Implementierung

Inhaltsverzeichnis

Verwendete Tools

Für die Implementierung wurden mehrere Tools verwendet.

- VMware Workstation 16 Player mit Ubuntu 20.04 LTS 64 bit
- Redis 5.0.7
- Python 3.8.5
- redis-py (Python-Bibliothek für Redis)
- redis-commander (Web-UI für Redis Server)
- RedisInsight (Web-UI für Redis Server)

Implementierung der Struktur

Aufgrund der Eigenschaften einer Schlüssel-Werte-Datenbank, insbesondere der Flexibilität von Datenstrukturen, wird die Struktur mit dem Laden der Daten aufgebaut. Damit ist sichergestellt, dass nur tatsächlich benötigte Strukturen gespeichert werden. Die Implementierung der Struktur erfolgt somit während der Laufzeit im Datenloader, die Grundstruktur ist eine leere Redis-Instanz.

Implementierung der Datenloader

Es gibt 2 verschiedene Datenloader, einen für Input- und einen für Output-Datensätze. Diese bekommen vom Watchdog den Pfad mit der Datei, in dem der jeweiligen Datensatz steht, im Funktionsaufruf übergeben. Anschließend wird die Datei als csv-Datei mit Trennzeichen ";" behandelt. Besitzt der Datensatz keine Seriennummer, so wird der Datensatz als "defekter" Datensatz gespeichert (siehe Entwurf "Rohdatensätze ohne Seriennummer" und "Liste aller Rohdatensätze ohne Seriennummer"). Besitzt er eine Seriennummer, so wird je nach Typ (Input/Output) unterschieden. In den folgenden Vorgehensweisen wird von ordentlichen

Rohdatensätzen ausgegangen.

Input-Datensatz

Zuerst wird der Rohdatensatz als Hash mit entsprechenden Feld-Werte-Paaren abgespeichert. Besitzt er keine Seriennummer, so wird er als defekter Datensatz abgelegt. Anschließend wird der Schlüssel dieses Hash an die Liste der zugehörigen SNR angehängt, um den Datensatz für den Output verknüpfbar zu machen. Danach wird die Verknüpfung für diesen In-Datensatz als Liste angelegt, wobei als Zeitstempel 2-mal das Datum in Sekunden genommen wird und der Wert der Schlüssel des Rohdatensatzes ist. Diese Verknüpfung wird an die Liste aller Verknüpfungen angehangen. Nun wird jede vorgegebene Eigenschaft durchgegangen. Falls die Eigenschaft noch nicht oder noch nicht in dieser Ausprägung vorhanden ist, wird sie angelegt. Das 1-Bit wird in den entsprechenden Datensätzen an der entsprechenden Position angelegt. Der Zeitstempel des Rohdatensatz wird in die Zeitverknüpfung als erstes Element der Liste eingetragen und der Zähler (inCounter) inkrementiert.

