Algorithmie Avancée Projet Licence 3 Informatique

Adrien Leroy-Lechat

05/12/2018

Première partie

Construction du Graphe

0.1 Structures Utilisées

0.1.1 Graphe_t

nb_noeud : contient le nombre de noeuds présents dans le graphe

vec_sommets : contient un pointeur sur le vecteur de noeuds

0.1.2 Noeud_t

line : contient le numéro de ligne de la station

nom_station : contient le nom de la station

nb_voisins : contient le nombre de stations connexes à la station

tab_indice_voisins : contient un vecteur d'indices de ces stations connexes

vu : contient la valeur (0,1) pour pouvoir identifier au moment du Monté-Carlo si le noeud a déjà été parcouru

0.1.3 Graphe_l_t

nb_noeud : contient le nombre de noeuds présents dans le graphe

vec_sommets : contient un pointeur sur le vecteur de noeuds

0.1.4 Node_t

line : contient le numéro de ligne de la station

nom_station : contient le nom de la station

nb_voisins : contient le nombre de stations connexes à la station

voisins : contient un pointeur sur une Structure contenant le premier élément de la liste

vu : contient la valeur (0,1) pour pouvoir identifier au moment du Monté-Carlo si le noeud a déjà été parcouru

0.1.5 Save_t

first : contient le pointeur sur le premier maillon de la liste

0.1.6 List_t

indice_voisin : contient l'indice de la station connexe

next : contient un pointeur vers le maillon suivant

Les deux construction de graphe sont à peut pres equivalantes en espace memoire utilisé. Mais le graphe de liste est legerement plus volumineux du fait du stokage des pointeurs suplementaires. Cette augmentation est compensé par la modulabilitée de ce dit espace memoire.

0.2 La récuperation des données

J'ai choisis d'utiliser un fichier format csv (metro.csv) de type
1 ;Chatelet ;9 ;299 ;21 ;77 ;79 ;152 ;35 ;21 ;166 ;23

respectivement : ligne ;nom_station ;nb_voisins ;voisin1 ;voisin2

Le nombre de stations est récupérées est présent dans un fichier nbStation.txt

Avec ces données les fonctions get_nodes et get_list remplissent le graphe.

Deuxième partie

Recherche du plus cour chemin

0.3 L'algorithme

```
tant que rollout < nb_rollout:
  tant que sation != station_final || noeuds connexes déjà exploré
    sélection d'une nouvelle station
    enregistrement de la nouvelle station dans la liste temporaire
    si station == station_final && nombre d'arrets de la liste temporaire <
      enregistrement de la liste temporaire dans la liste meilleure solutio
  mise à zero de liste temporaire
  rollout++
affichage du chemin le plus cour
```

0.4 Le code

J'ai baser la recherche du plus court chemin sur la fonction get_way et get_list. Ces fonctions sont identiques exepter dans leurs arguments ou l'une manipule un graphe_t et l'autre un graphe_l_t.
De même pour print_way et print_way_list.

Les grands changements sont surtout operés dans la manipulation des tab_indice_voisins et voisins dans les structure noeud_t et node_t.

Les fonctions test_node et test_node_list correspondent à 'tant que sation $= station\_final || noeuds connexes déjà explorés'$.

Les fonctions new_station et new_station_list correspondent à 'sélection d'une nouvelle station'.

0.5 Commentaires / Améliorations

Il est possible, du fait que l'algorithme soit entièrement basé sur le graphe construit en debut d'execution lui même basé sur le fichier metro.csv, que certaines erreurs presentent dans ce fichier faussent la qualité des resultats et ne representent pas la realité.

On observe en moyenne que la lecture du graphe à vecteur de successeurs par la fonction get_way est plus rapide que la lecture du graphe à liste de successeurs par la fonction get_list. Il est vrai aussi que l'ecriture de cette partie du code ma pauser plus de soucis que la simple manipulation de vecteur.

Les ameliorations à apporter à ce programme peuvent êtres multiples.

L'utilisation de MC pourait etre remplacé par un MCTS plus efficace dans la recherche du plus court chemin.

Une meilleur utilisation du numeros de ligne des stations, totalement mis de côter dans cette implementation du programme.

