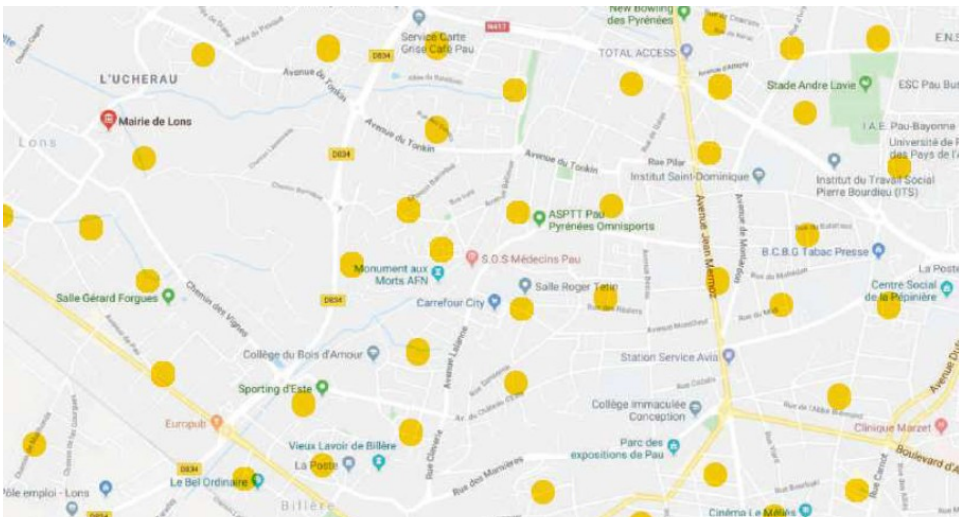
TP série N°2: optimisation du coût d’un réseau de fibre optique

Christoph Samuel – Jankowiak Matthias

I-Position du problème:

1- L'objectif de ce TP est celui de **l'optimisation du coût d'un réseau de fibre optique.** En effet, dans le cadre d'un projet de déploiement, par un opérateur, d'un réseau de fibre optique sur une zone non **uniformément urbanisée,** l'optimisation du **coût global** de déploiement est **primordiale** pour la rentabilité du projet. Ainsi, une estimation préalable de l'investissement à réaliser est un facteur déterminant dans la **prise de décision de l'opérateur.**

2- Le problème posé peut se ramener à un **problème de recherche de couverture minimale** de la fermeture transitive **d’un graphe non orienté** représentatif du réseau à équiper, en effet, on veut assurer une connexion avec un **coût global minimum,** les liaisons assurent la connexité de tous les sommets et l'absence de liaisons cycliques. Il nous faut alors déterminer **le plus petit ensemble de relations** nous permettant de parcourir **l’ensemble S des sommets du graphe,** en effet, on cherche à **supprimer les liaisons les moins rentables.**



3- Le problème réel dans le cadre de la théorie des graphes consiste à choisir entre **plusieurs algorithmes ACM.** Le meilleur choix se fait en fonction de la **complexité de l'algorithme.** Nous avons choisi dans le cadre du TP, de nous appuyer sur l’algorithme **Kruskal** qui permet la recherche d’arbres de recouvrement **de manière optimale** dans un **graphe non orienté** et de **faible densité,** on pourra éventuellement, dans le cadre de ce TP, comparer différents algorithmes ACM.

