PAD Projekt

Dokumenation

Roman Rezvan, Branko Ribic, Ines Petrusic

# Spezifikation

## Zielsetzung

Im Rahmen des Themenkreises Nummer 4 wollen wir einen Virus-Tracker implementieren, indem wir zuerst eine Testdatenerstellung kreieren. Diese Testdatenerstellung besitzt eine Personendatenbank, die Personen mit ID, Namen, Kontaktzeit und Kontaktpersonen generiert. Dafür wird eine vorhandene Namensdatenbank verwendet. Dazu kriegt man durch eine Benutzereingabe die Möglichkeit, dass man die erste Person (die infizierte Person, Patient Null) und die letzte Person (das Ziel, welches auch infiziert ist) mit Namen eingibt.

Die Auswertung der Testdaten erfolgt durch einen Algorithmus, der die Kontaktpersonen untereinander auf ID und Zeit prüfen wird. Wenn Beispielsweise Person A infiziert ist und mehr als 20 Minuten mit Person B Kontakt hatte, speichert das Programm Person B als markiert ab und muss für den Benutzer deutlich machen, dass Person B auf einen Virus getestet werden sollte. Nachdem die Person B markiert wurde, wird geprüft mit wem Person B noch über 20 Minuten Kontakt hatte, um die nächste Person zu markieren.

Dieser Vorgang führt sich so lange aus, bis die letzte Verdachtsperson gefunden wurde. Man sollte am Ende den Weg, den der Virus von der ersten bis zur letzten Person (über andere Personen) zurückgelegt hat, nachvollziehen können.

Zum Ende hin möchten wir eine Visualisierung einbringen, damit der Benutzer einen visuellen Überblick über die Gesamtlage haben kann. Die Auswertung des Programms wird durch Blasen, welche Namen beinhalten, bildlich dargestellt. Damit wird gesehen, wie sich der Virus über welche Personen von Patient Null bis zum letzten Infizierten übertragen konnte.

## Risiken

* Es besteht die Möglichkeit, dass es keine verdächtigen Personen finden kann, falls die Personen nur unter der festgelegten Zeit (20 Minuten) Kontakt hatten.
* Die Algorithmen, sowohl A\*-Algorithmus als auch Dijkstra-Algorithmus sind ursprünglich dafür gedacht, dass man den schnellstmöglichen Weg zwischen A und B findet. In unserer Idee muss man diese Prinzipien etwas umschreiben, dass man, zum einen alle Verdachtspersonen mit 20+ Minuten Kontakt zum Infizierten aufnimmt und abspeichert und in der Visualisierung anzeigen kann. Zum anderen, den Weg nachverfolgen kann, bis zur letzten Person, bei der Verdacht besteht. (Start bis Ende)
* Es besteht die Gefahr, dass die Visualisierung bei einer zu hohen Anzahl an Personen unübersichtlich wird.

## Alternativen

Sollten wir bei der Visualisierung nicht mit Graphviz arbeiten können, würden wir ein anderes Tool aufsuchen, um damit die Aufgabe zu bewältigen.

# Team

An sich konnte im Team ganz gut gearbeitet werden. Da die Mitglieder des Teams nicht unbedingt die erfahrensten im Programmieren sind, stellte sich die Aufgabenstellung in manchen Bereichen natürlich doch als etwas herausfordernder heraus als Anfangs gedacht.

## Aufteilung

* Testdatenerstellung: Ines Petrusic
* Auswertung (Dijkstra-Algorithmus): Roman Rezvan
* Graphische Darstellung (Graphviz): Branko Ribic

## Teamtreffen

**12.5.**

* erste Besprechung
* Ideen gesammelt (Virus-Tracker)

**13.5.**

* ausführlichere Besprechung über Zielsetzung
* Schreiben der Spezifikation

**14.5.**

* Spezifikation weiter ausgeführt
* Details hinzugefügt

**20.5.**

* Algorithmen-Wahl (Dijkstra)
* Besprechung über mögliche Umsetzungen der Zielsetzung
* Aufteilung der Aufgabenstellung

**29.5.**

* weitere Details über Umsetzung / Schnittstellen besprochen

**30.5.**

* erste Umsetzungen besprochen

**9.6.**

* Besprechung der bisherigen Implementierungen
* Besprechung über Schnittstelle
* weitere Implementierungen besprochen

# Probleme und Lösungen

Während der Umsetzung unserer Zielsetzung und der Implementierung von Funktionen tauchten folgende Probleme auf:

## Testdatenerstellung

Anfangs stellte sich bei der uns zur Verfügung gestellten Namensdatenbank die Frage, wie man die Namen am Besten entnimmt und dabei Namen, die mit diversen Sonderzeichen (oder Namen, die mit unserem Alphabet nicht richtig dargestellt werden können) weglässt. Schnell stellte sich heraus, dass die Namensdatenbank eine Funktion besitzt, mit der man die Namen eines bestimmten Landes in einer Datei ausgeben kann. Für unsere Implementierungen haben wir uns also nur österreichische Namen in eine csv-Datei ausgeben lassen.

