Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Volker Heun

Wintersemester 2023/24
Semestralklausur
28. März 2024

Algorithmen auf Sequenzen

Vorname	Name	Matrikelnummer

Reihe	Platz	Unterschrift

Allgemeine Hinweise zur Semestralklausur

- Vor der Prüfung ist diese Seite mit Vornamen, Namen, Matrikelnummer, Reihe und Platz **leserlich mit Druckbuchstaben** zu versehen und zu unterschreiben.
- Bitte **nicht** in *roter* oder *grüner* Farbe bzw. **nicht** mit Bleistift schreiben.
- Der Studentenausweis und ein amtlicher Lichtbildausweis sind bereit zu halten.
- Die reine Bearbeitungszeit beträgt **120 Minuten**.
- Es sind insgesamt 40 Punkte zu erreichen, zum Bestehen sind 17 Punkte nötig.

Viel Erfolg!

Hörsaal verlassen von ..... bis ..... von ..... bis .....

Vorzeitig abgegeben um .....

	Hz	A1	A2	A3	A4	A5	$\sum$
Erstkorrektur							
Nachkorrektur							
Zweitprüfer							

Vorname: _____ Name: _____ Matrikelnummer: _____

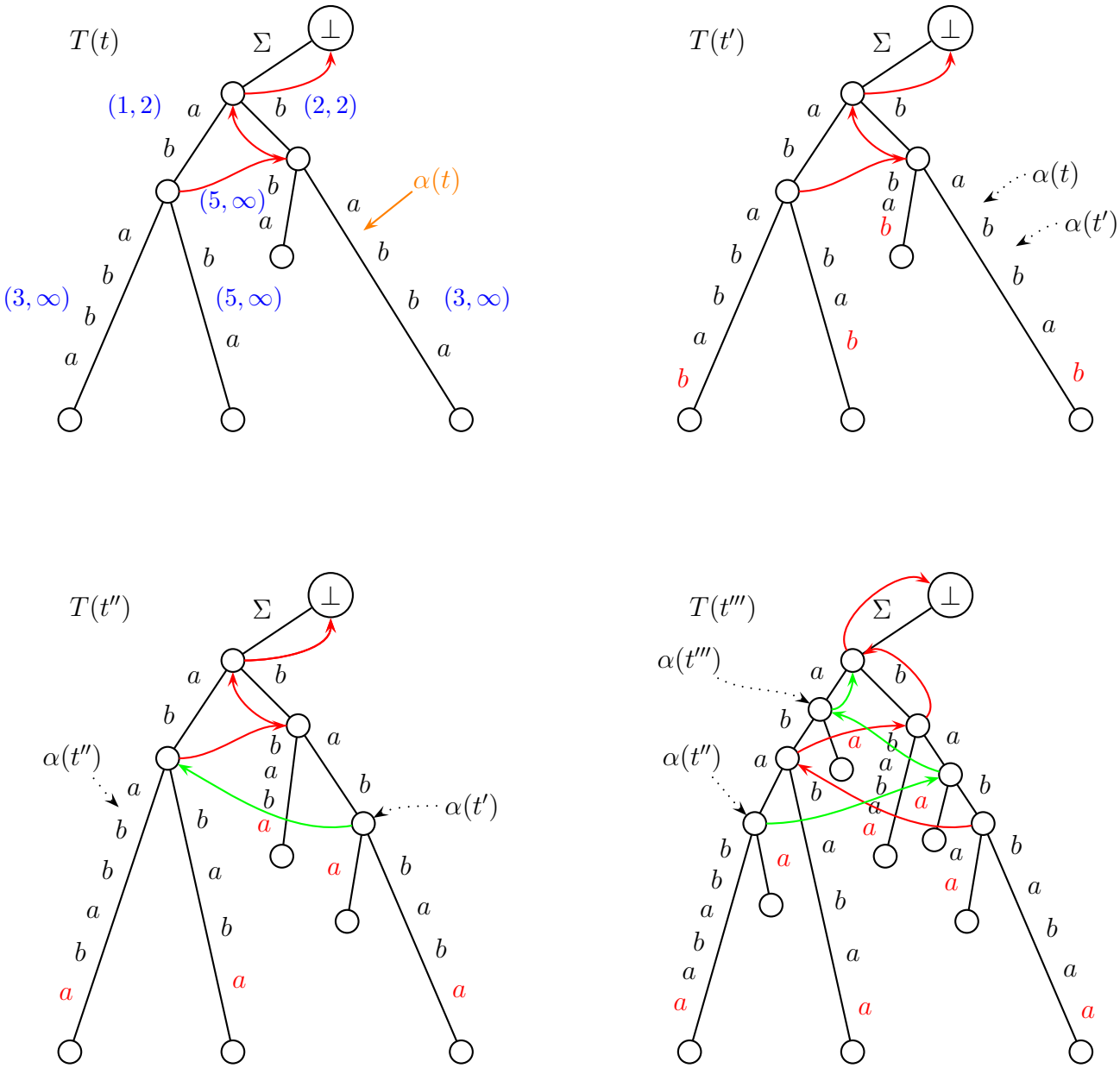
Aufgabe 1 (8 Punkte)

