Nachname:

Vorname:

Matrikelnummer:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders 14.03.2024

Klausur Algorithmen II

Aufgabe 1.	Kleinaufgaben	9 Punkte
Aufgabe 2.	Stringology: Burrows-Wheeler-Transformation	10 Punkte
Aufgabe 3.	Parametrisierte Algorithmen: Closest String	10 Punkte
Aufgabe 4.	Externe Algorithmen: Crowd-Computing	10 Punkte
Aufgabe 5.	Approximationsalgorithmen: Mehrdimensionaler Rucksack	12 Punkte
Aufgabe 6.	Online-Algorithmen: Unkrautbekämpfung	9 Punkte

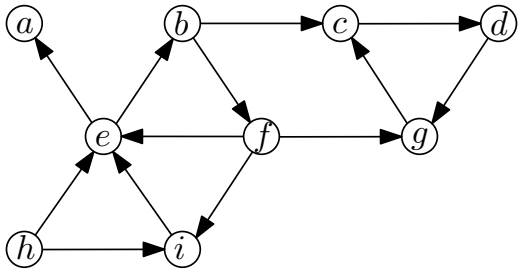
Bitte beachten Sie:

- 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 **15 Seiten**.
- Zum Bestehen der Klausur sind 30 Punkte hinreichend.

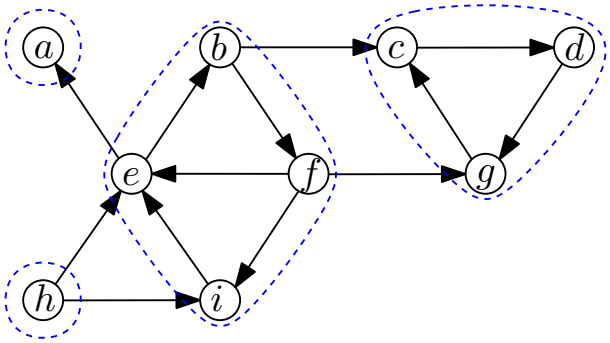
Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [9 Punkte]

a. Markieren Sie im unten abgebildeten Graphen alle starken Zusammenhangskomponenten (SCCs). Kann man durch Hinzufügen einer einzigen Kante erreichen, dass der Graph aus einer einzigen SCC besteht? Falls ja, geben Sie eine solche Kante an. Falls nein, begründen Sie. [2 Punkte]



Lösung



Nein, es kann keine Kante hinzugefügt werden, so dass der Graph nur eine einzelne SCC hat. Das sieht man z.B. daran, dass der Schrumpfgraph keinen Pfad hat, der alle Knoten enthält. Somit kann eine einzelne Kante keinen Kreis induzieren, der alle SCCs enthält.

b. Sortieren Sie die aus der Vorlesung bekannten Algorithmen für die folgenden Probleme aufsteigend nach erwarteter asymptotischer Laufzeit in Abhängigkeit von n. Alle Punkte bzw. Strecken liegen jeweils in der Ebene.

- *RangeCount*-Query auf einem bereits konstruierten Wavelet-Tree mit n Punkten
- Berechnung von *Streckenschnitten* für n Strecken mit $k \in \Theta(n)$ Schnittpunkten
- Berechnung der kleinsten einschließenden Kugel (*smallestEnclosingBall*) für n Punkte

[2 Punkte]

Lösung

- *RangeCount* (Laufzeit $\mathcal{O}(\log n)$)
- *smallestEnclosingBall* (erwartete Laufzeit $\mathcal{O}(n)$)
- *Streckenschnitte* (Laufzeit $\mathcal{O}(n \log n)$)

c. Gegeben seien zwei parallele Algorithmen A und B, die dasselbe Problem lösen. Sei n die Eingabegröße und p die Anzahl verfügbarer PEs. Der beste sequentielle Algorithmus habe Laufzeit $T_{\text{seq}}(n) = n$. Die Laufzeiten der parallelen Algorithmen seien:

$$T_A(n,p) = \frac{n \log n}{p} \qquad \text{bzw.} \qquad T_B(n,p) = \frac{n}{\sqrt{p}}$$

Geben Sie die Effizienz der beiden Algorithmen an. Wie viele PEs müssen in Abhängigkeit von n mindestens vorliegen, damit Algorithmus A effizienter ist als Algorithmus B? [3 Punkte]

Lösung

Algorithmus	Speedup	Effizienz
A	$\frac{p}{\log n}$	$\frac{1}{\log n}$
B	$\sqrt{p}$	$\frac{1}{\sqrt{p}}$

Algorithmus A ist effizienter als Algorithmus B, falls

$$E_A(n,p) > E_B(n,p) \iff \frac{1}{\log n} > \frac{1}{\sqrt{p}} \iff \sqrt{p} > \log n \iff p > (\log n)^2$$

d. Gegeben sei der folgende Zustand eines Radix-Heaps. Geben Sie das entfernte Element und den Zustand des Heaps nach einer sowie nach zwei `deleteMin()`-Operationen an. [2 Punkte]

