Aufgabe 3

33.Bundeswettbewerb Informatik 2014/'15

Der Script-Tim

Inhaltsverzeichnis

1 Problem

Das Problem hierbei besteht darin, Teilnehmer und Silben unabhängig voneinander zu verwalten und dabei die verschiedenen Entscheidungen des Geburtstagskindes zu simulieren („Was-Wäre-Wenn..“).

2 Lösungsidee

Meine Lösungsidee ist, dass eine Funktion das Aufsagen des Satzes simuliert. So lange, bis vom Geburtstagskind eine Entscheidung gefordert wird. Dann ruft sich die Funktion **zwei** mal selbst auf; einmal mit der Entscheidung für eine Silbe, dann mit der für zwei. Der Satz wird dann durch die Funktion dort fortgeführt, wo er angehalten hat, und wieder solange, bis eine Entscheidung gefordert wird. Die weitere Ausführung der „Mutterfunktion“ ist dann nicht mehr notwendig, da ja alle möglichen Entscheidungen durch die aufgerufenen Funktionen simuliert werden. Wird das Geburtstagskind selber getroffen, wird die Schleife selbstverständlich abgebrochen. Da im Kreis immer weniger Personen stehen, kann daraus keine Endlosschleife entstehen.

3 Umsetzung in ein Programm

3.1 *entscheidung*

Die Funktion nenne ich treffender Weise *entscheidung*. Bei jeder Entscheidung gibt es eine Art „Pfad“ der Entscheidungen, er angibt, wann bereits wie entschieden wurde. Dieser Entscheidungspfad wird bei jeder Entscheidung fortgeführt. Am Ende selektieren sich dann diejenigen Pfade heraus, die zu einer Lösung geführt haben. Da es sich hierbei immer um Entscheidungen mit zwei Optionen handelt, kann der Pfad folgend definiert werden:

```
19 vector<bool> pfad
```

false für nur eine, *true* für zwei Silben. Da die Funktion das Ziel hat, die Möglichkeit der letzten Entscheidung „weiter zu spinnen“, werden dafür verschiedene weitere Informationen benötigt, um den Satz dort fortsetzen zu können, wo er angehalten hat

- **personenImKreis**: Anzahl der Teilnehmer, die noch im Kreis stehen von rechts des Geburtstagskindes aus, das Geburtstagskind mitgerechnet (es ist die 1)
- **silbe**: Wo bzw. bei welcher Silbe sind wir stehengeblieben? Nummer der Silbe [1...16]
- **entscheidungZweiSilben**: Die Entscheidung, ob zwei oder eine Silbe gesprochen werden sollen. Diese Entscheidung wird am Anfang der Funktion zum Entscheidungspfad hinzugefügt;

```
25 pfad.push_back(entscheidungZweiSilben);
```

- **platz**: Die Anzahl derjenigen, die bereits zum Buffet gehen durften

Folglich definiere ich die Funktion formal so:

```
24 void entscheidung(vector<bool> pfad, unsigned int personenImKreis, int silbe, bool
       entscheidungZweiSilben, int platz){ ... }
```

Zunächst wird die Entscheidung in *entscheidungZweiSilben* ausgeführt:

```
27 silbe++; //"Normale" Silbe
28 if (entscheidungZweiSilben) silbe++; //Entscheidung ausführen
```

Die Person *person* ist logischer Weise 2, da Person 1 das Geburtstagskind ist und dieses soeben seine Silbe(n) „gesprochen“ hat.

```
30 unsigned int person = 2;
```

Dann wird der Spruch praktisch dort fortgeführt, wo er unterbrochen wurde

```
32 for(; silbe <= 16; silbe++){
33     if (personenImKreis <= 1) return;
34     if (silbe == 16){ // Satz zu Ende?
35         if (person == 1){
36             //Geburtstagskind wurde getroffen
37             loesung temp;
38             temp.platz = platz + 1;
39             temp.pfad = pfad;
40             loesungen.push_back(temp);
41             return;
42         }else{
43             //Irgend jemand anderes wurde getroffen
44             personenImKreis--;
45             platz++;
46             silbe = 0;
47         }
48     }else{
49         if (person == 1){ //Es wird eine Entscheidung verlangt
50             if (personenImKreis >= 16){ //Entscheidungen treffen
51                 entscheidung(pfad, personenImKreis, silbe, true, platz );
52                 entscheidung(pfad, personenImKreis, silbe, false, platz );
53                 return;
54             }
55
56         }
57     }
58     person = (person % personenImKreis)+1; //Neue Person ermitteln
59 }
```

