technische universität dortmund

SoSe 2018

M. Buchin
A. Krivošija, D. Mezlaf
L. Pfahler, A. Rey

DAP2 – Heimübung 5

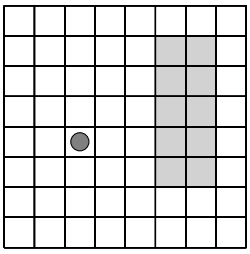
Ausgabedatum: 04.05.2018 — Abgabedatum: Mo. 14.05.2018 bis 12 Uhr

Abgabe:

Schreiben Sie unbedingt immer Ihren vollständigen Namen, Ihre Matrikelnummer und Ihre Gruppennummer auf Ihre Abgaben! Beweise sind nur dort notwendig, wo explizit danach gefragt wird. Eine Begründung der Antwort wird allerdings *immer* verlangt.

Aufgabe 5.1 (5 Punkte): (Teile und Herrsche)

Wir spielen Topfschlagen, aber ohne vom Sofa aufzustehen. Der Topf liegt in einem quadratischen Gebiet, das in kleine Quadrate aufgeteilt ist. Wir suchen ihn mit verbundenen Augen. Wir nennen die Seitenlänge n und nehmen an, dass n eine Zweierpotenz ist. Im folgenden Beispiel ist $n = 8$:



Bei der Suche hilft uns ein Orakel, das die Position des Topfes kennt, uns diese aber nicht direkt verrät. Das Orakel beantwortet aber Fragen, ob der Topf in einem bestimmten Rechteck liegt. Oben ist ein Rechteck eingezeichnet, das wir anfragen könnten, und in dem der Topf nicht liegt.

Ein Rechteck kann man zum Beispiel durch die linke untere und rechte obere Ecke beschreiben. Eine Anfage an das Orakel kann man also durch zwei Ecken (i_1, j_1) und (i_2, j_2) mit $1 \leq i_1 \leq i_2 \leq n$ und $1 \leq j_1 \leq j_2 \leq n$ ausdrücken. Die Antwort ist dann ja, wenn sich der Topf an einer Position (i, j) mit $i \in \{i_1, \ldots, i_2\}$ und $j \in \{j_1, \ldots, j_2\}$ befindet, und nein sonst.

Wir möchten den Topf jetzt mit möglichst wenigen Anfragen an das Orakel finden.

a) Entwerfen Sie einen Teile-und-Herrsche Algorithmus mit der Name `Topfschlagen`, der den Topf findet, und beschreiben Sie ihn mit eigenen Worten. Die volle Punktzahl erreicht ein Algorithmus, der mit $\mathcal{O}(\log n)$ Anfragen an das Orakel auskommt und den Topf immer findet.

b) Geben Sie eine Implementierung Ihres Algorithmus in Pseudocode an. Das Orakel kann dabei mit `Orakel`(i_1, j_1, i_2, j_2) aufgerufen werden und antwortet mit 1, wenn der Topf im angegebenen Rechteck liegt, und mit 0, wenn der Topf dort nicht liegt.

c) Beweisen Sie die Korrektheit Ihres Algorithmus.

Aufgabe 5.2 (5 Punkte): (Teile und Herrsche)

Bob steht im Lernraum 4.029 und blickt über die Häuser Dortmunds. Er möchte die Skyline, also den Umriss oder die Kontur der Stadt, zeichnen, doch er ist kein Künstler, sondern Informatiker.

Er abstrahiert: Die Stadt besteht aus einer Menge von n rechteckigen, sich teils überlappenden Häusern, wobei das i-te Haus durch das Tupel (x_i, b_i, h_i) beschrieben ist. Dabei ist x_i die horizontale Position des linken Rands des Hauses, $b_i > 0$ die Breite und $h_i > 0$ die Höhe. Ein Punkt (x, y) mit $y > 0$ liegt genau dann unter der Skyline, wenn (x,y) in mindestens einem der n Häuser liegt.

Bob verwendet eine Zeichen-Funktion, die zwei Punkte (x, h) und (x', h') mit $x' > x$, verbindet, indem zunächst eine Horizontale von (x, h) nach (x', h) gezeichnet wird und dann eine Vertikale von (x', h) nach (x', h'). Er braucht also einen Algorithmus, der, gegeben eine Liste von n Häusern, eine Liste mit den Eckpunkten der Skyline ermittelt. Mit der Laufzeit seines ersten Algorithmus, der ein Haus nach dem anderen in die Skyline integriert, ist er nicht zufrieden. Kann ein Teile-und-Herrsche Ansatz helfen?

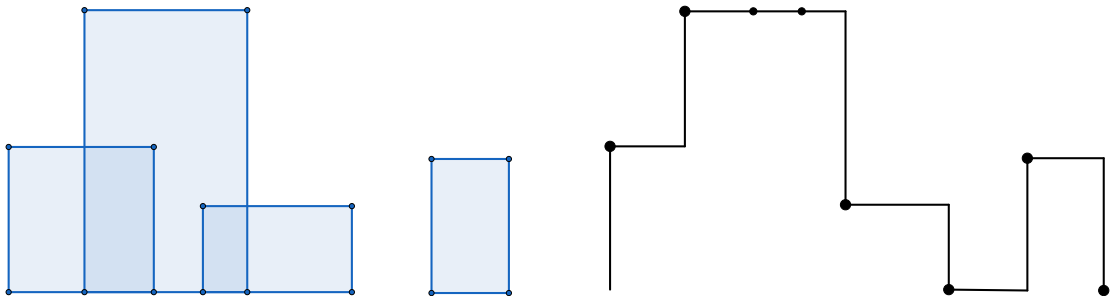


Abbildung 1: Beispiel: Links eine Menge von $n = 4$ Häusern, rechts die dazugehörige Skyline. Die dick markierten Punkte sind die ausgegebenen Ecken der Skyline (von links nach rechts). Die dünn markierten Punkte dürfen auch ausgegeben werden.

Wir nehmen an, dass keine Positionen doppelt vorkommen, also $x_i \neq x_j$ und $x_i + b_i \neq x_j$ für alle $i \neq j$ und dass alle Häuser unterschiedlich hoch sind. Weiterhin gilt, dass $n = 2^k$ mit $k \in \mathbb{N}_0$.

a) Entwickeln Sie einen Teile-und-Herrsche Algorithmus `Skyline`(X, B, H, i, n), der die Skyline berechnet. Beschreiben Sie die Funktionsweise und geben Sie eine Implementierung in Pseudocode an. Die volle Punktzahl erreicht ein Algorithmus, der Worst-Case-Laufzeit in $\mathcal{O}(n \log n)$ erreicht.
Tipp: Orientieren Sie sich an Merge-Sort.

n) Beweisen Sie, dass Ihr Algorithmus die geforderte Laufzeit erfüllt.

c) Beweisen Sie induktiv die Korrektheit des Algorithmus.