**RWTH Aachen** **WS 2012/13**
**Lehrgebiet Theoretische Informatik** Übungsblatt 9
Reidl–Ries–Rossmanith–Sanchez–Tönnis 10.12.2012

# Übung zur Vorlesung Berechenbarkeit und Komplexität

## Aufgabe T20

Beweisen Sie die folgende Aussage: Die Entscheidungsvariante von Bin Packing ist genau dann in polynomieller Zeit lösbar, wenn dies auch für die Optimierungsvariante gilt.

## Lösungsvorschlag.

Wir nehmen also an, dass die Entscheidungvariante des BPP in $\mathsf{P}$ ist und konstruieren einen Polynomialzeit-Algorithmus $\mathcal{A}_{\text{OPT}}$, der eine optimale Lösung des BPP berechnet, und dabei den Polynomialzeit-Algorithmus für die Entscheidungsvariante $\mathcal{A}_{\text{ENT}}$ als Unterprogramm benutzt.

Wir nutzen aus, dass man zwei Objekte $x$ und $y$ zusammenkleben kann, indem man beide löscht und durch ein neues Objekt $x'$ mit Gewicht $w(x') = w(x) + w(y)$ ersetzt. Damit wird erzwungen, dass $x$ und $y$ im gleichen Behälter landen. Falls sich dadurch der Wert der Lösung nicht ändert, so gibt es eine Verteilung, die bei gleichbleibender Qualität $x$ und $y$ dem gleichen Behälter zuordnet. Beachte, dass die Lösung durch das Zusammenkleben nicht besser werden kann.

Im Folgenden nehmen wir an, dass der Wert $b_{opt}$ der optimalen Lösung durch ein Orakel gegeben ist. Wir können dieses Orakel später durch eine binärer Suche ersetzen und damit $b_{opt}$ in Polynomialzeit finden. Der Algorithmus $\mathcal{A}_{\text{OPT}}$ zur Berechnung der optimalen Verteilung geht nun wie folgt vor:

- $\mathcal{A}_{\text{OPT}}$ iteriert über alle Paare $(x, y)$ von Objekten.
- Dabei klebt $\mathcal{A}_{\text{OPT}}$ das Paar $(x, y)$ zusammen und prüft mit Hilfe von $\mathcal{A}_{\text{ENT}}$, ob es eine Verteilung der Objekte auf $b_{opt}$ Behälter gibt.
  - Falls ja, bleiben $x$ und $y$ zusammengeklebt und der Algorithmus wird rekursiv mit dem zusammengeklebten Objekt fortgesetzt bis die Anzahl der Objekte $b_{opt}$ ist.
  - Falls nein, dürfen $x$ und $y$ nicht zusammengeklebt werden und es wird das nächste Paar gewählt.

Die auf diese Art erhaltene Verteilung ordnet jedem Behälter ein zusammengeklebtes Objekt zu. Zerlegt man diese Objekte wieder in seine Bestandteile, d.h. die einzelnen Objekte, aus denen es zusammengeklebt worden ist, erhält man die genaue Verteilung.

In jedem Rekursionsschritt wird die Eingabe um ein Objekt verringert; also werden, wenn $n$ die Anzahl der ursprünglichen Objekte ist, $n - b_{opt}$ Rekursionen durchgeführt. Dabei benötigt jeder Rekursionsschritt maximal $n \cdot (n-1)$ viele Aufrufe von $\mathcal{A}$. Folglich ist die Laufzeit von $\mathcal{A}_{\text{OPT}}$ polynomiell in der Eingabelänge beschränkt.

**Aufgabe T21**

Wir betrachten folgendes Problem: Als Eingabe sind ganzzahlige kartesische Koordinaten von $n$ Punkten in der Ebene sowie weitere Zahlen $k$ und $A$ gegeben. Die Frage ist, ob es $k$ Kreise in der Ebene gibt, welche zusammen alle $n$ Punkte umfassen und deren Flächensumme höchstens $A$ ist.

Beschreiben Sie, wie eine nichtdeterministische Turingmaschine dieses Problem in polynomieller Zeit lösen kann?

