Estructura de Datos y Algoritmos

ITBA 2024-Q2

Algoritmos para texto

Hemos analizado algunos de los algoritmos que manejan "similitud" entre strings o textos. Es decir, algoritmos que no necesariamente buscan matching exacto.

Pero sobre el procesamiento de textos hay muchos más desafíos.

Por ejemplo: "búsqueda exacta"

Algoritmos para texto

Búsqueda exacta: ideal, por ejemplo para cuando realizamos una búsqueda (opcionalmente para realizar un reemplazo) en un editor de texto.

Ej: notepad, vi, etc.

Problema 1

Dado dos arreglos de chars (no strings por ahora) target y query, queremos un código Java que permita calcular la primera aparición de source en target, o -1 si no hay tal aparición.

Ej: target="abracadabra" query="ra" , entonces se obtiene 2
Ej: target="abracadabra" query="abra" , entonces se obtiene 0
Ej: target="abracadabra" query="aba" , entonces se obtiene -1

Java viene con indexOf para Strings. Pero antes de conocer qué estrategia siguieron, veamos varias existentes.

From scratch (no usar API)

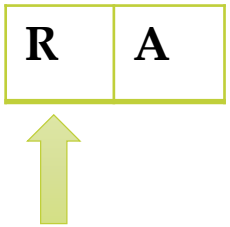
Lo invocaríamos así: ExactSearch.indexOf(char[] query, char[] target)

```
char[] target= "abracadabra".toCharArray();
char[] query= "ra".toCharArray();
System.out.println( ExactSearch.indexOf( query, target) );  //2

char[] target= "abracadabra".toCharArray();
char[] query= “abra".toCharArray();
System.out.println(ExactSearch.indexOf( query, target) );  //0

char[] target= "abracadabra".toCharArray();
char[] query= “aba".toCharArray();
System.out.println(ExactSearch.indexOf( query, target) );  //-1

char[] target= "ab".toCharArray();
char[] query= “aba".toCharArray();
System.out.println(ExactSearch.indexOf( query, target) );  //-1

char[] target= “xa".toCharArray();
char[] query= “aba".toCharArray();
System.out.println(ExactSearch.indexOf( query, target) );  //-1

char[] target= "abracadabras".toCharArray();
char[] query= “abras".toCharArray();
System.out.println(ExactSearch.indexOf( query, target) );  //7
```

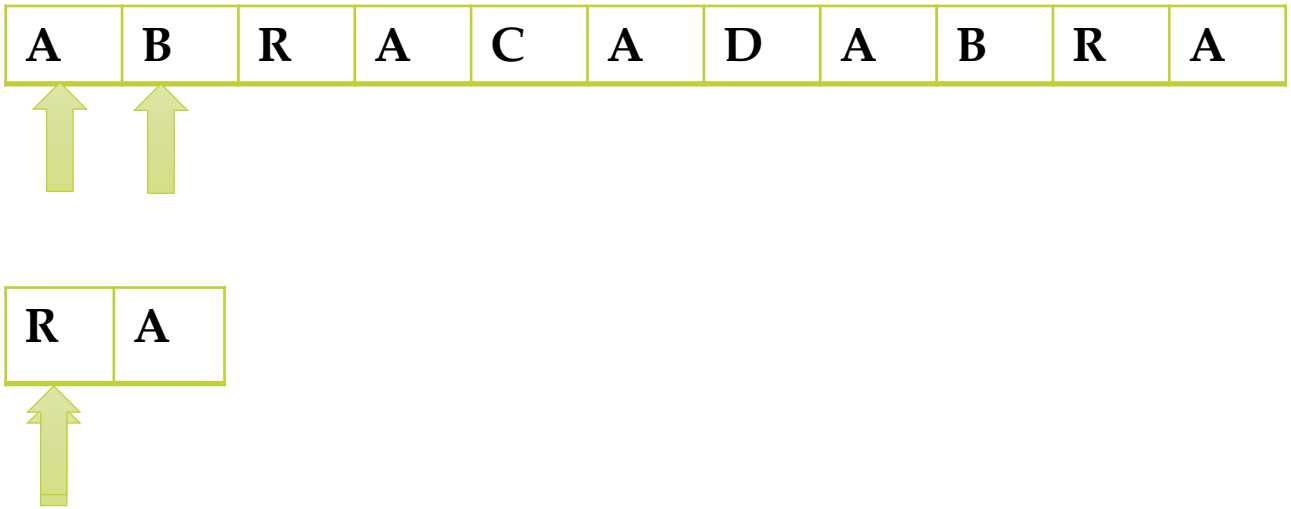
TP 2B- Ejer 1.1

Implementarlo (sin usar métodos Java, es un arreglo de chars!!!)

