Assignment 9 – Selected Model Answers

Exercise 1.

(i) Describe how to implement an algorithm to determine a maximum spanning tree, i.e. the sum of the edge costs shall be maximal.

(ii) What happens in the case of Prim's and Kruskal's algorithms, if negative edge costs are permitted? Is it still sensible to talk about minimum spanning trees, if negative edge costs are permitted?

Solution.

(i) Let the edge-labelled graph be $G = (V, E, c)$, and let $C = \max\{c(v, w) \mid (v, w) \in E\} + 1$. Then apply an MST algorithm, e.g. the algorithm by Kruskal or Prim, to the graph $G' = (V, E, c')$ with $c'(v, w) = C - c(v, w) > 0$. Let the result be (V, E'), which is a minimum spanning tree for G'.

It is also a maximum spanning tree for G. To see this, let (V, F) be any other spanning tree. Then we have

$$\sum_{(v,w)\in E'} c'(v, w) \leq \sum_{(v,w)\in F} c'(v, w) \ ,$$

which implies

$$\begin{aligned}
\sum_{(v,w)\in F} c(v, w) &= \sum_{(v,w)\in F} (C - c'(v, w)) \\
&= (|V| - 1)C - \sum_{(v,w)\in F} c'(v, w) \\
&\leq (|V| - 1)C - \sum_{(v,w)\in E'} c'(v, w) \\
&= \sum_{(v,w)\in E'} (C - c'(v, w)) \\
&= \sum_{(v,w)\in E'} c(v, w)
\end{aligned}$$

(ii) As we request to obtain a spanning tree, it still makes sense to look for a minimal spanning tree, which will have a cost $\geq (|V| - 1) \cdot \min\{c(v, w) \mid (v, w) \in E\}$.

If we take $C < \min\{c(v, w) \mid (v, w) \in E\}$ and replace $c(v, w)$ by $c'(v, w) = c(v, w) - C > 0$, then we can applu the algorithm of Kruskal or Prim to obtain a minimum spanning tree (V, E') with respect to c'. As costs for all edges differe uniformly by C, this is also a minimal spanning tree with respect to c.

Furthermore, as the order of the edges remains the same for the cost functions c and c', we can get this MST by applying the algorithm directly for the original c.

Exercise 2.

(i) Kruskal's algorithm also works for graphs that are not connected, in which case the result is a *spanning forest*. Modify Prim's algorithm to work as well for non-connected graphs without referring to multiple calls of the algorithm.

(ii) Show that if all edge costs are pairwise different, the minimum spanning tree is uniquely defined.

(iii) A set T of edges spans a connected graph G, if (V, T) is connected. Is a minimum cost spanning set of edges necessarily a tree? Is it a tree if all edge costs are positive?

Solution.

(i) Let $G = (V, E)$ have m connected components $(V_1, E_1), \dots, (V_m, E_m)$. Choose arbitrary vertices $v_i \in V_i$ for $1 \leq i \leq m$ and add edges $E' = \{\{v_i, v_{i+1}\}, \mid 1 \leq i \leq m-1\}$. Also extend the cost function c by $c(v_i, v_{i+1}) = d > 0$ for some constant d.

Prim's algorithm will at some point add all the new edges. The resulting MST has additional costs $(m-1) \cdot d$. Omitting the additional edges gives a MST for the original graph G.

(ii) Without loss of generality we can assume that $G = (V, E)$ is connected. Let (V, E_1) be a minimal spanning tree, and let $e \in E_1$ be the edge with largest costs. Then removing e gives rise to a partition of V into disjoint vertex sets V_1 and V_2, with $e = (v_1, v_2)$ for some $v_i \in V_i$.

If there exists a different minimal spanning tree (V, E_2) not containing the edge e, then there must be an edge $e' \in E_2$ with $e' = (w_1, w_2)$ and $w_i \in V_i$. If we replace e in E_1 by e', we obtain another spanning tree with lower costs contradicting the minimality.

(iii) Take $G = (V, E)$ with $V = \{a, b, c\}$ and $E = \{\{a, b\}, \{b, c\}, \{a, c\}\}$ and costs $c(a, b) = -1$, $c(b, c) = -2$ and $c(a, c) = -3$. Clearly, $T = E$ defines a minimum cost spanning set, but it is not an MST.

