ARBRE COUVRANT DE POIDS MINIMAL PARTIE 3 : ALGORITHME DE KRUSKAL

Alain Cournier : Licence Informatique, Université de Picardie

Algorithme de Kruskal Idées

- Il est fondé sur le second principe
- Pour chaque cycle on souhaite que les arêtes de poids maximal soient étudiée en dernier
  - Conséquence : On trie dans une liste L les arêtes dans l'ordre croissant des pondérations

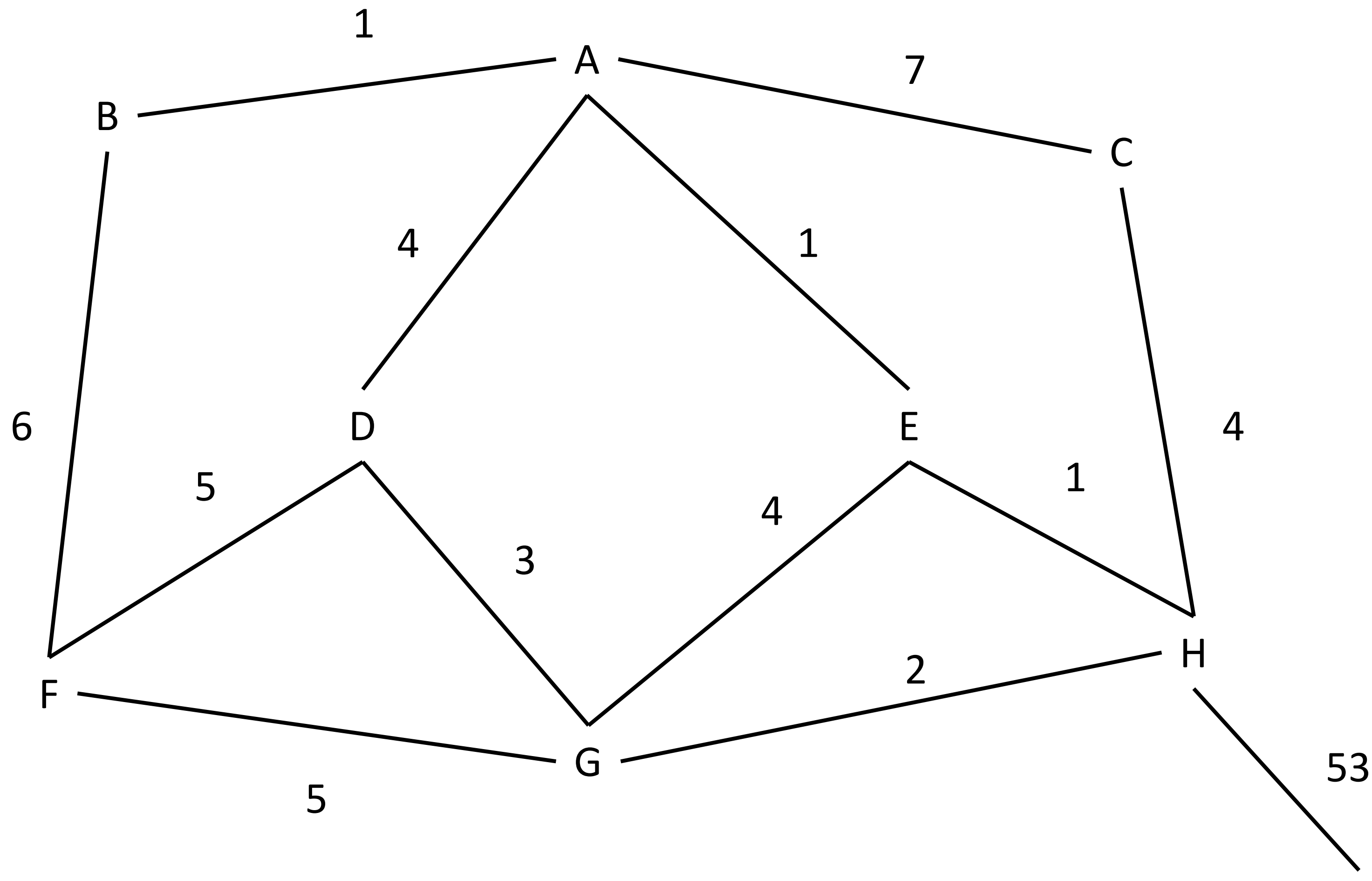
Algorithme de Kruskal Idées

- Partant d'une forêt de n arbres
- On ajoute les arêtes une à une dans la forêt en respectant l'ordre donné par les pondérations.