Bucket ID	-1	0	1	2	3
Radix Heap	0010_b			0110_b 0101_b 0100_b 0111_b	1110_b

Lösung

Nach erstem `deleteMin()`:

Entferntes Element: 0010_b

Bucket ID	-1	0	1	2	3
Radix Heap				0110_b 0101_b 0100_b 0111_b	1110_b

Nach zweitem `deleteMin()`:

Entferntes Element: 0100_b

Bucket ID	-1	0	1	2	3
Radix Heap		0101_b	0110_b 0111_b		1110_b

Lösungsvorschlag

Aufgabe 2. Stringology: Burrows-Wheeler-Transformation [10 Punkte]

a. Geben Sie alle zyklischen Rotationen der Strings $S_2 = \texttt{b\$}$, $S_3 = \texttt{ab\$}$, $S_4 = \texttt{dab\$}$ und $S_5 = \texttt{cdab\$}$ an und sortieren Sie diese, um die Burrows-Wheeler-Transformation zu berechnen. Markieren sie die Burrows-Wheeler-Transformation.

Zwei Kopien für jede Größe. Wenn Sie mehr als eine Tabelle einer Größe beschriften, machen Sie deutlich, welche Kopie korrigiert werden soll. Andernfalls wird diese Teilaufgabe mit 0 Punkten bewertet. [4 Punkte]

Lösung

b	$
$	b

$	b
b	$

a	b	$
b	$	a
$	a	b

$	a	b
a	b	$
b	$	a

d	a	b	$	
a	b	$	d	
b	$	d	a	
$	d	a	b	

$	a	b	d	
d	b	$	a	
a	$	d	b	
b	d	a	$	

c	d	a	b	$
d	a	b	$	c
a	b	$	c	d
b	$	c	d	a
$	c	d	a	b

$	a	b	c	d
c	b	$	d	a
d	$	c	a	b
a	c	d	b	$
b	d	a	$	c

Wir möchten nun einen in-place Algorithmus für die Konstruktion der Burrows-Wheeler-Transformation entwerfen, welcher die BWT schrittweise erweitert. Gehen Sie im Rest dieser Aufgabe davon aus, dass alle Zeichen des Textes paarweise verschieden sind.

Gegeben sei ein Text T und die Burrows-Wheeler-Transformation des Textes $BWT(T)$. Nun fügen wir ein Zeichen α vorne an den Text an und erhalten $T' = \alpha T$. Wir wollen die folgenden beiden Schritte vervollständigen, sodass sie $BWT(T)$ zu $BWT(T')$ überführen:

1. Ersetze ein Zeichen x in $BWT(T)$ durch α.
2. Füge das Zeichen x zwischen $BWT(T)[i-1]$ und $BWT(T)[i]$ neu ein.

b. Welches Zeichen x in der BWT wird durch α ersetzt und warum handelt es sich dabei immer um dieses Zeichen? [2 Punkte]

Lösung

Da α vorn am Text angehängt wird, bleiben alle existierenden Suffixe von T erhalten. Die Reihenfolge der existierende Suffixe ändert sich nicht. Die zyklischen Rotationen jedes existierenden Suffix werden um das Zeichen α nach dem Sentinel \$ ergänzt, denn α steht an erster Stelle von T' also zyklisch nach \$. Damit steht α in der BWT von T' also nun an der Stelle, wo zuvor das Sentinel $x := \$$ stand.

c. Wie kann der gesuchte Index i für das erneute Einfügen von x ermittelt werden?

Hinweis: Denken Sie daran, dass alle Zeichen paarweise verschieden sind. [2 Punkte]

Lösung

Neben den existierenden Suffixen kommt genau ein neues Suffix, nämlich T', hinzu. Der Rang von T' bzgl. der lexikographischen Ordnung der Suffixe von T' (entspricht $SA(T')[1]$) bestimmt die gesuchte Position i des Sentinels \$ in der BWT. Da alle Zeichen paarweise unterschiedlich sind, lässt sich der Rang bestimmen, indem man jene Elemente in $BWT(T)$ zählt, die lexikographisch kleiner als α sind.

d. Entwerfen Sie einen in-place Konstruktionsalgorithmus, der die Burrows-Wheeler-Transformation in $\mathcal{O}(n^2)$ Zeit konstruiert. Der Algorithmus darf lediglich den Text überschreiben und konstant viel zusätzlichen Platz benötigen. Begründen Sie, warum Ihr Algorithmus die geforderte Laufzeit und den geforderten Platzbedarf erfüllt. [2 Punkte]

Lösung

Die Grundidee ist, die Burrows-Wheeler-Transformation von rechts nach links zu konstruieren und dabei den Text zu überschreiben. In jeder Iteration fügen wir ein Zeichen zu der Burrows-Wheeler-Transformation hinzu. Es steht also $T[1..n-i] \cdot BWT(T[n-i..n])$ in der i-ten Iteration im Text. Wir können mit $i = 3$ beginnen, da die Burrows-Wheeler-Transformation für $T[n-1..n]$ das Suffix selber ist.

