Name: Matrikelnummer:

Nachklausur Algorithmen II, 7. Oktober 2011 Blatt 1 von 1

Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [8 Punkte]

a. Sei k ein Parameter und n die Eingabegröße. Welche der folgenden Laufzeiten machen ein Problem *fixed-parameter-tractable*? Begründen Sie Ihre Antwort jeweils kurz.

1. $O(\sqrt[9]{k^2} \cdot n^{\pi})$

2. $O(n^{\log n} \cdot k^6 \cdot n^{1.387536})$

3. $O(k \cdot n^{\log k})$

[3 Punkte]

Lösung

Ein parametrisiertes Problem Π heißt *fixed-parameter-tractable*, wenn es einen Lösungsalgorithmus zu Π mit Laufzeit $O(f(k) \cdot p(n))$ gibt. Dabei ist p ein Polynom, und f eine berechenbare Funktion, die nur von k abhängt.

1. Ja, denn $\sqrt[9]{k^2}$ hängt nur von k ab und n^{π} ist ein Polynom.
2. Nein, denn $n^{\log n}$ ist kein Polynom.
3. Nein, denn $n^{\log k}$ hängt von n und k ab.

Lösungsende

b. Geben Sie eine asymptotisch *optimale* untere Schranke für die Zahl der I/Os beim vergleichsbasierten, externen Sortieren an. [1 Punkte]

Lösung

$\Omega(\frac{n}{B}(1 + \lceil \log_{M/B} \frac{n}{M} \rceil)$ I/Os, Vorlesungsfolie 270.

Lösungsende

c. Was ist der Unterschied zwischen Las Vegas und Monte Carlo Algorithmen? [1 Punkte]

Lösung

- Las Vegas: Ergebnis immer korrekt, Laufzeit ist Zufallsvariable.
- Monte Carlo: Laufzeit fest, Ergebnis mit bestimmter Wahrscheinlichkeit p inkorrekt.

Lösungsende

Lösungsvorschlag

Fortsetzung von Aufgabe 1

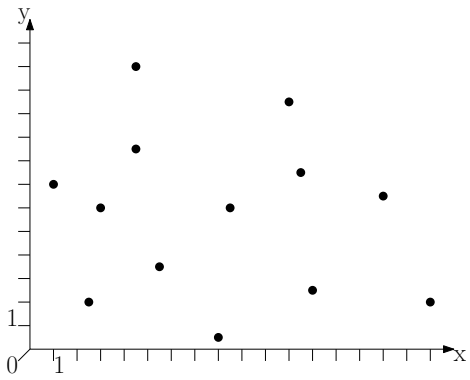
d. Was muss für die Kosten der Kanten in einem Graphen gelten, damit *Bucket Queues* als Prioritätsliste in Dijkstras Algorithmus benutzt werden können? [1 Punkte]

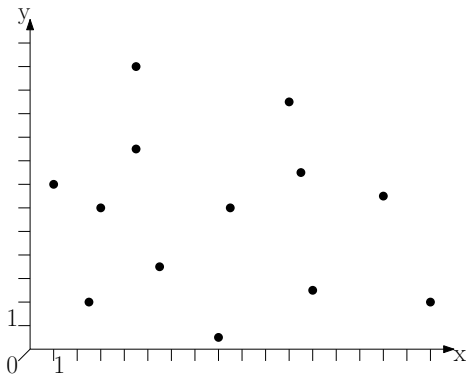
Lösung

Alle Kosten müssen ganzzahlig und positiv sein.
(Außerdem muss eine endliche, maximale Kantenlänge existieren. Dies ist bei explizit gegebenen Graphen immer der Fall.)

