TP 2 : Arbres couvrants

lionel.rieg@ens-lyon.fr

Ce TP se focalise autour de la création d'un arbre couvrant de poids minimum par l'algorithme de Kruskal sur un ensemble de points aléatoirement réparti dans un rectangle. Les paramètres du problème sont :

- h et w les hauteur et largeur du rectangle,
- n le nombre de points dans ce rectangle.

L'algorithme de Kruskal sélectionne un sous-ensemble des arêtes pondérées d'un graphe connexe pour créer un arbre couvrant de poids minimal. Ici, comme on veut que les points reliés soient les plus proches au sens de la distance euclidienne, on va utiliser le graphe complet (c. à d. celui avec toutes les arêtes) et le poids d'une arêtes sera la distance entre ses deux extrémités.

1 Création d'un ensemble de point aléatoires

Les points seront stockés par un tableau **point** de taille $n \times 2$.
En se servant de la méthode de la semaine dernière, créer un ensemble de n points aléatoires dans un rectangle de taille $h \times w$.

2 Calcul de toutes les distances entre les points

1. Combien y a-t-il d'arêtes dans le graphe complet à n sommets ?
2. Écrire une fonction qui calcule les distances entre toutes les paires de points, c. à d. qui calcule le coût des arêtes de notre graphe. Le tableau **edge** aura ici trois lignes : deux pour les extrémités de l'arête et une pour son poids.

3 Tri des arêtes

Trier le tableau des arêtes par poids croissant. On pourra utiliser n'importe quel tri (tri bulle par exemple) car la complexité n'est pas cruciale ici.

4 Calcul de l'arbre couvrant

En se basant sur l'implémentation du calcul des composantes connexes de la semaine dernière, calculer un arbre couvrant de poids minimal.

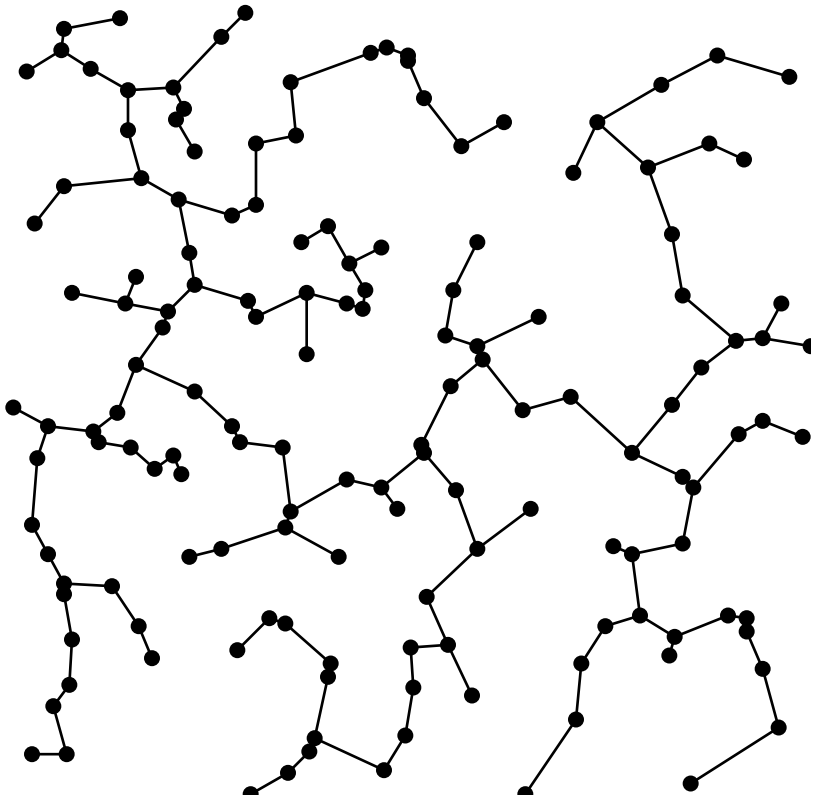
5 Affichage

Pour afficher votre arbre, vous pouvez utiliser la fonction suivante

```
void show(int h, int w, int n, int point[][2], int arbre[][2]) {
    ofstream output;
    output.open("Tree.ps", ios::out);
    output << "%!PS-Adobe-3.0" << endl;
    output << "%%BoundingBox: 0 0 " << h << " " << w << endl;
    output << endl;
    for(int i=0;i<n;i++) {
        output << point[i][0] << " " << point[i][1] << " 3 0 360 arc" << endl;
        output << "0 setgray" << endl;
```

```
            output << "fill " << endl;
            output << "stroke "<< endl;
            output << endl;
        }
        output << endl;
        for(int i=0;i<n-1;i++) {
            output << point[arbre[i][0]][0] << " " << point[arbre[i][0]][1]
                   << " moveto" << endl;
            output << point[arbre[i][1]][0] << " " << point[arbre[i][1]][1]
                   << " lineto " << endl;
            output << "stroke " << endl;
            output << endl;
        }
        output.close();
}
```

qui affiche dans le fichier « Tree.ps » l'arbre couvrant que vous avez construit. Vous obtiendrez un résultat de ce genre :



6 Diamètre

Calculer le diamètre de l'arbre couvrant que vous avez construit avec l'algorithme de double parcours en profondeur. Comment évolue-t-il par rapport à n ?

7 Voyageur de commerce

Comment se servir de notre algorithme de calcul d'arbre couvrant pour obtenir une 2-approximation du problème de voyageur de commerce ?