Exercise sheet 10.

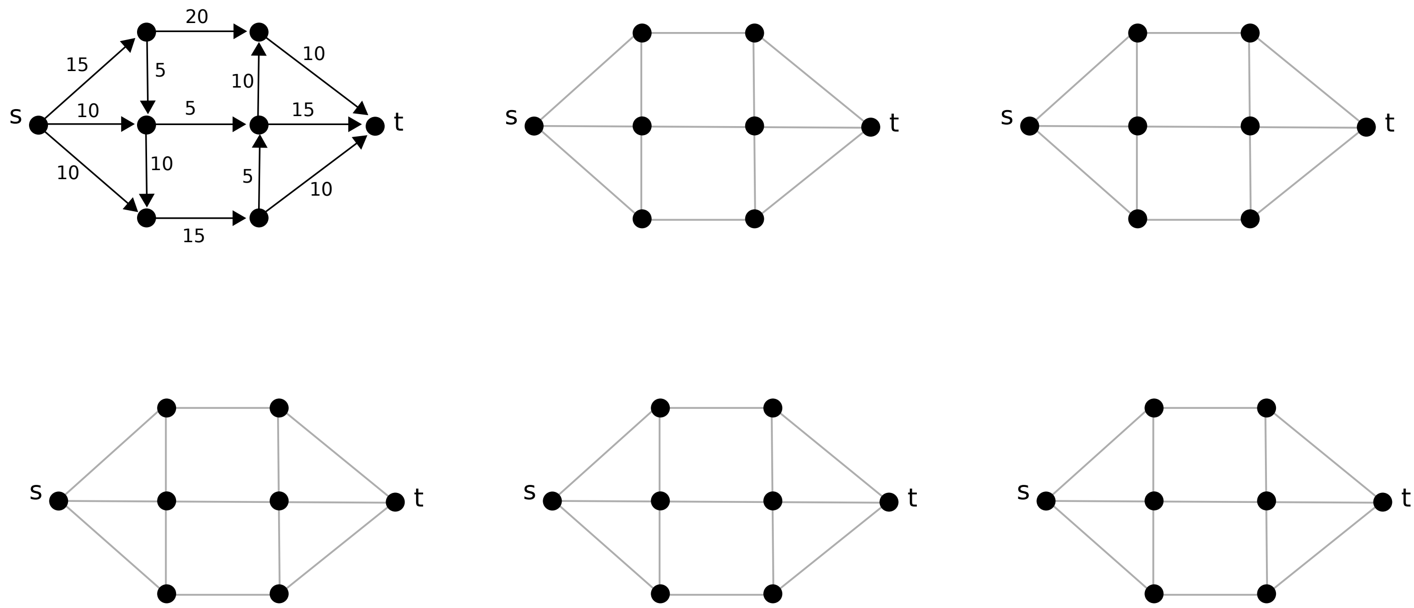
Advanced Algorithms WiSe 2020/21

Instructors: László Kozma, Mahmoud Elashmawi

Due 18:00, January 31st, 2021

Exercise 1 Maximum flow *12 pts*

(a) (6 pts)
Compute the maximum s-t flow by successive augmentations in the network shown on the figure (capacities written on the edges). Indicate intermediate steps, residual graphs, etc. as needed. Argue why the obtained flow is optimal. You can choose the augmenting paths arbitrarily.

