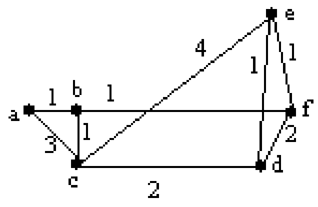
Due Monday 12/05/16

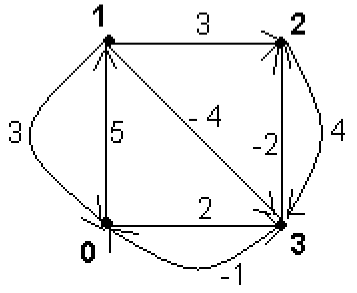
Greedy algorithm HW

On paper: #3, 4 (for the Instructor)
On paper: #1, 2, 5, 7 (for the TA)
Submit via Canvas: #6

(5 pts) 1. Trace Dijkstra's algorithm (break ties alphabetically) on the graph below with source node = a. Show the values for *p* and *IN* and the *d*-values and *s*-values for each pass through the while loop, as in Example 1.



(5 pts) 2. (a) Give the results after running Dijkstra's algorithm on the directed graph below with source node of 0. Just show the final d and s tables.



(b) Did Dijkstra's algorithm work correctly? Explain.

(10 pts) 3. (a) Prove that the single-source shortest path problem has no solution in a graph with a negative weight cycle.

(b) Prove that the shortest path in a graph with n nodes cannot be longer than n - 1 arcs.

(16 pts) 4. The Bellman-Ford algorithm also solves the single-source shortest path problem, but unlike Dijkstra's algorithm (see problem 2), it can work with negative-weight arcs as long as there is no negative-weight cycle (see problem 3). Because a shortest path has maximum n - 1 arcs, the algorithm computes in turn all shortest paths with number of arcs from 1 to n - 1. (Because arcs can have negative weights, adding more arcs to a path could reduce the overall weight of the path.) The pseudocode is below; in this algorithm, the main diagonal adjacency matrix entries A[i, i] must be 0. Also, the notation M[v, i] represents the shortest path from x to v that uses no more than i arcs. The big recurrence relation (highlighted) to find M[v, i] expresses the following idea:

For any vertex $z \neq x$, you know the value of M[z, i-1], the length of the shortest path from x to z using i-1 or fewer arcs. Add to that A[z, v] and you have a path with one more arc from x to v. Find the shortest such path by trying all possible z's. Compare this with the shortest path of length i-1 from x to v, represented by M[v, i-1], and take the smaller value.

```
Bellman-Ford Algorithm (n × n matrix A; source node x)
Local variables:
2-D n × n-1 array M                //here entry M[v,i] represents the
                                   //shortest path from x to v
                                   //that uses no more than i arcs
indices i, j
vertices v, z

//initialize M array
for j = 0 to n - 1,
   set M[x,j] = 0                  //shortest path from x to x of any
                                   //number of arcs is
                                   //always length 0

for v = any vertex ≠ x
   set M[v,1] = A[x,v]             //shortest path of length 1 from x to v
                                   //is an arc from x to v

for i from 2 to n-1                //successively more arcs in paths
   for v = any vertex ≠ x
         M[v,i] = min(M[v,i-1], min_{all z ≠ x}(M[z,i-1] + A[z,v]))
```

(a) Is this a greedy algorithm or a dynamic programming algorithm? If the former, where in the algorithm are you being greedy? If the latter, what are the optimal subproblems?

(b) Use the Bellman-Ford algorithm on the graph of problem #2, again using 0 as the source node. Fill in the M[v, i] table below. Also fill in the d/s table.

M	1	2	3
0			
1			
2			
3			

	0	1	2	3
d				
s				

(c) Does the Bellman-Ford algorithm solve the shortest path problem correctly for this particular graph? Explain.

(d) Give an order-of-magnitude upper bound for the work done by the Bellman-Ford algorithm ***as written above***.

(5 pts) 5. a. Construct the Huffman tree for the following characters and percent frequencies:

Character	b	n	p	s	w
Frequency	6	32	21	14	27

b. Find the Huffman codes for these characters.

(15 pts) 6. Write a C++ program called *Huffman.cpp* to create Huffman codes. Program input is a file called *freq.txt* (make up your own file for testing) that contains data on the characters in some cleartext file in the form of each character's non-zero frequency of occurrence in the cleartext file. (Note that these are frequency counts, not percentages, and it doesn't matter if the values do not sum to 100.) You can assume that *freq.txt* contains only characters from the standard ASCII 128 character set and that the results are ordered by the ASCII integer for each character. The end-of-line character in a text file is a non-printable line feed character, with ASCII code 10, and will be written in the *freq.txt* file as LF. Thus the freq.txt file might look like

```
LF 2
. 1
M 7
c 3
d 5
e 14
f 2
```

etc.

Your program should create a "code table" that gives each character and its binary Huffman code. Save this information in an external file called *codetable.txt* where each line of the file is a character and its code, ordered by the ASCII value of the character.

Turn in your Huffman.cpp file via Canvas. If your program is incomplete or does not work properly, please note the details when you submit it.

(10 pts) 7. Here is a description of the *Interval Scheduling Problem*. You have a set S of n requests, S = {1, 2, …n}. Each request i corresponds to an interval of time that starts at s(i) and finishes at f(i). A subset A of S is compatible if no two requests overlap in their time intervals. The problem is how to select a compatible subset A that contains the maximum number of requests. Because we want to maximize the number of requests (an optimization problem), we try a greedy strategy of filling A with requests that have the smallest time interval requirements so that we can fit the most in. Here is pseudocode for an algorithm using this strategy.

```
Set A to the empty set
While S is not empty
      Add to A the request i in S with the smallest time interval
      Remove all requests from S with time intervals incompatible with i
Return A
```

Either prove that this algorithm always gives an optimal solution or give an example where it fails to give an optimal solution.