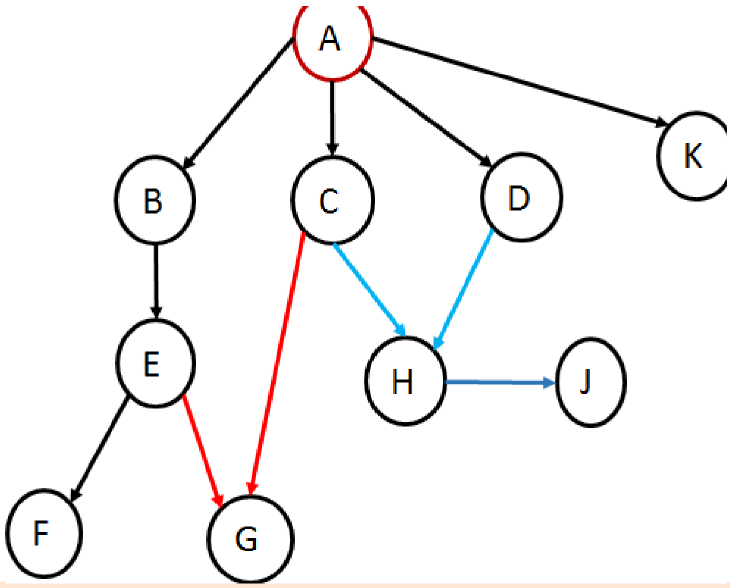
ECOLE SUPERIURE D'ÉDUCATION ET DE LA FORMATION

RAPPORT SUR LES GRAPHES EN C



Réalisé par:

- LAILA HAMZA
- AMAL ADERDOUR
- ASMA ELFAHIM

Structure des données

Sommaire

Introduction **1**

1. Exemples d'application pouvant être modelées par un graphe : ***2***

a. Le problème des ponts de Königsberg 2

b. Choix d'un itinéraire : 3

c. Organisation d'une session d'examens.......... 4

d. Planification de travaux.......... 5

2. Terminologies: ***6***

3. Représentation d'un graphe ***8***

a. Utilisation de matrices.......... 8

b. Matrice d'incidence 9

c. Utilisation des listes d'adjacence 10

4. Algorithmes : ***10***

a. Algorithme de Dijkstra:.......... 10

b. Algorithme de Kruskal : 12

c. Algorithme de Prim : 13

5. Fonctions : ***14***

a. Structure de liste de nœuds 14

b. Structure de liste adjacente 14

c. Structure de graphe 14

d. Création d'un graphe.......... 15

e. Vérifier si le graphe est vide 16

f. Création d’un nœud.......... 16

g. Insertion du sommet du graphe.......... 16

h. Affichage du graphe :.......... 17

i. Supprime un Graphe.......... 18

Conclusion.......... 19

Introduction

Dans le cadre de notre deuxième année en **E**cole **S**upérieure de l'**E**ducation et de la **F**ormation , nous avons eu pour tâche la réalisation d'un programme en langage C. Pour poursuivre la découverte des différentes structures de données, nous allons maintenant nous attarder sur les graphes.

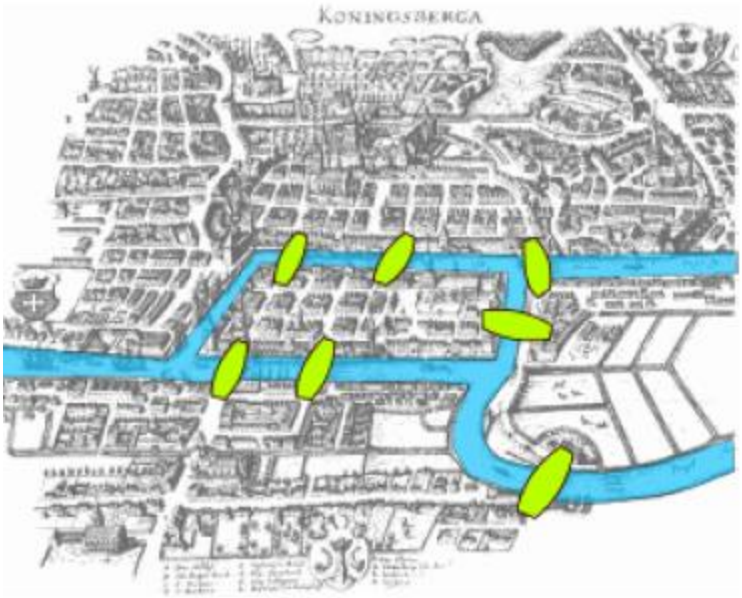
La notion de graphe est une structure combinatoire permettant de représenter de nombreuses situations rencontrées dans des applications faisant intervenir des mathématiques discrètes et nécessitant une solution informatique. Circuits électriques, réseaux de transport (ferrés, routiers, aériens), réseaux d'ordinateurs, ordonnancement d'un ensemble de tâches sont les principaux domaines d'application où la structure de graphe intervient.

