Rapport NOMMATIERE
INTITULEMATIERE

Rapport récapitulatif du tp courant d'info0605

Créé par :

PRENOMAUTEURDOC NOMAUTEURDOC

Ufr Sciences Exactes Et Naturelles
URCA

2021

Rapport NUMEROTP NOMMATIERE INTITULEMATIERE

NOMAUTEURDOC, PRENOMAUTEURDOC
ADRESSEEMAILDOC

13 mai 2021

Table des matières

1 chapitre de test

test

2 démo code inclus en C

2.1 la section ci dessous est un exemple

Je montre seulement les inclusions, vous pouvez appliquer le même principe pour chaque langage de programmation.

2.2 inclusion du code de la cellule

2.2.1 headers d'une cellule

Listing 2.1 – codes/liste_chainee/cellule.h

```
#ifndef __CELLULE_H__
#define __CELLULE_H__

typedef struct cell
{
    struct cell* pred;
    struct cell* succ;
    int cle;
} cellule_t;

void creer_cellule(cellule_t* c, int val);
void detruire_cellule(cellule_t** c);

#endif
```

2.2.2 code d'une cellule

Listing 2.2 – codes/liste_chainee/cellule.c

```c
#include <stdlib.h> /* pour NULL */
#include "cellule.h"

void creer_cellule(cellule_t* c, int val){
    c->succ = NULL;
    c->pred = NULL;
    c->cle = val;
}

void detruire_cellule(cellule_t** c){
    free(*c);
    *c = NULL;
}
```

2.3 inclusion du code du graphe

2.3.1 headers d'un graphe

Listing 2.3 – codes/graphe/graphe.h

```
#ifndef __GRAPHE_H__
#define __GRAPHE_H__

#include "liste.h"
#include "file.h"
#include "pile.h"

typedef struct graphe {
    int n_sommets;
    int oriente;
    int value;
    liste_t* l_adj;
    int** m_adj;
} graphe_t;

void initialiser_graphe(graphe_t* graphe, char* str);
void creer_matrice_adj(graphe_t* graphe, int** arretes, int nb_arretes);
void afficher_liste_adj(graphe_t* graphe);
void detruire_liste_adj(graphe_t* graphe);
void creer_liste_adj(graphe_t* graphe, int** arretes, int nb_arretes);
void afficher_matrice_adj(graphe_t* graphe);
void detruire_matrice_adj(graphe_t* graphe);
void afficher_graphe(graphe_t* graphe);
void detruire_graphe(graphe_t* graphe);

void parcours_largeur_matrice(graphe_t* graphe, sommet_t* sommets,
   sommet_t* s);
void parcours_largeur_liste(graphe_t* graphe, sommet_t* sommets,
   sommet_t* s);
void afficher_chemin(graphe_t* graphe, sommet_t* s, sommet_t* v);

void parcours_profondeur_matrice(graphe_t* graphe, sommet_t* sommets);
void visiter_pp_matrice(graphe_t* graphe, sommet_t* sommets, sommet_t* u
   , int* date);
//void parcours_profondeur_liste(graphe_t* graphe, sommet_t* sommets);
//void visiter_pp_liste(graphe_t* graphe, sommet_t* sommets, sommet_t* u
   , int* date);
void afficher_parcours_profondeur(graphe_t* graphe, sommet_t* sommets);
void parcours_profondeur_iteratif(graphe_t* graphe, sommet_t* sommets);
   /* le deuxieme du cours */

#endif //__GRAPHE_H__
```

2.3.2 code d'un graphe

Listing 2.4 – codes/graphe/graphe.c

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

#include "graphe.h"

void initialiser_graphe(graphe_t* graphe, char* str){
    char buffer[41] = "";
    int tmp, i, nb_arretes = 0, fin = 0;
    liste_t l1, l2;
    cellule_t *c1, *c2, *tmp1, *tmp2;
    int **arretes;
    FILE* f;

    f = fopen(str,"r");
    fscanf(f,"%s %d",buffer,&(graphe->n_sommets));
    fscanf(f,"%s %d",buffer,&(graphe->oriente));
    fscanf(f,"%s %d",buffer,&(graphe->value));
    fscanf(f,"%s",buffer);

