Carnet de Travaux Pratiques
Algorithmique et structures de données
Licence 2 Informatique

Julien Bernard

Table des matières

Travaux Pratiques d'Algorithmique n° 1

Exercice 1 : Mon premier programme en C

Le langage C est le langage que nous allons utiliser et apprendre pendant le cours d'Algorithmique. C'est un langage compilé, c'est-à-dire qu'il est nécessaire d'appeler un compilateur pour produire un exécutable qui sera fonctionnel uniquement sur la machine sur laquelle il a été compilé.

Question 1.1 Recopier le code source suivant dans un fichier appelé `hello.c`.

```
#include <stdio.h>

int main() {
  printf("Hello World\n");
  return 0;
}
```

Question 1.2 Compiler le programme avec la ligne de commande suivante :

```
gcc -Wall -std=c99 -O2 -g -o hello hello.c
```

Question 1.3 Exécuter le programme :

```
./hello
```

Exercice 2 : Découverte du langage C

Le but de cet exercice est de découvrir le langage C. Ce langage fait partie de la même famille que Java au niveau syntaxique, beaucoup de constructions sont similaires, de même que certains types.

Question 2.1 Recopier le code source suivant dans un fichier appelé `arg.c`.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("Un argument est nécessaire !\n");
    return 1;
  }

  int arg = atoi(argv[1]);
  printf("L'argument est : %d\n", arg);

  return 0;
}
```

Dans ce programme, on utilise la ligne de commande pour fournir un nombre au programme. S'il n'y a pas assez d'argument, un message d'erreur est renvoyé et le programme s'arrête. Sinon, l'argument est transformé en nombre grâce à la fonction `atoi(3)`. Puis, on affiche le nombre grâce à `printf(3)`. Cette fonction prend comme argument une chaîne de caractères entre guillemets, puis éventuellement des noms d'identifiants de variables. Lors de l'exécution, la chaîne de caractères est affichée ainsi que la valeur de chaque variable à l'endroit indiqué. Les variables doivent apparaître dans l'ordre de leur apparitions dans la chaîne de caractères. Un entier est marqué grâce à `%d`, un flottant grâce à `%f`, un caractère grâce à `%c`.

```
$ ./arg 32
L'argument est : 32
```

On utilisera ce code comme base pour les questions suivantes.

Question 2.2 Faire un programme qui affiche la suite de Collatz. L'argument sera l'élément initial de la suite. Pour rappel, la suite de Collatz est définie par un élément initial u_0 strictement positif et :

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est divisible par 2} \\ 3u_n + 1 & \text{sinon} \end{cases}$$

On utilisera une boucle `while` et on s'arrêtera quand u_n vaudra 1.

```
$ ./collatz 46
46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

Question 2.3 Faire un programme qui affiche les nombres de 1 à n où n est passé en argument. Dans cette suite, on remplace les multiple de 3 par «Fizz», les multiple de 5 par «Buzz» (et donc les multiples de 3 et 5 par «FizzBuzz»). On utilisera une boucle `for`.

```
$ ./fizzbuzz 16
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16
```

Question 2.4 Faire un programme qui permet d'afficher un triangle d'une longueur qu'on précisera en argument. On utilisera une double boucle `for`. Par exemple, pour un argument de 7, le programme affichera :

```
$ ./triangle 7
#
# #
# # #
# # # #
# # # # #
# # # # # #
# # # # # # #
```

Exercice 3 : Bibliothèque de fonctions sur les chaînes de caractères

Cet exercice fait l'objet d'une **évaluation notée**. Le travail est à réaliser entièrement en **monôme**.

1. Télécharger l'archive `strings.tar.gz` sur MOODLE et l'extraire dans votre répertoire de travail.

   ```
   tar zxvf strings.tar.gz
   ```

2. Entrer dans le répertoire `strings` créé et construire le projet.

   ```
   cd strings
   make
   ```

3. Exécuter les tests.

   ```
   ./stringslib
   ```

4. Compléter l'implémentation des fonctions dans le fichier `stringslib.c`. La documentation est dans `stringslib.h`. Vous ne devez pas inclure l'entête `<string.h>` (et de toute façon, il y a de légère subtilités entre les fonctions de la bibliothèque standard et celles qui vous sont demandées). Vous ne devez pas modifier le prototype des fonctions.