Zeichne im unteren Suffix-Baum $T(t)$ für $t = t_1 \cdots t_6 = ababba$ die Suffix-Links ein und erweitere diesen nach Ukkonens Algorithmus für $t' = ababba\underline{b}$, $t'' = ababba\underline{ba}$ und $t''' = ababba\underline{baa}$. Ergänze hierzu die unten angegebenen Suffix-Bäume.

Es sind auch jeweils die neuen Suffix-Links und die Position des aktiven Suffixes sowohl vor als auch nach Ukkonens Erweiterungsschritt einzuzeichnen.

Gib für den Baum $T(t)$ (oben links) die Kantenmarkierungen an, die bei einer echten Implementierung hierfür verwendet werden.

Lösungsskizze (nicht ausreichend für die volle Punktzahl)



Vorname: _____ Name: _____ Matrikelnummer: _____

Aufgabe 2 (8 Punkte)

Betrachte das Wort $t\$ = t_1 \cdots t_{10}\$ = \text{ANANASANNA}\$$.

a) Konstruiere die Burrows-Wheeler-Transformierte $\hat{t}$ zu $t\$$.

b) Gib die zugehörige LF-Funktion für $\hat{t}$ an.

c) Bestimme die Werte $C(\cdot)$ und $Occ(\cdot, \cdot)$.

d) Suche nach $s = s_1 \cdots s_3 = \text{NNA}$ im FM-Index für t mit Hilfe des in der Vorlesung angegebenen Algorithmus unter Verwendung von C und Occ.

Hinweise: Für Teil a) und b) fülle die unten angegebene Tabelle korrekt aus. In dieser Aufgabe gilt: $\$ < A < N < S$.

Lösungsskizze (nicht ausreichend für die volle Punktzahl)

i	$A[i]$	$t^{A[i]}$	$\hat{t}_i$	LF[i]
0	11	\$ANANASANNA	A	1
1	10	A\$ANANASANN	N	6
2	1	ANANASANNA\$	\$	0
3	3	ANASANNA\$AN	N	7
4	7	ANNA\$ANANAS	S	10
5	5	ASANNA\$ANAN	N	8
6	9	NA\$ANANASAN	N	9
7	2	NANASANNA\$A	A	2
8	4	NASANNA\$ANA	A	3
9	8	NNA\$ANANASA	A	4
10	6	SANNA\$ANANA	A	5

$Occ(\cdot, \cdot)$	\$	A	N	S
0	0	1	0	0
1	0	1	1	0
2	1	1	1	0
3	1	1	2	0
4	1	1	2	1
5	1	1	3	1
6	1	1	4	1
7	1	2	4	1
8	1	3	4	1
9	1	4	4	1
10	1	5	4	1

$C(\cdot)$	i
\$	0
A	1
N	6
S	10

$[\ell, r] = [0 : 10]$, $i = 3$, $s_i = \text{A}$
$\ell' = C(A) + Occ(A, 0 - 1) = 1 + 0 = 1$
$r' = C(A) + Occ(A, 10) - 1 = 1 + 5 - 1 = 5$

$[\ell, r] = [1 : 5]$, $i = 2$, $s_i = \text{N}$
$\ell' = C(N) + Occ(N, 1 - 1) = 6 + 0 = 6$
$r' = C(N) + Occ(N, 5) - 1 = 6 + 3 - 1 = 8$

$[\ell, r] = [6 : 8]$, $i = 1$, $s_i = \text{N}$
$\ell' = C(N) + Occ(N, 6 - 1) = 6 + 3 = 9$
$r' = C(N) + Occ(N, 8) - 1 = 6 + 4 - 1 = 9$

$[\ell, r] = [9, 9]$.

