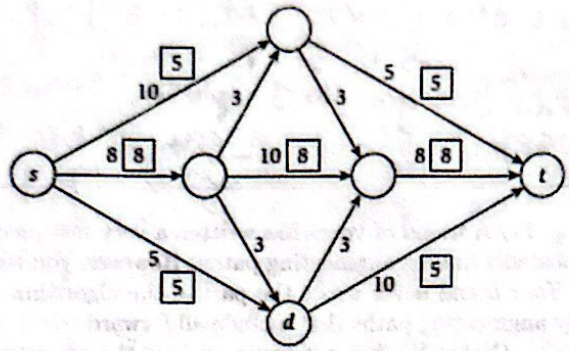
CS 577 **Assignment 5 – Network Flow** Summer 2023

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Gurusharan Kuruvath Wisc id: kuruvath

Network Flow

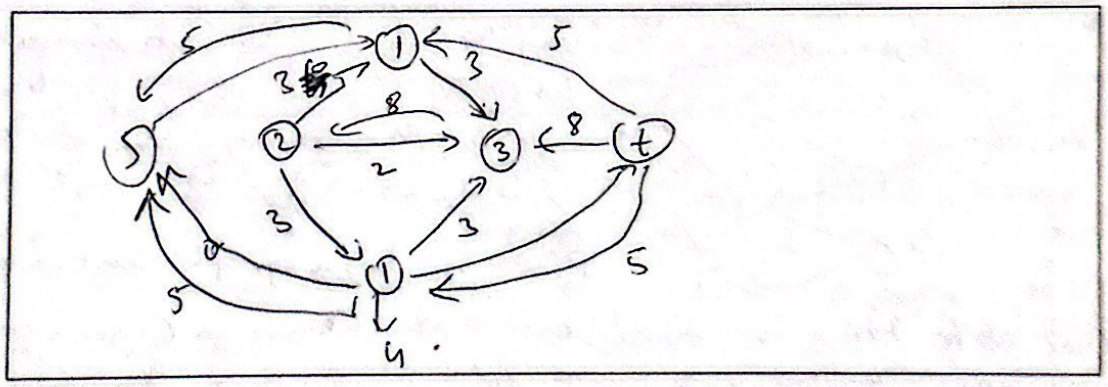
1. *Kleinberg, Jon. Algorithm Design (p. 415, q. 3a)* The figure below shows a flow network on which an $s-t$ flow has been computed. The capacity of each edge appears as a label next to the edge, and the flow is shown in boxes next to each edge. An edge with no box has no flow being sent down it.



(a) What is the value of this flow?

$5+8+5=18$

(b) Please draw the **residual graph** associated with this flow.



(c) Is this a maximum $s-t$ flow in this graph? If not, describe an augmenting path that would increase the total flow.

flow 3 $s \to 1 \to 3 \to 2 \to 4 \to t$

2. *Kleinberg, Jon. Algorithm Design (p. 419, q. 10)* Suppose you are given a directed graph $G = (V, E)$. This graph has a positive integer capacity c_e on each edge, a source $s \in V$, a sink $t \in V$. You are also given a maximum $s - t$ flow through G: f. You know that this flow is *acyclic* (no cycles with positive flow all the way around the cycle), and every flow $f_e \in f$ has an integer value.

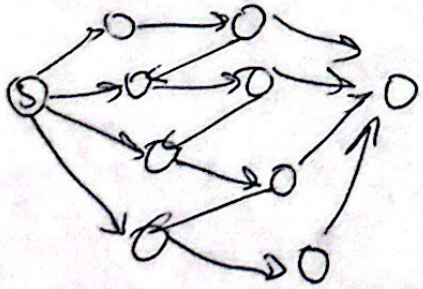
   Now suppose we pick an edge e^* and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting graph G^* in time $O(m + n)$, where $n = |V|$ and $m = |E|$.

The maximum flow M can stay unchanged or decrease by 1. Since we have f ① If $f(e^n) < Ce^n$ then stays unchanged. ② If $f(e^n) = Ce^n$, find a flow one path from t to s go through e^n in resulting graph. More specifically, consider $e^n = (v, v)$, Let $f'(e^n) = f(e^n) - 1$ then find flow one path $u \to v$, $t \to v$, adjust f to f'. Now we try to find some augmentary path from s-t with new residual graph of f' If we can, find some augmentary path from s-t with new residualgraph of f'.

