Dennis Paust 17.06.2020

# Dijkstra Dokumentation

## *Programmumfang*

### Bedienung der Website

- Es lässt sich mittels des Dijkstra-Algorithmus die jeweils kürzeste Route von einem Punkt zum anderen Bestimmen.
- Der Nutzer kann Knotenpunkte (Nodes) sowie Verbindungen (Vertex / Edge) zwischen ihnen in wechselseitige Richtungen (directed Edge) erstellen verschieben und löschen.
- Die Distanzen zwischen ihnen (weight) ergibt sich durch den Abstand [px] von den jeweiligen Mitten der Knotenpunkte. Der Umfang eines Knotens, sowie dessen Namensgebung und Farbe, sind trivial.
- Bedienungshinweise befinden sich zusammen mit einem Verweis zum Quellcode auf der Website); sollte es Probleme geben, muss ggf. der Suchmodus (aktiver Dijkstra-Algorithmus) durch Betätigen des Buttons oben rechts deaktiviert werden

### Der Dijkstra-Algorithmus

- Es lässt sich jede Iteration des Dijkstra-Algorithmus (bis auf die Initialisierung natürlich) einzeln ausführen oder an jedem gegebenen Schritt bis zum Ende überspringen.
- Der Nutzer kann während der Ausführung des Dijkstra-Algorithmus keine Änderungen an der Website vornehmen, um die Integrität vorheriger Schritte des Algorithmus zu wahren.
- Jede Iteration gibt visuellen Aufschluss über
    - nicht erreichte Knotenpunkte (ungesättigte Farbe),
    - aktuell unzugängliche Verbindungen (transparent-weiß),
    - den kürzesten Weg vom Startpunkt zu jedem einzelnen bereits erreichten Knotenpunkt (grüne Verbindungen) und
    - die kürzeste Route vom Start- zum Endpunkt nach Abschluss des Dijkstra-Algorithmus (hellgrün mit hervorgehobenen Richtungs- und Distanzangaben derselben Farbe)
- Es wird empfohlen die Konsolenausgaben (mit Tastaturbefehl: F12) zu verfolgen, um zusätzliche Informationen zu
    - erstellten Knotenpunkten und Verbindungen zwischen ihnen mit ihren zugehörigen Referenzen zu jeweiligen Klasseninstanzen, welche sämtliche relevante Information zu einem Objekt enthalten, zu erhalten,
    - der Prioritätsschlange des Dijkstra-Algorithmus einzusehen und
    - der Auswertung der Ergebnisroute (kein möglicher Weg existent / kürzester Weg mit Distanzangabe und nummerierter Ausgabe jedes Objektes auf der Route) zu bekommen.

### *Betriebsplattform, Programmumgebung & Sprachen*

- Das Projekt ist als Website unter https://dango301.github.io/Dijkstra öffentlich zugänglich. Durch Nutzung des Parcel Moduls ist der Code auf ECMA5-Version kompiliert und auf allen Browsern funktional.
- Verwendete Sprachen sind HTML, SASS und Typescript (durch Parcel automatisch auf Browser-kompatibles CSS bzw. Javascript kompiliert). Der Verweis zum Quellcode ist durch Klicken auf den Titeltext auf der Website oder unter https://github.com/dango301/Dijkstra verfügbar.
- Die Programmierung erfolgte durch Node.js welche über den Node-Package-Manager (NPM) das Parcel Modul und andere bereitstellte, obwohl keine weiteren externe Module oder Plugins genutzt wurden. Parcel kompiliert die Superset-Sprache Typescript auf unkonventionellem ECMA6 Standard mit erweiterten Funktionen auf die Mindestanforderungen zeitgemäßer Browser (ES5), sodass die Funktionalität des Codes gewährleistet ist.
- Der verwendete IDE ist Visual Studio Code, welcher u. a. aufgrund der anpassbaren IntelliSense-Codevervollständigung, Farbgervorhebungen und vereinfachtes Refactoring für sämtliche Programmiersprachen meine Standardwahl ist und bemerkenswert personalisiert an meinen Workflow geworden ist.