- Lorsqu'on envisage l'arête xy si x et y sont dans deux arbres distincts de la forêt on insère l'arête xy dans la forêt qui compte alors un arbre de moins (xy a fusionné deux arbres)

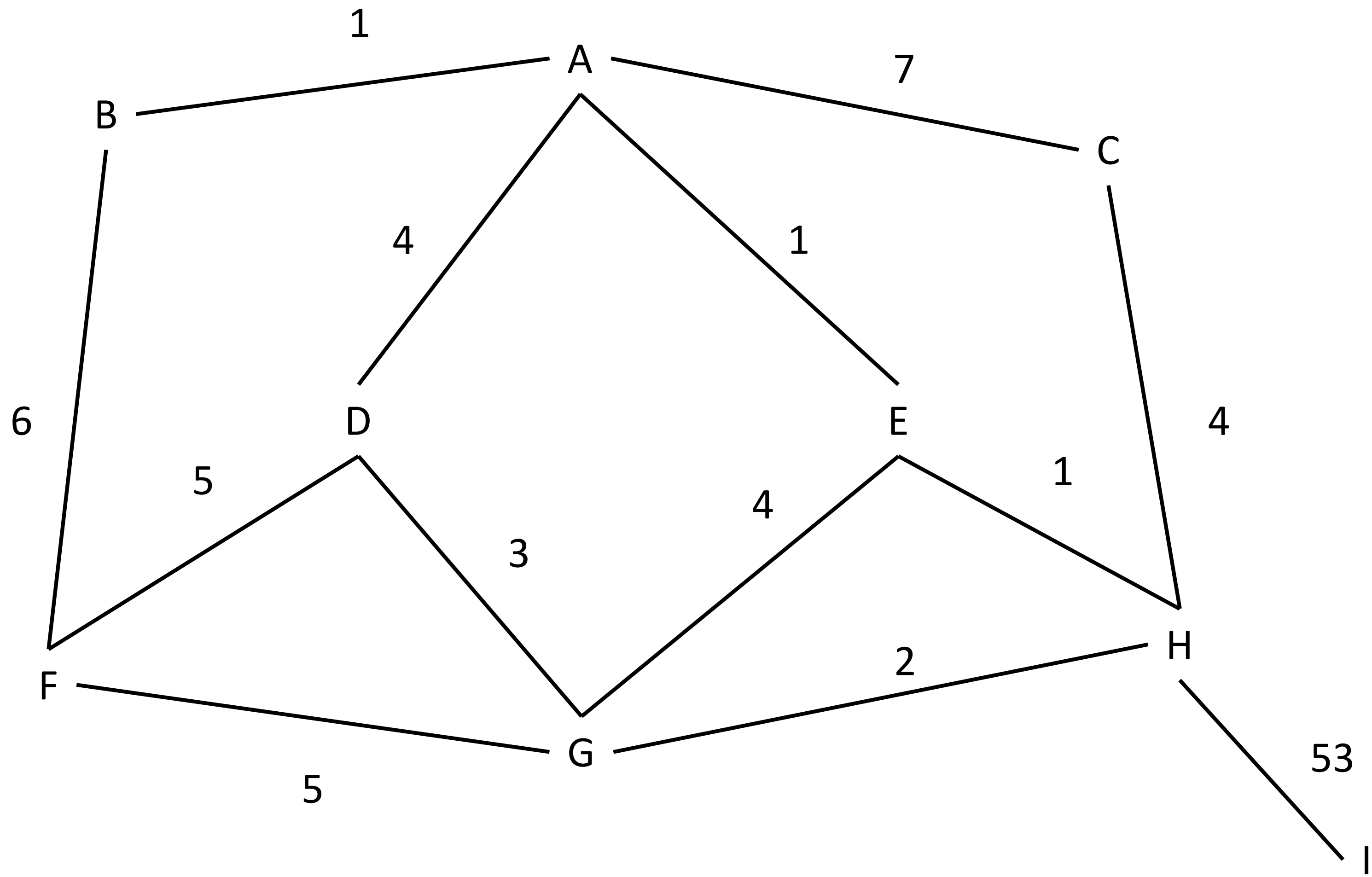
Algorithme de Kruskal Idées

- Il est fondé sur le second principe
- On crée le graphe ACPM = (X, U’,V)
- On trie dans une liste L les arêtes dans l’ordre croissant des pondérations
- Initialement : U’ = {}
- Pour chaque arête uv de L (dans l’ordre crois.)
