Programming Assignment #1

TOTAL POINTS 1

1. In this assignment you will implement one or more algorithms for the all-pairs shortest-path problem. Here are data files describing three graphs:

   1 point

   g1.txt

   g2.txt

   g3.txt

   The first line indicates the number of vertices and edges, respectively. Each subsequent line describes an edge (the first two numbers are its tail and head, respectively) and its length (the third number). NOTE: some of the edge lengths are negative. NOTE: These graphs may or may not have negative-cost cycles.

   Your task is to compute the "shortest shortest path". Precisely, you must first identify which, if any, of the three graphs have no negative cycles. For each such graph, you should compute all-pairs shortest paths and remember the smallest one (i.e., compute $\min_{u,v \in V} d(u,v)$, where $d(u,v)$ denotes the shortest-path distance from u to v).

   If each of the three graphs has a negative-cost cycle, then enter "NULL" in the box below. If exactly one graph has no negative-cost cycles, then enter the length of its shortest shortest path in the box below. If two or more of the graphs have no negative-cost cycles, then enter the smallest of the lengths of their shortest shortest paths in the box below.

   OPTIONAL: You can use whatever algorithm you like to solve this question. If you have extra time, try comparing the performance of different all-pairs shortest-path algorithms!

   OPTIONAL: Here is a bigger data set to play with.

   large.txt

   For fun, try computing the shortest shortest path of the graph in the file above.

   Enter answer here

Programming Assignment #2

TOTAL POINTS 1

1. In this assignment you will implement one or more algorithms for the traveling salesman problem, such as the dynamic programming algorithm covered in the video lectures. Here is a data file describing a TSP instance.

   tsp.txt

   The first line indicates the number of cities. Each city is a point in the plane, and each subsequent line indicates the x- and y-coordinates of a single city.

   The distance between two cities is defined as the Euclidean distance --- that is, two cities at locations (x, y) and (z, w) have distance $\sqrt{(x-z)^2+(y-w)^2}$ between them.

   In the box below, type in the minimum cost of a traveling salesman tour for this instance, *rounded down to the nearest integer.*

   OPTIONAL: If you want bigger data sets to play with, check out the TSP instances from around the world here. The smallest data set (Western Sahara) has 29 cities, and most of the data sets are much bigger than that. What's the largest of these data sets that you're able to solve --- using dynamic programming or, if you like, a completely different method?

   HINT: You might experiment with ways to reduce the data set size. For example, trying plotting the points. Can you infer any structure of the optimal solution? Can you use that structure to speed up your algorithm?

   Enter answer here

☐ I, **Veinstin Furtado**, understand that submitting another's work as my own can result in zero credit for this assignment. Repeated violations of the Coursera Honor Code may result in removal from this course or deactivation of my Coursera account.

Save Submit

Programming Assignment #3

TOTAL POINTS 1

1. In this assignment we will revisit an old friend, the traveling salesman problem (TSP). This week you will implement a heuristic for the TSP, rather than an exact algorithm, and as a result will be able to handle much larger problem sizes. Here is a data file describing a TSP instance (original source: http://www.math.uwaterloo.ca/tsp/world/bm33708.tsp).

   nn.txt

   The first line indicates the number of cities. Each city is a point in the plane, and each subsequent line indicates the x- and y-coordinates of a single city.

   The distance between two cities is defined as the Euclidean distance --- that is, two cities at locations (x, y) and (z, w) have distance $\sqrt{(x-z)^2 + (y-w)^2}$ between them.

   You should implement the *nearest neighbor* heuristic:

   1. Start the tour at the first city.
   2. Repeatedly visit the closest city that the tour hasn't visited yet. *In case of a tie, go to the closest city with the lowest index.* For example, if both the third and fifth cities have the same distance from the first city (and are closer than any other city), then the tour should begin by going from the first city to the third city.
   3. Once every city has been visited exactly once, return to the first city to complete the tour.

   In the box below, enter the cost of the traveling salesman tour computed by the nearest neighbor heuristic for this instance, *rounded down to the nearest integer.*

   [Hint: when constructing the tour, you might find it simpler to work with squared Euclidean distances (i.e., the formula above but without the square root) than Euclidean distances. But don't forget to report the length of the tour in terms of standard Euclidean distance.]

   Enter answer here

Programming Assignment #4

TOTAL POINTS 1

1. In this assignment you will implement one or more algorithms for the 2SAT problem. Here are 6 different 2SAT instances: 1 point

2sat1.txt

2sat2.txt

2sat3.txt

2sat4.txt

2sat5.txt

2sat6.txt

The file format is as follows. In each instance, the number of variables and the number of clauses is the same, and this number is specified on the first line of the file. Each subsequent line specifies a clause via its two literals, with a number denoting the variable and a "-" sign denoting logical "not". For example, the second line of the first data file is "-16808 75250", which indicates the clause $\neg x_{16808} \vee x_{75250}$.

Your task is to determine which of the 6 instances are satisfiable, and which are unsatisfiable. In the box below, enter a 6-bit string, where the ith bit should be 1 if the ith instance is satisfiable, and 0 otherwise. For example, if you think that the first 3 instances are satisfiable and the last 3 are not, then you should enter the string 111000 in the box below.

DISCUSSION: This assignment is deliberately open-ended, and you can implement whichever 2SAT algorithm you want. For example, 2SAT reduces to computing the strongly connected components of a suitable graph (with two vertices per variable and two directed edges per clause, you should think through the details). This might be an especially attractive option for those of you who coded up an SCC algorithm in Part 2 of this specialization. Alternatively, you can use Papadimitriou's randomized local search algorithm. (The algorithm from lecture is probably too slow as stated, so you might want to make one or more simple modifications to it --- even if this means breaking the analysis given in lecture --- to ensure that it runs in a reasonable amount of time.) A third approach is via backtracking. In lecture we mentioned this approach only in passing; see Chapter 9 of the Dasgupta-Papadimitriou-Vazirani book, for example, for more details.

Enter answer here