Lösungsende

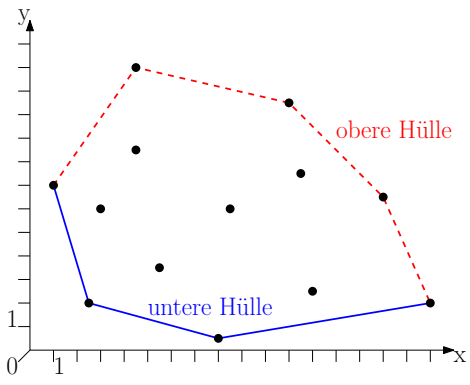
e. Gegeben sei die unten abgebildete Punktmenge. Tragen Sie die obere und untere Hülle dieser Punkte in eines der Koordinatensysteme ein.
(Falls Sie beide Abbildungen benutzt haben, machen Sie deutlich, welche gewertet werden soll.)





[2 Punkte]

Lösung



Lösungsende

(weitere Teilaufgaben auf dem nächsten Blatt)

Lösungsvorschlag

Aufgabe 2. Algorithmen-Entwurf: Zusammenhang [11 Punkte]

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph. Der Knoten eines Graphen wird als Gelenkpunkt bezeichnet, wenn dessen Entfernen die Zahl der Zusammenhangskomponenten erhöht.

a. Zeigen Sie, dass es in einem Graphen **ohne Gelenkpunkte** immer mindestens ein Knoten-Paar $(i, j); i, j \in V$ gibt, so dass zwei Pfade $P_1 = \langle i, \ldots, j \rangle$ und $P_2 = \langle i, \ldots, j \rangle$ existieren, die bis auf die Endpunkte knotendisjunkt sind, d.h.: $P_1 \cap P_2 = \{i, j\}$. [3 Punkte]

Lösung

Sei v kein Gelenkpunkt und i und j zwei zu v benachbarte Knoten. Ein Weg zwischen i und j geht offensichtlich über den Pfad $P_1 = \langle (i, v), (v, j) \rangle$. Wird v nun entfernt fallen auch diese Kanten weg. G bleibt zusammenhängend sonst wäre v ein Gelenkpunkt. Dies bedeutet, dass es also einen weiteren Pfad P_2 zwischen i und j geben muß und dass dieser disjunkt zu P_1 ist, da Knoten v nicht mehr vorhanden ist.

Lösungsende

b. Zeigen Sie in einem Graphen **mit Gelenkpunkten** die Existenz eines Knotens v, für den folgendes gilt: Man kann einen Knoten w entfernen, so dass es von v aus keine Pfade mehr zu mindestens der Hälfte der verbleibenden Knoten gibt. [3 Punkte]

Lösung

Anmerkung: Hier wird das Ergebnis aus Teilaufgabe a) angewendet.

Sei w ein Gelenkpunkt. Dann zerfällt G nach Wegnahme von v in zwei oder mehr Komponenten. Eine Komponente K hat die minimale Anzahl von Knoten unter allen Komponenten. Da es mindestens zwei Komponenten gibt und K die kleinere ist, kann K nicht mehr als $|K| := \frac{|V \setminus \{v\}|}{2}$ Knoten besitzen. Wählt man aus K einen Knoten v, so hat dieser offensichtlich zu weniger als der Hälfte der verbleibenden Knoten einen Pfad.

Lösungsende

(Weitere Teilaufgaben auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 2

c. Zeigen Sie, dass in einem zusammenhängenden, ungerichteten Graphen $G = (V, E)$ stets ein Knoten v existiert, so dass G nach Entfernen von v weiterhin zusammenhängend ist. [2 Punkte]

Lösung

Blätter aus einem Spannbaum können immer entfernt werden, ohne dass der zugrundeliegende Graph zerfällt.