  – Si non ($CC_{ACPM}(u) = CC_{ACPM}(v)$) alors
    - Insérer uv dans U’

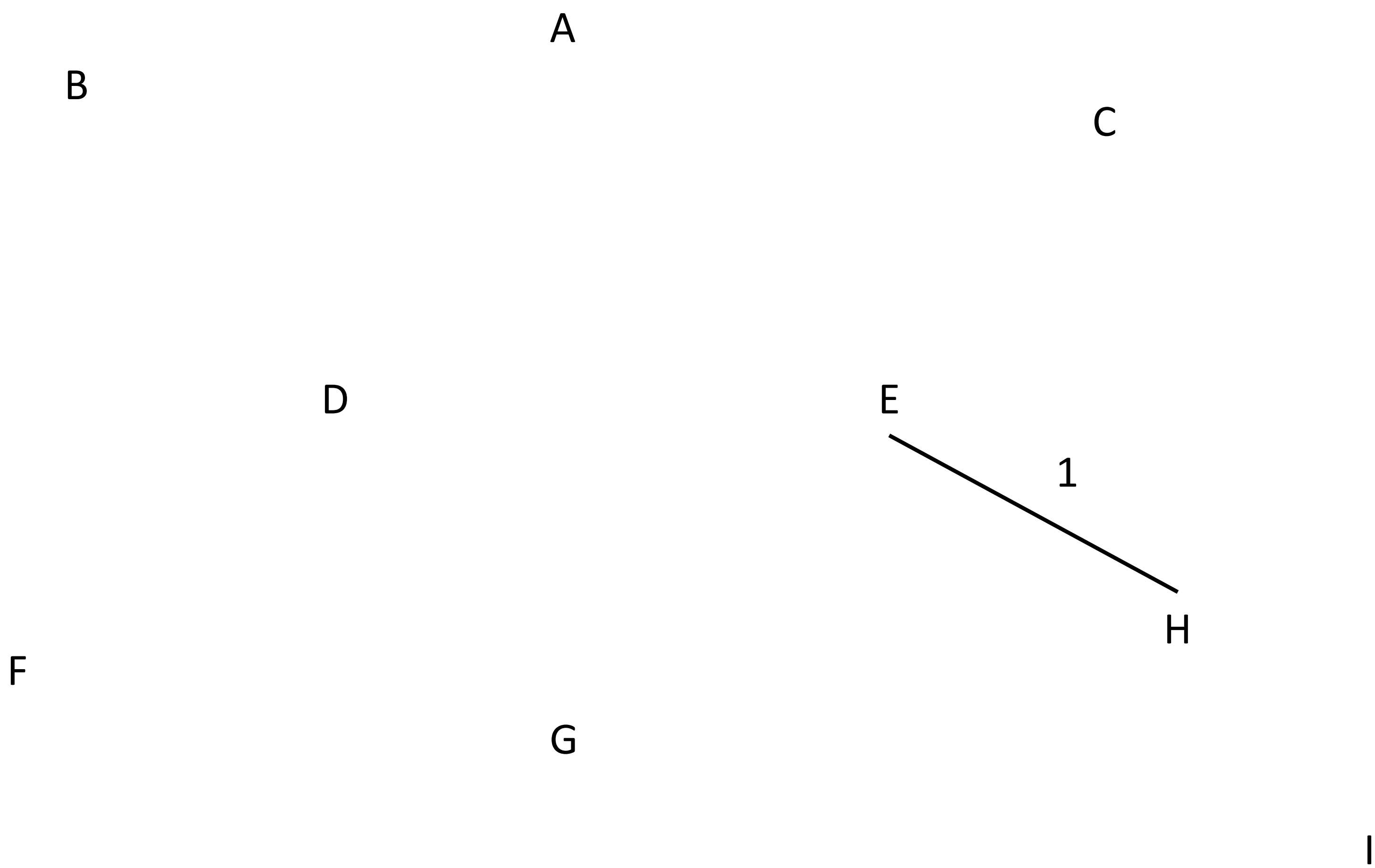
