Aufgabe3

the_reapers

verwendetet Actors:

- Reader
- Master
- Worker
- Collector
- Reaper

Grundkonzept:

- Erst wenn alle Hinweise zu einem Passwort gelöst sind, wird das Passwort geknackt
- Master teilt Hinweise auf Worker auf
- Informationen über das Charset, welches verwendet werden soll, um das Passwort zu knacken, werden in Bloomfilter verwaltet -> gesetztes Bit im Bitset bedeutet, dass der Char beim entsprechenden Index im Passwort nicht vorkommt;
    - Bsp,: c = „ABCDE“ und 3 ist im BloomFilter vorhanden -> c[3] = „D“ ist im Passwort nicht vorhanden

Reader:

- Liest Zeilen aus CSV ein, nicht durch uns verändert

Master:

- Erhält BatchMessage mit Zeilen vom Reader
- Teilt Hinweise aus den Zeilen entsprechend auf Worker auf als Reaktion auf deren RegistrationMessage: WelcomeMessage wird gesendet
    - Enthält u.a. aktuellen Bloomfilter mit Informationen über das Charset und den entsprechenden Hash
- Wenn Hinweis gelöst, wird eine HintMessage vom Worker erhalten, wo das passende Bit im BloomFilter gesetzt ist, Master nutzt merge()-Funktionen, um die Information zu erhalten und sendet die HintMessage weiter an alle anderen Worker, die an der Zeile arbeiten, um redundantes Rechnen zu verhindern
- Alle Hinweise zu einer Zeile gelöst: PasswordMessage an einen Worker:
    - Enthält u.a. Bloomfilter mit allen Informationen und den zu knackenden Hash
- Bei geknacktem Passwort wird das Ergebnis an Collector geschickt mit CollectMessage

Worker:

- Erhält WelcomeMessage bei Start mit zu knackendem Hinweis, speichert u.a. Bloomfilter und den Hash
- Ermittelt mit nextClearBit()-Funktion den nächsten Char, der aus dem Charset zu entfernen ist und versucht mit Permutationsbildungen den Hash zu knacken
- Bei Erfolg wird HintMessage an Master gesendet mit aktualisiertem Bloomfilter (passendes Bit wird gesetzt)
- Bei Misserfolg wird YieldMessage an sich selbst gesendet, um zu unterbrechen und auf Hinweis der anderen Worker zu hoffen
- Wird HintMessage erhalten, wird auch hier mit merge()-Funktionen der Bloomfilter aktualisiert
- Bei PasswordMessage wird das Passwort mit gegebenem Charset geknackt und PasswordResultMessage an Master gesendet

Collecter:

- Sammelt Ergebnisse, nicht durch uns verändert

Reaper:

- Ist für Shutdown des Systems verantwortlich, nicht durch uns verändert

Designentscheidungen:

- Beim Knacken der Hinweise startet jeder Worker bei einem zufälligem Char. Starteten alle beim gleichen Char, hätte das Weiterleiten der Hinweise wenig Sinn, da sie den entsprechenden Char in den meisten Fällen auch schon ausprobiert hätten.
- Wir haben die rekursiven Funktionen für die Bildung der Permutationen und die Bildung der Kombinationen durch iterative ersetzt. Bei den Passwörtern brachte und das einen großen Vorteil, da wir nach jeder Iteration gehasht haben und somit nicht alle Kombinationen und Hashes berechnen mussten. Bei den Hinweisen brachte uns das nur einen kleinen Vorteil, da in den meisten Fällen trotzdem alle Permutationen und somit auch Hashes berechnet werden mussten.
- Generell ist unsere Lösung sehr „straight forward“. Es werden immer alle Kapazitäten so gut wie es geht genutzt. Daher skaliert unsere Lösung sehr gut mit mehreren Worker-Systemen. Nur am Ende, wenn auf einige Hinweise noch gewartet wird oder das letzte Passwort berechnet wird, sind die Worker IDLE. Lediglich der Verwaltungsaufwand des Masters ist recht hoch. Wir haben versucht das möglichst effizient zu gestalten.