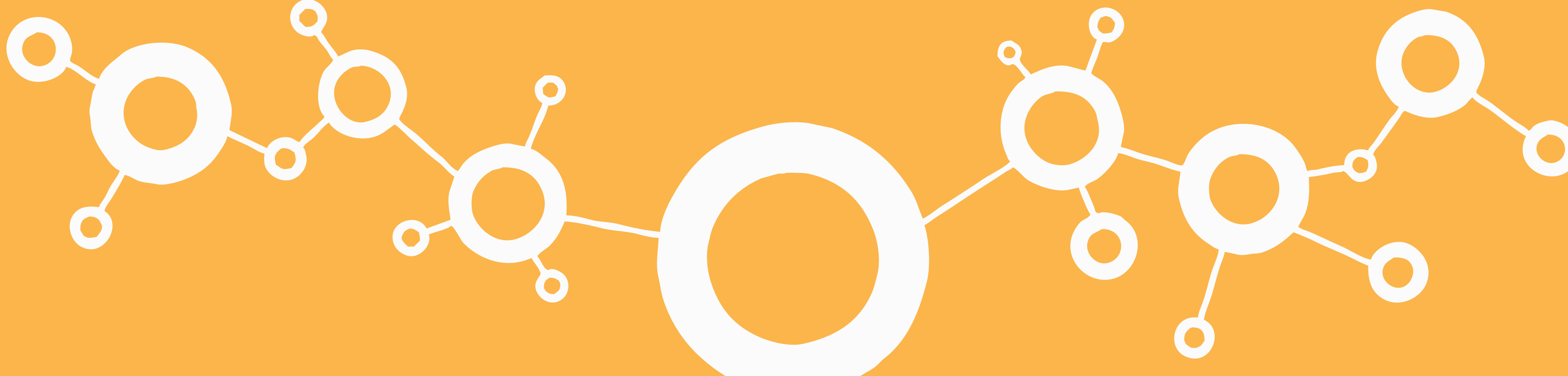
Ch. 7 : Parcours, arbres couvrants et plus courts chemins



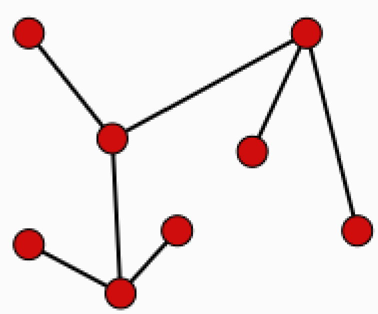
Parcours de graphes

Arbres

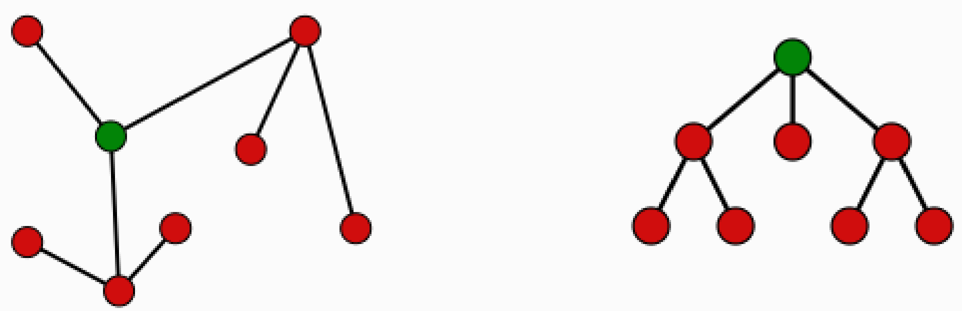
TERMINOLOGIE

Arbre

- mathématiques : *graphe* connexe (= « en un morceau ») et sans cycle
- informatique : arbre *enraciné* ou *arborescence*



arbre



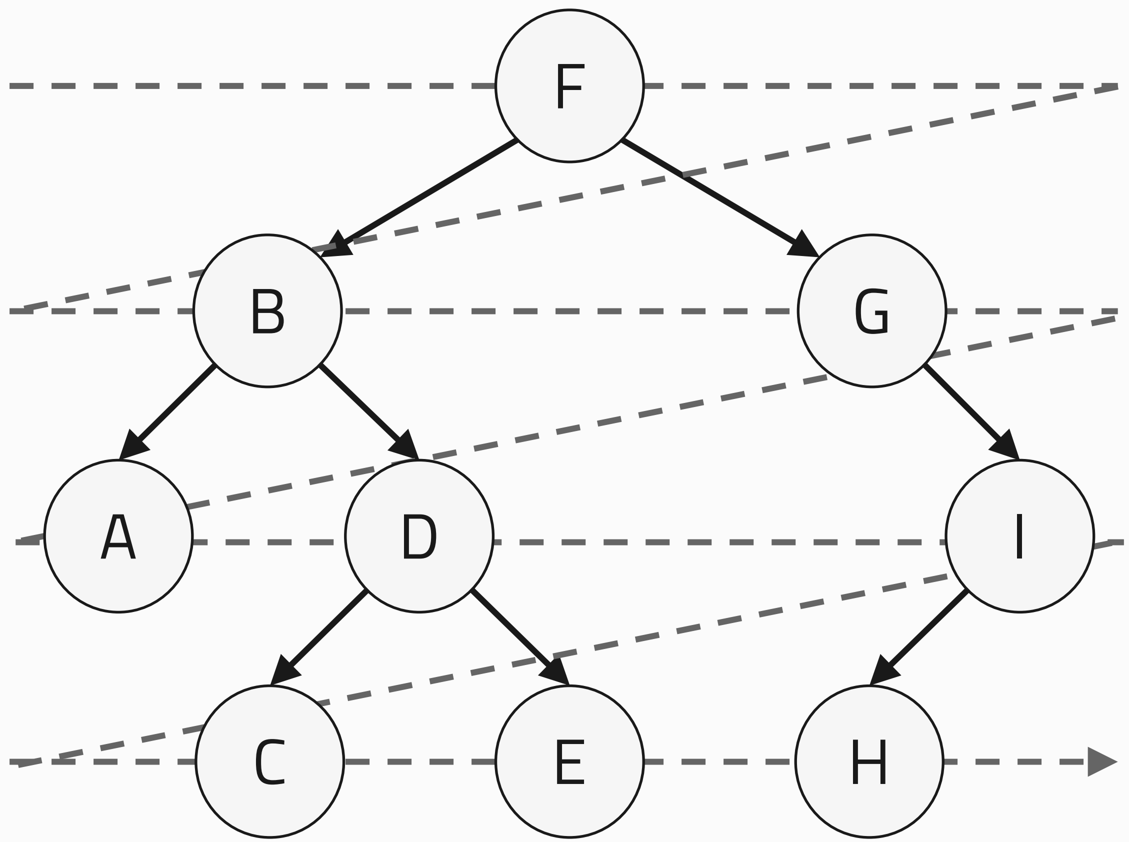
arbre enraciné ou arborescence

Un arbre est une généralisation d'une liste chaînée

Arbres : parcours en largeur

OU BFS (BREADTH-FIRST SEARCH)

Principe : on parcourt les nœuds niveau par niveau, de gauche à droite :

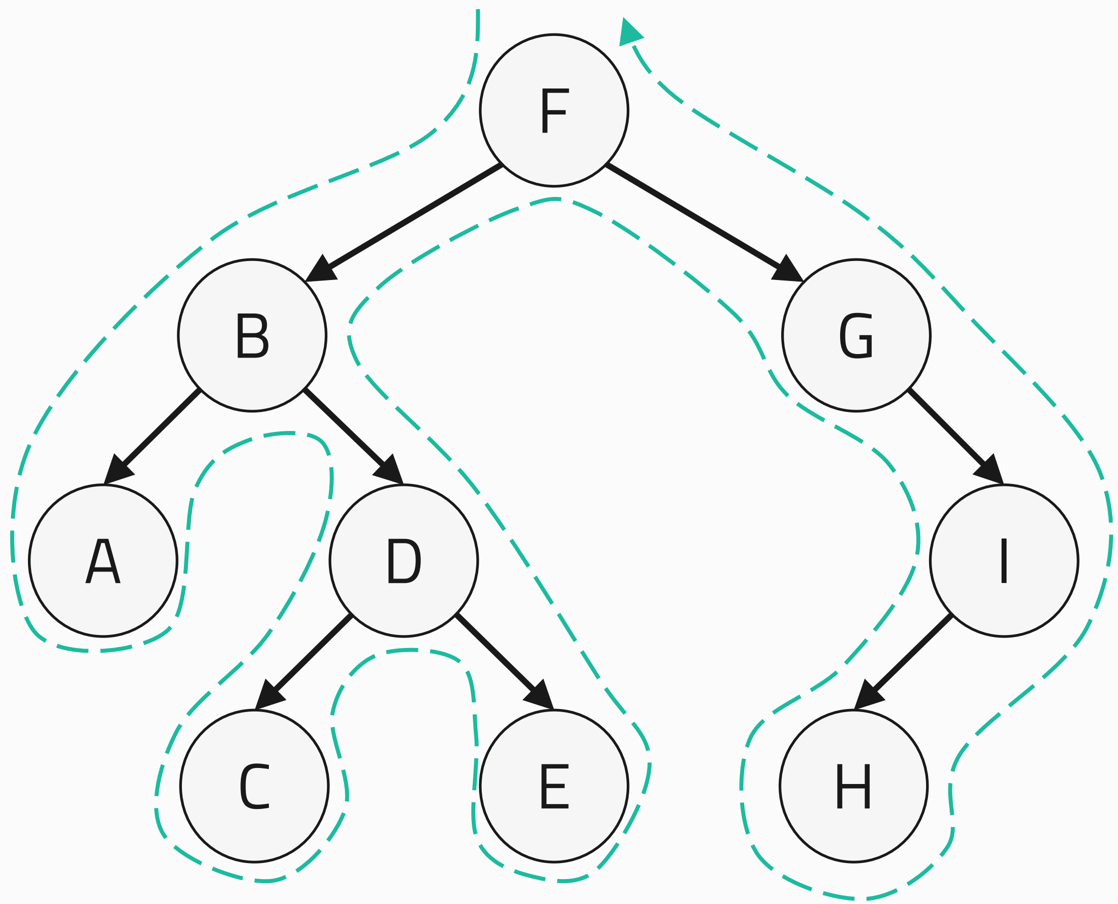


Ordre de parcours : F B G A D I C E H

Arbres : parcours en profondeur

OU DFS (DEPTH-FIRST SEARCH)

