	Algorithmen - Verteilen		AnPr	V 1.0
	Name	Klasse	Datum	

1 Allgemeines zu „Verteilen“

Bei Verteilalgorithmen müssen in der Regel Werte aus einem Array (oder einer anderen Form von Listen) in ein anderes Array zugeordnet werden. Hierbei gilt es in der Aufgabenstellung folgende Details zu beachten:

- Muss die Verteilung in irgendeiner Form optimiert sein?
- Ist die Quellliste unendlich, oder begrenzt?
- Ist die Quellliste veränderbar, bzw. können die Elemente der Liste geflaggt werden?
- Sind die Listen dynamisch, oder statisch?

2 Verteilung ohne Optimierung

Gehen wir bei der Analyse der Algorithmen von folgender Situation aus. Wir haben ein Array mit Daten – sagen wir das Gewicht in kg von Paketen:

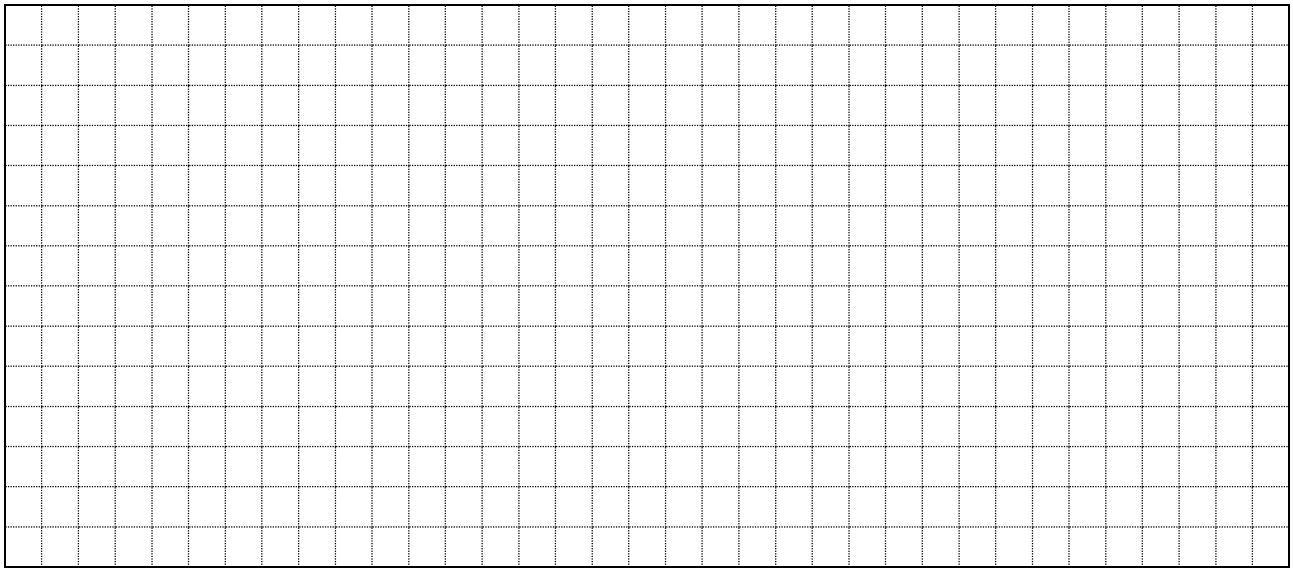
values:	459	573	877	914	722	582	67	169	381	262	774	194	210	196	473	408	838	87	627	977

Das Ganze soll auf LKWs verteilt werden, die einen Maximalwert von 2000 kg laden können. Gehen wir davon aus, dass wir 10 LKWs haben, somit verteilen wir die Zahlen in Array von 10 Elementen. Folgendes Struktogramm ermöglicht eine einfache, nicht optimierte Verteilung.

Folgende Erkenntnisse sind hier wichtig:

- Da wir keine Optimierung haben, können wir einfach alle Werte in `values` sequenziell abarbeiten, wir brauchen also eine Schleife durch `values`.
- Dadurch, dass wir `values` sequenziell abarbeiten, müssen wir das Array `values` nicht verändern um bspw. zu markieren, dass ein Wert bereits berücksichtigt wurde.
- Da die Zielliste eine feste Größe hat, müssen wir den Fall des „overflows“ berücksichtigen – also den Fall, dass wir zu viele Pakete für die vorhandenen LKWs haben.

Zeichnen Sie das entsprechende Struktogramm und realisieren Sie anschließend den Code.



