus out of it, so it is "connected" to one of the base stations. On the other hand, the outflow and thus by conservation the inflow of any base station cannot be more than L, so not more than L clients are connected to one base station. Because there are only connections from clients that are within range r of their connected base stations, we conclude that a max-flow of value n (note that more than n is not possible since sum{ c(e): e out of S} = n, so there is not more capacity anyway) means that there is a connection possible.

On the other hand, if there is a connection possible, we can make a flow of value n by setting f(Cl_i, St_j) = 1 for all (Cl_i,St_j) such that Cl_i is connected to St_j, setting f(S,CL_i) = 1 for all i and setting f(St_j, T) = sum{f(e): e into St_j}. We see that this satisfies conservation by definition, and: Since there are less than L clients connected to every station, sum{f(e): e into St_j} <= L, so it also satisfies capacity on all edges (including both the edges with capacity 1 and those with capacity L).

So if the max-flow has value < n, we can conclude that such a connection is not possible. Because if it was, we would have found a max-flow of value n.

Running time:

Constructing the graph takes forming n+k edges and n+k+ at most n*k edges (if every client is within range of every base station), and solving the flow-problem with edmonds-karp is O(E^2 * V), where E <= (n+k+nk) again and V = (n + b + 2), so this is O((n+k+nk)^2(n+k+2)) which is polynomial in n and k.

Exercise 4

We can formulate this as a flow problem.

Define a graph G' = (V',E') where

- V' is the union of V with an extra node ss, the "super-source"
- E' has a directed edge (u,v) for every edge (u,v) in E, and additionally contains for each s in S an extra edge (ss,s)
- we give capacities 1 to every forward and backward edge created from an original edge in E. We give capacity k_s to every edge (ss,s)

We then solve the max-flow-problem on this graph, where we take source node ss and sink node t. Again, the max-flow of this graph is integer because all capacities are integer.

The upper bound for this max flow is again sum {c(ss,s): s in S} = sum {k_s : s in S}. If this flow is achieved, then there is also a flow of k_s out of every s in S, and since all edges out of nodes that are not ss have capacity 1, this means there are k_s edges saturated. Every edge can be taken as the initial edge of a path, and then we continue to add edges from the next node onwards, until we reach v.

All edges that are added on the way, are deleted from the graph. So:

```
let G' = graph G with added vertex ss and connections (ss,s) for s in S
let c = as described in text: c(e) = 1 for all e in E and c(ss,s) = k_s for
all s in S

f = edmonds-karp(G,c)


if val(f) = sum{k_s : s in S}

    // Procedure to prove correctness, not used in complexity calculation
    let G'' : graph G with only edges where f(e) == 1

    paths = []

    for s in S:
        for i = 1 upto k_s
            find a path p from s to t using edges in G''
            paths.add(p)
            G <- delete the edges e in this path p from G''
    // End procedure

    // Just return true:

    return true

else:
    return false
```

Note that the exercise only requires to return a boolean, I only included the procedure to find the actual paths to argue that the algorithm is correct. Because:

At every iteration of the path-finding procedure, there should still be a path from s that has outflow to t because otherwise conservation is not satisfied (then, there would not be a flow of sum {k_s : s in S} into v, because some paths never reach v).

Why does this give edge-disjoint paths? Suppose that there are paths p and q that share an edge. Then, we would at some point have deleted all edges in q from G'' in the above procedure, but then we will never find p using this procedure. Contradiction, since at every iteration there should be paths findable due to conservation.

Another way to see this, is that if an edge is shared between two paths, then this means that two paths merge into one at some node. But this node then would have two extra inflowing edges of f(e) = 1 but only one (shared) outflowing edge of f(e) = 1, thereby also violating conservation!

The complexity of max-flow implemented as Edmonds-Karp is O((V+1)(E+S)^2), since we add S edges and one extra source vertex ss. So the total complexity (ignoring the follow-up procedure used to prove correctness) is O((V+1)(E+S)^2).

