Tutorial on Suffix Arrays

AUTH
hackerearth

A Suffix Array is a sorted array of all suffixes of a string. For instance, all the suffixes of the string "hacker" are:

```
0 hacker
1 acker
2 cker
3 ker
4 er
5 r
```

Now, a suffix array of the above suffixes would look like:

```
1 acker
2 cker
4 er
0 hacker
3 ker
5 r
```

Note that the suffix array will contain integers that represent the starting indexes of the all the suffixes of a given string. A naive approach of generating all suffixes and sorting them to construct a Suffix Array seems simple, but it is not efficient enough. In this method, sorting the suffixes of an array of size N will take $O(NlogN)$ time. Since comparing two suffixes is done in $O(N)$, the final time complexity will reach $O(N^2logN)$.

As a first step towards building Suffix Array, a slightly efficient algorithm is discussed below.

Algorithm:

The algorithm is mainly based on maintaining the order of the string's suffixes sorted by their 2^k long prefixes. Execute $m = [log_2 n]$ (ceil) steps, computing the order of the prefixes of length 2^k at the k^{th} step. It is used an $m \times n$ sized matrix. Let's denote by A_i^k the subsequence of A of length 2^k starting at position i. The position of A_i^k in the sorted array of A_j^k subsequences $(j = 1, n)$ is kept in $P(k, i)$.

When passing from step k to step $k + 1$, all the pairs of subsequences A_i^k and $A_{i+2^k}^k$ are concatenated, therefore obtaining the substrings of length 2^{k+1}. For establishing the new order relation among these, the information computed at the previous step must be used. For each index i it is kept a pair formed by $P(k, i)$ and $P(k, i + 2^k)$. After sorting, the pairs will be arranged conforming to the lexicographic order of the strings of length $2^k + 1$. One last thing that must be remembered is that at a certain step k there may be several substrings $A_i^k = A_j^k$, and these must be labeled identically ($P(k, i)$ must be equal to $P(k, j)$).

Implementation: (source)

```
#include <bits/stdc++.h>
using namespace std;

#define MAXN 65536
#define MAXLG 17

char A[MAXN];
struct entry {
    int nr[2], p;
} L[MAXN];

int P[MAXLG][MAXN], N, i, stp, cnt;

int cmp(struct entry a, struct entry b)
{
    return a.nr[0] == b.nr[0] ? (a.nr[1] < b.nr[1] ? 1 : 0)
}
int main(void)
{
    gets(A);
    for (N = strlen(A), i = 0; i < N; i ++)
    {
        P[0][i] = A[i] - 'a';
    }
    for (stp = 1, cnt = 1; cnt >> 1 < N; stp ++, cnt <<= 1
    {
```

```
        for (i = 0; i < N; i ++)
        {
            L[i].nr[0] = P[stp - 1][i];
            L[i].nr[1] = i + cnt < N ? P[stp - 1][i + cnt]
            L[i].p = i;
        }
        sort(L, L + N, cmp);
        for (i = 0; i < N; i ++)
        {
            if(i > 0 && L[i].nr[0] == L[i - 1].nr[0] && L[i
            {
                P[stp][L[i].p] = P[stp][L[i - 1].p];
            }
            else
            {
                P[stp][L[i].p] = i;
            }
        }
    }
    return 0;
}
```

The suffix array will be found on the last row of matrix P. Searching the k^{th} suffix is now immediate, so we won't return to this aspect. The quantity of memory used may be reduced, using only the last two lines of the matrix P. It is a trade-off, as in this case the structure will not be able to execute efficiently the following operation.

A more efficient algorithm wil be discussed in this space in coming says. Till then, try and solve the question beloew with a brute force approach.

TEST YOUR UNDERSTANDING

Suffix Array - Substring Occurrences

Given a string S and string P, find all the indices of occurrence of T in S. Follow 0-based indexing of string.

Input:
First line contains string S.
Second line contains string T.

Output:
Print all the indices separated by space.

Constraints:
$1 \leq |S| \leq 1000$
$1 \leq |T| \leq 100$

SAMPLE INPUT

```
cincinnati
```

```
in
```

SAMPLE OUTPUT

```
1 4
```

Enter your code or Upload your code as file.

All changes saved

Visual Basic (mono vbnc 4.0.1)

```
Loading...
```

1:1

COMPILE & RUN

SUBMIT

Press ctrl-space for autocomplete suggestions.

POWERED BY codetable

ABOUT US

- Blog
- Engineering Blog
- Updates & Releases
- Team
- Careers
- In the Press

HACKEREARTH

- API
- Chrome Extension
- CodeTable
- HackerEarth Academy
- Developer Profile
- Resume
- Get Badges

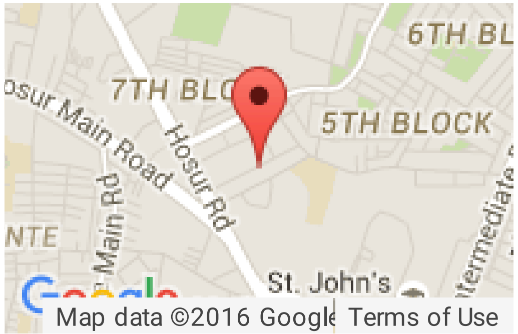
DEVELOPERS

- AMA
- Code Monk
- Judge Environment
- Solution Guide
- Problem Setter Guide
- Practice Problems
- HackerEarth Challenges

EMPLOYERS

- Developer Sourcing
- Lateral Hiring
- Campus Hiring
- Hackathons
- FAQs
- Customers

REACH US



IIIrd Floor,
Salarpuria Business Center,
4th B Cross Road,
5th A Block,
Koramangala Industrial Layout,
Bangalore,
Karnataka 560095,
India.

Challenges

Campus Ambassadors

Get Me Hired

Privacy

Terms of Service

College Challenges

College Ranking

Organise Hackathon

Hackathon Handbook

Competitive Programming

Open Source

contact@hackerear

+91-80-4155-4695

+1-650-461-4192