Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Moritz Kobitzsch, Dennis Schieferdecker

5. Übungsblatt zu Algorithmen II im WS 2011/2012

http://algo2.iti.kit.edu/AlgorithmenII.php
{kobitzsch,sanders,schieferdecker}@kit.edu

Musterlösungen

Weihnachtsblatt – mit extra vielen Aufgaben :)

Aufgabe 1 (*Pflichtaufgabe (*)*)

a) Machen Sie Ihren Übungsleitern ein schönes Weihnachtsgeschenk.

Musterlösung:

a) Kameras, Objektive, Ultrabooks, Pads, Smartphones, Fahrräder, ...

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.

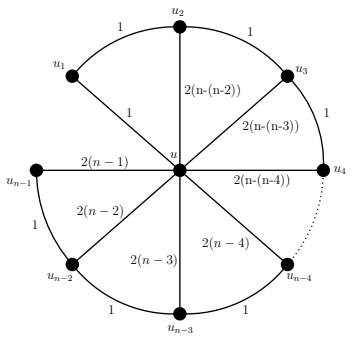
c) Bei der Durchführung von *Dijkstras Algorithmus* können kürzeste Wege gespeichert werden, indem Vorgängerknoten gespeichert werden. Diese werden immer dann geändert, wenn eine bessere vorläufige Distanz gefunden wird. Dieses kann auch auf einem Graphen durchgeführt werden, der negative Kantengewichte 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!

Musterlösung:

a) Der abgebildete Graph mit n Knoten ist ein mögliches Beispiel.



Jeder kürzeste Weg zwischen zwei Knoten geht entlang des äußeren Kreises. Wir bezeichnen den mittleren Knoten mit u und die Knoten auf dem Kreis mit u_1 bis u_{n-1} entsprechend der Abbildung. Betrachten wir den Pfad $u_i, \dots u_j$ mit $i > j$ und kürzester Weglänge $\mu(i,j) = i - j$. O.b.d.A sei $s = u_i$ und $t = u_j$. Die Vorwärtssuche führt zwischen $3(i-j)/4$ und $(i-j)/4$ Schritte entlang des Pfades aus (die Suchrichtungen wechseln sich ab und besuchen ggf. auch Knoten außerhalb des Pfades). Bis auf den ersten Schritt führt jeder dieser Schritte zu einer `decreaseKey` Operation auf Knoten u. Schritte außerhalb des Pfades führen zu keiner `decreaseKey` Operation auf Knoten u.

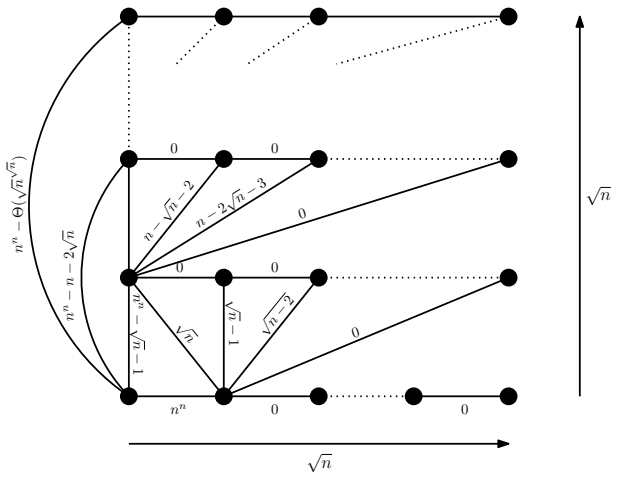
b) Es ist zu zeigen:

- π ist eine untere Schranke der Distanzfunktion: Es gilt: $\pi_1(u) \leq \mu(u,t)$, sowie $\pi_2(u) \leq \mu(u,t)$ für alle Knoten $u \in V$. Daraus folgt: $\pi_1(u) + \pi_2(u) \leq 2\mu(u,t)$ und damit $\pi \leq \mu(u,t)$.
- Die Kantengewichte sind nicht negativ: Für jede Kante $(u,v) \in E$ gilt: $c((u,v)) + \pi_1(v) \geq \pi_1(u)$, sowie $c((u,v)) + \pi_2(v) \geq \pi_2(u)$. Daraus folgt direkt: $2c((u,v)) + \pi_1(u) + \pi_2(u) \geq \pi_1(v) + \pi_2(v)$ und somit $c((u,v)) + \pi(u) \geq \pi(v)$.

