Name:

Vorname:

Matrikelnummer:

Klausur-ID:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders 22.02.2017

Klausur Algorithmen II

Aufgabe 1.	Kleinaufgaben	10 Punkte
Aufgabe 2.	Prioritätslisten: Verallgemeinerung des Radix Heaps	10 Punkte
Aufgabe 3.	Fluss Algorithmen	10 Punkte
Aufgabe 4.	Online Algorithmen: Kantenfärbung	11 Punkte
Aufgabe 5.	String Algorithmen: Muster im Text erkennen	10 Punkte
Aufgabe 6.	Fixed Parameter Algorithmen: Kleinvereine	6 Punkte
Aufgabe 7.	Punkte auf dem Einheitskreis	3 Punkte

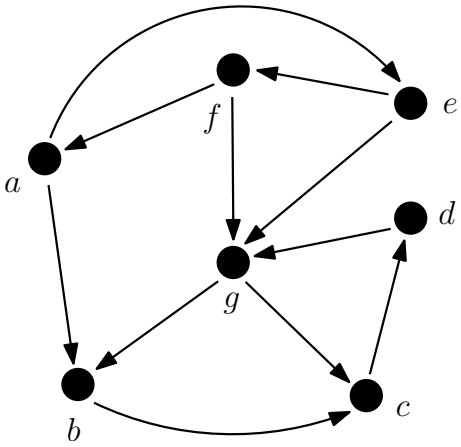
Bitte beachten Sie:

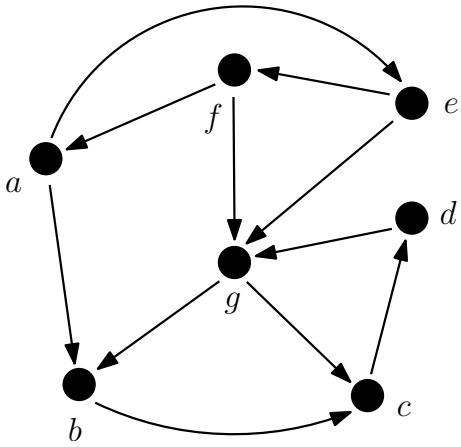
- Als Hilfsmittel ist nur **ein** DIN-A4 Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
- **Schreiben** Sie auf **alle** Blätter der Klausur und Zusatzblätter Ihre **Klausur-ID**.
- Merken Sie sich Ihre **Klausur-ID** auf dem Aufkleber für den Notenaushang.
- Die Klausur enthält **23 Blätter**.
- Zum Bestehen der Klausur sind 20 Punkte hinreichend.

Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [10 Punkte]

a. Gegeben sei folgender Graph (zwei Kopien):

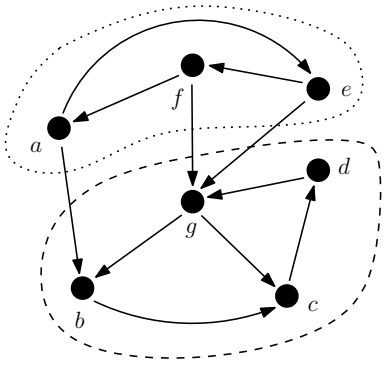




Markieren Sie die starken Zusammenhangskomponenten (SCCs) des Graphen (1 Punkt). Kann man durch Umdrehen einer Kante erreichen, dass der Graph aus fünf SCCs besteht? Falls ja, geben Sie eine solche Kante an (1 Punkt). Falls Sie beide Graphen beschriften, markieren Sie bitte welcher gewertet werden soll.

[2 Punkte]

Lösung



Der Graph besteht aus zwei SCCs – hier durch gestrichelte und gepunktete Umrandungen angedeutet. Dazwischen gibt es drei gerichtete Kanten (a,b), (f,g), (e,g). Das Umkehren der Kante (d,g) zerlegt den Graphen in fünf SCCs.
Weitere Erläuterung: Das Umkehren der Kante (d,g) zerlegt den Graphen in die Komponenten $\{a,f,g\}$, $\{b\}$, $\{c\}$, $\{d\}$ und $\{g\}$. Das Umkehren der Kante (c,d) zerlegt den Graphen ebenfalls in fünf SCCs.

b. Ein Algorithmus habe eine Laufzeit von $O(f(n))$. Sein berechnetes Ergebnis weiche um den Faktor $g(n)$ vom optimalen Wert ab. Geben Sie für die folgenden Fälle an, ob ein PTAS, FPTAS oder APX vorliegt. Begründen Sie Ihre Antworten kurz.

1. $f(n) = \log(n^{\frac{1}{\varepsilon}}) + n^2,\ g(n) = 1+\varepsilon$
2. $f(n) = \log(n)^{\frac{1}{\varepsilon}} + n,\ g(n) = 1+2\varepsilon$
3. $f(n) = \log(n^{\frac{1}{\varepsilon}}) + n,\ g(n) = 1+\sqrt{\log n}$

[3 Punkte]

Lösung

Ein APX (*approximable*) ist ein Algorithmus, der in polynomieller Zeit in der Eingabegröße eine Approximation mit konstantem Approximationsfaktor berechnet. Ein PTAS (*polynomial time approximation scheme*) ist ein Algorithmus, der eine $(1\pm\varepsilon)$-Approximation in polynomieller Zeit in der Eingabegröße berechnet. Dahingegen ist ein FPTAS (*fully polynomial time approximation scheme*) zusätzlich polynomiell in $\frac{1}{\varepsilon}$.

