Table des matières

Sujet

Le drapeau roumain

Le problème posé est le problème dit du « Drapeau Roumain », sur une machine de Turing à une bande

On cherche à obtenir à partir d'un mot de la forme XX[…]XXX de longueur k avec k > 0 le mot RRR[…]RRRJJJ[…]JJJBBB[…]BBB (R^rJ^jB^b), avec r <= J <= b <= r+1 avec k = r+j+b.

Par exemple, pour :

- k = 9 on obtient RRRJJJBBB
- k = 8 on obtient RRJJJBBB
- k = 7 on obtient RRJJBBB

Questions posées

Deux questions principales étaient posées :

1. Donner un premier algorithme simple pour le problème
2. Donner un algorithme efficace pour le même problème (en temps O(nlogn))

Ces deux algorithmes seront donc détailles dans ce document, par des explications, des exemples, et une analyse de la complexité de chacun.

Algorithme simple

Cet algorithme, pour lequel aucune contrainte n'était imposée, se devait juste de résoudre le problème posé. Son but était de nous permettre de nous familiariser avec l'outil Visual Turing.

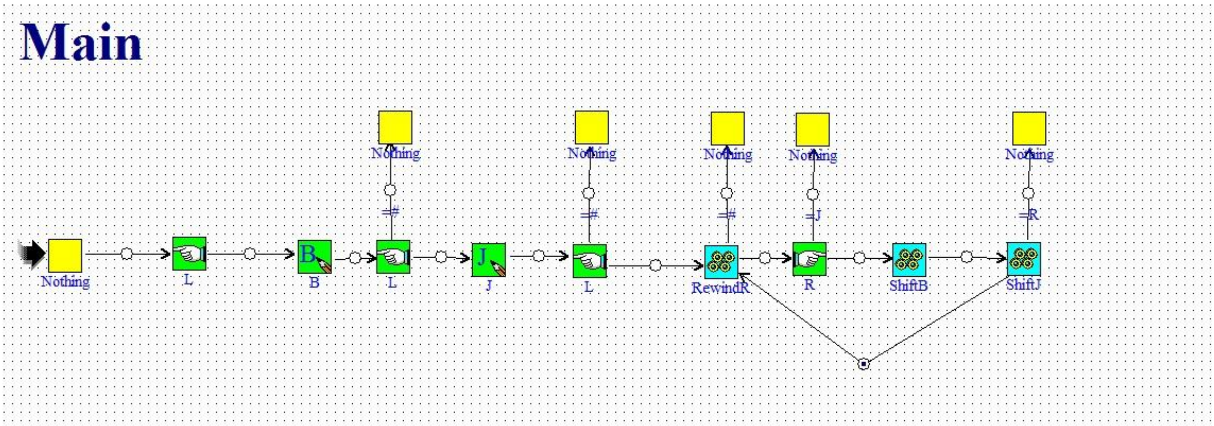

Figure 1Le corps principal de la machine

Explication de l'algorithme

Initialisation

Le principe de cet algorithme est très simple, il prend comme base l'illustration placé plus haut.
Ce morceau de machine ne représente pas son intégralité, celle-ci ayant été divisé en sous machine pour faciliter sa lisibilité.

Accompagnons son explication d'un exemple

Prenons le mot suivant : XXXXXXXX avec k = 8

Nous partons de la droite de la bande, et considérons que le mot est encadré par le symbole #

#XXXXXXXX#

On commence par écrire un B sur le premier X, puis un J juste derrière

#XXXXXXJB#

Maintenant que l'on a écrit ces deux symboles, on parcourt le mot jusqu'au dernier X pour y inscrire un R

