DUT Informatique Graphique
Le Puy en Velay

Graphe
TP Noté

département informatique graphique

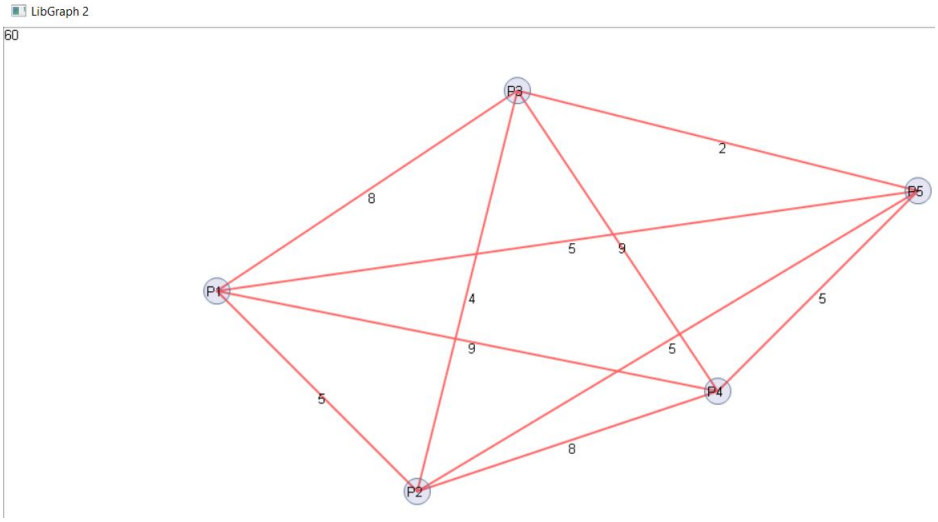
Imagerie 1 — Pierre-Yves Bischoff — 2024

Arbre de recouvrement de poids maximum

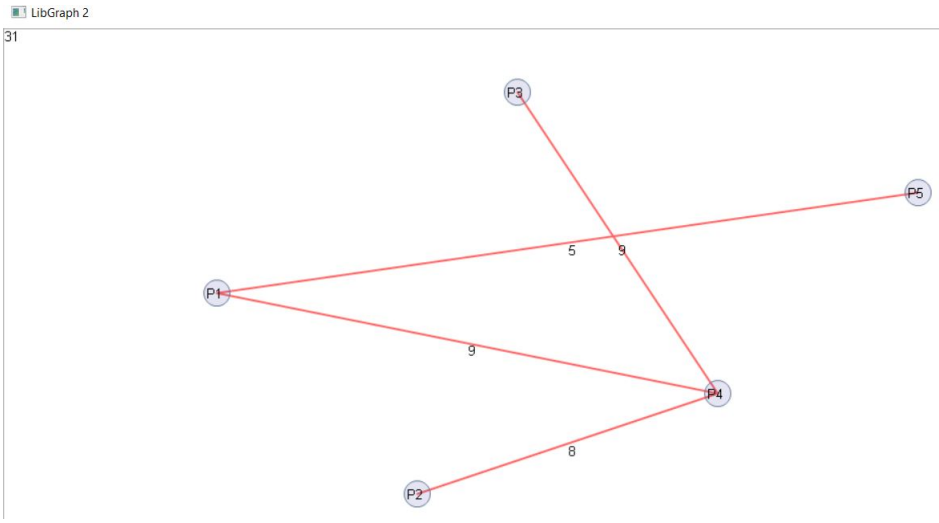
Exercice 1 : Positionnement du problème

Dans un graphe connexe non-orienté et pondéré, chaque arête possède un poids qui est un nombre qui représente le coût de cette arête. Dans un tel graphe, un arbre recouvrant est un sous-graphe connexe sans cycle qui contient tous les sommets du graphe. Le poids d'un tel arbre est la somme des poids des arêtes qui le compose.

En plus simple, imaginons une compagnie aérienne dont les arêtes correspondent aux vols assurés, et les nœuds les villes d'escale, qui désire simplifier son offre. Elle veut pouvoir relier toutes les villes avec le moins de vols possibles, tout en maximisant ses bénéfices. Le poids de l'arête correspond au bénéfice moyen du vol en million d'euros.



