Warren Hammock

CS520 Homework 2

2/6/18

1. (10 pts) Consider this instance of the knapsack problem. The weights are: 20, 24, 14, 20, 18, 20, 10, 6 and the prices are: 15, 9, 27, 12, 36, 12, 9, 12 while the capacity is M = 80. Apply the greedy method algorithms separately (1-0 and fractional) to find the optimal solution. For full credit you must show the process of 2 deriving the solution step-by-step.

\*\*\*I had completed this question in program form if you would like to see it. I have attached it. The output of the files are here side by side for comparison.

Zero and One Algorithm

Starting List:

ItemList is :

Item: 1 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 2 Weight: 24.0 Price: 9.0 Value: 0.375

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 4 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 5 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 6 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 7 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 8 Weight: 6.0 Price: 12.0 Value: 2.0

KnapsackList is :

After Sorting by Value in Desending Order:

ItemList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 4 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 5 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 6 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 7 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 8 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

ItemList is :

Item: 1 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 2 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 3 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 4 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 5 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 6 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 7 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Knapsack Weight is: 6

ItemList is :

Item: 1 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 2 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 3 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 4 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 5 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 6 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Knapsack Weight is: 24

ItemList is :

Item: 1 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 2 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 3 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 4 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 5 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Knapsack Weight is: 38

ItemList is :

Item: 1 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 2 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 3 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 4 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 4 Weight: 10.0 Price: 9.0 Value: 0.9

Knapsack Weight is: 48

ItemList is :

Item: 1 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 2 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 3 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 4 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 5 Weight: 20.0 Price: 15.0 Value: 0.75

Knapsack Weight is: 68

Left Over Space: 12.0

Fractional Knapsack

Starting List:

ItemList is :

Item: 1 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 2 Weight: 24.0 Price: 9.0 Value: 0.375

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 4 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 5 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 6 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 7 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 8 Weight: 6.0 Price: 12.0 Value: 2.0

KnapsackList is :

After Sorting by Value in Desending Order:

ItemList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 4 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 5 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 6 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 7 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 8 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

ItemList is :

Item: 1 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 2 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 3 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 4 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 5 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 6 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 7 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Knapsack Weight is: 6

ItemList is :

Item: 1 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 2 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 3 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 4 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 5 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 6 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Knapsack Weight is: 24

ItemList is :

Item: 1 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 2 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 3 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 4 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 5 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Knapsack Weight is: 38

ItemList is :

Item: 1 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 2 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 3 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 4 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 4 Weight: 10.0 Price: 9.0 Value: 0.9

Knapsack Weight is: 48

ItemList is :

Item: 1 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 2 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 3 Weight: 24.0 Price: 9.0 Value: 0.375

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

Item: 4 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 5 Weight: 20.0 Price: 15.0 Value: 0.75

Knapsack Weight is: 68

ItemList is :

Item: 1 Weight: 20.0 Price: 12.0 Value: 0.6

Item: 2 Weight: 24.0 Price: 9.0 Value: 0.375

Item: 3 Weight: 8.0 Price: 4.800000000000001 Value: 0.6000000000000001

KnapsackList is :

Item: 1 Weight: 6.0 Price: 12.0 Value: 2.0

Item: 2 Weight: 18.0 Price: 36.0 Value: 2.0

Item: 3 Weight: 14.0 Price: 27.0 Value: 1.9285714285714286

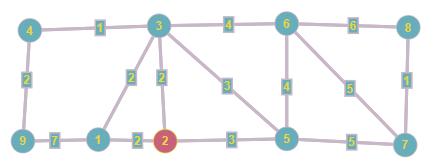
Item: 4 Weight: 10.0 Price: 9.0 Value: 0.9

Item: 5 Weight: 20.0 Price: 15.0 Value: 0.75

Item: 6 Weight: 12.0 Price: 7.199999999999999 Value: 0.6

Knapsack Weight is: 80

2. (20 pts) Let G = (V ,E) be the following weighted graph: V = {1, 2, 3, 4, 5, 6, 7, 8, 9} and E = {[(1, 2) 2], [(2, 3) 2], [(1, 3) 2], [(1, 9), 7], [(9, 4), 2], [(3, 4) 1], [(2, 5) 3], [(5, 3) 3], [(3, 6) 4], [(5, 6) 4], [(5, 7) 5], [6, 7) 5], [(6, 8) 6], [(7, 8) 1]}, where [(i, j) a] means that (i,j) is an edge of weight a. After you draw the graph: a. Select and apply the appropriate algorithm to find the minimum spanning tree of G. b. Using the greedy single source shortest path algorithm, find the distance between node 1 and all other nodes. Show the values of the DIST array at every step. To receive credit, your answer must be presented like the table in Fig. 4.18 of the textbook.



