Projet d'algorithmique du texte

Fonction ReplaceALL() avec KMP

Martin Sanchez
Mathis CAPISANO
Mathieu TRANQUART

Explication du Code

```
public static String KMP(String chaine , String occurence , String remplace){
```

KMP prend en paramètre une variable :
chaine : qui est la chaine de caractère à modifier
occurence : est l'occurence qui doit-être remplacer
remplace : par quoi l'occurence vat-être remplacé.

Attention le code KMP et replace présenté n'est pas exactement celui dans notre code source. Nous avons enlevé toutes les parties qui concerne la gestion du surlignement en rouge.

Les variables dans la fonction KMP

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

On transfert la chaine mit en paramètre dans une variable global

Permet de savoir sur qu'elle caractère de « occurrence »on est situé

La variable bouleenne fini permettra de savoir si on continue de parcourir les caractères de l'occurence

La variable fini sera mit sur false si on trouve un caractères différent entre la chaine et l occurrence a cherché ou si toute l'occcurence aura été trouvé (pour éviter de boucler à l'infini dans le while)
Et remis sur true à chaque nouveau caractère de la chaine

Les boucles dans la fonction KMP

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

Cette boucle permet de parcourir la chaine de caractères qui doit être modifié

La boucle while sera parcourus si fini est a true (donc cf diapo précedente) ou si l'occurrence peut déborder au dela de la chaine(par soucis d'optimisation)

Les conditions dans la fonction KMP

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

Si les caractères sont identitques on peut continuer dans la boucle while (car pour l'instant les caractères sont identiques)

Sinon cela veut dire qu'un caractères est différent on sort de la boucle while (en mettant fini a false) et on ce replace à l'emplacement 0 de l'occurence

Les conditions dans la fonction KMP

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

Si on arrive jusqu'à la fin de l'occurrence cela veut dire qu'on a trouvé une occurrence dans la chaine de caractère

Alors on remplace dans dest l'occurrence trouvé par la chaine de remplacement

On reajuste i à la nouvelle taille de dest

Et on quitte la boucle while en ce plaçant au caractères 0 de l'occurrence et ainsi passer au caractères suivant de la chaine

Les conditions dans la fonction KMP

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

Et sinon on ce place au prochain caractères d'occurrence pour vérifier si ils sont identique eux aussi a la chaine

Analyse du code replace

```java
public void replace(String cible, int index, String motif) {
    String partA = dest.substring(0, index) ;
    String partB = motif;
    int borne = index + cible.length() ;
    String partC = dest.substring(borne, dest.length()) ;
    dest =  partA + partB + partC;
}
```

On découpe la chaine de caractère dest en 3 partis
Parti A qui est situé avant la parti qu'on doit remplacer
Partis B : qui sera remplacé par le motif qu'on souhaite
La parti C : qui est situé après la parti qu'on veut modifié

Et on réassemble les 3 parties entre elles.

