Name:

Vorname:

Matrikelnummer:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders ID 1.3.2012

Klausur Algorithmen II

Aufgabe 1.	Kleinaufgaben	14 Punkte
Aufgabe 2.	Kürzeste Wege: A* Suche	8 Punkte
Aufgabe 3.	Parallele Algorithmen: Strahlenbelastung	12 Punkte
Aufgabe 4.	String Algorithmen: `abababba`	7 Punkte
Aufgabe 5.	Geometrische Algorithmen: Maximum von Funktionen	11 Punkte
Aufgabe 6.	Fixed Parameter Algorithmen: Summer School	8 Punkte

Bitte beachten Sie:
– Als Hilfsmittel ist nur **ein** DIN-A4 Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
– **Schreiben** Sie auf **alle** Blätter Ihren Namen und Ihre Matrikelnummer.
– Merken Sie sich Ihre Klausur-ID für den Notenaushang.
– Die Klausur enthält **16 Blätter**.
– Zum Bestehen der Klausur sind 20 Punkte hinreichend.

Lösungsvorschlag

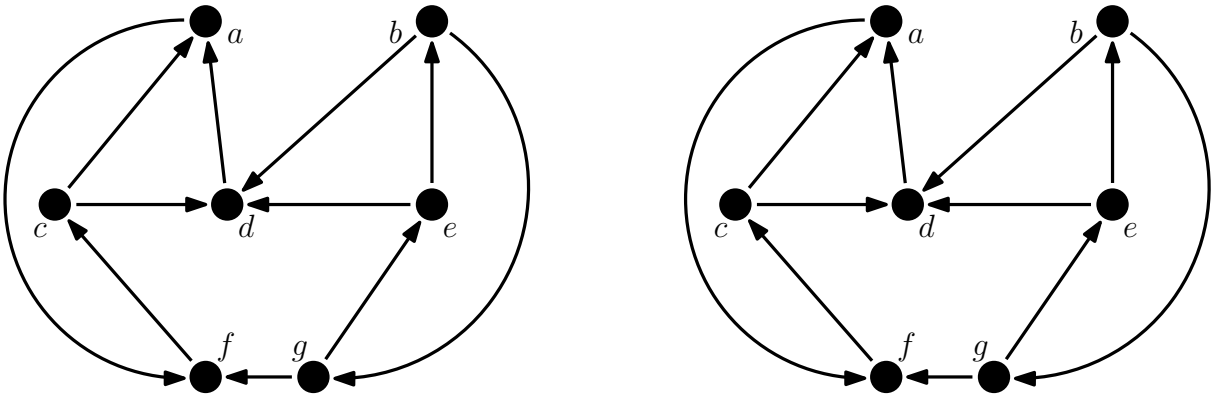
Aufgabe 1. Kleinaufgaben [14 Punkte]

a. Zeigen Sie, dass eine vergleichsbasierte *Priority Queue* die beiden Operationen `insert` und `deleteMin` nicht gleichzeitig in amortisiert $O(\log\log n)$ unterstützen kann. [1 Punkt]

Lösung

Könnten beide Operationen gleichzeitig in amortisiert $O(\log\log n)$ abgearbeitet werden, könnte man sich einen vergleichsbasierten Sortieralgorithmus konstruieren, der $O(n\log\log n)$ Laufzeit benötigt: Einfügen aller n Elemente in die Datenstruktur mit anschließendem –sortierten– Herausnehmen. Allerdings ist $\Omega(n\log n)$ die untere Schranke für die Laufzeit eines solchen Sortieres und damit sind die genannten Laufzeiten für `insert` und `deleteMin` nicht möglich.

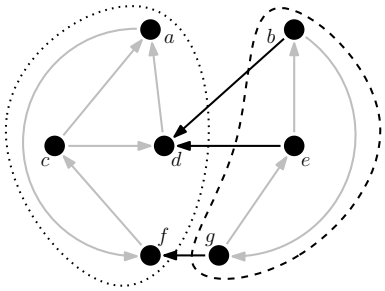
b. Gegeben sei folgender Graph (zwei Kopien):



Markieren Sie die starken Zusammenhangskomponenten (SCCs) des Graphen (1 Punkt). Kann man durch Umdrehen einer Kante erreichen, dass der Graph nur noch aus einer SCC 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

