Algorithmique et Structure de Données II

Arbres Binaires de Recherche

03 mai 2023

Version 1.3

Fabrice BOISSIER <fabrice.boissier@epita.fr>

Copyright

Table des matières

1 Consignes Générales

Les informations suivantes sont très importantes :

Le non-respect d'une des consignes suivantes entraînera des sanctions pouvant aller jusqu'à la multiplication de la note finale par 0.

Ces consignes sont claires, non-ambiguës, et ont un objectif précis. En outre, elles ne sont pas négociables.

N'hésitez pas à demander si vous ne comprenez pas une des règles.

Consigne Générale 0 : Vous devez lire le sujet.
Consigne Générale 1 : Vous devez respecter les consignes.
Consigne Générale 2 : Vous devez rendre le travail dans les délais prévus.

Consigne Générale 3 : Le travail doit être rendu dans le format décrit à la section Format de Rendu.

Consigne Générale 4 : Le travail rendu ne doit pas contenir de fichiers binaires, temporaires, ou d'erreurs (`*~`, `*.o`, `*.a`, `*.so`, `*#*`, `*core`, `*.log`, `*.exe`, binaires, ...).

Consigne Générale 5 : Dans l'ensemble de ce document, la casse (caractères majuscules et minuscules) est très importante. Vous devez strictement respecter les majuscules et minuscules imposées dans les messages et noms de fichiers du sujet.

Consigne Générale 6 : Dans l'ensemble de ce document, `login.x` correspond à votre login (donc `prenom.nom`).

Consigne Générale 7 : Dans l'ensemble de ce document, `nom1-nom2` correspond à la combinaison des deux noms de votre binôme (par exemple pour Fabrice BOISSIER et Mark ANGOUSTURES, cela donnera `boissier-angoustures`).

Consigne Générale 8 : Dans l'ensemble de ce document, le caractère ␣ correspond à une espace (s'il vous est demandé d'afficher ␣␣␣, vous devez afficher trois espaces consécutives).

Consigne Générale 9 : Tout retard, même d'une seconde, entraîne la note non négociable de 0.

Consigne Générale 10 : La triche (échange de code, copie de code ou de texte, ...) entraîne **au mieux** la note non négociable de 0.

Consigne Générale 11 : En cas de problème avec le projet, vous devez contacter le plus tôt possible les responsables du sujet aux adresses mail indiquées.

Conseil : N'attendez pas la dernière minute pour commencer à travailler sur le sujet.

Consigne Exceptionnelle : Tout code dans les headers (`.h`) sera sanctionné par un 0 (excepté les prototypes, constantes, et globales, si ceux-ci sont nécessaires).

2 Format de Rendu

Responsable(s) du projet :	**Fabrice BOISSIER** `<fabrice.boissier@epita.fr>`
Balise(s) du projet :	**`[ALG][BT]`**
Nombre d'étudiant(s) par rendu :	1
Procédure de rendu :	Devoir sur Moodle
Nom du répertoire :	`login.x-MiniProjet4`
Nom de l'archive :	`login.x-MiniProjet4.tar.bz2`
Date maximale de rendu :	03/05/2023 23h42
Durée du projet :	1 mois
Architecture/OS :	Linux - Ubuntu (x86_64)
Langage(s) :	C
Compilateur/Interpréteur :	**`/usr/bin/gcc`**
Options du compilateur/interpréteur :	**`-W -Wall -Werror -std=c99 -pedantic`**

Les fichiers suivants sont requis :

`AUTHORS`	contient le(s) nom(s) et prénom(s) de(s) auteur(s).
`Makefile`	le Makefile principal.
`README`	contient la description du projet et des exercices, ainsi que la façon d'utiliser le projet.
`configure`	le script shell de configuration pour l'environnement de compilation.

Un fichier **`Makefile`** doit être présent à la racine du dossier, et doit obligatoirement proposer ces règles :

`all`	*[Première règle]* lance la règle `libmybintree`.
`clean`	supprime tous les fichiers temporaires et ceux créés par le compilateur.
`dist`	crée une archive propre, valide, et répondant aux exigences de rendu.
`distclean`	lance la règle `clean`, puis supprime les binaires et bibliothèques.
`check`	lance le(s) script(s) de test.
`libmybintree`	lance les règles `shared` et `static`
`shared`	compile l'ensemble du projet avec les options de compilations exigées et génère une bibliothèque dynamique.
`static`	compile l'ensemble du projet avec les options de compilations exigées et génère une bibliothèque statique.

