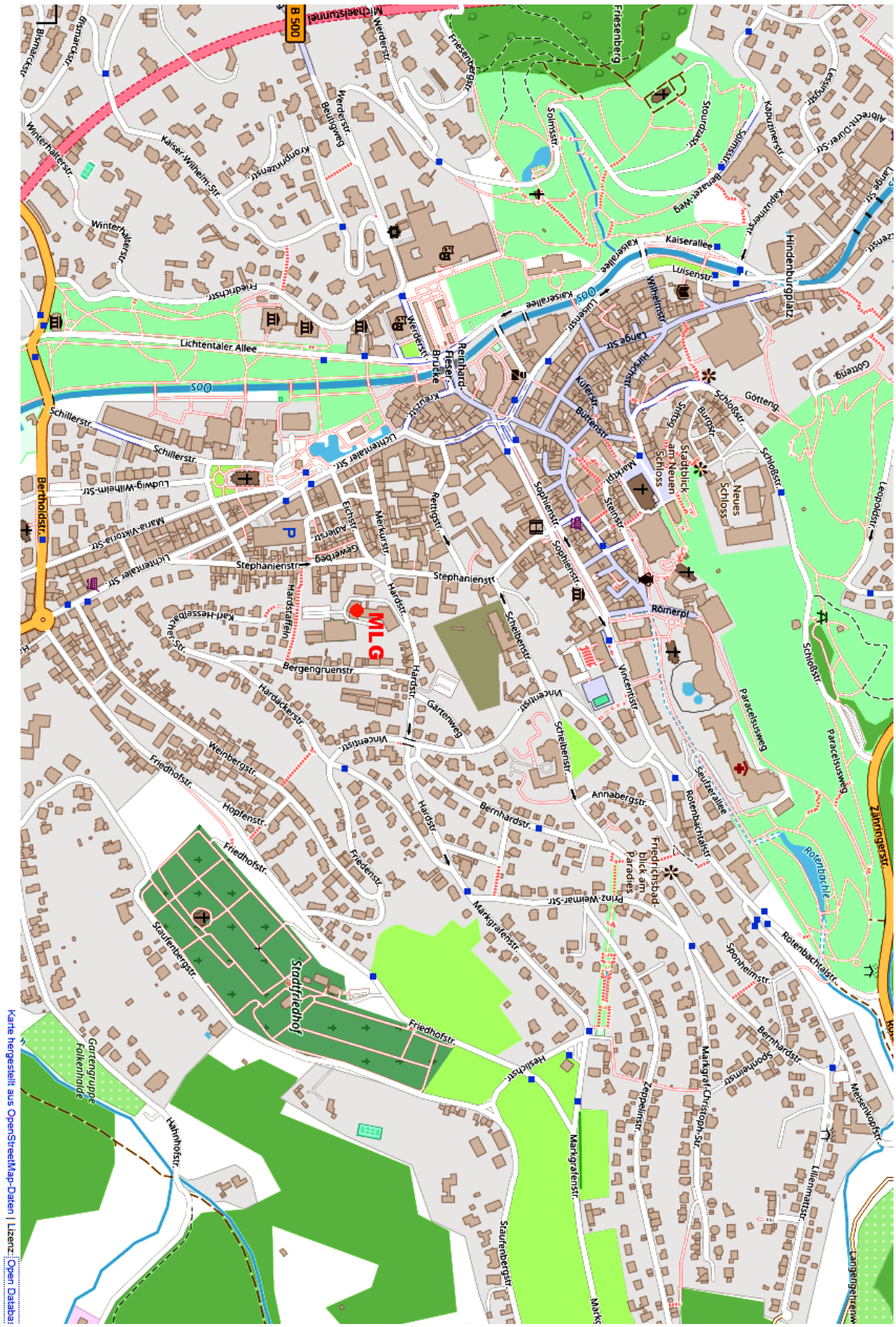
Karte von Baden-Baden



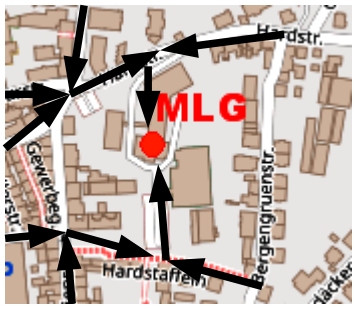
Kartenausschnitt Baden-Baden Karte hergestellt aus OpenStreetMap-Daten (Lizenz: Open Database License (ODbL))

Problemstellung

Das Markgrafen Ludwig Gymnasium (MLG) in Baden-Baden möchte seinen neuen Fünftklässlern einen neuen Service anbieten: Sie bekommen eine Karte, auf der für jedes Haus in Baden-Baden der kürzeste Fußweg zur Schule eingezeichnet ist. Der Informatikkurs der Oberstufe soll ein Programm entwerfen, das diese Karte berechnet.