Die *for*-Schleife startet bei dem Punkt, wo sie aufgehört hat: bei *silbe*. Zunächst wird geprüft, ob überhaupt noch jemand im Kreis ist, der dem Geburtstagskind das Essen streitig machen kann. Ist dies nicht der Fall, darf das Geburtstagskind zwar auch zum Buffet gehen, muss sich aber als letzter Teilnehmer mit abgegriffenen Essensresten zufriedengeben, weshalb diese Lösung nicht zu den erfolgreichen Entscheidungspfaden in *loesungen* hinzugefügt und der weiteren Ausführung der Schleife per *return* ein Ende gesetzt wird.*([Z.33])*. Bei jeder Silbe wird überprüft, ob sie die letzte des Satzes war

[Z.34]). Ist die aktuelle Person auch noch das Geburtstagskind, wird der Entscheidungspfad zu den Lösungen in *loesungen* hinzugefügt (*[Z.36-40]*) und die weitere Ausführung der Schleife wird nicht mehr benötigt (*[Z.41]*). Wurde ein anderer Teilnehmer getroffen, nimmt die Anzahl der Teilnehmer im Kreis ab (er 'verlässt' den Kreis) (*[Z.44]*), der Platz, den das Geburtstagskind minimal erreichen kann, wird um einen angehoben (*[Z.45]*) und der Satz wird wieder von vorne angefangen (*[Z.46]*; *silbe* wird wieder auf 0 gesetzt).

Ist das Ende des Satzes noch nicht erreicht, kann es trotzdem sein, dass vom Geburtstagskind eine Entscheidung erwartet wird. Die Funktion *entscheidung* ruft sich dann zweimal auf: Erst mit Ein-, dann mit Zwei-Silben-Entscheidung (*[Z.51f]*). Danach ist es nicht weiter notwendig, die Schleife fortzuführen (*[Z.53]*);

Nach jeder Silbe wird der nächste Teilnehmer ermittelt; hierbei wird der *modulo*-Operator(%) verwendet, da es sich ja um einen Kreis handelt und deshalb ggf. vom Ende zum Anfang gewechselt werden muss (*[Z.58]*).

3.2 Ein Behältnis für die Lösungen

Zur Speicherung der ermittelten Lösungen wird ein globaler Container einer eigens definierten Struktur definiert:

```
17 struct loesung{
18     int platz;
19     vector<bool> pfad;//Entscheidungspfad
20 };
21
22 vector<loesung> loesungen;
```

3.3 Eingabe der Teilnehmerzahl

Das Programm benötigt bevor es den Lösungspfad ermitteln kann die Anzahl der Teilnehmer im Kreis. Hierbei habe ich folgende Möglichkeiten eingeräumt:

1. **Kommandozeilenparameter**: Da es sich bei diesem Programm um eine Kommandozeilenanwendung handelt, liegt nichts näher, als diese Information über die Kommandozeilenparameter zu übergeben. Verwenden Sie einfach

   ```
   Aufgabe3.exe <Teilnehmeranzahl>
   ```

   Dies ist besonders nützlich, wenn Sie die Ausgabe in eine Datei umleiten wollen.

2. **Eingabe im Programm**: Sind keine oder ungültige Kommandozeilenparameter übergeben worden, erfolgt die Eingabe durch den Benutzer innerhalb des Programmes.(Fallback)

```
74 unsigned int personenImKreis = 0;
75 bool kommandozeilenparameterOK = false;
76
```

```
77  if (argc == 2){
78      stringstream sstream;
79      sstream.clear();
80      sstream << argv[1];
81
82      sstream >> personenImKreis;
83      if (personenImKreis > 0) kommandozeilenparameterOK = true;
84  }
85
86  if (kommandozeilenparameterOK == true){
87      cout << "\nKommandozeilenparameter uebergeben: " << personenImKreis << " Personen
            ";
88  }else{
89      cout << "\nKeine oder ungueltige Kommandozeilenparameter uebergeben. Manuelle
            Eingabe:";
90      do{
91          cout << "\nAnzahl der Personen im Kreis:";
92          cin >> personenImKreis;
93      }while(personenImKreis <= 0);
94  }
```

