Theoretische Grundlagen der Informatik II
Blatt 5

Markus Vieth, David Klopp, Christian Stricker

26. November 2015

Aufgabe 1

Beh: INDEPENDENT-SET $\in$ NP:

Bew:

GUESS

Rate nicht-deterministisch eine Menge von Knoten der Größe k.

CHECK

```
vorhandene_kanten = []
FOR v IN V: // iteriert über alle Knoten
    FOR k IN v.getKanten(): // iteriert über alle Kanten eines Knotens
        FOR i IN vorhandene_kanten: // iteriert über alle Kanten des Arrays
            IF i == k:
                return Nein // zwei Knoten die zu einer Kante passen
            ELSE:
                vorhandene_kanten.add(k)
return Ja // jede Kante der Lösung ist nur einmal vertreten
```

Laufzeit:

GUESS:

Im worst-case, also für k=n beträgt die Laufzeit n.

CHECK:

Falls alle Knoten untereinander verbunden sind, beträgt die Laufzeit zum Durchlaufen aller Kanten: $n \cdot \sum_{i=0}^{n-1} i$. Nach Gauß folgt daraus: $n \cdot \frac{(n^2-n)}{2}$.

Gesamt:

Die gesamt Laufzeit beträgt somit: $n \cdot \frac{(n^2-n)}{2} + n \in O(n^3)$
q.e.d.

Reduktion: CLIQUE $\leq_p$ INDEPENDENT-SET

Vorgehen:

Verbinde alle Knoten untereinander, die durch keine Kante verbunden sind. Lösche alle ursprünglichen Kanten und wende nun das INDEPENDENT-SET Problem mit der Größe k an.

Formal:

Eingabe: Graph G(V,E), ganze Zahl k

$E' := \{< v, w > | \forall\ v \neq w\ and\ v, w \in V\} \setminus E$
$G' := G(V, E')$
$INDEPENDENT-SET(G', k)$

Laufzeit:

1. Bilde die Menge alle Kante $\Rightarrow \sum_{i=1}^{n}(n-i) = \sum_{i=0}^{n-1} i = \frac{n^2-n}{2}$
2. Entferne aus der Menge aller möglichen Kanten jene, die in E liegen. Iteriere hierzu über die Menge aller möglichen Kanten, so wie über E $\Rightarrow (\frac{n^2-n}{2})^2$

$\Rightarrow$ Laufzeit: $O(n^4)$

Äquivalenzbeweis:

" $\Rightarrow$ " Geg: CLIQUE
Alle Knoten in einer Clique sind untereinander verbunden. Durch die Komplementbildung der Kanten sind diese Knoten nun alle nicht mehr miteinander verbunden. Es ergibt sich somit ein Independent-Set.

" $\Leftarrow$ " Geg: Independent-Set
Alle Knoten eines Independent-Set weisen untereinander keine Kanten auf. Nach der Bildung des Komplements sind alle Knoten untereinander verbunden. Es ergibt sich eine Clique.

q.e.d

Aufgabe 2

Annahme: $P \neq NP$, da SAT nur mit einem exponentiellen Aufwand gelöst werden kann und SAT in NP liegt.
Widerspruch: Um zu zeigen, dass NP = P gilt, reicht es aus, ein Algorithmus zu finden, der ein NP-hartes Problem in P löst.
Der Gegenbeweis wäre, es gibt keinen einzigen Algorithmus, der ein NP-hartes Problem in P löst.
Daraus folgt, dass es längst nicht ausreicht, nur einen einzigen Algorithmus zu finden, für den gilt: NP = P.
Da wahrscheinlich bisher noch nicht alle Algorithmen gefunden wurden, kann man nicht für jeden Algorithmus zeigen, dass $NP \neq P$ gilt,womit die Annahme hinfällig ist.

Aufgabe 3

Vorüberlegung:

$$P = NP \Rightarrow SAT \in NP = P \Rightarrow SAT \in P$$

zu Zeigen:

Es existiert ein Algorithmus, welcher für eine erfüllbare aussagenlogische Formel Φ in polynomieller Zeit eine erfüllende Belegung ausgibt.

Beweis:

Sei die Laufzeit des polynomiellen SAT-Algorithmus = p
Sei $X := \{x_i | \forall x_i, \overline{x_i} \in \Phi\}$ die Liste aller Literale in Φ, |X| ist im worst-case = n.

```
if (SAT(Φ(X)){ //Teste ob Φ erfüllbar ist; (Laufzeit ist p)
  for(int i=0; i < |X|; i++) { //Setze jedes x_i auf einen festen Wert, (im worst-case n Durchläufe)
     X[i]=0; //Teste ob Φ erfüllbar, wenn x_i = 0
     if(!(SAT(Φ(X)))) { //Wenn nicht: (Laufzeit = p)
        X[i]=1; //, dann muss x_i gleich 1 sein
     } //Wenn SAT mit x_i = 0 erfüllbar ist, bleibt x_i gleich 0
  } //Wenn dies für alle x_i erfolgreich durchgeführt wurde, besitzt X eine erfüllende Belegung
   String result =““;
   for (int i = 0; i < |X| ; i++)
     result = result+X[i]+““;
  return result;
}
else
  return “nicht erfüllbar“;
```

Die erste Überprüfung dauert p lange, die folgende Schleife hat im worst-case n Durchläufe und in jedem Durchlauf einen SAT-Aufruf mit der Laufzeit p, somit beträgt die Laufzeit $T(n) = p + p \cdot n \Rightarrow$ die Laufzeit des oben genannten Algorithmus ist selbst wieder polynomiell, da die Laufzeit höchstens ein Polynom vom Grad $grad(p) + 1$ ist. Das die Ausgabe ein legitimes Ergebnis ist, ist trivial und geht aus dem Algorithmus hervor.