Exemple



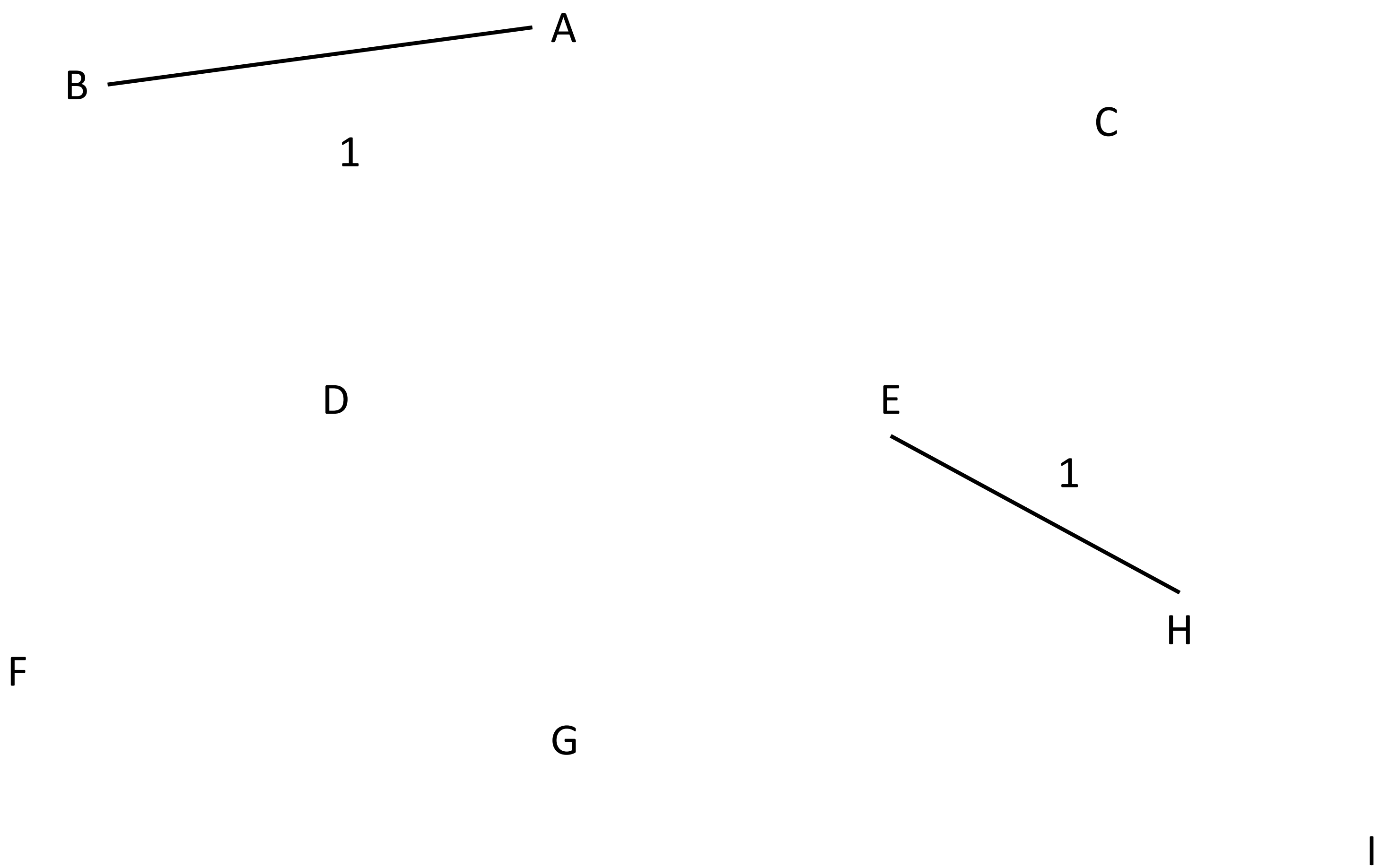
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