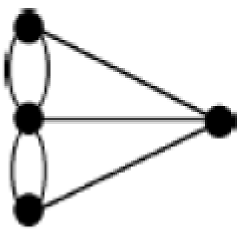
En tant que structures de données, les graphes sont une généralisation des structures qu'on a vues jusqu'à présent, dans la mesure où chaque élément d'un graphe peut posséder plusieurs prédécesseurs et plusieurs successeurs.

1. Exemples d’application pouvant être modelées par un graphe :

a. Le problème des ponts de Königsberg

La ville de Königsberg en Prusse (maintenant Kaliningrad) comprenait 4 quartiers, séparés par les bras du Prégel. Les habitants de Königsberg se demandaient s’il était possible, en partant d’un quartier quelconque de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir à leur point de départ.
Le plan de la ville peut se modéliser à l’aide d’un graphe ci-dessous, les quartiers sont représentés par les 4 sommets, les 7 ponts par des arêtes :





Un cycle eulérien, c'est un chemin qui passe une unique fois par toutes les arrêtes et qui revient à son point de départ. Une chaîne eulérienne, c'est comme un cycle, mais il n'y a pas besoin de retourner au point de départ.

b. Choix d'un itinéraire :

Sachant qu'une manifestation d'étudiants bloque la gare de Poitiers, et connaissant la durée des trajets suivants :

Bordeaux ➔ Nantes 4 h

Bordeaux ➔ Marseille 9 h

Bordeaux ➔ Lyon 12 h

Nantes ➔ Paris-Montparnasse 2 h

Nantes ➔ Lyon 7 h

Paris Montparnasse ➔ Paris Lyon 1 h (en autobus)

Paris-Lyon ➔ Grenoble 4 h 30

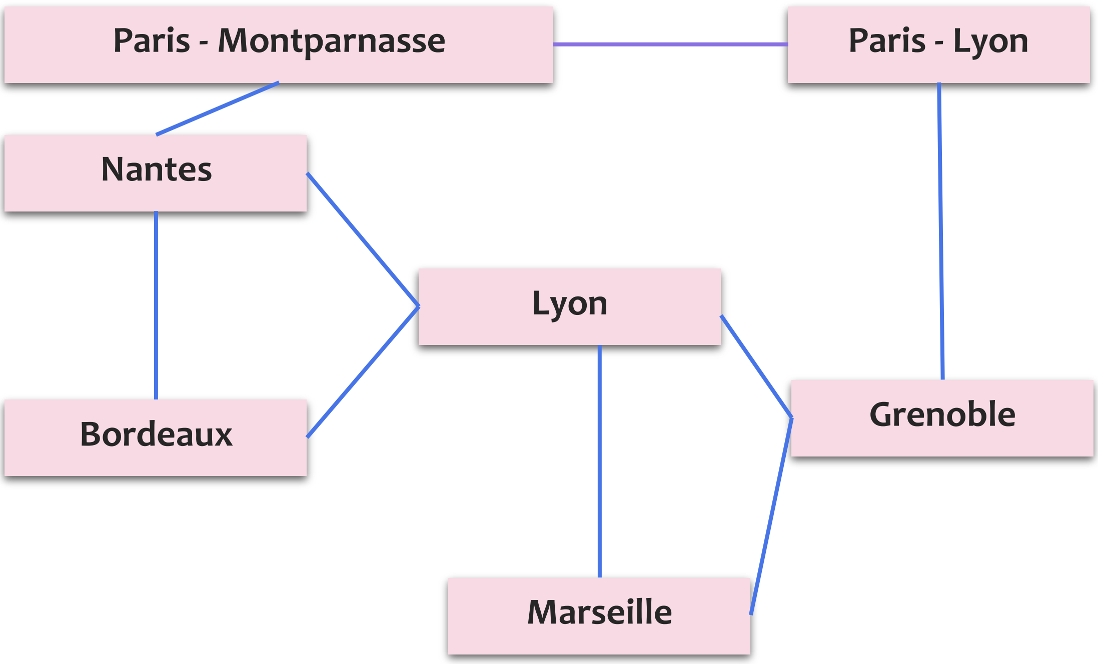
Marseille ➔ Lyon 2 h 30

Marseille ➔ Grenoble 4 h 30

Lyon ➔ Grenoble 1 h 15

Comment faire pour aller le plus rapidement possible de Bordeaux à Grenoble ?

Les données du problème sont faciles à représenter par un graphe dont les arêtes sont étiquetées par les durées des trajets :



Il s'agit de déterminer, dans ce graphe, le plus court chemin (ou l'un des plus courts chemins, s'il existe plusieurs solutions) entre Bordeaux et Grenoble.