Votre code sera testé automatiquement, vous devez donc scrupuleusement respecter les spécifications pour pouvoir obtenir des points en validant les exercices. Votre code sera testé en générant un exécutable ou des bibliothèques avec les commandes suivantes :

```
./configure
make
```

Suite à cette étape de génération, les exécutables ou bibliothèques doivent être placés à ces endroits :

login.x-MiniProjet4/libmybintree.a
login.x-MiniProjet4/libmybintree.so

L'arborescence attendue pour le projet est la suivante :

login.x-MiniProjet4/
login.x-MiniProjet4/AUTHORS
login.x-MiniProjet4/README
login.x-MiniProjet4/Makefile
login.x-MiniProjet4/configure
login.x-MiniProjet4/check/
login.x-MiniProjet4/check/check.sh
login.x-MiniProjet4/src/
login.x-MiniProjet4/src/bt_basics.c
login.x-MiniProjet4/src/bt_basics.h
login.x-MiniProjet4/src/bt_bst.c
login.x-MiniProjet4/src/bt_bst.h

Vous ne serez jamais pénalisés pour la présence de makefiles ou de fichiers sources (code et/ou headers) dans les différents dossiers du projet tant que leur existence peut être justifiée (des makefiles vides ou jamais utilisés sont néanmoins pénalisés).

Les points de la notation sont distribués ainsi :

— Exercice 1 - Arbres Binaires : 5 points
— Exercice 2 - Bibliothèques statique et dynamique : 4 points
— Exercice 3 - Arbres Binaires de Recherche : 5 points
— Exercice 4 - Tests : 6 points

3 Aide Mémoire

Le travail doit être rendu au format ***.tar.bz2***, c'est-à-dire une archive **bz2** compressée avec un outil adapté (voir **`man 1 tar`** et **`man 1 bz2`**).
Tout autre format d'archive (zip, rar, 7zip, gz, gzip, ...) ne sera pas pris en compte, et votre travail ne sera pas corrigé (entraînant la note de 0).

Pour générer une archive *tar* en y mettant les dossiers *folder1* et *folder2*, vous devez taper :

```
tar cvf MyTarball.tar folder1 folder2
```

Pour générer une archive *tar* et la compresser avec GZip, vous devez taper :

```
tar cvzf MyTarball.tar.gz folder1 folder2
```

Pour générer une archive *tar* et la compresser avec BZip2, vous devez taper :

```
tar cvjf MyTarball.tar.bz2 folder1 folder2
```

Pour lister le contenu d'une archive *tar*, vous devez taper :

```
tar tf MyTarball.tar.bz2
```

Pour extraire le contenu d'une archive *tar*, vous devez taper :

```
tar xvf MyTarball.tar.bz2
```

Pour générer des exécutables avec les symboles de debug, vous devez utiliser les flags **`-g -ggdb`** avec le compilateur. N'oubliez pas d'appliquer ces flags sur *l'ensemble* des fichiers sources transformés en fichiers objets, et d'éventuellement utiliser les bibliothèques compilées en mode debug.

```
gcc -g -ggdb -c file1.c file2.c
```

Pour produire des exécutables avec les symboles de debug, il est conseillé de fournir un script **`configure`** prenant en paramètre une option permettant d'ajouter ces flags aux **`CFLAGS`** habituels.

```
./configure
cat Makefile.rules
CFLAGS=-W -Wall -Werror -std=c99 -pedantic
./configure debug
cat Makefile.rules
CFLAGS=-W -Wall -Werror -std=c99 -pedantic -g -ggdb
```

Pour produire une bibliothèque statique *libtest.a* à partir des fichiers *test1.c* et *file.c*, vous devez taper :

```
cc -c test1.c file.c
ar cr libtest.a test1.o file.o
```

