PROJET IPF

Rapport du projet

Réalisé par : Imane EL MOUL

03 Mai 2019

Introduction

Dans le cadre de notre première année du cycle ingénieurs à l'ENSIIE, il nous est proposé par Mr. Julien FOREST un projet en programmation fonctionnelle permettant de mettre en pratique nos aquis pendant le deuxième semestre.
Le but de ce projet est d'implanter un protocole de communication avec les habitants des mondes lointains.
Plus précisément, il s'agira de prendre un message entrée et de donner la suite d'instructions permettant de piloter une (ou plusieurs) antenne(s) afin d'envoyer ledit message.
Dans l'urgence de cette nouvelle, nous avons développé un système d'antennes rudimentaire composées chacune d'une roue comportant les 26 lettres de l'alphabet et du caractère espace. Nous pouvons donc voir l'ensemble des symboles utilisables comme un cycle. Le caractère suivant A est B, ..., le caractère suivant Y est Z, le caractère suivant Z est espace suivi de A.
Toutes les antennes sont initialement positionnées sur le caractère espace.
Les commandes disponibles pour manipuler une antenne sont au nombre de 3 :
"N" : passe au caractère suivant. ;
"P" : revient au caractère précédent ;
"E" : envoie la commande d'émission du caractère courrant.

1 Les fonctions utilisé :

1.1 Phase 1 :implantation de la communication de base.

Pour cette phase, nous n'aurons à notre disposition qu'une unique antenne.
Le seul souci était de sélectionner le plus optimisé entre N et P par exemple pour aller de ' ' à 'Z' on préfére PE que NNNN...E (26 N).
Pour le faire j'avais besoin de calculer la distance entre la position actuel de la roue(initialement positionnée sur espace) et chaque lettre du message, et pour calculer ces distances j'avais besoin de créer les roues pour le faire j'ai utilisé les fonctions **index_char** qui prend en paramétres un char et renvoie un indice qui représente la position de la lettre dans la roue tel que ' ' -> 0 ; 'A' -> 1 etc... et **char_index** qui fait le contraire tel que 0 -> ' ' ;1 -> 'A' etc... ces fonctions vont me permettre par la suite d'accéder à la lettre précédente et suivante dans la roue par exemple char_index(index_char('B')-1) nous donne 'A' (le caractère précédant) et char_index(index_char('B')+1) nous donne 'c' (le caractère suivant).

Les fonctions qui calculent les distances sont :.
- **distance_N** qui permet le calcule de la distance en comptant le nombre de lettres suivant la position de l'antenne par exemple entre 'A' et 'c' il y a deux sauts de 'A' à 'B' et de 'B' à 'C' alors la distance est 2 ce que j'apelle la distance dans le sens des aiguilles d'une montre et qui va nous aidé par la suite de générer les N.
- **distance_P** qui calcule la distance entre la position de l'antenne et la lettre mais dans le sens inverse des aiguilles d'une montre ce qui va nous aidé de générer les P.
La fonction **instruction_lettre** va me permettre d'avoir l'instruction pour passer d'une lettre à une autre par exemple instruction_lettre ' ' 'A' donne NE, cette fonction est basé sur les distances si la **distance_N** est plus petite que 13 alors forcément la distance dans l'autre sens (**distance_P**) et supérieure à 13 vu qu'on est dans une roue alors elle va choisir N sinon P.
Cette fonction va être utilisé dans la fonction publique **instruction_msg** qui donne la commande total pour un message en concaténant l'instruction pour passer d'un alphabet à celui qui le suit du message. la position de la roue se mis à jour et prend la valeur de chaque lettre du message.

```
let rec instruction_msg pos msg =
match msg with
[] -> ""
|t : :q -> instruction_lettre pos t ^ instruction_msg t q ;;
```

1.2 Phase 2 :plusieurs antennes..

Dans cette phase on aura n antennes. En fonction du nombre d'antennes entré par l'utilisateur j'ai pensé à créer une liste des couple tel que le premier élément de chaque couple représente le numéro de l'antenne et le deuxième est la position de cette antenne, j'avais besoin d'une fonction qui génére la liste initial des roues identiques.
liste_roue prend en paramétre le nombre d'antennes ,la position initial des antennes (espace dans notre projet) et le numéro de la première antenne (pour nous la première antenne est s0 donc le numéro est bien 0),
liste_roue 2 ' ' 0 va nous renvoyé la liste [(0,' ') ;(1,' ')] c'est à dire deux antennes numéroté par 0 et 1 avec la position initial ' '.
le premier souci était le changement de position de ces antennes, sachant que les positions de ces roues se changent à chaque fois alors le deuxième élément des couples doit être mise à jou.Vu que

c'est impossible au point de vu fonctionnelle de modifier le contenue d'une liste j'ai utilisé une fonction qui renvoie une nouvelle liste avec le changement de la position de la roue qui a réagit.

