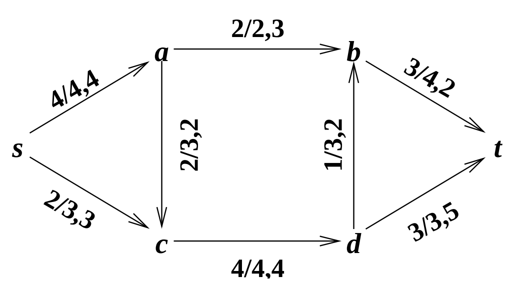
Due: In tutorial on April 7 by 12:10pm

1. [15 marks]

   In this question, we modify the standard Network Flow problem to deal with networks where there is a cost per unit flow associated with each edge of the network and we want to find a maximum flow with smallest total cost. In particular, each edge e is assigned a cost $d(e)$ (a positive integer) which indicates the cost per unit flow of sending flow along this edge (we use $d(e)$ to represent the cost of edge e because $c(e)$ already denotes the capacity of edge e; think of d as "dollars", for the cost). Given a maximum flow f, we define the cost of an edge e as $d(e)f(e)$ and the cost of f as the sum of the costs of the edges in the network, *i.e.*, $d(f) = \sum_{e \in E} d(e)f(e)$. Our problem is to find a maximum flow f that minimizes $d(f)$. To do this we introduce the concept of an *augmenting cycle* $\mathcal{C}$ which is a cycle in the corresponding undirected network with vertices $v_0, v_1, \ldots, v_{k-1}, v_k = v_0$, such that $\Delta_f(v_i, v_{i+1}) > 0$ for $i = 0, 1, \ldots, k-1$. (Note that $\Delta_f(v_i, v_{i+1})$ is the residual of edge (v_i, v_{i+1}) and is calculated in the same way as with augmenting paths for increasing flow, *i.e.*, you have to distinguish between forward and backward edges.) The residual of $\mathcal{C}$, denoted $\Delta_f(\mathcal{C})$ is the minimum of the residuals of the edges in $\mathcal{C}$.

   The cost of the cycle $\mathcal{C}$, denoted $d(\mathcal{C})$ is the sum of the costs of the forward edges minus the sum of the costs of the backward edges in $\mathcal{C}$. Our hope is to find an augmenting cycle that has negative cost. Once we have such a cycle $\mathcal{C}$, we augment f with respect to $\mathcal{C}$, yielding the new flow f'—for this question, you are expected to come up with your own reasonable definition of what it means to augment f with respect to $\mathcal{C}$. You will demonstrate your understanding of this notion in part (a) below.

   (a) Consider the network below and the maximum flow indicated for it. Each edge is labelled with three integers $i/j, k$ where i is the flow through the edge, j is the capacity of the edge and k is the cost of the edge. The maximum flow for this network is 6 and the total cost of the flow shown below is 71. Find a maximum flow that has the smallest total cost. Do this by iterating through the steps of finding an augmenting cycle of negative cost and augmenting the existing flow with respect to this cycle. This process continues until there are no augmenting cycles of negative cost.



   (b) Show that if an arbitrary network has maximum flow f and an augmenting cycle $\mathcal{C}$ of negative cost, then the flow f' that results from augmenting f with respect to $\mathcal{C}$ satisfies $|f'| = |f|$.

   (c) Show that if an arbitrary network has maximum flow f and an augmenting cycle $\mathcal{C}$ of negative cost, then the flow f' that results from augmenting f with respect to $\mathcal{C}$ satisfies $d(f') = d(f) + d(\mathcal{C})\Delta_f(\mathcal{C})$.

2. [10 marks]

   In the Traveling Salesman Problem (TSP), we are given a directed graph $G = (V, E)$ with an integer weight $w(e)$ for each edge $e \in E$, and we are asked to find a simple cycle over all the vertices (a "circuit") with minimum total weight. (Note that the weights $w(e)$ can be positive or negative.)

