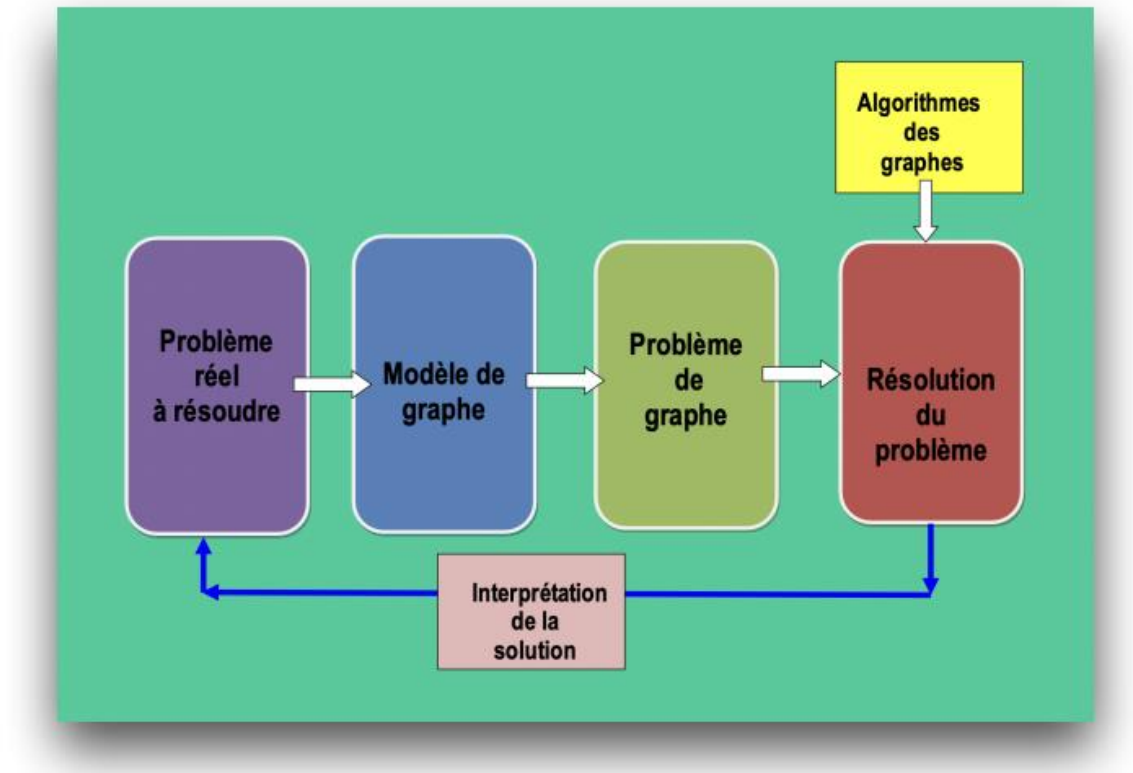
PALACIO Vincent
ROBIN Florian

Compte Rendu du TP2

Sujet : Optimisation du coût d'un réseau de fibre optique
Objectif du TP : Étude de la connexité d'un réseau

I -Position du problème :

Le problème exposé ici concerne le déploiement optimal d'un réseau de fibre optique par un opérateur réseau, en effet pour que le projet soit rentable et viable il doit respecter certains critères de conception. Ici l'ingénieur informaticien doit proposer un modèle de graphe pour représenter le réseau



Il faut donc une couverture totale en affichant les coûts correspondants de chaque connexion entre deux nœuds. Pour que celle-ci soit optimale, ce graphe devra avoir une couverture minimale afin d'engendrer un minimum de coût.

Plusieurs outils ont été utilisés pour l'étude du graphe :

• Il existe deux algorithmes afin de déterminer la couverture minimale d'un graphe, l'algorithme de Kruskal et de Prim. L'algorithme de Kruskal assure l'absence de cycle tandis que l'algorithme de Prim assure la connexité. Nous allons donc exposer le principe des deux algorithmes :

- **Algorithme de Kruskal :**
    - **Etape 1** : L'algorithme impose d'abord de trier les m arêtes de G par ordre croissant de leur coût, il faut ensuite partir d'un graphe G' vide.
    - **Etape 2** : Les arêtes sont considérées, une par une, dans l'ordre du tri. Si l'ajout d'une arête ai dans G' n'introduit pas de cycle alors on l'ajoute sinon on passe à l'arête suivante ai+1.
- **Algorithme de Prim** :
    - Cet algorithme consiste à fusionner deux à deux les sommets de G, c'est-à-dire remplacer deux sommets par un seul pour obtenir un seul sommet représentant l'arbre couvrant à construire.

