

# Tables de routage

Bouton Nicolas, Dedarally Taariq, Gia Tâm

Mai 2019

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Objectif . . . . .	3
1.2	Instruction de compilation . . . . .	3
1.3	Contenu du projet . . . . .	3
<b>2</b>	<b>Structure de données</b>	<b>4</b>
2.1	Structure du Graphe . . . . .	4
2.2	Tableau du graphe . . . . .	4
2.3	Insert . . . . .	4
2.3.1	Compteur . . . . .	4
2.3.2	Proba . . . . .	4
2.4	Table de routage . . . . .	5
2.4.1	Poids . . . . .	5
2.4.2	Père . . . . .	5
<b>3</b>	<b>Description du projet</b>	<b>6</b>
3.1	Création du graphe . . . . .	6
3.1.1	Initialisation du graphe . . . . .	6
3.1.2	Calcul du graphe . . . . .	6
3.2	Vérification de la connexité . . . . .	6
3.3	Création de la table de routage . . . . .	7
3.4	Reconstitution du chemin . . . . .	7
3.5	Affichage du graphe . . . . .	7
3.5.1	Description . . . . .	7
3.5.2	Fonctionnement de l’affichage . . . . .	7
<b>4</b>	<b>Annexe</b>	<b>8</b>
4.1	Makefile . . . . .	8
4.2	Création du graphe . . . . .	11
4.3	Vérification de la connexité . . . . .	19
4.4	Création de la table de routage . . . . .	22
4.5	Affichage . . . . .	26
4.6	Les autres fichiers . . . . .	38

4.6.1	Constantes . . . . .	38
4.6.2	Bibliothèque flame11 . . . . .	40

# Chapitre 1

## Introduction

### 1.1 Objectif

Le but de ce projet est de créer une application qui calcule la table de routage de chaque noeuds d'un réseaux de 100 noeuds (graphe de 100 noeuds).

### 1.2 Instruction de compilation

Le programme est écrit en C. Pour compiler, il suffit de taper "make", puis pour exécuter taper "./graph".

### 1.3 Contenu du projet

Description du contenu dans le projet :

- main.c programme principal
- graph.c/h pour la création du graphe
- connexe.c/h pour le parcours du graphe
- routage.c/h tableau de routage
- const.h contient les constantes utilisées
- affiche.c/h pour afficher le graphe dans une fenetre Canvas
- moteur\_graphique.c/h bibliothèque fils de flame11
- flame.c/h bibliothèque basée sur X11 crée par Yaspr
- pi.h outils de la bibliothèque fournit par Yaspr
- Makefile permet de compiler le programme

## Chapitre 2

# Structure de données

Nous avons décidé d'utiliser des pointeurs vers les structures au lieu de les passer en argument sans pointeurs.

### 2.1 Structure du Graphe

Cette structure contient 2 champs :

- un tableau qui représente le graphe
- une structure qui permet de bien initialiser le graphe

### 2.2 Tableau du graphe

Ce tableau représente le graphe. S'il y a une arête entre  $i$  et  $j$ , alors le poids de l'arête est noté, sinon il y a -1 dans  $list[i][j]$  et  $list[j][i]$ .

### 2.3 Insert

Cette structure permet de bien initialiser le graphe et contient 2 champs :

- un tableau qui compte le nombre d'arrête vers le même tier
- un tableau où est noté le nombre d'arrête qu'il faut avoir vers le même tier (est utilisé uniquement pour le tier 2 et 3)

#### 2.3.1 Compteur

Ajoute 1 a la valeur au sommet  $i$  à chaque fois qu'on ajoute une arête vers le même tier.

#### 2.3.2 Proba

Ce tableau est initialisé au moment où on initialise le graphe et permet de savoir combien d'arêtes il peut y avoir vers un noeuds du même tier. Pour

le tier 2 et 3 lorsque on ajoute une arête entre le sommet  $i$  et  $j$  il faut aussi l'ajouter de  $j$  vers  $i$  et sauvegarder qu'il y a une arête vers le même tier.

## 2.4 Table de routage

Cette structure contient 2 champs :

- un tableau qui représente la table de routage avec le poids
- un tableau qui représente les successeurs

### 2.4.1 Poids

Ce tableau représente le poids minimum pour aller d'un sommet vers un autre. Par exemple, prenons  $i$  le sommet de départ et  $j$  le sommet d'arrivée.  $\text{poids}[i][j]$  indique le poids minimum pour aller de  $i$  à  $j$ .

### 2.4.2 Père

Ce tableau représente le père du sommet d'arrivée en prenant compte du plus petit chemin.

Par exemple on prend 3 sommet  $i, j$  et  $k$ . Le chemin le plus court pour aller de  $i$  à  $j$  est :  $i \rightarrow \dots \rightarrow k \rightarrow j$ . Donc dans le tableau  $\text{pere}[i][j]$  il y aura  $k$ .

## Chapitre 3

# Description du projet

### 3.1 Création du graphe

#### 3.1.1 Initialisation du graphe

On alloue la mémoire nécessaire pour créer un graphe. On initialise tout le tableau du graphe à -1 pour dire qu'il n'y a pas d'arêtes pour l'instant. Maintenant, on initialise ces arêtes :

- les Backbones (Tier 1) avec aucune arête entre eux pour l'instant.
- les opérateurs de niveau 2 (Tier2) avec un nombre d'arêtes associés au même niveau qui varie entre 2 et 3 ;
- les opérateurs de niveau 3 (Tier3) avec un nombre d'arêtes associé au même niveau qui vaut 1.

Pour chaque sommet, on initialise à 0 le nombre d'arêtes liés d'un sommet à un autre de même rang.

#### 3.1.2 Calcul du graphe

En parcourant les listes, en vérifiant qu'on ne tombe pas sur la diagonale et qu'il y a une arête, on définit aléatoirement le poids de l'arête :

- Tier1 : entre 5 et 10.
- Tier2 : entre 10 et 20.
- Tier3 : entre 15 et 50.

Et on augmente le compteur du sommet traité à chaque fois qu'on définit un poids d'un lien qui mène vers un sommet de même rang.

