Les graphes

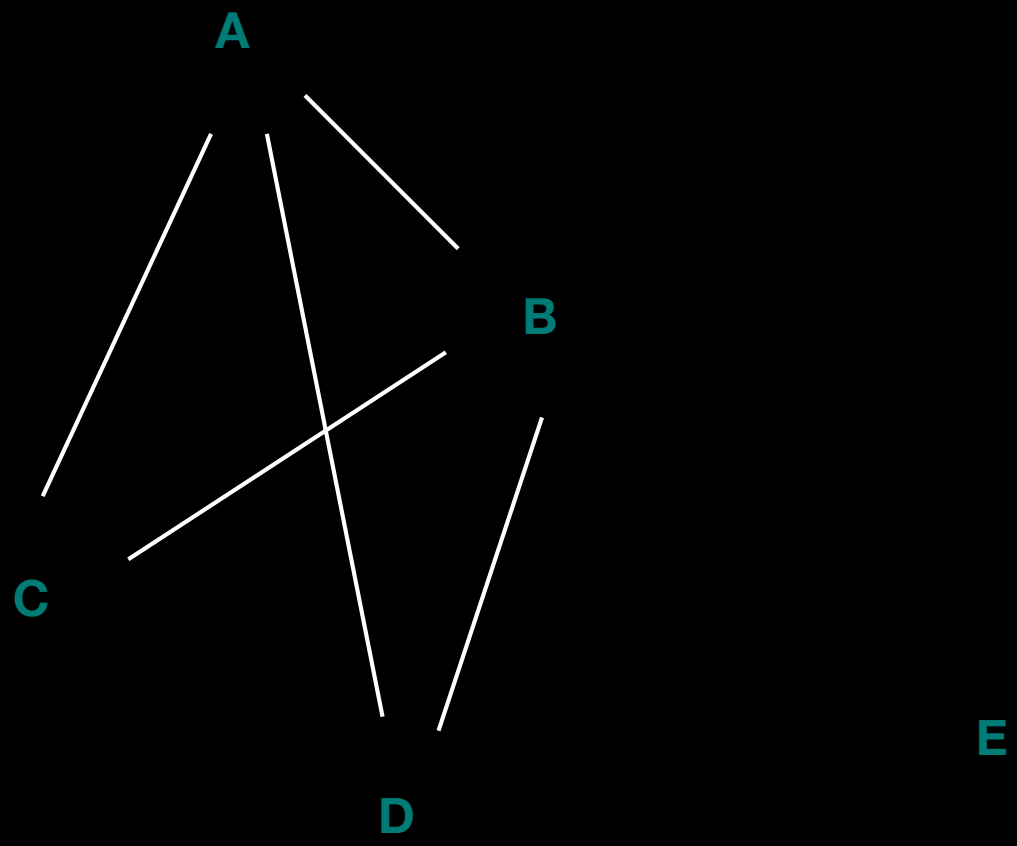
Dorian.H Mekni

27 | MAR | 2020

Les graphes

- Concept tiré des mathématiques, et plus précisément nous parlons de théorie des graphes
- Un graphe est une structure composée d'objets dans laquelle certaines paires d'objets sont en relation
- Les objets correspondent à des abstractions mathématiques et sont appelés sommets
- En C, le graphe est une structure de données composée d’un groupe de sommets, et d’un ensemble de relations entre ces derniers.

Un graphe á 5 sommets



struct

Notre fichier en tête comportant toutes nos struct d'initialisation de graphe ainsi que nos prototypes->

```c
#ifndef graphes_h
#define graphes_h
#include <stdio.h>

// Le booléen
typedef enum
{
    false,
    true
}Bool;

// Un noeud est un sommet
typedef struct NodelistObject
{

    int object;
    struct NodelistObject *next;

}NodeListObject, *NodeList;

// Une liste d'adjacence
typedef struct AdjacencyListObject
{

    NodeListObject *start;

}AdjacencyListObject, *AdjacencyList;

// Le graphe
typedef struct GraphObject
{
    /* Savoir si le deplacement est possible entre sommet 1 et sommer 2, on dit que les sommets sont orientés. Ils sont connectés */
    Bool is_connected;

    /* Savoir, mesurer combien de sommets dans notre graphe*/
    int nb_vertices;

    /* Le tableau */
    AdjacencyList tab_adjacencies;
    FILE *graph_file;

}GraphObject, *Graph;
```

Prototypes .h

```c
// Prototypes
Graph new_graph(int vertices, Bool is_connected);
Bool is_empty_graph(Graph g);
NodeList add_node(int x);
void add_edge(Graph g, int src, int dest);
void clear_graph(Graph g);
void print_graph(Graph g);
void display_graph(Graph g);// For 2D display

#endif /* graphes_h */
```

new_graph()

```c
Graph new_graph(int vertices, Bool is_connected)
{
    int i;
    GraphObject *object;

    object = malloc(sizeof(*object));

    if(object == NULL)
    {
        fprintf(stderr, "Error : Dynamic allocation issue");
        exit(EXIT_FAILURE);
    }

    object->is_connected = is_connected;
    object->nb_vertices = vertices;

    object->tab_adjacencies = malloc(vertices * sizeof(AdjacencyListObject));

    if(object->tab_adjacencies == NULL)
    {
        fprintf(stderr, "Error : Dynamic allocation issue");
        exit(EXIT_FAILURE);
    }
    // On definit nos sommets par default en partant de 1.
    for(i = 1; i < object->nb_vertices + 1; i++)
        object->tab_adjacencies[i-1].start = NULL;
    //Rajout à la création du graph
    if(object->is_connected)
        object->graph_file = fopen("digraph.out", "w");
    else
        object->graph_file = fopen("graph.out", "w");

    if(object->graph_file == NULL)
    {
        fprintf(stderr, "Error : Dynamic allocation issue");
        exit(EXIT_FAILURE);
    }

    if(object->is_connected)
        fprintf(object->graph_file, "digraph my_graph\n{\n}");
    else
        fprintf(object->graph_file, "digraph my_graph\n{\n}");
    // Fin du rajout
    return object;
}
```

is_empty_graph()

