Institut für Informatik

Lehrstuhl für Technische Informatik
Prof. Dr. Björn Scheuermann

Digitale Systeme 2017

C-Programmierprojekt

1 Aufgabenstellung

Die berühmte Bergsteigerin Junko Tabei hat es geschafft, sie steht auf dem Gipfel des K2. Nun steht der Abstieg bevor. Das Basislager ist zu weit entfernt, um es in einer Tagestour zu erreichen. Sie muss den Abstieg in zwei Etappen bewerkstelligen. Da ihr Zelt kaputt ist und sie noch Proviant für genau eine Nacht dabei hat, muss sie in einer ersten Tagestour eine von etlichen dort befindlichen Schutzhütten ansteuern, dort nächtigen und am zweiten Tag das Basislager erreichen.

Vom Gipfel aus überblickt sie die möglichen Pfade und Schutzhütten und hält einen Moment inne um zu überlegen, welche der Schutzhütten als Zwischenziel in Frage kommen. Deine Aufgabe ist es, Junko zu helfen!

Für diese Aufgabe sind folgende Daten verfügbar:

- Die möglichen begehbaren Wege. Das Wegenetz wird als kantengewichteter, gerichteter, planarer Graph mit N Knoten und M Kanten modelliert. Die Knoten-IDs sind ganze Zahlen in $[0, N)$. Die Kantengewichte sind ebenfalls nicht-negative ganze Zahlen und geben die Zeit an, die die Traversierung einer Kante erfordert.
- Die Knoten-ID von Gipfel und Basislager.
- Die maximale Zeit $d_{\max}$, die sich Junko pro Tag fortbewegen kann.
- Eine Liste von Knoten-IDs, auf denen sich Schutzhütten befinden.

Ihr Programm soll die Menge derjenigen Knoten berechnen, die als Zwischenlager für die Nacht in Frage kommen.

2 Ein- und Ausgabeformate

Ihr Programm soll seine Eingabe von der Standardeingabe lesen. Die Eingabedaten sind ein ASCII-codierter Bytestrom und auf folgende Weise formatiert:

- Die erste Zeile enthält das Tripel <Startknoten,Zielknoten,$d_{\max}$>.
- Darauf folgen M Zeilen, die je eine Kante des Graphen beschreiben. Eine Kante ist gegeben als Tripel <von,nach,Kantengewicht>.
- Darauf folgen K Zeilen, die jeweils die ID eines Knotens mit Schutzhütte enthalten.

Dabei werden folgende Trennzeichen verwendet:

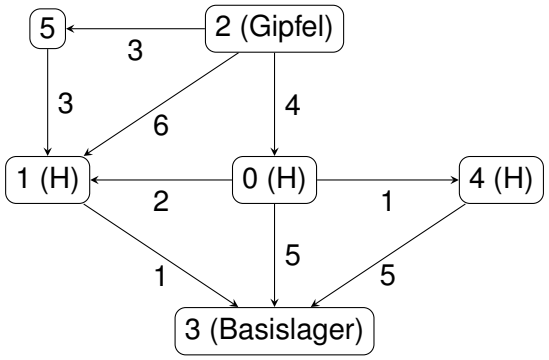
- Die Zahlen innerhalb einer Zeile sind durch einzelne Leerzeichen getrennt.

- Zwei Zeilen sind durch ein Linefeed (`'\n'`) getrennt.
- Die letzte Zeile wird möglicherweise, aber nicht zwingend durch ein Linefeed beendet.

Jede auftretende Ganzzahl ist kleiner als $4 \cdot 10^9$.

Eine mögliche Eingabe sähe also folgendermaßen aus:

```
2 3 5
4 3 5
2 0 4
1 3 1
2 1 6
0 1 2
2 5 3
0 4 1
0 3 5
5 1 3
1
0
4
```



Ihr Programm soll eine Liste von Knoten-IDs aller in Frage kommenden Schutzhütten auf die Standardausgabe schreiben. Diese Liste soll vollständig sein, aber keine Doppelungen enthalten. Die Knoten-IDs in der Ausgabe sollen durch Linefeeds getrennt sein. Die Reihenfolge der Knoten-IDs in der Ausgabe ist irrelevant. Die letzte Ausgabezeile darf durch ein Linefeed abgeschlossen werden. Falls die Menge der in Frage kommenden Hütten leer ist, soll nichts ausgegeben werden, auch kein Leerzeichen oder Linefeed.

Eine mögliche korrekte Ausgabe für die obige Beispieleingabe wäre:

```
4
0
```

3 Beispieldaten

Wir stellen Ihnen auf der Homepage der Lehrveranstaltung einige Beispiel-Eingabedateien zum Testen Ihres Programms zur Verfügung, sowie dazugehörige Ausgaben. Die Beispiel-Eingaben decken viele – aber nicht alle – Grenzfälle ab, mit denen Ihr Programm zurecht kommen muss. Denken Sie beim Vergleich der Ausgabe Ihres Programms mit diesen Lösungen daran, dass die Ausgabe als *Menge* von natürlichen Zahlen übereinstimmen muss, wobei die Reihenfolge unterschiedlich sein darf.

Für die Überprüfung Ihrer Abgabe werden andere Datensätze zum Einsatz kommen.