Da das doch sehr viele Häuser sind, soll zunächst nur für jede Kreuzung der beste Weg bestimmt werden.

Aufgabe:



1. *Zeichne diese Karte. Markiere dazu für jeden Wegabschnitt (von Kreuzung zu Kreuzung), in welche Richtung man laufen soll. Beachte, dass Baden-Baden viele Treppenaufgänge hat, die rot gestrichelt eingezeichnet sind.*

   Individuelle Lösung. Die kürzesten Wege sind so schwer zu ermitteln. Man vergleicht am Ende mit dem Ergebnis des Dijkstra-Algorithmus.

2. *Alle Schüler, die an einer Straße wohnen, die nun mit einem Pfeil markiert ist, müssen nur den Pfeilen folgen. Was machen die Schüler, die in einer Straße wohnen, die nicht markiert ist?*

   Diese Schüler können zu einer der beiden nächstgelegenen Kreuzungen laufen. Die Entfernung zur Kreuzung plus die Entfernung von der Kreuzung zur Schule entscheidet, welche Kreuzung besser ist.

Modellierung

Die Ausgangssituation soll nun als Graph modelliert werden.

3. *Entscheide, welche der folgenden Informationen wichtig für die Bestimmung sind:*
   - Position des Markgrafen-Ludwig-Gymnasiums
   - genauer Verlauf der Straßen
   - Name der Straßen
   - Verknüpfung der Straßen über Straßenkreuzungen
   - Abstände der Kreuzungen voneinander
   - Positionen der Häuser in der Straße
   - Art des Weges (Bundesstraße, Fußgängerzone, Treppen)

   Es ist wichtig, wie die Straßen untereinander verbunden sind und wie lang die Entfernung von Kreuzung zu Kreuzung ist. Damit kann man die Entfernungen vom MLG zu den Kreuzungen bestimmen. Die Positionen der Häuser sind wichtig, wenn man entscheiden möchte, zu welcher Kreuzung man als erstes läuft.

Modellierung

Knoten:

Die Knoten repräsentieren die Straßenkreuzungen. Es wäre auch möglich, jedes einzelne Haus als Knoten zu modellieren. Der Wert eines Knotens repräsentiert die Entfernung zum Startknoten.

Kanten:

Zwei Knoten sind durch eine Kante verbunden, wenn eine Straße zwischen diesen Kreuzungen verläuft. Die Entfernung der beiden Knoten ist das Gewicht der Kante. Hinweis: man könnte statt der Entfernung in Metern auch die benötigte Zeit verwenden.

Kürzeste Entfernung in ungerichteten Graphen

Geben ist ein Graph und ein Startknoten.
Bestimme die kürzeste Entfernung zu allen anderen Knoten unter Berücksichtigung des Gewichts der Kanten.

Weiterführende Fragen

Aufgaben

4. *Untersuche, wie man erreichen könnte, dass Fußgängerzonen bevorzugt und Bundesstraßen möglichst vermieden werden, wenn der Weg dadurch nicht zu lang wird.*

   Man könnte die Entfernungen in Fußgängerzonen mit einem Faktor kleiner 1 gewichteten und die Bundesstraßen mit einem Faktor größer 1. Dadurch würden bei gleicher oder ähnlicher Entfernung Fußgängerzonen bevorzugt und Bundesstraßen vermieden.

