Universität Bonn, Institut für Informatik I
MUSTERLÖSUNG Klausur Informatik IV SS 2007
18.7.2007, 90 Minuten

NAME	VORNAME	Matrikelnummer
MUSTER-	LÖSUNG	

Nachstehende Tabelle bitte NICHT ausfüllen!

1	2	3	4	5	6	7	8	Summe	Note
10	10	10	10	10	10	10	10	80	

Aufgabe 1 [Multiple Choice, 10 Punkte]

(pro Teilaufgabe 2 Punkte) Kreuzen Sie alle zutreffenden Aussagen an! Es können zu einem Unterpunkt mehrere Aussagen, aber auch keine einzige Aussage richtig sein.

a) Für welche der folgenden Probleme wurde in der Vorlesung ein Sweep-Algorithmus vorgestellt?

- × Berechnung aller Schnittpunkte von n Liniensegmenten
- ☐ Traveling-Salesman-Problem
- × Finden einer maximalen Teilsumme in einer Zahlenfolge
- ☐ Planaritätstest von Graphen

b) Für die zum regulären Ausdruck $\alpha = (a|aa|aaa)^*bb(bb)^*((a|aaaa)aa)^*$ gehörige Sprache gilt ...

- × $L(\alpha) = \{a^k b^l a^m \mid k \in \mathbb{N}_0, l = 2n, n \in \mathbb{N}_{>0}, m \in \mathbb{N}_0, m \text{ durch 3 teilbar}\}$
- ☐ $L(\alpha) = \{a^k b^l a^m \mid k \in \mathbb{N}_0, l = 2(k+1), m = 3k\}$
- × $L(\alpha) = L(\ (a)^*bb(\epsilon|bb)^*(aaa)^*\ )$
- × L ist kontextfrei.

c) Die Frage, ob eine gegebene natürliche Zahl mit m Stellen eine Primzahl ist, ...

- ☐ ist unentscheidbar.
- ☐ ist bewiesenermaßen NP-hart.
- × lässt sich in polynomieller Zeit in m randomisiert mit hoher Wahrscheinlichkeit richtig beantworten.
- × lässt sich in polynomieller Zeit in m deterministisch beantworten.

d) Welche der folgenden in der Vorlesung besprochenen Algorithmen verwenden das Prinzip der dynamischen Programmierung?

- ☐ Quicksort
- × Floyd-Warshall-Algorithmus
- × Algorithmus zur exakten Lösung des Traveling-Salesman-Problems.
- ☐ Berechnung aller Schnittpunkte von n Liniensegmenten.

e) Quicksort ...

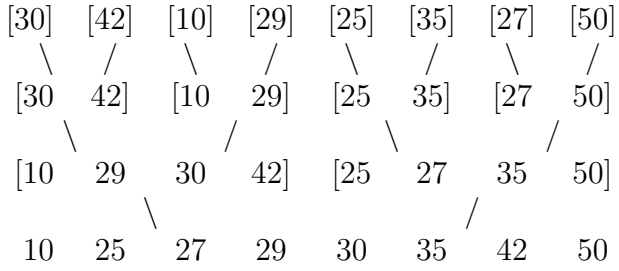
- × ist ein Las-Vegas-Algorithmus.
- × hat im Worst-Case eine Laufzeit in $\Omega(n^2)$.
- × hat im Erwartungswert eine Laufzeit in $O(n \log n)$.
- ☐ ist ein Monte-Carlo-Algorithmus.

