



LOG2810

Été 2021

TP1

Graphe-Automate

Groupe 1

[2017113] - [Yun Ji Lao]

[2028622]-[Johary Rakotomalala]

[1956802]-[Dawut Esse]

Soumis à : Mohameth Ndiaye

18 juin 2021

Introduction:

Graphe:

De nos jours, tout le monde est déjà très familiers avec le GPS. On l'utilise depuis toujours lorsqu'on ne connaît pas le chemin pour aller à un endroit. Non seulement le GPS est en mesure de nous mener du point A au point B, mais il permet aussi de trouver le plus court chemin pour y arriver. Nous pourrions donc éviter de se casser la tête pour aller chercher le chemin dans un map en papier. Ainsi, on pourra étudier, regarder une série sur Netflix, ou même prendre une sieste pendant ce temps. Similairement à un aéroport, si un avion veut aller à un autre aéroport, il faut que le GPS calcule la distance et le plus court chemin. Malheureusement, en raison de la pandémie plusieurs aéroports des pays sont fermés, on souhaite d'aider les voyageurs à trouver le plus court chemin pour aller dans un autre pays sans passer par certains pays si besoin.

Dans le cadre du cours de Structures discrètes, nous devons concevoir un système de navigation permettant d'assurer de trouver le plus court parcours dans une carte donnée qui soit optimal pour un voyageur en évitant de passer par les pays à éviter. Par exemple, si les États-Unis sont interdits de traverser, un voyageur canadien devra passer par le Japon pour aller à Cuba. De toute façon, le système devra s'assurer de ne pas passer par les États-Unis.

Tout d'abord, nous verrons un aperçu de la tâche que nous avons à réaliser, en prenant soin de noter les particularités et éléments requis de notre système. Nous verrons certaines conditions de fonctionnement de ce système, qui sera suivi par l'explication du fonctionnement des graphes dans notre système et comment nous avons exploité certaines propriétés. Nous verrons aussi par après les fonctions obligatoirement à implémenter dans notre système. Finalement, nous présenterons les principales difficultés que nous avons rencontrées ainsi que les éventuelles solutions apportées.

Automate:

Nous avons dû concevoir, à partir d'un lexique (un ensemble de mots d'un langage), un automate à états finis minimal qui doit contenir, entre autres, le nombre minimal d'états finis. Cet automate devait ensuite être utilisé pour faire la complétion de mots, c'est-à-dire suggérer des mots au fur et à mesure que l'on commence à l'écrire. Par exemple, un utilisateur qui commence à écrire les lettres « cas », les suggestions : case, cases, caser, casier et casiers

doivent lui être suggérés. Il devait gérer les mots mal orthographiés de 1 faute aussi comme par exemple si on tapait "bonjiur" on devait pouvoir voir le bon mot qui est "bonjour". Ajoutons que l'alphabet à considérer est celui de la langue française (26 lettres) et que les lettres accentuées ne sont pas prises en compte dans ce programme. Six lexiques ont été fournis dans le cadre de ce TP.

Presentation du travaux:

Lors de la réalisation du système, il fallait implémenter les fonctions obligatoires suivantes:

1. Une fonction "créer Graphe()" qui permettra la lecture d'un fichier fourni contenant les informations sur les pays et les sommets. Le nom du fichier est passé en paramètre.
2. Une fonction "lire Graphe()" qui permettra d'afficher le graphe lu du fichier.
3. Une fonction "colorierGraphe()" qui permettra de mieux visualiser la carte, ainsi que de déterminer les pays interdits..
4. Une fonction "ExtractionGraphe()" qui permettra d'extraire le sous-graphe résultant d'un graphe coloré, auquel on veut retirer une certaine couleur passée en paramètre.
5. Une fonction "plusCourtChemin()" qui, pour un point de départ et pour un couleur voulu passé en paramètre, générera le chemin ayant la distance minimale pour y parvenir (Inspiré de Dijkstra). Permet aussi l'affichage de ce chemin.
6. Une interface qui affichera un menu qui contiendra les options créer Graphe, lire graphe, déterminer les frontières et déterminer le plus court chemin et enfin l'option quitter qui quitte l'interface.
7. Créer un automate à partir d'un lexique. Le lexique est donné sous la forme d'un fichier .txt.
8. Implémenter la fonctionnalité de suggestion qui permettra de suggérer à l'utilisateur des mots à partir des premières lettres d'un mot qu'il écrit.
9. Implémenter la fonctionnalité de correction qui permettra de corriger un mot mal orthographié par l'utilisateur (en vous plaçant dans le contexte précis ou le mot mal orthographié diffère d'une seule lettre du mot qui doit le remplacer).