Nous avons décidé de développer les différents programmes aidant à l'analyse du graphe en C++, à l'aide de l'environnement de développement intégré Visual Studio Code.

Nous avons également utilisé la librairie BoostGraph afin de visualiser les graphes qui seront créés.

Nous pouvons donc diviser la résolution de ce problème en 3 étapes, tout d'abord nous allons présenter le modèle du graphe du système étudié en lien avec l'optimisation du coût du réseau. Ensuite il suffit d'appliquer cette modélisation à notre cas d'étude. Pour finir nous montrerons que le problème exposé ci-contre se ramène à un problème de couverture minimale.

II- Réalisation :

1) Présenter le modèle du graphe en lien avec le système étudié :

Voici la configuration du modèle de graphe pour le cas étudié :
La décision a été prise d'utiliser un graphe G pour modéliser le réseau.

De plus G = (S, A) où :

• S est l'ensemble des sommets s ∈ S représentant chacun un nœud du réseau.

• A est l'ensemble des arcs (s i, s j) ∈ A représentant chacun une connexion entre le nœud représenté par son extrémité initiale s i et le nœud représenté par son extrémité terminale s j.

il faut également déterminer si on utilise un graphe orienté ou non orienté, d'après le cas d'étude exposé à l'aide de la matrice ci-dessous, nous pouvons dire qu'il faut utiliser un modèle de graphe non orienté.

	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		

G=(S,A,) où S: ensemble fini de sommets, A: un ensemble fini d'arêtes. Ils sont définis tel que :

- Chaque sommet Si (i=1…,34) représente un nœud de connexion.
- Chaque arête (Si, Sj) correspond à un branchement entre 2 nœuds de connexion.

2) Montrer de quelle manière le problème exposé se rapporte à un problème relevant de la théorie des graphes :

Étudier la connexion optimale d'un réseau tout en assurant sa connexité consiste à ramener le problème à un problème de recouvrement minimum.

Tout d'abord définissons la connexité d'un graphe :

Un graphe G=(S,A) orienté (non orienté) est dit connexe SI x, y$\in$ S $\Rightarrow \exists$ une chaîne reliant x et y.

Il faut aussi faire en sorte de ne pas générer des branchements inutiles, il faut donc s'assurer que le graphe ne possède pas de cycle. Cela implique donc l'utilisation d'un arbre couvrant.

Pour finir nous devons utiliser le moins de branchements possible, le coût doit être minimum.

Un arbre couvrant minimum correspond à :

Soit un graphe non orienté connexe G; on peut toujours trouver un arbre couvrant en supprimant de G les arêtes qui forment un cycle.

