

Aufgabe 1: Graphentheorie (2 + 1 + 2 Punkte)

Gegeben ist der folgende ungerichtete Graph $G_n = (V, E)$ mit

$$V = \bigcup_{i=0}^{n-1} V_i \quad \text{wobei} \quad V_i = \{v_{i,0}, \dots, v_{i,m_i}\}, \quad E = \{(v_{i,j}, v_{x,y}) \mid i \neq x\}.$$

Hinweis: Der Graph besteht aus n Knotengruppen, die jeweils aus $m_0, \dots, m_{n-1} > 0$ Knoten bestehen. Dabei sind die Knoten innerhalb einer Gruppe nicht miteinander verbunden, aber mit allen anderen Knoten außerhalb der Gruppe.

- Wieviele Kanten besitzt G_n abhängig von n und $m_0, \dots, m_{n-1} \in \mathbb{N}$.
- Die Kante $(x, y) \in E$ ist *transitiv*, wenn $\exists z \in V ((x, z) \in E \wedge (z, y) \in E)$. Wieviele transitiven Kanten hat G_n ?
- Skizzieren sie eine Adjazenzmatrix von G_n .

Aufgabe 2: Haskell (5 Punkte)

Betrachten Sie folgende Grammatik in EBNF Form:

$S = 'x' \mid XSX$.
 $X = 'a' \mid 'b' \mid \dots \mid 'z'$.

Implementieren Sie eine funktion `isValid :: String -> Bool` die überprüft ob ein Wort zur Sprache dieser Grammatik gehört. Beispiele:

```
isValid "baxab" = True
isValid "axxa"  = False
isValid "baab"  = False
isValid "xxxxx" = True
```

Hinweis: Sie dürfen die Standardbibliothek verwenden, folgende Funktionen könnten Ihnen behilflich sein:

```
head :: [a] -> a           -- Liefert das erste Element einer Liste
last  :: [a] -> a           -- Liefert das letzte Element einer Liste
tail  :: [a] -> [a]        -- Entfernt das erste Element aus einer Liste
init  :: [a] -> [a]        -- Entfernt das letzte Element aus einer Liste
```

Aufgabe 3: Prädikatenlogik (1 + 4 + 2 Punkte)

- Sei $F = \forall x_1 \forall x_2 (P_1(x_1, x_2) \wedge \neg(\forall x_3 P_2(x_1, x_2, x_3)))$ eine Prädikatenlogische Formel. Ist die Pränexnormalform von F identisch zur Skolemnormalform von F ? (1 Punkt)
- Sei $F = P(x, f(y), b)$. Geben Sie den kleinstmöglichen Unifikator von F mit folgenden Formeln an.
 - $G_1 = P(z, f(w), b)$

(ii) $G_2 = P(x, f(a), b)$

(iii) $G_3 = P(g(z), f(a), b)$

(iv) $G_4 = P(c, f(a), a)$

(c) Gegeben sei die Struktur (A, R, P) mit

$$A = \{1, 2, 3, 4, 5, 6, 7\}$$

$$R = \{(1, 2), (2, 3), (4, 5), (5, 6), (6, 7), (7, 4)\}$$

$$P = \{2, 4\}$$

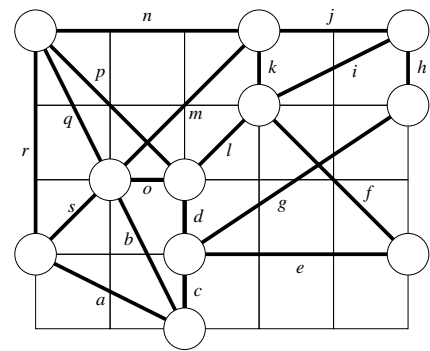
Beweisen oder widerlegen Sie die folgenden Aussagen.

(i) $\forall v_1 \exists v_2 R(v_1, v_2)$

(ii) $\forall v_1 \left(P(v_1) \rightarrow \exists v_2 \exists v_3 (R(v_2, v_1) \wedge R(v_1, v_3)) \right)$

Aufgabe 4: Algorithmus von Kruskal (4 + 4 Punkte)

- (a) Geben Sie in Pseudocode den Algorithmus von Kruskal zur Bestimmung des minimalen spannenden Baums in einem Graphen mit Kantengewichten an.
- (b) Geben Sie eine Reihenfolge an, in der der Algorithmus von Kruskal, angewandt auf den folgenden Graphen, die Kanten in den minimal spannenden Baum aufnimmt. In dem Graphen entspricht die Länge einer Kante ihrem Gewicht.



Aufgabe 5: Rekurrenzrelationen (3 Punkte)

Geben Sie die Rekurrenzrelation für den Aufwand (Rechenzeit) $T(n)$ für den Aufruf `foo(x, x+n)` an. Auflösen der Rekurrenz wird *nicht* verlangt. Die Komplexität von `bar(...)` sei konstant $O(1)$.

Hinweis: Der folgende Quelltext ist nicht als Java, sondern als Pseudo-Code zu verstehen.

```
void foo(int i, int j)
{
    if (j-1 < 4) return;
    bar (1, i);
    foo (i, i+(j-i)/4);
    for (int k=0; k<=j-i; k++) bar (k, j);
    foo (i+(j-i)/4 + 1, j);
}
```

Aufgabe 6: Rekurrenzen (4 + 4 Punkte)

Fortsetzung von Aufgabe 6 aus der Probeklausur:

- (d) Berechnen Sie mit Hilfe der Methode der generierenden Funktion eine geschlossene Form für f_n .
- (e) Beweisen Sie ausgehend von Teil (c) mit Hilfe vollständiger Induktion, dass Ihr Ergebnis aus Teil (d) richtig ist.

Aufgabe 7: Rekurrenzen (6 + 4 Punkte)

- (a) Gegeben sei folgende Rekurrenz: $f_0 = 0$, $f_n = f_{n-1} + (n - 1)$. Finden Sie für f_n eine geschlossene Form g_n und beweisen sie Ihr Ergebnis mit Hilfe vollständiger Induktion.
- (b) Schreiben Sie eine Haskell-Funktion, bei der sich die Anzahl der Additionen oder Multiplikationen nach der oben genannten Rekurrenz berechnet. Geben Sie auch an, was Ihre Funktion berechnet.

Aufgabe 8: Quicksort (1 + 2 Punkte)

Quicksort gilt als schnelles Sortierverfahren. Allerdings kann es in manchen Situationen nicht diesen Ansprüchen entsprechen. Die Wahl des *Pivotelements* ist für die Laufzeit des Algorithmus entscheidend.

In der Implementierung in dieser Aufgabe erhält die Sortierfunktion eine Liste, die mit Quicksort sortiert werden soll. Dabei wird als Pivotelement das erste Element der Liste gewählt.

- (a) Geben Sie eine Liste von 8 Elementen an, die Quicksort mit maximalen Aufwand sortiert (worst case).
- (b) Führen anhand ihrer Liste Quicksort durch, wobei als Pivotelement immer das erste Element in der Liste gewählt werden soll.
Mit welchem Aufwand wird die Liste sortiert (im O-Kalkül)? Es muss nicht jeder Schritt angegeben werden, aber Teilergebnisse müssen erkennbar sein.