Institut für Theoretische Informatik
Lehrstuhl Prof. Dr. D. Wagner

Adrian Feilhauer, Michael Zündorf

Übungsblatt

Praktikum Algorithm Engineering – Routenplanung (WS 23/24)

Ausgabe Mittwoch, den 25. Oktober 2023
Abgabe Dienstag, den 21. November 2023, 23:59 (CET/MEZ)

Deadline & Punkte

Wir geben alle Deadlines hinreichend früh bekannt. Wir bestehen deswegen darauf, dass das Übungsblatt und auch sonstige Materialien rechtzeitig abgegeben werden. Jede Abgabe nach der Deadline wird als nicht abgegeben gewertet!

Wer auf diesem Übungsblatt weniger als 2,5 Millionen Punkte erreicht, ist im Praktikum durchgefallen.

Allgemeines

Mit deinem ITI-Account kannst du dich an den Rechnern im Poolraum anmelden. Alternativ kannst du auch per SSH auf die Rechner zugreifen. Um von Außen ins Lehrstuhlnetzwerk zu kommen, verbinde dich mit `i11ssh.iti.kit.edu`. Auf dieser Maschine soll aber nicht gerechnet werden! Verbinde dich deswegen zu einem der Poolraumrechner weiter. Die Rechner haben die Adressen `i11poolX.iti.kit.edu`, wobei `X` von 1 bis 12 geht. Die Befehle für den SSH-Zugriff sind folgende:

```
ssh username@i11ssh.iti.kit.edu
ssh i11poolX.iti.kit.edu
```

Falls der SSH-Zugang nicht funktioniert, kann es sein, dass der entsprechende Rechner ausgeschaltet wurde. Versuche dann einen anderen. Ehe du Berechnungen durchführst, vergewissere dich bitte per `htop`, dass der Rechner nicht bereits von jemand anderem genutzt wird. Falls doch, wechsle bitte auf einen anderen Rechner.

Dein ITI-Account wird mit einem Standardpasswort angelegt. Bevor du loslegt, ändere bitte das Passwort. Führe dazu `passwd` auf einem der Poolraumrechner aus.

Um Dateien per SSH zu übertragen, kannst du `scp` verwenden:

```
scp von_datei_daheim username@i11ssh.iti.kit.edu:nach_datei_im_poolraum_home
scp username@i11ssh.iti.kit.edu:von_datei_im_poolraum_home nach_datei_daheim
```

Auf `i11ssh` (nicht auf den Poolraumrechnern) unter `/algoDaten/praktikum/graph` liegen diverse für das Praktikum relevante Eingabedateien. Bitte verwende diese nur im Rahmen des Praktikums. Für die Codeverwaltung stellen wir dir ein Git-Repository zur Verfügung. Die Abgabe des Übungsblatts erfolgt auch über das Repository. Zu jeder Aufgabe sollst du den Quellcode deines Programms und die berechneten Lösungen abgeben.

Implementierung

Zur Implementierung der Aufgaben kannst du Rust oder C++ verwenden. Für beide Sprachen stellen wir dir ein kleines Basisframework zur Verfügung, das du für I/O etc. benutzen kannst (s.u.). Die Datei `README.md` im jeweiligen Repository enthält weitere Informationen zur Nutzung.

C++

Der Quellcode soll durch das Ausführen von `./compile.sh` auf einem der Poolraumrechner kompiliert werden können. Auf den Poolraumrechnern ist GCC 10.3 installiert. Dieser unterstützt C++17 vollständig. Neuere C++-Features, die nicht vom Compiler unterstützt werden, sind nicht erlaubt. Das Verwenden externer Bibliotheken ist nicht erlaubt. Die C++-Standardbibliothek gilt nicht als extern.