1. $f(n) = \log(n^{\frac{1}{\varepsilon}}) + n^2,\ g(n) = 1+\varepsilon$:
   FPTAS, da $(1+\varepsilon)$-Approximation und Laufzeit polynomiell in n und $\frac{1}{\varepsilon}$
   (da $\log(n^{\frac{1}{\varepsilon}}) = \frac{1}{\varepsilon}\log n$)
2. $f(n) = \log(n)^{\frac{1}{\varepsilon}} + n,\ g(n) = 1+2\varepsilon$:
   PTAS, da $(1+\varepsilon)$-Approximation und Laufzeit polynomiell in n, aber nicht polynomiell in $\frac{1}{\varepsilon}$. Beachten Sie, dass der Approximationsfaktor $g(n) = 1+2\varepsilon$ durch einen neuen Approximationsfaktor $h(n) = 1+\varepsilon'$ ersetzt werden kann.
3. $f(n) = \log(n^{\frac{1}{\varepsilon}}) + n,\ g(n) = 1+\sqrt{\log n}$:
   Kein Approximationsalgorithmus, da die Laufzeit zwar polynomiell in n und $\frac{1}{\varepsilon}$ ist, jedoch garantiert $g(n)$ keine konstante Approximation.

c. Gegeben sei der Text $T = \mathrm{a}^n$. Geben Sie sowohl die Länge der (naiven) Lempel-Ziv Kompression von T als auch die Länge des längsten Eintrags im aufgebauten Wörterbuch in Abhängigkeit von n im O-Kalkül an. Begründen Sie Ihre Antwort. [2 Punkte]

Lösung

Der komprimierte Text besteht somit aus $O(\sqrt{n})$ Zeichen und der längste Eintrag im aufgebauten Wörterbuch ist $O(\sqrt{n})$ Zeichen lang.
Begründung: In jedem Schritt wird der längeste Eintrag im Wörterbuch verwendet und ein neuer Eintrag hinzugefügt. Der neue Eintrag ist genau ein Zeichen länger als der zuvor verwendete Eintrag. Somit ist die Größe des komprimierten Texts dasjenige s, welches $\sum_{i=1}^{s} i \geq n$ minimiert. Die Lösung folgt aus $\sum_{i=1}^{s} i = \frac{s(s+1)}{2} \in O(s^2)$.

d. Sei π eine Permutation der Zahlen von 1 bis n. Gegeben ist ein Wavelet-Tree, welcher die n Punkte der Form (i, π_i) verwaltet. Im Wavelet-Tree werden die Punkte bzgl. ihrer y-Koordinate rekursiv aufgeteilt. Skizzieren Sie grob einen Algorithmus, welcher in Zeitkomplexität $O(\log n)$ für ein i den Wert π_i rekonstruiert.
Hinweis: Der Wavelet-Tree besteht aus n Bitvektoren, wobei die rank-Operation in $O(1)$ ausgeführt werden kann. [3 Punkte]

Lösung

Rekursiver Algorithmus. Sei b der Bitvector auf der aktuellen Ebene und ℓ dessen Länge. Falls $b[i] = 0$ so fahre rekursiv auf dem linken Teibaum mit $x = \text{rank}_0[i]$ fort und gebe das Ergebnis aus der Rekursion zurück. Sonst fahre rekursiv auf dem rechten Teilbaum mit $i = \text{rank}_1[i]$ fort und gebe das Ergebnis aus der Rekursion + $\lceil \ell/2 \rceil$ zurück. Im Basisfall ($|b| = 2$) wird $b[i]$ zurückgegeben.

Lösungsvorschlag

Aufgabe 2. Prioritätslisten: Verallgemeinerung des Radix Heaps [10 Punkte]

Für ein $b \geq 2$ und ein $K := 1 + \lfloor \log_b C \rfloor$ ist ein b-K-Radix Heap definiert als eine Datenstruktur bestehend aus einem 2D-Array von Listen $B[i][j]$, einem Hilfsarray $H[j]$ mit $0 \leq i < b$ und $-1 \leq j \leq K$ und einem Schlüssel *min*. Die Schlüssel im b-K-Radix Heap werden in dem Stellenwertsystem zur Basis b dargestellt. Der Schlüssel *min* ist der Schlüssel des zuletzt entfernten Elements. Zu jedem Zeitpunkt enthält ein b-K-Radix-Heap ausschließlich Elemente mit Schlüsseln aus dem Wertebereich $[\min, \min + C]$. Für zwei Schlüssel x und y mit $x \neq y$ ist $\mathrm{msd}_b(x,y)$ die Position der höchstwertigen Ziffer im Stellenwertsystem zur Basis b an der sich die beiden Schlüssel unterscheiden. Für zwei gleiche Schlüssel x und y gilt $\mathrm{msd}_b(x,y) = \mathrm{msd}_b(x,x) = -1$. Das Element (val, key) in der Datenstruktur soll jederzeit an Position $B[i][j]$ stehen, wobei $j := \min(\mathrm{msd}_b(min, key), K)$ gilt und die j'te Ziffer von Schlüssel *key* gleich i ist. Der Eintrag $H[j] := \sum_{i=0}^{b-1} |B[i][j]|$ im Hilfsarray speichert die aktuelle Anzahl an Elementen, die in der j'ten Spalte gespeichert sind.

a. Gegeben sei der Ausgangszustand eines 3-3-Radix Heaps mit (Wert, Schlüssel)-Paaren:

		j				
		-1	0	1	2	3
i	0					
	1			$(a, 11\mathbf{1}0_3)$		
	2			$(b, 11\mathbf{2}0_3)$, $(c, 11\mathbf{2}1_3)$	$(d, 1\mathbf{2}22_3)$	$(e, \mathbf{2}100_3)$
H		0	0	3	1	1
min		1101_3				

Führen Sie nacheinander die folgenden drei Operationen auf dem Ausgangszustand des 3-3-Radix Heaps aus. Geben Sie dazu den Zustand der Datenstruktur nach jeder Operation an:

- 2x `DeleteMin()`
- `decreaseKey(e,`1122_3`)`

`DeleteMin()`:

		j				
		-1	0	1	2	3
i	0					
	1					
	2					
H						
min						

`DeleteMin()`:

		j				
		-1	0	1	2	3
i	0					
	1					
	2					
H						
min						

`decreaseKey(e,`1122_3`)`:

		j				
		-1	0	1	2	3
i	0					
	1					
	2					
H						
min						

[3 Punkte]

Lösung

Die Operationen ändern den Zustand des 3-3-*Radix Heaps* wie folgt:

`DeleteMin()`:

		j				
		-1	0	1	2	3
i	0					
	1					
	2			$(b, 1120_3)$, $(c, 1121_3)$	$(d, 1222_3)$	$(e, 2100_3)$
H		0	0	2	1	1
min		1110_3				

`DeleteMin()`:

		j				
		-1	0	1	2	3
i	0					
	1		$(c, 1121_3)$			
	2				$(d, 1222_3)$	$(e, 2100_3)$
H		0	1	0	1	1
min		1120_3				