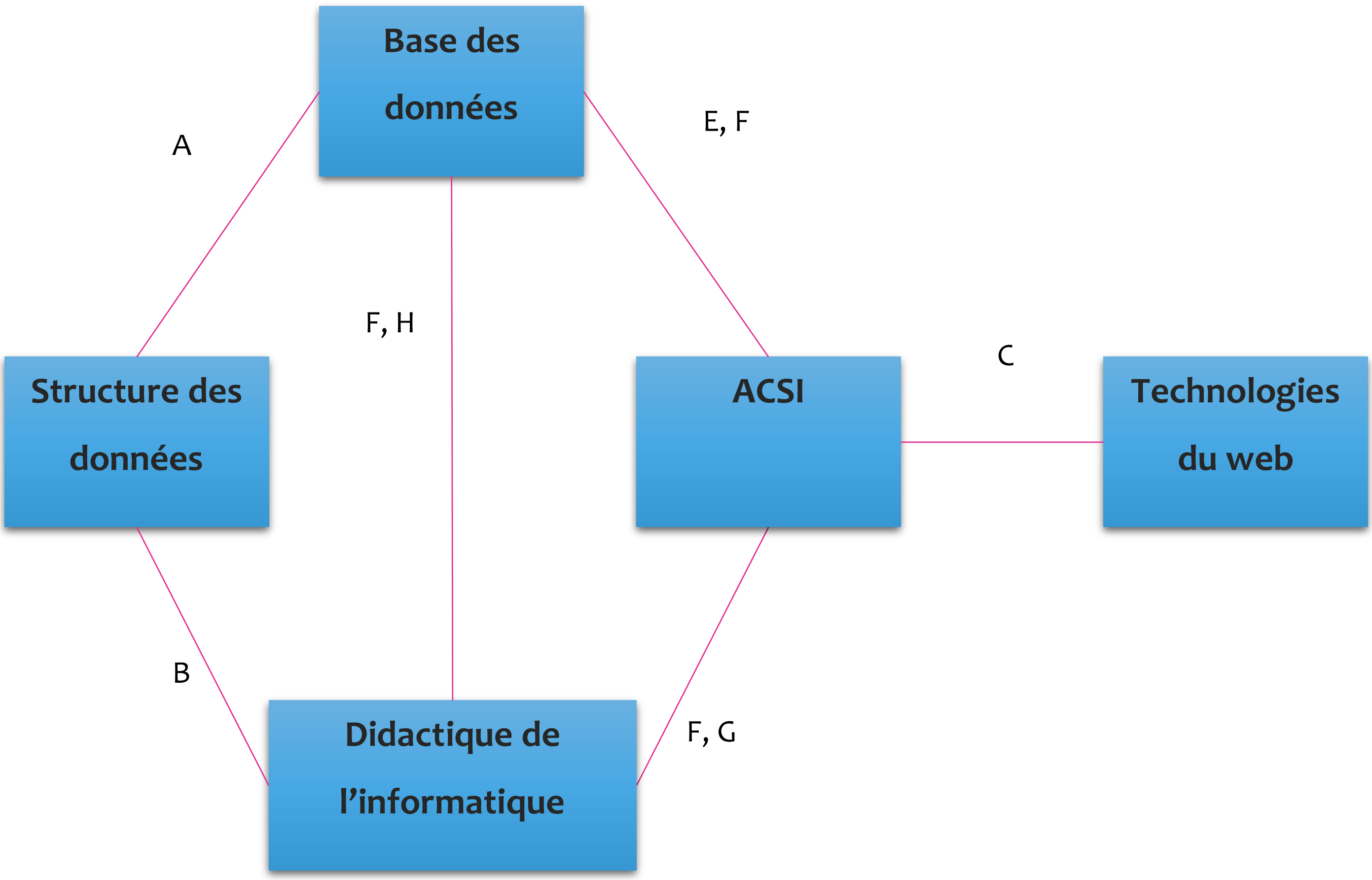
c. Organisation d'une session d'examens

Des étudiants A, B, C, D, E et F doivent passer des examens dans différentes disciplines, chaque examen occupant une demi-journée :

- Structure des données : étudiants A et B
- Technologies du web : étudiants C et D
- Analyse et conception des systèmes d'information : étudiants C, E, F et G
- Base des données : étudiants A, E, F et H
- Didactique de l'informatique : étudiants B, F, G et H

On cherche à organiser la session d'examens la plus courte possible.

On peut représenter chacune des disciplines par un sommet, et relier par des arêtes les sommets correspondant aux examens incompatibles (ayant des étudiants en commun) : Il s'agit alors de colorier chacun des sommets du graphe en utilisant le moins de couleurs possible, des sommets voisins (reliés par une arête) étant nécessairement de couleurs différentes.