### 3.2 Vérification de la connexité

On fait un parcours en profondeur du graphe sur chaque sommet à l'aide de la coloration pour vérifier si le graphe est connexe. On a 3 couleurs.

- 0 = non traité
- 1 = en cours de traitement

— 2 = déjà traité

Un compteur est présent pour indiquer si le graphe est connexe. Initialisé à 0, il augmente de 1 à chaque fois qu'on a fait un parcours. Dans l'idéal, faire ce parcours une seule fois uniquement indique que tous les sommets ont tous été traités du 1<sup>er</sup> coup et donc que le graphe est connexe.

### 3.3 Création de la table de routage

Pour la création du chemin on crée une table de routage. La table de routage est composée de deux matrices : une pour les successeurs et une autre pour les poids. On applique ensuite l'algorithme de Floyd-Warshall.

### 3.4 Reconstitution du chemin

Pour restituer le chemin on prend la table de routage calculée et on cherche le plus court chemin dans le tableau de hashage, fonction des identifiants entrée (l'expéditeur/destinataire).

### 3.5 Affichage du graphe

#### 3.5.1 Description

L'affichage du jeu se fait sur une fenêtre graphique : les noeuds sont représentés par des cercles et pour afficher le chemin le plus court entre deux noeuds il faut cliquer sur les cercles. Le plus court chemin sera affiché en jaune. Pour quitter proprement la fenêtre graphique il faut appuyer sur 'q' du clavier.

#### 3.5.2 Fonctionnement de l'affichage

La fenêtre graphique est gérée par la bibliothèque `flame11` créée par Yaspr (lien github en annexe pour consulter le code source) basée sur X11. On a créé une bibliothèque `fls` (`moteur_graphique.c/h`) pour plus facilement faire la gestion de l'affichage d'un graphe.



## Chapitre 4

# Annexe

Lien du code source sur github :

`https://github.com/Sholde/ProjetAlgo/tree/master/src`

Ici dessous les différents fichiers imprimés en pdf :

### 4.1 Makefile

Fichier de compilation :

```
1  # Evite conflit:
2  .PHONY: all lib compil clean
3
4  # Constantes:
5  WARN = -Wall
6  OPTIM = -g3 -Ofast
7  GCC = @gcc
8  MSG = @echo compilation en cours: création de $@
9  FLAME = -std=c99 -O2 -ffast-math flame.c -lm -lX11
10 # all
11 all: graph main.o affiche.o graph.o connexe.o routage.o flame.o moteur_graphique.o
    clean
12
13 # Run:
14 run:
15     @echo -n lancement du programme:
16     ./graph
17     @echo fin de programme:
18
19 # Edition de lien du programme principal
20 graph: main.o graph.o routage.o connexe.o affiche.o moteur_graphique.o
21     ${GCC} -o $@ $^ ${OPTIM} ${WARN} ${FLAME}
22     ${MSG}: fichier executable
23
24 # Compilation du programme principal
25 main.o: main.c *.h
26     ${GCC} -c ${OPTIM} ${WARN} *.c
27     ${MSG}
28
29 # Compilation des fichiers de gestions du programme
30 affiche.o: affiche.c *.h
31     ${GCC} -c ${OPTIM} ${WARN} *.c
32     ${MSG}
33
34 graph.o: graph.c *.h
35     ${GCC} -c ${OPTIM} ${WARN} *.c
36     ${MSG}
37
38 connexe.o: connexe.c *.h
39     ${GCC} -c ${OPTIM} ${WARN} *.c
40     ${MSG}
41
42 routage.o: routage.c *.h
43     ${GCC} -c ${OPTIM} ${WARN} *.c
44     ${MSG}
45
46 flame.o: flame.c *.h
47     ${GCC} -c ${OPTIM} ${WARN} *.c
48     ${MSG}
49
50 moteur_graphique.o: moteur_graphique.c *.h
51     ${GCC} -c ${OPTIM} ${WARN} *.c
52     ${MSG}
53
54 # Netoyage des fichiers
55 clean:
56     @echo netoyage des données suppression de: *.o
```

57 @rm \*.o  
58

## 4.2 Création du graphe

Fichiers pour la création du graphe :

```

1  #include "const.h"
2
3  /**
4   * Algorithme de création de graphe selon les modalités de l'énoncé
5   * */
6
7  typedef struct {
8      int compteur[TAILLE_GRAPHE][3];    // >compte nb voisin vers chaque tier dans la liste
9      int proba[TAILLE_GRAPHE][1];        // >nb de noeuds vers le meme tier
10 } insert;
11
12 // Graphe représenté par une liste d'adjacence
13 typedef struct {
14     int list[TAILLE_GRAPHE][TAILLE_GRAPHE];    // >pointe vers 100 listes** qui pointe chacune vers 3 liste* ( qui sont les liste de voisin vers un tier1, tier2, tier3) qui pointe vers une liste
15     insert I;    // >permet de bien initialiser le graphe
16 } graphe;
17
18 // Génère un graphe
19 // @return graphe graphe generé
20 graphe* init_graphe();
21
22
23 // Verifie si le sommet i est dans la liste de j et inversement
24 // ( meme si ca sert a rien car quand on ajoute un voisin a un sommet on ajoute aussi le sommet au voisin ) est dans la liste
25 // @param G le graphe
26 // @param i sommet
27 // @param etat_i dans quel tier est le sommet i
28 // @param j sommet
29 // @param etat_j dans quel tier est le sommet j
30 // @return booléens
31 int verifSiSommetInListe(graphe* G, int i, int j);
32
33
34 // Test si le nb de noeuds max est atteint
35 // @param G le graphe
36 // @param i sommet
37 // @param noeudsMax le noeuds maximum
38 // @param etat dans quel tier appartient le sommet i
39 // @return booléens
40 int test_noeuds_max(graphe* G, int i, int noeudsMax, int etat);
41
42 // Calcul quel noeuds choisir
43 // @param G le graphe
44 // @param sommet un sommet du graphe
45 // @param deb debut
46 // @param fin fin
47 // @param noeudsMax nb de noeuds max du sommet vers le meme tier
48 // @param etat_sommet dans quel tier appartient le sommet dans laquelle on cherche un voisin
49 // @param etat_i dans quel tier appartient le sommet i ( le voisin )
50 // @return noeuds
51 int calcul_noeuds(graphe* G, int sommet, int deb, int fin, int noeudsMax, int etat_sommet, int etat_i);

