Technische Universität München
Zentrum Mathematik
Prof. Dr. P. Gritzmann, Dipl.-Inf. Dipl.-Math. S. Borgwardt, Dipl.-Math. M. Ritter

Optimierung 2, WS 2008/09

Übungsblatt 8

Aufgabe 8.1
Für diese Aufgabe arbeiten Sie bitte in Gruppen zu je zwei Personen zusammen.

a) Ihr Nachbar denkt sich eine natürliche Zahl zwischen 0 und 1 000. Sie sollen diese Zahl bestimmen, indem Sie die Frage „Ist die gedachte Zahl kleiner oder gleich k?“ für beliebige Werte von k stellen, ihr Nachbar (der die Rolle des Orakels übernimmt) muss wahrheitsgemäß antworten. Verwenden Sie binäre Suche, um die gedachte Zahl zu bestimmen.

b) Sei $G = (V, E)$ ein Digraph, $s, t \in V$ zwei verschiedene Knoten und $\phi : E \to \mathbb{N}_0$ eine Kantengewichtung. Sie haben folgendes Orakel zu Verfügung:

Orakel (Weglängen-Orakel)
Input: *Eine Kantengewichtung $\varphi : E \to \mathbb{N}_0$, eine natürliche Zahl $K \in \mathbb{N}_0$.*

Output: *„Ja“, falls es einen s-t-Weg in G gibt, der bezüglich φ höchstens Länge K besitzt, „Nein“ sonst.*

Entwerfen Sie einen möglichst effizienten Algorithmus, der das Orakel verwendet, um die Länge eines (bezüglich ϕ) kürzesten s-t-Weges in G zu bestimmen. Welche Laufzeit hat ihr Algorithmus höchstens?

c) Entwerfen Sie einen Algorithmus, der das Orakel aus b) und ggf. Ihren Algorithmus aus b) verwendet, um einen kürzesten s-t-Weg in G zu bestimmen (den Weg selbst, nicht nur dessen Länge). *Hinweis: Verändern Sie die Kantengewichtung ϕ passend, bevor Sie das Orakel aufrufen, d. h. $\phi \neq \varphi$ ist erlaubt.*

Lösung zu Aufgabe 8.1

a) *Zwei TUM-Studierende, Anna und Bernd, sitzen sich gegenüber, ein „Optimierung 2“-Übungsblatt liegt vor ihnen auf dem Tisch. Sie blicken sich etwas ratlos an.*

Anna Also, ich denk mir eine Zahl aus. (*kurze Pause, schaut nachdenklich*) Ok, ich habs. Fang an!
Bernd (*etwas ärgerlich*) Wieso denkst Du Dir die Zahl aus? Ich weiss doch auch nicht mehr, wie das mit der binären Suche ging.
Anna (*mit besonders süßem Lächeln*) Das kriegst Du schon hin.
Bernd (*freundlicher*) Also gut. Ist die Zahl kleiner oder gleich 500?

ANNA Ja.
BERND Also zwischen 1 und 500. Ist sie kleiner oder gleich 250?
ANNA Ja.
BERND Und auch kleiner oder gleich 125?
ANNA Ja.
BERND Jetzt geht's nicht mehr auf – was hat der Gritzmann denn in der Vorlesung gesagt, muss man da auf- oder abrunden?
ANNA (*unsicher*) Ja?

Die Tutorin kommt zufällig im richtigen Moment vorbei, um die Frage mitzubekommen. Bernd wirft ihr einen hilfesuchenden Blick zu.

LUCIA (*bestimmt*) Abrunden! Aber eigentlich ist das egal.
BERND Danke! (*wieder zu Anna gewandt*) Also, ist die Zahl kleiner oder gleich (*grübelt kurz*) 62?
ANNA Ja.
BERND Ist die Zahl kleiner oder gleich 31?
ANNA Nein.
BERND Dann liegt sie also zwischen 31 und 62. Ist die Zahl kleiner oder gleich 46?
ANNA Ja.
BERND Hm, 31 plus 46 ist 77, und die Hälfte davon wäre 38½. Ist die Zahl kleiner oder gleich 38?
ANNA Nein.
BERND Hm, 31 plus 46 ist 77, und die Hälfte davon wäre 38½. Ist die Zahl kleiner oder gleich 38?
ANNA Nein.
BERND Also 38 plus 46 durch 2, das ist 42. Ist die Zahl kleiner oder gleich 42?
ANNA Ja.
BERND Ist sie kleiner oder gleich 40?
ANNA Nein.
BERND Ist sie kleiner oder gleich 41?
ANNA Nein.
BERND Dann bleibt nur noch eine Möglichkeit, Du hast Dir 42 gedacht.
ANNA Stimmt. Diese binäre Suche ist doch eigentlich ganz einfach.
BERND Ja. Aber die zweite Aufgabe mache ich jetzt nicht mehr alleine.

