Nachname:

Vorname:

Matrikelnummer:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders 11.03.2025

Klausur Algorithmen II

Aufgabe 1.	Kleinaufgaben	13 Punkte
Aufgabe 2.	Shortest Path: Multi-Direktionaler Dijkstra	10 Punkte
Aufgabe 3.	Approximationsalgorithmen: Independent Set	9 Punkte
Aufgabe 4.	Geometrische Algorithmen: Luftballons	9 Punkte
Aufgabe 5.	Stringology: Teilstrings und LZ77	9 Punkte
Aufgabe 6.	Parallele Algorithmen: Akkumulation	10 Punkte

Bitte beachten Sie:

- Als Hilfsmittel ist nur **ein** DIN-A4 Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- Schreiben Sie Ihre Antworten nur in blau oder schwarz, mit dokumentenechtem Stift.
- **Schreiben** Sie auf **alle** Blätter der Klausur und Zusatzblätter Ihre **Matrikelnummer**.
- Die Klausur enthält **17 Seiten**.
- Zum Bestehen der Klausur sind 30 Punkte hinreichend.

Lösungsvorschlag

EK

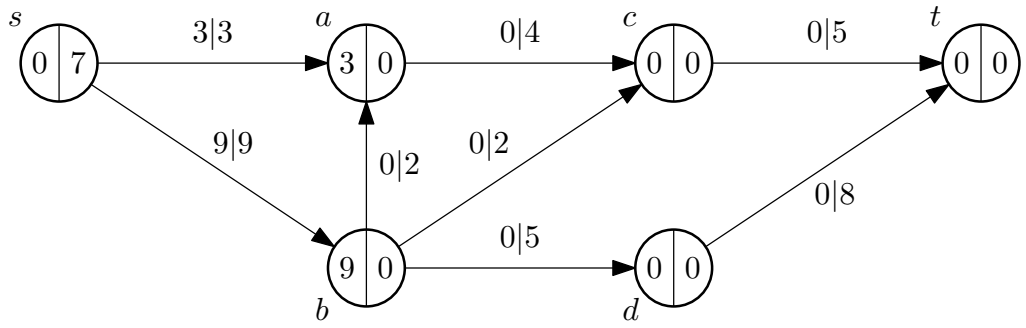
ZK

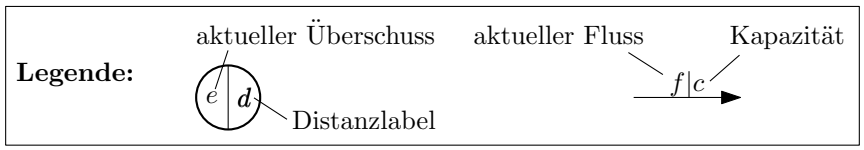
Aufgabe 1. Kleinaufgaben [13 Punkte]

a. Gegeben sei das folgende Flussnetzwerk mit Quelle s und Senke t. Es beschreibt den Zustand des preflow-push Algorithmus nach der Initialisierung. Die Knoten sind mit ihrem aktuellen Überschuss und ihrem Distanzlabel beschriftet. Die Kanten sind mit ihrem aktuellen Fluss und ihrer Kapazität beschriftet.
Berechnen Sie mittels des *generic preflow-push* Algorithmus einen maximalen Fluss f, indem Sie eine Operationsfolge mit folgenden beiden Operationen angeben:

- `push`$((u,v),x)$: schiebe x Fluss über die Kante (u,v).
- `relabel`(v,d): setze Distanzlabel von Knoten v auf d.

Sie dürfen den Algorithmus abbrechen, sobald der Fluss bei t maximal ist. [4 Punkte]





Lösung

1. `relabel`$(b,1)$	6. `relabel`$(a,1)$	11. `relabel`$(c,1)$
2. `push`$((b,d),5)$	7. `push`$((a,c),4)$	12. `push`$((c,t),5)$
3. `push`$((b,c),2)$	8. `relabel`$(d,1)$	13. ...
4. `push`$((b,a),2)$	9. `push`$((d,t),5)$	

b. Für eine Eingabe der Größe n mit Parameter k betrachten wir einen Algorithmus, der tiefenbeschränkte Suche nutzt. Der resultierende Suchbaum habe k Level (exklusive Wurzel) und Verzweigungsgrad d. In jedem Suchschritt müssen zusätzliche Berechnungen mit Laufzeit $\Theta(n)$ ausgeführt werden. Geben Sie für die folgenden Werte von d die asymptotische Laufzeit des Algorithmus an, sowie ob diese Laufzeit das Problem *fixed-parameter tractable* (FPT) in k macht.

