

## 4. Übungsblatt zu Algorithmen II im WS 2022/2023

[http://algo2.iti.kit.edu/AlgorithmenII\\_WS22.php](http://algo2.iti.kit.edu/AlgorithmenII_WS22.php)  
{sanders, moritz.laupichler, hans-peter.lehmann}@kit.edu

### Aufgabe 1 (Kleinaufgaben: Laufzeiten)

a) Sei  $T(n, \varepsilon)$  die Laufzeit eines Approximationsalgorithmus und  $g(n, \varepsilon)$  seine Approximationsgarantie. Geben Sie für die folgenden Fälle an, ob der Algorithmus ein PTAS, FPTAS oder keines von beiden ist. Begründen Sie Ihre Antwort jeweils kurz.

- $T_1(n, \varepsilon) = \frac{1}{\varepsilon} \cdot (4n^3 + n^2)$ ,  $g_1(n, \varepsilon) = (1 - \varepsilon)$
- $T_2(n, \varepsilon) = \frac{1}{\varepsilon} \cdot n^2$ ,  $g_2(n, \varepsilon) = (1 + 2\varepsilon)$
- $T_3(n, \varepsilon) = \sqrt{n} + n^{\frac{3}{2}}$ ,  $g_3(n, \varepsilon) = 2 + \frac{1}{n}$
- $T_4(n, \varepsilon) = n \cdot \log \frac{1}{\varepsilon}$ ,  $g_4(n, \varepsilon) = (1 - \varepsilon)$
- $T_5(n, \varepsilon) = \varepsilon + e^{\log n} + n^5$ ,  $g_5(n, \varepsilon) = (1 + \varepsilon)$
- $T_6(n, \varepsilon) = n^{\frac{1}{\varepsilon}} + n^5$ ,  $g_6(n, \varepsilon) = (2 + \varepsilon)$

Anmerkung: Für  $g_6(n, \varepsilon)$  können Sie annehmen, dass der Approximationsalgorithmus mit beliebigem  $\varepsilon \in (-1, 0)$  spezifiziert werden kann.

b) Sei  $f(n, k)$  die Laufzeit eines Algorithmus mit  $n$  der Eingabegröße des Problems und  $k$  ein beliebiger Parameter. Geben Sie an welche der folgenden Laufzeiten ein Problem *fixed-parameter-tractable* machen. Begründen Sie Ihre Antwort jeweils kurz.

- $f_1(n, k) = 3k^2n^2$
- $f_2(n, k) = n^k \cdot k^2 \cdot \sqrt{n^e}$
- $f_3(n, k) = 3n^2 + 2nk$
- $f_4(n, k) = e^k \cdot \sqrt{n}$
- $f_5(n, k) = e^n \cdot \sqrt{k}$
- $f_6(n, k) = k^3 \cdot \log n^2$

**Aufgabe 2** (Analyse+Rechnen: Vertex-Cover)

Gegeben sei folgender Algorithmus zur Berechnung eines *vertex cover*  $C$  für einen Graph  $G = (V, E)$ :

1. Initialisiere die Ergebnismenge  $C = \emptyset$  als leere Menge.
2. Wähle Kante  $(u, v) \in E$  beliebig.
3. Füge  $u, v$  zu  $C$  hinzu.
4. Entferne  $u, v$  und alle inzidenten Kanten aus  $G$ .
5. Wiederhole solange  $G$  noch Kanten hat

Nach Abschluss des Algorithmus ist  $C \subseteq V$  ein *vertex cover*, d.h. für jede Kante  $(u, v) \in E$  ist einer ihrer beiden Knoten in  $C$ . Falls o.b.d.A.  $u \in C$  sagt man auch Knoten  $u$  überdeckt Kante  $(u, v)$ .

- a) Zeigen Sie, dass der angegebene Algorithmus ein korrektes *vertex cover* berechnet.
- b) Geben Sie ein Beispiel an, in dem der Algorithmus ein minimales *vertex cover* liefert.
- c) Geben Sie ein Beispiel an, in dem der Algorithmus kein minimales *vertex cover* liefert.
- d) Zeigen oder widerlegen Sie, dass der Algorithmus eine 2-Approximation für *vertex cover* berechnet, d.h. dass er höchstens doppelt so viele Knoten auswählt als minimal nötig.