3.4 Initialisierung

Die Initialisierung der *entscheidung*-Funktion erfolgt folgendermaßen:

```
96  vector<bool> pfad; //Leeren Pfad erstellen
97  cout << "\n\nVorgang gestartet, bitte warten :P ...";
98
99  entscheidung(pfad, personenImKreis, 0, false, 0 );
100 entscheidung(pfad, personenImKreis, 0, true, 0 );
```

3.5 Ermittlung und Ausgabe des besten Entscheidungspfades

Alle Entscheidungspfade, die irgendwie zum Buffet führen, sind ungeordnet in *loesungen* gespeichert. Der im Sinne der Aufgabe beste Entscheidungspfad wird nun ermittelt und ausgegeben.:

```
102 //Ausgabe der Anzahl aller ermittelten Loesungen
103 cout << "\nErgebnisse: " << loesungen.size() << "\n\n";
104
105 if (loesungen.size() == 0){
106         cout << "\n\nEs wurden leider keine Loesungen gefunden."
107         <<  "Das Geburtstagskind ist spaetestens " << personenImKreis << ".";
108     }else{
109         int besterPlatz = personenImKreis;
110
111         for(unsigned int loesung = 0; loesung < loesungen.size(); loesung++){
112             if (loesungen[loesung].platz < besterPlatz){ besterPlatz = loesungen[
                    loesung].platz; }
113         }
114
```

```
115         cout << "Bester Platz: " << besterPlatz;
116
117         for(unsigned int loesung = 0; loesung < loesungen.size(); loesung++){
118             if (loesungen[loesung].platz == besterPlatz){
119                 cout << "\n\n----------\n";
120                 for(unsigned int schritt = 0; schritt <
121                   loesungen[loesung].pfad.size(); schritt++){
122                     cout << "\n" << schritt +1 << ":"
123                      << ((loesungen[loesung].pfad[schritt]) ? "2 Silben" : "1 Silbe");
124                 }
125             }
126         }
127 }
```

4 Beispiele

Hier einige Beispiele

Teilnehmerzahl	**Bester Platz**	**Schritte (u.A.)**
20	8	1,2,2,2,1
21	10	2,1,2,1,2,2
22	5	2,1,1,2
23	4	1,2,1
24	12	1,2,1,1,1,1,2,1
25	3	1,2
26	7	1,2,1,1,1
27	6	2,1,1,1
28	9	1,2,2,2,2,2
29	5	1,2,1
30	8	1,1,1,1,1
31	7	1,2,2,2
32	2	2
33	2	1
34	4	2,1
35	6	2,2,1

