ETH Zürich
Institut für Theoretische Informatik
Prof. Dr. Angelika Steger
Ralph Keusch, Florian Meier

HS 2016

Algorithmen und Komplexität
Ferienserie

Die Ferienserie ist als Übungsmaterial gedacht und wird nicht korrigiert. Mit * markierte Aufgaben sind besonders schwer.

Wir wünschen frohe Weihnachten und ein gutes neues Jahr!

$\star \ \star \ \star$

Aufgabe 1

Sofern nichts anderes angegeben ist, verstehen sich alle Limes für $n \to \infty$. Beweisen oder widerlegen Sie:

a) $\binom{n}{k} = \Theta(n^k)$ für $k \in \mathbb{N}$ konstant

b) $\binom{2n}{n} = \Omega(2^n)$

c) $2^{2^n} = \omega(n^n)$

d) $x \log x = o(1/x)$ für $x \to 0$

e) $(-2)^n = \Omega(n)$

f) $1 + (-1)^n = \Omega(1)$

Aufgabe 2

a) Schreiben Sie ein *rekursives* Programm zur Berechnung von $n!$.

b) Schreiben Sie ein *iteratives* Programm zur Berechnung von $n!$.

c) Warum bringt ein *dynamisches* Programm hier keinen Vorteil?

d) *Tail recursion* ist Rekursion, die nur "am Ende" vorkommt, also in der Form

return $f(\ldots)$

Beweisen Sie, dass tail recursion äquivalent ist zu Schleifen, d.h. geben Sie eine Transformation zwischen den beiden Konstruktionen an, die den Sinn des Programms erhält. Kann diese Transformation auf Ihr Programm von Teil (a) angewandt werden?

Aufgabe 3

Seien S und T zwei Zeichenfolgen.

a) Die *Hamming-Distanz* zwischen S und T, für $|S| = |T|$, ist die Anzahl Zeichen, die in S geändert werden müssen, um T zu erhalten. Schreiben Sie ein Programm, das die Hamming-Distanz berechnet. Welche Laufzeit erreichen Sie?

b) Die *Levenshtein-Distanz* zwischen S und T (wobei i.A. $|S| \neq |T|$) ist die minimale Anzahl Zeichen, die in S geändert, gelöscht, oder hinzugefügt werden müssen, um T zu erhalten. Zum Beispiel ist die Distanz zwischen "suchen" und "super" 3.

Schreiben sie ein dynamisches Programm, das die Levenshtein-Distanz berechnet. Welche Laufzeit erreichen Sie?

Aufgabe 4

Angenommen wir haben einen Algorithmus $\mathcal{A}$, der in polynomieller Zeit das *Entscheidungsproblem* SAT löst. Schreiben Sie einen Algorithmus, der $\mathcal{A}$ verwendet und für eine beliebige KNF-Formel F in polynomieller Zeit eine erfüllende Belegung findet, sofern F erfüllbar ist.

Aufgabe 5

Ein *Vertex Cover* eines Graphen $G = (V, E)$ ist eine Teilmenge $W \subseteq V$, so dass für jede Kante $\{u, v\} \in E$ mindestens einer der Endknoten u oder v in W liegt.

Das VERTEX COVER Entscheidungsproblem ist wie folgt definiert:

- *Eingabe:* Graph G und $k \in \mathbb{N}$.
- *Ausgabe:* JA, falls G ein Vertex Cover der Grösse k besitzt; NEIN, sonst.

Beweisen Sie, dass VERTEX COVER $\mathcal{NP}$-vollständig ist,

a) indem Sie von CLIQUE (siehe Serie 12) reduzieren.

Hinweis: Ein *Independent Set* ist eine Teilmenge $U \subseteq V$, so dass jeder Knoten in U *keinen* Nachbarn in U besitzt. Es gibt direkte Beziehungen zwischen Independent Set und Vertex Cover bzw. Clique.

b)* indem Sie von 3-SAT reduzieren.