Automatiser la création du fichier metro.csv pour limité les erreurs humaine.

Troisième partie

Le code source

0.6 Header

```
#ifndef PROJET_H
#define PROJET_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>

#define BUFF_SYZE 1024
#define DELIM ";"
#define DEBUG 0      //0 pas d'affichage //1 affichage find.c //2 affichage main.c
#define ROLLOUT 100000


typedef struct list_sommets{
  int indice_voisin;
  struct list_sommets *next;
}        list_t;

typedef struct list_save{
  list_t *first;
}        save_t;

typedef struct list_node{
  int line;
  char *nom_station;
  int nb_voisins;
  save_t *voisins;
  int vu;
}        node_t;

typedef struct noeud{
  int line;
  char *nom_station;
  int nb_voisins;
```

```
  int *tab_indice_voisins; //recuperable dans vec_sommets
  int vu;
}        noeud_t;

typedef struct graphe_list{
  int nb_noeud;
  node_t **vec_sommets;
}        graphe_l_t;

typedef struct graphe{
  int nb_noeud;
  noeud_t **vec_sommets;
}        graphe_t;

typedef struct chemin{
  int nb_arret;
  int *way;
}        chemin_t;

//fonctions de find.c
int new_station(graphe_t *metro, int indice_station);
int test_node(graphe_t *metro, int indice_station, int indice_end);
void get_way(graphe_t * metro, int indice_start, int indice_end);
void print_way(chemin_t *best_way, graphe_t *metro);

//fonctions de find_list.c
int new_station_list(graphe_l_t *metro, int indice_station);
int test_node_list(graphe_l_t *metro, int indice_station, int indice_end);
void get_way_list(graphe_l_t *metro, int indice_start, int indice_end);

//Fonctions de recup.c
int recup_indice(char station[30], graphe_t *metro);
int test_station(char station_start[30], char station_end[30], graphe_t *me

//fonctions de main.c
noeud_t **get_nodes(char *filename, graphe_t *metro);
node_t **get_list(char *filename, graphe_t *metro);
void clear_buf();
```

```
void free_graphe(graphe_t *metro);


//je considere que tt les statio non un poid de 0 (assez representatif du metro par

#endif
```

