Timur Garifullin
Due: 11:59pm Thursday, Feb 22

Homework 5

Spring 2024, Bhakta
CS 315, Algorithms

This assignment covers the network flow problem and the Ford-Fulkerson / Edmonds-Karp algorithms for finding the maximum flow.

For these questions, you may need to submit graph drawings. You should either

- Learn how to use the TikZ markup to create graph drawings in LaTeX. This isn't too hard (there are examples in this homework), but is harder than the other options.
- Draw a graph using other drawing software, or draw it by hand and take a picture. Then use the includegraphics command to add the resulting image to your homework pdf file.

0. (0 points) (NOT OPTIONAL) Who did you work with on this homework? What sources did you consult? How long did this assignment take?

Solution: Consulting: Artur, Konstantinos
Resources: Blackboard
Time: 25 hours

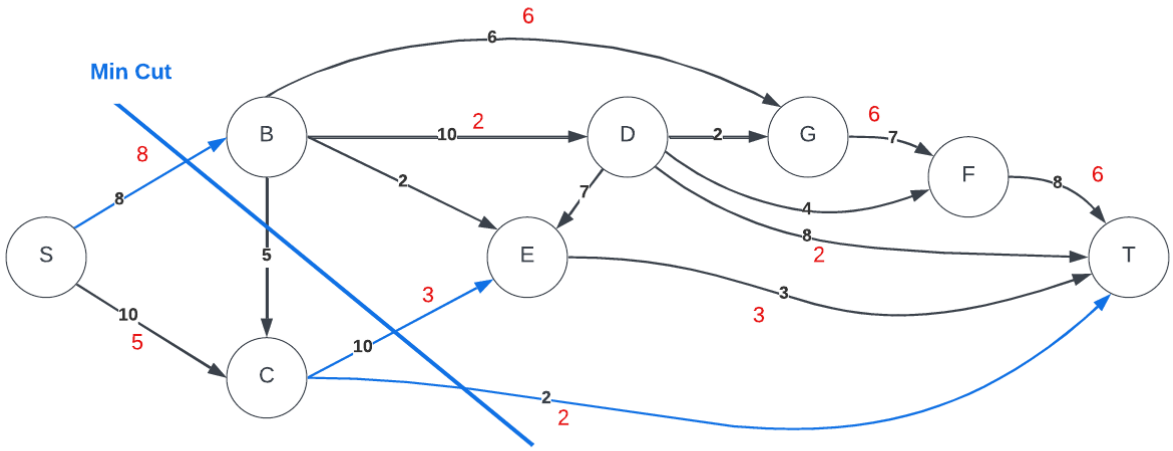
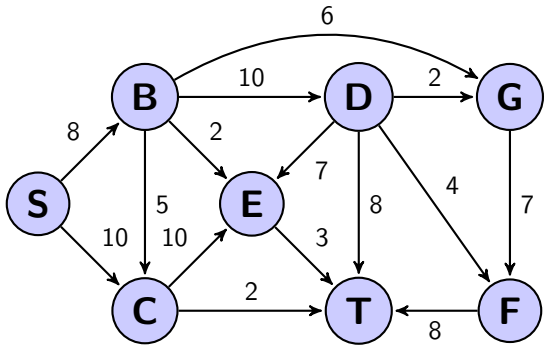


Figure 1: The maximum amount of flow in the network graph

1. (5 points) Find a max-flow from s to t in the below network flow capacity graph by any means (you don't have to follow the steps of Ford Fulkerson, Edmonds Karp, or any algorithm), and prove that it is optimal by finding a corresponding s-t min-cut with the same value. Draw both the flow graph, and give the min-cut.



Solution:
1) The figure above shows the optimal paths through the network graph. The maximum amount of flow achieved through all the paths is 13.
2) The corresponding s-t min-cut was found (marked on the graph), and it also has a capacity of 13.
Because statements 1) and 2) above have both resulted in the capacity of 13, this proves that this is an optimal max-flow from s to t i in the network graph.

2. (OPTIONAL) Say you are given a flow graph G with multiple sources and sinks (none of the sources are also sinks). The flow from any source can go into any sink, and the capacities on the edges bound the total flow through that edge from all sources to all sinks. Give an algorithm that can find the maximum total flow from all sources to all sinks in this graph.

