CS170-SP2015 Final Project Write-up

Team Name: kadut (pronounced: *kah-doot)*

Login: cs170-xd

Team members:

- Cecilia Astrid Maharani – 25483551
- Christian Dennis – 25483266
- Theodorus Budiyanto – 25477407
- Dionysius Andi Hanubrata – 25482771

Brief Description of General Algorithm/ Approach:

The algorithm our team used is a combination of algorithms, consist of brute-force and modified Kruskal algorithm to make a modified MST then fix this MST to meet the TSP color requirements. Our approach is if the input graph is inside a considerable size, which is less than 15 nodes in total, we use brute-force. Otherwise if the data is too complex to brute-force, we directly use the modified MST algorithm to approximate the path.

Our Brute-force algorithm:
Firstly, we apply a brute-force with backtracking method to search for every possible valid path of the input graph. After that, we take the minimum cost out of all possible paths - this guarantees an optimum cost of a valid path.

Our Modified MST algorithm:
Firstly, we create a valid-color-MST using modified Kruskal algorithm, then process this modified MST to meet the TSP color requirement. We modify it such that when the algorithm is searching for the minimum edge, it also considers the color restrictions such that no three vertices with the same color can be adjacent to each other. But this modified MST might have an/some nodes with degree more than two (3 or more). As shown in the book, if we just follow the path from MST, we will go through that node/s more than once. To fix this, for all vertices with degree more than 2, we move one of its edges to another node with the following methods:

> Cheapest Successor Connection method: Let x be the node with degree more than two. Then, we try to connect x's neighbors with all x's successors except the successors of the node that we are currently working on and choose the cheapest connection among all possible connections. However, this might create an infinite loop so we pass the partially fixed graph to the **Cheapest Leaf Connection** after some reasonably large number of iterations.

Cheapest Leaf Connection method: This method is similar to the previous one, but instead of finding the minimum cost possible out of all nodes, it searches for a **leaf** that has the minimum cost possible. If the algorithm still finds difficulty to find such minimum cost, it passes the partially fixed graph to our next algorithm, the **Direct Leaf connection.**

Direct Leaf Connection method: Since the previous method failed, we know that the input graph's configuration is a complex one, we then cut our minimum cost priority and connect the vertex to a possible leaf without taking into account the cost. If by any chance this algorithm fails, we rebuild the MST of our original input graph and directly apply this DLC method.

Brief Description of Generated Input Files:

Out of the three inputs, two of our generated inputs are randomly generated inputs. The last one is named as MST-trap input, the name tells the story; the input graph has edges with the weight of 1 coming out from node 1 to all other nodes, while all other edges have weights larger than 1. The reason behind this is when MST-approach algorithm solve this input, it will create an MST where all edges of the MST connect node 1 to other nodes. It makes it hard for finding paths after creating the MST.

Brief Description of codes organization and running instructions:

The first step of our code is to parse the input file and create a 2-D list of distance between nodes. We then convert these lists to generate a graph using tools from networkx library. The code then process the graphs using the algorithm mentioned above. It outputs the shortest path into a text file named answer.out for every iteration of the input graphs.

Running instructions: We have created a user interface that after the python runs the code, it asks for how many test cases are you going to solve (how many input graphs) and the user-interface gives a real-time information of what algorithm failed/being used, then automatically writes the output to the specified answer.out file.

Libraries/ Resources:

Networkx - https://networkx.github.io
Kruskal with Path Compression -
http://www.cs.cmu.edu/~ckingsf/class/02713-s13/src/mst.py