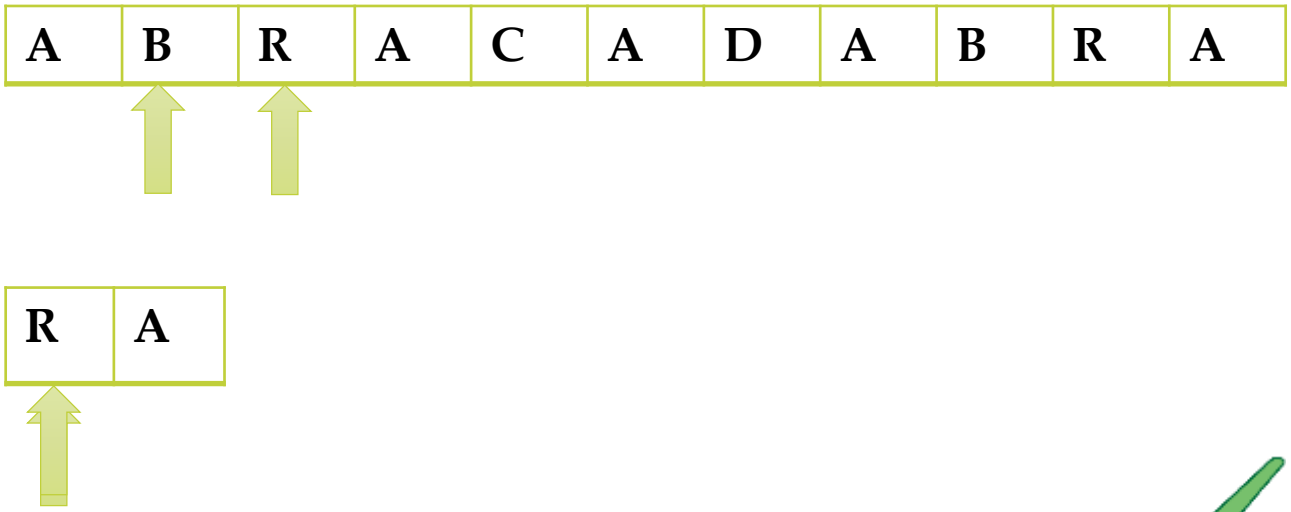
Algoritmo Fuerza Bruta o Naïve. Ej:



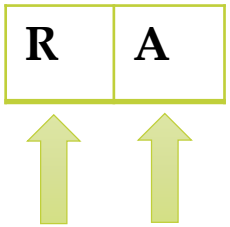
Algoritmo Fuerza Bruta o Naive



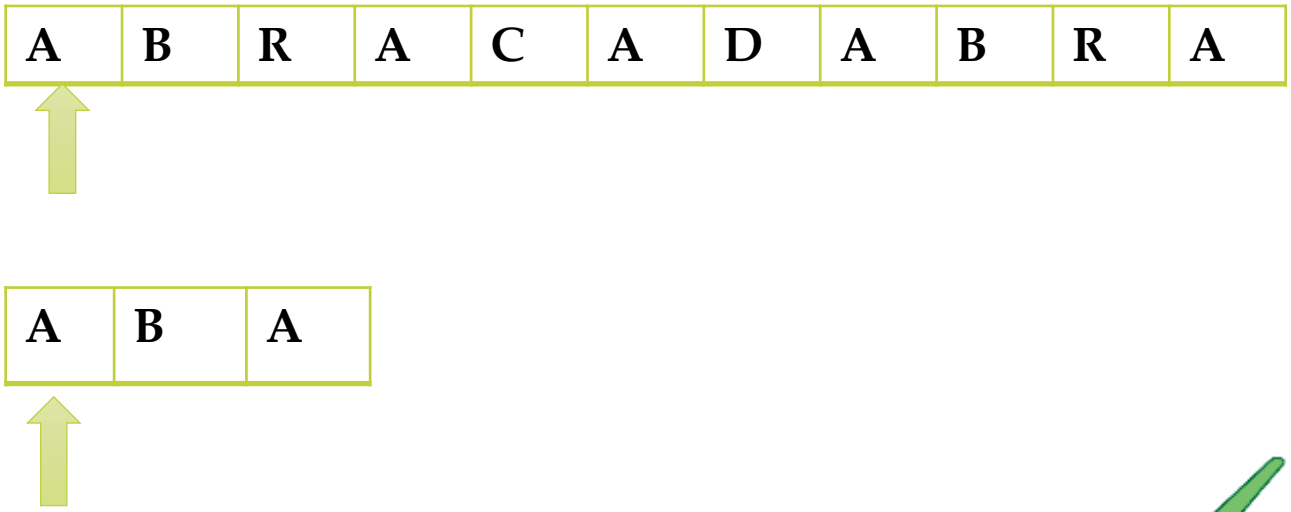
Algoritmo Fuerza Bruta o Naive



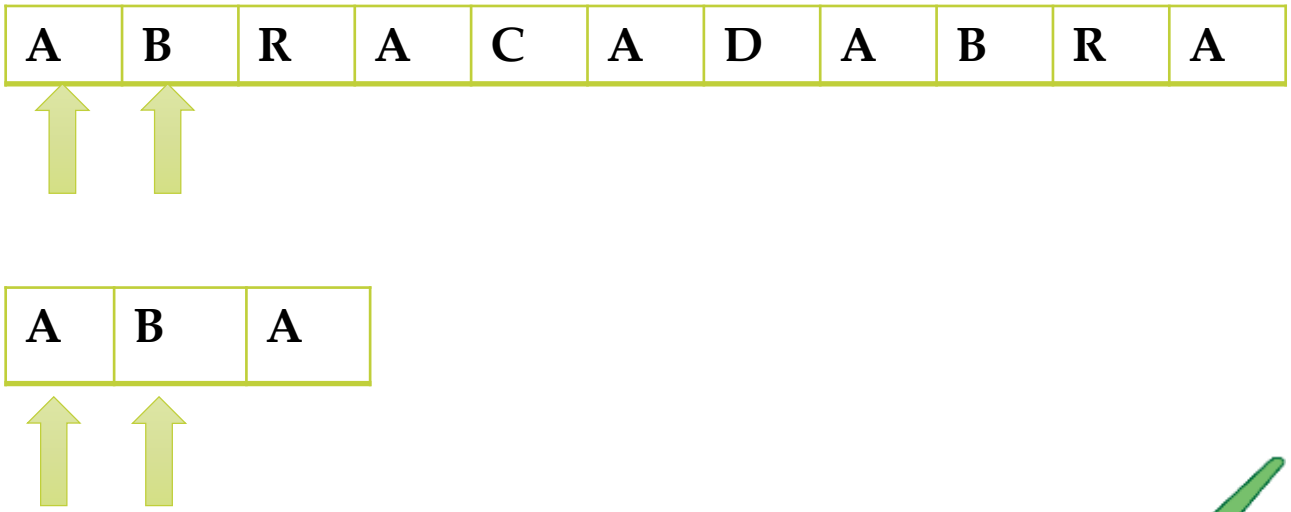
Algoritmo Fuerza Bruta o Naive



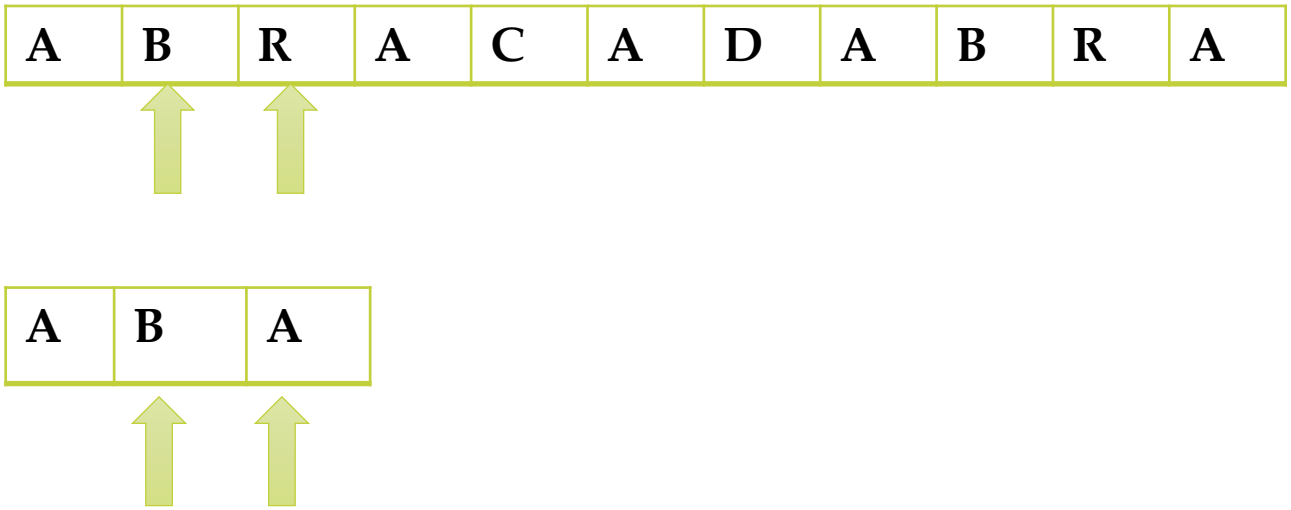
Algoritmo Fuerza Bruta o Naïve. Ej:



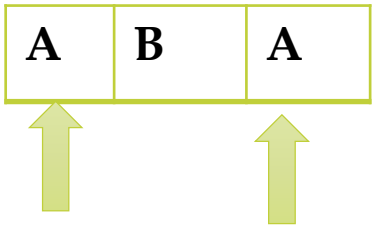
Algoritmo Fuerza Bruta o Naïve. Ej:



Algoritmo Fuerza Bruta o Naïve. Ej:



Algoritmo Fuerza Bruta o Naïve. Ej:



Y así continua…

Posible solución:

```
public static int indexOf(char[] query, char[] target)
{
        int idxTarget= 0;
        int idxQuery= 0;

        while(idxTarget < target.length &&  idxQuery < query.length)  {
                if (query[idxQuery] == target[idxTarget])  {
                        idxQuery++;
                        idxTarget++;
                }
                else  {
                        idxTarget= idxTarget - idxQuery + 1;
                        idxQuery = 0;
                }
        }

        if (idxQuery == query.length) // found!
                return idxTarget-idxQuery;
        return -1;
}
```

Otra posible implementación

```
public static int indexOf(char[] query, char[] target)
{
        int idxTarget= 0;
        int idxQuery 0;