1. $d = 2$
2. $d = k$
3. $d = \sqrt{n}$

[3 Punkte]

Lösung

1. Laufzeit $\Theta(2^k n)$, ist FPT
2. Laufzeit $\Theta(k^k n)$, ist FPT
3. Laufzeit $\Theta(n^{\frac{k}{2}+1})$, ist nicht FPT

c. Wir betrachten folgende Variante des Ski Rental Problems: Der Einkaufspreis einer Ski- Ausrüstung liege bei 500 Euro und der Mietpreis bei 50 Euro pro Woche. Der Online-Algorithmus arbeitet wie folgt: Die ersten 5 Ski-Fahr-Wochen wird gemietet. In der 6. Ski-Fahr-Woche wird gekauft. Was ist der kompetitive Faktor dieser Strategie gegenüber einer optimalen Strategie? Begründen Sie Ihre Antwort kurz. [2 Punkte]

Lösung

Der kompetetive Faktor ist 2.5. Begründung: Das Verfahren ist optimal wenn wir $x \leq 5$ mal Ski fahren. Wenn wir 6 mal Ski fahren zahlt der Algorithmus $500 + 5 \times 50 = 750$ Euro während der optimale Algorithmus nur $6 \times 50 = 300$ Euro zahlt – $750/300 = 2.5$. Wenn wir öfter fahren wird das Verhältnis zwischen optimalem und online Algorithmus kleiner.

d. Es seien n Ganzzahlen im Bereich $[1,n]$ gegeben, die nicht in den Hauptspeicher passen. Sortieren Sie die folgenden externen Algorithmen zur Entfernung von Duplikaten aufsteigend nach asymptotischer Anzahl IOs im Worst-Case. Begründen Sie Ihre Lösung, indem Sie für jeden Algorithmus die asymptotische Anzahl IOs im Worst-Case angeben. Gehen Sie davon aus, dass $B \gg \log n$ für die Blockgröße gilt.

1. Scan mit Deduplizierung durch Zugriff auf ein externes Array der Größe n
2. Scan mit Deduplizierung durch Zugriff auf einen externen Suchbaum
3. Externes Sortieren, dann Scan mit Deduplizierung durch Vergleich benachbarter Zahlen

[4 Punkte]

Lösung

- Externes Sortieren ($\mathcal{O}\left(\frac{n}{B}\log_{M/B}\frac{n}{M}\right)$ IOs)
- Array ($\mathcal{O}(n)$ IOs)
- Suchbaum ($\mathcal{O}\left(n\log_B\frac{n}{M}\right)$ IOs)

Dass externes Sortieren schneller als das Array ist, ergibt sich aus $B \gg \log n$.

Lösungsvorschlag

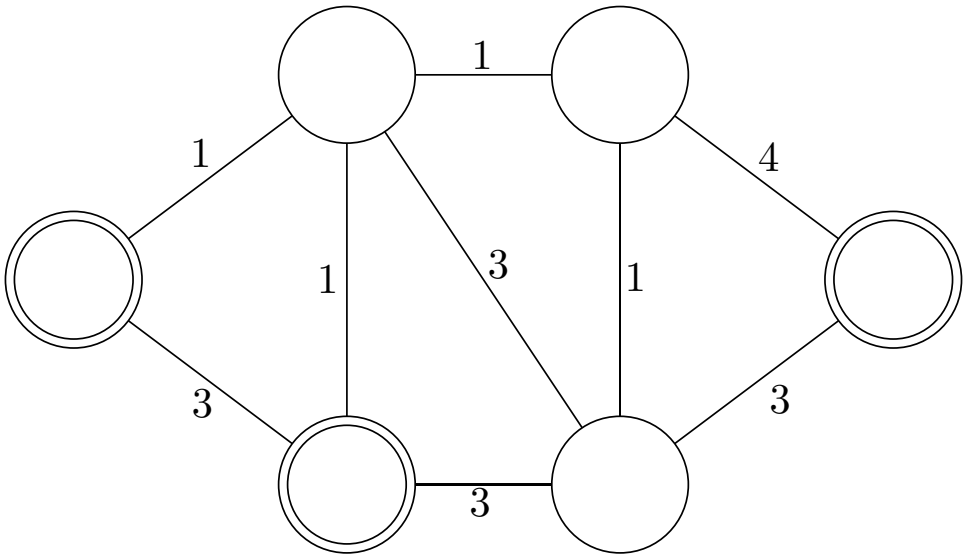
EK

ZK

Aufgabe 2. Shortest Path: Multi-Direktionaler Dijkstra [10 Punkte]

Gegeben sei ein Straßennetz in Form eines ungerichteten zusammenhängenden Graphen (V,E,c) mit n Knoten, m Kanten und Kostenfunktion $c : E \to \mathbb{R}_+$. Um gemeinsam zu lernen, möchten sich $k > 1$ Kommilitonen an einem Knoten $z \in V$ treffen, der die gleiche Distanz zu allen hat. Dabei sei $K \subseteq V$ die Menge der Startknoten der Kommilitonen und $\mu(u,v)$ die Länge des kürzesten Pfades von Knoten u nach Knoten v. Für einen gesuchten Treffpunkt z soll also gelten, dass $\mu(k,z)$ für alle $k \in K$ denselben Wert hat.

a. Es sei folgender Graph gegeben, wobei die doppelt umrandeteten Knoten die Startknoten K sind. Die Kanten sind mit ihren jeweiligen Kosten beschriftet. Markieren Sie alle möglichen Treffpunkte.