Rust

Der Quellcode soll durch das Ausführen von `cargo build --all` mit dem aktuellen stabilen Compiler (1.55.0) kompiliert werden können. Auf den Poolraumrechnern ist kein Rust-Compiler vorinstalliert. Du kannst aber für deinen Nutzer lokal `rustup` und damit dann einen aktuellen Compiler installieren.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh # rustup installieren
```

Die Nutzung von nicht stabilen Nightly-Features ist nicht erlaubt. Das Verwenden externer Crates ist nicht erlaubt. Die Rust-Standardbibliothek gilt nicht als extern.

Git

Um auf dein Git-Repository zuzugreifen, führe auf deinem lokalen Rechner folgenden Befehl aus:

```
git clone https://username@i11git.iti.kit.edu/git/Praktika/Routenplanung/Teilnehmer/<username>
```

Dies erstellt ein lokales Verzeichnis `<username>`. Dieses ist zu Beginn leer. Importiere nun mit einem der folgenden Befehle das C++- bzw. Rust-Basisframework:

```
git pull https://username@i11git.iti.kit.edu/git/Praktika/Routenplanung/cpp-base master # C++
git pull https://username@i11git.iti.kit.edu/git/Mitarbeiter/Tim-Zeitz/stud-rust-base master # Rust
```

Jetzt kannst du in deinem lokalen Verzeichnis beliebige Änderungen vornehmen. Um diese Änderungen mit unserem Server zu synchronisieren, musst du sie committen. Markiere dazu zunächst die veränderten oder hinzugefügten Dateien:

```
git add file1.cpp
git add file1.h
```

Anschließend bündelst du die Veränderungen zu einem Commit:

```
git commit -m "Created file1"
```

Dieser Commit existiert zunächst nur bei dir lokal. Um ihn mit dem Server zu synchronisieren, führe den folgenden Befehl aus:

```
git push
```

Weitere Informationen zu Git findest du unter `https://rogerdudler.github.io/git-guide/index.de.html`.

Wichtig: Packe keine Dateien mit mehr als 100MB in ein Git-Repository.

Graphformat

Knoten und Kanten werden durch numerische IDs identifiziert, die von 0 bis $n-1$ bzw. $m-1$ gehen, wobei n die Anzahl an Knoten und m die Anzahl an gerichteten Kanten ist. Wir speichern gewichtete, gerichtete Graphen in einer Adjazenzarray-Darstellung. Ein gewichteter, gewichteter Graph besteht aus den Vektoren `first_out`, `head` und `weight`. Um über die ausgehenden Kanten eines Knoten zu iterieren, kannst du beispielsweise den folgenden C++-Code verwenden (für Rust gibt es in der `README` Beispielcode):

```
vector<unsigned> first_out = load_vector<unsigned>("first_out_file_name");
vector<unsigned> head = load_vector<unsigned>("head_file_name");
vector<unsigned> weight = load_vector<unsigned>("weight_file_name");

