TD No4

Exercice 1 tri fusion

Question 1 : exemple

Appliquez l'algorithme du tri fusion vu en cours sur la liste linéaire suivante: 56,4,6,2,7,8,3,1,7,12

Question 2 couper une liste en deux

Dans la suite de cet exercice, les listes linéaires seront représentées par des listes chaînées. Ecrire une fonction C permettant de répartir les éléments d'une liste l dans deux listes l1 et l2 dont la taille diffère d'au plus 1.

Eléments de correction:

On a plusieurs tactiques possibles. La plus simple consiste à déterminer la taille t de la liste et à mettre les t/2 premiers élément dans l1 et les éléments restants dans l2.

Voici l'algorithme correspondant:

```
int taille(liste l){
  if (estVide(l))
    return 0;
  else return 1+taille(l->suivant);
}

void coupeEnDeux(liste l, liste * l1, liste * l2){
  int t=taille(l), cb;
  if ((estVide(l)) || (estVide(l->suivant))){ // 0 ou 1 élément
    *l2=listeVide();
    *l1=l;
  } else{
    liste ltmp=l;
    *l1=l;
    for (cb=0; cb < t/2-1; cb++)
      ltmp=ltmp->suivant;
    *l2=ltmp->suivant;
    ltmp->suivant=listeVide();
  }
}
```

Une alternative consiste à remarquer que l'on n'est pas obligé de couper la liste en son milieu mais que l'on peut répartir les éléments en parcourrant la liste : un élément dans l1, le suivant dans l2 et ainsi de suite. Si l=3;5;4;6;2;8 alors l1=3;4;2 et l2=5;6;8. Voici l'algorithme correspondant :

```
void coupeEnDeux(liste l, liste * l1, liste * l2){
  if ((estVide(l))|| (estVide(l->suivant))) { // 0 ou 1 élément
    *l2=listeVide();
    *l1=l;
  } else {
    *l1=l;
```



```
    l=l->suivant;
    *l2=l;
    l=l->suivant;
    coupeEnDeux(l,&((*l1)->suivant), &((*l2)->suivant));
  }
}
```

Question 3 fusion de listes triées

Ecrire une fonction ayant deux listes triées l1 et l2 comme arguments et retournant la liste obtenue en fusionnant l1 et l2. Votre fonction doit faire la fusion sur place en utilisant directement les éléments de l1 et de l2.

Éléments de correction: la fusion de liste triées est un algorithme classique vu en cours: pour déterminer le premier élément de la liste résultat, il suffit de comparer les premiers éléments de chacune des listes et de prendre le plus petit. Une implantation possible :

```
liste fusion(liste l1, liste l2){

  if (estVide(l1))
    return l2;
  else if (estVide(l2))
    return l1;
  else {
    if (l1->valeur < l2->valeur){
      l1->suivant=fusion(l1->suivant, l2);
      return l1;
    } else {
      l2->suivant=fusion(l1, l2->suivant);
      return l2;
    }
  }
}
```

Dans la prochaine version de ce corrigé, il faudra inclure une version itérative (ràf).

Question 4 tri fusion

Ecrire le fonction triFusion qui applique l'algorithme du tri fusion à une liste.

Éléments de correction:

```
liste triFusion(liste l){
  if ((estVide(l)) || (estVide(l->suivant)))
    return l;
  else {
    liste l1, l2;
    coupeEnDeux(l, &l1, &l2);
    l1=triFusion(l1);
    l2=triFusion(l2);
    return fusion(l1, l2);
```

```
    }
  }
```

Exercice 2 Listes de priorité

Question 1 : liste de priorités

Une liste de priorité est une structure de données contenant des objets ayant un champ priorité (un entier) et sur laquelle on peut réaliser efficacement les opérations suivantes :

- Inserer(e, l): insére un élément dans la liste de priorité
- max(l): Liste -> element: retourne l'élément qui a la plus grande priorité
- extraireMax(l): supprime l'élément qui a la plus grande priorité et retourne sa valeur

Proposer une structure de donnée pour représenter les listes de priorité.

Il nous faut étudiant la complexité des opérations proposées sur les structures de données connues :

- *tableau non triés:*
  - *inserer: O(1) en nb d'affectations (affectation à la fin)*
  - *max: O(n) en nombre de comparaisons car il faut parcourir tout le tableau pour détermine le max*
  - *extraireMax: on trouve l'indice du max et on échange le contenu de la cellule avec la dernière du tableau: O(n) en nb de comparaisons, O(1) en nb d'affectations.*
- *Tableaux triés:*
  - *inserer: au pire O(n) en nb d'affectations (il faut décaler les éléments situés après le point d'insertion) et O(log n) de comparaisons (recherche dichotomique)*
  - *max: O(1) car le max est à la fin*
  - *extraireMax: O(1) en nombre de comparaisons car le max est à la fin, idem pour les nombre d'affectations.*
- *Arbres binaires de recherche équilibrés (AVL ou ~)*
  - *inserer: au pire O(long n) en nb de comparaisons si ajout au feuilles, 0(1) en nombre d'affectations.*
  - *max: O(long n) en parcours d'éléments car le max est tout à droite. Pas de comparaisons ou d'affectations.*
  - *extraireMax: O(long n) en nombre de parcours d'éléments, O(1) en nombre d'affectations car le noeud qui héberge le max n'a pas de fils droit.*
- *Tas*
  - *inserer: au pire O(long n) comparaisons et affectations : on insère et on remonte en vérifiants à chaque fois si la propriété des tas est vérifiées. On va au pire jusqu'à la racine.*
  - *max: O(1): on retourne la valeur de la racine du tas;*
  - *extraireMax: O(long n) au pire en nombre de comparaisons et d'affectations: on échange la racine avec le dernier éléments et on vérifie que la propriété du tas est vérifiée en descendant.*

