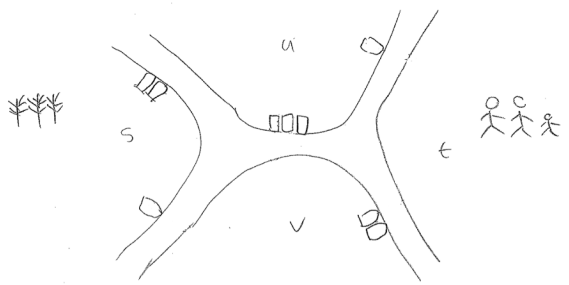
Algorithms on Graphs: Flow. Part 2

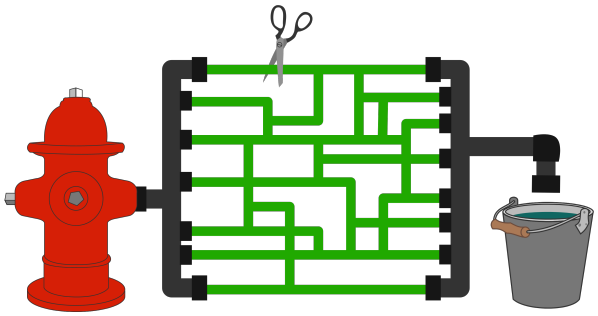
Prepared by Mark and Nikita on April 8, 2023

Part 1: A few puzzles to start

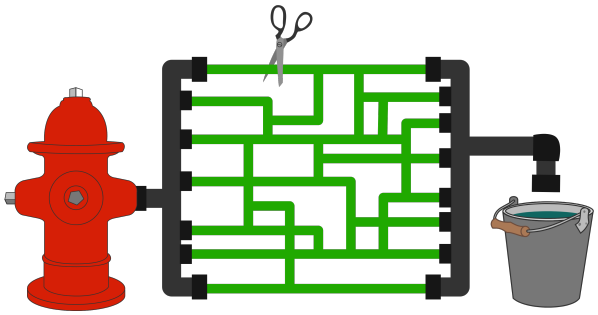
Problem 1:
How many loads of grain can you transport from point s to point t by crossing the river directly and without using the same boat more than once?



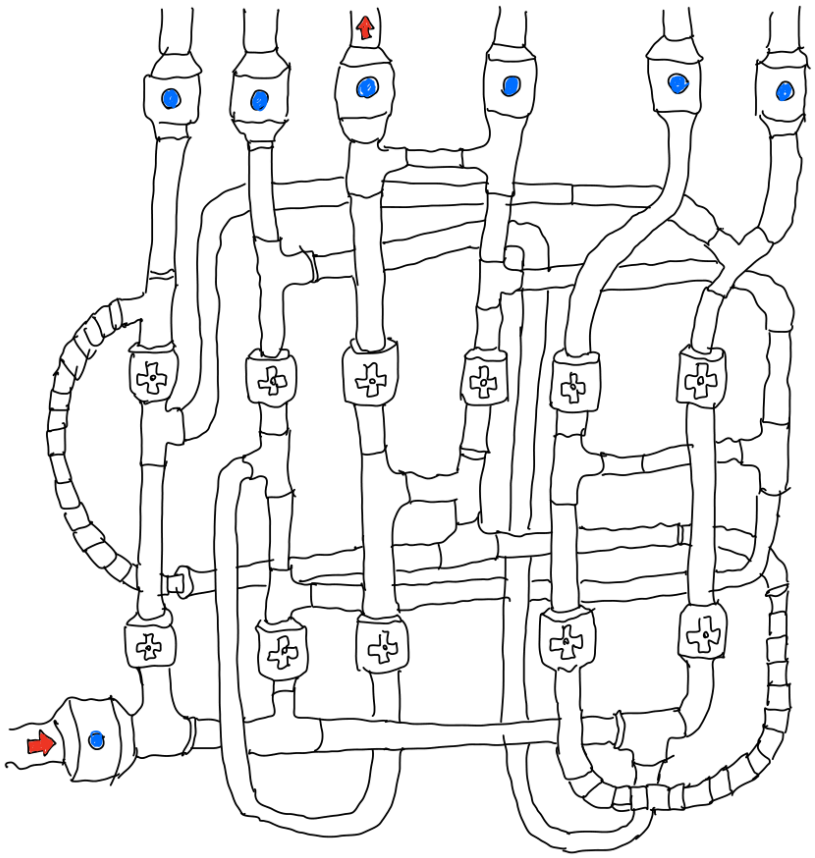
Problem 2:
What is the fewest number of green tubes that need to be cut so that no water can flow from the hydrant to the bucket?