```

```
52
53 // calcul les arc du tier 1
54 // @param G le graphe
55 // @return un graphe généré
56 void calculTier1(graphe* G);
57
58 // calcul les arc du tier 2
59 // @param G le graphe
60 // @return un graphe généré
61 void calculTier2(graphe* G);
62
63 // calcul les arc du tier 3
64 // @param G le graphe
65 // @return un graphe généré
66 void calculTier3(graphe* G);
67
```

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  //~ local
5  #include "graph.h"
6  #include "const.h"
7
8  graphe* init_graphe(){
9      graphe* G = malloc(sizeof(graphe));
10
11      int i, j, p = 0;
12
13      // struct liste
14      for(i = 0; i < TAILLE_GRAPHE; i++) // ini toute le tableau a -1
15      {
16          for(j = 0; j < TAILLE_GRAPHE; j++)
17          {
18              G->list[i][j] = -1;
19          }
20      }
21
22      // struct Insert
23      int found = 0;
24      int cmp2;
25      while(!found)
26      {
27          cmp2 = 0;
28          for(i = 0; i < TAILLE_GRAPHE; i++)
29          {
30              if( i < nbTier1 )
31              {
32                  G->I.proba[i][0] = 0;
33              }
34              if( i >= debTier2 && i < finTier2)
35              {
36                  p = rand()%2 + 2;
37                  G->I.proba[i][0] = p;
38              }
39              if( i >= debTier3 && i < finTier3)
40              {
41                  G->I.proba[i][0] = 1;
42              }
43              for(j = 0; j < 3; j++)
44              {
45                  G->I.compteur[i][j] = 0;
46              }
47          }
48
49          for(i = debTier2; i < finTier2; i++)
50          {
51              cmp2 += G->I.proba[i][0] = p;
52          }
53
54          if( cmp2 % 2 == 0 )
55          {
56              found = 1;
57          }
58      }
```

```

58     }
59     return G;
60 }
61
62 int verifSiSommetInListe(graphe* G, int i, int j) {
63     if(G->list[i][j] != -1)
64         return 1;
65     return 0;
66 }
67
68
69 int test_noeuds_max(graphe* G, int i, int noeudsMax, int etat) {
70
71     if(etat == tier1)
72     {
73         if(G->I.compteur[i][etat] < noeudsMax)
74             return 1;
75     }
76     else
77     {
78         if(G->I.compteur[i][etat] < noeudsMax && G->I.compteur[i][etat] <  2
79             G->I.proba[i][0])
80             return 1;
81     }
82     return 0;
83 }
84
85 int calcul_noeuds(graphe* G, int sommet, int deb, int fin, int noeudsMax, int  2
86 etat_sommet, int etat_i) {
87     int distance = fin - deb;
88     int pointeur[distance+1];
89     int i, j = 0;
90
91     for(i = 0; i < distance; i++)
92         pointeur[i] = 0;
93
94     for(i = deb; i < fin; i++)
95     {
96         if(G->I.compteur[i][etat_i] < noeudsMax
97             && !verifSiSommetInListe(G, sommet, i)
98             && i != sommet)
99         {
100             pointeur[j] = i;
101             j++;
102         }
103     }
104     if(j != 0)
105     {
106         int a, b;
107         b = rand();
108         a = b % j;
109         b = pointeur[a];
110         return b;
111     }
112     return -1;

```



```

113 }
114
115 void calculTier1(graphe* G) {
116
117     int i, j, k;
118     int p = 7500;
119
120     for(i = debTier1; i < finTier1 ; i++)
121     {
122         for(j = debTier1; j < finTier1; j++)
123         {
124             if(i != j && !verifSiSommetInListe(G, i, j))
125             {
126                 k = rand()%1000;
127                 if(k < p)
128                 {
129                     k = rand()%(poidsMaxTier2 - poidsMinTier2 + 1) + poidsMinTier2;
130
131                     G->list[i][j] = k;
132                     G->list[j][i] = k;
133                     G->I.compteur[i][0]++;
134                     G->I.compteur[j][0]++;
135                 }
136             }
137         }
138     }
139 }
140
141 void calculTier2(graphe* G){
142
143     int i, j;
144     int p, noeuds, k;
145
146     for(i = debTier2; i < finTier2; i++)
147     {
148         // pour les arc vers le tier precedent
149         p = rand()%2 + 1;
150         for(j = 0; j < p; j++)
151         {
152             noeuds = calcul_noeuds(G, i, debTier1, finTier1, 100, tier2, tier1);
153             k = rand()%(poidsMaxTier2 - poidsMinTier2 + 1) + poidsMinTier2;
154             if(noeuds != -1)
155             {
156                 G->list[i][noeuds] = k;
157                 G->list[noeuds][i] = k;
158             }
159         }
160
161         // pour les arc vers le tier current
162         for(j = 0; G->I.compteur[i][tier2] < G->I.proba[i][0] && j < G->I.proba[i][0]; j++)
163         {
164             if(test_noeuds_max(G, i, noeudsMaxTier2, tier2))
165             {
166                 noeuds = calcul_noeuds(G, i, debTier2, finTier2, noeudsMaxTier2,

```

