Lösungsvorschläge
zur Hauptklausur
„1662/1663 Datenstrukturen“

11.08.2007

Aufgabe 1 Algebra für Bilder

(a)

algebra *picture*

sorts	*card*, *color*, *picture*		
ops	*createPicture*	: *card* × *card*	→ *picture*
	insert	: *picture* × *picture* × *card* × *card*	→ *picture*
	rotate	: *picture*	→ *picture*
	height	: *picture*	→ *card*
	width	: *picture*	→ *card*
	getColor	: *picture* × *card* × *card*	→ *color*
	setColor	: *picture* × *color* × *card* × *card*	→ *picture*
	mirror	: *picture*	→ *picture*
	cut	: *picture* × *card* × *card* × *card* × *card*	→ *picture*

(b)

sets

$card = \{ x \in \mathbb{Z} \mid x \geq 1 \} \cup \{\bot\}$

$color = \{ (x, y, z) \in \mathbb{R}^3 \mid 0 \leq x \leq 1, 0 \leq y \leq 1, 0 \leq z \leq 1 \} \cup \{\bot\}$

$picture = \{ (b, h , c_{11} \ldots c_{b1} c_{12} \ldots c_{bh}) \mid b, h \in card, c_{ij} \in color, 1 \leq i \leq b, 1 \leq j \leq h \} \cup \{\bot\}$

functions

$$createPicture(b,h) = \begin{cases} \bot & \text{falls } b = \bot \vee h = \bot \\ (b,h,(1,1,1)\ldots(1,1,1)) & \text{sonst} \end{cases}$$

$height((b, h, c_{11} \ldots c_{bh})) = h$
$height(\bot) = \bot$
$width((b, h, c_{11} \ldots c_{bh})) = b$
$width(\bot) = \bot$

$$getColor((b,h,c_{11}\ldots c_{bh}),x,y) = \begin{cases} c_{xy} & \text{wenn } 1 \leq x \leq b \wedge 1 \leq y \leq h \\ \bot & \text{sonst} \end{cases}$$

$getColor(\bot, x, y) = \bot$

$$setColor((b,h,c_{11}\ldots c_{bh}),x,y,c) = \begin{cases} (b,h,c^{\bullet}_{11}\ldots c^{\bullet}_{bh}) & \text{falls } x \neq \bot \wedge y \neq \bot \wedge c \neq \bot \\ \bot & \text{sonst} \end{cases}$$

$$\text{mit } c^{\bullet}_{ij} = \begin{cases} c & \text{falls } i = x \wedge j = y \\ c_{ij} & \text{sonst} \end{cases}$$

$setColor(\bot, x, y, c) = \bot$

$mirror((b,h,c_{11}\ldots c_{bh})) = (b,h,c^{\bullet}_{11}\ldots c^{\bullet}_{bh})$

mit $c^{\bullet}_{ij} = c_{(b-i+1)j}$

$mirror(\perp) = \perp$

$rotate((b,h,c_{11}...c_{bh})) = (h,b,c^{\bullet}_{11}...c^{\bullet}_{hb})$

mit $c^{\bullet}_{ij} = c_{(b-j+1)i}$

$rotate(\perp) = \perp$

$$cut((b,h,c_{11}...c_{bh}),x_1,y_1,x_2,y_2) = \begin{cases}(b',h',c^{\bullet}_{11}...c^{\bullet}_{b'h'}) & \text{falls } b' \geq 1 \wedge h' \geq 1 \wedge x_i \neq \perp \wedge y_i \neq \perp \\ \perp & \text{sonst}\end{cases}$$

$b' = \min(b - x_1, x_2 - x_1) + 1$

$h' = \min(h - y_1, y_2 - y_1) + 1$

$c^{\bullet}_{ij} = c_{(i+x_1-1)(j+y_1-1)} 1 \leq i \leq b', 1 \leq j \leq h'$

$cut(\perp, x_1, y_1, x_2, y_2) = \perp$

$$insert((b_1,h_1,c_{11}...c_{b_1h_1}),(b_2,h_2,d_{11}...d_{b_2h_2}),x,y) = \begin{cases}(b_1,h_1,e_{11}...e_{b_1h_1}) & \text{falls } x \neq \perp \wedge y \neq \perp \\ \perp & \textit{sonst}\end{cases}$$

$$\text{mit } e_{ij} = \begin{cases}d_{(i-x+1)(j-y+1)} & \text{wenn } x \leq i < x + b_2 \wedge y \leq j < y + h_2 \\ c_{ij} & \text{sonst}\end{cases}$$