Musterlösung:

c) Nehmen wir an, dass es einen solchen Kreis geben würde und dieser wäre nicht negativ. Betrachten wir nun den Moment, in dem der Kreis zum ersten Mal geschlossen wird. Dies geschehe an Knoten u. Der Kreis sei dabei bezeichnet als $k = \{u = n_1, \ldots, n_k = u\}$. Wenn der Kreis ein positives Gewicht hätte, so wäre $d(s,u) + k \geq d(s,u)$ und somit würden wir den Vorgänger von u nicht auf n_{k-1} setzen.

d) Der abgebildete Graph ist ein mögliches Beispiel.



Im allgemeinen Labeling Algorithmus kann die Ausführungsreihenfolge beliebig schlecht gewählt werden. Der angegebene Graph erlaubt es, immer die komplette untere Reihe von Knoten abzulaufen, und dann dem zweiten Knoten dieser Reihe eine um 1 kleinere Distanz zuzuweisen. Dabei kann jede der $\sqrt{n}$ Zeilen dazu genutzt werden, den ersten Knoten der darunter liegenden Reihe $\sqrt{n}$ mal eine kürzere Distanz zuzuweisen. Rekursiv lässt sich somit eine Bearbeitungsreihenfolge festlegen. Wir starten mit der Bearbeitung der untersten Zeile (in Serie). Wenn wir n Zeilen bearbeiten können, so können wir $n+1$ Zeilen bearbeiten, indem wir $\sqrt{n}$ mal den zweiten Knoten der obersten Reihe der n Reihen eine um 1 kürzere Distanz als bisher bekannt zuweisen und danach die Bearbeitung der n Reihen wiederholen. Somit ergibt sich eine Gesamtbearbeitungszeit von $\sqrt{n}^{\sqrt{n}}$ Schritten.

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:

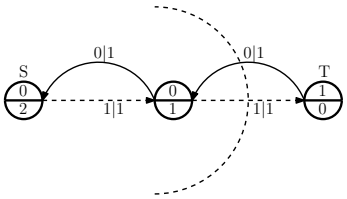
a) $\forall v \in T : d(v) < n$

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

Musterlösung:

a) Die Aussage ist wahr:
Angenommen es gibt einen Knoten $v \in T$ mit $d(v) \geq n$. Dann existiert im Residualgraph kein Weg von v zu t. Der entsprechende Fluss muss also zurück zu s geschoben werden. Daraus folgt aber auch direkt, dass der gegebene (s, t) Schnitt nicht minimal sein kann, da sonst der zusätzliche Fluss, der über den Schnitt geleitet wurde, zu t gelangen können muss (nach *max-flow-min-cut*-Theorem).

b) Die Aussage ist falsch:
Folgende Abbildung gibt ein Gegenbeispiel an.



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*.

Musterlösung:

a) Sei I ein *maximal* IS. I ist ein DS, da jeder Knoten $v \notin I$ zu einem Knoten $w \in I$ benachbart ist. Wäre dies nicht der Fall, so wäre I nicht maximal, denn man könnte v zum IS hinzunehmen. Damit ist auch gleichzeitig gezeigt, dass ein nicht-maximales IS kein DS ist.

b) Gegenbeispiel: In einer Kette von drei Knoten ist das *minimum* DS der mittlere Knoten. Das *maximum* IS sind die beiden äußeren Knoten.

c) $I = V/C$ ist ein IS. Wäre I kein IS, gäbe es eine Kante (u, v), so dass beide Knoten $u, v \in I$ wären. Damit wäre keiner der Eckpunkte der Kante (u, v) in C und C damit kein VC. Widerspruch! I ist ein *maximales* IS. Angenommen, Knoten v wäre nicht in I und man könnte ihn zu I hinzunehmen, um ein größeres IS zu erhalten. Dies würde bedeuten, dass kein Nachbar von v in I wäre. Damit wären $\{v\} \cup N(v)$ in C und damit C kein *minimales* VC, da man v aus C entfernen könnte und weiter ein VC hätte. Widerspruch!

d) Sei C ein *minimales* VC. Nach Definition ist mindestens ein Eckpunkt jeder Kante in C. Damit ist jeder nicht-isolierte Knoten entweder Teil von C oder zu einem Knoten in C benachbart. Also ist C ein DS für den Graph ohne isolierte Knoten. Fügt man alle isolierten Knoten zu C hinzu, so ergibt sich ein DS für G.

e) Gegenbeispiel: Betrachte die zu einem Dreieck verbundenen Knoten A, B, C. Ein *minimales* DS enthält genau einen dieser Knoten. Ein VC benötigt mindestens zwei der drei Knoten.

