Schubert Julian, Gruppe 4
Philipp Wahl, Gruppe 4

Abgabe zum 2. Übungsblatt (AGT 21)

Aufgabe 1:

a)

t	u	v	w	x	y	z
∞	∞	∞	∞	0	∞	∞
∞	**2**	3	6	0	∞	∞
∞		**3**	6	0	12	∞
∞			**5**	0	12	9
9				0	12	**7**
9				0	10	
				0	**10**	

b) Die erste Abbildung zeigt ein Beispiel bei dem der Algorithmus von Dijkstra fehl schlägt. Startet man hier vom Knoten q0 bekommt im nächsten Schritt der Knoten q1 eine Distanz von 1 zugeteilt. Jedoch sind die tatsächlichen Kosten des Pfades zu q1 minus unendlich da die Katne von q1 nach q2 ein Gewicht von minus 4 hat, man also immer weiter im Kreis gehen kann und der Weg immer billiger wird.
Die zweite Abbildung zeigt ein Beispiel in dem der Algorighmus von Dijkstra

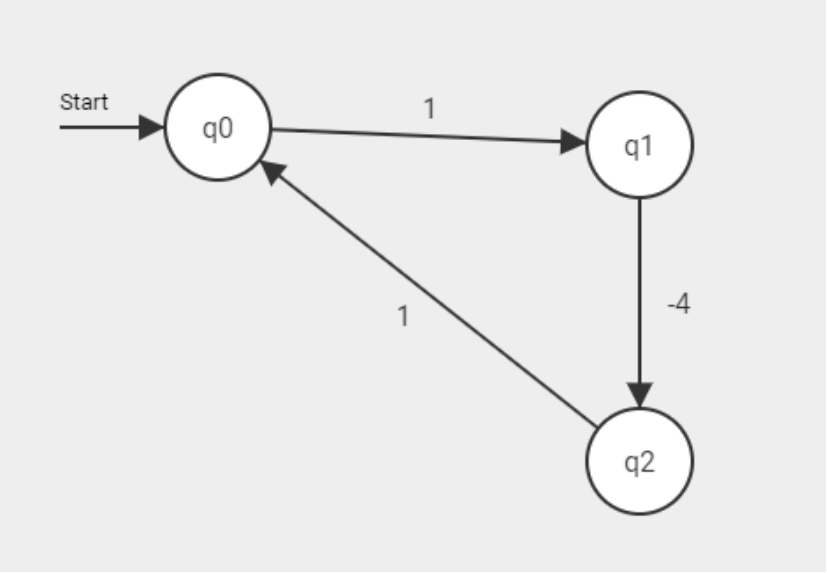

Abbildung 1: Hier schlägt Dijkstra fehl

funktioniert. Hier funktioniert der Algorithmus da nur ein einziges Mal eine negative Kante benutzt werden kann, und zwar die Kante die vom Start-Knoten 5 zum Knoten 1 führt

