String matching :

Problem : We have a string T, Text and another string P, pattern and we have to find out the occurrence of P in T. If there are several occurrences then we might have to return one or all the position where P starts in T.

For example : if T = ACGTGCTACGAT and P = TGCT. Then P occur from position 4 to position 7.

There are two type of string matching problems in algorithm.

1) Exact String Matching.
2) Approximate String Matching.

In exact string matching problem, we see whether the string P is a part of T and in approximate string matching, we find strings in T which matches with String P approximately.

Some Algorithms for String Matching Problem :

1) Naive String matching : In naive String matching, We take the pattern P, (let's say of length m) and start matching with the Text, T (say length n) starting from 1st position in the pattern. If the P matches with some part of the String in T, then we return the position of P in T, if it does not matches with any string in T, then we move P by one position in T and again start matching P with T starting from position 1. This is the most basic technique used. The time taken is O(m*(n-m)).

2) Knuth Morris Pratt Algorithm : This algorithm uses an array which is known as prefix table. This table is used to preprocess the pattern P.The ith entry in the prefix table is the length of proper prefix of P[0..i] which matches the suffix of P[0..i]. This algorithm is different from Naive String Matching because instead moving the pattern from one position, we can move the pattern by maximum amount we want to. By rigorous proof, we can analyze that one word is compared at most 2 times, so after preprocessing the prefix table, the time taken is O(2*n). The total time complexity of the algorithm is O(m + 2*n).

   ('m' is length of pattern and 'n' is length of text)

3) Rabin-Carp Algorithm : In this method, we use the hashing technique and instead of checking for each possible position in T, we check only if the hashing of P and hashing of ‘m’ characters of T gives the same result. Initially, we apply hash function to first ‘m’ characters of T and check whether this result and P’s hashing result is same or not. If they are not same then go to next character of T and apply function to next ‘m’ characters. If they are same then we compare those ‘m’ characters of T with P. In this method, we have to find a good hash function so that algorithm takes very less time.

4) Boyce-Moore algorithm : This algorithm also need some preprocessing. The algorithm scans characters of the pattern from right to left beginning with the rightmost character. During Pre-processing, we find the maximum we can shift a character in the pattern. If somewhere the pattern does not match then we will shift the pattern by some amount described in our pre processed array. The time complexity of this algorithm is O(m+n).