Programmation

Rapport de Projet

Professeur Encadrant:
M. Quentin BRAMAS
Maître de conférence en Informatique
ICUBE, Université de Strasbourg

-Présentation du projet:

-Projet de bibliothèque et de bibliographie numérique:

DBLP is a computer science bibliography website. Starting in 1993 at Universität Trier in Germany, it grew from a small collection of HTML files and became an organization hosting a database and logic programming bibliography site. [...] DBLP listed more than 5.4 million journal articles, conference papers, and other publications on computer science in December 2020, up from about 14,000 in 1995 and 3.66 million in July 2016. All important journals on computer science are tracked. Proceedings papers of many conferences are also tracked. It is mirrored at three sites across the Internet.[...]
DBLP originally stood for DataBase systems and Logic Programming. As a backronym, it has been taken to stand for Digital Bibliography Library Project; however, it is now preferred that the acronym be simply a name, hence the new title "The DBLP Computer Science Bibliography".

Wikipedia

-Objetcif du projet:

Le but du projet est de d'analyser la base de données des articles de recherche en informatique provenant de la DBLP. Le programme, écrit en C, doit ouvrir la base de données, la parser, la stocker dans une structure adaptée, la stocker dans un autre fichier au format binaire et l'analyser à l'aide d'algorithmes de graphe. L'analyse permettra notamment de calculer le plus court chemin entre deux auteurs.

-Plan du programme:

Le programme comporte quatre phases distinctes: le lancement du programme, le parsage de la base de donnée, le stockage des informations recueillies et enfin le traitement de ces informations.

Le programme laisse à l'utilisateur de le choix des actions qu'il souhaite réaliser avec le programme. Un parseur d'arguments détermine quelles options ont été appelées ainsi que les paramètres associés. Ce parseur utilise **getopt** pour remplir de façon adéquate les champs d'une structure qui enregsitre les options (**options_t**).

Il faut ensuite parser la base de données...

Au fur et à mesure que le parseur lit les publications (**publication_t**), elles sont enregistrées dans un fichier binaire. Simultanément, chaque auteur de chaque publication est ajouté dans une liste chaînée d'auteurs (**author_t**). Un auteur est ajouté une seule fois dans la liste, s'il est l'auteur de plusieurs publications alors ses informations personnelles sont mises à jour. Ainsi chaque auteur possède un numéro d'identification qui lui est unique: son indice de position dans la liste chaînée.
Une fois que toute les publications de la base de donnée ont été traitées, la liste chaînée d'auteurs est écrite à la suite dans le même fichier. Puis cette liste d'auteurs est convertie en un graphe (sous forme de liste d'adjacence, correspondant à une structure **graph_t**) dont chaque sommet est un entier qui correspond à l'identifiant d'un auteur. Ce graphe représentant les relations de co-auteurs est aussi écrit à la suite dans le fichier.

Par la suite les opérations sur les graphes seront faites sur ce dernier graphe dont les sommets sont identifiés par de simples entiers. Il est alors possible de mettre en place des algorithmes pour étudier les plus courts chemin et la connexité dans ce graphe.

-Manuel d'utilisation:

DBLP PARSING AND ANALYSING TOOL

outils de parsage et d'analyse de la dblp - parse le fichier base de donnée, enregistre les informations nécessaires dans un fichier, permet d'effectuer des opérations sur le graphe des auteurs

SYNOPSIS

dblp-tool [OPTION D'AIDE]
dblp-tool [OPTION DE FICHIER] [OPTION D'ACTION]

dblp-tool [-h]
dblp-tool [-i fichier_base_de_donnée -o fichier_de_sortie] ...
dblp-tool [-s fichier_enregistré] ...
dblp-tool ... [-c]
dblp-tool ... [-p auteur1 -p auteur2]
dblp-tool ... [-l mot]
dblp-tool ... [-a auteur]
dblp-tool ... [-a auteur -n distance]

DESCRIPTION

-h Afficher le menu d'aide à l'utilisateur
-i Définir le fichier base de donnée à utiliser
-o Définir le fichier dans lequel enregistrer les résultats
-s Définir le fichier de résultat à utiliser (le fichier choisi doit avoir été créé avec le programme)
-c Calculer le nombre de composantes connexes dans le graphe des auteurs
-p Calculer le plus court chemin entre deux auteurs
-l Rechercher un mot parmi les auteurs et les publications
-a Afficher les informations d'un auteur
-n Afficher tous les auteurs suffisamment proche d'un auteur choisi

-Documentation:

-Parser d'arguments:

Le parser d'arguments permet de reccueillir les paramètres que l'utilisateurs transmet au programme lors de l'éxecution en ligne de commande (voir **Manuel d'utilisation**).
Le parser doit donc enregistrer chaque option et, le cas échéant, l'argument lié à cette option. Pour cela le parser d'arguments utilise la commande **getopt**:

> **DESCRIPTION**
> getopt is used to break up (parse) options in command lines for easy parsing by shell
> procedures, and to check for valid options. It uses the GNU getopt(3) routines to do this.