Aufgabe 2 [Mergesort und Quicksort, 10 Punkte]

30 42 10 29 25 35 27 50

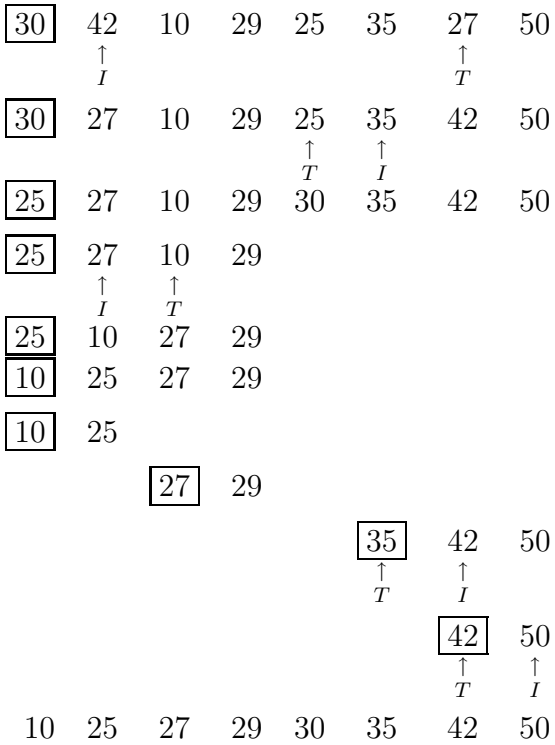
a) Sortieren Sie die oben angegebene Zahlenfolge mit dem Mergesort, der in der Vorlesung vorgestellt wurde.
Kennzeichnen Sie in jedem Schritt, welche Teilfolgen gemischt werden.

Antwort:



b) Sortieren Sie die oben angegebene Zahlenfolge mit dem Quicksort, der in der Vorlesung vorgestellt wurde.
Für eine Folge von Zahlen werde stets das **erste** Element dieser Folge als Pivotelement ausgewählt. Geben Sie in jedem Schritt die Indices I und T sowie das Pivotelement an, kennzeichnen Sie außerdem, welche Teilfolge gerade bearbeitet wird.

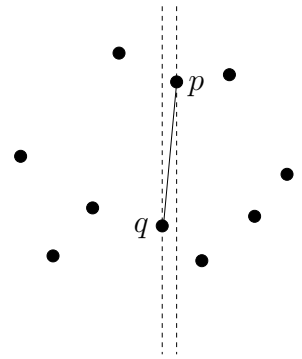
Antwort:



Aufgabe 3 [Divide-and-Conquer, 10 Punkte]

Gegeben sei eine Menge M von n Punkten in der Ebene. Bestimmen Sie mittels eines Divide-and-Conquer Algorithmus in Zeit $O(n \log n)$ ein Punktepaar p und q aus M, dessen Segment $\overline{p\,q}$ die betragsmäßig *größte* Steigung hat.

In der Abbildung hat das Segment $\overline{p\,q}$ der Punkte p und q die betragsmäßig größte Steigung unter allen Segmenten.



Beschreiben Sie die folgenden Elemente Ihres Algorithmus:

a) Divide-Schritt

b) Merge-Schritt

c) Korrektheit des Verfahrens (mit Begründung)

d) Laufzeitabschätzung (mit Begründung und Herleitung durch Rekursionsformel für $n = 2^k$)

Antwort:

a) Zunächst **sortieren** wir die Punktmenge einmalig **nach X-Koordinaten**. Falls zwei Punkte mit **gleichen X-Koordinaten** gefunden werden, geben diese die betragsmäßig maximale Steigung ∞ an und wir sind **fertig**.

   Sonst **splitten** wir in jedem Divide-Schritt **nach X-Ordnung** die zu behandelnde Punktmenge A in **zwei etwa gleichgroße Mengen (Abweichung ± 1) A_L und A_R**. Die Menge A_L liegt vollständig links von A_R.

b) Im Merge-Schritt berechnen wir zunächst **rekursiv jeweils die betragsmäßig größte Steigung in A_L und in A_R** und die dazugehörigen Punktpaare. Danach betrachten wir noch die betragsmäßige **Steigung des Segmentes des rechtesten Punktes p_{L_r} von A_L mit dem linkesten Punkt p_{R_l} von A_R**. Die drei Werte werden verglichen, das **Maximum der drei Werte** und das dazugehörige Punktepaar wird als Ergebnis zurückgegeben.