Aufgabe 5 (*Entwurf: Paketvermittlung*)

Der Weihnachtsmann ist spät dran mit dem Verteilen der Geschenke an seine Außenlager, von wo aus sie dann an Weihnachten in alle Wohnungen geliefert werden. Es ist ein ständiges Kommen und Gehen am Nordpol und der Lagermeister hat alle Hände voll zu tun. Gerade sind die folgenden Geschenke zum Versand freigegeben worden:

- 4× Lego City Feuerwehrlöschzug
- 1× Barbies Traumhaus
- 2× Ferngesteuerter Formel 1 Rennwagen von Ferrari
- 1× Rotkäppchen Kostüm
- 3× Wii Mario Edition (Red Edition)
- 3× Kosmos electronic Baukasten
- 1× Pony mit großen Kulleraugen

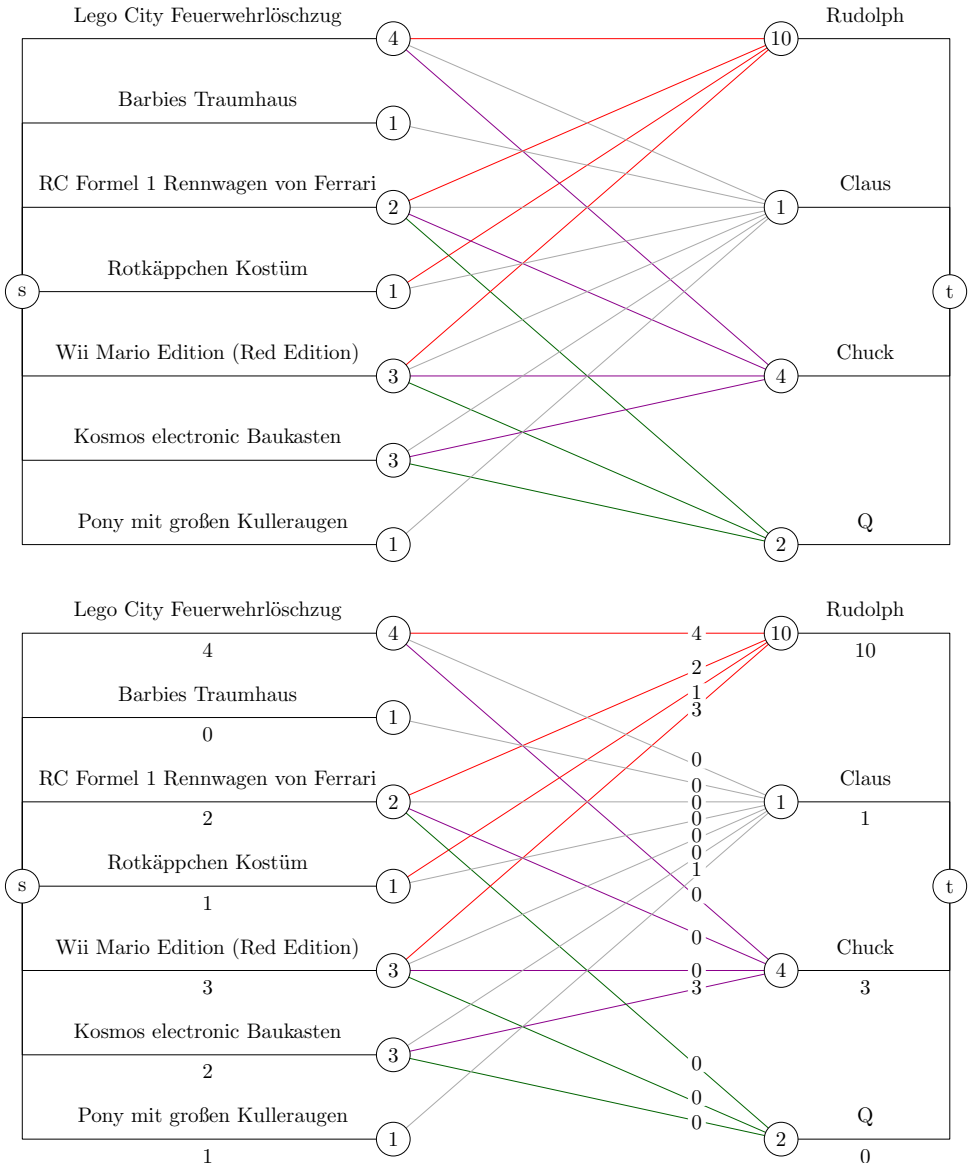
Es stehen auch einige Rentierschlitten zum Abtransport bereit. Allerdings haben die lieben Tiere so ihre Eigenarten, was es dem Lagermeister nicht leichter macht, seine Aufgabe effizient zu erfüllen.