```

        tier2, tier2);
169     k = rand()%(poidsMaxTier2 - poidsMinTier2 + 1) + poidsMinTier2;
170     if(noeuds != -1)
171     {
172         G->list[i][noeuds] = k;
173         G->list[noeuds][i] = k;
174
175         G->I.compteur[i][1]++;
176         G->I.compteur[noeuds][1]++;
177     }
178 }
179 }
180 }
181 }
182
183 void calculTier3(graphe* G){
184
185     int i, j, k;
186     int p, noeuds;
187
188     for(i = debTier3; i < finTier3; i++)
189     {
190         // pour les arc vers le tier precedent
191         p = 2;
192         for(j = 0; j < p; j++)
193         {
194             noeuds = calcul_noeuds(G, i, debTier2, finTier2, 100, tier3, tier2);
195             k = rand()%(poidsMaxTier3 - poidsMinTier3 + 1) + poidsMinTier3;
196
197             if(noeuds != -1)
198             {
199                 G->list[i][noeuds] = k;
200                 G->I.compteur[i][tier2]++;
201
202                 G->list[noeuds][i] = k;
203                 G->I.compteur[noeuds][tier3]++;
204             }
205         }
206
207         // pour les arc vers le tier current
208         for(j = 0; G->I.compteur[i][tier3] < G->I.proba[i][0] && j < 2)
209             G->I.proba[i][0]; j++)
210         {
211             if(test_noeuds_max(G, i, noeudsMaxTier3, tier3))
212             {
213                 noeuds = calcul_noeuds(G, i, debTier3, finTier3, noeudsMaxTier3,
214                                         tier3, tier3);
215                 k = rand()%(poidsMaxTier3 - poidsMinTier3 + 1) + poidsMinTier3;
216
217                 if(noeuds != -1)
218                 {
219                     G->list[i][noeuds] = k;
220                     G->I.compteur[i][tier3]++;
221
222                     G->list[noeuds][i] = k;
223                     G->I.compteur[noeuds][tier3]++;
224                 }
225             }
226         }
227     }
228 }

```

```
223         }
224     }
225 }
226 }
227
```

### 4.3 Vérification de la connexité

Fichiers pour vérifier la connexité du graphe :

```
1  /**
2   * Algorithme de recherche en profondeur
3   * Les algorithmes implémentés permettent de vérifier la connexité du
4   * graphe en appliquant l'algorithme de recherche en profondeur
5   * */
6
7  /**
8   * Parcours du graphe par chaque sommet
9   * @param Graphe
10  * @param entier
11  * @param couleur
12  * @param pere
13  * */
14  void parcours_sommet(graphe* G, int s, int *couleur, int *pere);
15
16  /**
17   * Parcours le graphe en profondeur pour savoir s'il est connexe
18   * @param graphe
19   * @return si connexe
20   * */
21  int parcours_graphe(graphe* G);
22
```

```
1  #include "const.h"
2  #include "graph.h"
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  void parcours_sommet(graphe* G, int s, int *couleur, int *pere){
8      int i;
9      couleur[s] = 1;
10     for(i = 0; i < TAILLE_GRAPHE ; i++)
11         if(i != s && couleur[i] == 0 && G->list[s][i])
12             {
13                 pere[i] = s;
14                 parcours_sommet(G, i, couleur, pere);
15             }
16     couleur[s] = 2;
17 }
18
19 int parcours_graphe(graphe* G){
20     int i;
21     int *couleur = calloc(TAILLE_GRAPHE, sizeof(int)); // 0 est blanc, 1 gris et 2 noir
22
23     int *pere = malloc(sizeof(int)*TAILLE_GRAPHE);
24     for(i = 0; i < TAILLE_GRAPHE ; i++)
25         pere[i] = TAILLE_GRAPHE;
26
27     int compteur = 0;
28     for(i = 0; i < TAILLE_GRAPHE ; i++)
29         if(couleur[i] == 0)
30             {
31                 parcours_sommet(G, i, couleur, pere);
32                 compteur++;
33             }
34
35     free(couleur);
36     free(pere);
37     return compteur;
38 }
39
```

## 4.4 Création de la table de routage

Fichiers pour la création de la table de routage :

```
1  #include "const.h"
2  /**
3   * Algorithme de recherche du plus court chemin dans une graphe
4   * */
5
6  /**
7   * Structure de donnée pour sauvegarder la table de root (2 matrices)
8   * */
9  typedef struct {
10     int succ[TAILLE_GRAPHE][TAILLE_GRAPHE];
11     int poids[TAILLE_GRAPHE][TAILLE_GRAPHE];
12 } routage;
13
14 /**
15  * Initialisation de la table de rootage
16  * @param graphe
17  * @param taille du graphe
18  * */
19 routage* init(graphe* G, int taille);
20
21 /**
22  * Application de l'algorithme Floyd Warshall
23  * @param routage table de rootage à remplir
24  * @param taille du graphe
25  * */
26 void Floyd_Warshall(routage* R, int taille);
27
28 /**
29  * Libération de mémoire de la table de rootage
30  * @param table de root
31  * */
32 void libere_routage(routage* R);
33
34 /**
35  * Affiche le plus court chemin entre une paire de noeuds
36  * @param table de root
37  * @param premier noeud de départ
38  * @param dernier noeud d'arrivé
39  * */
40 void afficher_chemin(routage* R, int deb, int fin);
41
```



```
1  #include "const.h"
2  #include "graph.h"
3  #include "routage.h"
4
5  #include <stdlib.h> // rand ()
6
7  #include <stdio.h> // printf ()
8
9  #define inf 9999
10
11
12  routage* init(graphe* G, int taille) {
13      int i, j;
14      routage* R = malloc(sizeof(routage));
15
16
17      for(i = 0; i < taille; ++i)
18      {
19          for(j = 0; j < taille; ++j)
20          {
21              R->poids[i][j] = G->list[i][j];
22
23              if(R->poids[i][j] == -1)
24              {
25                  R->poids[i][j] = inf;
26                  R->succ[i][j] = -1;
27              }
28              else
29              {
30                  R->succ[i][j] = j;
31              }
32          }
33      }
34
35      for(i = 0; i < taille; ++i)
36      {
37          R->poids[i][i] = 0;
38          R->succ[i][i] = i;
39      }
40      return R;
41  }
42
43  void Floyd_Warshall(routage* R, int taille) {
44      int i, j, k;
45
46      for(k = 0; k < taille; k++)
47      {
48          for(i = 0; i < taille; i++)
49          {
50              for(j = 0; j < taille; j++)
51              {
52                  if(R->poids[i][k] != inf && R->poids[k][j] != inf
53                     && (R->poids[i][j] > (R->poids[i][k] + R->poids[k][j])) )
54                  {
55                      R->poids[i][j] = R->poids[i][k] + R->poids[k][j];
56                      R->succ[i][j] = R->succ[i][k];
57                  }
58              }
59          }
60      }
61  }
```