`decreaseKey(e,`1122_3`)`:

		j				
		-1	0	1	2	3
i	0					
	1		$(c, 1121_3)$			
	2		$(e, 1122_3)$		$(d, 1222_3)$	
H		0	2	0	1	0
min		1120_3				

b. Ein neuer 3-3-*Radix Heaps* und $min = 0001_3$ enthalte die (Wert, Schlüssel)-Paare $(a, 0002_3)$, $(b, 0011_3)$, $(c, 0012_3)$, $(d, 0022_3)$, $(e, 1000_3)$. Geben Sie den Zustand des neuen 3-3-Radix Heaps an.

		j				
		-1	0	1	2	3
i	0					
	1					
	2					
H						
min						

[3 Punkte]

Lösung

Der Ausgangszustand des 3-3-*Radix Heaps* ist wie folgt:

	-1	0	1	2	3
0					
1			$(b, 0011_3)$, $(c, 0012_3)$		$(e, 1000_3)$
2		$(a, 0002_3)$	$(d, 0022_3)$		
H	0	1	3	0	1

min = 0001_3

c. Geben Sie Pseudocode an, welcher die Operation `DeleteMin()` in amortisiert $O(b+K)$ Zeit ausführt. Nehmen Sie an, dass `Insert()` amortisiert $O(K)$ Zeit benötigt und somit jeweils $O(K)$ Token für die Analyse zur Verfügung stellt. Begründen Sie die Laufzeit Ihres Algorithmus kurz. Eine Implementierung mit schlechterer Laufzeit wird mit maximal 2 Punkten bewertet. [4 Punkte]

Lösung

Algorithm 1 `DeleteMin()`

$i \leftarrow \min\{i | -1 \leq i \leq K \wedge H[i] > 0\}$ ▷ $O(K)$
$j \leftarrow \min\{j | 0 \leq j < b \wedge B[i][j] \neq \emptyset\}$ ▷ $O(b)$
if $i > -1$ **then**
  $H[i]-- = |B[i][j]|$ ▷ 1x pro Element $\rightarrow$ *amor.* $O(1)$
  $min \leftarrow \min(B[i][j])$ ▷ 1x pro Element $\rightarrow$ *amor.* $O(1)$
  for all $x \in B[i][j]$ **do** ▷ 1x pro Element $\rightarrow$ *amor.* $O(1)$
    $m \leftarrow \min(\text{msd}_b(min, x), K)$
    $B[m][x[m]] \leftarrow x$
    $H[m]++$
  $B[i][j] \leftarrow \emptyset$
$min \leftarrow \text{Pop}(B[-1][j])$ ▷ $O(1)$
$H[-1] \leftarrow H[-1] - 1$
return *min*

Die Analyse der Laufzeit stützt sich maßgeblich auf folgende Erkenntnisse:

- Steht in der ersten Spalte ($B[\cdot][-1]$) ein Element, so kann ein beliebiges Element aus der ersten nicht-leeren Zeile der Spalte "-1" zurückgegeben werden. Dieses Element wird das neue Minimum und alle anderen Elemente sind schon an der korrekten Position gespeichert.
- Wird das erste nicht-leere Element im Container $B[i][j]$ mit $j \geq 0$ gefunden, so müssen alle Elemente aus diesem Container in Container $B[\cdot][k]$ verschoben werden, wobei jetzt $k < j$ gelten muss. Somit muss jedes Element höchstens K-mal verschoben werden. Die Kosten für die Verschiebeoperationen können somit mit der Operation `Insert` amortisiert werden.

Aus diesen beiden Erkenntnissen und den Kommentaren im Pseudocode ergibt sich eine amortisierte Laufzeit der Operation `DeleteMin()` von $O(b+K)$.

Lösungsvorschlag

Aufgabe 3. Fluss Algorithmen [10 Punkte]

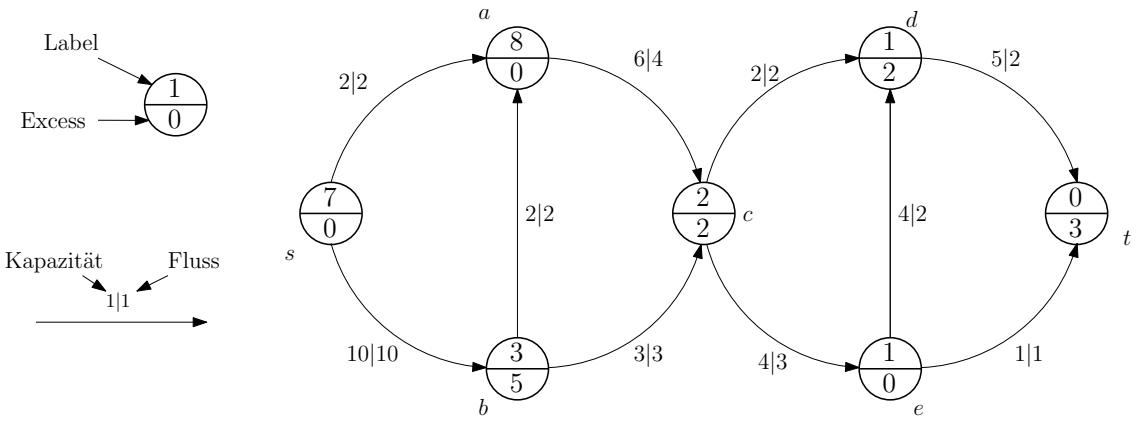
a. Sei G_f der Residualgraph während der Ausführung des Preflow-Push-Algorithmus. Betrachten Sie die Kante $e = (u, v) \in G_f$ mit einer Restkapazität von c_e^f. Sei excess(u) der Excess auf Knoten u und $d(u)$ der Level von Knoten u. Weiter sei $d(v)$ der Level von Knoten v. Geben Sie die drei Bedingungen für c_e^f, $d(u)$, $d(v)$ und excess(u) an, sodass auf der Kante e direkt die Operation `Push(`e`, min(excess(`u`), `c_e^f`))` aufgerufen werden kann.

[2 Punkte]

Lösung

1. Excess auf Knoten u: `Excess(`u`)` > 0
2. Positive Restkapazität der ausgehenden Kante e: $c_e^f > 0$
3. Der Level von Knoten u muss größer sein als der Level von Knoten v: $d(u) > d(v)$

b. Betrachten Sie das unten abgebildete Flussnetzwerk mit Quelle s und Senke t. Auf dem Flussnetzwerk wurde schon ein Teil des Preflow-Push-Algorithmus ausgeführt. Geben Sie die Kanten an, deren Fluss sofort durch die Operation `Push()` erhöht werden kann. Geben Sie auch jeweils den Wert des erhöhenden Flusses an.

