Algorithmique et Structures de Données - Avancé

Plan du cours

- Intro : fonctionnement du cours et projets ;
- Arbres :
  - Types ;
  - Application au stockage et à la recherche rapide.
- Graphes ;
  - Modèles et implémentations ;
  - Plus court chemin.
- Recherche de patterns (KMP) ;
- Algorithmes d'Encodage / de Compression de données.

Références

- **Algorithms in C** — par Robert Sedgewick chez Addison-Wesley
- **Algorithms in Java** — par Robert Sedgewick chez Addison-Wesley (en deux parties 1-4 et 5)
- **C Programming Language** — par Brian W. Kernighan et Dennis M. Ritchie chez Prentice Hall

```c
#define MAX 100
extern void initFile(void);
extern void enfiler(int v);
extern int defiler(void);
extern int vide(void);
```

```c
#include "file.h"
static int file[MAX + 1], debut = 0, fin = 0;
extern void initFile(void) {
  debut = fin = 0;
}
extern void enfiler(int v) {
  file[fin++] = v;
  if(fin > MAX) fin = 0;
}
extern int defiler(void) {
  int v = file[debut++];
  if(debut > MAX) debut = 0;
  return v;
}
extern int vide(void) {
  return debut == fin;
}
```

FIGURE 1 – **(Haut)** `file.h` **(bas)** `file.c`

```c
#include "file.h"
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
typedef struct node_t node_t;
struct node_t {
  int v;
  struct node_t * fg, * fd;
};
void addNode(node_t ** ptr, int v) {
  while(*ptr) {
    if(v > (*ptr)->v)
      ptr = &((*ptr)->fd);
    else
      ptr = &((*ptr)->fg);
  }
  *ptr = malloc(sizeof ** ptr);
  assert(*ptr);
  (*ptr)->v = v;
  (*ptr)->fg = NULL;
  (*ptr)->fd = NULL;
}
void printTree(node_t * ptr) {
  if(ptr) {
    printTree(ptr->fg);
    printf("%d ", ptr->v);
    printTree(ptr->fd);
  }
}
void enfilerTree(node_t * ptr) {
  if(ptr) {
    enfilerTree(ptr->fg);
    enfiler(ptr->v);
    enfilerTree(ptr->fd);
  }
}
int main(void) {
  node_t * tree = NULL;
  addNode(&tree, 5); addNode(&tree, 3); addNode(&tree, 4); addNode(&tree, 2);
  addNode(&tree, 1); addNode(&tree, 9); addNode(&tree, 7); addNode(&tree, 6);
  addNode(&tree, 8); addNode(&tree, 11); addNode(&tree, 12); addNode(&tree, 10);
  printTree(tree);
  printf("\n");
  enfilerTree(tree);
  while(!vide())
    printf("%d ", defiler());
  printf("\n");
  return 0;
}
```

FIGURE 2 – Arbre binaire (`main.c`)

```c
#include "binTree.h"
#include <sys/types.h>
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
static int recuDir(bnode_t ** pbt, char * path, char * name) {
  DIR * dir;
  char buf[BUFSIZ];
  struct dirent * dirent;
  if((dir = opendir(path)) == NULL) return 0;
  while((dirent = readdir(dir))) {
    if(!strcmp(dirent->d_name, ".") || !strcmp(dirent->d_name, "..")) continue;
    sprintf(buf, "%s/%s", path, dirent->d_name);
    addNode(pbt, dirent->d_name, path);
    recuDir(pbt, buf, dirent->d_name);
  }
  closedir(dir);
  return 1;
}
int main(int argc, char ** argv) {
  bnode_t * bt = NULL;
  if(argc != 3) {
    fprintf(stderr, "usage : %s path expression\n", argv[0]);
    exit(1);
  }
  recuDir(&bt, argv[1], ".");
  printPath(bt, argv[2]);
  return 0;
}
```

```
SHELL = /bin/sh
#definition des commandes utilisees
CC = gcc
ECHO = echo
RM = rm -f
TAR = gtar
MKDIR = mkdir
CHMOD = chmod
CP = cp
#declaration des options pour gcc
PG_FLAGS = -pg
CFLAGS = -Wall -O3 $(PG_FLAGS)
CPPFLAGS = -I.
LDFLAGS = $(PG_FLAGS)
#definition des fichiers et dossiers
PROGNAME = miniFind
PACKAGE=$(PROGNAME)
VERSION = 0.1
distdir = $(PACKAGE)-$(VERSION)
SOURCES = repArbo.c binTree.c
HEADERS = binTree.h
OBJ = $(SOURCES:.c=.o)
DISTFILES = $(SOURCES) Makefile $(HEADERS)

all: $(PROGNAME)

$(PROGNAME): $(OBJ)
        $(CC) $(OBJ) $(LDFLAGS) -o $(PROGNAME)

%.o: %.c
        $(CC) $(CPPFLAGS) $(CFLAGS) -c $<

dist: distdir
        $(CHMOD) -R a+r $(distdir)
        $(TAR) zcvf $(distdir).tgz $(distdir)
        $(RM) -r $(distdir)
distdir: $(DISTFILES)
        $(RM) -r $(distdir)
        $(MKDIR) $(distdir)
        $(CHMOD) 777 $(distdir)
        $(CP) -rf $(DISTFILES) $(distdir)
clean:
        $(RM) $(PROGNAME) $(OBJ) *~ $(distdir).tgz gmon.out
```

FIGURE 3 – **(Haut)** `miniFind/repArbo.c` **(bas)** `miniFind/Makefile`

```c
typedef struct bnode_t bnode_t;
struct bnode_t {
  char * name, * path;
  struct bnode_t * fg, * fd;
};
extern int       addNode   (bnode_t ** pbt, char * name, char * path);
extern bnode_t * findNode  (bnode_t *   bt, char * name);
extern void      printPath (bnode_t *   bt, char * name);
```

```c
#include "binTree.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
extern int addNode(bnode_t ** pbt, char * name, char * path) {
  int cmp;
  bnode_t ** ptr = pbt;
  while(*ptr) {
    if((cmp = strcmp((*ptr)->name, name)) > 0)
      ptr = &((*ptr)->fg);
    else if(cmp < 0)
      ptr = &((*ptr)->fd);
    else {
      if((cmp = strcmp((*ptr)->path, path)) > 0)
        ptr = &((*ptr)->fg);
      else if(cmp < 0)
        ptr = &((*ptr)->fd);
      else
        return 0;
    }
  }
  *ptr = malloc(sizeof * *ptr);
  assert(*ptr);
  (*ptr)->name = strdup(name);
  (*ptr)->path = strdup(path);
  (*ptr)->fg = (*ptr)->fd = NULL;
  return 1;
}
extern bnode_t * findNode(bnode_t * bt, char * name) {
  int cmp;
  if(bt == NULL) return NULL;
  if((cmp = strcmp(bt->name, name)) > 0)
    return findNode(bt->fg, name);
  else if(cmp < 0)
    return findNode(bt->fd, name);
  return bt;
}
extern void printPath(bnode_t * bt, char * name) {
  if((bt = findNode(bt, name)) == NULL) return;
  printf("%s/%s\n", bt->path, bt->name);
  fflush(stdout);
  printPath(bt->fg, name);
  printPath(bt->fd, name);
}
```

FIGURE 4 – **(Haut)** `miniFind/binTree.h` **(Bas)** `miniFind/binTree.c`