#RXXXXXJB#

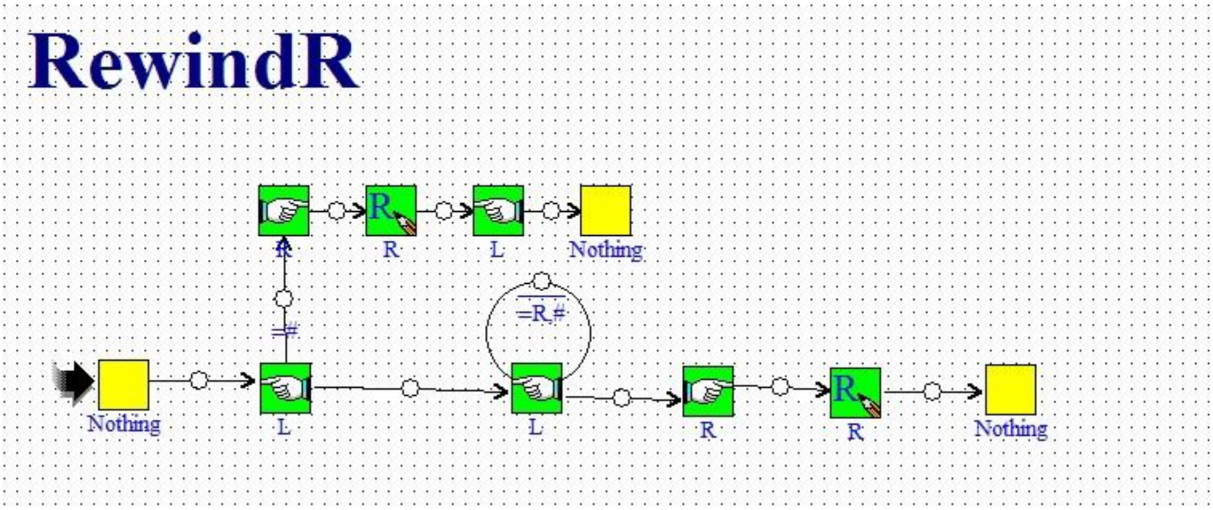

Figure 2Machine chargé d'inscrire un R au début du mot

En procédant ainsi, on s'assure de respecter la contrainte r <= J <= b <= r+1 car on pose le B avant le J, lui-même posé avant le R. On s'assure ainsi, même dans le cas où la bande fait une taille inférieur à 3, que l'on respectera bien la contrainte (#B# pour k =1, ou encore #JB# pour k = 2)

Déroulement

Une fois cette base posé, on procède de la façon suivante :

Ajout *B*

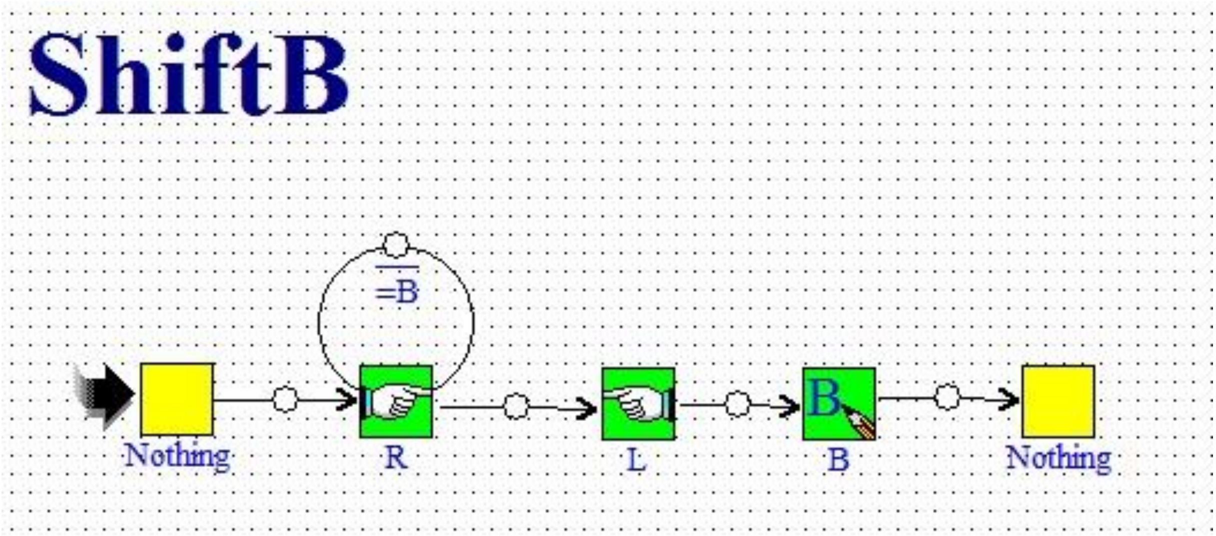

Figure 3Machine chargé d'écrire les B

La séquence démarre avec le curseur positionné sur le dernier R posé.

On décale la position sur la bande vers la droite jusqu'à rencontrer un B. Une fois positionné sur le B, on se redécale une fois vers la gauche et on ajoute un B. On obtient le mot suivant

#RXXXXXBB#

Ajout J

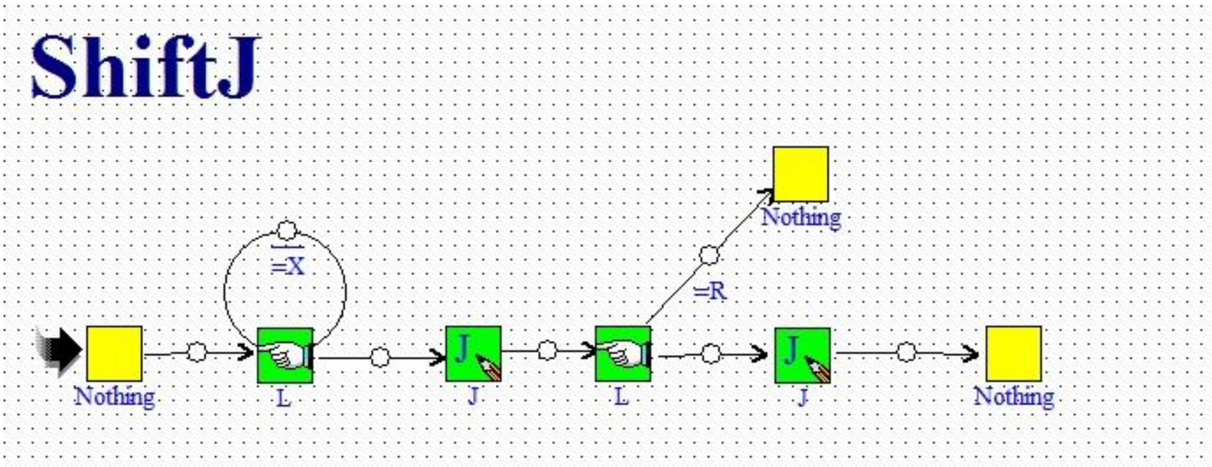

Figure 4Machine chargé d'écrire les J

Le curseur sur le dernier B posé, on se positionne sur le premier X que l'on rencontre.