Lösungsende

(weitere Teilaufgaben auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 2

d. Vervollständigen sie folgenden Algorithmus, der in Zeit $O(V+E)$ alle Gelenkpunkte eines Graphen berechnet. Schreiben Sie dazu, was die Funktionen init, root(s), traverseTreeEdge(v,w) und backTrack(u,v) machen, in die vorbereitete Tabelle.

```
Depth-first search of graph G = (V,E)
unmark all nodes
init
for all s ∈ V do
  if s is not marked then
    mark s
    root(s)
    DFS(s,s)
  end if
end for

Procedure DFS(u,v : NodeID)
for all (v,w) ∈ E do
  if w is marked then
    traverseNonTreeEdge(v,w)
  else
    traverseTreeEdge(v,w)
    mark w
    DFS(v,w)
  end if
end for
backtrack(u,v)
```

Hinweis: Benutzen Sie die DFS-Nummerierung und die Fertigstellungszeit (finishing time). Die nicht initialisierten Arrays dfsNum[], finishTime[] und result[] sind in passender Größe vorhanden.

(Lösungsvorlage auf der nächsten Seite)

Lösungsvorschlag

Fortsetzung von Aufgabe 2

init:	dfsPos= 1; finishingTime= 1
root(s):	
traverseTreeEdge(v,w):	
traverseNonTreeEdge(v,w):	no operation
backtrack(u,v):	

Lösung

Das Problem kann per DFS gelöst werden. Folgende Beobachtung liefert den Schlüssel zur Lösung: Ein Knoten v ist immer dann ein Gelenkpunkt, wenn im DFS-Subbaum unterhalb von v keine Querkante existiert. Querkanten können erkannt werden, wenn DFS-Nummerierung und finishing times bekannt sind.

init:	dfsPos= 1; finishingTime= 1
root(s):	dfsNum[s]:=dfsPos++
traverseTreeEdge(v,w):	dfsNum[w]:=dfsPos++
traverseNonTreeEdge(v,w):	no operation
backtrack(u,v):	finishTime[v]:=finishingTime++
	result[v] = ((dfsNum[v] $\geq$ dfsNum[w]) $\cup$ (finishTime[w] < finishTime[v])) *wedge* (result[w] $\wedge (v,w) \in E$)

Eine kreuzende Kante liegt vor, wenn (dfsNum[v] $\geq$ dfsNum[w]) $\cup$ (finishTime[w] < finishTime[v]) gilt. Deswegen wird im backtracking-Schritt diese Eigenschaft getestet. Siehe: Kurt Mehlhorn, Peter Sanders. „Algorithms and Data Structures – The Basic Toolbox“, Springer 2008, S. 179ff.

Lösungsende

[3 Punkte]

Lösungsvorschlag

Aufgabe 3. Online-Algorithmen: Schiffsverkehr [12 Punkte]

Sie haben sich zusammen mit Freunden ein kleines Segelschiff über den Sommer gemietet. Da das Schiff schon recht betagt ist, benötigen Sie vor jeder Ausfahrt eine Stunde, um das Schiff auf die Fahrt vorzubereiten. Sie könnten aber auch einmalig einen Tag Arbeit (24 Stunden) in das Schiff stecken, um es zu überholen. Danach könnten Sie mit nur 15 Minuten Vorbereitung in See stechen. Da das Wetter diesen Sommer allerdings sehr durchwachsen ist und nicht klar ist, wie oft Sie überhaupt segeln gehen werden, stellt sich die Frage ob bzw. wann sich die Überholung des Segelschiffs lohnt.

a. Wenn Sie das Schiff nach der Hälfte des Sommers überholen, was ist der schlechteste kompetitive Faktor (*competitive ratio*), der sich ergeben kann? [2 Punkt]

Lösung

Der schlechteste competetive ratio ist ∞, wenn das Schiff zwar überholt, aber nie benutzt wird. Weder vor Überholung noch danach. Ein optimaler Algorithmus würde das Schiff dann nie überholen.

$$\frac{Alg_{worse}}{Alg_{opt}} = \frac{24h}{0h} = \infty$$

Lösungsende

b. Geben Sie die optimale Offline-Strategie an? [3 Punkt]

