Algorithmen & Datenstrukturen - Aufgaben zum 23. November 2015 (Blatt 04)

Tobias Knöppler (6523815), Nico Tress (6378086)

23.11.2015

4.1

1.

Es existiert genau dann eine Partitionierung von U, die die geforderten Eigenschaften hat, wenn $\forall u \in U(s(u) \leq K)$ gilt. Deshalb genügt der folgende Algorithmus für die Entscheidung, ob es eine Partitionierung gibt, sodass für jede Partition $\sum\limits_{u \in U_i} s(u) \leq K$ erfüllt ist:

Pseudocode:

```
ExistsPartition(U, K):
  for each u in U
    if s(u) > K
      return false
  return true
```

Es lässt sich leicht sehen, dass dieser Algorithmus in NP ist; seine Laufzeit ist sogar in O(—U—), da er nur eine Schleife über alle Element in U enthält.

2.

Für diese Entscheidung genügt es, diejenigen Teilworte der Worte aus S zu betrachten, deren Länge exakt n ist, da alle längeren Teilworte mindestens eines der Teilworte der Länge n selbst als Teilwort enthielten.
Die Anzahl der Teilworte der Länge n eines Wortes $x \in S$ ist nun gegeben durch $|x| - n + 1$. Dies lässt sich leicht einsehen, wenn man überlegt, wie oft man eine "Schablone" der Länge n auf ein Wort x legen und diese verschieben könnte, ohne dass sie darüber hinaus ragt. für eine Schablone der Länge 1 gäbe es dafür exakt $|x|$ Möglichkeiten ($|x| - 1 + 1$); für eine um y längere Schablone entsprechend (y-1) Optionen mehr.
Mit dieser Vorüberlegung lässt sich der folgende Algorithmus formulieren:

Geg. Menge von Worten S und Länge n

```
ExistiertTeilwort(S, n)
  # iteriere über alle Worte in S
  for each x1 in S
    # Für jedes Teilwort w der Länge n von x1...
    for i=0 to (|x1| - n + 1)
      w = x1[i to (i + n-1)
      # ...iteriere über jedes Wort (x2) in S und...
      for each x2 in S
        # ...vergleiche w mit jedem Teilwort von x2.
        found_w = false
        for j=0 to (|x2| - n +1)
          # Falls x2 w enthält setze found_w = true
          if x2[i to (i + n - 1)] = w
            found_w = true
        # Falls ein Wort w nicht enthielt, setze w_in_all_x2 auf false
        if not found_w
          w_in_all_x2 = false
    # Falls eines der Teilworte in allen Worten aus S enthalten war, gebe true zurück.
    if w_in_all_x2
      return true
```

Liese man diesen Algorithmus nicht-deterministisch ausführen, indem jede Iteration der ersten for-Schleife gleichzeitig ausgeführt wird, und true zurückgegeben wird, sobald eine For-Schleife true zurückgibt (bzw. false, wenn alle false zurückgeben), so lässt sich leicht sehen, dass der Algorithmus in NP ist (auch wenn es wesentlich effizientere Algorithmen gäbe).

4.2

1.

$L_1, L_2 \in P \Rightarrow L_1 \in O(n^a) \wedge L_2 \in O(n^b) \Rightarrow L_1 \cdot L_2 \in O(n^a + n^b) \in O(n^{max(a,b)+1}) \Rightarrow L_1 \cdot L_2 \in P$ $\square$

Seien L_1, L_2, L_3, L_4 Sprachen, sodass $L_1, L_2 \in P \wedge L_3, L_4 \in NP$.
Dann gibt es zwei DTMs S,T, und zwei NTMs U,V, sodass S alle Worte von L_1, T alle Worte von L_2, U alle Worte von L_3 und V alle Worte von L_4 akzeptiert.
Für jede der TMs existiert zudem ein Polynom, dass dessen Laufzeit beschränkt.