A partir de là nous avons deux chose à faire :

- Il faut d'abord ré-inscrire un J sur la bande pour compenser celui effacé par l'ajout du B
- Il faut ensuite en ajouter un autre pour continuer à écrire le mot recherché

On obtient donc le mot suivant :

#RXXXJJBB#

Dans le cas où il n'y pas la place d'inscrire deux J, on en inscrit un seul. On a donc plus J^j = B^b, mais on respecte encore la contrainte initiale qui stipule que le mot est valide tant que le nombre de J est inférieur ou égale au nombre de B.

Il est aussi impossible d'inscrire un B sur un J non suivi d'un X, car cela voudrait dire qu'il est suivi d'un R. Or si jamais le nombre de X entre R et J est nul, alors la routine de la machine s'arrête car le mot est terminé.

Ajout R

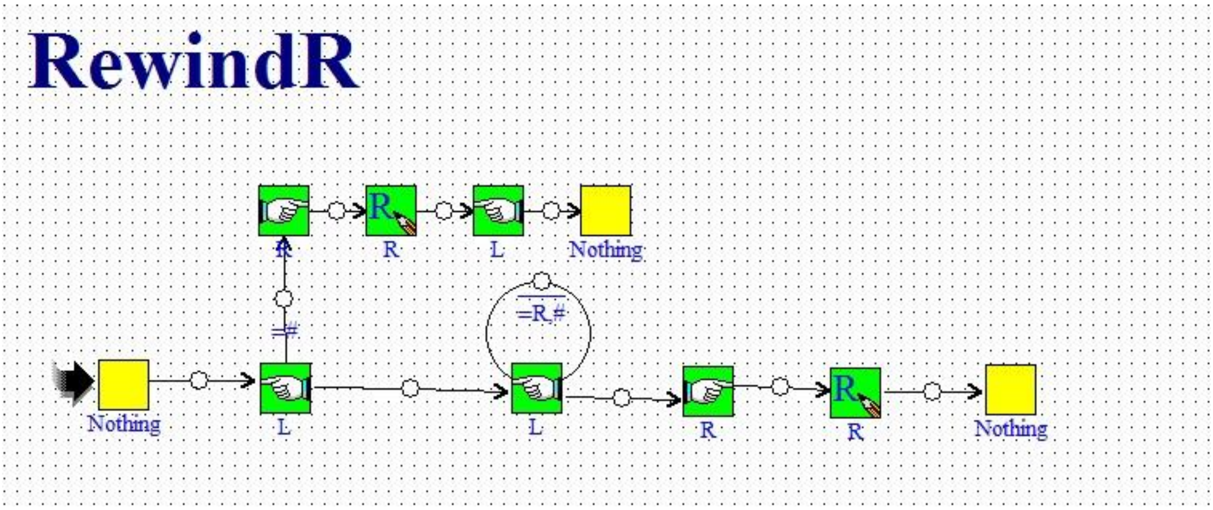

Figure 5Machine chargé d'inscrire les R à la suite du dernier R en tête

On parcourt ensuite le mot jusqu'au premier R que l'on rencontre, et on ajoute un nouveau R sur le X à sa droite.
Si jamais le R est suivi d'un J, alors on arrête d'écrire le mot puisque que celui est « terminé »

On obtient ensuite le mot suivant

#RRXXJJBB#

On répète les trois étapes ci-dessus jusqu'à atteindre un mot qui respecte les contraintes posées.
Le mot deviendra donc

N° itération	Mot
1	**#RRXXJJBB#**
2	**#RRJJJBBB#**

On obtient donc le mot #RRJJJBBB# qui respecte bien les conditions posées

Complexité

Cet algorithme est d'une complexité $O(n^2)$ dans le pire des cas.

En effet, on a n aller-retours sur la bande, chacun de ces allers-retours parcourant n-i de la bande, avec i allant de 0 à 2/3 (le dernier parcours avant a la fin du traitement survolera seulement la partie J du mot, soit 1/3 de la bande) de la bande de taille n, soit 2n/3.

On a donc une complexité au plus petit de n(2n/3), soit $2n^2/3$, donc n^2 puisque l'on peut ignorer les entiers constants dans le calcul de la complexité.

On a donc bien une complexité de $O(n^2)$.

Algorithme en O(nlog(n))

L'algorithme effectué pour la version simple bien que répondant au problème, n'est pas le plus efficace.