Une fois que vous aurez fini, vous devrez rendre votre travail en respectant scrupuleusement les consignes qui suivent :

1. Rendre le fichier `stringslib.c` sur le dépôt MOODLE prévu à cet effet avant la date indiquée.
2. Rendre uniquement ce fichier là, sans le renommer, sans le compresser, sans le mettre dans un répertoire.
3. Rendre le fichier avant l'heure indiquée, après il sera trop tard, donc ne pas attendre la dernière minute.
4. Rendre un fichier qui compile correctement et qui produit un exécutable qui peut s'exécuter sans erreur de segmentation, c'est-à-dire qui s'exécute correctement.

L'évaluation tiendra compte des éléments suivants :
— la correction de vos algorithmes ;
— l'absence de corruption mémoire ;
— l'absence de fuite mémoire ;
— le respect des consignes.
Vous serez corrigés en partie par une moulinette automatique qui sera sans pitié si certaines consignes ne sont pas respectées.

Travaux Pratiques d'Algorithmique n° 2

Exercice 4 : Tableaux

Pour chaque algorithme, on donnera l'opération considérée et sa complexité. On utilisera chaque fonction dans `main` sur un exemple quelconque.

Question 4.1 Écrire une fonction qui alloue un tableau d'entiers à l'aide de `calloc(3)`. Utiliser la fonction `rand(3)` pour remplir le tableau à l'aide de valeur entre 0 et 99. Dans `main`, on appellera cette fonction avec 10000 comme argument.

```
int *array_new(size_t size);
```

Question 4.2 Écrire une fonction qui renvoie l'indice de l'élément le plus grand du tableau.

```
size_t array_index_max(const int *data, size_t size);
```

Question 4.3 Écrire une fonction qui calcule la somme des éléments du tableau.

```
int array_sum(const int *data, size_t size);
```

Question 4.4 Écrire une fonction qui renvoie le nombre d'occurrences dans le tableau d'une valeur passées en paramètre.

```
size_t array_count(const int *data, size_t size, int value);
```

Question 4.5 Écrire un algorithme qui effectue un décalage du tableau d'une case vers la gauche, la première valeur étant placée à la fin du tableau.

```
void array_shift_left(int *data, size_t size);
```

Question 4.6 Écrire une fonction qui renvoie l'indice de la plus grande suite de nombres pairs dans le tableau.

```
size_t array_longest_even_seq(const int *data, size_t size);
```

Exercice 5 : Tableaux avancés

Pour chaque algorithme, on donnera l'opération considérée et sa complexité. On utilisera chaque fonction dans `main` sur un exemple bien choisi.

Question 5.1 Écrire une fonction qui renverse l'ordre des éléments d'un tableau.

```
void array_swap(int *data, size_t size);
```

Question 5.2 Écrire une fonction qui supprime les éléments consécutifs égaux pour n'en garder qu'un seul. La fonction renverra la nouvelle taille du tableau.

```
size_t array_uniq(int *data, size_t size);
```

Question 5.3 Écrire une fonction qui détermine si le tableau suivant représente une permutation, c'est-à-dire qu'il contient une et une seule fois tous les nombres compris entre 0 et (`size` - 1).

```
bool array_is_permutation(const int *data, size_t size);
```

Travaux Pratiques d'Algorithmique n° 3

Exercice 6 : Bibliothèque de structures de données

Cet exercice fait l'objet d'une **évaluation notée**. Le travail est à réaliser entièrement en **monôme** sur votre temps libre tout au long du semestre.

Il consiste à implémenter une bibliothèque contenant toutes les structures de données vues en cours ainsi que les algorithmes associés. Cette bibliothèque vous sera utile dans les TP (notamment les projets). Elle contient un squelette de toutes les fonctions ainsi que des tests unitaires, c'est-à-dire des tests qui vérifient que les fonctions sont conformes aux spécifications. Cette bibliothèque utilise des entiers comme données. Il se peut que vous ayez à adapter les structures de données si jamais vous utilisez ces fonctions dans un projet.

1. Télécharger l'archive `biblialgo.tar.gz` sur MOODLE et l'extraire dans votre répertoire de travail.

   ```
   tar zxvf biblialgo.tar.gz
   ```

2. Entrer dans le répertoire `biblialgo` créé et construire le projet.

   ```
   cd biblialgo
   make
   ```

3. Exécuter les tests.

   ```
   ./algorithms
   ```

4. Compléter l'implémentation des fonctions dans le fichier `algorithms.c`. La documentation est dans `algorithms.h`. Vous ne devez pas modifier le prototype des fonctions.

