# **Projet** de programmation fonctionnelle

## `Planificateur de tâches dans une base martienne`

réalisé en *OCaml*

Justin BABONNEAU

5 janvier 2023

# Introduction

L'objectif de ce projet était de proposer un planificateur de déplacements dans une base martienne, avec pour principale contrainte de le réaliser en utilisant exclusivement la partie fonctionnelle d'OCaml.

Une base martienne est composée de modules tous suffisamment spacieux pour contenir la totalité de la colonie. Le passage entre deux modules se fait via un système de tunnels de longueurs variables et chacun ne permettant que le passage d'une personne à la fois. Bien entendu, nous désirons que le système s'adapte à tout plan de base que nous lui soumettons.

Notre colonie s'est développée en trois étapes :

1. dans un premier temps, pour des problèmes de sécurité, notre colonie a été réduite à un seul individu ;
2. dans un second temps, le développement aidant, de nombreux individus ont peuplé notre colonie mais le système avait la simple tâche de réguler leurs déplacements ;
3. enfin, la capacité de calcul aidant, le système a permis la planification totale des déplacements.

# Table des matières

**1 Phase I** **4**
1.1 Structures de données . . . . . 4
1.2 Algorithme de recherche de parcours optimal . . . . . 4

**2 Phase II** **6**
2.1 Structures de données . . . . . 6
2.2 Algorithme de recherche de meilleure combinaison d'itinéraires . . . . . 6

**3 Phase III** **8**
3.1 Structures de données . . . . . 8
3.2 Algorithme de gestion des voyageurs . . . . . 8

**A Annexes** **9**
A.1 Compléments de code . . . . . 9
A.2 Codes de tests . . . . . 10
A.2.I Phase I . . . . . 10
A.2.II Phase II . . . . . 14
A.2.III Phase III . . . . . 16

# 1 Phase I

Pour cette phase, seul le déplacement d'un unique individu souhaitant aller d'un module de la station à un autre à travers le réseau de la base doit être géré. L'objectif est de trouver l'itinéraire qui minimise le temps de parcours.
Pour ce faire, nous connaissons son emplacement d'origine et celui qu'il souhaite atteindre, ainsi que le plan de la base, récupéré sous forme d'une liste de triplets indiquant le trajet entre deux modules et son temps de parcours.

## 1.1 Structures de données

Puisque les tunnels reliant les modules sont à double sens, une structure capable de récupérer le temps de trajet entre deux stations s'avérait nécessaire. Elle devait également permettre à l'algorithme de recherche de rapidement trouver le temps séparant deux modules, sans avoir à parcourir l'ensemble des données jusqu'à trouver celle souhaitée (si elle existe).
C'est pourquoi, le plan de la base était au début enregistré sous la forme d'une `Map` de `Maps` d'entiers, et dont la clé associée à un élément dans une `Map` est une chaîne de caractères.

La clé d'un élément de la `Map` principal correspond au nom d'un module. Cet élément est une sous-`Map` dont la clé de chaque élément est le nom d'un module vers lequel on peut se rendre directement à partir du premier. Ces éléments étaient au début des entiers correspondant au temps de parcours entre ces deux modules. Cependant, à partir de la phase II, l'utilisation d'un couple d'entiers s'est avérée nécessaire, ce qui explique que l'algorithme de la phase I utilise cette nouvelle structure de données, mais sans s'occuper du deuxième entier du couple.

## 1.2 Algorithme de recherche de parcours optimal

Cet algorithme prend en entrée le plan de la base, la station de départ et celle d'arrivée du voyageur, inscrites comme un couple de chaînes de caractères.

Pour trouver l'itinéraire le plus rapide, le plan de la base est considéré comme un arbre dont la racine est la station de départ. Chaque branche de l'arbre fait apparaître au maximum une fois une station. La recherche se fait en profondeur.
Le premier but est d'arriver jusqu'à un noeud de l'arbre dont la clé est la station d'arrivée. Tant qu'il n'y est pas parvenu, il parcours l'ensemble des chemins. Lorsqu'il trouve un noeud correspondant à la station d'arrivée, il s'arrête et enregistre le temps de parcours comme temps minimal.
Le prochain objectif est de trouver un itinéraire dont le temps de parcours est strictement inférieur à celui trouvé précédemment. Pour ce faire, l'algorithme parcourt l'ensemble des branches qui n'ont pas encore été explorées. S'il n'est pas encore tombé sur un noeud de la station d'arrivée et que le temps de trajet dépasse le temps minimal trouvé jusqu'à présent, la recherche s'arrête dans cette branche. En revanche, s'il trouve un itinéraire entre les deux modules dont le temps est plus court que l'ancien, on l'enregistre comme nouveau temps minimal.
L'action est répétée jusqu'à ce que toutes les branches de l'arbre aient été explorées. Si plu-

sieurs itinéraires présentent le même temps de parcours, celui trouvé en premier sera renvoyé. Si aucun chemin n'est trouvé, il lève une exception.