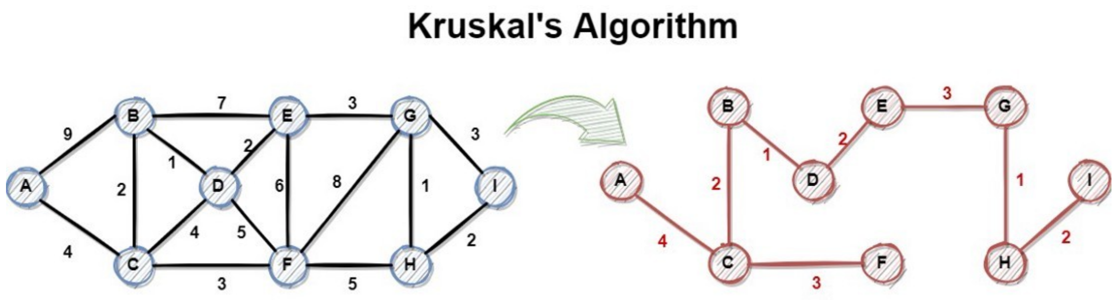
II-Réalisation:

1- Une façon de résoudre ce problème est de **modéliser le réseau** à l'aide d'un **graphe non orienté valué et connexe G = (S,A,C).** Chaque sommet S appartenant à S de G représente **un nœud du réseau** de fibre optique, chaque arc A appartenant à A de G **modélise une connexion adéquate entre deux nœuds du réseau** depuis un nœud vers un autre nœud, pour finir chaque coût c appartenant à C de G représente **le coût engendré par une connexion.**

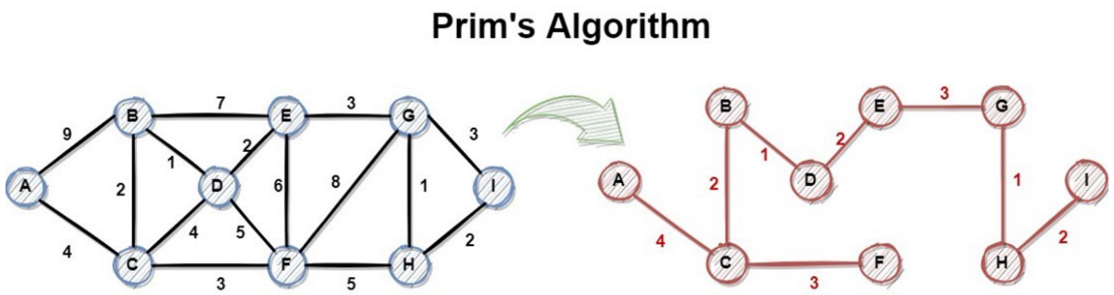
2- Le problème du test de connectivité des réseaux critiques est soulevé dans le cadre d'un projet de déploiement, par un **opérateur d'un réseau de fibre optique** sur une zone non uniformément urbanisée, il s'agit de trouver comment relier tout les nœuds à un point de départ **avec un coût minimal,** on cherche alors le plus petit ensemble de connexions permettant l'intersectionnalité de **l'ensemble du réseau de fibre optique.** Ce problème se ramène alors à un **problème classique de recouvrement minimal dans graphe non orienté valué et connexe.** Ainsi, nous pouvons établir **l'arbre couvrant minimum** du réseau à l'aide d'outils informatiques, de la librairie Boost Graph développé en C++ et un algorithme tel que prim ou kruskal.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
1		8	13			12					1																							
2			10						21																									
3				11			13																								40			
4					4	2	22																											
5			4																															
6					5																													
7								2	8																									
8													1																					
9								10		9																								
10											1	14	2																					
11												14	23																					
12													7																					
13							12							10																				
14							8								18																			
15																11	9																	
16																						9								5	6	13	11	13
17																		10																
18																			7				7											
19																				6				8										
20																					9													
21																									2									
22																				10													7	
23																								4		8								
24																									15	6								
25																											3	4						
26																													10					
27																												2						
28																													7					
29																																		
30																															2	10		
31																																		8
32																																	1	

3- L'essentiel du problème consiste à choisir entre les **différents algorithmes ACM,** afin de résoudre ce problème nous avons la possibilité d'utiliser un des deux que nous avons étudié **celui de Kruskal et celui de Prim,** la différence étant que **Prim s'initialise avec un arbre ainsi qu'un ensemble de sommets isolés** et s'étendent d'un nœud à un autre tandis que **Kruskal démarre en partant d'une forêt** de manière à ce que la position de l'arête ne soit pas basée sur la dernière étape. La finalité des deux algorithmes est la même, cependant, la complexité de ces deux procédures est différente, en effet, on peut observer que:



L’algorithme de **Kruskal** a une complexité de **O(mlog(m))** avec m nombre d’arcs



L’algorithme de **Prim** a une complexité de **$O(n^2)$** avec n nombre de sommets.