L'implémentation se fera tout au long du semestre :

— Une fois les tableaux vus en cours, implémenter les algorithmes relatifs à `struct array` (sauf les algorithmes de tris et les algorithmes de tas).
— Une fois les listes vues en cours, implémenter les algorithmes relatifs à `struct list`.
— Une fois les tris vus en cours, implémenter algorithmes de tris relatifs à `struct array` (sauf le tri par tas) et `struct list`.
— Une fois les arbres binaires de recherche vus en cours, implémenter les algorithmes relatifs à `struct tree`
— Une fois les tas vus en cours, implémenter les algorithmes de tas relatifs à `struct array`.

Une fois que vous aurez fini, vous devrez rendre votre travail en respectant scrupuleusement les consignes qui suivent :

1. Rendre le fichier `algorithms.c` sur le dépôt MOODLE prévu à cet effet avant la date indiquée.
2. Rendre uniquement ce fichier là, sans le renommer, sans le compresser, sans le mettre dans un répertoire.
3. Rendre le fichier avant l'heure indiquée, après il sera trop tard, donc ne pas attendre la dernière minute.
4. Rendre un fichier qui compile correctement et qui produit un exécutable qui peut s'exécuter sans erreur de segmentation, c'est-à-dire qui s'exécute correctement.

L'évaluation tiendra compte des éléments suivants :
— la correction de vos algorithmes ;
— l'absence de corruption mémoire ;
— l'absence de fuite mémoire ;
— le respect des consignes.

Vous serez corrigés en partie par une moulinette automatique qui sera sans pitié si certaines consignes ne sont pas respectées.

Exercice 7 : Crible d'Ératosthène

Un nombre est dit premier s'il admet exactement 2 diviseurs distincts (1 et lui-même). 1 n'est donc pas premier. On désigne sous le nom de crible d'Ératosthène une méthode de recherche des nombres premiers plus petits qu'un entier naturel n donné. La méthode est la suivante :

1. On supprime tous les multiples de 2 inférieurs à n.
2. L'entier 3 n'ayant pas été supprimé, il ne peut être multiple des entiers qui le précèdent, il est donc premier. On supprime alors tous les multiples de 3 inférieurs à n.
3. L'entier 5 n'ayant pas été supprimé, il ne peut être multiple des entiers qui le précèdent, il est donc premier. On supprime alors tous les multiples de 5 inférieurs à n.
4. Et ainsi de suite jusque n. Les valeurs n'ayant pas été supprimées sont les nombres entiers plus petits que n.

Pour programmer cette méthode, on va utiliser un tableau `is_prime` de n entiers qui contiendra des 1 et des 0. On donne à ces 1 et 0 le sens suivant : si `is_prime[i]` vaut 0 alors `i` n'est *pas premier*, et si `is_prime[i]` vaut 1 alors `i` est *premier*. Lors de la méthode, on élimine les nombres non premiers au fur et à mesure qu'on les rencontre. Donc au départ, on suppose que tous les entiers sont premiers.

Question 7.1 Récupérer la taille n sur la ligne de commande.

Question 7.2 Allouer un tableau `is_prime` de taille n à l'aide de `calloc(3)`.

Question 7.3 Initialiser ce tableau avec des 1.

Question 7.4 Implémenter la méthode du crible d'Ératosthène.

Question 7.5 Afficher les nombres premiers inférieurs à n.

```
$ ./eratosthene 100
Nombres premiers inférieurs à 100 :
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Exercice 8 : Tri d'un tableau binaire

On considère un tableau dont les éléments appartiennent à l'ensemble $\{0, 1\}$. On se propose de trier ce tableau. À chaque étape du tri, le tableau est constitué de trois zones consécutives, la première ne contenant que des 0, la seconde n'étant pas triée et la dernière ne contenant que des 1.

zone de 0	zone non triée	zone de 1

On range le premier élément de la zone non triée si celle-ci n'est pas encore vide : si l'élément vaut 0, il ne bouge pas ; si l'élément vaut 1, il est échangé avec le dernier élément de la zone non triée. Dans tous les cas, la longueur de la zone non triée diminue de 1.

Question 8.1 Créer un tableau de 100000 éléments de 0 et de 1 tirés au hasard.

Question 8.2 Compter le nombre de 1 du tableau.

Question 8.3 Trier le tableau selon la méthode indiquée.

Question 8.4 Vérifier que le nombre de 1 est identique au nombre de 1 précédemment calculé.

Question 8.5 Quel est la complexité de cet algorithme ?

Travaux Pratiques d'Algorithmique n° 4

Exercice 9 : Implémentation d'une pile avec une liste chaînée

Le but de cet exercice est de proposer une implémentation d'une pile grâce à une liste chaînée.

On dispose de la structure de donnée suivante :

