# CS31005: Algorithms-II
# Programming Assignment 1
# Due: October 22, 2020

Consider a flow problem where, unlike the maximum-flow problem you have seen so far, there are no special vertices *s* (source) and *t* (sink). However, each vertex may produce some amount of an item or consume some amount of the item. In addition, the vertices allow items to flow through them for distribution to other vertices. Let $n(i)$ denote the amount that a vertex $i$ produces or consumes. Thus, if $n(i) = 0$, the vertex is neither a producer nor a consumer, the vertex is a consumer if $n(i) > 0$, and the vertex is a producer if $n(i) < 0$. The items produced are shipped along the edges to satisfy the needs of the consumers. The flow assignments along the edges should be such that in addition to the standard capacity constraints, if vertex $i$ is a producer then its total outflow must be greater than its total inflow by $|n(i)|$ at any time, if vertex $i$ is a consumer then its total inflow must be greater than its total outflow by $|n(i)|$ at any time, and the total inflow must be equal to the total outflow for all other vertices.

**Your input is:**

A directed graph with non-negative integral capacities on every edge, and need $n(i)$ on every vertex *i*. The graph will be given in a file in a specific format that your program will first read into an adjacency list. You can assume that the $n(i)$ values will be positive or negative integers or 0.

**Your program should output:**

An assignment of flow *f* on every edge subject to the above conditions if possible, or say that it is not possible. For example, as a trivial case, if all $n(i)$ are positive, no flow assignment is possible satisfying the above conditions (as there are no producers for the consumers).

Your program should contain the following types/functions etc. Please adhere strictly to the type/function names, function prototypes etc. so that your program runs properly when TAs test it. Do not use the structure field names for any other variable names in your program.

1. Define a C type called ***EDGE*** that represents one edge of a flow network. The structure that you typedef should store 3 fields: an integer $y$ storing the endpoint vertex $y$ of an edge $(x, y)$ (edge from x to $y$), an integer $c$ storing the capacity of the edge, and an integer $f$ storing the flow value to be assigned on the edge.
2. Define a C type called ***VERTEX*** to represent one vertex of the flow network. The structure that you typedef should contain 3 fields, an integer $x$ storing the id of the vertex, an integer $n$ storing the need value of the vertex, and a pointer $p$ storing a pointer to an ***EDGE*** node (basically $p$ will point to the first node in the adjacency list of the vertex).
3. Define a C type called ***GRAPH*** that stores the directed graph. The graph will be stored as an adjacency list. The C structure that you typedef should store 4 things: an integer $V$ storing the number of vertices, an integer $E$ storing the number of edge, and a pointer $H$ storing a pointer to an array of ***VERTEX*** nodes (basically $H$ stores a pointer to an array containing the vertices of the adjacency list, with each vertex pointing to its edge list).

4. Define a function ***GRAPH *ReadGraph(char *fname)*** that reads the graph from a file whose name is given in the null-terminated string *fname*. It should also initialize the fields appropriately. The function should return a pointer to the graph formed. The format of the input file is described at the end.
5. Define a function ***void PrintGraph(GRAPH G)*** that prints the graph *G*. A sample format for printing the graph is described at the end.
6. Define a function ***void ComputeMaxFlow(GRAPH *G, int s, int t)*** that takes as parameter a pointer to a graph *G*, the id of the source vertex s and the id of the sink vertex t, and computes the maximum flow on it. The function should fill up the *f* field in every edge in the graph with final flow value assigned on that edge when the maximum flow is found. In addition, it should print the value of the maximum flow with a nice message. You should use the Ford-Fulkerson method, with the constraint that at every step, you should choose the augmenting path with the maximum residual capacity among all shortest augmenting paths (i.e., find the set of all augmenting paths with smallest number of edges, and among them choose the one with the maximum residual capacity).
7. Define a function ***void NeedBasedFlow(GRAPH *G)*** that takes as parameter a pointer to a graph *G* and computes the flow values on every edge for the need-based flow. The function should fill up the *f* field in every edge in the graph with the flow assigned on that edge. This function must call the ***ComputeMaxFlow function()*** (so you should see how you can use the max flow problem to solve this, do not design a totally different algorithm of your own for this assignment, that will not be graded).
8. Design a main function that does the following exactly in this order:
    a. Read in the file name from the keyboard into a string *S* (you can assume that the maximum size of the file name is 20 characters)
    b. Call the function ***ReadGraph()*** to read and initialize the graph
    c. Call the function ***PrintGraph()*** to print the graph
    d. Read in the identifiers of the source and the sink (in that order)
    e. Call the function ***ComputeMaxFlow()*** to compute and print the maxflow in the graph, taking the ids read in as source and sink
    f. Call the function ***PrintGraph()*** to print the graph
    g. Call the function ***ReadGraph()*** again to read and initialize the graph (as the original graph has been changed by the *ComputeMaxFlow()* call)
    h. Call the function ***NeedBasedFlow()*** to compute the need-based flow in the graph
    i. Call the function ***PrintGraph()*** to print the graph

**Input File Format**

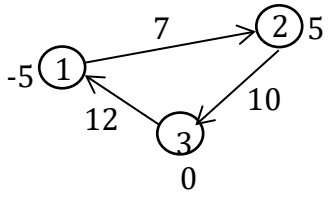
If the graph has *N* vertices, the vertices will be identified by the integers 1, 2, ….,*N*.

The first line of the file contains 2 integers, *N* followed by *M* (separated by spaces), where *N* is the number of vertices and *M* is the number of edges

The second line contains *N* integers, with integer *i* representing the need value of vertex *i*.

This is followed by *M* lines, with each line containing three integers *x*, *y*, *c* separated by spaces where *x* and *y* represent the vertex identifiers of the edge from *x* to *y*, and *c* represents the capacity of the edge.

For example, consider the following graph with 3 vertices and 3 edges. The identifiers for each vertex are shown inside the circle representing the vertex, and the need of the vertex is shown against it just outside the circle.




Then the graph will be represented in the file by the following lines:

3 3

-5 5 0

1 2 7

3 1 12

2 3 10

**Output Print Format**

Print the graph as a normal adjacency list in the following format, with one line for the edge list of each vertex 1, 2, 3,….,N

1 -> (*x*, *c*, *f*) -> (*x*, *c*, *f*)->….

2 -> (*x*, *c*, *f*) -> (*x*, *c*, *f*)->….

.

N -> (*x*, *c*, *f*) -> (*x*, *c*, *f*)->….

For example, if you want to find a need-based flow, the example graph will be printed as

1 -> (2, 7, 5)

2-> (3, 10, 0)

3-> (1, 12, 0)

You can verify that the flow assignments satisfy the constraints of a need-based flow for the example graph shown.