TP3 - Algorithmes de recherche

1. Simulations (papier/crayon)

Vous disposez de ce tableau de valeurs :

-4	-4	-1	0	3	3	10	12	17	21	25	28	23	29	30	33	40	45	45	45	45	50
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*

1. Simuler l'algorithme de recherche dichotomique pour chercher la valeur 45. Utiliser pour cela le tableau fourni dans le fichier Excel.
   - Quel indice sera retourné ?
   - Combien de valeurs ont été vérifiées ?
   - Combien de valeurs aurait-il fallu vérifier avec une recherche séquentielle ?
   - Combien de valeurs faut-il vérifier au pire avec une recherche séquentielle ? Quelle est la complexité temporelle de cet algorithme ?
   - Combien de valeurs faut-il vérifier au pire avec une recherche dichotomique ? Quelle est la complexité temporelle de cet algorithme ?

2. On souhaite trouver l'indice de la première occurrence de 45. Reprendre la diapo 15 du poly d'algo et simuler l'algorithme proposé (utiliser à nouveau les tableaux fournis dans le fichier Excel). Quel indice sera retourné ?

3. Faire de même pour trouver le dernier indice.

2. Recherche dichotomique dans un vecteur d'entiers

Créer un nouveau projet Code::Blocks à partir du fichier `mainIntroRecherche.cpp` fourni sur Moodle.

1. Étudier le code fourni et exécuter le programme pour être sur de bien le comprendre.
2. Définir les deux fonctions de recherche dont le prototype vous est donné. La première `rechercheDicho` doit implémenter la recherche dichotomique de base pour trouver une valeur dans un vecteur d'entiers et retourner un indice où elle se trouve, sinon -1. La deuxième `recherchePetit` doit implémenter une version de la recherche dichotomique permettant de trouver l'indice de la première occurrence d'une valeur dans un vecteur d'entiers. Inspirez vous des simulations de l'exercice précédent et des informations fournies dans le poly d'algo.
3. Veiller à lancer autant de recherches que nécessaire pour tester tous les cas particuliers.

3. La recherche dichotomique dans une classe

On reprend les classes `Pièce` et `Appartement` de la feuille de TP précédente. Pour cela utiliser le corrigé du TP2 fourni sur Moodle. Le but est de rechercher une chambre (instance de la classe `Pièce`) dans une résidence universitaire (instance de la classe `Appartement`), en supposant que les chambres sont triées alphabétiquement par leur nom.

Rappel : les chaînes de caractères (`string`) peuvent être comparées avec les opérateurs de comparaison habituels (<, >, =. >=, <=). Mais attention: "*chambre10*" < "*chambre9*" parce que le 8ème caractère de la 1ère chaine ('1') est plus petit que le 8ème caractère de la 2ème chaine ('9'). Il faut donc écrire *chambre09* par exemple si vous utilisez des noms avec des chiffres > 9.

1. Ajouter au fichier d'entête de la classe `Appartement` les deux méthodes suivantes :
   - `recherche` : retourne un indice de la pièce dont le nom est passé en paramètre si cette pièce est présente dans l'appartement cible, et -1 sinon.
   - `recherchePremier` : retourne le premier indice de la pièce dont le nom est passé en paramètre si cette pièce est présente dans l'appartement cible, et -1 sinon.

2. Déclarer une résidence universitaire *r1*, instance de la classe `Appartement`, et y ajouter 10 chambres nommées "*chambre0*" à "*chambre9*", instances de la classe `Pièce`, en veillant à les ajouter en ordre alphabétique pour obtenir un vecteur de `Piece` trié par ordre croissant des noms des pièces.
   - Définir la méthode `recherche` de la question 1 en effectuant une recherche dichotomique de la pièce dont le nom est passé en paramètre. Reprendre et adapter le code que vous avez écrit pour l'exercice précédent. Tester dans le programme principal.

3. Déclarer une résidence universitaire *r2*, instance de la classe `Appartement`, et y ajouter 10 chambres, instances de la classe `Piece`, en veillant de nouveau à les ajouter en ordre alphabétique pour obtenir un vecteur de `Piece` trié par ordre croissant des noms des pièces, mais en ajoutant cette fois-ci plusieurs occurrences d'une même chambre (5 *"chambre5"* par exemple).
   - Définir la méthode `recherchePremier` de la question 1 en effectuant une recherche dichotomique version "petit" afin de trouver ? ce de la première pièce dont le nom est passé en paramètre. Reprendre et adapter le code que vous avez écrit pour l'exercice précédent.

