Datenstrukturen und Algorithmen, SS 2024
Prof. Dr. P. Rossmanith
D. Cloerkes, M. Gehnen, D. Mock, N. Speetzen

Theoretische Informatik | RWTH AACHEN UNIVERSITY

Abgabe: 05.07.2024

Übungsblatt mit Lösungen 11

Aufgabe T37

Wiederholen Sie die folgenden Konzepte (jeweils bezogen auf die euklidische Ebene):

a) Euklidische Ebene

b) Punkt

c) Segment

d) Polygon

Lösungsvorschlag

a) $\mathbf{R}^2$

b) $(x_0, x_1) \in \mathbf{R} \times \mathbf{R}$

c) Alle Punkte zwischen zwei gegebenen Punkten (Linearkombination)

d) Alle Punkte zwischen den gegebenen Punkten (Konvexkombination)

Aufgabe T38

Sei $X \subseteq \mathbb{R}^n$ eine endliche Menge an Punkten im n-dimensionalen Raum. Geben Sie einen Algorithmus an, der die Punkte $x, y \in X$ $(x \neq y)$ findet, sodass deren euklidische Distanz $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$ minimal ist. Ihr Algorithmus sollte höchstens eine Laufzeit von $O(n \cdot |X|^2)$ haben.

Lösungsvorschlag

Wir berechnen einfach alle Distanzen von zwei Punkten in X. Anschließend wählen wir einfach das Paar mit der kleinsten Distanz. Wegen der Monotonie der Wurzelfunktion muss die Wurzel nicht gezogen werden; es können auch die quadrierten Instanzen verglichen werden.
Die Distanz zu berechnen geht in $O(n)$ Laufzeit. Wir haben $|X|^2$ viele Paare. Für jedes dieser Paare müssen wir deren Distanz zueinander berechnen. Sobald wir alle Distanzen berechnet haben, können wir linear in der Anzahl der Distanzen das Minimum suchen. Insgesamt haben wir also eine Laufzeit von $O(n \cdot |X|^2)$.

Aufgabe T39

Entwerfen Sie einen Algorithmus, der für zwei gegebenen Polygone entscheidet, ob eines im anderen enthalten ist. Die Polygone sind gegeben als Tupel von Punkten in der Reihenfolge, in der sie auch durch die Kanten des Polygons verbunden werden. Die Laufzeit des Algorithmus soll $O(n \log n)$ betragen, wobei n die Anzahl der Punkte ist, die die Polygone definieren.
Hinweis: Sie dürfen annehmen, dass Sie (etwa durch Befragung eines Orakels) in konstanter Zeit einen Punkt ermitteln können, der von einem gegebenen Eckpunkt eines Polygons erreichbar ist, ohne einen Eckpunkt des anderen Polygons zu durchlaufen.

Lösungsvorschlag

Zunächst prüfen wir, ob die beiden Polygone einen nicht-leeren Schnitt haben, der aber auch nicht einem der beiden Polygone entspricht. Dies tun wir, indem wir mit der Sweepline-Technik die Kanten der Polygone auf Schnitte überprüfen. (Wichtig ist hierbei, dass die Segmente an je einem Eckpunkt offen definiert sein müssen, sodass sich zwei benachbarte Polygonkanten nicht schneiden.) Gibt es einen Schnitt, liegt keines der beiden Polygon im anderen. Dieser erste Schritt ist wegen der Sweepline-Technik in $O(n \log n)$ durchführbar.

Gibt es keinen Schnitt, müssen wir noch feststellen, ob ein Polygon im anderen liegt, oder ob beide schnittfrei sind. Dazu für wir für beide Polygone folgendes durch: Wir wählen einen Eckpunkt P und außerdem einen Punkt Q, der außerhalb beider Polygone liegt. Diesen Punkt Q erhalten wir durch Befragung des Orakels. Nun prüfen wir (wieder mit dem Sweepline-Verfahren), wie oft das Segment zwischen P und Q das andere Polygon schneidet*: Ist es eine gerade Anzahl, wurde das andere Polygon lediglich betreten und wieder verlassen (ggf. mehrmals), sodass P nicht im anderen Polygon liegen kann. Die Polygone sind demnach schnittfrei. Ist es eine ungerade Anzahl, liegt P im anderen Polygon und somit das ganze erste Polygon. Auch dieser zweite Schritt ist wegen der Sweepline-Technik ebenalls in $O(n \log n)$ durchführbar.

*: Wichtig ist sicherzustellen, dass das andere Polygon nicht in einem Eckpunkt geschnitten wird, denn dann wissen wir nicht, ob überhaupt ein Übergang von innen nach außen bzw. andersherum stattgefunden hat, oder ob das andere Polygon nur von außen berührt worden ist. Hierzu befragen wir einfach das Orakel.

Aufgabe T40

Zeigen sie, dass ein ungerichteter Graph mit n Knoten, in dem jeder Knoten mindestens einen Grad von $(n-1)/2$ besitzt, zusammenhängend sein muss.

Lösungsvorschlag

Angenommen der Graph sei nicht zusammenhängend, d.h. es gibt mindestens zwei Komponenten A und B. Eine der beiden Komponenten muss weniger oder genau die Hälfte an Knoten haben. Nehmen wir also an an $|A| \leq \lfloor (n/2) \rfloor$.

Der maximale Knotengrad eines Knoten in der Komponente A kann also höchstens $\lfloor (n/2) \rfloor - 1$ sein.

$$\left\lfloor \frac{n}{2} \right\rfloor - 1 \leq \frac{n}{2} - 1 < \frac{n}{2} - \frac{1}{2} = \frac{n-1}{2}$$

Damit ist gezeigt, dass die geforderte Bedingung, dass jeder Knoten Grad $(n-1)/2$ besitzt nicht erfüllt sein kann. Also muss der Graph zusammenhängend sein.

