Strings.

Material de referencia.

Índice

1. Arreglo Z.

Consideremos un string s. El arreglo Z de s es un arreglo de enteros que guarda la longitud del substring más largo que empieza en la posición i y es prefijo de s.

En particular, podemos encontrar todas las ocurrencias de un patrón P en un texto T calculando el arreglo Z del string $P + \diamond + T$, donde $\diamond$ es algún carácter que no aparece ni en P ni en T.

1.1. Implementación

Complejidad: $O(|s|)$.

```
#include <iostream>
#include <algorithm>
using namespace std;

#define maxn 100000 //Longitud maxima de los strings.

string text, pattern, str; //Texto, patron a buscar y string auxiliar.
int Z[maxn];               //Arreglo Z.

//Construye el arreglo Z de str.
void buildZ() {
    int l = 0, r = 0;
    for (int i = 1; i < str.size(); ++i) {
        Z[i] = 0;
        if (i <= r)
            Z[i] = min(r - i + 1, Z[i - l]);
        while (i + Z[i] < str.size() && str[Z[i]] == str[i + Z[i]])
            ++Z[i];
        if (i + Z[i] - 1 > r) {
            l = i;
            r = i + Z[i] - 1;
        }
    }
}

int main() {
    ios_base::sync_with_stdio(0); cin.tie();
    getline(cin, text);
    cin >> pattern;
    str = pattern + '$' + text;

    //Imprime todas las ocurrencias.
    buildZ();
    for (int i = 0; i < text.size(); ++i)
        if (Z[i + pattern.size() + 1] == pattern.size())
            cout << "Patron encontrado en la posicion " << i << '\n';

    return 0;
}
```

Entrada	Salida
AABAACAADAABAABA AABA	Patron encontrado en la posicion 0 Patron encontrado en la posicion 9 Patron encontrado en la posicion 12

2. Arreglo de sufijos.

Consideremos un string s. El arreglo de sufijos de s es un arreglo de enteros que guarda las posiciones iniciales de los sufijos de s en orden lexicográfico.

2.1. Implementación

Complejidad: $O(|s| \log |s|)$.

```
#include <iostream>
#include <algorithm>
using namespace std;

#define maxn 100000 //Longitud maxima del string.

string word;                //String.
int n, SuffixArray[maxn]; //Arreglo de sufijos.

int rnk[maxn][2], bucket[maxn];     //Rango (SuffixArray) y Cubeta (RaxixSort).
int tempSA[maxn], tempRA[maxn][2]; //Arreglos temporales.

//Ordena de acuerdo a los rangos.
void RadixSort() {
    int M = max(n, 256);
    for (int k = 1; k >= 0; --k) {
        fill_n(bucket, M, 0);

        for (int i = 0; i < n; ++i)
            ++bucket[rnk[i][k]];
        for (int i = 1; i < M; ++i)
            bucket[i] += bucket[i - 1];

        for (int i = n - 1; i >= 0; --i) {
            int nxt_id = --bucket[rnk[i][k]];
            tempSA[nxt_id] = SuffixArray[i];
            tempRA[nxt_id][0] = rnk[i][0];
            tempRA[nxt_id][1] = rnk[i][1];
        }
        for (int i = 0; i < n; ++i) {
            SuffixArray[i] = tempSA[i];
            rnk[i][0] = tempRA[i][0];
            rnk[i][1] = tempRA[i][1];
        }
    }
}

//Construye el arreglo de sufijos.
void buildSA() {
    n = word.size();

    for (int i = 0; i < n; ++i) {
        SuffixArray[i] = i;
        rnk[i][0] = word[i];
    }
    RadixSort();

```

```
48      for (int k = 1; k < n; k *= 2) {
49          int curr = 0, prev = rnk[0][0];
50          rnk[0][0] = curr;
51          tempSA[SuffixArray[0]] = 0;
52
53          for (int i = 1; i < n; ++i) {
54              if (rnk[i][0] != prev || rnk[i][1] != rnk[i - 1][1])
55                  ++curr;
56              prev = rnk[i][0];
57              rnk[i][0] = curr;
58              tempSA[SuffixArray[i]] = i;
59          }
60
61          for (int i = 0; i < n; ++i) {
62              int nxt_id = SuffixArray[i] + k;
63              rnk[i][1] = (nxt_id < n) ? rnk[tempSA[nxt_id]][0] : 0;
64          }
65          RadixSort();
66      }
67  }
68
69  int main() {
70      ios_base::sync_with_stdio(0); cin.tie();
71      cin >> word;
72
73      buildSA();
74      for (int i = 0; i < n; ++i) {
75          cout << SuffixArray[i] << ' ';
76          for (int j = SuffixArray[i]; j < n; ++j)
77              cout << word[j];
78          cout << '\n';
79      }
80
81      return 0;
82  }
```

Entrada	**Salida**
banana	5 a 3 ana 1 anana 0 banana 4 na 2 nana