Compte Rendu du TP5 : Marquage Topologique

Ce fichier respecte la norme DokuWiki Realise par EL HIMDI Yasmine & PARMENTIER Laurent Groupe 2.

Difficulté

La difficulté qu'on a rencontré pendant la réalisation de ce Tp se résume surtout dans la compréhension de l'algorithme de marquage topologique. Nous avons passé beaucoup de temps dans la réalisation d'exemples et de cas pratiques pour facilité la compréhension.

Trace d'exécution

Ajout de la variable nb_pred[1)] dans la structure **Cell_som** qui renseigne sur le nombre de predecesseur de chaque sommet.

```
struct Cell_som {
    int sommet;
    int nb_pred;
    struct Cell_arc *psucc;
    struct Cell_som *suivant;
};
```

La determination du nombre de predecesseur se fait dans la fonction

```
 cell_som* charge_graphe(char *nom_fichier);
```

comme suit :

S'il n'y a pas de pointeur vers le sommet origine, on le crée, et on met son nombre de predecesseur à 0

```
        if (!pori)
        {
        pori = creer_sommet(ori,graphe);
            pori->nb_pred = 0;
        graphe = pori;
        }
```

Même chose pour le pointeur vers le sommet extrémité :

```
        if (!pext){
            pext = creer_sommet(ext,graphe);
            pext->nb_pred = 0;
            graphe = pext;
        }
```

Sinon, s'il existe, on incrémente son nombre de prédecesseur :

```
pext->nb_pred++
```

Marquage topologique : Avant de procéder à l'implémentation de la fonction qui gère le marquage topologique, il faut vérifier avant que le marquage est envisagable dans ce graphe. [2)] La vérification se fait à partir de la fonction

```
int marquage_topologique_envisageable(cell_som* graphe);
```

qui retourne un booléen : 1 dès qu'il existe un sommet possédant un nombre de predecesseur null, 0 sinon.

La fonction réalisant le marquage topologique nécessite l'utilisation d'une pile.

L'implémentation des fonctions de gestion de pile est faite dans le dossier

```
./lib/lifo/src
```

de façon à l'utiliser comme bibliotheque pour ce projet, ou dans n'importe quel autre projet.

Remarquez que toutes les fonctions et variables utilisées sont de type

```
void*
```

, et ce pour le but de pouvoir en faire plusieurs usages comme c'est le cas de toute librairie.

L'implementation de la fonction de marquage topologique :

```
int marquage_topologique(cell_som* graphe, char* file);
```

a respectée les étapes énumérée dans le sujet du TP.

En effet :

1. On vérifie d'abord que le graphe est marquable topologiquement
2. On initialise notre pile : Lifo lifo_no_pred = lifo_init();
3. On parcours notre graphe pour determiner les sommets sans pred. Ceux-ci sont ensuite retirer_virtuellement_sommet(psom, lifo_no_pred); du graphe.
4. On vérifie que le nombre de pred est à -1, ceci permet de vérifier si le marquage a été possible ou non.
5. On enregistre le marquage dans un fichier : ecrire_marquage(file, lifo_no_pred);

Tests

Nous avons réalisé un ensemble de tests qui permettent de vérifier que notre algorithme s'éxecute correctement, et que les fonctions retournent bien les résultats attendu.

Lifo

- test_lifo.c

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

int main(){

    int tests = 0;
    int* elmt;

    Lifo l = lifo_init();

/* test empty lifo */
    tests = lifo_is_empty(l) == 1;
#if DEBUG_AFFICHE == 1
    printf("empty : %d == 1\n", lifo_is_empty(l));
#endif

    elmt = (int*) malloc(sizeof(int));
    *elmt = 25;

    lifo_add_elmt(l, elmt);

    elmt = (int*) malloc(sizeof(int));
    *elmt = 27;

    lifo_add_elmt(l, elmt);

    elmt = (int*) malloc(sizeof(int));
```

```
    *elmt = 15;

    lifo_add_elmt(l, elmt);

    elmt = (int*) malloc(sizeof(int));
    *elmt = 17;

    lifo_add_elmt(l, elmt);


    elmt = (int*) malloc(sizeof(int));
    *elmt = 5;

    lifo_add_elmt(l, elmt);

    elmt = (int*) malloc(sizeof(int));
    *elmt = 7;

    lifo_add_elmt(l, elmt);

/* test function lifo_length */
    tests = lifo_length(l) == 6 && tests;

#if DEBUG_AFFICHE == 1
    printf("length : %d == 6\n", lifo_length(l));
#endif

/* test function lifo_head_elmt */
    tests = *(elmt = lifo_head_elmt(l)) == 7 && tests;