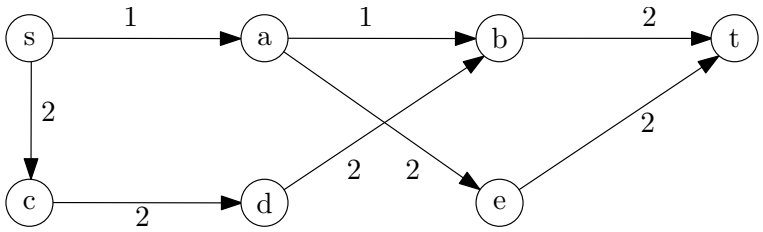


[2 Punkte]

Lösung

- Kante: (c, e) erhöhender Fluss: 1
- Kante: (d, t) erhöhender Fluss: 2

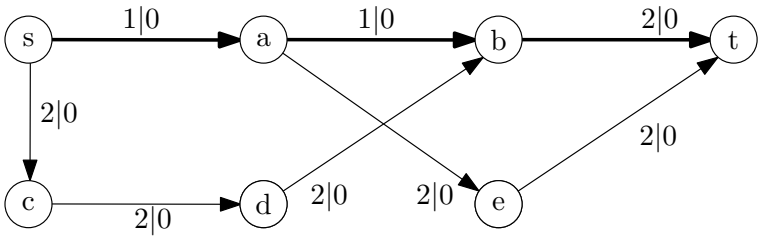
c. Betrachten Sie das unten abgebildete Flussnetzwerk. Die Kanten sind mit ihrer Kapazität beschriftet. Berechnen Sie mit dem *Ford Fulkerson* Algorithmus den maximalen Fluss von Knoten s zu Knoten t. Augmentieren Sie jeweils den **kürzesten Pfad** im Residualgraphen.
Geben Sie dabei für jeden Schritt den **erhöhenden Pfad** in Form einer Knotenliste **und** den **Wert** des erhöhenden Pfades an. Geben Sie zusätzlich den **Wert des maximalen Flusses** nach der Ausführung an.



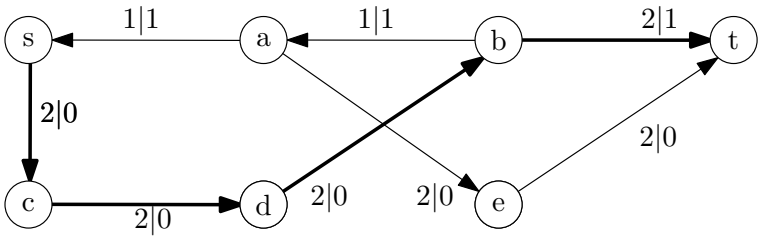
[3 Punkte]

Lösung

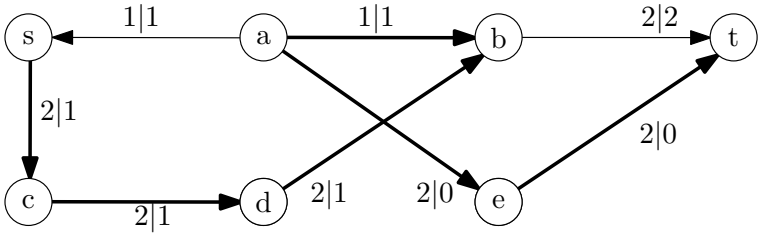
1. Erhöhender Pfad 1: $s \to a \to b \to t$ Wert 1



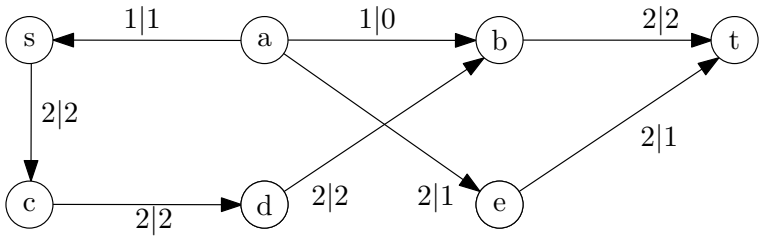
2. Erhöhender Pfad 2: $s \to c \to d \to b \to t$ Wert 1



3. Erhöhender Pfad 3: $s \to c \to d \to b \to a \to e \to t$ Wert 1



Folgendes Flussnetzwerk zeigt den blockierenden Fluss:



Wert des maximalen Flusses: 3

Anmerkung: Die gezeigten Residualgraphen sind nicht Teil der geforderten Lösung, sondern dienen ausschließlich zur Verdeutlichung derselben.

d. Gegeben sei ein Flussnetzwerk $N = (G = (V,E), s, t, c)$ und ein maximaler Fluss f in N. Geben Sie einen Algorithmus an, der in $O(|V| + |E|)$ Zeit prüft, ob sich der maximale Fluss f durch Entfernen der Kante $e := (u,v) \in E$ um genau $f(e)$ reduziert. Für eine Lösung mit schlechterer Laufzeit gibt es maximal einen Punkt. [3 Punkte]

Lösung

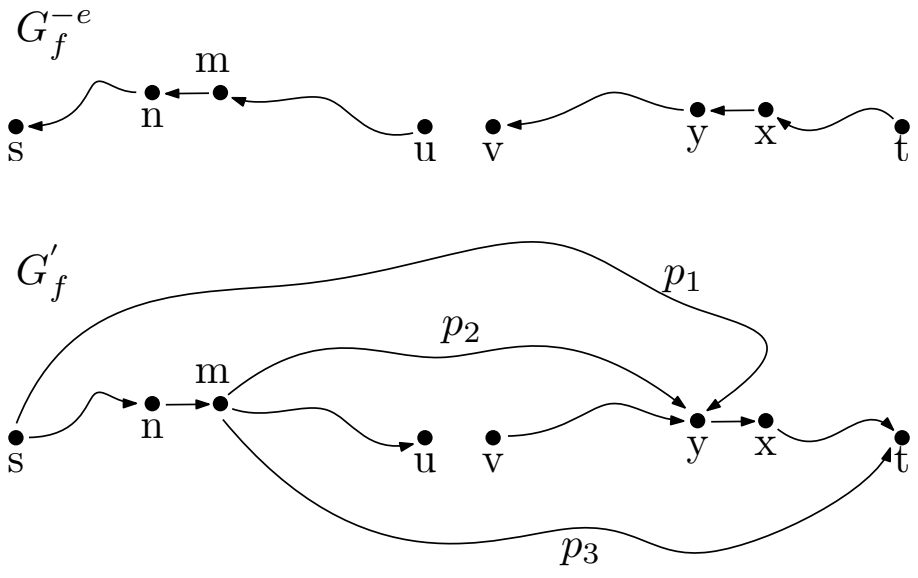
Entferne die Kante e aus dem Residualgraph. Suche danach mittels Breitensuche einen Pfad im Residualgraphen von u nach v. Der maximale Fluss f reduziert sich genau um $f(e)$, genau dann wenn kein Pfad von u nach v gefunden wurde. Die Breitensuche benötigt $O(|V|+|E|)$ Zeit.