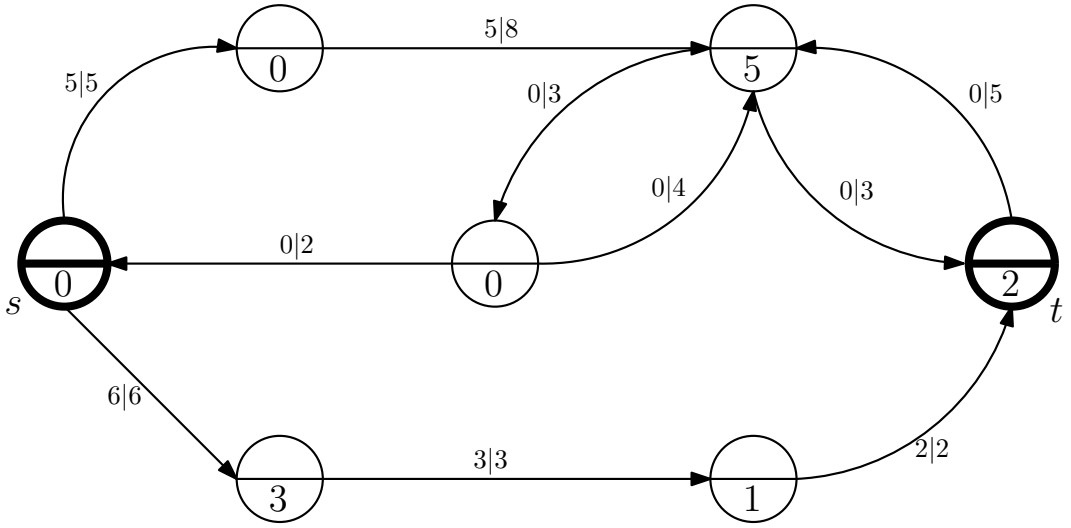


Der Graph besteht aus zwei SCCs – hier durch gestrichelte und gepunktete Umrandungen angedeutet. Dazwischen gibt es drei gerichtete Kanten (b,d), (e,d), (g,f). Das Umkehren einer dieser drei Kanten ermöglicht es, den Graphen auf eine SCC zu reduzieren.

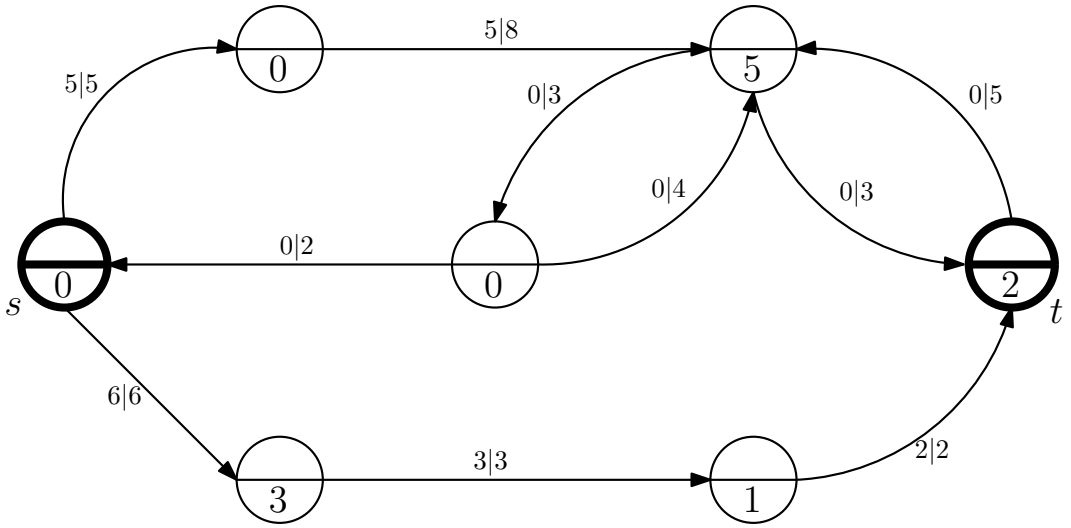
c. Gegeben sei der folgende (unvollständige) Flussgraph. Er beschreibt einen Zwischenzustand des *preflow-push* Algorithmus. Knoten sind mit ihrem momentanen Überschuss im unteren Halbkreis beschriftet, Kanten mit ihrem aktuellen Fluss und ihrer Kapazität. Quelle s befindet sich auf der linken, Senke t auf der rechten Seite des Graphen (beide fett markiert).

Weisen Sie allen Knoten gültige Level zu, markieren Sie die aktiven Knoten und zeichnen Sie die Residualkanten mit Restkapazität sowie deren Kapazitäten ein.

Flussgraph:

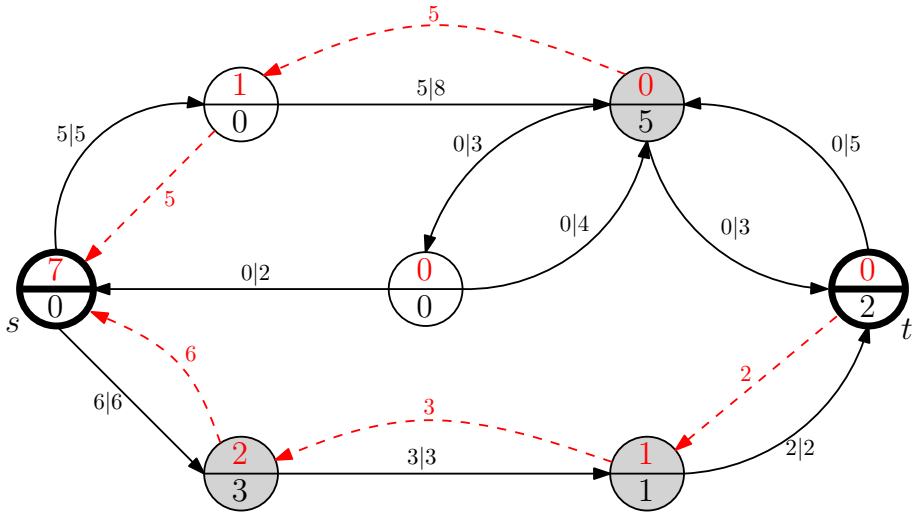


Kopie:



Falls Sie beide Graphen beschriften, markieren Sie bitte, welcher gewertet werden soll.
[3 Punkte]

Lösung



Aktive Knoten sind grau hinterlegt. Einzutragende Residualkanten sind gestrichelt gezeichnet. Knotengrade geben eine der möglichen korrekten Belegungen an.

d. Direkt nach der Rückkehr von seiner fünften Skiausfahrt entscheidet sich ein Algorithmiker, eine eigene Ausrüstung zu kaufen. Der Einkaufspreis für eine Skiausrüstung liege bei 400 Euro, der Mietpreis für eine Ausfahrt bei 40 Euro. Wie groß ist der kompetitive Faktor für diese Strategie gegenüber einer optimalen Strategie? Begründen Sie Ihre Antwort kurz. [2 Punkte]

Lösung

Die angegebene Strategie benötigt 400 Euro $+ 5 \cdot 40$ Euro $= 600$ Euro. Der schlimmste Fall tritt ein, wenn nach dem Kauf der Ausrüstung keine weitere Ausfahrt mehr stattfindet. Die optimale Strategie hätte sich die Ausrüstung immer geliehen und insgesamt $5 \cdot 40$ Euro $= 200$ Euro ausgegeben. Damit ist der kompetitive Faktor $c = 600/200 = 3$.

e. 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 keines von beiden vorliegt. Begründen Sie Ihre Antworten kurz.

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

[3 Punkte]

Lösung

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. $f(n) = n^2 + \log n$, $g(n) = 1 + \varepsilon$:
   FPTAS, da $(1+\varepsilon)$-Approximation und Laufzeit polynomiel in n und $\frac{1}{\varepsilon}$.
   (genauer: Laufzeit sogar unabhängig von ε)
2. $f(n) = \sqrt{\log(\frac{1}{\varepsilon} + n)}$, $g(n) = 1 - \varepsilon$:
   FPTAS, da $(1-\varepsilon)$-Approximation und Laufzeit polynomiell in n und $\frac{1}{\varepsilon}$.
   (genauer: $f(n) = \frac{1}{2}\log\frac{1}{\varepsilon} \cdot \log n = O(\frac{1}{\varepsilon} \cdot n)$)
3. $f(n) = n^3 + 2$, $g(n) = \log n$:
   Weder PTAS noch FPTAS, da Approximationsgüte von n abhängt.

f. Aus der Vorlesung ist Ihnen der externe Stack bekannt. Erläutern Sie, wie nur mit Hilfe von externen Stacks eine externe Queue möglichst effizient realisiert werden kann (2 Punkte). Wie viele I/O Operationen werden (amortisiert) pro `push` / `pop` benötigt (1 Punkt)? [3 Punkte]

Lösung

Eine Queue kann durch zwei Stacks realisiert werden. Bei einem `push` werden die Elemente auf den ersten Stapel gelegt. Bei einem `pop` muss zwischen zwei Fällen unterschieden werden. Ist der zweite Stapel leer werden alle Elemente des ersten Stapels per `pop` vom ersten Stapel geholt und auf den zweiten per `push` gelegt. Danach kann, genau wie in dem Fall, dass der Stapel nicht leer war, per `pop` das vorderste Element vom zweiten Stapel genommen werden. Dabei wird jedes Element einmal auf den ersten, einmal auf den zweiten Stapel gelegt und von jedem Stapel einmal heruntergenommen. Somit ergeben sich amortisiert weiterhin Kosten von $O(1/B)$ I/O Operationen pro `push` / `pop`.

Lösungsvorschlag

Aufgabe 2. Kürzeste Wege: A* Suche [8 Punkte]

Im Folgenden sollen verschiedene theoretische und praktische Teilaspekte von Kürzeste Wege Algorithmen anhand der A* Suche näher betrachtet werden:

a. Die A* Suche benötigt eine Potentialfunktion, die jedem Knoten einen Wert zuweist. Aus der Vorlesung ist Ihnen bekannt, dass eine Funktion p zwei Bedingungen erfüllen muss, um eine gültige Potentialfunktion zu sein.

Diese lauten

- $\forall u \in V : p(u) \leq \mu(u,t)$, (mit $\mu(u,t)$ kürzeste Distanz von u nach t)
- $\forall (u,v) \in E : c(u,v) + p(v) \geq p(u)$

wobei $G = (V,E)$ der betrachtete Graph und $t \in V$ das Ziel der Suche ist.

Zeigen Sie, sind p_1 und p_2 gültige Potentialfunktionen, so ist

$$p_{\max} := \max\{p_1, p_2\}$$

ebenfalls eine gültige Potentialfunktion. [3 Punkte]

Lösung

Einfach zu sehen ist, dass das Maximum $p_{\max}$ zweier unterer Schranken p_1 und p_2 weiterhin eine untere Schranke bildet.

