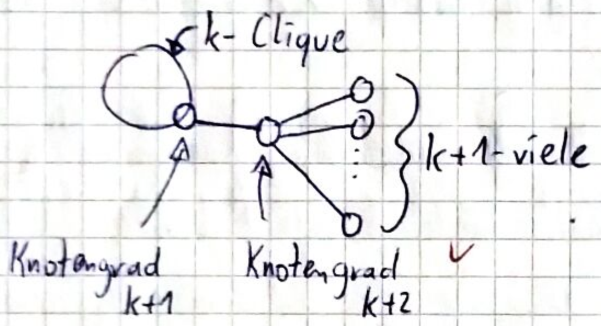
8.4 Fall 1: Der gewählte Knoten v mit maximalem Knotengrad k ist in einer Clique, dann ist diese Clique die größte im Graphen und wurde vom Algorithmus approximateClique gefunden.

4/4

=> 1- Approximativ

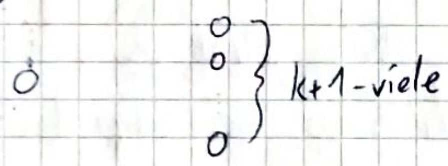
Fall 2: Der gewählte Knoten v mit maximalem Knotengrad k+2 ist nicht in einer Clique, wobei der Graph der Eingabe eine k-große Clique enthält.

Worst-case-Graph:



✓

In diesem Fall fliegt die komplette Clique raus, es bleiben folgende Knoten übrig:



Alle Knoten haben nun einen Knotengrad von 0 und sind nicht adjazent, d.h. es ist egal, welchen Knoten approximateClique als nächstes wählt, danach ist $V = \emptyset$ und der Algorithmus gibt $|C| = 2$ zurück.

=> Die k-Clique wird komplett übersehen, 2 wird als Ergebnis ausgegeben. Das heißt approximateClique ist im Worst-case $\frac{2k+2}{2} = \frac{n}{4}$ - approximativ (2k+2 war die Anzahl der Knoten)

=> approximateClique ist $\leq \frac{n}{2}$ -approximativ.

✓

8.2 Baue Graphen G mit: V_G stelle die Kreuzungen des Stadtplans dar
4/4
und E_G seien die Straßen zwischen den Kreuzungen.

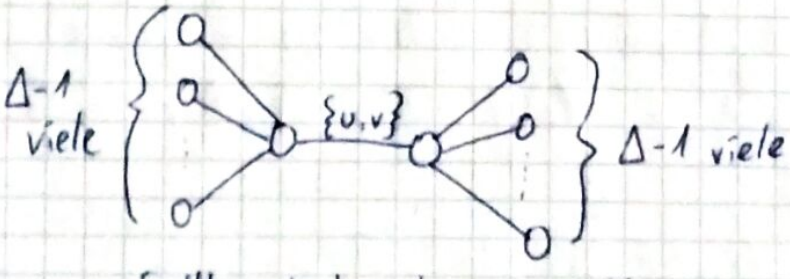
k sei die erhaltene Schranke für die Anzahl der Überwachungskameras.

Nun entspricht SURVEILLANCE dem VERTEX-COVER, das im Schnitger-Skript laut Satz 6.4 auf Seite 113 als NP-Vollständig bewiesen wurde.

8.3 a) Idee für den Greedy-Algorithmus: 6/6

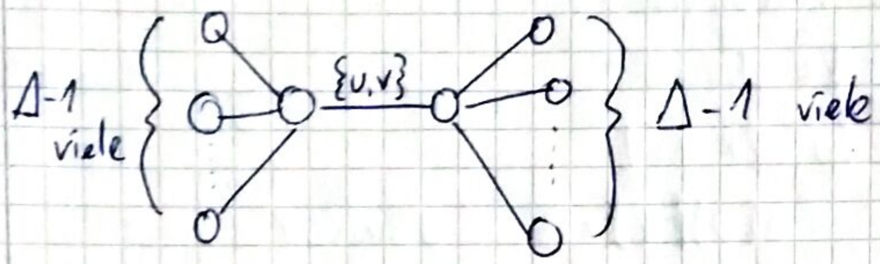
1) Wähle (irgend-) einen Knoten, der (noch) nicht gefärbte Kanten besitzt
2) Streiche alle Farben als "in-Frage-kommende", die bereits an diesem Knoten einer Kante zugeteilt wurden.
   Ebenso fallen alle Farben weg, die an dem Knoten bereits hängen, zu dem die einzufärbende Kante führt.
3) Färbe die Kante mit der ersten in Frage kommenden Farbe unter den übrig gebliebenen.

Warum hält der Algorithmus die Schranke von $2\Delta-1$ Farben ein? Betrachten wir den Worst-case:



Sollten links, als auch rechts, jeweils $\Delta-1$ viele Farben zum Einfärben der Kante $\{u,v\}$ wegfallen, so bleiben noch $(2\Delta-1)-(2\Delta-2) = 1$ Farbe übrig (Schubfachargument).
D.h. es können nie mehr als $2\Delta-2$ Farben für den Algorithmus wegfallen, er findet also immer für jede Kante unter $2\Delta-1$ Farben eine freie.

8.3 b) Erneut blicken wir auf den Worst Case aus dem Aufgabenteil (8.3a):



Dieser Graph hätte auch mit Δ-vielen Farben gefärbt werden können. Unser Algorithmus färbt ihn im Worst-case mit 2Δ-1 vielen. Es folgt folgender Approximationsfaktor:

Schön $$\frac{2\Delta-1}{\Delta} = \frac{2\Delta}{\Delta} - \frac{1}{\Delta} = 2 - \frac{1}{\Delta} \leq 2 \Rightarrow \text{2-Approx.}$$

8.5) Verfahre nach folgenden Basis- und Rekursionsfällen

7/8 $text(0,L) = 0$ // keine Worte mehr zu formatieren ? 0 entspricht einer optimal schönen Formatierung

$text(n,L) = \infty$, falls L negativ // Zeile darf nicht überfüllt werden

$$text(n,L) = \begin{cases} text(n, \text{Länge neue Zeile}), & \text{falls } L < 0 \\ \min\{text(n-1, L-|w|_n) + ugly(n-1, zeilenEnde, L), \\ \quad text(n, \text{Länge neue Zeile})\}, & \text{sonst} \end{cases}$$

Pseudocode auf der Rückseite

```
                         int i                          int L
Algorithmus   text(Anzahl zu formatierende Worte, Restlänge der Zeile){
                init Matrix a[i][L]
                int leereZeile = L
                int zeilenEnde = i
                for k = 1.. i
                    for l = 1.. L
                        a[0][l] = 0        //Basisfall
                for k = 1.. i
                    for l = 1.. L
                        if (L < 0)         //Zeile voll, starte neue Zeile
                            a[k][l] = a[k][leereZeile] *
                        else
                            a[k][l] = min{a[k-1][l-|w|_k] + ugly(k-1, zeilenEnde, l),
                                          a[k][leereZeile]} *
```

* sollte der Algo eine neue Zeile beschriften, also einen Zeilenumbruch setzen, muss die Variable zeilenEnde auf k gesetzt werden.

Schön! Laufzeit wird eingehalten, weil ...?