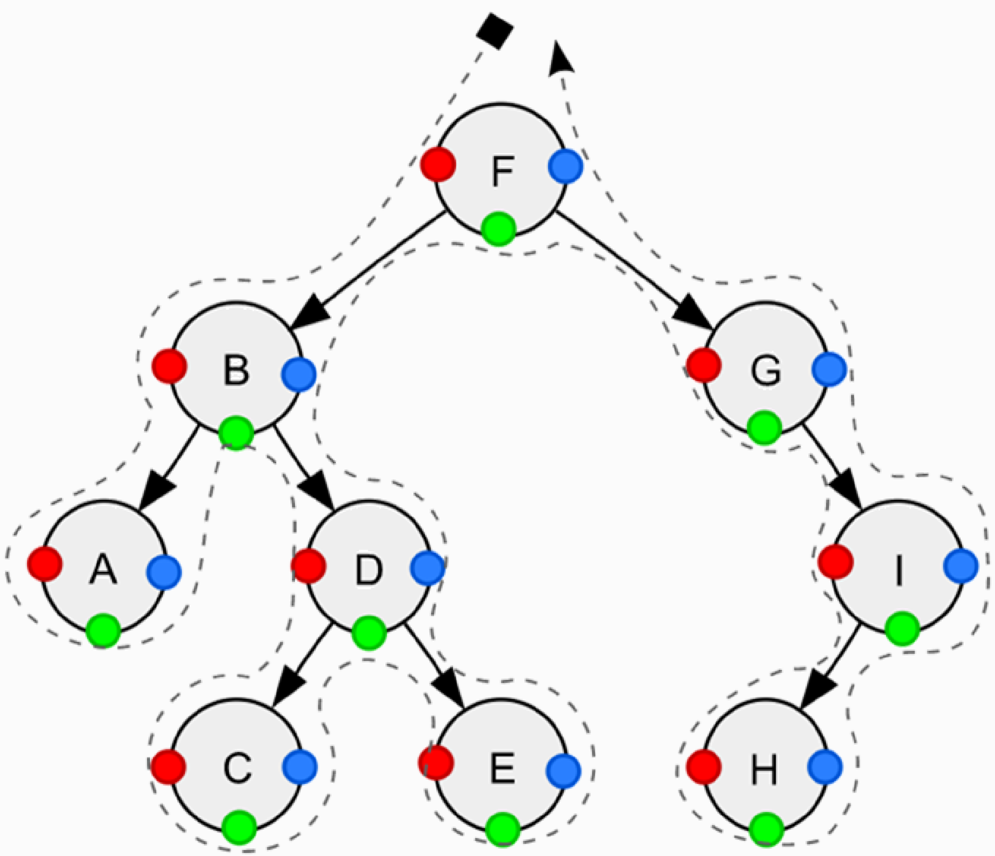
Principe : on descend le plus possible vers les enfants d'un nœud avant de passer au nœud frère



Arbres : parcours en profondeur

OU DFS (DEPTH-FIRST SEARCH)

3 ordres possibles pour marquer les sommets rencontrés :



Ordre préfixe : on note un nœud dès qu'on passe à gauche :

F B A D C E G I H

Ordre infixe : on note un nœud dès qu'on passe dessous :

A B C D E F G H I

Ordre postfixe : on note un nœud dès qu'on passe à droite :

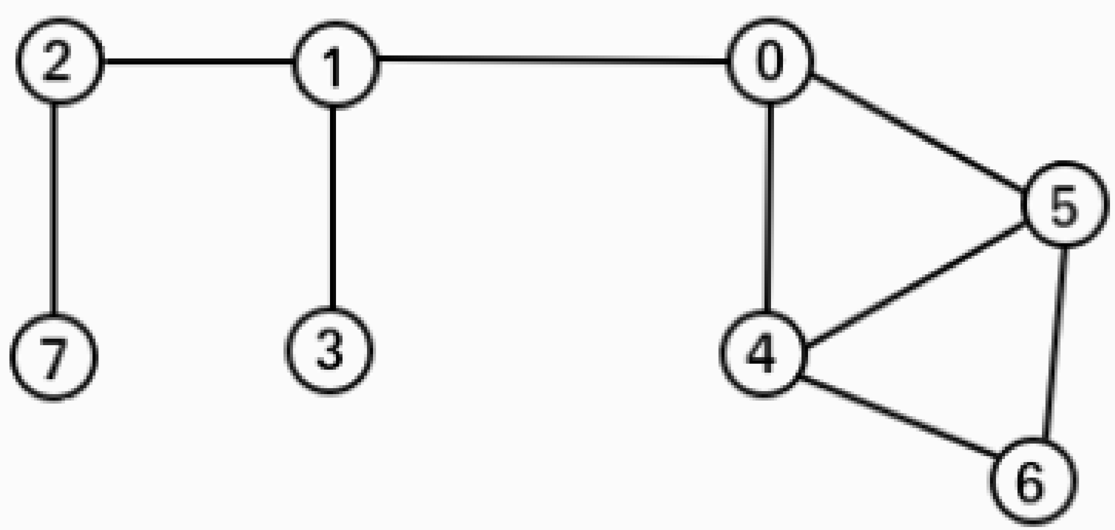
A C E D B G H I F

Parcours en largeur d'un graphe

OU BFS (BREADTH-FIRST SEARCH)

Principe : comme pour les arbres, on procède par *distance depuis un sommet de départ*

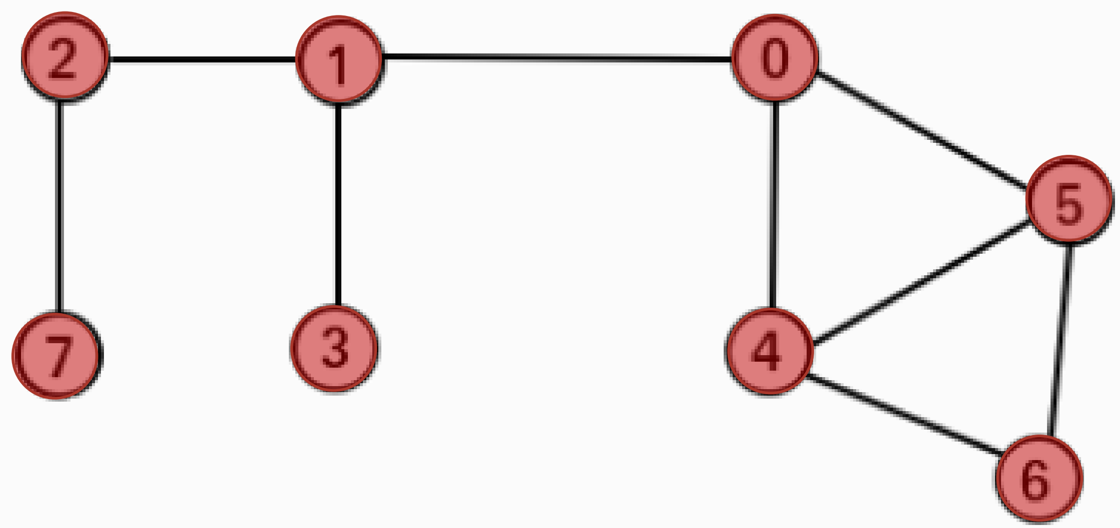
- on note un sommet de départ
- puis tous ses voisins
- puis tous les voisins de ses voisins
- etc.



Parcours en largeur

OU BFS (BREADTH-FIRST SEARCH)

Implémentation : on utilise une file d'attente



Exemple : parcours en largeur depuis le sommet 0

File d'attente

0	1	4	5	2	3	6	7

Parcours en profondeur

OU DFS (DEPTH-FIRST SEARCH)

Principe : comme pour les arbres, on va de plus en plus loin avant de passer à un nœud voisin.

Implémentation non récursive :

1. on ajoute un nœud de départ *s* dans une pile P et on le marque comme « visité »
2. on pousse dans P un voisin non visité du nœud au sommet de la pile et on le marque comme « visité »
3. on répète l'étape 2 tant que possible, sinon on passe à l'étape 4
4. on retire le nœud au sommet de la pile

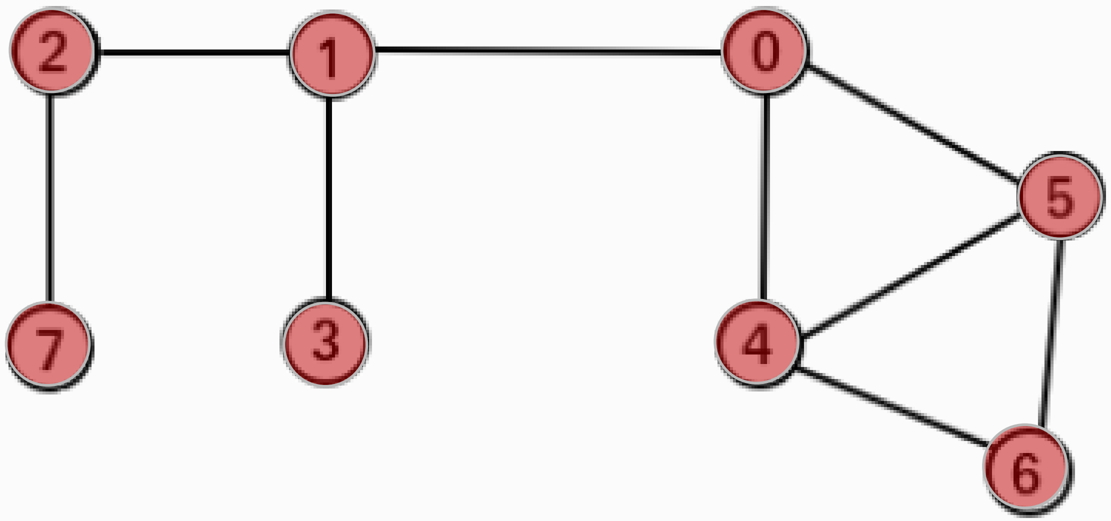
Remarque : comme pour les arbres, il existe plusieurs façons d'ordonner les nœuds dans le résultat :