Tester dans le programme principal.

4. (optionnel, pour les plus rapides) La recherche à double critère

Les étudiants d'une promotion sont enregistrés dans une instance de la classe `Promotion`, contenant un nom (par exemple *`"S1"`*) et un vecteur d'instances de la classe `Etudiant`. Un étudiant est représenté par son nom, son prénom, son numéro d'étudiant et son groupe (par exemple *`"1J"`*). Le vecteur est trié par groupe, puis par ordre alphabétique des noms d'étudiants à l'intérieur d'un groupe. Si plusieurs étudiants du même nom, alors ils sont dans des groupes différents (par exemple *`"BOUTRY"`* dans le code fourni).

Exemple (avec des noms réduits à leur 1ère lettre) :

groupe	...	1J	1J	1J	1K	1K	1K	1K	1K	1K	1L	1L	1L	...
nom	...	"G"	"P"	"T"	"C"	"G"	"L"	"M"	"R"	"T"	"B"	"O"	"V"	...
prénom	...	"s"	"v"	"a"	"t"	"a"	"l"	"m"	"d"	"z"	"x"	"p"	"a"	...
numéro	...	321	272	183	275	835	463	472	824	371	528	714	296	...
	...	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*	*33*	*34*	*35*	...

Les informations concernant les étudiants sont enregistrées dans un fichier, `listeEtudiants.txt`, qui sera lu pour initialiser le vecteur d'étudiants de la promotion S1 (les numéros d'étudiants sont factices). Le code nécessaire pour cette opération vous est fourni.

Créer un nouveau projet avec les fichiers disponibles sur Moodle (ClassePromotion-FichiersEtudiants) et enregistrer le fichier `listeEtudiants.txt` dans le même répertoire que les fichiers de code. Étudier les 2 classes fournies puis traiter les 2 questions qui suivent, en faisant les ajouts nécessaires à ces classes. Vous complèterez bien sûr le programme principal avec des exemples et des affichages indiquant clairement les résultats.

Ne pas oublier de gérer les exceptions en cas de groupe invalide. Le code fourni contient un tableau constant initialisé avec les groupes valides du S1 ainsi qu'une fonction permettant de déterminer si un numéro de groupe est valide.

1. On souhaite connaître l'indice d'un étudiant dont on connaît le groupe et le nom. Si plusieurs étudiants du même nom, alors ils sont dans des groupes différents.
   - Écrire une fonction `rechercheEtudiant`, membre de la classe `Promotion`, qui prend en paramètre un groupe (string) et un nom d'étudiant (string), et qui retourne l'indice de l'étudiant ayant le nom donné et appartenant au groupe donné, s'il existe, sinon retourner -1. La solution la plus appropriée est d'utiliser un algorithmique dichotomique de type "classique", avec un double critère de recherche.
   - Après avoir complété votre programme principal pour afficher l'indice d'un étudiant dont on connait le groupe et le nom, ajoutez les instructions nécessaires pour afficher les informations le concernant (s'il existe).

2. On souhaite connaître le nombre d'étudiants d'un groupe donné, ainsi que les indices du premier et du dernier étudiant de ce groupe dans l'ordre alphabétique des noms.
   - Écrire une fonction `rechercheGroupe`, membre de la classe `Promotion`, qui prend en paramètre un groupe donné (string) et qui renvoie par le biais des paramètres les indices du premier et du dernier étudiant de ce groupe, s'ils existent (sinon leur affecter la valeur -1). La fonction renvoie vrai s'il y a au moins un étudiant dans le groupe et faux sinon. La solution la plus appropriée est d'utiliser un algorithmique de type "recherche dicho petit" pour trouver le premier indice et un algorithmique de type "recherche dicho grand" pour trouver le dernier indice.
   - Après avoir complété votre programme principal pour afficher le premier et le dernier indice des étudiants d'un groupe de votre choix, ajoutez les instructions nécessaires pour afficher les étudiants concernés.

Modifié le: lundi 11 décembre 2023, 16:47