# Implementation

In the following section we will describe how each step of the algorithm is performed in pseudo code. Then, in order to describe the MapReduce flow, we shall explore the Pig script written for the algorithm.

## Algorithm

The algorithm will take a few input parameters from the user before running

- a list of sequences $S = s_0, s_1, ..., s_{N-1}$ of length $N$
- $k$-mer size $k$.
- $H$, the number of hash functions in the sketch
- $\epsilon$, the threshold for similarity.

Initially, the sequences had to be transformed into meaningful input for the hash functions used to produce the sketches **MM** and $\mathbf{MM}\frac{1}{2}$. Natural numbers fit the purpose well, and are exactly what $k$-mer transformations constitute. The function `kmerTransformation`$(s, k)$, as seen in Alg. 1, are how the transfomations were got. The function `transformChar` uses the same transformation as described in the Tools section.

**Algorithm 1** Transforms sequence $s$ into its $k$-mer transformation

**Input:** a sequence $s$, a $k$-mer size $k$
**Functions:** `transformChar`$(s, k)$ maps the characters of a $k$-mer to 2-bit values, `generateKmer`$(s, k)$ returns the $k$-mer of $s$.
**Output:** The transformed $k$-mer in a list kmerTransformed.

```
1:  function kmerTransformation(s, k)
2:      kmer ← generateKmer(s, k)                    ▷ saves k-mer of s
3:      for gram in kmer do
4:          prod ← 1
5:          sum ← 0
6:          for i ← 0 to gram.length − 1 do
7:              sum ← sum + prod · transformChar(gram.charAt(i))
8:              prod ← prod · 4
9:          kmerTransformed.append(sum)
10:     return kmerTransformed
```

Once the transformations were done, we needed to produce **MM** and $\mathbf{MM}\frac{1}{2}$. As we concluded in the Hash Performance Test, we were to use the Carter Wegman hash function (Eq. **??**), which are referred to as $h_i(x)$ from now on. For the sketch, we generate $H$ hashfunctions for **MM** and $H/2$ hash functions for $\mathbf{MM}\frac{1}{2}$, each with a different $a$ and $b$. Alg. 2 shows how **MM** was produced. As we can see, it runs through all $H$ hash functions to find the value that is smallest and highest in the transformed $k$-mer list. Alg. 3 shows the pseudocode for calculating $\mathbf{MM}\frac{1}{2}$, where the only change needed was to divide the number of hash functions by two.

**Algorithm 2** Uses transformed $k$-mer to find **MM** of a sequence

**Input:** a list of transformed $k$-mer $kT$ of a sequence, $H$ number of hashfunctions.
**Parameters:** $h_i(x)$ the ith hash function of the $H$ hash functions initiated.
**Output:** **MM** of a sequence.

```
1:  function toMMSketch(kT)
2:      MM ← Ø
3:      for i ← 0 to H − 1 do
4:          hmin ← ∞
5:          hmax ← −∞
6:          for kt in kT do
7:              val ← h_i(kt)
8:              if val < hmin then hmin = val
9:              if val > hmax then hmax = val
10:         MM.append([hmin,hmax])
11:     return kmerTransformed
```

**Algorithm 3** Uses transformed $k$-mer to find $\mathbf{MM}\frac{1}{2}$ of a sequence

**Input:** a list of transformed $k$-mer $kT$ of a sequence, $H$ number of hashfunctions.
**Parameters:** $h_i(x)$ the ith hash function of the $H/2$ hash functions initiated.
**Output:** $\mathbf{MM}\frac{1}{2}$ of a sequence.

```
1:  function toMMSketch(kT)
2:      MMhalf ← Ø
3:      for i ← 0 to H/2 − 1 do
4:          hmin ← ∞
5:          hmax ← −∞
6:          for kt in kT do
7:              val ← h_i(kt)
8:              if val < hmin then hmin = val
9:              if val > hmax then hmax = val
10:         MMhalf.append([hmin,hmax])
11:     return kmerTransformed
```

The sketches ready, what remained was to compare all sequences' sketches to each other. As speed was a main concern, a greedy algorithm was developed for both **MM** and $\mathbf{MM}\frac{1}{2}$.

The greed lays in that if the similarity between two sequences is above $\epsilon$, they are immediately placed in the same cluster, regardless of whether there are other clusters that the other sequence is more similar to.

