Assignment 10 – Selected Model Answers

Exercise 1. Suppose the edges of a graph are presented to you only once (for example over a network connection) and you do not have enough memory to store all of them. The edges do not necessarily arrive in sorted order.

Outline an algorithm that nevertheless computes a minimum spanning tree using space in $O(|V|)$.

Solution. Maintain a possibly partial set of edges E' that is intially empty, and an array of length V for the representation of connected components.

Let $e \in E$ be a new incoming edge. If $E' \cup \{e\}$ contains a cycle—this is the case, if both endpoints v and w of e are in the same connected component—then choose the edge $e' \in E'$ in this cycle with maximal cost $c(e')$ and replace e' by e. In this way we guarantee that E' never contains more than $|V| - 1$ edges, so the space complexity is in $O(|V|)$.

When we have only one connected component, we have reached a spanning tree, so following incoming edges can at most reduce the costs. This is the same as what would have resulted, if e had been explored before e', which is what Kruskal's algorithm does. Hence the final result will be a minimum spanning tree.

Exercise 2.

(i) In some instances of the greedy TSP algorithm it is possible to find a shorter optimal tour if the salesman is allowed to visit a city more than once. Construct an example for this.

(ii) A distance matrix is *Euclidean* iff it satisfies the triangle inequality: $d(i, j) \leq d(i, k) + d(k, j)$ for all i, j, k. Show that in this case it is never advantageous to pass through the same city several times.

(iii) Specify a greedy algorithm for the TSP with a Euclidean distance matrix, which produces tours with at most twice the length of an optimal tour.

Hint: Start by constructing a mimimal spanning tree, then use (ii).

Solution.

(i) Take an edge-weighted graph with $V = \{1, 2, 3, 4, 5\}$ defined by the following incidence matrix:

$$\begin{pmatrix} 0 & 3 & 29 & 21 & 1 \\ 3 & 0 & 117 & 101 & 2 \\ 29 & 117 & 0 & 167 & 25 \\ 21 & 101 & 25 & 0 & 200 \\ 1 & 2 & 167 & 200 & 0 \end{pmatrix}$$

Then the edges $\{1, 2\}, \{2, 5\}, \{1, 5\}, \{1, 3\}, \{3, 4\}, \{1, 4\}$ define a complete tour with total costs 81, in which vertex 1 is visited twice. The greedy TSP algorithm will construct this tour, if multiple visit are permitted.

If vertex 1 is not visited twice, at least one of the remaining edges must be used, so the costrs will exceed 100.

(ii) Suppose we have a complete tour, in which one vertex v is visited multiple times. Then there are edges e_1, e_2, e_3, e_4 that are in the tour and incident to v. Let $e_3 = \{v, w_1\}$ and $e_4 = \{v, w_2\}$. If we replace e_3 and e_4 by the edge $e_5 = \{w_1, w_2\}$, the costs do not increase, as $d(w_1, w_2) \leq d(v, w_1) + d(, w_2)$, and the double visit is omitted from the tour.

(iii) First, create a MST (V, E') using the greedy algorithm by Kruskal or Prim. Then E' defines a tour through G following each edge in E' exactly twice:procedd in depth-first way and baxcktrack, when there is no more incident edge in E' that has not yet been explored.

As every edge is used exactly twice, the cost of this tour is $2 \sum_{e \in E'} c(e)$. Then for every backtracking path apply the replacement from (ii), which may only reduce the costs.

If we have an optimal tour with edges in E'', then the omission of a single edge e from E'' defines a spanning tree, and we must have $\sum_{e \in E'} c(e) \leq \sum_{e \in E''} c(e)$. Hence the costs for the constructed tour are at most $2 \sum_{e \in E''} c(e)$ as claimed.

Exercise 3.

(i) The Ford-Fulkerson algorithm when executed in a depth-first fashion has some redundancy. First, all outgoing edges are pushed onto the stack and then the last is popped off to be followed by the algorithm. Modify the Ford-Fulkerson algorithm such that the first edge coming out of a certain vertex is immediately followed, and the second is followed only if the first does not lead to the sink. Consider using recursion.

(ii) The Ford-Fulkerson algorithm assumes that it terminates. Do you think such an assumption is safe?

Solution.

(i) If the first outgoing edge is immediately followed, the algorithm avoids the computation steps $\mathit{Label}(w) := (v, \min(\pi_2(\mathit{Label}(v)), \mathit{ResCap}(v, w)))$ and $\mathit{Path}(w) := \mathit{Path}(v) + [w]$ for the other edges. However, this computation requires constant time, so the performance improvement is marginal.

(ii) In each step the Ford-Fulkerson algorithm finds an augmenting path p in the residual network G_f, and the flow f is replaced by the augmented flow $f \uparrow f_p$. For the flow value we get $|f \uparrow f_p| = |f| + |f_p| > |f|$, so the sequence of flow values is strictly monotone increasing. We also have $f_p > m$, where m is the minimum capacity value in the network. So the algorithm must either terminate with a maximjum flow after finitely many steps or create an unbounded sequence of flow values. As all flow values are bounded by the capacity of a minimum cut, the algorithm must terminate.

Exercise 4.

(i) Implement Kruskal's algorithm for the construction of a minimum spanning tree.

(ii) Implement a *partition data structure* (aka *union-find data structure*) to support your implementation in (i). For this use a list L, in which entries correspond to vertices. If $L[i] = j$, then vertex i is in the connected component of j. In this way each connected component is represented as a tree, so can easily check if the endpoints of the current edge under consideration are already in the same connected component. Show how to realise the merging of connected components.

SOLUTION. See the C++ header and program files in the archives `Ass10_Ex4isolution.zip` and `Ass10_Ex4iisolution.zip`.