Algorithmique et Structures de données

L2 2021-2022

Travaux Pratiques 6 - Pile et File v2

`gcc mon_fichier.c -std=c11 -Wall -Wextra -o mon_programme`
Les exercices marqués d'une étoile sont optionnels et à faire dans un second temps.

Dans ce tp on travaille sur les piles et files d'arbres. On utilisera les définitions de types suivantes pour les arbres.

```
typedef struct noeud
{
   int val;

   struct noeud* g;
   struct noeud* d;
} Noeud;
typedef Noeud* Arbre;
```

On implantera les files et piles en utilisant des tableaux obtenus avec `malloc` et `realloc`.

```
typedef struct file
{
   int queue; // place de la queue de la file
   int tete;  // place de la tête de file
   Arbre* tab; // tableau de taille tMax destiné à des pointeurs sur Noeuds
      // modifié par rapport à la version 1
   int tMax; // capacité maximum de la file
} File;

typedef struct pile
{
   int sommet;  // pointe sur le sommet de la pile
   Arbre* tab; // tableau de taille tMax destiné à des pointeurs sur Noeuds
      // modifié par rapport à la version 1
   int tMax; // capacité maximum de la pile
} Pile;
```

Exercice 1. *File*

Implanter les opérations :

- `File* initialiserF(void);` qui alloue et renvoie une file vide.
- `void enfiler(File* f,Arbre a);` ajoute un élément et s'agrandit si la capacité Max de la pile est atteinte (dans un tel cas utiliser `realloc` et augmenter la capacité d'au moins 100 cases en plus).
- `Arbre Dfiler(File* f);` renvoie la tête de file.
- `int estVideF(File* f);` indique si la file est vide.
- `void detruireF(File* f);` qui détruit une file et libère toute la mémoire utilisée pour la file `f`.
- * écrire les fonctions `Arbre sommet(File* f)` qui renvoie le sommet sans modifier la file, `vider(File* f)` qui vide complètement la file.

Exemple :

```
Arbre a,b;
File* f=initialiserF();
```

```
.
.
enfiler(f,a);
.
.
b=Dfiler(f);
.
.
if(estVide(f)==...){
.
.
}
.
detruireF(f);
```

Exercice 2. *En largeur*

Implanter `void parcoursLargeur(Arbre a)` qui parcourt un arbre en largeur en affichant les étiquettes des nœuds. Par exemple un parcours en largeur de l'arbre dessiné plus bas donnera : 7 4 6 2 3 5.

Exercice 3. *Pile*

Implanter les fonctions de manipulation de pile d'arbres :

- `Pile* initialiserP(void);`.
- `void empiler(Pile* p,Arbre a);`.
- `Arbre Dpiler(Pile* p);`.
- `int estVideP(Pile* p);`.
- `void detruireP(Pile* p);`.

Exercice 4. *Profondeur sans récursion*

Écrire une fonction `void parcoursPrefixe(Arbre a)` sans appel récursif qui affiche les étiquettes des noeuds de l'arbre `a` en un parcours en profondeur préfixe.

Exercice 5. **Construction suffixe*

On souhaite construire une fonction `Arbre construireSuffixe(int* tab)` qui construit un arbre étiqueté par des entiers positifs à partir d'un parcours suffixe de ses noeuds. **La fonction ne doit pas utiliser d'appel récursif.** L'arbre vide sera codé par -1 et la fin du parcours par un entier inférieur ou égale à -2. (Aide : s'inspirer de l'exercice 3 du TD3). Une suite invalide (qui ne correspond pas à un parcours suffixe) renverra NULL et n'allouera pas de zones mémoire supplémentaires.

Exemple :

```
int T[]={-1,-1,2,-1,-1,3,4,-1,-1,5,-1,6,7,-2,-1,412,44};
Arbre a=construireSuffixe(T);
```

Construira l'arbre suivant :