    /* remplacer par liste_t car on ne connait pas la nb_arretes avant
       le parcours */
    initialiser_liste(&l1);
    initialiser_liste(&l2);
    /*tab1 = (int*)malloc(sizeof(int)*(graphe->n_sommets));
    tab2 = (int*)malloc(sizeof(int)*(graphe->n_sommets));*/

    if((strcmp(buffer,"DEBUT_DEF_ARETES") == 0) || (strcmp(buffer,"
       DEBUT_DEF_ARCS") == 0)){
        while(fin == 0){
            fscanf(f,"%s",buffer);
            if((strcmp(buffer,"FIN_DEF_ARETES") != 0) && (strcmp(buffer,
               "FIN_DEF_ARCS") != 0)){
                c1 = (cellule_t*)malloc(sizeof(cellule_t));
                creer_cellule(c1,atoi(buffer));
                inserer(&l1,c1);
                fscanf(f,"%d",&tmp);
                c2 = (cellule_t*)malloc(sizeof(cellule_t));
                creer_cellule(c2,tmp);
                inserer(&l2,c2);
                ++nb_arretes;
            } else {
                fin = 1;
            }
```

```
        }
    }

    arretes = (int**)malloc(sizeof(int*)*nb_arretes);
    for(i = 0; i < nb_arretes; ++i){
        arretes[i] = (int*)malloc(sizeof(int)*2);
    }

    i=0;

    while(l1.tete != NULL){
        arretes[i][0] = l1.tete->cle;
        tmp1 = l1.tete;
        l1.tete = l1.tete->succ;
        free(tmp1);
        tmp1 = NULL;

        arretes[i][1] = l2.tete->cle;
        tmp2 = l2.tete;
        l2.tete = l2.tete->succ;
        free(tmp2);
        tmp2 = NULL;

        ++i;
    }

    creer_liste_adj(graphe,arretes,nb_arretes);
    creer_matrice_adj(graphe,arretes,nb_arretes);

    for(i = 0; i < nb_arretes; ++i){
        free(arretes[i]);
    }

    free(arretes);
    fclose(f);
}

void creer_liste_adj(graphe_t* graphe, int** arretes, int nb_arretes){
    int i;
    cellule_t *cell1 = NULL, *cell2 = NULL;
    graphe->l_adj = (liste_t*)malloc(sizeof(liste_t)*graphe->n_sommets);

    for(i = 0; i < graphe->n_sommets; ++i){
        initialiser_liste(&graphe->l_adj[i]);
    }

    for(i = 0; i < nb_arretes; ++i){
        cell1 = (cellule_t*)malloc(sizeof(cellule_t));
```

```
            creer_cellule(cell1,arretes[i][1]);
            inserer(&graphe->l_adj[arretes[i][0]],cell1);
            if(graphe->oriente == 0){
                cell2 = (cellule_t*)malloc(sizeof(cellule_t));
                creer_cellule(cell2,arretes[i][0]);
                inserer(&graphe->l_adj[arretes[i][1]],cell2);
            }
        }
}

void afficher_liste_adj(graphe_t* graphe){
    int i;
    printf("Listes d'adjacences : \n");
    for(i = 0; i < graphe->n_sommets; ++i){
        printf("%d ---> ",i);
        afficher_liste(&graphe->l_adj[i]);
    }
    printf("\n");
}

void detruire_liste_adj(graphe_t* graphe){
    int i;
    for(i = 0; i < graphe->n_sommets; ++i){
        detruire_liste(&graphe->l_adj[i]);
    }
    free(graphe->l_adj);
}

void creer_matrice_adj(graphe_t* graphe, int** arretes, int nb_arretes){
    int i, j;
    graphe->m_adj = (int**)malloc(sizeof(int*)*graphe->n_sommets);
    for(i = 0; i < graphe->n_sommets; ++i){
        graphe->m_adj[i] = (int*)malloc(sizeof(int)*graphe->n_sommets);
    }

    for(i = 0; i < graphe->n_sommets; ++i){
        for(j = 0; j < graphe->n_sommets; ++j){
            graphe->m_adj[i][j] = 0;
        }
    }

    for(i = nb_arretes - 1; i >= 0; --i){
        graphe->m_adj[arretes[i][0]][arretes[i][1]] = 1;
        if(graphe->oriente == 0){
            graphe->m_adj[arretes[i][1]][arretes[i][0]] = 1;
        }
    }
}
```