$insert(\perp, p, x, y) = \perp$
$insert(p, \perp, x, y) = \perp$

(c)
$width(P) = height(rotate(P))$
$width(cut(P, x_1, y_1, x_2, y_2)) \in \{width(P), \perp\}$

$$getColor(createPicture(b,h),x,y) = \begin{cases}(1,1,1) & \text{falls } 1 \leq x \leq b \wedge 1 \leq y \leq h \\ \perp & \text{sonst}\end{cases}$$

$getColor(mirror(P), x, y) = getColor(P, width(P) - x + 1, y)$
$getColor(insert(P_1, P_2, x, y), x, y) \in \{getColor(P_2, 1, 1), \perp\}$

Aufgabe 2 Hashing

(a)

Bei einer Tabellengröße von m=10, besteht der mittlere auszuschneidende Block nur aus einer einzelnen Ziffer. Als mittleres Element wurde hier dasjenige gewählt, welches rechts der „tatsächlichen Mitte" der Ziffernfolge von k^2 liegt. Die Auswahl des links davon liegenden Elements ist natürlich genauso richtig. Die Zahlen werden auf die folgenden Behälternummern abgebildet:

k	k^2	$h(k)$	$h_1(k)$	$h_2(k)$	$h_3(k)$	$h_4(k)$	$h_5(k)$
120	14400	4					
163	26569	5					
19	361	6					
84	7056	5	6	9			

69	4761	6	7				
67	4489	8					
175	30625	6	7	0			
59	3481	8	9	2			
13	169	6	7	0	5	2	1

Der Inhalt der Tabelle nach Einfügen aller Zahlen ist:

0	1	2	3	4	5	6	7	8	9
175	13	59		120	163	19	69	67	84

Aufgabe 3

(a)

Da die Zeitstempel aus 3 Komponenten bestehen, benötigt Radixsort 3 Phasen. In der ersten Phase wird nach Sekunden zur Basis 60, in der zweiten Phase nach Minuten zur Basis 60 und in der letzten Phase nach Stunden zur Basis 24 sortiert. In den folgenden Tabellen werden leere Behälter nicht angegeben.

Phase 1:

B_1	B_3	B_{14}	B_{59}
23:59:01 12:11:01 03:27:01 13:28:01	12:11:03	13:31:14 17:23:14	17:15:59

Die Ausgabe der ersten Phase dient als Eingabe für die zweite Phase, deren Ausgabe in der folgenden Tabelle dargestellt ist.

B_{11}	B_{15}	B_{23}	B_{27}	B_{28}	B_{31}	B_{59}
12:11:01 12:11:03	17:15:59	17:23:14	03:27:01	13:28:01	13:31:14	23:59:01

Nach der dritten Phase zeigt sich folgende Behälterkonfiguration:

B_3	B_{12}	B_{13}	B_{17}	B_{23}
03:27:01	12:11:01 12:11:03	13:28:01 13:31:14	17:15:59 17:23:14	23:59:01

Als Ergebnisfolge wird

03:27:01, 12:11:01, 12:11:03, 13:28:01, 13:31:14, 17:15:59, 17:23:14, 23:59:01

ausgegeben.