0.7 Main.c

```
#include "Includes/projet.h"

noeud_t **get_nodes(char *filename, graphe_t *metro){
FILE *file;
noeud_t *station = NULL;
noeud_t **vec_sommets = NULL;
int tmp_line;
int tmp_voisins;
int *tmp_tab_indice_voisin;

char *token = NULL;
char buff[BUFF_SYZE];
int i, j, k, alloc_voisins, conteur_voisins;
char *ptr;



k = 0;
alloc_voisins = 0;
file = fopen(filename, "r");
vec_sommets = (noeud_t **)malloc(sizeof(noeud_t *) * metro->nb_noeud);
if (file != NULL){
while ((fgets(buff, BUFF_SYZE, file)) != NULL){
ptr = buff;
j = 0;
i = 0;
conteur_voisins = 0;
```

```
station = calloc(1, sizeof(*station));
if (station != NULL){
while ((token = strtok(ptr, DELIM)) != NULL){
if (j == 0)
ptr = NULL;
if (i == 0){
tmp_line = atoi(token);
station->line = tmp_line;
}
if (i == 1)
station->nom_station = strdup(token);
if (i == 2){
station->nb_voisins = atoi(token);
}
if (station->nb_voisins == 0 && alloc_voisins == 0){
tmp_tab_indice_voisin = malloc(8 + sizeof(int) * station->nb_voisins);
alloc_voisins = 1;
}
if (i > 2){
tmp_voisins = atoi(token);
tmp_tab_indice_voisin[conteur_voisins] = tmp_voisins;
conteur_voisins++;
}
i++;
}
station->tab_indice_voisins = tmp_tab_indice_voisin;
}
vec_sommets[k] = malloc(sizeof(noeud_t));
vec_sommets[k]->line = station->line;
vec_sommets[k]->nom_station = strdup(station->nom_station);
vec_sommets[k]->nb_voisins = station->nb_voisins;
vec_sommets[k]->tab_indice_voisins = malloc(sizeof(int) * vec_sommets[k]->n
for (int o = 0; o < vec_sommets[k]->nb_voisins; o++)
vec_sommets[k]->tab_indice_voisins[o] = station->tab_indice_voisins[o];
vec_sommets[k]->vu = 0;
k++;
}
free(tmp_tab_indice_voisin);
```

```
}
return vec_sommets;
}

void add_list(save_t *list, int indice_voisin){
list_t *new;

new = malloc(sizeof(list_t*));
new->indice_voisin = indice_voisin;
new->next = list->first;
list->first = new;
}


node_t **get_list(char *filename, graphe_t *metro){
FILE *file;
node_t *station = NULL;
node_t **vec_sommets = NULL;
int tmp_line;
int tmp_voisins;
int *tmp_tab_indice_voisin;

char *token = NULL;
char buff[BUFF_SYZE];
int i, j, k, y, alloc_voisins, conteur_voisins;
char *ptr;
list_t *tmp_list_voisin;



k = 0;
alloc_voisins = 0;
file = fopen(filename, "r");
vec_sommets = (node_t **)malloc(sizeof(node_t *) * metro->nb_noeud);
if (file != NULL){
while ((fgets(buff, BUFF_SYZE, file)) != NULL){
ptr = buff;
j = 0;
```

```
i = 0;
conteur_voisins = 0;

station = calloc(1, sizeof(*station));
if (station != NULL){
while ((token = strtok(ptr, DELIM)) != NULL){
if (j == 0)
ptr = NULL;
if (i == 0){
tmp_line = atoi(token);
station->line = tmp_line;
}
if (i == 1)
station->nom_station = strdup(token);
if (i == 2){
station->nb_voisins = atoi(token);
}
if (station->nb_voisins == 0 && alloc_voisins == 0){
tmp_tab_indice_voisin = malloc(8 + sizeof(int *) * station->nb_voisins);
alloc_voisins = 1;
}
if (i > 2){
tmp_voisins = atoi(token);
tmp_tab_indice_voisin[conteur_voisins] = tmp_voisins;
conteur_voisins++;
}
i++;
}
tmp_list_voisin = malloc(sizeof(tmp_list_voisin));
station->voisins = malloc(sizeof(save_t *));
tmp_list_voisin->indice_voisin = tmp_tab_indice_voisin[0];
tmp_list_voisin->next = NULL;
station->voisins->first = tmp_list_voisin;
for (y = 1; y < station->nb_voisins; y++){
add_list(station->voisins, tmp_tab_indice_voisin[y]);
}
}
vec_sommets[k] = malloc(sizeof(noeud_t));
```

```
vec_sommets[k]->line = station->line;
vec_sommets[k]->nom_station = strdup(station->nom_station);
vec_sommets[k]->nb_voisins = station->nb_voisins;
vec_sommets[k]->voisins = station->voisins;
vec_sommets[k]->vu = 0;
k++;
}
free(tmp_tab_indice_voisin);
}
return vec_sommets;
}

void clear_buf(){
int c = 0;

while(c != '\n' && c !=  EOF)
c = getchar();
}

void free_graphe(graphe_t *metro){
int i = 0;
while (i < metro->nb_noeud && metro->vec_sommets != NULL){
free(metro->vec_sommets[i]->tab_indice_voisins);
i++;
}
free(metro->vec_sommets);
}


int main(void){
FILE *f;
char buf[4];
graphe_t *metro;
graphe_l_t *metre;
int nb_station;
int d = 0;
int indice_start;
int indice_end;
```

```
char station_start[30];
char station_end[30];
int fin = 0;
int valide = 0;
double debut_vec = 0;
double fin_vec = 0;
double result_vec = 0;
double debut_list = 0;
double fin_list = 0;
double result_list = 0;


srand(time(NULL));

f = fopen("nbStation.txt", "rt");
if (f == NULL){
printf("Error : fopen return NULL\n");
return 0;
}

fgets(buf, 4, f);
metro = malloc(sizeof(*metro) * 1);
metre = malloc(sizeof(*metre) * 1);
fclose(f);

nb_station = atoi(buf);
metro->nb_noeud = nb_station;
metre->nb_noeud = nb_station;
metro->vec_sommets = get_nodes("metro.csv", metro);
metre->vec_sommets = get_list("metro.csv", metro);

while (fin == 0){
while (valide == 0){
printf("Veuillez entrer le nom de la station de depart:\n");
fgets(station_start, sizeof station_start, stdin);
printf("Veuillez entrer le nom de la station d'arriver:\n");
fgets(station_end, sizeof station_end, stdin);
printf("Vous avez choisis de partir de %s Pour aller a %s", station_start,
```

```
for (d = 0; d < 30; d++){
if (station_start[d] == '\n')
station_start[d] = '\0';
if (station_end[d] == '\n')
station_end[d] = '\0';
}

if (test_station(station_start, station_end, metro) == 0){
printf("La station de depart ou d'arriver choisis n'est pas repertoriee\n");
printf("Consultez le fichier stations.txt");
printf("Veuillez relancer la recherche\n");
clear_buf();
}
else
valide = 1;
}

indice_start = recup_indice(station_start, metro);
indice_end = recup_indice(station_end, metro);
valide = 0;
debut_vec = (double)clock();
get_way(metro, indice_start, indice_end);
fin_vec = (double)clock();
debut_list = (double)clock();
get_way_list(metre, indice_start, indice_end);
fin_list = (double)clock();
result_list = fin_list - debut_list;
result_vec = fin_vec - debut_vec;
printf("L'algorithme MC a trouver un chemins en %ftics pour un graphe a vecteur de
printf("L'algorithme MC a trouver un chemins en %ftics pour un graphe a liste de su
printf("Voulez-vous effectuer une autre recherche? [0]YES [1]NO\n");
scanf("%d", &fin);
clear_buf();
}


free_graphe(metro);
```

```
return 0;
}
```