   Show how to represent an arbitrary instance of the TSP as an integer program. Justify that your representation is correct, and describe how to obtain a solution to the instance of the TSP from solutions to your integer program.

3. [15 marks]

   Your company has recently discovered that several of its problems can be solved using linear programming. Your company doesn't want to write their own solver program, since many efficient programs are available for sale. But your boss has been reading his spam again and went out and purchased the MELPSE system (Most Efficient Linear Program Solver Ever!) in a fit of misdirected leadership.

   As expected, the claim is slightly overstated, and the package comes with some serious limitations. From the advertisement:

   > MELPSE is the fastest and most streamlined LP solver ever! Using the latest technology, it will find the best non-negative values for all your variables, get the biggest value for your objective function, and it will even make sure that $\mathbf{A}x \leq b$!

   In fact, this is all that MELPSE can do: it only supports non-negative variables (it implicitly enforces a constraint $x_i \geq 0$ on all variables x_i), only allows maximizations of the linear objective function, and insists that all constraints are in the form $\sum_{i=1}^{n} a_{j,i} x_i \leq b_j$ (where $a_{j,i}$ and b_j are real numbers, perhaps negative).

   The linear programs you need to solve are usually minimization problems, where variables occasionally take negative values, and some constraints are expressed with equality or greater-than-or-equal. The boss has already spent your entire budget, so to save your team you will need to figure out how to use the MELPSE system to solve your problems.

   (a) One of your problems fits the restrictions of MELPSE except that you need to *minimize* your objective function. Describe precisely how to convert your program into one MELPSE can solve; that is, describe how to create an equivalent LP where the objective is being *maximized*. Briefly explain how to use a solution to your new program to find a solution to your original problem.

   (b) Another of your problems contains a constaint of the form

   $$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \geq b.$$

   Describe precisely how to convert this to a constraint of the form

   $$a'_1 x_1 + a'_2 x_2 + \cdots + a'_n x_n \leq b'$$

   as required by MELPSE.

(c) This problem also has a constraint of the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b.$$

Describe precisely how to create an equivalent LP that can be solved by MELPSE (or in the form of part (b)).

(d) From the previous parts we know how to solve a minimization problem with $\geq$ or $=$ constraints. We can assume that all constraints are in the form $a_1x_1+a_2x_2+\cdots+a_nx_n \leq b$.

But we have a problem where one of the variables, x_1, could be negative. Describe precisely how to construct an equivalent LP where we replace x_1 with two new variables which can only take non-negative values. Briefly explain how to use a solution to your new program to find a solution to your original problem.

(e) Write an efficient high-level algorithm that will convert a linear program that might be a maximization problem, might contain greater-than-or-equal or equality constraints, and may contain variables lacking a non-negativity constraint, into an equivalent linear program that can be solved by MELPSE. Give a good bound on the size (the size being the number of variables and number of constraints) of your new linear program and briefly justify it.

4. [15 marks]

Consider the Bin Packing problem: given positive integers K and B, and a list of integers $A = [a_1, \ldots, a_n]$ (where $0 < a_i \leq B$), determine if there is a way to partition A into at most K cells such that for each cell of the partition, the sum of the elements in the cell is at most B.

The FirstFit algorithm takes each object in turn (**note that we do NOT sort the objects by size**) and places it into the first bin that can accommodate it.

For example, suppose $B = 10$ and $A = [2, 2, 7, 8, 3, 6, 3, 2, 6]$. FirstFit would use 5 bins and fill them as follows: Bin(1) contains 2, 2, 3 and 3; Bin(2) contains 7 and 2; Bin(3) contains 8; Bin(4) contains 6; and Bin(5) contains 6.

Let $S = \sum_{i=1}^{n} a_i$.

(a) Argue that the optimal number of bins required is at least $\lceil S/B \rceil$.

(b) Argue that the FirstFit algorithm leaves at most one bin less than half full.

(c) Prove that the number of bins used by the FirstFit algorithm is never more than $\lceil 2S/B \rceil$.

(d) Prove an approximation ratio of 2 for the FirstFit algorithm.