[2 Punkte]

Lösung

Es gibt nur einen Treffpunkt: von den vier mittleren Knoten der rechts untere Knoten.

b. Beschreiben Sie, wie Dijkstra's Algorithmus angepasst werden kann, sodass am Ende der Ausführung für jeden Knoten $v \in V$ gespeichert ist, welcher Knoten aus K die kürzeste Distanz zu v hat. Falls es mehrere Knoten mit gleicher Distanz gibt, soll ein beliebiger davon gespeichert werden. Dabei soll die asymptotische Laufzeit von Dijkstra's Algorithmus beibehalten werden (also $\mathcal{O}(n \log n + m)$ mit Fibonacci-Heaps). Begründen Sie die Laufzeit Ihrer Lösung.

Hinweis: Betrachten Sie eine Modifikation von Dijkstra's Algorithmus, bei der von allen Knoten in K gleichzeitig gesucht wird. [4 Punkte]

Lösung

Algorithmus
Wir suchen in einem Graphen mit einem Hilfsstartknoten s, der mit allen Knoten $u \in K$ durch eine Kante (s,u) mit Gewicht 0 verbunden wird. Implementiert wird dies indem die Prioritätsliste mit Einträgen $(u,0)$ für $u \in K$ initialisiert wird. Distanzlabels sind nun Paare von Entfernungen und Labels aus K. Letztere speichern einen Knoten aus K der die kürzeste Entfernung hat. Also wird für $u \in K$ die Distanz $d[u]$ durch $(0,u)$ initialisiert. Bei einer Relaxation einer Kante (w,v) mit Länge c von einem Knoten w mit Label (x,u) zu einem Knoten v mit Label (x',u') wird $d[v] = (x+c,u)$ gesetzt falls $x+c < x'$.

Korrektheit (nicht gefordert)
Füge in den Graphen einen Hilfsknoten hinzu der mit Kosten 0 mit allen K verbunden ist. Führe nun Dijkstra's Algorithmus mit parent pointer aus mit dem Hilfsknoten als Startknoten. Für jeden Knoten ist der nächste Startknoten korrekt, als transitive Auflösung der parent pointer bis auf den vorletzten Knoten welcher aus K sein muss. Zusätzlich ist bekannt, dass parent pointer sich nicht mehr ändern sobald der Knoten gescant wird, es ist also korrekt die Transitivität im Moment des Relaxierens aufzulösen.

Laufzeit
Die Laufzeit von Dijkstra's Algorithmus wird beibehalten, da weiterhin nur konstant viel Zeit pro Relaxation benötigt wird.

c. Wir nehmen nun an, dass genau ein Lösungsknoten z existiert.

Entwerfen Sie einen Algorithmus, der innerhalb einer Laufzeit von $\mathcal{O}(n \log n + km)$ den Knoten z bestimmt. Begründen Sie die Laufzeit Ihrer Lösung.

Hinweis: Erweitern Sie den Algorithmus aus der vorigen Teilaufgabe, so dass dieser zu einem Knoten v die Menge *aller* Startknoten mit kürzester Distanz berechnet. [4 Punkte]

Lösung

Algorithmus
Verwalte für jeden Knoten in einem Bitvektor der Länge k welche K zu diesem den kürzesten Pfad haben. Initialisiere den Bitvektor von jedem Startknoten k so, dass nur das k-te bit gesetzt ist. Immer wenn durch das relaxieren der Kante (u,v) sich die Distanz von v reduziert, ersetze den Bitvektor von v mit dem von u. Wenn die Distanz beibehlaten wird, dann ist der neue Bitvektor von v das bitwise-or von den beiden Bitvektoren von v und u. Gebe v als Lösung aus wenn alle bits gesetzt sind.

Laufzeit
Für jeden Relaxation einer Kante müssen bis zu k bit Operationen durchgeführt werden.

Anmerkungen
Der Algorithmus lässt ein Graph-Voronoi-Diagramm entstehen. Die Voronoi Flächen sind dadurch gegeben, dass alle Knoten einer Fläche denselben Bitvektor haben, bei dem nur das Bit von dem in der Fläche befindlichen Startknoten gesetzt ist. Dabei sind die Ränder der Vorornoi Flächen die Knoten in der mehr als ein Bit gesetzt ist. Mittels Bitparallelismus lässt sich die Laufzeit auf $\mathcal{O}(n\log n + m + km/\log n)$ reduzieren.