Prims algorithm steps:

Visited list: 2, 1, 3, 4, 9, 5, 6, 7, 8

Cost: 2 + 2 + 1 + 2 + 3 + 4 + 5 + 1 = 20

Start at 2. Evaluate all the connecting verticies that are not on the visited list and pick the smallest one (2, 2, 3). I choose verticies 1 and add the weight. Now compare all the weight to vertices connected to 1 and 2 that are not on the visited list (7, 2, 2, 3). I choose to connect 2 to 3. Add 3 to visited list and add weight of 2. Now do the same for vertices 2, 1, 3 giving you weights of (7, 1, 4, 3, 3). I choose 1, connect 4 and add weight of 1. I now do the same for 1, 2, 3, 4 giving me weights of (7, 2, 4, 3, 3). I choose the smallest and connect vertex 9. I now do the same for 2, 1, 3, 4, 9. Due to exclusion of vertices on the visited list, the weights to compare now are (4, 3, 3). I randomly choose one of the 3s and add 5 to the list. I now do the evaluation with 2, 1, 3, 4, 9, 5 giving me weights of (4, 4, 5). I randomly choose one of the 4s and add 6 to the list. I now do the evaluation giving me weights of (6, 5, 5). I randomly choose a 5 and add 7 to the list. I now do the evaluation of and add 8 to the list. This completes tree.

Tree is 2 -> 1, 2 -> 3, 3 -> 4, 4 -> 9, 2 -> 5, 3 -> 6, 5 -> 7, 7 -> 8

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iteration | S | Vertex Selected | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Initialize | -- | -- | 2 | 0 | 2 | +∞ | 3 | +∞ | +∞ | +∞ | +∞ |
| 1 | {2} | 3 | 2 | 0 | 2 | 3 | 3 | 6 | +∞ | +∞ | +∞ |
| 2 | {2,3} | 4 | 2 | 0 | 2 | 3 | 3 | 6 | +∞ | +∞ | 5 |
| 3 | {2,3,4} | 9 | 2 | 0 | 2 | 3 | 3 | 6 | +∞ | +∞ | 5 |
| 4 | {2,3,4,9} | 5 | 2 | 0 | 2 | 3 | 3 | 6 | 8 | +∞ | 5 |
| 5 | {2,3,4,9,5} | 6 | 2 | 0 | 2 | 3 | 3 | 6 | 8 | 12 | 5 |
| 6 | {2,3,4,9,5,6} | 7 | 2 | 0 | 2 | 3 | 3 | 6 | 8 | 9 | 5 |
| 7 | {2,3,4,9,5,6,7} | 8 | 2 | 0 | 2 | 3 | 3 | 6 | 8 | 9 | 5 |

3. Let S = {1, 2, 3,...​n​ } be a set of ​n activities that need to be scheduled among a large number of lecture halls. Activity ​i​ has to start at time s​i​ and finish at time f​i​, where s​i​ < f​i​. a. (20 pts) Develop pseudocode for a greedy algorithm and explain how it works (ex. explain data structures used). Exercise your algorithm by applying it to the following input: 5 activities where s​1​ = 1, f​1​ = 3, s​2​ = 2, f​2​ = 5, s​3​ = 4, f​3​ = 5, s​4​ = 6, f​4​ = 7, s​5​ = 4, f​5​ = 7. b. (40 pts) Develop an efficient, greedy algorithm that determines which activity should go to which lecture hall so that the number of lecture halls used is minimized. No two activities can use the same hall at the same time. Your code should accept as input a text file in the following format: 3 2,5 3,6 3,7 where the first entry is n and each entry below reflects the activity start and finish time pairs. You may assume reading valid input, meaning that both n and the number of entries is correct, all entries are positive integers and the property si < fi is maintained. c. (5 pts) Provide and explain how you derived the time complexity of your algorithm.