C'est pourquoi il était nécessaire de développer une solution plus efficace, qui toutefois est bien moins intuitive aux premiers abords.

Explication de l'algorithme

Le principe de cet algorithme est qu'à chaque parcours de la bande, nous allons effectuer le compte des caractères à écrire.

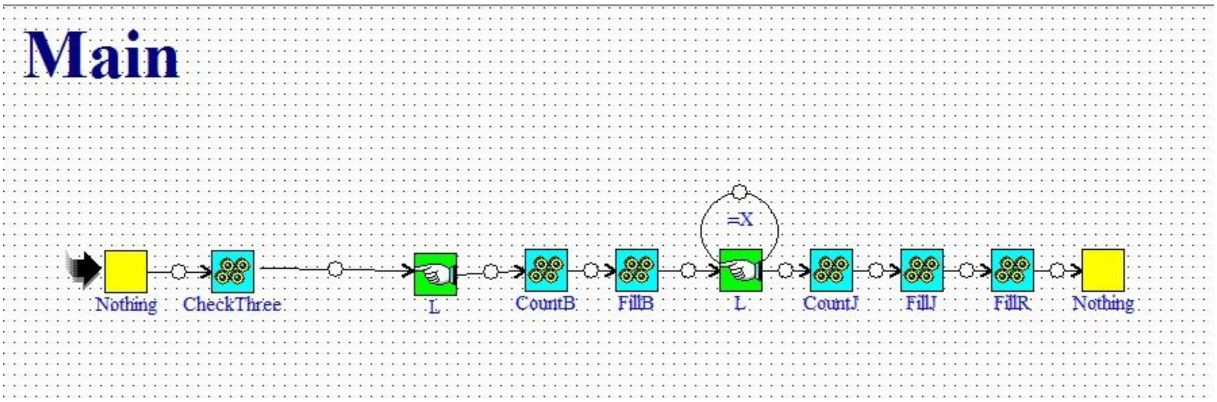

Figure 6Machine principale de l'algorithme en nlogn

Hors, il se trouve qu'une machine de Turing ne peux techniquement compter puisqu'elle ne dispose pas de mémoire.
Pour pallier à ce problème, nous tiendrons au fur et à mesure du parcours de la bande un compteur binaire, afin de garder une trace de tous les symboles à écrire une fois le parcours terminé.