Lösungsvorschlag

EK

ZK

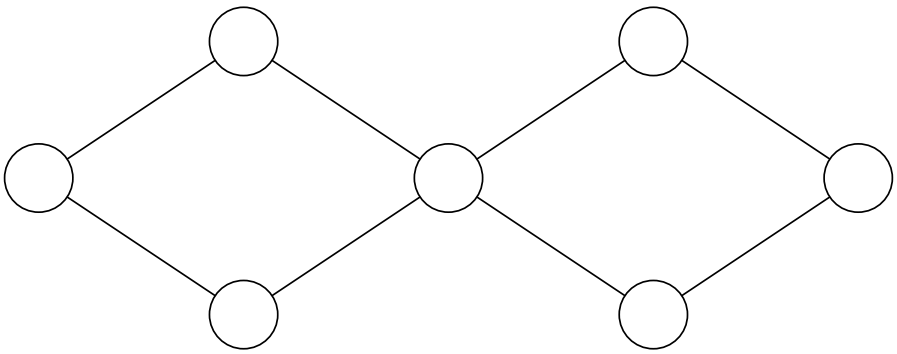
Aufgabe 3. Approximationsalgorithmen: Independent Set [9 Punkte]

Für einen gegebenen ungerichteten Graphen $G = (V, E)$ ist ein Independent Set eine Knotenmenge $I \subseteq V$, so dass keine Kante zwischen zwei Knoten aus I existiert. Wir suchen nun nach Independent Sets maximaler Größe.

Gegeben sei der folgende Greedy-Algorithmus:

1. Wähle einen (beliebigen) Knoten v mit minimalem Grad und füge ihn zu I hinzu.
2. Lösche v und alle benachbarten Knoten aus G.
3. Falls noch Knoten übrig sind, gehe zu 1., sonst gebe I aus.

a. Die Größe des Independent Set, das durch den Greedy-Algorithmus bestimmt wird, ist im Allgemeinen nicht eindeutig und hängt von der Wahl der Knoten ab. Geben Sie die Größe $|I|$ des kleinsten und des größten Independent Set an, das durch den Greedy-Algorithmus für den folgenden Graphen bestimmt werden kann.



Kleinstes $|I| =$ Größtes $|I| =$ [2 Punkte]

Lösung

Kleinstes $|I| = 3$, Größtes $|I| = 4$

Nun soll das Approximationsverhalten des Greedy-Algorithmus in Abhängigkeit des maximalen Knotengrades $\Delta := \max\{d(v) \mid v \in V\}$ untersucht werden.

b. In einem beliebigen Schritt des Greedy-Algorithmus bezeichne v den Knoten, der in diesem Schritt gelöscht wird, und $N(v)$ die Menge der benachbarten Knoten von v (**ohne** bereits gelöschte Knoten).

Zeigen Sie, dass maximal Δ Knoten aus $N(v) \cup \{v\}$ Teil der optimalen Lösung sein können, sofern $\Delta \geq 1$. [2 Punkte]

Lösung

Es gilt $|N(v) \cup \{v\}| \leq \Delta + 1$, wir müssen dabei nur den Fall $|N(v) \cup \{v\}| = \Delta + 1$ betrachten. Da v Kanten zu seinen Nachbarn hat (wobei $N(v) \neq \emptyset$), können nicht v und ein Nachbar von v gleichzeitig Teil der Lösung sein – dies würde die Definition eines Independent Set verletzen.

c. Zeigen Sie, dass der Greedy-Algorithmus für $\Delta \geq 1$ einen Approximationsfaktor von Δ erreicht. [2 Punkte]

Lösung

Nach Teilaufgabe **b.** wählt der Greedy-Algorithmus 1 Knoten und die optimale Lösung maximal Δ Knoten aus der Nachbarschaft jedes greedy gewählten Knoten. Da die Vereinigung der Nachbarschaften alle Knoten des Graphen ergibt, kann der insgesamte Faktor zwischen greedy Lösung und optimaler Lösung nicht größer als Δ sein.

Formale Lösung
Sei $v_1, \ldots, v_k$ die Folge der vom Greedy-Algorithmus gewählten Knoten, $N(v_i)$ bezeichne wieder die Menge der benachbarten Knoten von v_i ohne bereits gelöschte Knoten. I_{OPT} sei ein Independent Set maximaler Größe. Nach Teilaufgabe **b.** gilt

$$|I_{OPT}| = |I_{OPT} \cap \bigcup_{i=1}^{k} N(v_i) \cup \{v_i\}| = \sum_{i=1}^{k} |I_{OPT} \cap (N(v_i) \cup \{v_i\})| \leq \sum_{i=1}^{k} \Delta = k\Delta$$

Da der Greedy-Algorithmus k Knoten wählt, ist dies genau die gewünschte Aussage.

d. Begründen Sie, dass der Greedy-Algorithmus für $\Delta \leq 2$ immer die optimale Lösung findet. Wir gehen O.B.d.A. davon aus, dass G zusammenhängend ist. [3 Punkte]

Lösung

Für $\Delta \leq 2$ ist G entweder ein Pfad oder ein Kreis.

