Programming Assignment #1

TOTAL POINTS 3

1. In this programming problem and the next you'll code up the greedy algorithms from lecture for minimizing the weighted sum of completion times.. 1 point

   Download the text file below.

   jobs.txt

   This file describes a set of jobs with positive and integral weights and lengths. It has the format

   [number_of_jobs]

   [job_1_weight] [job_1_length]

   [job_2_weight] [job_2_length]

   ...

   For example, the third line of the file is "74 59", indicating that the second job has weight 74 and length 59.

   You should NOT assume that edge weights or lengths are distinct.

   Your task in this problem is to run the greedy algorithm that schedules jobs in decreasing order of the difference (weight - length). Recall from lecture that this algorithm is not always optimal. IMPORTANT: if two jobs have equal difference (weight - length), you should schedule the job with higher weight first. Beware: if you break ties in a different way, you are likely to get the wrong answer. You should report the sum of weighted completion times of the resulting schedule --- a positive integer --- in the box below.

   ADVICE: If you get the wrong answer, try out some small test cases to debug your algorithm (and post your test cases to the discussion forum).

   Enter answer here

2. For this problem, use the same data set as in the previous problem. 1 point

   Your task now is to run the greedy algorithm that schedules jobs (optimally) in decreasing order of the ratio (weight/length). In this algorithm, it does not matter how you break ties. You should report the sum of weighted completion times of the resulting schedule --- a positive integer --- in the box below.

   Enter answer here

3. In this programming problem you'll code up Prim's minimum spanning tree algorithm.

1 point

Download the text file below.

edges.txt

This file describes an undirected graph with integer edge costs. It has the format

[number_of_nodes] [number_of_edges]

[one_node_of_edge_1] [other_node_of_edge_1] [edge_1_cost]

[one_node_of_edge_2] [other_node_of_edge_2] [edge_2_cost]

...

For example, the third line of the file is "2 3 -8874", indicating that there is an edge connecting vertex #2 and vertex #3 that has cost -8874.

You should NOT assume that edge costs are positive, nor should you assume that they are distinct.

Your task is to run Prim's minimum spanning tree algorithm on this graph. You should report the overall cost of a minimum spanning tree --- an integer, which may or may not be negative --- in the box below.

IMPLEMENTATION NOTES: This graph is small enough that the straightforward O(mn) time implementation of Prim's algorithm should work fine. OPTIONAL: For those of you seeking an additional challenge, try implementing a heap-based version. The simpler approach, which should already give you a healthy speed-up, is to maintain relevant edges in a heap (with keys = edge costs). The superior approach stores the unprocessed vertices in the heap, as described in lecture. Note this requires a heap that supports deletions, and you'll probably need to maintain some kind of mapping between vertices and their positions in the heap.

Enter answer here

□ I, **Veinstin Furtado**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

Learn more about Coursera's Honor Code

Save

Submit

Programming Assignment #2

TOTAL POINTS 2

1. In this programming problem and the next you'll code up the clustering algorithm from lecture for computing a max-spacing k-clustering.

1 point

Download the text file below.

clustering1.txt

This file describes a distance function (equivalently, a complete graph with edge costs). It has the following format:

[number_of_nodes]

[edge 1 node 1] [edge 1 node 2] [edge 1 cost]

[edge 2 node 1] [edge 2 node 2] [edge 2 cost]

...

There is one edge (i, j) for each choice of $1 \leq i < j \leq n$, where n is the number of nodes.

For example, the third line of the file is "1 3 5250", indicating that the distance between nodes 1 and 3 (equivalently, the cost of the edge (1,3)) is 5250. You can assume that distances are positive, but you should NOT assume that they are distinct.

Your task in this problem is to run the clustering algorithm from lecture on this data set, where the target number k of clusters is set to 4. What is the maximum spacing of a 4-clustering?

ADVICE: If you're not getting the correct answer, try debugging your algorithm using some small test cases. And then post them to the discussion forum!

Enter answer here

2. In this question your task is again to run the clustering algorithm from lecture, but on a MUCH bigger graph. So big, in fact, that the distances (i.e., edge costs) are only defined *implicitly*, rather than being provided as an explicit list.

1 point

The data set is below.

clustering_big.txt

The format is:

[# of nodes] [# of bits for each node's label]

[first bit of node 1] ... [last bit of node 1]

[first bit of node 2] ... [last bit of node 2]

...

For example, the third line of the file "0 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1" denotes the 24 bits associated with node #2.

The distance between two nodes u and v in this problem is defined as the *Hamming distance*--- the number of differing bits --- between the two nodes' labels. For example, the Hamming distance between the 24-bit label of node #2 above and the label "0 1 0 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1" is 3 (since they differ in the 3rd, 7th, and 21st bits).