        while(idxTarget < target.length &&  idxQuery < query.length) {
                if (query[idxQuery] == target[idxTarget])   {
                        idxQuery++;
                        idxTarget++;
                        if (idxQuery == query.length)
                                return idxTarget-idxQuery;
                }
                else  {
                        idxTarget= idxTarget - idxQuery + 1;
                        idxQuery = 0;
                }
        }
        return -1;
}
```

TP 2B- Ejer 1.2 y 1.3

Cuál es el peor caso?

Complejidad espacial?

Complejidad temporal?

El algoritmo anterior se denomina “naïve”.

¿Cuál es el peor caso? Que el query no se encuentre en el target

¿Complejidad temporal? $O(n * m)$ Sea $|target| = n$ y $|source| = m$

¿Complejidad espacial? $O(1)$

TP 2B- Ejer 1.4

Buscar cómo implementa Java el String.indexOf

Es el algoritmo naive?

¿Habrá otro algoritmo? ¿Mejor en lo temporal o en lo espacial?

El algortimo Naïve no aprovecha lo que aprendió durante el recorrido cuando encuentra un mismatch. Hace **backtracking en el query y en el target**.

El algoritmo Knuth-Morris-Pratt: no vuelve a chequear un caracter que ya sabe que matcheó! No hace backtracking en el target.