Für die zweite Bedingung machen wir folgenden Ansatz:

$$\begin{aligned} c(u,v) + p_{\max}(v) &= c(u,v) + \max\{p_1(v), p_2(v)\} \\ &= \max\{c(u,v) + p_1(v), c(u,v) + p_2(v)\} \\ &\geq \max\{p_1(u), p_2(u)\} \\ &= p_{\max}(u). \end{aligned}$$

b. Wie Dijkstras Algorithmus benötigt auch die A* Suche eine *Priority Queue* Datenstruktur. Diese kann zum Beispiel durch einen *Radix Heap* implementiert werden. Betrachten wir nun folgenden bereits teilweise gefüllten *Radix Heap*:

Der aktuelle Zustand des *Radix Heaps* mit $C = 12$ [01100] und $min = 11$ [01011] enthalte die (Wert, Schlüssel) Paare $(a, 12$ [01100]), $(b, 16$ [10000]), $(c, 18$ [10010]), $(d, 19$ [10011]), $(e, 23$ [10111]). Geben Sie den Zustand des Heaps an (1 Punkt).

`Ausgangszustand:`

-1	0	
∅		

min = 11 [01011]

Führen Sie die folgenden Operationen auf dem Heap aus:
Achtung: Verwenden Sie die `deleteMin()` Implementierung aus dem Buch/der Übung (s.h. Blatt 19!)

- `deleteMin()`
- `deleteMin()`
- `decreaseKey(e,17)`
- `insert(f,16)`

Geben Sie den Zustand der Datenstruktur nach jeder Operation an. Füllen Sie die vorbereiteten Vorlagen dafür vollständig aus (4 Punkte).

`deleteMin():`

-1	0	

min =

`deleteMin():`

-1	0	

min =

`decreaseKey(e,17):`

-1	0	

min =

`insert(f,16):`

-1	0	

min =

Falls Sie zusätzlichen Platz benötigen:

-1	0	

min = [5 Punkte]

Lösung

$C = 12$ bedeutet $(\lfloor \log 12 \rfloor + 2) + 1 = 6$ Buckets. Der Ausgangszustand des *Radix Heaps* ist also wie folgt:

-1	0	1	2	3	4
			(a, 12)		(b, 16), (c, 18), (d, 19), (e, 23)

min = 11 [01011]

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

`deleteMin():`

-1	0	1	2	3	4
					(b, 16), (c, 18), (d, 19), (e, 23)

min = 12 [01100]

`deleteMin():`

-1	0	1	2	3	4
		(c, 18), (d, 19)	(e, 23)		

min = 16 [10000]

`decreaseKey(e, 17):`

-1	0	1	2	3	4
	(e, 17)	(c, 18), (d, 19)			

min = 16 [10000]

`insert(f, 16):`

-1	0	1	2	3	4
(f, 16)	(e, 17)	(c, 18), (d, 19)			

min = 16 [10000]

Lösungsvorschlag

Aufgabe 3. Parallele Algorithmen: Strahlenbelastung [12 Punkte]

In einem weitläufigen Gebiet soll die Strahlenbelastung gemessen und ausgewertet werden. Hierfür werden n Meßstellen in einem gleichmäßigen Gitter der Größe $\sqrt{n} \times \sqrt{n}$ errichtet. Jede Meßstelle kann die Strahlenbelastung in seiner Gitterzelle bestimmen. Die Meßwerte werden zentral gesammelt und in einer Matrix B gespeichert.
Zur Auswertung benötigt man die summierten Strahlenbelastungen in beliebigen, rechteckigen Teilgebieten des überwachten Gebiets. Diese sollen mit Hilfe eines großen Parallelrechners mit p Prozessoren effizient berechnet werden.

a. Geben Sie einen *sequentiellen* Linearzeitalgorithmus an, der die aufsummierten Strahlenbelastungen S von **allen** rechteckigen Teilgebieten der Form $(0,0)$:(x,y), für $x,y \in \{0,\ldots,\sqrt{n}\}$, aus den Meßwerten in Matrix B berechnet (3 Punkte). Begründen Sie, warum die lineare Laufzeit optimal ist (1 Punkt).

Die Notation (a,b):(c,d) beschreibt ein rechteckiges Teilgebiet des Gitters durch die beiden gegenüberliegenden Eckpunkte (a,b) und (c,d).

Hinweis: Durchlaufen Sie Matrix B strukturiert beginnend bei $(0,0)$. [4 Punkte]

Lösung

Jedes Element muss mindestens einmal betrachtet werden. Folglich ist die bestmögliche Laufzeit linear in n. Für einen solchen Algorithmus scannen wir B von $(0,0)$ ausgehend zeilenweise.

