



**UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG**

PROJET 2014

**Concurrent, Parallel and Distributed Computing
Erlang**

PETRESCU Alina

RADOVANOVIC Nevena

19 mai 2014

Rapport du projet

**Département d'Informatique - Université de Fribourg – Boulevard de Pérolles 90 – 1700
Fribourg - Suisse**

But

Le but de notre travail a été de permettre à l'utilisateur de définir de manière interactive le graphe à étudier, pour lequel on calcule le chemin le plus court selon l'algorithme de Chandy-Misra. Cette facilitée évite à l'utilisateur de décrire ce graphe à l'aide d'un fichier texte dont le contenu doit respecter une certaine syntaxe peu conviviale.

Structure de la solution

La solution est écrite en Erlang et comporte deux modules:

- le module `cm` qui correspond aux séries d'exercices 8-10 et qui contient l'implémentation proprement dite de l'algorithme de Chandy-Misra;
- le module `sp` qui représente notre challenge et qui contient des fonctions pour créer un nouveau graphe ou pour modifier un graphe existant.

De cette façon, nous avons "segmenté" le code ce qui nous permet d'apporter de manière indépendante d'éventuelles modifications ou des mises à jour dans chacun des deux modules.

Au bout du compte, grâce au module `sp`, l'utilisateur lance la commande `sp:start()` sur la ligne de commande de la console Erlang. Ainsi, l'utilisateur est pris en charge grâce à un menu qui lui permet de choisir les opérations à effectuer. Suite aux indications données par l'utilisateur, le programme génère finalement le fichier texte *graph_cm1.txt*.

Le lien entre les deux modules est réalisé par la fonction `sp:shPath(G)` qui appelle la fonction qui va calculer le chemin le plus court pour le graphe créé par l'utilisateur, à savoir la fonction `cm:master("graph_cm1.txt", "res.txt", X, 1)` qui déploie l'algorithme de Chandy-Misra.

Le module `sp`

Tout système minimal Erlang consiste au moins du noyau (Kernel) et des bibliothèques standards `STDLIB`. Afin d'écrire nos fonctions qui assurent la partie interactive, nous avons utilisé toute une série de fonctions BIF qui sont prédéfinies dans des modules standards d'Erlang et que nous mentionnons brièvement ci-dessous.

Du module `file` qui fournit une interface pour le système de fichiers, nous avons utilisé les méthodes `write_file/2` et `write_file/3`.

Du module `io`, qui fournit une interface vers les serveurs standards I/O d'Erlang, nous avons utilisé les fonctions `fread/2` et `format/2`.

Du module `io_lib` qui contient des fonctions permettant la conversion à partir de ou vers des chaînes de caractères (des strings), nous avons utilisé la méthode `fwrite/2`.

Du module `lists` qui contient les fonctions nécessaires pour le traitement des listes, nous avons utilisé les méthodes `foreach/2` (qui utilise les fonctions anonymes `fun`s), `keyreplace/4`, `nth/2` et `last/1`.

Du module `ets` qui contient des fonctions BIF pour le stockage des grandes quantités des données, nous avons utilisé les méthodes `new/2`, `lookup/2` et `insert/2`.

Du module `random` qui permet la génération des nombre pseudo-aléatoires, nous avons utilisé la méthode `uniform/1`.

Le module `digraph` qui implémente une version de graphes orientés "labellisés" a été utilisé pleinement. Ainsi, nous avons employé les méthodes:

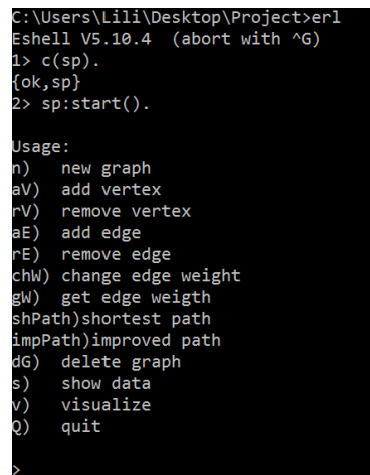
- `new/0` qui retourne un graphe vide;
- `edge/2` qui retourne une arête entre deux sommets, avec une identité propre;
- `edges/1` qui retourne une liste de toutes les arêtes du graphe;
- `add_edge/5` qui crée ou modifie une arête du graphique;
- `del_edge/2` qui supprime une arête du graphe;
- `in_edges/2` qui retourne une liste de toutes les arêtes qui arrivent à un sommet donné;
- `vertices/1` qui retourne une liste de tous les sommets du graphe;
- `add_vertex/2` qui ajoute ou modifie un sommet du graphe;
- `del_vertex/2` qui supprime un sommet du graphe ainsi que toutes les arêtes associées à ce sommet;
- `delete/1` qui supprime l'entier du graphe créé (y compris les tables `ets` associées);
- `out_neighbours/2` qui retourne une liste de tous les voisins vers lesquels des arêtes partent d'un sommet donné.

