**Problem Statement:**

Perform Burrows-Wheeler transform or block-sorting compression on the below-mentioned sentence.

“Curneu MedTech Innovation is a health care technology firm based at Heidelberg, Germany. We work on a motive of building affordable and innovative healthcare solutions that address the clinical needs thereby bringing better lives for the needy.”

Neat Documentation is expected with from-scratch implementation with C++ and output characters.

**Solution:**

**Abstract:**

The most important application of BWT is found in biological sciences where genomes(long strings written in A, C, T, G alphabets) don’t have many runs but they do have many repeats. The idea of the BWT is to build an array whose rows are all cyclic shifts of the input string in dictionary order and return the last column of the array that tends to have long runs of identical characters. The benefit of this is that once the characters have been clustered together, they effectively have an ordering, which can make our string more compressible for other algorithms like run-length encoding and Huffman Coding. The remarkable thing about BWT is that this particular transform is reversible with minimal data overhead.

**Introduction:**

Michael Burrows and David Wheeler recently released the details of a transformation function that opens the door to some revolutionary new data compression techniques. The Burrows-Wheeler Transform, or BWT, transforms a block of data into a format that is extremely well suited for compression. It does such a good job at this that even the simple demonstration programs I’ll present here will outperform state-of-the-art programs.

**The Burrows-Wheeler Transform**

Michael Burrows and David Wheeler released a research report in 1994 discussing work they had been doing at the Digital Systems Research Center in Palo Alto, California. Their paper, “A Block-sorting Lossless Data Compression Algorithm” presented a data compression algorithm based on a previously unpublished transformation discovered by Wheeler in 1983.

While the paper discusses a complete set of algorithms for compression and decompression, the real heart of the paper consists of the disclosure of the BWT algorithm.

The BWT is a data transformation algorithm that restructures data in such a way that the transformed message is more compressible. Technically, it is a lexicographical reversible permutation of the characters of a string. It is first of the three steps to be performed in succession while implementing the Burrows-Wheeler Data Compression algorithm that forms the basis of the Unix compression utility bzip2.

**Algorithm :**

The Burrows-Wheeler transform (BWT) is not actually a compression algorithm. In fact BWT requires that a small amount of additional information be stored with the transformed data so that the transformation may be reversed. This makes the transformed data larger than its original form.

BWT is useful because it converts the data into a format that is generally more compressible by run length encoders and statistical encoders with order greater than 0. By additionally applying move-to-front coding, the data will be in a format which is generally more compressible by even zero order statistical encoders such as traditional implementations of Huffman coding or arithmetic coding.

steps involved in BWT algorithm

Let’s take the word “banana$” as an example.

•Step 1:Form all cyclic rotations of the given text banana$

$ b $banana

a a a$banan

Cyclic rotations ----------> na$bana

n n ana$ban

a nana$ba

anana$b

Step 2: The next step is to sort the rotations lexicographically. The ‘$’ sign is viewed as first letter lexicographically, even before ‘a’.

$banana a$banan

a$banan Sorting ana$ban

na$bana ----------> anana$b

ana$ban alphabetically banana$

nana$ba na$bana

anana$b nana$ba

Step 3:The last column is what we output as BWT.

BWT(banana$) = annb$aa

**Example**:

Input: text = “abracadabra$”

Output: Burrows-Wheeler Transform = “ard$rcaaaabb”

**Coding:**

#include <iostream>

#include <stdlib.h>

#include <string.h>

// Structure to store data of a rotation

struct rotation {

int index;

char\* suffix;

};

// Compares the rotations and sorts the rotations alphabetically

int cmpfunc(const void\* x, const void\* y)

{

struct rotation\* rx = (struct rotation\*)x;

struct rotation\* ry = (struct rotation\*)y;

return strcmp(rx->suffix, ry->suffix);

}

// Takes text to be transformed and its length as arguments and returns the corresponding suffix array

int\* computeSuffixArray(char\* input\_text, int len\_text)

{

// Array of structures to store rotations and their indexes

struct rotation suff[len\_text];

// Structure is needed to maintain old indexes of

// rotations after sorting them

for (int i = 0; i < len\_text; i++) {

suff[i].index = i;

suff[i].suffix = (input\_text + i);

}

// Sorts rotations using comparison function defined above

qsort(suff, len\_text, sizeof(struct rotation),cmpfunc);

// Stores the indexes of sorted rotations

int\* suffix\_arr = (int\*)malloc(len\_text \* sizeof(int));

for (int i = 0; i < len\_text; i++)

suffix\_arr[i] = suff[i].index;

// Returns the computed suffix array

return suffix\_arr;

}

// Takes suffix array and its size as arguments and returns the Burrows - Wheeler Transform of given text

char\* findLastChar(char\* input\_text,int\* suffix\_arr, int n)

{

// Iterates over the suffix array to find

// the last char of each cyclic rotation

char\* bwt\_arr = (char\*)malloc(n \* sizeof(char));

int i;

for (i = 0; i < n; i++) {

// Computes the last char which is given by input\_text[(suffix\_arr[i] + n - 1) % n]

int j = suffix\_arr[i] - 1;

if (j < 0)

j = j + n;

bwt\_arr[i] = input\_text[j];

}

bwt\_arr[i] = '\0';

// Returns the computed Burrows - Wheeler Transform

return bwt\_arr;

}

// Driver program to test functions above

int main()

{

char input\_text[1000] ;

std::cin.get(input\_text,1000);

int len\_text = strlen(input\_text);

// Computes the suffix array of our text

int\* suffix\_arr = computeSuffixArray(input\_text, len\_text);

// Adds to the output array the last char of each rotation

char\* bwt\_arr = findLastChar(input\_text, suffix\_arr, len\_text);

std::cout <<"Input text :"<< input\_text<<"\n";

std::cout <<"Burrows - Wheeler Transform :"<<bwt\_arr<<"\n";

return 0;

}

**Output :**

