Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Hans-Peter Lehmann, Daniel Seemaier

8. Übungsblatt zu Algorithmen II im WS 2021/2022

http://algo2.iti.kit.edu/AlgorithmenII_WS21.php
{sanders, hans-peter.lehmann, daniel.seemaier}@kit.edu

Musterlösungen

Aufgabe 1 (*Analyse: ADAC Mitgliedschaft*)

Der "**A**utomobil **D**urch **A**lgorithmiker **C**lub" (ADAC) leistet auf Autobahnen Pannenhilfe. Ein Autofahrer hat in seiner Zeit als Verkehrsteilnehmer n Pannen, $n \in \mathbb{N}_{\geq 0}$, für die er die Hilfe des ADAC in Anspruch nehmen muss. Für jede geleistete Pannenhilfe verlangt der Club eine Aufwandsentschädigung abhängig von der Schwere der Panne. Mitglieder beim ADAC müssen lediglich ein Viertel dieser Kosten bezahlen. Eine lebenslange Mitgliedschaft kann man sich durch eine Einmalzahlung in Höhe von 1000 DM (**D**ijkstra **M**ark) sichern.
Da nicht schon mit Erwerb des Führerscheins klar ist, wie viele Pannen man in seinem Leben haben wird und wie schwerwiegend diese sein werden, stellt sich die Frage, ab wann es sich lohnt eine Mitgliedschaft beim ADAC zu erwerben.

a) Geben Sie eine Strategie an, die einen kompetitiven Faktor (competitive ratio) von ∞ erreicht. Begründen Sie kurz.

b) Wie gut ist die Strategie, sich nie eine Mitgliedschaft beim ADAC zu sichern? Begründen Sie.

c) Zeigen Sie, dass folgende Strategie einen kompetitiven Faktor $c = 3$ hat. Die Strategie ist, sich beim Pannenhelfer eine Mitgliedschaft zu kaufen, wenn die momentan von ihm bearbeitete Panne die Gesamtausgaben für Pannen (ohne Mitgliedschaft) auf über 500 DM anheben würde.

Hinweis: Verwenden Sie die summierten Gesamtkosten K über alle Pannen (ohne Mitgliederrabatt).

Musterlösung:
Ein Algorithmus ALG wird als *streng c-kompetitiv* bezeichnet, wenn für alle Eingaben I gilt

$$c = \sup_I \frac{ALG(I)}{OPT(I)}$$

Dieser Wert wird als *kompetitiver Faktor* (*competitive ratio*) bezeichnet.

Der Übersichtlichkeit halber wird im Folgenden ohne Einheiten gerechnet:

a) Wenn man sofort mit Erhalt der Fahrerlaubnis eine Mitgliedschaft beim ADAC erwirbt, gibt man im schlimmsten Fall 1000 DM aus, nimmt aber die Hilfe des ADAC nie in Anspruch. Dies ergibt einen kompetitiven Faktor von $c = \frac{1000}{0} = \infty$.

b) Wenn man sich nie eine Mitgliedschaft kauft, gibt man offensichtlich höchstens 4 mal soviel für den ADAC aus wie ein Mitglied. Die summierten Gesamtkosten für alle Pannen seien mit K bezeichnet. Für $K \to \infty$ konvergiert der kompetitive Faktor gegen $c = \frac{K}{1000+K/4} \to 4$.

c) Die summierten Gesamtkosten für alle Pannen seien mit K und die summierten Kosten vor Beitritt zum ADAC mit $K_1 \leq 500$ bezeichnet. Die eigene Strategie liefert

$$ALG = \begin{cases} K_1 + 1000 + (K - K_1)/4 & K > 500, \\ K & \text{sonst} \end{cases}$$

in Abhängigkeit davon, ob man jemals über 500 DM Kosten für Pannen hat oder nicht. Die optimale Strategie ist durch

$$OPT = \begin{cases} 1000 + K/4 & K \geq 1333\frac{1}{3}, \\ K & \text{sonst} \end{cases}$$

gegeben. Entweder kauft man sich sofort eine Mitgliedschaft oder nie. Die Grenzkosten ergeben sich durch Lösen von $1000 + K/4 \stackrel{!}{=} K$. Der kompetitive Fakor ist der maximale Quotient von ALG und OPT. Allgemein gilt

$$\frac{ALG(K)}{OPT(K)} = \begin{cases} \frac{K}{K} & K \leq 500, \\ \frac{1375+K/4}{1000+K/4} & K \geq 1333\frac{1}{3}, \\ \frac{1375+K/4}{K} & \text{sonst} \end{cases}$$