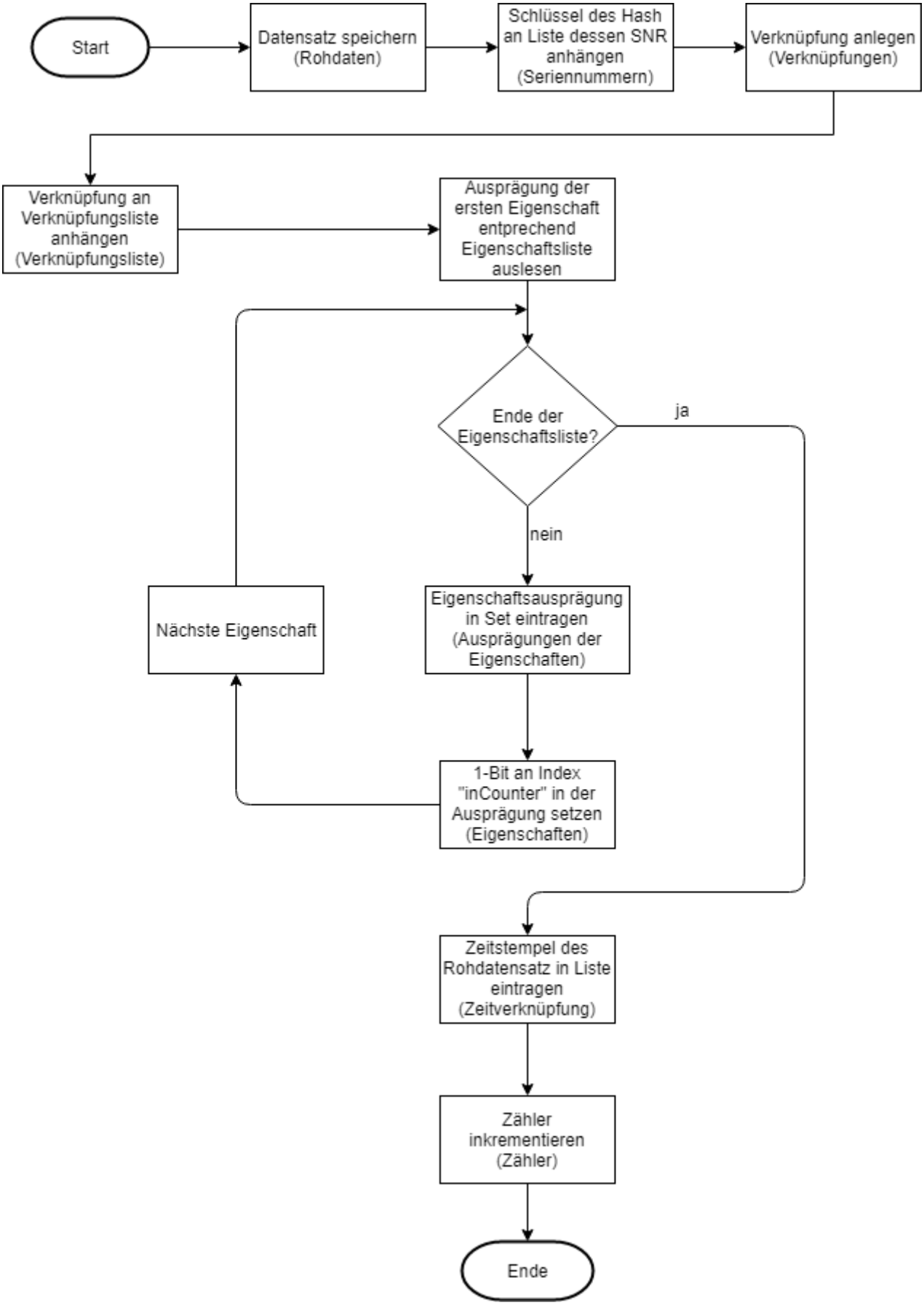

Figure 1. Einlesealgorithmus Input-Datensatz

Output-Datensatz

Der Rohdatensatz wird analog zu den Input-Datensätzen als Hash abgespeichert. Im nächsten Schritt werden alle Input-Datensätze aus der Liste der Seriennummer des Out-Datensatzes durchgegangen und die Zeitdifferenzen des Input und Output berechnet. Der Output-Datensatz wird dem Input mit der kleinsten positiven Zeitdifferenz zugeordnet. Im Anschluss wird die Zeitverknüpfung dieses Input-Datensatzes als Liste geladen. Gibt es in dieser Liste nur 1 Element, so wurde dem Input noch kein Output zugeordnet und der Zeitstempel des Output-Datensatzes wird angehangen. Ist der Zeitstempel des Outputs größer als der des bisherig größten Outputs in der Zeitverknüpfung, so wird dieser Zeitstempel mit dem des neuen ersetzt. Entsprechend wird der Schlüssel der Verknüpfung mit diesem neuen Zeitstempel versehen. Anschließend wird der Schlüssel des Output-Rohdatensatzes als Element in die Verknüpfung aufgenommen und der Schlüssel der Verknüpfung in der Liste aller Verknüpfungen entsprechend mit dem aktuellen Schlüssel ersetzt.

