Til Mohr, 405959
Andrés Montoya, 405409
Marc Ludevid Wulf, 405401

Aufgabe 4

a) Um zu zeigen, dass k-Independent Set in NP liegt, bauen wir uns einen Verifizierer:
Das Zertifikat hierbei ist eine Teilmenge von V, die wir V' nennen werden. V' soll hier genau k-viele Knoten besitzen und kann wie folgt kodiert werden:
$bin(u'_1)\$...bin(u'_i)\$...bin(u'_m)$, wobei alle u'_i Knoten aus V' sind, $0 \leq i \leq m \leq n := |V|$.
Der Verifizierer überprüft nun zuerst, ob die Codierung der Eingabe korrekt ist, dass V' tatsächlich eine Untermenge von V ist und, dass $m = k$ gilt. Dies ist offensichtlich in linearer Zeit möglich. Anschließend wird für alle Paare $e := (u'_i, u'_j)$ aus $V' \times V'$ überprüft, ob $e \notin E$ gilt. Dies ist in quadratischer Zeit möglich da alle Paare von Knoten einimal durchlaufen werden müssen.
Da die Laufzeit des Verifizierers polynomiell ist, liegt k-Independent Set in NP.

b) Unser Algorithmus geht wie folgt vor:
Der Algorithmus überprüft zunächst ob es eine Lösung gibt indem er den Graphen V an k-Independent übergibt. Wenn eine Lösung nicht möglich ist terminiert das Programm ohne eine gefundene Lösung.
Andernfalls startet der Algorithmus beim ersten Knoten des Graphens V und entfernt diesen. Dabei muss auch darauf geachtet werden, dass die Kanten entsprechend angepasst werden. Dieser modifizierte Graph wird an k-Independent übergeben. Falls dieser den Graphen verwirft heisst das, dass der entfernte Knoten in der Lösung sein muss und wir machen die Modifizierung wieder rückgängig und wiederholen diesen Prozess beim nächsten Knoten. Falls der modifizierte Graph jedoch akzeptiert wird fahren wir ohne den entfernten Knoten fort.
Der Algorithmus terminiert sobald alle Knoten durchlaufen wurden und gibt den modifizierten Graphen zurück.
Die Laufzeit ist offensichtlich polynomiell da der polynomielle Algorithmus der die Entscheidungsvariante löst linear oft aufgerufen wird.

Korrektheit:
Der zurückgegebene Graph hat offensichtlich zwei Eigenschaften: er wird von k-Independent akzeptiert und ist minimal, im Sinne von wenn ein weiterer Knoten entfernt wird ist er nicht mehr in k-Independent.
Dass dies ausreichend ist um eine Lösung zu produzieren ist nicht ganz so offensichtlich. Klar ist, dass der zurückgegebene Graph auf jeden Fall eine Untermenge von V ist und dass es mindestens k Knoten hat, da der zurückgegebene Graph von k-Independent akzeptiert wird. Zu beweisen ist aber noch, dass es genau k Knoten gibt und das keine dieser Knoten über eine Kante verbunden sind.
Dass es genau k Knoten gibt liegt daran das der zurückgegebene Graph minimal ist. O.b.D.A: Angenommen der zurückgegebene Graph hätte $k + 1$ Knoten. Da dieser Graph von k-Independent akzeptiert wird muss es eine Untermenge von diesem Graphen geben das ein k-Independent Set ist. Also müsste der Algorithmus k-Independent nach entfernen eines der Knoten immer noch akzeptiert haben. Somit hätte unser Suchalgorithmus einen der Knoten nicht wieder hizugefügt. Somit ist bewiesen das der produzierte Graph auf jeden Fall k Knoten hat.
Nun ist es auch offensichtlich, dass alle Knoten des zurückgegebenen Graphs nicht mit Kanten verbunden sind, da der Graph k Knoten hat und von k-Independent akzeptiert wird. Die in k-Independent gesuchte Untermenge kann nur der übergebene Graph sein da die Untermenge k Knoten haben muss.