Lösung

Beachte, dass eine optimale Strategie entweder sofort das Schiff überholt oder gar nicht, also $cost(0) = 24h + 0,25h \cdot n$ und $cost(\infty) = 1h \cdot n$. Für beide Möglichkeiten gilt, dass Ihre Kostenfunktionen linear in n sind und dass es sich ab einem gewissen $n = n_0$ lohnt die 24h zu investieren. Setzt man beide Kostenfunktionen gleich, ergibt sich

$$\begin{aligned} 24h + 0,25h \cdot n &= 1h \cdot n \\ 24h &= 0,75h \cdot n \\ 32 &= n \end{aligned}$$

als Punkt ab dem es sich lohnt das Schiff zu überholen. Damit folgt als optimale Strategie in Abhängigkeit von n.

$$OPT(n) = \begin{cases} \text{nicht überholen} & n < 32 \\ \text{sofort überholen} & else \end{cases}$$

Lösungsende

(weitere Teilaufgaben auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 3

c. Sie überlegen sich, dass Schiff zu überholen, sobald Sie n Ausfahrten gemacht haben. Dabei ist n nicht von vorn herein fest gelegt. Analysieren Sie den kompetetiven Faktor dieser Strategie in Abhängigkeit von n. [4 Punkte]

Lösung

Die gewählte Strategie in Abhängigkeit von n ist das Segelschiff immer zum letztmöglichen Zeitpunkt zu überholen. Für diese Strategie sind die Kosten

$$cost(n) = n \cdot 1h + 24h.$$

Der kompetetive Faktor einer von n abhängigen Strategie ist also

$$comp(n) = \frac{n \cdot 1h + 24h}{\min\left[(24 + 0,25 \cdot n); (1 \cdot n)\right]}$$

Lösungsende

d. Geben Sie den Wert für n an, bei welchem der kompetetive Faktor optimal ist [3 Punkte]

Lösung

Gesucht wird der Punkt der die Funktion $comp(n)$ minimiert. Das ist offensichtlich das Minimum der Funktion $\min\left[(24 + 0,25 \cdot n); (1 \cdot n)\right]$ und ist wie bereits berechnet $n = 32$.

Lösungsende

Lösungsvorschlag

Aufgabe 4. Algorithmen-Entwurf: Größte Zahlen [6 Punkte]

Gegeben sei ein Stream aus Integerwerten, der über die Netzwerkschnittstelle empfangen wird. Der Stream, welcher nur einmal gesendet und nicht wiederholt wird, ist zu groß um ihn auch nur annähernd im Hauptspeicher oder auf Festplatte zu speichern.

Sie möchten die k größten Werte aus diesem Stream extrahieren, allerdings brauchen diese k Werte viel mehr Platz als der Hauptspeicher bietet. Sie müssen deshalb extern auf Platte gespeichert werden. Skizzieren Sie einen I/O-effizienten Algorithmus für das Problem. Wieviele I/Os werden benötigt? Begründen Sie kurz.

Lösung

Eine externe Priority-Queue wird benutzt. Ist das aktuelle Element größer als das Minimum wird es eingefügt und eine DeleteMin-Operation ausgeführt.

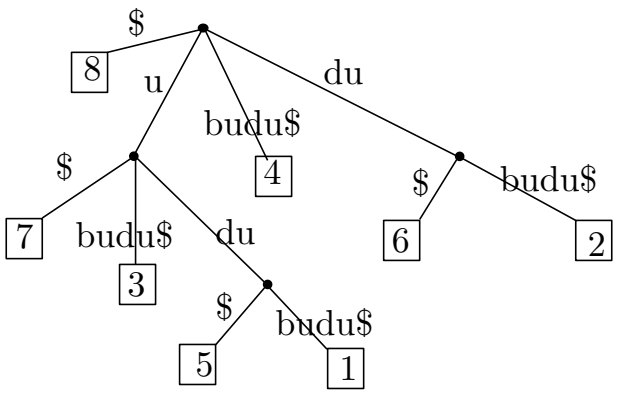
- Vergleich des aktuellen Elements: Kein I/O notwendig, denn es braucht nur konstanten Hauptspeicher. Das kleinste Element wird also immer vorgehalten im Speicher.
- Einfügen eines neuen Elements: Insert in $\approx \frac{2n}{B}(1+\lceil \log_{M/B} \frac{n}{M} \rceil)$ I/Os, DeleteMin: "amortisiert umsonst", Folie 275