- *Rudolph* kann 10 Pakete transportieren, allerdings weigert er sich alles zu transportieren, das nicht überwiegend rot ist.
- Der alte *Claus* kann nur noch 1 Paket mitnehmen, dafür ist er nicht wählerisch, was den Inhalt des Pakets angeht. Hauptsache es gibt nach getaner Arbeit einen Sack voll Hafer.
- *Chuck* transportiert kein Mädchenspielzeug. Das passt einfach nicht zu ihm, darüber hinaus auch nicht mehr als 4 Pakete.
- *Q* ist ein Technikfreak. Er liebt alles was Strom benötigt. Leider kann er nur 2 dieser Gerätschaften mit seinem tiefergelegten Schlitten ziehen.

Wie verteilt der Lagermeister diese Geschenke optimal auf die Schlitten? Können alle Geschenke mit nur einer Beladung jedes Schlittens abtransportiert werden?
Modellieren Sie die Problem als Flussgraph und beantworten Sie die obigen Fragen.

Hinweis: Das Flussproblem kann wieder durch scharfes Hinsehen gelöst werden. Es muss kein Flussalgorithmus ausgeführt werden.

Musterlösung:
Modellierung des *Matching* Problems als Flussgraph:
(Werte in Knoten sind Knotenkapazitäten, Werte an Kanten sind Flussmengen)



(*Chuck* steht zwar auf das Rotkäppchen Kostüm, für die Musterlösung zählen wir es aber trotzdem als Mädchenspielzeug)

Eine optimale Verteilung ist anhand des Flusses ablesbar. Jede mit Fluss f belegte Kante (Geschenk, Schlitten) gibt an, dass der jeweilige Schlitten mit f dieser Geschenke beladen wird. Da der Gesamtfluss kleiner als die Gesamtanzahl Geschenke ist, können die vorhandenen Schlitten nicht alle Geschenke auf einmal abtransportieren. Das kann man auch leicht wie folgt sehen: Das Pony und das Traumhaus werden beide nur von Claus transportiert. Er kann aber nur ein Geschenk auf einmal transportieren. Daher braucht man mindestens zwei Beladungen.

Aufgabe 6 (*Kleinaufgaben: Parallele Algorithmen*)

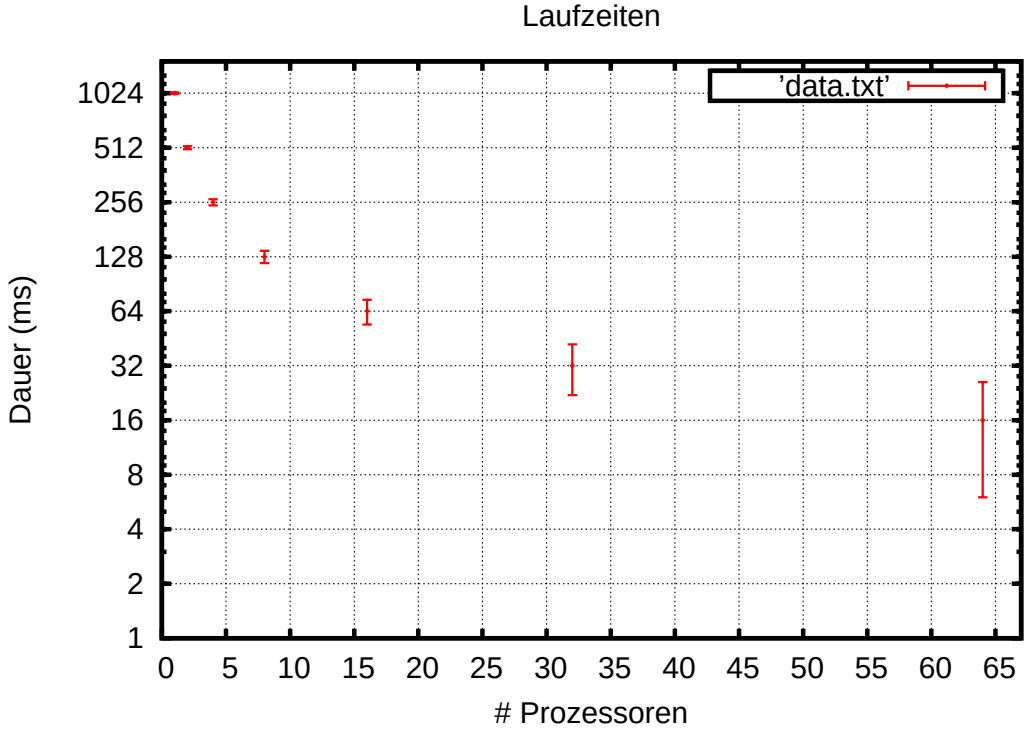
a) Gegeben sei ein paralleler vergleichsbasierter Sortieralgorithmus zum Sortieren von n komplexen Objekten auf p Prozessoren mit einer Laufzeit von