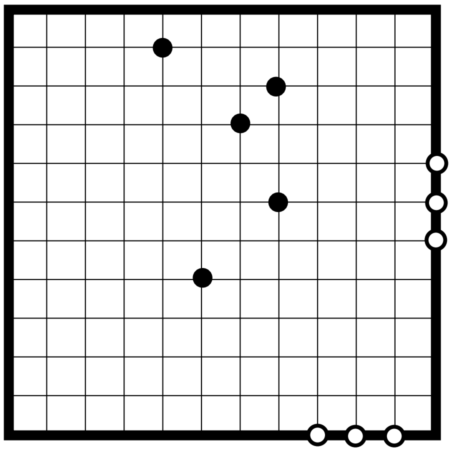


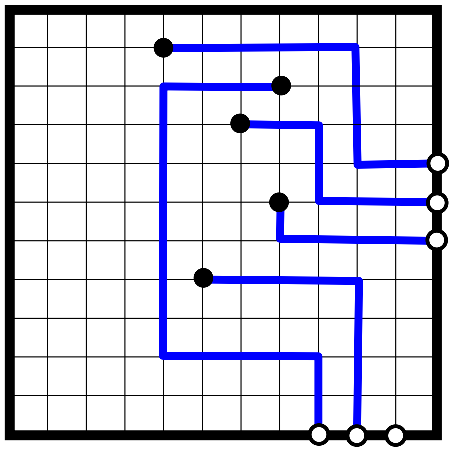
(b) (6 pts) Given is a network G with integer capacities, a source s, a sink t, and a maximum s-t flow f. Suppose that the capacity of a single edge of G is changed. Can you compute the maximum flow in the changed network more efficiently than computing it from scratch? Note that if the capacity of an edge *decreases*, the previous flow f may become invalid. How would you update the flow if the capacity of an edge changes only by one unit?

Consider the four cases separately: the capacity of an edge either (i) *increases* by 1, (ii) *decreases* by 1, (iii) *increases* by an arbitrary amount, (iv) *decreases* by an arbitrary amount.

Exercise 2 Escape routes *6 pts*

Consider the square with corners $(0, 0)$ and (n, n). We are given k points with integer coordinates inside this square and m *exit points* with integer coordinates on the boundary of the square ($m \geq k$). Describe an algorithm that finds *disjoint* escape routes for all k points, or reports that no such routes exist. An escape route connects a point to an exit point, and consists of horizontal and vertical line segments of length one, between neighboring integer points. Routes may not intersect. See figure for an example input (left) and a possible solution (right).





Exercise 3 Widest s-t path *4 pts*

Last week we considered the problem of finding *widest paths* from a single source to every other vertex, by modifying Dijkstra's algorithm. (Recall that the width of a path is the *minimum* edge-weight along the path.)

Describe a simpler algorithm (i.e. without using advanced data structures) to find a widest s-t path in a directed graph (as needed in one of the maximum flow algorithms discussed in the lecture).

Hint: you can target a running time of $O(m \log n)$.

Total: 22 points. Have fun with the solutions!

From: Yumeng Li and Thore Brehmer

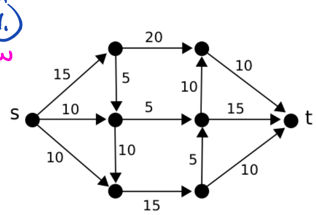
Exercise 1 Maximum flow *12 pts*

(a) (6 pts)

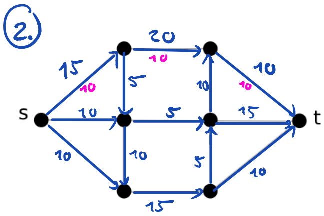
Compute the maximum s-t flow by successive augmentations in the network shown on the figure (capacities written on the edges). Indicate intermediate steps, residual graphs, etc. as needed. Argue why the obtained flow is optimal. You can choose the augmenting paths arbitrarily.

1.

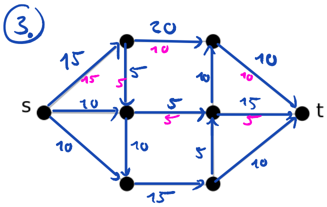
Pink = Flow



2.



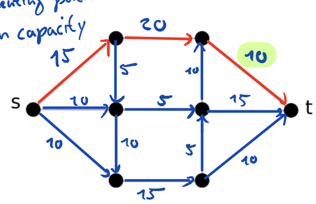
3.



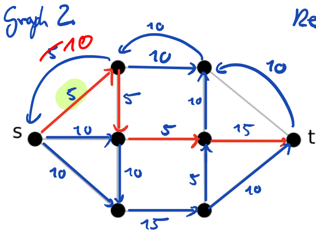
Residual Graph 1.