Arraycode: `int[] values = {459, 573, 877, 914, 722, 582, 67, 169, 381, 262, 774, 194, 210, 196, 473, 408, 838, 87, 627, 977};`

Nach der Abarbeitung dieses Algorithmus, müssten wir im Array `lkw` folgende Werte finden:

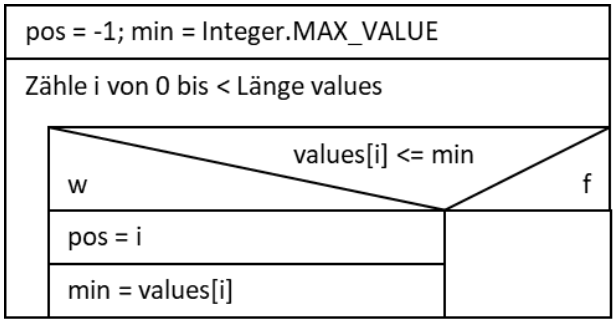
lkw:	1909	1636	1461	1847	1960	977	0	0	0	0

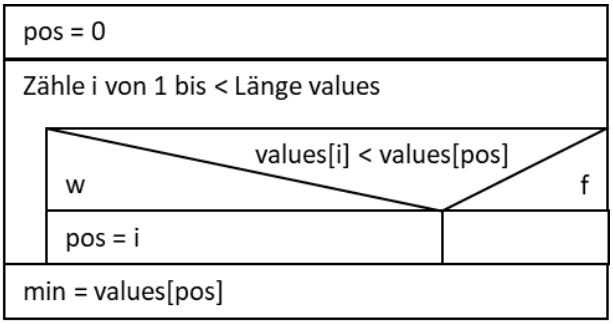
Wir benötigen also sechs LKWs für den Transport der Waren.

3 Suchen nach Minimum/Maximum

Wenn wir uns die Daten genauer ansehen erkennen wir, dass wir eigentlich mit 4 LKWs auskommen würden. Hierfür müssen wir nun aber die Datenverarbeitung einer Optimierung unterziehen. Diese wird zwar nicht jeden Fall abdecken, aber durchaus für die allermeisten Situationen ein vertretbares Ergebnis liefern. Hierfür müssen wir aber immer das Paket suchen, mit dem wir am nächsten an die Maximalbeladung von 2000 kg herankommen. Für die Suche nach dem Minimum (bzw. Maximum) – auch als „lineare Suche“ bekannt gibt es wiederum einen Standardalgorithmus.

Hierbei vergleichen wir jeden Wert der Liste mit einem Referenzwert. Wenn bei der Minimumsuche der Referenzwert größer ist als der aktuelle Listenwert, so wird dieser zum neuen Referenzwert. Von zentraler Bedeutung ist an dieser Stelle damit die Initialisierung (also erstmalige Belegung) des Referenzwertes. Sehen wir uns hierfür zwei Lösungen für das Problem an:





In der linken Variante initialisieren wir die Referenzvariable „`min`“ mit dem größtmöglichen Wert des genutzten Datentyps (in unserem Fall Integer). Nun suchen wir im Array die Position, in der ein kleinerer Wert existiert, was dann der neue Referenzwert ist. Sollten wir jedoch ein Array mit ausschließlich `Integer.MAX_VALUE` haben und in der Variable `pos` einen echten Wert (also ungleich -1) benötigen, so müssen wir beim Vergleich auch das Istgleich mit einbeziehen, da wir sonst nie in die Zeile pos = i gelangen würden.

Im rechten Fall initialisieren wir die Positionsvariable `pos` gleich auf die erste Position des Arrays in der Annahme, dass dies der kleinste Wert ist. In der Schleife prüfen wir dann, ob nicht eine andere Position im Array einen kleineren Wert führt. Hier ist es dann entsprechend sinnvoll, die Schleife bei 1 und nicht bei 0 beginnen zu lassen. Wichtig ist aber zu verstehen, dass wir hier immer mindestens einen Wert im Array benötigen. Arrays mit der Größe 0 würden zu einem Fehler führen, da der Zugriff auf values[pos] in der Verzweigung unmöglich wäre.