(b)

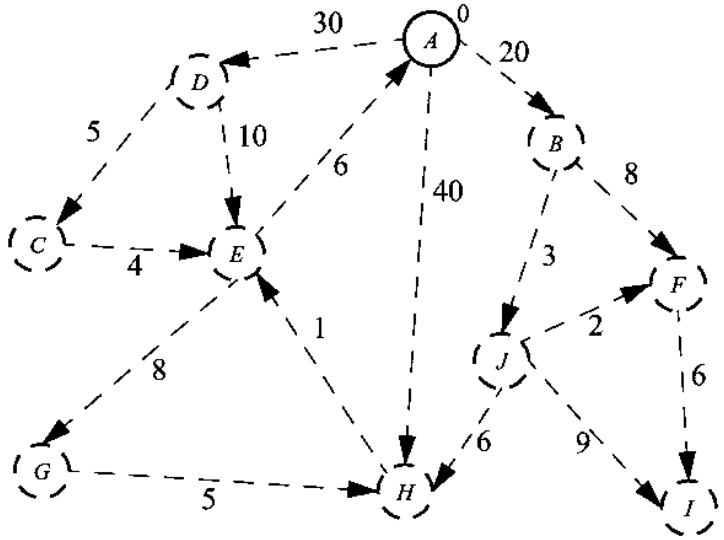
Radixsort müßte zur Basis m=10 k=100 Phasen durchlaufen, also insgesamt 100 mal 1000 Zahlen auf Buckets verteilen. Wählt man als Basis m=100 so ist k=50, also müßten 50.000 Zahlen auf Buckets verteilt werden. Der minimale Wert würde für m=1000 und k=34 erreicht; dabei müßten insgesamt nur 34.000 Zahlen auf Buckets verteilt werden.

Heapsort benötigt dagegen weniger als $2 \cdot n \cdot \log n + n$ Vergleiche also ungefähr $2 \cdot 9{,}96 \cdot 1000 + 1000 \leq 21.000$ Vergleiche. Wenn die Kosten für den Vergleich zweier Zahlen und das Vertauschen im Array in etwa den Kosten für das Verteilen auf Buckets entsprechen, wäre Heapsort vorzuziehen.

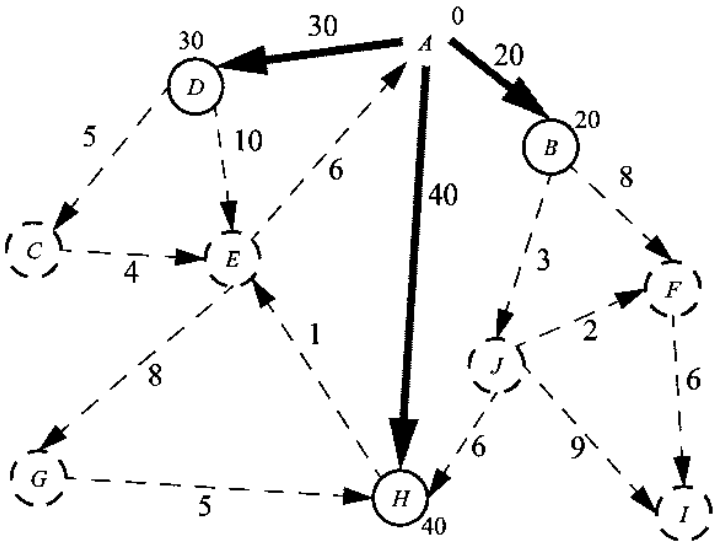
Aufgabe 4 (Bestimmung kürzester Wege)

Wir verwenden folgende Notation:

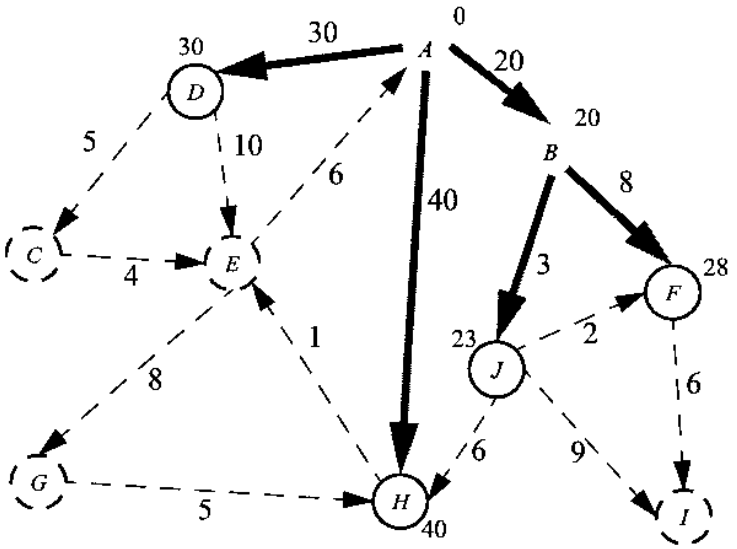
Im initialen Schritt wird Knoten A gelb gefärbt und $dist(A)$ auf 0 gesetzt. Alle anderen Knoten und Kanten sind ungefärbt. Wir markieren jeden Knoten zusätzlich mit seiner aktuellen kürzesten Distanz:



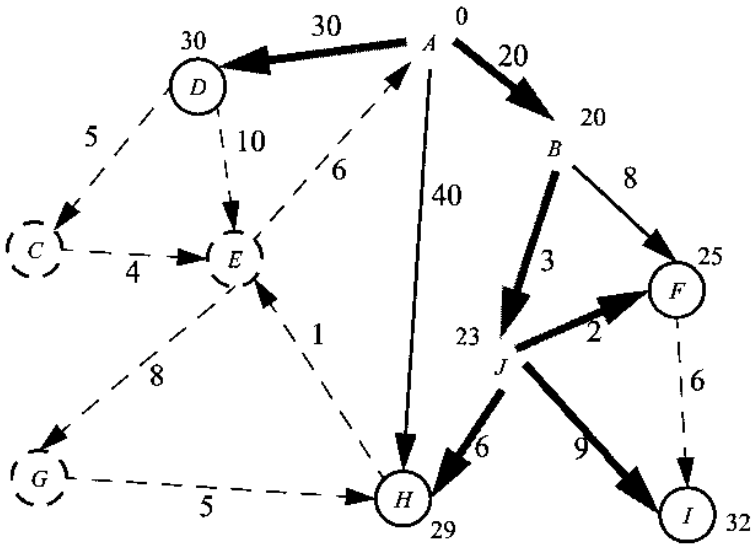
Wir wählen Knoten *A*, färben die Kanten zu seinen Nachfolgern auf rot und färben die Nachfolger gelb ein:



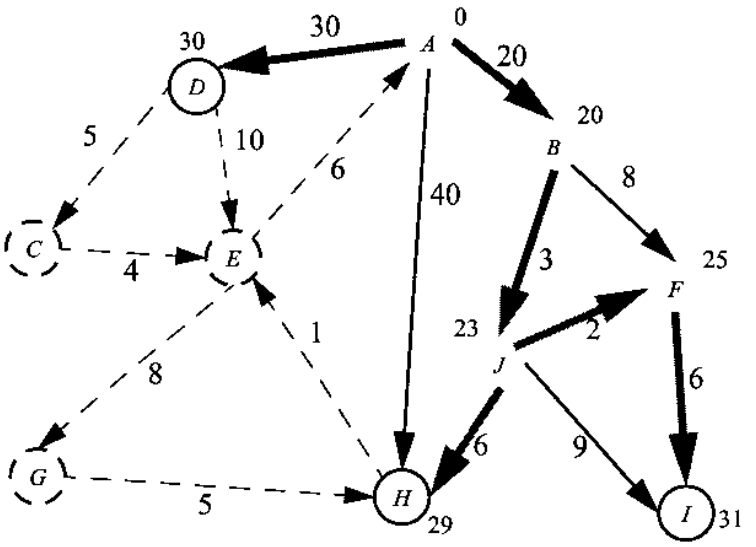
Als nächstes wird Knoten *B* grün gefärbt:



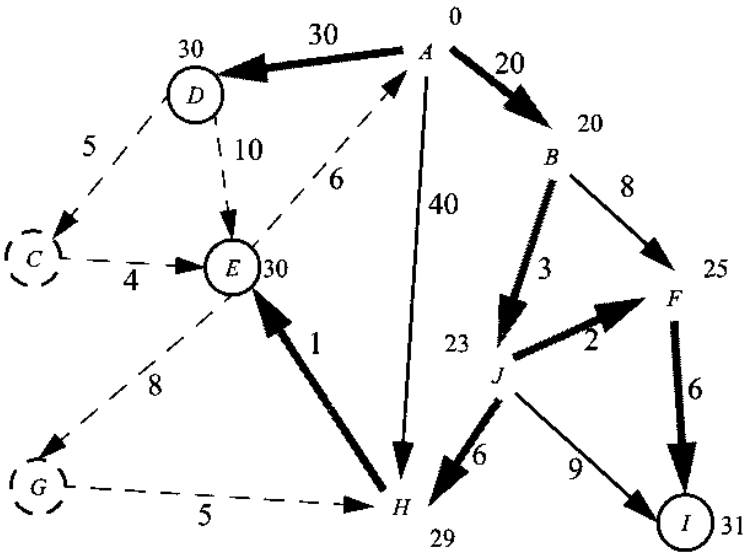
Nun wird Knoten *J* grün gefärbt. Dadurch ändern sich die bisherigen kürzesten Wege zu den Knoten *F* und *H*.



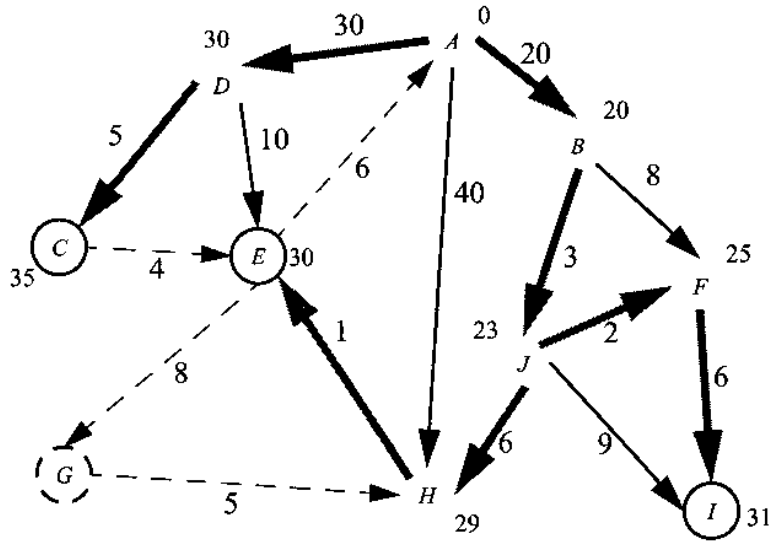
Wir wählen Knoten *F*, wodurch sich die Distanz zu Knoten *I* ändert.



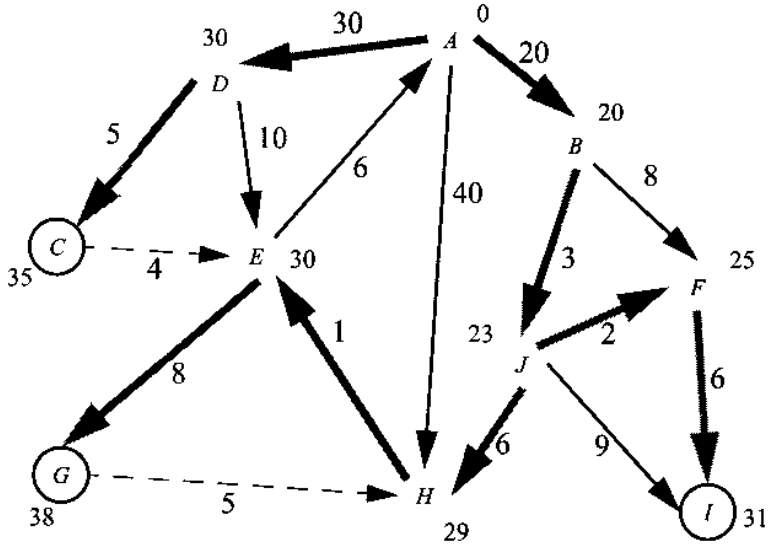
Als nächstes wird Knoten *H* grün gefärbt und *E* in die Menge der gelben Knoten aufgenommen.



