Max Wisniewski , Alexander Steen

Tutor: Lena Schlipf

Aufgabe 1

Sei P eine Menge von n Punkten in der Ebene, In der Vorlesung haben Sie einen Algorithmus kennengelernt, der ein engstes Paar von p in Zeit $O(n \log n)$ bestimmt. Dabei haben wir angenommen, dass alle x-Kooridnate in P verschieden sind. Zeigen Sie, wie man den Algorithmus anpassen kann, damit wir diese Annahme nicht mehr benötigen. Die Laufzeit soll immer noch $O(n \log n)$ betragen.

Lösung:
Das Problem, wenn wir Punkte mit der selben x-Koordinate zulassen, besteht darin, dass wir bei der Aufteilung anhand des Medians der x-Kooridnaten nicht mehr 2 gleich große Menge mit der halben Größe der Ursprünlgichen Menge erhalten. Im Extremfall, dass alle x Kooridnaten den selben Wert haben, hätten wir keine Partitionierung erreicht und müssten mit einem $\delta = \infty$ in den Conquer Schritt gehen, was für uns bedeuten würde, dass wir jeden Punkt mit jedem anderen vergleichen müssten.

Dies ist aber nicht nötig. Da wir den Split anhand der x - Achse des Medians vornehmen, können wir einfach die eine Hälfte der gleichen Werte in den linken Teil und die andere Hälfte in den rechten Teil tun und auf diesen beiden Mengen weiter rechenen. Wir erhalten ein δ zurück, mit dem wir die Rechtecke aus dem Beweis bilden können, und vergleichen nun je 10 Elemente paarweise alle Elemente aufsteigend nach y Koordinate durch.

Den paarweisen Vergleich haben wir schon im orginal Algorithmus gehabt, ändert also nichts an der Laufzeit. Bleibt also nur noch der andere Schnitt zu betachten.

Die optimierte Fassung sah vor, ein sortiertes Feld mitzuführen und im Divide Schritt die Grenzen zu verschieben, nähmlich immer auf die Mitte zwischen dem kleinsten und dem größten Index. Sollten wir nun mehrere gleiche Elemente als Median haben, so werden diese an dieser Stelle schon richtig geteilt und es wird korrekt fortgefahren.

Daraus ergibt sich, das er in der Vorlesung vorgestellte Algorithmus das Problem für nicht verschiedene x-Koordinaten schon gelöst hat. Dieser hatte wie in der Aufgabe gesagt die Laufzeit $O(n \log n)$, an der sich aufgrund mangelnder Veränderungen für den neuen Algorithmus nichts getan hat.

Damit haben wir ein Algorithmus für das Problem mit Laufzeit $O(n \log n)$

Aufgabe 2

Sei $G = (V, E)$ ein Gittergraph mit n Zeilen und n Spalten. Formal heißt das:
$V = \{\{(i, j), (i', j')\} | \, |i - i'| + |j - j'| = 1\}$. Desweiteren sei jeder Knoten $v \in V$ mit einer Zahl $x_v \in \mathbb{R}$ beschriftet, so dass die x_v paarweise verschieden sind. Ein *lokales Minimum* von G ist ein Knoten, dessen Beschriftung kleiner ist als die Beschriftung seiner Nachbern.

(a) Zeigen Sie, dass immer ein lokales Minimum existiert.

Lösung:
Der Graph G hat nach Definition n^2 viele Knoten. Dies bedeutet insbesondere, dass $W = \{x_v \mid v \in V\}$ eine endliche, nicht-leere Menge ist.
Für jede endliche, niicht-leere Teilmenge über einer total geordneten Grundmenge gilt, dass diese eine (globales) Minimum und Maximum besitzt[1]. Nun ist $\mathbb{R}$ eine total geordnete Menge und $W \subset \mathbb{R}$. Damit besitzt W ein (globales) Minumum.

Sei $m \in M$ nun der Knoten mit $x_m = \min W$. Da gilt, dass die Elemente aus W alle paarweise verschieden sind, gilt ins besonderen, dass
$\forall v \in V \, : \, x_v \leq x_m$ oder $\forall v \in V \setminus \{m\} \, : \, x_v < x_m$.

Dies gilt insbesondere auch für die Nachberknoten von m. Damit ist m auch sowohl *globales Minimum* als auch *lokales Minimum* in diesem Graphen.

(b) Der folgende Algorithmus heißt *lokale Suche*: Beginne bei einem beliebigen Knoten $v \in V$. Falls ein v ein *lokales Minimum* ist, sind wir fertig. Ansonsten hat v einen Nachbern w mit $x_w < x_v$ (wenn es mehr als einen solchen Nachbern gibt, wählen wir denjenigen mit dem kleinsten x_w). Gehe zu w und wiederhole, bis ein lokales Minimum erreicht ist.

Zeigen Sie, dass lokale Suche im schlimmesten Fall $\Omega(n^2)$ viele Schritte benötigt.

Lösung:

Um diese Laufzeit zu erreichen Konstruieren wir uns einen Fall, indem bei gegebenem $v \in V$ der Algorithmus jedes $winV$ besuchen muss.

(c) Zeigen Sie, wie man ein lokales Minimum in $O(n)$ Zeit finden kann. Begründen Sie Korrektheit und Laufzeit Ihres Algorithmus.

Lösung:

Die Idee dieser Lösung sieht wie folgt aus:
Wir legen in jedem Schritt ein Kreuz über das Feld, so dass es das gesammte Gitter in 4 (inetwa) gleichgroße Teile zerlegt. Als nächstes suchen wir uns das Minimum dieser $2 \cdot n$ Elemente, die auf dem Kreuz liegen. Von dem Knoten des Minimums aus,

[1] Wie in Mafi I gezeigt wurde

Aufgabe 3

Für eine gegebene Folge $M_1, M_2, ..., M_n$ von n Matrizen ist das *Matrizenkettenprodukt* $M_1 \cdot M_2 \cdot \cdots \cdot M_n$ zu berechnen. Die Matrizen haben dabei verschiedene Dimensionen. M_1 ist eine $(p_{M_1})_1 \times (p_{M_1})_2$ Matrix, M_2 ist eine $(p_{M_2})_1 \times (p_{M_2})_2$ usw.

Um eine $a \times b$ Matrix mit einer $b \times c$ Matrix zu multiplizieren, benötigen wir bekanntlich acb Multiplikationen und $ac\,(b-1)$ Additionen, also insgesammt $ac\,(2b-1)$ Additionen, also insgesamt $ac\,(2b-1)$ elementare Operationen.

(a) Bezeichne mit $P[i,j]$ die Kosten für eine optimale Klammerung des Matrizenkettenprodukts $M_i \cdot \cdots \cdot M_j$ $(1 \leq i \leq j \leq n)$. Unser Ziel ist, $P[1,n]$ zu berechnen. Finden Sie eine geeignete Rekursionsgleichung für $P[i,j]$.
Lösung

(b) Benutzen Sie Ihre Rekursion, um Pseudecode für einen Algorithmus anzugeben, welcher die optimalen Kosten $P[1,n]$ bestimmt. Analysieren Sie Laufziet und Platzbedarf.

(c) Erweitern Sie Ihren Algorithmus so, dass er auch eine optimale Klammerung ausgibt.

(d) Implementieren Sie ein Programm, das eine optimale Klammerung für die matrizenmultiplikation bestimmt und die Multiplikation durchführt. Vergleichen SIe Ihr Programm mit einer naiven Implementierung, welche die Matrizen von links nach rechts multipliziert.