- Ordre préfixe : on note les nœuds dès qu'on les insère dans la pile
- Ordre postfixe : on note les nœuds quand on les sort de la pile

Parcours en profondeur

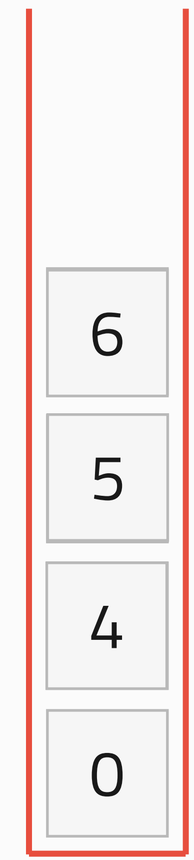
OU DFS (DEPTH-FIRST SEARCH)

Exemple : ordre postfixe



Exemple : parcours en profondeur depuis le sommet 0

7	2	3	1	6	5	4	0

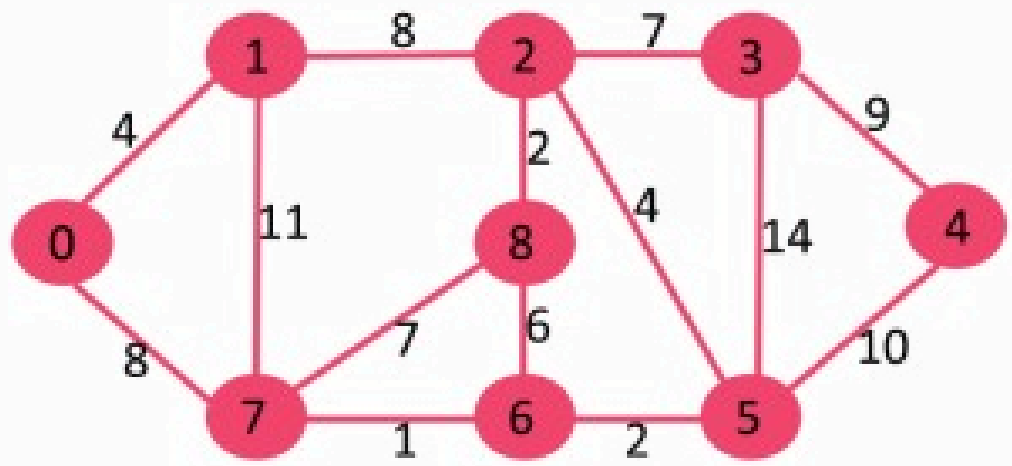


Pile

Problème de l’arbre couvrant de poids minimal

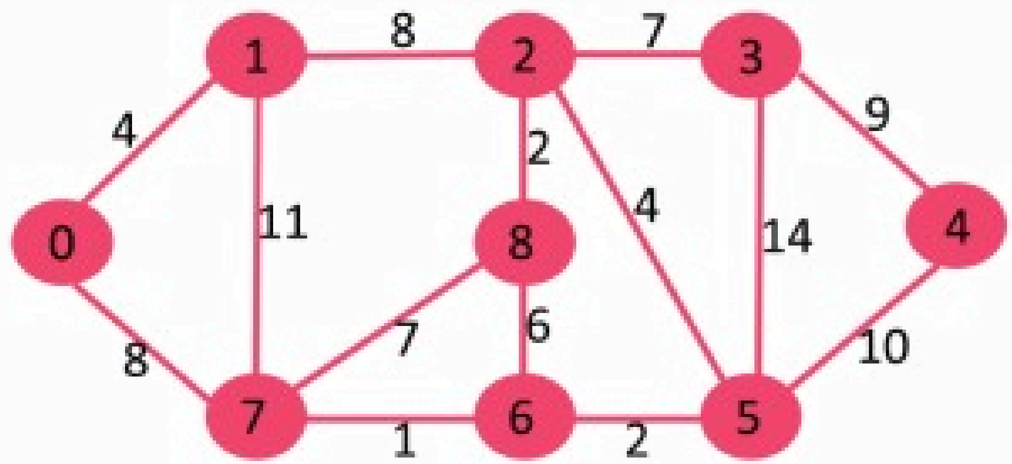
Graphe pondéré *(weighted graph)*

Graphe dans lequel chaque arête possède un *poids*



Enigme du jour

Une entreprise dispose d'un ensemble de filiales, dont certaines sont reliées entre elles par des routes (exprimées ici en centaines de km) :

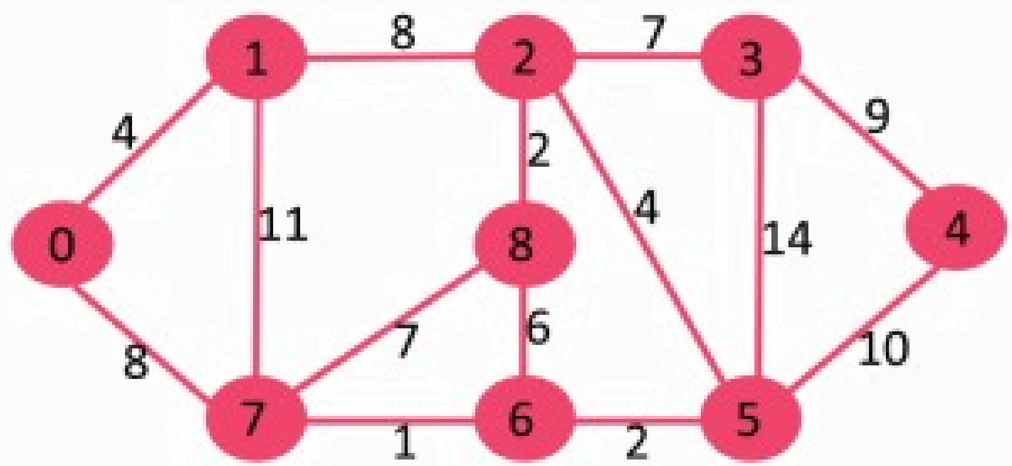


Cette entreprise souhaite connecter l'ensemble de ses filiales par des liaisons ultra rapides et hautement sécurisées. Chaque liaison ne peut se faire qu'en suivant les tronçons routiers (pour des raisons de coûts d'infrastructures) et le coût d'une liaison est proportionnel à la longueur de la route. Quelle est la configuration optimale, càd celle qui coûtera le moins cher à l'entreprise ?

Arbre couvrant de poids minimal

MODELISATION

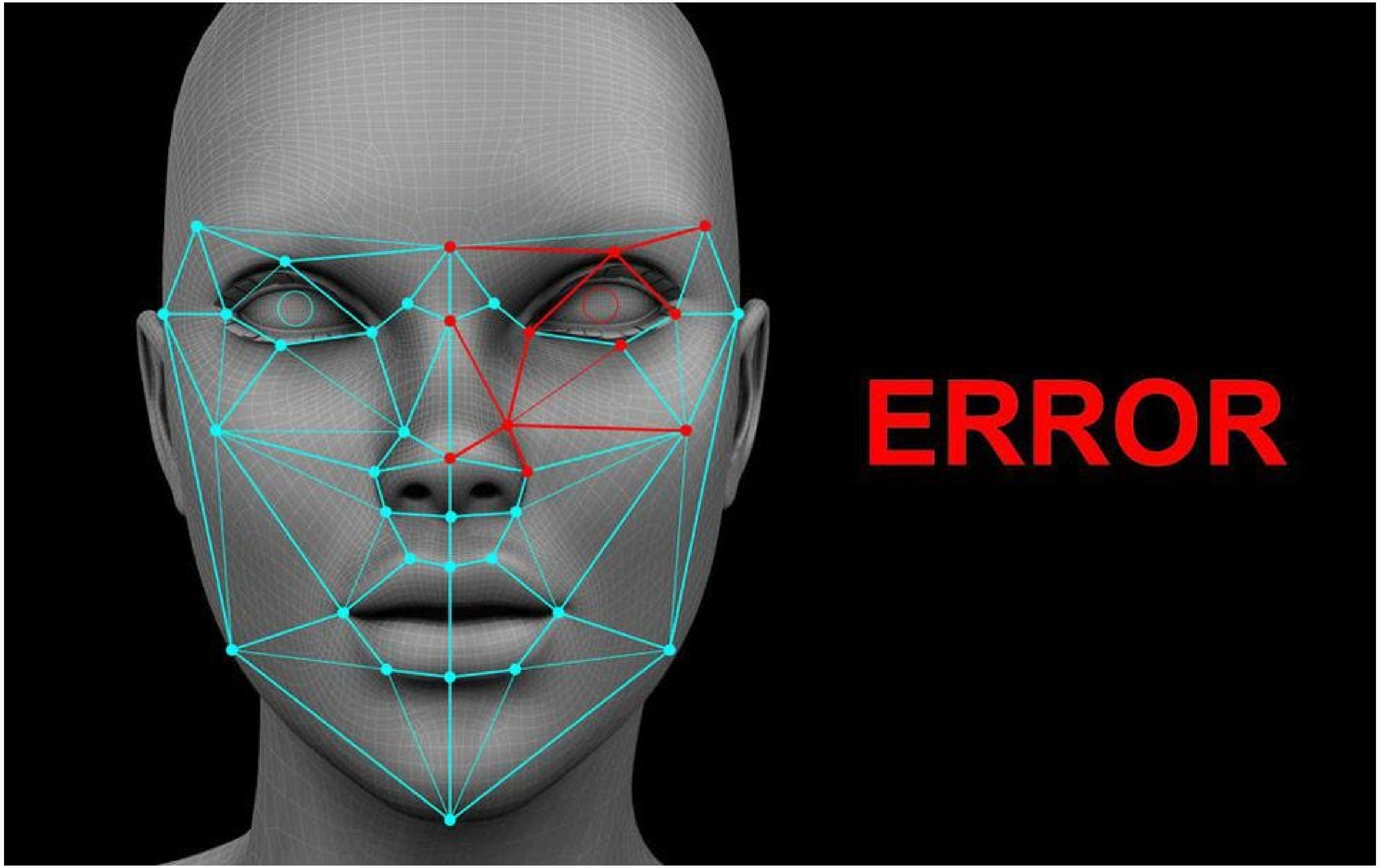
On modélise le problème par un graphe pondéré (ou *réseau*) *connexe* :



- Problème de l'arbre couvrant de poids minimum : trouver le sous-graphe *couvrant* (càd touchant tous les sommets) connexe de poids minimum
- Pourquoi est-ce nécessairement un *arbre* ?