Wir definieren $S((0,0):(-1,y)) = S((0,0):(x,-1)) := 0$ für alle möglichen Werte von x,y. Mit diesen *Sentinel* Werten gilt:

$$S((0,0):(x,y)) = S((0,0):(x-1,y)) + S((0,0):(x,y-1)) - S((0,0):(x-1,y-1)) + B(x,y).$$

Durch den linearen Scan sind alle benötigten Werte für $S((0,0):(x,y))$ bereits berechnet. Somit ergibt sich ein Linearzeitalgorithmus, der damit optimal ist.

b. Entwerfen Sie einen *parallelen* Algorithmus, der für hinreichend große n in Zeit $O(n/p)$ die Werte aus Teilaufgabe **a.** berechnet. [4 Punkte]

Lösung

Bilde über jede Zeile der Wertematrix B die Präfixsumme. Dies geschieht für eine Zeile in $O(\sqrt{n}/p + \log p)$, für alle Zeilen in $\sqrt{n}$ facher Zeit. Anschließend bilde über die erhaltenen Werte spaltenweise die Präfixsumme. Auch dies geschieht für eine Spalte in $O(\sqrt{n}/p + \log p)$ und für alle Spalten in $\sqrt{n}$ facher Zeit. Insgesamt ergibt sich die Laufzeit zu $O(n/p + \sqrt{n}\log p)$. Für große n dominiert der erste Term und man erhält die gesuchte Laufzeit $O(n/p)$.

c. Geben Sie Speedup und Effizienz Ihres parallelen Algorithmus als Funktion von n, p an. Falls Sie für Teilaufgabe **a.** oder **b.** keine Lösung gefunden haben, verwenden Sie die vorgegebenen Laufzeitschranken. [2 Punkte]

Lösung

Speedup ist definiert als $S(p) = \frac{T_{seq}}{T(p)}$. Damit ist $S(p) = \frac{O(n)}{O(n/p)} = O(p)$.
Für die Effizienz ergibt sich $E = \frac{S(p)}{p} = O(1)$.

d. Nachdem in Teilaufgabe **a.** bzw. **b.** Strahlenbelastungen S für alle rechteckigen Teilgebiete mit Ursprung $(0,0)$ berechnet worden sind, soll diese Berechnung verallgemeinert werden:

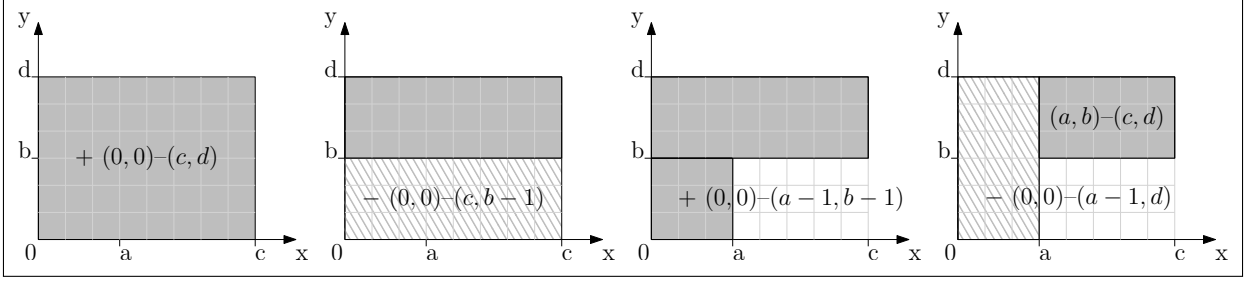
Zeigen Sie, wie in konstanter Zeit mit Hilfe der zuvor berechneten Werte, die Strahlenbelastung für ein beliebiges, rechteckiges Teilgebiet $(a,b):(c,d)$, $a,b,c,d \in \{0,\ldots,\sqrt{n}\}$, bestimmt werden kann. Geben Sie die Formel zur Berechnung direkt an. [2 Punkte]

Lösung

Die geforderte Formel entspricht dem *RangeCount* der *Wavelet Trees*:

$$S((a,b):(c,d)) = \\ S((0,0):(c,d)) - S((0,0):(a-1,d)) - S((0,0):(c,b-1)) + S((0,0):(a-1,b-1))$$

Bildlich dargestellt:



Lösungsvorschlag

Aufgabe 4. String Algorithmen: `abababba` [7 Punkte]

Der Algorithmus von Knuth, Morris und Pratt (KMP-Algorithmus) ermöglicht es, Muster p in Texten t effizient zu finden. Vor der eigentlichen Suche müssen aber zuerst einige Hilfswerte berechnet werden.

a. Tragen Sie die für den KMP-Algorithmus benötigten Hilfswerte ($\hat{=}$ *border-array*) für das Muster $p =$ `abababba` in die Vorlage ein:

p	a	b	a	b	a	b	b	a
border[]								

p	a	b	a	b	a	b	b	a
border[]								

Falls Sie beide Vorlagen beschriften, markieren Sie bitte welche gewertet werden soll. [1 Punkt]

Lösung

p	a	b	a	b	a	b	b	a
border[]	-1	0	0	1	2	3	4	0

b. Das obige Muster p werde in Text $t =$ `abbababababba` gesucht. Tragen Sie die Stellen, an denen der KMP-Algorithmus p an t anlegt, in die Vorlage ein:

	1	2	3	4	5	6	7	8	9	10	11	12	13	
t	a	b	b	a	b	a	b	a	b	a	b	b	a	
														Stelle __
														Stelle __
														Stelle __
														Stelle __
														Stelle __

	1	2	3	4	5	6	7	8	9	10	11	12	13	
t	a	b	b	a	b	a	b	a	b	a	b	b	a	
														Stelle __
														Stelle __
														Stelle __
														Stelle __
														Stelle __

Falls Sie beide Vorlagen beschriften, markieren Sie bitte welche gewertet werden soll.
[3 Punkte]

Lösung

t	a	b	b	a	b	a	b	a	b	a	b	b	a	
	a	b	a	b	a	b	b	a						Stelle 1
			a	b	a	b	a	b	b	a				Stelle 3
				a	b	a	b	a	b	b	a			Stelle 4
						a	b	a	b	a	b	b	a	Stelle 6

c. Komprimieren Sie das obige Muster p mit dem Lempel-Ziv Algorithmus (2 Punkt). Geben Sie außerdem das erzeugte Wörterbuch an (1 Punkt). [3 Punkte]

Lösung

Komprimierter String: `c = 12334`.

