Seminar: Bottom-up Datenverarbeitung auf der Unix Kommandozeile auch bekannt als Kommando Bimberle
Prozessbingo: Lottozahlen und Prozess-IDs

Alexander Vogelgsang, Manuel Marschsll, Richard Sailer

Lehrprofessur für Informatik, Universität Augsburg

30.09.2014

Abstract.

Wir zeigen anhand des Fallbeispiels "Prozessbingo: Lottozahlen und Prozess-ID's" wie man mithilfe der **BASH**, Textdaten aus einer Website lesen und strukturiert weiterverarbeiten kann. Außerdem werden wir aus Prozessnummern die maximal mögliche Menge an Enthropie gewinnen sowie daraus ein Sechstupel an Lottozahlen und eine Superzahl generieren.

Inhaltsverzeichnis

1 Definitionen

Im Folgenden referenziert "Lottoziehung" ein 6 - Tupel $x = (x_1, \ldots, x_6)$ mit folgenden Eigenschaften:

1. $\forall x_i, x_j \in x : x_i \neq x_j$ (In einer Ziehung kommt keine Zahl zwei Mal vor)
2. $\forall x_i, x_j \in x : i < j \Rightarrow x_i, x_j$ (Die Tupel mit denen wir arbeiten sind sortiert)

2 Extraktion der Lottozahlen aus dem Archiv

2.1 Erste Überlegungen

Um bestimmen zu können ob man jemals im Lotto gewonnen hätte, müssten die Lottozahlen erst in einem geeigneten Format zur Weiterverarbeitung vorliegen. Dazu wählen wir eine passende Internetseite. Wir entscheiden uns in diesem Fall für die Seite von ,,Lottozahlenonline", weil die dort vorgefundenen Daten gut strukturiert sind und in Folge dessen einfach ausgelesen und die Jahre der Lottoziehungen durch einfache Anpassung der URL durchgeschaltet werden können.

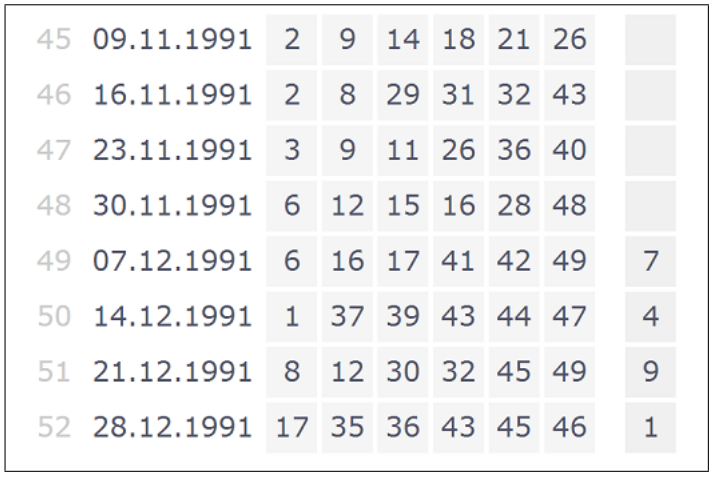
2.2 Arbeitsschritte

Für jedes Jahr werden für die jeweilige Seite folgende Arbeitsschritte ausgeführt:

1. Herunterladen der Seite in eine temporäre Datei
2. Vorformatierung des HTML-Codes
3. Entfernen des unnötigen HTML-Codes
4. Extrahieren der Lottozahlen und des Datums in eine Tabelle
5. Nachformatierungen der extrahierten Daten und Speichern in temporäre Datei

Ist dieser Ablauf für jede Seite ausgeführt, werden alle Daten in einer Tabelle zusammengefasst. Die resultierende Datei liegt nun in dem schließlich gewünschten Format vor und es können relationale Anfragen darauf ausgeführt werden.

Wir nutzen nun allerdings das Tool **wget**, um die Webseiten aus dem Internet herunterzuladen, da dieses verbreiteter zu sein scheint. Dazu bekommt



45	09.11.1991	2	9	14	18	21	26	
46	16.11.1991	2	8	29	31	32	43	
47	23.11.1991	3	9	11	26	36	40	
48	30.11.1991	6	12	15	16	28	48	
49	07.12.1991	6	16	17	41	42	49	7
50	14.12.1991	1	37	39	43	44	47	4
51	21.12.1991	8	12	30	32	45	49	9
52	28.12.1991	17	35	36	43	45	46	1

Abbildung 1: Aufbau der Internetseite. Man sieht folgende Daten (von links nach rechts): Nummer der Ziehung im Jahr, Datum der Ziehung und die dazugehörigen sechs Gewinnzahlen sowie Superzahl (falls vorhanden).

wget die URL der Seite, gefolgt von der jeweiligen Jahreszahl der Lottoziehung. Die Ausgabe wird in eine Datei mit „$Jahreszahl.html" geschrieben.

