Compte rendu TP1

Martin Ducoulombier

Développement des fonctions pour les graphes

Question 1 : Faites les déclarations de types nécessaires. Collez le code C de la déclaration dans le compte-rendu

```c
// représentation sous forme matrice d'adjacence
typedef struct
{
   unsigned int nbSommets;
   int ** matrice;
}T_graphMD;
```

Question 2 : Développez la fonction T_grapheMD *newGraphMD(int c); qui crée un nouveau graphe. Collez le code C de la fonction dans le compte-rendu

```c
T_graphMD * newGraphMD(int n) {

   // A compléter
   T_graphMD * g;
   int i, j;
   g = (T_graphMD *) malloc( sizeof(T_graphMD) );
   g->nbSommets = n;
   g->matrice = (int **)malloc(n * sizeof(int *));
   for (i = 0; i < n; i++) g->matrice[i] = (int *)malloc(n *
sizeof(int));
   for (i = 0; i < n; i++)
       for (j = 0; j < n; j++)
           g->matrice[i][j] = INT_MAX;
   for (i = 0; i < n; i++)
       g->matrice[i][i] = 0;
   return g;
}
```

Question 3 : Construire le graphe ci-dessous. Afficher ce graphe sur la sortie standard en format : sommet->(poids)->sommet. Collez le code C de la fonction, ainsi que du résultat obtenu dans le compte-rendu

```
T_graphMD * g = newGraphMD(5);
   addEdgeMD(g, 0, 1, 10);
   addEdgeMD(g, 0, 4, 5);

   addEdgeMD(g, 1, 2, 1);
   addEdgeMD(g, 1, 4, 2);

   addEdgeMD(g, 2, 3, 4);

   addEdgeMD(g, 3, 0, 7);
   addEdgeMD(g, 3, 2, 6);

   addEdgeMD(g, 4, 1, 3);
   addEdgeMD(g, 4, 2, 9);
   addEdgeMD(g, 4, 3, 2);

Affichage sur la console du graphe créé

      0 -> ( 10 ) -> 1
      0 -> ( 5 ) -> 4
      1 -> ( 1 ) -> 2
      1 -> ( 2 ) -> 4
      2 -> ( 4 ) -> 3
      3 -> ( 7 ) -> 0
      3 -> ( 6 ) -> 2
      4 -> ( 3 ) -> 1
      4 -> ( 9 ) -> 2
      4 -> ( 2 ) -> 3
```

Question 4 : Développez la fonction void dumpGraphe(FILE *fP, T_grapheMD *g); qui permet d'afficher la structure du graphe dans un fichier (FILE *), cette fonction servira à générer le .dot. Collez le code C de la fonction, ainsi que du résultat obtenu dans le compte-rendu

```c
void dumpGraphMD(FILE * fp, T_graphMD * g) {
   // Affiche la structure du graphe dans un FILE *
   // Tester avec stdout
   for (int i = 0; i < g->nbSommets; i++) {
       for (int j = 0; j < g->nbSommets; j++) {
           if (g->matrice[i][j] != INT_MAX && g->matrice[i][j] != 0) {
               fprintf(fp, "%d -> %d [label = \"%d\"]\n", i, j,
g->matrice[i][j]);
           }
       }
   }
}
```

