5. Übung

1. In der 4. Übung wurde das Rucksack–Problem betrachtet, zur Erinnerung:
   Gegeben sind n–Elemente und zugehörig zwei Arrays $p = (p_1, \ldots, p_n)$ und $w = (w_1, \ldots, w_n)$ (üblicherweise) positiver natürlicher Zahlen, sowie ein $C \in \mathbb{R}$. Typischerweise interpretiert man p_i als den Preis und w_i das Gewicht eines i–ten Gegenstandes.
   Angenommen ein Dieb kann nun maximal C Kilogramm tragen, so ergibt sich die Frage: Welche der n Güter soll er mitnehmen, um maximale Beute zu machen?
   Etwas präzieser: Gesucht ist $f(n, C)$, wenn

$$f(m, C') = \max \left\{ \sum_{i=1}^{m} x_i p_i \;\Big|\; \sum_{i=1}^{m} x_i w_i \leq C' \text{ und } x_i \in \{0, 1\} \right\}$$

   für $m \in \{1, \ldots n$ und $C' \in \{0, \ldots, C\}$.

   (a) Angenommen, es gäbe 6 Gegenstände, deren Preis–Array wäre $(100, 50, 45, 20, 10, 5)$ und deren Gewicht $(40, 35, 18, 4, 10, 2)$. Weiters sei $C = 100$.
   Überlegen Sie an diesem Beispiel, welchen Gewinn

   i. ein greedy Vorgehen nach dem Wert, d.h. in jedem Schritt wird der wertvollste Gegenstand (solange C nicht überschritten wird) genommen,
   ii. ein greedy Vorgehen nach dem Gewicht, d.h., in jedem Schritt wird der jeweils verfügbare leichteste Gegenstand (solange C nicht überschritten wird) genommen,
   iii. ein greedy Vorgehen nach dem Wert pro kg (relativen Wert), d.h., in jedem Schritt wird jener Gegenstand mit höchstem Wert pro kg (solange C nicht überschritten wird) genommen,
   iv. die optimale Lösung

   liefert. Wo liegt das Problem des greedy Algorithmus hier?

   (b) Betrachtet wird nun das sogenannte reelle Rucksackproblem, d.h. gesucht ist $f(n, C)$, wenn

$$f(m, C') = \max \left\{ \sum_{i=1}^{m} x_i p_i \;\Big|\; \sum_{i=1}^{m} x_i w_i \leq C' \text{ und } x_i \in [0, 1] \right\}$$

   für $m \in \{1, \ldots n$ und $C' \in \{0, \ldots, C\}$. (Beachten Sie den Unterschied: x_i ist nun reell, d.h. es können auch Bruchstücke der jeweiligen Gegenstände genommen werden.
   Begründen Sie, dass beim reellen Rucksackproblem ein greedy Algorithmus einen maximalen Gewinn bringt und geben Sie einen entsprechenden Algorithmus an, welcher das reelle Rucksackproblem löst.

2. Der Winter kommt: Für eine Gruppe von n Skiläufern mit Körpergröße $g_1, \ldots, g_n \in \mathbb{N}$ gibt es n Paar Ski der Längen $l_1, \ldots, l_n \in \mathbb{N}$.
   Dem i–ten Skiläufer wird der Ski der Länge l_{a_i} zugeordnet. Man möchte nun die durchschnittliche Differenz zwischen Körpergröße und Skilänge minimieren, d.h., der Ausdruck $\frac{1}{n} \sum_{i=1}^{n} |g_i - l_{a_i}|$ soll minimal werden.
   Gegeben sind die folgenden beiden Greedy–Algorithmen zur Lösung des Problems, einer stimmt (welcher und warum?), einer nicht (welcher und warum?):

   (G1) Paarweise wird nach und nach (greedy–typisch) Ski–Skiläufer so zugeordnet, dass in jedem Schritt die Differenz Skilänge–Körpergröße minimal ist.

   (G2) Kleinster Skifahrer bekommt die kürzesten Ski, zweitkleinster Skifahrer die zweitkürzesten Ski, usw.

   Hinweis: Für den falschen Algorithmus finden Sie ein zweielementiges Gegenbeispiel, die Korrektheit des anderen Algorithmus beweisen sie mittels Kontraposition, d.h. nehmen Sie an, der Algorithmus würde (für einen Input) keine optimale Lösung liefern und führen Sie das auf einen Widerspruch.

3. Als Präfix eines binären Wortes $a_1 a_2 \ldots a_n \in \{0, 1\}^n$ versteht man einen Anfangsabschnitt des Wortes. In einer binären präfix–freien Codierung von Elementen einer endlichen Menge M ist jedem Element $c \in M$ ein binäres Wort $w(c)$ so zugeordnet, dass $w(c)$ Präfix keines anderen zugeordneten Wortes $w(c')$, $c \neq c' \in M$, ist.