10. Créer une interface graphique qui permet à l'utilisateur de saisir du texte.

Explication des solutions:

La classe Graphe possède un map qui liste ses sommets et leurs noms. Chacun de ces sommets est une classe Sommet qui a une couleur initialisée à "neutre" et son nom comme variable ainsi qu'un map de string et d'arc qui représente les voisins.

La classe Sommet peut représenter plusieurs choses, tel que un pays ou une ville. Bref, ils représentent les différents points de départ et d'arrivée possibles. Chaque sommet possède l'information sur sa couleur initialisée à neutre, son nom et ses sommets adjacents qui est un map contenant le nom du sommet adjacent et la distance entre les deux. Ce map sera très utile lorsqu'on a besoin de connaître les sommets adjacents d'un sommet. Le map a la distance comme deuxième paramètre qui indique la distance entre le sommet et le sommet adjacent. L'interface est effectuée en programmation procédurale dans le fichier main.cpp

La classe Arc est seulement une petite classe qui a la distance comme variable privée pour enregistrer la distance entre les sommets.

Le main contiendra un menu qui représentera les fonctions créergraphe() liregraphe() colorer() extrairegraphe() et determinerlepluscourtchemin()(shortestpath()) en utilisant ma classeGraphe.

L'option 1 va permettre de créer le graphe avec la fonction créer graphe. La fonction créergraphe() consiste à créer un graphe en prenant en paramètre le nom du fichier contenant les pays avec leur distance. Il va à l'aide d'une boucle while parcourir le fichier et ajouter ses voisins et retourne au final un pointeur vers le graphe. Le pointeur sera nul si le fichier n'existe pas.

L'option 2 va lire le graphe avec la fonction liregraphe(). La fonction liregraphe() va parcourir l'attribut graphe_ de la classe on fait sortir le nom du sommet actuel et sa couleur et après pour ce sommet on affiche le nom et la couleur de ses voisins. L'iteration se fait sur chaque sommet du graphe.

L'option 3 va permettre de colorier les sommets avec la fonction colorer() qui utilise le théorème des quatre couleurs. Dans la fonction colorer() il existe une méthode privée colorerGraphe() qui colorie selon les 4 couleurs choisis en paramètre et

prend en paramètre un map de string et de sommet qui représentera plus tard l'attribut privée `graphe_`. La fonction `colorerGraphe()` va manipuler la map et pour cela il va d'abord classer les sommets par ordre de degré décroissants en utilisant la méthode `classerSommet()` qui va retourner un vecteur de sommet classé en ordre décroissant. Après cela on fait une boucle while qui fait l'opération A tant que tous les sommets ne sont pas colorés. L'opération a consisté à choisir une couleur donnée . Attribuer cette couleur au premier sommet de la liste non encore coloré . Parcourir la liste, jusqu'à la fin, en attribuant cette couleur à tout sommet non coloré et qui n'est adjacent à aucun sommet coloré avec cette couleur. On refait l'itération en prenant soins de choisir une autre couleur à la prochaine itération avec le nombre de couleur limite étant à 4.