```
struct stack_node {
  int data;
  struct stack_node *next;
};

struct stack {
  struct stack_node *first;
};
```

Question 9.1 Donner le code d'une fonction qui initialise une pile vide :

```
void stack_create(struct stack *self);
```

Question 9.2 Donner le code d'une fonction qui examine si la pile est vide :

```
bool stack_is_empty(const struct stack *self);
```

Question 9.3 Donner le code d'une fonction qui empile un élément :

```
void stack_push(struct stack *self, int data);
```

Question 9.4 Donner le code d'une fonction qui donne la valeur du premier élément d'une pile non-vide :

```
int stack_top(const struct stack *self);
```

Question 9.5 Donner le code d'une fonction qui dépile un élément :

```
void stack_pop(struct stack *self);
```

Question 9.6 Donner le code d'une fonction qui détruit une pile :

```
void stack_destroy(struct stack *self);
```

Exercice 10 : Manipulation de liste chaînée

On utilise la structure de liste suivante :

```
struct list_node {
  int data;
  struct list_node *next;
};

struct list {
  struct list_node *first;
};
```

Question 10.1 Donner le code d'une fonction qui prend en paramètre une liste et retourne une liste miroir, c'est-à-dire avec les mêmes éléments en ordre inverse. Quelle est sa complexité ?

```
void list_mirror(const struct list *self, struct list *res);
```

Question 10.2 Donner le code d'une fonction qui prend en paramètre une liste et retourne une copie de la liste. Quelle est sa complexité ?

```
void list_copy(const struct list *self, struct list *res);
```

Question 10.3 Donner le code d'une fonction qui prend en paramètre deux listes et qui renvoie la concaténation des deux listes. Quelle est sa complexité ?

```
void list_concat(const struct list *l1, const struct list *l2,
    struct list *res);
```

Travaux Pratiques d'Algorithmique n° 5

Exercice 11 : Tours de Hanoï

Le problème des tours de Hanoï est un grand classique illustrant l'usage de la récursivité. On a trois tours A, B et C et n disques numérotés 1 , 2, ..., n. En position initiale, tous les disques sont sur la tour A, chaque disque reposant sur un disque de taille plus grande. À chaque étape, on a le droit de déplacer le disque du haut d'une tour pour le mettre sur une autre tour, à condition qu'un disque ne soit jamais posé sur un disque plus petit.

Le but du jeu est de déplacer les n disques de la tour A vers la tour B comme illustré par la figure 1.

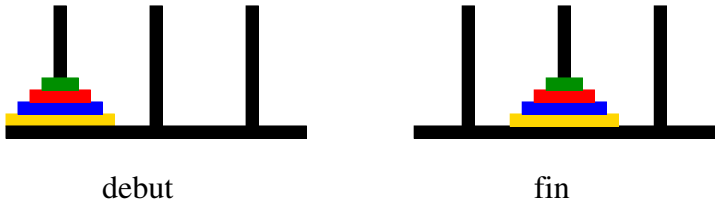

FIGURE 1 – Position de début et de fin pour les tours de Hanoï

Question 11.1 Proposez une solution dans les cas $n = 1, 2, 3, 4$.

Question 11.2 On voit se dessiner une approche récursive. Proposez un algorithme pour ce problème qui va décrire à l'utilisateur les mouvements à effectuer.

Question 11.3 Analysez le nombre de mouvements de disques.

Question 11.4 On considère la structure suivante pour représenter une tour. La structure `struct disc` représente un disque, et en particulier, sa taille (`width`). La structure `struct tower` est alors une pile de disques.

```
struct disc {
  int width;
  struct disc *next;
};
struct tower {
  struct disc *first;
};
```

Coder les fonctions de bases suivantes qui permettent de manipuler la structure.

```
// create an empty tower
void tower_create(struct tower *self);
// push a disc of size k on top of the tower
void tower_push_disc(struct tower *self, int width);
// pop the top disc and return its size
```

```
int tower_pop_disc(struct tower *self);
// print the content of the tower
void tower_print(const struct tower *self);
```

Question 11.5 Le jeu comprend trois tours. Proposer un algorithme pour initialiser le jeu avec n disques.

```
struct hanoi {
  struct tower towers[3];
};
void hanoi_create(struct hanoi *self, int n);
```

Question 11.6 Proposer un algorithme qui affiche le jeu.

```
void hanoi_print(const struct hanoi *self);
```

Question 11.7 Proposer un algorithme qui déplace le disque de la tour i vers la tour j.