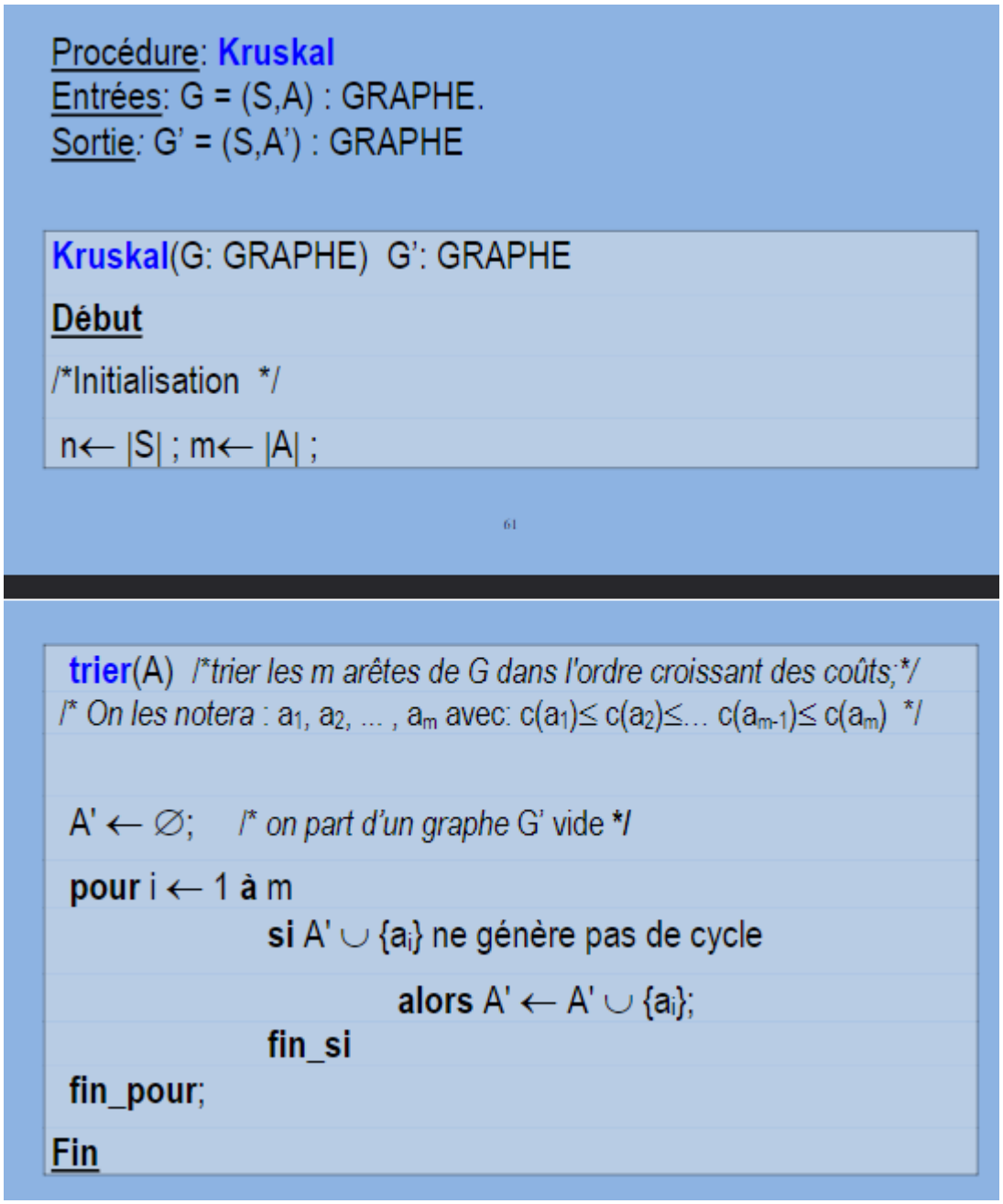
Il existe un nombre fini d'arbres couvrants pour G. Si G est valué, l'existence d'un arbre couvrant de coût minimum est donc assurée.

3) Exposer la solution retenue en mettant en œuvre une solution algorithmique issue de la théorie des graphes :

Nous nous sommes donc appuyés sur l'algorithme kruskal, qui permet d'obtenir le recouvrement minimum d'un graphe. Nous utilisons cet algorithme plutôt que l'algorithme de Prim car la rapidité de l'algorithme

de Kruskal est en fonction de la structure du graphe. En effet sa complexité est celle d'un tri de m arêtes donc elle est en O (m log(m)). L'algorithme de Kruskal est d'autant plus rapide que le graphe connexe est pauvre en arêtes (graphe de degré faible) tandis que l'algorithme de Prim est en $O(n^2)$. En conclusion l'algorithme de kruskal est plus intéressant à utiliser dans ce contexte.

Voici son algorithme :