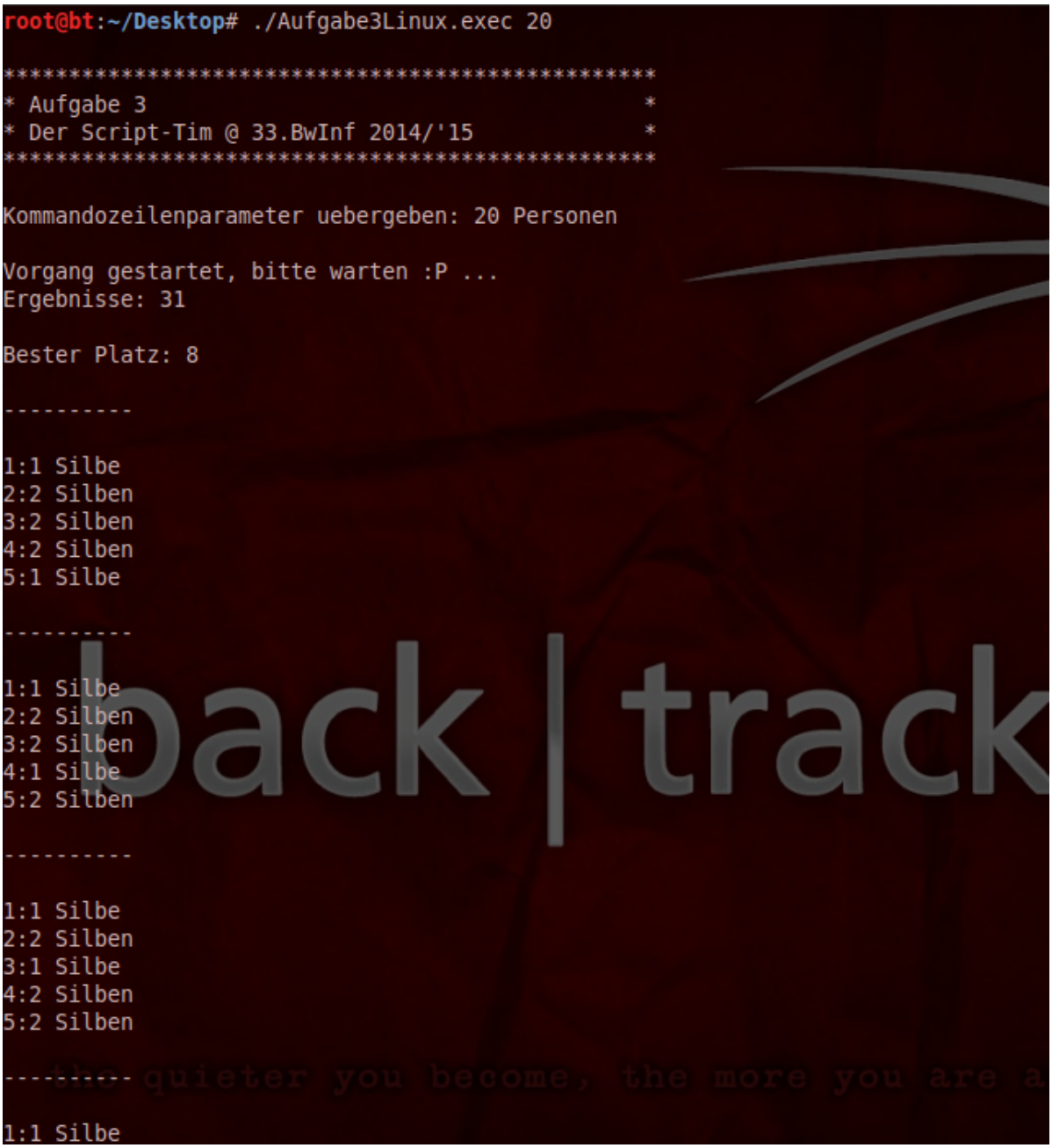

Abbildung 1: Anwendung auf 20 Teilnehmer

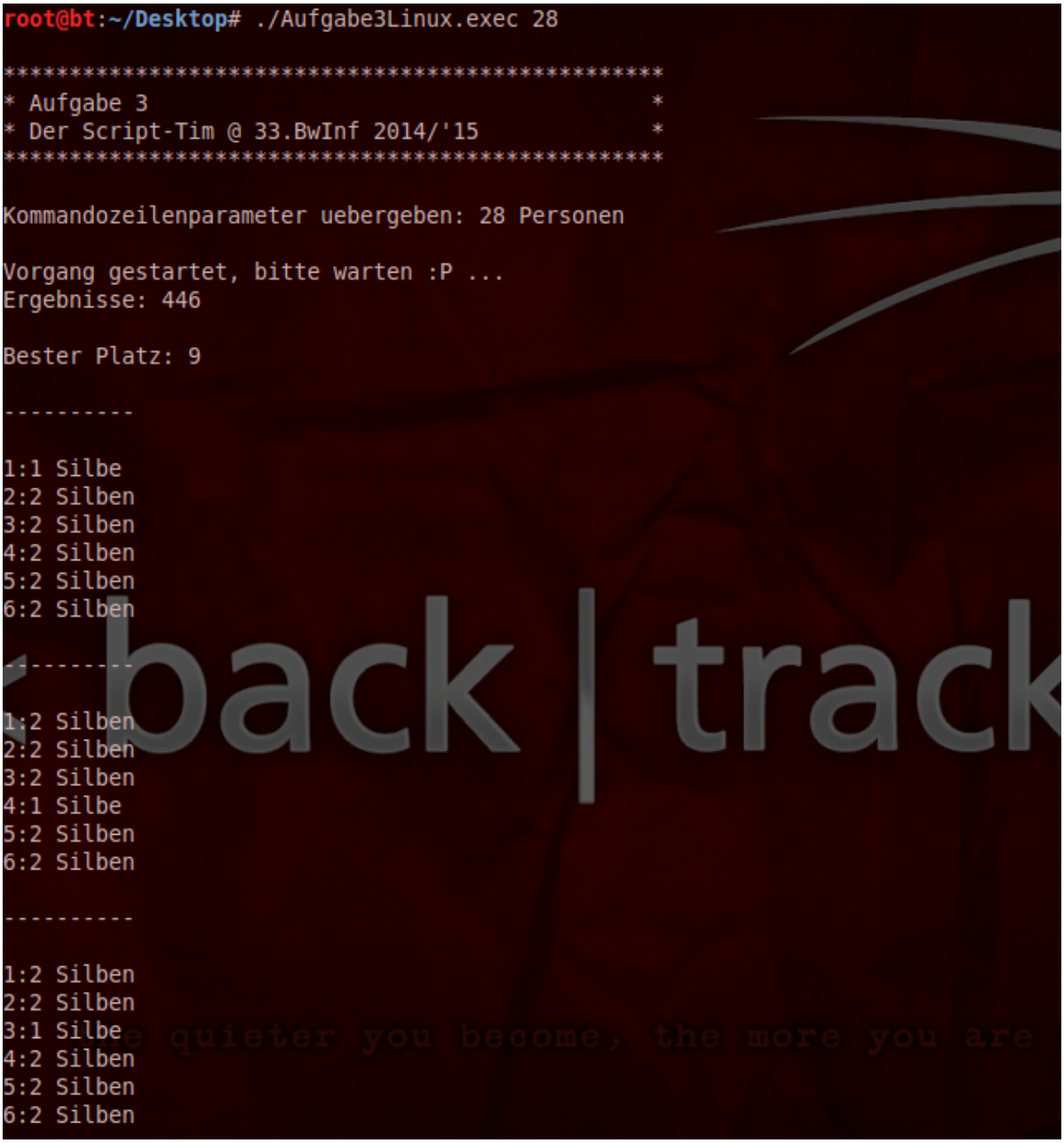