Erzeugtes Wörterbuch:

Index	Zeichenfolge
1	`a`
2	`b`
3	`ab`
4	`ba`
5	`aba`
6	`abb`

Lösungsvorschlag

Aufgabe 5. Geometrische Algorithmen: Maximum von Funktionen [11 Punkte]

Eine stückweise lineare, stetige Funktion f lässt sich durch eine nach x-Werten sortierte Liste von n Stützpunkten der Form $f_{support} = \{ (x_1,y_1), \ldots, (x_n,y_n) \}$ beschreiben. Der Funktionswert an Stelle x ergibt sich nach

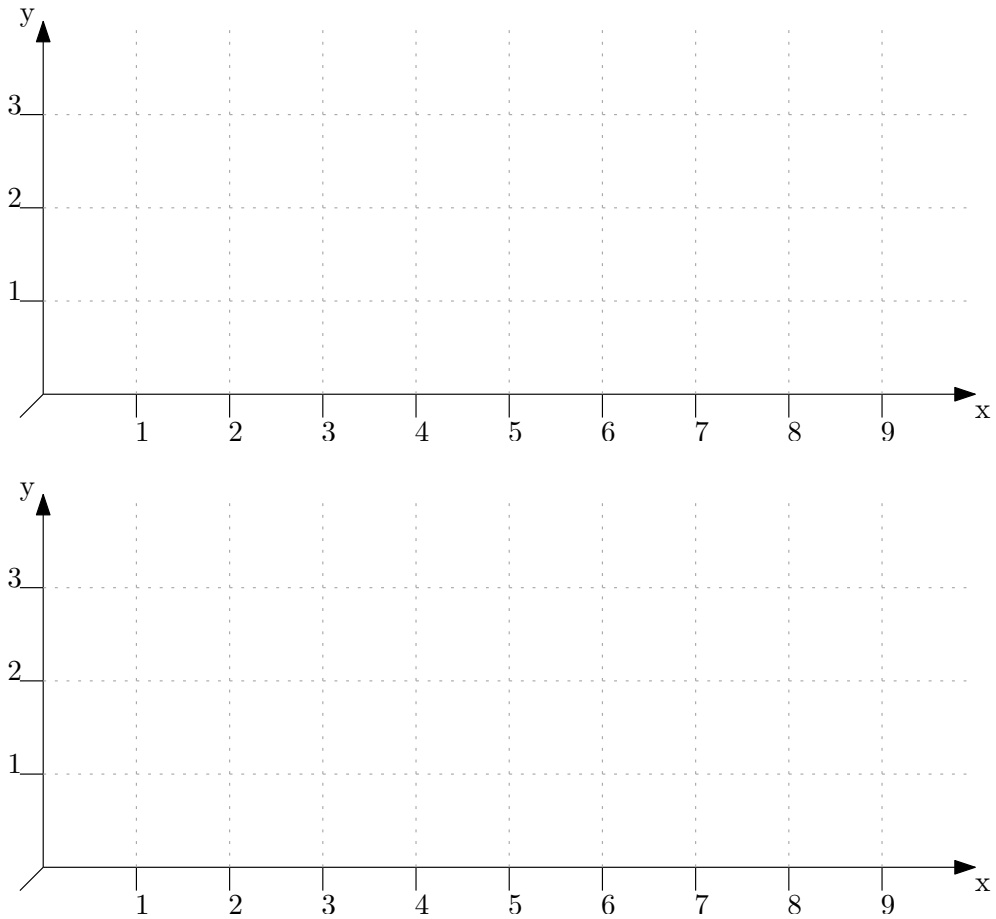
$$f(x) = \begin{cases} y_1 & x < x_1 \\ y_n & x > x_n \\ (1-\alpha)\cdot y_i + \alpha \cdot y_{i+1} & \alpha = \frac{x-x_i}{x_{i+1}-x_i},\ i \in \{1,\ldots,n-1\} \text{ für } x \in [x_i,x_{i+1}]. \end{cases}$$

Die Funktion beschreibt also Geradensegmente zwischen den gegebenen Stützpunkten.

a. Gegeben seien zwei stückweise lineare, stetige Funktion f, g mit

$f_{support} = \{ (0,0), (2,3), (4,1), (7,2), (9,0) \}$
$g_{support} = \{ (0,1), (2,1), (5,3), (6,0) \}$