```
void afficher_matrice_adj(graphe_t* graphe){
    int i, j;
    printf("Matrice d'adjacences : \n");
    for(i = 0; i < graphe->n_sommets; ++i){
        for(j = 0; j < graphe->n_sommets; ++j){
            printf("[%d]",graphe->m_adj[i][j]);
        }
        printf("\n");
    }
}

void detruire_matrice_adj(graphe_t* graphe){
    int i;
    for(i = 0; i < graphe->n_sommets; ++i){
        free(graphe->m_adj[i]);
    }
    free(graphe->m_adj);
}

void afficher_graphe(graphe_t* graphe){
    printf("Nombre de sommets : %d\n",graphe->n_sommets);
    if(graphe->oriente == 0){
        printf("Non oriente\n");
    } else {
        printf("Oriente\n");
    }
    if(graphe->value == 0){
        printf("Non value\n");
    } else {
        printf("Value\n");
    }
    afficher_liste_adj(graphe);
    afficher_matrice_adj(graphe);
}

void detruire_graphe(graphe_t* graphe){
    detruire_liste_adj(graphe);
    detruire_matrice_adj(graphe);
}

/*
* ##### Parcours Largeur #####
*/

void parcours_largeur_matrice(graphe_t* graphe, sommet_t* sommets,
   sommet_t* s){
    int i;
```

```
    file_t file;
    sommet_t *u, *v;
    /*
    * j'initialise tous les sommets a BLANC et SANSPERE
    */
    for(i = 0; i < graphe->n_sommets; ++i){
        initialiser_sommet_largeur(&sommets[i],i);
    }
    s->couleur = GRIS;
    s->distance = 0;

    initialiser_file(&file,graphe->n_sommets);
    enfiler(&file,s);
    while(!file_vide(&file)){
        u = defiler(&file);
        for(i = 0; i < graphe->n_sommets; ++i){
            if(graphe->m_adj[u->id][i] == 1){
                v = &sommets[i];
                if(v->couleur == BLANC){
                    v->couleur = GRIS;
                    v->pere = u;
                    v->distance = u->distance + 1;
                    enfiler(&file,v);
                }
            }
        }
        u->couleur = NOIR;
    }

    /* printf("\n");
    for(i = 0; i < graphe->n_sommets; ++i){
        if(sommets[i].pere != NULL){
        printf("{%d,%d,%d,%d}\n",sommets[i].id,sommets[i].couleur,
           sommets[i].distance,sommets[i].pere->id);
        } else {
            printf("{%d,%d,%d,-1}\n",sommets[i].id,sommets[i].couleur,
               sommets[i].distance);
        }
    } */

    free(file.sommets);
}

void parcours_largeur_liste(graphe_t* graphe, sommet_t* sommets,
   sommet_t* s){
    int i;
    file_t file;
    sommet_t *u, *v;
```

```
    liste_t tmp;
    /*
    * j'initialise tous les sommets a BLANC et SANSPERE
    */
    for(i = 0; i < graphe->n_sommets; ++i){
        initialiser_sommet_largeur(&sommets[i],i);
    }
    s->couleur = GRIS;
    s->distance = 0;

    initialiser_file(&file,graphe->n_sommets);
    enfiler(&file,s);
    while(!file_vide(&file)){
        u = defiler(&file);
        for(i = 0; i < graphe->n_sommets; ++i){
            tmp = graphe->l_adj[u->id];
            while(tmp.tete != NULL){
                v = &sommets[tmp.tete->cle];
                if(v->couleur == BLANC){
                    v->couleur = GRIS;
                    v->pere = u;
                    v->distance = u->distance + 1;
                    enfiler(&file,v);
                }
                tmp.tete = tmp.tete->succ;
            }
        }
        u->couleur = NOIR;
    }
    free(file.sommets);
}

/* TODO : finir afficher_chemin */

void afficher_chemin(graphe_t* graphe, sommet_t* s, sommet_t* v){
    if(v->id == s->id){
        printf("%d ",s->id);
    } else {
        if(v->pere == NULL){
            printf("Il n'existe aucun chemin de S a V\n");
        } else {
            afficher_chemin(graphe,s,v->pere);
            printf("%d ",v->id);
        }
    }
}