In jeder Iteration müssen wir zunächst die Position des neuen Zeichens $c = T[i]$ in der schon konstruierten Burrows-Wheeler-Transformation finden. Nach Teilaufgabe **b.** entspricht dies der Position des Sentinels \$ in der bekannten BWT $BWT(T[n-i+1..n])$. Die Position lässt sich also in $\mathcal{O}(n)$ finden.
Anschließend müssen wir noch die neue Position des Sentinels finden. Nach Teilaufgabe **c.** kann man diese Position bestimmen, indem man alle Elemente in $BWT(T[n-i+1..n])$ zählt, welche lexikographisch kleiner als c sind. Sowohl das Zählen, als auch das Einfügen des Sentinels an der richtigen Position (durch Verschieben der Elemente links vom neuen Sentinel) sind in $\mathcal{O}(n)$ möglich.
Alle Schritte benötigen insgesamt $\mathcal{O}(n)$ Zeit. Da wir die Schritte $\mathcal{O}(n)$ mal ausführen müssen ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(n^2)$.

Lösungsvorschlag

Aufgabe 3. Parametrisierte Algorithmen: Closest String [10 Punkte]

Gegeben sei eine Menge $S = \{s_1, \ldots, s_m\}$ von m Strings der Länge n über einem Alphabet $\mathscr{A}$. Für zwei Strings x, y bezeichnen wir mit $d(x,y) \coloneqq |\{j \in \{1,\ldots,n\} \colon x[j] \neq y[j]\}|$ die Anzahl Stellen, in denen sich x und y unterscheiden (Hamming-Distanz).

Das Problem Closest String fragt für einen gegebenen Parameter k, ob ein String $z \in \mathscr{A}^n$ existiert, der zu jedem Eingabestring höchstens Distanz k hat (also $d(s_i,z) \leq k$ für $1 \leq i \leq m$).

Wir betrachten einen Algorithmus, der Closest String mittels tiefenbeschränkter Suche löst. Gegeben sei der unten abgebildete Pseudocode für den Algorithmus. Hierbei wird initial $x = s_1$ und $i = 0$ übergeben. Der Rückgabewert $\perp$ bedeutet, dass die Instanz keine Lösung hat.

Algorithmus 1 `closestStringRecursion` $(S = \{s_1,\ldots,s_m\},\ k,\ x \in \mathscr{A}^n,\ i \leq k)$

```
1  if ∀s ∈ S : d(x,s) ≤ k then
2      return x
3  if i = k then
4      return ⊥
5
6  s ← s ∈ S mit d(x,s) > k                          ▷ wähle s mit minimalem Index
7  J ← {j_1, ... j_{k+1}} so dass x[j_ℓ] ≠ s[j_ℓ] für alle ℓ ≤ k+1   ▷ wähle die j_ℓ minimal
8  for ℓ = 1 to k+1 do
9      x' ← x
10     x'[j_ℓ] ← s[j_ℓ]                              ▷ neuer Lösungskandidat
11     z ← closestStringRecursion(S, k, x', i+1)
12     if z ≠ ⊥ then
13         return z
14 return ⊥
```

a. Führen Sie den Algorithmus auf der im Folgenden gegebenen Instanz für $k = 2$ aus. Wählen Sie dabei in Zeile 6 immer den String mit minimalem Index (d.h. $s = s_i$ für das kleinste i mit $d(x,s_i) > k$) und tragen Sie den gewählten String als $s = s_1/s_2/s_3$ ein. Wählen Sie die Indizes in Zeile 7 immer minimal möglich und in aufsteigender Reihenfolge. Tragen Sie alle Werte ein, die x' während der Ausführung annimmt (nach der Zuweisung in Zeile 10).

Prüfen Sie jeweils direkt, ob der Wert für x' bereits eine gültige Lösung ist. Falls darauf noch rekursive Aufrufe folgen, die nicht mehr ausgeführt werden, lassen Sie diese unausgefüllt. Markieren Sie den String, der vom Algorithmus zurückgegeben wird.