And one more copy for your attempts:

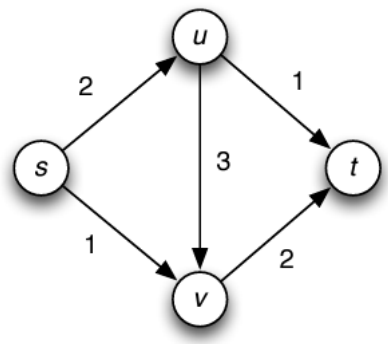


Problem 3:
Identify three valves to close in order to prevent water from flowing from the bottom pipe to the third pipe at the top.



Part 2: Ford–Fulkerson recap

All these puzzles have something in common. They are asking you to find a maximal flow in some network. For example, Problem 1 asks about the following network:

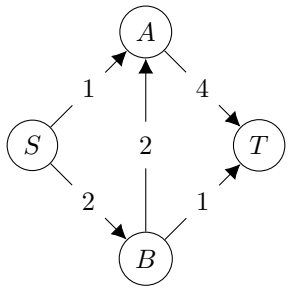


Networks Say we have a network: a sequence of pipes, a set of cities and highways, an electrical circuit, etc.

We can draw this network as a directed weighted graph. If we take a transporation network, for example, edges will represent highways and nodes will be cities. A valid network graph must satisfy certain conditions:

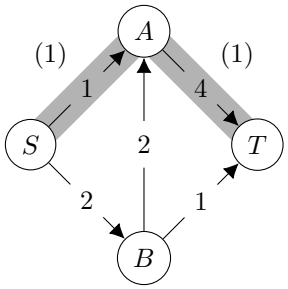
- The weight of each edge represents its capacity, e.g, the number of lanes in the highway.
- Edge capacities are always positive integers.[1]
- Node S is a *source*: it produces flow.
- Node T is a *sink*: it consumes flow.
- All other nodes *conserve* flow. The sum of flow coming in must equal the sum of flow going out.

Here is an example of such a graph:



Flow

In our city example, traffic represents *flow*. Let's send one unit of traffic along the topmost highway:



There are a few things to notice here:

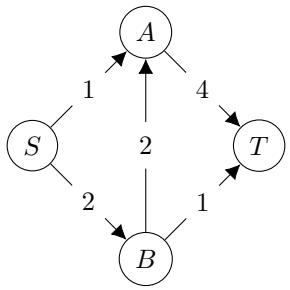
- Highlighted edges carry flow.
- Numbers in parentheses tell us how much flow each edge carries.
- The flow along an edge is always positive or zero.
- Flow comes from S and goes towards T.
- Flow is conserved: all flow produced by S enters T.

[1] An edge with capacity zero is equivalent to an edge that does not exist; An edge with negative capacity is equivalent to an edge in the opposite direction.

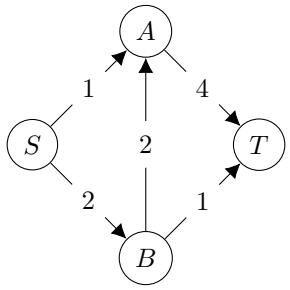
The *magnitude* of a flow is the number of "flow-units" that go from S to T.
We are interested in the *maximum flow* through this network: what is the greatest amount of flow we can get from S to T?

Problem 4:
What is the magnitude of the flow above?

Problem 5:
Find a flow with magnitude 2 on the graph below.

