Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

17. November 2016

Algorithmen & Datenstrukturen Lösungen zu Blatt P8 HS 16

Lösung P8.1 *Windräder.*

Für die Lösung dieser Aufgabe benützen wir Dynamische Programmierung: für alle $i \in \{0, \ldots, n\}$, sei m_i die beste Lösung wenn nur Positionen $\{d_1, \ldots, d_i\}$ benützt werden, also ist $m_0 = 0$ und wir interessieren uns für m_n.

Es gibt zwei verschiedene Arten wie die beste Lösung für Positionen $\{d_1, \ldots, d_i\}$ aussehen kann: entweder Windrad d_i wird benützt oder nicht. Wenn der Algorithmus Windrad d_i benützt, dann ist der Wert m_i die Addition von e_i and der besten Lösung vor d_i, welche genug weit entfernt ist; nennen wir diese m_z. Wenn dieses Windrad an Position d_i nicht benützt wird, dann haben wir $m_i = m_{i-1}$. Zusammen genommen gilt folgende Formel:

$$m_i = \max\{e_i + m_z, m_{i-1}\},$$

wobei z maximal ist, so dass $d_z \leq d_i - D$ (oder 0 falls $d_1 > d_i - D$). Nun können wir die Werte m_i vom m_1 bis m_n berechnen und müssen nur den passenden Wert z in jedem Schritt finden.

Ein Weg, um z zu finden, ist alle Werte $0, \ldots, i-1$ in jedem Schritt zu betrachten und d_z mit $d_i - D$ zu vergleichen, doch diese Vorgehensweise führt zu eine Laufzeit von $\mathcal{O}(n^2)$, zu langsam für die Testfälle. Wir können die Lösung verbessern, indem wir eine Binäre Suche auf $d_1, \ldots, i-1$ anwenden, um den letzten Wert vor $d_i - D$ (oder 0 falls kein solcher Wert existiert) zu finden, somit haben wir eine Laufzeit von $\mathcal{O}(n \log n)$.

Doch der effizienteste Weg, um z zu finden, ist auch der einfachste: Wir können beobachten, dass der Wert z für Schritt i mindestens so gross ist wie der Wert z, welcher zur Berechnung von m_{i-1} verwendet wurde. Also können wir einfach den vorherigen Wert z speichern und bei der Berechnung von m_i erhöhen wir z um 1 in einer `while` Schleife bis wir $d_{z+1} > d_i - D$ erreichen.

Diese Vorgehensweise führt zu einer Laufzeit von $\mathcal{O}(n)$ für das Problem: Es kann geschehen, dass wir in einem einzigen Schritt von d_{i-1} zu d_i z öfters erhöhen müssen (und dabei immer $d_{z+1} > d_i - D$ überprüfen), aber da z bei 0 startet und nie höher als $n-1$, können wir z höchstens n Mal erhöhen im ganzen Algorithmus, so die Laufzeit benötigt für diese `while` Schleife ist $\mathcal{O}(n)$ im ganzen Programm.

Lösungen

Auf der Vorlesungswebseite könnt ihr eine Lösung finden, welche eine Laufzeit von $\mathcal{O}(n)$ benötigt und den letzten erklärten Lösungsansatz verwendet. Die Implementierung der Lösung beinhaltet weitere Kommentare über die Implementation.

Es gibt eine zweite Lösung, welche einen alternativen Ansatz mit Dynamischer Programmierung beschreibt: sei t_i die beste Lösung von Windrädern an Positionen $d_1, \ldots, d_i$, welche Windrad d_i

benützt. Bei der Berechnung von t_i lassen wir $t_i = e_i + \max\{0, t_1, t_2, \ldots, t_z\}$, wobei z dieselbe Definition hat wie im obigen Ansatz. Diese Lösung betrachtet alle t_j für jeden Schritt, was eine Laufzeit von $\mathcal{O}(n^2)$ zur Folge hat. Doch auch bei dieser Lösung kann man eine Laufzeit von $\mathcal{O}(n)$ erreichen, wenn man nicht nur das zuletzt verwendete z sondern auch das vorhergehende Maximum von $\{0, t_1, t_2, \ldots, t_z\}$ speichert und zusammen mit der Erhöhung von z aktualisiert. Wir überlassen die Details dem Leser oder der Leserin.

Daten

Jeder Test im Judge beinhaltet 1-3 kleine Spezialfälle, beispielsweise Fälle mit nur einer Position, Fälle mit allen Positionen zu eng beieinander (also nur ein Windrad kann aufgestellt werden) oder Fälle mit allen Windrädern weit genug auseinander (alle können aufgestellt werden).

`judge1` $n = 100$, $D = 1\,000\,000$, alle Positionen in Konflikt.

`judge2` $n = 100$, $D = 1$, keine Positionen in Konflikt.

`judge3` $n = 100$, $D = 200\,000$, jede Position in Konflikt mit ca. $2/5$ der anderen.

`judge4` $n = 1\,000$, $D = 50\,000$, zufällige Positionen von $0 \ldots 1\,000\,000$.

`judge5` $n = 100\,000$, $D = 5\,000$, zufällige Positionen von $0 \ldots 1\,000\,000$. Ein quadratisches Programm sollte hier nicht funktionieren.