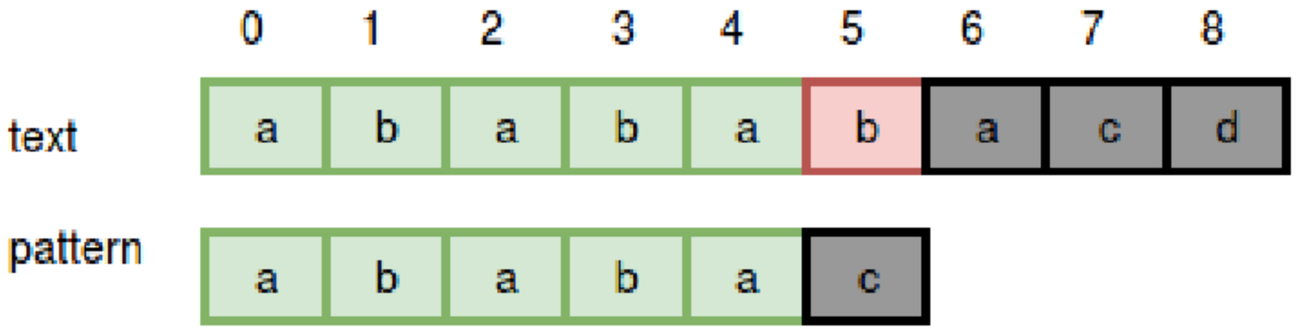
KMP & Z

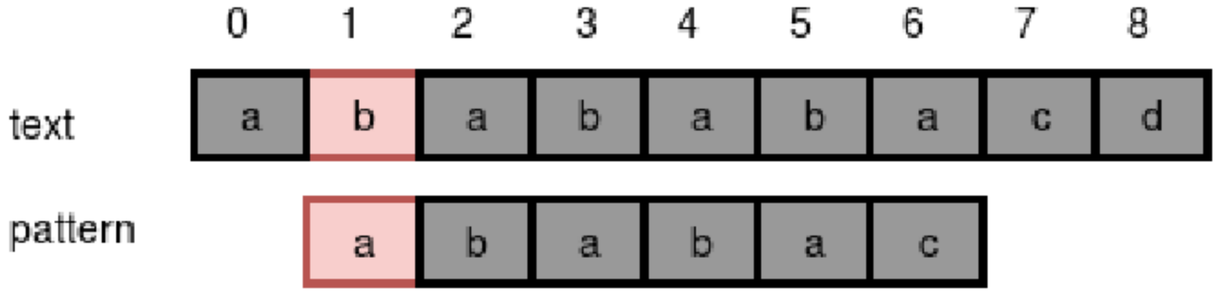
KMP Algorithm

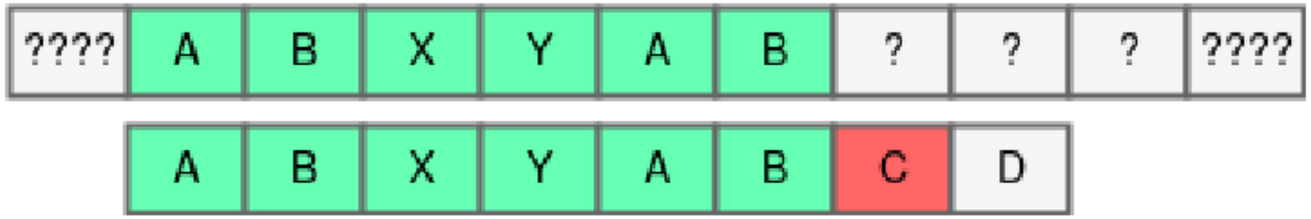
Calculate the indices of the occurrences of string s in t.

