Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Sebastian Lamm, Demian Hespe

5. Übungsblatt zu Algorithmen II im WS 2018/2019

http://algo2.iti.kit.edu/AlgorithmenII_WS18.php
{sanders, lamm, hespe}@kit.edu

Weihnachtsblatt – mit extra vielen Aufgaben :)

Aufgabe 1 (*Pflichtaufgabe (*)*)

a) Machen Sie Ihren Übungsleitern ein schönes Weihnachtsgeschenk.

Aufgabe 2 (*Analyse: Kürzeste Wege (Wiederholung)*)

a) Betrachten Sie eine Suche mit bidirektionalem Dijkstra. Geben Sie eine Familie von Graphen an, bei der ein ausgezeichneter Knoten u eine Anzahl an `decreaseKey` Operationen erfährt, die linear in der Länge des kürzesten Weges für jede mögliche Anfrage ist.

b) Bei manchen Algorithmen kann es sinnvoll sein, unterschiedliche Potentialfunktionen zu kombinieren. Zeigen Sie in diesem Zusammenhang: Sind π_1 und π_2 gültige Potentialfunktionen, so ist auch $\pi = \frac{\pi_1+\pi_2}{2}$ eine gültige Potentialfunktion.

c) Bei der Durchführung von *Dijkstras Algorithmus* können kürzeste Wege gespeichert werden, indem Vorgängerknoten gespeichert werden. Diese werden immer dann geändert, wenn eine bessere vorläufige Distanz gefunden wird. Dieses kann auch auf einem Graphen durchgeführt werden, der negative Kantengewichte enthält. Das Stopkriterium von Dijkstras Algorithmus muss dafür durch ein schwächeres Kriterium ersetzt werden: Es wird gestoppt, sobald keine Verbesserung mehr gefunden werden kann. Zeigen Sie, wenn auf diese Art ein Kreis entsteht, so ist dieser negativ.

d) (*) Dijkstras Algorithmus ist ein Spezialfall des allgemeinen *Labeling Algorithmus*. Der allgemeine Labeling Algorithmus wählt eine beliebige Kante aus, die das Label des Zielknotens verbessert. Geben Sie einen Graphen sowie eine Reihenfolge der Kanten an, so dass bei positiven Kantengewichten eine exponentielle Zahl von Schritten ausgeführt wird.

Hinweis: Achtung, schwierige Knobelaufgabe!

Aufgabe 3 (*Analyse:* preflow-push *Algorithmus (Wiederholung)*)

Sei durch S, T ein minimaler (s, t) Schnitt gegeben. Zeigen oder widerlegen Sie folgende Eigenschaften der Distanzfunktion:

a) $\forall v \in T : d(v) < n$

b) $\forall v \in S : d(v) \geq n$

Aufgabe 4 (*Analyse: Eigenschaften spezieller Knotenmengen*)

Gegeben sei ein ungerichteter Graph $G = (V, E)$. Sei $N(v) = \{w|(v, w) \in E\}$ die Menge aller Nachbarn von Knoten v. Man definiert folgende Teilmengen der Knotenmenge V des Graphen:

- *Vertex Cover (VC).* Ein VC ist eine Teilmenge $C \in V$, so dass f.a. Kanten $(u, v) \in E$ mindestens einer ihrer Endpunkte in C enthalten ist. Üblicherweise ist ein minimales VC gesucht.
- *Dominating Set (DS).* Ein DS ist eine Teilmenge $D \in V$, so dass f.a. Knoten $v \in V$ entweder v oder mindestens ein $w \in N(v)$ in D enthalten ist. Üblicherweise ist ein minimales DS gesucht.
- *Independent Set (IS).* Ein IS ist eine Teilmenge $I \in V$, so dass für jeden Knoten $v \in I$ gilt, alle Knoten $w \in N(v)$ sind nicht in I enthalten. Üblicherweise ist ein maximales IS gesucht.

Man unterscheidet weiter zwischen einer *minimum* (*maximum*) und einer *minimalen*(*maximalen*) Teilmenge. Erstere bezeichnet ein globales Optimum, d.h. auf diesem Graphen kann keine kleinere (größere) Menge mit der geforderten Eigenschaft gefunden werden. Letztere gibt ein lokales Optimum an, d.h. man kann keinen Knoten aus der Menge entfernen (zu der Menge hinzunehmen) und die geforderte Eigenschaft erhalten.

Zeigen oder widerlegen Sie, ...

a) ein *Independent Set* ist genau dann ein *Dominating Set*, wenn es *maximal* ist.

b) ein *minimum Dominating Set* ist ein *maximum Independent Set*.

c) ist C ein *minimales Vertex Cover*, so ist V/C ein *maximales Independent Set*.

d) ein *minimales Vertex Cover* mit allen isolierten Knoten von G ist ein *Dominating Set*.

e) ein *minimales Dominating Set* ohne isolierte Knoten ist ein *Vertex Cover*.