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

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?



Musterlösung:

a) Die besten sequentiellen vergleichsbasierten Sortierverfahren benötigen $T_{seq} = \Theta(n \log n)$ Laufzeit. Damit ergibt sich für den absoluten *speed-up*

$$S(p) = \frac{T_{seq}}{T(p)} = \frac{\Theta\left(n \log n\right)}{\Theta\left(\frac{n^2 \cdot \log^2 n}{p^2}\right)} = \Theta\left(\frac{p^2}{n \log n}\right).$$

Mit der Definition der *efficiency* ergibt sich

$$E(p) = \frac{S(p)}{p} = \frac{\Theta\left(\frac{p^2}{n \log n}\right)}{p} = \Theta\left(\frac{p}{n \log n}\right).$$

b) Für einen konstanten *speed-up* unabhängig von der Prozessoranzahl muss gelten

$$S(p) = \Theta(\frac{p^2}{n \log n}) \stackrel{!}{=} \Theta(1)$$

und damit muss die Anzahl Prozessoren mit der Eingabegröße nach $p = \Theta(\sqrt{n \log n})$ wachsen.

c) Ignoriert man die Fehlerbalken, so weisst der Algorithmus eine Halbierung der Laufzeit bei Verdopplung der Prozessoren auf. Der relative *speed-up* ist also

$$S_{rel}(p) = \frac{T(1)}{T(p)} = p.$$

Der absolute *speed-up* skaliert auch linear mit p, über den genauen Faktor und über die Skalierung mit n kann man keine Aussage treffen. Die *efficiency* ist unabhängig von p.

Aufgabe 7 (*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.

Musterlösung:

a) Die Variable zum Speichern des Resultats wird mit `result = false` initialisiert. Jeder Prozessor p_i überprüft für eine Zahl $a[i]$, ob diese durch 7 teilbar ist. Ist dies der Fall, setzt der Prozessor `result = true`. Ansonsten macht er nichts. Der Algorithmus läuft in $T(p) = O(1)$.

b) Jeder Prozessor muss sich seriell um n/p Speicherstellen kümmern. Ansonsten läuft der Algorithmus gleich ab und benötigt daher $T(p) = O(n/p)$ Zeit.

c) Im CREW Modell kann nicht mehr parallel auf eine Speicherstelle geschrieben werden. In einer einfachen Lösung würde für alle Prozessoren je ein Schreibzugriff auf `result` nacheinander ausgeführt werden. Dies würde $O(n)$ Zeit benötigen.

   Geschickter ist es, das Ergebnis für jede Zahl $a[i]$ in einem Array $b[\cdot]$ zu speichern - 0 für `false` und 1 für `true`. Dies geschieht in $O(1)$. Dann wird die Summe $\sum_{i=1}^{n} b[i]$ per Reduktion in $O(\log n)$ berechnet. Ist die Summe ungleich 0, so wird `result=true` gesetzt, sonst auf `false`. Die gesamte Laufzeit ist $T(p) = O(\log n) = O(\log p)$.

d) Der optimale sequentielle Algorithmus muss im schlimmsten Fall jede Zahl prüfen, benötigt also $T_{seq} = O(n)$. Damit ergibt sich für den absoluten *speed-up*

$$S(p) = \frac{T_{seq}}{T(p)} = O(n) = O(p) \text{ (a)}; \; O(p) \text{ (b)}; \; O(n/\log n) = O(p/\log p) \text{ (c)}.$$

