TP5 : structures, chaînes de caractères

Algorithmique et projet de programmation — M1 mathématiques

31 octobre 2016

1 Listes chaînées

Les listes chaînées sont une structure standard de l'algorithmique et sont complémentaires aux tableaux. Un tableau est une structure efficace quand on sait à l'avance (une borne sur) le nombre d'éléments à stocker : on peut alors accéder aux éléments par leur index en $O(1)$. Ce qui permet une telle efficacité est la contiguité des cases du tableau en mémoire, ramenant l'accès à la case i à une simple addition sur les pointeurs. En revanche, ils sont très inefficace concernant l'ajout ou la suppression d'éléments : pour ajouter un élément en tête d'un tableau de taille n, il faut allouer un tableau de taille $n+1$, puis copier l'entièreté du tableau dans les n dernières cases avant enfin d'affecter la première case à la valeur à ajouter... tout cela se fait au mieux en $O(n)$!

Les listes chaînées prennent le point de vue opposées. Au diable la contiguité en mémoire et l'arithmétique des pointeurs : une case est maintenant stockée avec un pointeur sur la case suivante. Rajouter une valeur en tête d'une telle liste est alors un jeu d'enfant : on alloue une case mémoire où bon nous semble dans la mémoire (le plus souvent sur le tas) et on lui associe un pointeur vers l'ancienne première case de la liste. Et tout ça en $O(1)$! Bien entendu, on a perdu l'accès à la i-ème case en temps constant, il faut maintenant partir du premier élément et suivre les pointeurs successifs jusqu'à atteindre le i-ième élément... Tout dépend donc de ce que l'on veut faire, il n'y a pas de solution miracle.

Exercice 1. Implémenter une structure `C` codant les listes châinées d'entiers. Indice : au même titre qu'en `C` un tableau d'entiers est en fait un pointeur vers le premier entier de ce tableau, une liste chaînée d'entiers sera un pointeur vers la première instance d'une structure à deux champs :

```
typedef list struct node *;

struct node {
  // two fields
};
```

Écrire une fonction renvoyant une liste vide :

```
list new_list ();
```

Exercice 2. Écrire une fonction prenant en paramètres une liste et un entier et renvoyant la concaténation de cet entier en tête de liste :

```
list push (list l, int val);
```

Si les tableaux étaient adaptés à un traitement itératif, les listes chaînées, elles, se prêtent plus facilement à un traitement recursif.

Exercice 3. Écrire une fonction calculant la longueur d'une liste :

```
int length (list l);
```

Exercice 4. Écrire une fonction retournant une liste (le dernier élément est maintenant le premier, l'avant-dernier le deuxième, etc.) :

```
void reverse (list l);
```

Attention, cette fonction devra avoir une complexité en temps de $O(n)$ (avec n la longueur de la liste).

2 Manipulation de fichiers

On va maintenant s'atteler à recoder (une version simple de) la commande `sort` des sytèmes UNIX. C'est un petit programme qui prend en paramètre un chemin d'accès à un fichier texte et affiche à l'écran l'ensemble de ses lignes triées alphabétiquement. Ainsi, si un fichier `phonebook` contient :

```
Zaid 0673894765
Bernard 0663896735
Lea 0612673546
Clement 0678900034
Adrien 0694367833
Claudia 0642356732
```

L'exécution de la commande `sort phonebook` affichera :

```
Adrien 0694367833
Bernard 0663896735
Claudia 0642356732
Clement 0678900034
Lea 0612673546
Zaid 0673894765
```

Exercice 5. Écrire une fonction qui compare deux chaînes de caractères dans l'ordre alphabétique :

```
char compare_str (char* s, char* t);
```

Elle renverra `1` si `s` est inférieur à `t` au sens large et `0` sinon. On rappelle que `s` $\leq$ `t` si et seulement si

- `s` est un préfixe de `t`
- il existe $i \geq 0$ tel que `s[k] = t[k]` pour tout $0 \leq k \leq i-1$ et `s[i]` $\leq$ `t[i]`.

Exercice 6. Écrire une fonction adaptée des fonctions de tri vues la dernière fois :

```
void sort_str (char** strs);
```

qui trie alphabétiquement un tableau de chaînes de caractères.

Rappelons que pour traiter les fichiers textes, le langage C dispose des fonction suivantes, disponibles dans la bibliothèque `stdio.h` :

```c
/*
  fopen takes the file path as a string and the opening mode that
  could be multiple thing: for our use, we just consider mode to be
  "r", opening the file in read-only mode.

  It returns a pointeur to a FILE structure, that we shall consider
  opaque.

  E.g.: FILE* f = fopen("path/to/my/file", "r");
*/
FILE* fopen (const char* path, const char* mode);

/*
  fgetc takes a pointer to a FILE and returns either a character just
  read in the file f or EOF, a constant meaning the End Of the File is
  reached.

  Beware that, when it does returns a character, it casts it to an
  int.

  E.g. : if( (c = fgetc(f)) != EOF ) printf("%c", (char) c);
*/
int fgetc (FILE* f);

/*
  fgets takes a string s, a buffer length size and a pointer to a FILE
  f; it reads in f a string of length size-1 (stopping to the first
  '\n' or EOF if encountered) and puts it inside s. It writes a '\0'
  as the last character of s. It returns s if everything went right
  and NULL if not or if EOF is encountered before reading anything.

  E.g. : char s[10]; if ( fgets(s,10,f) != NULL ) printf("%s",s);
*/
char* fgets (char* s, int size, FILE* f);

/*
  rewind puts the current position in the file back to the beginning
  of the file pointed by f.

  It behaves like closing (see below) and reopening the file.
*/
void rewind (FILE* f);

/*
  fclose frees the memory allocated for the FILE structure pointed by
  f; it returns 0 upon success (and something else upon failure we are
  not interested in here).
*/
int fclose(FILE* f);
```

Exercice 7. En déduire un programme implémentant la commande `sort` décrite précédemment. En particulier, on déterminera le nombre de lignes et la taille maxi-

male des lignes du fichier passé en paramètre pour allouer un tableau de taille suffisante.

3 Pour le 7 novembre 2016

Déterminer des binômes et commencer à regarder le projet. Préparer des questions si besoin à poser la fois prochaine.