Falls G ein Pfad ist, wird in jedem Schritt ein Endpunkt des Pfades ausgewählt und sein Nachbar entfernt. Also wird in jedem Schritt der Pfad um zwei Knoten kürzer und und das Independent Set um eins größer. Insgesamt entsteht ein Indendent Set der Größe $\lceil \frac{1}{2}|V| \rceil$. Das ist optimal – in einem Pfad kann höchstens jeder zweite Knoten für das Independent Set ausgewählt werden.

Falls G ein Kreis ist, wird im ersten Schritt ein Knoten für das Independent Set ausgewählt und seine zwei Nachbarn entfernt. Danach verläuft das Argument wie für Pfade. Das entstehende Independent Set hat $\lfloor \frac{1}{2}|V| \rfloor$ Knoten.

Lösungsvorschlag

EK

ZK

Konzeptpapier

Lösungsvorschlag

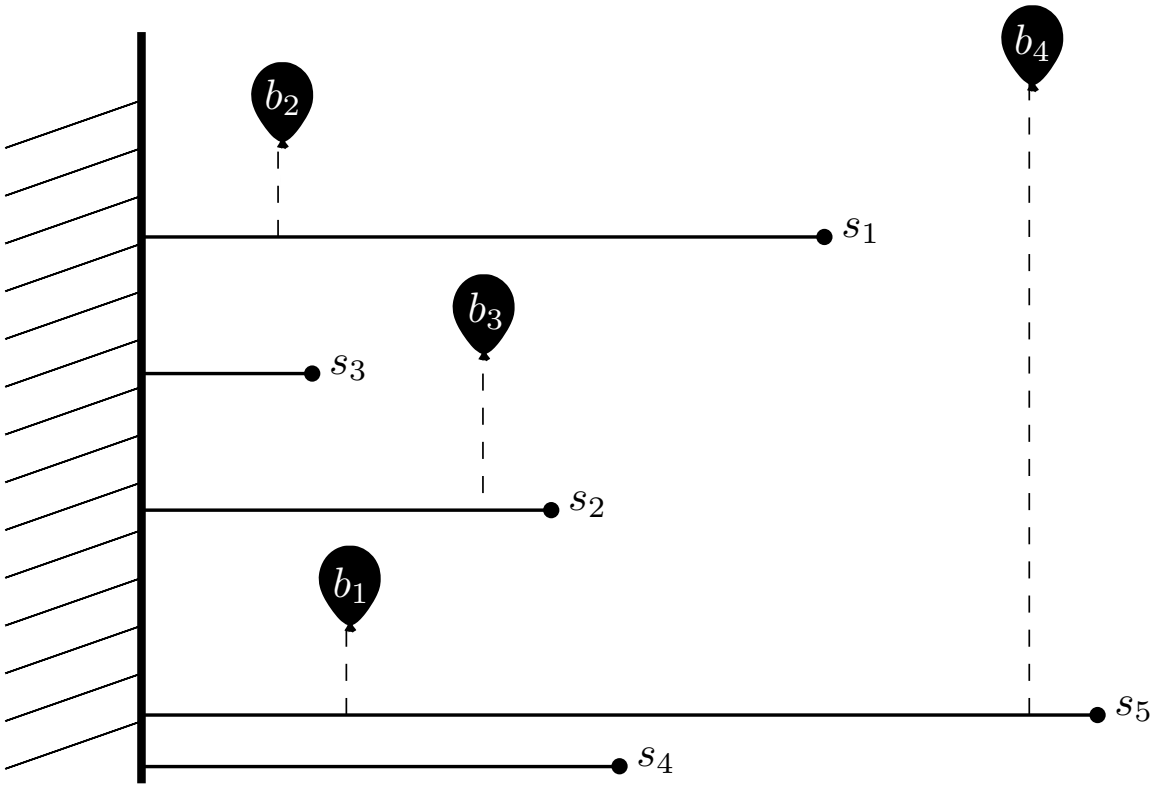
EK

ZK

Aufgabe 4. Geometrische Algorithmen: Luftballons [9 Punkte]

Gegeben ist eine vertikale (parallel zur y-Achse) verlaufende Wand an der Stelle $x = 0$. An der Wand sind horizontal (parallel zur x-Achse) n Stäbe $S = \{s_1, ..., s_n\}$ befestigt, wobei jeder Stab $s_j = (l_j, y_j)$ durch eine Befestigungshöhe y_j und eine Stablänge $l_j > 0$ relativ zur Wand beschrieben wird. Das Ende eines Stabes s_j befindet sich damit an Koordinate (l_j, y_j). Zusätzlich sind m Ballons $B = \{b_1, ..., b_m\}$ gegeben, wobei jeder Ballon durch seine Koordinate $b_i = (x_i, y_i)$ mit $x_i > 0$ beschrieben wird. Ballons sind idealisiert als Punkt anzunehmen. Jeder Ballon soll am nächsten vertikal darunterliegendem Stab befestigt werden. Gehen Sie davon aus, dass dieser immer existiert. Gehen Sie auch davon aus, dass alle Koordinaten aus S und B paarweise verschiedene x und y-Werte haben.

a. Es sei wie in der Grafik dargestellt S und B gegeben. Geben Sie in der Tabelle für jeden Ballon an, an welchem Stab er befestigt wird.