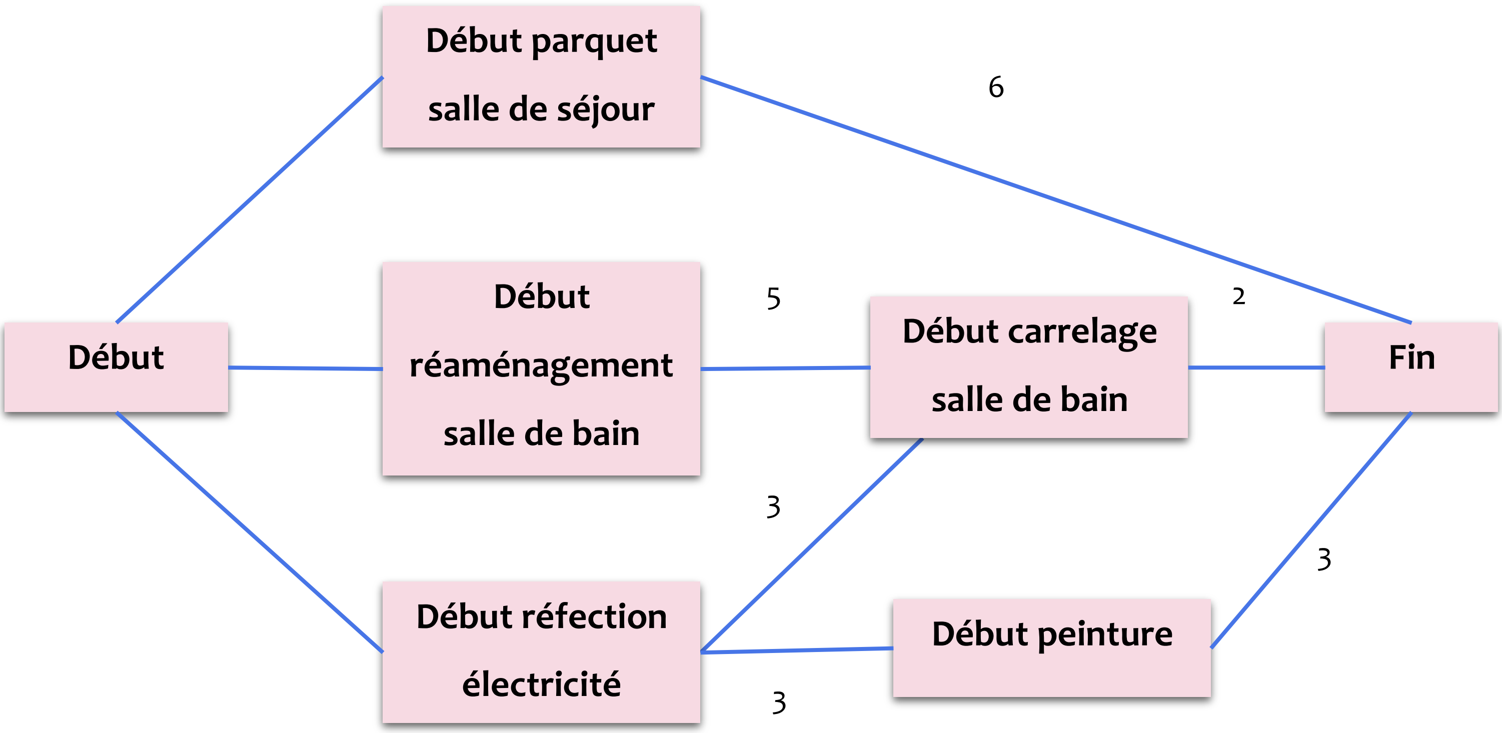


d. Planification de travaux

Pour rénover une maison, il est prévu de refaire l'installation électrique (3 jours), de réaménager (5 jours) et de carreler (2 jours) la salle de bains, de refaire le parquet de la salle de séjour (6 jours) et de repeindre les chambres (3 jours), la peinture et le carrelage ne devant être faits qu'après réfection de l'installation électrique. Si la rénovation est faite par une entreprise et que chacune des tâches est accomplie par un employé différent, quelle est la durée minimale des travaux ?

On peut représenter les différentes étapes de la rénovation sur un graphe dont les arcs sont

étiquetés par la durée minimale séparant deux étapes



Il s'agit de déterminer la durée du plus long chemin du début à la fin des travaux.

De manière générale, un graphe permet de représenter simplement la structure, les connexions, les cheminements possibles d'un ensemble complexe comprenant un grand nombre de situations, en exprimant les relations, les dépendances entre ses éléments. En plus de son existence purement mathématique, le graphe est aussi une structure de données puissante pour l'informatique .