(mit $K_1 = 500$ gesetzt, da wir nur am maximalen Wert interessiert sind). Das Supremum dieses Quotienten ist 3 für $K \to 500, K > 500$. Damit ist der kompetitive Faktor dieser Strategie

$$c = \sup_K \frac{ALG(K)}{OPT(K)} = 3.$$

Aufgabe 2 (*Analyse: Online-gaming-Algorithmen*)

Angestellte in einem Rechenzentrum haben einen recht eintönigen Job. Ihnen stehen zwar die größten Rechner zur Verfügung, Sie dürfen diese aber nicht selbst verwenden. Stattdessen heißt es nur, die Maschinen möglichst gut auszulasten und Rechnungen zu schreiben. Ein ziemlich langweiliger Job möchte man meinen – zum Glück gibt es noch Computerspiele.
Ein Mitarbeiter hat zu Weihnachten ein neues Computerspiel erhalten, dass er liebsten andauernd spielen würde. Diesem Wunsch steht leider seine Arbeit im Wege. Eine neue Dienstanweisung besagt, dass die teuren Großrechner zu mindestens 50% ausgelastet sein müssen. Da das Scheduling in diesem Rechenzentren noch von Hand durchgeführt wird, muss der spielfreudige Mitarbeiter die laufenden Jobs überwachen und schnell neue Jobs auf leerlaufende Maschinen verteilen. So bleibt leider nur wenig Zeit zum Spielen am Arbeitsplatz.
Jeder Job hat eine Mindestlaufzeit von 5 Minuten. Die Zeit zum Verteilen der Jobs kann für die Bestimmung der Auslastung vernachlässigt werden. Der Mitarbeiter kann in dieser Zeit aber nicht spielen. Ein Job hat während seiner Ausführung die Maschine exklusiv. Es gibt keine automatischen Benachrichtigungen über das Ende eines Jobs. Der Mitarbeiter muss dies selbst periodisch überprüfen.

a) Entwerfen Sie einen *online scheduling* Algorithmus, der die freie Zeit des Mitarbeiters unter Einhaltung der Nebenbedingungen maximiert, wenn er einen Großrechner zu betreuen hat.

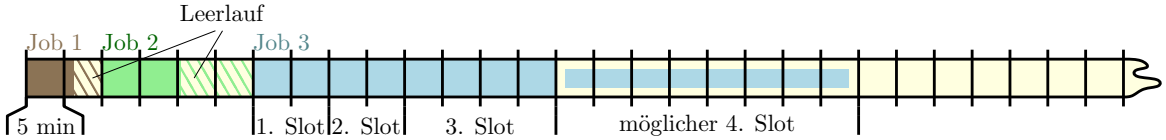
b) Zeigen Sie, dass Ihr Algorithmus optimal bzgl. der Freizeit des Mitarbeiters ist.

Musterlösung:

a) Wir definieren den Algorithmus induktiv. Durch die Mindestlaufzeit von 5 Minuten kann man jedem Job zunächst ein Zeitslot von 10 Minuten zuweisen. Nach Ablauf dieses Zeitslots muss der Mitarbeiter nachschauen, ob der Job bereits abgeschlossen ist, in der Zwischenzeit kann er spielen. Falls der Job in dieser Zeit beendet wurde, stand die Maschine höchstens 50% der Zeit still bis der Mitarbeiter dies erkannt und einen neuen Job gestartet hat. Andernfalls muss ein neuer Zeitslot für den noch laufenden Job zugewiesen werden.