Red = augmenting path

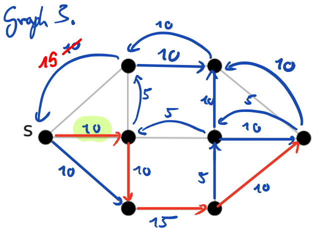
Green = min capacity



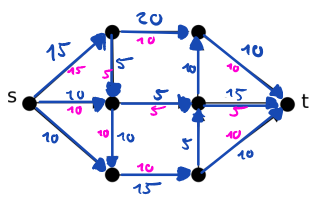
Residual Graph 2.



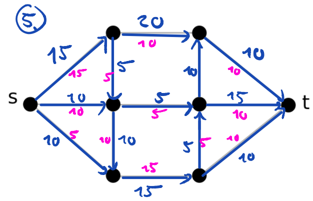
Residual Graph 3.

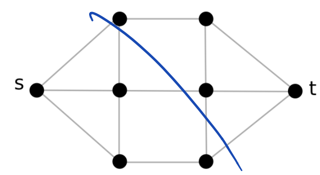


4.

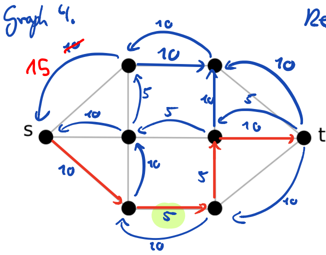


5.

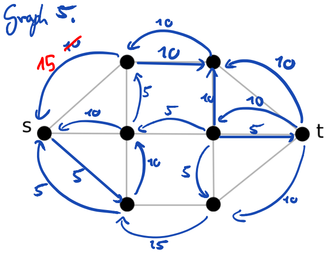


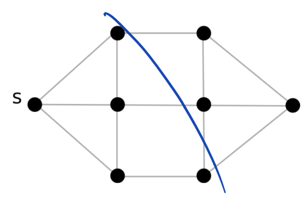


Residual Graph 4.



Residual Graph 5.





Optained flow is optimal, becaus there is no path (s,t) in the residual graph 5

6

(b) (6 pts) Given is a network G with integer capacities, a source s, a sink t, and a maximum s-t flow f. Suppose that the capacity of a single edge of G is changed. Can you compute the maximum flow in the changed network more efficiently than computing it from scratch? Note that if the capacity of an edge *decreases*, the previous flow f may become invalid. How would you update the flow if the capacity of an edge changes only by one unit?

Consider the four cases separately: the capacity of an edge either (i) *increases* by 1, (ii) *decreases* by 1, (iii) *increases* by an arbitrary amount, (iv) *decreases* by an arbitrary amount.

i) if the capacity of an edge increases by 1

1. Search for an augmenting path(s,t) in the residual graph

2. find it with e.g. DFS

- if there is none

$\Rightarrow$ current flow is still the optimal flow
$\Rightarrow$ increased edge has no impact on the flow

(if there is a new augmenting path it has to include the increased edge. Because there was no augmenting path befor we increased the edge)

3. else update the flow corresponding to the residual graph ✓

($\Leftrightarrow$ just use the Ford-Fulkason alg once for one path)

iii)

just lik i)

- do 1. 2. 3 until there is no augmenting path ✓

ii) if the capacity of an edge decreases by 1

1. test if capacity on the changed edge (x,y) < flow

2. if capacity ≥ flow

⇒ the decreased value has no impact on the flow (the edge is not used or not fully utilized)

⇒ the network can still hold the same flow

3. if capacity < flow

- search for an augmenting path(s,x) in the residual graph

3.1 if there is such a path (adjust the path / increase path(s,x))

- update the flow corresponding to the path

⇒ the network can still hold the same flow

Conservation at y still violated ?