Aufgabe H35 (10 Punkte)
Geben Sie eine Folge von Union- und Find-Operationen an, die zu einem Baum der Höhe vier führt (vier Knoten übereinander). Verwenden Sie dabei sowohl die Rangheuristik als auch Pfadkompression.
Nehmen Sie an, dass bei der Operation $union(A, B)$ der Baum B in A eingehängt wird, falls beide gleichen Rang haben. Geben Sie das Zwischenergebniss nach jeder Operation an.

Lösungsvorschlag
Wir betrachten die Menge $\{1, \dots 16\}$. Da sich bei Benutzung der Rangheuristik der Rang nur erhöht, wenn wir *union* auf zwei Bäumen mit dem gleichen Rang anwenden, werden wir keine Bäume mergen die unterschiedlichen Rang haben.
Desweiteren wollen wir nicht, dass *find* auf Elemente ausgeführt wird die nicht in der Wurzel stehen, damit die Bäume nicht durch Pfadkompression geglättet werden.
Der erste durchlauf ist also: *union*(1,2), *union*(3,4), *union*(5,6), *union*(7,8), ..., *union*(15,16).
Um Bäume mit jeweils Rang eins zu erzeugen.
Und jetzt führen wir weitere merges benachbarter Bäume durch.
union(1,3), *union*(5,7), *union*(9,11), *union*(13,15).
union(1,5), *union*(9,13).
union(1,9).
Da sich bei jedem *union* der Rang um eins erhöht ist die Gesamthöhe danach vier.

Aufgabe H36 (15 Punkte)
Entwerfen Sie einen Algorithmus, der für eine gegebene Menge von n Kreisen mit gleichem Radius den kleinsten umschreibenden Kreis berechnet, also den kleinsten Kreis, in dem alle gegebenen Kreise enthalten sind. Die Laufzeit des Algorithmus soll in $O(n)$ liegen. Sie dürfen selbst bestimmen, mittels welcher (eindeutiger) Informationen die Kreise gegeben sein sollen. Begründen Sie die Korrektheit Ihres Algorithmus.
Hinweis: Es gibt Algorithmen, die den kleinsten umschreibenden Kreis für eine gegebene Menge von Punkten berechnen.

Lösungsvorschlag
Es gibt einen entsprechenden Algorithmus für den kleinsten umschreibenden Kreis für eine gegebene Menge von Punkten mit Laufzeit in $O(n)$. Diesen rufen wir für die Menge der Mittelpunkte der uns gegebenen Kreise auf. (Dafür benötigen wir also den Mittelpunkt eines jeden Eingabe-Kreises, weswegen wir die Kreise mittels Mittelpunkt und Radius r eindeutig gegeben bekommen wollen.) Der Mittelpunkt des kleinsten umschreibenden Kreises dieser gegebenen Mittelpunkte ist dann auch der Mittelpunkt des kleinsten umschreibenden Kreises der Kreise, seinen Radius nennen wir R. Der gesuchte Kreis hat dann denselben Mittelpunkt, aber den Radius $R + r$.
Korrektheitsbeweis per Widerspruch: Angenommen es gibt einen kleineren umschreiben Kreis um die gegebenen Kreise. Dann muss dieser einen Radius $R' < R + r$ haben, um kleiner zu sein. Wenn wir diesen um r verringern, müssen die Mittelpunkte der gegebenen Kreise immer noch im verkleinerten Kreis (mit Radius $R' - r$) liegen. Da aber $R' - r < R$, ist das ein Widerspruch zur Minimalität des kleinsten umschreibenden Kreises der Mittelpunkte der gegebenen Kreise. Also ist der konstruierte Kreis doch der kleinste umschreibende Kreis für die gegebenen Kreise.

Aufgabe H37 (15 Punkte)
Gegeben sei eine einfach verkettete Liste mit n Elementen, deren Länge sie nicht kennen. Entwerfen Sie einen Algorithmus, mit dem sich testen lässt, ob die Liste einen Kreis enthält. Zeigen Sie die Korrektheit ihres Algorithmus. Wie ist seine Laufzeit? Ist dieses auch in Zeit $O(n)$ möglich?

Lösungsvorschlag

Eine Möglichkeit sieht so aus: Mit zwei Zeigern l_1, l_2 durchlaufen wir die Liste. In jedem Schritt wird, falls möglich, l_1 um ein Element weiterverschoben, l_2 um zwei Elemente. Wenn dies nicht möglich ist und das Ende der Liste erreicht ist, liegt offenbar kein Zykel vor. Wenn dieses möglich ist, wird verglichen, ob $l_1 = l_2$ ist, also l_1 und l_2 auf das gleiche Element zeigen. Wenn ja, wurde ein Zykel gefunden.

Zur Analyse: Wenn die Liste keinen Zykel enthält, terminiert das Verfahren, sobald das Ende der Liste erreicht wurde. Wenn die Liste jedoch einen Zykel der Länge k enthält, dann ist nach höchstens $2n$ Iterationen die Abbruchbedingung $l_1 = l_2$ erfüllt: Nach höchstens $n - k$ Schritten befinden sich beide Zeiger bereits im Zykel. Nach höchstens k weiteren Schritten hat sich mindestens einmal die Situation ergeben, dass $l_1 = l_2$, denn irgendwann wird l_2 l_1 überholen. D.h., irgendwann wird die Situation entstehen, dass sie höchstens ein Feld weit auseinanderstehen, also $l_2 = l_1 - 1$. Dann gilt im nächsten Schritt aber $l_2 = l_1$.

Damit ist dann auch die Laufzeit in $O(2n) = O(n)$.