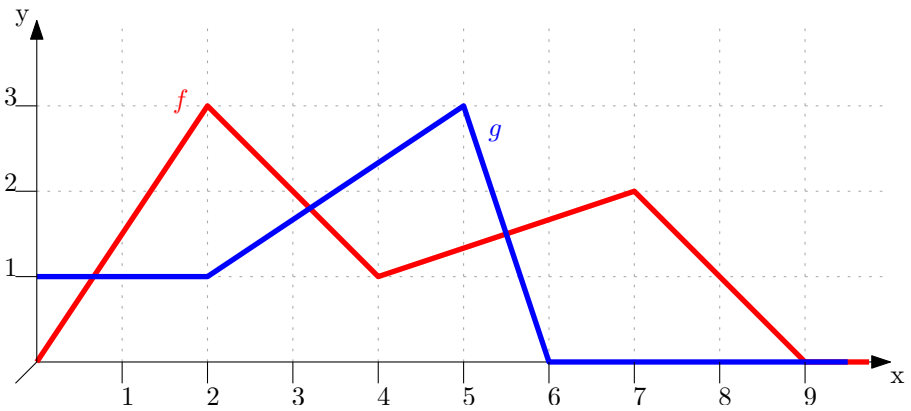
Zeichnen Sie die beiden Funktionen f,g in das obere Koordinatensystem (1 Punkt), sowie die Funktion $h := \max(f,g)$ in das untere Koordinatensystem (1 Punkt).



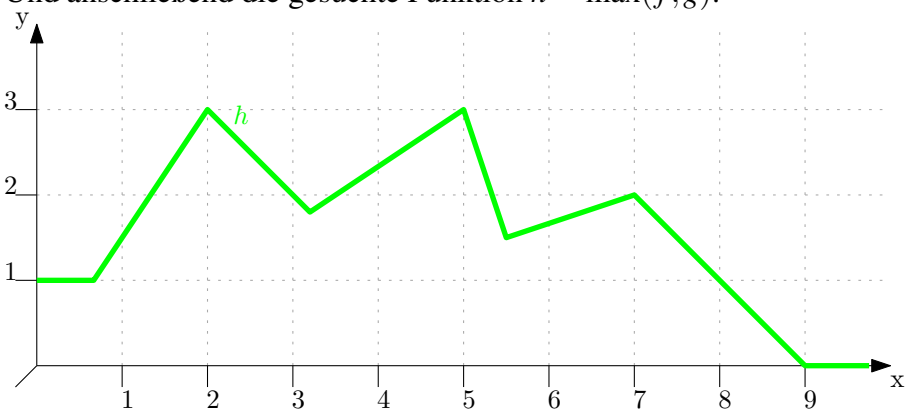
[2 Punkte]

Lösung

Der Übersichtlichkeit sind zunächst die Funktionen f und g gezeichnet:



Und anschließend die gesuchte Funktion $h = \max(f,g)$:



b. Geben Sie einen Algorithmus an, der die Stützpunkte von $h = max(f,g)$ in Zeit $O(n+m)$ berechnet (4 Punkte). Ein Algorithmus längerer Laufzeit wird mit maximal 2 Punkten bewertet. Begründen Sie die Laufzeit Ihres Algorithmus (1 Punkt).

Hinweis: Betrachten Sie die Stützpunkte in sukzessiver Reihenfolge mit wachsendem x Wert.

[5 Punkte]

Lösung

Folgender *line sweep* Algorithmus löst das Problem:
(angelehnt an den Algorithmus zum Finden von Linienschnitten)

```
MAX_PLF(f_support, g_support)
P := f_support.x ∪ g_support.x          {Betrachte Stützstellen beider Funktionen}
m = MaxFunc(f, g, P[0])                 {Wähle Funktion, die an Position P[0] größer ist}
S = (P[0], m(P[0]))                     {Füge erste Stützstellen der Maximumsfunktion ein}
for i = 1; i < P.size; i++ do
  m' = MaxFunc(f, g, P[i])              {Wähle Funktion, die an Position P[i] größer ist}
  if m' ≠ m then                        {Kreuzen sich beide Funktionen?}
    S = S ∪ CUT(f, g, P[i-1], P[i])     {Füge Schnitt von f, g in [P[i-1],P[i]] ein}
    m = m'                              {Wechsele zur aktuell größeren Funktion}
  end if
  S = S ∪ (P[i], M(P[i]))               {Füge Stützpunkt der aktuell größeren Funktion ein}
end for
return S                                {Gebe Stützstellen der Maximumsfunktion zurück}
```

Der Algorithmus läuft linear in der Anzahl Stützstellen, also in $O(n+m)$. Immer wenn sich die maximale Funktion zwischen zwei Stützstellen ändert, wird ein Schnitt eingefügt. Ansonsten genügt es, den Stützpunkt der aktuell maximalen Funktion einzufügen.

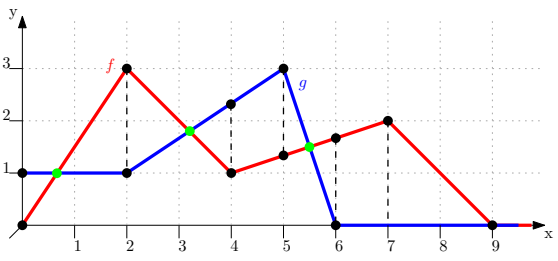
Die Funktion $MaxFunc(f,\ g,\ P[i])$ gibt die Funktion aus $\{f,g\}$ zurück, die an der Stelle $P[i]$ den größeren Wert hat. *MaxFunc* kann in amortisiert konstanter Zeit implementiert werden, wenn man sich die Position der Stützstelle in $f_{support}$ bzw. $g_{support}$ merkt, bis zu der f,g bereits betrachtet wurden.