Zwei Kopien. Wenn Sie beide beschriften, machen Sie deutlich, welche Kopie korrigiert werden soll. Andernfalls wird diese Teilaufgabe mit 0 Punkten bewertet. [4 Punkte]

Instanz ($k = 2$)

$s_1 =$ b a c b a $\quad$ $s_2 =$ c a a b b $\quad$ $s_3 =$ b b a c c

Lösung

Initialer Aufruf mit $i = 0$ $\quad x =$ b a c b a $\quad s = s_2$

$x' =$ | c | a | c | b | a | , | b | a | a | b | a | , | *b* | *a* | *c* | *b* | *b* |

Rekursion mit $i = 1, \ell = 1$ $\quad x =$ | c | a | c | b | a | , $\quad s = s_3$

$x' =$ | b | a | c | b | a | , | c | b | c | b | a | , | c | a | a | b | a |

Rekursion mit $i = 1, \ell = 2$ $\quad x =$ | b | a | a | b | a | , $\quad s = s_3$

$x' =$ | b | b | a | b | a | , | b | a | a | c | a | , | b | a | a | b | c |

Rekursion mit $i = 1, \ell = 3$ $\quad x =$ | | | | | | , $\quad s = -$

$x' =$ | | | | | | , | | | | | | , | | | | | |

Anmerkung: Der dritte Wert für x' im initialen Aufruf kann bzw. soll alternativ auch leer gelassen werden. Da es unpraktikabel ist, an dieser Stelle schon auf Abbruch des Algorithmus zu prüfen, sind beide Varianten akzeptabel.

b. Was ist die asymptotische Laufzeit des Algorithmus in Abhängigkeit von n, m und k? Begründen Sie. [2 Punkte]

Lösung

Es gibt insgesamt k Level, auf denen sich die Suche verzweigt (da i mit 0 initialisiert ist und für $i = k$ nicht mehr verzweigt wird) und der Verzweigungsgrad ist durch $|J| = k+1$ beschränkt. Jeder Verzweigungschritt besteht aus einer Iteration durch alle Strings, hat also Laufzeit $\mathcal{O}(nm)$. Das ergibt eine Gesamtlaufzeit von $\mathcal{O}\big((k+1)^k nm\big)$.

c. Sei $N := n \cdot m$ die insgesamte Eingabelänge. Macht ein Algorithmus mit der Laufzeit aus Teilaufgabe **b.** das Problem *fixed-parameter tractable* (FPT) für die Eingabelänge N und den Parameter k? Begründen Sie kurz. [1 Punkt]

Lösung

Eingesetzt in Teilaufgabe **b.** ergibt sich $\mathcal{O}\big((k+1)^k N\big)$ als Laufzeit. Da $\mathcal{O}\big((k+1)^k N\big) = \mathcal{O}(f(k)p(N))$ mit $f(k) = (k+1)^k$ berechenbar und $p(N) = N$ polynomiell, macht dies das Problem FPT.

d. Wir betrachten eine *gültige Lösung* z für eine gegebene lösbare Instanz. Wir wollen zeigen, dass der Algorithmus Fortschritt in Richtung dieser Lösung macht. Für einen rekursiven Aufruf des Algorithmus mit einem String $x \in \mathcal{A}^n$ sei $i < k$ und es gebe einen Eingabestring $s \in S$ mit $d(x,s) > k$.

Zeigen Sie für die vom Algorithmus berechnete Menge von modifizierten Lösungskandidaten: Es gibt ein x' mit $d(x',z) < d(x,z)$.

[3 Punkte]

Lösung

Da z eine gültige Lösung ist, gilt $d(s,z) \leq k$. Mit $|J| = k+1$ existiert somit ein ℓ mit $s[j_\ell] = z[j_\ell]$. Nach Definition von j_ℓ ist $x[j_\ell] \neq s[j_\ell] = z[j_\ell]$, andererseits aber $x'[j_\ell] = s[j_\ell] = z[j_\ell]$. Hieraus folgt $d(x',z) = d(x,z) - 1$.

Lösungsvorschlag

EK

ZK

Aufgabe 4. Externe Algorithmen: Crowd-Computing [10 Punkte]

In einem Crowd-Computing-Projekt arbeiten m teilnehmende Kleinrechner gemeinsam an einem schwierigen Problem. Die Verwaltung des Projekts verteilt dazu eine Folge $j_1, \ldots, j_n$ von kleineren Berechnungs-Jobs mit zugehörigen Arbeitsvolumen $v_1, \ldots, v_n$ auf die Kleinrechner. Jeder Job soll sofort nach seinem Eintreffen jenem Kleinrechner zugewiesen werden, welchem bisher das geringste Gesamtvolumen zugewiesen wurde.

