

1. Zwei Supermarktketten (A und B) haben sich geeinigt in einem noch unberührten Land (d.h. ohne Supermärkte) ihre Läden folgendermaßen zu errichten:

- In jedem Ort soll es genau einen Supermarkt geben.
- Sind zwei Orte benachbart, so sollen die Supermärkte je zu einer Firma gehören.

Sie sollen nun einen Algorithmus (Pseudocode + verständliche Erklärung) entwerfen, welcher überprüft, ob eine solche Aufteilung möglich ist. Zu jedem Ort x gibt es eine Liste $x \rightarrow n[1 \dots k]$ der k Nachbarorte (der Einfachheit halber nehmen wir an daß jeder Ort genau k Nachbarorte besitzt). Sie können beliebige Zusatzinformationen in Zusammenhang mit einem Ort speichern. Zum Beispiel könnte $x \rightarrow m$ angeben zu welcher Supermarktkette (A oder B) der Supermarkt im Ort x schließlich gehört. Weiters dürfen Sie die drei Funktionen `push(S, x)`, `pop(S)` und `is_empty(S)` für einen beliebigen Stack S verwenden. Analysieren Sie weiters die Laufzeit Ihrer Lösung.

Idee:

gehe alle Orte durch (1...n);

setze all Nachbarn auf die Andere Marke (1...k)

gehe zum naechsten Ort

setze nie einen Nachbarn mit kleinerem Index auf einen andere Marke

breche ab wenn ein Ort bereits einen Nachbarn der selben Marke hat

C Code:

```
village(1) = A;
for (count_village=1, count_village<n; count_village++)
{
  if(village(count_village) == A)
    for(count_neighbour; count_neighbour < k; count_neighbour++)
    {
      if(neighbour(count_Village) != A)
        neighbour(count_Village) = B;
      else
        return -1;
    }
  else
    for(count_neighbour; count_neighbour < k; count_neighbour++)
    {
      if(neighbour(count_Village) != B)
        neighbour(count_Village) = A;
      else
        return -1;
    }
}
return 0;
```

Psydo Code:

village(1) = A;

PROVEVILLAGE(village,neighbor)

for count_village = 1 to n step 1 do

if village(count_village) == A then

SETTYPE(B,neighbor(count_village));

if village(count_village) == B then

SETTYPE(A,neighbor(count_village));

SETTYPE(newType,neighbor)

for count_neighbor = 1 to k step 1 do

if neighbor(count_neighbor) != newType

neighbor(count_neighbor) = newType;

else

$\rightarrow \Theta(1)$

$\rightarrow \Theta(1)$
 $\rightarrow \Theta(n)$

$\Theta(1) + \Theta(1)$

$\rightarrow \Theta(1)$

$\rightarrow \Theta(k)$ bzw. $\Theta(n)$

$\Theta(1) + \Theta(1)$

$$T(n) = \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n)$$

$$= 7 \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n^2)$$

$$= \mathcal{O}(n^2) \quad \neq k=n \dots \text{muss case jeder Ort Nachbar anwenden } \mathcal{O}(n \cdot k)$$

Das Beispiel ist lösbar in Anlehnung an die 4 Farben Theorie für Graphen;

Lässt sich ein Kreis finden dessen Anzahl an Kanten umgekehrt ist, ist es nicht lösbar.

Knoten = Ort

Kante = Beziehung

2. Es werden online n Zahlen geliefert, d.h. die Zahlen können nur genau einmal der Reihe nach eingelesen werden. Entwerfen Sie einen Algorithmus, der daraus die k kleinsten Elemente bestimmt. Dabei gilt, daß n viel zu groß ist um alle n Zahlen zu speichern, k aber wesentlich kleiner als n ist. Ihr Algorithmus muß mit $O(k)$ Speicher auskommen und soll maximal $O(n \log k)$ Zeit benötigen. Analysieren Sie die Laufzeit und den Speicherbedarf Ihrer Lösung. (Tipp: eine bestimmte in der Vorlesung besprochene Datenstruktur kann hilfreich sein)

mit den ersten h Elementen eine Halde bauen bei der das erste Element das grösste ist; $A(n) \approx [A(2n), A(2n+1)]$

daneben das nächste ankommende Element mit $A(1)$ vergleichen; ist es kleiner ($x < A(1)$) vertauschen und anschließend verhalten

```

K_LITTLE_ELEMENT(x, A)
  for count = 1 to k do
    A(count) = getNextElement(); } h · O(1)

BAUE_HALDE(A);
  for count = k+1 to n do
    if x = getNextElement() < A(1)
      vertausche(A(1), x); → O(h)
      Verhalde(A); → (n-h) O(1)
                    → (n-h) O(log(h))

```

$$\begin{aligned}
 T(n) &= h \cdot O(1) + O(h) + (n-h) \cdot O(1) + (n-h) \cdot O(\log(h)) \\
 &= \underbrace{O(h)}_{\text{count}} + \underbrace{O(h)}_{\text{count}} + \underbrace{O(n-h)}_{O(n)} + \underbrace{O((n-h) \log(h))}_{O(n \log(h))} \\
 &= O(n) + O(n \log(h)) \\
 &= O(n \log(h))
 \end{aligned}$$

3. Sie haben einen randomisierten Algorithmus entworfen. Für eine Eingabe der Länge n hat Ihr Algorithmus eine Laufzeit von kni für eine Konstante $k > 1$ mit Wahrscheinlichkeit

$$\frac{2(n-i+1)}{n(n+1)}$$

für $i = 1, \dots, n$. Führen Sie eine average case analysis der Laufzeit durch.

$$T(n) = k \cdot i \cdot n$$

$$T(n) = p_{n1} \cdot T(1) + p_{n2} \cdot T(2) + \dots + p_{nn} \cdot T(n)$$

$$\sum_{i=0}^{n-1} \underbrace{\frac{2(n-i+1)}{n(n+1)}}_{p_{ni}} [k \cdot i \cdot n]$$

$$= \frac{2k}{(n+1)} \sum_{i=1}^{n-1} ni - i^2 + i$$

$$= \frac{2k}{(n+1)} \sum_{i=0}^n (-i^2 + ni + i) + 0$$

$$= \frac{2k}{(n+1)} \left(\sum_{i=0}^n -i^2 + n \cdot \sum_{i=0}^n i + \sum_{i=0}^n i \right)$$

$$= \frac{2k}{(n+1)} \cdot \left(-\frac{n(n+1)(2n+1)}{6} + \frac{n^2(n+1)}{2} + \frac{n(n+1)}{2} \right)$$

$$k \cdot \left(-\frac{2n^2 - n}{3} + n^2 + n \right)$$

$$= k \cdot \left(\frac{-2n^2 - n + 3n^2 + 3n}{3} \right)$$

$$= k \cdot \left(\frac{n^2 + 2n}{3} \right) \Rightarrow \mathcal{O}(n^2)$$