Abbildung 2: Anwendung auf 28 Teilnehmer

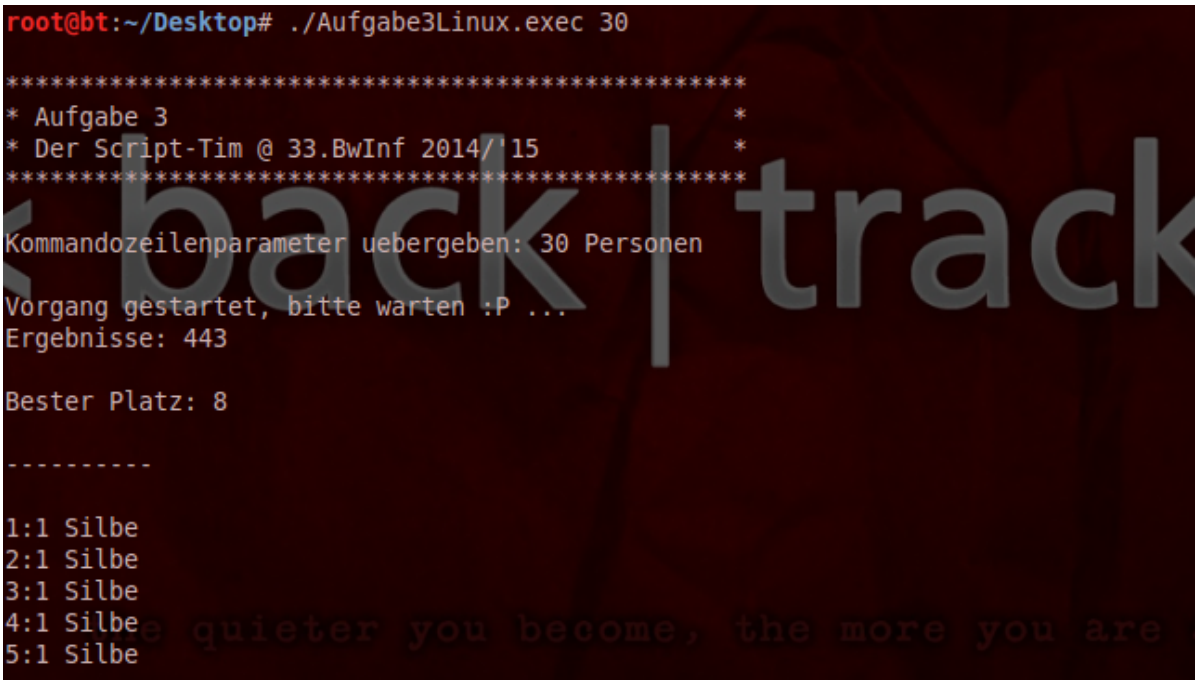

Abbildung 3: Anwendung auf 30 Teilnehmer