Um die spätere Verarbeitung zu vereinfachen, löschen wir Escape-Sequenzen aus dem Stream. Diese würden nämlich eine fehlerhafte Ausgabe hervorrufen und ihr Auftreten ist inkonsistent. Dazu nutzen wir **tr**, das einzelne Zeichen löscht oder ersetzt. Dadurch ist es für diese Aufgabe perfekt geeignet.

Nun entfernen wir denjenigen Teil des HTML-Codes, der keine Beziehung zu den Lottozahlen aufweist. Dazu gehört alles bis auf die in Abbildung 2 gezeigten Abschnitte. Wir benutzen dazu das Programm **grep**. **grep** dient dazu, mittels regulären Ausdrücken Text zu finden und die komplette Zeile des Textes auszugeben. Mit dem Parameter –o wird jedoch nur die gefundene Textpassage ausgegeben. Den verwendeten regulären Ausdruck zeigt Abbildung 3.

Um die Daten tatsächlich in tabellarischer Form zu erhalten, löschen wir mithilfe des Programms **sed** den HTML-Teil des verbleibenden Codes. Übrig bleiben die durch Leerzeichen getrennten Lottozahlen inklusive Datum und Ziehungsnummer. Dies erreichen wir durch den Substitutionsbefehl von sed. Mit diesem ersetzen wir die inneren divs durch Leerzeichen sowie das erste und letzte durch nichts. Das Ergebnis dieser Ausgabe zeigt Abbildung 3.

Schließlich fügen wir einen Stellvertreterwert für alle Ziehungen hinzu, in denen keine Superzahl gezogen wurde. Dies geschieht aus Konsistenzgründen.

```
...
<div class="zahlensuche_nr">47</div><div class="
zahlensuche_datum">23.11.1991</div><div class="
zahlensuche_zahl">3</div><div class="
zahlensuche_zahl">9</div><div class="
zahlensuche_zahl">11</div><div class="
zahlensuche_zahl">26</div><div class="
zahlensuche_zahl">36</div><div class="
zahlensuche_zahl">40</div><div class="
zahlensuche_zz"></div>
...
```

Abbildung 2: Page-Code

```
div class="zahlensuche_nr">[0-9]+</div>
<div class="zahlensuche_datum"[^<]*</div>
(<div class="zahlensuche_zahl">[^<]*</div>){6}
<div class="zahlensuche_zz">[^<]*</div>
```

Abbildung 3: Verwendeter regulärer Ausdruck. Dieser besteht aus vier Teilen. Der erste steht für die Ziehung, der zweite für das Datum der Ziehung, der dritte für die sechs Lottozahlen und der vierte für die Superzahl. Jeder dieser Teile hat die Form: ??? . Der Inhalt der ,,divs" stellt die zu extrahierenden Daten dar.

```
...
46 16.11.1991 2 8 29 31 32 43
47 23.11.1991 3 9 11 26 36 40
48 30.11.1991 6 12 15 16 28 48
49 07.12.1991 6 16 17 41 42 49 7
50 14.12.1991 1 37 39 43 44 47 4
51 21.12.1991 8 12 30 32 45 49 9
52 28.12.1991 17 35 36 43 45 46 1
```

Abbildung 4: Resultierende Tabelle. Man sieht die Daten liegen wie in Abbildung 2 vor, aber im Textformat.

2.3 Anpassungen

Im endgültigen Programm vergleichen wir Lottozahlen zum einen mit der Superzahl, zum anderen ohne Superzahl mit unseren generierten Lottozahlen. Dazu erstellen wir zwei verschiedene Tabellen, einmal ohne Zusatz und einmal mit. Hierfür ist unser Stellvertreterwert sehr hilfreich, weil man mit ihm alle Ziehungen auswählen kann, die keine Superzahl beinhalten. Hierfür nutzen wir eine Kombination aus **grep** und **cut**. **grep** extrahiert alle Zahlen mit beziehungsweise ohne Superzahl und **cut** entfernt unnötige Spalten. In unserem Fall sind das die Ziehungsnummer und eventuell die Superzahl. Anschließend werden die Ergebnisse in zwei unterschiedliche Textdateien exportiert.

