1 Arbre couvrant de poids minimum (ACPM)

1.1 Définition du problème

Soit $G = (V, E, w)$ un graphe. w *(pour weight)* est la fonction de pondération *(le poids)* des arêtes.

$$w : E \mapsto \mathbb{R}^+$$

Formellement, trouver un ACPM revient à minimiser

$$w(H) = \sum_{e \in E(H)} w(e)$$

avec $H \subseteq G$.
(H est un arbre couvrant : un graphe partiel de G, acyclique maximal)

1.2 Méta-algorithme

- Règle Rouge : Sélectionner un cycle C sans aucune arête rouge. Colorier en rouge l'arête e sans couleur de C qui a le poids maximum.
- Règle Bleue : Sélectionner une coupe $\partial(x)$ qui ne comporte aucune arête bleue. Colorier en bleu l'arête sans couleur de poids minimum de $\partial(x)$. $\partial(x)$ L'ensemble des arêtes qui ont une extrémité dans x et l'autre dans $\overline{x}$.

1.3 Algorithmes

Algorithm 1 Kruskal

Entrée : $G = (V, E, w)$, un graphe pondéré
Sortie : $T = (V, F)$ un Arbre Couvrant de Poids Minimum de G

Complexité : $O(m \log_2(n) + n + m \cdot \alpha(n, m))$

$L =$ la liste des arêtes triées par poids croissant

for each $e \in L$ (pris dans l'ordre) **do**
  if e a ses deux extrémités dans deux Composantes Connexes différentes **then**
    $F := F \cup \{e\}$

return F

Algorithm 2 Jarnik-Prim

Entrée : $G = (V, E, w)$, un graphe pondéré
Sortie : $T = (V, F)$ un Arbre Couvrant de Poids Minimum de G

Complexité : $O(n \log_2(n) + m)$ *avec tas de Fibonacci*

Soit s un sommet de G quelconque.

$F := \emptyset$

$C := \{s\}$

while $|C| \neq n - 1$ **do**
  On applique la règle bleue sur $\partial(C)$

  Soit e l'arête choisie à l'instruction précédente
  $F = F \cup \{e\}$ $(avec\ e = \{x, y\})$
  $C = C \cup \{x, y\}$

return F

2 Plus courts chemins

2.1 Définition du problème

Trouver tous les plus courts chemins à partir d'un sommet s vers tous les autres. (Plus court chemin à Source Unique) Le résultat formera un Arbre des plus courts chemins enraciné en s.

Algorithm 3 Dijkstra

Entrée : $G = (V, E, w)$, un graphe pondéré et s un sommet de départ
Sortie : $T = (V, F)$ un Arbre de Plus Court Chemin

Complexité :

Partie 1 : Initialisation
for each $v \in V(G)$ **do**
    $d[v] := +\infty$
    $p[v] := \emptyset$

$d[s] := 0$
In $:= \emptyset$
Out $:= V(G)$

Partie 2 : Calcul des plus courts chemins
while $|\text{In}| < n$ **do**
    Soit x le sommet de Out avec la plus petite valeur $d[x]$
    In $:=$ In $\cup\{x\}$
    Out $:=$ Out$\setminus\{x\}$

    for each $z \in N(x)$ **do**
        if $(d[z] > d[x] + w(x, z)$ **then**
            $d[v] := d[x] + w(x, z)$
            $p[z] := x$

return p

Algorithm 4 Bellman *(Bellman-Ford)*

Entrée : $G = (V, E)$, un graphe orienté valué et un sommet s
Sortie : T une arborescence de plus court chemin ou Impossible si pas possible

Complexité : $O(n \times m)$

Partie 1 : Initialisation
for each $v \in V(G)$ **do**
  $d[v] := +\infty$
  $p[v] := \emptyset$

Partie 2 : Calcul des plus courts chemins
for $i = 1$ to $|V(G)| - 1$ **do**
  for each arc (u, v) **do**
    if $(d[v] > d[u] + w(u, v))$ **then**
      $d[v] := d[u] + w(u, v)$
      $p[v] := u$

Partie 3 : Détection de cycle de poids négatif
for each arc (u, v) **do**
  if $(d[u] > d[u] + w(u, v))$ **then**
    return Impossible *(cycle de poids négatif)*

return p

Algorithm 5 Edmonds-Karp

Au lieu de prendre n'importe quel chemin améliorant de s à t, on prend le plus court en nombre d'arcs et on fait un BFS sur le graphe résiduel G_f.

Entrée : $G = (V, E, c)$, un graphe avec une capacité pour chaque arc
Sortie : f, le flot maximum

Complexité :

while $\exists P$ un chemin $f-$améliorant dans G **do**
  On augmente le flot le long de P

return f