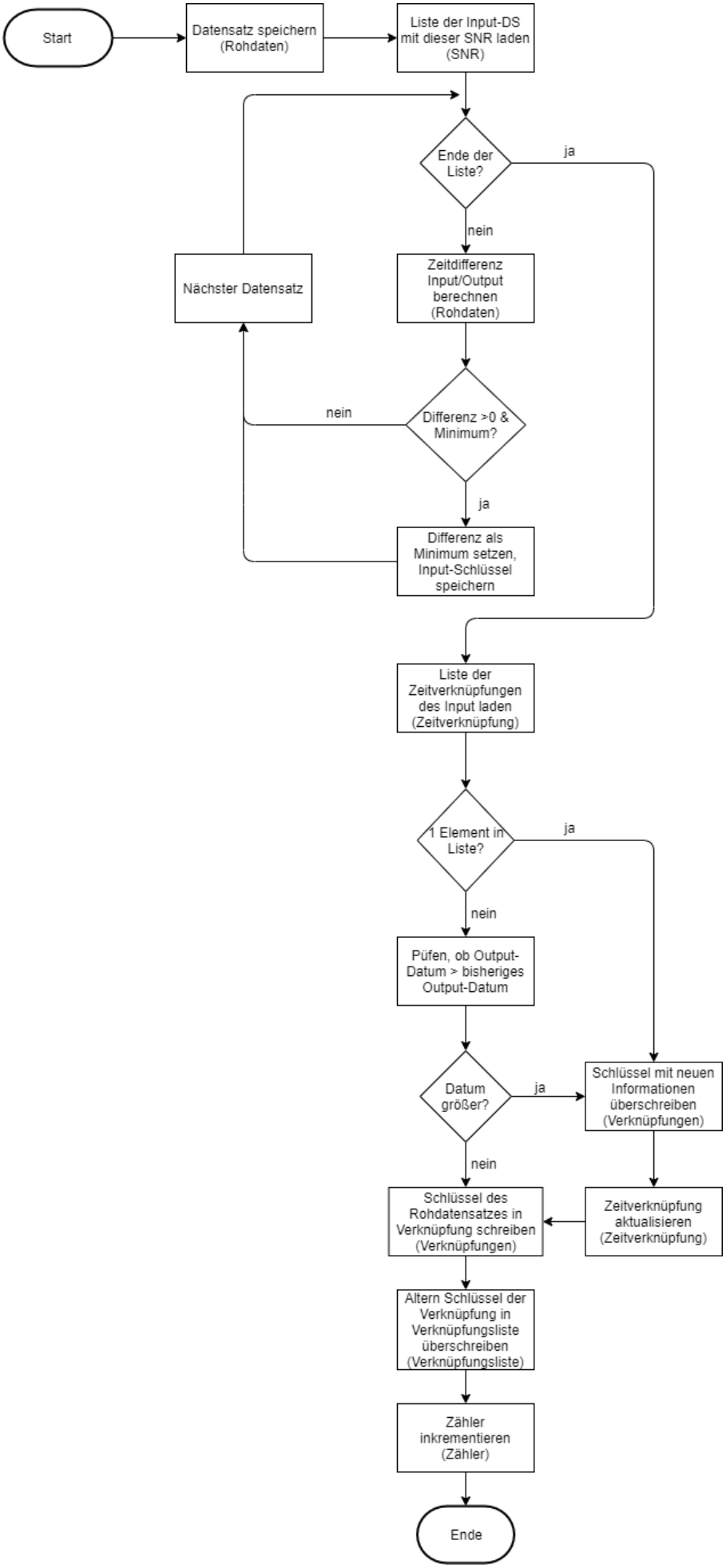

Figure 2. Einlesealgorithmus Output-Datensatz

Implementierung der Analysen

Vorgehensweise

Die Analysen folgen einem Grundgerüst, welches sich zwar in der Ausprägung, nicht aber in der Struktur unterscheidet. Am Anfang werden die Listen mit den benötigten Daten geladen. Diese sind zum einen die Liste mit allen Verknüpfungen, die Verknüpfungsliste, und die Sets der gefragten Eigenschaften mit den Ausprägungen als Elemente dieser.

Laden der benötigten Daten

```
import redis
r = redis.Redis(decode_responses=True)

allCons = r.lrange("con",0,-1)
allConLen = len(allCons)
allTeil = r.smembers("TEIL")
allFa = r.smembers("FA")
```

Danach wird die Ausgabedatei vorbereitet.

Ausgabedatei vorbereiten

```
writer = ""
diffFile = open("takt1.txt", "w")
diffFile.write("TEIL;FA;COUNT;MIN;MAX;AVG\n")
```

Anschließend wird ein Lua-Skript vorbereitet und in Redis eingebunden. Das Skript ist notwendig, da es in Redis keine Funktionalität dafür gibt die 1-Positionen einer Bitmap sinnvoll auszulesen. In unserem Fall brauchen wir die 1-Positionen der Bitmap "opCon", welche das Ergebnis der bitweisen AND Operation speichert. Redis ermöglicht es Lua-Skripte einzubinden und diese serverseitig auszuführen. Das Skript ist in den Quelltexten der Analysen aufgeführt, im folgenden Beispiel durch drei Punkte ersetzt.