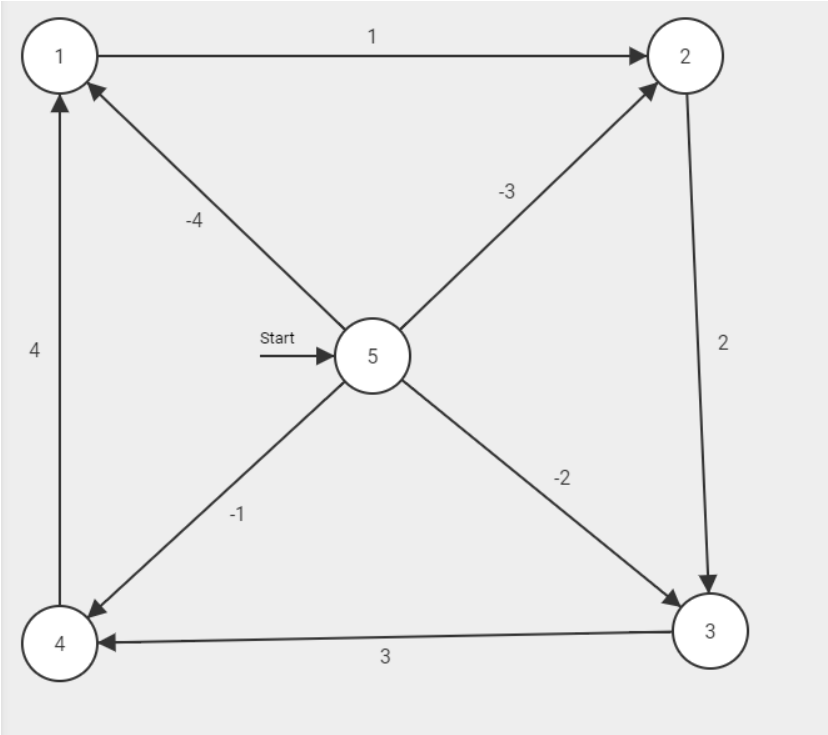

Abbildung 2: Hier funktioniert Dijkstra

1 Aufagbe 2

a) HatKreis(G, start)

```
// Speichert alle schon besuchten Knoten in Reihenfolge
besucht = []
besucht.append(start)
counter = 0
while besucht.length != |V| and counter != -1 do Wir besuchen jeden
 Knoten maximal ein mal
    // Nimm den ersten Knoten aus der Adjazenzliste vom aktuellen
     Knoten
    adj = besucht[counter].adj[0]
    if adj == nil then Zurück zum letzten Knoten gehen
        counter -=1
        continue
    // Entferne den Knoten von eben aus der Adjazenzliste damit er
     nicht nocheinmal besucht wird
    vistited[counter].adj.remove(adj)
    if adj.schonGesehen == true then Wenn wir auf einem Knoten
     gelandet sind den wir schonmal besucht haben existiert ein Kreis
        return true
    // Speichern das wir den aktuellen Knoten schon besucht haben
    ajd.schonGesehen = true
    // Zum nächsten Knoten gehen
    counter += 1
    visited.append(adj)
// Es wurde kein Kreis gefunden
return false
```

Die Idee des Algorithmus ist die folgende:
Man speichert jeden schonmal besuchten Knoten in eine Liste, das sorgt dafür, dass die While-Schleife maximal $|V|$-mal läuft. Wir wählen dann den ersten Knoten in der Adjazenzliste des aktuellen Knotens aus. Wenn dieser Knoten nicht existiert gehen wir zurück, besuchen also den vorletzen besuchten Knoten. Dann entfernen wir den gerade Besuchten Knoten aus der Adjazenzliste. Haben wir den Knoten schon einmal gesehen so sind wir im Kreis gelaufen, es existiert also ein Kreis. Sonst markieren wir den Knoten als bereits gesehen und gehen dann zum nächsten Knoten (welches wieder der erste Knoten aus der Adjazenzliste des aktuellen Knotens ist)

b) Laufzeit: Da wir jeden noch nicht besuchten Knoten sofort zu der Liste Hinzufügen wird die While-Schleife maximal $|V|$-mal ausgeführt.
Korrektheit: Da wir jeden Knoten besuchen und abspeichern ob der Knoten zuvor schon besucht wurde ist der Algorithmus korrekt.

c) Wenn unser counter auf -1 fällt bedeutet das im Algorithmus von oben das wir keinen Knoten mehr haben von dem wir aus weiter gehen könnten. Im nicht zusammenhängenden Graphen könnten wir eine zweite Liste einführen, in der zunächst alle Knoten gespeichert sind (also eine Kopie von $|V|$). Wird ein Knoten zu besucht hinzugefügt wird er aus der zweiten Liste entfernt. Wenn dann counter gleich -1 ist (es also von allen besuchten Knoten aus keinen weg mehr gibt den man gehen kann) wählen wir einen zufälligen Knoten aus der zweiten Liste aus (da diese Liste alle Knoten enthält die wir noch nicht besucht haben) und führen den Algorithmus von dort fort. Das stellt sicher das wir auch im nicht zusammenhängenden Graphen alle Knoten besuchen und somit alle Kreise finden.

2 Aufgabe 3

a) CompleteHamilton wählt (beginnend am Startknoten s) und fügt den Knoten mit billgster Distanz zu C hinzu. Prim wählt ebenfalls zunächst den Startknoten, da dieser mit Distanz 0 initialisert wird.
In der While-Schleife wählen wir in Complete-Hamilton den Knoten aus, der am nähesten an C dran ist und fügen diesen zu C hinzu. Prim fügt ebenfalls den Knoten der am nähesten an den bereits behandelten Knoten ist hinzu. Dies ist gegeben da wir in Relax alle Distanzen zu den jeweilig infrage kommenden Knoten updaten, und mit u = Q.ExtractMin() dann eben genau den Knoten wählen, der noch nicht besucht wurde und die geringste Distanz zu den bereits besuchten Knoten hat.

b) Um aus einem minimalen Spannbaum eine 2-Approximation für TSP zu erhalten werden zunächst alle Kanten verdoppelt, bereits besuchte Knoten übersprungen und dann eine Abkürzung eingefügt. Der Algorithmus completeHamilton erstellt bereits den Graphen mit den Abkürzungen, er erstellt also einen Hamiltonpfad. Um aus dem Pfad einen Kreis zu machen müssen nun nur noch Start und Endknoten verbunden werden. Da der Graph vollständig ist existiert eine solche direkte Kante. Durch die Dreiecksungleichung ist diese Kante sogar optimal (es gibt keinen Kürzeren Weg vom Startknoten zum Endknoten).