Grâce à cette fonction, lors de l'appel d'une option valide avec un argument cohérent, les paramètres de l'options sont enregistrés dans une structure **options_t**. Si les paramètres ne sont pas valides, une message d'erreur est affiché et le programme s'arrête. De même, si l'option n'est pas reconnue, le programme s'arrête.

Enfin, si plusieurs options (appélées correctement) sont incompatibles, la fonction **checkOptions** affichera un message d'erreur contenant une explication avant d'arrêter le programme.

Après avoir parsé les arguments, le programme exécutera au choix la fonction **option_io** (si l'option -i a été renseignée) ou **option_s** (si l'option -s a été renseignée).

Remarque: les deux options sont incompatibles, mais l'utilisateur doit obligatoirement fournir un fichier base de donnée.

Chacune de ses fonction ouvrira les fichiers nécessaires. Finalement, c'est la fonction **callActionFunction** qui permettra d'éxecuter l'action choisie par l'utilisateur.

```c
/**
 * @brief Options that can be sent in the command line
 */
typedef struct options_t
{
    char *prgm_name;        //Program name
    int help;               //Help indicator
    char *input_file_name;  //Name of the data base file
    char *output_file_name; //Name of the file to store results
    char *saved_file_name;  //Name of the file of results to load
    int find_connex;        //Option -c indicator
    char *p_author1;        //Name of the first author in option -p
    char *p_author2;        //Name of the second author in option -p
    char *searched_word;    //Searche word via option -l
    char *a_author;         //Name of the author in option -a
    int distance;           //Distance in option -n
    FILE *db_file;          //Data base file
    FILE *prgm_file;        //File to store results
} options_t;
```

Quelques cas particuliers:

-Si aucun ficier de sortie n'est renseigné avec l'option -o, alors un fichier de sortie par défault sera créé à l'emplacement prédéfini **./out/default**.

-Si aucune action (-c, -p, -l, -a, -n) n'est renseignée par l'utilisateur, alors le progrmmme effectura seulement le parsage de la base de donnée fournie et enregistrera les infomrations dans le fichier de sortie choisi (ou par defaut).

-Parser xml:

Le parser XML permet d'obtenir des objets **publication_t**, indiquant leur type, leur titre, leur année de publication, leur mois de publication et la liste chaînée de ses auteurs.

```c
/**
 * Publication Structure
 */
typedef struct publication_t
{
    publication_type_t type; //Publication's type
    char *title;             //Publication's title
    int year;                //Year of publication
    int month;               //Month of publication
    ll_list *authors;        //List of author(s) [name]
} publication_t;
```

Figure 2: Structure publication_t renvoyée par le parseur

Tout commence par l'appel de la fonction **parserSetup** dans le fichier **option-io.c**. Cet appel permet de dfinir les variables essentielles au parseur, notamment deux buffers : **payload.tag** pour le stockage temporaire des tags et **payload.txt** pour le stockage du texte entre les tags.
Ensuite, le parseur est appelé dynamiquement pour lire chaque nouvelle publication par la fonction **getPublication** qui renvoie 1 si elle a lu une publication et 0 sinon.

-Graphe:

Comme expliqué précédemment, le graphe utilisé est un graphe non orienté dont les sommets sont des entiers (l'indice d'un auteur dans la liste chaînée d'auteurs créée lors de la lecture des publications).
La structure théorique de graphe utilisée est une liste d'adjacence. C'est à dire que le graphe est représenté par deux listes. Dans le programme, cela correspond à la structure **graphe_t**, qui contient deux listes chaînées **deg** et **adj**. La première liste contient, à l'emplacement d'indice correspondant au sommet, l'indice qui correpsond au dernier voisin du sommet. La seconde liste contient donc les voisins de chaque sommets.