Zusatzinformation:
Sei G_f^{-e} der resultierende Graph, wenn aus dem Residualgraph G_f die Kante e entfernt wird. Sei weiter G_f' der resultierende Graph, wenn im Graphen G_f^{-e} der Fluss $f(e)$ von u zurück nach s gepusht wurde und der Fluss $f(e)$ von t zurück nach v gepusht wurde. Seien E_s die Kanten, deren Fluss sich ändert, wenn der Fluss $f(e)$ von u zurück nach s gepusht wird. Seien E_t die Kanten, deren Fluss sich ändert, wenn der Fluss $f(e)$ von t zurück nach v gepusht wird. Es ist offensichtlich, dass sich der maximale Fluss f nicht um genau $f(e)$ reduziert, wenn es im Graphen G_f^{-e} einen Pfad von u nach v gibt.
Es reicht also zu zeigen, dass sich der Fluss um genau $f(e)$ reduziert, wenn es im Graphen G_f^{-e} keinen Pfad von u nach v gibt (Gegenrichtung). Wir zeigen diese Aussage, indem wir zeigen, dass es in G_f^{-e} einen Pfad $u \to v$ gibt, wenn sich der maximale Fluss f um weniger als $f(e)$ reduziert. Wenn sich der maximale Flusse f um weniger als $f(e)$ reduziert, dann muss es in G_f' einen Pfad von s nach t geben. Es reicht also zu zeigen, dass wenn es in G_f' einen Pfad p von s nach t gibt, dann gibt es in G_f^{-e} einen Pfad $u \to v$. Es ist offensichtlich, dass der Pfad p mindestens eine der Kanten aus E_s oder E_t enthalten muss. Wäre dies nicht der Fall, würde dieser Pfad p auch im ursprünglichen Residualgraphen G_f existieren und f wäre kein maximaler Fluss in G_f. Es existieren also drei Fälle, welche wir im folgenden betrachten:

1. $(n, m) \in p \cap E_s \wedge (y, x) \in p \cap E_t$: Es gibt also einen Pfad $\{s, \cdots, n, m, p_2, y, x, \cdots, t\}$ in G_f'. Somit gibt es aber auch einen Pfad $\{u, m, p_2, y, v\}$ in G_f^{-e}.

2. $(n, m) \in p \cap E_s \wedge p \cap E_t = \emptyset$: Es gibt also einen Pfad $\{s, \cdots, n, m, p_3, t\}$ in G_f'. Somit gibt es aber auch einen Pfad $\{u, m, p_3, t, v\}$ in G_f^{-e}.

3. $p \cap E_s = \emptyset \wedge (y, x) \in p \cap E_t$: Es gibt also einen Pfad $\{s, p_1, y, x, \cdots, t\}$ in G_f'. Somit gibt es aber auch einen Pfad $\{u, s, p_1, y, v\}$ in G_f^{-e}.

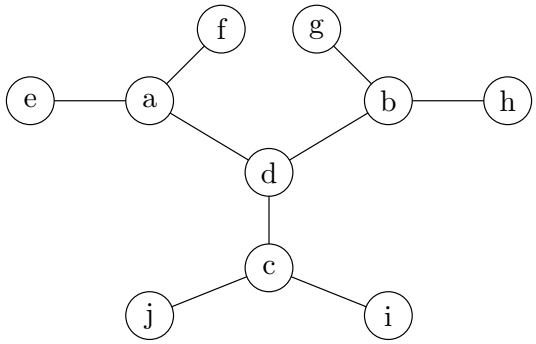


Lösungsvorschlag

Aufgabe 4. Online Algorithmen: Kantenfärbung [11 Punkte]

Eine **Kantenfärbung** eines Graphen $G = (V,E)$ weist **jeder Kante** $e \in E$ eine Farbe $c(e)$ zu, unter der Bedingung, dass für zwei inzidente Kanten[1] e_1 und e_2 gilt, dass $c(e_1) \neq c(e_2)$.
Ein Algorithmus, welcher eine Kantenfärbung online berechnet, hat initial kein Wissen über den Eingabegraphen. Wird dem Algorithmus eine neue Kante e aus dem Graphen übergeben, so weist der Algorithmus der neuen Kante e eine Farbe zu. Die Farben bisher eingelesener Kanten dürfen über den Verlauf des Algorithmus nicht geändert werden.

a. Wie viele Farben benötigt eine Kantenfärbung mindestens, um den unten abgebildeten Graph zu färben (1 Punkt)? Geben Sie eine solche Kantenfärbung an, indem Sie die Farben im Form von Zahlen an die Kanten schreiben (1 Punkt).