Vorname: _____ Name: _____ Matrikelnummer: _____

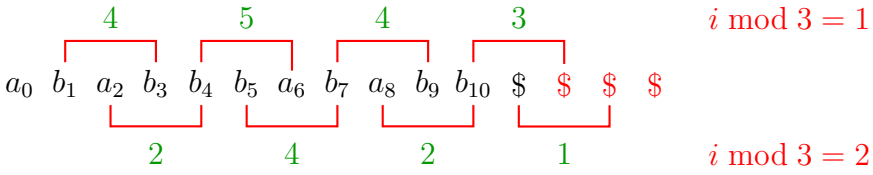
Aufgabe 3 (8 Punkte)

Erstelle für das Wort $t\$ = t_0 \cdots t_{10}\$ = ababbbababb\$$ ein Suffix-Array nach dem Algorithmus von Kärkkäinen und Sanders. Gib dabei alle Zwischenschritte an, wobei der rekursive Aufruf von Hand sortiert werden darf.

Hinweis: Gib beim Mischen von A_0 mit A_{12} für jede der Ergebnispositionen in A an, ob 1 oder 2 Zeichenvergleiche erforderlich waren oder ob auf die Ordnung von A_{12} zurückgegriffen wurde.

Lösungsskizze (nicht ausreichend für die volle Punktzahl)

Zuerst konstruieren wir alle Tripel, die an Position $i \bmod 3 \neq 0$ beginnen, und sortieren diese



Wir erhalten nun

$$\begin{aligned} t^{(1)} &= 4 \cdot 5 \cdot 4 \cdot 3 \\ t^{(2)} &= 2 \cdot 4 \cdot 2 \cdot 1 \\ \tilde{t} &= 4 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 4 \cdot 2 \cdot 1 \\ \tilde{t} \cdot 0 &= 4 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 4 \cdot 2 \cdot 1 \cdot 0 \end{aligned}$$

Wir erhalten nach dem rekursiven Sortieren der Suffixe von $\tilde{t} \cdot 0 = 4 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 4 \cdot 2 \cdot 1 \cdot 0$:

$$\begin{aligned} A'[0] &= 8 \quad \hat{=}\, 0 && \hat{=}\, \epsilon \\ A'[1] &= 7 \quad \hat{=}\, 10 && \hat{=}\, \$\$\$ \\ A'[2] &= 6 \quad \hat{=}\, 210 && \hat{=}\, abb\,\$\$ \\ A'[3] &= 4 \quad \hat{=}\, 24210 && \hat{=}\, abb\,bab\,abb\,\$\$ \\ A'[4] &= 3 \quad \hat{=}\, 324210 && \hat{=}\, b\$\$\cdots \\ A'[5] &= 5 \quad \hat{=}\, 4210 && \hat{=}\, bab\,abb\,\$\$\$ \\ A'[6] &= 2 \quad \hat{=}\, 4324210 && \hat{=}\, bab\,b\$\$\cdots \\ A'[7] &= 0 \quad \hat{=}\, 454324310 && \hat{=}\, bab\,bba\,bab\,b\$\$\cdots \\ A'[8] &= 1 \quad \hat{=}\, 54324210 && \hat{=}\, bba\,bab\,b\$\$\cdots \end{aligned}$$

Damit erhalten wir A_{12}:

$$\begin{aligned} A_{12}[0] &= 2 + 3 \cdot (\mathbf{7} - 4) &&= 11 && \hat{=}\, \$ \\ A_{12}[1] &= 2 + 3 \cdot (\mathbf{6} - 4) &&= 8 && \hat{=}\, abbb\$ \\ A_{12}[2] &= 2 + 3 \cdot (\mathbf{4} - 4) &&= 2 && \hat{=}\, abbbababb\$ \\ A_{12}[3] &= 1 + 3 \cdot (\mathbf{3}) &&= 10 && \hat{=}\, b\$ \\ A_{12}[4] &= 2 + 3 \cdot (\mathbf{5} - 4) &&= 5 && \hat{=}\, bababb\$ \\ A_{12}[5] &= 1 + 3 \cdot (\mathbf{2}) &&= 7 && \hat{=}\, babb\$ \\ A_{12}[6] &= 1 + 3 \cdot (\mathbf{0}) &&= 1 && \hat{=}\, babbababb\$ \\ A_{12}[7] &= 1 + 3 \cdot (\mathbf{1}) &&= 4 && \hat{=}\, bbababbb\$ \end{aligned}$$