আমাদের প্রবলেম হলো একটা দুটি স্ট্রিং text এবং pattern দেয়া আছে, আমাদের বলতে হবে text এর ভিতর pattern স্ট্রিংটি সাবস্ট্রিং হিসাবে আছি কিনা। যেমন ধরো টেক্সটটি হলো "MOD", এই স্ট্রিংটার ৬টা সাবস্ট্রিং আছে "M", "O", "D", "MO", "OD" এবং "MOD", এখন যদি pattern = "MO" খুজতে বলে আমরা true রিটার্ন করবো।

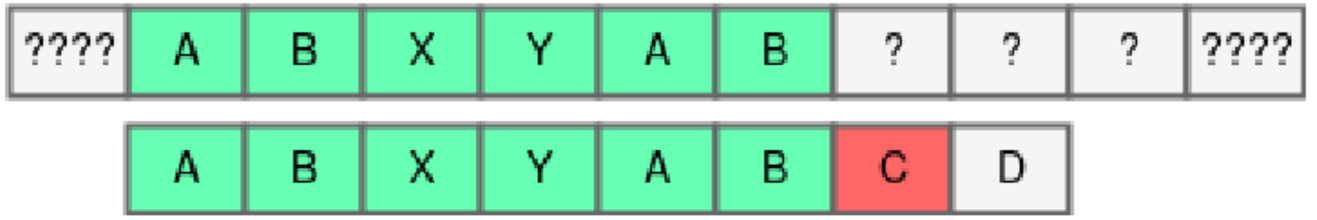
ব্রুটফোর্স অ্যালগরিদম ব্যবহার করে স্ট্রিং ম্যাচিং করার টাইম কমপ্লেক্সিটি $O(n * m)$, যেখানে n এবং m হলো টেক্সট ও প্যাটার্নের দৈর্ঘ্য। কিন্তু কেএমপি ব্যবহার করে $O(n + m)$ কমপ্লেক্সিটিতে প্যাটার্ন খুজে বের করা যায়।


