In [10]:

```
# === Algorithms for DNA Sequencing ====
# On Coursera, provided by John Hopkins School of Medicine
# Programming Homework 2

# Provides by staff:
import bm_preproc
import kmerindex
import subseq
# Binary Search module
import bisect


def readGenome(filename):
    genome = ''
    with open(filename, 'r') as f:
        for line in f:
            if not line[0] == '>':
                genome += line.rstrip()
    return genome


# provided in the problem statment:
# Do Boyer-Moore matching. p=pattern, t=text, p_bm=BoyerMoore object for p

def boyer_moore(p, p_bm, t):
    i = 0
    occurrences = []
    comparisons = 0
    alignments = 0
    while i < len(t) - len(p) + 1:
        alignments += 1
        shift = 1
        mismatched = False
        for j in range(len(p)-1, -1, -1):
            comparisons += 1
            if p[j] != t[i+j]:
                skip_bc = p_bm.bad_character_rule(j, t[i+j])
                skip_gs = p_bm.good_suffix_rule(j)
                shift = max(shift, skip_bc, skip_gs)
                mismatched = True
                break
        if not mismatched:
            occurrences.append(i)
            skip_gs = p_bm.match_skip()
            shift = max(shift, skip_gs)
        i += shift
    return occurrences, comparisons, alignments


def naive(p, t):
    occurrences = []
    comparisons = 0
    alignments = 0
    for i in range(len(t) - len(p) + 1):
        alignments += 1
        match = True
        for j in range(len(p)):
            comparisons += 1
            if t[i+j] != p[j]:
                match = False
                break
        if match:
            occurrences.append(i)
    return occurrences, comparisons, alignments

def approximate_match(p, t, n):
    segment_length = int(round(len(p) / (n+1)))
    all_matches = set()
    p_idx = kmerindex.Index(t, segment_length)
    idx_hits = 0
    for i in range(n+1):
        start = i*segment_length
        end = min((i+1)*segment_length, len(p))
        matches = p_idx.query(p[start:end])

        for m in matches:
            idx_hits += 1
            if m < start or m-start+len(p) > len(t):
                continue

            mismatches = 0

            for j in range(0, start):
                if not p[j] == t[m-start+j]:
                    mismatches += 1
                    if mismatches > n:
                        break
            for j in range(end, len(p)):
                if not p[j] == t[m-start+j]:
                    mismatches += 1
                    if mismatches > n:
                        break

            if mismatches <= n:
                all_matches.add(m - start)
    return list(all_matches), idx_hits


def approximate_match_subseq(p, t, n, ival):
    segment_length = int(round(len(p) / (n+1)))
    all_matches = set()
    p_idx = subseq.SubseqIndex(t, segment_length, ival)
    idx_hits = 0
    for i in range(n+1):
        start = i
        matches = p_idx.query(p[start:])

        for m in matches:
            idx_hits += 1
            if m < start or m-start+len(p) > len(t):
                continue

            mismatches = 0
```

```
            for j in range(0, len(p)):
                if not p[j] == t[m-start+j]:
                    mismatches += 1
                    if mismatches > n:
                        break

            if mismatches <= n:
                all_matches.add(m - start)
    return list(all_matches), idx_hits



def main():
    chr1 = readGenome('chr1.GRCh38.excerpt (1).fasta')
    p = 'GGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGG'
    p_bm = bm_preproc.BoyerMoore(p)
    # Question 1
    print(naive(p, chr1)[2])
    # Question 2
    print(naive(p, chr1)[1])
    # Question 3
    print(boyer_moore(p, p_bm, chr1)[2])
    p = 'GGCGCGGTGGCTCACGCCTGTAAT'
    # Question 4
    print(len(approximate_match(p, chr1, 2)[0]))
    # Question 5
    print(approximate_match(p, chr1, 2)[1])
    # Question 6
    print(approximate_match_subseq(p, chr1, 2, 3)[1])



if __name__ == '__main__':
    main()
```

```
799954
984143
127974
19
90
79
```

In [ ]: