DAP2 UB6
Gr. , Briefkasten

Springenberg, 177792

May 27, 2017

0.1 Gierige Algorithmen

0.1.1

Gegeben ist:
Distanz in Kilometern: l
Anzahl der maximalen Kilometer pro Tag: k
Index der Raststaette: $i, 1 \leq i \leq n$
Array, der jedem i eine Anzahl von kilometern zuordnet: A
$A[n] = l$
$0 < A[i+1] - A[i] \leq k$
Damit ist A aufsteigend sortiert.

Dem Algorithmus wird A uebergeben und es soll ein Array B zurueck gegeben werden , sodass $|B|$ minimal ist.
Gesucht wird also immer eine Raststaette die A[i_n] moeglichst weit weg ist, wobei von ihr aus wieder eine Rasstaette in der Distanz k erreichbar sein muss.
Wir wissen, dass gilt:
$0 < A[i+1] - A[i] \leq k$
Damit koennen wir nach dem Index i_x, bei dem der Abstand von A[i_0] zu A[i_0] moeglichst nahe kleiner gleich k ist, zuchen.
Diesen erhalten wir, an der stelle i mit:
A[i] $\leq$ k and A[i+1] ¿ k
da A aufsteigend sortiert ist. Anshliessend erhoehen wir k um A[i], da die Distanz immer zum Startpunkt gemessen wurde.

```
Minimize(A)
    k ←VALUE
    tmpK ←k
    counter ←2
    for i ←1 to length[A] - 1 do
        if A[i] ≤ k and A[i+1] ¿ k then
            counter ←counter + 1
            tmpK ←k + A[i]
```

```
B ←B[1 ... counter]
tmpK ←k
for i ←1 to length[A] - 1 do
      if A[i] ≤ k and A[i+1] ¿ k then
            B[i] ←A[i]
            tmpK ←k + A[i]
B[counter] ←A[length[A]]
return B
```

0.2 Gierige Algorithmen

0.2.1

Gegeben ist:
Simon kann sich nur eine Muenze pro Monat kaufen.
Array mit Muenzen:
A[1 ... n]
Array mit Faktoren fuer die Muenzen:
P[1 ... n]
Startpreis aller Muenzen:
20 (Euro)
Preissteigung je Monat:
A[i] ←A[i] * P[i]

Ferner gilt fuer den Preis p der Muenze am Index i nach m Monaten:
$p = 20 * P[i]^{m-1}$

Gesucht wird nun in Abhaengigkeit von P der minimale Gesamtbetrag, den Simon ausgeben muss, wenn er sich pro Monat eine Muenze kauft.
Dazu muss zunachst ein Array B mit den Indice von A ueber ihren Wert in P aufsteigend sortiert angelegt werden.
Anschliessend wird A durchgelaufen und von der Gleichung:
$p = 20 * P[i]^{m-1}$
gemacht und fuer jeden wert in A nach der durch B vorgegebenen Reihenfolge der Gesamtbetrag aufaddiert.

```
MinMoney(P)
    new Array B [n ... length[P]]
    B = mergeSort(B)
    endBetrag ←0
    monthCounter ←1
    for i ←1 to length(B) do
        endBetrag ←endBetrag + (20 * P[i]^(monthCounter−1))
        monthCounter ←monthCounter + 1
    return endBetrag
```