If all edge costs are positive and there is a cycle in a spanning set, then removing any edge of the cycle preserves the connectivity property, but reduces edge costs. Hence, if all edge costs are positive, a minimum cost spanning set can only be a tree.

Exercise 3. Implement a class DiGraph of directed graphs:

(i) Define basic operations for the insertion and deletion of edges and the determination of outgoing/incoming edges.

(ii) Implement depth-first and breadth-first search on directed graphs.

(iii) Implement a program that checks, if a given directed graph is acyclic.

Solution. See the C++ header and program files in the archive `Ass9_Ex3solution.zip`.

EXERCISE 4. For the bipartite matching problem we are given a finite bipartite graph (V, E), where the set V of vertices is partitioned into two sets *Boys* and *Girls* of equal size. Thus, the set E of edges contains sets $\{x, y\}$ with $x \in \mathit{Boys}$ and $y \in \mathit{Girls}$. A *perfect matching* is a subset $F \subseteq E$ such that every vertex is incident to exactly one edge in F. A *partial matching* is a subset $F \subseteq E$ such that every vertex is incident to at most one edge in F. So the algorithm will create larger and larger partial matchings until no more unmatched boys and girls are left, otherwise no perfect matching exists.

We use functions girls_to_boys and boys_to_girls turning sets of unordered edges into sets of ordered pairs:

$$\begin{aligned} \text{girls\_to\_boys}(X) &= \{(g, b) \mid b \in \mathit{Boys} \wedge g \in \mathit{Girls} : \{b, g\} \in X\} \\ \text{boys\_to\_girls}(X) &= \{(b, g) \mid b \in \mathit{Boys} \wedge g \in \mathit{Girls} : \{b, g\} \in X\} \end{aligned}$$

Conversely, the function unordered turns a set of ordered pairs (b, g) or (g, b) into a set of two-element sets:

$$\text{unordered}(X) = \{\{x, y\} \mid (x, y) \in X\}$$

We further use a predicate reachable and a function path. For the former one we have reachable(b, X, g) iff there is a path from b to g using the directed edges in X. For the latter one path(b, X, g) is a set of ordered pairs representing a path from b to g using the directed edges in X.

Then an algorithm for bipartite matching can be realised by iterating the following rule:

```
par if          $mode$ = init
    then   par  $mode$ := examine
                $partial\_match := \emptyset$
           endpar
    endif
    if          $mode$ = examine
    then   if   $\exists b \in Boys. \forall g \in Girls. \{b, g\} \notin partial\_match$
           then $mode$ := build-digraph
           else par  $Output$ := true
                     $Halt$ := true
                     $mode$ := final
                endpar
           endif
    endif
    if          $mode$ = build-digraph
    then   par  $di\_graph$ := girls_to_boys($partial\_match$)
                           $\cup$ boys_to_girls($E - partial\_match$)
                $mode$ := build-path
           endpar
    endif
    if          $mode$ = build-path
    then choose $b \in \{x \mid x \in Boys : \forall g \in Girls. \{b, g\} \notin partial\_match\}$
           do   if $\exists g' \in Girls. \forall b' \in Boys. \{b', g'\} \notin partial\_match$
                           $\wedge$ reachable($b$, $di\_graph$, $g'$)
                then choose $g \in \{y \mid y \in Girls. \forall x \in Boys. \{x, y\}$
                    $\notin partial\_match \wedge$ reachable$(b, di\_graph, y)\}$
                    do  par $path :=$ path$(b, di\_graph, g)$
                            $mode :=$ modify
                        endpar
                    enddo
                else    par $Output :=$ false
                            $Halt :=$ true
                            $mode :=$ final
                        endpar
                endif
            enddo
        endif
        if      $mode =$ modify
        then    par $partial\_match = (partial\_match -$ unordered$(path))$
                                      $\cup($unordered$(path) - partial\_match)$
                    $mode :=$ examine
                endpar
        endif
    endpar
```

(i) Implement a class BiPartiteGraph of bipartite graphs covering basic operations for insertion and deletion of vertices and edges and for the determination of edges incident to a given vertex.

(ii) Using the class BiPartiteGraph from (i) implement the above algorithm for the determination of a perfect matching on a bipartite graph (provided such a matching exists).

Solution. See the C++ header and program files in the archive `Ass9_Ex4solution.zip`.