TP6 : Structures de données

Préambule

L'objectif de cette séance est de se familiariser avec les structures de données vus en cours. À la fin de ce TP, vous devriez être capable de mettre en œuvre :

- le tri fusion sur des listes,
- une liste chaînée avec des éléments alloués en tableau.

Quelques informations importantes :

- Le langage de programmation à utiliser est le C.
- Ce TP utilise les signaux (pour interrompre les tris lorsqu'ils dépassent une certaine limite de temps), il doit être fait sous Linux.
- Des outils pour tester les fonctions sont fournis dans le fichier zip `algo-tp05-code.zip`.
- Pour chaque exercice, exécutez les tests dans les fichiers fournis pour vérifier le code écrit.
- Il est fortement recommandé de **compiler dans le terminal** (en utilisant `make`).

Exercice 1 : TP5. Cette fiche est la suite du TP5, et ses exercices seront difficiles à réaliser sans l'avoir terminé.

Avant de continuer, résolvez les exercices du TP5, en particulier ceux qui concernent les listes chaînées.

1 Le tri par fusion

Dans cette section, vous allez mettre en œuvre deux variantes du tri par fusion : le première utilise la longueur de la liste, donnée en paramètre, pour séparer la liste en deux moitiés ; la seconde suit le pseudo-code que nous avons vu en classe et évite l'utilisation de la longueur de la liste.

Exercice 2 : Tri par fusion 1 (**). Écrivez et testez les fonctions nécessaires au fonctionnement du tri par fusion, utiliser la longueur de la liste pour la diviser directement en deux moitiés. :

1. `liste_dbl fusionner_1(liste_dbl l1, int n, liste_dbl l2, int m)`, qui fusionne les n premiers éléments de la liste `l1` avec les m premiers éléments de `l2` ;
2. `int tri_fusion_1(liste_dbl* lst, int n)`, qui trie les n premiers éléments de la liste `lst`.

Pour la deuxième version du tri par fusion, nous devons introduire une nouvelle fonction, appelée `separer_liste`, qui prend une liste comme paramètre et la sépare en deux parties à peu près égales.

L'idée principale est la suivante : la fonction utilise deux pointeurs, qui pointent initialement sur le premier élément de la liste et qui avancent itérativement à des vitesses différentes : le premier (appelé pointeur lent) prend l'adresse de l'élément suivant, tandis que le second (appelé pointeur rapide) prend l'adresse de l'élément deux positions après (le suivant du suivant, s'il existe, ou `NULL`). Lorsque le pointeur rapide atteint la fin de la liste, le pointeur lent se trouve approximativement au milieu de la liste. La fonction peut alors remplacer le suivant du pointeur

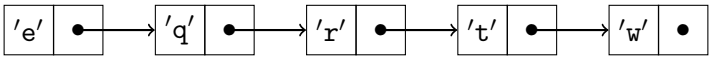
lent par `NULL` et renvoyer un pointeur vers le premier élément et un autre vers l'ancien suivant du pointeur lent.

Exercice 3 : Tri par fusion 2 (**). Écrivez et testez les fonctions nécessaires au fonctionnement du tri par fusion, en utilisant l'idée ci-dessus :

1. `void separer_liste(liste_dbl lst, liste_dbl* l1, liste_dbl* l2);`
2. `liste_dbl fusionner_2(liste_dbl l1, liste_dbl l2)`, qui fusionne les listes `l1` et `l2`;
3. `int tri_fusion_2(liste_dbl* lst)`, qui trie la liste `lst`.

2 Listes chaînées sur des tableaux

Une liste chaînée peut être représentée par deux tableaux, l'un contenant des valeurs (`contenu`) et l'autre contenant des indices (`suivant`). Le chaînage sera effectué de la façon suivante : l'élément suivant `contenu`$[k]$ aura `suivant`$[k]$ comme indice dans le tableau `contenu`. L'accès à la liste se fait par l'indice dans le tableau `contenu` du premier élément. L'élément situé en fin de liste sera associé à une valeur spéciale dans le tableau `suivant` (-1 par exemple). De cette manière, la liste ci-après peut être représentée dans les tableaux ci-dessous :



`lst` = 2

	contenu	suivant
0	'q'	3
1	'w'	-1
2	'e'	0
3	'r'	4
4	't'	1

La valeur à la tête de la liste (`lst`) est l'élément 2 du tableau `contenu`, c'est-à-dire 'e'. Le suivant est l'élément 0 ('q') et ainsi de suite jusqu'au dernier élément 'w' associé à -1 indiquant la fin de la liste.

Pour organiser les données de la liste, nous utilisons la structure suivante :

```
1 typedef struct {
2     double *contenu;
3     int    *suivant;
4     int    p_tete;
5     int    p_libre;
6 } lstbl_dbl;
```

Les variables `contenu` et `suivant` pointent vers des tableaux (qui doivent être alloués) stockant les informations sur les maillons ; les variables `p_tete` et `p_libre` indiquent les positions du premier élément et de la première place libre, respectivement.

Exercice 4 : Opérations simples (**). Mettre en œuvre des opérations (sauf suppression) sur la structure `lstbl_double`, en suivant les opérations implémentées pour les listes chaînées.

Pour cet exercice, vous n'écrivez pas les fonctions de suppression, et on peut supposer que la place libre augmente linéairement (de 1 à chaque fois qu'un élément est inséré).

Si nous ajoutons des opérations de suppression, les places libres se comportent de manière plus complexe. À cette fin, nous représentons les places libres sous la forme d'une liste chaînée (en utilisant le tableau `suivant`). De cette manière,

- Ajouter un élément à une liste consiste à enlever un élément de la liste des places libres et l'inclure dans la liste modifiée ;
- Supprimer un élément d'une liste consiste à l'enlever de la liste active et l'ajouter à la liste des places libres.

Exercice 5 : Liste chaînée des places libres (***). Modifiez les fonctions pour que les places libres soient stockées sous forme de liste chaînée (en utilisant le tableau `suivant`) et ajoutez les opérations de suppression.

Exercice 6 : Liste chaînée sur tableaux dynamiques (***). Écrivez une fonction pour redimensionner les tableaux utilisés dans cette structure, et modifiez les opérations pour redimensionner les tableaux si nécessaire.

3 Les Tas

Exercice 7 : Arbre-tas (***). Implémentez un tas arborescent, sans oublier d'écrire des fonctions pour ajouter un élément, extraire le maximum et vérifier si le tas est vide.

Exercice 8 : Tri par tas en arbre (**). Utilisez le tas en arbre que vous avez implémenté dans l'exercice précédent pour implémenter un tri par tas.

Testez votre code soit par des tests manuels, soit en l'insérant dans le programme de TP3, ce qui vous permettra de comparer votre implémentation au tri par tas en tableau.