Algorithmen und Komplexität
Lösungsskizze Klausur Winter 2017

Lösungsvorschlag zu Aufgabe 1

(a) Wahr. Da $\exp(-1) \leq \exp(\cos(n)) \leq \exp(1)$.

(b) Falsch. Merge Sort hat Laufzeit $\mathcal{O}(n \log n)$. Wenn Quick-Sort immer das kleinste verbleibende Element auswählt, hat Quick-Sort Laufzeit $\Theta(n^2)$.
Bemerkung: Untere Schranke für Quicksort (Ω oder Θ) muss angegeben werden .

(c) Falsch. Das CONSILIDATE kann $\omega(1)$ Zeit benötigen, zum Beispiel wenn $\omega(1)$ Elemente ohne Kinder in der Wurzelliste sind.

(d) Wahr. TRIPARTITESMATCHING ist in $\mathcal{NP}$ und 3-SAT ist $\mathcal{NP}$ vollständig.

Punkteschema:
Richtige Antwort je 1 Punkt
Richtige Begründung je 2 Punkte (1 Punkt bei kleinem Fehler)

Lösungsvorschlag zu Aufgabe 2

(a) Falsch. Gegenbeispiel: Drei Dreiecke, die mit jeweils einer Kante mit einem zusätzlichen Knoten verbunden sind.

(b) Falsch. Betrachte den Baum mit Schlüsseln $1, 3, 6, 7$, der Tiefe 2 hat.

(c) Wahr. Nehme an es existiert ein minimaler Spannbaum T welcher die Kante $e = (u, v)$ mit dem kleinsten Gewicht nicht enthält. Fügt man e zu T hinzu so bildet sich ein Kreis C. Wenn man eine andere Kante aus C entfernt so bleibt der Graph zusammenhängend und hat wieder $|V| - 1$ Kanten. Somit haben wir einen Spannbaum mit kleinerem Gewicht als T erhalten. Widerspruch.

(d) Wahr. Zum Beispiel: $a = (1, 1, 1, 1)$.

Punkteschema:
Richtige Antwort je 1 Punkt
Richtige Begründung je 3 Punkte (2 oder 1 Punkt bei kleinem Fehler)

Lösungsvorschlag zu Aufgabe 3

(a) Nehme an der kürzeste Kreis $p_1, \ldots, p_\ell$ hat Länge $\ell > 3$. 1. Fall: $(p_3, p_1) \in A$, dann ist p_1, p_2, p_3 ein Kreis der Länge 3. 2. Fall: $(p_1, p_3) \in A$, so bildet $p_1, p_3, p_4, \ldots p_\ell$ einen Kreis der Länge $\ell - 1$. Widerspruch zur Annahme. Es folgt, dass es entweder einen Kreis der Länge 3 gibt, oder gar keinen Kreis.

(b) Wir benutzen $\textsc{Select}(a[1 \ldots n], \lceil \log n \rceil)$ um das $\lceil \log n \rceil$ kleinste Element x zu finden. Anschliessend vergleichen wir alle a_i mit x und geben die a_i aus die kleinergleich x sind.

Korrektheit: Laut Vorlesung ist $\textsc{Select}$ korrekt, und wir geben die Elemente aus die kleinergleich dem $\lceil \log n \rceil$ kleinstem Element sind.

Laufzeit: Sowohl $\textsc{Select}$ als auch der zweite Schritt benötigt Zeit $\mathcal{O}(n)$.

Punkteschema:
a) 8 Punkte
Idee, dass man einen Kreis kleiner machen kann 4 Punkte
Formal korrekt begründet 4 Punkte
b) 8 Punkte
Korrekter Algorithmus 8 Punkte
Idee: $\textsc{Select}(a[1 \ldots n], \lceil \log n \rceil)$ aufzurufen. 2 Teilpunkte
Falls Korrektheit oder Laufzeit fehlt, je minus 2 Punkte

Lösungsvorschlag zu Aufgabe 4

Wir zeigen zuerst, dass SUPER-SAT in $\mathcal{NP}$ liegt.

Angenommen eine KNF Formel F ist in der Sprache SUPER-SAT. Dann gibt es eine Belegung ω, sodass in jeder Klausel zwei Literale erfüllt sind. Wir benutzen ω als Zeugen. Da ω lediglich jeder Variabel 0 oder 1 zuordnet, ist die Länge des Zeugen $|w|$ polynomiell in $|F|$. Gegeben ω können wir in polynomieller Zeit jede Klausel durchgehen und schauen ob dort mindestens zwei Literale erfüllt sind.

Als zweites zeigen wir, dass SUPER-SAT $\mathcal{NP}$-schwer ist, indem wir von der Sprache SAT reduzieren.

Gegeben eine KNF Formel $F = \wedge_{i=1\ldots m} C_i$. Erzeuge daraus eine KNF Formel $F' = \wedge_{i=1\ldots m} C_i'$, wobei $C_i' = C_i \vee y$ (y ist eine Hilfsvariable).

Behauptung: F besitzt genau dann eine erfüllende Belegung, wenn F' eine Belegung besitzt in der jede Klausel mindestens 2 erfüllte Literale besitzt.

Beweis: Wenn F eine erfüllende Belegung besitzt, so nehme dieselbe Belegung und setze $y = 1$. Dann hat F' zwei erfüllte Literale pro Klausel. Wenn F' eine Erfüllende Belegung mit zwei erfüllten Literalen pro Klausel besitzt, so erfüllt diese Belegung (ohne y) auch F.

Klarerweise kann F' in polynomieller Zeit aus F erzeugt werden. Es folgt dass SUPER-SAT $\mathcal{NP}$-schwer ist.

Punkteschema:
SUPER-SAT $\in \mathcal{NP}$, 5 Punkte
SUPER-SAT ist $\mathcal{NP}$-schwer, 11 Punkte

Lösungsvorschlag zu Aufgabe 5

Definition: Sei $a_{i,j}$ die Länge des grössten monotonen Matchings, wenn nur die ersten i Stellen von x und die ersten j Stellen von y verwendet werden dürfen.

Rekursionsformel: Falls $x_i \neq y_j$ oder $i = j$, dann $a_{i,j} = \max\{a_{i-1,j}, a_{i,j-1}\}$. Sonst $a_{i,j} = 1 + a_{i-1,j-1}$.

Begründung Rekursionsformel: Falls $x_i \neq y_j$ oder $i = j$, so ist x_i nicht mit y_i verbunden. Darum können x_i und y_j nicht gleichzeitig im monotonen Matching vorkommen (sonst würden sich zwei Linien überkreuzen). Es folgt, dass entweder x_i oder y_j weggelassen werden kann. Im anderen fall sind x_i und y_j verbunden. Sei (x_ℓ, y_k) die letzte kannte die im grössten monotonen Matching enthalten ist. Ersetzen wir diese Kante durch (x_i, y_j) erhalten wir ein Matching derselben Grösse, demnach ist (x_i, y_j) in mindestens einem grössten monotonen Matching enthalten und es gilt $a_{i,j} = 1 + a_{i-1,j-1}$.

Algorithmus :

Algorithm 1 GRÖSSTES MONOTONES MATCHING(gleiche Arrays $x[1..n], y[1..n]$)

```
Initialisieren:
for i = 0...n do
  a_{i,0} = a_{0,i} = 0
end for
Berechnung aller a_{i,j}:
for i = 1...n do
  for j = 1...n do
    if x_i ≠ y_j oder i = j then
      a_{i,j} = max{a_{i-1,j}, a_{i,j-1}}
    else
      a_{i,j} = 1 + a_{i-1,j-1}
    end if
  end for
end for
```

Punkteschema:
Definition: 2 Punkte
Rekursionsformel: 5 Punkte
Begründung Rekursionsformel: 5 Punkte
Algorithmus: 8 Punkte
Laufzeit fehlt minus 2 Punkte
Falsche Initialisierung minus 2 Punkte

Lösungsvorschlag zu Aufgabe 6

Bezeichne $a[v]$ die Anzahl kürzester s-v Pfade in G. Wir modifizieren die Breitensuche, sodass für jeden Knoten $a[v]$ berechnet wird.

Initialisiere $a[v]$ für alle $v \in V$: $a[v] = 0$ und setze $a[s] = 1$.

Innerhalb der while-Schleife modifizieren wir die for-Schleife folgendermassen:

```
for all v ∈ Γ(v) do
  if d[u] = ∞ then
    d[u] = d[v] + 1
    pred[u] = v
    Q.INSERT(u)
  end if
  if d[u] = d[v] + 1 then
    a[u] = a[u] + a[v]
  end if
end for
```

Korrektheitsbeweis: Wir haben in der Vorlesung gesehen, dass $d[v]$ die Distanz von s nach v angibt. Ein kürzester s-v Pfad ist demnach eine Abfolge von $d[v]+1$ Knoten $s = p_0, p_1, \ldots, p_{d[v]} = v$ mit $(p_i, p_{i+1}) \in E$. Es gilt $d[p_i] = i$. Die Anzahl kürzester s-v Pfade können wir folgendermassen darstellen.

$$\begin{aligned}
a[v] &= |\{p_0, \ldots, p_{d[v]} | s = p_0, v = p_{d[v]}, (p_i, p_{i+1}) \in E \text{ for } i = 0, \ldots, d[v]-1\}| \\
&= |\{p_0, \ldots, p_{d[v]-1} | s = p_0, (p_{d[v]-1}, v) \in E, (p_i, p_{i+1}) \in E \text{ for } i = 0, \ldots, d[v]-2\}| \\
&= \sum_{u \in \Gamma(v) | d[u] = d[v]-1} |\{p_0, \ldots, p_{d[v]-1} | s = p_0, u = p_{d[v]-1}, (p_i, p_{i+1}) \in E \text{ for } i = 0, \ldots, d[v]-2\}| \\
&= \sum_{u \in \Gamma(v) | d[u] = d[v]-1} a[u]
\end{aligned}$$

Der Algorithmus berchnet für $a[v]$ genau diese Summe und liefer deshalb das korrekte Resultat.

Laufzeitanalyse: Analog zur Breitensuche, ausser, dass wir hier nur die Zusammenhangskomponente von s durchsuchen müssen. Deshalb $\mathcal{O}(|E|)$.

Punkteschema:

Algorithmus: 10 Punkte

-Bemerkung, dass Breitensuche die Distanz $d[v]$ berechnet oder Breitensuch findet kürzesten Pfad, 2 Punkte
-Idee $a[v]$ einzuführen und während der Breitensuche irgendwie aufzusummieren, 3 Punkte
-Falls Laufzeitanalyse fehlt max 8/10 Punkte möglich für Algorithms.

Korrektheitsbeweis: 10 Punkte