Question 5 : Développez la fonction void createDotPOT(T_graphMD *g, const char *basename); qui permet de produire des image png avec graphiz, elle fera appelle à la fonction dumpGraphe.
NB : Pour cela vous pouvez vous inspirez de la fonction createDotPOT utilisée lors des TP 4 et 5, voir également la page 17 du slide du cours sur les graphes pour la partie .dot.
Collez le code C de la fonction, ainsi que du résultat obtenu dans le compte-rendu

```c
void createDotPOT(T_graphMD * g,  const char *basename) {
   static char oldBasename[FILENAME_MAX + 1] = "";
   static unsigned int noVersion = 0;

   char DOSSIER_DOT[FILENAME_MAX + 1];
   char DOSSIER_PNG[FILENAME_MAX + 1];

   char fnameDot [FILENAME_MAX + 1];
   char fnamePng [FILENAME_MAX + 1];
   char    cmdLine [2 * FILENAME_MAX + 20];
   FILE *fp;
   struct stat sb;


   // Au premier appel, création (si nécessaire) des répertoires
   // où seront rangés les fichiers .dot et .png générés par cette
fonction
```

```
   // il faut créer le répertoire outputPath s'il n'existe pas
   if (stat(outputPath, &sb) == 0 && S_ISDIR(sb.st_mode)) {
   } else {
       printf("Création du répertoire %s\n", outputPath);
       mkdir(outputPath, 0777);
   }

   // il faut créer les répertoires outputPath/png et /dot
   sprintf(DOSSIER_DOT, "%s/dot/",outputPath);
   sprintf(DOSSIER_PNG, "%s/png/",outputPath);

   if (oldBasename[0] == '\0') {
       mkdir(DOSSIER_DOT,  S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH |
S_IXOTH);
       mkdir(DOSSIER_PNG,  S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH |
S_IXOTH);
   }

    // S'il y a changement de nom de base alors recommencer à zéro
    // la numérotation des fichiers

   if (strcmp(oldBasename, basename) != 0) {
       noVersion = 0;
       strcpy(oldBasename, basename);
   }

   sprintf(fnameDot, "%s%s_v%02u.dot", DOSSIER_DOT, basename,
noVersion);
   sprintf(fnamePng, "%s%s_v%02u.png", DOSSIER_PNG, basename,
noVersion);

   CHECK_IF(fp = fopen(fnameDot, "w"), NULL, "erreur fopen dans
saveDotBST");

   noVersion ++;

   // Ecrire ici l'entête pour le fichier Graphiz comme indiqué dans la
page 17 du cours

   fprintf(fp, "digraph %s {\n", basename);
   fprintf(fp,
   "\trankdir = \"LR\" \n"
   "\tnode [\n"
```

```
        "\t\tfontname = \"Arial\" \n"
        "\t\tstyle = \"filled\"\n"
        "\t\tshape = \"circle\"\n"
        "\t\tcolor = \"lightblue\"\n"
        "\t];\n"
    "\n"
    "\tedge [\n"
        "\t\tcolor = \"red\"\n"
    "\t];\n\n"
    );


    dumpGraphMD(fp, g);
    // et appel de la fonction dumpGraphMD


    fprintf(fp, "}\n");

    fclose(fp);

    sprintf(cmdLine, "dot -Tpng  %s -o %s", fnameDot, fnamePng);
    system(cmdLine);

    printf("Creation de '%s' et '%s' ... effectuee\n", fnameDot,
fnamePng);
}
```

Question 6 :

Développez la fonction bellmanFord dont les paramètres sont le graphe et le numéro du sommet d'origine, cette fonction devrait retourner un tableau contenant les plus courtes distances du sommet s, ainsi qu'un tableau permettant de connaître les prédécesseurs des sommets sur les plus courts chemins
NB : construire un typedef struct contenant deux pointeurs, l'un pour les distances et l'autre pour les prédécesseurs. Ce typedef sera le type de la fonction bellmanFord.

L'algorithme de BelmanFord est présenté à la page 47 du cours.
Dans la fonction bellmanFord faire une allocation dynamique pour ces deux pointeurs.
Testez votre programme pour le graphe de la question 3 en prenant comme origine, le sommet 1 et en affichant les distances minimales depuis ce sommet aux autres ainsi que le sommet prédécesseur comme indiqué ci-dessous.
Collez le code C de la fonction, ainsi que du résultat obtenu dans le compte-rendu.

```c
typedef struct
{
   int *d;
   int *p;
} T_dp;



T_dp bellmanFord(T_graphMD * g, int s) {
   T_dp dp;
   int i;
   int j;
   int k;
   dp.d = (int *)malloc(g->nbSommets * sizeof(int));
   dp.p = (int *)malloc(g->nbSommets * sizeof(int));
   for (i = 0; i < g->nbSommets; i++)
   {
       dp.d[i] = INT_MAX;
       dp.p[s] = s;

   }
   dp.d[s] = 0;

   for (k = 1; k < g->nbSommets; k++)
   {
       for (i = 0; i < g->nbSommets; i++)
       {
           for (j = 0; j < g->nbSommets; j++)
           {
               if (g->matrice[i][j] != INT_MAX && dp.d[i] != INT_MAX &&
dp.d[j] > dp.d[i] + g->matrice[i][j])
               {
                   dp.d[j] = dp.d[i] + g->matrice[i][j];
                   dp.p[j] = i;
               }
           }
       }
   }
   return dp;
}

void affiche_ppcBellman(T_graphMD * g, T_dp dp, int s) {
```

```
    printf("Plus court chemin depuis %d : \n", s);
    for (int i = 0; i < g->nbSommets; i++) {
        printf("d[%d] = %d (s%d)\n", i, dp.d[i], dp.p[i]);
    }
}
```

Question 7 : Développez une fonction d'affichage qui aura comme paramètres, le graphe ainsi que la structure que vous avez créé dans la question précédente pour faire un affichage sous forme de tableau de résultats comme indiqué ci-dessous

Collez le code C de la fonction, ainsi que du résultat obtenu dans le compte-rendu