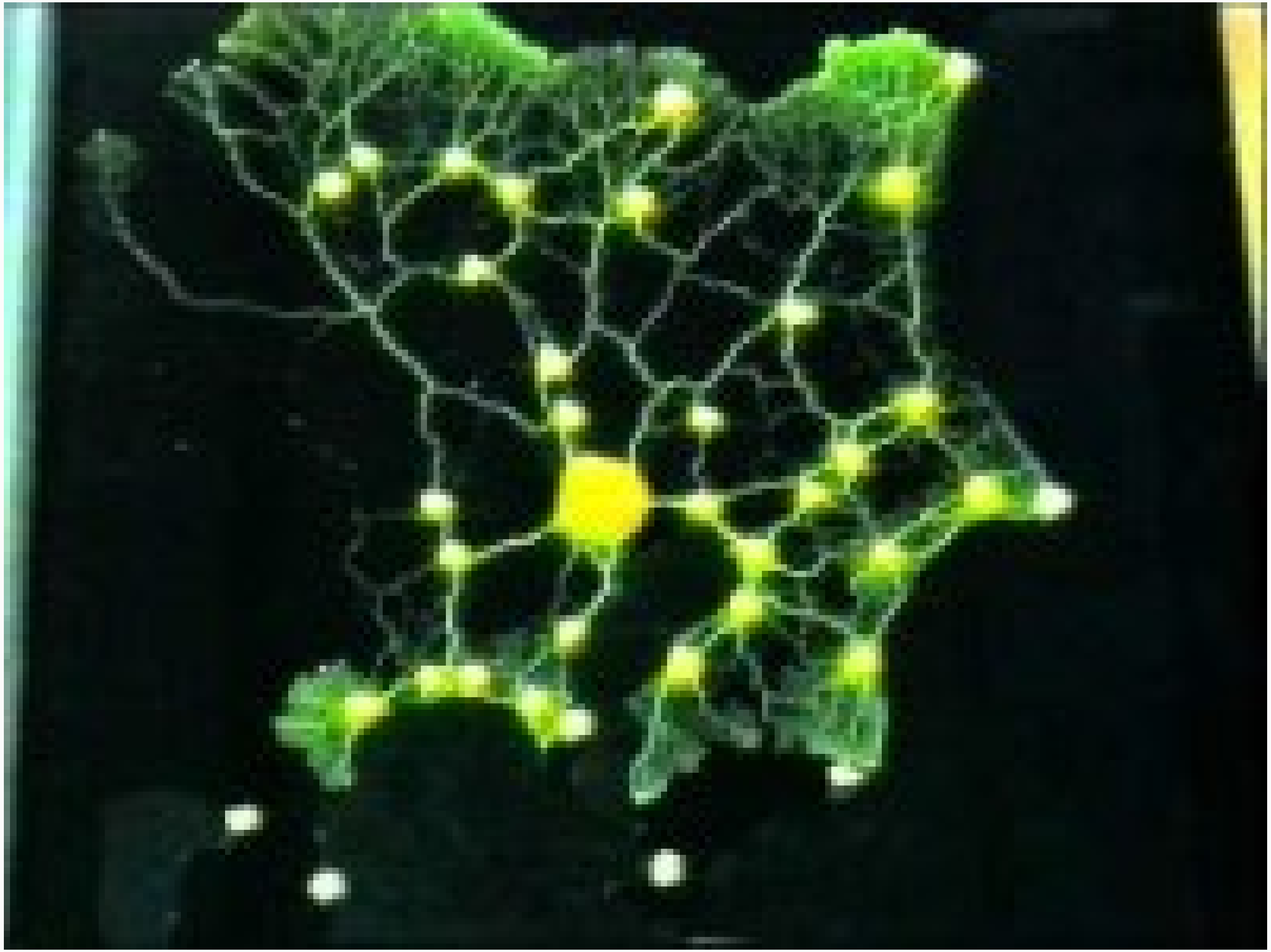
Arbre couvrant de poids minimal

UNE AUTRE APPLICATION



Arbre couvrant de poids minimal

RESOLUTION A L'AIDE D'UN BLOB (EN 26 HEURES…)



Arbre couvrant de poids minimal

ALGORITHME DE KRUSKAL

Algorithme de Kruskal :

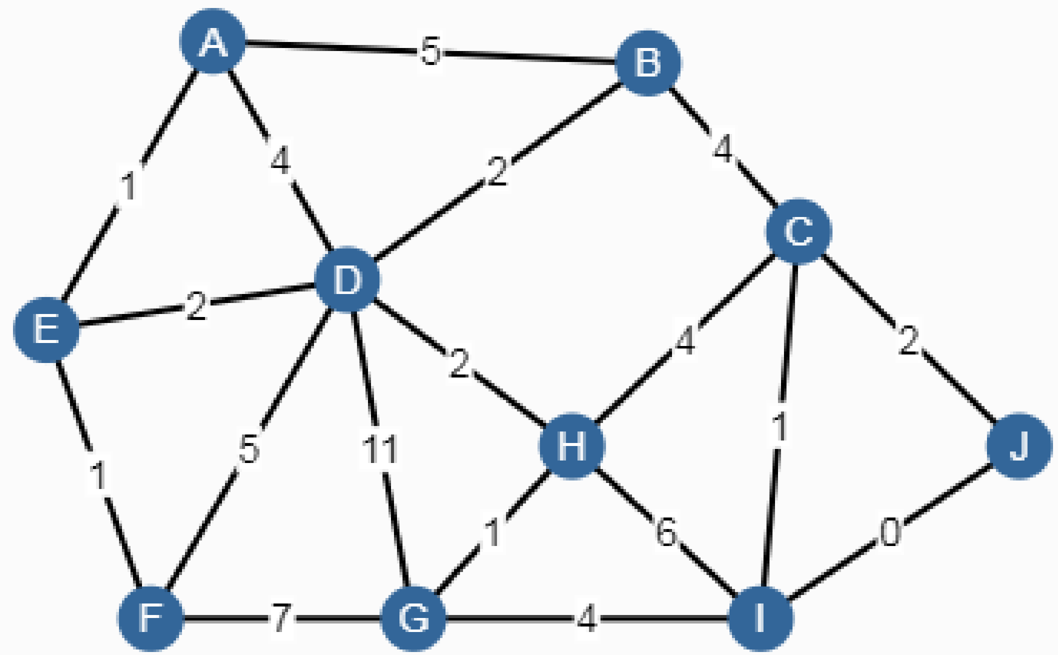
1. Trier les arêtes par poids croissant
2. Prendre l'arête de poids minimum, et vérifier si elle forme un cycle avec l'arbre déjà construit :
   - si *non*, l'ajouter à l'arbre
   - si *oui*, la rejeter
3. Répéter l'étape 2 jusqu'à ce que tous les sommets soient reliés, càd avoir $n - 1$ arêtes dans l'arbre

Cas d'algorithme glouton optimal !

Arbre couvrant de poids minimal

ALGORITHME DE KRUSKAL

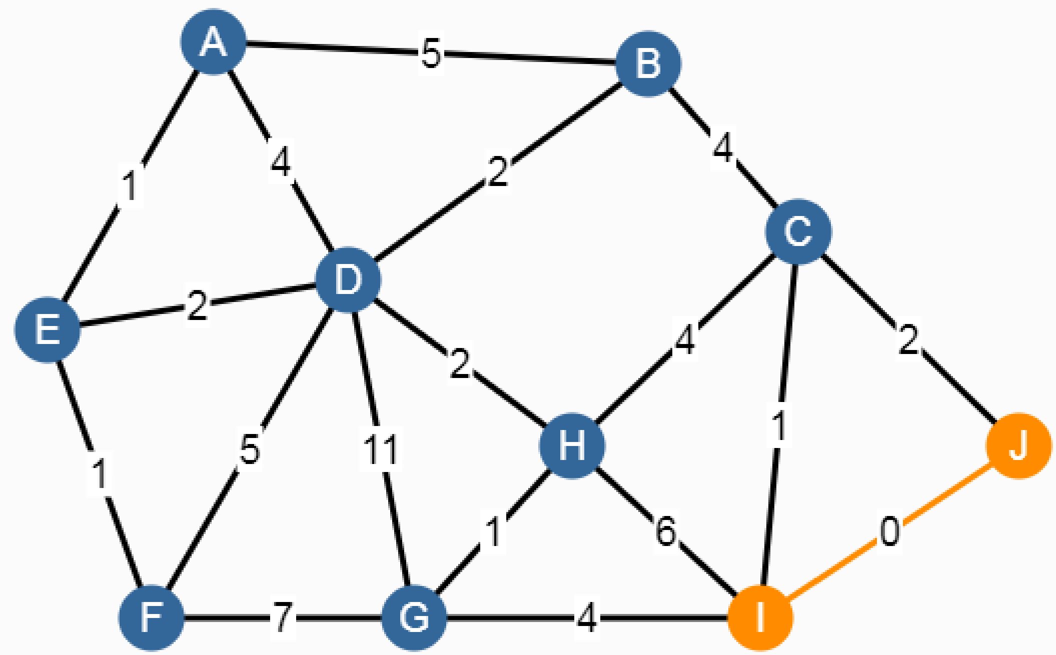
Exemple :