unsigned my_node = 42;
for(unsigned out_arc = first_out[my_node]; out_arc < first_out[my_node+1]; ++out_arc){
    cout<< "There is an arc from " << my_node
        << " to " << head[out_arc]
        << " with weight " << weight[out_arc]
        << endl;
}
```

Hinweis: `head` und `weight` haben m Elemente. `first_out` hat $n+1$ Elemente. Das erste Element von `first_out` ist immer 0 und das letzte ist immer m. Für alle Graphen gibt es zwei unterschiedliche Kantengewichte: Reisezeit und Distanz. Desweiteren gibt es für manche Graphen zusätzlich für jeden Knoten die geographische Position. Diese wird als zwei `float`-Vektoren abgespeichert, die für jeden Knoten den Längen- und Breitengrad angeben.

Im Verzeichnis `/algoDaten/praktikum/graph` findest du mehrere Graphen in diesem Format. Manche dienen nur zu Testzwecken, während andere zur Aufgabenbewertung verwendet werden. Die Testgraphen entsprechen ganz grob Stupferich, Karlsruhe & Umgebung, Deutschland & Umgebung und (West-)Europa. Die Aufgabengraphen haben die Größe des Deutschlandgraphen.

Achtung: Der Europagraph könnte zu groß sein für den Hauptspeicher von manchen Poolraumrechnern.

Aufgabe 1: Dijkstras Algorithmus (2 000 Punkte)

Implementiere Dijkstras Algorithmus, um die kürzeste Distanz zwischen zwei Knoten in einem Graph zu berechnen.

Um die Korrektheit deiner Implementierung zu testen, gibt es in den `test`-Unterverzeichnissen der 4 Testgraphen jeweils Testdaten. Diese bestehen aus jeweils 4 Vektoren: `source`, `target`, `travel_time_length` und `geo_distance_length`. Sie können mit `load_vector<unsigned>` geladen werden. Die Daten hängen wie folgt zusammen: An der Stelle `travel_time_length[i]` beziehungsweise an der Stelle `geo_distance_length[i]` steht die Reisezeit bzw. Distanz von Knoten `source[i]` zu `target[i]`. Falls es keinen Pfad gibt, ist der Eintrag `inf_weight`. Wenn deine Implementierung für jede Testanfrage und jeden Graph denselben Wert wie die vorgegebenen Referenzlösungen berechnet, dann ist dein Programm wahrscheinlich korrekt.

Für Aufgabe 1 gibt es in `/algoDaten/praktikum/graph/aufgabe1` einen Graph, dessen `test`-Unterverzeichnis weder eine `travel_time_length`- noch eine `geo_distance_length`-Datei enthält. Es ist deine Aufgabe, diese beiden Dateien zu erstellen. Lege die von dir berechneten Dateien in deinem Git-Repository unter `/abgabe/aufgabe1/travel_time_length` und `/abgabe/aufgabe1/geo_distance_length` ab.

Wir bewerten die abgegebenen Dateien `/abgabe/aufgabe1/travel_time_length` und `/abgabe/aufgabe1/geo_distance_length` durch einen Vergleich mit unserer Musterlösung. Für jede korrekt beantwortete Anfrage gibt es einen Punkt. Es können bei Aufgabe 1 also bis zu 2 000 Punkte erreicht werden, da es 1 000 Anfragen und zwei Kantengewichte gibt. Weicht eine Antwort vom korrekten Wert auch nur leicht ab, dann gibt es für diese Anfrage keine Punkte.

Wichtig: Erzeuge die Dateien mit der `save_vector`-Funktion (C++) oder `Vec::write_to()` (Rust). Für selbst ausgedachte Dateiformate gibt es keine Punkte.

Aufgabe 2–4: Contraction Hierarchies (je 2 000 000 Punkte)

Diese Aufgaben folgen demselben Schema wie die erste: Berechne die `travel_time_length`- und `geo_distance_length`-Vektoren für die restlichen Graphen. Hier gibt es aber jeweils 1 000 000 Anfragen, weshalb Dijkstras Algorithmus für die Berechnung zu langsam ist. Verwende deswegen Contraction Hierarchies (CH), um die Berechnungen zu beschleunigen. Die drei Aufgaben unterscheiden sich darin, welche CH-Teile du implementieren sollst.

- Bei Aufgabe 2 sind die Knotenordnung und der mit Shortcuts augmentierte Graph gegeben. Die Daten liegen in den `travel_time_ch`- und `geo_distance_ch`-Unterverzeichnissen. Du musst in dieser Aufgabe also nur den Anfragealgorithmus implementieren. Die Ordnung ist gegeben als eine Liste von Knoten-IDs in der Reihenfolge, in der sie kontrahiert wurden (und nicht als Rank jedes Knotens). Um die Ranks zu bekommen, musst du diese Permutation invertieren.
- Bei Aufgabe 3 ist der augmentierte Graph nicht mehr gegeben, sondern nur noch die Knotenordnung. Du musst also den augmentierten Graph berechnen. Die Implementierung des Anfragealgorithmus kannst du aus Aufgabe 2 übernehmen.
- Bei Aufgabe 4 gibt es nur noch den Eingabegraphen. Du musst also sowohl den augmentierten Graph als auch die Knotenordnung bestimmen.

Details zum CH-Algorithmus findest du in den Einführungsfolien.

Abgabecheckliste

Bevor du deine Antworten abgibst, vergewissere dich bitte, dass …

- … die Abgabefrist noch nicht verstrichen ist. Verspätete Abgaben werden nicht gewertet.
- … die abgegebenen Dateien 4 000 bzw. 4 000 000 Byte lang sind und dem vorgeschriebenen Format entsprechen. Wir vergeben keine Punkte, wenn wir nicht feststellen können, welche Antwort zu welcher Anfrage passt.
- … die Abgabedateien in deinem Git-Repository an folgender Stelle liegen:

```
/abgabe/aufgabe{1,2,3,4}/{travel_time,geo_distance}_length
```

- ... das Git-Repository den von dir geschriebenen Quellcode enthält.
- ... ein Aufruf von `./compile.sh` oder `cargo build --all` alle Programme übersetzt.
- ... dein Code sich auf einem Poolraumrechner kompilieren lässt.

Schreibe dann eine E-Mail an `michael.zuendorf@kit.edu` mit einer Nachricht, was du wann und wo abgegeben hast.