Es stehe der Verwaltung eine Maschine mit Hauptspeicher der Größe M, sowie externer Speicher mit Blockgröße B zur Verfügung. Die Anzahl m der Kleinrechner sei zu groß, um in den Hauptspeicher passen. Jedoch gelte $m \ll M^2/B$. Jeder Kleinrechner habe eine eindeutige ID zur Identifizierung.

a. Geben Sie einen Algorithmus an, der mit amortisiert $\mathcal{O}((n+m)/B)$ I/O-Operationen die korrekte Zuweisung für jeden Job bestimmen kann. Begründen Sie, wieso diese asymptotische Anzahl I/O-Operationen eingehalten wird. [4 Punkte]

Lösung

Nutze eine externe PQ Q (mittelgroße PQ aus Vorlesung). Füge alle Kleinrechner mit ihrem initialen Volumen 0 zu Q hinzu. Für jeden Job a_i, hole den Kleinrechner mit geringstem Volumen durch `deleteMin()` auf Q, addiere v_i auf dessen Volumen und füge Kleinrechner wieder mit neuem Gesamtvolumen ein. Es sind nie mehr als m Elemente in der PQ, also ist es möglich, die mittelgroße PQ aus der Vorlesung zu verwenden, da $m \ll M^2/B$. Insgesamt $\mathcal{O}(n+m)$ mal `insert()` und $\mathcal{O}(n)$ mal `deleteMin()` => amortisiert in $\mathcal{O}((n+m)/B)$ I/Os.

Kleinrechner treten gelegentlich aus dem Projekt aus. Ein Kleinrechner, der austreten möchte, teilt dies der Verwaltung mit und arbeitet seine bereits zugeteilten Jobs noch ab, aber nimmt keine neuen Jobs mehr an.

b. Erweitern Sie Ihren Algorithmus aus Teilaufgabe **a.** unter Verwendung eines externen Suchbaums so, dass er mit austretenden Kleinrechnern umgehen kann.
Wie verändert sich die asymptotische Anzahl I/O-Operationen, wenn insgesamt $k < m$ Kleinrechner austreten? Begründen Sie. [3 Punkte]

Lösung

Wenn ein Kleinrechner r austritt, füge die ID von r zum externen Suchbaum hinzu. Bei jedem `deleteMin()` auf Q, prüfe, ob der resultierende Kleinrechner im Suchbaum enthalten ist. Falls ja, ignoriere den Kleinrechner, entferne ihn aus dem Suchbaum, füge ihn nicht neu in Q ein, und wähle den nächsten Kleinrechner aus Q mit `deleteMin()`.

Ein externer Suchbaum der Größe k benötigt $\mathcal{O}(\log_B k/M)$ I/Os pro Einfügen, Suchen, und Entfernen, also insgesamt $\mathcal{O}(n \log_B k/M)$ I/Os zusätzlich.

c. Gehen Sie nun davon aus, dass jeder Kleinrechner bei seinem Austritt auch sein insgesamt zugewiesenes Arbeitsvolumen an die Verwaltung übermittelt. Sie dürfen annehmen, dass die Arbeitsvolumen der ausgetretenen Rechner paarweise verschieden sind.
Geben Sie einen Algorithmus an, der mithilfe dieser Information mit amortisiert $\mathcal{O}((n+m)/B)$ I/O-Operationen die korrekten Zuweisungen aller Jobs bei beliebiger Anzahl Austritte $k < m$ ausgibt. Begründen Sie, wieso diese Schranke eingehalten wird. [3 Punkte]

Lösung

Vom Moment des Austritts an, ändert sich das Arbeitsvolumen nicht mehr. Nehme an, dass Kleinrechner r mit Volumen v_r kündigt und Kleinrechner r' und r'' bereits mit Volumen $v_{r'} < v_r$ bzw. $v_{r''} > v_r$ gekündigt haben. Beim wiederholten Ausführen von `deleteMin()` auf Q wird r dann nach r' jedoch vor r'' das nächste Mal auftauchen. Die Ordnung der ausgetretenen Kleinrechner nach aufsteigendem Volumen entspricht also ihrer relativen Ordnung in Q. Wir merken uns diese Ordnung der ausgetretenen Kleinrechner in einer zusätzlichen externen PQ Q'. Dann genügt es, nach jedem `deleteMin()` auf Q zu prüfen, ob der resultierende Kleinrechner das gleiche wie an Q'`.min()` ist. Falls ja, hat der Kleinrechner bereits gekündigt und kann für die Zuweisung eines Auftrags ignoriert werden. Entferne den Kleinrechner dann auch aus Q'.
Insgesamt wird je $\mathcal{O}(k)$ mal `insert()` und `deleteMin()` sowie $\mathcal{O}(n)$ mal `min()` auf Q' ausgeführt. Es entstehen amortisiert $\mathcal{O}(k/B)$ zusätzliche I/Os. Wegen $k < m$ bleibt die Schranke $\mathcal{O}((k+n+m)/B) = \mathcal{O}((n+m)/B)$ erhalten.