Primero la idea…

Algoritmo Knuth-Morris-Pratt.

- Scanea el target de izquierda a derecha, pero usa conocimiento sobre los caracteres comparados antes de determinar la próxima posición del patrón a usar.

- Preprocesa el query antes de la búsqueda una vez, con el objetivo de analizar la estructura (las características del patrón query). Para ello construye una tabla Next del mismo tamaño del query.

- La tabla de Next tiene en cada posición "i" la longitud del **borde propio** más grande para el substring query desde 0 hasta i.

Algoritmo Knuth-Morris-Pratt

Cálculo de Next

query	A	B	A
Next			

Next[2]?

Next[1]?

Next[0]

Algoritmo Knuth-Morris-Pratt

Ejercicio 1. Calcular el next en cada caso

query	X	E	E	E
Next				

query	B	C	B	C
Next				

query	A	B	X	A	B	U
Next						

Algoritmo Knuth-Morris-Pratt. Cálculo de Next

query	**A**	**B**	**R**	**A**	**C**	**A**	**D**	**A**	**B**	**R**	**A**
Next											

query	**S**	**A**	**S**	**S**
Next				

Revisando propiedades de los next

query	A	B	A
Next	0	0	1

query	B	C	B	C
Next	0	0	1	2

query	X	E	E	E
Next	0	0	0	0

query	A	B	X	A	B	U
Next	0	0	0	1	2	0

query	A	B	R	A	C	A	D	A	B	R	A
Next	0	0	0	1	0	1	0	1	2	3	4

N[i] <= N[i - 1] + 1 Pero puede ser que N[i]= 0

query	S	A	S	S
Next	0	0	1	1

Implementacion Original

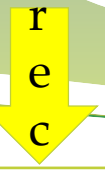
Queremos calcular Next para un cierto String.

Aprovechando la propiedad de Next

```
private static int[] nextComputation(char[] query) {

    int next[] = new int[query.length];
    next[0]= 0;     // Always. There's no proper border.
    int border = 0;  // Length of the current border

    for (int rec = 1; rec < query.length; rec++) {
        while ((border > 0) && (query[border] != query[rec]))
            border = next[border-1];     // Improving previous computation
        if (query[border] == query[rec])
            border++;
        // else border = 0;  // redundant
        next[rec]= border;
    }
    return next;
}
```

Otra forma

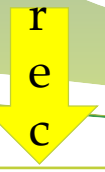
```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```



A	A	T	A	A	A	T	U
0							

Border= 0

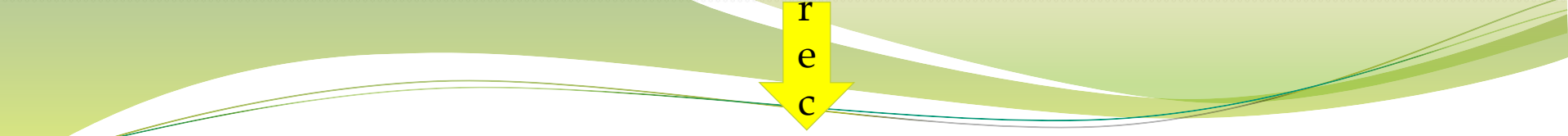
```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

    int border=0;  // Length of the current border

    int rec=1;
    while(rec < query.length){
        if(query[rec]!=query[border]){
            if(border!=0)
                border=next[border-1];
            else
                next[rec++]=0;
        }
        else{
            border++;
            next[rec]=border;
            rec++;
        }
    }
    return next;
}
```



A	A	T	A	A	A	T	U
0							

Border= 0

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```



A	A	T	A	A	A	T	U
0	1						

Border= 1

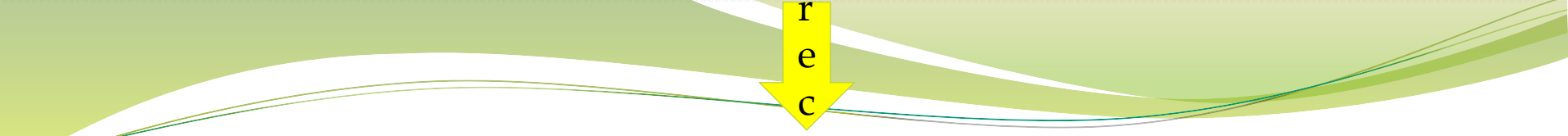
```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1						