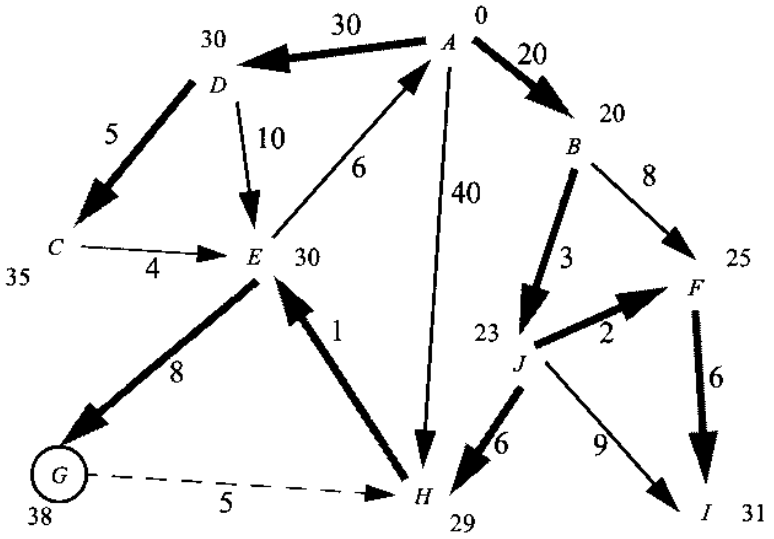
Als nächstes wird Knoten *D* ausgewählt, da dieser alphabetisch kleiner als Knoten *E* ist.



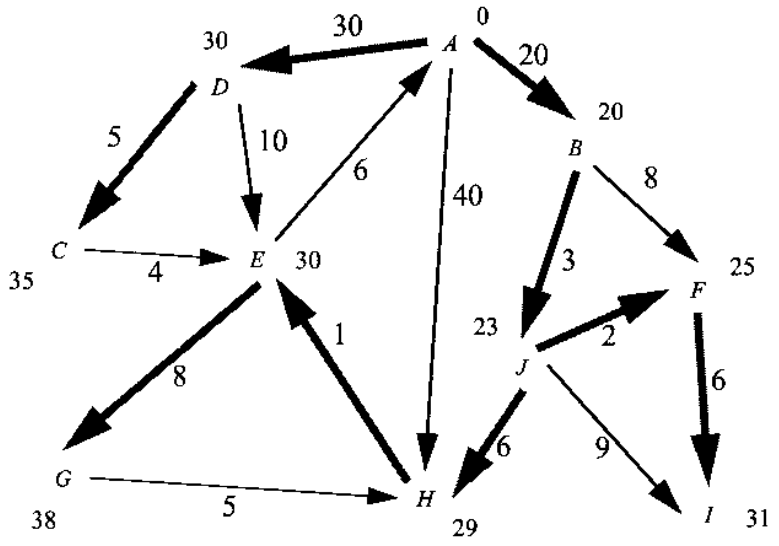
Nun wird Knoten *E* in die Menge der grünen Knoten aufgenommen.



Knoten *I* wird nun grün gefärbt, ohne daß sich Änderungen bei Kanten ergeben. Anschließend wird Knoten *C* gewählt.



Zum Schluß wird noch Knoten *G* grün gefärbt und der Algorithmus endet.



Aufgabe 5 Geometrische Algorithmen

(a)

Gegeben sei eine gemischte Menge von Punkten und (achsenparallelen) Rechtecken. Finde alle Paare (Punkt, Rechteck), bei denen der Punkt im Rechteck liegt. Falls die Objektmenge p Punkte und r Rechtecke enthält, kann es maximal $p \cdot r$ Punkteinschlüsse geben.

(b)

algorithm *Punkteinschluss*(*R*, *P*)

```
{ R: eine Menge von Rechtecken {(x_l, y_l, x_r, y_r) ∈ ℝ^4},
  P: eine Menge von Punkten {(x, y) ∈ ℝ^2}}
i = 0;
while R ≠ ∅ do
  Entnehme r aus R;
  i = i + 1; /* Erzeuge einen eindeutigen Schlüsselwert für r */
  SE.append( (left, i, r, x_l) );
  SE.append( (right, i, r, x_r) );
  B.insert(i,r);
done
while P ≠ ∅ do
  Entnehme p aus P;
  SE.append( (point, p, x) );
done
/* Sortiere S bezüglich der x-Koordinate der darin gespeicherten Objekte */
SE.sort(); SE.startScan();
while (s = SE.getNext()) ≠ null do
  if (s = (left, j, r, x_l)) then /* linke Rechteckseite */
    SS.insert(j, [r.y_l, r.y_r]);
  else if (s = (right, j, r, x_r)) then /* rechte Rechteckseite */
    SS.remove(j, [r.y_l, r.y_r]);
  else /* Punkt s = (point, (x, y), x)*/
    SS.find(y);
    while (r = SS.getNext()) ≠ null do
      r = B.get(j);
      Gib das Paar (p, r) aus;
    done
  fi
done
```