**Aufgabe T22**

Wir betrachten das Problem SORTING BY REVERSALS. Eine gegebene Permutation $\pi$ soll durch höchstens $k$ viele Vertauschungen (*reversals*) $\rho$ in eine zweite gegebene Permutation $\sigma$ verwandelt werden. Ein *reversal* $\rho(i,j)$ vertauscht dabei die Reihenfolge aller Elemente im Intervall $[i,j]$. Formal

$$\rho(i,j)\colon (\pi_1 \dots \pi_{i-1}\pi_i\pi_{i+1} \dots \pi_{j-1}\pi_j\pi_{j+1} \dots \pi_n) \mapsto (\pi_1 \dots \pi_{i-1}\pi_j\pi_{j-1} \dots \pi_{i+1}\pi_i\pi_{j+1} \dots \pi_n)$$

Geben Sie ein Zertifikat und einen Polynomialzeitverifizierer an. Beschreiben Sie dazu im Detail die Kodierung und die Länge des Zertifikates und die Arbeitsweise des Verifizierers.

Die Motivation dieses Problems kommt aus der Bioinformatik: Viele Mutationen auf DNA-Strängen finden ausschließlich durch solche *reversals* statt. Eine kürzeste Folge dieser reversals entspricht wahrscheinlich den Mutationen, die wirklich stattfanden.

**Lösungsvorschlag.**

*Zertifikat und Länge:* Das Zertifikat sind die Indizies der reversals. Diese werden binärkodiert und durch entsprechende Trennsymbole separiert. Da höchstens $k$ reversals erlaubt sind, ergibt sich eine Länge von $2k \log n$ + Trennzeichen. *Polynomialzeitverifizierer:* Der Zertifizierer muss testen, ob das Zertifikat die richtige Form hat und ob die Hintereinanderausführung der reversals auf $\pi$ zu $\sigma$ führen.

**(1)** Hat das Zertifikat die Korrekte Form?

**(2)** Führe die Vertauschungen hintereinander auf $\pi$ aus,

**(3)** Vergleiche $\pi$ und $\sigma$.

*Laufzeit:* In Schritt 1 wird die gesamte Eingabe überprüft, was in Zeit $\mathcal{O}(n)$ erledigt wird. Eine Vertauschung von zwei Elementen realisieren wir wie folgt: Das erste Element wird auf eine extra Spur geschrieben, und danach dieses Element in $\pi$ überschrieben. Nun kann das zwischengespeicherte Element an die Position des zweiten Elementes geschrieben werden, dies dauert Zeit $O(\log n)$. Auch müssen wir immer wieder den Zähler welche Elemente nun vertauscht werden müssen aktualisieren ($O(\log n)$). Für jedes reversal müssen wir schimmstenfalls $n/2$ viele Vertauschungen durchführen, Also dauert ein reversal $O(n \log n)$. Dies gibt eine Laufzeit von $O(k \cdot n \cdot \log n)$ für Schritt 2. Im letzten Schritt müssen wir noch zwei Strings der Länge $O(n \log n)$ vergleichen. Der Verifizierer läuft folglich in Polynomialzeit.

**Aufgabe H22 (10 Punkte)**
Beweisen Sie die folgende Aussage: Die Entscheidungsvariante des Travelling Salesman Problems (TSP) ist genau dann in polynomieller Zeit lösbar, wenn dies auch für die Optimierungsvariante gilt. Die Gewichte mögen durch rationale Zahlen in Binärkodierung kodiert sein.