Replace : cette fonction prend en paramétre la liste à modifier la position de l'antenne qui a réagit et le nouveau couple représentant l'antenne avec sa nouvelle position.
replace [(0,' ');(1,' ')] 0 (0,'A') renvoie la liste [(0,'A');(1,' ')]. cette fonction va nous permettre de mettre à jour les antennes après chaque utilisation.

Le deuxième souci était le choix de l'antenne la plus proche de la lettre par exemple si on a la liste des roues suivante [(0,'A');(1,' ')] et on veut atteindre 'Z' la commande PE en utilisant la roue numéro 1 est plus optimisé que PPe en utilisant la roue numéro 0. alors on doit calculer la distance entre la position de chaque roue evec la lettre cible et prendre celle qui représente la distante minimum ; en plus de faire le bon choix de la roue il faut bien sur faire le bon choix entre N et P également.
Pour le faire j'avais, avant tous, besoin de deux listes la première représente les distances entre les positions et la lettre cible dans le sens des aiguilles d'une montre et une autre liste pour les distance dans le sens contraire ; on va prendre le minimum de chaque liste qui représente le min des distances en utilisant N et le min des distance en utilisant P, entre ces deux valeur on doit choisir le plus optimisé c'est à dire la distance la plus petite entre les deux.
grouper_distance_N et **grouper_distance_P** : Elles prennent en paramétres la liste des antennes et la lettre cible et renvoient respectivement une liste de distances entre la position de l'antenne et la lettre dans le sens des aiguilles d'une montre et dans le sens inverse.
min_N_P : prend en paramètre deux listes et renvoie le minimum des minimums des ces deux listes.
Ces fonctions sont des fonctions auxilières de celle qui doit séléctionner la bonne roue.
choix_roue_lettre : prend en paramètre une liste d'antennes et une lettre cible et en utilisant les fonctions précédentes, elle renvoie le couple représentant la bonne roue.

```
let rec choix_roue_lettre l c = match l with
[] -> (0,' ')
|(a,b) : :q -> let ln = grouper_distance_N l c in let lp=grouper_distance_P l c in
let min_d = min_N_P ln lp in
if (min_d == (distance_N b c) || min_d == (distance_P b c) ) then (a,b)
else choix_roue_lettre q c;;
```

Si le minimum des minimums des deux listes ln et lp est égal à la distance entre la position du couple actuel en utilisant N ou P alors la fonction renvoie ce couple qui représente le bon choix.

Maintenant, après avoir choisit la bonne antenne il nous reste que construire la fonction qui renvoie l'instruction total d'un message en utilisant tous les fonctions précédentes.
instruction_msg_nroue : cette fonction prend en paramétre une liste de roues,un message représenté par une liste de caractères et un entier qui va être utilisé pour distingué la première vu que pour la première lettre on utilisera toujours l'antenne S0. **SI** cet entier est égal à 0(ce qui va être vrai que pour la première fois) alors on va directement remplacer la position de la première antenne par la première lettre du message et concaténer avec l'insruction générer par instruction_lettre (qui représente l'instruction du passage de ' ' à la première lettre du message). **SINON** il doit choisir la bonne roue (en utilisant choix_roue_lettre), construire une nouvelle liste dans laquel on va changer la position de la roue active par sa nouvelle position(en utilisant replace), ensuite concaténer "s", le numéro de l'antenne active, l'instruction de passage à la lettre courante par rapport à la position de la roue active et récursivement faire la même chose avec le reste des lettres et à chaque fois en utilisant la liste des antennes mise à jour.

```
let rec instruction_msg_nroue l msg i =
match msg with
[] -> ""
|t : :q ->if (i == 0 ) then
let l'=replace l 0 (0,t) in instruction_lettre ' ' t ^ instruction_msg_nroue l' q (i+1)
else let roue_active = choix_roue_lettre l t in let l'=replace l (fst(roue_active)) (fst(roue_active),t)
in
"s"^ string_of_int(fst(roue_active))^ instruction_lettre (snd(roue_active)) t ^ instruction_msg_nroue l' q
(i+1);;
```