Pour produire une bibliothèque dynamique *libtest.so* à partir des fichiers *test1.c* et *file.c*, vous devez taper (pensez aussi à **-fpic** ou **-fPIC**) :

```
cc -c test1.c file.c
cc test1.o file.o -shared -o libtest.so
```

Pour compiler un fichier en utilisant une bibliothèque dont le **.h** se trouve dans un dossier spécifique, par exemple la *libxml2*, vous devez taper (pensez à vous assurer que les *includes* de la bibliothèque ont été entourés de chevrons **< >**) :

```
cc -c -I/usr/include test1.c
```

Pour lier plusieurs fichiers objets ensemble et avec une bibliothèque, par exemple la *libxml2*, vous devez d'abord indiquer dans quel dossier trouver la bibliothèque avec l'option **-L**, puis indiquer quelle bibliothèque utiliser avec l'option **-l** (n'oubliez pas de retirer le préfixe *lib* au nom de la bibliothèque, et surtout, que l'ordre des fichiers objets et bibliothèques est important) :

```
cc -L/usr/lib test1.o -lxml2 file.o -o executable.exe
```

Si la bibliothèque existe en version *dynamique* (**.so**) et en version *statique* (**.a**) dans le système, l'éditeur de lien choisira en priorité la version *dynamique*. Pour forcer la version *statique*, vous devez l'indiquer dans la ligne de commande avec l'option **-static** :

```
cc -static -L/usr/lib test1.o -lxml2 file.o -o executable.exe
```

Dans ce sujet précis, vous ferez du code en C et des appels à des scripts shell qui afficheront les résultats dans le terminal (donc des flux de sortie qui pourront être redirigés vers un fichier texte).

4 Exercice 1 - Arbres Binaires

Nom du(es) fichier(s) :	**bt_basics.c**
Répertoire :	**login.x-MiniProjet4/src/**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions autorisées :	**malloc(3)**, **free(3)**, **printf(3)**

Objectif : Le but de l'exercice est d'implémenter les outils essentiels aux abres binaires.

Vous devez écrire plusieurs fonctions permettant de construire, parcourir, et vider des arbres binaires en ajoutant des nœuds. Un fichier **bt_basics.h** contenant toutes les fonctions exportables à implémenter vous est fourni en annexe. La structure **bt_p** est déjà déclarée dedans. Cette structure prend en charge la représentation des arbres sous forme de pointeurs grâce aux champs *lc* (pointeur vers le fils gauche) et *rc* (pointeur vers le fils droit). Chaque nœud dispose également d'un champ *key* contenant la clé identifiant l'élément stocké, d'un champ *elt* pointant vers l'élément stocké, et d'un champ *len_elt* représentant la taille de l'élément.

Vous aurez probablement besoin d'au moins une file. Vous pouvez ajouter deux fichiers supplémentaires **.c** et **.h** pour y implémenter votre file et ainsi appeler ces fonctions.

Vous devez implémenter les fonctions suivantes :

```
bt_p *create_bt_p(int key,
                  int elt_size,
                  void *elt,
                  bt_p *left_child,
                  bt_p *right_child);
int get_key_bt_p(bt_p *node);
int get_elt_size_bt_p(bt_p *node);
void *get_elt_bt_p(bt_p *node);

int deepness_node_bt_p(bt_p *node, bt_p *T);
int size_bt_p(bt_p *T);
int height_bt_p(bt_p *T);

void print_dfs_preorder_bt_p(bt_p *T);
void print_dfs_inorder_bt_p(bt_p *T);
void print_dfs_postorder_bt_p(bt_p *T);

void print_hierarchical_bt_p(bt_p *T);
int hierarchical_number_bt_p(bt_p *T, int key);

int clear_bt_p(bt_p *T);
```

`bt_p *create_bt_p(int key, int elt_size, void *elt, bt_p *left_child, bt_p *right_child)`

Cette fonction crée un nœud en remplissant ses différents champs ainsi qu'en indiquant les adresses de ses fils gauche et droit. En cas d'erreur (pas assez de mémoire), cette fonction doit renvoyer un pointeur **`NULL`**. *Vous n'avez pas à allouer d'espace pour stocker l'élément* ***`elt`*** *: c'est à l'utilisateur de votre bibliothèque de le faire.*

`int get_key_bt_p(bt_p *node)`

Cette fonction récupère le numéro de la clé contenue dans le nœud donné en paramètre. Si le nœud donné en paramètre est **`NULL`**, la fonction doit renvoyer -1.

