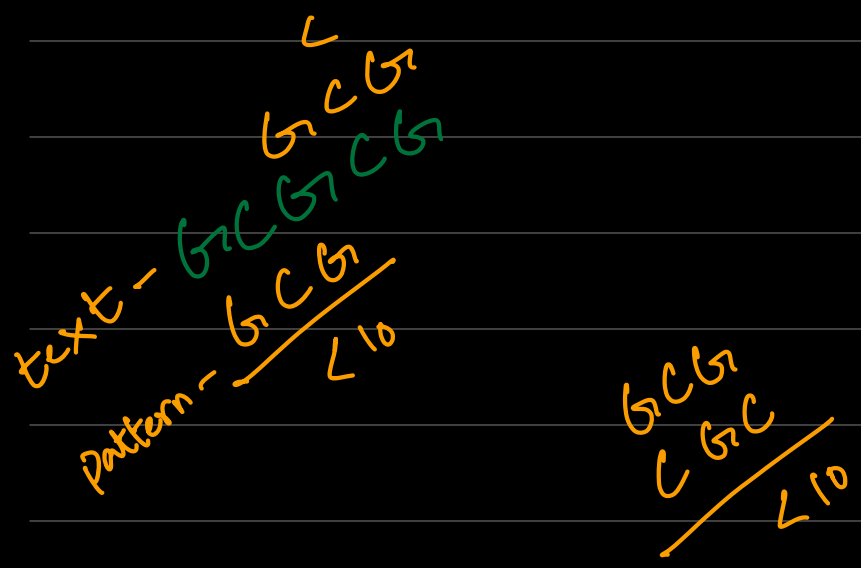
- Hamming Distance

```
def hamming_distance(p, q):
    mismatch = 0
    for i in range(0, len(p)):
        if p[i] != q[i]:
            mismatch += 1

    return mismatch
```

GTCG — 3

GACGACG — 5

```
def approxfind(pattern, text, d):
    indices = []
    for i in range(0, len(text) - len(pattern)
                   + 1):
        if hamming(text[i:i+len(pattern)],
                   pattern) <= d:
            append i to indices

    return indices.
```



→ This approach will only work if pattern appears in the string.

```
def frequent_words_with_mismatches(text, k, d):
    k_mers = { }

    for i in range(len(text) - k + 1):
        k_mer = text[i: i + k]
        count = approxfind(text[i:i+k], text, d)
        k_mers[k_mer] = count

    maximum = max(k_mers.values())
    L = [key for key in k_mers if k_mers[key] == maximum]

    return L
```

AGCAT CCACT AAAA ⋮ CAGA

can be grouped using group anagrams method.

⇒ Pattern might not appear in the string. Therefore, will have to generate all K_mers of length K as specified in the problem statement and store their "occurences" in a dictionary and return the maximum or most frequent K_mer.

```
bases = ["A", "T", "G", "C"]
def generate_kmers(K, bases):
    if K == 1:
        return bases
    small_kmers = generate_kmers(K-1, bases)
```

```
k_mers = []
for kmer in small_kmers:
    for b in bases
        append (k_mer + b) to
k_mers
```

return kmers

4-length kmer : $\_ \times \_ \times \_ \times \_ = 4 \cdot 4 \cdot 4 \cdot 4$

$= 4^4 = 256$ kmers

↑ 4 options since bases = ['A', 'G', 'T', 'C']

$= 4^k$ kmers

$\_ \times \_ \times \_ \times \_ \times \_ = 4^5 = 1024$ kmers

AAT → AAT, ATT

ATGCATTACGTATTCA

– Frequent Words with Mismatches and Reverse Complements.

```
def foo(text, K, d):
    bases = ['A', 'T', 'G', 'C']
    s = ""
    visited = set()
    K_mers = generate_Kmers(K, bases)
    K_mers_count = {}
    for Kmer in K_mers:
        if Kmer not in visited:
            r_c_Kmer = reverse_complement(Kmer)
            Count1 = len(approx_find(Kmer, text, 2)[1]
            Count2 = len(approx_find(____)[1]
            K_mers_count[Kmer] = sum(Count1,
                                     Count2).

    # return Kmers with most frequency.
```

Have to make sure if Kmer and its reverse complement same add once.