Aufgabe 5 (*Kleinaufgaben: Parallele Algorithmen*)

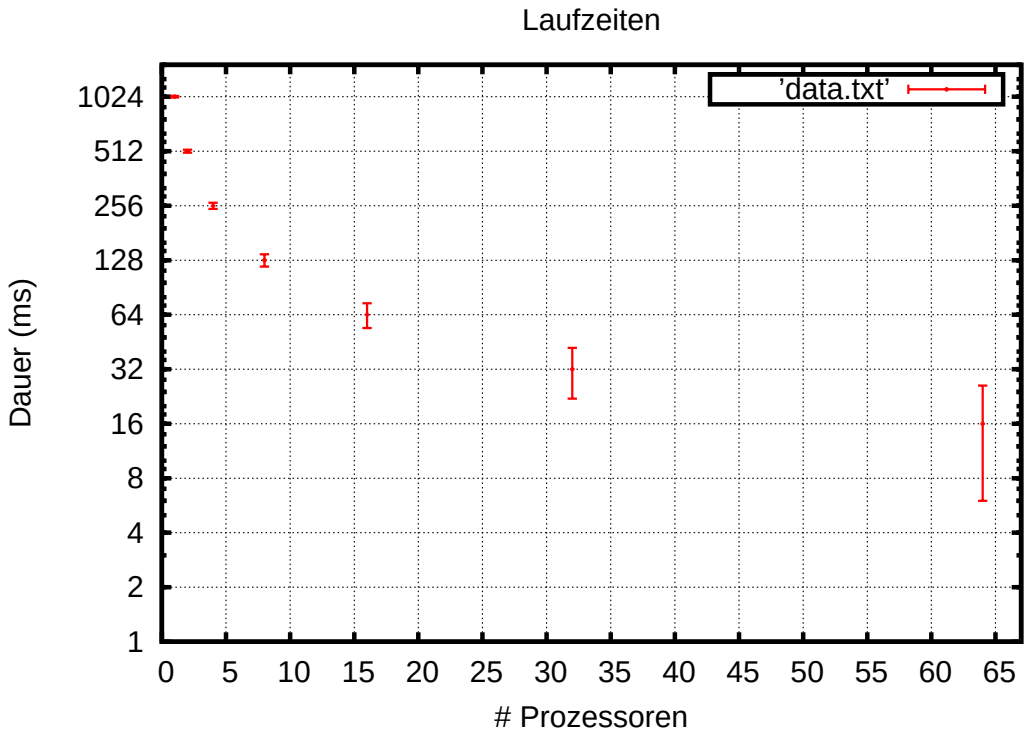
a) Gegeben sei ein paralleler vergleichsbasierter Sortieralgorithmus zum Sortieren von n komplexen Objekten auf p Prozessoren mit einer Laufzeit von

$$T(p) := \Theta(\frac{n^2 \log^2 n}{p^2}).$$

Geben Sie den absoluten *speed-up* und die *efficiency* an.

b) Wie muss in der vorherigen Teilaufgabe die Prozessorzahl p mit der Eingabegröße n wachsen, damit der absolute *speed-up* konstant bleibt?

c) Sie haben für einen parallelen Algorithmus folgende Laufzeiten bei unterschiedlicher Prozessorzahl gemessen. Was können Sie über die Skalierung dieses Algorithmus aussagen?


Laufzeiten
'data.txt'
Dauer (ms)
1024
512
256
128
64
32
16
8
4
2
1
0
5
10
15
20
25
30
35
40
45
50
55
60
65
Prozessoren

Aufgabe 6 (*Entwurf+Analyse: CRCW und CREW Modelle*)

Gegeben sei ein Array $a[\cdot]$ im gemeinsamen Speicher, der n Zahlen hält.

a) Beschreiben Sie einen möglichst schnellen parallelen Algorithmus, der überprüft, ob eine der Zahlen durch 7 teilbar ist. Gehen Sie von $p = n$ Prozessoren und dem CRCW *common* Modell aus. Außerdem sei die Teilbarkeit in $O(1)$ prüfbar.

b) Wie ändert sich die Laufzeit, wenn Sie nur noch $p < n$ Prozessoren zur Verfügung haben?

c) Wieviel Zeit würde Ihr Algorithmus im CREW Modell benötigen (wieder $p = n$ Prozessoren)?

d) Geben Sie den absoluten *speed-up* und die *efficiency* für die vorherigen Teilaufgaben an.

e) Im Folgenden soll berechnet werden, wieviele der Zahlen in $a[\cdot]$ durch eine andere Zahl in $a[\cdot]$ (nicht durch sich selbst!) teilbar sind. Sie haben das CRCW Modell und $p = n^2$ Prozessoren zur Verfügung.