```
58
59     }
60 }
61 }
62 }
63
64 void afficher_chemin(routage* R, int deb, int fin) {
65     int stock_deb = deb;
66     int voisin[TAILLE_GRAPHE] = {-1};
67
68     int i, suiv = R->succ[deb][fin];
69     for(i = 0; suiv != fin && i < TAILLE_GRAPHE; i++)
70     {
71         deb = R->succ[deb][fin];
72         voisin[i] = deb;
73         suiv = R->succ[deb][fin];
74     }
75
76     if(suiv == fin)
77     {
78         voisin[i] = fin;
79
80         printf("\nChemin de %d à %d :\n%d", stock_deb, fin, stock_deb);
81         for(int j = 0 ; j < i+1; j++)
82         {
83             printf(" -> %d", voisin[j]);
84         }
85         printf("\n");
86     }
87     else
88     {
89         printf("error\n");
90     }
91 }
92
93 void libere_routage(routage* R) { free(R); }
94
```

## 4.5 Affichage

Fichiers pour l’affichage graphe :

```
1  /**
2   * Gestion de l'Affichage sous forme de fenetre graphique
3   * */
4  #include "moteur_graphique.h"
5
6  /**
7   * Affichage des connexions entre les noeuds
8   * @param graphe
9   * @param objet
10  * @param cercle
11  * @param les noeuds
12  * @param le début du tier
13  * @param la fin du tier
14  * */
15  void afficher_voisin(graphe* G, flame_obj_t * fo, cercle_t * c,int sommet, int deb, int fin);
16
17  /**
18   * Sauvegarde des données dans le tableau cercle pour l'affichage
19   * Chaque tiers ont des couleurs différentes (tier 1 : rouge, tier2 :
20   * bleu, tier 3 : vert)
21   * @param cercle
22   * @param le début du tier
23   * @param la fin du tier
24   * @param x représente les coordonnées en abscisse
25   * @param y représente les coordonnées en ordonnée
26   * */
27  void init_affichage_tier(cercle_t * c, int debut,int fin, int tier,int * x,int * y);
28
29  /**
30   * Permet l'initialisation des objets graphiques:
31   * initialisation des cercles et des connexions des noeuds*
32   * @param graphe
33   * @param objet
34   * @param cercle
35   * */
36  void initialisation_objets_graphique(graphe *G,flame_obj_t * fo,cercle_t * c);
37
38  /**
39   * Recherche l'identifiant du cercle dans la fenetre graphique
40   * @param x abscisse
41   * @param y ordonnée
42   * @return identifiant du cercle
43   * */
44  int trouve_id(int x,int y);
45
46  /**
47   * Affiche le plus court chemin
48   * @param objet
49   * @param la table de routage
50   * @param cercle
51   * @param l'identifiant de l'expediteur
52   * @param l'identifiant du destinataire
53   * @param la couleur
54   * */
55  void affiche_chemin(flame_obj_t * fo,routage* R, cercle_t * c,int deb, int fin,enum couleur coul);
```

```
56
57 /**
58  * Permet la gestion de l'interaction utilisateur et machine
59  * @param graphe
60  * @param objet
61  * @param la table de routage
62  * @param cercle
63  * */
64 void interaction_user(graphe * G,flame_obj_t * fo,routage * R,cercle_t * c);
65
66 /**
67  * Fonction principale
68  * Gestion de la fenetre graphique
69  * @param graphe
70  * @param la table de routage
71  * */
72 void gestion_fenetre_graphique(graphe* G, routage *R);
73
```

```

1  #include <stdio.h>
2
3  //~ local
4  #include "const.h"
5  #include "graph.h"
6  #include "moteur_graphique.h"
7  #include "routage.h"
8
9  void afficher_voisin(graphe* G, flame_obj_t * fo, cercle_t * c,int sommet, int deb, int fin)
10 {
11     int j;
12     for(j = deb; j < fin; j++) if(G->list[sommet][j] != -1) {
13         afficher_connexion(fo,c,sommet,j,GRIS); }
14 }
15
16 void init_affichage_tier(cercle_t * c, int debut,int fin, int tier,int * x,int * y)
17 {
18     int i;
19
20     for( i = debut ; i < fin ; i ++ )
21     {
22         c[i].rad = TAILLE_CERCLE;
23         if(tier == tier1)
24         {
25             colorer_cercle(&c[i],ROUGE);
26             c[i].pos_x = *x * ( 3 * TAILLE_CERCLE);
27             c[i].pos_y = *y * ( 3 * TAILLE_CERCLE);
28
29         }
30         if(tier == tier2)
31         {
32             colorer_cercle(&c[i],BLEU);
33             c[i].pos_x = *x * ( 3 * TAILLE_CERCLE);
34             c[i].pos_y = *y * ( 3 * TAILLE_CERCLE);
35         }
36         if(tier == tier3)
37         {
38             colorer_cercle(&c[i],VERT);
39             c[i].pos_x = *x * ( 3 * TAILLE_CERCLE);
40             c[i].pos_y = *y * ( 3 * TAILLE_CERCLE);
41         }
42         *x+=1;
43         if(*x > 10)
44         {
45             *y += 1;
46             *x = 1;
47         }
48     }
49 }
50
51 void initialisation_objets_graphique(graphe *G,flame_obj_t * fo,cercle_t * c)
52 {
53     int noeud;
54
55     int x = 1;

```