Border= 1

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```



A	A	T	A	A	A	T	U
0	1						

Border= 0

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1						

Border= 0

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0					

Border= 0

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

A	A	T	A	A	A	T	U
0	1	0					

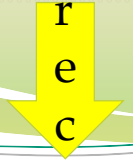
Border= 0

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

A	A	T	A	A	A	T	U
0	1	0					

Border= 0

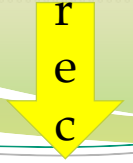
```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```



A	A	T	A	A	A	T	U
0	1	0	1				

Border= 1

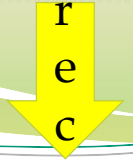
```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```



A	A	T	A	A	A	T	U
0	1	0	1				

Border= 1

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```



A	A	T	A	A	A	T	U
0	1	0	1				

Border= 1

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2			

Border= 2

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2			

Border= 2

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2			

Border= 1

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2			

Border= 1

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2	2		

Border= 2

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2	2		

Border= 2

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2	2	3	

Border= 3

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2	2	3	

Border= 0

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2	2	3	

Border= 0

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

rec

A	A	T	A	A	A	T	U
0	1	0	1	2	2	3	0

Border= 0

```
private static int[] nextComputation(char[] query) {
    int[] next = new int[query.length];

     int border=0;  // Length of the current border

     int rec=1;
     while(rec < query.length){
          if(query[rec]!=query[border]){
               if(border!=0)
                    border=next[border-1];
               else
                    next[rec++]=0;
          }
          else{
               border++;
               next[rec]=border;
               rec++;
          }
     }
     return next;
}
```

El algoritmo que calcula next tiene

Complejidad especial: O(m)

Complejidad temporal: O(m)

TP 2B- Ejer 2.1

Escribir la clase KMP

Agregar el método de clase nextComputation

Una vez calculada la tabla Next, ¿cómo se usa para calcular search sin hacer backtracking en el text?

Idea:

Supongamos un **rec** que apunta al caracter en **target** y que **pquery** que apunta a un caracter en **query**

Mientras haya coincidencia, avanzo en ambos.

Cuando no la haya se "shiftea" query a next[pquery-1], salvo que pquery sea 0, en cuyo caso hay que avanzar **rec** en **target**

S	A	B	A	S	A	B	A	B	A

query	**A**	B	**A**	**B**
Next	0	0	1	2

Si pquery>0 cambiaría solo pquery por next[pquery-1]
Pero en este caso es 0, o sea, sólo avanza **rec**

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Como coinciden, avanzan los 2

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Como coinciden, avanzan los 2

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Como coinciden, avanzan los 2

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Si pquery>0 cambiaría solo pquery por next[pquery-1]
Entonces, pquery apunta a la “b” de la posicion 1

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Si pquery>0 cambiaría solo pquery por next[pquery-1]
Entonces, pquery apunta a la “a” de la posición 0

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Si pquery>0 cambiaría solo pquery por next[pquery-1]
Pero es 0. Entonces solo avanza rec

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Como coinciden, avanzan los 2

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Como coinciden, avanzan los 2

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Como coinciden, avanzan los 2

S	A	B	A	S	A	B	A	B	A

query	A	B	A	B
Next	0	0	1	2

Como coinciden, pero alcancé el final de query, lo encontre!!!
Cuál es el cálculo de la primera posición encontrada? Rec - pquery

Otro ejemplo

Otro ejemplo

A	B	X	A	B	X	A	B	U	F

query	**A**	B	**X**	**A**	**B**	**U**
Next	0	0	0	1	2	0

El next[pquery-1] dice de donde seguir

Si pquery>0 cambiaría solo pquery por next[pquery-1]
Pero en este caso es 0, o sea, sólo avanza **rec**

Otro ejemplo

A	B	X	A	B	X	A	B	U	F

query	A	B	X	A	B	U
Next	0	0	0	1	2	0

Y sigue…

Si pquery>0 cambiaría solo pquery por next[pquery-1]
Pero en este caso es 0, o sea, sólo avanza **rec**