Lösungsvorschlag

Aufgabe 5. Approximationsalgorithmen: Mehrdimensionaler Rucksack [12 Punkte]

Wir betrachten im Folgenden das mehrdimensionale (unbeschränkte) Rucksack-Problem. Gegeben ist eine nicht-leere Menge von n Items, wobei jedes Item einen Profit $p_i \in \mathbb{R}_{\geq 0}$ sowie ein mehrdimensionales Gewicht $w_i = (w_i^1, \ldots, w_i^d) \in [0,1]^d$ hat. Jedes Gewicht ist in mindestens einer Dimension ungleich null. Jedes Item kann beliebig oft ausgewählt werden, im Folgenden wird die gewählte Anzahl für das i'te Item mit $x_i \in \mathbb{N}_0$ bezeichnet.

Das Ziel ist die Maximierung des Gesamtprofits $P := \sum_{i=1}^{n} x_i p_i$. Dabei hat der Rucksack in jeder Dimension Kapazität 1, die Lösung muss also $\sum_{i=1}^{n} x_i w_i^j \leq 1$ für $1 \leq j \leq d$ erfüllen.

a. Gegeben sei die folgende Instanz mit $d = 2$:

i	p_i	w_i^1	w_i^2
1	6	0,6	0
2	13	0,6	0,9
3	8	0,5	0,5

Geben Sie die optimale Lösung sowie ihren Profit P_{OPT} an: [2 Punkte]

Lösung

$x_1 = 0$ $x_2 = 0$ $x_3 = 2$ $P_{OPT} = 16$

Wir betrachten den folgenden Approximationsalgorithmus: Für jedes Item wird die Profitdichte $q_i = \frac{p_i}{\hat{w}_i}$ berechnet, wobei $\hat{w}_i := \sum_{j=1}^{d} w_i^j$ das summierte Gewicht ist. Der Algorithmus wählt greedy das Item mit der höchsten Profitdichte aus, das die Gesamtkapazität nicht verletzt. Der Algorithmus terminiert, sobald kein Item mehr ausgewählt werden kann.

b. Zeigen Sie, dass der Approximationsfaktor des Algorithmus für jede Dimension $d \geq 1$ nicht besser als $2d$ ist, indem Sie eine Familie von Instanzen konstruieren, auf denen der Gesamtprofit P_A der vom Algorithmus berechneten Lösung beliebig nahe an $\frac{1}{2d} P_{OPT}$ herankommt. Begründen Sie, dass dies gilt.

Hinweis: Wie verhält sich der Algorithmus auf der Instanz aus Teilaufgabe **a.**? Verallgemeinern Sie mit einem Parameter k und benutzen Sie zwei Items, wobei $p_1 = k+1$ und $w_1^1 = \frac{k+1}{2k}$. [4 Punkte]

Lösung

Mit Hilfe des Hinweises konstruieren wir eine d-dimensionale Instanz mit zwei Items. Wir wählen $p_1 = k+1$, $w_1^1 = \frac{k+1}{2k}$ und $w_1^j = 0$ für $j > 1$. Für das zweite Item nutzen wir $p_2 = 2dk - 1$ und $w_2^j = 1$ für alle j. Entscheidend ist bei der Wahl dieser Werte, dass die Profitdichte geringfügig kleiner als für das erste Item ist.

Es ergeben sich Profitdichten $q_1 = (k+1)(\frac{k+1}{2k})^{-1} = 2k$ und $q_2 = \frac{2dk-1}{d} = 2k - \frac{1}{d}$. Folglich wählt der Algorithmus das erste Item aus (die Kapazität erlaubt danach kein weiteres Item), was einen Gesamtprofit von $P_A = k+1$ ergibt. Die optimale Lösung wählt das zweite Item mit $P_{OPT} = 2dk - 1$.

Wir folgern $\frac{P_{OPT}}{P_A} = \frac{2dk-1}{k+1} = \frac{k}{k+1} 2d - \frac{1}{k+1}$ und somit $\frac{P_{OPT}}{P_A} \to 2d$ für $k \to \infty$.

c. Zeigen Sie: Sei s der Index des Items, das vom Algorithmus als erstes ausgewählt wird. Nach Ausführung des Algorithmus gibt es eine Dimension j, deren Kapazität mehr als zur Hälfte von diesem Item genutzt wird, also $x_s w_s^j > \frac{1}{2}$. [2 Punkte]

Lösung

(Nebenbemerkung: Es wird immer mindestens ein Item ausgewählt, da die Items nicht-leer sind und Gewicht ≤ 1 haben.)
Wir nehmen an, es gilt $x_s w_s^j \leq \frac{1}{2}$ für jede Dimension. Daraus folgt insbesondere $w_s^j \leq \frac{1}{2}$. Das bedeutet aber, dass x_s eins größer gewählt werden kann, ohne die Kapazität zu verletzen. Der Algorithmus hätte dieses Item also öfter als x_s-mal ausgewählt, was einen Widerspruch zur Annahme ergibt.

