CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL
Institut für Informatik, Arbeitsgruppe Theorie der Parallelität
Prof. Dr. K. Jansen, K.-M. Klein, M. Rau, F. Land

13. Juli 2016

Übungsaufgaben zur Vorlesung »Algorithmen und Datenstrukturen«

SS 2016

Aufgabe 1 (Plateau-Problem)

Gegeben sei ein nichtleeres Array A von n natürlichen Zahlen. Gesucht ist ein Element, das am häufigsten in A vorkommt.

Geben Sie einen Algorithmus mit Laufzeit $\mathcal{O}(n \log n)$ für dieses Problem an. Beschreiben Sie dazu kurz Ihre Idee und geben Sie den Algorithmus in Pseudocode an. Sie dürfen Algorithmen als Subroutinen verwenden, die aus der Vorlesung bekannt sind.

Geben Sie außerdem einen Algorithmus an, der, falls das Array A nur Zahlen $\leq n$ enthält, die Laufzeit $O(n)$ hat.

Lösung. Idee: Sortiere zunächst das Array. Danach kann man die Häufigkeit jedes Elementes leicht zählen, da alle Vorkommen hintereinander liegen. Als Sortierverfahren wird ein beliebiger Algorithmus mit Worst-Case-Laufzeit $O(n \log n)$ verwendet, z.B. Mergesort.

```
1   sort(A);
2
3   int plateau_value = A[0];
4   int plateau_length = 1;
5
6   int current_value = A[0];
7   int current_length = 1;
8   int i = 1;
9   while(i < n) {
10      if(A[i] == current_value) {
11          current_length++;
12          if(current_length > plateau_length) {
13              plateau_value = current_value;
14              plateau_length = current_length;
15          }
16      } else {
17          current_value = A[i];
18          current_length = 1;
19      }
20      i++;
21  }
22  return plateau_value;
```

Die Laufzeit wird durch das Sortieren dominiert. Das finden des größten Plateaus benötigt im sortierten Array Zeit $O(n)$.

Gilt für alle Array-Einträge $A[i] \leq n$, so kann das Array mittels Bucketsort in Zeit $O(n+n) = O(n)$ sortiert werden. Der gesamte Algorithmus braucht dann ebenfalls nur noch Zeit $O(n)$. □

Aufgabe 2 (Algorithmenanalyse)

Betrachten Sie untenstehenden Algorithmus für Eingaben $a, b \in \mathbb{N}$ mit $a, b \geq 1$. Was berechnet er? Bestimmen Sie außerdem seine Laufzeit (mit Beweis).

```
1    while(a != b) {
2      if(a > b) {
3        a = a - b;
4      } else {
5        b = b - a;
6      }
7    }
8    return a;
```

Lösung. Der Algorithmus ist eine iterative Variante des Euklidischen Algorithmus und berechnet den größten gemeinsamen Teiler von a und b. Die Laufzeit beträgt $O(a+b)$.

Beweis. Wir zeigen zunächst, dass nach jeder Iteration $a, b \geq 1$ gilt. Fall 1: $a > b$. Dann ist $a - b \geq 1$. Fall 2: $a \not> b$. Dann muss, da $a \neq b$ ist, schon $b > a$ gelten, und es folgt $b - a \geq 1$. Also sind die neuen Werte für a und b mindestens 1.

Da in jeder Iteration $a, b \geq 1$ gilt, wird die Summe $a + b$ mit jeder Iteration um mindestens 1 kleiner. Weil $a + b \geq 2$ gilt, terminiert der Algorithmus nach spätestens $a + b - 2$ Iterationen. Jede Iteration hat konstant viele Operationen, sodass die Laufzeit $O(a+b)$ ist. □

Bemerkung Es ist $O(a+b) = O(\max(a,b))$, und für $b = 1$ benötigt der Algorithmus $a - 1 = \Omega(a) = \Omega(\max(a,b))$ Iterationen, d.h. es kann keine kleinere Schranke für die Laufzeit gezeigt werden. □

Aufgabe 3 (Backtracking)

Sei $G = (V, E)$ ein Graph mit $V = \{1, \ldots, n\}$. Ein *Hamiltonkreis* in G ist ein Kreis der Länge $|V| = n$ in G, d.h. eine Folge $v_1, \ldots, v_n \in V$ von paarweise verschiedenen Knoten mit $\{v_i, v_{i+1}\} \in E$ für alle $i = 1, \ldots, n-1$ und $\{v_n, v_1\} \in E$.

Enwerfen Sie einen auf Backtracking basierenden Algorithmus, der einen Hamiltonkreis in G findet, oder korrekt feststellt, dass keiner existiert. Geben Sie Ihren Algorithmus in Pseudocode an.