   Hint: Modify G appropriately.

   Solution:

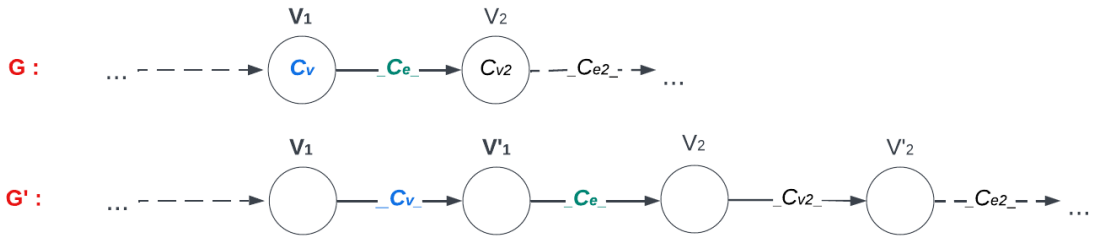


Figure 2: Graphs G and G' for exercise 3

3. (10 points) Say you are given a directed flow graph G with capacities c_e not only on edges (the max amount of flow that can pass through that edge), but also c_v on vertices (the max amount of flow that can pass through that vertex). Give a max-flow based algorithm that can find the maximum total flow in this graph with edge and vertex constraints. Justify the correctness of your algorithm and provide a bound on its running time in terms of the number of vertices and edges $|V|, |E|$ of the *input* graph.

   Hint: Construct a new graph G' appropriately.

Solution:

- Construct a new graph G' following these rules:
  - For each vertex V_1 with weight c_v, create a new "prime" vertex V_1'.
  - Insert each vertex V_1' after each vertex V_1, and connect them with an edge.
  - Assign each weight c_v of each vertex V_1 as a weight on the EDGE that connects vertices V_1 and V_1'.
  - Connect each vertex V_1' with neighbor vertex V_2 and assign the weight of c_e to the connecting edge.
  - Repeat the above step for each vertex that has a weight $c_v n$

  The reason we should construct the graph G' with the rules above is because these rules create a "standard" graph (only edges have weights). Because this is a "standard" graph, it is compatible with a known-to-us max-flow algorithm.

- Run Edmonds Karp max-flow algorithm on the graph G'

Runtime of the algorithm:
- Constructing a new graph G' : $O(|V| + |E|)$
- Running Edmonds Karp algorithm on a new graph G' : $O(|V||E|^2) = O(2|V|(|E| + |V|)^2)$
(2 before $|V|$ comes from the fact that one vertex on G' is now two vertices in G')

Total Runtime of our algorithm for exercise e : $O(2|V|(|E| + |V|)^2)$

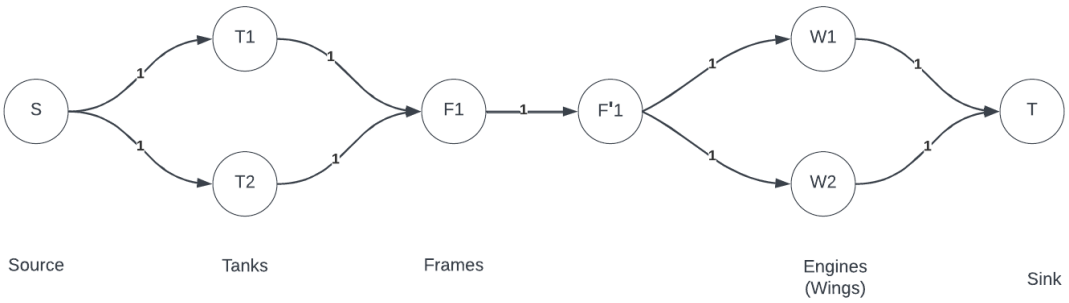


Figure 3: Flow graph G' for the exercise 4

4. (15 points) You are trying to assemble model planes. Each plane is made of one frame, one engine, and one gas tank. You have a bunch of spare frames, engines, and tanks, and your goal is to build as many model planes as possible. However, not all these parts fit togehter. Each frame f will only fit some subset of the engines and some subset tanks. Each frame has its own compatibilities. Engines and gas tanks don't touch each other, so don't get in each other's way.