/* test order of pop */
#if DEBUG_AFFICHE == 1
    printf("return : %d == 7\n", *(elmt = lifo_head_elmt(l)));
#endif

    tests = *(elmt = lifo_pop_elmt(l)) == 7 && tests;

#if DEBUG_AFFICHE == 1
    printf("return : %d == 5\n", *(elmt = lifo_head_elmt(l)));
#endif

    tests = *(elmt = lifo_pop_elmt(l)) == 5 && tests;


/* test function lifo_length */
    tests = lifo_length(l) == 4 && tests;

#if DEBUG_AFFICHE == 1
    printf("length : %d == 4\n", lifo_length(l));
#endif
```

```
    if(tests == 1)
        printf("lifo: \033[32mOK\033[0m\n");
    else
        printf("lifo: \033[31mproblemes durant les tests\033[0m\n");

    return 0;
}
```

```
yasmine@yasmine-HP-Pavilion-dv6-Notebook-PC:~/Ensicaen/ensicaen.git/media/s2
/algo_avancee/tps/tp5/lib/lifo$ make tests
./bin/test_lifo
empty : 1 == 1
length : 6 == 6
return : 7 == 7
return : 5 == 5
length : 4 == 4
lifo: OK
```

Marquage topologique

- test_ll_adj_topo.c

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

int main()
{
    int tests = 0;
    cell_som *graphe;

    graphe =
charge_graphe("extra/test_graphe_non_marquable_boucle.txt");

    tests = marquage_topologique(graphe, NULL) == 0;

    graphe =
charge_graphe("extra/test_graphe_non_marquable_boucle.txt");

#if DEBUG_AFFICHE == 1
    printf("graphe non marquable : %d == 1\n",
marquage_topologique(graphe, NULL) == 0);
#endif

    /**
```

```
 * test du nombre de predecesseurs (implémenté dans la fonction charge_graphe),
 * sur chaque sommet du graphe marquable
 */
    graphe = charge_graphe("extra/test_graphe_marquable.txt");

    tests = nb_pred(0, graphe) == 0 && tests;

#if DEBUG_AFFICHE == 1
    printf("nb_pred sommet 0 : %d == 0\n", nb_pred(0, graphe));
#endif

    tests = nb_pred(1, graphe) == 1 && tests;

#if DEBUG_AFFICHE == 1
    printf("nb_pred sommet 1 : %d == 1\n", nb_pred(1, graphe));
#endif

    tests = nb_pred(2, graphe) == 1 && tests;

#if DEBUG_AFFICHE == 1
    printf("nb_pred sommet 2 : %d == 1\n", nb_pred(2, graphe));
#endif

    tests = nb_pred(3, graphe) == 1 && tests;

#if DEBUG_AFFICHE == 1
    printf("nb_pred sommet 3: %d == 1\n", nb_pred(3, graphe));
#endif

    tests = nb_pred(4, graphe) == 1 && tests;

#if DEBUG_AFFICHE == 1
    printf("nb_pred sommet 4 : %d == 1\n", nb_pred(4, graphe));
#endif

    tests = nb_pred(5, graphe) == 2 && tests;

#if DEBUG_AFFICHE == 1
    printf("nb_pred sommet 5 : %d == 2\n", nb_pred(5, graphe));
#endif

    tests = nb_pred(6, graphe) == 1 && tests;

#if DEBUG_AFFICHE == 1
    printf("nb_pred sommet 6 : %d == 1\n", nb_pred(6, graphe));
#endif

    graphe = charge_graphe("extra/test_graphe_marquable.txt");
```

```
    tests = marquage_topologique(graphe,
"extra/test_marquage_topologique_graphe_marquable.txt") && tests;

    graphe = charge_graphe("extra/test_graphe_marquable.txt");

#if DEBUG_AFFICHE == 1
    printf("graphe marquable : %d == 1\n", marquage_topologique(graphe,
NULL) == 1);
#endif

/**
 * On vérifie que le marquage est réalisé dans un bon ordre
 */

#if DEBUG_AFFICHE == 1

affiche_marquage_topo(
"extra/test_marquage_topologique_graphe_marquable.txt");
#endif

    if(tests == 1)
        printf("ll_adj: \033[32mOK\033[0m\n");
    else
        printf("ll_adj: \033[31mproblemes durant les tests\033[0m\n");

    return 0;
}
```