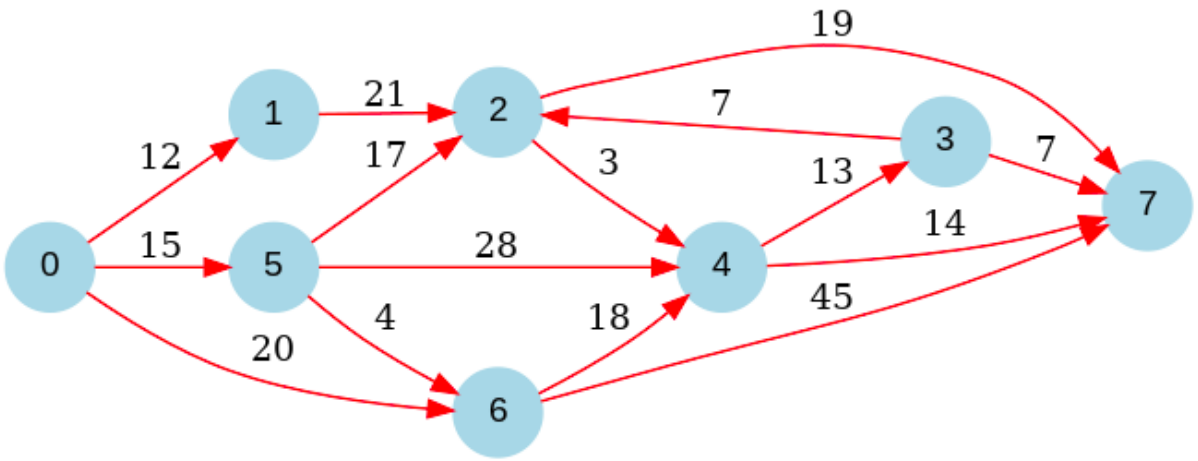
```
void affiche_ppcBellman2(T_graphMD * g, T_dp dp, int s ) {
    char * chemin;
    printf("Plus court chemins (Bellman) depuis le sommet %d : \n", s);
    printf("+-------------+------+--------------------------------+
\n");
    printf("| org -> dest | dist | chemin                         |
\n");
    printf("+-------------+------+--------------------------------+
\n");
    for (int i = 0; i < g->nbSommets; i++) {
        if (dp.d[i] == INT_MAX)
        {
            printf("| %3d -> %3d  | %4s | %s", s, i, "inf", "pas de
chemin                  ");
        }
        else
        {
            printf("| %3d -> %3d  | %4d | ", s, i, dp.d[i]);
            chemin = getChemin(dp, s, i);
            printf("%s", chemin);
            for (int i = 0; i < 30 - strlen(chemin); i++)
            {
                printf(" ");
            }
        }

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

```
    printf("+-------------+------+--------------------------------+
\n");
}

void afficherChemin(T_dp dp, int s, int d) {
    while (d != s)
    {
        printf("%d ", d);
        d = dp.p[d];
    }
    printf("%d ", s);
}

char * getChemin(T_dp dp, int s, int d) {
    static char chemin[100];
    char tmp[100];
    chemin[0] = '\0';
    tmp[0] = '\0';

    while (d != s)
    {
        sprintf(tmp, "%d ", d);
        strcat(tmp, chemin);
        strcpy(chemin, tmp);
        d = dp.p[d];
    }
    sprintf(tmp, "%d ", d);
    strcat(tmp, chemin);
    strcpy(chemin, tmp);

    return chemin;

}
```

Question 8 : Testez vos fonctions de bellmanFord et affichage pour le graphe suivant. Collez le résultat obtenu dans le compte-rendu

```
T_graphMD * g = newGraphMD(8);
   addEdgeMD(g, 0, 1, 12);
   addEdgeMD(g, 0, 5, 15);
   addEdgeMD(g, 0, 6, 20);

   addEdgeMD(g, 1, 2, 21);

   addEdgeMD(g, 2, 4, 3);
   addEdgeMD(g, 2, 7, 19);

   addEdgeMD(g, 3, 2, 7);
   addEdgeMD(g, 3, 7, 7);

   addEdgeMD(g, 4, 3, 13);
   addEdgeMD(g, 4, 7, 14);

   addEdgeMD(g, 5, 2, 17);
   addEdgeMD(g, 5, 4, 28);
   addEdgeMD(g, 5, 6, 4);

   addEdgeMD(g, 6, 4, 18);
   addEdgeMD(g, 6, 7, 45);
```

On obtient le graphe suivant avec la fonction createDotPOT :



On obtient le résultat suivant apres utilisation de bellman Ford et de l’affichage :

```
Plus court chemins (Bellman) depuis le sommet 0 :
+-------------+------+--------------------------------+
| org -> dest | dist | chemin                         |
+-------------+------+--------------------------------+
|   0 ->   0  |  0 | 0                                 |
|   0 ->   1  |   12 | 0 1                               |
|   0 ->   2  |   32 | 0 5 2                             |
|   0 ->   3  |   48 | 0 5 2 4 3                         |
|   0 ->   4  |   35 | 0 5 2 4                           |
|   0 ->   5  |   15 | 0 5                               |
|   0 ->   6  |   19 | 0 5 6                             |
|   0 ->   7  |   49 | 0 5 2 4 7                         |
+-------------+------+--------------------------------+
```

NB : L’affichage de toutes les valeurs sont aligné dans l’affichage console, la mise en forme du document déforme cette alignement.