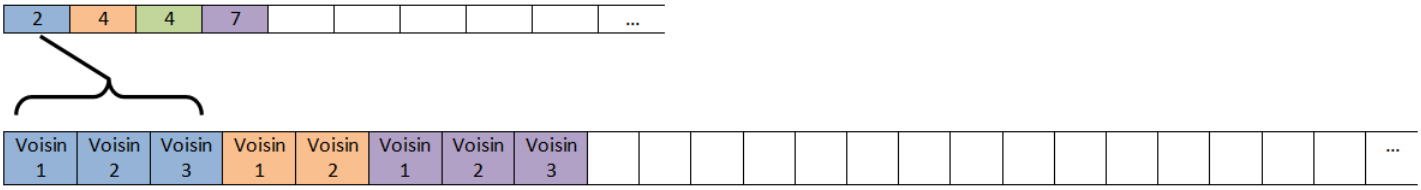

Figure 3: Représentation d'un graphe sous forme de liste d'adjacence

Cette structure est particulièrement adaptée aux graphes ayant une structure lacunaire. En effet, contrairement aux matrices d'adjacence ou d'incidence, cette structure utilise uniquement l'espace mémoire nécessaire (il y a de nombreux emplacements vides dans une matrice lorsqu'un graphe est lacunaire). Cependant, les opérations sur le graphe sont parfois plus lentes étant donné que nous n'avons pas accès à un sommet en une seule instruction comme c'est le cas avec une matrice.

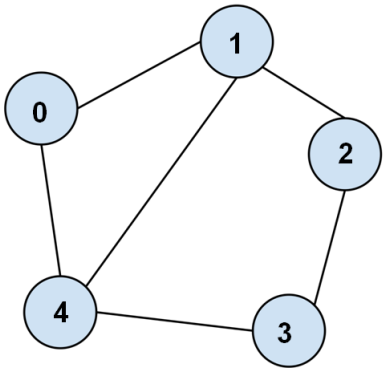

Figure 4: Représentation graphique du graphe n°.1

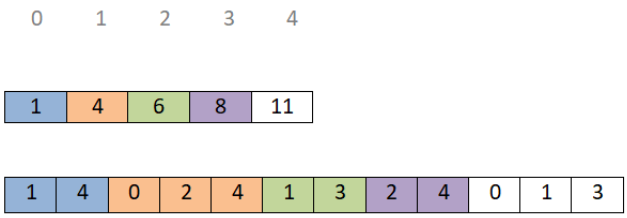

Figure 5: Liste d'adjacence du graphe n°.1

Chaque publication possédant un nombre limité d'auteurs (i.e.: chaque auteur a peu de voisins), le graphe obtenu sera lacunaire. La structure de liste d'adjacence est donc adaptée.

-Stockage:

Le stockage des informations est une étape cruciale du programme car il doit être efficace tout en restant adapté à une lecture simple par la suite pour faciliter l'accès aux informations.

-Publication:

Les premières informations à écrire dans le fichier sont les publications. Elles sont écrites une à une au fur et à mesure du parsage de la base de donnée.
Chaque publication est écrite dans le fichier suivant le format suivant:

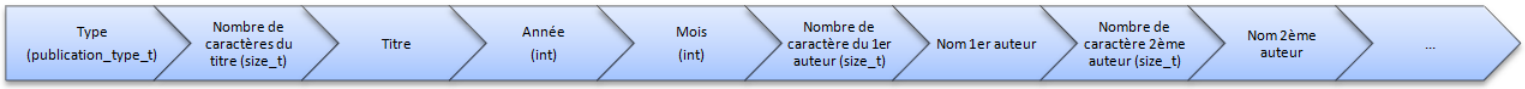

Figure 6: Représentation en mémoire d'une publication (publication_t)

Le type, le titre, la date et les auteurs de chaque publication sont écrits à la suite dans le fichier. Les éléments dont le type est de taille fixe (**int**, **size_t**...) sont simplement écrits. Les chaines de caratères sont précédées de leur nombre de caractère (i.e.: de leur taille dans le fichier) écrits sous la forme d'un **size_t**.

-Auteur:

Ce sont ensuite les auteurs qui sont écrits dans le fichier.
De la même façon que pour une publication les arguments de la structure **author_t** dont la taille en octet est fixe sont simplements écrits alors que les chaines de caractères sont précédées de leur nombre de caractère (un **size_t**).

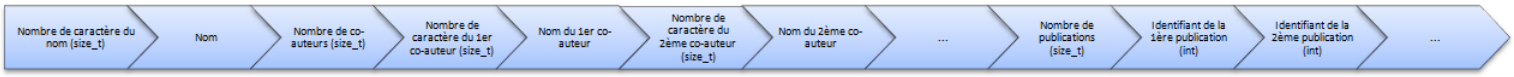

Figure 7: Représentation en mémoire d'un auteur (auteur_t)

-Graphe:

La dernière structure à écrire dans le fichier est la structure **graph_t** qui contient le graphe sur lequel sont réalisées les opérations sur les graphes.
Les deux listes **graph.deg** et **graph.adj** sont des listes chaînées d'entiers. Chaque élément (dont la taille est connue: **sizeof(int)**) de chaque liste est écrit dans le fichier. Chaque liste est précédée de sa taille (i.e.: le nombre d'éléments qu'elle contient) écrite sous la forme d'uin **size_t**.
Ainsi la taille d'une liste dans le fichier correspond à son nombre d'éléments multiplié par **sizeof(int)**).