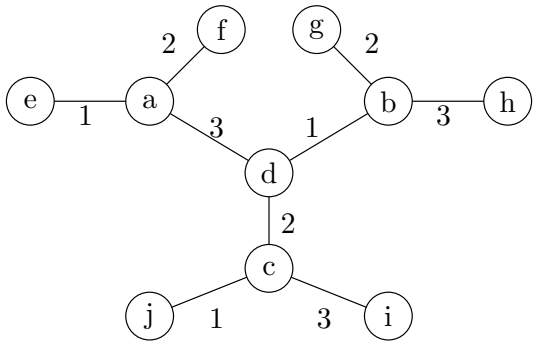


Minimale Anzahl an Farben:

[2 Punkte]

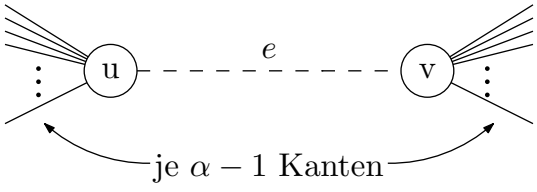
Lösung

Es existiert eine 3-Kantenfärbung für diesen Graph. Dies ist auch die kleinste Kantenfärbung, da die Anzahl an Farben durch den maximalen Knotengrad (hier 3) nach unten beschränkt ist. Eine 3-Kantenfärbung kann wie folgt aussehen:



[1] Zwei Kanten sind inzident, falls sie einen gemeinsamen Endknoten teilen.

b. Sei α der maximale Knotengrad im Graph nach Einfügen der Kante e. Geben Sie einen deterministischen Onlinealgorithmus an, welcher beim Einfügen der Kante e im schlechtesten Fall eine $2\alpha - 1$ Kantenfärbung in amortisiert $O(\alpha)$ Zeit berechnet. Begründen Sie die Laufzeit kurz. Beweisen Sie zusätzlich, dass Ihr Onlinealgorithmus, im schlechtesten Fall eine $2\alpha - 1$ Kantenfärbung berechnet.
Tipp: Betrachten Sie den folgenden Teilgraph:



[6 Punkte]

Lösung

Algorithmus:
Unser Onlinealgorithmus verwendet folgende Datenstruktur: Die bisher gesehenen Knoten werden in einer Hashmap H verwaltet. Zu jedem bisher gesehenen Knoten v speichert die Hashmap eine Liste von Kanten L_v. In der Liste L_v sind die Farben der zu Knoten v adjazenten Kanten in aufsteigender Reihenfolge gespeichert. Wird dem Onlinealgorithmus eine neue Kante (u, v) übergeben, so werden die folgenden drei Schritte ausgeführt:

1. Wir prüfen, ob die Knoten u und v schon in der Hashmap H verwaltet werden. Falls Knoten u bzw. Knoten v nicht in der Hashmap enthalten sind, so fügen wir die Knoten in die Hashmap ein.

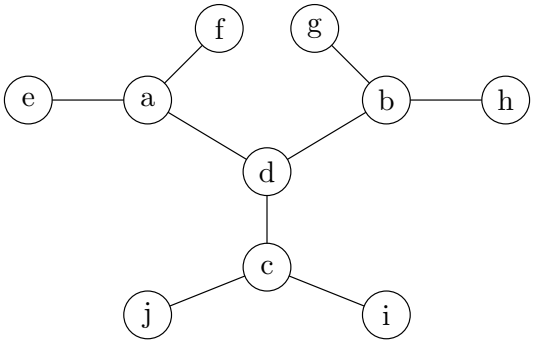
2. Wir iterieren parallel über die sortierten Farblisten L_u und L_v und suche die Farbe mit kleinstem Wert, die nicht in L_u und nicht in L_v vorkommt. Dabei wird für jeden Knoten die letzte besuchte Farbe gespeichert (wir gehen von einem Dummy-Element am Listenanfang aus).

3. Füge die gefundene Farbe in die beiden sortierten Listen hinter die jeweils letzte besuchte Farbe ein und gebe die Farbe aus.

Laufzeit:
Ein neuer Knoten kann in amortisiert $O(1)$ Zeit in die Hashmap eingefügt werden. Die Suche nach der kleinsten nicht vergebenen Farbe iteriert über zwei sortierte Farblisten der Größe $O(\alpha)$ und benötigt somit $O(\alpha)$ Zeit. Das Einfügen der neuen Farbe in die sortierte Liste erfolgt in $O(1)$ Zeit, da die Einfügepositionen in den Listen bekannt sind. Somit ergibt sich eine amortisierte Gesamtlaufzeit von $O(\alpha)$.

$2\alpha - 1$ **Kantenfärbung:** Beweis per Induktion über die hinzugefügten Kanten.
Induktionsannahme: Sei α der maximale Grad durch die bis jetzt hinzugefügten Kanten. Dann wurden ausschließlich Farben aus dem Intervall $[1, 2\alpha - 1]$ vergeben.
Induktionsanfang: Nach Hinzufügen der ersten Kante e gilt $\alpha = 1$. Die Kante e wurde mit der Farbe 1 gefärbt.
Induktionsschritt: Nehmen wir an, dass die Kante $e = (u, v)$ hinzugefügt wird. Fall 1: Der maximale Grad erhöht sich durch Hinzufügen von e nicht. In diesem Fall haben die Knoten u und v einen maximalen Grad von jeweils $\alpha - 1$. Ihre ausgehenden Kanten sind also mit höchstens verschiedenen $2(\alpha - 1) = 2\alpha - 2$ gefärbt. Somit existiert mindestens eine valide Farbe aus dem Intervall $[1, 2\alpha - 1]$, mit der die Kante e gefärbt werden kann. Der Algorithmus färbt e mit der kleinsten validen Farbe. Fall 2: Der maximale Grad erhöht sich durch hinzufügen von e auf $\alpha + 1$. Der neuen Kante wird eine Farbe aus dem Intervall $[1, 2\alpha]$ zugewiesen, da nach Induktionsannahme bisher nur Farbe aus dem Intervall $[1, 2\alpha - 1]$ vergeben wurden.

c. Gegeben sei wieder der Graph aus der ersten Teilaufgabe:



Anzahl an Farben: ☐

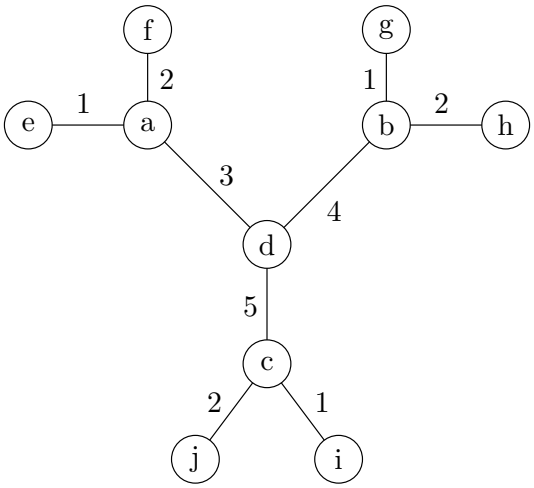
Ihrem Algorithmus werden die Kanten in folgender Reihenfolge übergeben:

$$(a,e),(a,f),(b,g),(b,h),(c,i),(c,j),(d,a),(d,b),(d,c)$$

Führen Sie Ihren Onlinealgorithmus aus der vorletzten Teilaufgabe aus und tragen Sie die Kantenfärbung in den abgebildeten Graph ein. Wie viele Farben benötigt Ihr Onlinealgorithmus? Was sagt Ihnen die Kantenfärbung aus dieser Teilaufgabe und aus der ersten Teilaufgabe über den *Competitive Ratio* Ihres Onlinealgorithmus? [3 Punkte]

Lösung

Die Kantenfärbung des Onlinealgorithmus besteht aus 5 verschiedene Farben.