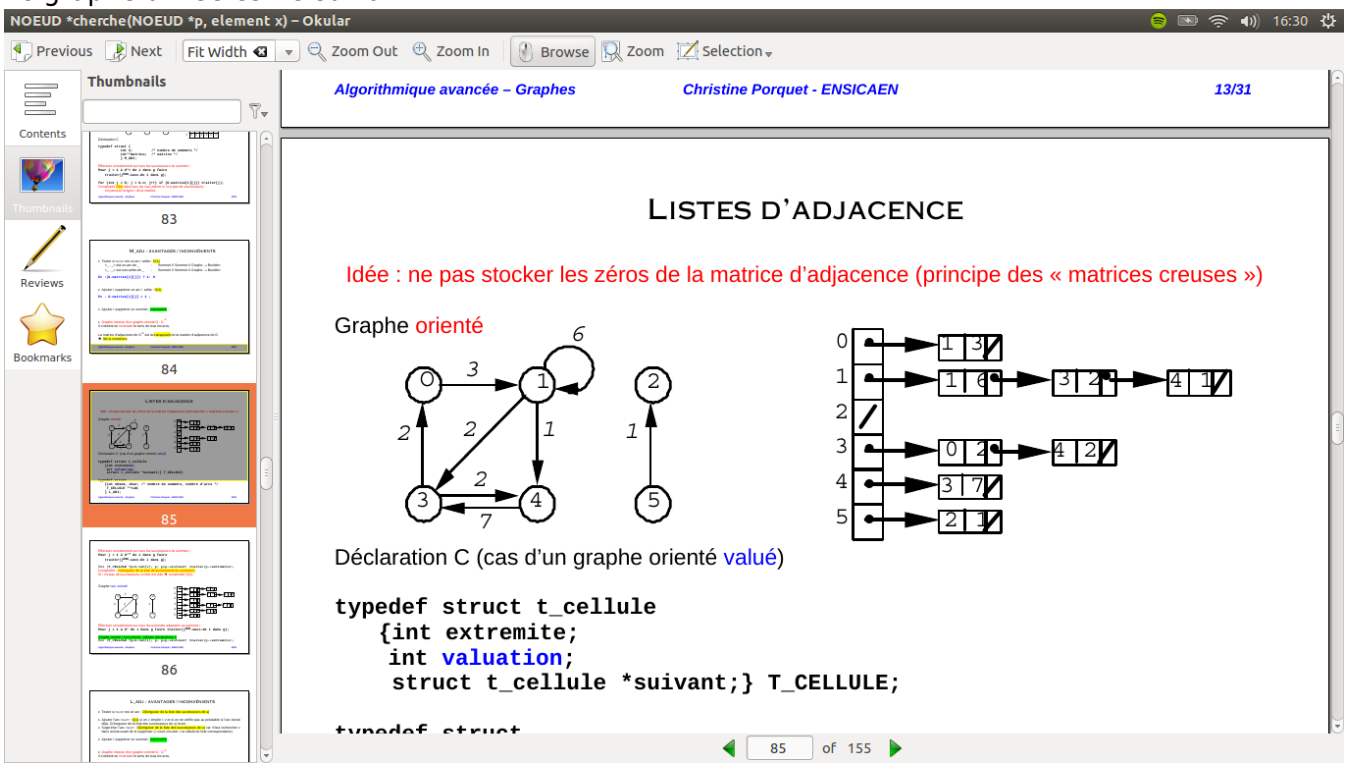
```
yasmine@yasmine-HP-Pavilion-dv6-Notebook-PC:~/Ensicaen/ensicaen.git/media/s2
/algo_avancee/tps/tp5$ make tests
gcc -Wall -Wextra -ansi -pedantic -std=c99 -I ./inc -I ./lib/*/inc -
DDEBUG_AFFICHE=1 -c src/ll_adj.c -o obj/ll_adj.o
gcc -Wall -Wextra -ansi -pedantic -std=c99 -I ./inc -I ./lib/*/inc -
DDEBUG_AFFICHE=1 src/test_ll_adj_topo.c obj/ll_adj.o lib/lifo/obj/lifo.o -o
bin/debug/test_ll_adj_topo
./bin/debug/test_ll_adj_topo
graphe non marquable : 1 == 1
nb_pred sommet 0 : 0 == 0
nb_pred sommet 1 : 1 == 1
nb_pred sommet 2 : 1 == 1
nb_pred sommet 3: 1 == 1
nb_pred sommet 4 : 1 == 1
nb_pred sommet 5 : 2 == 2
nb_pred sommet 6 : 1 == 1
graphe marquable : 1 == 1
Le sommet 5 est attribue au numero 7 d'apres l'attribution topologique
Le sommet 2 est attribue au numero 6 d'apres l'attribution topologique
Le sommet 3 est attribue au numero 5 d'apres l'attribution topologique
Le sommet 6 est attribue au numero 4 d'apres l'attribution topologique
Le sommet 1 est attribue au numero 3 d'apres l'attribution topologique
```

```
Le sommet 4 est attribue au numero 2 d'apres l'attribution topologique
Le sommet 0 est attribue au numero 1 d'apres l'attribution topologique
ll_adj: OK
```

On voit bien que l'ensemble des tests sont réalisés avec succès.

