## Design Entscheidungen:

### GUI:

Die GUI hat eine kleine Überarbeitung bekommen:

* es gibt eine JMenuBar.
* In der MenuBar gibt es den Punkt „load“.
* Die Dropdownliste wurde entfernt und ihre Unterpunkte als Einträge unter „load“ eingefügt.
* Ein einfacher Button neben dem Dropdown wurde verworfen. Stattdessen wurde in der MenuBar ein Unterpunkt „algorithms“ angelegt.
* Dem Menu Punkt „algorithms“ wurden 3 Unterpunkte zugeordnet („BFS“, „DIJ“, „A\*“)
* Die Algorithmen können über ihre zugehörigen Menu Punkte aufgerufen werden.
* Die Auswahl zweier Knoten des Graphen wird nun über die GUI gemacht.
* Bei ungenügender Selektierung gibt es eine Fehlermeldung an den Benutzer(prompt)
* Ein gefundener Weg wird in einer Ausgabe dem Benutzer gemeldet.
* Ein nicht vorhandener Weg wird ebenfalls auf den Bildschirm gezeigt.

### Algorithmen:

Wir entschieden uns dazu die ShortestPath Algorithmen unter einem gemeinsamen Interface zu implementieren, da wir hier offensichtlich gleiche Interface Anforderungen an alle Implementationen haben.

Des Weiteren haben wir einen Teil der Implementation abstrahieren können und in einer abstrakten Klasse vereint.

Wir haben ein eigenes Material erstellt, welches zur Rückgabe eines Pfades verwendet werden kann, und somit die weitere Bearbeitung wesentlich vereinfachen würde.

Es kommt allerdings noch nicht in den ShortestPath Algorithmen zum Einsatz.

Alle Algorithmen und dazugehörige Materialen und Tests wurden innerhalb eines „Algorithm“ Pakets gekapselt.

### Tools:

Bei der Implementation des Graphenerzeugungswerkzeugs sind wir über die Aufgabenstellung hinausgegangen und haben zusätzlich einige Methoden hinzugefügt, welche in Zukunft hilfreich sein könnten.

Diese ermöglichen uns einen Graphen entweder mit einem Rad oder einem Kreis zu befüllen.

Weitere Methoden dieser Art sind geplant.

## 2. Textantworten:

### 4.(a):

Sofern mehrere kürzeste Wege im Graphen vorhanden sind kann nicht garantiert werden, dass alle *ShortestPath* Algorithmen denselben Pfad zurückliefern.

Es ist nur gegeben, dass alle eventuell zurückgegebenen Pfade die gleiche Länge bzw. Gewicht besitzen.

Diese Unterschiede hängen von der genauen Implementierung der Algorithmen ab.

Jedoch sollte jeder Algorithmus egal wie oft er auf demselben Graphen ausgeführt wird immer denselben Pfad zurückgeben.

### 4.(b):

Für einen Graphen mit negativem Kantengewicht kann nicht garantiert werden, dass tatsächlich ein kürzester Pfad gefunden wird.

Sollte eine negative Kante im Graphen enthalten sein, dann könnte der *DijkstraShortestPath* Algorithmus fälschlicherweise einen Knoten als „Ok“ markieren, obwohl es noch einen kürzeren Weg zu diesem gibt.

Daher ist die Tatsache, dass ein Graph keine negativen Kanten enthält, eine der Vorbedingungen welche erfüllt sein muss, damit der *DijkstraShortestPath* und auch *A\*ShortestPath* garantieren kann, dass ein kürzester Weg gefunden wird.

### 4.(c):

Ja, unsere Implementation generiert einen vollständig zufälligen Graphen.

Dieser Graph beinhaltet genau die angegebene Anzahl an Knoten und baut anschließend genau die angegebene Anzahl an zufälligen Kanten zwischen diese Knoten auf.

### 4.(d):

Wir testen generell zuerst unsere Implementationen gegeneinander, so können wir zumindest feststellen, ob alle Lösungen vermutlich korrekt sind(Sofern alle Implementationen einen Pfad der gleichen Länge zurückgeben).

Um nun relative Gewissheit zu erlangen erweitern wir unsere Tests mit einer weiteren Implementation, welche allerdings nicht von uns entwickelt wurde, sondern aus der JGraph Bibliothek eingebunden wurde.

Da diese Implementation schon lange Bestand in einer seriösen Bibliothek hatte, kann mit hoher Wahrscheinlichkeit erwartet werden, dass sie korrekt ist.

Wenn nun die Ergebnisse unserer Implementationen mit der aus JGraph Bibliothek übereinstimmen, dann können wir davon ausgehen, dass diese richtig sind.

### 4.(e):

Um die Menge aller kürzesten Wege zu berechnen dürfte der Algorithmus nicht nach auffinden einer Lösung aufhören, und müsste stattdessen weiterrechnen alle möglichen Wege berechnet wurden.

Man könnte die Berechnungen möglicherweise noch einschränken indem man nur genau solange rechnet bis alle Wege welche *startVertex* und *endVertex* miteinander verbinden, jedoch könnte der zusätzliche Aufwand um dies zu überprüfen den Aufwand die restlichen Wege zu prüfen übersteigen.