The question is: what is the largest value of k such that there is a k-clustering with spacing at least 3? That is, how many clusters are needed to ensure that no pair of nodes with all but 2 bits in common get split into different clusters?

NOTE: The graph implicitly defined by the data file is so big that you probably can't write it out explicitly, let alone sort the edges by cost. So you will have to be a little creative to complete this part of the question. For example, is there some way you can identify the smallest distances without explicitly looking at every pair of nodes?

Enter answer here

□ I, **Veinstin Furtado**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

Learn more about Coursera's Honor Code

Save Submit

Programming Assignment #3

TOTAL POINTS 3

1. In this programming problem and the next you'll code up the greedy algorithm from the lectures on Huffman coding. 1 point

Download the text file below.

huffman.txt

This file describes an instance of the problem. It has the following format:

[number_of_symbols]

[weight of symbol #1]

[weight of symbol #2]

...

For example, the third line of the file is "6852892," indicating that the weight of the second symbol of the alphabet is 6852892. (We're using weights instead of frequencies, like in the "A More Complex Example" video.)

Your task in this problem is to run the Huffman coding algorithm from lecture on this data set. What is the maximum length of a codeword in the resulting Huffman code?

ADVICE: If you're not getting the correct answer, try debugging your algorithm using some small test cases. And then post them to the discussion forum!

Enter answer here

2. Continuing the previous problem, what is the minimum length of a codeword in your Huffman code?

1 point

Enter answer here

3. In this programming problem you'll code up the dynamic programming algorithm for computing a maximum-weight independent set of a path graph.

1 point

Download the text file below.

mwis.txt

This file describes the weights of the vertices in a path graph (with the weights listed in the order in which vertices appear in the path). It has the following format:

[number_of_vertices]

[weight of first vertex]

[weight of second vertex]

...

For example, the third line of the file is "6395702," indicating that the weight of the second vertex of the graph is 6395702.

Your task in this problem is to run the dynamic programming algorithm (and the reconstruction procedure) from lecture on this data set. The question is: of the vertices 1, 2, 3, 4, 17, 117, 517, and 997, which ones belong to the maximum-weight independent set? (By "vertex 1" we mean the first vertex of the graph---there is no vertex 0.) In the box below, enter a 8-bit string, where the ith bit should be 1 if the ith of these 8 vertices is in the maximum-weight independent set, and 0 otherwise. For example, if you think that the vertices 1, 4, 17, and 517 are in the maximum-weight independent set and the other four vertices are not, then you should enter the string 10011010 in the box below.

Enter answer here

□ I, **Veinstin Furtado**, understand that submitting work that isn’t my own may result in permanent failure of this course or deactivation of my Coursera account.

Learn more about Coursera’s Honor Code

Save

Submit

Programming Assignment #4

TOTAL POINTS 2

1. In this programming problem and the next you'll code up the knapsack algorithm from lecture.

1 point

Let's start with a warm-up. Download the text file below.

knapsack1.txt

This file describes a knapsack instance, and it has the following format:

[knapsack_size][number_of_items]

[value_1] [weight_1]

[value_2] [weight_2]

...

For example, the third line of the file is "50074 659", indicating that the second item has value 50074 and size 659, respectively.

You can assume that all numbers are positive. You should assume that item weights and the knapsack capacity are integers.

In the box below, type in the value of the optimal solution.

ADVICE: If you're not getting the correct answer, try debugging your algorithm using some small test cases. And then post them to the discussion forum!

Enter answer here

2. This problem also asks you to solve a knapsack instance, but a much bigger one.

1 point

Download the text file below.

knapsack_big.txt

This file describes a knapsack instance, and it has the following format:

[knapsack_size][number_of_items]

[value_1] [weight_1]

[value_2] [weight_2]

...

For example, the third line of the file is "50074 834558", indicating that the second item has value 50074 and size 834558, respectively. As before, you should assume that item weights and the knapsack capacity are integers.

This instance is so big that the straightforward iterative implemetation uses an infeasible amount of time and space. So you will have to be creative to compute an optimal solution. One idea is to go back to a recursive implementation, solving subproblems --- and, of course, caching the results to avoid redundant work --- only on an "as needed" basis. Also, be sure to think about appropriate data structures for storing and looking up solutions to subproblems.

In the box below, type in the value of the optimal solution.

ADVICE: If you're not getting the correct answer, try debugging your algorithm using some small test cases. And then post them to the discussion forum!

Enter answer here

□ I, **Veinstin Furtado**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

Learn more about Coursera's Honor Code

Save

Submit