DUT Informatique Graphique
Le Puy en Velay

Graphe
TP 2

département informatique graphique

Imagerie 1 | Pierre-Yves Bischoff | 2021

Graphe visuel

En réalisant ce TP vous pourrez affichez des nœuds et des arêtes en utilisant LibGraph2. Par défaut vous affichez des nœuds en cliquant ! ! ! !

Exercice 1 : Recopie de vos fonctions

Vous récupérez les fonctions du TD/TP précédents. J'ai rajouté les fonctions que je trouve nécessaire. J'ai aussi ajouté presque toute la partie graphique excepté l'ajout d'arête.

Exercice 2 : CNode

On ajoute une variable statique `m_uiNbNodes` qui totalisera les instances de $CNode$ initialisée à 0. Cette variable qui s'incrémente dans le constructeur, permet d'affecter un nom différent à chaque nœud : P1, P2, P3, ...
Normalement les nœuds n'ont pas la connaissance des arêtes. Mais comme nous allons utiliser plus tard un algorithme de coloration qui devra trier les nœuds par degré décroissant. Il sera souhaitable de stocker dans la classe `CNode` cette information qui sera mise à jour à chaque fois qu'on ajoute ou enlève une arête.
On ajoute ainsi une variable `m_uiDegre` initialisée à 0, avec ses accesseurs. On ajoute enfin les coordonnées du points.

```
size_t m_uiDegre = 0;
static  size_t m_uiNbNodes;
float m_fPosX = 0;
float m_fPosY = 0;
```

On ajoute les accesseurs sur ces variables et les opérateurs d'égalité.

```
size_t GetNbNodes() const { return m_uiNbNodes; }
size_t GetDegre()const { return m_uiDegre; }
void AddDegre(size_t degreToAdd) { m_uiDegre += degreToAdd; }
void SubDegre(size_t degreToSub) { m_uiDegre -= degreToSub; }
float GetX()const { return m_fPosX; }
float GetY()const { return m_fPosY; }
void SetX(float X) { m_fPosX = X; }
void SetY(float Y) { m_fPosY = Y; }
bool operator==(const CNode& node) { return m_sName == node.m_sName; }
bool operator==(const CNode::pNode& pNode) { return m_sName == pNode->m_sNam
```

Exercice 3 : CEdge

La classe CEdge est très peu impactée par ce TP. Vous aurez juste à ajouter un opérateur == qui renverra vrai s'ils ont même poids et mêmes nœuds et les geteurs sur le milieu de l'arête.

```
bool operator==(const CEdge & edge)
float GetMilieuX()const { return (m_apNodes[0]->GetX() + ...) / 2; }
float GetMilieuY()const { return (m_apNodes[0]->GetY() + ...) / 2; }
```

Exercice 4 : CGraph

Ajouter une fonction qui retourne un pointeur sur le nœud le plus proche du point (x, y) Ajouter une fonction qui retourne un pointeur sur l'arête la plus proche du point (x, y)

```
const CNode::Pointer GetNodeNear(unsigned int x, unsigned int y) const;
const CEdge::Pointer GetEdgeNear(unsigned int x, unsigned int y) const;
```

Un point (x, y) est le plus proche d'un nœud si sa distance euclidienne est la plus petite :

$$\sqrt{(x - \text{pNode->GetX()})^2 + (y - \text{pNode->GetY()})^2}$$

Un point est le plus proche d'une arête s'il est le plus proche de son milieu.

Exercice 5 : CTP

- La classe CTP va faire la liaison entre le graphe qu'elle possède en donnée membre et les fonctions de feedback.
- Elle va gérer toutes les fonctionnalités de base : Ajouter arête et nœud, et les supprimer.
- Elle permet aussi de dessiner le Graphe.

Dans ce TP vous ajouterez une fonction qui transforme un graphe à n nœuds, en son graphe complet K_n.

```
void CreateGraphFull();
```

Exercice 6 : prog.cpp

C'est le fichier contenant la fonction $main$. Il gère tous les événements. Le seul cas que vous aurez à programmer c'est l'ajout d'une arête.