(Solche präfix freien Codes sind vorteilhaft, da eine Entcodierung offenbar leicht möglich ist). Angenommen, den Elementen von M wären zustzlich Gewichte $f(c)$ zugeordnet.
Ziel ist es nun, eine päfix–Codierung von M zu finden, welche möglichst kleines mittleres Gewicht hat.
Der folgende Algorithmus induziert (bottom up) eine Binärbaumdarstellung B (ein nicht voller binärer Baum), wobei jedes Element von M als ein Blatt von B auftritt. Gibt man den linken Kanten von B den Wert 0 und den rechten den Wert 1, so liefert der Pfad (Wurzel von B,Blatt c) die Codierung von c, $w(c)$.
Dieses greedy Verfahren liefert – wie sich zeigen lässt – eine optimale Lösung obiger Problemstellung.
Der Algorithmus beginnt mit den $|M|$ Blättern und führt Verknüpfungsoperationen durch, welche neue Knoten liefern, welche sich nach und nach zu einem binären Baum ergänzen.

$Präfix(M)$:
1 $n \leftarrow |M|$
2 $Q \leftarrow M$
3 für $i \leftarrow 1$ to $n-1$
4 do neuen Knoten z einfügen
5 left$[z] \leftarrow x \leftarrow$ EXTRACT-MIN(Q)
6 right$[z] \leftarrow y \leftarrow$ EXTRACT-MIN(Q)
7 $f(z) \leftarrow f(x) + f(y)$
8 INSERT(Q, z)
9 return EXTRACT-MIN(Q)

Erklären Sie die Funktionsweise des Algorithmus anhand der Beispiele

(a) $M = \{a, b, c, d, e\}$ und $f(a) = 15$, $f(b) = 7$, $f(c) = 6$, $f(d) = 6$, $f(e) = 4$. Wie lauten in diesem Beispiel die zugeordneten binären Wörter?

(b) $M = \{a_1, \ldots, a_n\}$ und $f(a_{n-i}) = F_i$, die i–te Fibonacci Zahl, $0 \leq i < n$.

4. Gegeben sei der vollständige Graph mit den Knoten $\{A, B, C, D, E, F\}$ und folgenden Kantengewichten:

	A	B	C	D	E	F
A	0	6	9	11	5	9
B	6	0	3	6	5	2
C	9	3	0	0	4	4
D	11	6	0	0	6	6
E	5	5	4	5	0	8
F	9	2	4	6	8	0

(a) Bestimmen Sie einen minimal spanning tree mit Kruksal's Algorithmus.

(b) Bestimmen Sie einen minimal spanning tree mit Prim's Algorithmus (beginnend in F).

Hier die Pseudocodes der beiden Algorithmen (danke Wikipedia):

(a) Algorithmus von Prim:
G: Graph
VG: Knotenmenge von G
w: Gewichtsfunktion fr Kantenlänge
r: Startknoten ($r \in VG$)
Q: Prioritätswarteschlange
$p[u]$: Elternknoten von Knoten u im Spannbaum
$Adj[u]$: Adjazenzliste von u (alle Nachbarknoten)
$wert[u]$: Abstand von u zum entstehenden Spannbaum

Prim(G, w, r)
01 $Q \leftarrow VG$ //Initialisierung
02 für alle $u \in Q$

```
03      wert[u] ← ∞
04      p[u] ← 0
05 wert[r] ← 0
06 solange Q ≠ ∅
07      u ← extractmin(Q)
08      für alle v ∈ Adj[u]
09          wenn v ∈ Q und w(u, v) < wert[v]
10              dann p[v] ← u
11                   wert[v] ← w(u, v)
```

(b) Algorithmus von Kruskal:
$G = (V, E, w)$: ein zusammenhängender, ungerichteter, kantengewichteter Graph

```
Kruskal(G)
01 E' ← ∅
02 L ← E
03 Sortiere die Kanten in L aufsteigend nach ihrem Kantengewicht.
04 solange L ≠ ∅
05      wähle eine Kante e ∈ L mit kleinstem Kantengewicht
06      entferne die Kante e aus L
07      wenn der Graph (V, E' ∪ {e}) keinen Kreis enthält
08          dann E' ← E' ∪ {e}
09 M = (V, E') ist ein minimaler Spannbaum von G.
```

5. Gegeben sei ein ungerichteter, gewichteter Graph $G = (V, E)$, mit positiver Gewichtsfunktion $w : E \to \mathbb{R}^+$. Ein spanning tree mit der Eigenschaft, dass sein größtes vorkommendes Kantengewicht minimal unter allen spanning trees ist, soll im folgenden K(anten)–M(inimal)–S(panning)–T(ree) genannt.

   (a) Ist jeder KMST ein M(inimal)–S(panning)–T(ree)? Falls ja, warum? Falls nein, finden Sie ein Gegenbeispiel eines gewichteten Graphen mit vier Ecken und vier Kanten.

   (b) Ist jeder MST ein KMST? Falls ja, warum? Falls nein, finden Sie ein Gegenbeispiel eines gewichteten Graphen mit vier Ecken und vier Kanten.