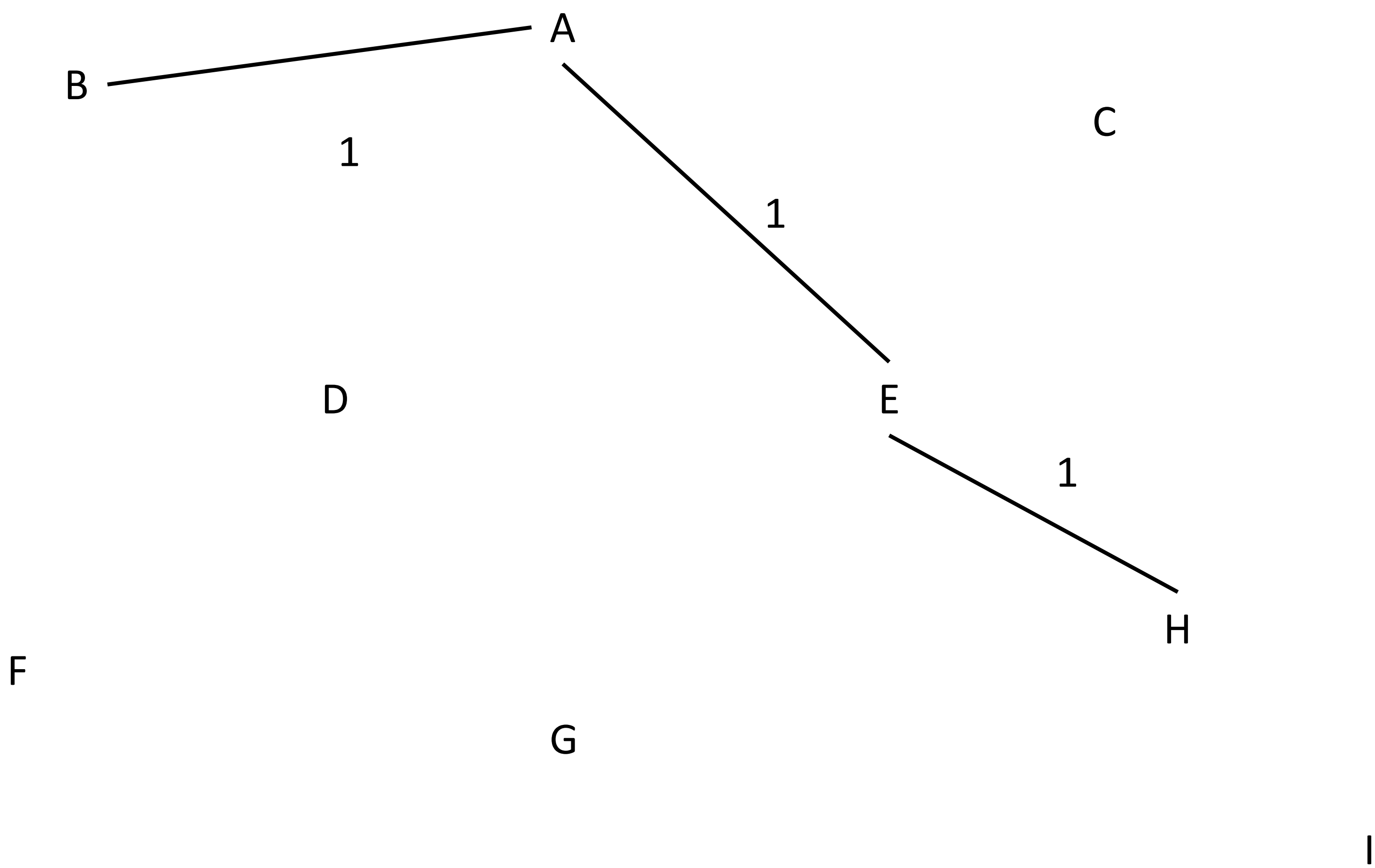
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