Arbre couvrant de poids minimal

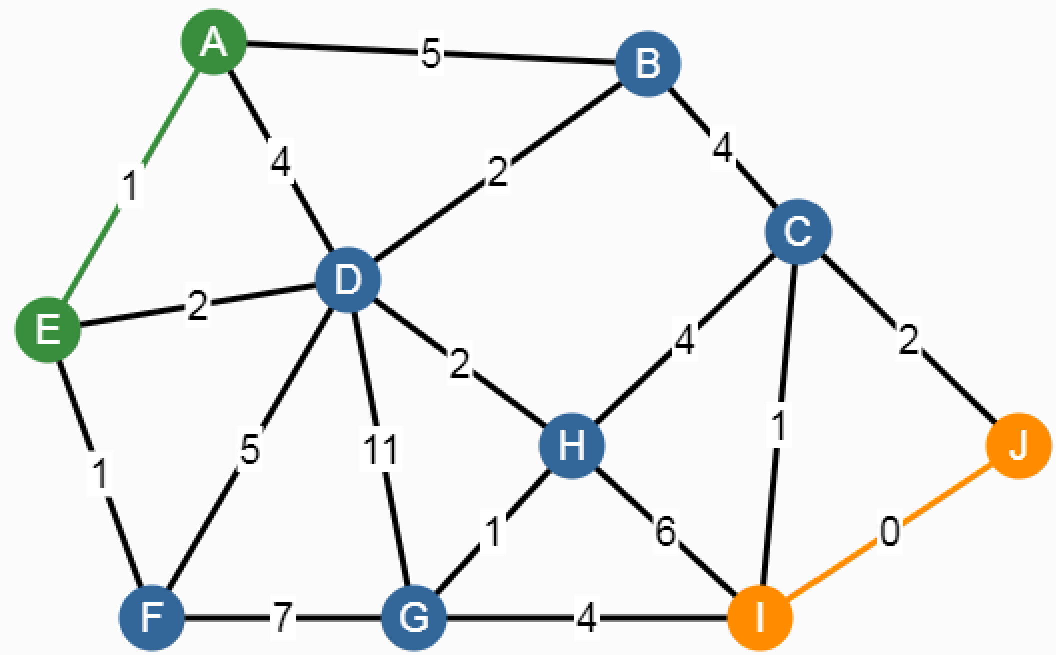
ALGORITHME DE KRUSKAL

Solution :



Arbre couvrant de poids minimal

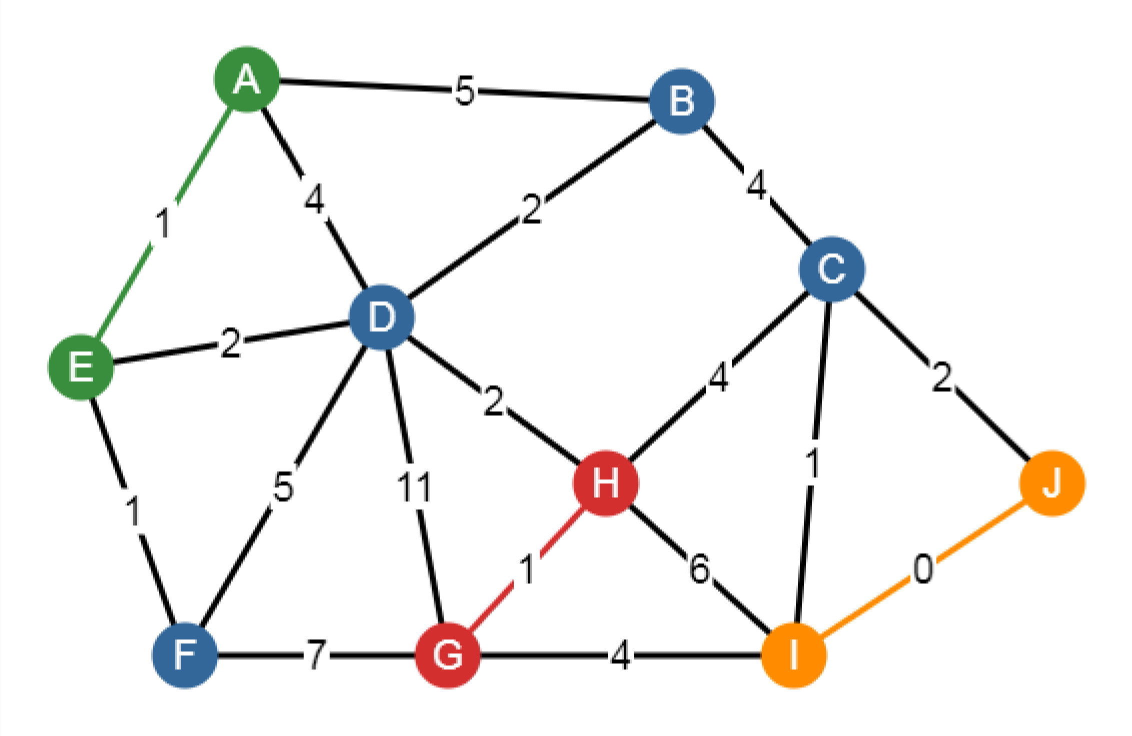
Solution :



Arbre couvrant de poids minimal

ALGORITHME DE KRUSKAL

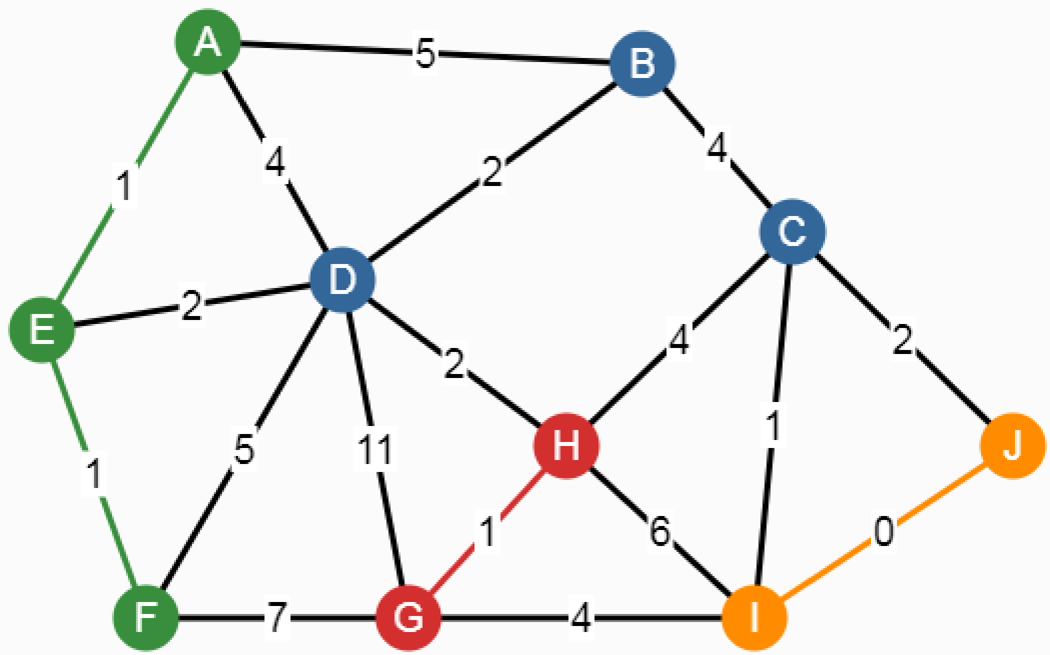
Solution :



Arbre couvrant de poids minimal

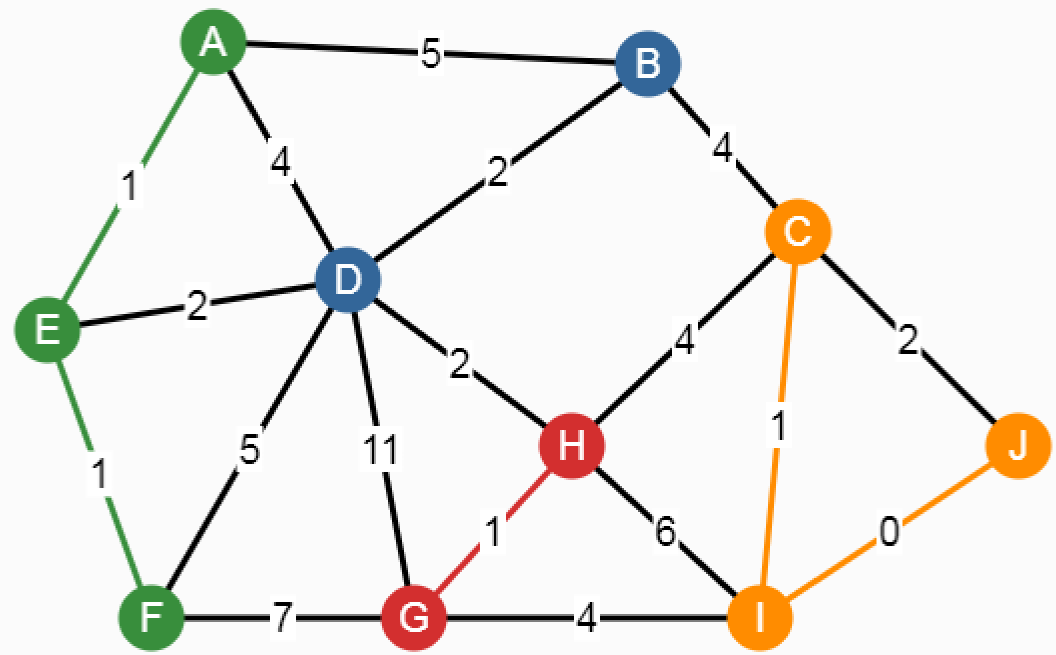
ALGORITHME DE KRUSKAL

Solution :