Figure 8: Représentation en mémoire d'une graphe (graph_t)

-Structure du fichier:

Afin de faciliter et d'optimiser le procédé de lecture du fichier, des infomrations utiles sont écrites au début du fichier et à des emplacements stratégiques.

En début de fichier, on retrouve deux **size_t**, qui correspondent dans l'ordre à la position du curseur dans le fichier pour lire les auteurs et celle pour lire le graphe. (La position dans le fichier pour lire les publications vaut **2 * sizeof(size_t)**).

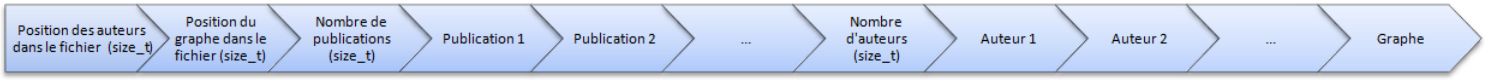

Figure 9: Représentation de la structure du fichier

On retrouve aussi juste avant les publications et les auteurs (respectivement aux positions **2 * sizeof(size_t)** et celle indiquée par le premier **size_t** écrit au début du fichier), le nombre d'éléments (publication ou auteur) qui sont écrits à la suite.

Afin de permettre une lecture rapide du fichier, chaque publication et chaque auteur est précédé de sa taille dans le fichier en octect écrit sous la forme d'un **size_t**.

-Bilan général:

Au cours de ce projet, nous avons rencontré de nombreuse difficultés. Certaines étaient prévisibles, par exemple la complexité du programme à fournir, d'autres ont été sous-estimées (rendre un projet complet et bien structuré). Enfin, certaines difficultés se sont introduites au cours du projet et ont entravé notre progression. Nous n'avions jamais travaillé en équipe sur un même programme et cela s'est avéré plus difficile qu'il n'y parait.

Ce projet s'est avéré être un réel défis. N'ayant jamais créé un programme réalisant autant de fonctionnalités différentes dans un language de programmation dont nous connaissons encore mal les capacités et les limites, nous avons eu du mal à trouver des repères et une méthode de travail adaptée. (Exemples: Comment séparer les fichiers? Quelle fonction créer? Comment gérer les erreurs?...)
Déterminer les fonctionnalités du programme, leur mise en pratique et les programmer tout en conservant une cohérence globale dans toutes les portions de code (nom de variables, structures...) a été une tâche difficle.

À cela s'est ajouté le partage des fichiers du projets. Afin de travailler sur des fichiers communs et s'accorder sur les caractéristiques techniques du programme, nous avons utilisé le gestionnaire de version Git. Cet outils, bien que très utile et extrêmement performant, n'est pas réellement facile à prendre en main.

Comme en témoigne l'utilisation de Git, la difficulté du projet a été amplifiée par le facteur *"travail en équipe"*. Développer différentes parties de codes ayant des fonctionnalités différentes mais compatibles nécessite une bonne communication.

Finalement nous avons réussi à fournir un programme fonctionnel et simple d'utilisation. L'utilisateur peut facilement indiquer au programme quelles actions effectuer. Le programme parvient alors à parser la base de donnée si nécessaire puis réalise les actions demandées grâce à diverse méthode de lecture et d'écriture dans un fichier et à des algorithmes sur le graphe.
Néanmoins de nombreux aspects du porgrammes restent à améliorer. L'utilisation des listes chaînées entrave les performances du porgramme à de multiples reprises (notamment lorsqu'il s'agit de retrouver un auteur, opération effectuée pour chaque publication lors du parsage de la base de donnée). L'implémentation d'une table de hachage aurait grandement diminué le temps de recherche et ainsi aurait amélioré les performances du parser de la base de donnée.
De plus, certaines fuites de mémoire auraient pu être évitées en prévoyant plus précisemment les fonctionnalités du programme. Il vaut mieux écrire un pseudo code précis des fonctionnamlités principales en amont afin de ne rien omettre, plutôt que de programmer au fur et à mesure en modifiant toutes les fonctions précédentes à l'ajout de chaque nouvelle fonctionnalité nécessaire...
Pour terminer, l'utilisation du programme aurait put être enrichie d'une interface graphique qui aurait rendu le programme plus simple et plus agréable à l'utilisation.