`int get_elt_size_bt_p(bt_p *node)`

Cette fonction renvoie la taille de l'élément stocké dans le nœud donné en paramètre. Si le nœud donné en paramètre est **`NULL`**, la fonction doit renvoyer -1.

`void *get_elt_bt_p(bt_p *node)`

Cette fonction renvoie l'adresse de l'élément stocké dans le nœud donné en paramètre. Si le nœud donné en paramètre est **`NULL`**, la fonction doit renvoyer **`NULL`**.

`int deepness_node_bt_p(bt_p *node, bt_p *T)`

Cette fonction renvoie la profondeur du nœud donné en paramètre (c'est-à-dire le niveau du nœud). Si le nœud ou l'arbre donnés en paramètres sont **`NULL`**, la fonction doit renvoyer -1. Si le nœud est la racine, la fonction doit renvoyer 0.

`int size_bt_p(bt_p *T)`

Cette fonction renvoie la taille de l'arbre donné en paramètre (c'est-à-dire le nombre de nœuds qu'il contient). Si l'arbre donné en paramètre est **`NULL`**, la fonction doit renvoyer 0.

`int height_bt_p(bt_p *T)`

Cette fonction renvoie la hauteur de l'arbre donné en paramètre (c'est-à-dire le niveau du nœud le plus profond). Si l'arbre donné en paramètre ne contient qu'un seul élément (la racine), la fonction doit renvoyer 0. Si l'arbre donné en paramètre est **`NULL`**, la fonction doit renvoyer -1.

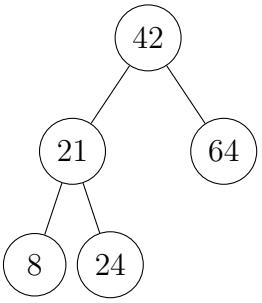
`void print_dfs_preorder_bt_p(bt_p *T)`

Cette procédure exécute un *parcours en profondeur* main gauche de l'arbre donné en paramètre, et affiche les clés dans l'ordre *préfixe*. Chaque clé doit être suivie d'un retour à la ligne. Si l'arbre donné en paramètre est **`NULL`**, la procédure ne fait rien.
Le format attendu est le suivant :

***`clé`*`\n`**

Ce qui donnerait cet affichage pour l'arbre suivant :

```
$ ./bt_example1
42
21
8
24
64
$
```



`void print_dfs_inorder_bt_p(bt_p *T)`

Cette procédure exécute un *parcours en profondeur* main gauche de l'arbre donné en paramètre, et affiche les clés dans l'ordre *infixe*. Chaque clé doit être suivie d'un retour à la ligne. Si l'arbre donné en paramètre est **`NULL`**, la procédure ne fait rien.

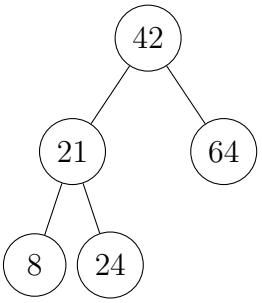
`void print_dfs_postorder_bt_p(bt_p *T)`

Cette procédure exécute un *parcours en profondeur* main gauche de l'arbre donné en paramètre, et affiche les clés dans l'ordre *suffixe*. Chaque clé doit être suivie d'un retour à la ligne. Si l'arbre donné en paramètre est **`NULL`**, la procédure ne fait rien.

`void print_hierarchical_bt_p(bt_p *T)`

Cette procédure exécute un *parcours en largeur* de l'arbre donné en paramètre, et affiche les clés au fur et à mesure du parcours. Chaque clé doit être suivie d'un retour à la ligne. Si l'arbre donné en paramètre est **`NULL`**, la procédure ne fait rien.
Ce qui donnerait cet affichage pour l'arbre suivant :

```
$ ./bt_example2
42
21
64
8
24
$
```



`int hierarchical_number_bt_p(bt_p *T, int key)`

Cette fonction récupère le numéro hiérarchique du nœud contenant la clé donnée en paramètre, puis elle le retourne. La racine a comme numéro hiérarchique 1. Si l'arbre donné en paramètre est **`NULL`**, ou que la clé n'existe pas, la fonction doit renvoyer -1.