Arbre couvrant de poids minimal

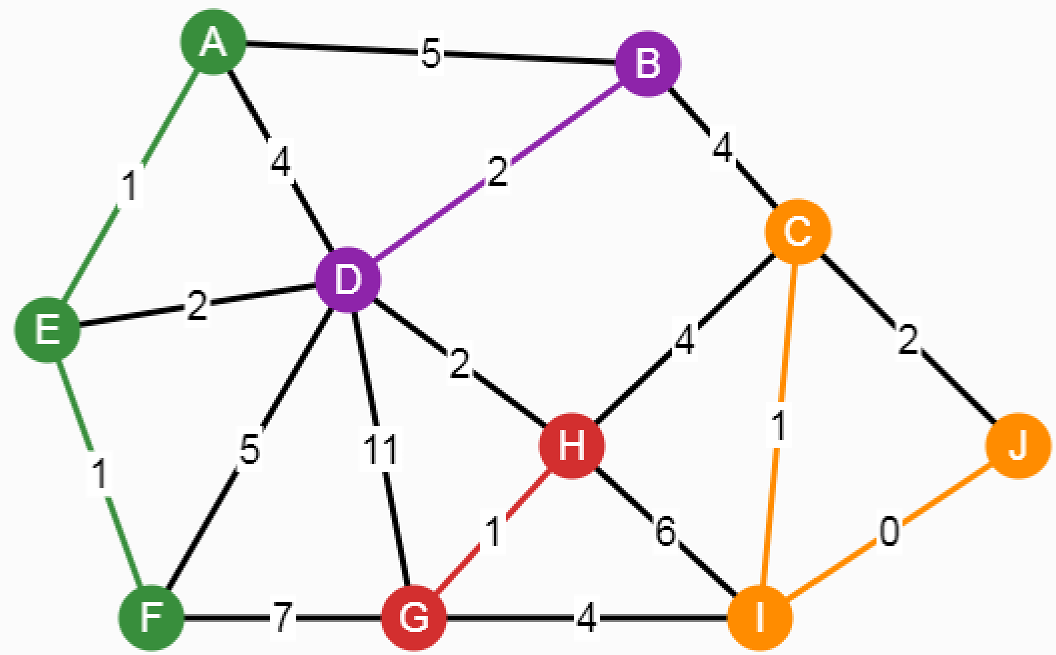
Solution :



Arbre couvrant de poids minimal

ALGORITHME DE KRUSKAL

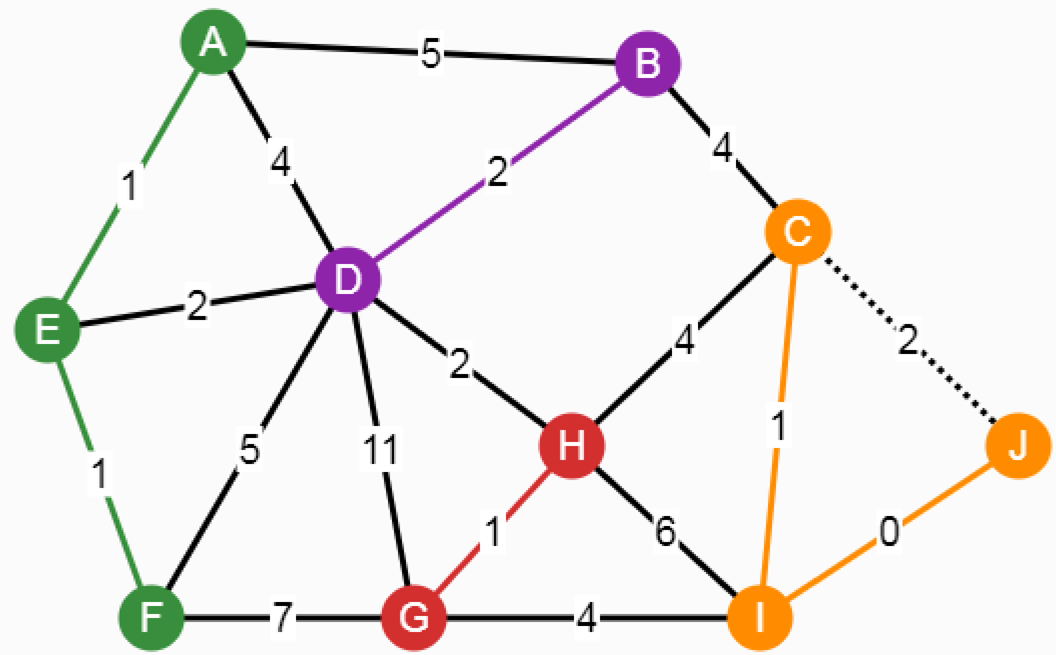
Solution :



Arbre couvrant de poids minimal

ALGORITHME DE KRUSKAL

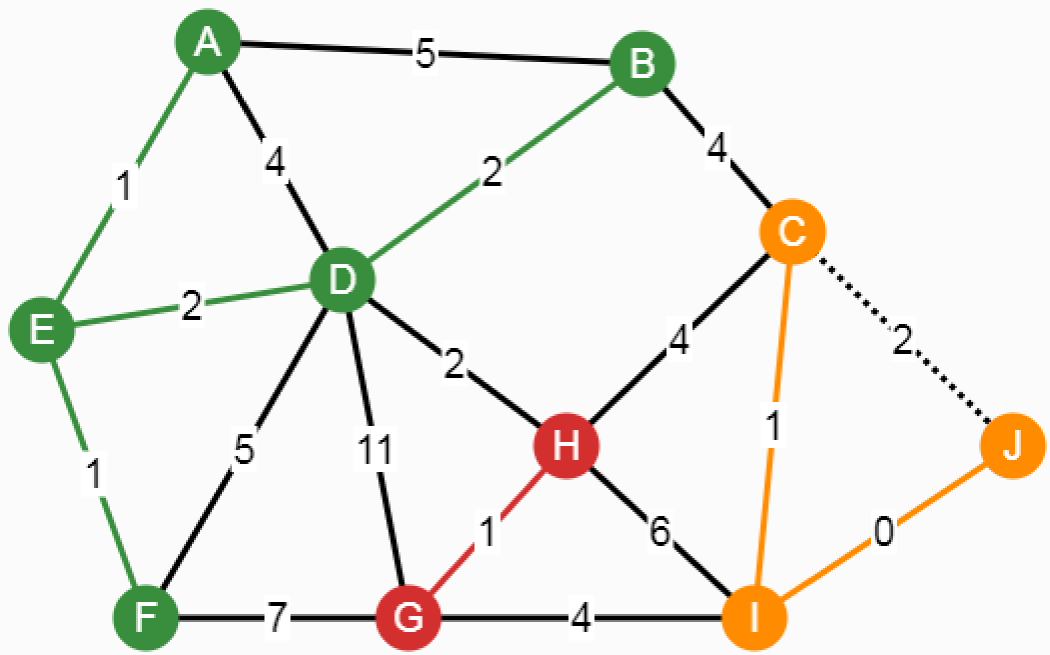
Solution :



Arbre couvrant de poids minimal

ALGORITHME DE KRUSKAL

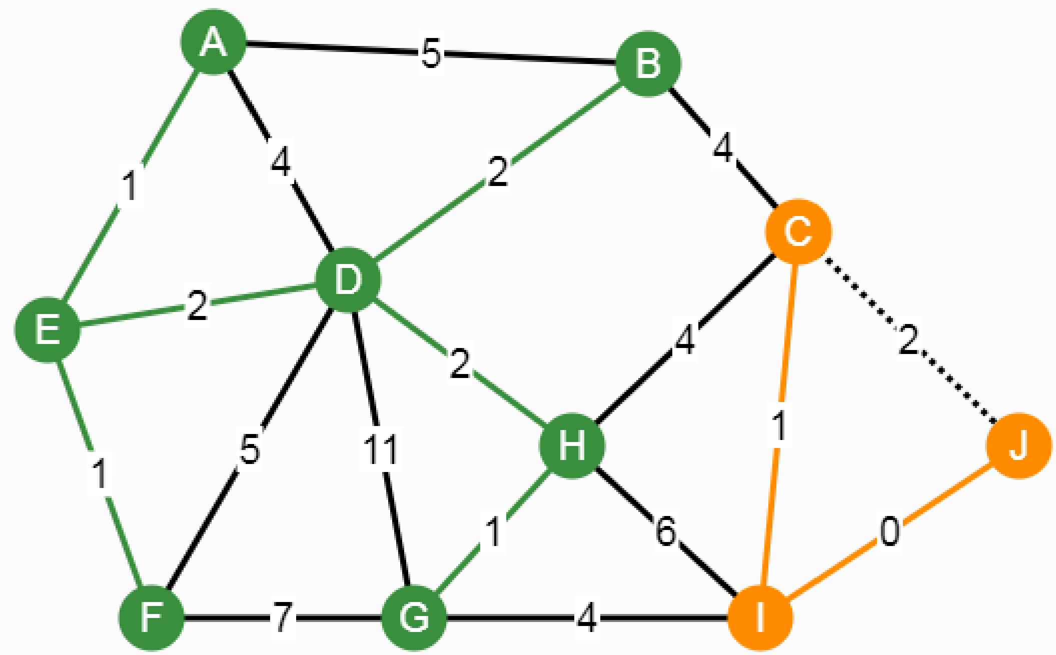
Solution :