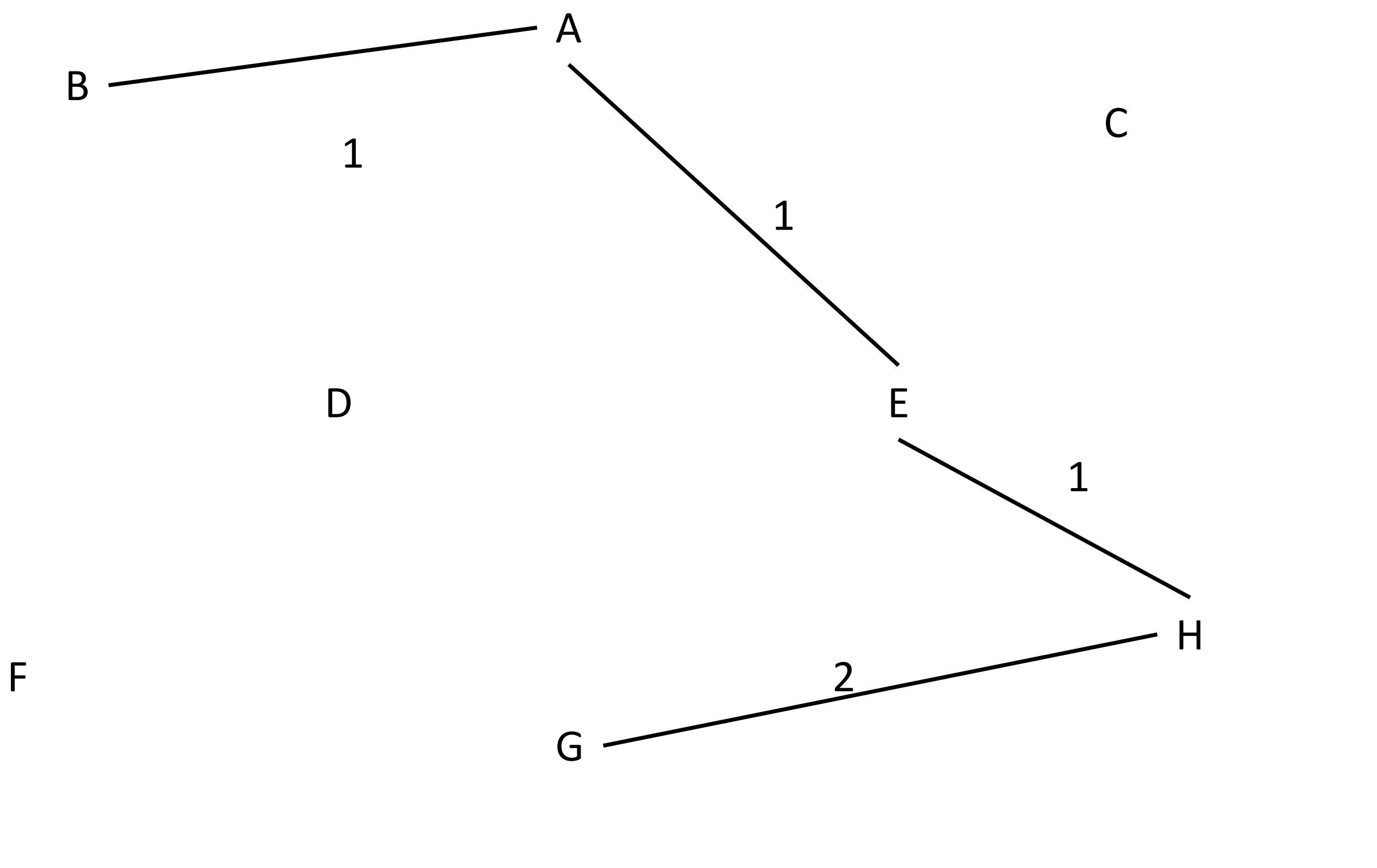
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



