Aufgabe 1 [10 · 4 Punkte] Welche der folgenden Aussagen gelten? Begründen Sie Ihre Aussage jeweils kurz. Ohne korrekte Begründung gibt es keine Punkte.

(a) $\log(n!) = \Theta(n \log n)$

(b) Für jede Teilmenge $M \subseteq \mathbb{N}$ gibt es ein C++-Programm P_M, das zu einer gegebenen natürlichen Zahl entscheidet, ob sie zu M gehört.

(c) Seien $x_1 \cdots x_l$ und $y_1 \cdots y_l$ die l-stelligen Komplementdarstellungen zur Basis 2 zweier ganzer Zahlen p und q, so dass $z_i := x_i + y_i \leq 1$ für alle $i = 1, \ldots, l$. Dann ist $z_1 \cdots z_l$ die l-stellige Komplementdarstellung zur Basis 2 von $p + q$.

(d) Es gibt einen polynomiellen Algorithmus, der zu zwei gegebenen natürlichen Zahlen (in Binärdarstellung) das kleinste gemeinsame Vielfache berechnet.

(e) Nicht jede reelle, aber jede rationale Zahl hat eine endliche normalisierte 2-adische Darstellung.

(f) Sei n die kleinste natürliche Zahl, die nicht in F_{double} liegt. Dann sind $n-1$ und $n+1$ beide in F_{double}.

(g) Die Funktion $x \mapsto \ln(1 + |x - 1|)$ ist für alle $x \in \mathbb{R} \setminus \{0\}$ gut konditioniert.

(h) Es gibt einen polynomiellen Algorithmus, der zu einer gegebenen Zahl $n \in \mathbb{N}$ (in Binärdarstellung) die Zahl $k \in \mathbb{N}$ berechnet, für die $k \leq \sqrt{n} < k + 1$ gilt.

(i) Merge-Sort funktioniert im Allgemeinen nicht, wenn die partielle Ordnung nicht durch Schlüssel induziert ist.

(j) Jede Matrix, die genau eine LU-Zerlegung besitzt, ist nichtsingulär.

Aufgabe 2 [10 Punkte] Beweisen Sie: Jede quadratische Matrix, deren Einträge alle nichtnegative ganze Zahlen sind, und deren Zeilen- und Spaltensummen alle gleich 2017 sind, ist Summe von 2017 Permutationsmatrizen.

– bitte wenden –

Aufgabe 3 [10 Punkte] Gegeben sei ein gerichteter Graph G mit Kantengewichten $c : E(G) \to \mathbb{R}$, so dass jeder Kreis in G positives Gesamtgewicht hat, sowie zwei Knoten s und t. Geben Sie einen polynomiellen Algorithmus an, der die Vereinigung der Kantenmengen aller kürzesten s-t-Wege in (G, c) berechnet. Zeigen Sie die Korrektheit. Welche Laufzeit können Sie erreichen?

Aufgabe 4 [5+5+5+5 Punkte] Betrachten Sie das folgende Programmstück in C++; ein Ausdruck der aus der Vorlesung bekannten Klasse `Graph` liegt Ihnen vor. Der Funktion `was_ist_das` soll stets ein ungerichteter Graph G übergeben werden.

(a) Für genau welche ungerichteten Graphen G berechnet die Funktion `was_ist_das` das Ergebnis `true`? Geben Sie eine kurze Begründung.

(b) Welche Laufzeit hat die Funktion `was_ist_das`? Warum?

(c) Begründen Sie, warum man die Symbole `&` nicht entfernen sollte.

(d) Wie muss man den Programmcode (möglichst geringfügig!) ändern oder ergänzen, so dass die Funktion genau dann `true` ergibt, wenn der ungerichtete Graph G ein Wald ist?

```
#include "graph.h"
#include <vector>

void visit(const Graph::NodeId nodeid, const Graph & graph,
           std::vector<bool> & visited)
{
    if (not visited[nodeid]) {
        visited[nodeid] = true;
        for (auto neighbor: graph.get_node(nodeid).adjacent_nodes()) {
            visit(neighbor.id, graph, visited);
        }
    }
}

bool was_ist_das(const Graph & graph)
{
    int result = 0;
    std::vector<bool> visited(graph.num_nodes(), false);
    for (Graph::NodeId s=0; s<graph.num_nodes(); s++) {
        if (not visited[s]) {
            result++;
            visit(s, graph, visited);
        }
    }
    return (result > 1);
}
```

Aufgabe 1

(a) Wahr. Es gilt $\log(n!) \leq \log(n^n) = n \log n$ und für $n \geq 4$: $\log(n!) > \log(\frac{n}{2}^{\frac{n}{2}}) = \frac{n}{2} \log(\frac{n}{2}) \geq \frac{1}{4} n \log n = \Omega(n \log n)$.