Hinweis Sie können davon ausgehen, dass eine Funktion NACHBAR(v, w) gegeben ist, die angibt, ob Knoten v und w benachbart (durch eine Kante verbunden) sind.

Lösung. Idee:

- Die Wahl der Knoten wird über ein Array `solution` der Länge n geregelt.
- Für die aktuelle Position i wird nacheinander für alle Knoten getestet, ob `solution`$[0], \ldots,$ `solution`$[i]$ zu einem Hamiltonkreis fortgesetzt werden kann.
- Wenn für keinen Knoten an Position i ein Hamiltonkreis gefunden wird, wird der nächste Knoten an Position $i-1$ probiert (Backtracking).
- Ohne Einschränkung können wir davon ausgehen, dass ein Hamiltonkreis bei Knoten 1 beginnt.

Algorithmus FINDEHAMILTONKREIS(n)

```
1  Erzeuge Feld solution mit n Elementen
2  solution[0] = 1
3  for i=1 to n-1 do
```

```
4     solution[i] = 0
5   od
6   boolean success = Complete(solution, 1, n)
7   if success then
8     return solution
9   fi
10  return 'Kein Hamiltonkreis vorhanden'
```

Algorithmus COMPLETE(solution, i, n)

```
1   if i = n then
2     return Nachbar(solution[n-1], 1)
3   fi
4   for k=1 to n do
5     if KnotenGültig(solution, i, k) then
6       solution[i] = k
7       if Complete(solution, i+1, n) then
8         return true
9       fi
10      solution[i] = 0
11    fi
12  od
13  return false
```

Algorithmus KNOTENGÜLTIG(solution, i, k)

```
1   if not Nachbar(solution[i-1], k) then
2     return false
3   fi
4   for j=0 to i-1 do
5     if solution[j] = k then
6       return false
7     fi
8   od
9   return true
```

□

Aufgabe 4 (NP-Vollständigkeit)

Das Problem HALFCLIQUE besteht darin, zu entscheiden, ob ein gegebener Graph $G = (V,E)$ eine Clique der Kardinalität mindestens $|V|/2$ hat.

Zeigen Sie, dass dieses Problem NP-vollständig ist. Sie können ohne Beweis davon ausgehen, dass HALFCLIQUE $\in$ NP ist.

Lösung. Da HALFCLIQUE $\in$ NP ist, müssen wir noch zeigen, dass HALFCLIQUE NP-schwierig ist. Wir geben eine Reduktion von CLIQUE an. Sei (G,k) eine Instanz von CLIQUE, also $G = (V,E)$ ein Graph und $k \in \mathbb{N}$. Wir unterscheiden drei Fälle:

Fall 1 $k = |V|/2$. Dann gebe $G' = G$ als Instanz von HALFCLIQUE aus.

Fall 2 $k > |V|/2$. Dann gebe den Graphen G' aus, der aus G durch hinzufügen von $2k - |V| > 0$ isolierten Knoten entsteht.

Fall 3 $k < |V|/2$. Dann gebe den Graphen G' aus, der aus G durch hinzufügen von $|V| - 2k > 0$ Knoten entsteht, die jeweils mit allen Knoten (untereinander und zu V) verbunden werden.

Wir beweisen die Korrektheit der Reduktion. Sei V' die Knotenmenge von G'. Es ist zu zeigen, dass G eine Clique der Größe k genau dann besitzt, wenn G' eine Clique der Größe $|V'|/2$ besitzt.

Fall 1 $k = |V|/2$. Dann hat offensichtlich G eine Clique der Größe k genau dann, wenn G' eine Clique der Größe $k = |V'|/2$ hat.

Fall 2 $k > |V|/2$. Dann ist $|V'| = |V| + 2k - |V| = 2k$ und G besitzt eine Clique der Größe k genau dann, wenn G' eine Clique der Größe $k = |V'|/2$ besitzt.

Fall 3 $k < |V|/2$. Sei $W = V' \setminus V$ die Menge der hinzugefügten Knoten. Es habe zunächst G eine Clique C der Größe k. Dann ist $C \cup W$ eine eine Clique der Größe $k + |V| - 2k = |V| - k$ in G' und es ist $|V'| = |V| + |V| - 2k = 2(|V| - k)$. Es besitze nun G' eine Clique C der Größe $|V'|/2 = |V| - k$. Dann ist $C \setminus W$ eine Clique in G mit mindestens $(|V| - k) - |W| = (|V| - k) - (|V| - 2k) = k$ Knoten. □