Arbre couvrant de poids minimal

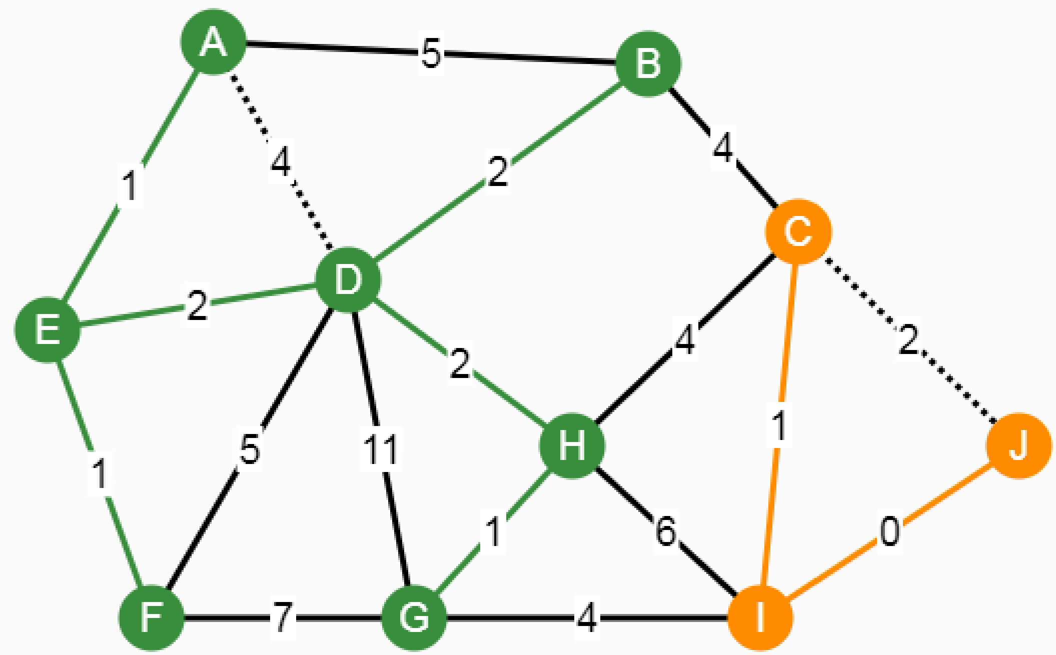
ALGORITHME DE KRUSKAL

Solution :



Arbre couvrant de poids minimal

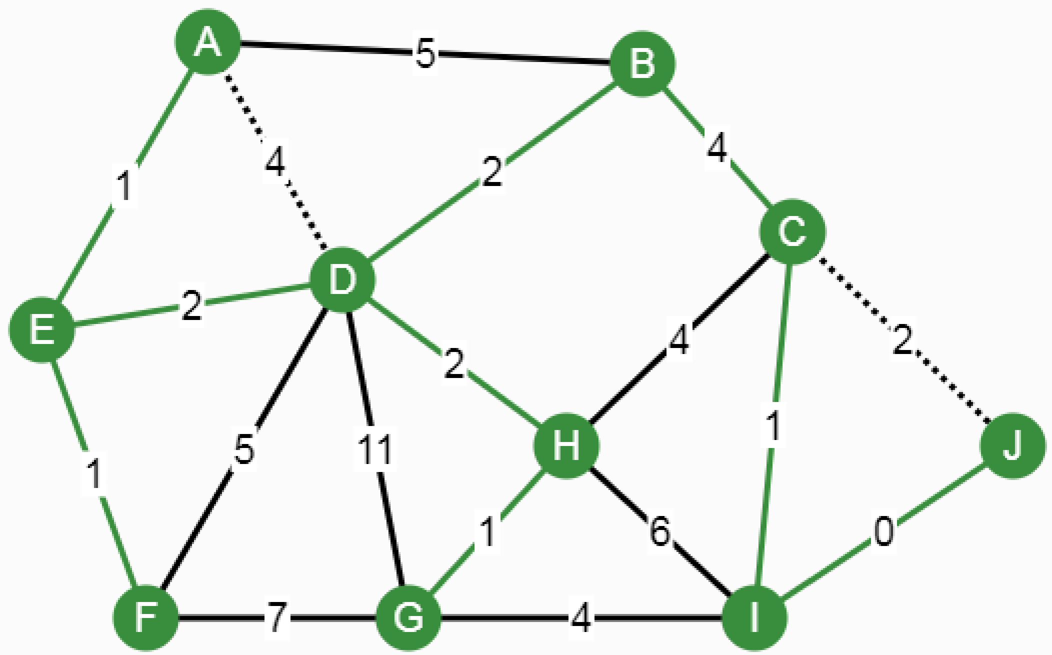
Solution :



Arbre couvrant de poids minimal

ALGORITHME DE KRUSKAL

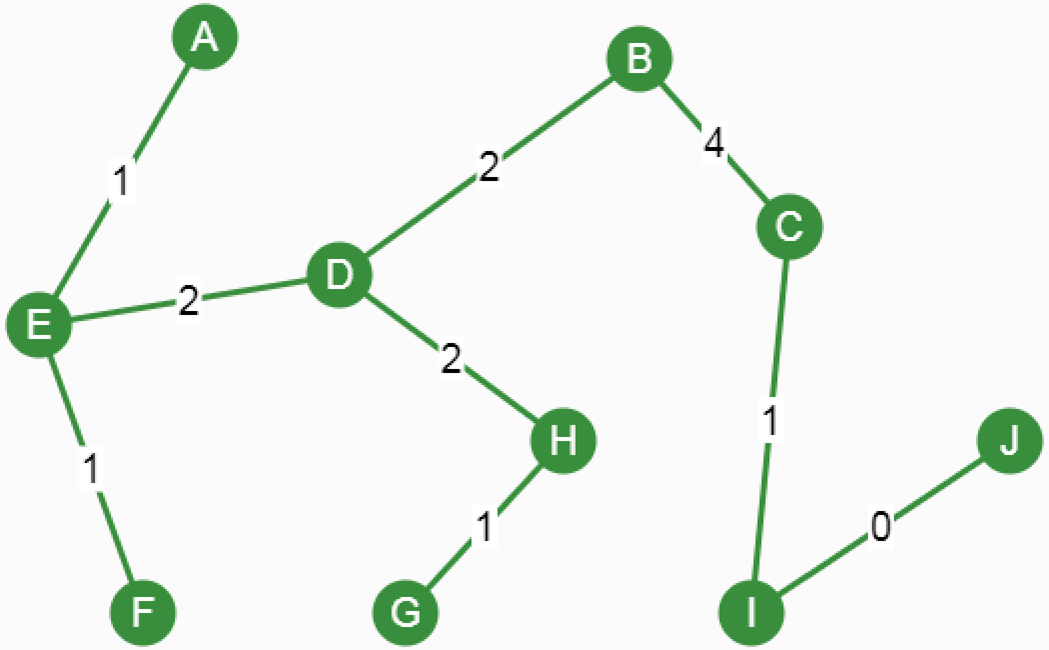
Solution :



Arbre couvrant de poids minimal

ALGORITHME DE KRUSKAL

Solution : arbre couvrant de poids 0 + 1 + 1 + 1 + 1 + 2 + 2 + 2 + 4 = 14



Question : comment détecter automatiquement les cycles ?

Réponse dans le TP !

Arbre couvrant de poids minimal

Un second algorithme : l'Algorithme de Prim :

1. Initialiser l'arbre avec un sommet arbitraire
2. Faire croître l'arbre en prenant l'arête de poids minimal reliant un sommet de l'arbre et un sommet en dehors de l'arbre
3. Répéter l'étape 3 tant qu'il reste un sommet hors de l'arbre

Autre cas d'algorithme glouton optimal !

Arbre couvrant de poids minimal

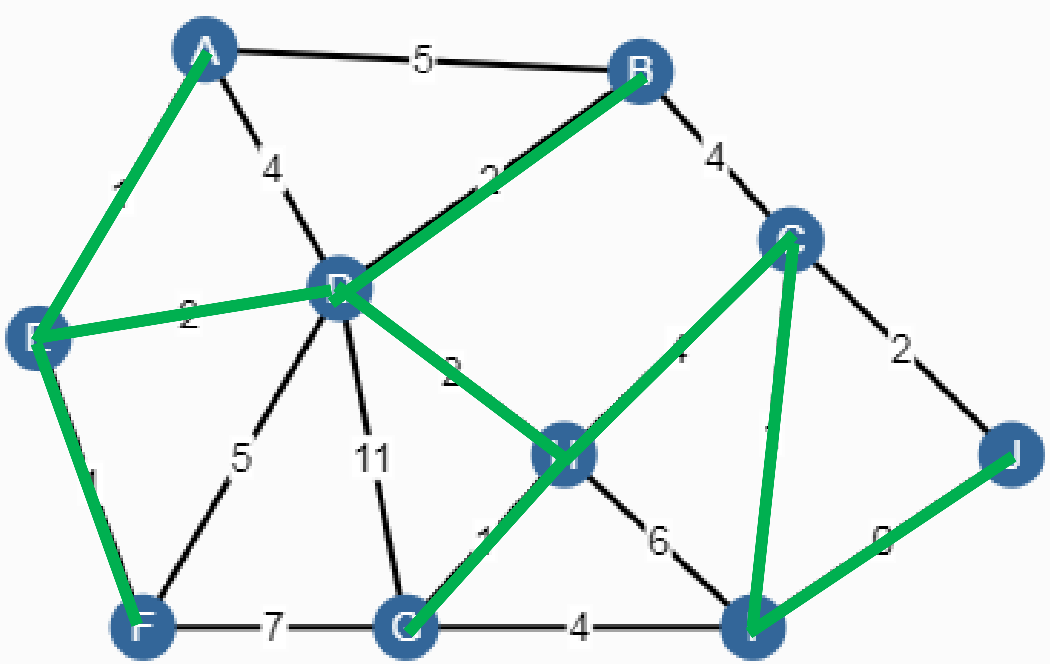
Pseudocode :

1. Créer un tableau *visités* afin de garder une trace des sommets visités, et un tableau *parents* afin de savoir quelles arêtes conserver
2. Attribuer à chaque sommet un *poids* égal à $+\infty$ sauf pour le sommet de départ, initialisé à 0
3. Tant qu'il existe un sommet qui n'est pas dans *visités*
   a) Prendre un sommet *v* absent de *visités* ayant le poids minimal
   b) L'ajouter à *visités*
   c) Mettre à jour les poids des voisins de *v* :
   pour chaque arête v-w, si le poids de cette arête est inférieur au poids actuel de w, mettre à jour le poids de w avec le poids de l'arête, et *parents[w]* = *v*