   Falls die Menge nur noch aus einem Punkt besteht werden ein negativer Wert und zwei Dummy-Punkte zurückgegeben.

c) Besteht die Menge vor dem **letzten Divide-Schritt** aus **zwei Punkten**, dann ist das Ergebnis die betragsmäßige Steigung dieses Paares und die wird vom Algorithmus **richtig ermittelt** (größer als negativer Wert).

Die betragsmäßig maximale Steigung einer **Menge** $A = A_L \cup A_R$ wird **zwischen zwei Punkten angenommen**, **die nach X-Ordnung benachbart sind**. **Beweis durch Widerspruch**: Angenommen für zwei Punkte $p = (p_x, p_y)$ und $q = (q_x, q_y)$ mit betragsmäßig maximaler Steigung läge im vertikalen Streifen zwischen den Geraden $X = p_x$ und $X = q_x$ noch ein weiterer Punkt r. Dann wird eine betragsmäßig größere Steigung entweder zwischen r und q oder zwischen r und p angenommen. Ein Widerspruch!

Deshalb genügt es im **Merge-Schritt** neben den **Werten von A_L und A_R**, die aus der Rekursion stammen, noch den **Übergang dieser Mengen** zu überprüfen.

Insgesamt wird in jedem Merge-Schritt das richtige Ergebnis geliefert, das Verfahren ist somit korrekt.

d) Das **Sortieren der Menge** kostet einmalig $O(n \log n)$. Für den **Merge-Schritt** müssen wir die **sortierte Menge A durchlaufen** und in zwei Hälften teilen, das kostet $O(|A|)$ Aufwand. Da wir die Menge stets in zwei in etwa gleichgroße Teilmengen aufteilen ergibt sich folgende **Rekursionsformel für $n = 2^k$ Punkte**:

$$\begin{aligned} T(n) &\leq 2T\left(\frac{n}{2}\right) + C \cdot n \\ &\leq 2\left(2T\left(\frac{n}{4}\right) + C \cdot \frac{n}{2}\right) + C \cdot n \\ &\vdots \quad \vdots \\ &\leq 2^k T(1) + C \sum_{i=1}^{k} n \in O(n \log n) \end{aligned}$$

Aufgabe 4 [Kontextfreie Sprachen, 10 Punkte]

Sei $G = (V, \Sigma, P, A)$ die kontextfreie Grammatik mit folgenden Produktionen:

$$\begin{aligned} A &\longrightarrow aBb \\ B &\longrightarrow CBD \mid ba \mid \varepsilon \\ C &\longrightarrow CC \mid b \mid \varepsilon \\ D &\longrightarrow aD \mid \varepsilon \end{aligned}$$

a) Geben Sie die Menge der Variablen N und die Menge der Terminalsymbole Σ an.

b) Geben Sie zwei Worte an, die in $L(G)$ liegen.

c) Geben Sie zwei Worte an, die *nicht* in $L(G)$ liegen.

d) Welche der folgenden Aussagen sind wahr, welche falsch?

	Wahr	Falsch
$B \underset{G}{\Longrightarrow} bbaa$		×
$B \overset{*}{\underset{G}{\Longrightarrow}} bbaa$	×	
$B \overset{*}{\underset{G}{\Longrightarrow}} bbaaaa$	×	
$B \overset{*}{\underset{G}{\Longrightarrow}} abab$		×

e) Beschreiben Sie informell, welche Wörter in L liegen.

f) Ist die Sprache $L(G)$ auch regulär? Begründen Sie Ihre Antwort!

Antwort:

a) $N = \{ A, B, C, D \}$, $\Sigma = \{ a, b \}$

b) z.B. ab, abab, abbaaab

c) z.B. a, b, aa

d) s.o.

