Travaux Dirigés d'algorithmique n°9

Cours d'Informatique de Deuxième Année

—Deug SCIENCES M3-MI—

Révisions

Cette feuille de travaux dirigés rassemble des exercices donnés lors de précédents examens. Nous finissons cette séance par une étude du tri rapide.

▸ **Exercice 1. (Septembre 96)** *On considère l'algorithme suivant :*

```
x(i), i=[1,n] est un tableau d'entier

pour i allant de 2 a n faire
  j = i ;
  tant que j>1 et x[j-1]>x[j] faire
    echanger(x[j],x[j-1]);
    j = j - 1 ;
  fin faire
fin faire
```

- *Quelle est la complexité de cet algorithme exprimée en fonction de* `n` *?*
- *quel est l'invariant de boucle que l'on peut définir pour la boucle par rapport à* `i` *?*
- *que fait cet algorithme ?*

▸ **Exercice 2. (Juin 96)** *Les listes chaînées d'entiers peuvent être représentées par la structure :*

```
typedef struct Cellule {
   int x;
   struct Cellule * Suivant;
   } * ListeEntier;
```

1. *Donner une fonction* `PremierDernier` *qui prend une liste en argument et qui renvoie une liste (sans recopie), qui est la même que la première, sauf que le premier élément est devenu le dernier. Si la première liste est vide, renvoyer la liste vide.*
2. *En déduire une et une seule fonction récursive* `PairImpair` *qui prend une liste en argument et qui renvoie (sans recopie) une liste identique à la première, à ceci près que tous les éléments pairs sont au début et tous les éléments impairs à la fin. L'ordre des éléments n'a pas d'importance.*

▸ **Exercice 3. (Janvier 97)** *On considère les déclarations suivantes pour une liste de caractères :*

```
typedef struct Cellule {
   char c ;
   struct Cellule * Suivant ;
   } StrCellule ;
typedef StrCellule * ListeCar ;
```

Ecrire une fonction `ExtraitLettres`*, ayant en entrée une liste* `L` *de type* `ListeCar`*, et retournant une autre liste dont les éléments sont toutes les lettres alphabétiques contenues dans* `L`*.*

Par exemple, si la liste `L` *est ('2', 'z', 'B', '-', 'p'), le résultat est ("z', 'B', 'p'). L'ordre dans lequel apparaissent les lettres dans la liste résultat n'a pas d'importance. La liste d'entrée ne doit pas être modifiée.*

▸ **Exercice 4. (Tri rapide)** *Dans le tri par insertion, nous insérons l'élément R_i dans la séquence ordonnée, $R_0, R_1, \ldots, R_{i-1}$ de sorte que la séquence résultante de taille i soit aussi ordonnée. Nous commençons par la séquence ordonnée R_0 et ensuite insérons successivement les enregistrements $R_1, R_2, \ldots, R_{n-1}$. Par conséquent, nous pouvons ordonner une liste de n enregistrements en effectuant $n-1$ insertions. La clé K_i (clé pivot) qui contrôle l'insertion est placée dans une position à droite dans la séquence $R_0, R_1, \ldots, R_{i-1}$.*

Dans le tri rapide, la clé pivot K_i est placée en position s_i de telle sorte que :

$$K_j < K_{s_i} \text{ pour } j < s_i, \text{ et } K_j \geq K_{s_i} \text{ pour } j > s_i.$$

Par conséquent, les séquences $R_0, R_1, \ldots, R_{s_1-1}$, $R_{s_1+1}, \ldots, R_{n-1}$ peuvent être triés indépendemment. Le programme suivant implémente le tri rapide de Hoare.

```
void quicksort(element list[], int left, int right)
/* list[left].key est arbitrairement chosi comme cle pivot. On suppose
list[left].key<=list[right+1].key */
{ int pivot, i, j ;
  element tmp ;
  if (left < right) {
    i = left ; j = right + 1 ; pivot = list[left].key ;
    do{
       do i++ ; while (list[i].key < pivot);
       do j-- ; while (list[j].key > pivot);

       if (i < j) swap(list[i],list[j],tmp);
    } while (i < j);
    swap(list[letf],list[j],tmp);
    quicksort(list,left,j-1);
    quicksort(list,j+1,right);
  }
}
```

Appliquez la fonction `quicksort` *sur la séquence [26, 5, 37, 1, 61, 11, 59, 15, 48, 19]. Que se passe-t-il si on interchange les éléments 61 et 26 ?*