Vorname: _____ Name: _____ Matrikelnummer: _____

Nun bestimmen wir A_0:

A_0	0	1	2	3
	0	6	3	9
11				
8				
2				
10			9	
5				
7	6			
1	6	0		
4			9	3
	6	0	9	3
	a	a	b	b
	b	b	b	b
	a	b	a	\$

Nun müssen wir noch A_0 und A_{12} mischen (hierbei sind in Blau bzw. Rot die Werte mit $i \bmod 3 = 1$ bzw. $i \bmod 3 = 2$) :

A_0	0	1	2	3
	6	0	9	3
	a	a	b	b
	b	b	b	b

A_{12}	0	1	2	3	4	5	6	7
	11	8	2	10	5	7	1	4
	\$	a	a	b	b	b	b	b
		b	b	\$	a	a	a	b

A	0	1	2	3	4	5	6	7	8	9	10	11
	11^1	6^*	0^*	8^1	2^1	10^*	5^2	7^*	1^*	9^*	4^*	3

Bei den Vergleichen gibt $\cdot^1$ bzw. $\cdot^2$ an, dass ein Vergleich des ersten bzw. zweiten Zeichens der entsprechenden Suffixe zur Anordnung beim Mischen ausreichend war, $\cdot^*$ zeigt an, dass die relative Ordnung der Suffixe in A_{12} erforderlich war. Bei den Werten ohne Zusatzinformation war kein Vergleich mehr erforderlich.

Vorname: _____ Name: _____ Matrikelnummer: _____

Aufgabe 4 (8 Punkte)

Betrachte die unorientierte Permutation $\pi = (3, 4, 6, 5, 7, 8, 1, 2)$.

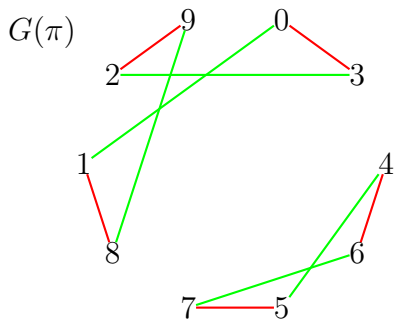
a) Zeichne den Breakpoint-Graphen $G(\pi)$ für π.

b) Gib die schärfere untere Schranke für die benötigte Anzahl von Reversionen zum Sortieren mit Reversionen aus der Vorlesung an und wende diese auf π bzw. den Breakpoint-Graphen $G(\pi)$ aus a) an.

c) Wende den Algorithmus aus der Vorlesung zur 2-Approximation für die minimale Reversal-Distanz auf π an. Gib dabei alle Zwischenschritte an und erläutere, warum eine bestimmte Reversion angewendet wird (bzw. nicht angewendet wird).

Lösungsskizze (nicht ausreichend für die volle Punktzahl)

a) Der Breakpoint-Graph für die erweiterte Permutation $(0, 3, 4, 6, 5, 7, 8, 1, 2, 9)$ von π sieht wie folgt aus:



b) Die untere scharfe Schranke für das Sortieren mit unorientierten Reversionen lautet für den Breakpoint-Graphen aus a):

$$d(\pi) \geq b(\pi) - c(\pi).$$

Für diese Permutation π gilt:

$$\begin{aligned} b(\pi) &= 5 \\ c(\pi) &= 2 \end{aligned}$$

Wie man leicht sieht, besteht der Breakpoint-Graphen $G(\pi)$ aus genau zwei alternierenden Kreise. Somit gilt

$$d(\pi) \geq b(\pi) - c(\pi) = 5 - 2 = 3.$$

Vorname: _____ Name: _____ Matrikelnummer: _____

c) Es ergibt sich der folgende Ablauf des Algorithmus:

0 3 4 6 5 7 8 1 2 9	Kleinstes Element in einem fallendem Strip finden und zugehörigen Breakpoint eliminieren; Ergebnis-Permutation enthält keine fallende Strips, also größtes Element in einem fallendem Strip finden und zugehörigen Breakpoint eliminieren; (liefert dieselbe Reversion)
0 3 4 5 6 7 8 1 2 9	Kein fallender Strip, also alles zwischen ersten und letzten Breakpoint umdrehen
0 2 1 8 7 6 5 4 3 9	Kleinstes Element in einem fallendem Strip finden und zugehörigen Breakpoint eliminieren
0 1 2 8 7 6 5 4 3 9	Kleinstes Element in einem fallendem Strip finden und zugehörigen Breakpoint eliminieren; Ergebnis-Permutation enthält keine fallende Strips, also größtes Element in einem fallendem Strip finden und zugehörigen Breakpoint eliminieren; (liefert dieselbe Reversion)
0 1 2 3 4 5 6 7 8 9	Fertig!