Ballon	b_1	b_2	b_3	b_4
befestigt an Stab				

[1 Punkt]

Lösung

$b_1 : s_5$, $b_2 : s_1$, $b_3 : s_2$, $b_4 : s_5$

In den folgenden zwei Teilaufgaben sollen nun Sweep-Line Algorithmen entworfen werden, die dieses Problem in verschiedener Laufzeit lösen.

In Teilaufgabe **b.** ist eine Zeit von $\mathcal{O}(n + m \log \mathbf{m})$ und in **c.** von $\mathcal{O}(n + m \log \mathbf{n})$ gefordert. Dabei ist m die Anzahl der Ballons und n die Anzahl der Stäbe.

b. Entwerfen Sie einen Sweep-Line Algorithmus, der unter Vernachlässigung des initialen Sortierschrittes der Ereignisse innerhalb von $\mathcal{O}(n + m \log m)$ für jeden Ballon den nächsten vertikal darunterliegenden Stab berechnet.

Begründen Sie die Laufzeit Ihrer Lösung. [4 Punkte]

Lösung

Algorithmus
Als Ereignisse werden die Punkte aus B und S verwendet. Die Sweep-Line geht von rechts nach links. Sortiere also die Ereignisse absteigend nach x-Koordinate. Verwende eine PQ als Sweep-Line-Datenstruktur. Die PQ ist absteigend nach y-Koordinate sortiert. Bei einem Ballon Ereignis wird dieser der PQ hinzugefügt. Bei einem Stab Ereignis wird solange deleteMax aufgerufen bis der nächste Ballon kleinere y-Koordinate hat als der Stab. Die gelöschten Ballons werden an diesem Stab befestigt.

Laufzeit
Jeder Ballon wird einmal in die PQ eingefügt und gelöscht, dass geht in $\mathcal{O}(m \log m)$. Für jeden Stab und für jeden Ballon wird einmal das Maximum der PQ betrachtet was jeweils konstante Zeit benötigt.

c. Entwerfen Sie einen Sweep-Line Algorithmus, der unter Vernachlässigung des initialen Sortierschrittes der Ereignisse innerhalb von $\mathcal{O}(n + m \log n)$ für jeden Ballon den nächsten vertikal darunterliegenden Stab berechnet.

Begründen Sie die Laufzeit Ihrer Lösung. [4 Punkte]

Lösung

Algorithmus
Als Ereignisse werden die Punkte aus B und S verwendet. Die Sweep-Line geht von unten nach oben. Sortiere also die Ereignisse aufsteigend nach y-Koordinate. Verwende einen Stack als Sweep-Line-Datenstruktur. Gehe bei einem Stab Ereignis wie folgt vor:

1. Lösche solange Stäbe vom Stack bis das oberste Element größere x-Koordinate hat.
2. Lege den Stab auf den Stack.

Bei einem Ballon Ereignis nutze binäre suche auf dem Stack um den Stab mit nächst größerer x-Koordinate zu finden. An diesem wird der Ballon befestigt.

Laufzeit
Jeder Stab wird einmal eingefügt und maximal einmal gelöscht. Bei einem Stack geht das in jeweils konstanter Zeit. Für jeden Ballon dann eine binäre Suche auf einem Stack der maximal alle Stäbe hält.

Lösungsvorschlag

EK

ZK

Aufgabe 5. Stringology: Teilstrings und LZ77 [9 Punkte]

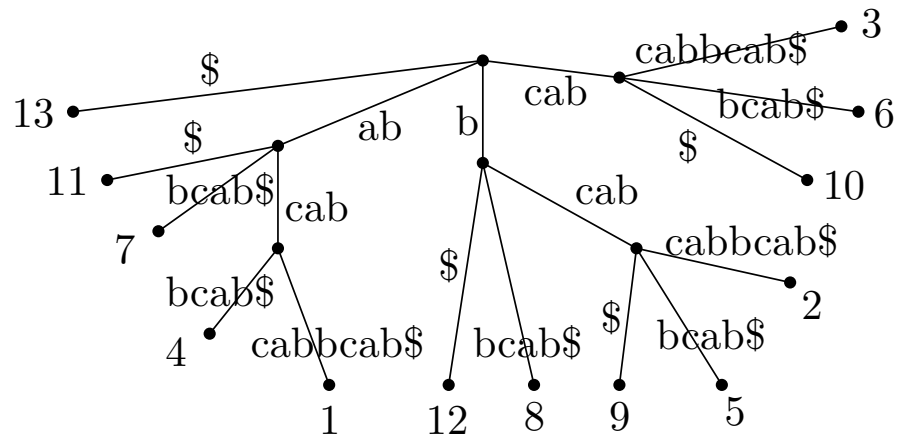
a. Bestimmen Sie alle LZ77-Faktoren des Textes $\mathtt{T} = \mathtt{abcabcabbcab\$}$. [1 Punkt]

Lösung

a|b|c|abcab|bcab|$

b. Erstellen Sie einen kompakten Suffix-Baum für den Text $\mathtt{T} = \mathtt{abcabcabbcab\$}$. Geben Sie an, wie viele unterschiedliche Substrings der Länge 4 der Text hat. [3 Punkte]

Lösung



Die Anzahl der Substrings ist 7 (abbc, abca, bbca, bcab, cabb, cab$, cabc). Das kann einfach aus dem Suffix-Baum abgelesen werden.

