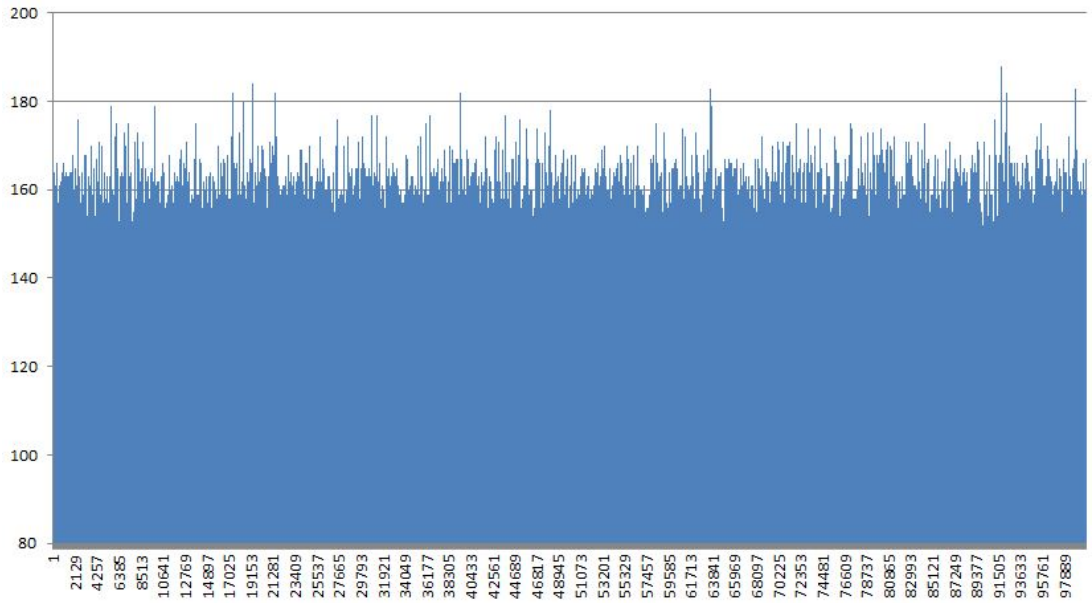
1 Lösungsbeschreibung

1. Es wird die gesamte Population von 66 Generationen à 100′000 Personen generiert.
2. Die Verwandschaftsbeziehungen werden festgelegt: Zu jeder beliebigen Person aus Generation p werden zwei Elternteile aus Generation $p-1$ ermittelt. Die Wahrscheinlichkeit für ein Individuum aus p, ein bestimmtes Individuum aus $p-1$ als Elternteil zu haben, richtet sich nach der Anzahl Personen, also ist sie pro Wahl[1] für ein Mitglied der Generation $p-1$ genau $\frac{1}{100000}$.
3. Von Generation p, beginnend mit 0, wird von jedem Individuum nach Nachkommen gesucht. Dies beginnt in $p+1$, wobei alle Nachkommen in einer Liste gespeichert werden. Von den resultierenden Personen wird wiederum nach deren Nachkommen gesucht. Dies wiederholt sich bis in die 66. Generation ($p = 65$).

2 Resultate

Die Resultate erstaunen. Folgende Kennwerte wurden ermittelt:

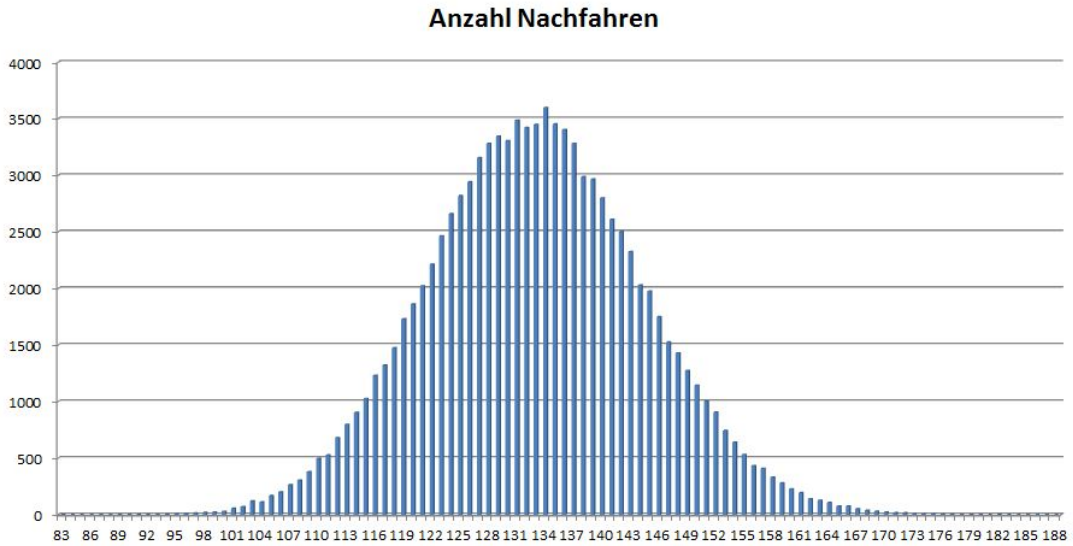
- Der **Median** von Nachkommen pro Person aus Generation 134.
- Der **Mittelwert** von Nachkommen pro Person aus Generation 0 beträgt 132.9.
- Der **Höchstwert** lag bei 188 Nachkommen.
- Der **Mindestwert** lag bei 83 Nachkommen.