Lösungsende

Lösungsvorschlag

Aufgabe 5. Suffix-Arrays und -Bäume [6 Punkte]

Gegeben sei folgender Suffix-Baum für einen String S. Es gilt $\$ < \mathtt{a} < \mathtt{b} < \ldots < \mathtt{z}$.



a. Geben Sie S an. [1 Punkte]

Lösung

ududubudu$

Lösungsende

b. Geben Sie das Suffix-Array für S an. [2 Punkte]

Lösung

i	$SA[i]$	Suffix
1	8	$
2	4	budu$
3	6	du$
4	2	dubudu$
5	7	u$
6	3	ubudu$
7	5	udu$
8	1	ududubudu$

Akzeptiert werden Lösungen, die mindestens die Spalte $SA[i]$ oder die sortierte Liste der Suffixe angeben.

Lösungsende

(weitere Teilaufgabe auf dem nächsten Blatt)

Lösungsvorschlag

Fortsetzung von Aufgabe 5

c. Geben Sie das Ergebnis der Burrows-Wheeler Transformation für S an.
Hinweis: Achten Sie darauf das Abschlusszeichen \$ zu verwenden.

[3 Punkte]

Lösung

Die BWT ist die letzte Spalte aus der lexikographischen Sortierung aller (zyklischen) Rotationen des Originalstrings:

i	Rotationen	Sortiert	Output
1	`udududu$`	`$udududu`	u
2	`dududu$u`	`dudu$udu`	u
3	`ududu$ud`	`du$ududu`	u
4	`dudu$udu`	`dududu$u`	u
5	`udu$udud`	`u$ududud`	d
6	`du$ududu`	`ududu$ud`	d
7	`u$ududud`	`udu$udud`	b
8	`$udududu`	`udududu$`	\$

Lösungsende

Lösungsvorschlag

Aufgabe 6. Algorithmen-Entwurf: Dyck-Sprache [8 Punkte]

Die Sprache der *wohlgeformten* Klammerausdrücke ist rekursiv definiert als:

- () ist wohlgeformt,
- sind K_1 und K_2 wohlgeformt, so sind dies auch (K_1), (K_2) und K_1K_2.

Geben Sie einen parallelen Algorithmus an, der in $O(\log p + n/p)$ prüft, ob ein gegebener Ausdruck A wohlgeformt ist. Begründen Sie die Ausführungszeit.

Lösung

Beobachtung: An jeder Stelle $A[i]$ in einem wohlgeformten Klammerausdruck gilt, dass die Zahl der $')'$ im Interall $[A[0]\ldots A[i]]$ immer kleiner gleich der Zahl der $'('$ ist.
Mit anderen Worten: **Es gehen nie mehr Klammern zu als zuvor aufgegangen sind**.
Nun wird die Präfixsumme für jede Stelle von A berechnet, wobei $'('$ $+1$ und $')'$ -1 entspricht.
Für eine wohlgeformte Klammerung muss gelten:

- Die Präfixsummen $S_A[i], 0 \leq i \leq n$ sind nie kleiner Null,
- $S_A[n-1] = 0$, $n = |A|$.

Die parallele Rechenzeit wird von der Zahl der Kommunikationsschritte dominiert. Für p Prozessoren ist diese für eine Präfixsumme gleich $O(\log p)$. Die lokale Überprüfung kostet noch einmal n/p Rechenzeit.