0 1 2 3 4 5 6 7 8
text
a b a b a b a c d
pattern
a b a b a c

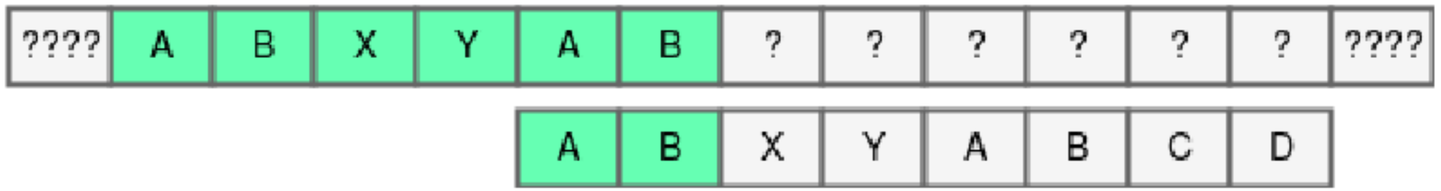

0 1 2 3 4 5 6 7 8
text
a b a b a b a c d
pattern
a b a b a c

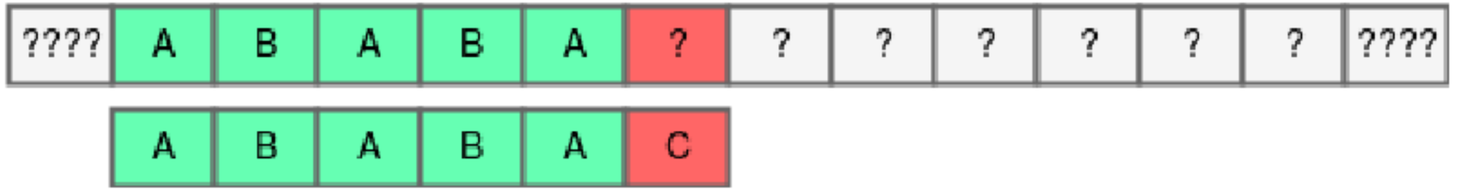

????
A
B
X
Y
A
B
?
?
?
????
A
B
X
Y
A
B
C
D

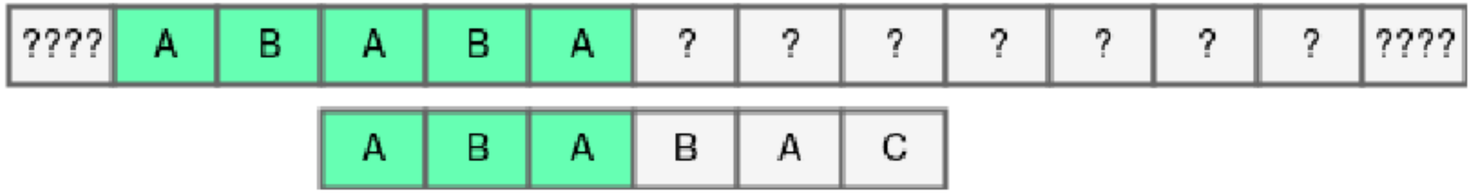

????
A
B
X
Y
A
B
?
?
?
????
A
B
X
Y
A
B
C
D

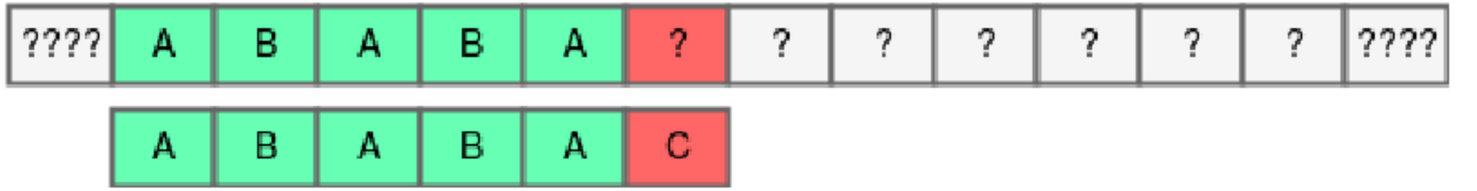

????
A
B
X
Y
A
B
?
?
?
????
A
B
X
Y
A
B
C
D

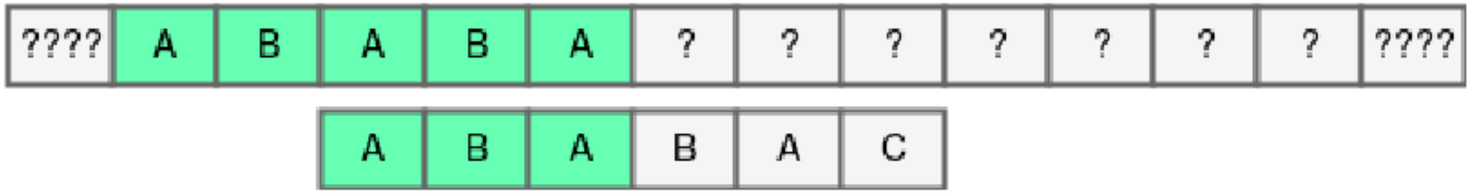

????
A
B
X
Y
A
B
?
?
?
????
A
B
X
Y
A
B
C
D

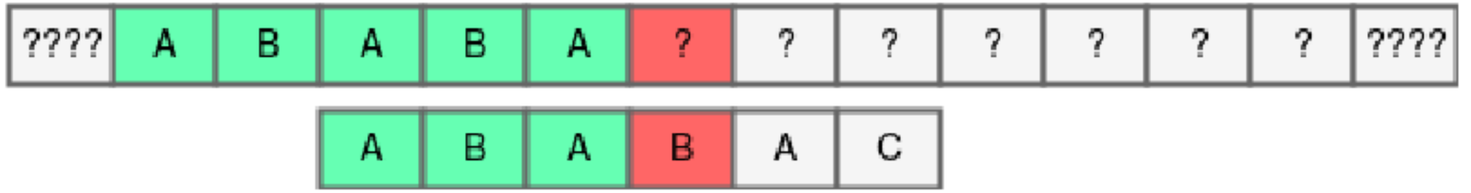

????
A
B
X
Y
A
B
?
?
?
?
?
?
????
A
B
X
Y
A
B
C
D

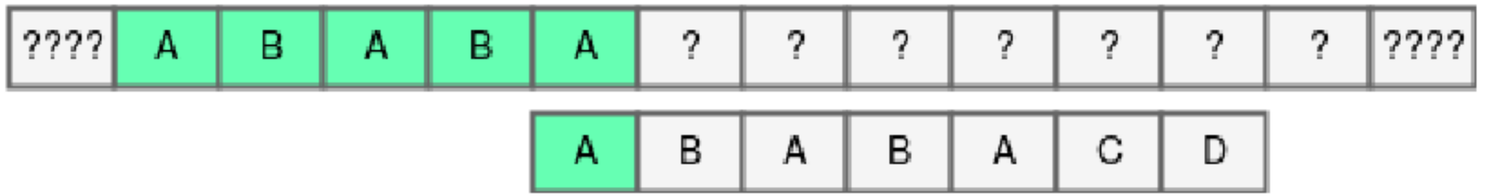