Aufgabe 6

Ein sortiertes Array ist eine Datenstruktur von n Elementen aus einem geordneten Universum, auf der man INSERT in $\mathcal{O}(n)$ und FIND (binäre Suche) in $\mathcal{O}(\log n)$ Zeit implementieren kann. Wir betrachten nun folgende Alternative:

Angenommen die Ziffern von n in Basis 2 sind $n_k \dots n_0$, wobei $k = \lfloor \log_2 n \rfloor$. Für $i = 0, \dots, k$ definieren wir A_i als ein sortiertes(!) Array der Grösse 2^i. Jedes A_i ist voll (enthält genau 2^i Elemente), falls $n_i = 1$; ansonsten ist A_i leer/ungenutzt. Somit enthält die Datenstruktur tatsächlich n Elemente. Zwischen A_i und A_j, $i \neq j$, oder ihren Elementen besteht keine Ordnungsrelation. (Siehe das Beispiel in Abbildung 1.)

a) Wie lässt sich auf dieser Datenstruktur FIND effizient implementieren? Was ist seine Laufzeit?

b)* Implementieren Sie INSERT mit einer amortisierten Laufzeit von $\mathcal{O}(\log n)$, und beweisen Sie diese Abschätzung. Was ist die worst-case Laufzeit?

Hinweis: Welche Laufzeit benötigen Sie, um zwei sortierte Arrays der Länge m zu einem zusammenzufügen?

Aufgabe 7

Der Chef der Firma POLYALGO möchte eine Weihnachtsfeier für die Angestellten organisieren. Die Firma ist streng hierarchisch organisiert: jeder der n Mitarbeiter (ausser der Chef) hat genau einen Vorgesetzten. Von jedem Mitarbeiter ist bekannt, wer sein Vorgesetzter und wer seine Untergebenen sind. Ausserdem hat die HR-Abteilung für jeden Mitarbeiter i einen Geselligkeitswert $f(i) \in \mathbb{R}$ erstellt, der aussagt, wie gut der Mitarbeiter der Firmenstimmung tut. Damit die Stimmung an der Weihnachtsfeier gut ist, darf kein Mitarbeiter zusammen mit seinem Vorgesetzten eingeladen werden.

Finden Sie einen Algorithmus, der in Laufzeit $O(n)$ eine Gästeliste $S \subset V$ herausfindet, so dass für jeden Mitarbeiter $i \in S$ sein Vorgesetzter j nicht in S enthalten ist, und so dass der gesamte Geselligkeitswert $f(S) := \sum_{i \in S} f(i)$ maximiert wird. Verwenden Sie dazu das Konzept der Dynamischen Programmierung, und finden sie zuerst den optimalen Wert $f(S)$ heraus.

Aufgabe 8*

Sei $G = (V, A)$ ein gerichteter Graph. Wir schreiben $u \rightsquigarrow v$, falls in G ein gerichteter Pfad von u nach v existiert. G heisst *stark zusammenhängend*, falls für alle $u, v \in G$ gilt: $u \rightsquigarrow_H v$ und $v \rightsquigarrow_H u$. Die *starken Zusammenhangskomponenten* (SZK) von G sind die maximalen stark zusammenhängenden Teilgraphen.

Entwerfen Sie einen Algorithmus, der G in seine SZK zerlegt.

Hinweis: Wir definieren die Transponierte $G^{\mathrm{T}} = (V, A^{\mathrm{T}})$ eines Graphen $G = (V, A)$ durch $A^{\mathrm{T}} = \{(v, u) \mid (u, v) \in A\}$. Wir schreiben ausserdem $f[v]$ für die Zeit, zu der die Tiefensuche den Knoten v und all seine Nachfolger vollständig abgesucht hat, und zum Vorgänger von v zurückgeht. Weiter definieren wir $f(U) = \max_{u \in U} f[u]$ für $U \subseteq V$. Beweisen Sie folgendes Lemma:

Lemma. *Seien C_1 und C_2 verschiedene SZK von $G = (V, A)$. Wenn es eine Kante $(u, v) \in A$ gibt mit $u \in C_1$ und $v \in C_2$, dann ist $f(C_1) > f(C_2)$.*

Benutzen sie dann je eine Tiefensuche auf G und G^{T}.

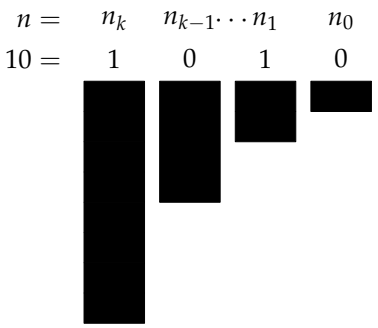

Abbildung 1: Beispiel-Datenstruktur