`int clear_bt_p(bt_p *T)`

Cette fonction vide l'arbre binaire donné en paramètre en libérant chacun de ses nœuds, puis elle renvoie le nombre de nœuds libérés. Si l'arbre donné en paramètre est **`NULL`**, la fonction doit renvoyer 0. *Attention, vous ne devez pas libérer les pointeurs des éléments stockés dans les nœuds : c'est à l'utilisateur de votre bibliothèque de s'en charger.*

5 Exercice 2 - Bibliothèques statique et dynamique

Nom du(es) fichier(s) :	**`Makefile`**
Répertoire :	**`login.x-MiniProjet4/`**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Outils recommandés :	**`gcc(1)`**, **`ar(1)`**

Objectif : Le but de l'exercice est de faire fonctionner l'ensemble de votre projet avec un makefile simple, et de produire une bibliothèque statique ainsi qu'une dynamique.

Une bibliothèque est un ensemble de fonctions et procédures prêtes à être utilisées (ayant éventuellement des variables statiques, ou encore faisant appel à `malloc(3)`) par un utilisateur ne connaissant pas leur implémentation. L'utilisateur développe et écrit son propre programme (utilisant un `main` pour s'exécuter) et il peut éventuellement s'appuyer sur des bibliothèques (n'ayant pas de `main`) pour réutiliser des implémentations existantes.

Les exercices suivants vont vous conduire à produire des fonctions manipulant des structures, afin d'offrir un service à des utilisateurs : des arbres binaires et leur interface pour les exploiter (une `API`). Vous devez donc écrire le (ou les) *makefile(s)* de votre projet et un fichier `configure` (éventuellement vide) afin de produire une bibliothèque statique nommée **`libmybintree.a`** et une bibliothèque dynamique nommée **`libmybintree.so`**.

Le *makefile* que vous allez écrire rendra votre projet complet et autonome. Pour cela, vous devez faire en sorte que plusieurs *cibles* soient présentent dans le makefile (celles-ci sont rappelées dans le paragraphe suivant).

Plusieurs stratégies existent pour compiler avec des makefiles, l'une d'entre elle consiste à placer un makefile par dossier afin que le makefile principal appelle les suivants avec la bonne règle (par exemple : le makefile principal va appeler le makefile du dossier *src* pour compiler le projet, mais le makefile principal appellera celui du dossier *check* lorsque l'on demandera d'exécuter la suite de tests).
Pour ce premier contact avec les makefiles, vous pouvez vous contenter d'un seul makefile à la racine exécutant des lignes de compilation toutes prêtes sans aucune réécriture ou extension.

Afin de générer des bibliothèques statiques et dynamiques, vous devez compiler vos fichiers pour produire des fichiers *objets* (des fichiers **`.o`**), mais vous ne devez pas laisser l'étape d'*édition de liens* classique se faire. À la place, la cible *static* de votre makefile devra produire une bibliothèque statique nommée **`libmybintree.a`**, et la cible *shared* devra produire une bibliothèque dynamique nommée **`libmybintree.so`**.

Pour générer une bibliothèque statique (*static library* en anglais), on utilise la commande **`ar`** qui créée des *archives*. Pour produire la bibliothèque statique *libtest.a* à partir des fichiers *test1.c* et *file.c*, on utilisera ces commandes :

```
cc -c test1.c file.c
ar cr libtest.a test1.o file.o
```

Pour générer une bibliothèque dynamique (*shared library* en anglais), on utilise l'option **-shared** du compilateur. Pour produire la bibliothèque dynamique *libtest.so* à partir des fichiers *test1.c* et *file.c*, on utilisera ces commandes :

```
cc -c test1.c file.c
cc test1.o file.o -shared -o libtest.so
```

La première commande génère des fichiers objets, et la deuxième les réunit dans un seul fichier. Il arrivera sur certains systèmes qu'il soit nécessaire d'ajouter l'option **-fpic** ou **-fPIC** (*Position Independent Code*) pour générer des bibliothèques dynamiques. Gardez en tête que *toutes* les bibliothèques **doivent** être préfixées par un *lib* dans le nom des fichiers **.a** et **.so** générés. Mais, lorsque l'on utiliser une bibliothèque dans un projet, on doit ignorer ce préfixe lors de la compilation et l'édition de liens.