3.2. if there is not such a path (decrease the flow path)

- find the augmenting paths: path(x,s) and path(t,y) in the residual graphs

(there always be such paths, because we had flow in the path (x,y))

- Update the graph corresponding to these Graphs (But just with 1 flow)

correct, but you might need to check G_f for augmenting paths afterwards

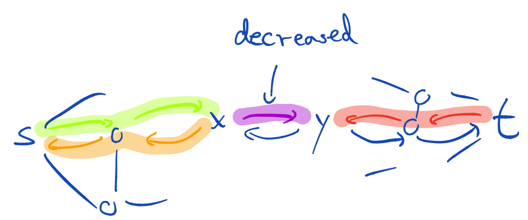
path (s,x)

path (x,s)

path (t,y)

path (x,y)

e.g. Graph j

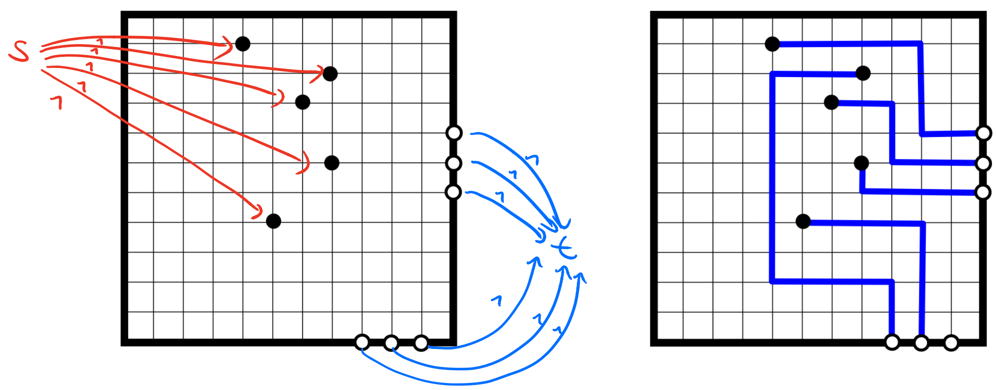


iv) just like ii) but do 3. until capacity ≥ flow ~~ff~~

missing runtime analysis 3

Exercise 2 Escape routes *6 pts*

Consider the square with corners $(0, 0)$ and (n, n). We are given k points with integer coordinates inside this square and m *exit points* with integer coordinates on the boundary of the square ($m \geq k$). Describe an algorithm that finds *disjoint* escape routes for all k points, or reports that no such routes exist. An escape route connects a point to an exit point, and consists of horizontal and vertical line segments of length one, between neighboring integer points. Routes may not intersect. See figure for an example input (left) and a possible solution (right).



1. Interpret all coordinates from (0,0) to (n,n) as vertices.

2. Every vertice has an edge with weight 1 to his neighbor (within reach of 1, just like in the picture)

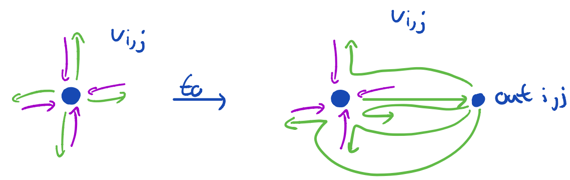
3. Create a new vertice s (outside the square) and and connect all the k given vertices with an edge of weight 1

4. Connect all the exit points to a new vertice t (outside the square) with edges of weight 1

5. All the vertices need the following adjustments:

   5.1 create a new out i,j vertice for the vertice $v_{i,j}$ which has the same outgoing edges as $v_{i,j}$

   5.2. Now delete all the outgoing edges of $v_{i,j}$ and create a new edge with weight 1 to out i,j e.g. for 5



- Now we can use Ford-Fulkason on the path (s,t)
  if we get a flow $\geq k$ $\Rightarrow$ there are "escape routes"
  if flow $< k$ $\Rightarrow$ no such routs exist

- Works because with the steps 2, 3 we can count the flow for the described problem.
  And with adjustment 5 we restricted that every vertice can only be visited once

6

Exercise 3 Widest *s*-*t* path *4 pts*

Last week we considered the problem of finding *widest paths* from a single source to every other vertex, by modifying Dijkstra's algorithm. (Recall that the width of a path is the *minimum* edge-weight along the path.)

Describe a simpler algorithm (i.e. without using advanced data structures) to find a widest *s*-*t* path in a directed graph (as needed in one of the maximum flow algorithms discussed in the lecture).

Hint: you can target a running time of $O(m \log n)$.