Projet IN403 - Tables de routage

DANKOU Mathis, DELORT Tristan et SOURSOU Adrien

Mai 2019

Sommaire

1 Implémentation

On a choisit d'utiliser le langage C afin de réaliser le projet. Pour la partie optionnelle graphique, nous utilisons la bibliothèque CSFML.

1.1 Découpage

La manipulation du graphe se trouve dans le fichier *graph.c*
Le réseau est géré dans fichier *network.c*

1.2 Structures de données

Les structures de données utilisées pour le graphe sont la structure *Node* et *Graph*.

La structure *Node* est donc définie par le numéro du sommet voisin, le poids correspondant aux temps de communication *Node* ainsi qu'un pointeur vers le *Node* voisin suivant.

```
typedef struct s_Node
{
    uint8_t id;
    uint8_t weight;
    struct node *next;
}   Node;
```

Graph est définie par sa taille *size* (le nombre de noeud qu'il possède) et un tableau de listes chaînées de *Node*. Ce tableau sert à enregistrer la liste des successeurs du noeud n à l'indice i du tableau ainsi que le temps de communication vers ces noeuds.

```
typedef struct s_Graph
{
    uint8_t size;
    Node **tab;
}   Graph;
```

La liste chaînée a été privilégiée en particulier pour son gain de place par rapport à la matrice d'adjacence. Cette dernière nous aurait forcée à enregistrer un très nombre d'informations inutiles puisque 72 % des noeuds ne feront que 3 connections.

1.3 Dijkstra

L'implémentation de l'algorithme de Dijkstra a été réalisée avec un tas binaire pour obtenir une complexité logarithmique. Le tas binaire est stocké sous forme de tableau et est utilisé comme une file de priorité.

1.4 Affichage

L'affichage est réalisé avec la bibliothèque graphique CSFML. La documentation est disponible sur le site de SFML en C++ ou en C dans le SDK de CSFML. Pour le placement des Node, nous avons opté pour un affichage *force-directed* car l'affichage en plaçant les Node aléatoirement n'était pas assez lisible.
Pour l'algorithme *force-directed*, le fonctionnement est très simple. Pour chaque Node on calcul la force totale causée par l'attraction des autres Node (Tels des Ressorts) et la force de répulsion (Comme des particules) et ,en supposant la masse des Node unitaire, on ajoute cette force a la vitesse du Node. La constante *DAMPING* effectue un "frottement" pour que les Node s'immobilisent et une fois que le mouvement total est bas, le Graph est affiché.
Les constantes pour modifier le comportement de l'algorithme se trouvent dans *force.h* et l'implémentation de l'algorithme et de l'affichage se trouvent dans *force.c*.
K_COULOMB pour la force de répulsion, *K_HOOKE* pour la force d'attraction, *DAMPING* pour le frottement, et *SPRING_LEN* pour la longueur des ressorts.

2 Utilisation

Cette partie regroupe les directives de compilation et d'exécution du programme

2.1 Dépendances

Le projet a besoin de gcc et CSFML sous Linux pour fonctionner. Ces dépendances peuvent être installées à partir de la commande :

```
make install
```

2.2 Compilation

Le programme se compile via la commande :

```
make rooting-table.out
```

2.3 Exécution

Le programme s'exécute via la commande :

```
make
```

ou

```
make run
```

2.4 Comportement du programme

Le programme génère le réseau tant qu'il n'est pas connexe (en appliquant l'agorithme du parcours en profondeur), puis calcule la table de routage du réseau. Enfin, l'utilisateur choisit son noeud de départ et d'arrivée afin d'obtenir le cheminement de son message, qui s'affiche à l'écran.