A part les modules standards mentionnés, nous avons dû installer aussi le module supplémentaire `mdrawer` qui permet de réaliser du graphisme en 2D (en utilisant des primitives graphiques comme *Mathematica*). Plus précisément, nous avons utilisé la méthode `start/0` nécessaire pour la visualisation du graphe indiqué par l'utilisateur.

Par la suite, nous passons en revue les principales fonctions ad-hoc définies dans le module `sp`.

La fonction `start/0` doit être appelée par l'utilisateur (une seule fois, tout au début du programme) quand il veut créer un nouveau graphe (qui pourra être étudié par la suite, à l'aide des fonctions prévues dans d'autres modules, comme par exemple pour calculer le chemin le plus court qui lui est associé). Après la création d'un graphe vide, cette fonction appelle la fonction `usage/1`.

La fonction `usage/1` reçoit en argument le graphe (vide) créé par `start/0`. Elle affiche les fonctionnalités mises à disposition de l'utilisateur (voir Figure 1) et, en fonction du choix de celui-ci, elle appelle la "bonne" fonction qui réalise l'opération souhaitée et affiche ensuite un message de confirmation adéquat. Après avoir rempli sa tâche, chacune de ces fonctions, à l'exception de la fonction `quit/0`, appelle à nouveau la fonction `usage/1` afin de permettre à l'utilisateur de faire d'autres choix.



```

C:\Users\Lili\Desktop\Project>erl
Eshell V5.10.4 (abort with ^G)
1> c(sp).
{ok,sp}
2> sp:start().

Usage:
n)  new graph
aV) add vertex
rV) remove vertex
aE) add edge
rE) remove edge
chW) change edge weight
gW) get edge weight
shPath) shortest path
impPath) improved path
dG) delete graph
s)  show data
v)  visualize
Q)  quit
>

```

Figure 1

La fonction `newGraph/0` crée un nouveau graphe (vide).

La fonction `addVertex/1` crée (dans le graphe courant) un nouveau sommet avec une identité précisée par l'utilisateur.

La fonction `removeVertex/1` supprime (du graphe courant) le sommet dont l'identité est demandée à l'utilisateur.

La fonction `addEdge/1` crée (dans le graphe courant) une nouvelle arête en demandant à l'utilisateur les identités des deux sommets à relier, l'identité de l'arête et son poids.

La fonction `removeEdge/1` supprime (du graphe courant) l'arête dont l'identité est demandée à l'utilisateur.

La fonction `changeWeight/1` change le poids d'une arête (du graphe courant) dont l'identité et le nouveau poids sont indiqués par l'utilisateur.

La fonction `getWeight/1` retourne le poids d'une arête (du graphe courant) dont l'identité est demandée à l'utilisateur.

La fonction `cm/1` n'est pas appelée directement suite à un choix de l'utilisateur mais dans le corps de la fonction `shPath/1`. Plus précisément, la fonction `cm/1` prépare le fichier *graph_cm1.txt* qui décrit le graphe courant et qui sera utilisé par la fonction `master/4` du module `cm` afin de calculer le chemin le plus court.

La fonction `shPath/1` demande à l'utilisateur le sommet "source" (ou master) du graphe courant et appelle la fonction `cm/1` de ce même module `sp` et la fonction `master/4` du module `cm`. Ainsi, le chemin le plus court est calculé à l'aide de l'algorithme de Chandy-Misra et les résultats sont affichés à l'écran.

La fonction `improvedPath/1` affiche ce qui se trouve dans le fichier *improvedRes.txt* généré après chaque ajout d'un sommet ou d'une arête (et sur cet aspect, nous y revenons plus loin).

La fonction `deleteGraph/1` supprime le graphe courant et crée un nouveau graphe (vide).

La fonction `display/1` affiche des informations concernant le graphe courant sous forme d'une liste de données (sommets et arêtes).

La fonction `makeList/1` n'est pas appelée directement suite à un choix de l'utilisateur mais dans le corps de la fonction `visualize/1`. Plus précisément, la fonction `makeList/1` associe à chaque sommet du graphe courant des coordonnées X et Y obtenues aléatoirement et stocke ces résultats dans une table ets.

La fonction `visualize/1` utilise cette dernière table et affiche la représentation graphique du graphe courant: chaque sommet est représenté par un petit cercle et chaque arête par une ligne sur laquelle le poids de l'arête est également affiché. Pour obtenir ce dessin, nous avons utilisé, comme déjà mentionné, la méthode `start/0` du module `mdrawer`.