   Your input is the sets of frames F, wings W and tanks T. You are also given, for each frame f, the set of compatible tanks $T_f \subseteq T$ and the set of compatible engines $E_f \subseteq E$.

   Design an algorithm to determine the maximum number of model planes that you can construct from your parts. Justify the correctness of your algorithm and provide a bound on its running time in terms of $|F|, |W|, |T|$.

   Hint: Make sure that your proposed algorithm will work when given one frame, two tanks, two engines, and the frame is compatible with both tanks and both engines. There is a common bug that this example should help you fix.

Solution: The algorithm for the problem above is as follows:

- Construct a new graph G' following these rules:
  - We connect the source to each tank
  - Each set of tanks $T_f \subseteq T$ connects with a compatible frame f_1
  - Following rules as in exercise 3, we create a new vertex f' for each compatible frame f_1
  - Each frame f' connects with the set of compatible engines $E_f \subseteq E$.
  - Each engine connects with a sink.

  Note that each weight of a vertex in the graph G' is 1. Placing 1 prevents the algorithm from selecting more than one of each part (tank, frame, engine). This avoids a common bug stated in the "Hint" section above.

  The reason we should construct the graph G' with the rules above is so that we can make sure that each compatible set of tanks $T_f \subseteq T$ AND a compatible set of engines

$E_f \subseteq E$ is connected to an appropriate (compatible) frame f. For this reason, the sea frames are placed in the middle of the graph G', after tanks and before engines.

- Run Edmonds Karp max flow algorithm on the graph G'

Runtime of the algorithm:
- Constructing a new graph G' : $O(|V| + |E|)$
- Running Edmonds Karp on the graph G : $O(|V||E|^2) = O(|V||E|^2) = O((|E| + 2|F| + |T| + 2) \times (|E| \times |F| + |F| \times |T| + |E| + |T| + |F|)^2)$

This is because:
Vertices: we have $|V| = |E| + 2|F| + |T| + 2$
(engine vertices, frames vertices $\times 2$, tank vertices, 1 source, 1 sink)
Edges: $|E| = |E| \times |F| + |F| \times |T| + |E| + |T| + |F|$
($|E|$ connected from source to all engines, each engine is connected to compatible frames |F|, each frame is attached to compatible tanks |T|)

Total Runtime of our algorithm for exercise 4 : $O((|E| + 2|F| + |T| + 2) \times (|E| \times |F| + |F| \times |T| + |E| + |T| + |F|)^2)$