D'après le sujet on sait que la deuxième méthode qui consiste sur l'utilisation de plusieurs antennes est plus rapide que la première, c'est pour ça que je dois mettre en place une fonction qui permet de calculer le temps nécessaire pour l'émission de message en utilisant chaque solution sachant que le changement de place d'une roue prend 3 secondes, la sélection d'une antenne prend 1 secondes et L'émission prend 5 secondes.
time :Cette fonction prend en paramétre une commande et renvooie le temps qu'elle demande. Elle va tout simplement analyser chaque lettre de la commande et rajouté le temps nécessaire pour chaque action (N, E, P,s).
communication_time :cette fonction prend en paramétre un message ,le nombre de roues et applique,selon n, la sortie de *instruction_msg* pour n=1 et *instruction_msg_nroue* pour n>1 sur la fonction time.
compare_time :selon la valeurenvoyée par communication_time cette fonction fait la comparaison entre l'utilisation d'une seule roue et deux roues. Si l'utilisation de plusieurs roues et plus optimisé que l'utilisation d'une seule roue elle va proposer à l'utilisateur de rajouter des antennes et vice versa.

1.3 Regrouper les deux phase

Afin d'avoir un seul projet englobant les deux phases j'ai choisis de créer une fonction **communication** qui prend en paramétre le nombre de roues et une liste de caractéres représentant le message et en fonction de n elle applique *instruction_msg* ou *instruction_msg_nroue*

```
let communication n msg =
if(n == 1) then instruction_msg ' ' msg
else let roues = liste_roue n ' ' 0 in instruction_msg_nroue roues msg 0;;
```

2 les Tests

Pour tester j'ai déclaré un couple dont le premier élément est n ,le nombre d'antennes ,et le deuxième est le message sous forme d'une liste de charcatères en utilisant la fonction parse_input() donnée dans le sujet.

```
let entre = parse_input();;
let msg = snd(entre);;
let n = fst(entre);;
```

ensuite afficher la fonction **communication** et **compare_time**

```
Printf.printf "%s %s" (communication n msg)(compare_time n msg);;
```

2.1 Mes tests

2.1.1 message = "IMANE ELMOUL"

le message IMANE ELMOUL prend 306s avec une seule roue et 212s avec deux roue alors il m'a proposé d'ajouter des roues.

```
Entrez le nombre de roues et votre message en MAJUSCULE
1
IMANE ELMOUL
NNNNNNNNNENNNNEPPPPPPPPPPPPPENNNNNNNNNNNNNEPPPPPPPPPEPPPPPENNNNNENNNNNNNENENNENNNNNNEPPPPPPPPPE
time=306s
en utlisant 2 roues je vais passé 212s
Pensez à ajouter des roues c'est plus optimisé!
```

le message IMANE ELMOUL prend 212s avec deux roues

```
Entrez le nombre de roues et votre message en MAJUSCULE
2
IMANE ELMOUL
NNNNNNNNNEs0NNNNEs1NEs0NEs1NNNNEs1PPPPPEs1NNNNNEs0PPEs0NEs0NNEs0NNNNNNEs1NNNNNNNE
time=212s
```

le message IMANE ELMOUL prend 182s avec trois roues

```
Entrez le nombre de roues et votre message en MAJUSCULE
3
IMANE ELMOUL
NNNNNNNNNEs0NNNNEs1NEs0NEs1NNNNEs2Es1Es0PPEs0NEs0NNEs0NNNNNNEs1NNNNNNNE
time=182s
```

dans cet exemple plus d'antenne -> plus rapide

2.1.2 message = "ENSIIE"

le message ENSIIE prend 129s avec une seule roue

```
Entrez le nombre de roues et votre message en MAJUSCULE
1
ENSIIE
NNNNNENNNNNNNNNENNNNNEPPPPPPPPPPEEPPPPE
time=129s
```

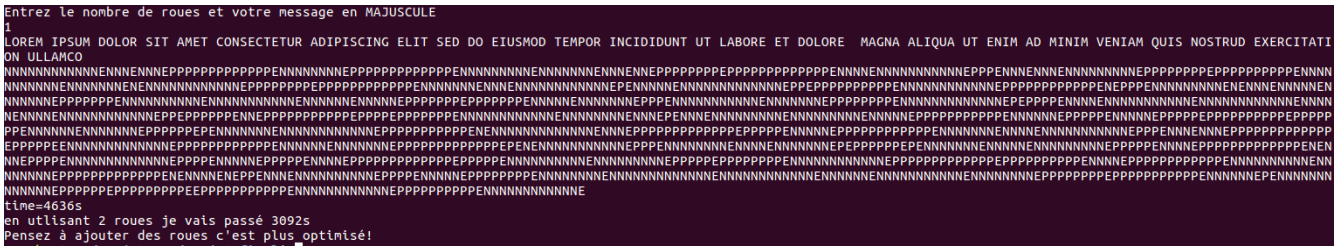
le message ENSIIE prend 131s avec deux roues

```
Entrez le nombre de roues et votre message en MAJUSCULE
2
ENSIIE
NNNNNEs0NNNNNNNNNEs0NNNNNEs1NNNNNNNNNEs1Es1PPPPE
time=131s
avec une seule roue je vais passer 129s
pour ce message c'est mieux d'utiliser une seule roue
```