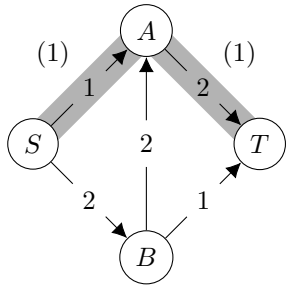


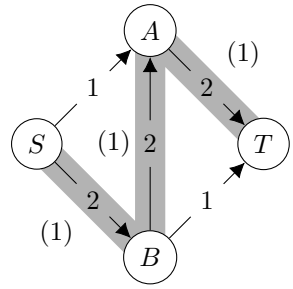
Problem 6:
Find a maximal flow on the graph below.
Hint: The total capacity coming out of S is 3, so any flow must have magnitude ≤ 3.



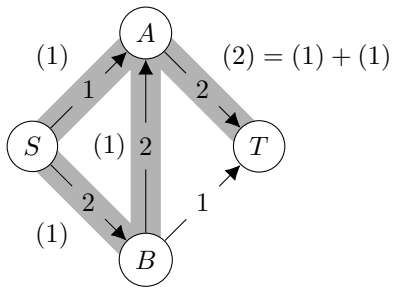
Part 3: Combining Flows

It is fairly easy to combine two flows on a graph. All we need to do is add the flows along each edge. Consider the following flows:





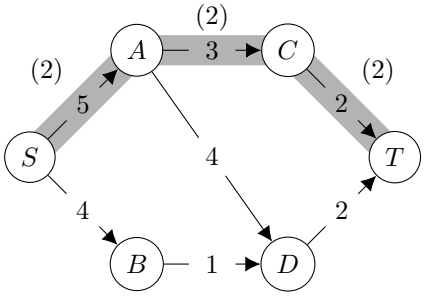
Combining these, we get the following:

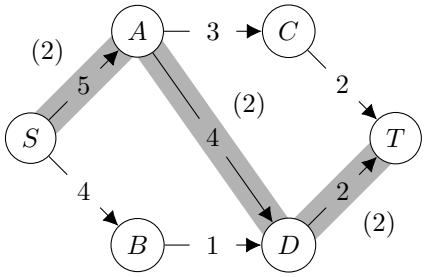


When adding flows, we must respect edge capacities.

For example, we could not add these graphs if the magnitude of flow in the right graph above was 2, since the capacity of the top-right edge is 2, and $2 + 1 > 2$.

Problem 7:
Combine the flows below, ensuring that the flow along each edge remains within capacity.



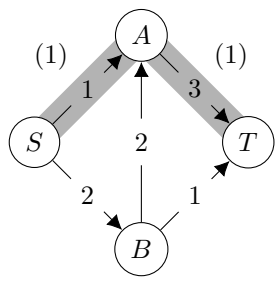


Part 4: Residual Graphs

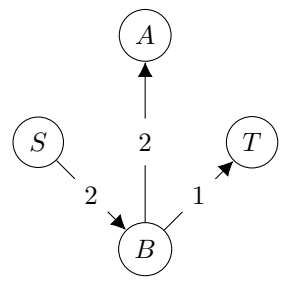
It is hard to find a maximum flow for a large network by hand.
We need to create an algorithm to accomplish this task.

The first thing we'll need is the notion of a *residual graph*.

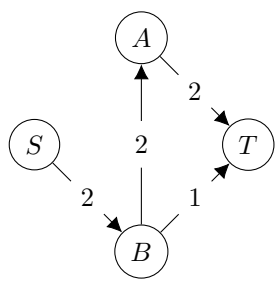
We'll start with the following network and flow:



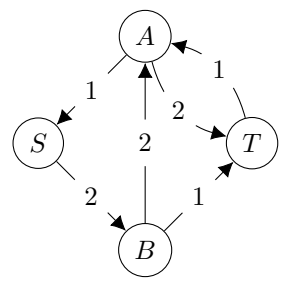
First, we'll copy all nodes and "unused" edges:



Then, we'll add the unused capacity of "used" edges: (Note that $3 - 1 = 2$)



Finally, we'll add "used" capacity as edges in the opposite direction:



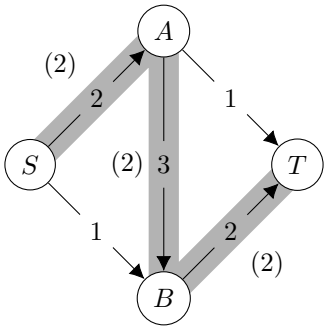
This graph is the residual of the original flow.