Le choix de l’algorithme se fera donc en fonction de la **densité du graphe.** Si la densité du graphe tend vers 1, alors m tend vers n2. Donc la complexité de l’algorithme de Kruskal est en O (n2 log(n)). Dans ce cas, le choix de l’algorithme sera celui de **Prim** qui devient plus efficace que l’algorithme de Kruskal. Au contraire si la densité du graphe tend vers 0, alors m < n2, ce sera donc l’algorithme de **Kruskal** qui sera le plus efficace.

D = m/n^2 avec m = nombre d’arêtes et n = nombre de sommet

Dans notre cas il sera plus judicieux de choisir l’algorithme de Kruskal, car **il y a peu d’arêtes par rapport au nombre de sommets,** en effet, nous avons 34 sommets et 59 arêtes soit:

D = $34/59^2$ = 0.00976 qui tend vers 0

La complexité de **Kruskal est dominée par le tri** donc sa complexité est celle d’un tri, elle est en O(m log(m)) avec m le nombre d'arêtes. L'algorithme de kruskal est d’autant plus rapide que le graphe est pauvre en arrête. L’algorithme de Kruskal est donc **très efficace dans ce cas.** Nous avons alors décidé d’utiliser l’algorithme de Kruskal dont voici le déroulement, il fonctionne en deux étapes:

- étape 1: **trier les m arête** de G par **ordre croissant de leur coût.**
- étape 2: partir du **graphe vide** de G et rajouter les arêtes en **suivant l’ordre de l’étape 1** en s’assurant qu’elles ne forment **pas de cycles.**

On rentre la **fermeture transitive** du graphe dans le programme et on lance l'algorithme de Kruskal pour obtenir un arbre couvrant. Avec l'algorithme codé en c++ suivant (tp2.cpp) on peut retrouver le **recouvrement minimum,** c'est-à-dire une liste des connexions optimales. Nous commençons alors par construire le modèle de présenté plus haut en utilisant la bibliothèque **Boost Graph Library** sur VisualStudio code de façon à obtenir son graphe G.

Pour cela nous commençons par inclure et créer l'ensemble des librairies boostgraph et propriétés des **différents nœuds et arcs (coût…)** des nécessaires pour l'analyse et l'affichage d'un cas concret.

```
// Pour std::cout.
#include <iostream>
using namespace std;
// Pour clock_t.
#include <time.h>
// Inclusion de la librairie Boost.
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/graph_utility.hpp>
#include <boost/graph/graph_traits.hpp>
#include <boost/graph/graphviz.hpp>
#include <boost/lexical_cast.hpp>
using namespace boost;
// Inclusion d'une recherche de couverture minimale avec l'algorithme de Kruskal.
#include <boost/graph/kruskal_min_spanning_tree.hpp>

```

```
// Dans la structure, on définit les propriété des vertex, c'est à dire des noeux.
struct VertexProperties
{
    // Un identifiant manuel (qui n'est pas assigné automatiquement par la librairie) donc 1,2,3, etc ...
    unsigned id;
    // Un constructeur par défaut si aucune donnée n'est rentrée lors de la création du noeud.
    VertexProperties() : id(0) {}
    // La fonction qui permet d'assigner les valeur saisie par l'utilisateur au noeud correspondant.
    VertexProperties(unsigned i) : id(i) {}
};