Cet algorithme permet de connaître à coup sûr l'itinéraire le plus rapide entre deux stations, puisqu'il essaye toutes les possibilités, mais le fait intelligemment en arrêtant le calcul sur un chemin lorsque le temps de trajet a dépassé le temps minimal trouvé précédemment.

# 2 Phase II

Lors de cette phase, on doit être capable de trouver le meilleur ordonnancement entre plusieurs itinéraires donnés en entrée du programme. Ce dernier devra alors afficher ces itinéraires en indiquant l'heure de départ des voyageurs dans chaque tunnel reliant un module à un autre. La principale contrainte de ce problème réside dans le fait que deux personnes ne peuvent pas se trouver dans le même tunnel en même temps, que ce soit dans un sens comme dans l'autre.

## 2.1 Structures de données

Afin de connaître l'heure de départ d'une station d'un voyageur, la structure d'un itinéraire utilisée pour le calcul est une liste de couples. Ces couples sont formés d'une chaîne de caractères permettant de stocker le nom du module, et d'un entier représentant le temps de départ correspondant. En revanche, la structure utilisée pour son affichage est un couple de listes : une pour le nom des stations et une pour les temps de départ.

L'ensemble des itinéraires est stocké dans une liste, permettant ainsi à l'algorithme de suivre un ordre de résolution précis.

La structure de données correspondant au plan de la base est la même que celle de la phase I. Il s'agit d'une `Map` de `Maps` de couples, eux-mêmes constitués de deux entiers : un représentant le temps de trajet entre deux modules, l'autre servant à stocker le temps restant avant que ce trajet ne soit libéré et puisse accueillir un autre usager.

## 2.2 Algorithme de recherche de meilleure combinaison d'itinéraires

La première action réalisée par l'algorithme à la réception des données est de classer la liste des itinéraires par temps de trajet décroissant, puis par nombre de modules décroissant ; le haut de la liste est donc constituée des trajets les plus longs en temps et passant par le plus de stations. Ces trajets sont considérés comme prioritaires dans la gestion des conflits, car ce sont eux qui pourront le moins s'adapter aux autres.
L'algorithme lance ensuite le voyage de nos usagers virtuels, en commençant par les itinéraires les plus longs. Lorsqu'un individu s'engage dans un tunnel, le temps de départ de ce dernier est mis à jour dans la liste correspondant à son itinéraire. Dans la structure de la base, le temps de traversée d'un tunnel est attribué au temps d'attente avant d'être à nouveau disponible. Dès qu'un voyageur souhaite s'engager dans un tunnel en cours d'utilisation, il est mis en attente jusqu'à que ce soit son tour.
Lorsque tous les voyageurs pouvant s'engager dans un tunnel l'ont fait, on incrémente le temps total de 1 et on décrémente tous les temps d'attente non nuls de la structure de la base.
Cette procédure est répétée jusqu'à ce que tous les voyageurs soient arrivés à destination. On aura alors accès à leur heure de départ pour chaque module, ainsi qu'au temps d'arrivée global.

L'algorithme utilisé est un algorithme glouton ; il ne donne pas forcément la solution la plus optimale mais s'en approche avec une faible utilisation des ressources. Compte tenu de la faible capacité de calcul présente à ce stade sur la base, ce type d'algorithme semble ainsi être un bon choix.

Dans le but d'obtenir de meilleurs résultats, mais avec une utilisation des ressources presque identique, une amélioration a été apportée. Il a été remarqué que le plus souvent, faire attendre pendant un cycle un voyageur arrivé dans une station pour faire partir un autre pas encore arrivé et dont le trajet est plus long, est souvent plus intéressant. C'est pourquoi, si le système remarque qu'un de ses utilisateurs est dans ce cas et arrive dans un module au prochain cycle, l'autre usager est mis en attente.

La dernière étape ayant lieu après la résolution du problème est la conversion de la structure des itinéraires, passant d'une liste de couples à un couple de listes. Cependant, un simple `List.split` ne suffit pas, étant donné que la liste des stations comprenant $n$ stations voit sa liste de temps associée être de longueur $n - 1$.

# 3 Phase III

Lors de cette phase, le système doit être capable d'ordonnancer le déplacement de plusieurs individus à la fois, avec pour seules données d'entrées le plan de la base et les modules de départ et d'arrivée des différents usagers.

## 3.1 Structures de données

Les structures de données utilisées sont les mêmes que celles employées lors de la phase II.