Create object workTimes variables int startTime and int endTime;

array of workTimes called workArray;

array of workTimes call jobs;

sort workTimes in ascending order (lowest to highest) by endTime;

iterator = 1;

workTimes object called currentObject;

for i = 0 to workArray.length do:

if (i == 0){ jobs[i] = workArray[0]; currentObject = workArray[0]; }

else{  
 if (workArray[i].getStartTime() => currentObject.getEndTime){  
 jobs[iterator] = workArray[i];

currentObject = workArray[i];

iterator++;

}

}

}

/\*the function has two arrays. One is made up of objects that have ALL the start and end times of the jobs. The second is empty and will have the jobs added to it as the function processes the data. The second array will keep a list of all the optimal amount of jobs that can be completed and their times. The first array is sorted by the jobs end times from low to high. The function processes the array. CurrentObject keeps track of the current job that will be compared to the incoming job. Each incoming job’s start time is compared to the currentObjects end time. If the start time is less than the end time, the job is skipped. If the start time is equal to or greater than the end time of the currentObject, the job is added to the jobs array and the current job becomes that job.

The algorithm puts the 5 activities into an array and sorts them from smallest end time to largest. Every job is created into an object which has a start time and an end time. The workArray array is sorted as 1(1-3), 2(2-5), 3(4-5), 4(6-7), 5(4-7), respectively. In the first pass of the for loop, workArray(0), value of 1-3, is assigned to the currentObject and also added to jobs(0). The second pass evaluates workArray(1), 2-5. The currentObject(1-3) evaluates its end time, 3, with workArray(1) start time, 2. Since 2 not equal to or greater than 3, the job is skipped. The next evaluation is workArray(2), 4-5. Since 4 is equal or greater than 3, workArray(2) now becomes the currentObject and added to the jobs(1). workArray(3), currentObject, 4-5, is now evaluated to workArray(4), 6-7. Since 6 is equal to or greater than 5, workArray(4) now becomes the currentObject and added to the jobs(2). workArray(4) is now evaluated to the currentObject. Since 4 is not equal to or greater than 7, nothing happens. The list is fully evaluated. The contents of the jobs array is workObject 1, 3, and 4, which should be the optimized list of jobs for this part of the question.

\*/

\*\*\*\*second part of question was done by code. I created a program to test validity of the algorithm and the phrase in the question “Your code should accept as input a text file in the following format: 3 2,5 3,6 3,7 where the first entry is n and each entry below reflects the activity start and finish time pairs” led me believe you wanted us to create a program. I am probably wrong. Below is the relevant code of the program. I created an algorithm that is not optimized for growth. This algorithm only works for up to 4 rooms. If I were to revise the code for growth, which I should have initially, I would have made each room an element in an array. Each comparison would have been done similarly as it was in this algorithm, however, the design would have been cleaner and all creations for new rooms would have followed the same logic without having to write specific code for the rooms.

File input: 3 2,5 3,6 3,7 5,8 2,3

Output: [Job: startTime: 2 endtime: 3 done in room 1,

Job: startTime: 2 endtime: 5 done in room 2,

Job: startTime: 3 endtime: 6 done in room 1,

Job: startTime: 3 endtime: 7 done in room 3,

Job: startTime: 5 endtime: 8 done in room 2]

Relivant code:

//used waterfall method of checking the rooms for compares.

**private** **void** implementGreedyAlgorithm() {

//sort the list ascending order with endtime

Collections.*sort*(jobsToBeDone);

**int** room = 1;

Jobs roomOneJob = **null**;

Jobs roomTwoJob = **null**;

Jobs roomThreeJob = **null**;

Jobs roomFourJob = **null**;

**for** (**int** i = 0; i < jobsToBeDone.size(); i++) {

**if** (roomOneJob == **null**) {

roomOneJob = jobsToBeDone.get(i);

String jobOneToString = ("Job: " + jobsToBeDone.get(i).toString() + " done in room " + room);

finalList.add(jobOneToString);

}

**else** **if** (jobsToBeDone.get(i).getStartTime() >= roomOneJob.getEndTime()){

roomOneJob = jobsToBeDone.get(i);

String jobOneToString = ("Job: " + jobsToBeDone.get(i).toString() + " done in room " + room);

finalList.add(jobOneToString);

}

**else** **if** (jobsToBeDone.get(i).getStartTime() < roomOneJob.getEndTime()) {

**if** (roomTwoJob == **null**) {

roomTwoJob = jobsToBeDone.get(i);

String jobTwoToString = ("Job: " + jobsToBeDone.get(i).toString() + " done in room " + (room + 1));

finalList.add(jobTwoToString);

}

**else** **if** (jobsToBeDone.get(i).getStartTime() >= roomTwoJob.getEndTime()){

roomTwoJob = jobsToBeDone.get(i);

String jobTwoToString = ("Job: " + jobsToBeDone.get(i).toString() + " done in room " + (room + 1));

finalList.add(jobTwoToString);