### *Leistung & Quellen*

- Es wurde keinerlei fremder Code kopiert oder übernommen.
- Die Konzeption und Darstellung des Projekts sind eigene Ideen, obwohl die konkrete Visualisierung einzelner Iterationsschritte später durch http://www.gitta.info/Accessibiliti/de/html/Dijkstra_learningObject2.html inspiriert wurde.
- Die einzige Quelle aus konkreter Recherche, bis auf zahlreiche Google-Suchen für einzelne sprachspezifische Befehle (z.B. “Array.includes() or Array.contains() ?”) und Recherche für Spezifikationen von Sets und Klassen in Typescript in deren Dokumentationen, ist: https://youtu.be/GazC3A4OQTE (Dijkstra-Algorithmus nach vorgestelltem Vorgehen konzipiert).

### *Programmstruktur*

Die beiden Referenzen zu den kompilierten CSS und JS Dateien befinden sich in "index.html" (ca. 50 Zeilen). Die Typescript Datei “main.ts” importiert 4 weitere Dateien, welche der Übersicht halber aufgeteilt wurden und auch hier nacheinander erklärt werden.

#### Styles in "main.sass" (ca. 220 Zeilen)

Die Datei besteht zur einen Hälfte aus standardmäßigen Eigenschaften für die gesamte Seite und zur anderen Hälfte aus situationsgemäßen Overrides für den aktiven Dijkstra-Algorithmus.

**"main.ts" (ca. 250 Zeilen)**

- weist allen Buttons und Mouseevents ihre Funktionen zu, zu erwähnen sind die:
    - Bestimmung von Start- und Endpunkt (nacheinander) des Dijkstra-Algorithmus sowie
    - die De- / Aktivierung der Bewegung des angeklickten Knotenpunktes, der Löschung eines Elements, der Erstellung einer neuen Verbindung und der Erstellung eines neuen Knotenpunktes
- globale Variablen sind:
    - cNode: HTMLElement → aktuell zu erstellendem Knotenpunkt zugewiesen
    - dragging: HTMLElement → aktuell zu verschiebendem Element zugewiesen
    - cVertex: HTMLElement → aktuell zu erstellender Verbindung zugewiesen
    - deletion = { timeout: float, el: HTMLElement } → zur Überprüfung des Zeitlimits beim Löschen mit Referenz zum zu löschenden Element
    - computing: boolean → stoppt bei aktiver Dijkstra-Suche Veränderungen an der Seite

    ⇒ bei Null/False-Werten einer dieser Variablen ist der jeweilige Modus soz. deaktiviert und hält den Nutzer davon ab mehrere Aktionen gleichzeitig durchzuführen

**“classes.ts” (ca. 170 Zeilen)**

- enthält Definitionen der Klassen _Node, Vertex, SeachNode und SearchVertex
- (Es wird stark empfohlen sich derer Eigenschaften bewusst zu sein und die Bedeutung der Attribute zu verstehen; da diese durch ihre Namen und Typen selbsterklärend sind, wird nur auf das Modul verwiesen, das hier einzusehen ist.)

**“node.ts” (ca. 100 Zeilen)**

- exportiert im Wesentlichen die Funktionen createNode, scaleNode und moveNode, welche aus den eventHandlern in “main.ts” gerufen werden
- hauptsächlich Positionierung (z.B. offset vom Ursprung des Elements in der oberen linken Ecke des Knotens zur Mitte des Elements beim Bewegen) und Festlegung wichtiger CSS Eigenschaften

**“vertices.ts” (ca. 80 Zeilen)**

- exportieren die Funktionenen createVertex, moveVertex und endVertex, welche aus den eventHandlern in “main.ts” sowie in moveNode aus “nodes.ts” (entsprechende Bewegung aller Verbindungen bei Bewegung eines Knotenpunkts) gerufen werden
- hauptsächlich Positionierung (z.B. korrekte bzw. leserliche Ausrichtung von Richtungspfeilen und Distanzwerten) und Herstellung von Referenzen zwischen verbundenen Knoten