Le tas est la structure de données choisie.

Question 2 : implantation

Implanter les opérations sur les listes de priorité en C.

éléments de correction :

```
// attention: le premier élément a l'indice 0 (et pas 1 comme dans le cours)
int pere(int n){
  return (n+1)/2-1;
}
int filsg(int n){

  return 2*(n+1)-1;
}

int filsd(int n){
  return 2*(n+1);
}

int estNoeudTas(int n, tableau t){
  return n < t.taille;
}

int indiceDuMax(int a, int b, tableau t){
  if (t.tab[a]>t.tab[b])
    return a;
  else return b;
}

void ordonner (tableau * pt, int noeud){
  int m=noeud;
  if (estNoeudTas(filsg(noeud), *pt)){
    m=indiceDuMax(m, filsg(noeud), *pt);
    if (estNoeudTas(filsd(noeud), *pt))
      m=indiceDuMax(m, filsd(noeud),*pt);
  }
  if (m != noeud){
    echangeElemTab(pt, m, noeud);
    ordonner(pt, m);
  }
}

void ajouteElementTas(tableau * pt, int e){
  int cb=pt->taille;
  pt->tab[pt->taille]=e;
  pt->taille++;
  while (estNoeudTas(cb,*pt) && (pt->tab[cb]>pt->tab[pere(cb)])){
```

```
      echangeElemTab(pt, cb, pere(cb));
      cb=pere(cb);
    }
  }
```

Exercice 3 Tri rapide

Question 1 : algorithme

Le tri rapide est un algorithme dont le principe de fonctionnement est le suivant :

- on choisit un élément appelé pivot dans la liste
- on réorganise la liste de façon à ce que les éléments inférieurs au pivot soient avant le pivot et que ceux qui lui sont supérieurs soient après lui dans la liste.
- On applique l'algorithme au deux partie de la liste (ceux avant et ceux après le pivot).

Quelle propriété vérifie la liste obtenue à la fin de l'algorithme ?

Elément de correction

Elle est triée. :-)

Question 2 : partition

Dans la suite de cet exercice, on suppose que les listes sont représentées par des tableaux. Ecrire une fonction ***partition*** ayant comme argument la partie de liste à traîter et le pivot et qui réorganise la liste en mettre les éléments plus petits que le pivot avant lui et les éléments plus grands après lui.

Éléments de correction:

```
/*                                                              */
/* placer :                                                     */
/* arguments : l'adresse du tableau, les indices des bornes du  */
/*             sous-tableau sur lequel on travaille.            */
/* le pivot est le premier élément du sous-tableau              */
int placer(tableau * t, int debut, int fin){
  int i, s;
  i=debut + 1;
  s=fin;
  while (i <= s)  {
    while (t->tab[s] > t->tab[debut])  s--;
    while ((t->tab[i] <= t->tab[debut]) && (i <= t->taille))  i++;
    if (i<s) {
      echangeElemTab(t,i,s);
      i+=1;
      s-=1;
    }
  }
  echangeElemTab(t,debut,s);
  return s;

}
```


Question 3 : tri rapide

Ecrire le reste des fonctions nécessaires à l'implantation de cet algorithme.

Éléments de correction:

```
/*                                                              */
/*               Le tri rapide                                  */
/*                                                              */
/*                                                              */
void trirapideAux(tableau * t, int debut, int fin){
  int indice_pivot;
  if (debut < fin) {
    indice_pivot=placer(t,debut,fin);
    trirapideAux(t,debut,indice_pivot-1);
    trirapideAux(t,indice_pivot+1,fin);
  }
}
void triRapide(tableau * pt){
      trirapideAux(pt, 0, pt->taille-1);
}
```

Question 4 : complexité au pire

Déterminez la complexité au pire de cet algorithme en nombre de comparaisons

éléments de correction:

Cas d'une liste déjà triée: complexité en n^2.

Question 5 : tri rapide sur les listes chaînées

Cet algorithme est-il implantable efficacement sur les listes chaînées ?

Éléments de correction:

ràf

Question 6 : choix du pivot

Si l'on connaît des informations sur les données, il est possible de choisir le pivot en fonction de ces informations. Indiquez le pivot que vous choisiriez en sachant que la liste à trier est déjà quasiment triée.

Éléments de correction: on prend l'élément qui est au milieu du tableau (indice taille/2).