2. DEZEMBER 2013

DOKUMENTATION

ZU AUFGABENBLATT 07 AUS DER VORLESUNGSREIHE „ALGORITHMEN UND DATENSTRUKTUREN“

HAW HAMBURG

Erstellt von Steffen Giersch, Birger Kamp und Maria Lüdemann

DOKUMENTATION

ZU AUFGABENBLATT 07 AUS DER VORLESUNGSREIHE „ALGORITHMEN UND DATENSTRUKTUREN“

ÜBUNGSAUFGABE 7.1

Sie arbeiten als Programmierer bei GG Investment. Die Firma testet verschiedene Investmentstrategien auf den Börsendaten der Vergangenheit.
Dabei sie mit einer Zeitreihe von n Kurswerten k1,...,kn. Ein Investment in diesem Zeitintervall ist ein Kauf zu einem Zeitpunkt i gefolgt von einem verkauf zum Zeitpunkt j (also mit i < j). Ein Investment ist sinnvoll, wenn der Verkaufskurs höher als der Einkaufskurs liegt. Das sinnvolle Investment ist optimal, wenn es unter allen sinnvollen Investments in diesem Zeitintervall maximalen Gewinn bietet.

TEILAUFGABE 1

Konzipieren Sie einen Algorithmus (d.h. Sie müssen ihn nicht ausprogrammieren), der die Zeitpunkte i und j für ein optimales Investment f(k1,...,kn) = (i,j) findet bzw. meldet, wenn kein Investment sinnvoll wäre.
Der Zeitbedarf Ihres Algorithmus soll in $O(n^2)$ liegen.

```
akku = -1
iopt = -1
jopt = -1
for i from 1 to n
	for j from i to n
		if(InvW(j) - InvW(i) > akku)
			iopt = i
			jopt = j
			akku = InvW(j) - InvW(i)
		endif
	endfor
endfor
if akku < 0
	throw error
endif
```

TEILAUFGABE 2

Implementieren Sie Ihren $O(n^2)$ Algorithmus.

```
public static List<Integer> investment(List<Integer> investments){
	int accu = -1;
	int iopt = 0;
	int jopt = 0;
	int diff = 0;
	List<Integer> result = new ArrayList<Integer>();
	for(int i = 0 ; i < investments.size()-1 ; i++){
		for(int j = i ; j < investments.size()-1 ; j++){
			diff = investments.get(j) - investments.get(i);
			if( diff > accu){
				iopt = i;
				jopt = j;
				accu = diff;
```

```
                    }
                }
            }
            result.addAll(Arrays.asList(iopt,jopt,accu));

            return result;
}
```

TEILAUFGABE 3

Entwerfen Sie nun einen Algorithmus nach dem Teile-und-Herrsche-Prinzip, der nur O(n log(n)) Zeit benötigt.

(Entwurf für Listen der Länge >= 4)
1. Teile die Ursprungsliste in Listen der Länge 4
2. Speichere diese in der ursprünglichen Reihenfolge in lists
3. Ersetze jede Liste in lists durch eine Liste der Form
   [min,max,bestmin,bestmax,best]

wobei min -> das Minimum der Liste
max -> das Maximum der Liste
bestMin -> Minimum des besten Intervalls der Liste
bestMax -> Maximum des besten Intervalls der Liste
best -> Größe des besten Intervalls der Liste

4. Rufe lists rekursiv auf und zerteile sie bei jedem Aufruf in zwei weitere Teillisten, bis die Teilliste genau zwei Listenelemente hat
5. Füge diese Zwei Listen zusammen, sodass sie wieder der obigen Form entsprechen. Tue dies solange, bis nur noch ein Listenelement übrig ist.
6. Wenn lists[4] kleiner als 0 ist, dann gib einen Fehler aus
7. Gib ein Array zurück mit den Elementen [indexOf(lists[2]),indexOf(lists[3]),list[4]]

TEILAUFGABE 4

Erläutern Sie die Funktionsweise Ihres Algorithmus im Detail.