   Mit der Definition von *efficiency* ergibt sich

$$E(p) = \frac{S(p)}{p} = O(1) \text{ (a)}; \; O(1) \text{ (b)}; \; O(1/\log p) \text{ (c)}.$$

e) Berechne Hilfsarray $b[\cdot]$. Es wird $b[i] =$ `true` gesetzt, wenn eine Zahl in $a[\cdot]$ von $a[i]$ geteilt wird, die nicht gleich $a[i]$ ist. Ansonsten wird $b[i] =$ `false` gesetzt. Dies geschieht analog zur ersten Teilaufgabe in $O(1)$ mit n Prozessoren für jedes i. Mit n^2 Prozessoren und n Einträgen in $b[\cdot]$ können alle Einträge in $O(1)$ berechnet werden. Anschließend wird über die Elemente in $b[\cdot]$ summiert, mit n Prozessoren und Reduktionsschema ist dies in $O(\log n)$ möglich. Die gesamte Laufzeit ist also $T(p) = O(\log n) = O(\log \sqrt{p})$.

Aufgabe 8 (*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?

Musterlösung:

a) Prozessor p_i prüft Bedingung $pred(a[i])$ und schreibt das Resultat als 0 (falsch) oder 1 (wahr) in ein neues Array $s[\cdot]$ an Stelle i. Anschließend wird die Präfixsumme über $s[\cdot]$ gebildet. Aus diesen Werten kann jeder Prozessor p_i die neue Position für sein Datenelement $a[i]$ ableiten:

- $a[i] \to a[s[i] - 1]$, wenn $a[i]$ das Prädikat erfüllt,
- $a[i] \to a[s[n-1] + i - s[i]]$, wenn $a[i]$ das Prädikat nicht erfüllt.

Prozessoren werden von 0 durchnummeriert.

b) Die Ausführungszeit beträgt $O(T(n))$ für die Berechnung von $s[\cdot]$, $O(\log n)$ für die Berechnung der Präfixsumme und $O(1)$ für die Umsortierung. Gesamt ergeben sich die Laufzeiten in Abhängigkeit von $T(n)$ zu $O(\log n)$, $O(\log n)$ bzw. $O(n)$, wobei $p = n$ gilt.

c) Stehen weniger als n Prozessoren zur Verfügung, muss man die Arbeit für jeweils n/p Objekte von einem Prozessor ausführen lassen. Es ergeben sich folgende Laufzeiten für die drei Schritte $O(n/p \cdot T(n))$, $O(n/p + \log p)$ und $O(n/p)$. Insgesamt ergeben sich die Laufzeiten wieder in Abhängigkeit von $T(n)$ zu $O(n/p + \log p)$, $O(n/p \cdot \log p)$ bzw. $O(n^2/p + \log p)$.

Aufgabe 9 (*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 $\bigodot_{i=1}^{n} a_i$ 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 $\bigodot_{i=1}^{n} a_i$ 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?

Musterlösung:

a) Der Operator ist offensichtlich assoziativ, da beide Koordinaten unabhängig voneinander sind und die ausgeführte Addition bzw. Multiplikation auf reelen Zahlen assoziativ ist.

b) Der Operator $\odot$ ist assoziativ und in konstanter Zeit ausführbar. Daher kann $\bigodot_{i=1}^{n} a_i$ mit Hilfe des Reduktionsschemas in $T(n, p) = O(n/p + \log p)$ berechnet werden. Für $p = n$ liegt die Ausführungszeit in $T(n, p) = O(\log n)$.

c) Auch hier ist $\odot$ assoziativ, benötigt aber Zeit proportional zur Länge beider Operanden (ein Scan über beide Operanden liefert die Anzahl öffnender und schließender Klammern, die anschließend geschrieben werden können). Das Reduktionsschema ist weiterhin anwendbar, die Laufzeit erhöht sich allerdings insgesamt zu $T(n, p) = O(p \cdot n/p + n \log p)$ bzw. zu $T(n, p) = O(n \log n)$ für $p = n$.

Ein schnellerer Lösungsansatz verwendet paralleles Sortieren. In diesem Fall ist die Laufzeit $T(n, p) = O(\frac{n \log n}{p} + \log^2 p)$ bzw. $T(n, p) = O(\log^3 n)$ für $p = n$.