```
void hanoi_move_one_disc(struct hanoi *self, int i, int j);
```

Question 11.8 Proposer un algorithme qui déplace les n disques de la tour i vers la tour j. La tour k qui n'est ni i, ni j pourra être calculé par la formule $k = 3 - i - j$.

```
void hanoi_move(struct hanoi *self, int n, int i, int j);
```

Question 11.9 Proposer un programme qui prend en paramètres le nombre n de disque et qui retourne une description de la solution.

```
$ hanoi 2

A: 2 1
B:
C:

A: 2
B:
C: 1

A:
B: 2
C: 1

A:
B: 2 1
C:
```

Travaux Pratiques d'Algorithmique n° 6

Exercice 12 : Génératon aléatoire d'arbres binaires

On considère la structure d'arbre suivante :

```
struct tree {
  struct tree *left;
  struct tree *right;
};
```

Question 12.1 Écrire une fonction qui crée un arbre à partir de deux arbres fils.

```
struct tree *tree_merge(struct tree *left, struct tree *right);
```

Question 12.2 Écrire une fonction qui génère un arbre aléatoire à n nœuds. Pour cela, on tirera un nombre q au hasard entre 0 et $n-1$ et on créera récursivement un sous-arbre gauche à q nœuds et un sous-arbre droit à $n-1-q$ nœuds.

```
struct tree *tree_generate(int n);
```

Question 12.3 Écrire une fonction qui calcule la hauteur d'un arbre binaire.

```
int tree_height(const struct tree *self);
```

Question 12.4 Écrire une fonction qui détruit un arbre.

```
void tree_destroy(struct tree *self);
```

Question 12.5 Quelle est la hauteur moyenne des arbres générées avec la méthode précédente ? Construire un histogramme des hauteurs obtenues sur la génération de 10000 arbres de taille 127.

Exercice 13 : Codage de Huffman

Le codage de Huffman est un algorithme de compression de données sans perte. Son implémentation repose sur la création d'un arbre binaire à partir d'un texte. Plus précisément, le principe de l'algorithme pour le codage de Huffman est le suivant :

1. À partir du texte, on fait une analyse statistique en comptant les occurrences de chaque caractère. On supposera que les caractères du texte sont uniquement des minuscules, sans accent, et l'espace.
2. Pour chaque caractère, on crée un nœud avec comme poids son nombre d'occurences. On trie l'ensemble des nœuds par ordre croissant de poids.
3. Tant qu'on a plus d'un nœud, on prend les deux nœuds de poids le plus faible et on crée un nouveau nœud avec comme fils les deux nœuds et comme poids la somme des poids des deux nœuds. On insère ce nouveau nœud dans la liste triée des nœuds.

4. Quand il n'y a plus qu'un seul nœud, on a un arbre binaire. Chaque caractère est alors codé par son chemin depuis la racine : si on va à gauche, on code avec un 0 et si on va à droite, on code avec un 1.

Ainsi, les caractères apparaissant le plus souvent auront un code binaire plus court que ceux apparaissant moins souvent.

Dans la suite, on donne des structures de données que vous pouvez utiliser. Vous devrez réfléchir aux fonctions à réaliser, en particulier leurs paramètres.

Question 13.1 Saisir un texte grâce à `fgets(3)` en le nettoyant du caractère de passage à la ligne.

Question 13.2 Compter le nombre d'occurrence de chaque caractères du texte.

```
#define CHARS_NUMBER 27

struct statistics {
  int count[CHARS_NUMBER];
};
```

Question 13.3 Construire les nœuds pour chaque caractère et insérer les nœuds dans une liste chaînée triée en fonction des occurrences, et en ne considérerant que les nœuds qui ont plus d'une occurrence. On réalise ainsi un tri par insertion sur une liste chaînée.

```
struct node {
  char c; // '#' for internal nodes
  int count;
  struct node *left;
  struct node *right;
};

struct list {
  struct node *data;
  struct list *next;
};
```

Question 13.4 Appliquer l'algorithme de Huffman décrit précédemment pour contruire l'arbre de Huffman.

Question 13.5 Pour chaque lettre, déterminer son codage en binaire. On parcourera l'arbre en profondeur en construisant le codage au fur et à mesure. Le codage sera stocké sous forme de chaîne de caractère avec des `'0'` et des `'1'`.

```
#define CODING_MAX 27

struct coding {
  char code[CHARS_NUMBER][CODING_MAX];
};
```

Question 13.6 Afficher le codage en binaire du texte initial. Combien de bits sont utilisés ? Combien d'octets fait le texte initial ? Quel est le taux de compression ?