Romain G. G2S1

Loïc W.

Rapport sur les choix des algos de la SAE C-Algo

(SAE S1.02)

En algorithme, on parle de complexité. On peut parler de deux types de complexité : la complexité en espace, et la complexité en temps.

La complexité en espace correspond à l’utilisation en mémoire que le programme, l’algorithme va utiliser et l’espace de stockage que le programme va se servir.

La complexité en temps correspond au temps que le programme va prendre afin de s’exécuter.

Afin de simplifier les choses, on se sert de O(n) avec n la complexité associée à l’algorithme, pour parler de complexité, ce qui correspond à une « complexité d’ordre de n ». Il existe différents types de complexité : O(n), O(n²), O(log(n)), O(n\*log(n)). La complexité qui sera la meilleure est une complexité dite « logarithmique » et la pire une complexité dite « exponentielle », cependant avec la puissance des ordinateurs de nos jours, même une complexité dite « quadratique » (de la forme O(n²)) sera acceptable.

Nous verrons tout d’abord l’algorithme retenu pour effectuer les recherches, avec des exemples, puis nous verrons quel algorithme nous avons choisi pour effectuer le tri, avec notre explication sur notre choix puis une explication sur son fonctionnement et enfin nous verrons quel algorithme nous avons retenu pour les filtres. Une annexe avec les sources est accessible à la toute fin de ce rapport.

La recherche :

En ce qui concerne la recherche, nous avons stocké le tableau qui est demandé d’afficher dans un tableau (à 1 ou 2 dimensions) donc pour la recherche nous avant juste à demander à l’utilisateur quelle lettre il désire chercher et faire une recherche dans la bonne colonne du tableau

Par exemple si l’utilisateur rentre dans un tableau qui contient 3 entrées listées ci-dessous le programme va demander dans quelle catégorie il veut chercher puis lui proposera de renseigner son champ de recherche le programme va alors ce placer dans la bonne colonne (ici en vert) et va regarder si l’entrée de l’utilisateur correspond à la première ligne puis passera à la suivante :

(Exemple pour le nom Pons : )

↓ le programme sélectionne cette colonne grâce à l’entrée nom

|  |  |  |  |
| --- | --- | --- | --- |
| Nom | Prénom | Ville | Numéro |
| Adelaire | Benoit | Grenoble | 04.16.23.32.39 |
| Laurent | Aubert | Lyon | 04.71.29.92.51 |
| Pons | Zoé | Grenoble | 05.87.79.20.71 |

Si le programme en se déplaçant détecte plusieurs lignes qui correspondent à l’entrée de l’utilisateur alors il les affichera comme cela vous est expliqué dans l’exemple suivant avec les paramètres d’entrées ville et Grenoble :

Le programme sélectionne cette colonne grâce à l’entrée ville

↓

|  |  |  |  |
| --- | --- | --- | --- |
| Nom | Prénom | Ville | Numéro |
| Adelaire | Benoit | Grenoble | 04.16.23.32.39 |
| Laurent | Aubert | Lyon | 04.71.29.92.51 |
| Pons | Zoé | Grenoble | 05.87.79.20.71 |

Le tri :

Pour la partie du programme destiné au tri d’un tableau donné par l’utilisateur nous avons fait des recherches sur les différents types d’algorithmes de tri et nous en avons conclus que le tri Shell serait la meilleure des solutions bien qu’il ait une complexité de Θ(N2) il se débrouille très bien que le tableau à trier soit dans la pire des façons, la meilleure des façons, ou complètement aléatoirement, il est donc assez polyvalent pour notre projet.

Ce programme de tri marche en séparent le tableau en plusieurs sous-tableau et compare ensuite les premiers éléments de chaque sous tableau entre eux puis passe au deuxième élément jusqu’au dernier puis réduit la taille du sous tableau et recommence à trier le premier élément du sous tableau et ainsi de suite jusqu’à que les sous tableau ne soit composé plus que d’un seul élément.

Les filtres

Concernant les filtres, on se sert d’un simple parcours dans le tableau, c’est efficace et vu que le tableau n’est pas forcément trié alors c’est sûrement la meilleure solution. On procède de la façon suivante :

* On demande à l’utilisateur quel critère il veut voir ;
* On demande à l’utilisateur quelle correspondance il veut voir ;
* L’algorithme stocke dans un tableau de la structure temporaire ;
* Le tableau est affiché.

Même avec l’annuaire de 5001 lignes il n’y a aucun problème de temps de latence, tout est très rapide. On en revient donc à ce qu’on disait dans l’intro, la puissance des machines de nos jours nous permet « d’abuser » des complexités un peu grandes afin de se servir d’algorithme simple à comprendre et simple à mettre en place.

ANNEXE :

I – Recherche

1 – Exemples

II – Tri

1 – Justification

2 – Explication de l’algorithme

III – Filtre

1 – Explication

2 - Exemple

Sources :

<http://lwh.free.fr/pages/algo/tri/tri.htm> : site qui nous a permis de faire notre choix sur l’algorithme du tri à utiliser, qui nous a présenté les différents algorithmes de tri qui existaient

<https://clarolineconnect.univ-lyon1.fr/resource/open/file/6327414> : site qui nous a permis de savoir quoi mettre dans ce rapport

<https://home.mis.u-picardie.fr/~furst/docs/2-Recherche-Tri.pdf> : explication pour l’algorithme de recherche dans un tableau non-trié