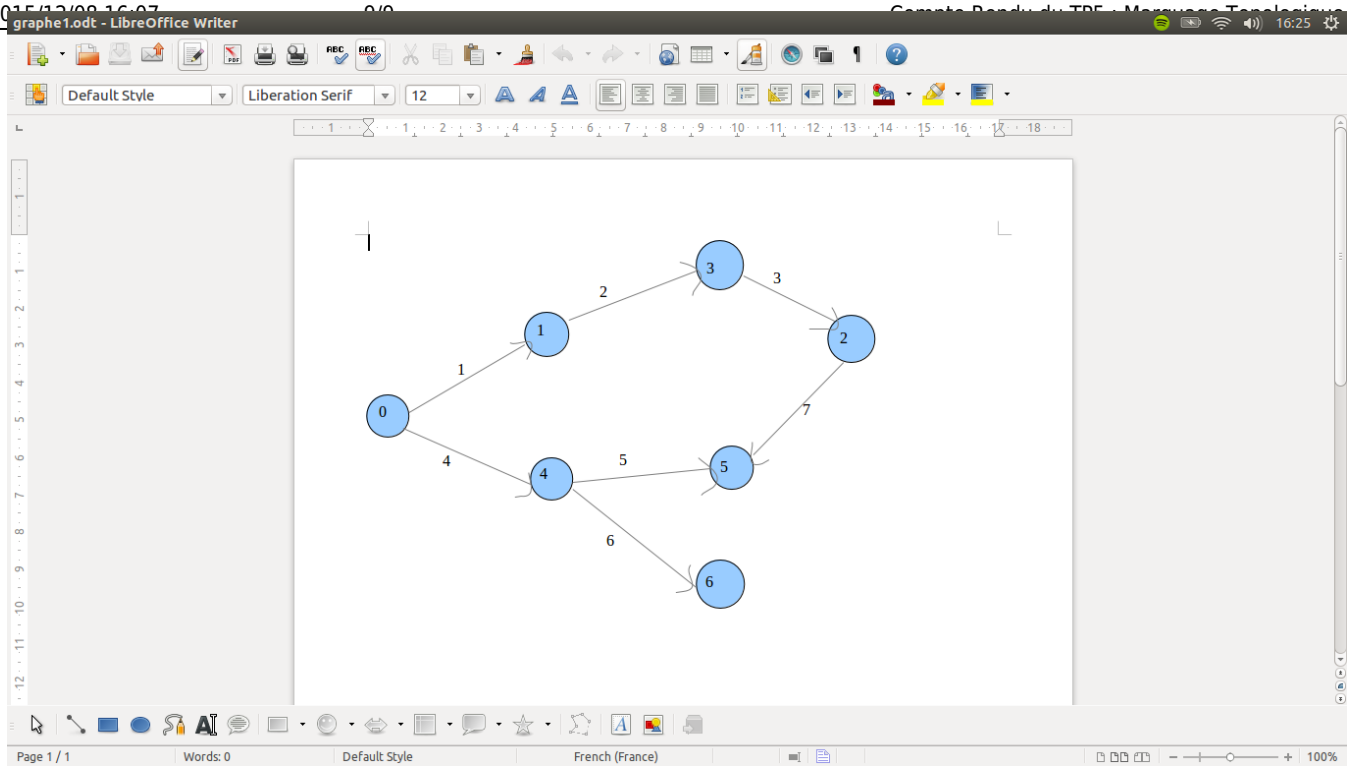
Nos tests réalisent les choses suivantes :

- L'attribution du nombre de predecesseur pour chaque sommet est correctement effectué lors de l'execution de `charge_fichier(…)`
- Le chargement d'un fichier contenant un graphe sur lequel on ne peut réaliser un marquage topologique nous retourne bien 0 lors de l'utilisation de la fonction `marquage_topologique(…)`. Le graphe utilisé est le suivant[3] :



. Fichier texte associé : 6
8
0 1 3
1 1 6
1 3 2
1 4 1
3 0 2
3 4 2
4 3 7
5 2 1

- Le chargement d'un fichier contenant un graphe sur lequel on peut réaliser un marquage topo nous retourne bien 1 lors de l'utilisation de la fonction `marquage_topologique(…)`. Le graphe utilisé est le suivant :



. Fichier texte associé : 7

```
7
0 1 1
0 4 4
1 3 2
3 2 3
4 5 5
4 6 6
2 5 7
```

[1] nombre de predecesseur

[2] **Rq** : Le marquage topo. ne se fait pas dans tous les graphes, les graphes où il y a des sommets possédant toujours un nombre non nul de predecesseurs ne peut pas être marqué.

[3] graphe du cours