2. Terminologies:

Adjacence	deux arcs sont adjacents s'ils ont une extrémité commune; deux sommets sont adjacents s'il existe un arc, ou une arête, les reliant
Arc	Ligne qui joint deux sommets consécutifs, distincts ou non, d'un graphe orienté.
Arête	nom d'un arc, dans un graphe non orienté
Boucle	Ligne qui joint deux sommets consécutifs, distincts ou non, d'un graphe orienté.
Chaîne	Nom d'un chemin dans un graphe non orienté ; séquence d'arcs avec une extrémité commune dans un graphe orienté.
Chemin	Suite d'arcs connexes reliant un sommet à un autre. Le nombre d'arcs d'un chemin détermine la longueur du chemin. Le plus long chemin d'un graphe orienté est le diamètre de ce graphe.
Chemin eulérien	désigne un chemin simple passant une fois et une seule par toutes les arêtes du graphe ; il n'existe pas toujours...
Chemin hamiltonien	désigne un chemin simple qui passe une fois et une seule par chaque sommet
Cycle hamiltonien	c'est un cycle passant une seule fois par tous les sommets d'un graphe et revenant au sommet de départ.
Circuit	chemin dont l'origine et l'extrémité sont identiques
Connexité	Un graphe est connexe s'il existe toujours une chaîne, ou un chemin, entre deux sommets quelconques. Par exemple le plan d'une ville doit être connexe.
Complet	un graphe est complet si quels que soient deux sommets distincts, il existe un arc (ou une arête) les reliant dans un sens ou dans l'autre
Cycle	Un cycle est une chaine qui permet de partir d'un sommet et revenir a ce sommet en parcourant une et une seule fois les autres sommets.
Degré d'un sommet	Le nombre de sommets avec lesquels il est en relation (on dit aussi Valence).
Diamètre	le diamètre d'un graphe est la plus grande chaîne (chemin) de toutes reliant deux sommets quelconques du graphe.
Distance	la distance entre deux sommets d'un graphe est la plus petite longueur des chaînes, ou des chemins, reliant ces deux sommets.
Graphe orienté	Graphe dans lequel chacune des arêtes reliant deux sommets est orientée (a un sens).
Graphe simple	Désigne un graphe non orienté n'ayant pas de boucle ni plus d'une arête reliant deux sommets. Sur le dessin, les liens entre les sommets sont des segments, et on ne parle alors plus d'arcs mais d'arêtes ; tout graphe orienté peut donc être transformé en graphe simple, en remplaçant les arcs par des arêtes.
Longueur d'un chemin (ou d'une chaîne)	nombre d'arcs du chemin (ou d'arêtes de la chaîne)
Ordre d'un graphe	nombre de sommets du graphe.
Prédécesseur	Dans l'arc (x;y), x est prédécesseur de y

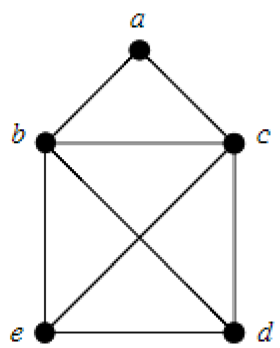
⊙ Graphe Eulérien et Graphe Hamiltonien :

Un graphe Hamiltonien est un graphe où au moins un cycle passe par tous les sommets. Ce cycle est appelé cycle hamiltonien. Il ne faut surtout pas confondre le graphe Hamiltonien au graphe Eulérien qui part d'un sommet quelconque et emprunte une et uniquement une fois chaque arête pour revenir au sommet d'origine, nous avons dans ce cas-là un cycle eulérien.

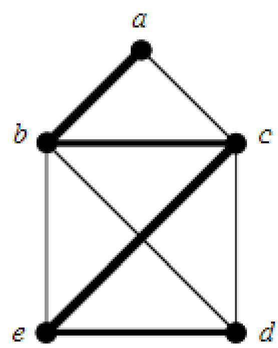
Définitions:

- Un chemin eulérien est un chemin dans le graphe qui passe par toutes les arêtes juste une seule fois. Si ce chemin est fermé, on parlera de cycle eulérien.
- Un chemin hamiltonien est un chemin dans le graphe qui passe par tous les sommets une et une seule fois. Si ce chemin est fermé (i.e. il existe une arête reliant le sommet de départ au sommet d'arrivée), on parlera de cycle hamiltonien.
- Un graphe est dit hamiltonien s'il possède un cycle hamiltonien.
- Un graphe est dit eulérien s'il possède un cycle eulérien.

Exemple:



chemin eulérien:
e-b-d-e-c-a-b-c-d
pas de cycle eulérien



chemin hamiltonien:
d-e-c-b-a
cycle hamiltonien:
d-e-b-a-c

3. Représentation d'un graphe

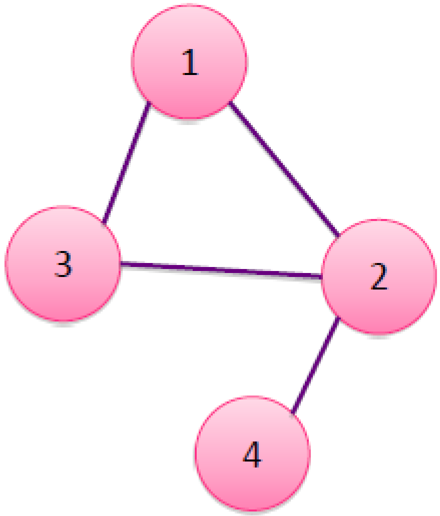
a. Utilisation de matrices

- L'ensemble des sommets du graphe n'évolue pas.
- On utilise un tableau de booléens, dite matrice d'adjacence, de dimension n x n où n= |V|.

L'élément d'indice i et j est vrai si et seulement si il existe un arc entre les sommets i et j.

Exemple: La matrice d'adjacence du graphe est comme suit:

0	1	1	0
1	0	1	1
1	1	0	0
0	1	0	0

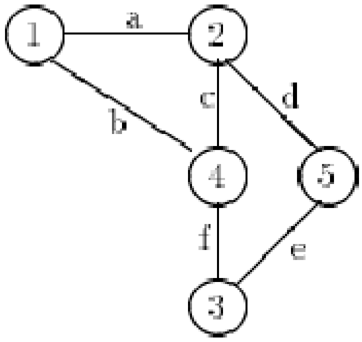


➔ **Avantages :** rapidité des recherches, compacité de la représentation, simplicité des algorithmes de calcul.

