ETH Zürich
Institut für Theoretische Informatik
Prof. Dr. Angelika Steger, Dr. Johannes Lengler
Übungsleitung: Florian Meier

HS 2019

Algorithmen und Komplexität
Ferienserie

Folgende Aufgaben dienen als zusätzliche Möglichkeit zur Vorbereitung auf die Prüfung. Sie werden von den Assistenten nicht korrigiert.

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

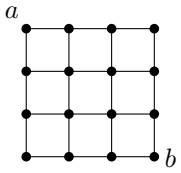
d) $x \log x = o(1/x)$ für $x \to 0$

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

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

Aufgabe 2

In dieser Aufgabe betrachten wir Gittergraphen $M_{n,n} = (V, E)$, wobei $V = \{v_{i,j} \mid 1 \leq i, j \leq n\}$ und $E = \{\{v_{i,j}, v_{i',j'}\} \mid |i - i'| + |j - j'| = 1\}$. Im Beispiel $n = 4$ sieht dieser Graph wie folgt aus:



$M_{4,4}$

Es seien weiter $a := v_{1,1}$ und $b := v_{n,n}$ zwei Knoten mit maximaler Distanz in $M_{n,n}$.

(a) Bestimmen Sie die Anzahl Knoten und die Anzahl Kanten in $M_{n,n}$.

(b) Berechnen Sie die Anzahl kürzester a-b-Pfade in $M_{n,n}$.

(c) Sei G ein zusammenhängender Teilgraph von $M_{n,n}$, der alle Knoten von $M_{n,n}$ enthält. Entwerfen Sie einen Algorithmus mit Laufzeit $O(n^2)$, der die Länge eines kürzesten $a - b-$Pfades in G berechnet. Zeigen Sie Korrektheit und Laufzeit.

Aufgabe 3

Unter einem Digraphen (engl. *directed graph*) $D = (V, A)$ versteht man einen Graphen, in dem die Kanten A Richtungen haben, d.h. $A \subseteq V \times V$ besteht aus *geordneten Paaren*. Eine *topologische Ordnung* von D ist eine Nummerierung der Knoten $f \colon V \to \mathbb{N}$ mit der Eigenschaft

$$(u, v) \in A \Rightarrow f(u) < f(v).$$

Beschreiben Sie einen Algorithmus, der in Laufzeit $\mathcal{O}(|V| + |A|)$ eine topologische Ordnung von D findet oder feststellt, dass eine solche nicht existiert. Geben Sie Ihren Algorithmus auch in Pseudo-Code an.

Hinweis: Sie dürfen annehmen, dass der Digraph als Adjanzenzliste gespeichert ist. Für jeden Knoten gibt es zwei Listen: eine Liste für alle durch *ausgehende* Kanten erreichbaren Nachbarn, und eine Liste für alle durch *eingehende* Kanten erreichbaren Nachbarn.

Aufgabe 4

Sei $G = (V, E)$ ein Graph. Für $u, v \in V$ sei die Distanz $d(u, v)$ als Länge eines kürzesten u-v-Pfades definiert. Weiter definieren wir für $v \in V$ seine *Exzentrizität*

$$ecc(v) = \max_{u \in V} d(v, u)$$

als die Länge des längsten kürzesten Pfades von v zu einem anderen Knoten u. Ein Knoten v mit

$$ecc(v) = \min_{u \in V} ecc(u)$$

heisst *Zentrum* von G. Man beachte, dass das Zentrum eines Graphen nicht eindeutig bestimmt ist.

Geben Sie einen Algorithmus an, der in Laufzeit $\mathcal{O}(|V|)$ ein Zentrum eines *Baumes* bestimmt.

Aufgabe 5

Es seien zwei aufsteigend sortierte Arrays $A = [a_1, \ldots, a_n]$, sowie $B = [b_1, \ldots, b_n]$ gegeben. Wir nehmen an, dass die Elemente a_i und b_i paarweise verschieden sind.

(a) Entwerfen Sie einen Algorithmus, der als Input ein Paar (i, k) mit $1 \leq k \leq 2n$ und $1 \leq i \leq n$ bekommt, und welcher in Zeit $\mathcal{O}(1)$ entscheidet, ob a_i das k-grösste[1] aller $2n$ Elemente ist oder nicht.

(b) Entwerfen sie einen Algorithmus, der das k-grösste Element aller $2n$ Elemente in $\mathcal{O}(\log n)$ Schritten findet.

(c) Entwerfen Sie einen Algorithmus, der eine Zahl x als Input bekommt, und welcher in Zeit $\mathcal{O}(n \log n)$ entscheidet, ob es $a \in A$ und $b \in B$ gibt, sodass $a + b = x$ gilt.