Übungsblatt 08

Til Mohr, 405959
Andrés Montoya, 405409
Marc Ludevid Wulf, 405401

Aufgabe 5

a) 1) Dieser Graph ist in 2-Colorability:

Knoten	Farbe
1	1
2	1
3	2
4	2
5	1
6	2
7	1

2) Dieser Graph ist nicht in 2-Colorability:
Würde man Knoten 1 als 1 färben (wobei es prinzipiell hier egal ist, als was wir den Knoten 1 färben), so müssen Knoten 3 und 7 als 2 gefärbt werden. Deswegen müssten Knoten 2, 4 und 6 wieder als 1 gefärbt werden. Jedoch sind dann aber Knoten 2 und 4 immer gleich gefärbt. Da sie aber eine Kante zwischen sich besitzen, ist der Graph nicht in 2-Colorability.

b) Eine einfache, leicht abgeänderte Breitensuche sollte hier genügen:

1) Färbe alle Knoten 0 (noch nicht gefunden)
2) Färbe den ersten Knoten k mit aktueller Farbe 0 aus V 1.
3) Führe `BFS` auf k aus. Färbe dabei alle benachbarten Knoten, die Farbe 0 sind, in der von sich selber verschiedenen Farbe. Ist ein benachbarter Knoten schon eine von sich selber verschiedene Farbe, so laufe über diesen nicht mehr (schon besucht). Ist ein benachbarter Knoten aber dieselbe Farbe wie die eigene, so **verwerfe**.
4) Falls noch Knoten in V als 0 gefärbt sind, kehre zu Schritt `2)` zurück.
5) **Akzeptiere**.

Die Laufzeit ist die selbe der Breitensuche und somit polynomiell.

Korrektheit des Algorithmus:

- Falls der Algorithmus einen Graphen akzeptiert, wurde eine korrekte Färbung gefunden. Somit liegt der Graph in 2-Colorability.
- Falls der Algorithmus einen Graph verwirft wurden zwei benachbarte Knoten gefunden die die gleiche Farbe haben. Da die Färbung innerhalb eines zusammenhängenden Graphs bis auf das Vertauschen der Farben eindeutig ist kann es keine Färung geben sodass der Graph in 2-Colorability liegt.

c) Um zu zeigen, dass 3-Colorability in NP liegt, bauen wir uns einen Verifizierer: Das Zertifikat hierbei ist die Zuordnung von allen Knoten aus V zu den Farben $\{1, 2, 3\}$. Dabei das das Zertifikat folgende Kodierung:
$bin(k_0)\$bin(f_0)\$...bin(k_i)\$bin(f_i)\$...bin(k_n)\$bin(f_n)$, wobei $0 \leq i \leq n \coloneqq |V|$ gilt. Hierbei ist f_i die Farbe aus $\{1, 2, 3\}$, die dem jeweiligen Knoten k_i aus V zugeordnet wurde.
Der Verifizierer überprüft nun folgende Dinge:

- Zertifikat korrekt formatiert
- Für jede Kante $(k_i, k_j) \in E$ $(0 \leq i, j \leq n, i \neq j)$, ob $f_i \neq f_j$.

Damit liegt 3-Colorability in NP, da diese Überprüfung in polynomieller Zeit geschieht, da der erste Schritt in linearer und der zweite in quadratischer Laufzeit möglich ist.

Übungsblatt 08

Til Mohr, 405959
Andrés Montoya, 405409
Marc Ludevid Wulf, 405401

Aufgabe 6

Die nichtdeterministische 1-Band-TM die das Komplement von $L_{PALINDROM}$ erkennt funktioniert folgendermassen:
Die TM speichert auf einer Spur das Eingabewort und auf einer weiteren einen Counter. Die TM durchläuft das Eingabewort Buchstabe für Buchstabe und zählt die gegangenen Schritte. Nach jedem Schritt gibt es zwei Möglichkeiten. Entweder die TM geht an das Ende des Eingabeworts und geht dann so viele Schritte zurück wie im Counter steht um zu überprüfen ob dort der gleiche Buchstabe steht um ggf. die Eingabe zu akzeptieren falls die Buchstaben verschieden sind oder es geht mit dem nächsten Buchstaben weiter. Da die TM nur akzeptiert falls es einen Index i gibt sodass der i-te und der $n - i$-te Buchstabe verschieden sind akzeptiert diese TM nur Wörter die keine Palindrome sind. Die Laufzeit dieses Algorithmus ist in $O(n*log(n))$, da das Durchlaufen des Eingabeworts und danach das Prüfen der Gleichheit zwei korrespondenter Buchstaben in linearer Zeit möglich wäre, falls es den Counter nicht gäbe. Das Mitbewegen des Counters ist in logarithmische Zeit möglich da die maximale Länge des Counter logarithmische zur Länge des Eingabewortes wächst.