d. Sei s wieder das zuerst gewählte Item und $y_1, \ldots, y_n$ eine optimale Lösung. Zeigen Sie, dass folgende Ungleichung für den Profit des Algorithmus P_A und den optimalen Profit P_{OPT} gilt:

$$\frac{P_A}{P_{OPT}} \geq \frac{x_s \hat{w}_s}{\sum_{i=1}^n y_i \hat{w}_i}$$

Hinweis: Nutzen Sie das Kriterium, nach dem die Items gewählt werden, um die Profitdichten abzuschätzen. [2 Punkte]

Lösung

Beobachtung: Die Greedy-Eigenschaft impliziert $q_s = \max_j q_j$. Wir folgern

$$\begin{aligned}\frac{P_A}{P_{OPT}} &\geq \frac{x_s p_s}{\sum_{i=1}^n y_i p_i} = \frac{q_s x_s \hat{w}_s}{\sum_{i=1}^n q_i y_i \hat{w}_i} \\ &\geq \frac{q_s x_s \hat{w}_s}{q_s \sum_{i=1}^n y_i \hat{w}_i} = \frac{x_s \hat{w}_s}{\sum_{i=1}^n y_i \hat{w}_i}\end{aligned}$$

e. Zeigen Sie mit Hilfe der vorigen Teilaufgaben: Der Algorithmus erreicht tatsächlich einen Approximationsfaktor von $2d$.

Hinweis: Das summierte Gewicht einer Lösung kann d nicht überschreiten. [2 Punkte]

Lösung

Wir kombinieren zuerst die Ergebnisse aus Teilaufgabe **c.** und **d.**.

$$\begin{aligned}\frac{P_A}{P_{OPT}} &\geq^{\mathbf{d}} \frac{x_s \hat{w}_s}{\sum_{i=1}^n y_i \hat{w}_i} \geq \frac{x_s w_s^j}{\sum_{i=1}^n y_i \hat{w}_i} \\ &>^{\mathbf{c}} \frac{1}{2 \sum_{i=1}^n y_i \hat{w}_i}\end{aligned}$$

Dabei wird j wie in Teilaufgabe **c.** gewählt. Der Hinweis liefert uns $\sum_{i=1}^n y_i \hat{w}_i \leq d$, gemeinsam mit der vorigen Abschätzung ergibt sich also

$$\frac{P_A}{P_{OPT}} \geq \frac{1}{2d}$$

Lösungsvorschlag

EK

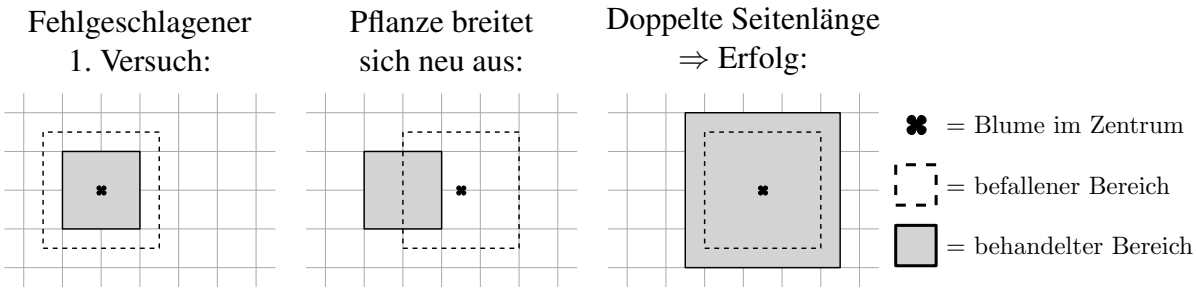
ZK

Aufgabe 6. Online-Algorithmen: Unkrautbekämpfung [9 Punkte]

Das Feld von Bauer Algor ist von der "Quadratwurzel" befallen. Dieses Unkraut befällt unterirdisch einen quadratischen Bereich, oberirdisch ist nur eine rote Blume im Mittelpunkt des befallenen Bereichs zu erkennen.

Bauer Algor geht mit einem Unkrautvernichter gegen die Quadratwurzel vor, wobei er möglichst wenig des umweltschädlichen Mittels versprühen möchte. Pro Quadratmeter Acker wird ein Liter Unkrautvernichter benötigt. Das Problem: Wenn der Bauer nicht die ganze infizierte Fläche besprüht, überlebt die Quadratwurzel und breitet sich um einen neuen Mittelpunkt erneut aus. Dabei bleibt die Größe der Ausbreitung bei jedem neuen Mittelpunkt gleich.