(d) Entwerfen Sie für die Aufgabe c) einen Algorithmus mit Laufzeit $\mathcal{O}(n)$.

Aufgabe 6

Gegeben sei eine aufsteigend sortierte Liste L von n ganzen Zahlen, und eine weitere ganze Zahl x. Entwerfen Sie einen Algorithmus, der in Zeit $\mathcal{O}(n)$ entscheidet, ob es zwei Elemente $a, b \in L$ gibt mit $a + b = x$.

Aufgabe 7

Alice und Bob spielen *Mastermind* in folgender Variante: Jeder Spielstein ist mit einer von 4 verschiedenen Farben aus $F = \{1, 2, 3, 4\}$ gefärbt. Vor dem Spiel wählt Alice, nicht einsehbar für Bob, eine Kombination von drei nicht notwendigerweise verschiedenfarbigen Spielsteinen, d.h. ein Tripel $a = (a_1, a_2, a_3) \in F^3$. Bobs Ziel ist es, dieses Tripel in möglichst wenigen Runden zu erraten. In jeder Runde wählt Bob ein Tripel $b = (b_1, b_2, b_3) \in F^3$. Danach erfährt er von Alice,

[1] das 1.-grösste ist das Maximum, das 2.-grösste ist das nächstkleinere, usw.

wie viele Übereinstimmungen es zwischen a und b gibt, d.h., für wie viele Indizes $i \in \{1,2,3\}$ die Gleichheit $a_i = b_i$ gilt. Gilt z.B. $a = (1,2,1)$ und $b = (1,1,1)$, so erfährt Bob, dass er zwei Übereinstimmungen hat (jedoch nicht, welche Positionen korrekt sind). Gilt hingegen $a = (1,2,1)$ und $b = (2,1,3)$, so erfährt Bob, dass er 0 Übereinstimmungen hat.

(a) Wieviele Tripel $a \in F^3$, die Alice zu Beginn des Spiels wählen kann, gibt es?

(b) Nach einigen Runden hat Bob die Zahl der verbleibenden Möglichkeiten (also die Zahl der Tripel a, die noch mit Alices Antworten vereinbar sind) auf x reduziert. Zeigen Sie:

Es ist unmöglich für Bob, so zu spielen dass die Zahl der verbleibenden Möglichkeiten nach der nächsten Runde mit Sicherheit kleiner ist als $\lceil \frac{x-1}{3} \rceil$.

Hinweis: Benutzen Sie u.a., dass es nur ein Tripel gibt, welches drei Übereinstimmungen mit a hat.

(c) Wir nehmen an, dass Bob mit einer festen Strategie spielt. Zeigen Sie, dass es ein Tripel a gibt, so dass Bob frühestens in der fünften Runde drei Übereinstimmungen erreicht.

Aufgabe 8

Es sei ein Graph $G = (V, E)$ mit einer Gewichtsfunktion $w : E \longrightarrow \mathbb{N}$ gegeben, und es sei ein minimaler Spannbaum T von G gegeben. Wir verringern das Gewicht $w(e)$ einer Kante e, die nicht in T enthalten ist, und nennen den so modifizierten Graphen G'. Geben Sie einen Algorithmus an, der ausgehend von T in Laufzeit $\mathcal{O}(|V|)$ einen minimalen Spannbaum von G' findet.

Aufgabe 9

Gegeben sei eine Folge von n Zahlen $a_1, a_2, \ldots a_n$, wobei $a_i \in \mathbb{Z}$. D.h. es gibt sowohl positive als auch negative Zahlen. Wir wollen Indizes $1 \leq k \leq \ell \leq n$ finden, so dass die Summe $\sum_{i=k}^{\ell} a_i$ maximiert wird.

Verwenden Sie das Konzept der Dynamischen Programmierung, um einen Algorithmus zu entwerfen, der zunächst den Wert einer solchen Summe berechnet und die Indizes k und ℓ herausfindet. Ihr Algorithmus soll Laufzeit $O(n)$ aufweisen.

Aufgabe 10

Es sind n Bücher gegeben. Jedes Buch hat eine Breite $b_i \in \{1, \ldots, 10\}$ ($1 \leq i \leq n$) und alle Bücher haben identische Höhe. Wir möchten nun die n Bücher so auf zwei Regale R_1 und R_2 verteilen, dass die Bücher in beiden Regalen möglichst den gleichen Platz einnehmen. Dazu müssen wir $\{1, \ldots, n\}$ so auf R_1 und R_2 verteilen, dass

$$\left| \sum_{i \in R_1} b_i - \sum_{i \in R_2} b_i \right|$$

minimiert wird.