```

```
// Création de la propriété nécessaire a la création du coût de chaques arêtes.
typedef property<edge_weight_t, int> EdgeWeightProperty;
// Passage des paramètres nécéssaires à adjacency_list afin d'obtenir le graph désiré.
typedef adjacency_list<listS, vecS, undirectedS, VertexProperties, EdgeWeightProperty> Graph;
// Définiton du type noeuds.
typedef typename graph_traits<Graph>::vertex_descriptor Vertex;
// Définiton du type arcs.
typedef typename graph_traits<Graph>::edge_descriptor Edge;

```

Ensuite on se base sur la cas d'application présenté du tp2, on créé donc un graphe correspondant à **l'optimisation du coût d'un réseau de fibre optique** en déterminant la **couverture minimale** de celui-ci, pour cela nous implémentons notre main comme suit. On commence par déclarer, le graphe g ainsi que chaque nœud et arcs pour notre cas voici un exemple pour les deux premiers nœuds et arcs.

```
// Fonction d'application.
int main()
{
    // Création du graphe.
    Graph g;
```

```
    // Déclaration de tous les noeuds de nom : "1", "2", "3" ... dans le Graphe g.
    Vertex v1 = add_vertex(VertexProperties(1),g);
    Vertex v2 = add_vertex(VertexProperties(2),g);
```

```
    // Déclaration de tous les arcs et leurs coûts : arc v1 qui est dirigé vers arc v2 de coût 8 dans le Graphe g.
    add_edge(v1, v2, EdgeWeightProperty(8),g);
    add_edge(v1, v3, EdgeWeightProperty(13),g);
```

On passe ensuite a l'initialisation et a la création des **fichiers** nécessaires pour nous ***afficher les différent graphes*** (graphe g, graphe sous kruskal…).

```
    // Initialisation du graphe avant passage dans kruskal sous graphe.dot.
    string graphDot = "graph.dot";
    ofstream FGraph(graphDot.c_str());
    // Initialisation du graphe résultat aprés passage dans kruskal sous kruskal.dot.
    string kruskalDot = "kruskal.dot";
    ofstream FKruskal(kruskalDot.c_str());

```

Nous avons également intégrer un élément de type **dynamic_properties** pour rendre notre code plus **simple d'utilisation et d'affichage.**

```
    // Initialisation d'une dynamic_properties pour afficher les noeuds et arcs de notre Graphe g.
    dynamic_properties dp;
    // L'élément dp récupére chaque noeuds et arcs et leurs donnent un nom (ou un coût) pour la création des graphes en png.
    dp.property("node_id", get(vertex_index, g));
    dp.property("label", get(&VertexProperties::id, g));
    dp.property("label", get(edge_weight, g));
    dp.property("weight", get(edge_weight, g));

