Projekt 1: Roberto Panizza & Loris Trifoglio

# Zielsetzung & Rahmen

Ziel war es, eine gemeinsame Schweizer Karte in einem einheitlichen CSV-Format zu erzeugen, bestehende Algorithmen auf diesem Datensatz auszuführen und zu beurteilen sowie einen zusätzlichen Algorithmus/Optimierung zu implementieren (hier: Beam Search). Die Resultate werden in einem Git-Repository abgelegt und reproduzierbar gemacht.

# Abgabe:

Der vollständige Code (inkl. aller Anpassungen) liegt unter:

<https://github.com/rpa-private/wi_softwareengineering_2023_roberto_loris>

wi\_softwareengineering\_2023\_roberto\_loris/src/main/java/ch/fhnw/richards/Week\_03/project1

Die CSV-Dateien liegen ebenfalls im ZIP-Ordner in diesem Repo. Das Zip File muss lokal entpackt werden und danach der Pfad zum Ordner Nodes\_Edges in der Datei MapData.java angegeben werden.

Es sind alle Dateien im Repo. Die Struktur und Abgabekriterien entsprechen der Aufgabenstellung.

# Phase 1: Datenaufbereitung

Hier ist unsere Karte: <https://s.geo.admin.ch/syvoz9cgz26e>

Damit wir nicht alles manuell abfüllen mussten, haben wir aus Zeitgründen mithilfe von ChatGPT aus dem kombinierten KML-File zwei CSV-Dateien erstellen lassen, mit allen Nodes und Edges. Diese befinden sich als Zip Datei ebenfalls im Repository.

Die CSV-Dateien müssen ausserhalb des Projektordners gespeichert werden und in der Datei MapData referenziert werden:

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Phase 2: testing existing algorithms

## Greedy Best-First

Hinweis: Hier gingen wir zuerst von einem anderen Algroithmus aus, daher haben wir zu Beginn aus Versehen die Greedy best-First umgesetzt. Im Anschluss haben wir noch BFS und DFS.

Kurze Beschreibung:

* Geht immer in Richtung Ziel nach Luftlinie
* Wählt jeweils den aktuell zielnächsten Ort
* Vermeidet Schleifen durch Merken besuchter Orte
* Stoppt sobald das Ziel erreicht ist und gibt den Weg zurück
* Zählt die betrachteten Orte und kann die Weglänge in Metern berechnen

Erkenntnisse:

* Schnell in der Praxis doch nicht immer der kürzeste Weg

## A\*

Kurze Beschreibung:

* Kombiniert Luftlinie zum Ziel mit bisheriger Weglänge
* Wählt den Weg mit der kleinsten Summe aus beidem
* Vermeidet Schleifen durch Markieren bereits besuchter/abgeschlossener Orte
* Stoppt am Ziel und rekonstruiert den kompletten Weg
* Zählt betrachtete Orte und kann die Weglänge in Metern berechnen

Erkenntnisse:

* Liefert bei realistischer Luftlinien-Schätzung meist den kürzesten Weg
* Gründlicher als Greedy aber meistens etwas mehr Rechenaufwand

## Breadth und Depth Search

Hinweis: Schlussendlich ist uns aufgefallen, dass wir bisher nur eine Greedy BestFirst Suche hatten und keine Breadth und Depth search, diese haben wir dann zum Schluss ebenfalls eingefügt.

### Breadth-First (BFS)

* Geht schrittweise Ebene für Ebene vor
* Findet den kürzesten Weg in Anzahl Kanten
* Vermeidet Schleifen durch Markieren besuchter Orte
* Stoppt am Ziel und gibt den kompletten Weg zurück
* Zählt betrachtete Orte und kann die Weglänge in Metern berechnen

Erkenntnisse:

* Gut, wenn wenige Schritte wichtig sind
* Benötigt oft mehr Speicher, da viele Wege parallel gehalten werden

Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Depth-First (DFS)

* Geht möglichst tief in eine Richtung, dann Backtracking
* Liefert den zuerst gefundenen Weg, nicht garantiert kürzest
* Vermeidet Schleifen durch Markieren und Pfadprüfung
* Stoppt am Ziel und gibt den kompletten Weg zurück
* Zählt betrachtete Orte und kann die Weglänge in Metern berechnen

Erkenntnisse:

* Braucht wenig Speicher, kann aber in Sackgassen abtauchen
* Ergebnis hängt stark von der Nachbarn-Reihenfolge ab
* Nicht geeignet, um minimale Wege sicher zu finden

Unterschiede und Auswirkungen:

* BFS priorisiert Breite und garantiert minimal viele Schritte, kostet mehr Speicher
* DFS priorisiert Tiefe, ist speichersparend, aber riskant für lange Umwege

Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Phase 3: Beam Search

Wir haben uns für die Erstellung der BeamSearch entschieden.