Sei $T^{\infty}[i] := T[i \bmod n]$ für einen Text T der Länge n die unendlich lange Wiederholung des Textes.

c. Wenn z die Anzahl der LZ77-Faktoren von T ist, wie groß ist die Anzahl der LZ77-Faktoren von T^{∞}? Wir nehmen hierfür an, dass ein LZ77-Faktor unendlich lang sein darf.

Hinweis: Z. B. für $T = \mathtt{ab\$}$ ist $T^{\infty} = \mathtt{ab\$ab\$ab\$a...}$ [2 Punkte]

Lösung

Die Anzahl ist $z+1$. Der letzte Faktor von T ist immer das $ als eindeutiges Zeichen. Danach wiederholt sich der Text nur noch unendlich oft. Diese Wiederholungen werden aber in einem einzelnen Faktor, nämlich T^{∞} selbst, dargestellt.

d. Sei $S_m := \{S \in \Sigma^m \mid S \text{ ist Teilstring von } T^\infty\}$ für einen Text T mit z LZ77-Faktoren. Begründen Sie, dass sich für $m \geq 2$ die Anzahl der eindeutigen Teilstrings durch $|S_m| \leq (m-1)z$ abschätzen lässt. [3 Punkte]

Lösung

Jeder eindeutige Teilstring mit Länge $m \geq 2$ muss die rechte Grenze eines LZ77-Faktors von T schneiden. Sollte dies nicht der Fall sein, gibt es ein Vorkommen des Teilstrings weiter links im Text. Da es je LZ77-Faktor höchstens $m-1$ mögliche Positionen gibt, die zu einem solchen Schnitt führen, folgt daraus die gesuchte Abschätzung.

Lösungsvorschlag

EK

ZK

Aufgabe 6. Parallele Algorithmen: Akkumulation [10 Punkte]

Es sei eine Liste L mit n Einträgen gegeben, die auf einem Parallelrechner mit p Prozessoreinheiten (PEs) gleichmäßig verteilt ist. Ein Eintrag $(i,x) \in L$ besteht aus einer ID i und einem Betrag x. Nun soll zu jeder ID die Summe aller Beträge mit dieser ID berechnet werden. Die Ausgabe ist also eine Liste, in der für jede in L vorkommende ID genau ein Eintrag existiert, nämlich mit der ID und dem zugehörigen aufsummierten Betrag.

Die Einträge sind bereits nach ID sortiert, wobei PEs mit kleinerem Index entsprechend Einträge mit kleinerer ID haben.

a. Es sei die folgende Beispieleingabe gegeben:

$$\langle\ (\texttt{A},2),(\texttt{B},1),(\texttt{B},-3),(\texttt{B},6),(\texttt{B},4),(\texttt{C},13)\ \rangle$$

Geben Sie die zugehörige Ausgabe an. [1 Punkt]

Lösung

Die korrekte Ausgabe ist:

$$\langle\ (\texttt{A},2),(\texttt{B},8),(\texttt{C},13)\ \rangle$$

b. Wir betrachten den folgenden Algorithmus: Jede PE berechnet zuerst lokal für jede ID den aufsummierten Betrag. Anschließend wird sequentiell und in aufsteigender Reihenfolge für jede PE j (außer der letzten) überprüft, ob die maximal vorkommende ID identisch zur minimalen ID auf PE $j+1$ ist. Falls ja, werden die beiden Einträge addiert und auf PE $j+1$ gespeichert.

Wir nehmen an, dass Nachrichten konstanter Länge zwischen zwei PEs in konstanter Zeit ausgetauscht werden können. Geben Sie die asymptotische Laufzeit der beiden Schritte sowie des kompletten Algorithmus in Abhängigkeit von n und p an, unter Annahme einer optimalen Implementierung. [2 Punkte]

Lösung

Die lokale Akkumulation funktioniert in $\mathcal{O}\left(\frac{n}{p}\right)$ Zeit: Die Daten sind gleichmäßig verteilt und aufsteigend sortiert. Die Akkumulation ist also mit einer einzelnen Iteration über die Einträge in aufsteigender Reihenfolge möglich.
Durch die Sortierung der Daten kann in konstanter Zeit auf den Eintrag mit minimaler bzw. maximaler ID für eine PE zugegriffen werden. Somit benötigt die Deduplikation insgesamt $\mathcal{O}(p)$ Zeit.
Insgesamt ergibt sich also die Laufzeit $\mathcal{O}\left(\frac{n}{p}+p\right)$.