Déroulement de l'algorithme

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

Chaine = RONDOUDOU
Occurrence = OUD
Remplace = MAMA

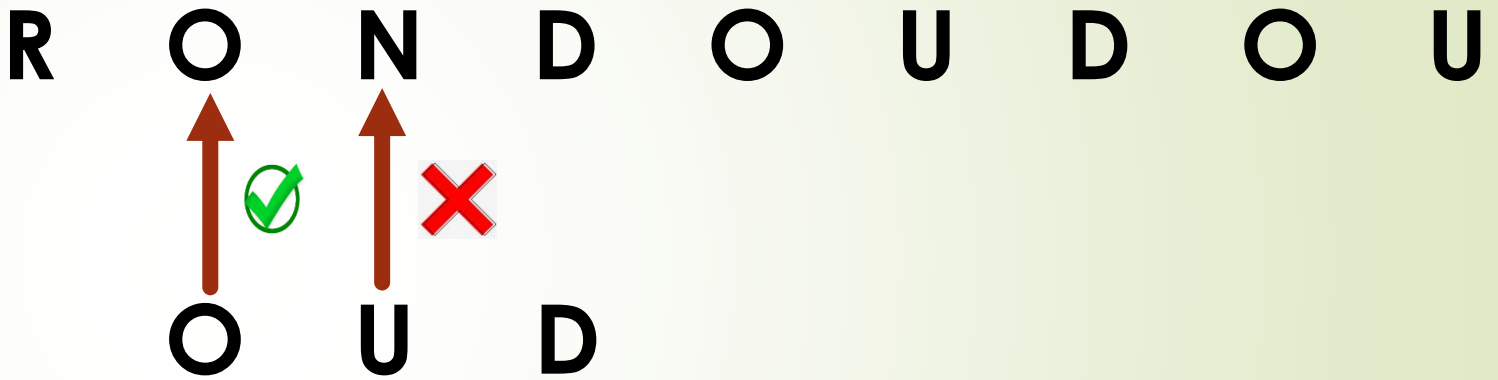
Déroulement de l'algorithme

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

I	J	fini
0	0	FAUX
X	X	X
X	X	X

Déroulement de l'algorithme

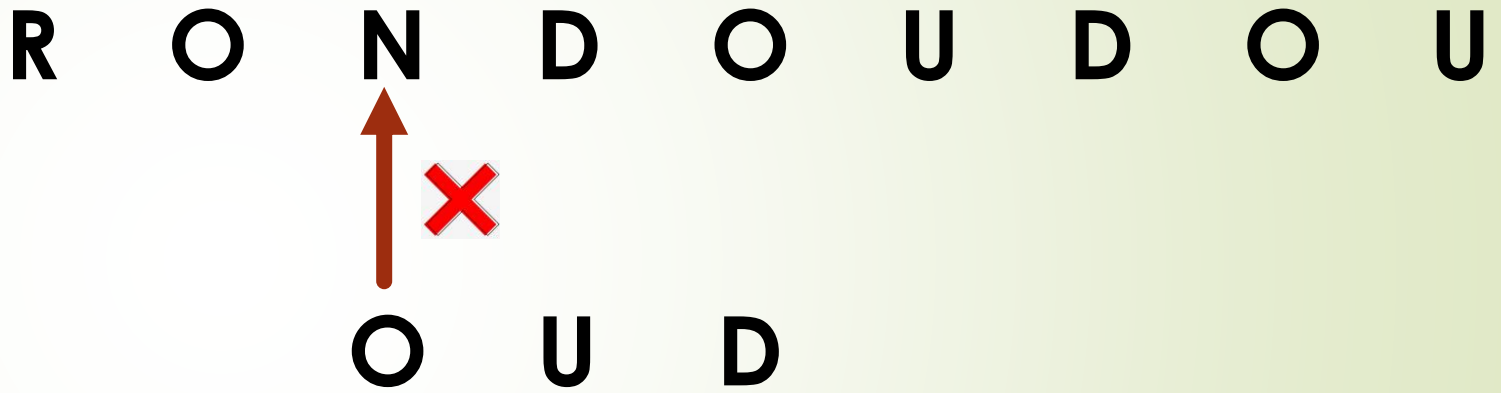
```java
public static String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+j) < dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    int index = i;
                    String cible = dest.substring(index, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```



I	J	fini
0	1	VRAI
1	2	FAUX
X	X	X

Déroulement de l'algorithme

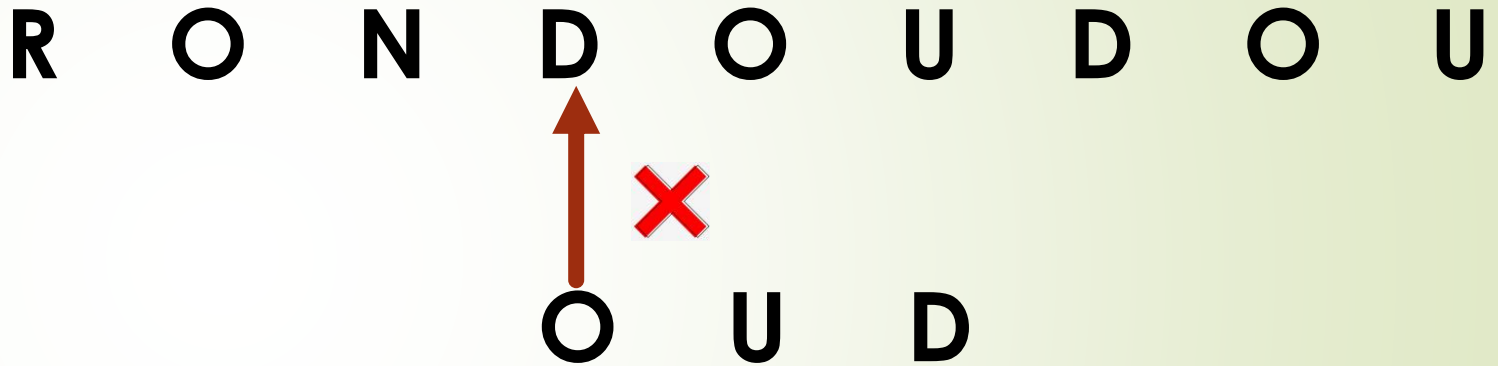
```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```



I	J	fini
0	2	FAUX
X	X	X
X	X	X

Déroulement de l'algorithme

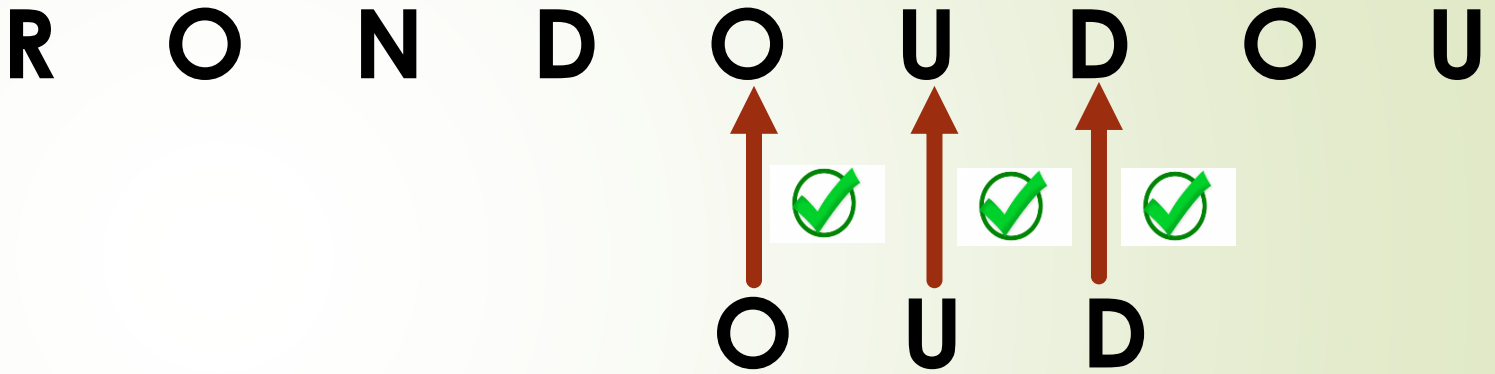
```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```



I	J	fini
0	3	FAUX
X	X	X
X	X	X

Déroulement de l'algorithme

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```



I	J	fini
0	4	VRAI
1	5	VRAI
2	6	VRAI

Déroulement de l'algorithme

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

R O N D M A M A O U

I	J	fini
X	X	X
X	X	X
X	X	X

On arrête le programme car l'occurence déborde sur la chaine de caractère.

Autre fonction du programme

Permet de générer l'interface graphique