Lua-Skript einbinden

```
lua = """ ... """
myLua = r.register_script(lua)
```

Nun erfolgt die eigentliche Analyse. Dabei wird immer mit den Bitmaps der Eigenschaften gearbeitet, da diese angeben welche Verknüpfungen diese Eigenschaft besitzen. Zur Veranschaulichung dient folgendes Beispiel:

- Gesucht: Alle Verknüpfungen mit den Eigenschaften "TEIL:B" und "FA:005830"
- Bitmap "TEIL:B" → [0101010]
- Bitmap "FA:005830" → [1100110]

Die 1-Bits einer Bitmap bedeuten, dass die Verknüpfung mit der Nummer "Index des 1-Bit" diese Eigenschaft besitzt (Erster Index einer Bitmap ist Index Nr. 1). In dem Beispiel besitzen also die Verknüpfungen 2,4 und 6 die Eigenschaft "TEIL:B" und Verknüpfungen 1,2,5,6 die Eigenschaft "FA:005830". Um nun diejenigen Verknüpfungen zu finden, welche beide Eigenschaften besitzen, müssen nur beide Bitmaps bitweise AND verknüpft werden. Das Ergebnis dieser Operation ist wieder eine Bitmap, welche unter dem Schlüssel "opCon" gespeichert wird.

- Bitmap "opCon" → [0100010]

Es besitzen also Verknüpfung Nummer 2 und 6 beide Eigenschaften. Das eingebundene Lua-Skript gibt nun alle 1-Positionen von "opCon" als Liste zurück, in dem Fall also "(2,6)". Diese Liste kann nun auf die Liste aller Verknüpfungen angewendet werden, wodurch nur Verknüpfungen mit diesen bestimmten Eigenschaften betrachtet werden. Im Folgenden noch das Codebeispiel.

Bitmap Operationen

```
#Bitweise AND für dieses Teil und diesen Fertigungsauftrag
r.bitop("AND","opCon",teil,fa)
#Lua-Skript auf 'opCon' anwenden
#So viele Bits auf 1en prüfen wie die Liste aller Verknüpfungen lang ist
result = myLua(keys=['opCon'],args=[1,allConLen])
#Für jede 1-Position die passende Verknüpfung aus Liste holen
for res in result:
    con = allCons[res-1]
    #...
```

Am Ende jeder Analyse wird das Ergebnis in einer entsprechenden Ausgabedatei festgehalten.

Ausgabedatei schreiben

```
writer = teilSplit+";"+faSplit+";"+str(menge)+";"+str(minDiff)
            +";"+str(maxDiff)+";"+str(round(avgGesTime,2))+"\n"
diffFile.write(writer)
```

Darstellung der Analysen in Pseudo-Code

Da Redis aufgrund der Eigenschaften einer Schlüssel-Werte-Datenbank selbst keine Möglichkeit für komplexe Abfragen bietet, muss die Abfragelogik zum großen Teil in Python geschehen. Das führt dazu, dass der Quellcode für die Analysen recht lang ist. Daher ist im Folgenden nur Pseudo-Code dargestellt, welcher die wichtigsten Schritte jeder Analyse vorgibt.

Legende:

- FA: Fertigungsauftrag
- SNR: Seriennummer
- LA: Ladungsträger

Analyse 1:

```
FOR EACH Teil in alleTeile {
    FOR EACH FA in alleFA {
        Verknüpfungen ermitteln, welche dieses Teil und diesen FA haben

        IF Verknüpfung(en) exisieren {
            FOR EACH Verknüpfung {
                IF Verknüpfung besitzt Ouput {
                    Zeitdifferenz Input/Output berechnen
                    IF Differenz <= 1 Stunde {
                        Differenz auf Minimum/Maximum prüfen
                        Differenz auf Gesamtzeit addieren
                        Ausschuss für diese SNR inkrementieren
                    }
                }
            }

            Menge an SNR ermitteln
            Maximum, Minimum und Durchschnitt Ausschuss berechnen
            Maximum, Minimum und Durchschnitt Zeiten berechnen
            In Ausgabedatei schreiben
        }
    }
}
```

Analyse 2:

```
FOR EACH Teil in alleTeile {
    Verknüpfungen ermitteln, welche dieses Teil haben

    FOR EACH Verknüpfung {
        Verbinde die Input/Output Zeitstempel mit der jeweiligen SNR
    }

    FOR EACH SNR in SNR-Zeitstempel-Verbindungen {
        Ausschuss berechnen
        Verbindungen nach Input-Zeitstempel sortieren

        IF Anzahl Verbindung > 1 {
            FOR EACH Verbindung {
                Berechne Zeitdifferenz letzter Output bis aktueller Input
                Differenz auf Maximum / Minimum prüfen
                Differenz auf Gesamtzeit addieren
            }
        }
    }

    Durchschnitte berechnen
    In Ausgabedatei schreiben
}
```

