Algorithmique III. L2 Informatique I41.

TD 10. Listes, piles, files et évaluation à l'aide d'une pile [1]

EXERCICE 1. On considère une liste L dont la structure de données associée `L` est une liste chaînée, une liste est donc une *référence*, elle n'est que la clé pour accéder à l'objet référencé noté ici `L»`. Les expressions `L»val` et `L»suiv` désignent respectivement la valeur contenue dans la cellule en tête de la liste L et la sous-liste suivante. On note `[]` la liste vide. On ne se préoccupe ni d'allocation mémoire ni de libération mémoire, contrairement au langage C.

1. Écrivez les algorithmes suivants :

- `InsQueue(@L,x)` qui rajoute une cellule contenant `x` en fin de liste.
- `SupQueue((@L)` qui renvoie la valeur contenue dans la dernière cellule et élimine cette cellule de la liste.
- `InsTete(@L,x)` qui rajoute une cellule contenant `x` en début de liste.
- `SupTete(@L)` qui renvoie la valeur contenue dans la cellule en tête de liste et élimine cette cellule de la liste.

2. Implantez ces algorithmes sous forme de fonctions en langage C à l'aide des structures prédéfinies suivantes (le type `tval` n'est pas spécifié) :

```
typedef struct tcell{
   tval val;
   struct tcell *suiv;
} tcell;

typedef tcell *tliste;
```

EXERCICE 2. Un mot sur l'alphabet $\Sigma := \{\texttt{\{}, \texttt{(}, \texttt{[}, \texttt{)}, \texttt{]}, \texttt{\}}\}$ est bien parenthésé s'il est défini inductivement par l'une des règles de construction suivantes :

(a) mot := ε (le mot vide).

(b) mot := $\langle$mot$\rangle$ où $(\langle, \rangle) \in \{(\texttt{\{}, \texttt{\}}), (\texttt{(}, \texttt{)}), (\texttt{[}, \texttt{]})\}$.

(c) mot := mot.mot (concaténation).

Par exemple, le mot `{(())}[()]` est bien parenthésé alors que les mots `())()` et `[{])}` ne le sont pas.

1. Écrivez un algorithme `EstBP(mot)` qui utilise une pile pour décider si un `mot` de Σ^* est bien parenthésé ou non.

2. Quelle est sa complexité ?

3. S'il n'y avait qu'un seul type de parenthèses, pourrait-on procéder plus simplement ?

EXERCICE 3. Écrivez un algorithme `Renverser(@pile)` qui renverse les éléments de la pile passée en paramètre et *in situ*. Autrement dit, après l'exécution, les valeurs apparaissent dans l'ordre inverse de celles de la liste originale. Écrivez cet algorithme en langage C.

EXERCICE 4. Dans la suite `L` désigne une liste chaînée.

1. Écrivez un algorithme `Inserer(@L,pos,A)` qui insère la liste atomique `A` *après* la cellule référencée par `pos` de la liste `L`. On suppose que `A` peut désigner n'importe quelle cellule de la liste `L`. Écrivez cet algorithme en langage C.

2. Écrivez une fonction `tliste Fusionner(tliste *A, tliste *B)` qui fusionne deux listes triées `A` et `B` et renvoie la liste fusionnée triée à partir des cellules de `A` et `B` qui seront vides à l'issue du fusionnement.

EXERCICE 5. Écrivez l'algorithme du tri insertion en langage C qui trie les valeurs d'une liste *doublement* chaînée dans l'ordre croissant. Indications : il suffit de rajouter le champ `struct tcell *prec;` dans la structure d'une cellule pour créer un chaînage double.

EXERCICE 6. On dispose d'une mémoire M de taille n codée par un tableau `M` indexé de 1 à n de cellules à deux champs, un booléen `libre` indiquant si la cellule est libre et une valeur `val` contenant l'information à stocker. Initialement toutes les cellules sont libres.

1. Écrivez un algorithme `Reserver(i,k)` (resp. `Liberer(i,k)`) qui réserve (resp. libère) les k cellules mémoire à partir de la position `i` en instanciant leurs champs `libre` à *faux* (resp. *vrai*).

1. Version du 5 juillet 2022, 12 : 20

(2) Écrivez un algorithme `ChercherLibre(k)` qui renvoie le plus petit indice i tel que les cellules $M[i+r]$ sont libres pour tout $r \in [\![0, k-1]\!]$. L'algorithme renvoie 0 si aucun bloc de taille k n'est libre.

2. Faites une preuve d'arrêt et justifiez l'algorithme.

3. Écrivez un algorithme `Allouer(k,@i)` qui cherche l'adresse i du premier bloc de k cellules libres de la mémoire et les réserve puis renvoie *vrai* s'il a pu les trouver et *faux* sinon.

4. Calculez la/les complexité(s) de l'algorithme d'allocation.

EXERCICE 7. Écrivez les expressions arithmétiques suivantes sous forme postfixe et calculez leurs valeurs à l'aide d'une pile et représentez l'évolution de cette pile lors de l'évaluation. Utilisez le symbole $\sim$ pour distinguer l'opérateur unaire moins de l'opérateur binaire $-$ dans les expressions postfixes.

(a) $(3-6)(2+1)$

(b) $4(1+2+3)$

(c) $(1+3)-5+(2+(-1))$

(d) $3+2((7+1)-(5+2))(-2)$.