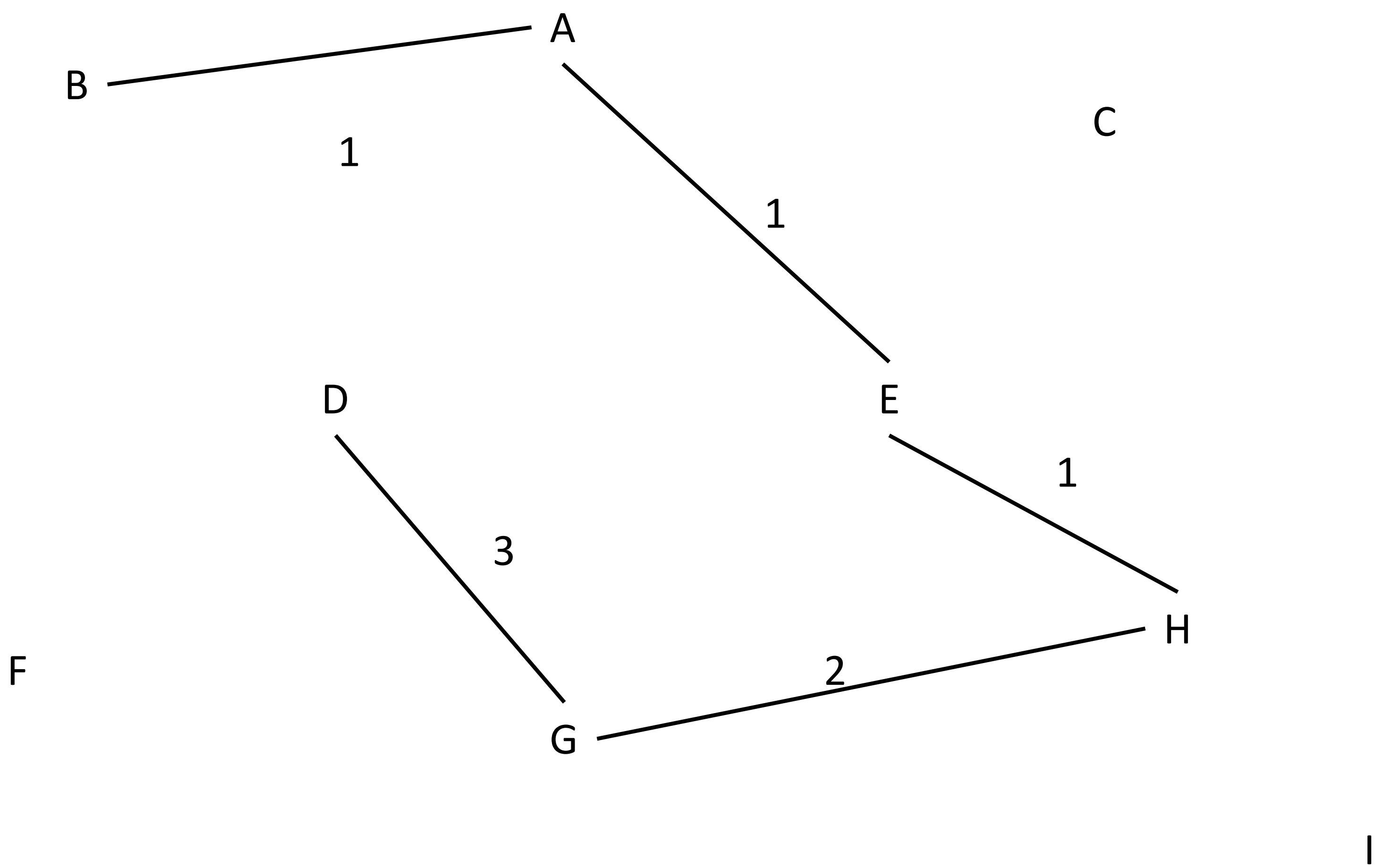
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