5. *Untersuche, was passiert, wenn man nicht den Knoten mit dem geringsten Wert als nächstes auswertet.*

   Wertet man einen Knoten mit höherem Wert zuerst aus, gibt es einen Knoten mit geringerem Gewicht, der danach ausgewertet wird. Es könnte passieren, dass von diesem Knoten eine Kante zu dem schon ausgewerteten führt, die einen kürzeren Weg zu diesem Knoten darstellt, als der zuvor berechnete Weg. Dann müsste dieser Knoten und alle davon abhängenden Knoten erneut ausgewertet werden. Dadurch würde die Laufzeit des Algorithmus extrem ansteigen.

6. *Untersuche, wie der Algorithmus damit zurechtkommt, wenn der Graph auch negative Kantengewichte enthält.*

   Der Dijkstra-Algorithmus kann damit nicht arbeiten. Die Grundvoraussetzung, dass der Knoten mit dem geringsten Wert nachträglich nicht mehr geändert werden muss, stimmt durch die negativen Kantengewichte nicht mehr.

Für viele Anwendungen verwendet man gerichtete Graphen, d.h. die Kanten haben eine Richtung (z.B. bei einem Stadtplan mit Einbahnstraßen). Die Kanten dürfen dann nur in der vorgegebenen Richtung durchlaufen werden.

Aufgaben

7. *Untersuche, wie sich der Algorithmus auf einem gerichteten Graph verhält.*

   Gerichtete Kanten sind für den Algorithmus kein Problem. Es sind keine Änderungen am Algorithmus notwendig.

Beschreibung des Dijkstra-Algorithmus

Um das Problem des kürzesten Pfades in einem gewichteten Graphen zu lösen, wird der Algorithmus A von E. Moore angepasst:

- Moore hat bei den unbesuchten Nachbarknoten den Wert des aktuellen Knotens plus 1 eingetragen. Dijkstra berechnet den Wert des Nachbarknotens als Wert des aktuellen Knotens plus das Gewicht der Kante dorthin. Außerdem schaut er sich auch schon besuchte (aber noch unmarkierte) Knoten an und passt deren Wert an, wenn der neu berechnete Wert kleiner als der bisher dort eingetragene ist (neue kürzeste Entfernung).
- Moore hat jeden besuchten Knoten in eine Schlange eingetragen und aus der Schlange den nächsten zu bearbeitenden Knoten entnommen. Dieser war automatisch einer der Knoten mit dem geringsten Wert. Bei Dijkstra können später in die ToDo-Liste eingefügte Knoten einen geringeren Wert haben als zuvor eingefügte. Trotzdem muss der Knoten der ToDo-Liste mit dem geringsten Wert als nächstes bearbeitet werden. Daher kann man die ToDo-Liste jedes Mal sortieren und dann den ersten entnehmen.
- Möchte man die kürzesten Wege sichtbar machen, muss eine Kante markiert werden, wenn diese verwendet wurde, um den Wert eines Knotens anzupassen. War schon eine andere Kante zu diesem Knoten markiert, kann man diese als gelöscht markieren. Führt die Auswertung einer Kante nicht zu einer Anpassung des Wertes des Nachbarknotens wird die Kante auch als gelöscht markiert.

Pseudocode des Dijkstra-Algorithmus

Dijkstra-Algorithmus:

```
Setze den Entfernungswert des Startknotens auf 0
Markiere ihn als besucht
Erzeuge eine leere ToDo-Liste und füge den Startknoten hinzu

Wiederhole solange die ToDo-Liste nicht leer ist:
  Sortiere die ToDo-Liste aufsteigend
  Nimm den ersten Knoten k aus der ToDo-Liste heraus
  Sein Entfernungswert ist d
  Markiere k
  Wiederhole für jeden seiner Nachbarn:
    Falls der Nachbarknoten n noch nicht markiert ist:
      Bestimme die Kante ka dorthin
      Falls n noch nicht besucht ist
        oder d + Gewicht von ka < Entfernungswert von n:
         Kennzeichne n als "besucht"
         Setze den Entfernungswert von n auf d+Gewicht von ka
         Füge n am Ende der ToDo-Liste hinzu, wenn n nicht schon in
                   der ToDo-Liste enthalten ist
      Ende-Falls
    Ende-Falls
  Ende-Wiederhole
Ende-Wiederhole
```