Voici le code en conséquence avec quelques explications :

```cpp
#include <boost/graph/graphviz.hpp>
#include <boost/graph/graph_traits.hpp>
#include <boost/graph/depth_first_search.hpp>
#include <boost/graph/reverse_graph.hpp>
#include <boost/graph/graph_utility.hpp>
#include <boost/graph/copy.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/kruskal_min_spanning_tree.hpp>
#include <iostream>
#include <fstream>
int main()
{
  using namespace boost;
  typedef adjacency_list < vecS, vecS, undirectedS,no_property, property < edge_weight_t, int > > Graph;
  typedef graph_traits < Graph >::edge_descriptor Edge;
  typedef graph_traits < Graph >::vertex_descriptor Vertex;
  typedef std::pair<int, int> E;
    const int num_nodes = 34; // nombre de noeuds
  E edge_array[] = { E(1, 3), E(1, 6), E(1, 11), E(2, 3), E(2, 9),
    E(3, 4), E(3, 7), E(3, 31), E(4,5), E(4,6), E(4,7), E(5,3), E(6,5), E(7,8), E(7,9), E(8,13), E(9,8), E(9,10), E(10,11), E(10,12)
    , E(10,13), E(11,12), E(11,13), E(12,13), E(13,7), E(13,14), E(14,7), E(14,15), E(15,16), E(15,17), E(16,22), E(16,30), E(16,31), E(16,32)
    , E(16,33), E(16,34), E(17,18), E(18,19), E(18,23), E(19,20), E(19,24), E(20,21), E(21,25), E(22,20), E(22,33), E(23,24)
    , E(23,26), E(24,25), E(24,26), E(25,27), E(25,28), E(26,29), E(27,28), E(28,29), E(30,31), E(30,32), E(31,34), E(32,33)
  };//liaison des noeuds

  int weights[] = { 13, 12, 1, 10,21,11,13,40,4,2,22,4,5,2,8,1,10,9,1,14,9,14,9,14,23,7,12,10,8,18,11,9,9,5,6,13,11,13,10,7,7,6,8,9,2,10,7,4,8,15,6,3,4,10,2,7,2,10,8,1};
  std::size_t num_edges = sizeof(edge_array) / sizeof(E);
  #if defined(BOOST_MSVC) && BOOST_MSVC <= 1300
    Graph g(num_nodes);
    property_map<Graph, edge_weight_t>::type weightmap = get(edge_weight, g);
    for (std::size_t j = 0; j < num_edges; ++j) {
      Edge e; bool inserted;
      boost::tie(e, inserted) = add_edge(edge_array[j].first, edge_array[j].second, g);
      weightmap[e] = weights[j];
    }
  #else
    Graph g(edge_array, edge_array + num_edges, weights, num_nodes);
    #endif
    property_map < Graph, edge_weight_t >::type weight = get(edge_weight, g);
    std::vector < Edge > spanning_tree;

    kruskal_minimum_spanning_tree(g, std::back_inserter(spanning_tree)); //utilisation librairie inclut dans boostgraph

  int i=0;
    std::cout << "Print the edges in the MST:" << std::endl;
    for (std::vector < Edge >::iterator ei = spanning_tree.begin();
         ei != spanning_tree.end(); ++ei) {
      std::cout << source(*ei, g) << " <--> " << target(*ei, g)
        << " with weight of " << weight[*ei]
        << std::endl;
        i +=  1;
    }
  // création du graphe et du graphe suite à l'algorithme de kruskal dans une seule image
    std::ofstream fout("kruskal-eg.dot");
    fout << "digraph g {\n"
      << " rankdir=TB\n"
      << " edge[style=\"bold\"]\n" << " node[shape=\"circle\"]\n";
    graph_traits<Graph>::edge_iterator eiter, eiter_end;

    for (boost::tie(eiter, eiter_end) = edges(g); eiter != eiter_end; ++eiter) {
      fout << source(*eiter, g) << " -> " << target(*eiter, g);
      if (std::find(spanning_tree.begin(), spanning_tree.end(), *eiter)
          != spanning_tree.end())
        fout << "[color=\"black\", label=\"" << get(edge_weight, g, *eiter)
             << "\"]\n";
      else
        fout << "[color=\"gray\",label=\"" << get(edge_weight, g, *eiter)
             << "\"]\n";
    }
    fout << "}\n";
      write_graphviz(fout, g);
      system("dot -Tpng kruskal-eg.dot > TP2-GrapheMin.png");
    return EXIT_SUCCESS;
  }
```

Pour implémenter l’algorithme de kruskal nous utilisons la librairie de boostGraph qui possède une partie dédiée à cet algorithme. De plus pour des raisons d’efficacité nous allons générer un seul graphe pour montrer le graphe initial et le graphe après l’exécution. En effet les nœuds qui seront en gras représenteront le graphe après l’exécution.

Voici le graphe générer par le code ci-dessus. (Voir figure 1).