(b) Falsch. Es gibt überabzählbar viele solche Mengen, aber nur abzählbar viele C++-Programme.

(c) Wahr. Die Komplementdarstellung von $x \in \{-2^{l-1}, \ldots, 2^{l-1} - 1\}$ ist die Binärdarstellung von $f(x) = \begin{cases} x & \text{falls } x \geq 0 \\ x + 2^l & \text{falls } x < 0. \end{cases}$
Da bei der Addition kein Übertrag anfällt, ist $0 \leq f(p) + f(q) < 2^l$ und somit $f(p+q) = f(p) + f(q)$. Wegen $x_1 + y_1 \leq 1$ kann höchstens eine der Zahlen p und q negativ sein, weshalb mit p und q auch $p + q$ in $\{-2^{l-1}, \ldots, 2^{l-1} - 1\}$ liegt.

(d) Wahr. Es gilt $\mathrm{kgV}(a \cdot b) = \frac{a \cdot b}{\mathrm{ggT}(a,b)}$. Sowohl für Multiplikation und ganzzahlige Division (Schulmethode) als auch für die Berechnung des $\mathrm{ggT}(a, b)$ (Euklidischer Algorithmus) gibt es polynomielle Algorithmen.

(e) Falsch. Z.B. hat $\frac{1}{3}$ die (wie immer eindeutige) Darstellung $1{,}\overline{01} \cdot 10^{-2}$.

(f) Wahr. $n-1$ liegt nach Definition von n in F_{double}. Weiter ist $n = 2^{53} + 1$, also ist auch $n + 1 = 2^{53} + 2 = 2^{53}(1 + 2^{-52})$ in F_{double}, da man für die Binärdarstelung mit 52 Bits in der Mantisse auskommt.

(g) Falsch. Für $x > 1$ ist das die Funktion $f(x) = \ln(x)$, und die Kondition ist $\frac{|f'(x)| \cdot |x|}{|f(x)|} = \frac{1}{\ln x}$, was beliebig groß wird. Für $x = 1$ ist die Kondition ∞.

(h) Wahr. Binäre Suche leistet das: wegen $k \in \{1, \ldots, n\}$ reichen $O(\log n)$ Iterationen, und man kann für jede Zahl m in polynomieller Zeit entscheiden ob $m \leq \sqrt{n}$ gilt, indem man (z.B. mit Schulmethode) m^2 berechnet.

(i) Wahr. Betrachte die Grundmenge $\{a, b, c\}$ mit der partiellen Ordnung $\preceq$, bei der $c \prec a$ ist, aber andere Paare verschiedener Elemente unvergleichbar sind. Wenn Merge-Sort die Liste a, b, c in die Listen a und b, c aufteilt, sind diese bereits korrekt sortiert. Dann kann beim Merge wegen $b \not\preceq a$ das Element a an die erste Stelle gesetzt werden. Es ergibt sich die falsche Ordnung a, b, c.

(j) Falsch. Ein Gegenbeispiel ist $\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$. Diese Matrix ist offenbar singulär. Aber mit $\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & 0 \\ a & 1 \end{smallmatrix}\right)\left(\begin{smallmatrix} b & c \\ 0 & d \end{smallmatrix}\right) = \left(\begin{smallmatrix} b & c \\ ab & ac+d \end{smallmatrix}\right)$ folgt $b = 1$, $c = 1$, $a = 1$ und $d = 0$; die Matrix hat also genau eine LU-Zerlegung.