Elle permet de vérifier si le graph est vide ou pas

```
Bool is_empty_graph(Graph g)
{
    if(g == NULL)
        return true;

    return false;
}
```

add_node()

add_node()-> contribue à l'ajout d'un noeud

```
// Rajout d'élément
NodeList add_node(int x)
{

    NodeList n = malloc(sizeof(NodeListObject));

    if(n == NULL)
    {
      fprintf(stderr, "Error : Dynamic allocation issue");
      exit(EXIT_FAILURE);
    }

    n->object = x;
    n->next = NULL;

    return n;
}
```

add_edge()

add_edge()-> ajout d'un sommet au graphe avec deux cas de figures : graphe orienté et non orienté

```c
void add_edge(Graph g, int src, int dest)
{
    /*Graphe orienté-> par défault : creation de liaison de source destination*/
    NodeList n = add_node(dest);
    n->next = g->tab_adjacencies[src-1].start;
    g->tab_adjacencies[src-1].start = n;

    /* Graphe non-orienté : Création de destination à source si il est non orienté */
    if(!g->is_connected)
    {
        n = add_node(src);
        n->next = g->tab_adjacencies[src-1].start;
        g->tab_adjacencies[src-1].start = n;
    }

    //Ajout d'un lien dans le fichier Graphviz
    if(g->is_connected)
        fprintf(g->graph_file, "\t%d -> %d;\n", src, dest);
    else
        fprintf(g->graph_file, "\t%d -- %d;\n", src, dest);
}
```

print_graph()

```c
void print_graph(Graph g)
{
    int i;

    if(is_empty_graph(g))
       {
           printf("Nothing to print.\n");
           return;
       }

    for(i = 1; i < g->nb_vertices + 1; i++)
    {
        NodeList n = g->tab_adjacencies[i-1].start;
        printf("(%d) : ", i);

        while(n != NULL)
        {
            printf("%d, ", n->object);
            n = n->next;
        }

        printf("NULL\n");
    }

}
```

clear_graph()

```c
void clear_graph(Graph g)
{
    if(is_empty_graph(g))
    {
        printf("Nothing to delete, No graph found.\n");
        return;
    }
    // Dans les cas où les sommets sont adjacents
    if(g->tab_adjacencies)
    {
        int i;

        for(i = 1; i < g->nb_vertices + 1; i++)
            {
                NodeList n = g->tab_adjacencies[i-1].start;

                while(n != NULL)
                {
                    NodeList tmp = n;
                    n = n->next;
                    free(tmp);
                }
        }

        // Libération de la liste d'adjacences
        free(g->tab_adjacencies);
    }

    //Fin et fermeture du fichier Graphviz
    fprintf(g->graph_file, "}\n");
    fclose(g->graph_file);

    // Libération du Graphe
    free(g);
}
```

Test de Compilation Basique

Test basique avec initiation de graph à 5 sommets - Pas de remplissage de données

```c
#include <stdio.h>
#include<stdlib.h>
#include "graphes.h"

int main(void)
{
  // Graphe à 5 sommets non orienté
    Graph g1 = new_graph(5, false);

    clear_graph(g1);

    return 0;
}

```

Graphe Orienté-> Console

```c
#include <stdio.h>
#include<stdlib.h>
#include "graphes.h"

int main(void)
{
  // Graphe à 5 sommets orienté
    Graph g1 = new_graph(5, true);

    // Chaque sommet est relié
    add_edge(g1, 1, 2);
    add_edge(g1, 1, 5);
    add_edge(g1, 2, 4);
    add_edge(g1, 2, 3);
    add_edge(g1, 3, 4);
    add_edge(g1, 4, 5);

    print_graph(g1);
    clear_graph(g1);

    return 0;
}
```

```
(1) : 5, 2, NULL
(2) : 3, 4, NULL
(3) : 4, NULL
(4) : 5, NULL
(5) : NULL
Program ended with exit code: 0
```

Graphe Non Orienté-> Console

```c
#include <stdio.h>
#include<stdlib.h>
#include "graphes.h"

int main(void)
{
  // Graphe à 5 sommets non orienté
    Graph g1 = new_graph(5, false);

    // Chaque sommet est relié
    add_edge(g1, 1, 2);
    add_edge(g1, 1, 5);
    add_edge(g1, 2, 4);
    add_edge(g1, 2, 3);
    add_edge(g1, 3, 4);
    add_edge(g1, 4, 5);

    print_graph(g1);
    clear_graph(g1);

    return 0;
}

```

```
(1) : 1, 5, 1, 2, NULL
(2) : 2, 3, 2, 4, NULL
(3) : 3, 4, NULL
(4) : 4, 5, NULL
(5) : NULL
Program ended with exit code: 0
```