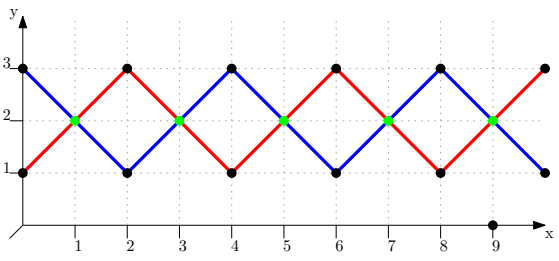
c. Gegeben seien zwei stückweise lineare, stetige Funktion f,g mit n bzw. m Stützstellen. Wieviele Stützstellen hat die Funktion $h := max(f,g)$ maximal im O-Kalkül? Geben Sie die bestmögliche Schranke an und begründen Sie Ihre Antwort (3 Punkt). Zeigen Sie insbesondere auch, dass es keine bessere Schranke geben kann (1 Punkte). [4 Punkte]

Lösung

Die Funktion $h = max(f,g)$ besitzt maximal $O(n+m)$ Stützstellen.

Begründung:
Wir geben f maximal m zusätzliche Stützstellen, indem wir an jeder Stützstelle von g eine Stützstelle für f einfügen. Genauso verfahren wir mit g. Auf diese Weise erhalten wir zwei Funktionen mit je $m+n$ Stützstellen. In jedem der so entstandenen $m+n$ Intervalle können sich die Geraden höchstens einmal schneiden. Somit erhalten wir maximal $O(m+n)$ Stützstellen. Das rechte Bild zeigt, dass keine bessere Schranke gefunden werden kann.





Lösungsvorschlag

Aufgabe 6. Fixed Parameter Algorithmen: Summer School [8 Punkte]

Für eine *Summer School* im sonnigen Sizilien bewerben sich n Studenten. Die Teilnehmerzahl ist allerdings auf k Personen beschränkt. Um die Vielfalt zu erhöhen, sollen zudem nur Personen eingeladen werden, die sich nicht kennen. Um dies zu garantieren, haben die Organisatoren bereits die Bekannten aller Bewerber mit Hilfe sozialer Netze ermittelt. Bei der Datenerhebung wurde außerdem festgestellt, dass kein Bewerber mehr als d Freunde hat, die sich auch für die *Summer School* bewerben.

Die Organisatoren stehen vor dem Problem, zu überprüfen, ob es überhaupt k Bewerber gibt, die ihre Bedingung erfüllen. Sie sind der arme Informatiker, dem diese Aufgabe zugeteilt wird.

a. Geben Sie eine graphentheoretische Formulierung des obigen Problems an. Gehen Sie auch darauf ein, welche Bedeutung Knoten und Kanten in Ihrer Darstellung haben. [2 Punkte]

Lösung

Jeder Bewerber entspricht einem Knoten im Graph. Bekanntschaften werden durch eine Kante symbolisiert. Gesucht ist ein *Independent Set* mit k Knoten, d.h. eine Teilmenge aller Knoten, die paarweise nicht benachbart sind.

b. Geben Sie einen FPT (*fixed parameter tractable*) Algorithmus an, der das Problem löst, d.h. der entscheidet, ob es eine Gruppierung aus genau k Bewerbern gibt, von denen keiner einen anderen dieser Bewerber kennt. Die Laufzeit des Algorithmus soll in $O(f(d,k)\cdot \text{poly}(n))$ liegen, mit $f(d,k)$ eine berechenbare Funktion über d,k und $poly(n)$ ein Polynom über n. [4 Punkte]

Lösung

Verwende eine tiefenbeschränkte DFS um ein *Independent Set* (IS) aus k Bewerbern zu bauen: Wähle Knoten v beliebig. Verzweige, indem v oder einen seiner Nachbarn zum IS hinzugefügt wird (einer dieser Knoten ist immer Teil eines *Maximum Independent Set*, kann also für das IS beschränkter Größe verwendet werden!). Entferne den hinzugefügten Knoten und seine Nachbarn aus dem Graph. Wiederhole bis k Knoten im IS sind (Erfolg) oder kein Knoten mehr vorhanden ist (Fehlschlag).

Bemerkung: Durch die beliebige Wahl von v ist das IS nicht eindeutig bestimmt!

c. Geben Sie die (asymptotische) Laufzeit Ihrer Lösung in Abhängigkeit von d,k,n an (1 Punkt), und begründen Sie diese (1 Punkt).

[2 Punkte]

Lösung

Der Algorithmus baut einen Suchbaum der Tiefe k. Der Verzweigungsfaktor ist immer kleiner gleich der maximalen Anzahl Nachbarn d plus eins. Der Umbau des Graphen kostet lineare Zeit. Damit ergbit sich eine asympotische Laufzeit von $O((d+1)^k \cdot n)$.