}

**else** **if** (jobsToBeDone.get(i).getStartTime() < roomTwoJob.getEndTime()) {

**if** (roomThreeJob == **null**) {

roomThreeJob = jobsToBeDone.get(i);

String jobThreeToString = ("Job: " + jobsToBeDone.get(i).toString() + " done in room " + (room + 2));

finalList.add(jobThreeToString);

}

**else** **if** (jobsToBeDone.get(i).getStartTime() >= roomThreeJob.getEndTime()){

roomFourJob = jobsToBeDone.get(i);

String jobFourToString = ("Job: " + jobsToBeDone.get(i).toString() + " done in room " + (room + 3));

finalList.add(jobFourToString);

}

}

}

**else** {System.***out***.println("Sorry, no rooms available");

}

}

System.***out***.println(finalList);

}

Complexity justified by code:

String[] input = DataIO.*readFromFile*(inputFile); O(2n)

**for**(**int** i = 1; i < input.length; i += 2) {

Jobs jobs = **new** Jobs(Integer.*parseInt*(input[i]), Integer.*parseInt*(input[i+1]));

jobsToBeDone.add(jobs);

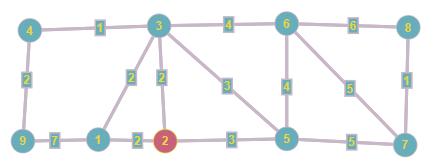
} O(n)

Relevant code: O(n)

Final Complexity: O(n)

4. (5 pts) Both Prim’s and Kruskal’s algorithms identify minimum spanning trees in graphs with positive edges. Do these algorithms work if some edges have negative weights? Justify your answer by either a proof or, counter example for each case. You may assume distinct edge weights.

The short answer is yes. To prove my answer, I will give each case with a negative number and a non negative number. If my answer is correct, both cases should result in the same answer. I will use my graph from question 2.



If my original answer is correct, the answer I am looking for is Tree is 2 -> 1, 2 -> 3, 3 -> 4, 4 -> 9, 2 -> 5, 3 -> 6, 5 -> 7, 7 -> 8.

I am going to swap out the 1 with -11 and the 2s with -2, to keep the numerical values in order. I will start at vertex 2. I will evaluate the weights (-2, -2, 3). I will choose vertex 1 and add it to the tree. I now evaluate edges not connected to vertex already known and get (7,2,3). For consistency, I will take the same edge chosen by the non negative minimal spanning tree from question 2, and take vertex 3. I now evaluate edges not connected to vertex already known and get (1,4,7,3,3). I will choose vertex 4 due to the -11 value of the edge. I now evaluate edges not connected to vertex already known and get (2,4,7,3,3). I will choose 9 due to the 2 value of the edge. I now evaluate edges not connected to vertex already known and get (4,3,3). I choose 5 with an edge value of 3. I now evaluate edges not connected to vertex already known and get (4,4,5). I choose vertex 3 to 6 that has with the same edge value as 5 to 6. I am trying to keep both chooses consistent. I now evaluate edges not connected to vertex already known and get (6,5,5). I choose 5 to 7 with a value of 5. I now evaluate edges not connected to vertex already known and get (6,-11). I lastly choose the lesser of the two values and conclude the MST.

The answer for some negative numbers is 2 -> 1, 2 -> 3, 3 -> 4, 4 -> 9, 2 -> 5, 3 -> 6, 5 -> 7, 7 -> 8. As you can see Prims algorithm is not affected if some edges have negative weight. I would demonstrate the same for the Kruskal’s algorithm, but both algorithms work on the same general evaluation process. I will explain. Both algorithms evaluate the smallest edge value at the time of evaluation. Prim’s evaluates a give subset of edges connected to a subset of vertices, and Kruskal’s evaluates the smallest edges of the total set. Neither algorithm is concerned with the total value of the edges (theoretically, both should produce the smallest total of edges in the tree). Giving any of the edges a negative value will not change the processing of the evaluation, given that the order of the numbers is still unbroken. -55 is less than -33 just as 3 is less than 5. Both will evaluate with the left being less than the right. It does not matter that the evaluation on the left is negatives and the right are positive, the evaluation is still one thing smaller than the other. Both algorithms execute a succession of evaluations of a list of values. The evaluations do not care if the numbers are positive or negative. I maybe stepping my bounds, but I would content that, if we take numerical weight out of the discussion, the edges could consist of words that could be evaluated alphabetically (‘a’ being highest).