University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithms and Datastructures
Winter Term 2024
Exercise Sheet 9

Due: Wednesday, June 26th, 2pm

Exercise 1: Minimum Spanning Trees *(10 Points)*

Let $G = (V, E, w)$ be an *undirected, connected, weighted* graph with pairwise distinct edge weights.

(a) Show that G has a *unique* minimum spanning tree. *(5 Points)*

(b) Show that the minimum spanning tree T' of G is obtained by the following construction:

Start with $T' = \emptyset$. For each cut in G, add the lightest cut edge to T'.

(5 Points)

Exercise 2: Travelling Salesperson Problem *(10 + 5* Points)*

Let $p_1, \dots, p_n \in \mathbb{R}^2$ be points in the euclidean plane. Point p_i represents the position of city i. The distance between cities i and j is defined as the euclidean distance between the points p_i and p_j. A *tour* is a sequence of cities $(i_1, \dots, i_n)$ such that each city is visited exactly once (formally, it is a permutation of $\{1, \dots, n\}$). The task is to find a tour that minimizes the travelled distance. This problem is probably costly to solve.[1] We therefore aim for a tour that is at most twice as long as a minimal tour.

We can model this as a graph problem, using the graph $G = (V, E, w)$ with $V = \{p_1, \dots, p_n\}$ and $w(p_i, p_j) := \|p_i - p_j\|_2$. Hence, G is undirected and complete and fulfills the triangle inequality, i.e., for any nodes x, y, z we have $w(\{x, z\}) \leq w(\{x, y\}) + w(\{y, z\})$. We aim for a tour $(i_1, \dots, i_n)$ such that $w(p_{i_n}, p_{i_1}) + \sum_{j=1}^{n-1} w(p_{i_j}, p_{i_{j+1}})$ is small.

(a) Let G be a weighted, undirected, complete graph that fulfills the triangle inequality. Show that the sequence of nodes obtained by a pre-order traversal of a minimum spanning tree (starting at an arbitrary root) is a tour that is at most twice as long as a minimal tour. *(5 Bonus Points)*

(b) Implement an algorithm that computes the pre-order ordering of a minimum spanning tree of G. You may use the templates `TSP.py` and `AdjacencyMatrix.py` as well as python modules for heap and union-find data structures[2]. Transfer the graph given in `cities.txt` into an adjacency matrix and run your algorithm on it. Compute the sum of distances of your tour and attach this value to your solution. *(10 Points)*

[1] The Travelling Salesperson Problem is in the class of $\mathcal{NP}$-complete problems for which it is assumed that no algorithm with polynomial runtime exists. However, this has not been proven yet.

[2] E.g., `heapq` and `networkx.utils.union_find`. In `heapq` the function `heappush` corresponds to the `insert` operation and `heappop` to the `delete-min` operation from the lecture. You can also use `heappush` and `heappop` on Python-lists (more details here). If you instantiated an object `uf` of the class `UnionFind`, the command `uf[`i`]` creates a new set $\{i\}$ if i does not exist in `uf` yet and else returns the representative of the set containing i (this combines the functions `make-set` and `find` from the lecture. More details here).