```

Nous utilisons ensuite la **procédure Kruskal** implémentée comme suit, chaque partie de l'algorithme est détaillée dans le code (calcul **temps de la procédure** Kruskal ainsi que son **coût total minimal** à la sortie du programme).

```
/////////////////////////////////////////////////////////////////// PROCEDURE KRUSKAL ///////////////////////////////////////////////////////////////////

    // Création du graphe kruskal.
    Graph kruskal;
    // Initialisation du graphe kruskal avec les valeurs du Graph g.
    vector<graph_traits<Graph>::edge_descriptor> matKruskal;
    // Ajout du coût de chaque noeuds du graphe.
    property_map<Graph, edge_weight_t>::type weight = get(edge_weight, g);

    // Initialisation du temps avant et après passage du graphe dans l'algorithme de Kruskal pour couverture minimal (kruskal_minimum_spanning_tree).
    clock_t beginKruskal = clock();
    kruskal_minimum_spanning_tree(g, back_inserter(matKruskal));
    clock_t endKruskal = clock();

    // Calcul du temps mis pour trouver la couverture minimale du graphe.
    double time_spent = (double)(endKruskal - beginKruskal);

    // Initialisation du coût total.
    int totalWeight = 0;
    // Calcul du coût total minimal lors du passage dans l'algorithme
    for(int i=0; i < matKruskal.size(); i++)
    {
        totalWeight += get(weight, matKruskal[i]);
        add_edge(source(matKruskal[i], g), target(matKruskal[i], g), EdgeWeightProperty(get(weight, matKruskal[i])), kruskal);
    }

    // Affichage dans le terminal du temps et du coût total dans l'application de notre graphe.
    cout << "\n" << "Temps d'exécution de l'algorithme Kruskal : " << time_spent <<endl;
    cout << "Coût total du poid avec l'algorithme de Kruskal : " << totalWeight << "\n" << endl;

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

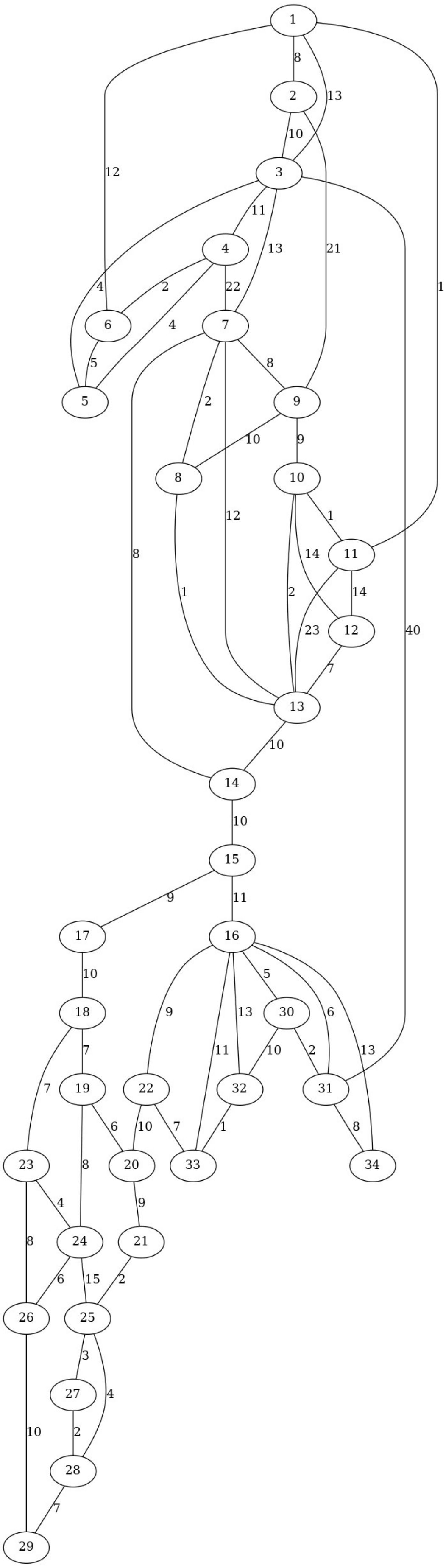
Pour finir nous affichons d'une part dans le **terminal** les éléments calculés lors du déroulement de notre programme et d'autre part la créations des images graph.dot avant passage dans Kruskal et kruskal.dot après passage dans Kruskal.

```
    // Création de l'image sous graph.png contenant le graphe g.
    write_graphviz_dp(FGraph, g, dp);
    system("dot -Tpng graph.dot > graph.png");
    // Création de l'image finale sous kruskal.png contenant la couverture minimale du graphe g utilisant la méthode kruskal.
    write_graphviz_dp(FKruskal, kruskal, dp);
    system("dot -Tpng kruskal.dot > kruskal.png");