3. *Kleinberg, Jon. Algorithm Design (p. 420, q. 11)* A friend of yours has written a very fast piece of code to calculate the maximum flow based on repeatedly finding augmenting paths. However, you realize that it's not always finding the maximum flow. Your friend never wrote the part of the algorithm that uses backward edges! So their program finds only augmenting paths that include all forward edges, and halts when no more such augmenting paths remain. (Note: We haven't specified *how* the algorithm selects forward-only augmenting paths.)

   When confronted, your friend claims that their algorithm may not produce the maximum flow every time, but it is guaranteed to produce flow which is within a factor of b of maximum. That is, there is some constant b such that no matter what input you come up with, their algorithm will produce flow at least $1/b$ times the maximum possible on that input.

   Is your friend right? Provide a proof supporting your choice.



My friend is false in general. Consider we go following graph with all edges. My friend would find ① → ① → ⋯ ① to have solution ① Part in blue graph best would be. We can add infinite new one (green path) to have higher maximum flow while my friends to find ① → ① ⋯ ② → ② → ①, still 1. So we can not find constant b to let it produce at least $\frac{1}{b}$ of the maximum flow.

4. *Kleinberg, Jon. Algorithm Design (p. 418, q. 8)* Consider this problem faced by a hospital that is trying to evaluate whether its blood supply is sufficient:

   In a (simplified) model, the patients each have blood of one of four types: A, B, AB, or O. Blood type A has the A antigen, type B has the B antigen, AB has both, and O has neither. Patients with blood type A can receive either A or O blood. Likewise patients with type B can receive either B or O type blood. Patients with type O can only receive type O blood, and patients with type AB can receive any of the four types.

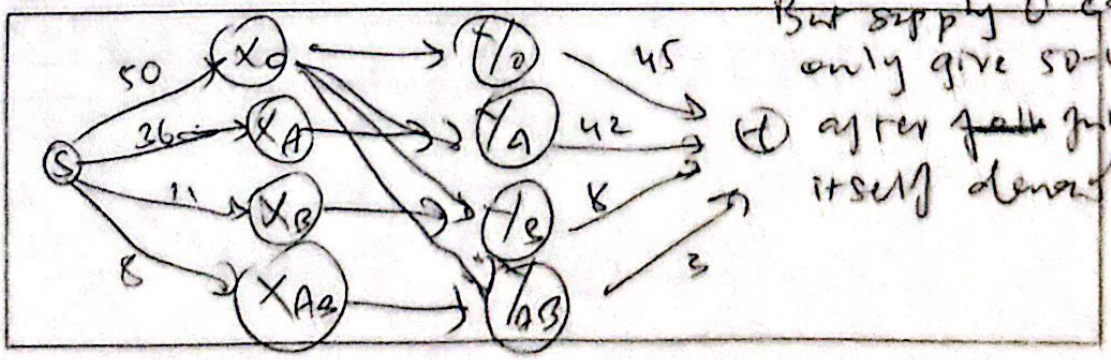
   (a) Let integers s_O, s_A, s_B, s_{AB} denote the hospital's blood supply on hand, and let integers d_A, d_B, d_O, d_{AB} denote their projected demand for the coming week. Give a polynomial time algorithm to evaluate whether the blood supply is enough to cover the projected need.

Input: $s_O, s_A, s_B, s_{AB}, d_O, d_A, d_B, d_{AB}$.
Output: Boolean variable
if $s_O < d_O$ return false
if $s_A + s_O - d_O < d_A$ return false
if $s_B + s_O - d_O < d_B$ return false
if $s_A + s_B + s_O - d_O < d_A + d_B$ return false
if $s_{AB} + (s_A + s_B + s_O - d_O - d_A - d_B) < d_{AB}$ return false
return true

   (b) Network flow is one of the most powerful and versatile tools in the algorithms toolbox, but it can be difficult to explain to people who don't know algorithms. Consider the following instance. Show that the supply is **insufficient** in this case, and provide an explanation for this fact that would be understandable to a non-computer scientist. (For example: to a hospital administrator.) Your explanation should not involve the words *flow*, *cut*, or *graph*.

blood type	supply	demand
O	50	45
A	36	42
B	11	8
AB	8	3

our supply can not fulfil demand A.
A can only receive O, B demand A must get 42-36=6 from supply O.
But supply O can only give 50-45=5 < 6 after fulfilling itself demand.