Aufgabe 2 Sei $\mathcal{M}_k$ die Menge der Matrizen, deren Einträge alle nichtnegative ganze Zahlen sind, und deren Zeilen- und Spaltensummen alle gleich k sind. Wir zeigen per Indutkion über k, dass für jedes $k \in \mathbb{N}$ jede Matrix in $\mathcal{M}_k$ Summe von k Permutationsmatrizen ist. Für $k = 0$ ist dies trivial, für $k = 2017$ ist dies die Behauptung.

Für den Induktionsschritt sei $k \geq 1$ und $A = (\alpha_{ij})_{i,j=1,\ldots,n} \in \mathcal{M}_k$. Betrachte den bipartiten Graph G mit Knotenmenge $\{z_1, \ldots, z_n\} \dot\cup \{s_1, \ldots, s_n\}$ und genau α_{ij} parallelen Kanten zwischen z_i und s_j $(i, j = 1, \ldots, n)$. Nach dem Heiratssatz enthält G ein perfektes Matching M, denn für $X \subseteq \{1, \ldots, n\}$ ist die Zahl der Nachbarn mindestens $\sum_{j=1}^{n} \frac{1}{k} \sum_{i \in X} \alpha_{ij} = \frac{k|X|}{k} = |X|$. Setzt man $\sigma(i) = j$ für $\{z_i, s_j\} \in M$, so erhält man eine Permutationsmatrix P mit $A - P \in \mathcal{M}_{k-1}$.

Aufgabe 3 Mit dem Moore-Bellman-Ford-Algorithmus berechnen wir zuerst (von s ausgehend) $\mathrm{dist}_{(G,c)}(s, v)$ für alle $v \in V(G)$ und dann (von t ausgehend nachdem bei jeder Kante die Richtung umgedreht wurde) $\mathrm{dist}_{(G,c)}(w, t)$ für alle $w \in V(G)$. Eine Kante $e = (v, w) \in E(G)$ liegt genau dann auf einem kürzesten s-t-Weg in (G, c), wenn $\mathrm{dist}_{(G,c)}(s, v) + c(e) + \mathrm{dist}_{(G,c)}(w, t) = \mathrm{dist}_{(G,c)}(s, t)$ ist, denn ein kürzester Kantenzug von s nach t muss ein Weg sein, weil das Entfernen etwaiger Kreise das Gewicht verringern würde. Die Laufzeit wird durch zwei Aufrufe von Moore-Bellman-Ford dominiert, ist also $O(mn)$.

Aufgabe 4

(a) Für unzusammenhängende. Das Programm implementiert Tiefensuche und markiert in der ersten Iteration der `for`-Schleife von `was_ist_das` alle vom Knoten 0 aus erreichbaren Knoten als visited. Der Wert von `result` ist dann 1 und wird in der Folge genau dann weiter erhöht, wenn es noch unerreichte Knoten gibt.

(b) Die Laufzeit ist linear, d.h. $O(n+m)$, wobei $n = |V(G)|$ und $m = |E(G)|$, denn in der `for`-Schleife von `visit` wird jede Kante genau zweimal besucht, und in der `for`-Schleife von `was_ist_das` werden alle Knoten durchlaufen.

(c) Das `&` vor `visited` ist notwendig, damit der Vektor `visited` beim Aufruf von `visit` nicht kopiert wird; der Vektor `visited` in `was_ist_das` würde dann nie geändert und das Programm würde nicht korrekt funktionieren. Ohne die anderen `&` würde der Graph bei jedem Funktionsaufruf kopiert und das Programm viel langsamer.

(d) Die folgende Funktion testet, ob der Graph ein Wald ist:

```
1  bool was_ist_das(const Graph & graph)
2  {
3      int result = 0;
4      int sum_degrees = 0;
5      std::vector<bool> visited(graph.num_nodes(), false);
6      for (Graph::NodeId s=0; s<graph.num_nodes(); s++) {
7          sum_degrees += graph.get_node(s).adjacent_nodes().size();
8          if (not visited[s]) {
9              result++;
10             visit(s, graph, visited);
11         }
12     }
13     return (sum_degrees/2 + result <= (int)(graph.num_nodes()));
14 }
```

(Dabei wird benutzt, dass die Zahl der Kanten genau zweimal die Summe der Knotengrade ist und dass ein Graph mit n Knoten, m Kanten und k Zusammenhangskomponenten genau dann ein Wald ist, wenn $m + k \leq n$ gilt.)