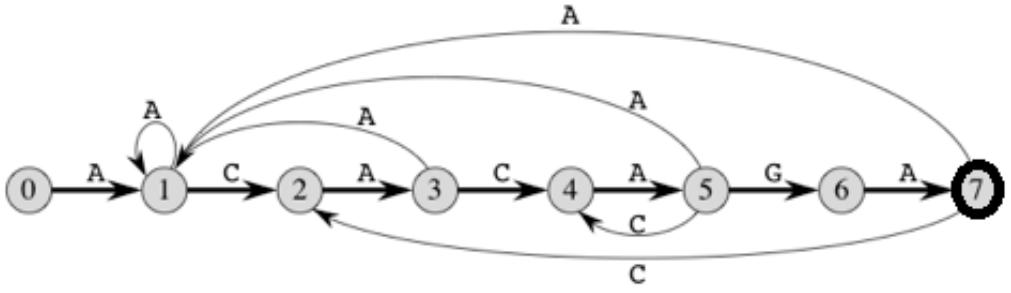
Pattern Searching | Set 6 (Efficient Construction of Finite Automata) - GeeksforGeeks

In the previous post, we discussed Finite Automata based pattern searching algorithm. The FA (Finite Automata) construction method discussed in previous post takes O((m^3)*NO_OF_CHARS) time. FA can be constructed in O(m*NO_OF_CHARS) time. In this post, we will discuss the O(m*NO_OF_CHARS) algorithm for FA construction. The idea is similar to lps (longest prefix suffix) array construction discussed in the KMP algorithm. We use previously filled rows to fill a new row.



state	character A	C	G	T
0	1	0	0	0
1	1	2	0	0
2	3	0	0	0
3	1	4	0	0
4	5	0	0	0
5	1	4	6	0
6	7	0	0	0
7	1	2	0	0

The abvoe diagrams represent graphical and tabular representations of pattern ACACAGA.

Algorithm:

1) Fill the first row. All entries in first row are always 0 except the entry for pat[0] character. For pat[0] character, we always need to go to state 1.

2) Initialize lps as 0. lps for the first index is always 0.

3) Do following for rows at index i = 1 to M. (M is the length of the pattern)

…..a) Copy the entries from the row at index equal to lps.

…..b) Update the entry for pat[i] character to i+1.

…..c) Update lps “lps = TF[lps][pat[i]]” where TF is the 2D array which is being constructed.

Implementation

Following is C implementation for the above algorithm.

```
#include<stdio.h>
#include<string.h>
#define NO_OF_CHARS 256

/* This function builds the TF table which represents Finite Automata for
a
   given pattern  */
void computeTransFun(char *pat, int M, int TF[][NO_OF_CHARS])
{
    int i, lps = 0, x;

    // Fill entries in first row
    for (x =0; x < NO_OF_CHARS; x++)
       TF[0][x] = 0;
    TF[0][pat[0]] = 1;

    // Fill entries in other rows
    for (i = 1; i<= M; i++)
    {
        // Copy values from row at index lps
        for (x = 0; x < NO_OF_CHARS; x++)
            TF[i][x] = TF[lps][x];

        // Update the entry corresponding to this character
        TF[i][pat[i]] = i + 1;

        // Update lps for next row to be filled
        if (i < M)
          lps = TF[lps][pat[i]];
    }
}

/* Prints all occurrences of pat in txt */
void search(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    int TF[M+1][NO_OF_CHARS];

    computeTransFun(pat, M, TF);

    // process text over FA.
    int i, j=0;
```

```
    for (i = 0; i < N; i++)
    {
       j = TF[j][txt[i]];
       if (j == M)
       {
         printf ("\n pattern found at index %d", i-M+1);
       }
    }
}

/* Driver program to test above function */
int main()
{
    char *txt = "GEEKS FOR GEEKS";
    char *pat = "GEEKS";
    search(pat, txt);
    getchar();
    return 0;
}
```

Output:

```
 pattern found at index 0
 pattern found at index 10
```

Time Complexity for FA construction is O(M*NO_OF_CHARS). The code for search is same as the previous post and time complexity for it is O(n).

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.