4 Optimierte Verteilung

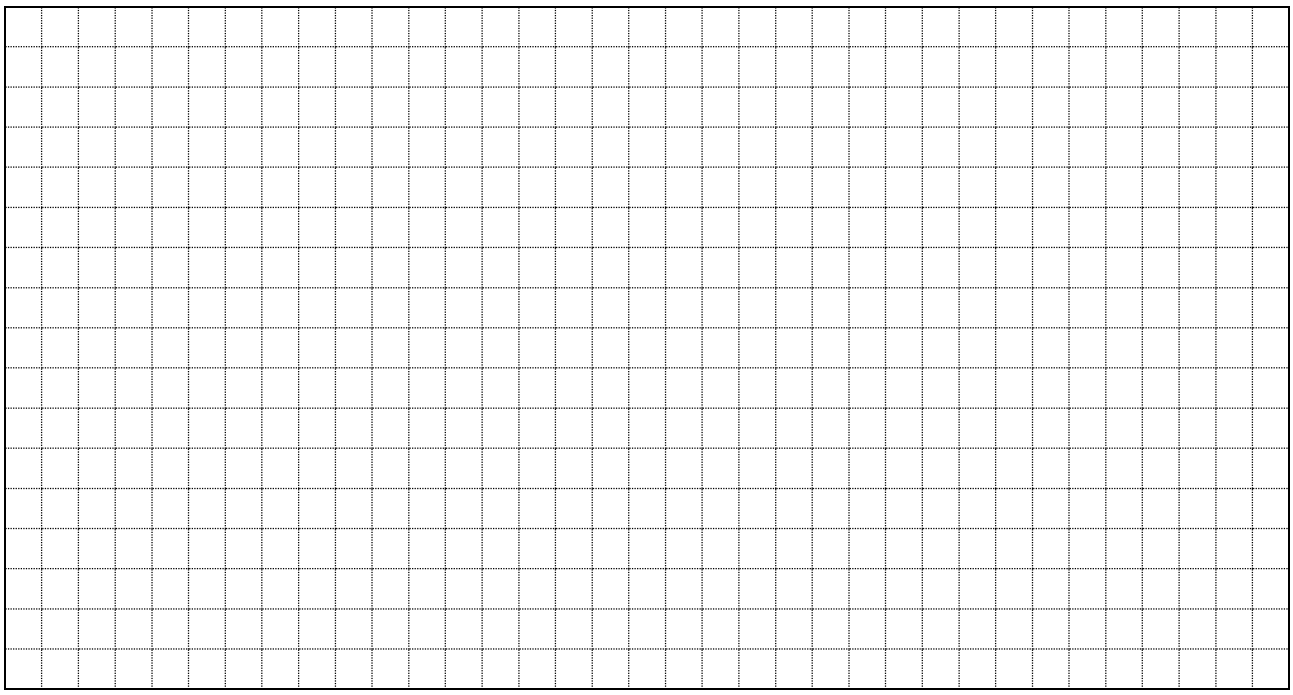
Wenn wir die Füllmenge der LKWs optimieren wollen, müssen wir im Vergleich zu Kapitel 2 die Schleifen umdrehen – wir gehen also nicht `values` sequenziell durch und verteilen alles, sondern wir gehen die LKWs durch und versuchen jeden einzelnen nacheinander optimal zu befüllen. Dies bedeutet aber, dass wir für einen LKW im gesamten `values` Array den nächsten optimalen Wert suchen müssen und demnach dort auch hinterlegen müssen, ob ein Paket bereits verladen wurde, oder nicht. Dies geht, indem die Pakete als Objekte realisiert werden und diese mit einem Flag versehen werden, oder mittels einem dynamischen Array, wo verladene Pakete gelöscht werden können.

Gehen wir für unserem Algorithmus von einem dynamischen Array für `values` aus, welches Stück für Stück verkürzt wird. Ziel ist es, jedes Paket optimiert in einen LKW zu laden. Wir arbeiten also solange ab, bis das `values` Array die Länge 0 hat. Ein Eintrag wird dann gelöscht, wenn wir das optimale Paket gefunden haben.

Ein Paket ist wiederum dann optimal, wenn die Beladung auf dem aktuellen LKW möglichst nah an das Optimum kommt, also die noch freie Kapazität nach der Beladung mit einem neuen Paket möglichst klein ist. Weiterhin darf der LKW aber nicht überladen wird, also die freie Kapazität darf nicht negativ sein. Diese Bedingungen können wir mathematisch formulieren:

Mögliche Beladung „`maxVal`“ ist `MAX – lkw[posLKW]`
Freie Kapazität nach der Beladung „`delta`“ ist `maxVal – values[i]`

Delta muss als Minimum optimiert werden. Zeichnen Sie hierfür nun die Struktogramme für die Optimierung und der eigentlichen Verteilung, welche die Optimierung aufruft:



Das Ergebnis im lkw-Array sieht dann wie folgt aus – wir brauchen also nur noch fünf LKWs:

lkw:	1978	1977	1969	1992	1874	0	0	0	0	0

Nun gibt es auch sortierte Eingangslisten. Bei unserem optimierten Algorithmus wird dies keinerlei Rolle spielen, da der Algorithmus immer den gleichen „optimierten“ Wert aus dem `values` Array holt. Auch bei der nicht optimieren Variante aus Kapitel 2 wird die Sortierung zu keiner wirklichen Verbesserung führen.

5 Unendliche Eingangslisten

Eine weitere Möglichkeit für Verteilalgorithmen sind unendliche Eingangslisten – wenn wir bspw. aus einem Lager stetig Pakete erhalten, die auf die LKW-Flotte verteilt werden müssen und wir bspw. zu ermitteln haben, wie viele Pakete pro Tag (wenn also pro Tag jeder LKW einmal fährt) verteilt werden können. Solche „unendliche“ Listen zeichnen sich dadurch aus, dass wir kein Array mit Index haben, sondern ein Objekt mit einer Methode „`getNext()`“.

Bei solch einem Algorithmus können wir nun nicht mehr das optimale Paket für einen LKW suchen, sondern den optimalen LKW für das aktuelle Paket. Wir nehmen also ein Paket vom endlosen Laufband und laufen solange durch die 10 LKWs, bis wir einen LKW gefunden haben, wo wir das Paket aufladen können. Erst wenn das aktuelle Paket keinen Platz mehr hat, ist die Verteilung beendet.