➔ **Inconvénients :** représentation ne convenant qu'aux graphes simples; redondance des informations pour les graphes non orientés; stockage inutile de cas inintéressants (les zéros de la matrice), à examiner quand on parcourt le graphe (pour la complexité des algorithmes, le nombre d’arêtes E est à remplacer par *V* 2).

b. Matrice d'incidence

Un graphe non orienté à *n* sommets numérotés et *p* arcs numérotés peut être représenté par une matrice (*n*, *p*) d'entiers : l'élément M[i][j] vaut 1 si le sommet i est une extrémité de l'arête j, 0 sinon :



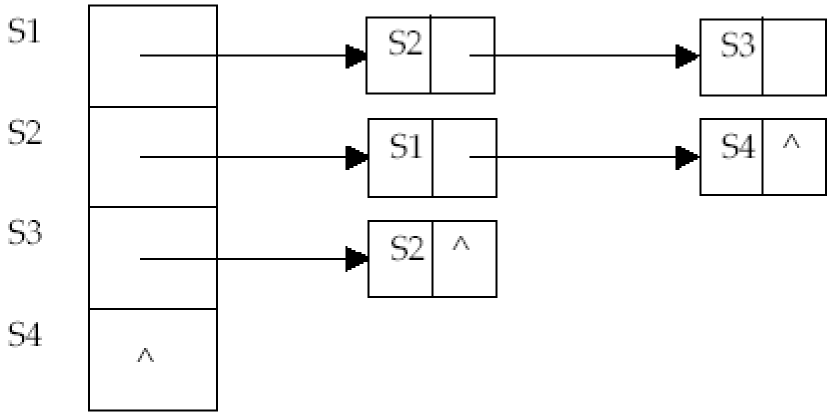
$$
\begin{array}{c}
\\ 1 \\ 2 \\ 3 \\ 4 \\ 5
\end{array}
\begin{array}{c}
\begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0
\end{bmatrix}
\end{array}
$$

Représentation par la matrice d’incidence.

- ➔ **Avantages :** rapidité des recherches, compacité de la représentation, informations non redondantes pour les graphes non orientés;
- ➔ **Inconvénients :** stockage et examen inutile de zéros; les calculs de matrices classiques ne s'appliquent pas.

c. Utilisation des listes d'adjacence

C'est un tableau de liste chaînée. La dimension du tableau est de *n.* Chaque sommet du tableau contient une liste chaînée de sommets qui lui sont adjacents.



4. Algorithmes :

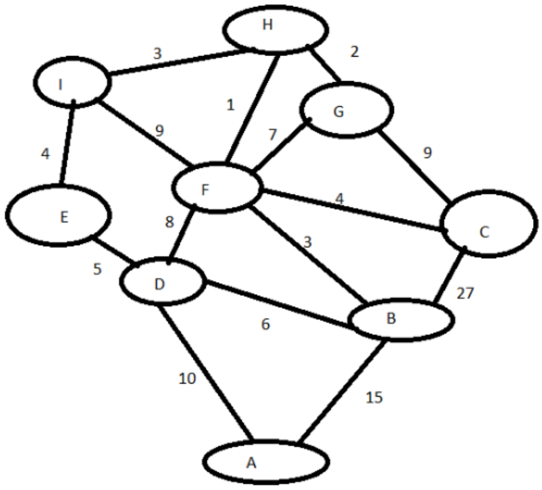
a. Algorithme de Dijkstra:

Le principe de cet algorithme nous permet de déterminer le chemin le plus court. cet algorithme est utilisé dans les GPS afin de trouver le chemin le plus court entre un point A et un point B. Les conditions pour que cet algorithme fonctionne sont :

- avoir un graphe orienté pondéré
- avoir des réels positifs (distance)
- avoir un sommet source.

Dans cet exemple, nous allons montrer à partir d'un sommet source A, atteindre le point d'arrivé H en prenant le chemin le plus court possible :

Il existe ici 2 chemins possible avec la même plus courte distance qui est égale à 19 : {A,D,F,H} ou {A,B,F,H}. Il y a une erreur à ne pas faire, c'est de regarder le segment le plus court à chaque fois, il se peut qu'en suivant ce genre de chemin qu'il y ait une distance plus longue à parcourir que les autres, par exemple ici si on suivait cette méthode, on aurait trouvé {A,D,E,I,H} ayant pour distance 22 ce qui est plus long que les 2 autres.