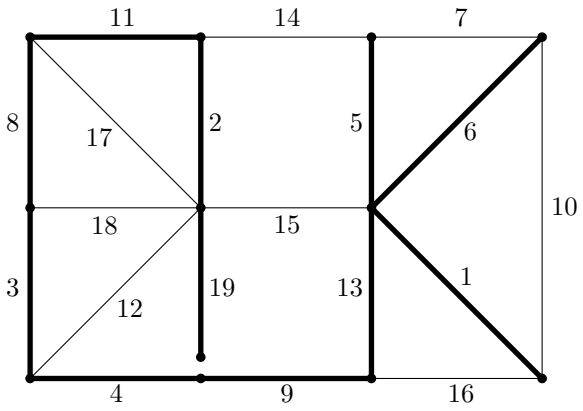
e) Wörter, die mit a beginnen, gefolgt von einer (evtl. leeren) Sequenz b's, einer (evt. leeren) Sequenz von a's und einem abschließendem b.

f) Ja, sie läßt sich durch den regulären Ausdruck ab^*a^*b angeben.

Aufgabe 5 [Kruskal-Verfahren, 10 Punkte]

a) Berechnen Sie mit dem Kruskal-Algorithmus einen minimalen Spannbaum des folgenden Graphen! Geben Sie nur an, in welcher Reihenfolge die Kanten betrachtet werden, und zeichnen Sie den erhaltenen minimalen Spannbaum in der Abbildung ein! (Die Zahlen an den Kanten sind die Gewichte, die Sie hier aber auch als eindeutige Kantenbezeichner verwenden sollen.)

Antwort: Die Kanten werden natürlich in der Reihenfolge aufsteigenden Gewichtes betrachtet, also $1, 2, 3, \ldots, 19$. Der minimale Spannbaum sieht dann so aus:



b) Beweisen oder widerlegen Sie folgende Aussage: Wenn ein beliebiger, zusammenhängender, ungerichteter, schlichter Graph G mit echt positiven Kantengewichten (> 0) genau eine Kante mit minimalem Gewicht hat, dann ist diese Kante in jedem minimalen Spannbaum von G enthalten.

Antwort: Widerspruchsbeweis: Annahme, für einen solchen Graphen $G = (V, E)$ gäbe es einen minimalen Spannbaum T mit Kantenmenge $E_T \subseteq E$ ohne die eindeutige Kante $e_{\min} = (v, w)$ minimalen Gewichtes.

Fügen wir $e_{\min}$ zu E_T hinzu, muss ein Kreis entstehen, denn sonst wäre T kein Spannbaum von G. Der Kreis enthält $e_{\min}$ und mindestens noch eine weitere Kante e. Für die Kantengewichte gilt $|e_{\min}| < |e|$.

Durch Löschen von e bleibt der Graph $\tilde{T} = (V, E_T \cup \{e_{\min}\} \setminus \{e\})$ wegen der restlichen Kanten im Kreis zusammenhängend. Er ist kreisfrei, weil es in T nur einen Weg von v nach w gab. Es handelt sich also um einen Spannbaum und das Gesamtgewicht hat um $|e| - |e_{\min}| > 0$ abgenommen.

Aufgabe 6 [Randomisierte Algorithmen, 10 Punkte]

Es werde nun folgendes einfache randomisierte Sortierverfahren betrachtet. Der Algorithmus wählt zufällig und gleichverteilt eine Permutation der gegebenen, paarweise verschiedenen n Zahlen und gibt diese aus.

a) Wie groß ist die Wahrscheinlichkeit, dass der Algorithmus ein falsches Ergebnis liefert? Wie oft muss man ihn unabhängig voneinander mindestens wiederholen, damit mindestens mit Wahrscheinlichkeit (1/2) mindestens eines der gesammelten Ergebnisse korrekt ist?

Antwort: Da der Algorithmus gleichverteilt eine von $n!$ Permutationen wählt, von denen genau eine das richtige Ergebnis ist, weil die Eingabewerte paarweise verschieden sind, ist das Ergebnis nur mit einer Wahrscheinlichkeit von $1/n!$ korrekt. Die Wahrscheinlichkeit für das Scheitern des Algorithmus liegt also bei $1 - 1/n! = (n! - 1)/n!$.