Pour utiliser une bibliothèque dynamique sur votre système, plusieurs méthodes existent selon le système où vous vous trouvez.

— L'une d'entre elle consiste à ajouter le dossier contenant la bibliothèque à la variable d'environnement **LD_LIBRARY_PATH** (comme pour la variable **PATH**). Selon votre système, il peut s'agir de la variable d'environnement **DYLD_LIBRARY_PATH**, **LIBPATH**, ou encore **SHLIB_PATH**. Pour ajouter le dossier courant comme dossier contenant des bibliothèques dynamiques en plus d'autres dossiers, vous pouvez taper :
export LD_LIBRARY_PATH=.:/usr/lib:/usr/local/lib

— Une autre méthode consiste à modifier le fichier **/etc/ls.so.conf** et/ou ajouter un fichier contenant le chemin vers votre bibliothèque dans **/etc/ld.so.conf.d/** puis à exécuter **/sbin/ldconfig** pour recharger les dossiers à utiliser.

— Enfin, vous pouvez demander les droits administrateur et installer votre bibliothèque dans **/usr/lib** ou **/usr/local/lib**.

Évidemment, à long terme, l'objectif est d'avoir une bibliothèque installée dans le répertoire **/usr/lib**. Mais, lors du développement, il est préférable d'utiliser la variable **LD_LIBRARY_PATH**. Pour vous assurer du fonctionnement de votre exécutable, vous pouvez utiliser **ldd(1)** avec le chemin complet vers un exécutable. **ldd** vous indiquera quelles bibliothèques sont requises, et où celui-ci les trouve. Si l'une d'entre elle n'est pas trouvée, **ldd** vous en informera :

```
ldd /usr/bin/ls
ldd my_main
```

Pour rappel voici les cibles demandées pour le **Makefile** principal à la racine de votre projet :

`all`	*[Première règle]* lance la règle `libmybintree`.
`clean`	supprime tous les fichiers temporaires et ceux créés par le compilateur.
`dist`	crée une archive propre, valide, et répondant aux exigences de rendu.
`distclean`	lance la règle `clean`, puis supprime les binaires et bibliothèques.
`check`	lance le(s) script(s) de test.
`libmybintree`	lance les règles `shared` et `static`
`shared`	compile l'ensemble du projet avec les options de compilations exigées et génère une bibliothèque dynamique.
`static`	compile l'ensemble du projet avec les options de compilations exigées et génère une bibliothèque statique.

6 Exercice 3 - Arbres Binaires de Recherche

Nom du(es) fichier(s) :	**bt_bst.c**
Répertoire :	**login.x-MiniProjet4/src/**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions autorisées :	**malloc(3)**, **free(3)**, **printf(3)**

Objectif : Le but de l'exercice est d'implémenter une mini bibliothèque d'arbres binaires de recherche (ABR).

Vous devez écrire plusieurs fonctions permettant d'effectuer des recherche, insertions, et suppressions de nœuds dans des ABR. Un fichier **bt_bst.h** contenant toutes les fonctions exportables à implémenter vous est fourni en annexe. Celui-ci s'appuie sur le fichier **bt_basics.h** de l'exercice 1 : vous devez avoir réalisé toutes les fonctions du premier exercice pour pouvoir faire celui-ci.

