Übungsblatt 7 für "Diskrete und geometrische Algorithmen"

1.) Bei einem vergleichenden Sortierverfahren werden ausschließlich Vergleiche zwischen den Elementen $a_1, \ldots, a_n$ einer Eingabefolge herangezogen (also Vergleichsoperationen der Art $a_i \leq a_j$) um Informationen über die Reihenfolge der Elemente zu erhalten, wohingegen z.B. keinerlei Informationen über die Werte a_i selbst verwendet werden. Im folgenden setzen wir immer voraus, daß die Elemente a_i alle paarweise verschieden sind. Solch ein vergleichendes Sortierverfahren kann man abstrakt als Entscheidungsbaum ansehen, d.h. einen binären Baum, der die Vergleiche zwischen Elementen repräsentiert, die von einem speziellen Sortieralgorithmus auf einem Eingabefeld einer vorgegebenen Größe durchgeführt werden. Im Entscheidungsbaum beinhalten die internen Knoten die Vergleiche zwischen Elementen, wobei $i : j$ den Vergleich zwischen a_i und a_j angibt. Jedes Blatt im Entscheidungsbaum bezeichnet eine Permutation $(\pi(1), \pi(2), \ldots, \pi(n))$. Wenn wir einen Pfad im Entscheidungsbaum von der Wurzel bis zu einem Blatt verfolgen, dann hat der Sortieralgorithmus durch die auf dem Pfad in den internen Knoten angegebenen Vergleiche, die im Blatt angeführte Permutation der Eingabedaten als Reihenfolge mittels $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ festgelegt.

   Bei einem korrekten Sortieralgorithmus muß jedes der $n!$ Permutationen von n Elementen als eines der Blätter des Entscheidungsbaumes vorkommen. Indem Sie sich überlegen, welche Höhe h ein binärer Baum mindestens haben muß, damit alle $n!$ Permutationen als Blätter des Entscheidungsbaumes vorkommen können, zeigen Sie

$$h \geq \log_2(n!),$$

   wodurch Sie folgende untere Schranke für die benötigte Anzahl der Vergleiche $T(n)$ für einen vergleichenden Sortieralgorithmus im worst-case erhalten:

$$T(n) \geq \log_2(n!) = \Omega(n \log n).$$

2.) Falls ein Sortierverfahren nicht nur Vergleiche zwischen den Elementen heranzieht, sondern auch zusätzliche Informationen über die Eingabedaten verwendet, existieren in bestimmten Fällen Algorithmen, welche die oben erhaltene Schranke für den Zeitaufwand unterbieten. Wir betrachten solch ein Sortierverfahren, genannt Countingsort, welches als Eingabe voraussetzt, daß alle n Eingabeelemente ganze Zahlen aus der Menge $\{0, \ldots, k\}$ sind, wobei k eine fest vorgegebene natürliche Zahl ist. Countingsort bestimmt dabei für jedes Eingabeelement x die Anzahl der Elemente, die kleiner sind als x und verwendet diese Information zum Sortieren. Countingsort verwendet dabei neben dem Eingabefeld $A[1..n]$ noch das Feld $B[1..n]$, in dem die sortierte Ausgabe abgespeichert wird, sowie das Feld $C[0..k]$, wo Informationen über die Anzahl der Elemente aus der Eingabemenge mit einer bestimmten Größe gespeichert werden. Der Algorithmus lautet wie folgt:

```
COUNTING-SORT(A, B, k)
n := A.länge
sei C[0..k] ein neues Feld
FOR i = 0 TO k DO
  C[i] := 0
END DO
FOR j = 1 TO n DO
  C[A[j]] := C[A[j]] + 1
END DO
FOR i = 1 TO k DO
  C[i] := C[i] + C[i-1]
END DO
FOR j = n DOWNTO 1 DO
  B[C[A[j]]] := A[j]
  C[A[j]] := C[A[j]] -1
END DO
```

(a) Man betrachte ein kleines selbst gewähltes Bsp. an Hand dessen man die Arbeitsweise von Countingsort veranschaulicht.

(b) Man erkläre, warum der Algorithmus korrekt arbeitet.

(c) Man überlege sich, daß Countingsort eine Gesamtlaufzeit von $\Theta(n+k)$ besitzt.

