Probeklausur zum Modul *Algorithmische Mathematik I* 22.1.2013

Name: **Matrikelnr.:**

Aufgabe 1 [2+2+2 Punkte] Geben Sie für folgende Aussagen jeweils ein Beispiel an, das die Aussage beweist, und begründen Sie kurz, warum.

(a) Es gibt eine Instanz des Kürzeste-Wege-Problems (mit teilweise negativen Kantengewichten), bei der Dijkstras Algorithmus nicht die korrekte Lösung findet.

(b) Es gibt eine reelle Zahl, die eine endliche Binärdarstellung hat, aber keine endliche Darstellung zur Basis b für irgendein ungerades $b \geq 3$.

(c) Es gibt eine Matrix, deren Einträge alle ungleich null sind und die keine LU-Zerlegung hat.

Aufgabe 2 [2+2+2 Punkte] Welche der folgenden Aussagen gelten? Begründen Sie Ihre Aussage jeweils kurz.

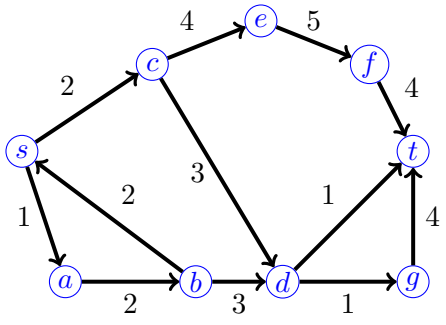
(a) Quicksort ist immer schneller als Merge-Sort.

(b) Addiert man zwei Maschinenzahlen x und y und rundet das Ergebnis zur nächsten Maschinenzahl, so ist der absolute Fehler, den die Rundung verursacht, höchstens $\min\{|x|, |y|\}$.

(c) Das Sieb des Erathosthenes ist ein polynomieller Algorithmus.

Aufgabe 3 [6 Punkte] Wie kann man in linearer Zeit entscheiden, ob es in einem gegebenen ungerichteten Graphen zwei Knoten gibt, die durch genau einen Weg verbunden sind?

Aufgabe 4 [6 Punkte] Berechnen Sie mit dem Edmonds-Karp-Algorithmus einen s-t-Fluss mit maximalem Wert in folgendem Netzwerk. Geben Sie das Endergebnis und die Knotenfolgen aller dabei benutzten augmentierenden Wege an.



Aufgabe 5 [6 Punkte] Sei G ein nichtleerer bipartiter Graph mit Bipartition $V(G) = A \dot{\cup} B$. Es gelte $\min\{|\delta_G(v)| : v \in A\} \geq \max\{|\delta_G(v)| : v \in B\}$. Zeigen Sie, dass G ein Matching enthält, das A überdeckt.

Aufgabe 6 [10 Punkte]

Betrachten Sie das folgende Programmstück in C++. Füllen Sie die Lücke (Zeile 18 bis 30) so, dass diese Funktion korrekt eine topologische Ordnung ausgibt, wenn der Digraph azyklisch ist, und andernfalls meldet, dass er einen Kreis enthält.

```
1  int topological_order(Graph const & graph)
2  {
3        std::vector<Graph::NodeId> sorted_nodes;
4        std::vector<int> indegrees(graph.num_nodes());
5
6        for (Graph::NodeId v = 0; v != graph.num_nodes(); v++)
7        {
8              if (graph.get_node(v).in_edges().empty())
9              {
10                   sorted_nodes.push_back(v);
11             }
12             indegrees[v] = graph.get_node(v).in_edges().size();
13       }
14
15       int i = 0;
16       while (i < sorted_nodes.size())
17       {
18
19
20
21
22
23
24
25
26
27
28
29
30
31             i++;
32       }
33
34       if (i < graph.num_nodes())
35       {
36             std::cout << "The digraph contains a circuit.\n";
37             return 1;
38       }
39       else
40       {
41             std::cout << "The graph is acyclic. Topological order:\n";
42             for (Graph::NodeId v = 0; v != graph.num_nodes(); v++)
43             {
44                   std::cout << sorted_nodes[v] << " ";
45             }
46             std::cout << "\n";
47             return 0;
48       }
49 }
```

Aufgabe 1

(a): Sei $G = (\{s, a, b, t\}, \{(s, a), (s, b), (a, b), (b, t)\})$ mit $c((s, a)) = 3$, $c((a, b)) = -3$ und $c((s, b)) = c((b, t)) = 1$. Dann wählt Dijkstras Algorithmus in dieser Reihenfolge s, b, t und a aus und berechnet daher 2 als Abstand von s nach t, obwohl 1 richtig wäre.

(b): Die Zahl $\frac{1}{2}$ hat Binärdarstellung $1 \cdot 2^{-1}$, aber für $b \geq 3$ ungerade ist $\frac{1}{2} = \sum_{i=1}^{\infty} \frac{b-1}{2} \cdot b^{-i}$.