Arbre couvrant de poids minimal

ALGORITHME DE PRIM

Exemple : on part du sommet D



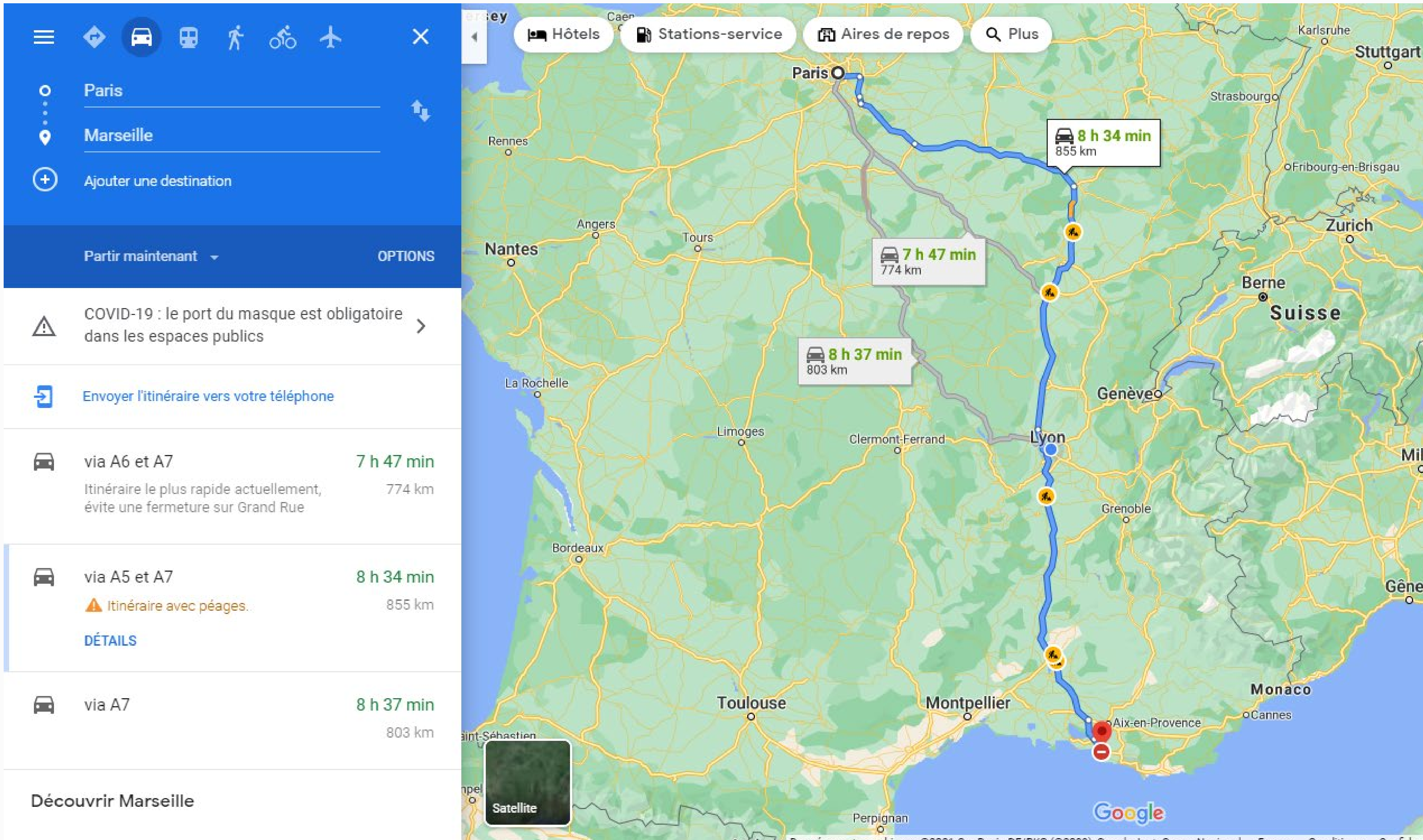
	A	B	C	D	E	F	G	H	I	J
Poids	∞	∞	∞	0	∞	∞	∞	∞	∞	∞
Poids	4/D	2/D	∞	0	2/D	5/D	11/D	2/D	∞	∞
Poids	4/D	2/D	4/B	0	2/D	5/D	11/D	2/D	∞	∞
Poids	1/E	2/D	4/B	0	2/D	1/E	11/D	2/D	∞	∞
Poids	1/E	2/D	4/B	0	2/D	1/E	11/D	2/D	∞	∞
Poids	1/E	2/D	4/B	0	2/D	1/E	7/F	2/D	∞	∞
Poids	1/E	2/D	4/B	0	2/D	1/E	1/H	2/D	6/H	∞
Poids	1/E	2/D	4/B	0	2/D	1/E	1/H	2/D	4/G	∞
Poids	1/E	2/D	4/B	0	2/D	1/E	1/H	2/D	1/C	2/C
Poids	1/E	2/D	4/B	0	2/D	1/E	1/H	2/D	1/C	0/I
Poids	1/E	2/D	4/B	0	2/D	1/E	1/H	2/D	1/C	0/I

En jaune : sommet dont le poids a diminué

7.3 Problème du plus court chemin entre deux sommets

Problème du plus court chemin

COMMENT FONCTIONNE GOOGLE MAPS ?



Problème du plus court chemin

ALGORITHME DE DIJKSTRA

Algorithme de Dijkstra :

1. Créer un tableau *PCC* afin de garder une trace des sommets visités
2. Attribuer à chaque sommet une *distance* $+\infty$, sauf pour le sommet de départ, initialisée à 0
3. Tant qu'il existe un sommet qui n'est pas dans PCC
   a) Prendre un sommet *v* absent de PCC ayant la distance minimale
   b) L'ajouter à PCC
   c) Mettre à jour les clés des voisins de *v* :

      pour chaque arête v-w, si $d(w) \geq d(u) + d(u, v)$, mettre à jour w

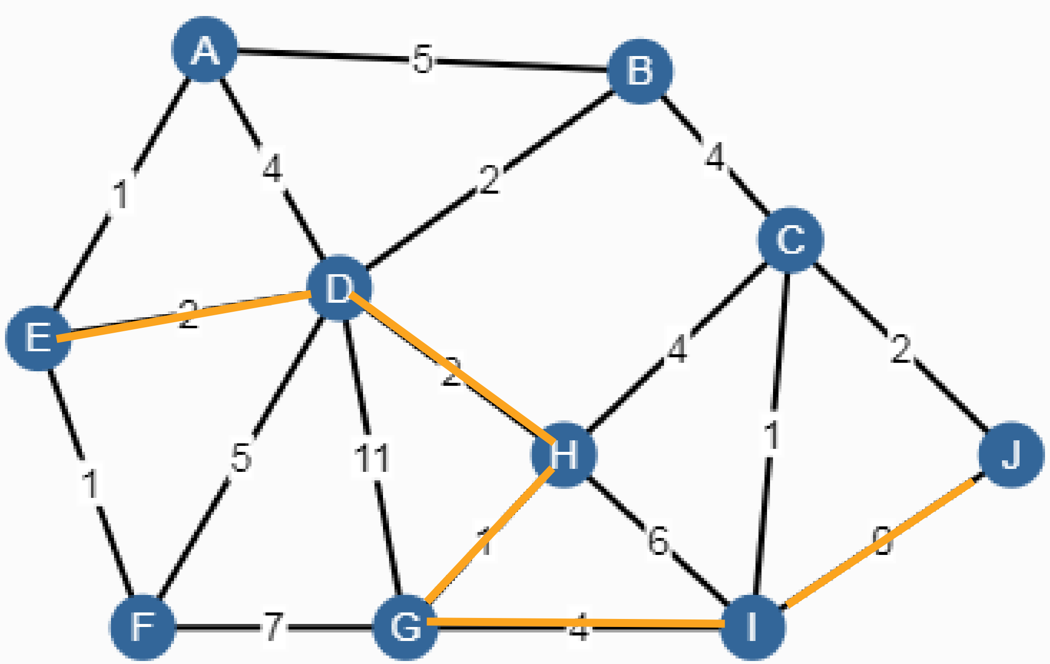
Très similaire à l'algorithme de Prim !

Donne le plus court chemin d'un sommet vers tous les autres

Problème du plus court chemin

ALGORITHME DE DIJKSTRA

Exemple : quel est le plus court chemin reliant les sommets E et J ?



	A	B	C	D	E	F	G	H	I	J
Poids	∞	∞	∞	∞	0	∞	∞	∞	∞	∞
Poids	1/E	∞	∞	2/E	0	1/E	∞	∞	∞	∞
Poids	1/E	6/A	∞	2/E	0	1/E	∞	∞	∞	∞
Poids	1/E	6/A	∞	2/E	0	1/E	8/F	∞	∞	∞
Poids	1/E	4/D	∞	2/E	0	1/E	8/F	4/D	∞	∞
Poids	1/E	4/D	8/B	2/E	0	1/E	8/F	4/D	∞	∞
Poids	1/E	4/D	8/B	2/E	0	1/E	5/H	4/D	10/H	∞
Poids	1/E	4/D	8/B	2/E	0	1/E	5/H	4/D	9/G	∞
Poids	1/E	4/D	8/B	2/E	0	1/E	5/H	4/D	9/G	10/C
Poids	1/E	4/D	8/B	2/E	0	1/E	5/H	4/D	9/G	9/I
Poids	1/E	4/D	8/B	2/E	0	1/E	5/H	4/D	9/G	9/I

En jaune : sommet dont le poids a diminué

Problème du plus court chemin

Remarques :

1. On peut arrêter l'algorithme dès que le sommet considéré est le sommet destination

2. Les poids peuvent représenter des distances, des temps de trajet, des coûts…

3. L'algorithme de Dijkstra ne fonctionne pas toujours si le graphe comporte des arêtes de poids *négatif* (peut arriver dans certaines applications)