



# RAPPORT DE PROJET
# Complexité et algorithmique :

## Découpage sur une machine de Turing à une bande

Chargé du cours :

M J. BOND

Intervenants : M D. MAZAURIC, M C. PAPAZIAN

Rédigé par

Smith DJAMOURA

Année universitaire : 2019-2020

# Table des matières

Introduction générale ..... 3
1. Description du problème ..... 3
2. Proposition de solutions ..... 3
2.1. Recherche de la médiane ..... 3
2.1.1. Description de l'algorithme ..... 4
2.1.2. Analyse de la complexité ..... 4
2.1.3. Représentation de la machine de Turing ..... 5
2.1.4. Quelques exemples ..... 7
2.2. Un algorithme en O(nlog(n)) pour la recherche de la médiane ..... 7
2.2.1. Description de l'algorithme ..... 8
2.2.2. Analyse de la complexité ..... 9
2.2.3. Représentation de la machine de Turing ..... 9
2.2.4. Quelques exemples ..... 11
2.3. Résolution du problème du découpage ..... 11
2.3.1. Description de l'algorithme ..... 12
2.3.2. Analyse de la complexité ..... 13
2.3.3. Représentation de la machine de Turing ..... 13
2.3.4. Quelques exemples ..... 15
Conclusion générale ..... 15
Bibliographie ..... 15

# Introduction générale

Planifier son trajet en voiture, trier ses chaussettes, résoudre un sudoku, trouver une bonne stratégie au jeu du socoban… Notre quotidien est jonché de problèmes à résoudre ; certains semblent faciles, d'autres beaucoup moins. La théorie de la complexité algorithmique vient à notre rescousse afin d'y voir un peu plus clair.

Ainsi, dans le cadre de notre cours intitulé « Algorithmique et complexité », nous effectuons un mini projet dont le problème à traiter est celui du découpage sur une machine de Turing.

## 1. Description du problème

Le travail qui nous est demandé dans ce mini projet est celui du découpage sur une machine de Turing. Le résultat attendu à l'issu de ce projet est qu'étant donné un mot de la forme XXX...XXX de longueur k (k > 0) nous puissions fournir un mot 111…111222…222333…333444…444 ($1^a\ 2^b\ 3^c\ 4^d$) tel que $a \leq b \leq c \leq d \leq a+1$ avec a+b+c+d = k. Il faut noter que l'algorithme de découpage proposé doit être en O(nlog(n)).

Ainsi, pour aboutir à ce résultat une démarche à suivre nous a été proposée :

- ✓ Trouver un algorithme de recherche du milieu de la donnée (Le mot en entrée)
- ✓ Trouver ensuite un algorithme efficace pour le même problème (recherche du milieu)
- ✓ Adapter le programme efficace obtenu pour résoudre le problème du découpage

## 2. Proposition de solutions

Tout comme l'indique le titre de cette section, nous allons y faire des propositions de solutions tout en respectant la démarche proposée pour la résolution du problème.

### 2.1. Recherche de la médiane

La principale difficulté lors de la résolution de ce problème n'était pas la représentation de la machine mais plutôt le fait de trouver un algorithme qui fournit le

résultat attendu tout en respectant la principale contrainte qui est l'utilisation d'une seule bande pour implémenter notre machine de Turing.

### 2.1.1. Description de l'algorithme

Posons le problème autrement : On pose sur une table n objets avec n>0 et n suffisamment grand pour que le résultat ne soit pas trivial et on vous demande de trouver l'objet qui se trouve au milieu et admettons avec une prime de 1000 euros si vous trouvez le milieu.

Le premier réflexe que vous auriez eu si l'on vous posait un tel problème serait bien évidemment de compter, ensuite de faire une division par deux pour trouver le milieu. Mais, imaginons que vous n'ayez pas de quoi faire le calcul et admettons que les divisions par deux mentalement ce n'est pas votre fort. Alors que faites-vous ? Comment empocher les 1000 euros ?

On constate que naturellement, vos yeux se baladent de gauche à droite en éliminant à chaque extrémité des n objets un objet à chaque fois tout simplement dans le but de tomber à la fin sur l'objet du milieu. Voilà en quoi consiste notre premier algorithme de recherche de la médiane.

### 2.1.2. Analyse de la complexité

Nous avons à ce niveau un algorithme qui marche, mais quel est son coût ? Ce sera l'objet de ce sous-chapitre.

Lorsque nous avons n éléments d'indices 1 à n et que nous déroulons notre algorithme de la gauche de l'entrée vers la droite, on a :

- Ecarter le 1$^{er}$ élément ne coûte rien, mais le n$^{em}$ à droite coute n-1
- Ecarter le 2$^{e}$ élément coute n-2 et le (n-1)$^{em}$ coute n-3

En continuant ainsi de suite jusqu'au dernier élément qui sera le milieu on se retrouve avec une somme des n entiers consécutifs que nous savons très bien calculer. Le résultat de cette somme donne **½ n (n + 1)** et en développant le plus grand monôme se trouve être en n2 d'où nous pouvons conclure que notre algorithme est en O(n$^2$).

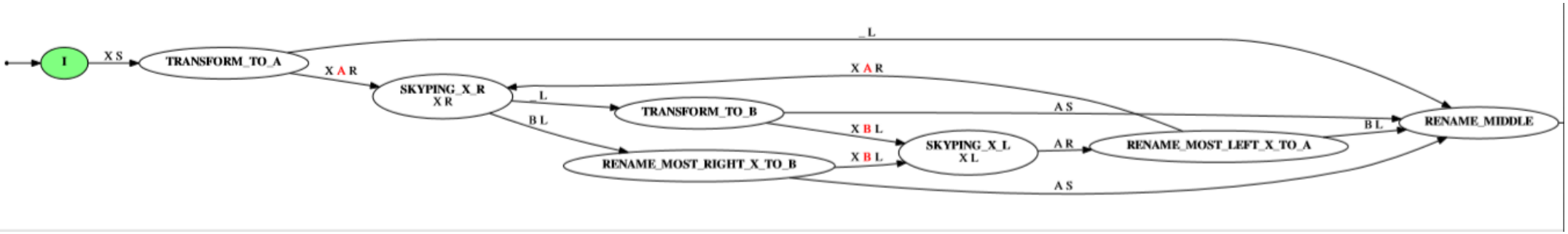
### 2.1.3. Représentation de la machine de Turing

Avant de passer à la représentation de la machine, nous rappellerons que son principe de fonctionnement est décrit dans le fichier programme.txt du rendu et qui est basé sur l'algorithme décrit précédemment.

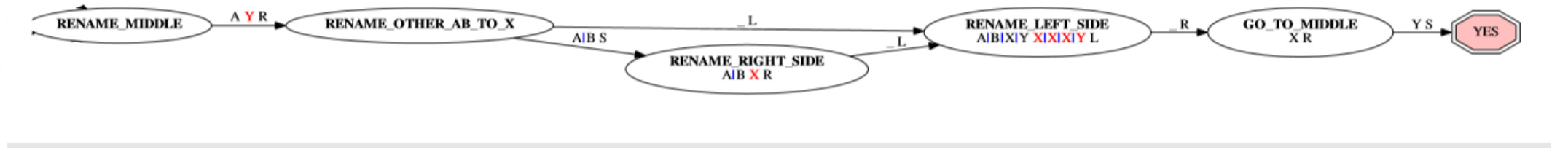
Ainsi, la machine remplace les caractères du côté gauche par des A et ceux du coté droit par des B bien sûr en faisant un remplacement à gauche puis un à droite et ainsi de suite jusqu'à trouver le milieu qui est remplacé par 'Y'. En cas de longueur pair, c'est le caractère le plus à gauche entre les deux au milieu qui est choisi comme milieu conformément à l'énoncé du problème.

La représentation fournit est celle que nous avons généré avec le simulateur mis à notre disposition.  La machine a été divisée en deux sous graphes pour une meilleure présentation.

Sous-graphe 1 ←→ Etat RENAME_MIDDLE ←→ Sous-graphe 2




1er sous-graphe




2e sous-graphe

### 2.1.4. Quelques exemples

| Entrée $X^K$ | Sortie | Transitions |
|---|---|---|
| **k=1** | Y | 9 |
| **k=2** | YX | 15 |
| **k=3** | XYX | 21 |
| **k=4** | XYXX | 28 |
| **k=5** | XXYXX | 36 |
| **k=6** | XXYXXX | 45 |
| **k=7** | XXXYXXX | 55 |
| **k=8** | XXXYXXXX | 66 |
| **k=9** | XXXXYXXXX | 78 |
| **k=10** | XXXXYXXXXX | 91 |
| **k=50** | $X^{24}YX^{25}$ | 1431 |
| **k=100** | $X^{49}YX^{50}$ | 5356 |
| **k=200** | $X^{99}YX^{100}$ | 20706 |
| **k=500** | $X^{249}YX^{250}$ | 126756 |
| **k=1000** | $X^{499}YX^{500}$ | 503506 |

- Petit zoom sur le déroulement pour k = 5

On a donc en entrée un mot m = « XXXXX »

- ✓ 1ere itération : XXXXX devient AXXXX (parcours de gauche à droite jusqu'au blanc)
- ✓ 2e itération : AXXXX devient AXXXB (parcours de droite à gauche jusqu'au prochain A)
- ✓ 3e itération : AXXXB devient AAXXB (parcours de gauche à droite jusqu'au prochain B)
- ✓ 4e itération : AAXXB devient AAXBB (parcours de droite à gauche jusqu'au prochain A)
- ✓ 5e itération : AAXBB devient AAABB (parcours de gauche à droite jusqu'au prochain B)

A ce stade au prochain parcours de la droite vers la gauche, on tombe sur un A juste après un B, c'est donc le milieu qui sera renommé en Y et ensuite on renomme tous les A, B restant en X puis on se repositionne au milieu.

## 2.2. Un algorithme en O(nlog(n)) pour la recherche de la médiane

Avoir un programme qui marche c'est bien, mais un programme qui marche et qui est rapide pour ne pas dire efficace c'est encore mieux. C'est pour cela qu'il nous a été demandé de trouver un deuxième algorithme qui lui sera efficace et qui devra être en O(nlog(n)).

### 2.2.1. Description de l'algorithme

La question est la suivante : comment peut-on à partir de ce que nous avons, obtenir un algorithme efficace ?

Nous devons chercher la médiane de notre mot avec un algorithme en O(nlog(n)) et comme par hasard il existe un algorithme de recherche qui se trouve être en log(n) : Il s'agit de l'algorithme de recherche dichotomique. Nous allons nous inspirer de cet algorithme.

Tout comme l'algorithme de recherche dichotomique nous allons réduire les éléments parmi lesquels peut se trouver l'élément recherché par 2 à chaque itération. Ceci implique que nous ferons au maximum $\log_2(n)$ itérations pour trouver l'élément cherché qui est dans notre cas le milieu.

Mais contrairement à la recherche dichotomique, où à chaque itération une seule opération est effectuée, nous effectuerons ici 2n opérations. Pourquoi 2n ? Pour le découvrir il faut passer sans plus tarder à la description de l'algorithme :

L'algorithme consiste à remplacer chaque lettre ($X^k$) du mot reçu en entrée par une suite alternée de A, B jusqu'à la fin du mot ce qui coûte n (n étant la taille de l'entrée). Ensuite, on supprime soit les A, soit les B en rebroussant chemin et en renommant les caractères qui ne seront pas supprimés par X à nouveau jusqu'au début du mot. Les caractères dits supprimés sont en fait juste renommés en un autre caractère muet (Z le caractère choisi dans notre algorithme) qui ne sera plus pris en compte lors de la prochaine itération. Ce deuxième parcours coute également n. voilà d'où viennent les 2n opérations mentionnées un peu plus haut.

Cependant, on a parlé de suppression pour ne pas dire de renommage de caractères mais lesquels doit-on supprimer ? Après quelques essais sur papier, on constate que lorsqu'après avoir écrit la série de A, B jusqu'au bout :

- ✓ Le nombre de A est pair : il faut donc supprimer les A, Le milieu est un des B que nous avons écrit.
- ✓ Le nombre de A est impair : il faut donc supprimer les B, Le milieu est un des A que nous avons écrit.

Ainsi on retrouve notre milieu lorsqu'il reste un caractère.

### 2.2.2. Analyse de la complexité

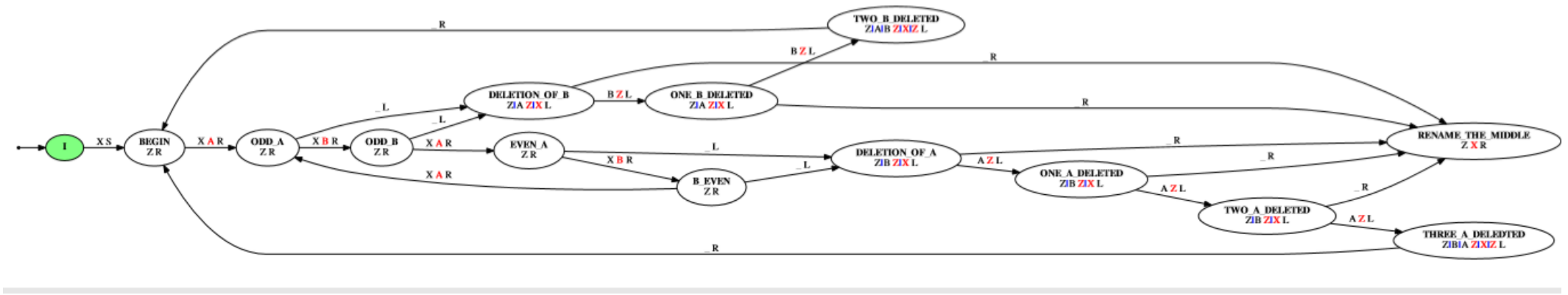
Dans la partie précédente, nous avons mentionné que nous trouvons notre milieu en $\log_2(n)$ itérations qui nous coute chacune $2n$. En fait après les $\log_2(n)$ itérations, on se retrouve au début du mot, avec le milieu marqué en X et les autres caractères sont des Z (les caractères supprimés). Il faut donc refaire un parcours de gauche à droite pour renommer les Z en X et le milieu X en Y ce qui nous coûte $n$ et après quitter la enfin du mot pour se repositionner en Y c'est-à-dire au milieu ce qui coûte $n/2$.

En définitif, on se retrouver avec un coût final de $2n\log_2(n) + n + n/2$ ce qui nous permet de dire que cet algorithme est en O(nlog(n)).

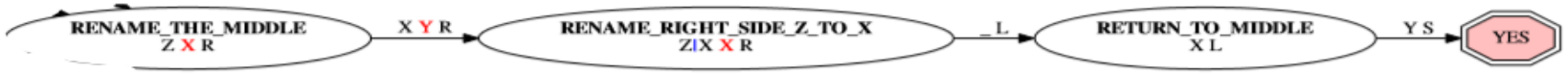
### 2.2.3. Représentation de la machine de Turing

La représentation fournit est celle que nous avons généré avec le simulateur mis à notre disposition. La machine a été divisée en deux sous graphes pour une meilleure présentation.

Sous-graphe 1 ←→ Etat RENAME_THE_MIDDLE ←→ Sous-graphe 2




1er sous-graphe




2e sous-graphe

### 2.2.4. Quelques exemples

| Entrée $X^K$ | Sortie | Transitions |
|---|---|---|
| **k=1** | Y | 8 |
| **k=2** | YX | 12 |
| **k=3** | XYX | 15 |
| **k=4** | XYXX | 19 |
| **k=5** | XXYXX | 34 |
| **k=6** | XXYXXX | 40 |
| **k=7** | XXXYXXX | 45 |
| **k=8** | XXXYXXXX | 51 |
| **k=9** | XXXXYXXXX | 76 |
| **k=10** | XXXXYXXXXX | 84 |
| **k=50** | $X^{24}YX^{25}$ | 588 |
| **k=100** | $X^{49}YX^{50}$ | 1365 |
| **k=200** | $X^{99}YX^{100}$ | 3117 |
| **k=500** | $X^{249}YX^{250}$ | 8769 |
| **k=1000** | $X^{499}YX^{500}$ | 19521 |

- Analyse du tableau d'exemple

Les résultats obtenus (valeurs de k et transitions) confirment que l'algorithme est en O(nlog(n)). On remarque aussi que la différence est très nette entre ces résultats et ceux de l'algorithme précédent.

## 2.3. Résolution du problème du découpage

Nous y voilà ! Après les algorithmes de recherche du milieu, la dernière étape de notre travail consiste à trouver une solution pour le problème de découpage posé.

Une solution naïve a ce problème est d'écrire une suite de 4, 3, 2, 1 jusqu'à la fin du mot, d'y ajouter un autre caractère pour marquer la fin qui servira de repère, puis à partir de ce repère recopier tous les 1 à gauche du repère à sa droite, puis les 2, puis les 3 et enfin les 4. Lorsqu'il n'y a plus de caractère à copier à gauche du repère placé alors c'est la fin on supprime tout ce qu'il y à gauche du repère et le repère lui-même et voilà le travail. Mais cet algorithme est-il efficace ? Mais même avant de se poser la question sur son efficacité, avons-nous respecté la consigne qui est de s'inspirer de l'algorithme obtenu au 2 pour trouver une solution au problème du découpage ? La réponse est non !

Néanmoins, cette méthode naïve nous conduira à constater quelque chose de très important. On distingue 4 principaux cas : k'>0

- ✓ k=4k', on a : a=b=c=d
- ✓ k=4k'+1, on a : a=b=c et d=a+1
- ✓ k=4k'+2, on a : a=b et c=d=a+1
- ✓ k=4k'+3, on a : b=c=d=a+1

En plus de ces cas s'ajoutent, les cas particuliers pour k = {1,2,3}. Le k=4 a été traité comme un cas particulier pour des soucis d'optimisation sur la machine fournie.

### 2.3.1. Description de l'algorithme

L'algorithme qui suit se base essentiellement sur le principe de recherche du milieu proposé précédemment. A l'aide de notre algorithme efficace de recherche du milieu nous allons diviser le mot d'entrée en quatre parties par 3 différents milieux (Un milieu N qui sépare le mot en deux parties, un milieu O qui sépare en deux la partie droite du mot et un milieu M qui logiquement sépare la partie gauche du mot en 2 parties) puis nous allons écrire une suite de 1 jusqu'au 1$^{er}$ milieu, puis une suite de 2 jusqu'au 2$^{e}$ milieu, puis une suite de 3 jusqu'au 3$^{e}$ milieu et de là une suite de 4 jusqu'à la fin du mot. Notons que l'opération d'écriture peut se faire dans l'autre sens et c'est la méthode que nous avons choisis dans notre programme. Pourquoi celle-là ? C'est simple dans le souci d'optimisation que j'explique :

Lors du premier parcours pour la recherche du 1$^{er}$ milieu, comme décrit précédemment on écrit une suite de A, B jusqu'au bout du mot. A cet instant précis on sait déjà si nous nous trouvons dans le cas 4k, 4k+1, 4k+2 ou 4k+3 en fonction de la parité de A et B. Naturellement, nous pensons à poser un nouveau caractère de fin (U :4k, V :4k+1 et 4k+2, W :4k+3) pour qu'après avoir trouvé les 3 milieux il nous rappelle dans quel cas nous nous trouvons. Après avoir donc lu ce caractère il est évident qu'il vaut mieux commencer la réécriture par la gauche immédiatement après.

- Analyse des différents cas :

- ✓ A impair : 4k+1, M inclus lors de la réécriture des 1, N non inclus dans la réécriture des 2, O non inclus dans la réécriture des 3
- ✓ B impair : 4k+2, M inclus lors de la réécriture des 1, N non inclus dans la réécriture des 2, O non inclus dans la réécriture des 3
- ✓ A pair :4k+3, M non inclus lors de la réécriture des 1, N non inclus dans la réécriture des 2, O non inclus dans la réécriture des 3
- ✓ B pair :4k, M inclus lors de la réécriture des 1, N inclus dans la réécriture des 2, O inclus dans la réécriture des 3

Cette analyse montre que les cas 4k+1 et 4k+2 sont similaires au niveau de la réécriture.

### 2.3.2. Analyse de la complexité

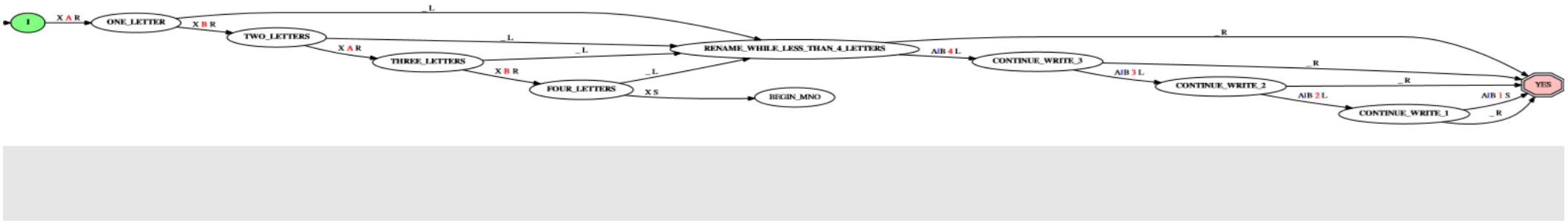
A ce stade, nous avons un algorithme qui résout le problème du découpage. Quelle est donc sa complexité ? Cet algorithme cherche le milieu d'une entrée de taille n (O(nlog(n))), puis cherche le milieu des deux côtés de l'entrée (O(nlog(n))) et ensuite fait un parcours de réécriture qui coûte n. L'algorithme est donc en O(nlog(n)).

### 2.3.3. Représentation de la machine de Turing

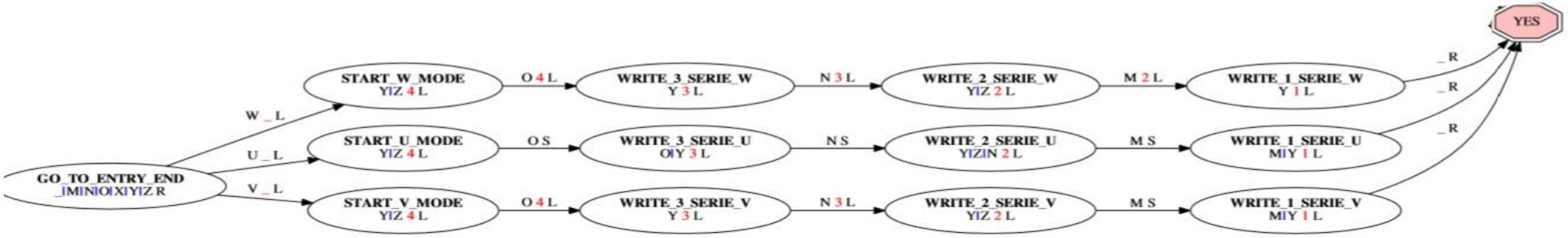
La représentation en totalité de cette machine de Turing n'est pas possible en raison de son grand nombre d'états. Tout comme pour les deux derniers nous ferons une représentation en sous-graphes :

| Sous-graphe 1 : résolution des cas particuliers.<br><br>Début : ITIALISATION / I<br><br>- K<5 Fin : ACCEPTATION / YES<br><br>- K>4 Fin : BEGIN_MNO | Sous-graphe 2 : recopie des séries de 1,2,3,4 après avoir trouvé les milieux et lu le caractère de fin.<br><br>Début : GO_TO_ENTRY_END<br><br>Fin : ACCEPTATION / YES | Sous-graphe 3 : Concaténation de trois automates de recherche de milieu pour chercher les trois milieux.<br><br>Début : BEGIN_MNO<br><br>Fin : GO_TO_ENTRY_END |
|---|---|---|

Nous représenterons uniquement les sous-graphes 1 et 2 que ne n'avons pas encore vu jusqu'ici mais pas le sous graphe 3 qui a un grand nombre d'états et est juste une combinaison de 3 machines E2 (programme efficace de recherche de milieu).




1er sous-graphe




2e sous-graphe

### 2.3.4. Quelques exemples

| Entrée $X^K$ | Sortie | Transitions |
|---|---|---|
| **k=1** | 4 | 4 |
| **k=2** | 34 | 6 |
| **k=3** | 234 | 8 |
| **k=4** | 1234 | 9 |
| **k=5** | 12344 | 60 |
| **k=6** | 123344 | 70 |
| **k=7** | 1223344 | 79 |
| **k=8** | 11223344 | 90 |
| **k=9** | 112233444 | 118 |
| **k=10** | 1122333444 | 142 |
| **k=50** | $1^{12}2^{12}3^{13}4^{13}$ | 1100 |
| **k=100** | $1^{25}2^{25}3^{25}4^{25}$ | 2580 |
| **k=200** | $1^{50}2^{50}3^{50}4^{50}$ | 5934 |
| **k=500** | $1^{125}2^{125}3^{125}4^{125}$ | 16788 |
| **k=1000** | $1^{250}2^{250}3^{250}4^{250}$ | 37542 |

- Analyse du tableau de résultat

De même que pour l'algorithme précédent, les résultats démontrent que l'algorithme est en O(nlog(n)).

# Conclusion générale

La réalisation de ce projet m'a permis d'approfondir mes connaissances en algorithmique et complexité, surtout sur la notion de machine de Turing.

# Bibliographie

Lors de la réalisation de ce projet, en plus des supports de cours que nous avons reçu, j'ai consulté les pages suivantes :


- https://fr.wikipedia.org/wiki/Machine_de_Turing
- https://fr.wikipedia.org/wiki/Analyse_de_la_complexit%C3%A9_des_algorithmes