c. Es sei auf 2-Tupeln die Operation $\oplus$ wie folgt definiert:

$$(i,x) \oplus (j,y) := \begin{cases} (i, x+y), & \text{falls } i = j \\ (j,y), & \text{falls } i \neq j \end{cases}$$

Wir gehen nun davon aus, dass $\oplus$ nur auf Tupel (i,x) und (j,y) angewandt wird, für die $i \leq j$ gilt. Der erste Tupel-Eintrag ist also in Reihenfolge der Anwendung von $\oplus$ monoton steigend. Zeigen Sie, dass $\oplus$ unter dieser Einschränkung assoziativ ist.

Hinweis: Betrachten Sie die Fälle $i = j = k$, $i < j = k$ und $i \leq j < k$. [3 Punkte]

Lösung

Wir rechnen die Assoziativität für die im Hinweis genannten Fälle nach.
Im Fall $i = j = k$ gilt

$$((i,x) \oplus (j,y)) \oplus (k,z) = (i, x+y+z) = (i,x) \oplus ((j,y) \oplus (k,z))$$

Im Fall $i < j = k$ gilt

$$((i,x) \oplus (j,y)) \oplus (k,z) = (j, y+z) = (i,x) \oplus ((j,y) \oplus (k,z))$$

Im Fall $i \leq j < k$ gilt

$$((i,x) \oplus (j,y)) \oplus (k,z) = (k,z) = (i,x) \oplus ((j,y) \oplus (k,z))$$

d. Entwerfen Sie einen Algorithmus, der das Problem innerhalb einer Laufzeit von $\mathcal{O}\left(\frac{n}{p} + \log p\right)$ löst. Hierfür stehen Ihnen als Baustein eine parallele Präfixsumme für n Elemente und konstant große Nachrichten mit Laufzeit $\mathcal{O}\left(\frac{n}{p} + \log p\right)$ zur Verfügung. Begründen Sie die Korrektheit und die Laufzeit Ihres Algorithmus.

Hinweis: Machen Sie sich zuerst klar, warum der zweite Schritt des Algorithmus aus Teilaufgabe **b.** nicht parallel ausgeführt werden kann. Beachten Sie außerdem, dass die Ausgabe keine Duplikate enthalten darf und dass eine Präfixsumme einen beliebigen assoziativen Operator verwenden kann. [4 Punkte]

Lösung

Wie zuvor beginnen wir damit, in $\mathcal{O}\left(\frac{n}{p}\right)$ Zeit die lokalen Einträge zu akkumulieren. Anschließen berechnen wir in Zeit $\mathcal{O}\left(\frac{n}{p}+\log p\right)$ eine globale Präfixsumme über alle verbleibenden Einträge, wobei wir den Operator $\oplus$ aus Teilaufgabe **c.** nutzen. Bei allen IDs, die noch mehrfach vorhanden sind (diese sind minimal oder maximal auf der jeweiligen PE!), steht nun der summierte Wert im letzten Eintrag. Nun überprüfen wir parallel für alle PEs, ob die maximale ID auf PE i identisch zur minimalen ID auf PE $i+1$ ist. Falls ja, löschen wir den Eintrag auf PE i. Dieser Schritt funktioniert in konstanter Zeit.

Die gewünschte Gesamtlaufzeit ergibt sich aus der jeweiligen Laufzeit der einzelnen Schritte.

Korrektheit *(wird nicht in diesem Detaillgrad erwartet)*: Nach der lokalen Akkumulation sind nur noch IDs mehrfach vorhanden, die auf mehreren PEs auftauchen – und folglich auf der PE entweder minimal oder maximal sind. Da $\oplus$ durch die gegebene Sortierung assoziativ ist, liefert die Präfixsumme an Position j den Wert

$$\bigoplus_{k\leq j}(i_k,x_k)=\bigoplus_{\substack{k\leq j\\ i_k\neq i_j}}(i_k,x_k)\oplus\bigoplus_{\substack{k\leq j\\ i_k=i_j}}(i_k,x_k)=\bigoplus_{\substack{k\leq j\\ i_k=i_j}}(i_k,x_k)=(i_j,\sum_{\substack{k\leq j\\ i_k=i_j}}x_k)$$

Dies bedeutet, wir erhalten die gewünschte Ausgabe, indem wir zu jeder ID nur den letzten Eintrag behalten. Da lokal keine Duplikate mehr vorhanden sind, wird genau das durch den letzten Schritt des Algorithmus erreicht.

Alternative: Es funktioniert ebenso gut, sofort die Präfixsumme zu berechnen. In diesem Fall braucht es im Anschluss daran sowohl einen lokalen Deduplizierungsschritt als auch eine Deduplizierung zwischen den PEs.