La fonction `quit/1` efface les données des fichiers *.txt* et permet à l'utilisateur de mettre fin au programme interactif.

Amélioration du calcul du chemin le plus court

Une question qui reste ouverte concerne la façon dont le programme pourrait prendre en compte l'effet des modifications de la structure d'un graphe sur la valeur du plus court chemin déjà calculée. Plus précisément, dans la solution actuelle, après la modification d'un graphe, le nouveau plus court chemin est recalculé "à partir de zéro". L'idée serait de garder certaines valeurs déjà calculées pour des chemins "partiels" et de recalculer seulement la partie qui est vraiment changée suite à la modification effectuée. Plus précisément, nous pouvons considérer un graphe pour lequel le plus court chemin a déjà été calculé. A titre d'exemple, imaginons le cas d'un nouveau sommet qui est créé d'abord et ajouté ensuite à ce graphe. En fait, l'utilisateur précise les arêtes orientées qui relient ce nouveau sommet avec des sommets déjà existants. Vu que les plus courts chemins vers les sommets prédécesseurs sont connus, il suffit d'y ajouter la

contribution des nouvelles arêtes (entrantes) pour obtenir le plus court chemin jusqu'au nouveau sommet. Chaque fois que nous avons ajouté un sommet ou une arête au graphe existant, le nouveau chemin le plus court a été lui aussi ajouté au fichier *improvedRes.txt*. Dans ce cas, le sommet duquel part la première arête est considéré comme source. De plus, la fonction *improvedPath/1* permet à l'utilisateur de consulter le contenu du fichier *improvedRes.txt*.

D'une manière plus générale, il serait bien de prévoir aussi la prise en compte de tout changement de la structure du graphe (par exemple suite à la suppression des sommets ou des arêtes, la modification des poids de certaines arêtes, etc.). Ainsi, quand la modification du graphe est finie, le calcul de la nouvelle valeur du plus court chemin peut commencer soit à la demande de l'utilisateur soit à l'initiative du programme. Grâce aux méthodes *in_edges/2*, *in_neighbours/2*, *out_edges/2* et *out_neighbours/2* du module *digraph*, le programme peut connaître les sommets prédécesseurs et successeurs d'un nouveau sommet, par exemple, ainsi que les poids des arêtes qui arrivent et qui partent de ce nouveau sommet. Vu que les valeurs des plus courts chemins partiels du sommet master aux prédécesseurs du nouveau sommet ne changent pas, le calcul du nouveau plus court chemin pourrait garder ces valeurs et démarrer avec ces sommets prédécesseurs et avancer vers les successeurs. Nous pouvons envisager deux stratégies légèrement différentes afin de ne pas refaire le calcul du chemin le plus court à partir de zéro. Soit on ne modifie pas la fonction *master/4* du module *cm*, mais le fichier *graph_cm1.txt* (généré par la fonction *cm/1* de notre module *sp*) correspondra à un graphe modifié qui garde seulement la partie affectée par les modifications faites par l'utilisateur. Soit on modifie la fonction actuelle *master/4* du module *cm*, en lui ajoutant un ou plusieurs arguments qui permettent à l'algorithme de Chandy-Misra de prendre en compte seulement les sommets et les arêtes affectés par les modifications de l'utilisateur.

Conclusion

A la fin de notre travail, nous pensons avoir appris beaucoup de choses intéressantes. De plus, l'implémentation de notre challenge nous a apporté du plaisir. Cependant, nous sommes conscientes que de nombreuses améliorations peuvent encore être ajoutées.

A part les modifications évoquées dans la section précédente, la partie interactive pourrait devenir encore plus conviviale dans la mesure où certaines manœuvres maladroites de la part de l'utilisateur produiraient des messages qui l'aideraient à se rendre compte de l'erreur commise. Nous pensons par exemple à la suppression d'un sommet ou d'une arête dont les identités n'existent pas ou à la création d'une arête entre deux sommets non valides.

Remerciements

Tout d'abord, nous tenons à remercier Monsieur le Professeur B  at Hirsbrunner pour ce projet extr  mement int  ressant et riche, et qui nous a permis de prendre contact avec un langage surprenant et puissant.   galement, nous sommes reconnaissantes vis-  -vis de Monsieur Christian G  ttel dont la comp  tence, disponibilit   et amabilit   nous ont beaucoup aid  es durant notre travail.

Bibliographie

Cours du projet 2014 : <http://diuf.unifr.ch/pai/pr4/>

Documentation d'Erlang : www.erlang.org/doc/man

Documentation pour le module `mdrawer` : <https://code.google.com/p/ermdrawer/>

J. Armstrong : "*Programming Erlang – Software for a Concurrent World*", Second Edition, The Pragmatic Bookshelf, Dallas, 2013.