```

56     int y = 1;
57
58     // > Initialise les coordonnées des cercles
59     init_affichage_tier(c,debTier1,finTier1,tier1,&x,&y);
60     init_affichage_tier(c,debTier2,finTier2,tier2,&x,&y);
61     init_affichage_tier(c,debTier3,finTier3,tier3,&x,&y);
62
63     for (noeud = 0; noeud < TAILLE_GRAPHE; noeud++)
64     {
65         afficher_cercle(fo, &c[noeud]);
66         // > Affichage des connexions mais c'est illisible
67         //~ afficher_voisin(G, fo, c, noeud, debTier1, finTier1);
68         //~ afficher_voisin(G, fo, c, noeud, debTier2, finTier2);
69         //~ afficher_voisin(G, fo, c, noeud, debTier3, finTier3);
70     }
71 }
72
73 int trouve_id(int x,int y)
74 {
75     x -= (2 * TAILLE_CERCLE);
76     y -= (2 * TAILLE_CERCLE);
77     return ((y / (3 * TAILLE_CERCLE))*10) + (x / (3*TAILLE_CERCLE));
78 }
79
80 void affiche_chemin (flame_obj_t * fo,routage* R, cercle_t * c,int deb, int
fin,enum couleur coul)
81 {
82     int voisin[TAILLE_GRAPHE] = {-1};
83
84     int i = 0, suiv = R->succ[deb][fin];
85     voisin[i] = deb;
86
87     for(i = 1; suiv != fin && i < TAILLE_GRAPHE; i++)
88     {
89         deb = R->succ[deb][fin];
90         voisin[i] = deb;
91         suiv = R->succ[deb][fin];
92     }
93
94     if(suiv == fin)
95     {
96         voisin[i] = fin;
97
98         int d, e, j;
99         for(j = 0 ; j < i; j++)
100         {
101             d = voisin[j]; e = voisin[j+1];
102             afficher_connexion(fo, c, d, e, coul);
103         }
104     }
105     else
106     {
107         printf("error\n");
108     }
109 }
110
111

```

```
112 void interaction_user(graphe * G, flame_obj_t * fo, routage * R, cercle_t * c)
113 {
114     XEvent event;
115     int cmp = 0;
116     int id_1 = 0;
117     int id_2 = 0;
118     int save_id_1 = 0;
119     int save_id_2 = 0;
120
121     int click_x, click_y;
122
123     while (1)
124     {
125         if (XPending(fo->display) > 0)
126         {
127             XNextEvent(fo->display, &event);
128             if(recupere_clavier(event) == 'q') { break; }
129             if (event.type == ButtonPress)
130             {
131                 // Récupere les coordonnées
132                 click_x = event.xkey.x;
133                 click_y = event.xkey.y;
134
135                 // colorie et affiche les cercles
136                 colorer_cercle(&c[trouve_id(click_x,click_y)],JAUNE);
137                 afficher_cercle(fo,&c[trouve_id(click_x,click_y)]);
138
139                 if(cmp == 0)
140                 {
141                     save_id_1 = id_1;
142                     save_id_2 = id_2;
143                     id_1 = trouve_id(click_x,click_y);
144
145                     // Permet d'effacer les traits
146                     affiche_chemin ( fo, R, c, save_id_1, save_id_2, NOIR);
147                     affiche_croix(fo, c[save_id_2].pos_x, c[save_id_2].pos_y, NOIR);
148                     initialisation_objets_graphique ( G, fo, c);
149
150                     // Coloris les cercles
151                     colorer_cercle( &c[id_1], JAUNE);
152
153                     // Affiche les cercles
154                     afficher_cercle(fo, &c[id_1]);
155                 }
156                 else
157                 {
158                     id_2 = trouve_id(click_x,click_y);
159
160                     // Coloris les cercles
161                     colorer_cercle( &c[id_2], JAUNE);
162
163                     // Affiche les cercles
164                     afficher_cercle(fo, &c[id_2]);
165                 }
166                 cmp ++;
167             }
168         }
```



```
169     else
170     {
171         if(cmp == 2)
172         {
173             cmp = 0;
174
175             // Affiche les chemins
176             afficher_chemin (R, id_1, id_2);
177             affiche_chemin (fo, R, c, id_1, id_2, JAUNE);
178
179             // Affiche debut
180
181             // Affiche arrivé
182             affiche_croix(fo, c[id_2].pos_x, c[id_2].pos_y, BLANC);
183         }
184     }
185 }
186 }
187
188 void gestion_fenetre_graphique(graphe* G, routage *R)
189 {
190     // > Initialisation du canvas:
191     flame_obj_t * fo = init_canvas();
192
193     // > Allocation de mémoire de la structure de donnée d'un cercle
194     cercle_t c[TAILLE_GRAPHE];
195
196     // > Initialise les connexions et les cercles
197     initialisation_objets_graphique(G,fo,c);
198
199     // > Gestion des interaction utilisateur machine
200     interaction_user(G,fo,R,c);
201
202     // > Fermeture du canvas
203     flame_close(fo);
204 }
205
```

```
1  #include "flame.h"
2
3  #define TAILLE_ECRAN_HAUTEUR 500
4  #define TAILLE_ECRAN_LARGEUR 500
5
6  #define TAILLE_CERCLE 15
7
8  /* Définir la couleur 8 bits */
9  typedef unsigned char byte;
10
11 /* Structure de données d'un cercle */
12
13 typedef struct cercle_s {
14
15     double pos_x;
16     double pos_y;
17
18     double rad;
19
20     //Couleur du cercle
21     byte r;
22     byte g;
23     byte b;
24
25 } cercle_t;
26
27 /* Enumération des couleurs */
28 enum couleur{
29     ROUGE,
30     VERT,
31     BLEU,
32     BLANC,
33     JAUNE,
34     GRIS,
35     NOIR
36 };
37
38 /* Permet d'afficher un cercle rempli*/
39 void remplir_cercle(flame_obj_t *fo, cercle_t *b);
40
41 /* Permet d'afficher un cercle */
42 void afficher_cercle(flame_obj_t *fo, cercle_t *b);
43
44 /* Permet d'afficher les connexions */
45 void afficher_ligne(flame_obj_t *fo,int x1,int y1,int x2,int y2);
46
47 /* Permet de récupérer les entrées du clavier */
48 char recupere_clavier(XEvent event);
49
50 /* Colorier un cercle */
51 void colorer_cercle(cercle_t * c,enum couleur coul);
52
53 /* Initialise la fenetre graphique */
54 flame_obj_t * init_canvas();
55
56 /* Permet d'afficher les connexion entre les objets symbolisé par un trait */
57 void afficher_connexion(flame_obj_t *fo,cercle_t * c,int id_1,int id_2,enum
```