    return 0;
}
```

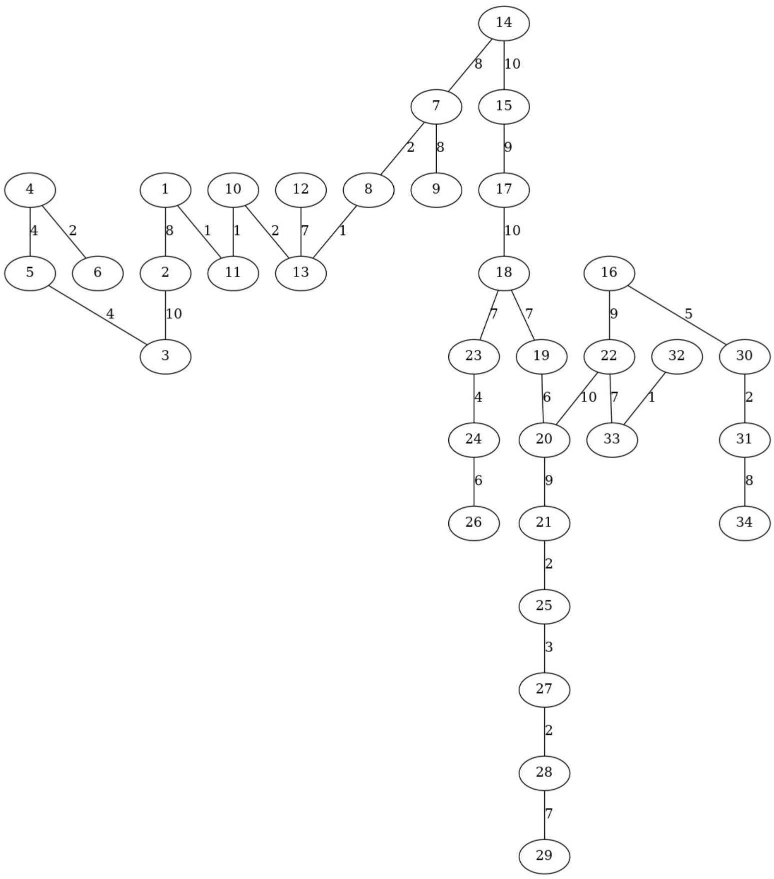
Le terminal nous affiche les **résultats attendu,** un temps égal à 59 pour un coût total minimum de 182. Nous observons également la création de **nos graphes pré et post Kruskal.**

```
schristoph@scinfe173 ~/Bureau/stockage/clef/tp2
./tp2.o

Cout total du poid avec l'algorithme de Kruskal : 182
Temps d'execution pour l'algorithme de Kruskal : 59
```

Affichage des résultats de la procédure kruskal sur notre terminal



Graph G (avant passage dans Kruskal)



Graph kruskal (après passage dans Kruskal)

4- Cet arbre couvrant minimum représente dans notre **cas réel** la couverture de **coût le plus faible** en fibre optique sur la zone urbaine. Ainsi l'opérateur pourra l'entièreté du réseau urbain à un moindre coût. Cependant cette couverture **n'est pas unique.** En effet, **plusieurs couvertures minimales sont possible** pour un coût équivalent. La solution présenté n'est qu'un exemple de solution possibles au problème. Cette démarche permet **en trouvant la stratégie optimale,** de **réduire le coût de déploiement** de la fibre sur le territoire.

III-Bilan/Conclusion:

1- Nous avons appris grâce à ce TP que certains **problèmes réels d'ingénierie,** concernant le déploiement de tout un réseau dans le souci d'un **coût optimal,** sont facilement traduisibles en un **problème de couverture minimale de la théorie des graphes**. Dans notre cas, le recouvrement minimum d'un graphe à l'aide d'un algorithme adapté, celui de **Kruskal,** nous a permis de mieux comprendre son fonctionnement et ses différentes étapes. Il en devient donc simple de résoudre de tels problèmes, à l'aide du **modèle de graphe** correspondant et des outils à dispositions.

2- Nous retenons également cette façon de faire, que nous pourrions vraisemblablement être confrontés à ce **type de problématique** dans le futur si nous sommes amenés à travailler dans le **domaine des télécoms** et le déploiement d'un **réseau optique** sur une zone non uniformément urbanisée.