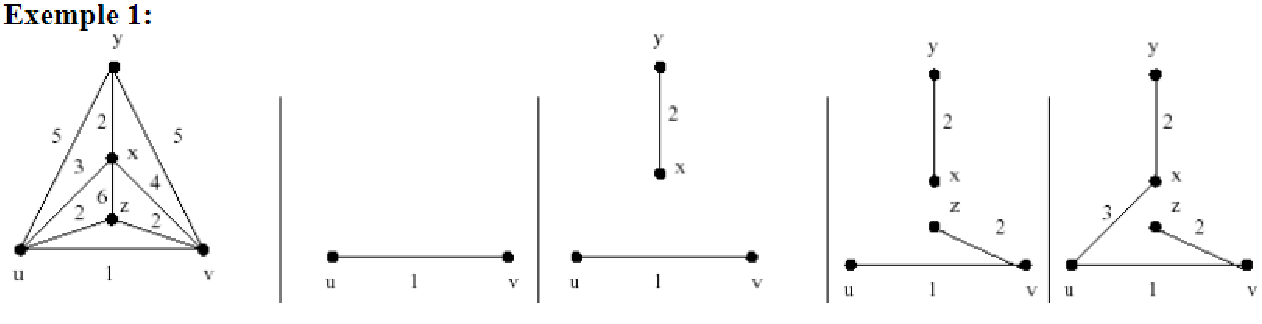

```
Pour k de 1 à n , faire :
Distaneces [ k ] ← cout( s, k) ;
predecesseurs [ k ]← s ;
     Fin Pour ;
M ← Supprimer (s, M);
TantQue (M non vide) Faire:
   i ← LePlusProche(M) ;
   si ( distances [ i ] = \inf ) Alors retour :
      M ← Supprimer( i, M ) ;
      Pour k de 1 à d+(i)  Faire:
               J← K-eme_successeur ( i ) ;
Si ( EstSupprime( j , M)<> 1)
 V ← distances [ i ] + cout( i, j );
 Si ( v < distances[ j ])
    Alors distances [ j ] ←v ;
    Alorspredecesseurs [ j ] ← i ;
  FinSi
                  Finsi
          FinPour
       FinSi.
    FinTantQue.
```

b. Algorithme de Kruskal :

La stratégie de cet algorithme consiste à construire l’arbre en question comme suit : on part d’une solution vide. On choisit, à chaque fois, une arête de G de poids minimum et qui ne crée pas de cycle. Soit E l'ensemble des sommets de G. On utilisera un ensemble d'arêtes T qui sera en sortie l'arbre couvrant minimal et un ensemble d'arêtes F qui représentera les arêtes qui peuvent être choisies.

```
T = { };
F =  E ;
tant que |T| < n - 1 faire
  trouver une arête e de F de poids minimal
  F = F - e
  si T + e est acyclique
    alors T = T + e
  finsi
fin tantQue.
```

Exemple 1:



Les différentes étape pour construire l’arbre T

c. Algorithme de Prim :

L'algorithme de Kruskal veille à maintenir la propriété d'acyclicité d'un arbre alors que l'algorithme de Prim se base sur la connexité d'un arbre. L'algorithme de Prim fait pousser un arbre couvrant minimal en ajoutant au sous-arbre T déjà construit une nouvelle branche parmi les arêtes de poids minimal joignant un sommet de T à un sommet qui n'est pas dans ce dernier. L'algorithme s'arrête lorsque tus les sommets du graphe sont dans T. Soit X l'ensemble de sommets du graphe de départ G. On utilisera un ensemble d'arêtes qui sera en sortie l'arbre recouvrant en question, et S un ensemble qui contiendra les sommets de T.

T = Ø

S= S U x

tant que S ≠ X faire

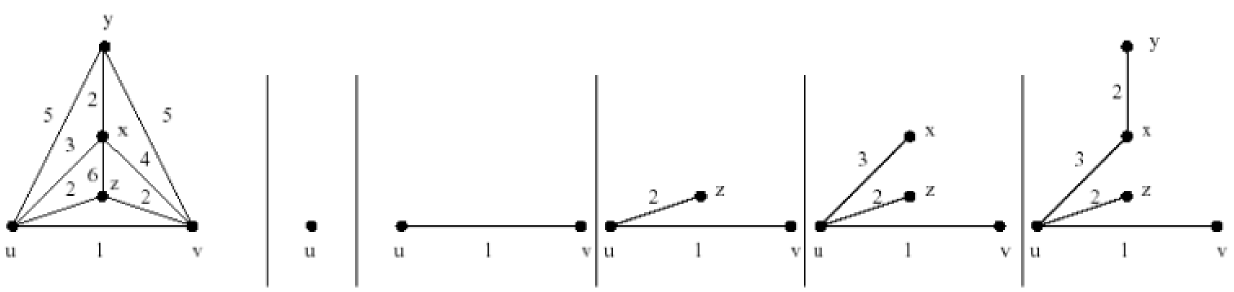
trouver une arête e = {y,s} de poids minimal telque y∈X-S et s∈S

T = T U{y,s}

S = S U y

findutantque

Exemple:



Les différentes étape pour construire l'arbre T

5. Fonctions :

a. Structure de liste de nœuds

Une liste de Nœuds (sommet) contient :

- Une valeur de type int.
- Un pointeur vers un élément du même type appelé Next.

```c
typedef struct NodeListElement
{
    int value;
    struct NodeListElement *next;
}NodeListElement, *NodeList;
```

b. Structure de liste adjacente

Une liste d'adjacence (tableau) contient un pointeur de type NodeListElement qui indique le premier élément de la liste.

```c
typedef struct AdjencyListElement
{
    NodeListElement *begin;
}AdjencyListElement, *AdjencyList;
```

c. Structure de graphe

Structure de graphe contient :