Der Bauer weiß, dass Quadratwurzeln immer eine Seitenlänge von mindestens 5m erreichen und sich in Nord/Süd- und Ost/West-Richtung ausbreiten. Er überlegt sich folgendes Vorgehen, um das Unkraut zu vernichten: Zunächst besprüht er einen quadratischen Bereich von 10m Seitenlänge um die rote Blume im Mittelpunkt. Sollte die Pflanze erneut auftauchen, verdoppelt er die Seitenlänge. Als nächstes besprüht er also 20×20m, dann 40×40m, usw. Dies wiederholt er, bis sich die Quadratwurzel nicht erneut zeigt. Die folgende Abbildung zeigt das Verfahren:



a. Nehmen Sie an, dass Bauer Algor gerade mit einer Quadratwurzel kämpft, die eine Seitenlänge von 21m hat. Wie viele Liter Unkrautvernichter verwendet Bauer Algor mit seinem Verfahren bis zur Vernichtung der Quadratwurzel? Wie viele Liter wären nötig, wenn Bauer Algor die Seitenlänge bereits kennen würde? [2 Punkte]

Lösung

Mindestens werden $21\text{m} \cdot 21\text{m} \cdot 1\frac{\text{L}}{\text{m}^2} = 441\text{L}$ benötigt.
Bauer Algor verwendet $\sum_{i=1}^{3}(5 \cdot 2^i\text{m})^2 \cdot 1\frac{\text{L}}{\text{m}^2} = 2100\text{L}$.

b. Zeigen Sie, dass das Verfahren von Bauer Algor im Allgemeinen höchstens $\frac{16}{3}$ mal so viel Unkrautvernichter benötigt, wie mindestens nötig wären, um die Quadratwurzel zu vernichten.
Hinweis: $\sum_{i=0}^{n} x^i = \frac{x^{n+1}}{x-1}$ für $x > 1$. [3 Punkte]

Lösung

Sei a die unbekannte Seitenlänge der Quadratwurzel. Der Bauer muss insgesamt $n := \lceil \log_2 \frac{a}{5} \rceil$ mal Sprühen bis die behandelte Fläche die Quadratwurzel überdeckt. Für die insgesamt besprühte Fläche A gilt:

$$
\begin{aligned}
A &= \sum_{i=1}^{n} (5 \cdot 2^i)^2 \\
&= 25 \cdot \sum_{i=1}^{n} 4^i \\
&\leq 25 \cdot \sum_{i=0}^{n} 4^i \\
&= 25 \cdot \frac{4^{n+1} - 1}{4 - 1} \\
&\leq 25 \cdot \frac{4}{3} \cdot 2^{2\lceil \log_2 \frac{a}{5} \rceil} \\
&\leq 25 \cdot \frac{4}{3} \cdot 2^{2(\log_2 \frac{a}{5} + 1)} \\
&= 25 \cdot \frac{16}{3} \cdot (2^{\log_2 \frac{a}{5}})^2 \\
&= 25 \cdot \frac{16}{3} \cdot (\frac{a}{5})^2 \\
&= \frac{16}{3} a^2
\end{aligned}
$$

Bemerkung: Der in der Klausur gegebene Hinweis war nicht ganz richtig, es müsste korrekt heißen $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ für $x > 1$. Der Fehler im Hinweis macht für die Lösung wegen der Abschätzung nach oben keinen Unterschied.

c. Die "Rechteckwurzel" ist eine nahe Verwandte der Quadratwurzel. Sie unterscheidet sich nur darin, dass sie sich in einem Rechteck statt einem Quadrat um ihren Mittelpunkt ausbreitet. Der Bauer möchte mit dem bekannten Vorgehen mit quadratischer behandelter Fläche gegen die Rechteckwurzel vorgehen. Kann so trotzdem garantieren werden, dass nur ein konstanter Faktor mehr Unkrautvernichter verbraucht wird als mindestens nötig? Begründen Sie. [2 Punkte]

Lösung

Es kann kein konstanter kompetitiver Faktor erreicht werden. Für ein Rechteck mit Seitenlängen a und b kann der Bauer erst aufhören, wenn die Seitenlänge der quadratischen behandelten Fläche größer als $\max\{a, b\}$ ist. Mit $\frac{a}{b} \to \infty$ wächst auch das Verhältnis aus behandelter Fläche und befallener Fläche unbeschränkt.

d. Nehmen Sie an, dass die Ausbreitung der Rechteckwurzel immer das gleiche Seitenverhältnis $\frac{a}{b}$ (mit unbekannten Seitenlängen $a > b \geq 5$m) hat. Der Bauer verwende weiterhin das bekannte Vorgehen. Geben Sie an, wie viel mehr Unkrautvernichter als mindestens nötig der Bauer in Abhängigkeit von $\frac{a}{b}$ verbraucht. [2 Punkte]

Lösung

Der Bauer benötigt höchstens $\frac{16}{3}\frac{a}{b}$ mal mehr Unkrautvernichter als mindestens nötig.