Bei der Erstellung von zufälligen Interkationen zwischen zwei Personen-ID war anfangs geplant, dass statt einer Minutenanzahl eine Start- und Endzeit (time() - seconds since epoch) generiert wird. Allerdings wurde dann doch dagegen entschieden, da wir uns für die weitere Verwendung sowieso die Minutenanzahl zwischen zwei Personen ausrechnen hätten müssen. Außerdem hätte dann noch in Bedacht gezogen müssen, dass eine Person nicht mit mehreren Kontakten gleichzeitig in Kontakt stehen hätte können. Man hätte also weiters implementieren müssen, dass sich die Zeitfenster einer Person sich nicht überlappen. Dieser Aufwand erschien dann doch als eher unnötig, da wie schon erwähnt, für die weitere Verwendung nur die Minutenanzahl zwischen zwei Personen notwendig war. (Die Variante mit Start- und Endzeit wurde zunächst quasi als Stretch Goal beiseitegelegt, dann schließlich aus zeitlichen Gründen verworfen.)

## Auswertung (Dijkstra-Algorithmus)

## Graphische Darstellung (Graphviz)

# Implementierung und Funktionen

## testdaten-library

### int count\_lines(FILE \* input)

Die Funktion bekommt eine Textdatei übergeben und gibt die Zeilenanzahl dieser Datei zurück. (Dabei zählt sie die Zeilenumbrüche ‚\n‘.)

### int id\_name(char \* input, char \* output, int n)

Die Funktion bekommt die Datei (*names.csv*), welche die österreichischen Namen aus der Namensdatenbank enthält, übergeben. Ebenfalls bekommt sie den Namen der Ausgabedatei (*id\_name.csv*) und die Anzahl an Personen, welche generiert werden sollen, übergeben. Die Funktion kopiert also zufällige Namen aus der Namensliste in eine neue Datei und gibt jedem Namen eine ID.

Format: *Person-ID | Name*

Falls alles funktioniert hat, liefert sie die Zahl „0“ zurück, bei einem Fehler beim Öffnen der *names.csv* liefert sie „-1“ zurück und bei einem Fehler beim Erstellen der neuen Datei *id\_name.csv* liefert sie „-2“ zurück.

### int interactions(char \* output, int n, int max\_i, int t)

Diese Funktion generiert Interaktionen in eine neue Ausgabedatei (*interactions.csv*). Sie bekommt also den Namen der neuen Datei, sowie die Personenanzahl, Anzahl der Interaktionen und der Zeitraum, in dem die Interaktionen stattfinden sollen (bzw. die maximale Zeit, die 2 Personen miteinander verbringen können) übergeben. Dabei wird jeder Interaktion auch eine Interaktions-ID zugeteilt.

Format: *Interaktions-ID | Person1-ID | Person2-ID | Minutenanzahl*

Falls alles funktioniert hat, liefert sie die Zahl „0“ zurück, bei einem Fehler beim Erstellen der Datei liefert sie „-1“ zurück.

### int search\_id(char \* input, char search\_name[])

Die Funktion bekommt eine Datei (die Datei, in der IDs und Namen stehen, *id\_name.csv*) und einen Namen (string) übergeben. Sie sucht in der Datei nach dem Namen, und falls gefunden, liefert sie die ID der Person zurück. Sonst liefert sie, wenn es einen Fehler beim Öffnen der Datei gab, „-1“ zurück oder, falls der Name nicht gefunden wurde, „-2“.

### char \*search\_name(char \* input, int search\_id)

Die Funktion bekommt eine Datei (die Datei, in der IDs und Namen stehen, *id\_name.csv*) und eine ID übergeben. Sie sucht in der Datei nach der ID, und falls gefunden, liefert sie den Namen der Person zurück. (Gegenstück zur int *search\_id* Funktion.)

## Auswertung (Dijkstra-Algorithmus)

## Graphische Darstellung (Graphviz)

# Ablauf der main-Funktion

# Testing

# Fazit