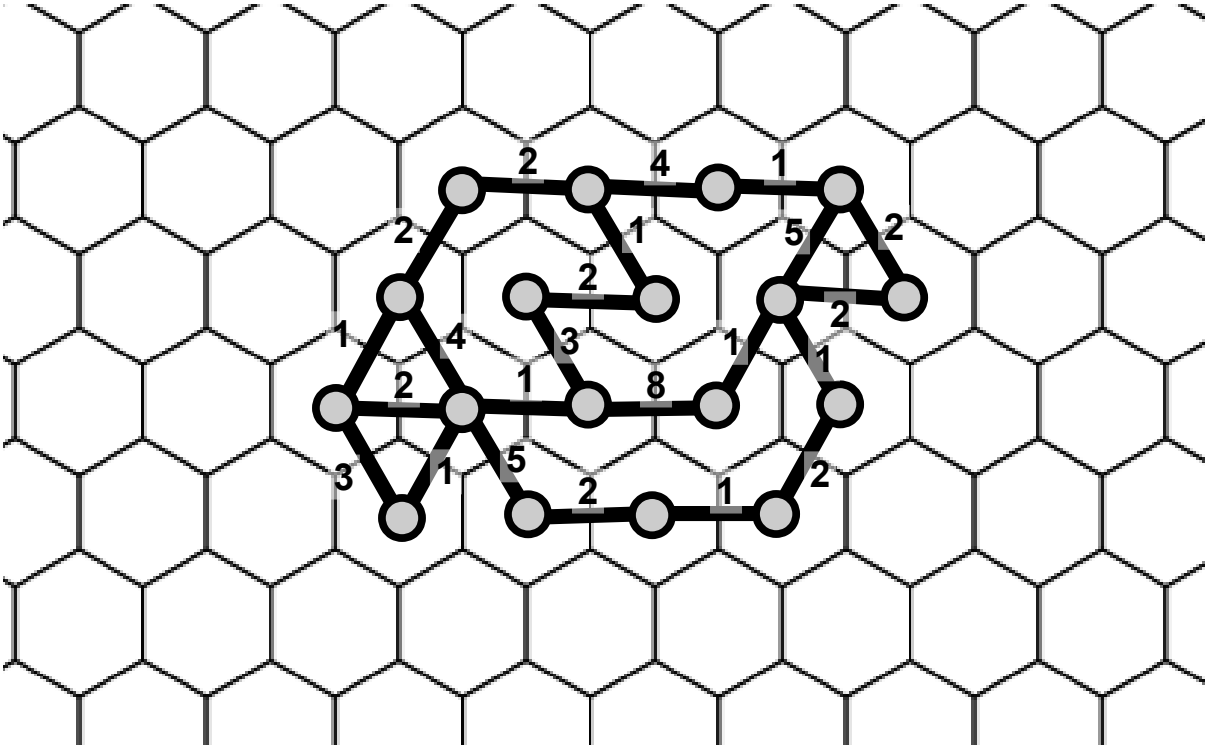
Quelltext des Dijkstra-Algorithmus

```
public void fuehreAlgorithmusAus() {
  if (g.getAnzahlKnoten()==0) {
    return;
  }

  ArrayList<Knoten> toDo = new ArrayList<Knoten>();
  getStartKnoten().setBesucht(true);
  getStartKnoten().setWert(0);
  toDo.add(getStartKnoten());

  while(toDo.size()>0) {
    Collections.sort(toDo);
    Knoten k = toDo.remove(0);
    k.setMarkiert(true);
    for(Knoten n : g.getNachbarKnoten(k)) {
      if(!n.isMarkiert()){
        Kante ka = g.getKante(k, n);
        if(!n.isBesucht() ||
           k.getDoubleWert()+ka.getGewicht() < n.getDoubleWert()){
          if(n.isBesucht()) {
            List<Kante> eingehend = g.getEingehendeKanten(n,
                ka2 -> !ka2.isGeloescht() && ka2.isMarkiert());
            Kante alterWeg = eingehend.get(0);
            alterWeg.setGeloescht(true);
            alterWeg.setMarkiert(false);
          }
          n.setWert(k.getIntWert()+ka.getGewicht());
          if(!toDo.contains(n)) toDo.add(n);
          ka.setMarkiert(true);
          n.setBesucht(true);
        } else {
          ka.setGeloescht(true);
        }
      }
    }
  }
}
```

Beispiel 1



Beispiel 2