## 3.2 Algorithme de gestion des voyageurs

L'algorithme de la phase I établissant le meilleur itinéraire entre deux modules, et celui de la phase II pouvant donner un ordonnancement presque optimal de plusieurs itinéraires, ces deux algorithmes ont été intégralement réutilisés pour la résolution de la phase III.
Pour commencer, on lance l'algorithme de recherche de parcours optimal sur l'ensemble des couples (station de départ, station d'arrivée) donnés en entrée, dont on stocke les résultats dans une liste. On donne ensuite cette liste à l'algorithme de recherche de meilleure combinaison d'itinéraires, qui se charge de l'ordonner et d'en tirer une bonne combinaison minimisant au mieux le temps de parcours global des voyageurs.

Cette solution comprend un algorithme (celui de la phase II) dont le résultat n'est pas toujours optimal, ce qui le rend lui-même imparfait. Cependant, même en considérant que l'algorithme utilisé donne une solution parfaite, l'utilisation du meilleur itinéraire d'un usager entre deux stations ne garantit pas que le temps obtenu par leur combinaison soit le meilleur possible, puisque la meilleure combinaison peut comprendre des parcours individuels non optimaux.
Un algorithme donnant une solution optimal pour la phase III devrait par conséquent prendre en compte l'ensemble des chemins permettant à un voyageur de se rendre d'un module à un autre, ainsi que la totalité de leur combinaisons, ce qui serait extrêmement lourd en calcul.

# A Annexes

## A.1 Compléments de code

Des fonctions présentes dans le fichier *print.ml* sont disponibles et permettent d'afficher les différentes structures de données utilisées dans *table.ml*, contenant l'ensemble des fonctions utilisées pour le fonctionnement des algorithmes.

## A.2 Codes de tests

### A.2.I Phase I

# Code de test 1

```
5
n2 n4 3
n1 n2 7
n1 n3 1
n2 n3 2
n3 n4 2
n1 n4
```

FIG. 1 – *Entrée du test 1*

```
3 : n1 -> n3 -> n4
```

FIG. 2 – *Sortie du test 1*

# Code de test 2

```
5
n1 n2 1
n2 n3 2
n3 n4 3
n4 n5 4
n5 n6 5
n1 n6
```

FIG. 3 – *Entrée du test 2*

```
15 : n1 -> n2 -> n3 -> n4 -> n5 -> n6
```

FIG. 4 – *Sortie du test 2*

# Code de test 3

```
4
n2 n4 3
n1 n2 7
n1 n3 1
n2 n3 2
n1 n4
```

FIG. 5 – *Entrée du test 3*

```
6 : n1 -> n3 -> n2 -> n4
```

FIG. 6 – *Sortie du test 3*

# Code de test 4

```
2
n2 n4 3
n1 n3 1
n1 n4
```

FIG. 7 – *Entrée du test 4*

```
No way from n1 to n4
```

FIG. 8 – *Sortie du test 4*

```
Fatal error: exception Table.No_way
```

FIG. 9 – *Erreur du test 4*

# Code de test 5

```
9
n1 n3 2
n2 n9 4
n3 n8 3
n4 n2 4
n5 n6 6
n6 n7 1
n7 n2 2
n8 n6 6
n9 n5 2
n1 n9
```

FIG. 10 – *Entrée du test 5*

```
18 : n1 -> n3 -> n8 -> n6 -> n7 -> n2 -> n9
```

FIG. 11 – *Sortie du test 5*

# Code de test 6

```
19
n7 n5 2704
n7 n6 1846
n7 n8 337
n7 n9 1464
n8 n9 1235
n6 n9 802
n8 n2 2342
n9 n2 1121
n9 n3 1391
n6 n1 621
n1 n2 946
n6 n3 740
n6 n5 867
n6 n4 849
n3 n5 187
n3 n4 144
n1 n3 184
n3 n2 1090
n2 n5 1258
n1 n9
```

FIG. 12 – *Entrée du test 6*

```
1423 : n1 -> n6 -> n9
```

FIG. 13 – *Sortie du test 6*

# Code de test 7

```
10
n1 n5 7
n1 n2 1
n1 n6 6
n2 n5 4
n2 n3 3
n3 n5 9
n3 n6 3
n3 n4 2
n4 n5 7
n4 n6 4
n5 n6
```

FIG. 14 – *Entrée du test 7*

```
10 : n5 -> n2 -> n3 -> n6
```

FIG. 15 – *Sortie du test 7*

# Code de test 8

```
2
n2 n4 3
n1 n2 1
n1 n4
```

FIG. 16 – *Entrée du test 8*

```
4 : n1 -> n2 -> n4
```

FIG. 17 – *Sortie du test 8*

# Code de test 9

```
10 : B -> I -> T -> E
```

FIG. 18 – *Entrée du test 9*

```
10 : B -> I -> T -> E
```

FIG. 19 – *Sortie du test 9*

# Code de test 10

```
3
A B 1
B C 1
A C 1000
A C
```

FIG. 20 – *Entrée du test 10*

```
2 : A -> B -> C
```

FIG. 21 – *Sortie du test 10*

### A.2.II Phase II

## Code de test 1

```
2
n2 n3 3
n1 n2 1
2
n1 -> n2 -> n3 -> n2 -> n1 -> n2
n3 -> n2
```

FIG. 22 – *Entrée du test 1*

```
n1 -0-> n2 -3-> n3 -6-> n2 -9-> n1 -10-> n2
n3 -0-> n2
11
```

