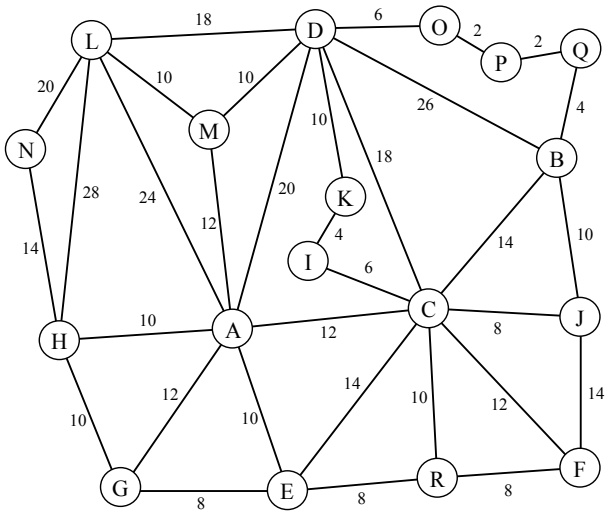
Übungsblatt 6

Abgabe: 09.07.2023

Aufgabe 1 Spannende Vernetzung (20 %)

In einer Stadt sind Kreuzungen (Knoten) durch Straßen (Kanten) verbunden. Jede Straße hat eine bestimmte Länge. Es soll nun ein Stromnetz aufgebaut werden, das alle Kreuzungspunkte mit Strom versorgt. Leitungen dürfen nur unter schon bestehenden Straßen verlegt werden, da das Aufreißen von Straßen, das Verlegen von Leitungen darunter, das anschließende Zubetonieren und die Leitungen selbst teuer sind. Es stellt sich die Frage, wie alle Kreuzungen durch ein möglichst kurzes, d.h. minimales Netz verbunden werden können.



Ermittelt den minimalen aufspannenden Baum aus dem Straßennetz mit dem Algorithmus von Jarnik, Prim und Dijkstra von Hand und erklärt schrittweise, wie ihr zu eurem Ergebnis gekommen seid.

Aufgabe 2 Sommer, Sonne, Routenplaner

Jede Reisende, die etwas auf sich hält, benutzt einen Routenplaner. Da Studierende bekanntermaßen nicht so viel Geld haben, müssen sie sich ihren selbst programmieren, und nutzen dazu natürlich gleich das, was sie gerade in der Vorlesung gehört haben, nämlich den Algorithmus von Dijkstra. Fehlen nur noch die Daten. Das Projekt *Open Street Map* bietet diese gratis an. Für die lokale Umgebung der Uni wurden diese Daten schon in Form der Dateien *nodes.txt* und *edges.txt* aufbereitet. Eine Datei *readme.txt* liegt auch dabei.

Generell dürft ihr in dieser Aufgabe Klassen aus der Laufzeitbibliothek verwenden. Achtet bei Sammlungen auf die Aufwandsklassen der Datenstrukturen.

Aufgabe 2.1 Karte aufbauen (30 %)

Erweitert die bereitgestellte Klasse *Map* so, dass ihr Konstruktor die beiden Dateien *nodes.txt* und *edges.txt* einliest und daraus einen Graphen konstruiert. Nutzt hierzu die ebenfalls bereitge-

stellten Klassen *Node* und *Edge*. Die Kanten in der Datei *edges.txt* sind ungerichtet, weshalb ihr sie in beide Richtungen in euren Graphen eintragen müsst. Implementiert die Methode *draw*, so dass diese die Karte zeichnet. Nach dem Starten des Programms *RoutePlanner* sollte die Karte nun im Fenster angezeigt werden.

Aufgabe 2.2 Positionen wählen (5 %)

Implementiert in der Klasse *Map* die Methode *getClosest*, die den dichtesten Knoten zu einer Position heraussucht. Wenn sie funktioniert, sollte es möglich sein, in die Karte zu klicken und ein Knoten in der Nähe wird rot markiert. Es sind maximal zwei Knoten rot markiert.

Aufgabe 2.3 Routenplanung (45 %)

Implementiert die Methode *shortestPath* in der Klasse *RoutePlanner* als eine kürzeste-Wege-Suche nach nach Dijkstra. Zeichnet die durchsuchten Kanten ein, z.B. in blau. Zeichnet den kürzesten Weg in einer anderen Farbe ein, z.B. rot. Abhängig davon, für welche Implementierung ihr euch entscheidet (Rand separat oder in den Knoten speichern), müsst ihr möglicherweise die bereitgestellten Klassen *Node* und *Map* zusätzlich erweitern, z.B. die Methode *Map.reset()* vervollständigen, die automatisch immer vor jedem Aufruf von *shortestPath* ausgeführt wird.

Testet interaktiv, d.h. wählt Start- und Zielknoten aus und beurteilt, ob das Ergebnis plausibel ist. Jeder Klick wählt einen neuen Zielknoten, das bisherige Ziel wird zum Startknoten. Beachtet, dass nicht alle Knoten in der Karte miteinander verbunden sind.

Aufgabe 3 Bonusmeilen

Aufgabe 3.1 Schneller Reisen (5 %)

Ordnet den verschiedenen Kantentypen unterschiedliche Geschwindigkeiten zu und nutzt diese in Kombination mit der Länge der Kanten zur Bewertung der Wege. Sucht dann nach dem schnellsten Weg statt nach dem kürzesten.

Aufgabe 3.2 Schneller Finden (5 %)

Ihr habt sicher schon gemerkt, dass sehr viel von der Karte durchsucht wird, sobald Start- und Zielpunkt weiter auseinanderliegen. Das kann deutlich reduziert werden, wenn etwas mehr Wissen eingebracht wird. Beim A^*-Suchalgorithmus wird der Rand nicht mehr nur allein nach den bereits angesammelten Kosten geordnet, sondern nach den bisherigen Kosten plus den Kosten, die *mindestens* bis zum Ziel noch entstehen werden. Für die Suche nach dem kürzesten Weg ist eine solche Abschätzung leicht zu finden. Implementiert dies. Warum wäre es für den schnellsten Weg deutlich schwieriger?