4 Hinweise zum Algorithmus

Es ist möglich, das Problem folgendermaßen zu lösen: Mit dem Algorithmus von Dijkstra findet man die Distanz vom Gipfel zu jeder Schutzhütte. Für jede Hütte, die weniger als $d_{\max}$ vom Gipfel entfernt ist, berechnet man danach mit einer weiteren Anwendung des Dijkstra-Algorithmus die Distanz bis zum Basislager. Diejenigen Hütten, für die beide Distanzen kleiner sind als die maximale Länge einer Tagesetappe, sind Teil der auszugebenden Menge.

Wenn Sie keine andere Lösung finden, dann können Sie dieses Verfahren implementieren. Es gibt allerdings auch (mindestens) eine sehr viel elegantere Lösung für das Problem, die außerdem auch etwas einfacher umzusetzen ist. Sie dürfen jedes (korrekte) Lösungsverfahren einsetzen, das zwei grundlegende Bedingungen erfüllt:

- Es soll nicht noch ineffizienter sein als das oben skizzierte Verfahren.
- Sie müssen es uns bei der Vorstellung Ihres Programms erläutern und begründen können.

5 Zeitplan

Datum	Beschreibung
23.06.2017	Empfohlene Deadline für eine erste Abgabe; bei Abgabe bis zu diesem Termin können wir im Falle von Problemen mit der Abgabe gewährleisten, dass nach einer ersten Rückmeldung noch Zeit zur Nachbesserung bis zur endgültigen Deadline bleibt
08.09.2017	Spätestmögliche Deadline für die Abgabe der Endversion, die alle Kriterien vollständig erfüllen muss
18.09.2017 – 29.09.2017	In diesem Zeitraum finden die mündlichen Testate in Einzelsitzungen statt. Eine frühere Abnahme ist nach individueller Absprache möglich.

6 Wichtige Hinweise

Beachten Sie bei der Erstellung Ihres Programms **unbedingt** folgende Punkte:

- Kommentieren Sie Ihren Quelltext in angemessenem Umfang.
- Halten Sie sich strikt an das oben definierte Ausgabeformat und die Aufrufkonventionen. Wir überprüfen Ihr Programm automatisiert auf die Korrektheit der ausgegebenen Lösungen. *Eine falsch formatierte Ausgabe wird nicht als korrekt anerkannt!* Im Zweifelsfall fragen Sie *rechtzeitig vor Abgabe* nach!
- Fangen Sie falsch formatierte Eingabedaten mit einer Fehlermeldung ab.
- Verwenden Sie keine Bibliotheken außer der C-Standardbibliothek und ausschließlich POSIX C11-Features.
- Verwenden Sie eine einzige C-Quelltextdatei mit dem Namen `loesung-123456.c`, wobei Sie `123456` durch Ihre Matrikelnummer ersetzen. Ihr Programm muss sich mit folgendem einzelnen GCC-6.3.1-Compileraufruf fehlerfrei übersetzen und linken lassen:
  `gcc -o loesung -O3 -std=c11 -Wall -Werror loesung-123456.c`
  Um Implementierungsfehler zu vermeiden, empfiehlt es sich, zusätzlich die Compiler-Flags `-Wextra` und `-Wpedantic` zu verwenden. Verlangt wird dies aber nicht.
- Ihr Programm muss auf der Kommandozeile ohne grafische Oberfläche lauffähig sein und darf keine Kommandozeilenargument erwarten, d.h. es soll mit folgendem Aufruf ausführbar sein:
  `./loesung < graph.in`
- Achten Sie darauf, dass Ihr Programm auf unterschiedlichen Architekturen lauffähig ist (z. B. mit unterschiedlich langen Integer-Typen).

- Wir stellen auf der Website ein VirtualBox-Image mit installiertem GCC-6.3.1 zur Verfügung. Dieses Image können Sie zum Testen verwenden. Die Korrektur wird ebenfalls anhand dieser Testumgebung erfolgen.
- Stellen sie sicher, dass die von Ihnen abgegebene C-Quelldatei folgende Identifikationszeile enthält:
  `// Matrikelnummer 123456 ; Vorname Nachname <nachnamv@informatik.hu-berlin.de>`
  (Setzen Sie Ihre Matrikelnummer, Ihren vollen Namen sowie Ihre Emailaddresse ein.)
- Legen Sie Ihr Programm so aus, dass es mit beliebig großen Eingabedaten umgehen kann. Hierfür ist es unbedingt notwendig, dass Sie dynamische Speicherverwaltung einsetzen. Wir werden Ihr Programm mit *großen* Datensätzen testen!
- Ihr abgegebenes Programm wird automatisch auf Speicherlecks überprüft. Stellen Sie sicher, dass es keine Speicherlecks aufweist. Dies können Sie beispielsweise mit dem Werkzeug `valgrind` bewerkstelligen. Geben Sie jeglichen dynamisch zugewiesenen Speicher vor Programmende korrekt wieder frei.
- Wir werden alle abgegebenen Lösungen im bereitgestellten VirtualBox-Image benchmarken. Die ressourcensparsamsten Lösungen werden am Ende des Semesters mit einem kleinen Preis ausgezeichnet. Wir werden hierfür die Lösungen in Ausführungszeit sowie Speicherverbrauch sowohl bei großen als auch bei kleinen Graphen untersuchen.

Viel Erfolg!