In [1]:

```
import numpy as np
from numba import njit
import edlib


def get_rc(s):
    map_dict = {'A': 'T', 'T': 'A', 'G': 'C', 'C': 'G', 'N': 'N'}
    l = []
    for c in s:
        l.append(map_dict[c])
    l = l[::-1]
    return ''.join(l)
def rc(s):
    map_dict = {'A': 'T', 'T': 'A', 'G': 'C', 'C': 'G', 'N': 'N'}
    l = []
    for c in s:
        l.append(map_dict[c])
    l = l[::-1]
    return ''.join(l)

def seq2hashtable_multi_test(refseq, testseq, kmersize=15, shift = 1):
    rc_testseq = get_rc(testseq)
    testseq_len = len(testseq)
    local_lookuptable = dict()
    skiphash = hash('N'*kmersize)
    for iloc in range(0, len(refseq) - kmersize + 1, 1):
        hashedkmer = hash(refseq[iloc:iloc+kmersize])
        if(skiphash == hashedkmer):
            continue
        if(hashedkmer in local_lookuptable):

            local_lookuptable[hashedkmer].append(iloc)
        else:
            local_lookuptable[hashedkmer] = [iloc]
    iloc = -1
    readend = testseq_len-kmersize+1
    one_mapinfo = []
    preiloc = 0
    while(True):

        iloc += shift
        if(iloc >= readend):
            break

        #if(hash(testseq[iloc: iloc + kmersize]) == hash(rc_testseq[-(iloc + kmersize): -iloc])):
            #continue

        hashedkmer = hash(testseq[iloc: iloc + kmersize])
        if(hashedkmer in local_lookuptable):

            for refloc in local_lookuptable[hashedkmer]:

                one_mapinfo.append((iloc, refloc, 1, kmersize))



        hashedkmer = hash(rc_testseq[-(iloc + kmersize): -iloc])
        if(hashedkmer in local_lookuptable):
            for refloc in local_lookuptable[hashedkmer]:
                one_mapinfo.append((iloc, refloc, -1, kmersize))
        preiloc = iloc



    return np.array(one_mapinfo)

def get_points(tuples_str):
    data = []
    num = 0
    for c in tuples_str:
        if(ord('0') <= c <= ord('9')):
            num = num * 10 + c - ord('0')
        elif(ord(',') == c):
            data.append(num)
            num = 0
    if(num != 0):
        data.append(num)
    return data

def calculate_distance(ref, query, ref_st, ref_en, query_st, query_en):
    A = ref[ref_st: ref_en]
    a = query[query_st: query_en]
    _a = rc(query[query_st: query_en])
    return min(edlib.align(A, a)['editDistance'], edlib.align(A, _a)['editDistance'])

def get_first(x):
    return x[0]
```

```
def calculate_value(tuples_str, ref, query):

    slicepoints = np.array(get_points(tuples_str.encode()))
    if(len(slicepoints) > 0 and len(slicepoints) % 4 == 0):
        editdistance = 0
        aligned = 0
        preend = 0
        points = np.array(slicepoints).reshape((-1, 4)).tolist()
        points.sort(key=get_first)
        for onetuple in points:
            query_st, query_en, ref_st, ref_en = onetuple
            if(preend > query_st):
                return 0
            if(query_en - query_st < 30):
                continue
            preend = query_en
            if((calculate_distance(ref, query, ref_st, ref_en, query_st, query_en)/len(query[query_st:query_en])) > 0.1):
                continue
            editdistance += calculate_distance(ref, query, ref_st, ref_en, query_st, query_en)
            aligned += len(query[query_st:query_en])
        return max(aligned - editdistance, 0)
    else:
        return 0


```

In [2]:
```
#实验1
ref = 'TATTATAGTCTTCATTCTGTGTATTAGATTACTAAAGCATATTACTTCTGTCTAAATGAAATTTTGTATCCTTTAGTCAGCATCTTCCCTTTCTCCATCCACTTTTCTCTCCCAGCCTCTGGTAATCAACATTCTACCACAATTCTGTGAGTTCTACTTTTTTAGATTCCCCATGTAAGTAAGATCATGCTGTATTTGTCTTTGTGTGCCTGGCTTATCTGAGTTAACG
query = 'TATTATAGTCTTCATTCTGTGTATTAGATTACTAAAGCATATTACTTCTGTCTAAATGAAATTTTGTATCCTTTAGTCAGCATCTTCCCTTTCTCCATCCACTTTTCTCTCCCAGCCTCTGGTAATCAACATTCTACCACAATTCTGTGAGTTCTACTTTTTTAGATTCCCCATGTAAGTAAGATCATGCTGTATTTGTCTTTGTGTGCCTGGCTTATCTGAGTTAA

```

In [3]:
```
data = seq2hashtable_multi_test(ref, query, kmersize=9, shift = 1)
data.shape
```

Out[3]: (63746, 4)

In [4]:
```
#实验2
ref = 'TGATTTAGAACGGACTTAGCAGACATTGAAACTCGAGGGGTATAGCAATAGATGCCCAAAAAGGTAAGCGCCATAAGCGTGGTTCTACGAGCCAGGTGCTCATGCCTAAGTTCTGCGCCTTCGCTGTCACTTGGAAATACTGTAATGGATCATGCCTAGGTTATGCGCCTTCGGGGTCACTTCAACATACTGTAATGGATCATGCCTAGGTTTTGCGTGTTCGCTGTCA
query = 'TGATTTAGAACGGACTTAGCAGACATTGAAACTCGAGGGGTATAGCAATAGATGCCCAAAAAGGTAAGCGCCATAAGCGTGTTTCTACGAGCCAGGTGCTCATGCCTAAGTTCTGCGCCTTCGCTGTCACTGGGAAATACTGTAATGGATCATCCGTAGGTTATGCGCCTTCGGGGTCACTTCAACATACTGTAACGGATCGTGCCTAGGTTTTGCGTATTCGCTGT

```

In [5]:
```
data = seq2hashtable_multi_test(ref, query, kmersize=9, shift = 1)
data.shape
```

Out[5]: (2347, 4)

In [ ]:
```

```

```
在这里设计你的算法
```

In [ ]:
```
#Design a algorithm
def function(data):
    return [[q_st, q_en, r_st, r_en], ...]
```

In [ ]:
```

```

```
Result
```

In [ ]:
```
tuples_str = str(youfunction(data))
```

In [8]:
```
#Score
calculate_value(tuples_str, ref, query)
```

Out[8]: 2090

In [ ]:
```

```