0.8 Find.c

```
#include "Includes/projet.h"

int new_station(graphe_t *metro, int indice_station){
int find = 0;
int indice_rand;
int result;

while (find == 0){
indice_rand = rand() % (metro->vec_sommets[indice_station]->nb_voisins);
result = metro->vec_sommets[indice_station]->tab_indice_voisins[indice_rand
if (metro->vec_sommets[result]->vu == 0)
find = 1;
}
return result;
}

int test_node(graphe_t *metro, int indice_station, int indice_end){
int i;
int nb_station_visit;

i = 0;
nb_station_visit = 0;
if (indice_station == indice_end)
return 0;
while (i < metro->vec_sommets[indice_station]->nb_voisins){
if (DEBUG){
printf("Passage dans la fonction test_node n%d\n", i);
printf("Vu de la station %s est a %d\n", metro->vec_sommets[metro->vec_somm
}
if (metro->vec_sommets[metro->vec_sommets[indice_station]->tab_indice_voisi
nb_station_visit++;
i++;
```

```
}
if (nb_station_visit == metro->vec_sommets[indice_station]->nb_voisins)
return 1;
return 2;
}


void print_way(chemin_t *best_way, graphe_t *metro){
int i;

i = 0;
printf("_____________________________Chemin trouve avec le graphe de vecteur de succe
for (i = 0; i < best_way->nb_arret; i++){
printf("Station : %s\n", metro->vec_sommets[best_way->way[i]]->nom_station);
printf("-------------------------------------------------------------\n");
}
}


void get_way(graphe_t *metro, int indice_start, int indice_end){
chemin_t *best_way;
int *way;
int nb_way;
int result_test;
int i;
int k;
int j;
int continu;
int indice_station;


way = malloc(sizeof(int) * 300);
nb_way = 0;
best_way = malloc(sizeof(chemin_t));
best_way->nb_arret = 300;
best_way->way = malloc(sizeof(int) * 300);
```

```
for (i = 0; i < ROLLOUT; i++){
continu = 1;
indice_station = indice_start;
while(continu == 1){
metro->vec_sommets[indice_station]->vu = 1;
way[nb_way] = indice_station;
nb_way++;
result_test = test_node(metro, indice_station, indice_end);
if (result_test == 2){
indice_station = new_station(metro, indice_station);
}
if (result_test == 0){
if (nb_way < best_way->nb_arret){
best_way->nb_arret = nb_way;
for (j = 0; j < nb_way; j++)
best_way->way[j] = way[j];
}
continu = 0;
}
if (result_test == 1)
continu = 0;
}
for (k = 0; k < metro->nb_noeud; k++)
metro->vec_sommets[k]->vu = 0;
for (k = 0; k < 300; k++)
way[k] = 0;
nb_way = 0;
}
if (best_way->nb_arret == 300){
printf("La recherche d'un chemins a echoue\n");
}
else
print_way(best_way, metro);
free(way);
}
```