Nun lässt sich leicht durch aneinanderfügen der Turingmaschinen S und T - indem von jedem Endzustand von S eine Kante zu dem Startzustand von T hinzugefügt wird und die jeweiligen Ein- und Ausgaben auf je einem eigenen Band liegen - eine weitere DTM erstellen, die S und T nacheinander lößt.
Ähnlich lässt sich für U und V - durch Verbinden jedes Endknotens von U mit dem Startknoten von V - eine NTM erzeugen, die $U \cdot V$ lößt.
Damit ist $\forall S, T(S \cdot T \in P)$ und $\forall U, V(U \cdot V \in NP)$ bewiesen. $\square$

2.

Geg. Aussagenlogische Formel F (n sei die Anzahl der Literale

```
getAssignment(F)
  i = 1
  while count(c of size i in F) != 0
    # Falls eine Klausel leer ist (d.h. sie enthält keine nicht zu false evaluierten Liteale),
    # gibt es keine gültige Belegung
    for each clause c in F # Schleifenkomplexität: |F|
      if c is empty
        return false
    # Iteriere über alle Klauseln der Länge i, wobei i der Länge der kleinsten Klauseln,
    # welche noch negierte Literale enthalten können, entspricht.
    for each clause c of length i in F # Schleifenkomplexität: |F|
      # Sobald ein negiertes Literal gefunden wird, wird es mit true (das nicht-negierte
      # also mit false) belegt.
      for each literal l in c # Schleifenkomplexität: |c|
        if l is negated
          for each clause d in F # Schleifenkomplexität: |F|
            # Dadurch kann es überall dort, wo es nicht-negiert vorkommt
            # (entspricht "or false") gelöscht werden;
            if d contains negated l # Komplexität: |d|
              remove d
            # überall, wo es negiert vorkommt
            # (entspricht "or true"), kann die Klausel gelöscht werden
            else if d contains non-negated l # Komplexität: |d|
              remove l from d
          # Danach wird die while-Schleife "neugestartet" (mit i=1)
          i = 1
          continue while;
```

n sei die Anzahl der Literale Die Laufzeit eines while-Schleifen-Durchlaufs ist im Worst Case in $O(n^2 + |F|) \in O(n^2)$, da die zweite For-Schleife (und die verschachtelten Schleifen) alle Literale mit allen Literalen vergleichen (im Worst Case) und die erste For-Schleife einmal über alle Klauseln iteriert.
Die while-Schleife selbst, hat die Komplexität $n + m$, wobei n der Anzahl Literale und m der maximalen Klausellänge entspricht.
Im Worst Case ist also offensichtlich n = m, woraus sich eine Gesamtkomplexität in $O(2n \cdot n^2 + |F|) \in O(n^3)$ ergibt.
Damit ist der Algorithmus in P. $\square$

Unter der Annahme $P = NP$ lässt sich sogar ein wesentlich einfacher Algorithmus finden: Eine NTM kann nicht-deterministisch jede mögliche Belegung durchprobieren und wird in n Schritten terminieren und die richtige Belegung zurückgeben, wenn diese existiert. (*)

4.3

1.

Sat-2 lässt sich folgendermaßen auf SAT reduzieren.
Man finde eine gültige Belegung für eine Formel F in Sat-2.
Nun sei $G = F \wedge \neg(a, b, c...) \wedge (d, e, f...)$, wobei $(a, b, c...)$ den Literalen entsprechen, die bei der ersten Belegung mit true belegt wurden und (d, e, f) den Literalen, welche bei der ersten Belegung mit false belegt wurden.
Bringt man nun G in die KNF und löst die daraus resultierende Formel (mittels SAT), so hat man

(falls möglich) eine zweite gültige Belegung erhalten. Falls dies möglich war, ist F in SAT-2, falls nicht, ist F nicht in SAT-2. Damit ist SAT-2 auf SAT reduziert, woraus $SAT - 2 \in NP - C$ folgt.
$\square$