/*
```

```
* ##### Parcours profondeur #####
*/

void parcours_profondeur_matrice(graphe_t* graphe, sommet_t* sommets){
    int i, date = 0;
    for(i = 0 ; i < graphe->n_sommets; ++i){
        initialiser_sommet_profondeur(&sommets[i],i);
    }

    for(i = 0; i < graphe->n_sommets; ++i){
        if(sommets[i].couleur == BLANC){
            visiter_pp_matrice(graphe,sommets,&sommets[i],&date);
        }
    }
}

void visiter_pp_matrice(graphe_t* graphe, sommet_t* sommets, sommet_t* u
    , int* date){
    int i;
    *date = *date + 1;
    u->date_d = *date;
    u->couleur = GRIS;
    for(i = 0; i < graphe->n_sommets; ++i){
        if((graphe->m_adj[u->id][i] == 1) && (sommets[i].couleur ==
            BLANC)){
            sommets[i].pere = u;
            visiter_pp_matrice(graphe,sommets,&sommets[i],date);
        }
    }
    *date = *date + 1;
    u->date_f = *date;
    u->couleur = NOIR;
}

void afficher_parcours_profondeur(graphe_t* graphe, sommet_t* sommets){
    int i;
    for(i = 0; i < graphe->n_sommets; ++i){
        if(sommets[i].pere != NULL){
        printf("Sommet : %d, date debut : %d, date fin : %d, pere : %d\n
            ",sommets[i].id,sommets[i].date_d,sommets[i].date_f,sommets[i
            ].pere->id);
        } else {
            printf("Sommet : %d, date debut : %d, date fin : %d, pere :
                -1\n",sommets[i].id,sommets[i].date_d,sommets[i].date_f);
        }
    }
}
```

```
void parcours_profondeur_iteratif(graphe_t* graphe, sommet_t* sommets){
    int i, date;
    sommet_t* v;
    pile_t pile;
    for(i = 0 ; i < graphe->n_sommets; ++i){
        initialiser_sommet_profondeur(&sommets[i],i);
    }
    date = 0;
    initialiser_pile(&pile, graphe->n_sommets * graphe->n_sommets);
    for(i = 0; i < graphe->n_sommets; ++i){
        if(sommets[i].couleur == BLANC){
            empiler(&pile,&sommets[i]);
            while(!pile_vide(&pile)){
                v = depiler(&pile);
                if(v->couleur == BLANC){
                    v->couleur = GRIS;
                    date = date + 1;
                    v->date_d = date;
                    for(i = graphe->n_sommets - 1; i >= 0; --i){
                        if((graphe->m_adj[v->id][i] == 1) && (sommets[i
                            ].couleur == BLANC)){
                            sommets[i].pere = v;
                            empiler(&pile,&sommets[i]);
                        }
                    }
                } else {
                    if(v->couleur == GRIS){
                        v->couleur = NOIR;
                        date = date + 1;
                        v->date_f = date;
                        depiler(&pile);
                    } else {
                        if(v->couleur == NOIR){
                            depiler(&pile);
                        }
                    }
                }
            }
        }
    }
    detruire_pile(&pile);
}

/* non NOIR : 0 1 2 3 5 8 11 */
/* TODO : trouver la taille optimale pour la pile */
/*void parcours_profondeur_iteratif(graphe_t* graphe, sommet_t* sommets)
   {
    int i, j, date, flag = 1;
```

```
sommet_t* v;
pile_t pile;
for(i = 0 ; i < graphe->n_sommets; ++i){
    initialiser_sommet_profondeur(&sommets[i],i);
}
date = 0;
initialiser_pile(&pile, graphe->n_sommets * graphe->n_sommets);
for(i = 0; i < graphe->n_sommets; ++i){
    if(sommets[i].couleur == BLANC){
        empiler(&pile,&sommets[i]);
        afficher_pile(&pile);
        printf("for\n");
        while(!pile_vide(&pile)){
            v = depiler(&pile);
            printf("\n\n");
            afficher_pile(&pile);
            if(v->couleur == BLANC){
                v->couleur = GRIS;
                date = date + 1;
                v->date_d = date;
                for(i = graphe->n_sommets - 1; i >= 0; --i){
                    if((graphe->m_adj[v->id][i] == 1) && (sommets[i
                        ].couleur == BLANC)){
                        sommets[i].pere = v;
                        empiler(&pile,&sommets[i]);
                    }
                }
            }
        }
    } else {
        if(sommets[i].couleur == GRIS){
            flag = 1;
            for(j = 0; j < graphe-> n_sommets; ++j){
                if((graphe->m_adj[i][j] == 1) && (sommets[i].couleur
                    != GRIS)){
                    flag = 0;
                }
            }
            if(flag){
                sommets[i].couleur = NOIR;
                date = date + 1;
                sommets[i].date_f = date;
            }
        } else {
            if(sommets[i].couleur == NOIR){
                depiler(&pile);
            }
        }
```

```
        }
    }
    detruire_pile(&pile);
}*/
```