Vous devez implémenter les fonctions suivantes :

```
bt_p *search_bt_p(int key, bt_p *T);
bt_p *search_verbose_bt_p(int key, bt_p *T);

bt_p *insert_leaf_bt_p(int key,
                       int len_elt,
                       void *elt,
                       bt_p *T);

bt_p *remove_node_bt_p(int key, bt_p *T);

bt_t *convert_bt_p_to_tab(bt_p *T);
void print_bt_t(bt_t *T_tab);

bt_p *insert_root_cut_bt_p(int key,        // BONUS
                           int len_elt,
                           void *elt,
                           bt_p *T);

bt_p *insert_root_rot_bt_p(int key,        // BONUS
                           int len_elt,
                           void *elt,
                           bt_p *T);
```

bt_p *search_bt_p(int key, bt_p *T)

Cette fonction recherche un nœud à partir de sa clé dans un arbre binaire en appliquant les contraintes des ABR. Si la clé est trouvée, on renvoie un pointeur vers le nœud contenant la clé. Si la clé n'est pas trouvée, ou que le l'arbre binaire est **NULL**, alors la fonction doit renvoyer un pointeur **NULL**.

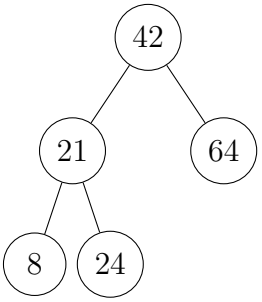
bt_p *search_verbose_bt_p(int key, bt_p *T)

Cette fonction recherche un nœud à partir de sa clé dans un arbre binaire en appliquant les contraintes des ABR et en affichant la clé de chaque nœud traversé. Lors du parcours, chaque nœud testé verra sa clé affichée et suivie d'un retour à la ligne. Si le nœud testé n'existe pas, il n'affichera rien (ni retour à la ligne, ni erreur). Si la clé est trouvée, on renvoie un pointeur vers le nœud contenant la clé. Si la clé n'est pas trouvée, ou que le l'arbre binaire est **NULL**, alors la fonction doit renvoyer un pointeur **NULL**.
Le format attendu est le suivant :

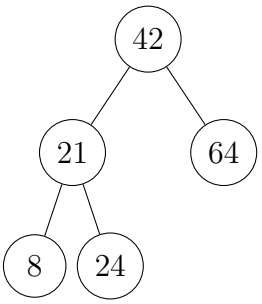
***clé*\n**

Ce qui donnerait ces affichages pour l'arbre suivant :

```
$ ./bt_example3 24
42
21
24
$
```




```
$ ./bt_example3 96
42
64
$
```



bt_p *insert_leaf_bt_p(int key, int len_elt, void *elt, bt_p *T)

Cette fonction insère un nouvel élément et sa clé dans l'arbre binaire donné en paramètre. L'insertion doit être faite en feuille en respectant les contraintes des ABR. Si l'arbre binaire en paramètre est vide, vous devez le créer et lui allouer un premier nœud qui servira de racine. Si la clé donnée en paramètre est inférieure ou égale à 0, alors la fonction ne fera rien et renverra l'arbre inchangé.

`bt_p *remove_node_bt_p(int key, bt_p *T)`

Cette fonction supprime un nœud de l'arbre binaire donné en paramètre à partir de la clé également donnée en paramètre. La suppression doit être faite en respectant les contraintes des ABR. Pour l'implémentation de cette suppression, dans le cas où le nœud visé a 2 fils, vous devrez utiliser le plus proche voisin *plus petit* pour le remplacer (donc aller chercher le plus grand nœud dans le sous-arbre gauche).
Vous devez libérer correctement la mémoire en libérant le nœud, mais sans libérer l'élément stocké dans le nœud (c'est à l'utilisateur de s'en charger). La fonction renvoie un pointeur vers la racine de l'arbre, mais si l'arbre est vide ou le devient, la fonction renvoie **`NULL`**.

`bt_t *convert_bt_p_to_tab(bt_p *T)`

Cette fonction transforme un arbre binaire au format pointeur vers le format tableau. La structure **`bt_t`** renvoyée par la fonction contiendra :

- **`int tab_len`** : un champs stockant la taille des tableaux (la taille étant obligatoirement au format $2^n - 1$ où n est le nombre de niveaux)
- **`int tab_used`** : un champs stockant le nombre de cases utilisées dans chaque tableau (c'est-à-dire le nombre de nœuds dans l'arbre)
- **`int *keys`** : un tableau d'entiers contenant les clés des nœuds
- **`int *elts_size`** : un tableau d'entiers contenant la taille des éléments stockés
- **`void **elts`** : un tableau de **`void*`** contenant les adresses des éléments stockés (c'est-à-dire les pointeurs vers les éléments)

Les cases représentant les nœuds vides doivent contenir -1. Si l'arbre donné en paramètre est **`NULL`**, la fonction doit renvoyer **`NULL`** sans rien allouer.