```
58     couleur coul);
59     /* Permet l'affichage d'une croix */
60     void affiche_croix(flame_obj_t *fo, int x, int y, enum couleur coul);
61
```

```
1  /**
2   * Bibliothèque de gestion d'un graphe:
3   * Cette bibliothèque permet de dessiner des cercles, des lignes et des croix.
4   * Elle a été créer afin de manipuler des graphes simplement.
5   * */
6
7  #include <time.h>
8  #include <math.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <stdint.h>
13 #include <unistd.h>
14 #include <inttypes.h>
15
16 #include "flame.h"
17 #include "moteur_graphique.h"
18
19 #include "pi.h"
20
21 #define A_STEP 0.01
22
23
24 void remplir_cercle(flame_obj_t *fo, cercle_t *b)
25 {
26     flame_set_color(fo, b->r, b->g,b->b);
27     for (double angle = 0.0; angle < 2 * PI; angle += A_STEP)
28     {
29         flame_draw_line(fo, b->pos_x, b->pos_y, b->pos_x + b->rad * cos(angle),
30             b->pos_y + b->rad * sin(angle));
31     }
32 }
33
34 void afficher_cercle(flame_obj_t *fo, cercle_t *b)
35 {
36     flame_set_color(fo, b->r, b->g,b->b);
37     for (double angle = 0.0; angle < 2 * PI; angle += A_STEP)
38     {
39         flame_draw_point(fo, b->pos_x + b->rad * cos(angle), b->pos_y + b->rad *
40             sin(angle));
41     }
42 }
43
44 void afficher_ligne(flame_obj_t *fo,int x1,int y1,int x2,int y2)
45 {
46     flame_draw_line(fo, x1, y1,x2,y2);
47 }
48
49 char recupere_clavier(XEvent event)
50 {
51     char c;
52     if (event.type == KeyPress)
53     {
54         c = XLookupKeysym(&event.xkey, 0);
55         return c;
56     }
57     return -1;
58 }
```

```
56 }
57
58 flame_obj_t * init_canvas()
59 {
60     return flame_open("Graphe",TAILLE_ECRAN_HAUTEUR,TAILLE_ECRAN_LARGEUR);
61 }
62
63 void colorer_cercle(cercle_t * c,enum couleur coul)
64 {
65     switch(coul)
66     {
67         case ROUGE :
68         {
69             c->r = 255;
70             c->g = 0;
71             c->b = 0;
72             break;
73         }
74         case BLEU :
75         {
76             c->r = 0;
77             c->g = 0;
78             c->b = 255;
79             break;
80         }
81         case VERT :
82         {
83             c->r = 0;
84             c->g = 255;
85             c->b = 0;
86             break;
87         }
88         case BLANC :
89         {
90             c->r = 255;
91             c->g = 255;
92             c->b = 255;
93             break;
94         }
95         case JAUNE :
96         {
97             c->r = 255;
98             c->g = 255;
99             c->b = 0;
100             break;
101         }
102         default:
103         {
104             perror("switch");
105             exit(EXIT_FAILURE);
106         }
107     }
108 }
109
110 void afficher_connexion(flame_obj_t *fo, cercle_t * c, int id_1, int id_2, enum
couleur coul)
111 {
```

```
112     if(coul == BLANC) flame_set_color(fo, 255, 255, 255);
113     if(coul == ROUGE) flame_set_color(fo, 255, 0, 0);
114     if(coul == GRIS) flame_set_color(fo, 200, 200, 200);
115     if(coul == NOIR) flame_set_color(fo, 0, 0, 0);
116     if(coul == JAUNE) flame_set_color(fo, 255, 255, 0);
117     afficher_ligne(fo, c[id_1].pos_x, c[id_1].pos_y, c[id_2].pos_x, c[id_2].pos_y);
118 }
119
120 void affiche_croix(flame_obj_t *fo, int x, int y, enum couleur coul) {
121     if(coul == BLANC) flame_set_color(fo, 255, 255, 255);
122     if(coul == NOIR) flame_set_color(fo, 0, 0, 0);
123
124     afficher_ligne(fo, x - 10, y - 10, x + 10, y + 10);
125     afficher_ligne(fo, x - 10, y + 10, x + 10, y - 10);
126 }
127
```

## **4.6 Les autres fichiers**

### **4.6.1 Constantes**

Fichier pour les définitions des constantes :

```
1  /**
2   * Ensemble de définition des constantes
3   * */
4
5  #define TAILLE_GRAPHE 100
6  #define tier1 0
7  #define tier2 1
8  #define tier3 2
9
10 #define nbTier1 8
11 #define nbTier2 20
12 #define nbTier3 72
13 #define ZERO 0
14
15 #define debTier1 0
16 #define finTier1 8
17 #define debTier2 8
18 #define finTier2 28
19 #define debTier3 28
20 #define finTier3 100
21
22 #define poidsMaxTier1 10
23 #define poidsMinTier1 5
24 #define poidsMaxTier2 20
25 #define poidsMinTier2 10
26 #define poidsMaxTier3 50
27 #define poidsMinTier3 15
28
29 #define noeudsMaxTier2 3
30 #define noeudsMaxTier3 1
31
```



### 4.6.2 Bibliothèque flame11

Code source de la bibliothèque flame11 crée par Yaspr, voici le lien github :

<https://github.com/yaspr/flame11>