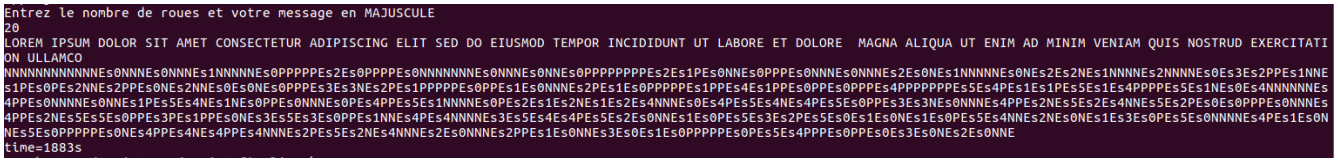
dans cet exemple plus d'antenne -> moins rapide Alors dans ce cas l'utilisation d'une seule roue est plus optimisé que deux roues Vu que les lettres de ENSIIE sont proche les uns des autres dans l'ordre des alphabets alors il va utiliser la même roue pour la plupart des lettres ce qui est équivalent à l'utilisation d'une seule roue mais à chaque fois il va se tarder 1s pour séléctionner la bonne roue.

2.1.3 message = du texte aléatoire (lorem impsum)

le texte aléatoire prend 4636s avec une seule roue
Il prend 3092s avec deux roues



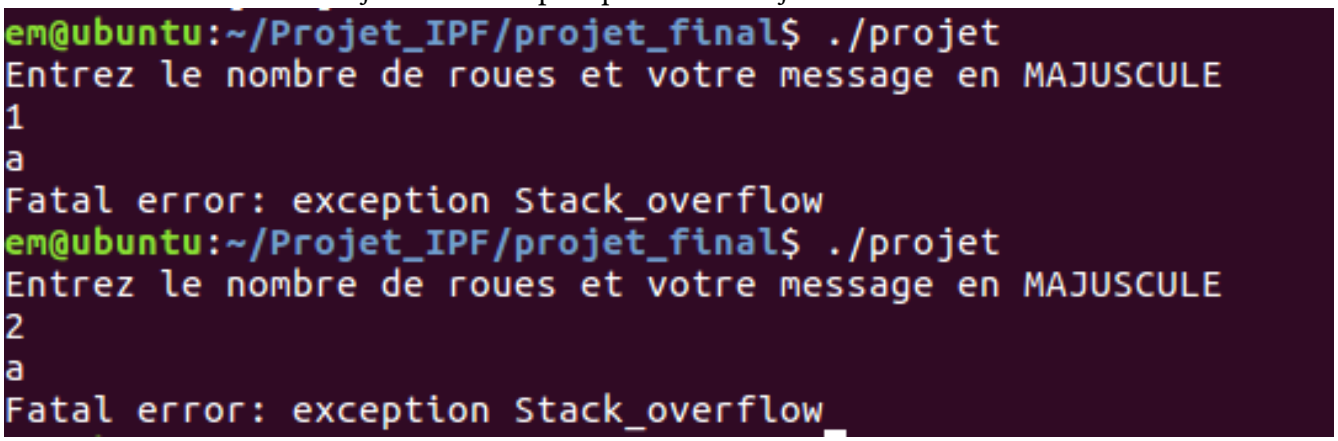
le texte aléatoire prend 1883s avec 20 roues



Dans cet exemple plus de roues -> plus rapide

2.2 Limites du projets

Ce code ne gère que les alphabets en majuscule en plus du caractère espace, pas des caractères spéciaux, pas des nombres, et surtout pas des minuscules, sinon on aura un fatal error à cause d'une récursion infinie. cette récursion infinie vient du fait que ocaml est sensible à la casse alors les fonctions des distances ne vont jamais arrêté puisque 'A' est toujours différente de 'a'.



Conclusion

Ce projet représentait une vraie mise en pratique de mes connaissances initiatiques en Programmation fonctionnelle en utilisant le langage ocaml. Durant ce projet j'ai rencontré des problèmes surtout avec la récursion des fonctions vu qu'il y avait un parcours des listes ou des chaines de charactères et pour moi un parcour était toujours lié aux boucles, sauf que la récursion des fonctions a bien fait l'affaire bien sûr avec l'aide d'aures fonctions auxilière qui sont eux même des fonctions récursives. Ce projet était une vraie opportunité pour moi de découvrir, d'une façon initial, le concept de la programmation fonctionnelle.