Exercise 5

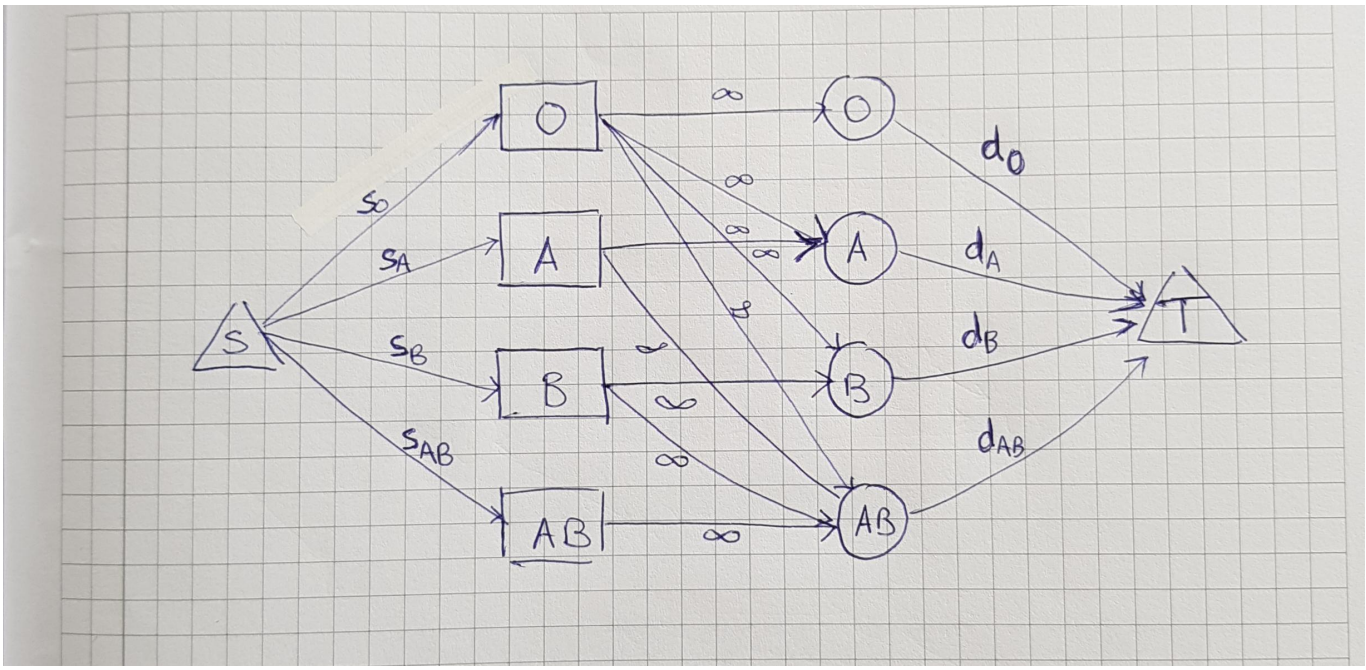
(a) We can formulate this as a max-flow problem, where we have a graph consisting of 10 nodes:

o, a, b, ab, O, A, B, AB, S and T. there is an edge from S into o, a, b, ab and an edge from each of O, A, B, AB into T, and there is an edge from a lower-case bloodgroup x into an uppercase bloodgroup Y if one can donate to a patient of group Y using blood of group x.

The capacities are set as follows:

- for every edge (s,x) we have capacity s_x
- for every edge (Y,T) we have capacicity d_Y.
- for every edge (x,Y) between a lower-case bloodgroup x and an uppercase bloodgroup Y, we have an unlimited capacity INFINITY.

Here is a picture of the graph, actually:



If we can set up transfusions to satisfy the projected need, this means that we can define donation(x, Y) such that

- sum{donation(x,Y) : x = o, a, b, ab} = d_Y for each Y = O, A, B, AB, i.e. demands are satisfied.
- sum{donation(x,Y) : x = O, A, B, AB} <= s_x for each x = o, a, b, ab, i.e. supply constraints are satisfied.

But then set

- f(x,Y) = donation(x,Y),

- f(S,x) = sum{donation(x,Y) : x = O, A, B, AB} <= s_x, and
- f(Y,T) = sum{donation(x,Y) : x = o, a, b, ab} >= d_Y,

This flow satisfies the capacity constraints on the flow problem and it satisfies conservation by how it is defined: because, sum(donation(x,Y) x = ...) = sum(f(e): e into Y) and sum(donation(x,Y), Y == ...) = sum(f(e): e out of x). Moreover, all edges to T are saturated, so the flow is clearly maximal. So if the blood on hand is sufficient, we get a max-flow that has

- f(e) = d_Y for e = (Y,T) for all Y

If the max-flow does not fully use (that is f(e) = c(e)) all edges (Y,T), then it has a lower value than a flow constructed from a valid donation schema `donation(x,Y)` would have given, so we also know that there cannot be such a valid donation schema otherwise the max-flow would have been higher.

We can find the max-flow using edmond-karps, of which the running-time complexity is O(E^2 V) = O(17^2 * 10) = O(1), it is constant time (because what is even the size parameter of our input here? The supplies and demands have no influence on the running time of edmonds-karp. If we used an arbitrary way of finding augmenting paths, then in the worst case the time complexity would depend on val(f*) which in the worst case would be sum(d_Y : Y in O, A, B, AB)). So it is certainly polynomial.

(b)

We will reason that certain allocations are strictly necessary and reduce the problem step-by-step until we find a solution or reach a dead end and conclude that there is no solution.

Blood group O can **only** receive blood from blood group O, so we have to give 45 units of O to O, leaving us with

Supply' = {O: 5, A: 36, B : 11, AB : 8} Demand' = {A : 42, B : 10, AB : 3}

We cannot donate AB to any other group than AB, so donate AB to AB and we are left with 5 units of AB which are of no use because none of the groups can receive it:

Supply'' = {O: 5, A: 36, B : 11} Demand'' = {A : 42, B : 10}

Now, we see that the demand of A can never be satisfied, since A can only receive from A and O, which can together provide 5 + 36 = 41 units, while A demands 42 units.

So there is no allocation that satisfies the needs.