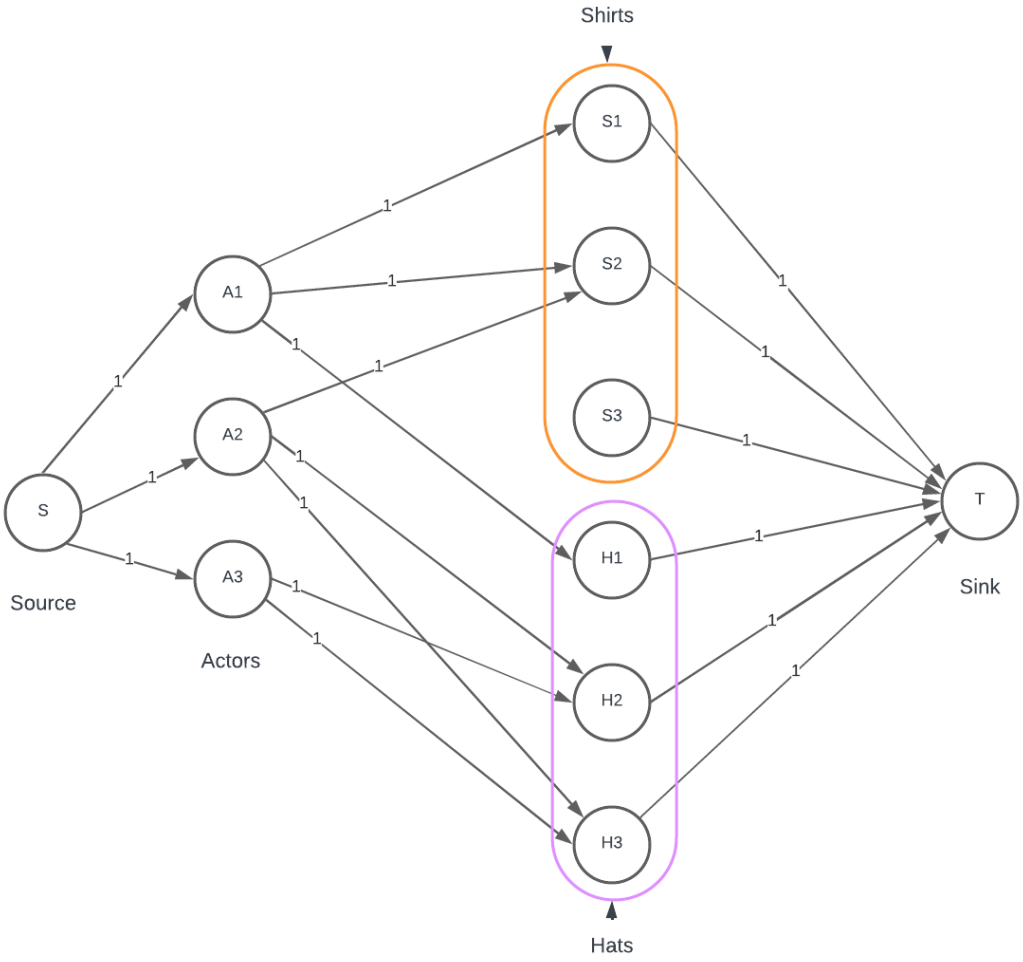


Figure 4: Flow graph G' for exercise 5

5. (20 points) You are doing costumes for a play. You have n actors, and have found n acceptable shirts S and n acceptable hats H. You need to assign each actor one shirt and one hat for the play.

   To you, it doesn't really matter who wears which of these shirts and hats, as they are all extras. However, your actors are picky about what they will wear on stage. Each actor i only likes some subset of the shirts S_i, and hats H_i, and each actor has their own preferences.

   An actor is *Happy* if they like *at least one* of their assigned shirt or hat.

   Design an algorithm to assign shirts and hats to maximize the number of happy actors. Justify the correctness of your algorithm and provide a bound on its running time in terms of n.

   Hint: If an actor is already happy because of one article of clothing, the other article of clothing doesn't matter. Also, be careful with the edge constraints.

Solution:
Graph to showcase the solution is on the next page.

Connect the Source with all the actors.

Connect all the actors with their respective subsets of hats and shirts that they *like*. Each actor has their own preferences.

Connect all of the hats and shirts with the Sink. This will tell us which items were picked by which actor. This will also let us know which items the actors still need (a hat or a shirt) and which items were not picked by anyone (leftovers).

Randomly choose an item from the leftovers that every actor is lacking and assign those items to them. This should result in every actor having a hat and a shirt, at least one of the two items is their preference.

In order to be happy an actor needs to *like* (prefer) at least one item out of the two. Letting each actor pick only one item that they *like* (prefer) increases the chances for every actor to be happy. Therefore we set all the edges in the max flow graph to have a weight of 1. This allows the actors to pick only one item that they *like* (prefer), which fulfills the condition of making an actor happy. Therefore, it does not matter if they *like* (prefer) the second item or not, thus we can randomly assign the second missing item based on the first item (if the actor chose a shirt that they *like*, then the algorithm will assign a random hat to that actor or vice versa).

Runtime bound:
Constructing the graph G the runtime is $O(3n + 2)$, which is the total number of actors, shirts and hats plus the source and the sink.
Running Edmonds-Karp algorithm on G the runtime is $O((3n + 2n^2)n)$
Total runtime is $O((3n + 2) + (3n + 2n^2)n)$,
Therefore $O(n^3)$

6. (50 points) You are programming a fleet of n self-driving taxis in your city C. You are given a weighted graph representing your city. The edges are roads, the vertices are points where edges connect or end, and the weight w_e on edge e is the length of the road e connecting the endpoints.