Die Idee ist, die Eingabe in Listen aus vier Elementen zu teilen und über diese vier Elemente Aussagen zu treffen. Diese Aussagen sind die maximale Größe, die minimale Größe und Informationen über den größt möglichen Intervall innerhalb jeder diese Teillisten.
Nun können wir zwei benachbarte Teillisten ohne Informationsverlust zusammenführen, indem wir
a) das Maximum beider Listen bestimmen und als das neue Maximum annehmen
b) das Minimum beider Listen bestimmen und als das neue Minimum annehmen
c) den größtmöglichen Intervall ersetzen durch den größten von
   - der linken Liste
   - der rechten Liste
   - dem entstehenden Intervall des Minimums der linken und Maximums der rechten Liste

Dadurch haben wir linear viele Vergleiche der Elemente untereinander um die Eigenschaften der jeweiligen 4-Elementigen Listen zu bestimmen mal logarithmisch viele Schritte um diese Teillisten zusammenzufügen

TEILAUFGABE 5

Erläutern Sie, warum die Funktionsweise korrekt ist:
a) Wenn es ein Optimum gibt, dann wird auch eines gefunden.
b) Wenn es kein Optimum gibt, dann wird ein Fehler gemeldet.
c) Und das Paar (i,j), das gefunden wird, ist auch ein Optimum.
d) Wenn ein Fehler gemeldet wird, dann gibt es kein Optimum.

a) Wenn es genau ein Optimum gibt, dann wird an irgendeiner Stelle entweder bei 3.3 oder 3.4 ein positiver Wert als Optimum abgespeichert. Hierdurch wird bei 3.6 kein Fehler geworfen und daraufhin bei 3.7 ein Ergebnis ausgegeben.

b) Wenn es kein Optimum gibt dann werden beim Zusammenfügen der Listen nur negative beste Intervalle gefunden. Dadurch wird dann bei Punkt 3.6 ein Fehler geworfen.

c) - wenn das Optimum innerhalb einer der bei 3.3 erzeugten Teillisten liegt, dann wird innerhalb dieser Teilliste in der [4] die Größe dieses Optimums und in [2] und [3] die Grenzen gespeichert. Beim Zusammenfügen der Teillisten wird nun immer dieses Optimum übernommen, weil es größer ist als alle anderen (zwangsweise) negativen besten Optima. Der Intervall wird in 3.7 zurückgegeben.
- wenn das Optimum in zwei verschiedenen Arrays liegt, dann wird bei 3.5 dieses Optimum irgendwann durch das Zusammenfügen gefunden, weil das Minimum der linken und das Maximum der rechten Liste dann dem besten Intervall entsprechen. Dieser wird in 3.7 zurückgegeben.

d) Die einzige Möglichkeit einen Fehler zu melden hat der Algorithmus, wenn nach dem bestimmen der Daten für die Intervalle in Punkt 3.3 bei jedem bestimmen eine negative Zahl gefunden wird und beim Zusammenfügen dieser Listen bei 3.4 das Minimum der linken Liste immer größer ist als das Maximum der rechten Liste. Dies kann ausschließlich der Fall sein, wenn die Liste monoton fallend ist, weil ansonsten entweder bei 3.3 oder 3.4 ein nicht negativer Wert für das beste Intervall gespeichert werden würde.

TEILAUFGABE 6

Weise Sie (z.B. unter Verwendung des Master-Theorems) nach, dass die Laufzeit T(n) tatsächlich in $\theta(n\log(n))$ *liegt.*

Elemente Aufteilen und Werte bestimmen = O(n/4)
Weil wir genau ein mal über die Liste iterieren und bei jedem Iterationsschritt nur Konstante Operationen ausführen.