```
1  #ifndef FLAME_H
2  #define FLAME_H
3
4  #include <unistd.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <X11/Xlib.h>
8
9  //
10 typedef struct flame_obj_s {
11
12     GC      gc;
13     Window  window;
14     Display *display;
15     Colormap colormap;
16     int     fast_color_mode;
17
18 } flame_obj_t;
19
20 int flame_close(flame_obj_t *fo);
21 int flame_event_waiting(flame_obj_t *fo);
22 void flame_flush(flame_obj_t *fo);
23 void flame_clear_display(flame_obj_t *fo);
24 void flame_draw_point(flame_obj_t *fo, int x, int y);
25 flame_obj_t *flame_open(char *title, int width, int height);
26 char flame_wait(flame_obj_t *fo, int *click_x, int *click_y);
27 void flame_set_color(flame_obj_t *fo, int red, int green, int blue );
28 void flame_draw_line(flame_obj_t *fo, int x1, int y1, int x2, int y2);
29 void flame_clear_color(flame_obj_t *fo, int red, int green, int blue );
30
31 #endif
32
```

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <X11/Xlib.h>
5
6  #include "flame.h"
7
8  //
9  int flame_flush_display(flame_obj_t *fo)
10 {
11     if (fo && fo->display)
12         return XFlush(fo->display), 1;
13     else
14         return 0;
15 }
16
17 //
18 int flame_close(flame_obj_t *fo)
19 {
20     if (fo)
21     {
22         XFreeGC(fo->display, fo->gc);
23         XCloseDisplay(fo->display);
24
25         free(fo);
26
27         return 1;
28     }
29     else
30         return 0;
31 }
32
33 //
34 flame_obj_t *flame_open(char *title, int width, int height)
35 {
36     XEvent e;
37     int vb = 0;
38     Visual *visual;
39     int blackColor, whiteColor;
40     XSetWindowAttributes attrib;
41     flame_obj_t *fo = malloc(sizeof(flame_obj_t));
42
43     fo->display = XOpenDisplay(0);
44
45     if (!fo->display)
46     {
47         fprintf(stderr, "flame_open: unable to open the graphics window.\n");
48         exit(1);
49     }
50
51     visual = DefaultVisual(fo->display, 0);
52
53     fo->fast_color_mode = (visual && visual->class == TrueColor);
54
55     blackColor = BlackPixel(fo->display, DefaultScreen(fo->display));
56     whiteColor = WhitePixel(fo->display, DefaultScreen(fo->display));
57 }
```

```

58 //fo->window = XCreateSimpleWindow(fo->display, DefaultRootWindow(fo->display),
    0, 0, width, height, 0, blackColor, blackColor);
59 fo->window = XCreateSimpleWindow(fo->display, DefaultRootWindow(fo->display), 0,
    0, width, height, 0, blackColor, blackColor);
60
61 attrib.backing_store = Always;
62
63 XChangeWindowAttributes(fo->display, fo->window, CWBackingStore, &attrib);
64
65 XStoreName(fo->display, fo->window, title);
66
67 XSelectInput(fo->display, fo->window, StructureNotifyMask | ExposureMask |
    KeyPressMask | ButtonPressMask | ButtonReleaseMask | PointerMotionMask);
68
69 XMapWindow(fo->display, fo->window);
70
71 fo->gc = XCreateGC(fo->display, fo->window, 0, 0);
72
73 fo->colormap = DefaultColormap(fo->display, 0);
74
75 XSetForeground(fo->display, fo->gc, whiteColor);
76
77 while(!vb)
78 {
79     XNextEvent(fo->display, &e);
80
81     vb = (e.type == MapNotify);
82 }
83
84 return fo;
85 }
86
87 //
88 void flame_draw_point(flame_obj_t *fo, int x, int y)
89 { XDrawPoint(fo->display, fo->window, fo->gc, x, y); }
90
91 //
92 void flame_draw_line(flame_obj_t *fo, int x1, int y1, int x2, int y2)
93 { XDrawLine(fo->display, fo->window, fo->gc, x1, y1, x2, y2); }
94
95 //
96 void flame_set_color(flame_obj_t *fo, int r, int g, int b)
97 {
98     XColor color;
99
100     if (fo->fast_color_mode)
101         color.pixel = ((b & 0xff) | ((g & 0xff) << 8) | ((r & 0xff) << 16));
102     else
103     {
104         color.pixel = 0;
105         color.red = r << 8;
106         color.green = g << 8;
107         color.blue = b << 8;
108         XAllocColor(fo->display, fo->colormap, &color);
109     }
110
111     XSetForeground(fo->display, fo->gc, color.pixel);

```

```
112 }
113
114
115 //
116 void flame_clear_display(flame_obj_t *fo)
117 { XClearWindow(fo->display, fo->>window); }
118
119 //
120 void flame_clear_color(flame_obj_t *fo, int r, int g, int b )
121 {
122     XColor color;
123     XSetWindowAttributes attrib;
124
125     color.pixel = 0;
126     color.red   = r << 8;
127     color.green = g << 8;
128     color.blue  = b << 8;
129     XAllocColor(fo->display, fo->colormap, &color);
130
131     attrib.background_pixel = color.pixel;
132     XChangeWindowAttributes(fo->display, fo->>window, CWBackPixel,&attrib);
133 }
134
135 //
136 int flame_event_waiting(flame_obj_t *fo)
137 {
138     XEvent event;
139
140     flame_flush_display(fo);
141
142     while (1)
143     {
144         if (XCheckMaskEvent(fo->display, -1, &event))
145         {
146             if (event.type == KeyPress)
147             {
148                 XPutBackEvent(fo->display, &event);
149
150                 return 1;
151             }
152             else
153             if (event.type == ButtonPress)
154             {
155                 XPutBackEvent(fo->display, &event);
156
157                 return 1;
158             }
159             else
160                 return 0;
161         }
162         else
163             return 0;
164     }
165 }
166
167 //
168 char flame_wait(flame_obj_t *fo, int *click_x, int *click_y)
```

```
169 {
170     XEvent event;
171
172     flame_flush_display(fo);
173
174     while (1)
175     {
176         if (XPending(fo->display) > 0)
177         {
178             XNextEvent(fo->display, &event);
179
180             if (event.type == KeyPress)
181             {
182                 /* printf("%c\n", XLookupKeysym(&event.xkey, 0)); */
183                 return XLookupKeysym(&event.xkey, 0);
184             }
185             else
186             if (event.type == ButtonPress)
187             {
188                 *click_x = event.xkey.x;
189                 *click_y = event.xkey.y;
190
191                 //Left click == 1, Right click == 3
192                 return event.xbutton.button;
193             }
194         }
195         else
196         {
197             ;
198         }
199     }
200 }
201
202
```

```
1  #ifndef PI_H
2  #define PI_H
3
4  #define PI 3.14159265359
5  #define PI_2 PI / 2.0
6  #define PI_3 PI / 3.0
7
8  #endif
9
```