Analyse 4:

```
FOR EACH LA in alleLA {
    Verknüpfungen ermitteln, welche diesen LA haben

    FOR EACH Verknüpfung {
        Zeitstempel auf Maximum / Minimum prüfen
    }

    Differenz von Maximum und Minimum berechnen
    In Ausgabedatei schreiben
}
```

Analyse 5:

```
FOR EACH Teil in alleTeile {
    FOR EACH LA in alleLA {
        Verknüpfungen ermitteln, welche dieses Teil und diesen FA haben

        IF Verknüpfung(en) existieren {
            FOR EACH Verknüpfung {
                SNR in Set aller SNR hinzufügen

                IF Verknüpfung besitzt Ouput {
                    Zeitdifferenz Input/Output berechnen
                    Differenz auf Gesamtzeit addieren
                    Differenz auf Minimum/Maximum prüfen
                }
            }

            Menge an SNR ermitteln
            Maximum, Minimum und Durchschnitt Zeiten berechnen
            In Ausgabedatei schreiben
        }
    }
}
```

Analyse 6:

```
FOR EACH Linie in alleLinien {
    FOR EACH FA in alleFA {
        Verknüpfungen ermitteln, welche diese Linie und diesen FA haben

        FOR EACH Verknüpfung {
            Zeitstempel auf Maximum / Minimum prüfen

            IF Maximum {
                FA Maximum zuordnen
            }
            IF Minimum {
                FA Minimum zuordnen
            }
        }
    }

    FA-Zeit-Verbindungen nach Zeitstempel des Input aufsteigend sortieren

    FOR EACH FA-Zeit-Verbindung {
        Zeitdifferenz letzter Input / aktueller Input berechnen
        Differenz auf Maximum / Minimum für diese Teilkombination prüfen
    }

    In Ausgabedatei schreiben
}
```

Messung der Performance

Für die Performancemessung kommen 2 Kennzahlen zum Einsatz, zum einen die Zeitdauer der Ausführung des Python-Skripts und zum anderen die Zeitdauer der Ausführungen der Operationen in Redis.

Für Python wurde die Zeit in Nanosekunden mithilfe der Funktion process_time_ns() des Moduls time gemessen. Diese misst nur die reine Prozesszeit des Programms. Der Timer startet direkt nach dem Einbinden der Bibliotheken und endet nach der letzten für die Analyse relevanten Operation.

Messung der Prozesszeit des Python Programms

```python
from time import process_time_ns
start = process_time_ns()
#...
stop = process_time_ns()
print(str((stop-start)/10**9))
```

Die Ausführungszeiten der Redis Operationen wurden mit der built-in Funktionalität SLOWLOG gemessen. Dieses System ermöglicht es, die Ausführungszeiten von Operationen zu messen welche eine bestimmte Dauer überschreiten. Um alle Operationen zu messen wurde die Schranke auf 0 Mikrosekunden gesetzt ("slowlog-log-slower-than 0"). Da sich in der Praxis allerdings gezeigt hat,

dass dieser Log für eine große Anzahl an Operationen nicht zuverlässig funktioniert (ab mehreren tausend Einträgen wurden alle Einträge gelöscht), wurden die Operationszeiten für jeden Durchlauf (z.B. ein Input-Datensatz wird vollständig eingelesen) gemessen und der SLOWLOG im Anschluss mittels "SLOWLOG RESET" bereinigt. Im Folgenden ist ein Beispiel für die Messung der Operationszeit beim einlesen eines Input-Datensatzes aufgeführt. Dabei ist anzumerken, dass "SLOWLOG RESET" und "SLOWLOG LEN" (Anzahl Einträge im Log) selbst gemessen werden und damit das Ergebnis verfälschen würden, weshalb diese in der if-Anweisung ausgeklammert werden.

Messung der Prozesszeit von Redis

```
timeLog = r.slowlog_get(r.slowlog_len())
for time in timeLog:
    if time['command'] != 'SLOWLOG RESET' and time['command'] != 'SLOWLOG LEN':
        timeFile.write(str(time['command'])+str(time['duration'])+'\n')
r.slowlog_reset()
```