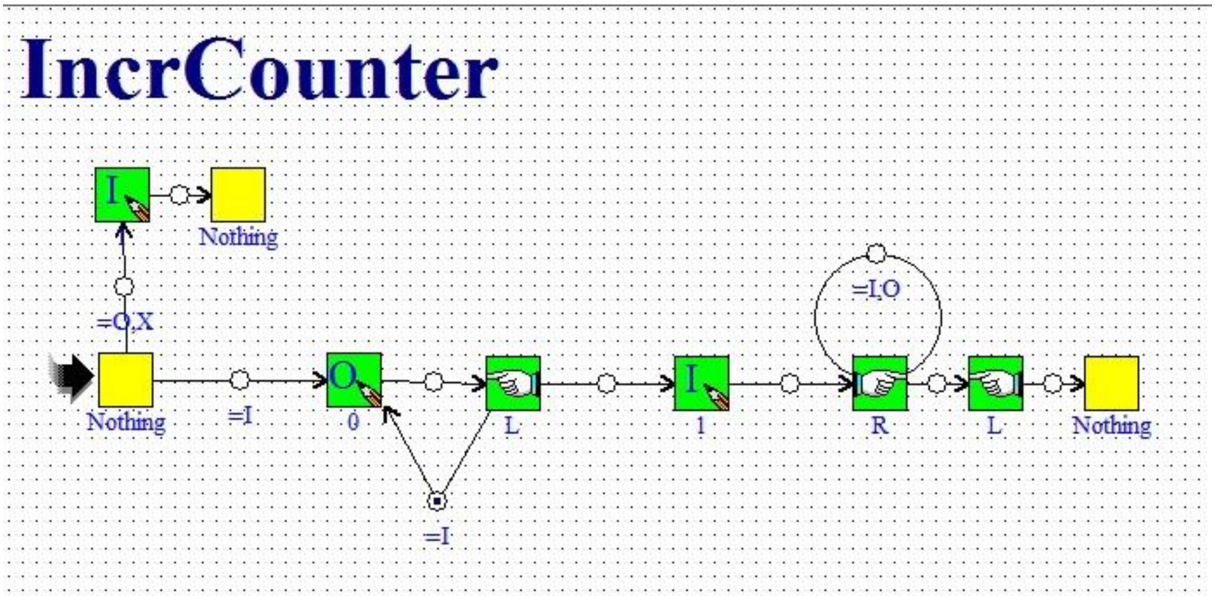

Figure 7Machine chargé d'incrémenter le compteur

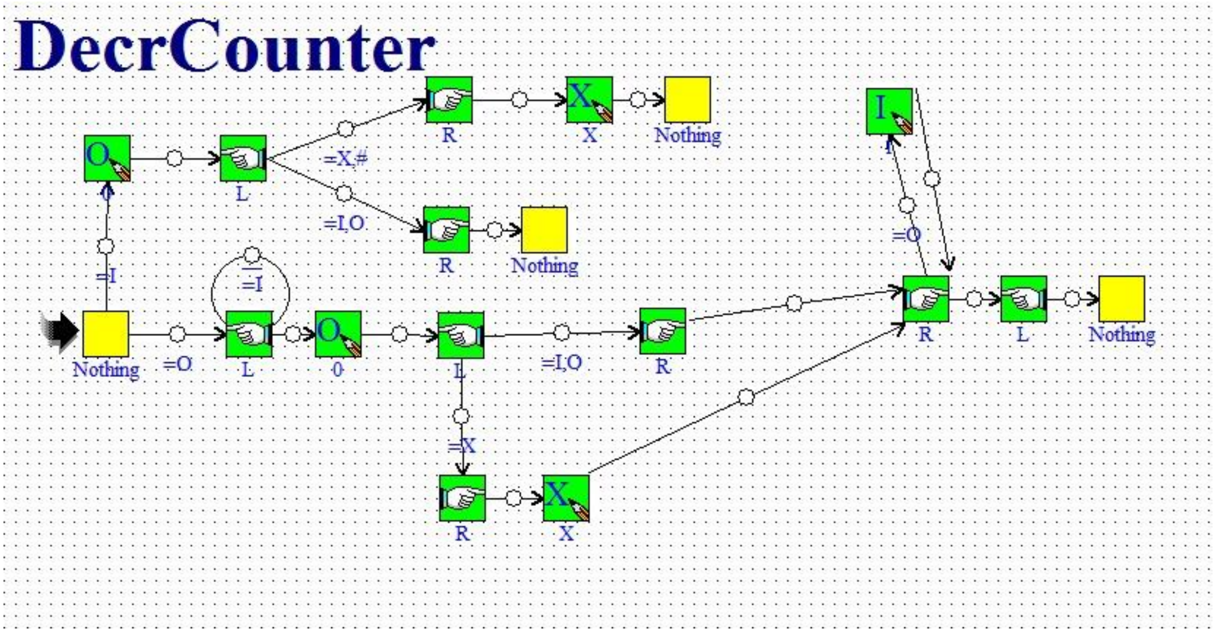

Figure 8Machine chargé de décrementer le compteur

Prenons en exemple la chaine #XXXXXX#

Compte des B

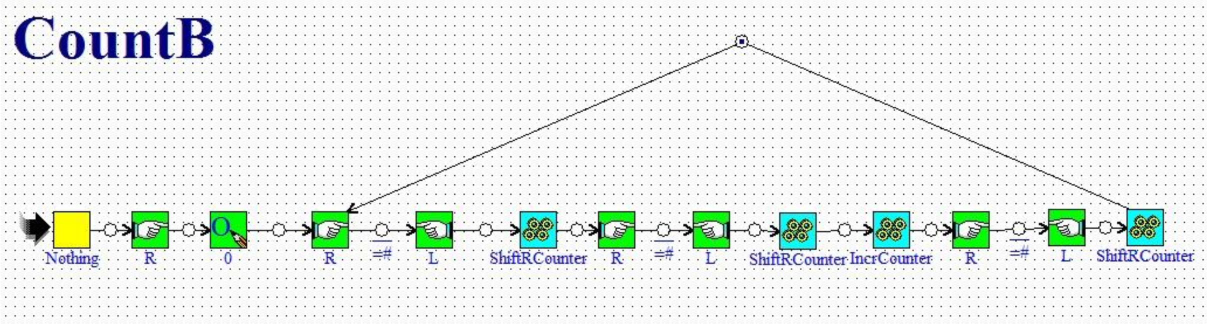

Figure 9Machine chargé de compter les B

On commence le parcours de la chaine.

Dès lors que l'on se place sur le premier X, on écrit un 1. C'est la base de notre compteur. Ainsi, à chaque déplacement du curseur, il faudra déplacer le compteur afin de toujours l'avoir sous la main pour permettre de compter le nombre de B à placer.