Die *Competitive Ratio* von dem Onlinealgorithmus aus der vorherigen Teilaufgabe ist somit $5/3$ oder höher.

Lösungsvorschlag

Aufgabe 5. String Algorithmen: Muster im Text erkennen [10 Punkte]

In dieser Aufgabe geht es darum, das Pattern P im Text T zu finden. Aufgrund eines Datenverlusts ist der Text T nicht mehr auffindbar. Sie haben jedoch noch eine Verschiebetabelle[2] (Border-Array) des Knuth-Morris-Pratt-Algorithmus gespeichert. Sie wissen auch, dass P aus m Zeichen und T aus n Zeichen bestand. Im Folgenden seien die Zeichen # und \$ eindeutige Trennsymbole.

a. Nehmen Sie an, dass das Border-Array B aus dem Text "$P\#T\$$" aufgebaut wurde. Das Border-Array von $P = ab$ und $T = cababcabrabcr$ würde wie folgt aussehen.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	b	#	c	a	b	a	b	c	a	b	r	a	b	c	r	\$
-1	0	0	0	0	1	2	1	2	0	1	2	0	1	2	0	0

Geben Sie einen Algorithmus an, welcher diejenigen Positionen im Text T ausgibt, an denen das Pattern P im Text T *endet*. Die Laufzeit des Algorithmus soll in $O(m+n)$ liegen. Beachten Sie, dass der Text T und das Pattern P nicht mehr vorliegen. Die Länge m von P, die Länge n von T und das Border-Array B stehen Ihnen jedoch zur Verfügung. [2 Punkte]

Lösung

Scanne über das Border-Array von Position $m+2$ bis $m+n+2$. Falls an Position i, mit $m+2 \leq i \leq m+n+2$, ein Feld mit Wert m gelesen wird, so kommt das Pattern P im Ausgangstext vor und endet an Position $i-m-2$.

b. Nehmen Sie nun an, dass das Border-Array B nur noch aus dem Text "$PT\$$" aufgebaut wurde. Geben Sie das Border-Array von "$PT\$$" für $P = ab$ und $T = cababcabrabcr$ an.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a	b	c	a	b	a	b	c	a	b	r	a	b	c	r	\$

[2 Punkte]

Lösung

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a	b	c	a	b	a	b	c	a	b	r	a	b	c	r	\$
-1	0	0	0	1	2	1	2	3	4	5	0	1	2	3	0

[2] Für einen Text S speichert das Border-Array an Position i die Länge des längsten Suffixes von $S[1,\ldots,i-1]$, das auch echtes Präfix von S ist.

c. Nehmen Sie an, dass das Border-Array B wieder nur aus dem Text "$PT\$$" aufgebaut wurde. Nennen Sie eine notwendige und hinreichende Bedingung, die für den Eintrag an Position i im Border-Array gelten muss, sodass an Position $i-m-1$ im Ausgangstext T ein Vorkommen von P endet. [3 Punkte]

Lösung

Bedingung in Form einer Berechnungsvorschrift:

Algorithm 2 `B: Border-Array, m: Länge des Patterns P, i: Index im Border-Array`

```
while B[i] > m do
    i ← B[i] + 1
if B[i] = m then
    True
else
    False
```

Entweder es gilt gleich $B[i] = m$ oder wir können die Vorschrift $i := B[B[i]+1]$ so lange anwenden, bis $i = m$ gilt.

d. Nehmen Sie wieder an, dass das Border-Array B aus dem Text "$PT\$$" aufgebaut wurde. Geben Sie einen Algorithmus an, der auf Grundlage von B alle Endpositionen von P im Text T in $O(m+n)$ Zeit ausgibt. Beachten Sie, dass der Text T und das Pattern P nicht mehr vorliegen. Die Länge m von P, die Länge n von T und das Border-Array B stehen Ihnen jedoch zur Verfügung. Korrekte Lösungen mit schlechterer asymptotischer Laufzeit bekommen maximal 1 Punkte. [3 Punkte]

Lösung

Linearzeitalgorithmus:

1. Erstelle ein Hilfsarray H der Größe $m+n+1$ mit 0 initialisiert.
2. Iteriere über die Einträge in B von links nach rechts.
   - Falls $B[i] = m$ oder $H[B[i]+1] = 1$, so setze $H[i] = 1$. In diesem Fall haben wir ein Vorkommen des Patterns P gefunden, welches im Text T an Position $i-m-1$ *endet*.

Nicht-Linearzeitalgorithmus:
Führe die Vorschrift aus der letzten Teilaufgabe für jedes Feld in B aus.

Lösungsvorschlag

Aufgabe 6. Fixed Parameter Algorithmen: Kleinvereine [6 Punkte]

In dieser Aufgabe geht es darum, ein Komitee zu bestimmen, welches eine Menge von Kleinvereinen repräsentiert. Per Definition besteht ein Kleinverein aus 10 oder weniger Mitgliedern. Eine Menge von Mitgliedern wird ein Komitee von einer Menge von Kleinvereinen genannt, falls aus jedem dieser Kleinvereine mindestens ein Mitglied im Komitee vertreten ist.

Formale Definition:

Sei M die Grundmenge von Mitgliedern.
v ist ein Kleinverein falls $v \subseteq M \wedge |v| \leq 10$.
Sei V eine Menge von Kleinvereinen ($V \subseteq \mathscr{P}(M)$), dann ist K ein Komitee von V, genau dann wenn $K \subseteq M \wedge (\forall v \in V : v \cap K \neq \emptyset)$.