FIG. 23 – *Sortie du test 1*

## Code de test 2

```
3
n1 n2 1
n2 n3 2
n2 n4 3
2
n1 -> n2 -> n4 -> n2 -> n1 -> n2
n3 -> n2 -> n4
```

FIG. 24 – *Entrée du test 2*

```
n1 -0-> n2 -1-> n4 -7-> n2 -10-> n1 -11-> n2
n3 -0-> n2 -4-> n4
12
```

FIG. 25 – *Sortie du test 2*

## Code de test 3

```
3
n1 n2 1
n2 n3 2
n2 n4 3
2
n4 -> n2 -> n1 -> n2 -> n4 -> n2 -> n3
n4 -> n2 -> n3 -> n2 -> n4
```

FIG. 26 – *Entrée du test 3*

```
n4 -0-> n2 -3-> n1 -4-> n2 -6-> n4 -9-> n2 -12-> n3
n4 -3-> n2 -14-> n3 -16-> n2 -18-> n4
21
```

FIG. 27 – *Sortie du test 3*

# Code de test 4

```
2
n2 n4 3
n1 n2 1
2
n1 -> n2 -> n4
n4 -> n2
```

FIG. 28 – *Entrée du test 4*

```
n1 -0-> n2 -3-> n4
n4 -0-> n2
6
```

FIG. 29 – *Sortie du test 4*

### A.2.III Phase III

## Code de test 1

```
5
n2 n4 3
n1 n2 7
n1 n3 1
n2 n3 2
n3 n4 2
2
n1 -> n4
n2 -> n4
```

FIG. 30 – *Entrée du test 1*

```
n1 -0-> n3 -1-> n4
n2 -0-> n4
3
```

FIG. 31 – *Sortie du test 1*

## Code de test 2

```
9
n1 n3 2
n2 n9 4
n3 n8 3
n4 n2 4
n5 n6 6
n6 n7 1
n7 n2 2
n8 n6 6
n9 n5 2
4
n1 -> n9
n3 -> n7
n6 -> n2
n3 -> n8
```

FIG. 32 – *Entrée du test 2*

```
n1 -0-> n3 -3-> n8 -6-> n6 -18-> n7 -20-> n2 -22->
n3 -0-> n8 -12-> n6 -19-> n7
n6 -0-> n7 -1-> n2
n3 -6-> n8
26
```

FIG. 33 – *Sortie du test 2*

# Code de test 3

```
19
n7 n5 2704
n7 n6 1846
n7 n8 337
n7 n9 1464
n8 n9 1235
n6 n9 802
n8 n2 2342
n9 n2 1121
n9 n3 1391
n6 n1 621
n1 n2 946
n6 n3 740
n6 n5 867
n6 n4 849
n3 n5 187
n3 n4 144
n1 n3 184
n3 n2 1090
n2 n5 1258
4
n1 -> n9
n3 -> n8
n6 -> n2
n3 -> n7
```

FIG. 34 – *Entrée du test 3*

```
n3 -0-> n9 -1391-> n8
n3 -0-> n6 -740-> n7
n6 -0-> n1 -621-> n2
n1 -621-> n6 -1242-> n9
2626
```

FIG. 35 – *Sortie du test 3*

# Code de test 4

```
10
n1 n3 2
n2 n9 4
n3 n8 3
n4 n2 4
n5 n6 6
n6 n7 1
n7 n2 2
n8 n6 6
n9 n5 2
n10 n11 5
5
n1 -> n9
n3 -> n7
n6 -> n2
n3 -> n8
n1 -> n10
```

FIG. 36 – *Entrée du test 4*

```
10
n1 n3 2
n2 n9 4
n3 n8 3
n4 n2 4
n5 n6 6
n6 n7 1
n7 n2 2
n8 n6 6
n9 n5 2
n10 n11 5
5
n1 -> n9
n3 -> n7
n6 -> n2
n3 -> n8
n1 -> n10
```

FIG. 37 – *Sortie du test 4*

```
Fatal error: exception Table.No_way
```

FIG. 38 – *Erreur du test 4*

# Code de test 5

```
2
n2 n4 3
n1 n2 1
2
n1 -> n4
n4 -> n2
```

FIG. 39 – ***Entrée du test 5***

```
No way from n1 to n10
```

FIG. 40 – ***Sortie du test 5***