Genauer könnte man die Verteilungsfunktion so beschreiben[2], dass sie eine Normalverteilung mit folgenden Parametern ist:

- Standardabweichung $\sigma = 10$
- Dichte $\mu = 133$

[1] Es findet insgesamt 2 Mal pro Individuum aus p eine Wahl statt, eine für den Vater, eine für die Mutter.
[2] mit der Einschränkung, dass die Berechnung nur auf einer einmaligen Ausführung des Programm beruht



2.1 Spezialfall: Keine Nachkommen

Für den Fall, dass jemand keine Nachkommen haben sollte, lässt sich die Wahrscheinlichkeit berechnen:

$$p = \frac{e^{\frac{-1}{200}(-133)^2}}{10\sqrt{2\pi}} \tag{1}$$

$$\Rightarrow p = \frac{1}{10e^{\frac{17689}{200}}\sqrt{2\pi}} \tag{2}$$

$$p \approx 1.54787 \cdot 10^{-40} \tag{3}$$

$$\approx 1 : 6'460'000'000'000'000'000'000'000'000'000'000'000'000 \tag{4}$$

Es ist also extrem unwahrscheinlich, dass jemand keine/n oder nur eine/n[3] Nachfahren hat.

3 Anhang: Java-Klassen

Im Anhang finden sich 3 Java-Klassen:

- **Main.java** - Das ausführbare Hauptprogramm, inkl. Tracing[4]-Algorithmus.
- **Generation.java** - repräsentiert eine Generation, welche mit Personen "bevölkert" wird.
- **Person.java** - repräsentiert eine Person mit festgelegten Attributen für die Elternteile.

[3] 1 Nachfahre: $p \approx 5.82337 \cdot 10^{-40}$
[4] Zur Funktionsweise: Siehe Funktion *trace()*

```java
package ancestor;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

public class Main {

	public static int GENERATIONS = 66;
	public static int POPULATION = 100000;
	public static double avg;
	public static int desc;
	public static int nodesc;

	static List<Generation> era = new ArrayList<Generation>();
	static List<Integer> generationZero = new ArrayList<Integer>();

	public static void main(String[] args) {
		System.out.print("Populating...");
		populate();
		System.out.print(" done.\n");

		System.out.print("Tracing children...");
		trace();
		System.out.print(" done.\n");

		System.out.print("Calculating results...");
		calculate();
		System.out.print(" done.\n\n");

		System.out.println("Average of descendants: " + avg);
		System.out.println("Ancestors with most descendant: " + desc);
		System.out.println("Number of individuals with no descendants: " + nodesc);

		FileWriter fstream = null;
		try {fstream = new FileWriter("out.txt");
		} catch (IOException e1) {e1.printStackTrace();}

		BufferedWriter out = new BufferedWriter(fstream);
		try {
			for (int i = 0; i < generationZero.size(); i++) {
				out.write(String.valueOf(generationZero.get(i))+"\n");
			}
		} catch (IOException e) {e.printStackTrace();}
		try {out.close();} catch (IOException e) {e.printStackTrace();}

	}

	private static void trace() {
		for (int i = 0; i < Main.POPULATION; i++) {
			generationZero.add(0);
		}

		//Walk through existing population (id=65)
		for (int i = 0; i < Main.POPULATION; i++) {
			System.out.println("Individuum "+i);
			int gen = 0;
			Generation nextGeneration = era.get(gen);

			Set<Integer> current = new HashSet<Integer>();
			Set<Integer> children = new HashSet<Integer>();

			current.add(i);

			for (int j = 0; j < Main.GENERATIONS; j++) {
				nextGeneration = era.get(j);
```



```java
				//search for children
				for (int k = 0; k< Main.POPULATION; k++) {
					if (i==nextGeneration.getDad(k) || i==nextGeneration.getMom(k))
						children.add(k);
				}

				//next generation
				current.addAll(children);
				children.clear();
			}
			generationZero.set(i, current.size());
		}
	}

	private static void calculate() {
		for (int i = 0; i < generationZero.size(); i++) {
			int nrOfDescendants = generationZero.get(i);

			// Who has no descendants?
			if (nrOfDescendants==0) nodesc++;

			// Who has the most?
			if (nrOfDescendants > desc) desc = nrOfDescendants;

			avg+=nrOfDescendants;
		}
		avg /= generationZero.size();
	}

	private static void populate() {
		for (int i = 0; i < Main.GENERATIONS; i++) {
			era.add(new Generation(i));
		}
	}
}
```

```java
package ancestor;

import java.util.ArrayList;
import java.util.List;

public class Generation {

	private int id;
	List<Person> generation;

	public Generation(int id) {
		populate();
		this.id = id;
	}

	public void populate() {
		generation = new ArrayList<Person>();

		for (int i=0; i < Main.POPULATION; i++) {
			int m = (int)(Math.random() * Main.POPULATION);
			int d = (int)(Math.random() * Main.POPULATION);
			generation.add(new Person(m, d));
		}
	}

	public int getID() {
		return id;
	}

	public int getDad(int id) {
		return generation.get(id).getDad();
	}

	public int getMom(int id) {
		return generation.get(id).getMom();
	}
}
```

```java
package ancestor;

public class Person {

	private int mom;
	private int dad;

	public Person(int mum, int dad) {
		this.mom = mum;
		this.dad = dad;
	}

	public void setMom(int newMom) {
		this.mom = newMom;
	}
	public int getMom() {
		return mom;
	}

	public void setDad(int newDad) {
		this.mom = newDad;
	}
	public int getDad() {
		return dad;
	}
}
```