   By city law, your cars aren't allowed to self-drive on the road at night. Because your cars are electric, they must get to a charging station parking spot every night so that they can charge overnight.

   It is now almost night, and it is time to send your cars to charging stations. Your goal is to write a program that can send as many cars as possible to an available charging station spot. For the cars that don't make it to a charging station, humans will have to help retrieve the car with a portable battery, and this is expensive.

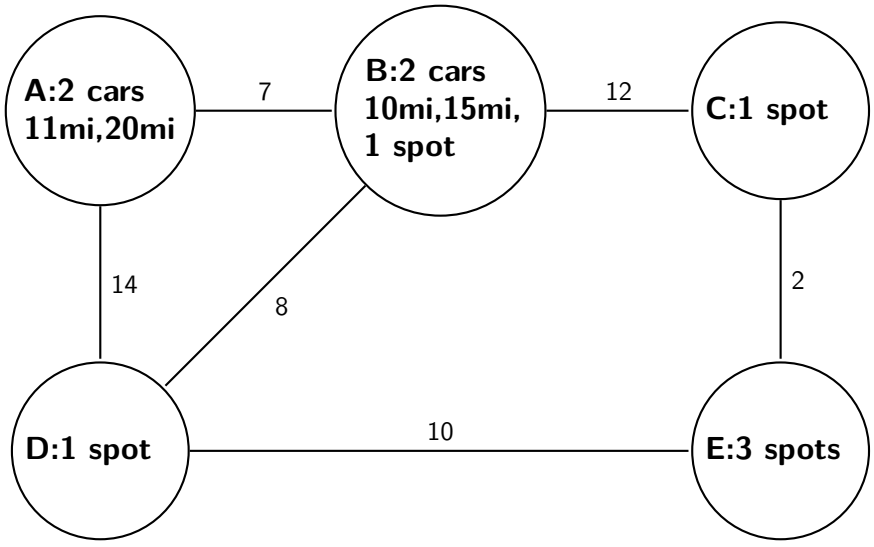
   Marked on your city graph, you are given the vertex locations of all the self-driving cars, the total distance that each car can travel before running out of charge, and the number of available parking spots at each vertex.

   You need to output the maximum number of cars that you can send to a charging station (to minimize the number of cars the humans have to retrieve.) The humans are paid per car, not by distance, so it doesn't matter where you leave the cars that don't end up at a charging spot.

   Importantly, the following reasonable, greedy algorithm is NOT optimal:

```
For each car c (in order of remaining travel distance):
    Find the closest station with at least 1 available parking space.
    If the car can travel the distance to that station:
        Send the car to that station
        Decrease the number of available spaces at that station by 1.
    Else:
        Give up on this car, continue to the next one.
```

   Think about the above approach vs optimal on this example graph.



   Variants of the above greedy approach will almost certainly also have counterexamples, that you are welcome to find.

(a) (20 points) Describe an algorithm that *can* correctly find the maximum number of cars that you can get to a charging spot.
Justify its correctness and analyze its running time in terms of the number of vertices, edges, cars, stations, etc., as appropriate.
Hint: There are *two* different graph algorithms that you've learned about that you will need here.

Solution:

```
From the input get the number of vertices, edges, and cars in the city
Read the input and store the values as n, m, and c respectively

From the input get the user for the number of available charging spots
at each vertex and store them in a list called spots

Initialize a dictionary called roads to represent the graph, where each
vertex maps to its neighboring vertices and distances
for each edge in range(m):
    Read input for vertex xi, vertex yi, and distance d
    if xi is not in the neighbors of yi:
        Add xi as a neighbor of yi with distance d
    if yi is not in the neighbors of xi:
        Add yi as a neighbor of xi with distance d

Create an empty dictionary called cars to store the starting vertices
and remaining mileage for each car
for each car in range(c):
    Read input for the starting vertex and remaining mileage of the car
    Add the car to the dictionary cars with the starting vertex as the
    key and remaining mileage as the value

Create a new dictionary called nfg to represent the network flow graph
Initialize the source node 's' with an empty dictionary
for each car in range(c):
    Add an edge from the source node 's' to the car node with a capacity
    of 1

for each car in cars:
    Extract the starting vertex and remaining mileage of the car
    Compute the shortest paths from the starting vertex to all other
    vertices using Dijkstra's algorithm
    for each reachable vertex within the car's remaining mileage and
    with available charging spots:
        Add an edge from the car node to the charging station node in
        the network flow graph

for each charging station in range(1, n):
    Add an edge from each charging station node to the sink node 't'
```

```
    with a capacity corresponding to the available parking spots

Run the Edmonds-Karp algorithm on the network flow graph to find the
maximum flow
Print the maximum flow, indicating the maximum number of cars that can
be sent to a charging station
```