- Un élément bool : true indique que le graphe de type orienté et false indique que le graph de type non orienté,
- Un élément de type int, indiquant le nombre de sommet dans le graphe.
- Un élément de liste adjacente.

```c
typedef struct GraphElement
{
    Bool is_oriented;
    int nb_vertices;
    AdjencyList tab_neighbours;
    FILE *graph_file;
}GraphElement, *Graph;
```

d. Création d’un graphe

Cette fonction aura comme paramètres deux variables vertices de type int indique le nombre des sommets, is_oriented de type Bool indique le type de graphe (orienté, non orienté)

Nous avons commencé par créer un élément de type graph (Créer un graphe) et lui réserver la mémoire

Ensuite, nous vérifions que l’allocation est bien passée, sinon, un message d’erreur s’affichera.

L’étape suivante est d’initialiser les champs :

- nombre des sommets
- type de graphes
- un élément de la liste adjacente, il dépend de nombre de sommet qu'on a choisi, on va devoir faire une allocation dynamique.

Nous vérifions que l’allocation de liste est bien passée, sinon, un message d’erreur s’affichera.

Puis on va initialiser notre liste

```
Graph new_graph(int vertices, Bool is_oriented)
{
    int i;
    GraphElement *element;

    element = malloc(sizeof(*element));

    if(element == NULL)
    {
        fprintf(stderr, "Erreur : Probleme creation Graphe.\n");
        exit(EXIT_FAILURE);
    }

    element->is_oriented = is_oriented;
    element->nb_vertices = vertices;

    element->tab_neighbours = malloc(vertices * sizeof(AdjencyListElement));

    if(element->tab_neighbours == NULL)
    {
        fprintf(stderr, "Erreur : Probleme creation Graphe.\n");
        exit(EXIT_FAILURE);
    }

    for(i = 1 ; i < element->nb_vertices + 1 ; i++)
        element->tab_neighbours[i-1].begin = NULL;
}
```

e. Vérifier si le graphe est vide

Le graphique prend comme paramètre, s'il est vide, alors nous retournons faux, sinon nous retournons vrai.

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

    return false;
}
```

f. Création d'un nœud

On a déjà vu cette fonction en les listes chainées.

```c
NodeList add_node(int x)
{
    NodeList n = malloc(sizeof(NodeListElement));

    if(n == NULL)
    {
        fprintf(stderr, "Erreur : Probleme creation Node.\n");
        exit(EXIT_FAILURE);
    }

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

    return n;
}

```

g. Insertion du sommet du graphe

La fonction prend comme paramètre

- g Le Graphe
- src Le premier sommet (ou source)
- dest Le second sommet (ou destination)

Premièrement on va créer un nœud, en utilisant la fonction addnode et nous passerons destination pour le sauvegarder.

Puis nous créerons la liaison entre la destination et la source, et si le graphe non orienté nous créerons également la liaison entre la source et la destination.

```c
void add_edge(Graph g, int src, int dest)
{
    NodeList n = add_node(dest);
    n->next = g->tab_neighbours[src-1].begin;
    g->tab_neighbours[src-1].begin = n;

    if(!g->is_oriented)
    {
        n = add_node(src);
        n->next = g->tab_neighbours[dest-1].begin;
        g->tab_neighbours[dest-1].begin = n;
    }
}
```

h. Affichage du graphe :

On doit afficher le contenu, tant qu'il existe. Pour ce faire, on a utilisé la boucle for pour mit l'élément de liste adjacente dans liste de nœuds, puis la boucle while si le nœud est non vide on affiche son valeur.

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

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

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

        printf("NULL\n");
    }
}
```

i. Supprime un Graphe

S’il existe des sommets adjacents, on va créer le nœud c'est parce qu'on a besoin de le récupérer

Ensuite si n non vide on va créer un élément temporaire qui sera donc la notion de sauvegarde le n, on se décaler n donc à l'élément suivant, puis on libère ce qui a été mis dans l'élément temporaire.

Puis on va libérer la liste adjacente, et on fin libérer le graphe

```c
void erase_graph(Graph g)
{
    if(is_empty_graph(g))
    {
        printf("Rien a effacer, le Graphe n'existe pas.\n");
        return;
    }

    if(g->tab_neighbours)
    {
        int i;

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

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

    free(g);
}
```

Conclusion

L'objectif de notre rapport est d'explorer les graphes, comprendre leurs applications et leurs algorithmes dont nous aurons besoin pour leur implémentation. De plus la programmation nous a permis d'améliorer nos connaissances du langage C.

Par ailleurs, nous pensons à réaliser un programme en langage C pour mieux comprendre l'utilisation des graphes. Pour cela, nous avons créé un ensemble de fonctions.

En dehors du cadre de notre rapport, un programme pourrait être encore amélioré en ajoutant de nouvelles fonctionnalités de gestion du graphe : ajout et suppression de sommets et d'arêtes.

D'autre part , il serait intéressant de pouvoir faire la manipulation autrement selon le choix d'un critère.