5. Kleinberg, Jon. *Algorithm Design (p.416 q.6)*. Suppose you're a consultant for the Ergonomic Architecture Commission, and they come to you with the following problem.

They're really concerned about designing houses that are "user-friendly", and they've been having a lot of trouble with the setup of light fixtures and switches in newly designed houses. Consider, for example, a one-floor house with n light fixtures and n locations for light switches mounted in the wall. You'd like to be able to wire up one switch to control each light fixture, in such a way that a person at the switch can see the light fixture being controlled.

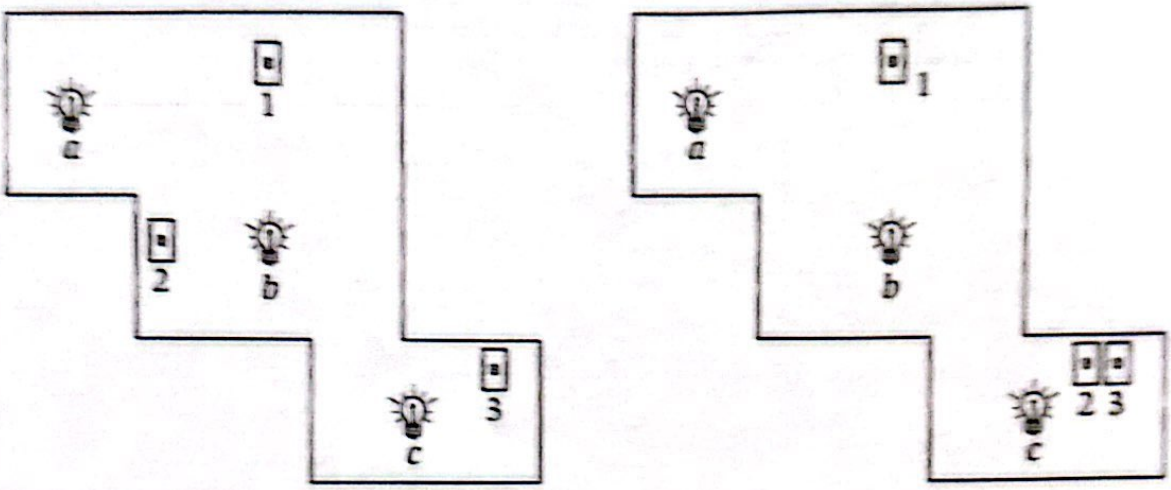


Figure 1: The floor plan in (a) is ergonomic, because we can wire switches to fixtures in such a way that each fixture is visible from the switch that controls it. (This can be done by wiring switch 1 to a, switch 2 to b, and switch 3 to c.) The floor plan in (b) is not ergonomic, because no such wiring is possible.

Sometimes this is possible and sometimes it isn't. Consider the two simple floor plans for houses in Figure 1. There are three light fixtures (labelled a, b, c) and three switches (labelled 1, 2, 3). It is possible to wire switches to fixtures in Figure 1(a) so that every switch has a line of sight to the fixture, but this is not possible in Figure 1(b).

Let's call a floor plan, together with n light fixture locations and n switch locations, ergonomic if it's possible to wire one switch to each fixture so that every fixture is visible from the switch that controls it. A floor plan will be represented by a set of m horizontal or vertical line segments in the plane (the walls), where the i-th wall has endpoints $(x_i, y_i), (x_i', y_i')$. Each of the n switches and each of the n fixtures is given by its coordinates in the plane. A fixture is visible from a switch if the line segment joining them does not cross any of the walls.

Give an algorithm to decide if a given floor plan is ergonomic. The running time should be polynomial in m and n. You may assume that you have a subroutine with $O(1)$ running time that takes two line segments as input and decides whether or not they cross in the plane.

we can use a graph-based approach. We will construct a visibility graph based on the given floor plan, and then check if there exists a perfect matching between the switches and fixtures in the graph. If such a matching exists, the floor plan is ergonomic; otherwise, it is not.