**Lösungsvorschlag.**
Wir nehmen also an, dass die Entscheidungvariante des TSP in $\mathsf{P}$ ist und konstruieren einen Polynomialzeit-Algorithmus $\mathcal{A}_{\mathrm{OPT}}$, der eine optimale Lösung des TSP berechnet, und dabei den Polynomialzeit-Algorithmus für die Entscheidungsvariante $\mathcal{A}_{\mathrm{ENT}}$ als Unterprogramm benutzt.
Als Eingabe erhält der Algorithmus $\mathcal{A}_{\mathrm{OPT}}$ einen gewichteten vollständigen Graphen $G$ mit $n$ Knoten. Wir bezeichnen mit $d_{\max}$ das maximale Kantengewicht in $G$. Jede Rundreise in $G$ hat höchstens Kosten von $nd_{\max}$. Folglich können wir eine Kante aus $G$ „löschen“, indem wir ihre Kosten auf $nd_{\max} + 1$ setzen.
Im Folgenden nehmen wir an, dass der Wert $b_{opt}$ der optimalen Lösung durch ein Orakel gegeben ist. Wir können dieses Orakel später durch eine binärer Suche ersetzen und damit $b_{opt}$ in Polynomialzeit finden. Der Algorithmus $\mathcal{A}_{\mathrm{OPT}}$ zur Berechnung der optimalen Rundreise geht nun wie folgt vor:

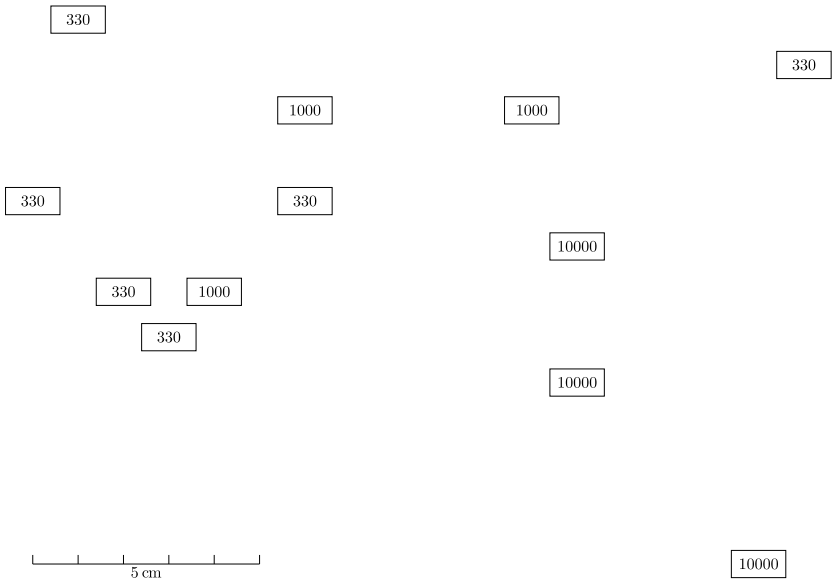
- $\mathcal{A}_{\mathrm{OPT}}$ iteriert über alle Kanten $e$ des Graphen $G$.
- Dabei „löscht“ $\mathcal{A}_{\mathrm{OPT}}$ die Kante $e$ und prüft mit Hilfe von $\mathcal{A}_{\mathrm{ENT}}$, ob es eine Rundreise mit Kosten $b_{opt}$ gibt.
  - Falls ja, dann gibt es eine optimale Rundreise, die nicht über $e$ führt. $\mathcal{A}_{\mathrm{OPT}}$ „löscht“ die Kante $e$ aus dem Graphen und wählt eine andere Kante bis nur noch $n$ Kanten vorhanden sind.
  - Falls nein, dann führt jede optimale Rundreise über die Kante $e$ und die Kante $e$ wird nicht gelöscht. $\mathcal{A}_{\mathrm{OPT}}$ wählt nun eine andere Kante.

Nachdem über alle Kanten iteriert wurde, sind nur noch die Kanten einer optimalen Rundreise im Graphen vorhanden. $\mathcal{A}_{\mathrm{OPT}}$ gibt diese als optimale Lösung aus.
Jede Kante wird höchstens einmal aus $G$ entfernt. Dabei wird jeweils der Algorithmus $\mathcal{A}_{\mathrm{ENT}}$ einmal aufgerufen. Folglich ist die Laufzeit von $\mathcal{A}_{\mathrm{OPT}}$ polynomiell in der Eingabelänge beschränkt.