Allgemeine Strategie: Man merkt sich immer eine obere und eine untere Grenze, mit jeder Frage (jedem Aufruf des Orakels) fragt man nach dem arithmetischen Mittel dieser Grenzen, ggf. abgerundet. Je nach Antwort passt man entweder die obere oder die untere Grenze an, bis der Bereich auf eine mögliche Antwort eingeschränkt ist. Für ursprüngliche obere Schranke M und untere Schranke m ist die Laufzeit der binären Suche $\mathcal{O}(\log_2(M-m))$.

b) Der Algorithmus verwendet binäre Suche, wie bereits oben beschrieben. Als Startschranken kann man 0 (da alle Kantengewichte nichtnegativ sind) bzw. $\sum_{e\in E}\phi(e)$ verwenden.

c) In b) haben wir eine Routine entwickelt, die für jede Kantengewichtung $\phi : E \to \mathbb{N}_0$ die Länge eines kürzesten s-t-Weges bezüglich ϕ ermittelt. Sei $E = \{e_1, \ldots, e_p\}$. Wir bestimmen damit folgendermaßen einen kürzesten Weg:

1) Bestimme die Länge L eines kürzesten Weges bzgl. $\phi_0 := \phi$.

2) Für $i = 1, \ldots, p$:

$$\phi_i'(e) := \begin{cases} \phi_{i-1}(e) + 1, & \text{falls } e = e_i, \\ \phi_{i-1}(e), & \text{sonst.} \end{cases}$$

Bestimme die Länge eines kürzesten Weges bzgl. ϕ_i'. Ist diese Länge gleich L, so wird e_i für den kürzesten Weg nicht benötigt und wir setzen $\phi_i := \phi_i'$. Sonst ist e_i in der gesuchten Lösung enthalten, in diesem Fall setzen wir $\phi_i := \phi_{i-1}$.

3) Der kürzeste Weg besteht genau aus den Kanten, deren Gewicht am Ende unverändert ist, für die also $\phi_p(e) = \phi(e)$ gilt.

Der Algorithmus kommt mit $|E| = p$ Aufrufen des Algorithmus aus Teilaufgabe b) aus und benötigt daher polynomiell viele Aufrufe des Weglängen-Orakels.

Aufgabe 8.2
Betrachten Sie die folgenden beiden Probleme:

Problem (SUBSET SUM)
Gegeben: *Eine Menge $S = \{s_1, ..., s_n\}$ mit $s_i \in \mathbb{Z}$ $\forall i \in \{1, ..., n\}$, $\beta \in \mathbb{Z}$.*

Auftrag: *Entscheide, ob eine Teilmenge $S_1 \subset S$ existiert, so dass*

$$\sum_{x \in S_1} x = \beta.$$

Problem (PARTITION)
Gegeben: *Eine Menge $S = \{s_1, ..., s_n\}$ mit $s_i \in \mathbb{Z}$ $\forall i \in \{1, ..., n\}$.*

Auftrag: *Entscheide, ob es zwei Teilmengen $S_1, S_2 \subset S$ mit $S_1 \cap S_2 = \emptyset$ und $S_1 \cup S_2 = S$ gibt, so dass*

$$\sum_{x \in S_1} x = \sum_{x \in S_2} x.$$

Beweisen oder widerlegen Sie:

a) PARTITION ist polynomiell reduzierbar auf SUBSET SUM.

b) SUBSET SUM ist polynomiell reduzierbar auf PARTITION.

Lösung zu Aufgabe 8.2
Beide Probleme sind NP-vollständig, die Reduktionsaussagen sind korrekt. Wir beweisen sie konstruktiv:

a) Wir wollen zunächst Partition mit einem Orakel für Subset Sum lösen. Für eine entsprechende Partition gilt $\sum_{x \in S_1} x = \sum_{x \in S_2} x = \frac{1}{2} \sum_{x \in S} x$. Mit $\beta = \frac{1}{2} \sum_{x \in S} x$ rufen wir also das Orakel für Subset Sum auf und übernehmen die zurückgelieferte Ja-Nein-Antwort für Partition.

Natürlich ist die Reduktion polynomiell, da wir (für β) nur einmal über alle Elemente von S aufsummieren müssen.

b) Wir wollen nun Subset Sum mit einem Orakel für Partition lösen. Dazu definieren wir die Menge $S' := S \cup \{s\}$, wobei $s = (\sum_{x \in S} x) - 2\beta$. Damit gilt $\sum_{x \in S'} x = 2(\sum_{x \in S} x) - 2\beta$. Wenn wir für S' das Orakel für Partition aufrufen, so beantwortet dieses die Frage, ob es zwei Teilmengen S_1, S_2 von S' gibt mit $\sum_{x \in S_1} x = \sum_{x \in S_2} x = \frac{1}{2} \sum_{x \in S'} x = (\sum_{x \in S} x) - \beta$. Für eine solche Partition muss o.B.d.A. $s \in S_1$ gelten. Mit der Definition von S' und s gilt dann $\sum_{x \in S_1, x \neq s} x = \beta$. Wir können also die Antwort des Partition-Orakels direkt für unser Subset Sum Problem übernehmen.