Algorithm:

1) Create an empty graph G.
2) For each switch, create a vertex in G.
3) For each fixture, create a vertex in G.
4) For each pair of switch and fixture vertices (u, v), check if the line segment joining them crosses any of the walls in the floor plan. If it does not cross any wall, add an undirected edge between u and v in G. This step requires the use of the subroutine that decides whether or not two line segments cross in the plane.
5) Use any polynomial-time algorithm to find a maximum cardinality matching M in G. This step ensures that we find a matching that covers as many fixtures as possible.
6) If the cardinality of M is equal to n, output "The floor plan is ergonomic". This means that there exists a perfect matching that covers all fixtures.
7) Otherwise, output "The floor plan is not ergonomic"

The runtime of this algorithm is dependent on the runtime of the subroutine that checks if two line segments across. Since the algorithm itself performs a constant number of operations for each switch-fixture pair, the overall runtime is polynomial in both the number of the line segments and the number of fixtures/switches

6. *Kleinberg, Jon. Algorithm Design (p.426 q.20).*

Your friends are involved in a large-scale atmospheric science experiment. They need to get good measurements on a set S of n different conditions in the atmosphere (such as the ozone level at various places), and they have a set of m balloons that they plan to send up to make these measurements. Each balloon can make at most two measurements. Unfortunately, not all balloons are capable of measuring all conditions, so for each balloon $i = 1, \ldots, m$, they have a set S_i of conditions that balloon i can measure. Finally, to make the results more reliable, they plan to take each measurement from at least k different balloons. (Note that a single balloon should not measure the same condition twice.) They are having trouble figuring out which conditions to measure on which balloon.

Example. Suppose that $k = 2$, there are $n = 4$ conditions labelled c_1, c_2, c_3, c_4, and there are $m = 4$ balloons that can measure conditions, subject to the limitation that $S_1 = S_2 = c_1, c_2, c_3$, and $S_3 = S_4 = c_1, c_3, c_4$. Then one possible way to make sure that each condition is measured at least $k = 2$ times is to have

- balloon 1 measure conditions c_1, c_2,
- balloon 2 measure conditions c_2, c_3,
- balloon 3 measure conditions c_3, c_4, and
- balloon 4 measure conditions c_1, c_4.

(a) Give a polynomial-time algorithm that takes the input to an instance of this problem (the n conditions, the sets S_i for each of the m balloons, and the parameter k) and decides whether there is a way to measure each condition by k different balloons, while each balloon only measures at most two conditions.

1) Construct a bipartite graph G with two sets of nodes: The left set representing the conditions and the right set representing the balloons.
2) For each condition node in the left set, connect it to all the balloon nodes in the right set that can measure that condition based on the input sets S_i.
3) Use any polynomial-time algorithm for finding a maximum matching in a bipartite graph to find a maximum cardinality M in G.
4) If the cardinality of M is equal to n, output "There exists a solution". This means that there is a matching that covers all conditions.
5) Otherwise, output "There is no solution".
The runtime of this algorithm is polynomial, as it depends on the runtime of the maximum matching algorithm used.

(b) You show your friends a solution computed by your algorithm from (a), and to your surprise they reply, "This won't do at all—one of the conditions is only being measured by balloons from a single subcontractor." You hadn't heard anything about subcontractors before; it turns out there's an extra wrinkle they forgot to mention...

Each of the balloons is produced by one of three different subcontractors involved in the experiment. A requirement of the experiment is that there be no condition for which all k measurements come from balloons produced by a single subcontractor.

Explain how to modify your polynomial-time algorithm for part (a) into a new algorithm that decides whether there exists a solution satisfying all the conditions from (a), plus the new requirement about subcontractors.

1) Construct a bipartite graph G with three sets of nodes: the left set representing the conditions and their copies for each subcontractor, the middle set representing the subcontractors, and the right set representing the balloons.
2) For each condition node in the left set, create a copy of that node for each subcontractor. Connect each copy of the condition node to all the balloon nodes in the right set that can measure that condition and are produced by a different subcontractor based on the input sets S_i.
3) Use any polynomial-time algorithm for finding a maximum matching in a bipartite graph to find a maximum cardinality matching M in G.
4) If the cardinality of M is equal to n, output "There exists a solution satisfying all conditions. This means that there is a matching that covers all conditions with balloons produced by different subcontractors.
5) Otherwise, output "There is no solution.

