Compte-rendu du TEA n°3

Valentin GAUTHIER
Alexandre TORRES–LEGUET

December 7, 2022

1 Introduction

Ce compte-rendu présente le travail réalisé dans le cadre du TEA 3 de l'enseignement d'AAP, qui se concentre sur le tri rapide et le tri fusion en C.

Nous avons vu en TD le principe de fonctionnement et le code de ces tris en C, notre tâche a alors été dans un premier temps d'adapter ce travail afin de le rendre compatible avec les outils de création de graphiques, i.e. avec la structure de données `T_data`.

Dans un second temps, nous avons implémenté les fonctions `fusionsort` et `quicksort` qui respectent le même prototype que la fonction `qsort` disponible nativement dans C. (sous réserve d'inclure `stdlib.h`)

Enfin, nous avons implémenter le tri fusion sur des listes chaînées.

- `list.h` et `list.c` qui implémentent les listes chaînées.
- `stack_cs.h` et `stack_cs.c` qui implémentent les piles en contigu statique.
- `stack_cd.h` et `stack_cd.c` qui implémentent les piles en contigu dynamique.
- `stack_cld.h` et `stack_cld.c` qui implémentent les piles avec une liste chaînée.

Ces fichiers n'ont pas eu à être modifié.
Nous avons eu a créer ou modifier les fichiers :

- `elt.h` et `elt.c` pour l'implémentation du `T_elt` RPN, qui est une structure possédant 2 champs, dont l'un deux est défini par un ensemble de constantes symboliques également définies dans `elt.h`. Ils présentent également la fonction toString pour afficher sous forme de string un `T_elt` RPN et genElt pour generer un `T_elt` RPN arbitraire.
- `rpn.c` et `rpn.h` pour l'implémentation des fonctions `s2list`, `rpn_eval` et `affiche_operations_rpn`, fonction qui affiche les opérations effectuées lors de l'évaluation d'une RPN, dans le format demandé dans l'énoncé.

- `stack_choix.h` qui permet de décider le type d'implémentation souhaité pour les piles (cs, cd, ou cld)
- `rec.h` et `rec.c` où réside l'algorithme de parcours en profondeur à la volée pour chercher la meilleure RPN.
- `main.c` pour gérer le programme (et notamment les paramètres passés correspondant aux cartons tirés et à la cible)

2 Développement

Dans le déroulé de notre travail, il s'est tout d'abord agit de revoir le cours en profondeur pour s'assurer de maîtriser les nouvelles définitions de types, structures et manières d'implémenter sous C pour plus d'efficacité lors de la réalisation pratique. Cette phase de théorie aura durée l'ensemble du weekend (26 et 27/11). Ensuite, Alexandre s'est chargé de rédigé un premier code en PYTHON pour se faire une première idée de l'algorithme proposé. Puis le lundi 28/11, nous nous sommes réunis pour décider des fichiers et exercices à disposition dans le TP que nous pouvions utiliser (intégralement, en partie ou plus ou moins modifié) pour le TEA. L'après midi du lundi aura suffit pour que nous codions les fonctions nécessaires pour la partie 1 du TEA . Sur cette partie Alexandre s'est chargé des fichiers `rpn.c` et `rpn.h`, et Valentin s'est lui occupé de la modification des fichiers `elt.c` et `elt.h`.

Puis pour la partie 2, nous nous sommes réunies rapidement mardi pour décider qui ferait quelle partie (implémentation de `rec.c` ou de `main.c`) et pour discuter de nos premières idées sur les fonctions. Alexandre est donc partie sur la réalisation de la fonction de recherche en profondeur et Valentin le fichier `main.c`. Nous avons décider de réaliser ce travail le soir même pour s'assurer d'avoir le temps de corriger les erreurs le lendemain, avant la remise du TEA.

Nous avons rencontré quelques difficultés lors de la manipulation des chaînes de caractères, notamment pour la concaténation (nécessaire lors du parcours en profondeur) ou encore pour convertir des nombres en chaîne de caractère ou vice-versa. Egalement, il a fallu s'accorder sur les conventions prises lors de la Partie 1 (que renvoie `rpn_eval` en cas d'une RPN non valide ? comment le détecter ? etc...) pour effectuer la Partie 2 puisque cette seconde partie se base sur les fonctions développées dans la première.

Un autre point de discussion important a été celui de la représentation des cartons restants. Nous avons au départ penser à utiliser une liste, mais nous nous sommes rendus compte par la suite que avoir accès qu'à l'élément en tête de liste allait rendre inutilement compliqué la boucle à faire sur les cartons restants. Nous avons donc décidé de passer, lors des appels récursifs, un tableau d'entiers ainsi que la longueur de ce tableau.

3 Résultats

Voici, pour 2 jeux d'essais, les temps d'exécutions constatés sur l'une de nos machines (par moyenne). Le premier exemple correspond à une cible parfaite, tandis que le second ne peut qu'être approché, d'où la longueur de l'exécution.

cartons	cible	temps
3 5 7 9 25 50	788	1.2s
2 2 3 4 6 10	631	23.6s

Il est intéressant de remarquer que ces temps d'exécutions sont bien meilleurs que ceux constatés avec notre implémentation PYTHON.

4 Schéma d'exploration

Notre algorithme, étant donné un RPN, cherche à lui ajouter un à un tous les opérateurs, puis un à un les cartons restants. Une amélioration que nous avons effectué est que, si jamais un RPN peut être évalué sous forme d'une valeur (exemple : 1 2 + qui peut être évalué à 3) alors lui ajouter un opérateur le rendra nécessairement invalide. Sur le schéma, on voit que cette amélioration évite l'exploration inutile d'un grand nombre de noeuds de l'arbre.

5 Conclusion

Au terme du travail, il vient que notre programme semble fonctionner grâce à plusieurs tests tous réalisés avec justesse. Nous restons conscients que c'est un algorithme de type "glouton".