# Algorithm description

Problem definition: Given a set of COG-spelled genomes S, where each genome is segmented into segments such that each segment could contain one or more operons. Also given parameters d, and q. Find all gene clusters 𝑤 (over the COG alphabet) of length 𝑑 that are conserved in ≥ 𝑞 of the genomes in the database 𝑆. Sort the gene clusters you found in decreasing order of abundance.

Algorithm input: S, d and q as described above.

Algorithm description:

1. Let Genome be an ordered pair <name, list>, where “name” is the genomes name and “list” contains all segments of the genome (a segment is a string over the COG alphabet)
2. Let genomeList be a list containing Genomes extracted from S.
3. Cluster w is a multiset where if σ is in 𝑤, then σ is in the COG alphabet.
4. Let clusterList be an empty list.
5. Let map(Cluster-> genomeList ) be a hash map, where “key” is a cluster, and “value” is a list containing strings. Note, for run time efficientcy and convenience, we will store each cluster in a tuple.
6. For each Genome on the list, go over its segments list.

For each segment:  
\* create a cluter from each d length subsequence called myCluster.  
\* check if myCluster appears in the map.  
if it does appear: check if its value (a list) contains current genomes name, and if it doesn’t, insert the Genomes name to the list map(cluster). In addition, check if list map(myCluster) size is q.

if it doesn’t appear: insert < myCluster, {Genomes’s name} > to map.

1. Sort map by the length of each value in the map (descending order) and return the 20 first clusters whom their genome list is equal or larger than q (if exists).

Runtime analysis

For the runtime analysis, wi will use pseudo code:

Algorithm(S, d,q):

1 Initialize map(Cluster->stringList) where set will represent a cluster and stringList will represent a genome sequence’s name list.

2 Initialize genomeList(S).

3 For each Genome in genomeList:   
 3.1 for each sequence in Genome.sequences:

3.2 for each subsequence of length d in sequence

3.2.1 Create multiSet from subsequence

3.2.2\* Map[cluster].add(sequence.name) on average

4. Sort map by value.length (where c is the number of clusters found)

5. Return clusterList

Overall, the algorithms run time is

Why is it correct and efficient?

This algorithm goes over the database, divides it into segments, and goes over every d-mer in it. For each d-mer, it creates a multiset from its COG letters, and the multiset (cluser) is used as a key in our hash map. In the hash map we document each genome the cluster (key) was found in.

Thus, the hash map documents every cluster found in the database. By sorting the hash map we get the most frequent clusters in it. Our algorithm requires one reading of the database. The number of actions done for each letter is proportional to d, which is usually small.