Algorithmique et Structures de données 1

L2 2021-2022

Travaux Pratiques 6

Site du cours : `https://defelice.up8.site/algo-struct.html`
Les exercices marqués de (@) sont à faire dans un second temps.
Un fichier écrit en langage C se termine conventionnellement par `.c`.
Une commande de compilation est `gcc fichier_source1.c fichier_source2.c fichier_source3.c`.
Voici des options de cette commande.

- `-o nom_sortie` pour donner un nom au fichier de sortie (par défaut `a.out`).
- `-Wall -Wextra` pour demander au compilateur d'afficher plus de Warnings
- `-std=c11` pour compiler selon la norme C11
- `-g -fsanitize=address` pour compiler avec information de débogage et en interdisant la plupart des accès a une zone mémoire non reservée.

Exemple : `gcc -Wall fichier1.c -o monprogramme`

Exercice 1. *Mauvais Code*

Exécuter ce programme sans option de compilation de sécurité (comme `-fsanitize=address`).

```c
int main(void)
{
    int a=0;
    int T[10];
    int b=0;
    T[10]=1;
    printf("%d",b);

    return 0;
}
```

1. L'exécution produit-t-elle toujours une erreur detectée par le système d'exploitation ?
2. Quelle est l'instruction qui pose problème ? et comment le problème peut-il se manifester ?
3. Pourquoi le programme se comporte ainsi ?

Exercice 2. *triqq.c*

Le fichier `triqq.c`, disponible sur le site du cours, contient la définition de la fonction `queFaisJe(int n,int* t,int num)`. Cette fonction effectue une opération (on ne sait pas laquelle) sur le tableau `t` de taille `n`. L'opération effectuée dépend de la valeur donnée à `num`. Le paramètre `num` doit prendre une valeur comprise entre 0 et `QUEFAISJENUMMAX`-1 (compris). Le but de l'exercice est de trouver les valeurs de `num` pour laquelle la fonction `queFaisJe` se comporte comme un `tri` dans l'ordre croissant et parmi ces tris trouver le (ou les plus) efficace en temps, en pratique. La déclaration de `queFaisJe` et la définition de `QUEFAISJENUMMAX` sont faites dans le fichier `triqq.h`.

On utilise la structure `C` suivante pour représenter un tas.

```c
#define MAX // taille maximum du tas

typedef struct s_tas_t
{
   int n; // taille du tas
   int valeurs[MAX]; // zone de mémoire contenant les valeurs du tas
}tas_t;
```

Exercice 3. *Est un tas*

Écrire une fonction `int estUnTas(int n,int* T)` qui renvoie 1 si le tableau `T` de taille `n` est un tas, 0 sinon.

Exercice 4. *Tas*

Implanter les méthodes

1. `void vider(tas_t* t)` qui initialise un tas vide.
2. `int extraireMin(tas_t* t)` qui extrait la plus petite valeur du tas en $O(\log n)$ opérations, n étant la taille du tas.
3. `void ajouter(tas_t* t,int v)` qui ajoute un élément au tas en $O(\log n)$ opérations, n étant la taille du tas.

Exercice 5. *Tri par tas*

Utiliser la structure de tas pour en déduire un algorithme de tri `void triParTas(int n,int* t)` (appelé tri par tas) d'un tableau en $\Theta(n \log n)$ opérations.

Idée :

— Au départ le tas est vide puis on le fait grandir en ajoutant une à une les valeurs du tableau. Ensuite, une fois que le tas contient toutes les valeurs, on extrait systematiquement la plus petite valeur du tas qui est mise juste après la fin du tas. Une fois le tas vide, le tableau est trié dans l'ordre décroissant.

— Le tas grandit à l'interieur du tableau, aucune zone mémoire en plus du tableau `t` n'est utilisée pour stocker le tas.