3.) Gegeben sind n ganze Zahlen von 0 bis k. Geben Sie einen Algorithmus an, der die Eingabe vorverarbeitet und dann jede Anfrage nach der Anzahl der Zahlen, die in einem Intervall $[a..b]$ liegen, in der Zeit $\mathcal{O}(1)$ beantwortet. Ihr Algorithmus sollte für den Vorverarbeitungsschritt die Zeit $\Theta(n+k)$ benötigen.
Hinweis: adaptieren Sie Countingsort.

4.) Die Inzidenzmatrix eines gerichteten Graphen $G = (V, E)$, der keine Schlingen (also Kanten (v, v), mit $v \in V$) enthält, ist eine $|V| \times |E|$-Matrix $B = (b_{i,j})$ mit

$$b_{i,j} := \begin{cases} -1, & \text{falls Kante } j \text{ aus dem Knoten } i \text{ austritt,} \\ 1, & \text{falls Kante } j \text{ in den Knoten } i \text{ eintritt,} \\ 0, & \text{sonst.} \end{cases}$$

Beschreiben Sie, was die Einträge des Matrizenprodukts $B \cdot B^T$ darstellen (mit B^T die zu B transponierte Matrix).

5.) Gegeben sei die Adjazenzmatrix A eines gerichteten Graphens $G = (V, E)$, welcher keine Schlingen und keine Mehrfachkanten enthält. Eine universelle Senke in solch einem gerichteten Graphen G ist ein Knoten s mit Hingrad $d^-(s) = |V| - 1$ und Weggrad $d^+(s) = 0$. Man zeige, daß es möglich ist, durch Untersuchen der Adjazenzmatrix A in Laufzeit $\mathcal{O}(|V|)$ festzustellen, ob G solch eine universelle Senke enthält oder nicht.

6.) Ein ungerichteter einfacher (das heißt, er besitzt keine Schlingen und keine Mehrfachkanten) Graph $G = (V, E)$ heißt bipartit, falls es eine Zerlegung der Knotenmenge V in zwei disjunkte Teilmengen V_1 und V_2 gibt, also $V = V_1 \cup V_2$ mit $V_1 \cap V_2 = \emptyset$, sodaß jede Kante in E einen Knoten aus V_1 mit einem aus V_2 verbindet (und somit keine Kanten innerhalb von Knoten aus V_1 bzw. V_2 existieren). Ein bekannter Satz der Graphentheorie sagt aus (ein Beweis ist hier nicht verlangt), daß G genau dann bipartit ist, wenn er keine Kreise ungerader Länge besitzt. Man überlege sich nun einen auf der Breitensuche basierenden Algorithmus, welcher in Laufzeit $\mathcal{O}(|V| + |E|)$ entscheiden kann, ob ein gegebener ungerichteter einfacher Graph G bipartit ist und in diesem Fall auch gleich eine Partitionierung von V in zwei Teilmengen V_1 und V_2 liefert.

7.) Man studiere den Beweis des sogenannten "Klammerungstheorems" für die Tiefensuche.

Bei jeder Tiefensuche in einem (gerichteten oder ungerichteten) Graphen $G = (V, E)$ ist für jedes Paar von Knoten u und v genau eine der folgenden drei Bedingungen erfüllt:

- die Intervalle $[u.d, u.f]$ und $[v.d, v.f]$ sind paarweise disjunkt und weder u noch v ist im Tiefensuchwald ein Nachfahre des anderen.
- das Intervall $[u.d, u.f]$ ist vollständig im Intervall $[v.d, v.f]$ enthalten und u ist im Tiefensuchwald ein Nachfahre von v.
- das Intervall $[v.d, v.f]$ ist vollständig im Intervall $[u.d, u.f]$ enthalten und v ist im Tiefensuchwald ein Nachfahre von u.

8.) Man studiere den Beweis des sogenannten "Theorems der weißen Pfade" für die Tiefensuche.

In einem Tiefensuchwald eines (gerichteten oder ungerichteten) Graphen $G = (V, E)$ ist der Knoten v genau dann ein Nachfahre des Knotens u, wenn es zum Zeitpunkt $u.d$, zu dem die Durchmusterung den Knoten u entdeckt hat, einen Pfad von u nach v gibt, der nur aus weißen Knoten besteht.