3 Die Generierung der Lottozahlen aus den Prozess-IDs

Die Menge der Prozess-IDs (im Folgenden PIDs) ist eine Menge $\mathbb{P}$ aus natürlichen Zahlen.

Die Spezifikationen für das generate_pid_lottotips.sh Skript sind also aus der gegebenen Instanz von $\mathbb{P}$, ein 6 - Tupel x zu erzeugen, sodass die oben genannten Bedingungen erfüllt sind und außerdem noch eine weitere Superzahl zu generieren. Des Weiteren haben wir uns das Ziel gesetzt, dies in einer Art zu tun, die:

1. die komplette in $\mathbb{P}$ enthaltene Enthropie nutzt.
2. zuverlässig (unabhängig von $|\mathbb{P}|$, somit der Anzahl der Prozesse auf dem System zum aktuellen Zeitpunkt) funktioniert.
3. performant ist; (insofern dies mit einem **Shellskript** möglich ist).

Um Vergleichszahlen zu erhalten, generieren wir unsere eigenen Lottozahlen anhand der Prozess-ID's. generate_pid_lottotips.sh erfüllt die Spezifikation.

3.1 Das Programm (generate_pid_lottotips.sh)

Wir simulieren das normale Vorgehen einer Lottoziehung aus einer Trommel, aufgrund dessen diese Mthode programmiertechnisch gesehen die am einfachsten zu implementierende Lösung ist, die doppelte Lottozahlen verhindert und damit Definition 1 genügt. Wir generieren uns ein Array **lotto_balls**, der die Trommel darstellt. Aus diesem ziehen wir zufällig und hintereinander sechs Zahlen. Der Zufall wird aus den **PID's** berechnet.

Zuerst nehmen wir alle **PID's**,wobei uns das **ps** Kommando hilft, und konkatenieren sie zu einem großen String. Anschließend hashen wir diesen String mit der Hashfunktion **sha512**. Nun werden alle Kleinbuchstaben a - f

in Großbuchstaben A - F umgewandelt. Dies dient einer erleichterten Weiterverarbeitung. Als Nächstes wandeln wir diese Hexadezimalzahl mit dem Kommandozeilenrechner **bc** in eine dezimale Zahl um. Für **bc** haben wir uns entschieden, weil dieses Werkzeug hervorragend mit extrem großen Zahl umgehen kann, da es die GNU BigNumberLibrary verwendet, im Gegensatz zu der in der BASH eingebauten **Arithmetic Evaluation**.

Gesetzt wurde die Variable ibase, um bc zu sagen welche Basis die über STDIN erhaltenen Zahlen haben. Die Verwendung der Variablen ibase und obase ist ein unter UNIX Nutzern beliebter Trick zur schnellen Umwandlung zwischen beliebigen Zahlensystemen auf der Kommandozeile. Da die Ausgabe Basis das Dezimalsystem ist (default) muss obase hier nicht angepasst werden.

Abbildung 5 zeigt welche Bälle später aus dem Array **lotto_balls** gezogen werden.

```
ball_to_pick_1=$(( (10#${hashed pids:0:2 } % 49) + 1 ))
ball_to_pick_2=$(( (10#${hashed pids:2:2 } % 48) + 1 ))
ball_to_pick_3=$(( (10#${hashed pids:4:2 } % 47) + 1 ))
ball_to_pick_4=$(( (10#${hashed pids:6:2 } % 46) + 1 ))
ball_to_pick_5=$(( (10#${hashed pids:8:2 } % 45) + 1 ))
ball_to_pick_6=$(( (10#${hashed pids:10:2} % 44) + 1 ))
```

Abbildung 5: Hier benutzen wir die **Arithmetic Evaluation** und die Stringbearbeitungsfeatures der **BASH**. $(var:a:b) selektiert b-Zeichen aus der Stringvariable var von der Stelle a ab. $(()) führt eine **Arithmetic Evaluation Enviroment** ein. Deswegen funktionieren die Rechenoperationen % und +. Das Sigel 10# vor dem $-Sigel ist notwendig, um zu verhindern, dass die **BASH** Zahlen, die mit einer führenden 0 beginnen, als Oktalzahl interpretiert werden und das Programm damit bei Zahlen wie 08 und 09 Fehlermeldungen ausgibt und damit terminiert.