????
A
B
A
B
A
?
?
?
?
?
?
?
????
A
B
A
B
A
C

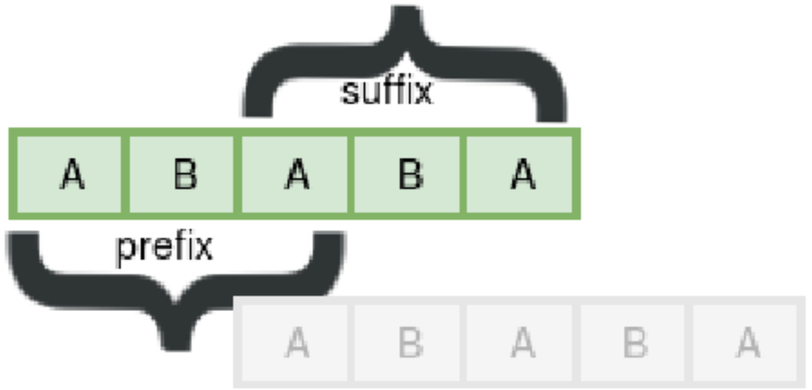

????
A
B
A
B
A
?
?
?
?
?
?
?
????
A
B
A
B
A
C


????
A
B
A
B
A
?
?
?
?
?
?
?
????
A
B
A
B
A
C


????
A
B
A
B
A
?
?
?
?
?
?
?
????
A
B
A
B
A
C


????
A
B
A
B
A
?
?
?
?
?
?
?
????
A
B
A
B
A
C


????
A
B
A
B
A
?
?
?
?
?
?
?
????
A
B
A
B
A
C
D



Longest prefix that is also a suffix

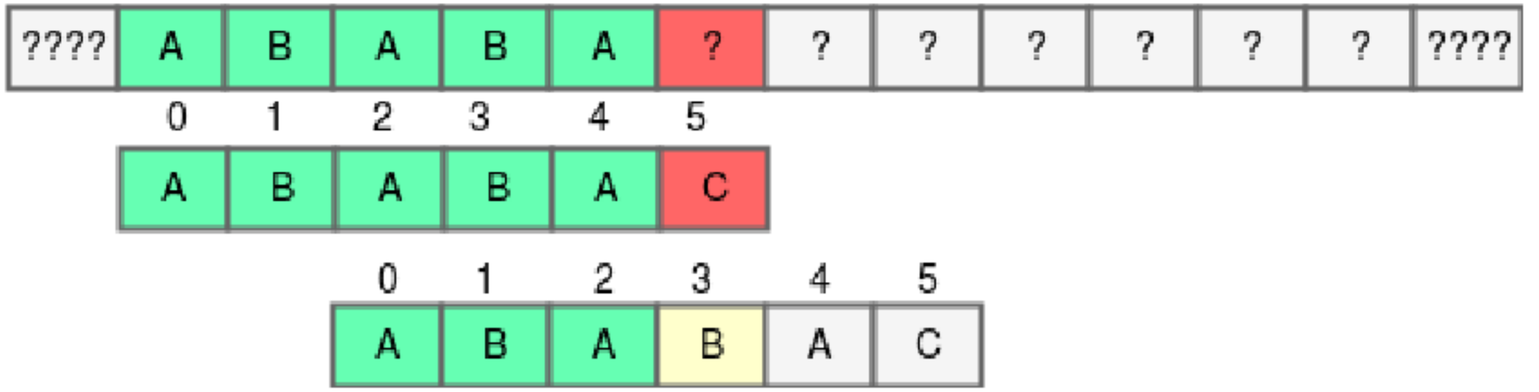
length	prefix	longest prefix length is also a suffix
0	""	0
1	A	0
2	AB	0
3	ABA	1
4	ABAB	2
5	ABABA	3
6	ABABAC	0

ABABAC

length	prefix	longest prefix length is also a suffix
0	""	0
1	A	0
2	AB	0
3	ABA	1
4	ABAB	2
5	ABABA	3
6	ABABAC	0