Alg. 4 shows the algorithm for **MM** greedy clustering. In line 1-7 the sketches are prepared for the comparison; the remainder is the greedy algorithm. The most interesting part of this algorithm is line 15-22, where the similarity between two strings is determined. Running through all $H$ hash functions, in line 16-20 it checks that if either $h_{min,i}$ or $h_{max,i}$ of both are equal to the other's counterpart, they are an intersection. When all hash functions have been run through, the similarity between the two strings is determined in line 21. If the similarity then surpasses the given $\epsilon$ the strings are placed in the same cluster.

Alg. 5 has a very similar method as Alg. 5. The main difference lies in line 15-22, where the intersections are determined by the number of $h_{min,i}$ that are equal plus the number of $h_{max,i]}$ that are equal. This means that each hash function can result in two intersections, instead of one in Alg. 4. Line 20 the shows the similarity measure, which is the same as in Alg. 4, since the potential number of intersections was doubled.

**Algorithm 4** Greedy Clustering using **MM**

**Input:** a list of DNA/RNA sequences $S = \{s_0, s_1, ..., s_{N-1}\}$ of length $N$.
**Parameters:** $H$ number of hash functions, $k$ size of $k$-mer, $\epsilon$ threshold for string similarity, $c$ current cluster
**Output:** A list of the cluster $C$ each sequence belongs to.

Initialize $h_i(x)$ for $i = 0, 1, ..., H$
Initialize $kT$,$C$, and $MM$ as lists of length $N$
**for** $i \leftarrow 0$ to $N - 1$ **do**
    $C$[i] $\leftarrow 0$
    $kT$[i] $\leftarrow$ kmerTransformation($s_i$)
    $MM$[i] $\leftarrow$ toMMSketch($kT$[i])
$c \leftarrow 0$
**for** $i \leftarrow 0$ to $N - 1$ **do**
    **if** $C$[i] == 0 **then**
        $c \leftarrow c + 1$
        $C$[i] $\leftarrow c$
        **for** $j \leftarrow 0$ to $N - 1$ **do**
            $intersections \leftarrow 0$
            **if** $C$[i] == 0 **then**
                **for** $k \leftarrow 0$ to $H - 1$ **do**
                    **if** $MM$[i][k][0] == $MM$[j][k][0] **then**
                        $intersections \leftarrow intersections + 1$
                    **else**
                        **if** $MM$[i][k][1] == $MM$[j][k][1] **then**
                            $intersections \leftarrow intersections + 1$
                **if** $\frac{intersections}{H} \geq \epsilon$ **then**
                    $C$[j] $\leftarrow C$[i]

**Algorithm 5** Greedy Clustering using $\mathbf{MM}^{\frac{1}{2}}$

**Input:** a list of DNA/RNA sequences $S = \{s_0, s_1, ..., s_{N-1}\}$ of length $N$.
**Parameters:** $H$ number of hash functions, $k$ size of $k$-mer, $\epsilon$ threshold for string similarity, $c$ current cluster
**Output:** A list of the cluster $C$ each sequence belongs to.

Initialize $h_i(x)$ for $i = 0, 1, ..., H/2$
Initialize $kT$,$C$, and $MM$ as lists of length $N$
**for** $i \leftarrow 0$ to $N - 1$ **do**
    $C$[i] $\leftarrow 0$
    $kT$[i] $\leftarrow$ kmerTransformation($s_i$)
    $MM$[i] $\leftarrow$ toMMhalfSketch($kT$[i])
$c \leftarrow 0$
**for** $i \leftarrow 0$ to $N - 1$ **do**
    **if** $C$[i] == 0 **then**
        $c \leftarrow c + 1$
        $C$[i] $\leftarrow c$
        **for** $j \leftarrow 0$ to $N - 1$ **do**
            $intersections \leftarrow 0$
            **if** $C$[i] == 0 **then**
                **for** $k \leftarrow 0$ to $H/2 - 1$ **do**
                    **if** $MM$[i][k][0] == $MM$[j][k][0] **then**
                        $intersections \leftarrow intersections + 1$
                    **if** $MM$[i][k][1] == $MM$[j][k][1] **then**
                        $intersections \leftarrow intersections + 1$
                **if** $\frac{intersections}{H} \geq \epsilon$ **then**
                    $C$[j] $\leftarrow C$[i]

**MapReduce Flow**