Intro to Data Science - Assignment 3

> Names: Dean Tahory (319045837) Omer Mazig (204614770)

Question 1 - K-means Implementation

K-means is an iterative algorithm where each iteration consists of two parts:

- Assignment Step: where for every $x \in S$ we find the index of the closest centroid and assigned the cluster to be that index.
- Re-Estimation Step: update each centroid to be the average of the points that have been assigned to it.

The assignment step can be done in a mapper function by finding the closest centroid to every point independently from other points, then emit the points centroid with the point itself. The shuffling phase aggregates the points of each centroids and by that the reducer function can calculate the next centroid for each cluster by averaging. Finally the reducer returns the (old_centroid, updated_centroid) so the centroids list would be updated for the next iteration.

```
def Mapper(key,p,centroids):
  p_centroid = centroids[0]
  for c in centroids:
      if norm(p,c) < norm(p,p_centroid):
          p_centroid = c
  yield (p_centroid, p)

def Reducer(key,values): #key = centroid of last iteration
  new_centroid = mean(values)
  if key!= new_centroid:
    yield (key, mean(values)) # (old_centroid, updated_centroid)
  else: # there is no update so we print all points with their centroid
    for val in value:
      yield(val, key) # print (point,centroid) pair
```

Stoping condition - where there was no update after iteration. i.e. old_centroid equals updated_centroid after all the reducers completed running.

Question 2 - CheckClique

Note: We assume that the given graph is connected.
otherwise we will have 2 MapReduce operations at least. That's because given a graph consists of number of cliques, they don't know nothing about each other so after we will check for every component if its clique, the reducer is going to output the cliques and a new mapper will need to map all the sub cliques to same key so the reducer could check if there is more then one sub clique and if so return "NO".

In order for a graph to be full clique every node should have edges to all the nodes in the graph. So by counting the number of unique neighbors for each node, and checking that it's equal to $n-1$ where n is the total number of nodes, we can check if a graph graph is full clique or not.

Note: we assume that the given graph is connected.

```
def Mapper(k,v):
  d= split('->')
  for neighbor in d[1):
    yield (d[0], neighbor)

def Reducer(k, values):
  if n-1 == len(set(values)): yield "YES"
  else yield "NO"
```

Question 3 - pseudo-synonyms detection

To solve the problem we need to know:

1. The pairs of words that appeared at the same location in a given sentence. i.e candidates for synonyms
2. How many sentences those pairs appeared together in.

There is no way to do these steps together in one MapReduce operation because they must contradicting keys in the Map stage - step 1 should have the the sentence without the candidates as a key so at the aggregation step the candidates will be together, step 2 should have the pairs as keys therefore contradicting step 1.

So we are going to implement 2 M/R operations:

Stage 1

The mapper will map each sentence by its first and last words as a key and a candidate synonym as the second word. The the reducer will return every possible pair of synonym of each sentence.

```
def Mapper(k,v):
  d = v.split(" ");
  yield (d[0],d[2]),d[1]

def Reducer(k,values):
    for x, y in lex_permutations(values):
      yield (x,y), 1
```

Stage 2

The mapper gets a candidate pair of synonyms and use them as a key, and the value will be the number of sentences they appeared together. the reducer will check if the aggregation of this number for the synonym is greater from 2 for each pair, and if so it will return it.

```
def Mapper(k,v):
  yield k,v

def Reducer(k,v):
  n = sum(v)
  if n >=2: yield f"{k[0]} - {k[1]} ({n})" # e.g. cheap - new (2)
```