Der rekursive Aufruf:
n beschreibt die Länge der Eingabeliste

$$T(n) = \begin{cases} 25, wenn\ n = 2 \\ 2 * T\left(\frac{n}{2}\right) + \ n, sonst \end{cases}$$

c = 25, weil dies ungefähr der Anzahl der Rechenschritt entspricht die für das Zusammenfügen der Listen benötigt wird.
a = 2, weil zwei Aufrufe gemacht werden müssen um die Teillösungen zu bestimmen
b = 2, weil die Größe der Eingabe in zwei Teillösungen geteilt wird

f(n) = n, weil genau ein mal über die Eingabe iteriert werden muss, um die Teileingaben zu erzeugen
Damit ist die Form auf den zweiten Fall des Mastertheorems zurückzuführen, weil

$$n \in \theta(n^{log_{2(2)}})$$

Somit ist die Komplexität unseres Algorithmus

Bzw.

$$O(n * \log(n))$$

TEILAUFGABE 7

Implementieren Sie Ihren Algorithmus. Testen Sie Ihn, indem Sie vergleichen, ob er die gleichen Resultate wie Ihr O(n^2) Algorithmus liefert.

```
public static List<Integer> investm(List<Integer> investments) {
	List<Integer> akku = new ArrayList<Integer>();
	List<List<Integer>> lists = new ArrayList<List<Integer>>();
	List<List<Integer>> lists2 = new ArrayList<List<Integer>>();

	if(investments.size() == 1) {
		throw new IllegalArgumentException("Das Array hat nur ein Element");
	} else if(investments.size() == 2){
		if(investments.get(0) <= investments.get(1)) {
			akku.addAll(Arrays.asList(0,1,investments.get(1) - investments.get(0)));
			return akku;
		} else {
			throw new IllegalArgumentException("Es gibt keinen gewinnbringenden
Zeitraum");
		}
	}
	//Aufteilen der Liste in 4er Arrays
	for(int i = 0; i < investments.size(); i += 4) {
		lists.add(Arrays.asList(investments.get(i), investments.get(i+1),
investments.get(i+2), investments.get(i+3)));
	}
	// Speichere in jedem 4er Tupel in der Form [min,max,bestmin,bestmax,best]
	for(List<Integer> list : lists) {
		int min = Integer.MAX_VALUE;
		int max = Integer.MIN_VALUE;
		for(int i = 0; i < 4; i++) {
			if(list.get(i) > max) {
				max = list.get(i);
			} else if(list.get(i) < min) {
				min = list.get(i);
			}
		}
		List<Integer> best = investment(list);
lists2.add(Arrays.asList(min,max,list.get(best.get(0)),list.get(best.get(1)),best.get(2)));
	}
	akku = investHelper(lists2);

	akku.set(2, investments.indexOf(akku.get(2)));
	for(int i = akku.get(2); i < investments.size();i++) {
		if(akku.get(3) == investments.get(i)) {
			akku.set(3, i);
```

```
                    break;
                }
        }
        return Arrays.asList(akku.get(2), akku.get(3), akku.get(4));
}

private static List<Integer> investHelper(List<List<Integer>> lists) {
        if(lists.size() == 1) {
                return lists.get(0);
        } else if (lists.size() == 2) {
                int num1 = lists.get(0).get(4);
                int num2 = lists.get(1).get(4);
                int num3 = lists.get(1).get(1) - lists.get(0).get(0);
                int max = lists.get(0).get(1) > lists.get(1).get(1) ? lists.get(0).get(1) :
lists.get(1).get(1);
                int min = lists.get(0).get(0) < lists.get(1).get(0) ? lists.get(0).get(0) :
lists.get(1).get(0);
                if(num1 > num2 && num1 > num3) {
                        lists.get(0).set(0, min);
                        lists.get(0).set(1, max);
                        return lists.get(0);
                } else if (num2 > num1 && num2 > num3) {
                        lists.get(1).set(0, min);
                        lists.get(1).set(1, max);
                        return lists.get(1);
                } else {
                        return
Arrays.asList(lists.get(0).get(0),lists.get(1).get(1),lists.get(0).get(0),lists.get(1).get(
1),lists.get(1).get(1) - lists.get(0).get(0));
                        }
        } else {
                List<List<Integer>> list1 = new ArrayList<List<Integer>>();
                List<List<Integer>> list2 = new ArrayList<List<Integer>>();
                for(int i = 0; i < lists.size(); i++) {
                        if(i <= lists.size()/2) {
                                list1.add(lists.get(i));
                        } else {
                                list2.add(lists.get(i));
                        }
                }
                        return
investHelper(Arrays.asList(investHelper(list1),investHelper(list1)));
        }
}
```