Un arbre couvrant maximum est un arbre couvrant dont le poids est supérieur ou égal à celui de tous les autres arbres couvrants du graphe. L'objectif de l'algorithme est de trouver un tel arbre de recouvrement de poids maximum.



Description de l'algorithme :

L'algorithme construit un arbre recouvrant en sélectionnant des arêtes par poids décroissant. L'algorithme considère toutes les arêtes du graphe par poids décroissant et pour chacune d'elle, il la sélectionne si elle ne crée pas un cycle.

Une solution permettant la visualisation d'un graphe à l'aide de libgraph vous est fournie. Cette solution correspond à ce que vous auriez dû finir à la fin du TP5. La solution est commentée et vous avez les recommandations pour le code à rajouter.

Exercice 2 : CNode

Ajouter un attribut de type `size_t` pour gérer à quelle composante connexe du graphe le nœud appartient.
Une composante connexe est un sous ensemble du graphe initial tel qu'il existe au moins un chemin (parfois de plusieurs arêtes) entre tous les nœuds de la composante.

Par exemple un graphe composé de 3 sommets et aucune arête a 3 composantes connexes : les 3 sommets. Si on rajoute 1 arête entre 2 sommets, il se retrouve avec 2 composantes connexes : la première qui contient les 2 sommets reliés et l'arête, et la deuxième le sommet isolé. Si on rajoute une arête pour relier le sommet isolé à l'un des deux autres sommets alors le graphe n'a plus qu'une composante connexe.

Par défaut, cet attribut doit être initialisé à sa valeur du nombre d'instance. Au début les nœuds ne sont pas reliés entre eux, donc ils appartiennent tous à une composante connexe différente.

Quand on rajoute une arête à un graphe elle est susceptible de faire baisser de un le nombre de composante connexe.

```
size_t m_connexe = 0;
```

Ajouter des accesseurs sur cette variable.

Exercice 3 : CEdge

La classe CEdge n'est presque pas impactée par ce TP, vous devrez juste rajouter un foncteur pour trier les arêtes par poids décroissant. (Vous pouvez vous inspirer de celui qui trie les nœuds vu dans le TP coloration).

Exercice 4 : CGraph

La classe Graphe est énormément impactée dans ce TP. Presque toutes les fonctions doivent être complétées. On ajoute pas mal de méthodes privées pour faciliter ce travail.
Les deux fonctions publiques que vous devrez programmer sont : **ComputeWeight** qui retourne la somme du poids de toutes les arêtes. Et **Simplify** qui crée l'arbre de recouvrement de poids maximum.

```
double ComputeWeight() const;
void Simplify();
```

Votre première fonction pour vous mettre en jambe sera de calculer le poids d'un graphe. Je vous ai mis directement dans le code les informations pour coder les fonctions privées.

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, le colorer.
Dans ce TP vous ajouterez une fonction qui lance l'algorithme de simplification sur le graphe.

```
void launchSimplify();
```

Vous aurez aussi à modifier la fonction **Draw** pour qu'elle affiche en haut à gauche le poids du graphe.

Exercice 6 : prog.cpp

C'est le fichier de la fonction $main$. Il gère tous les événements. Le seul cas que vous aurez à programmer c'est le lancement de l'algorithme de simplification.

Exercice 7 : Graphe complet

1. Maintenant vous allez pouvoir toucher au Graal : appliquez l'algorithme au graphe d'exemple et :

   Vérifiez que le graphe s'affiche correctement et que l'arbre de poids maximum est correct (réponse : 14).

2. Pour conclure ajoutez un point et faites le graphe complet, puis lancez l'algorithme de simplification.

   Vérifiez que le graphe s'affiche correctement et que l'arbre de poids minimum est correct ! ! !

Je vous rappelle que c'est un TP comme un autre et que vous disposez des ressources internet. Vous pouvez aussi faire appel au professeur qui est libre de répondre ou non aux question. Attention vous n'avez pas le joker j'appelle un ami ! ! ! !