Vorname: _____ Name: _____ Matrikelnummer: _____

Aufgabe 5 (8 Punkte)

Entwirf einen Linearzeit-Algorithmus für das folgende Problem.

Maximal Scoring Even Subsequence (MSES)

Eingabe: Eine Folge $(a_1, \ldots, a_n) \in \mathbb{R}^n$.
Ausgabe: Eine (zusammenhängende) Teilfolge $(a_\ell, \ldots, a_r)$ mit $\ell \leq r \in [1 : n]$, für die $r - \ell + 1$ gerade ist und die $\sigma(\ell, r)$ maximiert, wobei $\sigma(i, j) = \sum_{k=i}^{j} a_k$.

Hinweise: Korrektheitsbeweis und Laufzeitanalyse nicht vergessen.

Lösungsskizze (nicht ausreichend für die volle Punktzahl)

Wir adaptieren die Lösung für das Maximal Scoring Subsequence Problems aus der Vorlesung. Anstatt von *rmax* führen wir zwei Variable *rmax_even* und *rmax_odd* ein, die maximal bewertete Sequenz gerader bzw. ungerader Länge bis zur Position $i \in [1 : n]$ bezeichnet (mit zugehöriger Startposition *rstart_even* bzw. *rstart_odd*, d.h. $i - rstart\_even + 1$ bzw. $i - rstart\_odd + 1$ ist gerade bzw. ungerade). Wenn i inkrementiert wird, müssen diese beide Werte jeweils vertauscht werden (swap), bevor Sie um das Element a_i aktualisiert werden, da sich die Eigenschaft Gerade/Ungerade der zugehörigen Teilfolgen dabei ändert.

Zu Beginn ist für die einelementige Folge (a_1) die leere Folge mit Score 0 die einzige Folge gerader Länge, also $max = 0$, $\ell = 0$, $r = 1$. Die Folge mit maximalem Wert gerader bzw. ungerade Länge, die an Position $i = 1$ endet, ist dann die leere bzw. einelementige Folge, also $rmax\_even := 0$, $rstart\_even := 2$ bzw. $rmax\_odd := a_1$, $rstart\_odd := 1$.

Für die for-Schleife ($i \in [2 : n]$) wird dann jeweils zuerst *rmax_even* und *rmax_odd* vertauscht und dann aktualisiert und mit der kürzesten Folge gerader bzw. ungerader Länge, die an Position i endet verglichen; dies sind die leere Folge und die einelementige Folge a_i.

Für die anschließende Aktualisierung von *max* darf dann natürlich nur eine Folge gerade Länge, die an Position i endet, berücksichtigt werden, also *rmax_even*. Dieser Algorithmus ist im Pseudo-Code auf der folgenden Seite angegeben.

In jedem Schleifendurchlauf ist die Laufzeit konstant. Somit ist die Gesamtlaufzeit linear.

Auch wenn $n = 0$ ist, wird als korrekte Lösung die leere Folge mit $(\ell, r) = (1, 0)$ zurückgegeben, da die for-Schleife nicht durchlaufen wird. Hierbei wird der Einfachheit halber angenommen, dass der Zugriff auf a_1 keinen Laufzeitfehler auslöst.

Es gibt auch einige alternative Lösungen.

Vorname: _____ Name: _____ Matrikelnummer: _____

MSES (real $a[]$, int n)

```
begin
    int rmax_even := 0, rstart_even := 2;
    int rmax_odd := a_1, rstart_odd := 1;
    int max := 0, ℓ := 1, r := 0;

    for (i := 2; i ≤ n; i++) do
        swap(rmax_even, rmax_odd);
        swap(rstart_even, rstart_odd);
        if (rmax_even + a_i > 0) then
            rmax_even := rmax_even + a_i;
        else
            rmax_even := 0;
            rstart_even := i + 1;
        if (rmax_odd + a_i > a_i) then
            rmax_odd := rmax_odd + a_i;
        else
            rmax_odd := a_i;
            rstart_odd := i;

        if (rmax_even > max) then
            max := rmax_even;
            ℓ := rstart_even;
            r := i;
    return (ℓ, r, max);
end
```