L'option 4 va permettre de déterminer le plus court chemin en utilisant la fonction `extractiongraphe ()` et `shortestpath()`. La fonction `extractiongraphe()` va extraire de la graphe les sommets colorés d'une couleur donnée en paramètre. Cela représente les pays à éviter dans notre cas. Pour extraire le graphe on a créé un nouveau graphe et on a parcouru l'ancien. On insère chaque sommet qui ne contient pas la couleur donnée en paramètre. Après cela, on supprime tous les voisins des sommets ajoutés qui possèdent la couleur donnée en paramètre. `Shortestpath()` va mettre dans l'attribut `pathdijkstra_` le plus court chemin et mettre dans `distance` minimale la distance minimale. Après on parcourt le chemin et on l'affiche à l'aide de `std::cout`. Il est à noter qu'on peut choisir dans la couleur le mot neutre qui implique qu'on peut traverser tous les pays.

L'option 5 va permettre de quitter l'interface.

L'option 6 est pour accéder au jeu Instructif avec les mots. En rentrant dans cette option, il y aura l'option Correction et Quitter. Si on choisit Correction , le jeu commence est l'utilisateur rentre le mot , si le mot n'est pas dans le lexique on dressera une liste de suggestion selon le lexique choisi et si le mot comporte une faute on pourrait voir le mot bien orthographié dans la liste aussi.

Difficultés rencontrées:

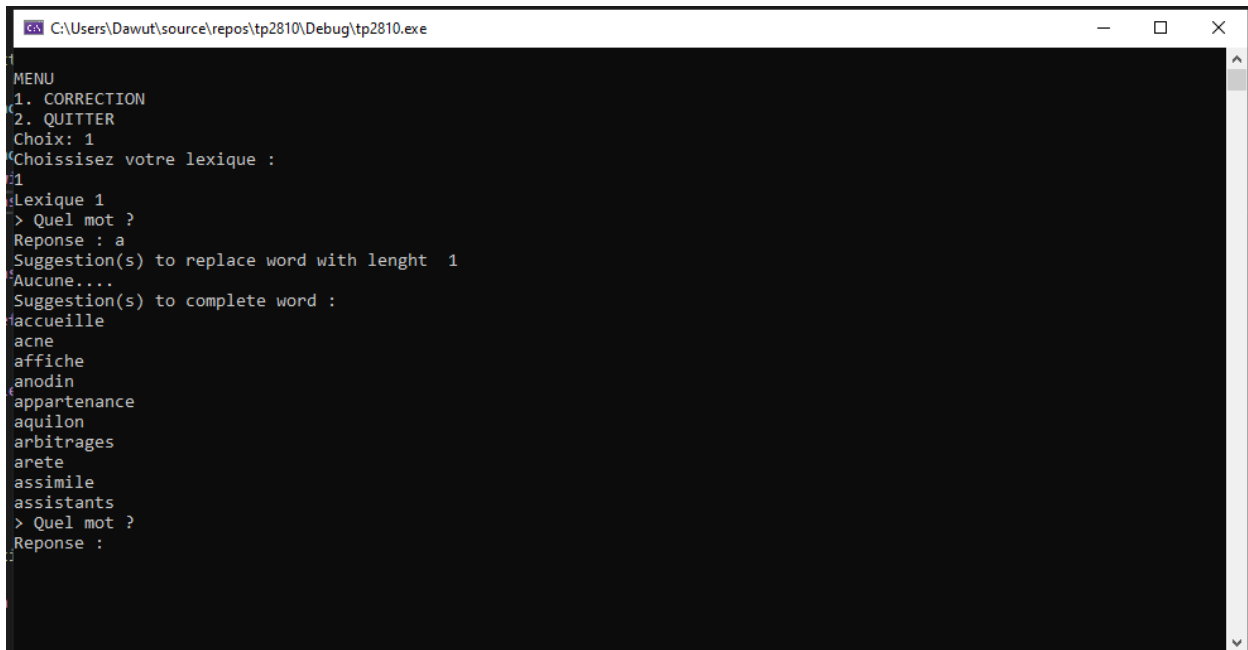
Nous nous sommes rendus compte que l'algorithme de Dijkstra ne calculait pas le meilleur chemin. En effet, l'algorithme fait en sorte que le sommet passe toujours par le plus arc et il ne calcule pas le plus court chemin en général. En conséquence, nous avons fait une nouvelle façon d' implémenter Dijkstra en utilisant `priority_queue`. En effet, une file d'attente prioritaire assure que lorsque nous explorons un sommet après l'autre, nous explorons toujours celui avec la plus petite distance.