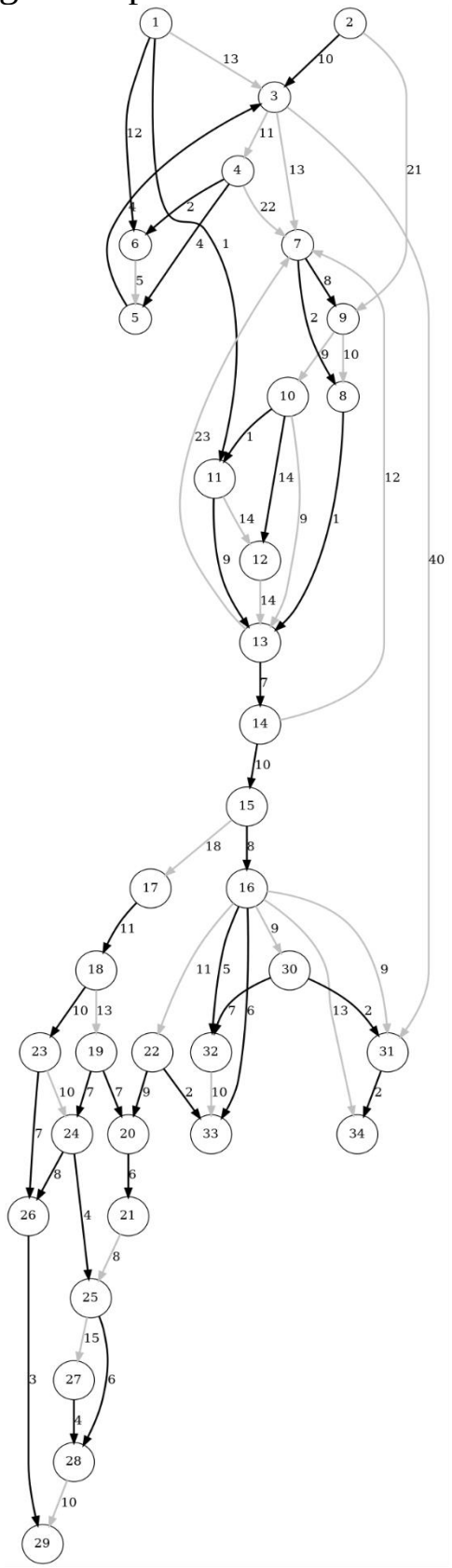
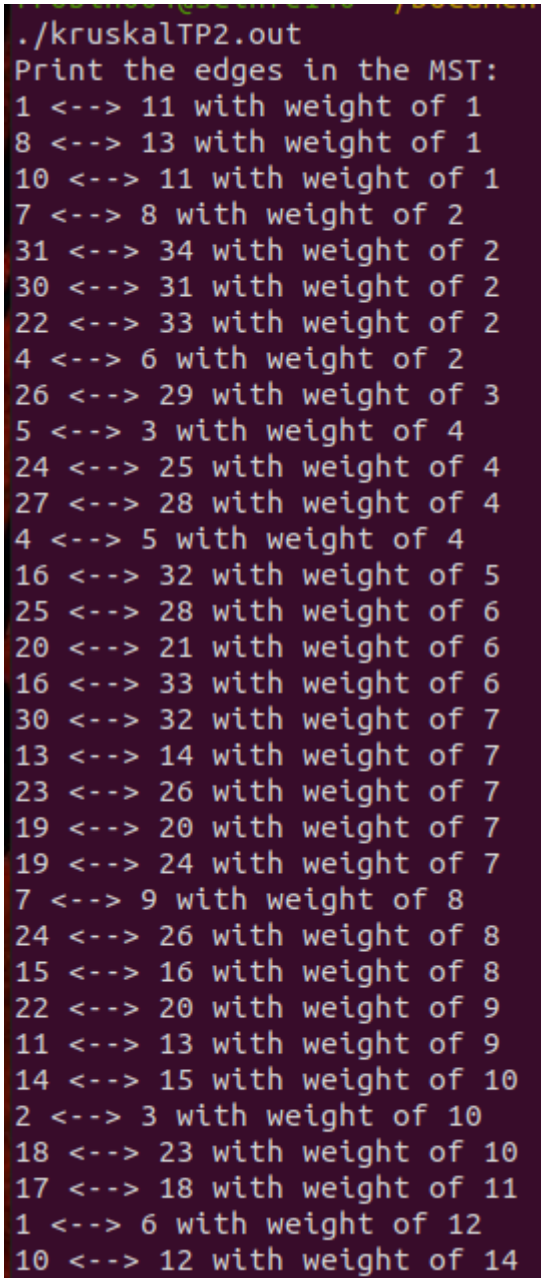


Figure 1 : graphe initial et graphe de kruskal (en gras)

De plus en gérant le code nous affichons dans la console le détail des nœuds à la suite de l'application de l'algorithme de kruskal comme ci-dessous.



Chaque liaison est affichée avec le poids associé.

III- Conclusion :

En réalisant cette série de travaux pratiques, nous avons développé nos compétences dans la programmation de modèles et d'algorithmes sur les graphes. Nous avons pu voir qu'il est primordial de savoir déterminer la couverture minimale d'un graphe afin d'optimiser le graphe.

Ainsi nous avons pris connaissance de l’algorithme de kruskal qui permet de déterminer la couverture minimale notamment grâce aux librairies déjà existante dans le langage C++.
En déterminant la couverture minimale, on peut donc éliminer les connexions redondantes du réseau, cela rien à éliminer tous les cycles et le graphe devient alors un arbre. De plus la connexion de tous les nœuds est assuré ce qui en fait un arbre couvrant total. Pour finir on optimise le coût global de connexion du réseau en recherchant le coût minimum.