(c)

Mittels des Punkteinschlußproblems kann man auch das Rechteckschnittproblem lösen. Damit erhält man alle Paare von Rechtecken, die sich schneiden. Berechnet man zu jedem Paar die Schnittfläche, so erhält man wieder eine Menge von Rechtecken. Wendet man auf diese Menge die Lösung für das Maßproblem an, so kann man damit die gesuchte Fläche berechnen.

Mit n Rechtecken von denen k Paare sich schneiden erhält man die Laufzeit $O(n \log n + k)$ für das Rechteckschnittproblem sowie $O(k \log k)$ für das Maßproblem, also insgesamt:

$$O(n \log n + k \log k)$$

Aufgabe 6 Externes Sortieren

(a)

g_1: 43, 71, 12, 35, 11, 80, 99, 47, 62, 15, 77, 26, 69, 93, 19, 45, 23, 38, 56, 91, 10

Die Eingangsfolge hat 21 Elemente, daher werden $\lceil \log_3(21) \rceil = \lceil 2{,}771 \rceil = 3$ Mischphasen benötigt.

Inital Runs:

f_1:	43 | 35 | 99 | 15 | 69 | 45 | 56
f_2:	71 | 11 | 47 | 77 | 93 | 23 | 91
f_3:	12 | 80 | 62 | 26 | 19 | 38 | 10

Phase 1:

g_1:	12 43 71 | 15 26 77 | 10 56 91
g_2:	11 35 80 | 19 69 93
g_3:	47 62 99 | 23 38 45

Phase 2:

f_1:	11 12 35 43 47 62 71 80 99
f_2:	15 19 23 26 38 45 69 77 93
f_3:	10 56 91

Phase 3:

g_1:	10 11 12 15 19 23 26 35 38 43 45 47 56 62 69 71 77 80 91 93 99
g_2:	
g_3:	

(b)

Für den Fall der Aufgabenstellung ist das Verfahren am effizientesten, welches die geringsten Kosten gemessen an Vergleichen und Lese- bzw. Schreibzugriffen erfordert. Da in jeder Phase alle n Datensätze einmal gelesen und geschrieben werden, entspricht dies den Kosten von n Vergleichen. Im folgenden geben wir die Kosten als Anzahl der Vergleiche in Abhängigkeit der Eingabegröße n an.

Da beim direkten Mischen in der i-ten Phase $n/2^i$ Läufe der Länge 2^{i-1} im parallelen Durchlauf verschmolzen werden, werden dabei weniger als $n \cdot 2^i$ Vergleiche benötigt.

$$f_{bin}(n) = \sum_{i=1}^{\lceil \log_2 n \rceil} (n2^i + n)$$

Das Vielweg-Mischen benutzt einen Heap, der mehrere Vergleiche produziert, dafür aber die Anzahl der Phasen reduziert. Dabei werden in der i-ten Phase k^i Elemente in einen Heap eingefügt, wofür jedes mal maximal 2 log k Vergleiche notwendig sind.

$$f_{vielweg}(n) = \sum_{i=1}^{\lceil \log_k n \rceil} (2 \log k \cdot n \cdot k^i + n)$$

Die Frage ist nun, ob es ein k gibt, so daß die Kosten für das Vielweg-Mischen kleiner sind als beim direkten Mischen. Die gesuchte Ungleichung dazu lautet $f_{vielweg}(n) \leq f_{bin}(n)$. Eine Auflösung nach k ist recht kompliziert und soll hier nicht diskutiert werden.