**Aufgabe H23 (15 Punkte)**
Eine selbstgebaute Bestückungsmaschine kann Widerstände mit verschiedenen Widerstandswerten an die richtigen Plätze in eine Platine stecken. Sie benötigt 2 Sekunden, um auf einen verschieden Wert zu wechseln, ansonsten kann sich der Bestückungskopf mit einem Zentimeter pro Sekunde in eine beliebige Richtung bewegen. Das eigentliche Einstecken eines Widerstands benötigt ebenfalls etwas Zeit, aber diese Zeit ist fest und hängt insbesondere nicht vom Widerstandswert ab oder davon, was die Maschine in der Vergangenheit machte.
Die folgende Tabelle zeigt die Positionen (in Zentimetern) und Widerstandswerte (in Ohm) von elf Widerständen, mit welchen eine Platine bestückt werden muß. Sie können

davon ausgehen, daß der Bestückungskopf sich anfangs bereits auf einer der zu bestückenden Positionen befindet und mit dem richtigen Widerstandswert geladen ist. Der erste Widerstand kann also sofort eingesteckt werden. Die Zeichnung neben der Tabelle zeigt, wie die Widerstände am Ende auf der Platine positioniert sein werden (aber nicht im Maßstab 1 : 1).

| $x$ | $y$ | *Wert* |
|---|---|---|
| 2 | 8 | 330 |
| 3 | 12 | 330 |
| 4 | 6 | 330 |
| 5 | 5 | 330 |
| 6 | 6 | 1000 |
| 8 | 10 | 1000 |
| 8 | 8 | 330 |
| 13 | 10 | 1000 |
| 14 | 4 | 10000 |
| 14 | 7 | 10000 |
| 18 | 0 | 10000 |
| 19 | 11 | 330 |




a) Finden Sie eine optimale Reihenfolge, in welcher alle Widerstände bestückt werden können und der Bestückungskopf anschließend wieder in die Ausgangslage zurückkehrt und wieder mit dem gleichem Widerstandswert wie am Anfang geladen wird. (So kann eine Serie von Platinen nacheinander auf die gleiche Weise bestückt werden.)

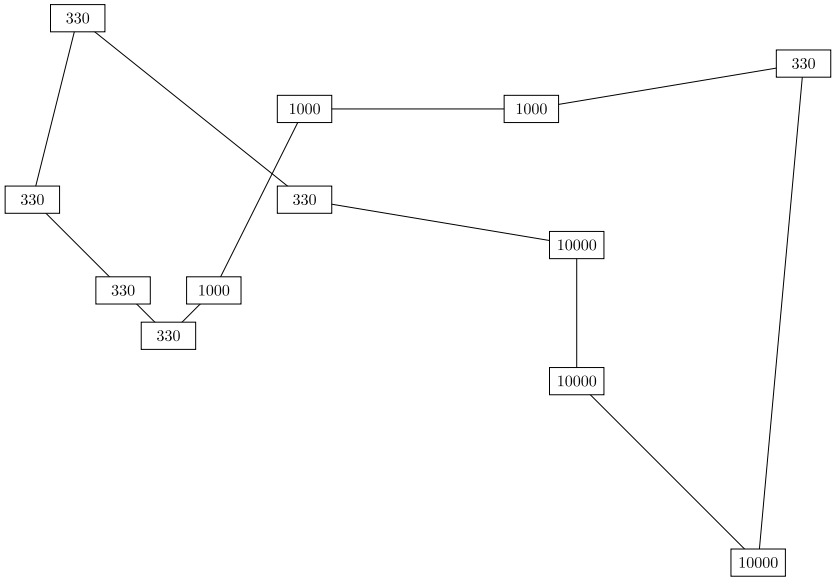
b) Wie können Sie garantieren, daß Ihre Lösung optimal ist?

c) Schätzen Sie, mit bis zu wievielen Widerständen man Ihre Lösungsmethode noch verwenden könnte, bevor die Laufzeit unpraktikabel groß würde.

d) Haben Sie eine Idee, was Sie machen würden, wenn es um hunderte von Widerständen geht?

*Hinweis:* Natürlich dürfen Sie ein Programm schreiben, welches bei der Lösung dieser Aufgabe hilft.

**Lösungsvorschlag.**


330
330
1000
1000
330
330
10000
330
1000
330
10000
10000