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

Algorithms and Datastructures
Winter Term 2023
Sample Solution Exercise Sheet 9

Due: Wednesday, January 10th, 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)

Sample Solution

(a) Let T and T' be two minimum spanning trees with edges $e_1, \ldots, e_{n-1}$ and $e'_1, \ldots, e'_{n-1}$, sorted by increasing weight. Assume we have $T \neq T'$. Let j be the largest index for which $e_j \neq e'_j$. As the weights are pairwise distinct, we also have $w(e_j) \neq w(e'_j)$. W.l.o.g. let $w(e_j) < w(e'_j)$. The graph $T' \setminus \{e'_j\}$ has two connected components with nodes S and $V \setminus S$. Let e_k be an edge in T connecting S and $V \setminus S$. As T' contains only one edge between S and $V \setminus S$, it must hold $k \leq j$ (as $e_k = e'_k$ for $k > j$). As $(T' \setminus \{e'_j\}) \cup \{e_k\}$ is a spanning tree and $w(e'_j) > w(e_j) \geq w(e_k)$, it has a smaller weight than T', contradicting the fact that T' is minimal.

(b) Let T be the MST of G and T' the set containing the lightest cut edges.

$T' \subseteq T$: Let $s \in T'$, i.e., s is the lightest cut edge of a cut $(S, V \setminus S)$ in G. Assume $s \notin T$. In T there is a (unique) path from x to y. Let e be an edge on that path which is a cut edge of $(S, V \setminus S)$. According to the assumption we have $e \neq s$ and s is the (unique) lightest cut edge of $(S, V \setminus S)$, so we have $w(s) < w(e)$. It follows that the spanning tree $(T \setminus \{e\}) \cup \{s\}$ is lighter than T contradicting that T is an MST.

$T \subseteq T'$: Let $e \in T$. The graph $T \setminus \{e\}$ has two connected components which define a cut in G. With an exchange argument as above one can show that e is the (unique) lightest cut edge of this cut, i.e., we have $e \in T'$.

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

Let $p_1, \ldots, p_n \in \mathbb{R}^2$ 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 $(i_1, \ldots, i_n)$ such that each city is visited exactly once (formally, it is a permutation of $\{1, \ldots, n\}$). 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 $V = \{p_1, \ldots, p_n\}$ 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 $(i_1, \ldots, i_n)$ 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)*

Sample Solution

(a) Let $R = (i_1, \ldots, i_n)$ be a minimal tour and $w(R) := w(p_{i_n}, p_{i_1}) + \sum_{j=1}^{n-1} w(p_{i_j}, p_{i_{j+1}})$. Let T be an MST, $w(T) := \sum_{e \in T} w(e)$ its weight and $\mathcal{P}_T$ its pre-order sequence of nodes. As the graph is complete, $\mathcal{P}_T$ is also a tour.

We add points to $\mathcal{P}_T$ as follows: If two subsequent nodes u and v are not connected in T by a tree edge, we add between u and v all nodes on the shortest path from u to v in T (these are all nodes from u to the first common ancestor w and from there to v). We write $\mathcal{P}'_T$ for the sequence that we obtain (this is formally not a tour as points are visited more than once).

In $\mathcal{P}'_T$, two subsequent nodes are neighbors in T, so we can consider this sequence as a sequence of edges in T. Each edge from T is contained in $\mathcal{P}'_T$ exactly twice (if you go from the last point back to the root). Thus we have $w(\mathcal{P'}_T) = 2\sum_{e \in T} w(e)$. The triangle inequality implies $w(\mathcal{P}_T) \leq w(\mathcal{P'}_T)$ and hence $w(\mathcal{P}_T) \leq 2\sum_{e \in T} w(e)$.

The minimal tour R defines a spanning tree T_R by taking the edges between subsequent nodes in R. As T is the minimum spanning tree we have $w(T) \leq w(T_R) \leq w(T_R) + w(p_{i_n}, p_{i_1}) = w(R)$ and hence $w(\mathcal{P}_T) \leq 2 \cdot w(R)$.

Remark: The above argumentation also works for the post-order traversal. However, if you want the tour to start at a predefined point, it is easiest to use this point as the root of a pre-order traversal.

(b) Cf. `TSP.py` for our implementation. The calculated round trip of our solution has a length of 363.11 (however, the pre-order of the tree order and therefore the length of the resulting tour length is not always the same). Since the unique MST has a weight of 248.03, your own estimate should be at least 248.03 and at most 496.06 (see exercise part (a)). Cf. figure 1 for an illustration.

[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).

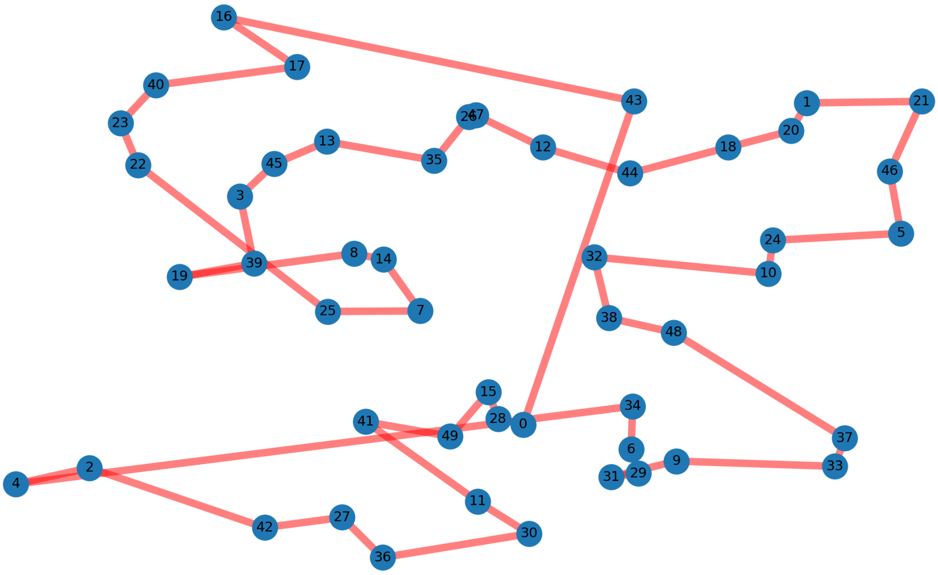

Figure 1: The approximated tour.