Wieder ist die Reduktion polynomiell, da wir nur einmal (für s) über alle Elemente von S aufsummieren müssen (und dann 2β subtrahieren).

Aufgabe 8.3 *Hausaufgabe*
Betrachten Sie das folgende Problem:

Problem (SORTIEREN)
Gegeben: *$n \in \mathbb{N}$, $\gamma_1, ..., \gamma_n \in \mathbb{Z}$ mit $\gamma_i \neq \gamma_j$ für alle $i \neq j$.*

Auftrag: *Bestimme eine Permutation $\pi \in S_n$ mit $\gamma_{\pi(1)} \leq ... \leq \gamma_{\pi(n)}$.*

Ein *vergleichsbasierter* Sortier-Algorithmus löst das Problem allein durch (wiederholtes) paarweises Vergleichen zweier Elemente aus $\gamma_1, ..., \gamma_n$.

a) Entwerfen Sie einen vergleichsbasierten Sortier-Algorithmus, der in maximal $\mathcal{O}(n \ln n)$ Schritten das Problem SORTIEREN löst. Begründen Sie die Korrektheit Ihres Algorithmus und beweisen Sie, dass die Laufzeit nicht schlechter als $\mathcal{O}(n \ln n)$ ist.

b) Zeigen Sie, dass jeder vergleichsbasierte Sortier-Algorithmus eine Laufzeit von mindestens $\Omega(n \ln n)$ besitzt.

Hinweis: Nach der Stirling-Formel gilt $\ln(n!) \approx n \ln n - n + 1$.

Lösung zu Aufgabe 8.3
Man beachte, dass Vergleiche nur auf den im Input gegebenen Zahlen durchgeführt werden, so dass sich die Kodierungslängen der auftretenden Zahlen nicht vergrößern. Daher genügt es bei unserer Laufzeitanalyse nur die Anzahl der Vergleiche zu zählen.

a) Um auf die angegebene Laufzeit zu bekommen, müssen wir einen Divide and Conquer-Ansatz benutzen, man kann ja nicht jedes Element mit jedem vergleichen ($n \cdot (n-1)$ Vergleiche). Z.B. Merge Sort hat eine worst-case Laufzeit von $O(n \ln n)$. Das Konzept von Merge Sort ist einfach:

- Hat die zu sortierende Liste kein oder ein Element, ist sie bereits sortiert.
- Ansonsten wird die Liste in zwei (möglichst) gleich große Teillisten aufgeteilt und jede Liste rekursiv durch Merge Sort sortiert.

  Diese zwei Teillisten werden, nachdem sie intern sortiert wurden, wieder zu einer (jetzt korrekt sortierten) großen Liste zusammengefasst (merge).

Damit ist nach Konstruktion klar, dass der Algorithmus korrekt arbeitet. Zur Laufzeit:

Das Mergen zweier Listen der Länge k_1 und k_2 lässt sich in höchstens $k_1 + k_2$ Vergleichen durchführen: Wir beginnen, indem wir die ersten beiden Elemente der Listen vergleichen. Das kleinere davon löschen wir aus der entsprechenden Liste und fügen es hinten an die (noch leere) zusammengesetzte Liste an, es ist das insgesamt kleinste Element, da beide Teillisten intern schon sortiert sind. Dann vergleichen wir wieder die kleinsten (noch übrigen) Elemente. In jedem Schritt wird durch einen Vergleich ein Element aus den Listen gelöscht. Daher sind wir nach spätestens $k_1 + k_2$ Vergleichen mit einem solchen Merge-Vorgang fertig.

Jedes einzelne Element der zu sortierenden Listen wird insgesamt höchstens $O(\ln n)$ mal gemergt, da wir unsere zu sortierende Liste rekursiv jeweils in möglichst gleich große Teile aufgeteilt hatten. Damit ist die Gesamtlänge aller zu mergenden Listen während des Algorithmus in $O(n \ln n)$.

b) Unser vergleichsbasierter Sortier-Algorithmus muss $n!$ Permutationen unterscheiden können. Durch jeden Vergleich zweier Elemente s_i, s_j können wir entscheiden, ob $\pi(i) < \pi(j)$ oder nicht, d.h. jeder Vergleich halbiert die Anzahl der noch möglichen Permutationen. Da wir am Ende auf diese Art und Weise die $n!$ Permutationen auf eine eindeutige reduziert haben müssen, brauchen wir mindestens $\log_2(n!) = const \cdot \ln(n!)$ Vergleiche. Es gilt $\ln(n!) = \sum_{j=1}^{n} \ln j \approx \int_{1}^{n} \ln x dx = n \ln n - n + 1 \in \Omega(n \ln n)$ (nach der Stirling Formel).