You can think of the residual graph as a "list of possible changes" to the original flow.
There are two ways we can change a flow:

- We can add flow along a path
- We can remove flow along another path

A residual graph captures both of these actions, showing us where we can add flow (forward edges) and where we can remove it (reverse edges). Note that "removing" flow along an edge is equivalent to adding flow in the opposite direction.

Problem 8:
Construct the residual of this flow.



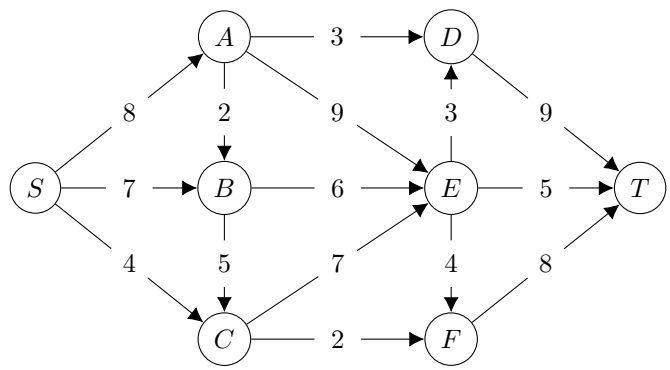
Problem 9:
Is the flow in Problem 8 maximal?
If it isn't, find a maximal flow.
Hint: Look at the residual graph. Can we add flow along another path?

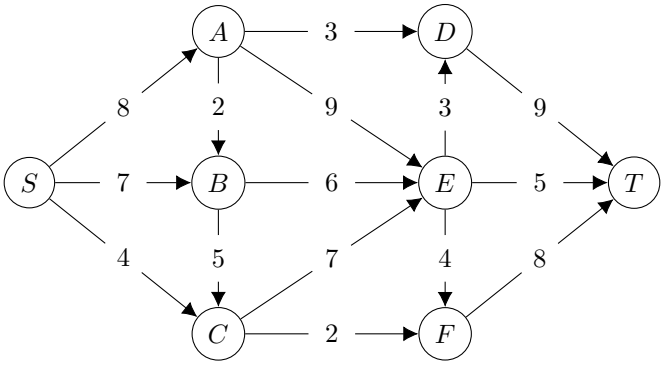
Part 5: The Ford-Fulkerson Algorithm

We now have all the tools we need to construct an algorithm that finds a maximal flow. It works as follows:

```
00 Take a weighted directed graph G.
01 Find any flow F in G
02 Calculate R, the residual of F.
03     If S and T are not connected in R, F is a maximal flow. HALT.
04 Otherwise, find another flow F_0 in R.
05 Add F_0 to F
06 GOTO 02
```

Problem 10:
Run the Ford-Fulkerson algorithm on the following graph.
There is extra space on the next page.




A
3
D
8
2
9
3
9
S
7
B
6
E
5
T
4
5
7
4
8
C
2
F

Problem 11:
Show that...

1. A maximal flow exists in every network with integral edge weights.
2. Every edge in this flow carries an integral amount of flow

Problem 12:
You are given a large network. How can you quickly find an upper bound for the number of iterations the Ford-Fulkerson algorithm will need to find a maximum flow?

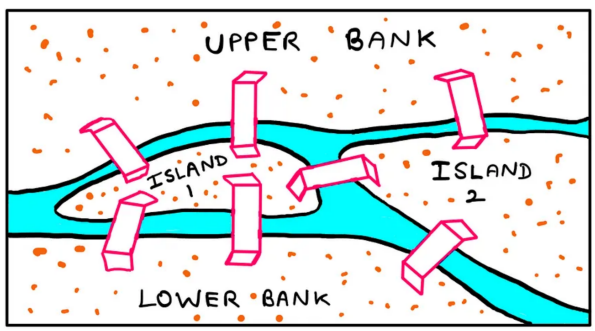
Problem 13:
Show that in any network the size of the maximal flow is equal to the sum of capacities in the minimal cut. This is called a max-flow min-cut theorem.