Beschreiben Sie einen Algorithmus, der möglichst effizient eine solche Aufteilung findet. Analysieren Sie die Laufzeit ihres Algorithmus und beweisen Sie die Korrektheit.

Aufgabe 11

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 12

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 13

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 14

Wir wollen für Fibonacci-Heaps eine weitere Funktion hinzufügen: CHANGE-KEY(H, x, k) soll den Schlüssel von Knoten x auf den Wert k ändern. Geben Sie einen effizienten Algorithmus für diese Operation an. Die worst-case-Laufzeit von Ihrem Algorithmus ist womöglich hoch. Analysieren Sie deshalb nicht diese Laufzeit, sondern die amortisierten Kosten Ihrer Operationen bezüglich dem Potential, das wir in der Vorlesung definiert haben.

Aufgabe 15

Alice und Bob unterhalten sich über dynamische Programmierung (DP). Bob behauptet, er habe mit DP einen Algorithmus A entworfen, der für jeden Graphen $G = (V, E)$ einen längsten Pfad (d.h. einen Pfad maximaler Länge) in G in Zeit $O(|V|^2 + |E|^2)$ berechnet. Er will seinen Algorithmus jedoch nicht verraten, und Alice ist skeptisch, ob sie Bob glauben soll. Helfen Sie Alice durch die folgenden Punkte, zu einer solideren Einschätzung zu kommen.

(a) Gegeben sei ein Graph $G = (V, E)$ und eine Kante $e \in E$. Konstruieren Sie einen Graphen $G' = (V', E')$ mit der folgenden Eigenschaft: Es gibt genau dann einen Hamiltonkreis in G, der die Kante e benutzt, wenn der längste Pfad im Graphen G' die Länge $|V'| - 1$ hat.

Hinweis 1: Konstruieren Sie G', indem Sie in geschickter Weise Knoten zu G hinzufügen.
Hinweis 2: Beachten Sie, dass ein Pfad der Länge k aus $k + 1$ Knoten und k Kanten besteht.

(b) Nehmen Sie an, dass Bobs Algorithmus korrekt ist, und dass sie ihn als Subroutine aufrufen dürfen. Wie können Sie für einen Graph $G = (V, E)$ und eine Kante $e \in E$ effizient entscheiden, ob es in G einen Hamiltonkreis gibt, der durch e geht?

(c) Unter der Annahme aus (b), wie können Sie entscheiden, ob es in G einen *beliebigen* Hamiltonkreis gibt? Die Laufzeit Ihres Algorithmus sollte höchstens $O(|V|^3 + |E|^3)$ sein.

(d) Was besagt (c) für die Plausibilität von Bobs Behauptung? Ist Alice zurecht skeptisch? Kann sie absolut sicher sein, dass Bob falsch liegt?

Aufgabe 16

Eine boolsche Formel F in disjunktiver Normalform (DNF) ist von der Bauart

$$F = D_1 \vee D_2 \vee \cdots \vee D_m,$$

wobei jede der m Klauseln eine Konjunktion $D_i = y_{i_1} \wedge y_{i_2} \wedge \cdots \wedge y_{i_k}$ von höchstens k Literalen ist. Wir nehmen an dass es total n Variablen gibt. Zeigen Sie, dass man in *linearer* Zeit in der Eingabegrösse entscheiden kann, ob eine gegebene boolsche Formel F in disjunktiver Normalform erfüllbar ist.

Aufgabe 17

Zeigen Sie, dass 3-SAT auch dann noch $\mathcal{NP}$-vollständig bleibt, wenn wir die Menge der möglichen Eingabe-Formeln dahingehend restringieren, dass jede Variable höchstens dreimal und jedes Literal höchstens zweimal in der Formel vorkommen darf.

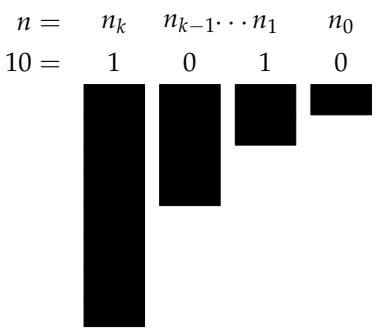


Abbildung 1: Beispiel-Datenstruktur

Hinweis: Nehmen Sie an, dass die Variable x k-mal in der Formel vorkommt, und ersetzen Sie jedes Vorkommen von x durch eine neue Variable $x_1, \ldots, x_k$. Erzwingen sie dann die Äquivalenz der beiden Formeln durch Hinzufügen geeigneter Klauseln.