Da **lotto_balls** mit jeder Ziehung um eine Zelle kleiner wird, ist auch die obere Schranke für die Indizes (der Divisor unserer Modulooperation) **ball_to_pick** für jede Ziehung um 1 kleiner gewählt.

Aus Performance Gründen haben wir uns hier, genause wie bei den String Operationen der vorherigen Schritte, dafür entschieden, möglichst viel über die bash internen Operatoren (bzw. hier die Arithmetik Evaluation Erweiterung der **BASH**) zu arbeiten. Dies ist gegenüber dem Aufruf von externen Programmen, die extra von der Festplatte oder aus einem Cache geladen werden müssen deutlich effizienter.

Die Extraktion der Superzahl aus dem hashed **PID** String verläuft auf ähnliche Art und Weise.

Im Folgenden nehmen wir aus der Tombola, simuliert durch das Array **lotto_balls**, sechs Zahlen.

Dies funktioniert genauso wie bei einer echten Tombola. Nachdem ein Element gezogen wurde, wird es aus dem Array entfernt.

Zuletzt werden die Lottozahlen in einen String überführt, der diese in sortierter Reihenfolge enthält (die Sortierung wird über GNU Sort mit Parameter -n erreicht).

Dies ist notwendig um die Suche nach einem Treffer in den Archivlottozahlen effizient und performant umzusetzen, da die Lottozahlen dort auch in sortierter Reihenfolge vorliegen.

4 Überprüfung auf Gewinnkombinationen und eine Benutzerschnittstelle

Wir haben nun sowohl die Archivzahlen als auch unsere im vorherigen Abschnitt erstellte Lottoziehung $\mathfrak{L}$.

In der Datei **mainbusinesslogic.sh** suchen wir mithilfe von grep in den Archivlottozahlen nach $\mathfrak{L}$.

Eine erfolgreiche Suche bedeutet, dass unser $\mathfrak{L}$ in den Archivlottozahlen vorkommt und wir mit $\mathfrak{L}$ im Lotto gewonnen hätten.

Der Zeitpunkt dieses spekulativen Lottogewinns wird zusammen mit anderen wertvollen Informationen über die Whiptail Benutzerschnittstelle in einer ansprechenden modernen und zeitgemäßen Form ausgegeben.

5 Fazit

In diesem Seminar haben wir die **BASH** und die Kommandozeilentools kennengelernt und daher vor allem wie man mit ihnen Daten, insbesondere Text in strukturierter Form, verarbeitet. Insgesamt sind wir zu dem Schluss gekommen, dass die BASH mit dazugehörigen POSIX-Programmen für diesen Zweck nur mittelmäßig gut geeignet sind. **Perl** wurde für genau diesen Zweck geschaffen und erlaubt für den Programmierer deutlich angenehmere und effizientere Möglichkeiten der Datenverarbeitung. Die Datenstrukturen und Datentypen in **BASH** sind wenig umfangreich und in Hinblick auf ihre Möglichkeiten nur eingeschränkt nutzbar. So können Funktionen zum Beispiel keine Arrays zurückgeben. Außerdem integriert der Array-Datentyp nur auf sehr plumpe Art und Weise mit externen Programmen und Build-In Methoden. Oftmals muss mit einer for-Schleife über alle Elemente interiert werden, bei Tätigkeiten, für die **Perl** sehr viel angenehmer zu benutzende Build-In Methoden bietet. Außerdem ist die Syntax schwierig zu durchschauen, da sie sehr vom Kontext abhängig ist. So ist es im Gegensatz zu vielen anderen Sprachen durchaus wichtig, an welcher Stelle man ein Leerzeichen setzt und an welcher nicht. Abschließend lässt sich sagen, dass es sicherlich einige Anwendungsfälle gibt, für die Shell-Skript gut geeignet ist, wie zum Beispiel Automatisierung von Administrationsaufgaben oder

das Arbeiten mit Dateien, komplexere Textverarbeitungsaufgaben gehören jedoch nicht dazu. Wir hatten viel Spaß am Kommando Bimberle xD.