(c): Die Matrix $\left(\begin{smallmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{smallmatrix}\right)$ hat keine LU-Zerlegung, denn sonst wäre $L = \left(\begin{smallmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ x & y & 1 \end{smallmatrix}\right)$ und $U = \left(\begin{smallmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & z \end{smallmatrix}\right)$, und x müsste gleichzeitig 1 und 2 sein.

Aufgabe 2

(a): falsch, denn Quicksort braucht z.B. bei bereits sortierter Eingabe $\Theta(n^2)$ Vergleiche, Merge-Sort hat aber Worst-Case-Laufzeit $O(n \log n)$.

(b): wahr, denn die zu $x + y$ nächste Maschinenzahl kann nicht weiter von $x + y$ entfernt sein als x und nicht weiter als y, also ist der absolute Fehler höchstens $|x+y-x|$ und höchstens $|x + y - y|$.

(c): falsch, denn es hat Laufzeit $\Omega(n)$, die Eingabe n kann aber (in Binärdarstellung) mit $O(\log n)$ Bits kodiert werden.

Aufgabe 3

Der gegebene Graph G sei o.B.d.A. zusammenhängend, da wir sonst das folgende Verfahren auf jede Zusammenhangskomponente einzeln anwenden können. Wir müssen nur nach Brücken in G suchen: die Endpunkte einer Brücke sind durch genau einen Weg miteinander verbunden; und falls zwei Knoten v und w durch nur einen Weg P miteinander verbunden sind, müssen alle Kanten von P Brücken sein.

Wir führen DFS mit beliebigem Startknoten r durch und erhalten einen DFS-Baum T. Wie nummerieren die Knoten $l : V(G) \to \mathbb{N}$ in der Reihenfolge, in der sie in die Menge Q aufgenommen werden. Nur Kanten aus T können Brücken sein.

Wir orientieren T als Arboreszenz T' mit Wurzel r. Für $v \in V(G) \setminus \{r\}$ sei $p(v)$ der Vorgänger von v in T', X_v die Menge der Knoten w, für die v auf dem r-w-Weg in T liegt (insbesondere ist also $v \in X_v$ und $p(v) \notin X_v$), und $k(v) := \min\{l(u) : \exists w \in X_v \text{ mit } \{u, w\} \in E(G) \setminus E(T)\}$ (wobei wie üblich $\min \emptyset = \infty$ sei). Diese Zahlen können rekursiv, beginnend bei den Blättern von T', in linearer Zeit berechnet werden, denn $k(v) = \min\big\{\min\{l(u) : \{u, v\} \in \delta(v) \setminus E(T)\}, \min\{k(w) : (v, w) \in \delta^+_{T'}(v)\}\big\}$.

Eine Kante $\{p(v), v\} \in E(T)$ (mit $v \in V(G) \setminus \{r\}$) ist offenbar genau dann Brücke von G, wenn $k(v) \geq l(v)$ ist.

Aufgabe 4

Man augmentiert (jeweils immer um 1) entlang des Weges s-c-d-t, dann entlang s-c-d-g-t, dann entlang s-a-b-d-c-e-f-t. Im Ergebnis tragen alle Kanten Fluss 1, außer (b, s) mit Fluss 0 und (s, c) mit Fluss 2.

Aufgabe 5

Für alle $X \subseteq A$ ist $|\Gamma(X)| \geq \frac{|\delta(X)|}{\max\{|\delta_G(v)|:v\in B\}} \geq \frac{\min\{|\delta_G(v)|:v\in A\}\cdot|X|}{\max\{|\delta_G(v)|:v\in B\}} \geq |X|$. Man kann also $|B| - |A|$ neue Knoten ergänzen und diese jeweils mit allen Knoten in B verbinden, und der entstehende Graph hat nach dem Heiratssatz ein perfektes Matching. Dieses enthält ein A überdeckendes Matching in G.

Aufgabe 6

```
int topological_order(Graph const & graph)
{
	std::vector<Graph::NodeId> sorted_nodes;
	std::vector<int> indegrees(graph.num_nodes());

	for (Graph::NodeId v = 0; v != graph.num_nodes(); v++)
	{
		if (graph.get_node(v).in_edges().empty())
		{
			sorted_nodes.push_back(v);
		}
		indegrees[v] = graph.get_node(v).in_edges().size();
	}

	int i = 0;
	while (i < sorted_nodes.size())
	{
		Graph::NodeId v = sorted_nodes[i];

		for (Graph::EdgeId e = 0; e != graph.get_node(v).out_edges().size(); e++)
		{
			Graph::NodeId head = graph.get_edge(graph.get_node(v).out_edges()[e
			    ]).get_head();

			indegrees[head]--;

			if(indegrees[head] == 0)
			{
				sorted_nodes.push_back(head);
			}
		}
		i++;
	}

	if (i < graph.num_nodes())
	{
		std::cout << "The digraph contains a circuit.\n";
		return 1;
	}
	else
	{
		std::cout << "The graph is acyclic. Topological order:\n";
		for (Graph::NodeId v = 0; v != graph.num_nodes(); v++)
		{
			std::cout << sorted_nodes[v] << " ";
		}
		std::cout << "\n";
		return 0;
	}
}
```