# Strings

Wilmer Emiro Castrillón Calderón

17 de septiembre de 2022

## 1. KMP

El string matching es un problema clásico de las ciencias de la computación, consiste en buscar si una cadena de texto $T$ de longitud $n$ contiene una subcadena $P$ de longitud $m$, la solución ingenua consiste en deslizar la cadena $P$ sobre toda la cadena $T$, comparando carácter por carácter si encajan, es decir, se empieza verificando si la subcadena $T[0, n]$ es igual a $P$, en caso de fallo(caracteres diferentes) se compara $T[1, n+1]$ con $P$ y así sucesivamente, dando como resultado una complejidad de $O(n * m)$, la cual resulta muy alta cuando los cadenas de texto son muy largas.

El algoritmo KMP(Knuth-Morris-Pratt) permite encontrar todas las apariciones de $P$ en $T$ utilizando una tabla de precalculo $B$ sobre la cadena $P$, el algoritmo inicia con dos punteros, un puntero $i$ sobre $T$ y otro puntero $j$ sobre $P$, ambos avanzan a la par mientras se comparan $T[i]$ y $P[j]$, en caso de fallo, el cursor $j$ se devuelve a la posición indicada en la tabla de precalculo($B[j]$), de esta forma solo el cursor sobre $P$ se devuelve y se evita devolver el cursor sobre $T$. El algoritmo KMP se divide en dos partes, una primera para calcular la tabla de precalculo $B$ y una segunda para aplicar la búsqueda de $P$ sobre $T$.

La tabla de precalculo consiste en buscar la longitud de los "bordes" de $P$, un borde se define como un substring que es prefijo de $P$ y sufijo de un substring $P[0, k]$, entonces $B[k+1]$ es igual a la longitud del borde del substring $P[0, k]$, por ejemplo en la figura 1 se observa el borde para $P =$ "ABRACABRAABRA", se recomienda hacer el proceso a mano para que se pueda entender mejor.

| idx | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | R | A | C | A | B | R | A | A | B | R | A | |
| Borde: | -1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |

Figura 1

Para el proceso de buscar las repeticiones de $P$ sobre $T$ se necesitan dos punteros como se había mencionado antes, el puntero $i$ sobre $T$ y $j$ sobre $P$, observemos la figura 2, los punteros avanzan hasta llegar a al indice 9, donde hay un carácter de fallo, en este punto se puede observar la mejora que ofrece el KMP, pues solamente se devuelve $j$ a la posición $B[j]$ para seguir comparando, como se observa en la figura 3.

La implementacion no es larga y consiste en realizar primero la búsqueda de los bordes de $P$ y posteriormente hacer la búsqueda de $P$ sobre $T$. El algoritmo resultante tiene complejidad $O(n + m)$.

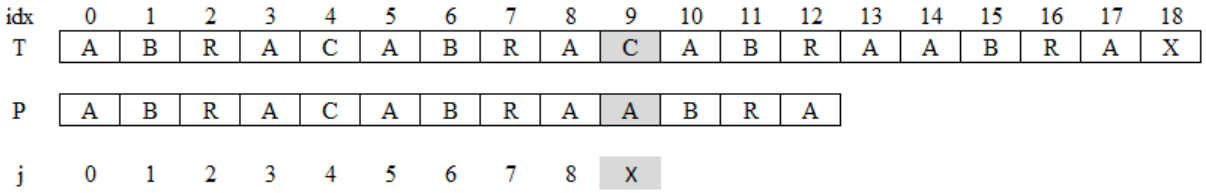


Figura 2

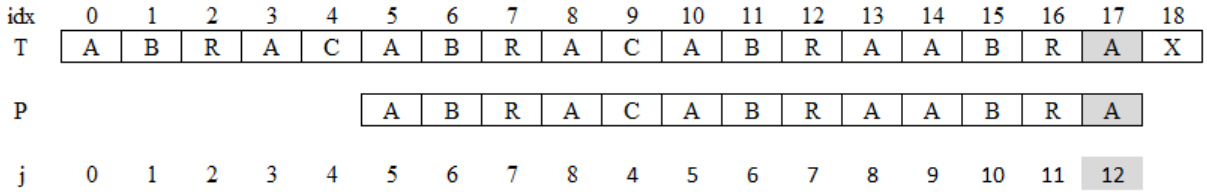


Figura 3

```
1  int B[MAX] = { -1 };
2
3  void bordes(string p) {
4    B[0] = -1; B[1] = 0;
5    int j = 0;
6    for(int i = 1; i < p.size(); i++) {
7      while(j >= 0 && p[j] != p[i]) j = B[j];
8      B[i + 1] = ++j;
9    }
10 }
11
12 void KMP(string t, string p) {
13   int j = 0;
14   for(int i = 0; i < t.size(); i++) {
15     while(j >= 0 && p[j] != t[i]) j = B[j];
16     j++;
17     if(j == p.size()) {
18       cout << "match en rango (" << (i - j + 1) << "," << i << ")\n";
19       j = B[j];
20     }
21   }
22 }
```

El KMP puede encontrar repeticiones con intervalos cruzados, por ejemplo con $P =$ "ABCABC" y $T =$ "DABCABCABCD", hay 2 repeticiones, en los rangos $(1, 6)$ y $(4, 9)$, en este caso se cruza el rango $(4, 6)$ el cual aparece ambas repeticiones. Si no se requieren intervalos cruzados se puede simplemente reiniciar el puntero $j$ a 0 al encontrar cada repetición (linea 19 del código), de esta manera se continuará buscando desde el inicio de $P$.