IUT CLERMONT AUVERGNE
Aurillac - Clermont-Ferrand - Le Puy-en-Velay
Montluçon - Moulins - Vichy

DÉPARTEMENT
INFORMATIQUE
Clermont-Ferrand

SAE 2.02

Exploration algorithmique d'un problème

Agostinho Alexandre

22 mars 2023

Table des matières

1 Partie 1

1.1 Question 1

Pour représenter ce réseaux, je propose la structure de données suivante :

```
#define NAME_MAX_LEN 40
#define NB_PTS_CARACT_MAX 10

typedef struct pointCaracteristique {
    char nom[NAME_MAX_LEN];
    struct pointCaracteristique **tPtsCaract;
    int nbPtsCaract;
} PointCaracteristique, *Rue;

PointCaracteristique *reseau[NB_PTS_CARACT_MAX];
```

La structure du point se compose de son nom et d'un tableau de pointeurs sur d'autre points. Le réseau est ainsi composé d'un tableau de pointeur sur des structures 'pointsCaracteristique', elles-même pointant sur une un ou plusieurs point de la liste.

Le réseau est caractérisé par ses points caractéristiques. Chaque point connaît les points accessibles de manière direct. Cela permet de représenter les rues et leur sens. Prenons par exemple, une rue qui part du point A et qui a pour destination le point B. Elle sera caractérisée par la présence du point B dans la liste des points accessibles du point A. De cette manière on peut facilement dessiner le réseau et calculer les trajets à suivre. En somme, les pointeurs servent à caractériser les rues et leurs sens.

1.2 Question 2

Le choix d'une telle structure est fait sur la base de la simplicité d'implémentation. En effet, cette structure est relativement simple. Elle ne travaille quasiment que sur de la manipulation de pointeurs. Le parcours des points caractéristiques est donc facile.

De plus, si on se penche sur la complexité de l'implémentation, le nombre d'opérations requise pour le parcours du réseau est proportionnel au nombre de points que l'on parcours. Au pire, tout les points seront parcouru : c'est-à-dire que la complexité maximale s'élève au nombre de point qui compose le réseaux.

1.3 Question 3

Pour savoir si A est accessible directement depuis le point B, il suffit juste de regarder si B se trouve dans le tableau des points caractéristiques accessibles directment.

Voici le code correspondant :

```
int isAccessDirect(PointCaracteristique *A, PointCaracteristique *B)
{
    for (int i = 0; i < A->nbPtsCaract; i++)
    {
        if (A->tPtsCaract[i] == B)
        {
            return 1;
        }
    }
    return 0;
}
```

1.4 Question 4

Pour savoir si B est accessible depuis A, directement ou non (par une ou plusieurs rues), il est possible de répéter récursivement la fonction précédante. Ainsi, elle va tester chaque

point independamment et si un chemain direct est trouvé, c'est que il y à forcément un chemain qui rend B accessible.

Voici le code correspondant :

```
// version recusrive
int isAccessREC(PointCaracteristique *A, PointCaracteristique *B)
{
    for (int i = 0; i < A->nbPtsCaract; i++)
    {
        if (isAccessDirect(A, B))
        {
            return 1;
        }
        else
        {
            return isAccessREC(A->tPtsCaract[i], B);
        }
    }
    return 0;
}
```

Voici maintenant une ébauche de travail qui visait à créé une version itérative de cette fonction :

```
int isAccessITE(PointCaracteristique *A, PointCaracteristique *B)
{
    int i = 0;

    PointCaracteristique *ptsMarque[NB_PTS_CARACT_MAX];
    int nbPtsMarque = 0;

    PointCaracteristique *ptsTest = A;

    // pour tout les points du point teste
    for (i = 0; i < ptsTest->nbPtsCaract; i++)
    {
        // test si point deja teste
        for (int j = 0; j < nbPtsMarque; j++)
        {
            if (ptsTest == ptsMarque[j])
            {
                break;
            }
        }

        // test si pour ce point le chemin est direct
        if (isAccessDirect(ptsTest, B))
        {
            return 1;
        }

        // le point viens d etre teste, il entre dans la liste
        ptsMarque[nbPtsMarque++] = ptsTest;

        // on teste ensuite le prochain
        ptsTest = ptsTest->tPtsCaract[i];
    }

    return 0;
}
```

Je me suis rendu compte que je n'arrivait pas à pensé l'algorithme autrement que récursivement, c'est pourquoi j'ai décidé de mettre cette ébauche de côté.
Neanmoin, il est quand même possible de réutiliser la partie de code qui teste si un point à déjà été testé. Ceci diviserait, par chaque points testé, la complexité de cette algorithme.

Voici maintenant une ébauche de travail qui visait à créé une version itérative de cette fonction :

```
int isInTab(PointCaracteristique *P, PointCaracteristique **TabP, int
    lenTabP)
{
    for (int i; i < lenTabP; i++)
    {
        if (TabP[i] == P)
        {
            return 1;
        }
    }
    return 0;
}

// version rapide
int isAccess(PointCaracteristique *A, PointCaracteristique *B,
    PointCaracteristique **ptsMarque, int *nbPtsMarque)
{
    for (int i = 0; i < A->nbPtsCaract; i++) // erreur
    {
        if (! isInTab(A, ptsMarque, *nbPtsMarque))
        {
            if (isAccessDirect(A, B))
            {
                return 1;
            }
            else
            {
                return isAccess(A->tPtsCaract[i], B, ptsMarque, nbPtsMarque
    ); // erreur
            }

        }
        ptsMarque[*nbPtsMarque++] = A;
    }
    return 0;
}
```

1.5 Question 5

2 Partie 2

2.1 Question 1