Algoritmo Boyer-Moore

Introducción

El **algoritmo de búsqueda de cadenas Boyer-Moore** es un eficiente algoritmo de búsqueda de cadenas. Fue desarrollado por Bob Boyer y J. Strother Moore en 1977. El algoritmo preprocesa la cadena que está siendo buscada, pero no en la cadena en que se busca:

- La comparación ahora se realiza de derecha a izquierda.
- Si hay una discrepancia en el último caracter del patrón y el caracter del texto no aparece en todo el patrón, entonces éste se puede deslizar *m* posiciones sin realizar ninguna comparación extra.
- No es necesario comparar los primeros m-1 caracteres del texto, lo cual indica que podría realizarse una búsqueda en el texto con menos de *n* comparaciones.
- Si el caracter discrepante del texto se encuentra dentro del patrón, éste podría desplazarse en un número menor de espacios.

Lógica

- Ante una discrepancia de un caracter, el caracter del texto se compara con el patrón de búsqueda para determinar el salto hacia la derecha.
- Si el caracter no existe en el patrón de búsqueda, se salta la cadena completa.
- Si la discrepancia se produce tras varias coincidencias, se salta en función de la repetición de patrones en la secuencia de búsqueda, y se alinea de nuevo usando ese valor.
- Se toma como salto el mayor de los dos valores.

Código C

```
void preBmBc(char *x, int m, int bmBc[]) {
  int i;


  for (i = 0; i < ASIZE; ++i)
```

```
    bmBc[i] = m;
  for (i = 0; i < m - 1; ++i)
    bmBc[x[i]] = m - i - 1;
}


void suffixes(char *x, int m, int *suff) {
  int f, g, i;

  suff[m - 1] = m;
  g = m - 1;
  for (i = m - 2; i >= 0; --i) {
    if (i > g && suff[i + m - 1 - f] < i - g)
      suff[i] = suff[i + m - 1 - f];
    else {
      if (i < g)
        g = i;
      f = i;
      while (g >= 0 && x[g] == x[g + m - 1 - f])
        --g;
      suff[i] = f - g;
    }
  }
}

void preBmGs(char *x, int m, int bmGs[]) {
  int i, j, suff[XSIZE];
```

```
  suffixes(x, m, suff);

  for (i = 0; i < m; ++i)
    bmGs[i] = m;
  j = 0;
  for (i = m - 1; i >= 0; --i)
    if (suff[i] == i + 1)
      for (; j < m - 1 - i; ++j)
        if (bmGs[j] == m)
          bmGs[j] = m - 1 - i;
  for (i = 0; i <= m - 2; ++i)
    bmGs[m - 1 - suff[i]] = m - 1 - i;
}


void BM(char *x, int m, char *y, int n) {
  int i, j, bmGs[XSIZE], bmBc[ASIZE];

  /* Preprocessing */
  preBmGs(x, m, bmGs);
  preBmBc(x, m, bmBc);

  /* Searching */
  j = 0;
  while (j <= n - m) {
    for (i = m - 1; i >= 0 && x[i] == y[i + j]; --i);
```

```
    if (i < 0) {
      OUTPUT(j);
      j += bmGs[0];
    }
    else
      j += MAX(bmGs[i], bmBc[y[i + j]] - m + 1 + i);
  }
}
```