In jedem Fall wäre die Berechnung aller kürzesten Wege mit erheblichem Aufwand verbunden.

### 4.(f):

Nicht-Determinismus ist nicht im Algorithmus umsetzbar.

Es handelt sich bei nicht-Determinismus lediglich um ein Konstrukt welches benutzt werden kann um in Modellen die Komplexität einer Aufgabe niedrig zu halten.

## 3. Theorieteil:

### Aufgabe IV:

1.)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **∞** | **∞** | **∞** | **∞** | **∞** | **∞** | **∞** | **∞** | **∞** |
| **Vorgaenger** | **v0** |  |  |  |  |  |  |  |  |  |
| **Ok** |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **2. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **∞** | **10** | **40** | **∞** | **∞** | **1** | **∞** | **∞** | **80** |
| **Vorgaenger** | **v0** |  | **v0** | **v0** |  |  | **v0** |  |  | **v0** |
| **Ok** | **t** |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **3. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **46** | **10** | **40** | **∞** | **∞** | **1** | **3** | **9** | **80** |
| **Vorgaenger** | **v0** | **v1** | **v0** | **v0** |  |  | **v0** | **v6** | **v6** | **v0** |
| **Ok** | **t** |  |  |  |  |  | **t** |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **4. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **46** | **10** | **40** | **∞** | **∞** | **1** | **3** | **9** | **80** |
| **Vorgaenger** | **v0** | **v1** | **v0** | **v0** |  |  | **v0** | **v6** | **v6** | **v0** |
| **Ok** | **t** |  |  |  |  |  | **t** | **t** |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **5. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **46** | **10** | **40** | **∞** | **∞** | **1** | **3** | **9** | **80** |
| **Vorgaenger** | **v0** | **v1** | **v0** | **v0** |  |  | **v0** | **v6** | **v6** | **v0** |
| **Ok** | **t** |  |  |  |  |  | **t** | **t** | **t** |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **6. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **35** | **10** | **40** | **30** | **∞** | **1** | **3** | **9** | **80** |
| **Vorgaenger** | **v0** | **v2** | **v0** | **v0** | **v2** |  | **v0** | **v6** | **v6** | **v0** |
| **Ok** | **t** |  | **t** |  |  |  | **t** | **t** | **t** |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **7. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **35** | **10** | **40** | **30** | **80** | **1** | **3** | **9** | **80** |
| **Vorgaenger** | **v0** | **v2** | **v0** | **v0** | **v2** | **v4** | **v0** | **v6** | **v6** | **v0** |
| **Ok** | **t** |  | **t** |  | **t** |  | **t** | **t** | **t** |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **8. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **35** | **10** | **40** | **30** | **65** | **1** | **3** | **9** | **80** |
| **Vorgaenger** | **v0** | **v2** | **v0** | **v0** | **v2** | **v1** | **v0** | **v6** | **v6** | **v0** |
| **Ok** | **t** | **t** | **t** |  | **t** |  | **t** | **t** | **t** |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **9. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **35** | **10** | **40** | **30** | **65** | **1** | **3** | **9** | **80** |
| **Vorgaenger** | **v0** | **v2** | **v0** | **v0** | **v2** | **v1** | **v0** | **v6** | **v6** | **v0** |
| **Ok** | **t** | **t** | **t** | **t** | **t** |  | **t** | **t** | **t** |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **10. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Entfernung** | **0** | **35** | **10** | **40** | **30** | **65** | **1** | **3** | **9** | **80** |
| **Vorgaenger** | **v0** | **v2** | **v0** | **v0** | **v2** | **v1** | **v0** | **v6** | **v6** | **v0** |
| **Ok** | **t** | **t** | **t** | **t** | **t** | **t** | **t** | **t** | **t** |  |