Die Länge des neuen Zeitslots wird gleich der bisherigen Gesamtlaufzeit des Jobs (10 Minuten) gewählt. Dies verdoppelt die mögliche Gesamtlaufzeit des Jobs bis zur nächsten Überprüfung durch den Mitarbeiter auf 20 Minuten. Dadurch wird sichergestellt, dass der Rechner maximal die Hälfte der Zeit leerläuft. Wenn der Job ε Zeiteinheiten nach Start des neuen Zeitslots endet, ist er insgesamt $10+\varepsilon$ Minuten gelaufen und der Rechner damit zu $\frac{10+\varepsilon}{20} > 50\%$ ausgelastet gewesen. Sollte der neue Zeitslot auch nicht genügen, wird dieses Vorgehen wiederholt, bis der Job endet (neuer Zeitslot von $20, 40, \ldots$ Minuten, maximale Gesamtlaufzeit von $40, 80, \ldots$ Minuten).

Folgende Abbildung verdeutlicht den Ablauf des Algorithmus:

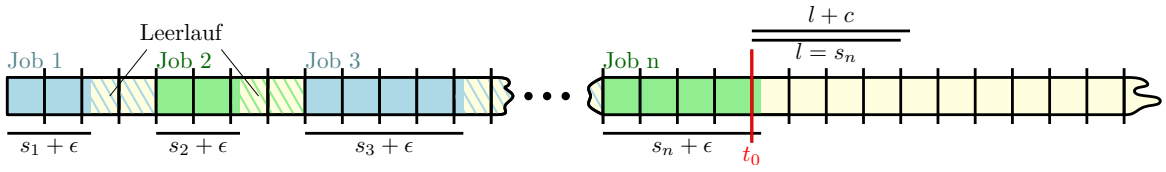


Kein Job erhält mehr als das Doppelte seiner Laufzeit an Zeitslots zugeteilt. Somit ist der Großrechner im Durchschnitt zu mindestens 50% ausgelastet – wie gefordert.

Auch wenn die Aufgabenstellung einen eher scherzhaften Hintergund hat, so ist die vorgestellte Technik der Verdopplung (*doubling*) ein nützliches Hilfsmittel beim Entwurf von *online* Algorithmen. Insbesondere Algorithmen die Suchschritte benötigen, können durch derartige Techniken oft schon recht gute Approximationsgarantien (bzw. kompetitive Faktoren) liefern.

Musterlösung:

b) Angenommen, es existiere ein Algorithmus, der längere Zeitslots zwischen zwei Überprüfungen erlaubt als unser Algorithmus. Dann existiert für eine Folge an Jobs ein Zeitpunkt t_0, an dem die zugeteilten Zeitslots beider Algorithmen erstmals voneinander abweichen. Unser Algorithmus weise einen Zeitslot der Länge l zu, der potentiell bessere Algorithmus einen Zeitslot der Länge $l' = l + c$. Betrachte konkret eine Folge an Jobs, für die unser Algorithmus genau die minimale Auslastungsgrenze von 50% einhält (siehe Grafik).



Job i benötigt Zeit $s_i + \epsilon$ mit s_i gleich einer Dauer, nach der eine Überprüfung stattfindet – also nach $10, 20, 40, 80, \ldots$ Minuten. Zum Zeitpunkt t_0 weist unser Algorithmus einen Zeitslot von $l = s_n$ als Verlängerung zu, der andere Algorithmus einen Zeitslot von $l' = l + c = s_n + c$. Wähle $\epsilon < \frac{c}{2n}$, so ergibt sich nach Abarbeitung dieses Zeitslots eine Auslastung von

$$\frac{\sum_{i=1}^{n} s_i + \epsilon}{c + \sum_{i=1}^{n} 2s_i} = \frac{n \cdot \epsilon + \sum_{i=1}^{n} s_i}{c + 2 \cdot \sum_{i=1}^{n} s_i} < \frac{\frac{c}{2} + \sum_{i=1}^{n} s_i}{2 \cdot \left(\frac{c}{2} + \cdot \sum_{i=1}^{n} s_i\right)} = \frac{1}{2}$$

für den alternativen Algorithmus. Beachte: Startet zu dem Zeitpunkt t_0 ein neuer Job, so ist das Ergebnis analog zu erreichen durch eine Wahl von $s_n = 10$, der doppelten Mindestlaufzeit des Algorithmus. Damit hält er nicht – wie gefordert – eine minimale Auslastung von 50% ein. Unser Algorithmus verwendet also bereits die maximal möglichen Zeitslots zwischen zwei Überprüfungen und ermöglicht dem Mitarbeiter die meiste Freizeit.