Runtime bound:
n vertices, m edges, and c cars.
We run Dijkstra's algorithm for each car hence the runtime: O(c * (n + m) log n)

Constructing the graph:
|V| = source s, sink t, c car vertices and n charging stations
Thus, |V| = c + n + 2
|E| = c * n edges as a result of Dijkstra's algorithm output.
Each station vertex n is connected to the sink and each car vertex c to
the source, so |E| = c * n + n + c.
Thus, with Edmonds-Karp algorithm,

The total runtime is:$O(|V||E|^2) = O((|c + n + 2|)(|c * n + n + c|)^2)$

(b) (30 points) Write working code that will solve this problem.

Your code must use the following input/output format using stdin/stdout.

Input Format:

The input will start with three numbers: n,m,c giving the number of vertices n in the graph, number of edges m in the graph, and the number of cars c in the city. The vertices are numbered 0 to $n-1$.

The next line will contain n numbers, giving the number of available charging spots at each vertex i, in order.

The next m lines will contain three numbers each $x_i y_i l_i$. This indicates that there is a road from vertex x_i to vertex y_i of length l_i. This is an undirected edge (your city has two-way streets).

The next c lines will contain two numbers each $v_j d_j$. This indicates the vertex where car j is currently located, and the distance d_j it can travel with its remaining charge.

Sample Input (representing the above problem):

```
5 6 4
0 1 1 1 3
0 1 7
0 3 14
1 2 12
1 3 8
2 4 2
3 4 10
0 11
0 20
1 10
1 15
```

Output Format:

Output a single integer indicating the maximum number of cars you can get to a charging station.

Sample Output:

```
4
```

Submit this code in a separate file.

Where did you source any algorithms/data structures/ideas that you used?

Solution: Dr. Prateek Bhakta, CMSC 315 slides

7. (10 points) (Bonus) You are the royal political mastermind for the nation of Computopia, and are trying to maximize your nation's wealth through diplomacy. Every other country $c \in \{1, \ldots, n\}$ that you sign an treaty with will either give you tribute or demand tribute, which will cause a net impact of $w_c \in \mathbb{Z}$ on your wealth. Due to their own political dealings, each country c also has a list $l_c \subseteq \{1, \ldots, n\}$ of other countries that they demand you also sign treaties with, if want to sign an agreement with c. These lists are not necessarily symmetric, that is if 1 demands that you sign a treaty with 3, it may not be true that 3 will demand that you sign a treaty with 1.

   Find an algorithm that can determine the optimal set of countries to sign treaties with to maximize your country's wealth. Give a proof of correctness.

   Hint: Think about this as a directed min-cut problem. Create a node for each country, and a source node and sink node. Add infinite capacity arcs to represent the prerequisites for each country. The countries on one side of the min cut will be the ones you sign with. Remember that the directed edges that count in an s-t min cut are only the ones that go from the side with s to the side with t.

   Solution:

8. (BONUS) You have to give clothes to n people with n available shirts, and n available pairs of jeans. You have to assign each person one shirt and one pair of jeans. The shirts and jeans are not identical, but they have patterns. Each person tells you which subset of the shirts and jeans they like. A person is "happy" with your choice if they like *at least* one of the two items of clothing. Your goal is to assign clothes to people in a way that maximizes the number of happy people. Design an appropriate algorithm, with justification of correctness and analysis of running time, that will help you find this.