* Begrenzt pro Ebene auf k beste Kandidaten
* Bewertet Wege mit f = g (bisherige Strecke) + h (Luftlinie)
* Hält Suche zielnah wie A\*, aber schlanker und breiter als Greedy

## Wichtige Bausteine:

### Pfaddatensatz inkl. Kosten

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Einstieg + Initialisierung

Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Heuristik (Luftlinie)

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Ablauf der Suche

### Hauptrunde:

expandiert aktuelle Beam-Pfade, generiert Kandidaten, schneidet auf Top-k

### Komfort-Wrapper:

public static List<String> search(MapData mapData, String start, String goal, int beamWidth) { return searchWithStats(mapData, start, goal, beamWidth).path(); }

## Gedanken/Entscheidungen

* Beam-Width steuert Qualität vs. Aufwand:
  + klein = schneller, riskanter;
  + gross = näher an A\*
* f = g + h hält Balance zwischen bisheriger Strecke und Zielnähe
* bestGForNode verhindert schlechtere Duplikate am gleichen Endknoten
* Dedup der Endknoten hält den Beam vielfältig statt redundant
* Frühe Rückgaben bei Zieltreffer sparen Zeit

# Testen

1. CSV ablegen und in MapData Pfad angeben. Ist oben beschrieben.
2. Dann die Test.java File ausführen.

In der Test.java können auch andere Routen getestet werden:

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Die Resultate werden folgendermassen ausgegeben:

Ein Bild, das Text, Screenshot, Schrift, Schwarzweiß enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Ergebnisse

## Von Rothrist/Schellbergweg/1 nach Aarau/Buchsstrasse/1

|  |  |  |  |
| --- | --- | --- | --- |
| Suche | Länge (m) | Zeit (ms) | Besuchte Knoten |
| DFS | 29 751.3 | 0.886 | 41 |
| BFS | 29 751.3 | 0.383 | 36 |
| Greedy Best-First | 21 232.9 | 1.696 | 24 |
| A\* | 21 188.1 | 1.392 | 46 |
| BeamSearch | 29 751.3 | 0.954 | 35 |

### Routenwahl:

* DFS, BFS und Beam nahmen exakt dieselbe lange A1-Route (~29.75 km).
* A\* und Greedy sind via Oberentfelden/Entfelderstrasse und fuhren deutlich kürzer (~21.2 km).

### Bestes Ergebnis:

* A\* war am kürzesten (21 188 m) – noch ca. 45 m kürzer als Greedy.

### Tempo vs. Qualität:

* BFS rechnete am schnellsten (0.291 ms), lieferte aber die lange Route.
* Greedy brauchte am längsten (1.740 ms), kam aber fast an A\* heran.

### Heuristik lohnt sich:

* A\* und Greedy sparen ~8.6 km gegenüber DFS/BFS/Beam.

### BeamSearch:

* Brachte hier keinen Vorteil, gleiche Route und Länge wie DFS/BFS.

### Knoten (besuchte Knoten):

* DFS 41, BFS 36, Greedy 24 (am wenigsten), A\* 46 (am meisten), Beam 35.
* Greedy prüfte am wenigsten Knoten, war aber dennoch langsamer als BFS, die Heuristik führte wohl zu Umordnungen/Neubewertungen. A\* inspizierte am meisten Knoten, fand dafür die beste Route.

Ein Bild, das Screenshot, Text, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Von Rothrist/Neue Aarburgerstrasse/2 nach Aarau/Allmendweg/1

|  |  |  |  |
| --- | --- | --- | --- |
| Suche | Länge (m) | Zeit (ms) | Besuchte Knoten |
| DFS | 30 581.8 | 0.598 | 52 |
| BFS | 20 461.2 | 0.262 | 66 |
| Greedy Best-First | 20 461.2 | 1.442 | 20 |
| A\* | 20 413.0 | 0.581 | 54 |
| BeamSearch | 20 461.2 | 1.289 | 59 |

### Routenwahl:

* DFS nahm die lange A1-Schleife (~30.58 km).
* BFS, Greedy und Beam wichen ab (via Oberentfelden → Entfelderstrasse → Obere/Vordere Vorstadt) und fuhren deutlich kürzer (~20.46 km).
* A\* fuhr fast gleich, bog am Ende aber über Pfrundweg → Allmendweg ab und war minimal kürzer (~20.41 km).

### Bestes Ergebnis:

* A\* war am kürzesten (20 413 m), rund 48 m kürzer als BFS/Greedy/Beam (20 461 m).

### Tempo vs. Qualität:

* BFS rechnete am schnellsten (0.277 ms) und fand hier bereits die kurze Route.
* Greedy war am langsamsten (1.828 ms) und landete bei der Strecke gleichauf mit BFS/Beam.
* A\* lag zeitlich dazwischen (1.009 ms).

### Heuristik lohnt sich:

* Die Abkürzung spart ~10.1 km gegenüber DFS (30 582 m → 20 461 m mit BFS/Greedy/Beam, 20 413 m mit A\*).
* A\* bringt zusätzlich ~48 m weniger Strecke.

### BeamSearch:

* Kein Vorteil gegenüber BFS/Greedy – gleiche Route und Länge, zudem langsamer als BFS (1.067 ms vs. 0.277 ms).

### Knoten (besuchte Knoten):

* DFS 52, BFS 66 (am meisten), Greedy 20 (am wenigsten), A\* 54, Beam 59.
* BFS exploriert breit (viele Knoten), ist aber pro Schritt sehr günstig → schnell trotz hoher Knotenzahl. Greedy braucht extrem wenige Knoten, erreicht aber „nur“ die zweitbeste Strecke. A\* betrachtet mehr Knoten als Greedy, nutzt die zusätzliche Suche aber für den minimalen Feinschliff.

Ein Bild, das Screenshot, Text, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.