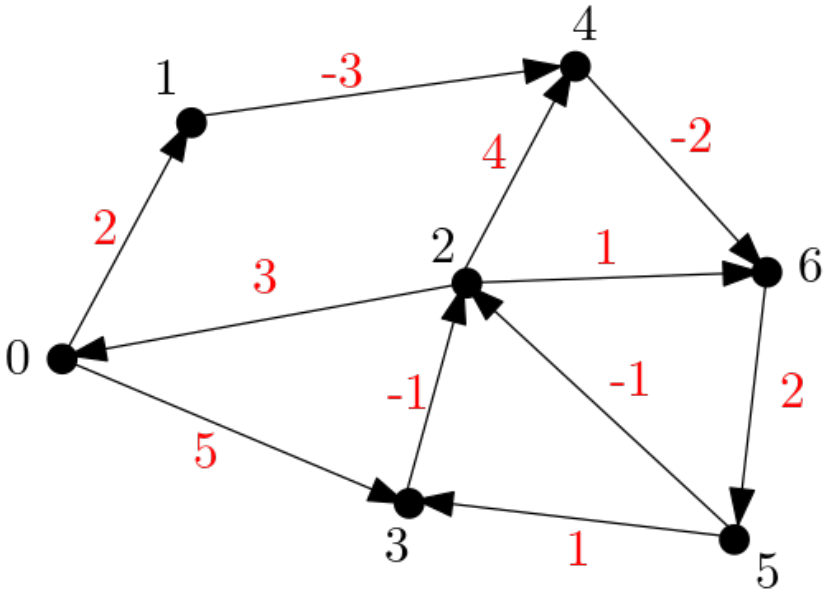
Analysis of Algorithms

Extra Credit Project – Bellman-Ford Algorithm

In this project, we will implement the Bellman-Ford algorithm to compute single source shortest paths in a graph that can have edge weights that are either positive or negative. In this project, we will assume that the graph is a directed graph. This project is worth up to 50 points that can be applied to the other 3 projects in the class. We covered this exact algorithm earlier in the class.

There are various ways to implement a graph data structure. **In this project, you should implement the graph with an adjaceny list representation.** If using java, I would recommend using ArrayLists to store the adjacency lists. You could do something very similar in python using lists. In C, you could either use a linked list data structure that you probably wrote for the data structures class, or you could read the entire file in once to count how many edges each vertex has, and then you can have the adjacency lists just be arrays of fixed size.

There are two input files for this project. The first is graphInput.txt which contains all of the information you need to construct the graph. The provided file gives the input for the following graph, but we will test your code on different graphs as well.



In this figure, the black numbers are the IDs of the vertices, and the red numbers are the weights on the edges. The graphInput.txt will first contain *n*, the number of vertices in the graph. You should assume that the vertices are labeled 0, 1, …, n-1. Then on each line there will be the information of a single edge of the graph in the following format:

startingVertex endingVertex edgeWeight

For example, the edge connecting vertex 0 to vertex 1 with a weight of 2 is given by "0 1 2".

The second input file is shortestPaths.txt which contains the shortest paths that we want to compute in the following format:

startingVertex destinationVertex

You then want to run the Bellman-Ford algorithm starting from *startingVertex* and output the length of the shortest path to *destinationVertex*. **You only need to report the length of a shortest path, not the actual path itself.** You then output in the following format:

A shortest path from *startingVertex* to *destinationVertex* has length *pathLength*.

So for example, if shortestPaths.txt has "0 4", you would want to run Bellman-Ford from 0 until we compute a shortest path to vertex 4. This path has length -1. So you would output:

A shortest path from 0 to 4 has length -1.

To keep the project a bit more simple, you can assume that a shortest path will always exist for the vertices we ask you to compute the paths for. Namely, you can assume there will not be a negative weight cycle in the graph, and that there will always be at least one path from the *startingVertex* to the *destinationVertex*.

How to code this project:

For the most part, you can code this project in any language you want provided that the language is supported on the Fox servers at UTSA. We want you to use a language that you are very comfortable with so that implementation issues do not prevent you from accomplishing the project. That said, we will be compiling and running your code on the Fox servers, so you need to pick a language that is already installed on those machines. See the PDFs on Blackboard in the Programming Projects folder for more information on how to connect to the Fox servers remotely (also covered in the Programming Project 1 overview lecture).

Since everyone is coding in their own preferred language, we are asking you to provide a bash script named *ExtraCredit.sh* that will act similarly to a makefile. I covered how bash scripts work in class in the Project 1 overview lecture, and I recorded a short follow-up to this here: https://youtu.be/CaIFJWiyU_U

In short, your bash script should contain the command to compile your code, and then on a different line, it should contain the line to execute your code. **In this project, we will only be using the graphInput.txt and shortestPaths.txt files, so these file names can be hard coded inside your program. No command line arguments are needed for this project.** So, the command to execute your code should look like this:

bash ExtraCredit.sh

Grading

We will grade according to the rubric provided on Blackboard. The majority of the points come from the following: did the student give a correct implmentation of the selction algorithm that runs in expected O(n) time that returns the correct answers for all possible inputs? Proper documentation may help the grader understand your code and earn you partial credit in the event you have some mistakes in the code.

Violations of the UTSA Student Code of Conduct will be penalized harshly. In particular, be very careful about sending code to a student who asks how you accomplished a particular task. I've heard this story several times recently: "They said they just wanted to see how to perform part X of the project. I didn't think they would submit my exact code." If this happens, you will both be penalized for cheating. To protect yourself and to more properly help your fellow student, send pseudocode, and not actual compilable code.

Also we know about the online sites where people upload projects and have a third party complete the project for you. This is a particularly egregious form of cheating (it's in the best interest of your career to not tolerate this). If you use a solution from one of these sites or submit a minor modification (minor is at the disgression of the instructor) of a solution from one of these sites, you will receive a 0 and will be reported to the university for a viloation of the UTSA Student Code of Conduct.

Submitting

Zip up your project folder and submit on the dropbox on Blackboard by the due date.