D'autre part, nous avons éprouvé de la difficulté à lire les fichiers textes en les stockant tout d'un coup dans une variable de type string. Éventuellement, un de nous a suggéré que nous utilisions un `istream`. Nous avons utilisé `getline` pour isoler les éléments entre les point-virgules « ; ».

Finalement, étant donné que notre programme était complexe. Il y avait beaucoup de bogues et de problèmes de compilation. Nous avons passé beaucoup de temps à gérer les exceptions. De plus, nous avons passé un certain temps à enlever des fonctions inutiles.

Du côté de l'automate, on a pas pu rendre le texte dynamique, il faut écrire le mot incomplet et ensuite taper sur Enter pour qu'un tableau de suggestion apparaisse et lister les mots possibles. Pareillement pour le correcteur de texte, il faut rentrer le mot avec la faute et ensuite un tableau va dresser le mot sans faute. On doit préciser que le code pour cette partie est strictement procédurale et non orienté-objet.

Figure 1 : Tableau de suggestion lorsqu'on rentre a comme première lettre



```
C:\Users\Dawut\source\repos\tp2810\Debug\tp2810.exe
MENU
1. CORRECTION
2. QUITTER
Choix: 1
Choisissez votre lexique :
1
Lexique 1
> Quel mot ?
Reponse : a
Suggestion(s) to replace word with lenght 1
Aucune...
Suggestion(s) to complete word :
accueille
acne
affiche
anodin
appartenance
aquilon
arbitrages
arete
assimile
assistants
> Quel mot ?
Reponse :
```

Figure 2: Tableau qui montre le mot corrigé lorsque le mot est mal orthographié par exemple "affishe"



```
C:\Users\Dawut\source\repos\tp2810\Debug\tp2810.exe
MENU
1. CORRECTION
2. QUITTER
Choix: 1
Choisissez votre lexique :
1
Lexique 1
> Quel mot ?
Reponse : a
Suggestion(s) to replace word with lenght 1
Aucune....
Suggestion(s) to complete word :
accueille
acne
affiche
anodin
appartenance
aquilon
arbitrages
arete
assimile
assistants
> Quel mot ?
Reponse : affiche
Suggestion(s) to replace word with lenght 7
affiche
Suggestion(s) to complete word :
Aucune....
> Quel mot ?
Reponse :
```

Conclusion:

Ce laboratoire nous a permis d'appliquer les notions sur les graphes que nous avons vu en classe telles que la notion de sommet, d'arc, de graphe connexe et de chemin. Il nous a aussi permis de comprendre comment des applications GPS comme Google Map fonctionne. Nous avons pu revoir les notions de la programmation en C++, ainsi que celles de la lecture de fichier. Nous avons appris à réfléchir sur papier avant de passer à l'écriture du code. Finalement. Nous sommes curieux de savoir comment les applications qui utilisent le GPS, par exemple Google Map, sont capables de calculer le plus court chemin pour une voiture aussi vite alors qu'il existe des millions de possibilités de chemin. Bref, on aimerait approfondir sur comment améliorer notre code afin qu'il s'exécute aussi rapidement que les GPS réels. De cette façon, on pourra l'appliquer dans la vraie vie, ce qui est très excitant. Pour conclure, ce travail pratique nous a permis de comprendre l'utilité des concepts sur les automates appliqués au quotidien et de permettre la prise en compte de son importance dans le travail d'un ingénieur en informatique et en logiciel. Nous permettant de mettre en pratique des notions théoriques en pratique, le travail fut un succès. Ainsi, notre système fait bel et bien la suggestion de mots et la correction de mots.