Alternative Idee:

Auf jeder CPU berechne:

- Zahl fehlender öffnender Klammern links, und
- Zahl fehlender schließender Klammern rechts.

Benachbarte Ausdrücke werden reduziert, wobei jeweils die Hälfte der CPUs nach einem Schritt fertig ist.

Achtung: Sonderfall leeres Wort.

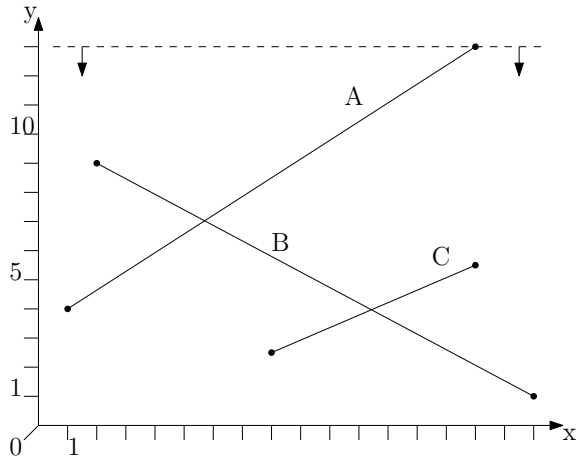
Lösungsende

Lösungsvorschlag

Aufgabe 7. Rechenaufgabe: Streckenschnitt [9 Punkte]

Im Folgenden ist ein Zustand des aus der Vorlesung bekannten *plane sweep* Algorithmus zum Finden von Streckenschnitten abgebildet. Links sehen Sie eine graphische Repräsentation, rechts den Inhalt der Prioritätswarteschlange Q und der sortierten Sequenz T. Q enthält die abzuarbeitenden Ereignisse als Tripel (y-Position, Typ, betroffene Strecke(n)), höchste y-Position zuerst. T enthält die von der *sweep line* geschnittenen Strecken in zunehmender x-Position.

Tragen Sie in die vorbereiteten Tabellen, die Zustände von Q und T für die folgenden drei Iterationen des Algorithmus ein. Zeichnen Sie ebenfalls die Sweep-Line ein. Q darf durch eine sortierte Liste repräsentiert werden.



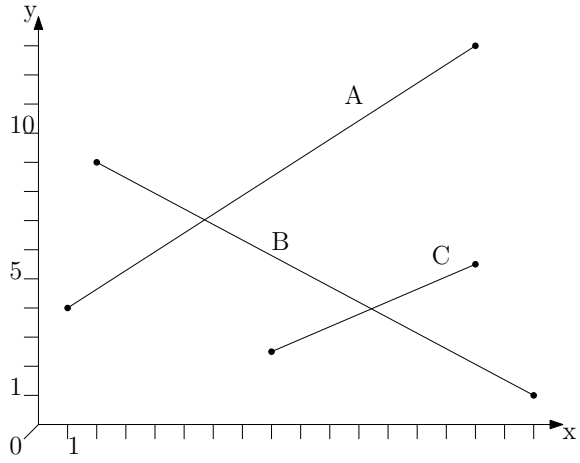
Q:

(9, start, {B})
(5, start, {C})
(4, finish, {A})
(2, finish, {C})
(1, finish, {B})

T:

A

Schritt 0 (gegeben)



Q:

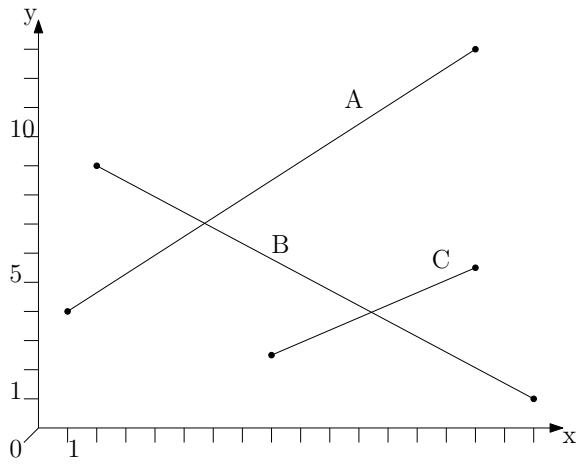
T:

Schritt 1

Fortsetzung auf der nächsten Seite

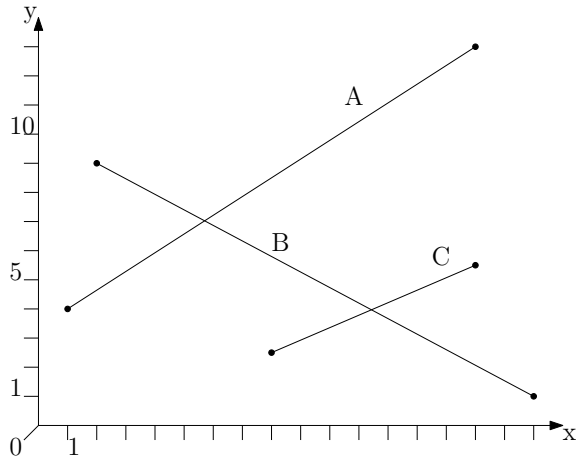
Lösungsvorschlag

Fortsetzung von Aufgabe 7



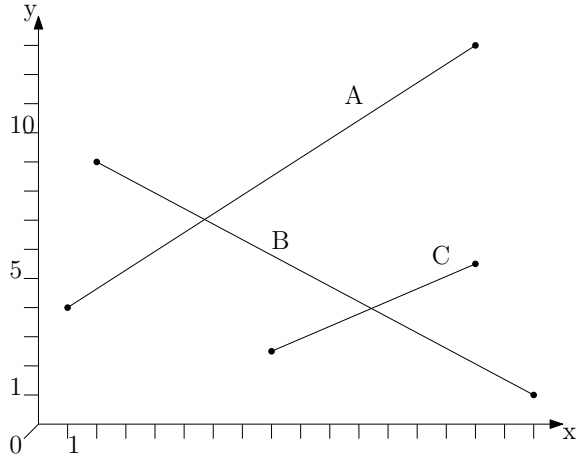
Q: T:

Schritt 2



Q: T:

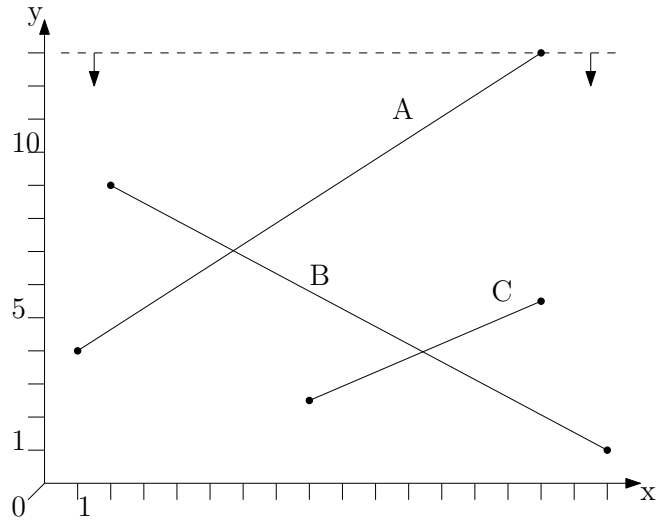
Schritt 3



Q: T:

zusätzlich, bei Bedarf verwenden

Lösung



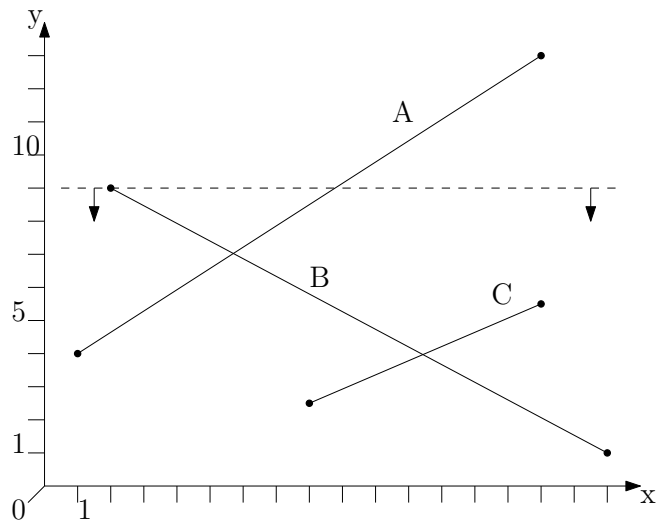
Q:

(9, start, {B})
(5, start, {C})
(4, finish, {A})
(2, finish, {C})
(1, finish, {B})

T:

A

Schritt 0 (gegeben)



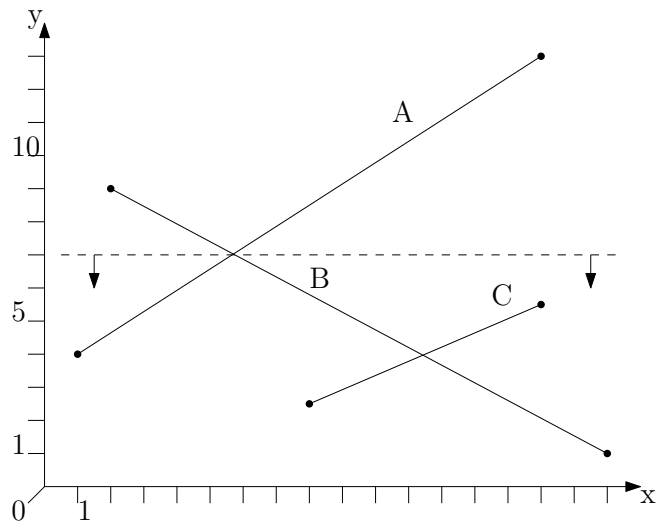
Q:

(7, i.sect, {B,A})
(5, start, {C})
(4, finish, {A})
(2, finish, {C})
(1, finish, {B})

T:

B
A

Schritt 1



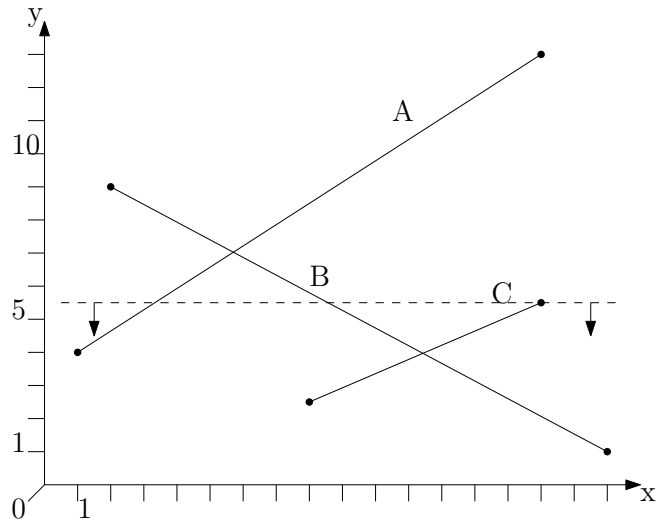
Q:

(5, start, {C})
(4, finish, {A})
(2, finish, {C})
(1, finish, {B})

T:

A
B

Schritt 2



Q:

(4, finish, {A})
(4, i.sect, {B,C})
(2, finish, {C})
(1, finish, {B})

T:

A
B
C

Schritt 3

Lösungsende