Aufgabe 7 (*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}`

a) Beschreiben Sie einen Algorithmus, der eine parallele `findif` Anweisung auf $a[\cdot]$ möglichst schnell ausführt. Sie haben $p = n$ Prozessoren zur Verfügung.

b) 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.

c) Wie verhalten sich die Laufzeiten, wenn Sie nur noch $p < n$ Prozessoren zur Verfügung haben?

Aufgabe 8 (*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.

a) 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.

b) Beschreiben Sie einen schnellen parallelen Algorithmus, der $\odot_{i=1}^{n} a_i$ berechnet und geben Sie dessen Laufzeit $T(n, p)$ an.

c) Nun sei $\odot$ wie folgt definiert: X beschreibe die Menge an möglichen Zeichenketten über dem Alphabet $\{(,)\}$. Die Operation $x \odot y$ verknüpfe beide Zeichenketten und schiebe alle öffnenden Klammern nach links, alle schließenden Klammern nach rechts (Bsp.: $()(() \odot (()) = ((((()))) $).

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 9 (*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.

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

Anmerkung: Für $g_6(n, \epsilon)$ können Sie annehmen, dass der Approximationsalgorithmus mit beliebigem $\epsilon \in (-1, 0)$ 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 10 (*Analyse+Rechnen: Vertex-Cover*)

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

1. Initialisiere die Ergebnismenge $C = \varnothing$ 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 soviele Knoten auswählt als minimal nötig.

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

1. Initialisiere die Ergebnismenge $C = \varnothing$ 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 soviele Knoten auswählt als minimal nötig.

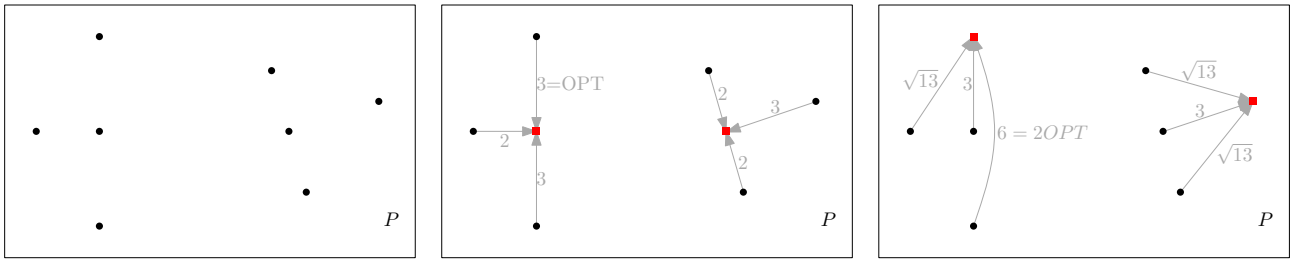
Aufgabe 11 (*Analyse: Metrisches k-Zentren Problem (*)*)

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 werdem, 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?

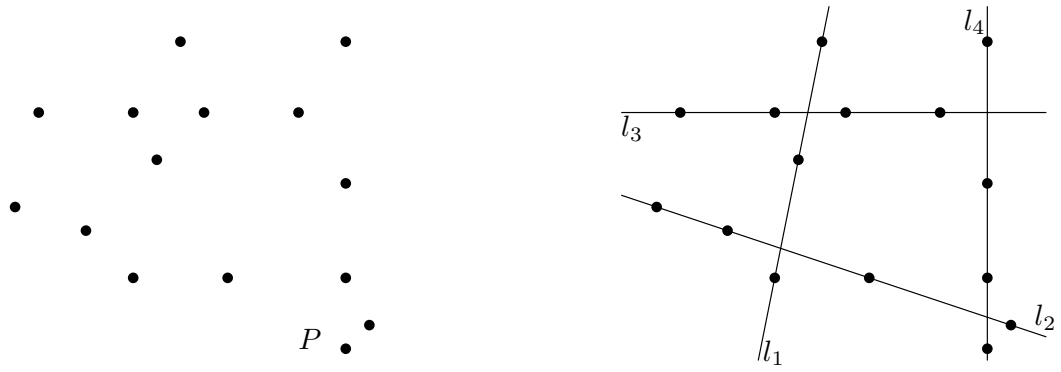
e) In einer Lösung des *greedy* Algorithmus sei der maximale Abstand eines Punktes $p \notin K$ zu seinem nächstgelegenen Zentrum $> l$. Zeigen Sie, dass l eine untere Schranke für den Abstand zwischen je zwei der Zentren $k_i,\ k_j \in K,\ i \neq j$ der Lösung darstellt.

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 12 (*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 $(P, 4)$.



a) 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.

b) Begründen Sie kurz, warum die Instanz $(P, 3)$ des *line shooting* Problems keine Lösung besitzt mit P wie in obiger Abbildung.

c) 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.

d) 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.

Ausgabe: 18.12.2016
Abgabe: keine Abgabe, keine Korrektur