2.)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Vorgaenger** | **v0** |  |  |  |  |  |  |  |  |  |
| **h** | **32** | **30** | **50** | **62** | **45** | **0** | **70** | **61** | **67** | **12** |
| **g** | **0** |  |  |  |  |  |  |  |  |  |
| **f** | **32** |  |  |  |  |  |  |  |  |  |
| **CL** |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **2. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Vorgaenger** | **v0** |  | **v0** | **v0** |  |  | **v0** |  |  | **v0** |
| **h** | **32** | **30** | **50** | **62** | **45** | **0** | **70** | **61** | **67** | **12** |
| **g** | **0** |  | **10** | **40** |  |  | **1** |  |  | **80** |
| **f** | **32** |  | **60** | **102** |  |  | **71** |  |  | **92** |
| **CL** | **t** |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **3. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Vorgaenger** | **v0** | **v2** | **v0** | **v0** | **v2** |  | **v0** |  |  | **v0** |
| **h** | **32** | **30** | **50** | **62** | **45** | **0** | **70** | **61** | **67** | **12** |
| **g** | **0** | **35** | **10** | **40** | **30** |  | **1** |  |  | **80** |
| **f** | **32** | **65** | **60** | **102** | **75** |  | **71** |  |  | **92** |
| **CL** | **t** |  | **t** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **4. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Vorgaenger** | **v0** | **v2** | **v0** | **v0** | **v2** | **v1** | **v0** |  |  | **v0** |
| **h** | **32** | **30** | **50** | **62** | **45** | **0** | **70** | **61** | **67** | **12** |
| **g** | **0** | **35** | **10** | **40** | **30** | **65** | **1** |  |  | **80** |
| **f** | **32** | **65** | **60** | **102** | **75** | **65** | **71** |  |  | **92** |
| **CL** | **t** | **t** | **t** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| **5. Schritt** | **v0** | **v1** | **v2** | **v3** | **v4** | **v5** | **v6** | **v7** | **v8** | **v9** |
| **Vorgaenger** | **v0** | **v2** | **v0** | **v0** | **v2** | **v1** | **v0** |  |  | **v0** |
| **h** | **32** | **30** | **50** | **62** | **45** | **0** | **70** | **61** | **67** | **12** |
| **g** | **0** | **35** | **10** | **40** | **30** | **65** | **1** |  |  | **80** |
| **f** | **32** | **65** | **60** | **102** | **75** | **65** | **71** |  |  | **92** |
| **CL** | **t** | **t** | **t** |  |  | **t** |  |  |  |  |

Der Pfad von v0 zu v5 ist: "v0-v2-v1-v5"

3.)

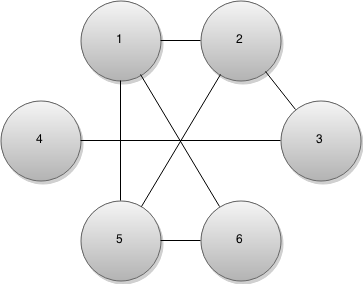
Wie an den beiden vorhergegangenen Tabellen zu erkennen haben beide Algorithmen den gleichen Pfad zurückgeliefert.

Zwar benötigt der A\* Algorithmus aufgrund der verwendeten Heuristik mehr Berechnungen pro Durchlauf, jedoch konnte er so den kürzesten Pfad in nur 5 Durchläufen erreicht.

Der Dijkstra brauchte im Vergleich schon 10 Durchläufe, ein nicht unerheblicherer zusätzlicher Leistungsaufwand.

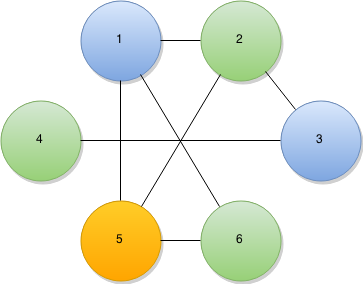
### Aufgabe V:

Um die Umstände zu verdeutlichen haben wir zuerst die Tabelle in einen Konfliktgraphen umgewandelt:



Anhand dieses Konfliktgraphen ist es nun einfach zu erkennen, dass wir hier an schlimmster Stelle einen Konflikt von 3 Sendern untereinander vorliegen haben, also eine 3er Clique.

Nun lässt sich aus dieser Information bereits schließen, dass der Graph 3-färbbar ist, wie in folgender Darstellung verdeutlicht wird:



Somit ist nun auch klar, dass mindestens 3 Fernsehkanäle benötigt werden, um sie der Aufgabenstellung gemäß auf die Stationen zu verteilen.

### Aufgabe VI:

1.)

Um eine Familie von Graphen, die kritisch *k*-chromatisch sind zu erzeugen nehmen wir einfach an, dass wie einen vollständigen Graphen *G* mit *|V| = k* besitzen.

Da in diesem Graph alle Knoten *v ∈ V* mit jedem anderen Knoten *v‘ ∈ V* verbunden sind werden genau *|V| = k* Farben benötigt um den Graphen zu färben, er ist also *k*-chromatisch.

Sollte nun irgendeine Kante aus diesem Graphen *G* entfernt werden, so besteht zwischen 2 Knoten *v, v‘ ∈ V* keine Verbindung mehr, sie können also dieselbe Farbe annehmen.

Somit benötigen wir nur noch *k-1* Farben um unseren Graphen einzufärben.

Unser Graph *G* ist also kritisch k-chromatisch■

2.)

Um eine Familie von Graphen mit *n = 3,5, …* Knoten, die kritisch *3*-chromatisch sind zu erzeugen nehmen wir einfach an, dass wir einen Graphen *G* mit *|V| = n* besitzen, welcher aus einem Kreis besteht.

Jeder Kreis mit einer ungeraden Anzahl an Knoten ist genau *3*-chromatisch, dass er auch genau kritisch *3*-chromatisch erkennen wir daran, dass wenn wir eine beliebige Kante aus dem Kreis entfernen dieser kein Kreis mehr ist, sondern lediglich eine Folge von Knoten welche genau 1zu1 verbunden sind.

Da wir nur noch Verbindungen zwischen genau *2* Knoten besitzen, also 2er Cliquen ist der gesamte Graph nun *2*-chromatisch.

Also war unser Ausgangsgraph kritisch *3*-chromatisch■