```
public void init() {
    JFrame f = new JFrame("KMP algo");
    f.setSize(800, 1600);
    JLabel l1,l2;
    JScrollPane texteAsc1;
    JScrollPane texteAsc2;

    GridBagLayout layout = new GridBagLayout();
    f.setLayout(layout);
    GridBagConstraints gbc = new GridBagConstraints();

    gbc.fill = GridBagConstraints.HORIZONTAL;

    passwordField1 = new JTextArea("");
    passwordField1.setPreferredSize(new Dimension(100, 25));
    passwordField2 = new JTextArea("");
    passwordField2.setPreferredSize(new Dimension(100, 25));

    grandeZone1.setPreferredSize(new Dimension(100, 700));
    grandeZone1.setMaximumSize(new Dimension(100, 700));
    redPainter = new DefaultHighlighter.DefaultHighlightPainter(Color.red);
    grandeZone2.setPreferredSize(new Dimension(100, 700));
    grandeZone2.setMaximumSize(new Dimension(100, 700));
    l1=new JLabel("Occurence");
    l2=new JLabel("remplacer");
```

Permet la gestion des boutons de l'interface

```
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source.equals(pickFile)) {
        File repertoireCourant = null;
        try {
            repertoireCourant = new File(".").getCanonicalFile();
        } catch (IOException e1) {
            e1.printStackTrace();
        }

        JFileChooser dialogue = new JFileChooser(repertoireCourant);

        dialogue.showOpenDialog(null);

        String emplacement = dialogue.getSelectedFile().getPath() ;
        String text = lecture(emplacement);

        grandeZone1.setText("");
        grandeZone1.setText(text);
```

Autre fonction du programme

Fonction servant à enregistrer le texte généré

```
public  void enregistrer(String emplacement) throws IOException {
    try {


        File f = new File(emplacement+".txt");



        if (f.createNewFile())
            System.out.println("File created");
        else
            System.out.println("File already exists");

        FileWriter fw = new FileWriter(f,true);
        fw.write(grandeZone2.getText());
        fw.close();
    }
    catch (Exception e) {
        System.err.println(e);
    }
}
```

Fonction servant a lire un fichier text

```
public String lecture(String emplacement) {
    try (BufferedReader in = new BufferedReader(new FileReader(emplacement))) {

            StringBuilder builder = new StringBuilder();
            String line;
        while ((line = in.readLine()) != null) {
            // Afficher le contenu du fichier
            builder.append('\n' + line);
        }

        return builder.toString();
    } catch (IOException e) {
            e.printStackTrace();
            return null;
    }
}
```

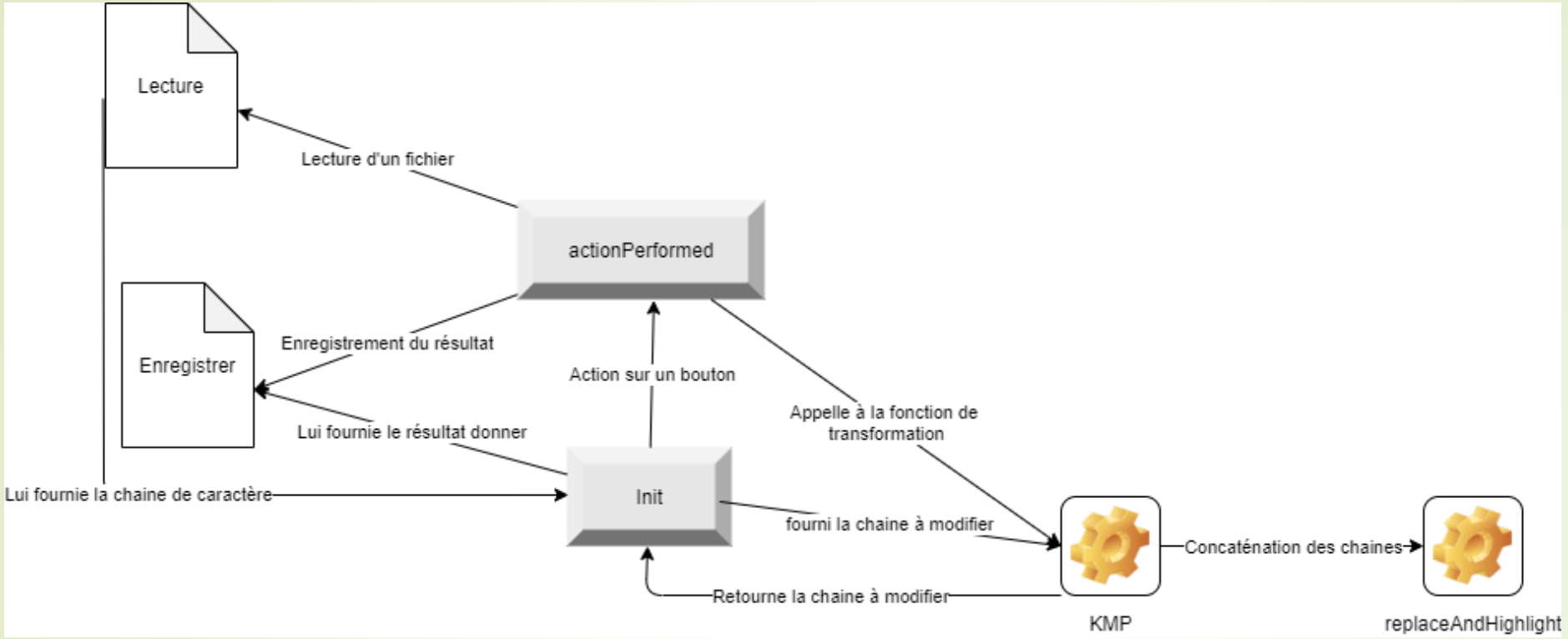


Schéma UML du programme

Complexité en temps

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

L'algorithme KMP à deux boucle imbriqué
Une boucle for qui parcours toute le text
Et une boucle while qui parcours l'occurrence
N = taille du texte
M = Taille de l'occurrence

Complexité en temps

```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

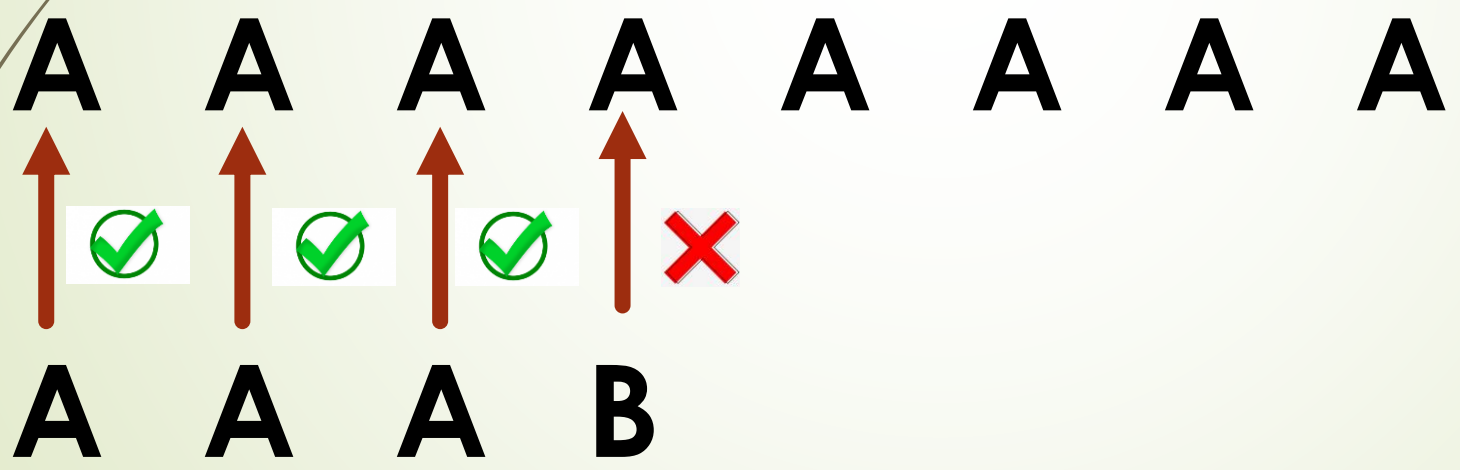
Vu qu'on ne parcours pas la boucle while si l'occurrence déborde sur la chaine de caractère alors cette complexité en fonction de l'occurence est en O(log(M))

Donc la complexité est de O(N*log(M)) dans le pire des cas

Complexité en temps

La complexité déprendra aussi du nombre d'occurrence trouvé et de caractère similaire.
Illustration avec un exemple:

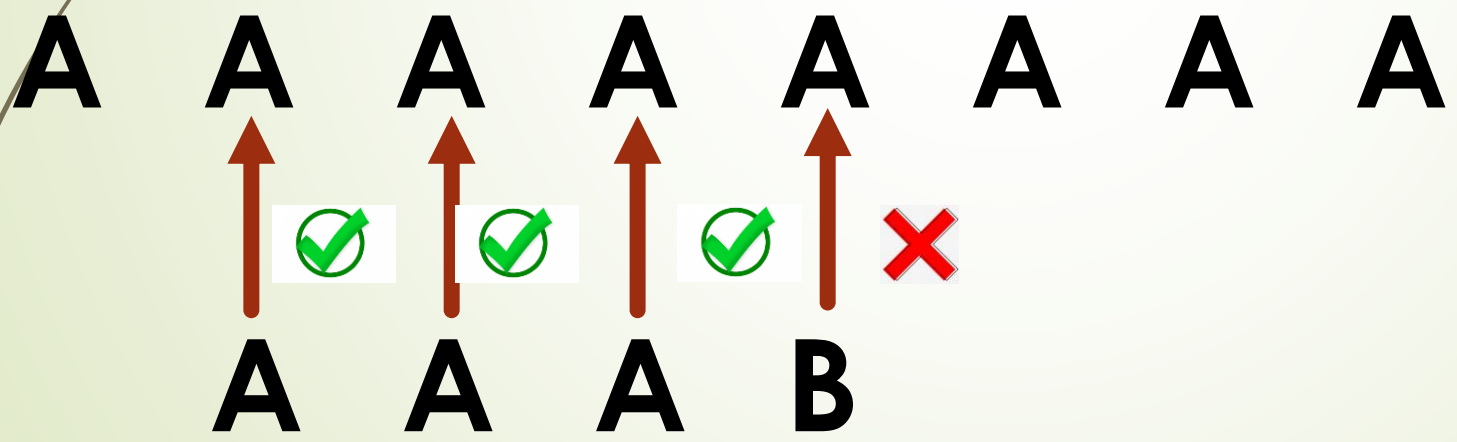
Tour de la boucle for : 1
Tour de la boucle while : 4



Complexité en temps

La complexité déprendra aussi du nombre d'occurrence trouvé et de caractère similaire.
Illustration avec un exemple:

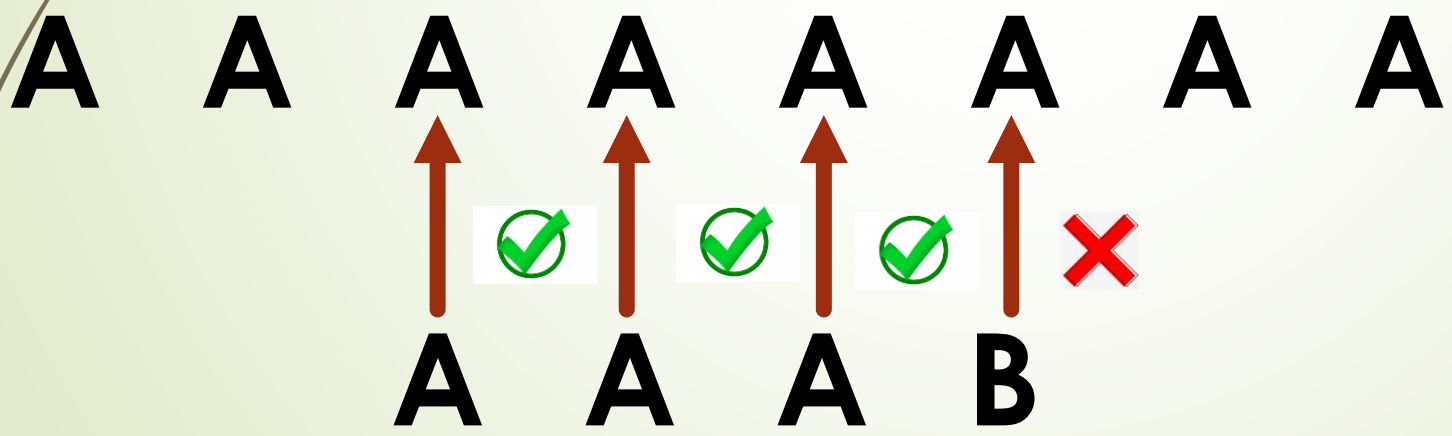
Tour de la boucle for : 2
Tour de la boucle while : 4



Complexité en temps

La complexité déprendra aussi du nombre d'occurrence trouvé et de caractère similaire.
Illustration avec un exemple:

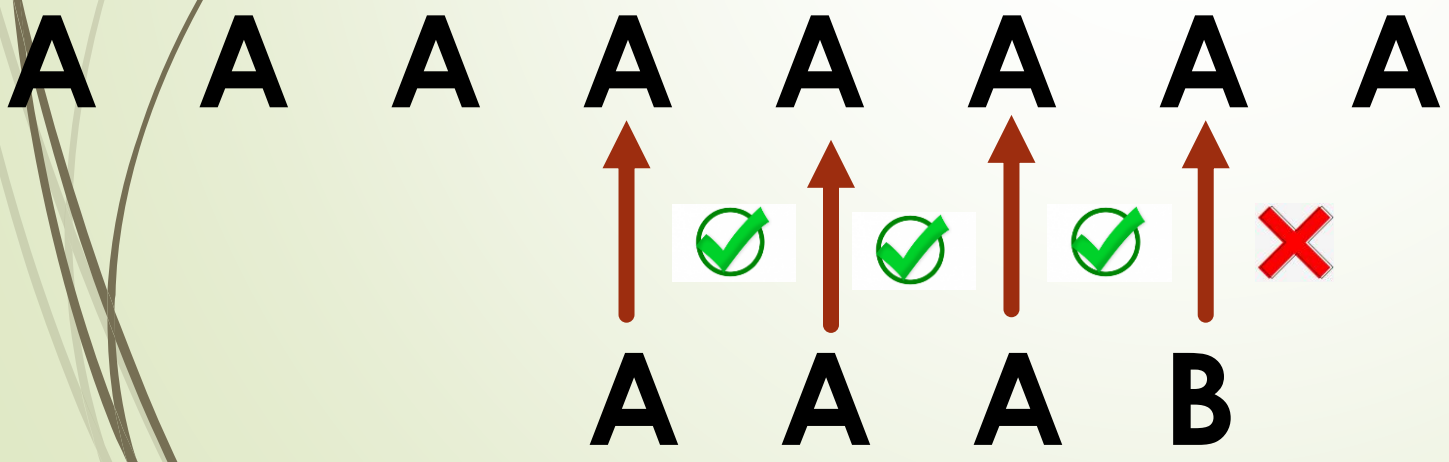
Tour de la boucle for : 3
Tour de la boucle while : 4



Complexité en temps

La complexité déprendra aussi du nombre d'occurrence trouvé et de caractère similaire.
Illustration avec un exemple:

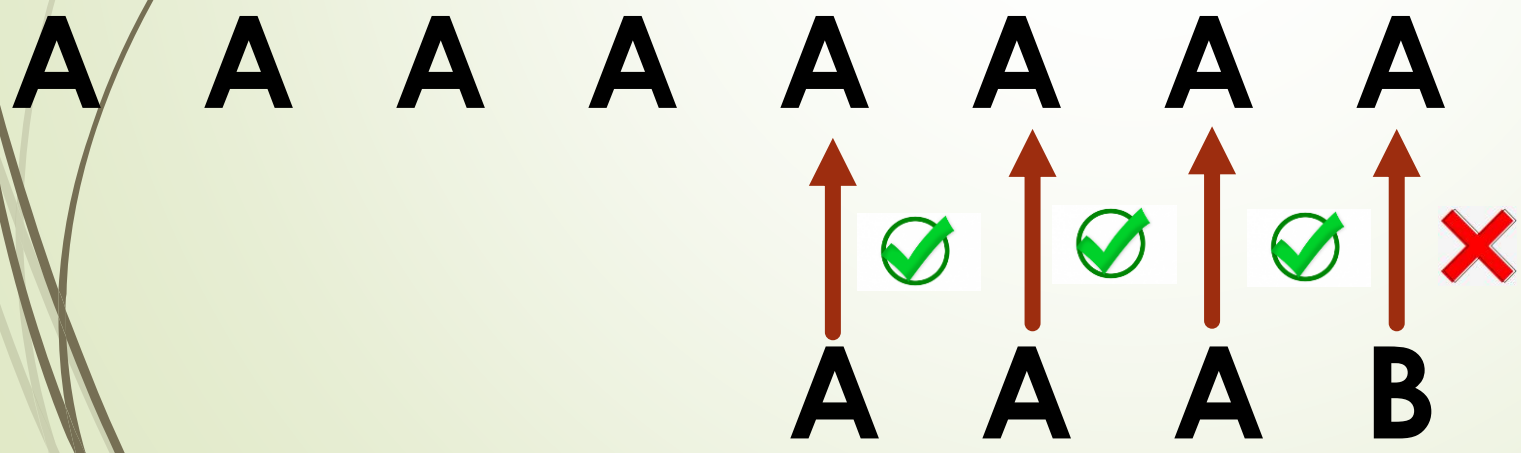
Tour de la boucle for : 4
Tour de la boucle while : 4



Complexité en temps

La complexité déprendra aussi du nombre d'occurrence trouvé et de caractère similaire.
Illustration avec un exemple:

Tour de la boucle for : 5
Tour de la boucle while : 4

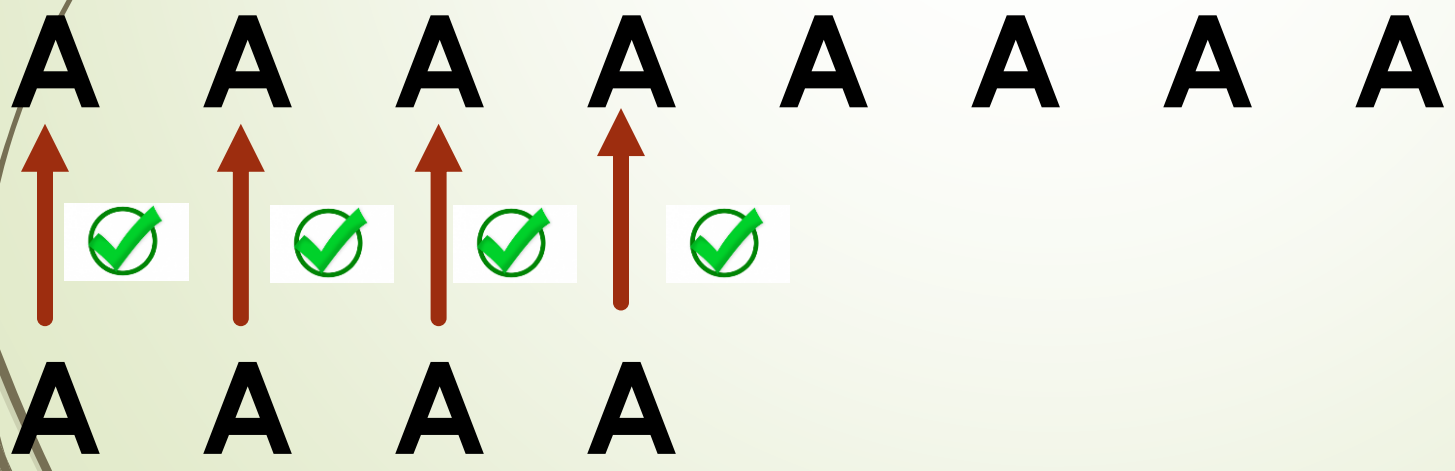


Au total il y a eu donc 20 tour de boucle

Complexité en temps

La complexité déprendra aussi du nombre d'occurrence trouvé et de caractère similaire.
Illustration avec un exemple:

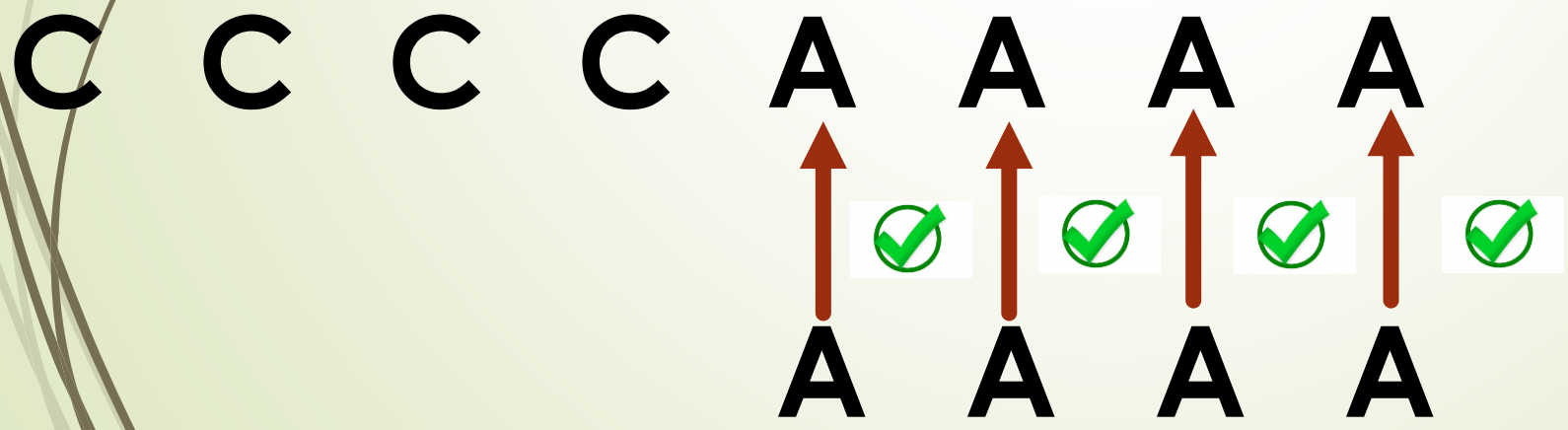
Tour de la boucle for : 1
Tour de la boucle while : 4



Complexité en mémoire

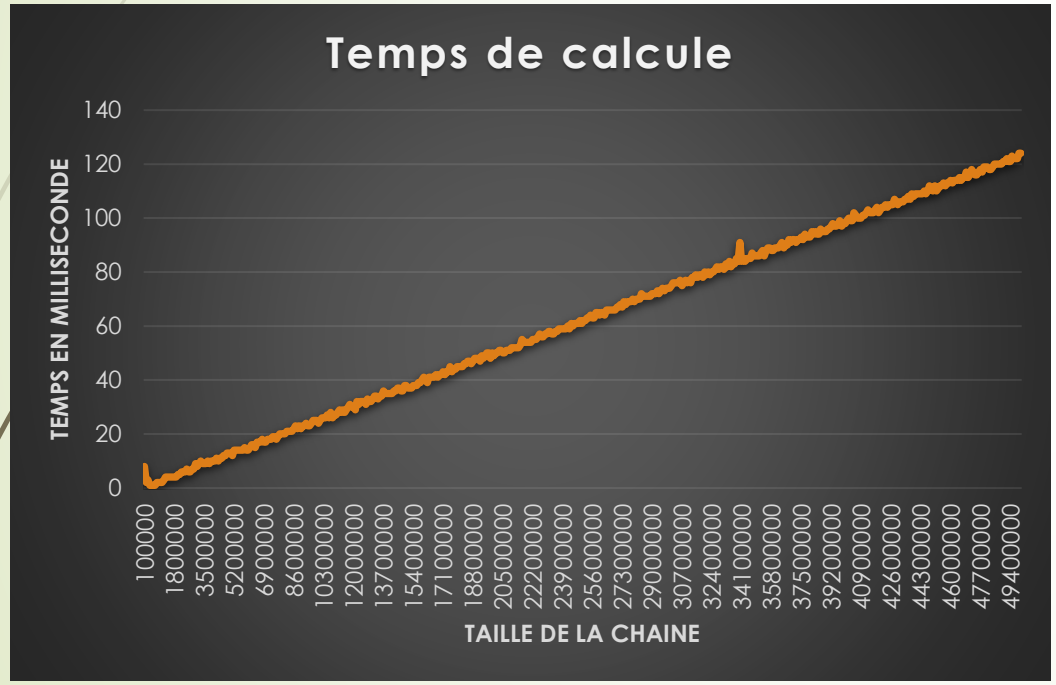
La complexité déprendra aussi du nombre d'occurrence trouvé et de caractère similaire.
Illustration avec un exemple:

Tour de la boucle for : 2
Tour de la boucle while : 4



Cette fois ci il y a eu de 8 tours de boucle alors que la taille de la chaine et de l'occurrence n'a pas bougé

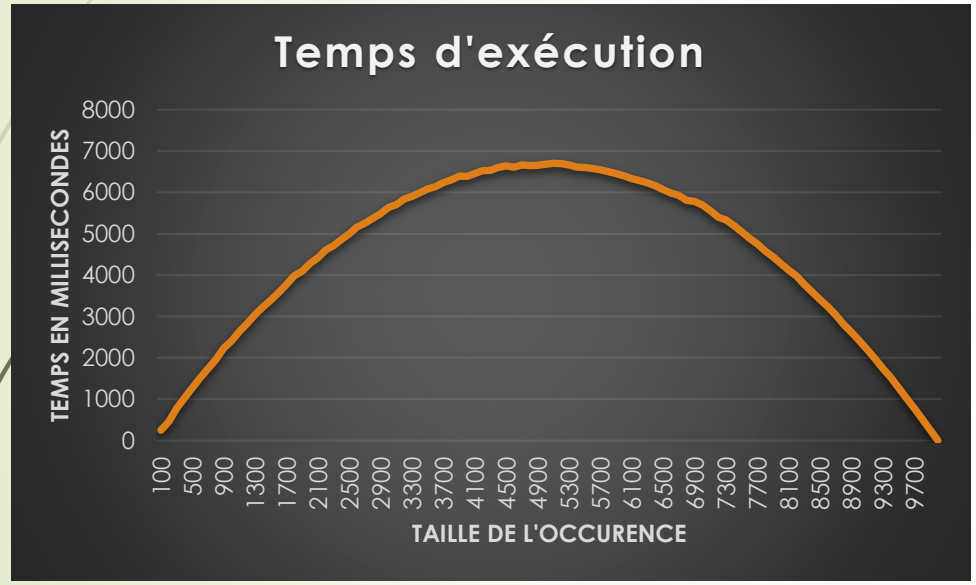
Temps d'éxections



Voici le graphique de nos test sur le temps d'exécution de notre programme KMP en fonction de la taille du texte passé en paramètre. On peut bien voir que en fonction de de la taille de la chaine de caractère la fonction est linéaire.(la courbe de prend pas en fonction de la taille de l'occurence cherché)

Les mesures de temps et d'exécution on était faite dans la pire des situations possible pour le programme.

Temps d'éxections



Voici le graphique des temps en fonctions de la taille de l'occurrence.Elle a une forme de parabole car plus la taille de l'occurrence grandit plus vite elle s'approche du bord du texte.

Complexité en mémoire

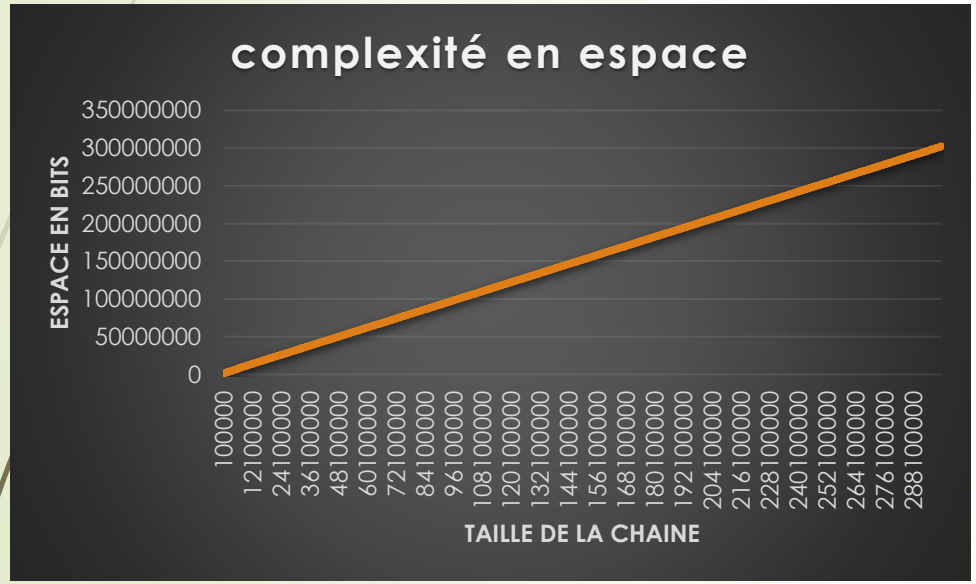
```java
public String KMP(String chaine , String occurence , String remplace){
    dest = chaine;
    int j = 0;
    boolean fini ;
    for(int i =0 ; i < dest.length() ; ++i) {
        fini = true;
        while(fini && (i+occurence.length()) <= dest.length()){
            if(dest.charAt(i+j) == occurence.charAt(j)) {
                if(j == occurence.length()-1) {
                    String cible = dest.substring(i, i+(occurence.length()));
                    replace(cible,index,remplace);
                    i = i + remplace.length()-1;
                    j=0;
                    fini = false;
                }
                else {
                    ++j;
                }
            }
            else{
                j = 0;
                fini = false;
            }
        }
    }
    return dest;
}
```

L'algorithme KMP retient en mémoire le texte , l'occurrence et le motif de remplacement

N = taille du texte
M = Taille de l'occurrence
R = Taille du motif de remplacement
K = nombre d'occurrence trouvé

Vu qu'on remplace dans le texte l'occurrence recherché par le motif à remplacer la place en mémoire du texte varie en
O(N+(R-M)*k)
Ou k est le nombre de fois que le motif est trouvé dans le texte.
Ce qui donne une complexité en mémoire de
O((N+(R-M)*k)+R+M)

Complexité en espace



Voici le graphique de l'espace en mémoire en fonction de la taille de la chaine de caractère.On voit bien que la complexité est linéaire.

Les mesures ont été faite dans le pire des cas et ne tient donc pas compte du fait que la chaine de caractères peut être racoursite pendant l'exécution du programme.