Die Wahrscheinlichkeit, dass bei k Versuchen nicht die richtige Reihenfolge ausgegeben wurde, liegt bei $(1 - 1/n!)^k$. Damit diese Wahrscheinlichkeit höchstens (1/2) ist, muss also

$$\left(1 - \frac{1}{n!}\right)^k \leq \frac{1}{2}$$

gelten, was äquivalent ist zu

$$k \geq \log_{(1-1/n!)} \left(\frac{1}{2}\right) = \frac{\ln\left(\frac{1}{2}\right)}{\ln\left(\frac{n!-1}{n!}\right)} = \frac{\ln\left(\frac{1}{2}\right)}{\ln(n! - 1) - \ln(n!)} = \frac{\ln 2}{\ln(n!) - \ln(n! - 1)}.$$

b) Konstruieren Sie mit dem Prinzip aus a) einen genauso primitiven Las-Vegas-Algorithmus zum Sortieren von n Zahlen! Was ist seine Laufzeit im schlimmsten Fall und im besten Fall? Ist er in einer dieser Laufzeiten besser als Merge-Sort?

Antwort: Im letzten Teil von a) steckt schon die Idee eines solchen Las-Vegas-Algorithmus. Man wiederholt die zufällige Wahl der Reihenfolge und testet jedes Mal, ob die erhaltene Ausgabe eine korrekt sortierte Folge ist. Dies geht durch einen linearen Durchlauf in einer Laufzeit von $\Theta(n)$. Man wiederholt das ganze so lange, bis das Ergebnis tatsächlich korrekt ist. Im Worst-Case kann der Las-Vegas-Algorithmus beliebig lange immer wieder falsche Permutationen anwenden. Es gibt also keine obere Laufzeitschranke für den Worst-Case. Hier ist Merge-Sort mit einer Worst-Case-Laufzeit in $O(n \log n)$ deutlich besser.

Im besten Fall wird beim Las-Vegas-Algorithmus sofort am Anfang die richtige Reihenfolge geraten. Durch den anschließenden Test liegt die Best-Case-Laufzeit in $\Theta(n)$. Offenbar ist bei dieser Laufzeit das skurrile Verfahren besser als Merge-Sort, da Merge-Sort auch im besten Fall eine Laufzeit in $\Omega(n \log n)$ hat.

Aufgabe 7 [Automaten, 10 Punkte]

Gegeben sei die Sprache $L \subset \{a, b\}^*$ aller Wörter, die den String $abab$ enthalten.

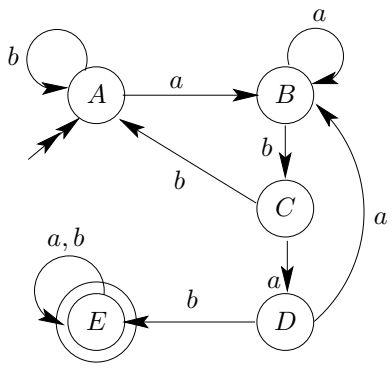
a) Geben Sie einen **minimalen** DEA an, der diese Sprache akzeptiert. D.h. geben Sie die Zustandsübergangsfunktion δ als Zustandsdiagramm an und geben Sie das Tupel an, das den Automaten beschreibt.

b) Geben Sie einen regulären Ausdruck für die Sprache L an.

c) Geben Sie die Definition der in der Vorlesung verwendeten Äquivalenzrelation $\sim_L$ über $\{a, b\}^*$ an.

d) Begründen Sie, dass Ihr Automat aus a) minimal ist. Verwenden Sie dabei die Relation $\sim_L$ aus c) und eine Aussage bezüglich $\sim_L$ aus der Vorlesung.

