École Nationale Supérieur
d'Informatique pour l'Industrie et
l'Entreprise

Projet Programmation Fonctionnelle

Communication avec nos voisins les extra-terrestres

Elève
M. Nicolas Makaroff

Encadrement
M. Julien Forest

Contents

1 Introduction

Le projet se décompose en deux parties, la première étant une version simplifiée de la seconde. Le but est de transmettre un message à travers la manipulation d'antenne à une forme de vie située sur une autre planète qui tente de rentrer en contact avec nous.
La solution proposée ici n'est pas optimale car elle ne prend pas en compte le temps d'envoie le plus court sur un mot mais seulement lettre par lettre. Une esquisse d'une autre implémentation est proposée en section 3. L'exécutable présent dans l'archive ne fonctionne qu'avec la partie 2 puisque qu'il suffit de donner une seule antenne pour se retrouver dans la situation de la partie 1.

2 Partie 1

J'ai pris la décision en abordant cette partie 1 de ne pas prendre en compte l'existence de la deuxième afin d'obtenir un code le plus clair possible. Les premiers temps ont été consacré à chercher une modélisation du problème à l'aide des outils informatiques que nous connaissons. J'ai d'abord entreprit de représenter les antennes par des *Zippers* mais cet outils m'est apparu par la suite inutilement complexe ce qui m'a conduit à me concentrer sur la structure de *List* fournit par la fonction de lecture d'entrée.
Les lettres étant transposable en entier le calcul des distances entre deux lettres semblait être une solution simple et efficace. Une nouvelle difficulté est alors apparu qui est le caractère *espace* car celui-ci est *loin* des lettres de l'alphabet. Il a donc fallu considéré à part ce cas particulier. La fonction faisant la majorité du travail sur cette partie *creation* se contente de calculer des distances et de faire le bon choix de sens de rotation.

3 Partie 2

La plupart de cette partie s'est faîte sur papier. Le travail a été divisé en trois étapes :

1. Compréhension de la difficulté
2. Choix de représentation des antennes
3. Construction de fonctions sur les antennes
4. Réutilisation de la Partie 1

3.1 Compréhension de la difficulté

L'apport de cette seconde partie de projet est la prise en compte d'un nombre inconnue d'antenne disponible pour transmettre le message et l'envie d'optimiser le temps d'envoie.
Il faut donc pouvoir considérer chaque antenne comme des entités distinctes. Une fois cette difficulté résolue, il faut être capable de faire switcher l'algorithme entre les antennes et d'indiquer quelle antenne est utilisée.

3.2 Choix de la représentation

J'ai fait le choix de représenter une antenne par un enregistrement contenant plusieurs informations sur celle-ci.

- un identifiant d'antenne : son nom avec nom $\in \{S0, ..SN\}$
- sa position courante initialisée sur l'espace
- un entier représentant la distance au prochain caractère du message initialisé en 0

Listing 1: type antenne

```
type antenne={num:char list; pos: char ; dis: int};;
```

3.3 Construction de fonctions sur les antennes

J'ai par la suite écrit quelques fonctions pour manipuler ce nouveau type ainsi qu'une liste d'antenne. Ces fonctions permettent essentiellement de simplifier la compréhension du code et d'empêcher les mauvaises manipulation.
Une description de leur fonctionnement se trouve dans le document d'explication des fonctions.

3.4 Réutilisation de la partie 1

L'envie ici a été de réutiliser les résultats fournis par la partie 1 mais sur les différentes antennes indépendamment. Cela se fait par tout d'abord être capable de choisir la bonne antenne puis appliquer la partie 1. Ceci n'a pas été très compliqué puisque le choix a été simplifier par la troisième entrée *dis* du type *antenne* en écrivant une fonction trouvant la distance minimum sur une liste d'antenne.

4 Pour aller plus loin

Comme expliqué en introduction, cette solution ne donnera pas le meilleur résultat sur certains mots particuliers comme par exemple $['E';' B';' Z']$ avec 2 antenne (ou encore $['A';' B';' A';' B';' A';' B';' A';' B';' A';' B']$). On obtiendra ici $31sec$ alors qu'on aurait au mieux $25sec$. On obtient ainsi quand même une différence de 6 sec ce qui pourrait ne pas être négligeable d'autant plus que le contre-exemple donné est relativement simple.
J'ai voulu alors trouvé d'autre modélisation afin de résoudre cette difficulté mais le réalisation n'ont pas été mené jusqu'au bout.
Deux choix se proposaient à moi :

- les arbres
- les graphes

Les arbres ont été mis de côté parce que la complexité en temps était très élevé est que mon niveau dans ce domaine me permettait pas de représenter optimalement le problème.

Par contre, les graphes me semblaient plus simple d'utilisation. Une modélisation du contre exemple donnerait en utilisant les graphes Potentiel-Tâches un graphe assez simple mais rapidement inutilisable. En effet, l'utilisation de Dijkstra amène à un mauvais résultat à cause du chemin entre $'E'$ et $'Z'$. Il a donc fallu rajouter des sommets fictifs afin de résoudre la mauvaise représentation. Le problème étant alors de faire comprendre à un ordinateur comment créer un tel graphe à partir d'une seule liste de caractères. On trouvera dans le fichier *autre.ml* une implémentation du graphe pour l'exemple $E; B; Z$ et l'application de l'algorithme de Dijkstra qui renvoie le parcours entre les lettres et les antennes afin de savoir quand bien utiliser un changement d'antenne.

5 Conclusion

Ce projet m'a permis de me confronter à de réelles difficultés de recherche de bonne solution et modélisation afin de ne pas écrire du code imbuvable au premier regard, ce qui comme passionné de mathématiques est particulièrement plaisant. La rigueur du language OCaml étant parfaitement adéquate pour cela.