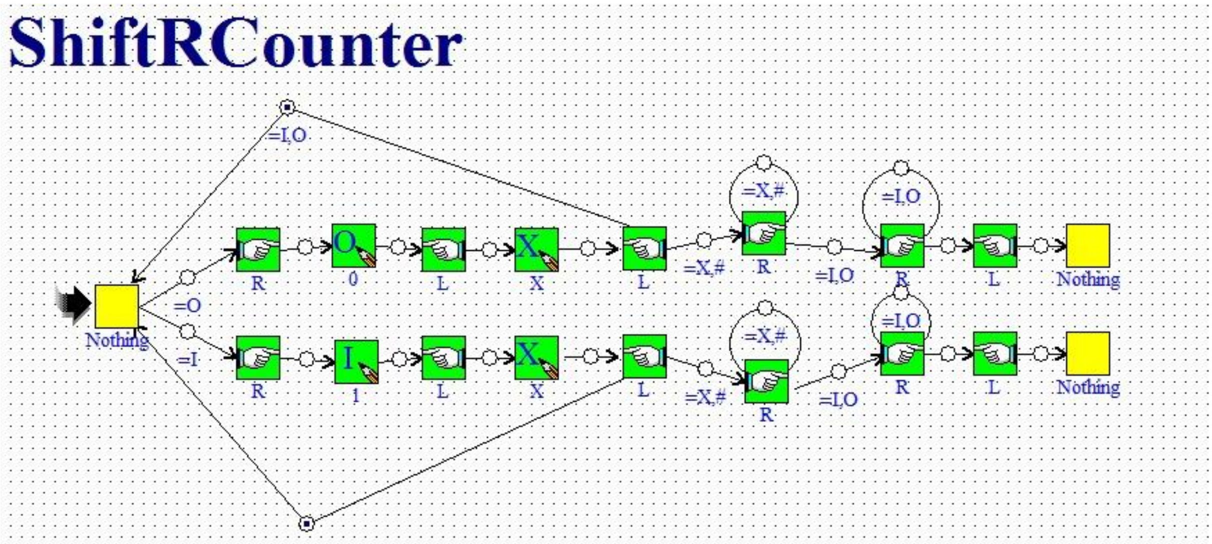

Figure 10Machine chargé de décaler le compteur vers la droite

On a donc #1XXXXX#

Il s'agira maintenant tous les 3 déplacements, d'incrémenter le compteur de 1. Ainsi on garantira le respect de la condition imposé au moment du placement des caractères.

On se retrouve donc avec #XX10XX# après trois déplacement, puis enfin #XXXX10#, avec le curseur sur le dièse.

Ecriture des B

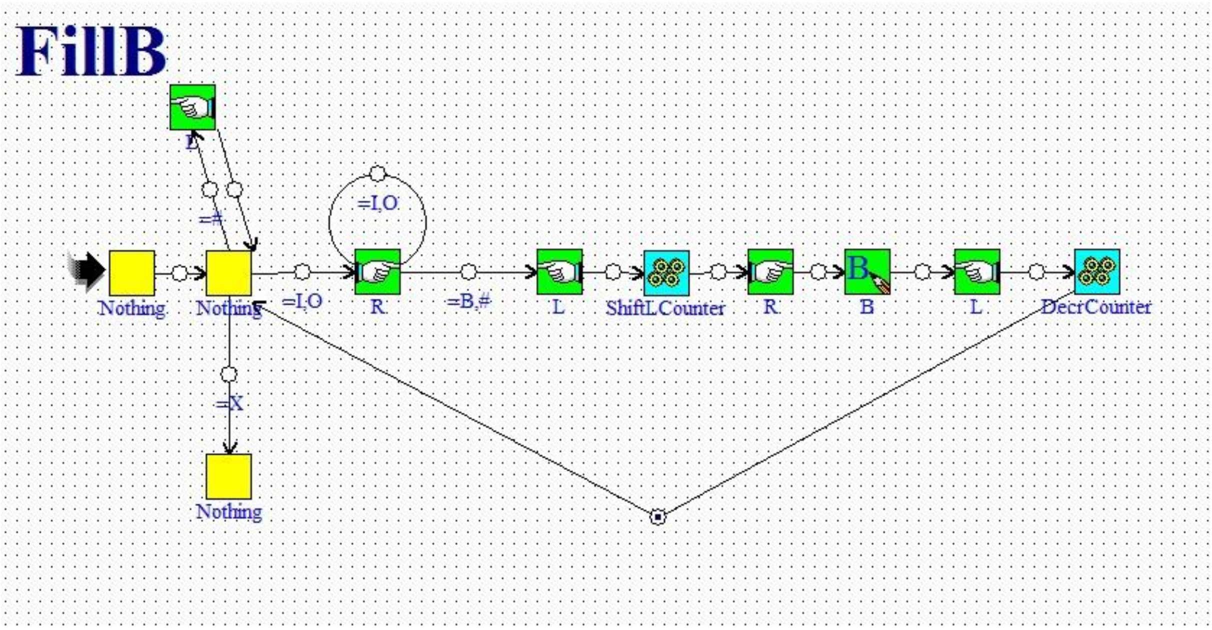

Figure 11Machine chargé d'écrire les B

Une fois la fin de la bande atteinte, on commence par consulter le compteur. S'il indique une valeur avec au moins un 1, alors c'est qu'il faut placer au moins un B.