Part 6: Additional problems

Problem 14:
Find the network corresponding to the Problem 3. *Hint:* The vertices in the network correspond to the connected components, and the edges correspond to the valves. Since the graph is undirected, each edge should be represented in both directions.

Game theory
In the game of rock-paper-scissors, a player has three possible moves: rock, paper, or scissors. A probabilistic strategy in this game means that a player randomly chooses between these three moves with specific probabilities.
Suppose one player chooses a probabilistic strategy with equal probabilities, while the other always chooses rock. The player who uses a probabilistic strategy will win 1/3 of the time (when they choose paper), lose 1/3 of the time (when they choose scissors), and tie 1/3 of the time (when they choose rock). Using a probabilistic strategy in games can introduce an element of unpredictability and make it harder for opponents to predict the player's next move.

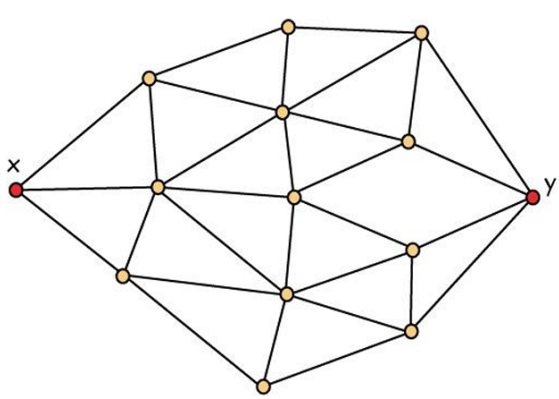
Problem 15:
Troll of Königsberg wants to eat Leonard Euler on his way from Island 1 to Island 2.



Euler can choose any path from Island 1 to Island 2 and the troll can choose any bridge to sit under.

1. Prove that Euler has a probabilistic strategy which allows him to be eaten with probability $\leq \frac{1}{3}$ regardless of troll's strategy.
2. Prove that Troll has a probabilistic strategy that allows him to eat Euler with probability $\geq \frac{1}{3}$ regardless of Euler's strategy.

Problem 16:
Prove the same for the new scheme and vertices x and y.

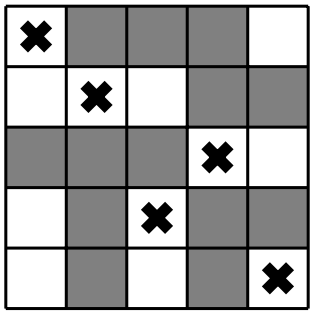


Problem 17:
Prove that for any graph, the maximal probability for the troll is equal to the minimal probability for Euler.

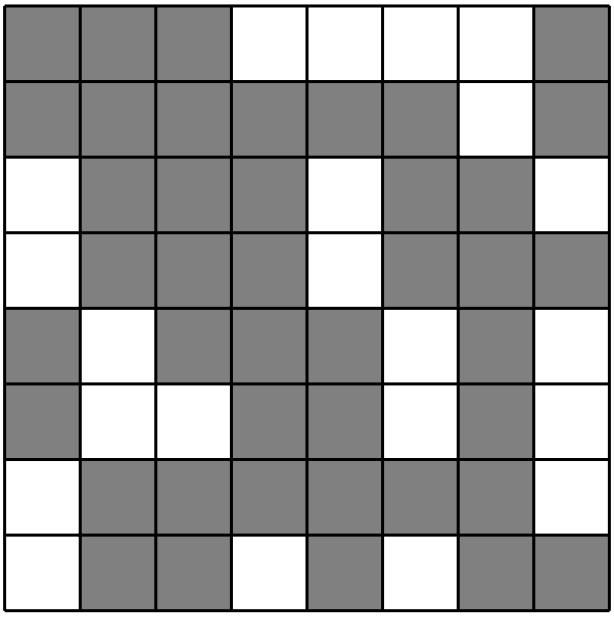
Part 7: Crosses

You are given an $n \times n$ grid. Some of its squares are white, some are gray. Your goal is to place n crosses on white cells so that each row and each column contains exactly one cross.

Here is an example of such a grid, including a possible solution.



Problem 18:
Find a solution for the following grid.



Problem 19:
Turn this into a network flow problem that can be solved with the Ford-Fulkerson algorithm.