**Northeastern University**
CS7180 – Algorithms
Spring 2024 Semester, Jonathan Mwaura

---

# Problem Set #1

This Problem Set #1 is due at **11.30PM on 29$^{th}$, Jan 2024** , and will be submitted via Canvas.

This Problem Set will be marked out of 100, and is worth 8% of the final course grade.

Two problems are to be completed **individually**. and the third one is group based.

Please type (or neatly handwrite) your solutions on standard $8.5 \times 11$ paper, with your name at the top of each solution. Ensure that you submit you solutions in one file PDF file on Gradescope. **each problem sets solution should be on in its own individual page, Gradescope will help ensure you submit each solution under its correct problem number**

Make sure you label your Problem Set #1 submissions appropriately using your lastname-1 (i.e. lastname 'hypen' number of problem set number) - e.g. Mwaura-1.pdf.

While a solution must be absolutely perfect to receive full marks, I will be generous in awarding partial marks for incomplete solutions that demonstrate progress.

So that there is no ambiguity, there are two non-negotiable rules. A violation of either rule constitutes plagiarism and will result in you receiving an F for this course.

(a) If you meet with a classmate to discuss any of the Individual Problems, your submission must be an individual activity, done in your own words, away from others. The process of finding a solution might take 3 - 5 iterations or even more BUT you learn from all these attempts and your confidence grows with each iteration.

(b) These problem sets might seem hard on a first look. They are designed to be so. We learn by attempting problems, struggling through them and coming on top. I encourage you to make this learning exercise worth your while. What do I mean? Open the problem sets as early as you get them, then do not look at hints or answers any where (including on the internet and consulting other students for direct answers), give it the best shot you can. If you get stuck come to Professor or TA's office hour and we shall be glad to listen to your rationale and work with you till you are able to tackle the problem sets.

## Problem #1 – INDIVIDUAL

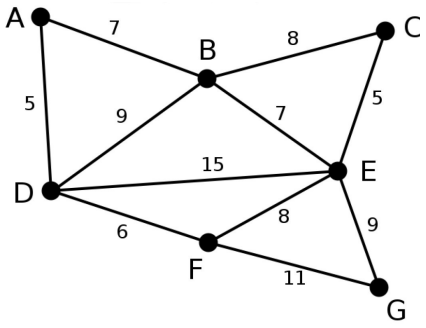
In this question you will explore the Traveling Salesperson Problem (TSP).

(a) In every instance (i.e., example) of the TSP, we are given $n$ cities, where each pair of cities is connected by a weighted edge that measures the cost of traveling between those two cities. Our goal is to find the optimal TSP tour, minimizing the total cost of a Hamiltonian cycle in $G$.

Although it is NP-complete to solve the TSP, we found a simple 2-approximation by first generating a minimum-weight spanning tree of $G$ and using this output to determine our TSP tour.

Prove that our output is guaranteed to be a 2-approximation, provided the Triangle Inequality holds. In other words, if $OPT$ is the total cost of the optimal solution, and $APP$ is the total cost of our approximate solution, clearly explain why $APP \leq 2 \times OPT$.

(b) Let $G$ be this weighted undirected graph, containing 7 vertices and 11 edges.




For each of the 10 edges that do not appear (AC, AE, AF, AG, BF, BG, CD, CF, CG, DG), assign a weight of 1000. It is easy to see that the optimal TSP tour has total cost 51.

Generate an approximate TSP tour using the algorithm from part (a), and calculate the total cost of your solution. Explain why your solution is <u>not</u> a 2-approximation.

(c) Consider a TSP instance with $n$ cities, where the edge weights do not satisfy the Triangle Inequality.

Create a simple polynomial-time algorithm to transform this instance into another TSP instance where: (i) the edge weights do satisfy the Triangle Inequality, and (ii) the two instances have the exact same set of optimal tours.

Apply your algorithm to the instance in part (b), and re-calculate the approximate TSP tour for your new instance, showing that indeed your approximate solution is a 2-approximation.

(d) Consider a TSP instance with $n$ cities, where the $n$ cities are points on the Cartesian 2-dimensional $x$-$y$ plane. Furthermore, suppose that the weight of each edge $uv$ is the Euclidean (Pythagorean) distance between points $u$ and $v$.

Consider an optimal TSP tour of this instance, which will consist of exactly $n$ edges. Prove that it is impossible for this optimal tour to cross (or intersect) itself.

## Problem #2 – INDIVIDUAL

Considering the TSP problem discussed in Question 1 and for which you carried out individual work. This questions requires you to explore heuristics that are used to solve such NP - Hard problems. Our problem of choice is TSP.

(a) An exact and inefficient algorithm to solve this problem is by dynamic programming (DP). There is a popular DP algorithm called Held–Karp algorithm. This according to Wikipedia is also called the Bellman–Held–Karp algorithm and it was proposed in 1962 "independently by Bellman and by Held and Karp to solve the traveling salesman problem (TSP)". We discussed this technique in the classroom. Your work in this part is to **design and implement the Held - Karp Algorithm** and use the algorithm to solve any arbitrary Metric TSP problem (We discussed Metric TSP as those TSP that are symmetrical and also satisfy triangular inequality.

(b) In class we explored various TSP specific heuristics algorithms (note that some of these heuristics algorithms could also be approximation algorithms). In this part of the question, you are expected to **design and implement** one of the following heuristics: (Note the choice is yours but you should check in with the other class members to ensure that you implement a different algorithm from them)

(i) Nearest Neighbour Algorithm
(ii) Nearest Addition /Insertion Algorithm
(iii) Random Addition/Insertion Algorithm
(iv) Furthest Addition/Insertion Algorithm
(v) Double Tree (or Twice around the Tree) Algorithm
(vi) Christophides Algorithm
(vii) 2- OPT Algorithm
(viii) 3-OPT Algorithm
(ix) Clarke-Wright Saving heuristic
(x) Lin-Kernighan-Helsgaun.

(c) Empirical Analysis: You will now need to run both your DP from (a) above and your Heuristic from (b) above. You need to run these for 20 runs for the selected TSP problem (i.e. run the algorithm 20 times repeatedly using the same TSP problem and record the running time each time the technique is run). You will have twenty lines of data points all associated with that one problem you will have used. You will need to compute the average time/speed for each run (**For uniformity, use the graph given in Question 1b.)**.

Now repeat this experimentation but now we are going to be a bit methodical. We would like to see the run time based on how many cities are in the TSP. You will need to go to this site: https://github.com/acu192/fun-tsp-challenge. There are 4 different TSP datasets, for our question we are interested in the medium data set ( https://github.com/acu192/fun-tsp-challenge). This contains a TSP with 100 cities. However, we want to work with it progressively. So we shall start with 10 cities and increment this with 10 cities until we reach 100 cities.

For each run we need to record the time spent (note in order to remove any randomness, we would like to repeat each experiment for 20 runs and record the average). Once we have the data (i.e. 10 rows - each problem per row and the average run time) we shall need to plot on Average runtime vs no of cities (see Problem below).

You are expected to output a graphical display of the running times as shown in the figure below. Additionally, you will need to report and discuss your intuition into these running times versus the actual fitness that you obtained, i.e. a direct comparison between your chosen technique and the DP technique. For instance:

(i) does your algorithm perform better with smaller city sizes? what about the larger ones?

(ii) DP will give an exact result, if you were to take the average solution achieved with your algorithm (averaged from the 20 runs per each problem), what is the do you obtain Relative approximation or Absolute approximation (see notes from class - slides on Approximation from Unit 1).