`void print_bt_t(bt_t *T_tab)`

Cette fonction affiche les clés de l'arbre binaire représenté sous forme de tableau. Le format d'affichage est le même que pour l'affichage hiérarchique de l'exercice 1 (donc une clé par ligne, et aucun affichage si l'arbre est vide).

[BONUS] **`bt_p *insert_root_cut_bt_p(int key, int len_elt, void *elt, bt_p *T)`**

Cette fonction insère un nouvel élément et sa clé dans l'arbre binaire donné en paramètre. L'insertion doit être faite en racine avec la méthode de coupe. Si l'arbre binaire en paramètre est vide, vous devez le créer et lui allouer un premier nœud qui servira de racine. Si la clé donnée en paramètre est inférieure ou égale à 0, alors la fonction ne fera rien et renverra l'arbre inchangé.

[BONUS] **`bt_p *insert_root_rot_bt_p(int key, int len_elt, void *elt, bt_p *T)`**

Cette fonction insère un nouvel élément et sa clé dans l'arbre binaire donné en paramètre. L'insertion doit être faite en racine avec la méthode des rotations. Si l'arbre binaire en paramètre est vide, vous devez le créer et lui allouer un premier nœud qui servira de racine. Si la clé donnée en paramètre est inférieure ou égale à 0, alors la fonction ne fera rien et renverra l'arbre inchangé.

7 Exercice 4 - Tests

Nom du(es) fichier(s) :	**`check.sh`**, **`tests.c`**, *[**`test_XYZ.c`**]*
Répertoire :	**`login.x-MiniProjet4/check/`**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions autorisées :	*(toutes)*

Objectif : Le but de l'exercice est de fournir un programme et un script qui testeront votre bibliothèque d'arbres.

Vous devez écrire une suite de tests vérifiant le bon fonctionnement de votre bibliothèque **`libmybintree`**. L'appel à la commande `make` à la racine du projet avec la cible *check* doit compiler un (ou des) fichiers C se trouvant dans le dossier **`check`** afin qu'ils utilisent la bibliothèque que vous avez écrite préalablement, puis le script **`check.sh`** doit lancer les tests.

Attention : la cible **`check`** doit avoir comme dépendance la cible **`libmybintree`** afin que les bibliothèques soient générées *avant* de compiler et lier les tests. Inclure les fichiers C des exercices précédents ne donnera pas tous les points/limitera la note de cet exercice : seuls les fichiers **`bt_basics.h`** et **`bt_bst.h`** doivent être utilisés dans un **`include`** pour cet exercice, mais pas les **`.c`** !

Vous devez donc écrire plusieurs scénarios de test qui construisent des arbres, les parcourent, et suppriment des nœuds (en re-faisant le parcours par la suite). Comme vous l'aurez constaté, le format de sortie texte est très précis pour justement permettre des comparaisons simples et aisées.
Une façon de rédiger les tests consiste à préparer un scénario sur papier en amont, d'écrire le code C réalisant ce scénario et écrivant en sortie standard à chaque étape l'état de l'arbre, puis, d'écrire dans un script shell les sorties attendues et les comparer.

Vous pouvez tout à fait écrire plusieurs scénarios dans plusieurs fichiers C et sh distincts. Il faut néanmoins que le script **`check.sh`** existe et lance les tests.

Les points attribués dans cet exercice dépendront de l'exhaustivité des tests, mais également de la façon de tester : comme indiqué plus haut, si vous n'utilisez pas l'une des bibliothèques résultantes, mais directement les sources de votre projet (**`bt_basics.c`**, **`bt_bst.c`**, et vos éventuels autres fichiers sources, ou vous copiez/collez le code, ou d'autres techniques encore...), alors vous n'aurez pas le maximum de points. Les tests doivent être exhaustifs pour ne rater aucun cas parmi ceux spécifiés dans ce sujet.