Failure Table or LPS array

A B A B A C



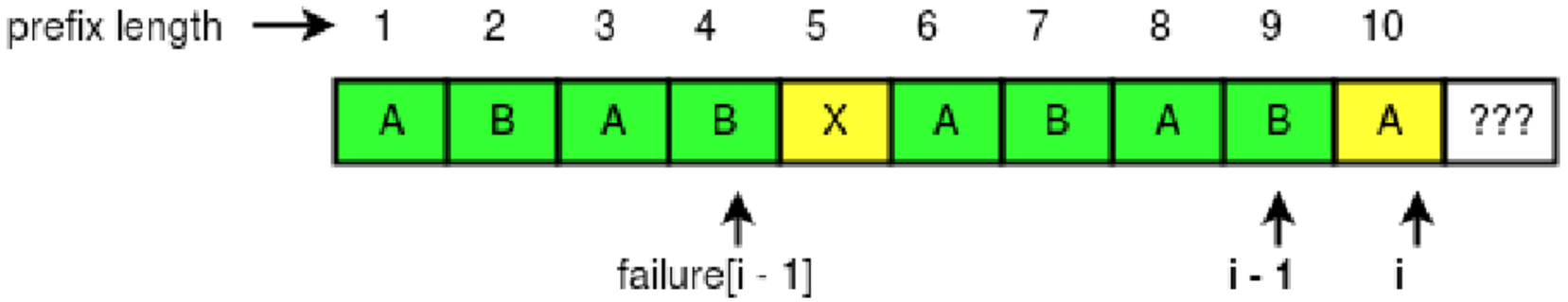
length	prefix	longest prefix length is also a suffix
0	""	0
1	A	0
2	AB	0
3	ABA	1
4	ABAB	2
5	ABABA	3
6	ABABAC	0