**“dijkstra.ts” (ca. 110 Zeilen)**

- Initialisierung, gerufen aus “main.ts”, und Umwandlung aller Instanzen von _Node zu SearchNode mit entsprechend angepassten Attributen (z. B. kombinierte Eigenschaften aus den Klassen _Node und Vertex sowie Eigenschaften, die nur bei der Dijkstra-Suche relevant sind)
- Funktionen nextStep und processNode, gerufen durch Button-Events in “main.ts”, verarbeiten jeweils das erste Element der Prioritätsschlange bis sie leer ist oder der Endpunkt gefunden ist

## *Reflexion*

### Leichtigkeiten

- Ohne größere Probleme ließen sich die HTML und SASS Dateien schreiben, da diese verhältnismäßig simpel sind und auf viel Erfahrung basieren. Das Schreiben der Styles war jedoch recht zeitintensiv.
- Der Dijkstra-Algorithmus war ironischerweise nicht der Schwierigste, den ich für dieses Projekt schreiben musste, da dieser recht simpel ist.
- Es hat Spaß gemacht Trigonometrie aus dem Schulunterricht so vielseitig, besonders aus dem Kopf und ohne Nachschlagen, anzuwenden :)

### Schwierigkeiten

- Eine neue Herausforderung war hier die Nutzung von SVG Dateien in Pseudoklassen (relative Positionierung, relative Dateipfade, Größe und Sichtbarkeit unter Text). Dies war jedoch eine nützliche Lektion für mich, da Pseudoklassen und -selektoren extrem nützlich im Repertoire sind.
- Die Dateien waren extrem unübersichtlich während des Schreibens, erschwert durch die Länge des Codes und Vielzahl an Modulen, und ließen sich erst am Ende refaktorisieren.
- Die Kommunikation zwischen den Typescript-Modulen war etwas umständlich (etliche Funktionsparameter, die weitergegeben werden müssen), scheint rückblickend, trotz ständiger Restrukturierungen des Codes, jedoch die eleganteste Lösung zu sein.
- Die korrekte Ausrichtung von Richtungspfeilen und Distanzwerten (besonders bei bidirektionalen Verbindungen!) in moveVertex war etwas frustrierend und besser durch Ausprobieren von Lösungswegen und ohne das Hineinsteigern in die Math-Library zu lösen.
- Ein Durchbruch war die Entdeckung von statischen Attributen und Funktionen in Klassen, wodurch die IDs eindeutig und ohne Fehlermöglichkeiten bei der Initialisierung einer Instanz zugewiesen werden können. Die Instanzen werden dabei einem Array als statisches Klassenattribut zugefügt, wodurch sie sich mit den HTML-Elementen identifizieren lassen und vice versa.

**Rückblick auf Leistungen**

- Unglücklicherweise wurde so viel Wert auf möglichst aussagekräftiges Debugging gelegt, dass die Darstellung teilweise interessanter Informationen (z.B. aktuelle Prioritätsschlange, Distanz, Routenschritte bei Dijkstra-Suche) auf der Website redundant wurde und nun "hinter den Kulissen" in den Konsolenausgaben des Fensters bleiben.
- Die Verwendung von globalen Variablen zur Kontrolle der Modi ist weniger elegant (ständige Konditionsabfragen), aber die einfachste (mir bekannte) Methode, um simultane und potentiell ambivalente User-Aktionen zu verhindern.
- Es wurde sehr viel Zeit in das Refaktorisieren und Verbessern der Klassen _Node und SearchNode investiert. Die Klassenvererbung (class inheritance), d. h. das _Node zum Super von SearchNode wird, wurde mehrfach versucht, stellte sich jedoch nicht als zielführend, sondern Komplexität um der Komplexität Willen heraus. Ihre Methode setPaths erforderte sehr viel Arbeit, welche schließlich nicht sichtbar ist, da der Code nun gut leserlich und prägnant ist, was sich schließlich auch auf die Kürze und Simplizität des Dijkstra-Algorithmus ausgewirkt hat.
- Es wurden alle Ideen und (zugegeben) "Gimmicks" exakt so ausgearbeitet, wie zu Beginn erdacht (was doch sehr selten und zugleich erfreulich ist).