Betrachten Sie abschließend diesen alternativen Algorithmus zur Bestimmung einer 2-Approximation von *vertex cover*:

1. Initialisiere die Ergebnismenge  $C = \emptyset$  als leere Menge.
2. Wähle Knoten  $u \in V$  mit minimalem Grad.
3. Füge  $u$  zu  $C$  hinzu.
4. Entferne  $u$  und alle inzidenten Kanten aus  $G$ .
5. Wiederhole solange  $G$  noch Kanten hat

Der Algorithmus berechnet offenbar –mit ähnlichen Argumenten wie in Teilaufgabe (a)– ein *vertex cover*. Es bleibt folgende Frage zu beantworten:

- e) Zeigen oder widerlegen Sie, dass der Algorithmus eine 2-Approximation für *vertex cover* berechnet, d.h. dass er höchstens doppelt so viele Knoten auswählt als minimal nötig.

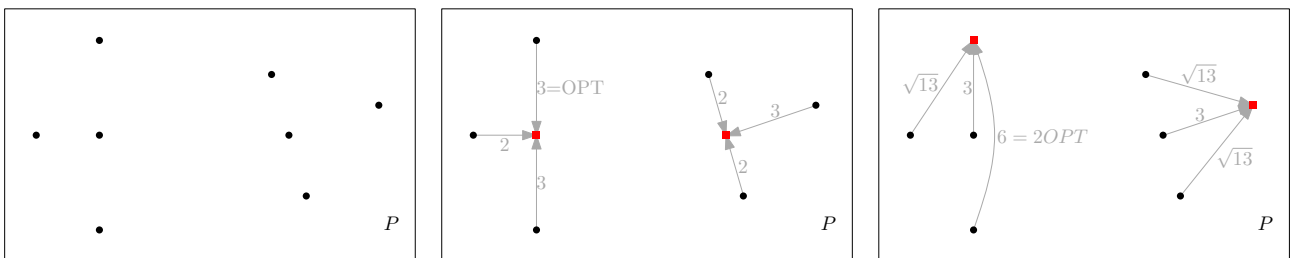
**Aufgabe 3** (Analyse: Metrisches  $k$ -Zentren Problem (\*))

Gegeben sei eine Menge an Punkten  $P \subset \mathbb{R}^2$  in der Ebene sowie eine Zahl  $k > 0$ . Gesucht ist eine  $k$ -elementige Teilmenge  $K \subset P$  dieser Punkte, genannt Zentren, so dass für jeden Punkt  $p \in P$  der maximale Abstand zu seinem nächstgelegenen Zentrum minimal ist.

Es existiert folgender *greedy* Algorithmus, der eine 2-Approximation des Problems berechnet:

1. Wähle beliebigen Punkt aus  $P$  als erstes Zentrum
2. Wähle Punkt aus  $P$  als nächstes Zentrum mit größter Entfernung zu allen bisherigen Zentren (d.h. der den maximalen kürzesten Abstand zu einem Zentrum besitzt)
3. Wiederhole bis  $k$  Zentren gewählt worden sind

Das folgende Beispiel veranschaulicht die Problemstellung für  $k = 2$ :



Links ist eine Punktmenge  $P$  abgebildet. In der Mitte ist eine optimale Lösung zu sehen. Die roten Quadrate sind die ausgewählten Zentren. Die Kanten geben das nächstgelegene Zentrum für jeden Knoten sowie den Abstand an. Rechts ist eine weitere aber suboptimale Lösung aufgezeigt.

Zunächst einige allgemeine Fragen zu diesem Algorithmus:

- a) Beschreiben Sie in Worten, welche Bedeutung  $OPT$  sowie die Aussage eine Lösung sei eine 2-Approximation des metrischen  $k$ -Zentren Problems, haben.
- b) Handelt es sich bei dem angegebenen Algorithmus um ein PTAS, ein FPTAS oder um keines von beiden? Begründen Sie kurz.

Im Folgenden soll gezeigt werden, dass der Algorithmus tatsächlich eine 2-Approximation berechnet. Dafür sind zunächst einige Vorüberlegungen nötig.