The runtime of this modified algorithm is still polynomial, as it depends on the runtime of the maximum matching algorithm used.

7. *Kleinberg, Jon. Algorithm Design (p.442, q.41).*

Suppose you're managing a collection of k processors and must schedule a sequence of m jobs over n time steps.

The jobs have the following characteristics. Each job j has an arrival time a_j when it is first available for processing, a length ℓ_j which indicates how much processing time it needs, and a deadline d_j by which it must be finished. (We'll assume $0 < \ell_j \leq d_j - a_j$.) Each job can be run on any of the processors, but only on one at a time; it can also be preempted and resumed from where it left off (possibly after a delay) on another processor.

Moreover, the collection of processors is not entirely static either: You have an overall pool of k possible processors; but for each processor i, there is an interval of time $[t_i, t'_i]$ during which it is available; it is unavailable at all other times.

Given all this data about job requirements and processor availability, you'd like to decide whether the jobs can all be completed or not. Give a polynomial-time (in k, m, and n) algorithm that either produces a schedule completing all jobs by their deadlines or reports (correctly) that no such schedule exists. You may assume that all the parameters associated with the problem are integers.

Example. Suppose we have two jobs J_1 and J_2. J_1 arrives at time 0, is due at time 4, and has length 3. J_2 arrives at time 1, is due at time 3, and has length 2. We also have two processors P_1 and P_2. P_1 is available between times 0 and 4; P_2 is available between times 2 and 3. In this case, there is a schedule that gets both jobs done.

- At time 0, we start job J_1 on processor P_1.
- At time 1, we preempt J_1 to start J_2 on P_1.
- At time 2, we resume J_1 on P_2. (J_2 continues processing on P_1.)
- At time 3, J_2 completes by its deadline. P_2 ceases to be available, so we move J_1 back to P_1 to finish its remaining one unit of processing there.
- At time 4, J_1 completes its processing on P_1. Notice that there is no solution that does not involve preemption and moving of jobs.

1) Sort the jobs in non-decreasing order of their deadlines.
2) Create an empty list of scheduled jobs
3) for each job j from the sorted list:
as I iterate over the available processors in increasing order of their availability intervals. b) for each processor i, check if it is available during the job's arrival time and its processing time. If it is, assign the job to the processor i and update its availability interval accordingly. c) If a processor is assigned to the job, add the job to the scheduled jobs list and break out of the loop. d) If the number of scheduled jobs is equal to n, output "There exists a schedule completing all jobs by their deadlines" and terminate.
e) Otherwise, output "No schedule".

8. *Kleinberg, Jon. Algorithm Design (p.444, q.45).*

Consider the following definition. We are given a set of n countries that are engaged in trade with one another. For each country i, we have the value s_i of its budget surplus; this number may be positive or negative, with a negative number indicating a deficit. For each pair of countries i, j, we have the total value e_{ij} of all exports from i to j; this number is always nonnegative. We say that a subset S of the countries is *free-standing* if the sum of the budget surpluses of the countries in S, minus the total value of all exports from countries in S to countries not in S, is nonnegative. Give a polynomial-time algorithm that takes this data for a set of n countries and decides whether it contains a nonempty free-standing subset.

1) Create a flow network G, where:
- set the source vertex s.
- set the sink vertex t.
- for each country i, create a vertex Vi.
- Create directed edges from s to each Vi with capacities equal to the budget surplus si if
- For each pair of countries i & j, create a directed edge for Vi to Vj with capa

2) Uses any polynomial-time maximum flow algorithm to find the maximum flow in G.

3) If the value of the maximum flow is non-negative, output "There exists a non empty free-standing subset." This means that there is a subset of countries that satisfies the free-standing condition.

4) Otherwise, output "There is no nonempty free-standing subset". This means that there is no subset of countries that satisfies the free-standing condition.

The runtime of the algorithm is polynomial, as it depends on the runtime of the maximum flow algorithm used. The maximum flow algorithm typically have a polynomial runtime complexity in the number of vertices & edges which is polynomial in n for this problem