On décale donc le compteur de 1 vers la gauche, et on inscrit un B sur la bande, puis on décrémente le compteur

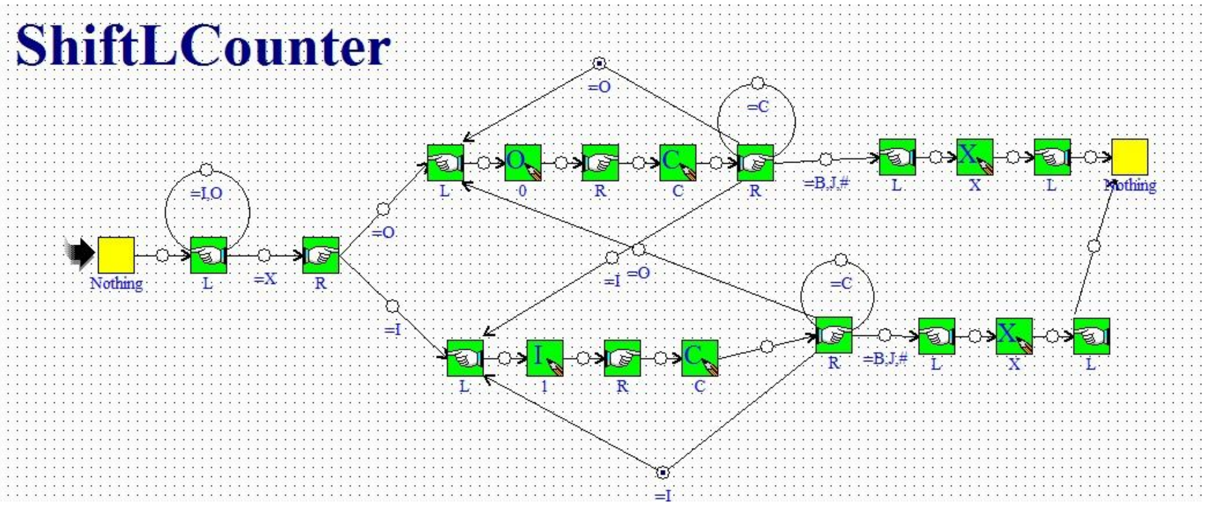

Figure 12Machine chargé de décaler le compteur vers la gauche

On obtient donc dans l'ordre :

Etat	Mot
Décalage compteur	#XXX10X#
Ecriture B	#XXX10B#
Soustraction compteur	#XXXX1B#

On continue jusqu'à disparition du compteur

#XXXXBB#

Puis on effectue la même manipulation pour les J

Compte des J

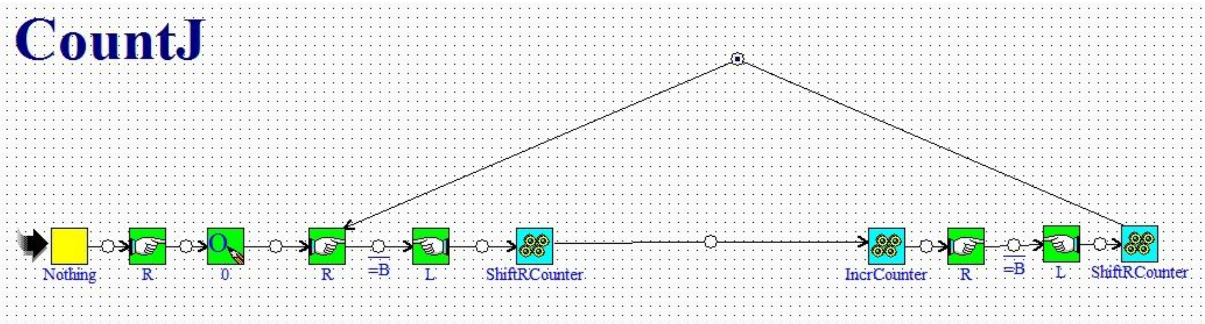

Figure 13Machine chargé de compter les J

Le compte des J s'effectue de la même manière, à la différence près que l'on incrément le compteur tous les deux shifts