- c) Zeigen Sie, bei einer Auswahl von  $k + 1$  Punkten aus  $P$  existieren immer mindestens 2 Punkte, die das gleiche nächstgelegene Zentrum haben.
- d) Gegeben eine optimale Lösung, wie groß kann der Abstand zwischen zwei Punkten maximal sein, wenn diese das gleiche nächstgelegene Zentrum besitzen?
- e) In einer Lösung des *greedy* Algorithmus sei der maximale Abstand eines Punktes  $p \notin K$  zu seinem nächstgelegenen Zentrum  $> l$ . Zeigen Sie, dass  $l$  eine untere Schranke für den Abstand zwischen je zwei der Zentren  $k_i, k_j \in K, i \neq j$  der Lösung darstellt.
- f) Zeigen Sie mit obigen Aussagen, dass der angegebene *greedy* Algorithmus eine 2-Approximation für das Problem berechnet. Nehmen Sie dazu an, in der Lösung des Algorithmus sei der maximale

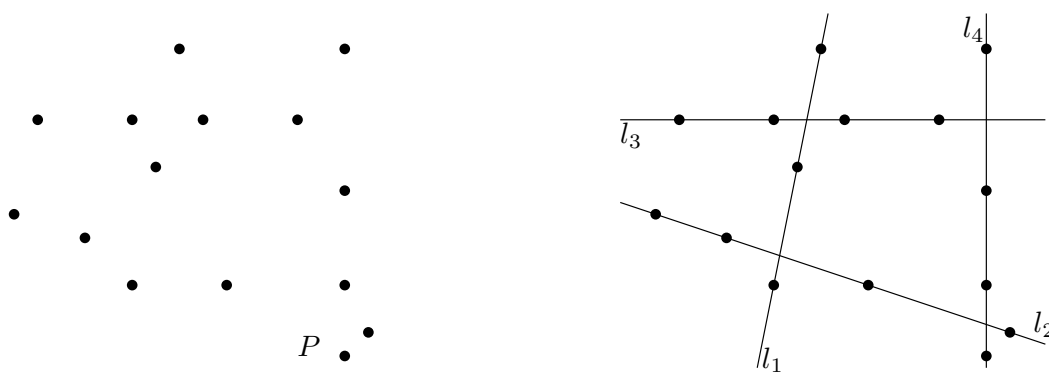
Abstand eines Punktes  $p \notin K$  zu seinem nächstgelegenen Zentrum  $> 2 \cdot OPT$ , und führen Sie diese Aussage zum Widerspruch.

**Hinweis:** Machen Sie zunächst eine Aussage über die paarweisen Abstände von  $k + 1$  speziell gewählten Punkten. Verwenden Sie anschließend einen Vergleich zu Abständen in der optimalen Lösung, um zum Widerspruch zu gelangen.

**Aufgabe 4** (Analyse: line shooting Problem)

Gegeben seien  $n$  Punkte  $P \subset \mathbb{R}^2$  in der Ebene sowie eine Zahl  $k > 0$ . Das *line shooting* Problem besteht darin zu bestimmen, ob es eine Menge  $L$  von  $k$  Geraden gibt, so dass jeder Punkt in  $P$  von mindestens einer dieser Geraden getroffen wird. Eine Probleminstanz wird durch das Tupel  $(P, k)$  charakterisiert.

In den Bildern sehen Sie links eine Punktmenge  $P$  und rechts eine mögliche Lösung für  $(P, 4)$ .



- Begründen Sie kurz, warum eine Gerade  $l$ , die mehr als  $k$  Punkte trifft, Teil einer Lösung für die Probleminstanz  $(P, k)$  sein muss bei beliebigem  $P$ .
- Begründen Sie kurz, warum die Instanz  $(P, 3)$  des *line shooting* Problems keine Lösung besitzt mit  $P$  wie in obiger Abbildung.
- Geben Sie einen Algorithmus an, der eine Instanz  $(P, k)$  des *line shooting* Problems exakt löst für beliebiges  $P$ . Bauen Sie dazu einen Suchbaum mit beschränkter Tiefe auf, der alle Kombination von  $k$  Geraden, die jeweils mindestens zwei Punkte treffen, generiert. Die Suchbaumtiefe und der Verzweigungsgrad sollen dabei polynomiell in  $k$ , der Aufwand pro Knoten polynomiell in  $n = |P|$  sein.

**Hinweise:** Verwalten Sie in jedem Knoten des Suchbaumes  $k$  Einträge, die jeweils bis zu zwei Punkte halten können (und damit eine Gerade definieren). Ein Suchbaum der Höhe  $O(k)$  genügt.

- Zeigen Sie, dass das *line shooting* Problem *fixed parameter tractable* bezüglich  $k$  ist. Geben Sie dazu die asymptotische Laufzeit Ihres Algorithmus in Abhängigkeit von  $n$  und  $k$  an.