Beispiel: Mitglieder $M := \{m_1, m_2, m_3, m_4\}$
Menge von Kleinvereinen $V = \{\{m_1, m_2, m_3\}, \{m_1, m_3\}, \{m_2, m_4\}\}$
Ein Komitee K von V wäre $\{m_2, m_3\}$

a. Es soll nun entschieden werden, ob aus einer gegebene Menge V, bestehend aus n verschiedenen Kleinvereinen, ein Komitee bestehend aus maximal k Personen gebildet werden kann. Geben Sie einen FPT (*fixed parameter tractable*) Algorithmus an, der das Entscheidungsproblem löst. Die Laufzeit des Algorithmus soll in $O(f(k) \cdot \text{poly}(n))$ liegen, mit $f(k)$ eine berechenbare Funktion über k und $poly(n)$ ein Polynom über n.
Tipp: Verwenden Sie das Prinzip der *tiefenbeschränkten Suche*, welches in der Übung vorgestellt wurde.

[4 Punkte]

Lösung

Der folgende Algorithms löst das Entscheindungsproblem:

Algorithm 3 `Komitee(`$V = \{v_1,\ldots,v_n\}$`: Menge von Kleinvereinen ohne Vertreter im Komitee, k: unbesetzte Stühle im Komitee)`

```
if V = ∅ then
    return True
else if k = 0 then
    return False
for all Mitglieder m ∈ v_1 do                ▷ Maximaler Verzweigungsgrad 10.
    V' ← ∅
    for all v ∈ V do                         ▷ Vereine aus V ohne Verein mit Mitglied m.
        if v ∩ {m} = ∅ then
            V' ← V' ∪ {v}
    success ← Komitee(V', k − 1)
    if success then
        return True
return False
```

b. Geben Sie die (asymptotische) Laufzeit Ihrer Lösung in Abhängigkeit von k und n an und begründen Sie diese. [2 Punkte]

Lösung

Der Algorithmus baut einen Suchbaum der Tiefe k auf. Der Verzweigungsfaktor ist immer kleiner gleich der maximalen Anzahl Mitglieder in einem Kleinverein. Da ein Kleinverein aus maximal 10 Mitgliedern besteht, ist der Verzweigungsfaktor kleiner oder gleich 10. Die neue Menge von Vereinen V' kann in Zeit $O(n)$ Zeit konstruiert werden, da die Größe von jedem der n Vereinen durch eine Konstente (hier 10) beschränkt ist. Damit ergibt sich eine asympotische Laufzeit von $O(10^k \cdot n)$.

Lösungsvorschlag

Aufgabe 7. Punkte auf dem Einheitskreis [3 Punkte]

a. Gegeben sei eine Punktmenge $P \subset \{(x,y) \in \mathbb{R} \times \mathbb{R} | y^2 + x^2 = 1\}$ mit $|P| = n$.[3] Die Punkte in der Punktmenge P liegen also auf dem Einheitskreis. Geben Sie einen Algorithmus an, der für die Punktmenge P in $O(n \log n)$ Zeit zwei Punkte mit kleinstem Abstand berechnet. [2 Punkt]

Lösung

Lösung 1: Zuerst sortieren wir die Punkte $p_j = (x_j, y_j) \in P$ nach ihrer Winkelkoordinate $\phi_j = arg(x_j + iy_j)$. Danach vergleichen wir jedes benachbarte Punktepaar und merken uns dasjenige Punktepaar (p_k, p_{k+1}), welches den geringsten Abstand d_k zueinander hat. Falls d_k kleiner ist als der Abstand zwischen dem ersten und dem letztem Punkt, (p_1, p_n), in der sortierten Folge, so geben wir (p_k, p_{k+1}) aus, andernfalls (p_1, p_n).
Lösung 2: Teile Punktemenge P in $P_{\leq} := \{y \leq 0 | (x,y) \in P\}$ und $P_{>} := \{y > 0 | (x,y) \in P\}$ Sortiere beide Punktmengen nach x-Koordinate. Bestimme den kleinsten Abstand eines Punktpaares (p_k, p_{k+1}) in $P_{\leq}$ und $P_{>}$. Berechne zusätzlich den Abstand des letzten Punktes in der sortierten Folge $P_{>}$ und ersten Punktes in $P_{\leq}$, sowie des lezten Punktes in $P_{\leq}$ und ersten Punktes in $P_{>}$. Gebe das Punktpaar mit kleinster Distanz aus.
Lösung 3: Sortiere Punkte nach x-Koordinate. Berechne für jeden Punkt p_k die Distanz zu seinem Nachfolger p_{k+1} und seinem Nachnachfolger p_{k+2} in der sortierten Liste; beachte, dass der Nachfolger von Punkt p_n Punkt p_1 ist. Gebe das Punktpaar mit kleinster Distanz aus.

b. Gegeben sei wieder eine Punktmenge $P \subset \{(x,y) \in \mathbb{R} \times \mathbb{R} | y^2 + x^2 = 1\}$ mit $|P| = n$.[3] Skizzieren Sie kurz einen **effizienten parallelen Algorithmus**, der für die Punktmenge P zwei Punkte mit kleinstem Abstand berechnet. [1 Punkt]

Lösung

Sei p die Anzahl an Threads/Prozessen. Zuerst sortieren wir die Punkte $x_j = (x_j, y_j) \in P$ nach ihrer Winkelkoordinate $\phi_j = arg(x_j + iy_j)$ parallel. Danach teilen wir die sortierte Folge in p gleich große Segmente auf. Nun suchen wir parallel in jedem Segment dasjenige benachbarte Punktepaar mit kleinstem Abstand. In diese Suche wird auch jeweils der erste Knoten im nächsten Segment mit einbezogen. Die Suche im letzte Segment bezieht den ersten Knoten im ersten Sement mit ein. Bisher wurden für jedes Segment (und dem folgenden Element) die zwei Punkte mit kleinstem Abstand berechnet. Zuletzt führen wir eine parallele Reduktion auf den potentiellen Kandidaten aus und finden dadurch die zwei Punkte mit kleinstem Abstand in der Puntkemenge P.
Die anderen Lösungen aus Teilaufgabe a) lassen sich nach dem gleichen Prinzip parallelisieren.
Alternative Lösungen:

- Vergleich mit p^2 Prozessoren und Reduktion ($O(\log p)$)
- Vergleich mit p^2 Prozessoren und schreiben der Distanz mit CRCW-combine und Min-Operator ($O(1)$)

[3] Nehmen Sie an, dass reelle Zahlen in einem Maschinenwort repräsentiert werden können und keine Rundungsfehler auftreten.