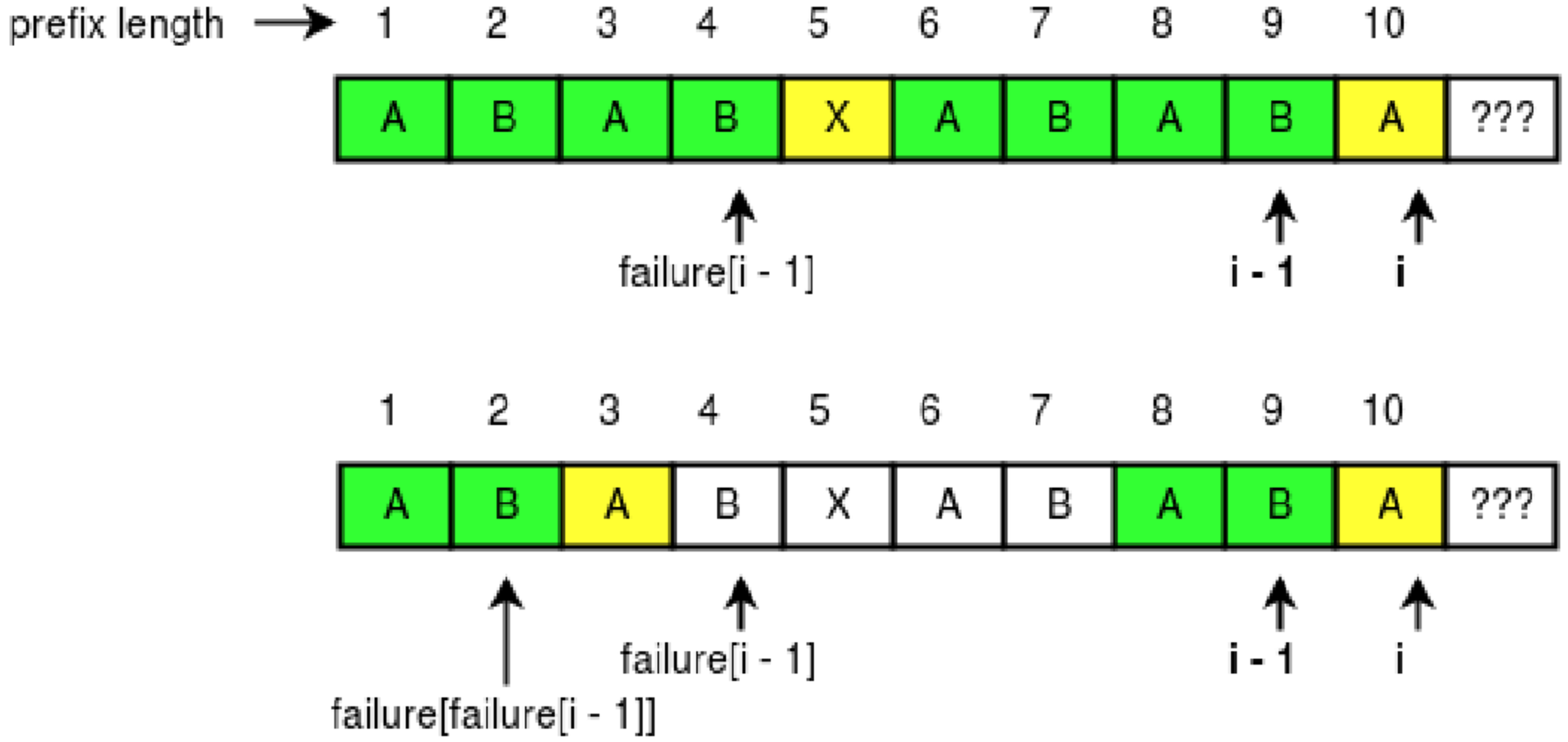
Failure Table or LPS array

ABABAC

$$failure[0] = 0$$
$$failure[1] = 0$$
$$failure[2] = 0$$
$$failure[3] = 1$$
$$failure[4] = 2$$
$$failure[5] = 3$$
$$failure[6] = 0$$

failure [i] = consider i length prefix


prefix length
1
2
3
4
5
6
7
8
9
10
A
B
A
B
X
A
B
A
B
A
???
failure[i - 1]
i - 1
i


prefix length
1 2 3 4 5 6 7 8 9 10
A B A B X A B A B A ???
failure[i - 1]
i - 1
i
1 2 3 4 5 6 7 8 9 10
A B A B X A B A B A ???
failure[failure[i - 1]]
failure[i - 1]
i - 1
i

```
// returns the longest proper prefix array of pattern p
// where lps[i]=longest proper prefix which is also suffix of p[0...i]
vector<int> build_lps(string p) {
  int sz = p.size();
  vector<int> lps;
  lps.assign(sz + 1, 0);
  int j = 0;
  lps[0] = 0;
  for(int i = 1; i < sz; i++) {
    while(j >= 0 && p[i] != p[j]) {
      if(j >= 1) j = lps[j - 1];
      else j = -1;
    }
    j++;
    lps[i] = j;
  }
  return lps;
}
```

```
vector<int>ans;
// returns matches in vector ans in 0-indexed
void kmp(vector<int> lps, string s, string p) {
  int psz = p.size(), sz = s.size();
  int j = 0;
  for(int i = 0; i < sz; i++) {
    while(j >= 0 && p[j] != s[i])
      if(j >= 1) j = lps[j - 1];
      else j = -1;
    j++;
    if(j == psz) {
      j = lps[j - 1];
      // pattern found in string s at position i-psz+1
      ans.push_back(i - psz + 1);
    }
    // after each loop we have j=longest common suffix of s[0..i] which is also prefix of p
  }
}
```

Z Algorithm

z[i]=number of elements prefix such that suffix=prefix ; suffix starts from idx i
z[i]= i theke shuru kore prefixer sathe koyta mil

- "aaaaa" - $[0, 4, 3, 2, 1]$
- "aaabaab" - $[0, 2, 1, 0, 2, 1, 0]$
- "abacaba" - $[0, 0, 1, 0, 3, 0, 1]$

String - a b c d e a a b d f a a b d f g

Pattern - a a b b

New-String - a a b b # a b c d e a a b b t a a b d f g

Z-array - 0 1 0 0 0 1 0 0 0 0 4 1 0 0 0 3 1 0 0 0

```
void zfunction(string &s) {
    ll n = s.size();
    z[0] = n;
    ll L = 0, R = 0;
    for (int i = 1; i < n; i++) {
        if (i > R) {
            L = R = i;
            while (R < n && s[R-L] == s[R]) R++;
            z[i] = R-L; R--;
        }
        else {
            int k = i-L;
            if (z[k] < R-i+1) z[i] = z[k];
            else {
                L = i;
                while (R < n && s[R-L] == s[R]) R++;
                    z[i] = R-L; R--;
                }
        }
    }
}
```