On obtient donc

#XX10BB#

Ecriture des J

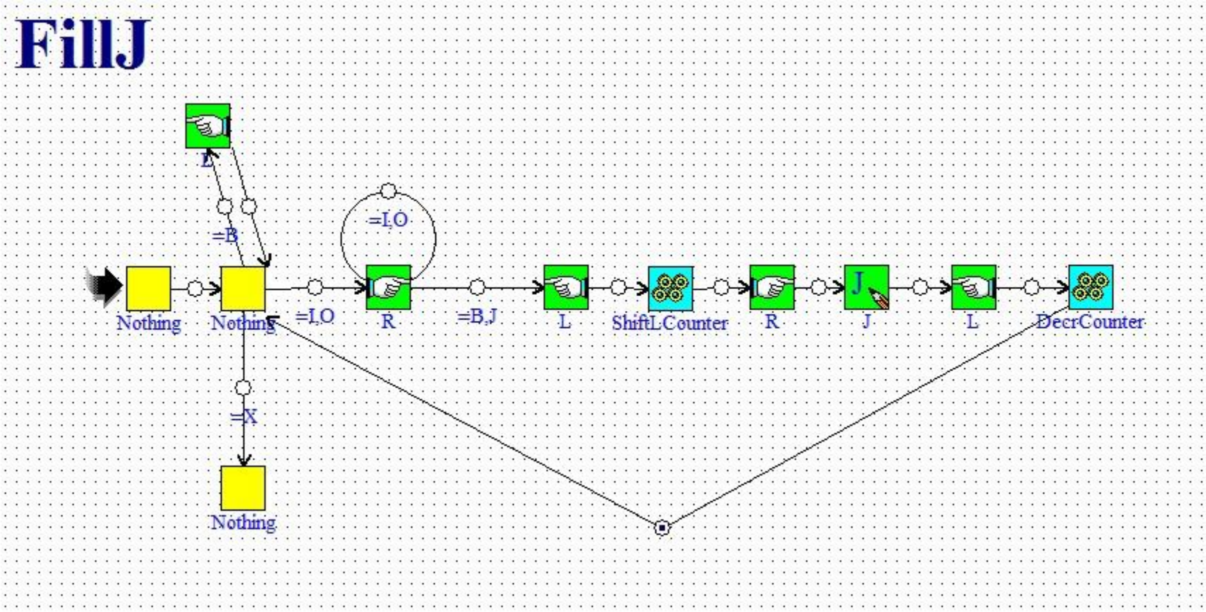

Figure 14Machine chargé d'écrire les J sur la bande

Une fois le compte effectué, même chose que pour les B, on décale et on réduit le compteur tout en écrivant les J sur la bande

#XXJJBB#

Ecriture des R

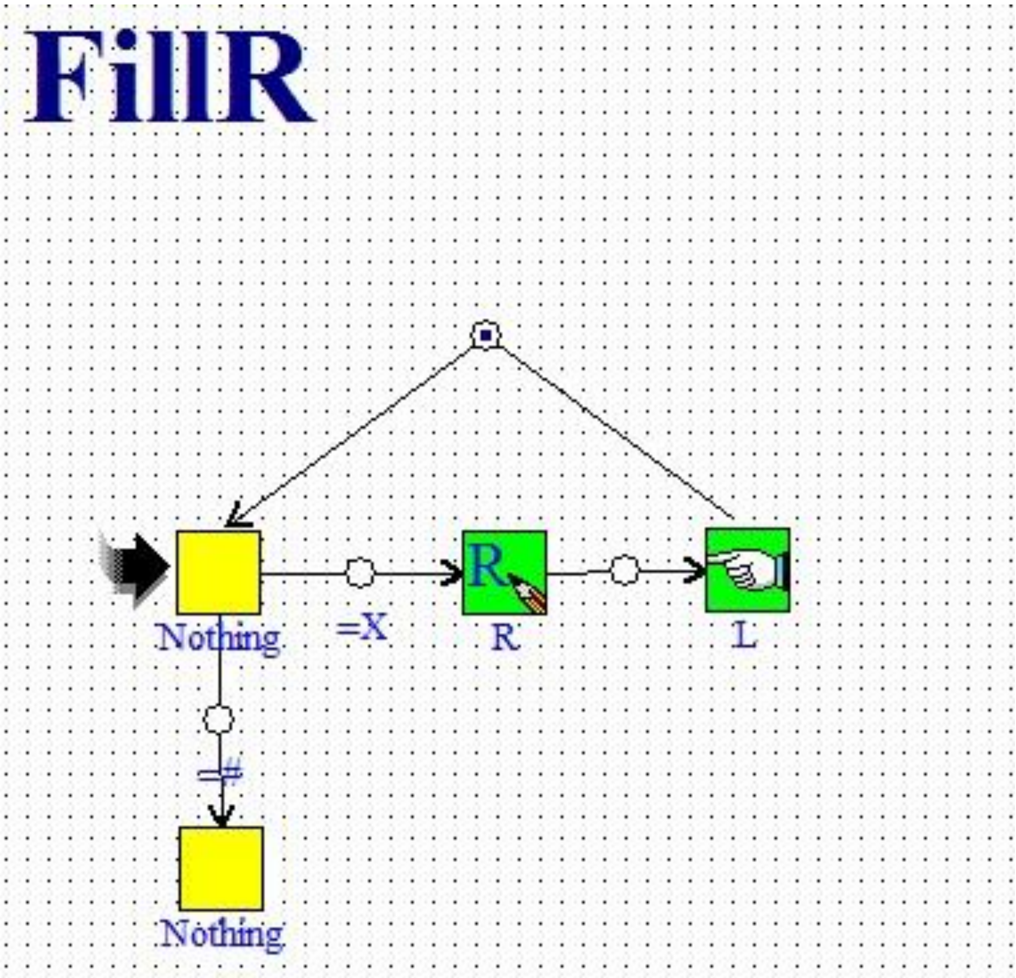

Figure 15Machine chargé d'écrire les R sur la bande

A ce moment-là, le curseur est sur le dernier J.

Il suffit donc de rembobiner vers la gauche jusqu’au dièse final, tout en remplaçant les X par des R

Compléxité

La complexité de cet algorithme est de O(nlogn) dans le pire des cas.

En effet comme nous avons pu le voir en cours, la complexité du compteur est de logn.

Le compteur est tenu tout le long des 4 parcours de la bande de longueur n.

Lors de l'écriture des B, on fait 2n parcours.

Lors de l'écriture des J, on effectue 4n/3 parcours.

Lors de l'écriture des R, on effectue n/3 parcours.

On a donc une complexité de (2n + 4n/3 + n/3) * logn

Soit 11n/3 logn, que l'on peut réduire à nlogn en éliminant les constantes.