0.9 Find_list.c

```
#include "Includes/projet.h"

int new_station_list(graphe_l_t *metro, int indice_station){
int find = 0;
int indice_rand;
int result;
list_t *save;
int i;

while (find == 0){
save = malloc(sizeof(list_t*));
save = metro->vec_sommets[indice_station]->voisins->first;
indice_rand = rand() % (metro->vec_sommets[indice_station]->nb_voisins);
i = 0;
while (i != indice_rand){
save = save->next;
i++;
}
result = save->indice_voisin;
if (metro->vec_sommets[result]->vu == 0)
find = 1;
}
return result;
}

int test_node_list(graphe_l_t *metro, int indice_station, int indice_end){
list_t *save;

save = malloc(sizeof(list_t *));
save = metro->vec_sommets[indice_station]->voisins->first;
if (indice_station == indice_end){
return 0;
}
while (save->next){
if (metro->vec_sommets[save->indice_voisin]->vu == 0){
return 2;
```

```
}
save = save->next;
}
if (metro->vec_sommets[save->indice_voisin]->vu == 0)
return 2;
return 1;
}

void print_way_list(chemin_t *best_way, graphe_l_t *metro){
int i;

i = 0;
printf("_____________________________Chemin trouve avec le graphe de liste de
for (i = 0; i < best_way->nb_arret; i++){
printf("Station : %s\n", metro->vec_sommets[best_way->way[i]]->nom_station)
printf("-------------------------------------------------------------\n");
}
}


void get_way_list(graphe_l_t *metro, int indice_start, int indice_end){
chemin_t *best_way;
int *way;
int nb_way;
int result_test;
int i;
int k;
int j;
int continu;
int indice_station;

way = malloc(sizeof(int) * 300);
nb_way = 0;
best_way = malloc(sizeof(chemin_t));
best_way->nb_arret = 300;
best_way->way = malloc(sizeof(int) * 300);

for (i = 0; i < ROLLOUT; i++){
```

```
continu = 1;
indice_station = indice_start;
while(continu == 1){
metro->vec_sommets[indice_station]->vu = 1;
way[nb_way] = indice_station;
nb_way++;
result_test = test_node_list(metro, indice_station, indice_end);
if (result_test == 2){
indice_station = new_station_list(metro, indice_station);
}
if (result_test == 0){
if (nb_way < best_way->nb_arret){
best_way->nb_arret = nb_way;
for (j = 0; j < nb_way; j++)
best_way->way[j] = way[j];
}
continu = 0;
}
if (result_test == 1)
continu = 0;
}
for (k = 0; k < metro->nb_noeud; k++)
metro->vec_sommets[k]->vu = 0;
for (k = 0; k < 300; k++)
way[k] = 0;
nb_way = 0;
}
if (best_way->nb_arret == 300){
printf("La recherche d'un chemins a echoue\n");
}
else
print_way_list(best_way, metro);
free(way);
}
```

0.10 Recup.c

```
#include "Includes/projet.h"


//retourne 0 si la station n est pas dans le graphe
int test_station(char station_start[30], char station_end[30], graphe_t *me
int i;
int valid_start;
int valid_end;

for (i = 0; i < metro->nb_noeud; i++){
if (strcmp(station_start, metro->vec_sommets[i]->nom_station) == 0)
valid_start = 1;
if (strcmp(station_end, metro->vec_sommets[i]->nom_station) == 0)
valid_end = 1;
}
if (valid_start == 1 && valid_end == 1)
return 1;
return 0;
}

int recup_indice(char station[30], graphe_t *metro){
int i;

for (i = 0; i < metro->nb_noeud; i++){
if (strcmp(station, metro->vec_sommets[i]->nom_station) == 0)
return i;
}
return (-1);
}
```