Antwort:



a)

$(Q, \Sigma, \delta, q_0, F)$ mit

$$Q = \{A, B, C, D, E\}$$
$$\Sigma = \{a, b\}$$
$$F = \{E\}$$
$$q_0 = A$$

b) $(a|b)^*abab(a|b)^*$

c)

$$x \sim_L y \quad :\Leftrightarrow (\forall z \in \{a, b\}^* : xz \in L \Leftrightarrow yz \in L)$$

d) In der Vorlesung wurde gezeigt, dass die maximale Anzahl an Zuständen eines akzeptierenden Automaten durch die Anzahl der Äquivalenzklassen von $\sim_L$ beschränkt ist. Es gibt fünf verschiedene Äquivalenzklassen, nämlich: $[\epsilon]_L, [a]_L, [ab]_L, [aba]_L$ und $[abab]_L$. Also werden mindestens fünf Zustände benötigt, die hat unser Automat.

Aufgabe 8 [Greedy Algorithmen, 10 Punkte]

Wir wollen Zeilenumbrüche berechnen. Gegeben sei eine Sequenz von n Wörtern. Die Länge des i. Wortes sei w_i. Wir wollen daraus eine Sequenz von Zeilen berechnen (die Reihenfolge der Wörter darf dabei nicht verändert werden, Wörter werden nicht getrennt!). Die Länge K_j der j. Zeile ist die Anzahl der darin enthaltenen Zeichen (zur Vereinfachung lassen wir Leerzeichen zwischen den Wörtern weg). Die maximale Länge einer Zeile sei L, also eine Zeile darf nie mehr als L Zeichen enthalten, weniger ist erlaubt. Die Kosten der j. Zeile mit K_j Zeichen betragen $L - K_j$.

Betrachten Sie folgenden Greedy-Algorithmus:

FOR $i := 1$ **TO** n **DO**
  IF i. Wort passt noch in die aktuelle Zeile
    THEN plaziere das i. Wort in der aktuellen Zeile
    ELSE plaziere das i. Wort in einer neuen Zeile
END FOR

a) Zeigen Sie: der Greedy-Algorithmus ist *nicht* optimal, wenn die Kosten des Algorithmus durch das *Maximum* der Zeilenkosten bestimmt werden.

Antwort:

Um dies zu zeigen genügt es, ein Beispiel anzugeben, bei dem der Greedy-Algorithmus nicht die optimale Lösung berechnet.

Sei $L = 5$ und die Worte AAA BB CC DDDD gegeben. Der Greedy-Algorithmus liefert:
AAABB
CC
DDDD
mit Gesamtkosten $\max\{0, 3, 1\} = 3$. Optimal wäre
AAA
BBCC
DDDD
mit Kosten $\max\{2, 1, 1\} = 2$.

b) Zeigen Sie: der Greedy-Algorithmus ist optimal, wenn die Kosten des Algorithmus durch die *Summe* der Zeilenkosten bestimmt werden.

(Tip: Konstruieren Sie aus einer beliebigen optimalen Lösung eine Lösung mit gleichen Kosten, die der Ausgabe des Greedy-Algorithmus entspricht.)

Antwort:

Wir zeigen dies durch Widerspruch. Angenommen, es gäbe eine optimale Lösung OPT, die sich von der Lösung G unseres Algorithmus unterscheidet. Seien OPT_j und G_j die Kosten der j. Zeile im Ergebnis von OPT bzw. Greedy und sei ℓ der Index der ersten Zeile, in der OPT und Greedy unterschiedliche Lösungen liefern. Da sich unser Algorithmus "greedy" verhält, gilt $G_\ell < OPT_\ell$.

Nun läßt sich aus OPT eine optimale Lösung OPT' konstruieren, indem das erste Wort aus Zeile $\ell + 1$ in Zeile ℓ verschoben wird. Sei w die Länge dieses Wortes, dann betragen die Kosten der Zeile ℓ in OPT': $OPT'_\ell = OPT_\ell - w$ und $OPT'_{\ell+1} = OPT_{\ell+1} + w$. Dies läßt sich für alle weiteren Zeilen fortführen. Die Kosten von OPT' und OPT sind also gleich, ebenso sind die Ausgaben OPT' und G gleich, also muß G eine optimale Lösung sein.