### Aufgabe 5 (Entwurf+Analyse: findif-Anweisung)

Gegeben sei ein Array  $a[\cdot]$  im verteilten Speicher der  $n$  Objekte hält. Gesucht ist ein Algorithmus, der eine parallele **findif** Anweisung auf  $a[\cdot]$  ausführt. Die Anweisung sortiert die Elemente von  $a[\cdot]$  anhand eines Prädikats  $pred(\cdot)$ , so dass Elemente, die das Prädikat erfüllen, vorne stehen. Die relative Ordnung der Elemente untereinander soll dabei erhalten bleiben.

Bsp.: `findif( {1,4,9,7,3}, is_bigger_than_3 ) = {4,9,7,1,3}`

- Beschreiben Sie einen Algorithmus, der eine parallele **findif** Anweisung auf  $a[\cdot]$  möglichst schnell ausführt. Sie haben  $p = n$  Prozessoren zur Verfügung.
- Untersuchen Sie die Laufzeit der Anweisung für den Fall, dass  $p = n$  Prozessoren zur Verfügung stehen und das Prädikat in  $T(n) = O(1)$ ,  $O(\log n)$  bzw.  $O(n)$  ausgewertet werden kann.
- Wie verhalten sich die Laufzeiten, wenn Sie nur noch  $p < n$  Prozessoren zur Verfügung haben?

### Aufgabe 6 (Entwurf+Analyse: Assoziative Operationen)

Gegeben sei ein Array  $A$  im gemeinsamen Speicher bestehend aus  $n$  Objekten vom Typ  $X$ . Auf den Objekten sei eine Operator  $\odot$  definiert. Es sei nach dem Ergebnis von  $\odot_{i=1}^n a_i$  gesucht.

- Sei  $X = \mathbb{R}^2$  und der Operator definiert als

$$(x_1, x_2) \odot (y_1, y_2) := (x_1 y_1, x_2 + y_2)$$

Zeigen Sie, dass der Operator  $\odot$  assoziativ ist.

- Beschreiben Sie einen schnellen parallelen Algorithmus, der  $\odot_{i=1}^n a_i$  berechnet und geben Sie dessen Laufzeit  $T(n, p)$  an.
- Nun sei  $\odot$  wie folgt definiert:  $X$  beschreibe die Menge an möglichen Zeichenketten über dem Alphabet  $\{(\cdot)\}$ . Die Operation  $x \odot y$  verknüpfe beide Zeichenketten und schiebe alle öffnenden Klammern nach links, alle schließenden Klammern nach rechts ( Bsp.:  $()(\cdot) \odot (\cdot) = (((((\cdot)))) )$  ). Können Sie den selben Lösungsansatz wie in der vorherigen Teilaufgabe verwenden? Falls nein, geben Sie einen neuen parallelen Algorithmus an. Wie lange dauert die Ausführung?

### **Aufgabe 7** (Analyse: Externer Stack)

In der Vorlesung wurde eine Implementierung von *Stack* als externe Datenstruktur vorgestellt. Eine äquivalente Implementierung besitzt folgende Struktur: Im Speicher wird ein Puffer  $P$  der Größe  $2B$  gehalten –  $B$  sei die Blockgröße beim Zugriff auf externen Speicher. Der Puffer ist in Form eines (internen) Stacks organisiert und enthält die neuesten gespeicherten Elemente. Folgende Operationen sind für die externe Datenstruktur definiert:

- pop** Falls  $P$  nicht leer, entferne das neueste Element aus  $P$ . Ansonsten, lese einen Block ein, um die Hälfte von  $P$  zu füllen bevor **pop** auf  $P$  ausgeführt wird.
- push** Falls  $P$  nicht voll, füge das neue Element direkt zu  $P$ . Ansonsten, schreibe die ältere Hälfte von  $P$  in den externen Speicher und verschiebe die aktuellere Hälfte an diese Stelle im Speicher. Anschließend führe ein **push** auf  $P$  aus.

Für die Analyse können Sie davon ausgehen, dass ein Block  $B$  Elemente des Stacks halten kann.

- a) Zeigen Sie, dass die Operationen **push** und **pop** amortisiert  $O(1/B)$  I/O Operationen benötigen.
- b) Warum genügt es nicht, nur einen Puffer mit Größe  $B$  zu verwenden?

**Ausgabe:** 20.12.2022

**Besprechung:** 17.01.2023