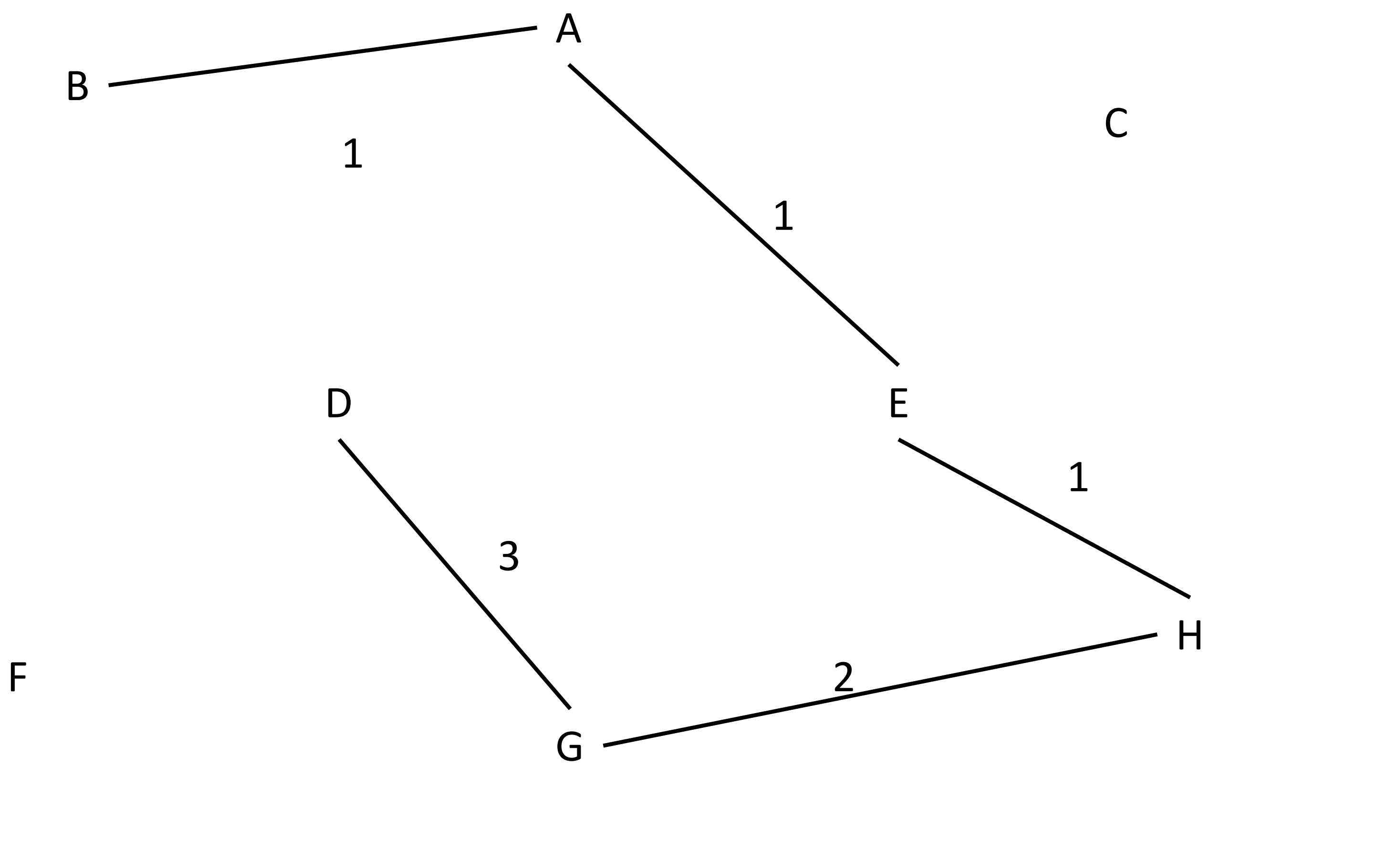
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



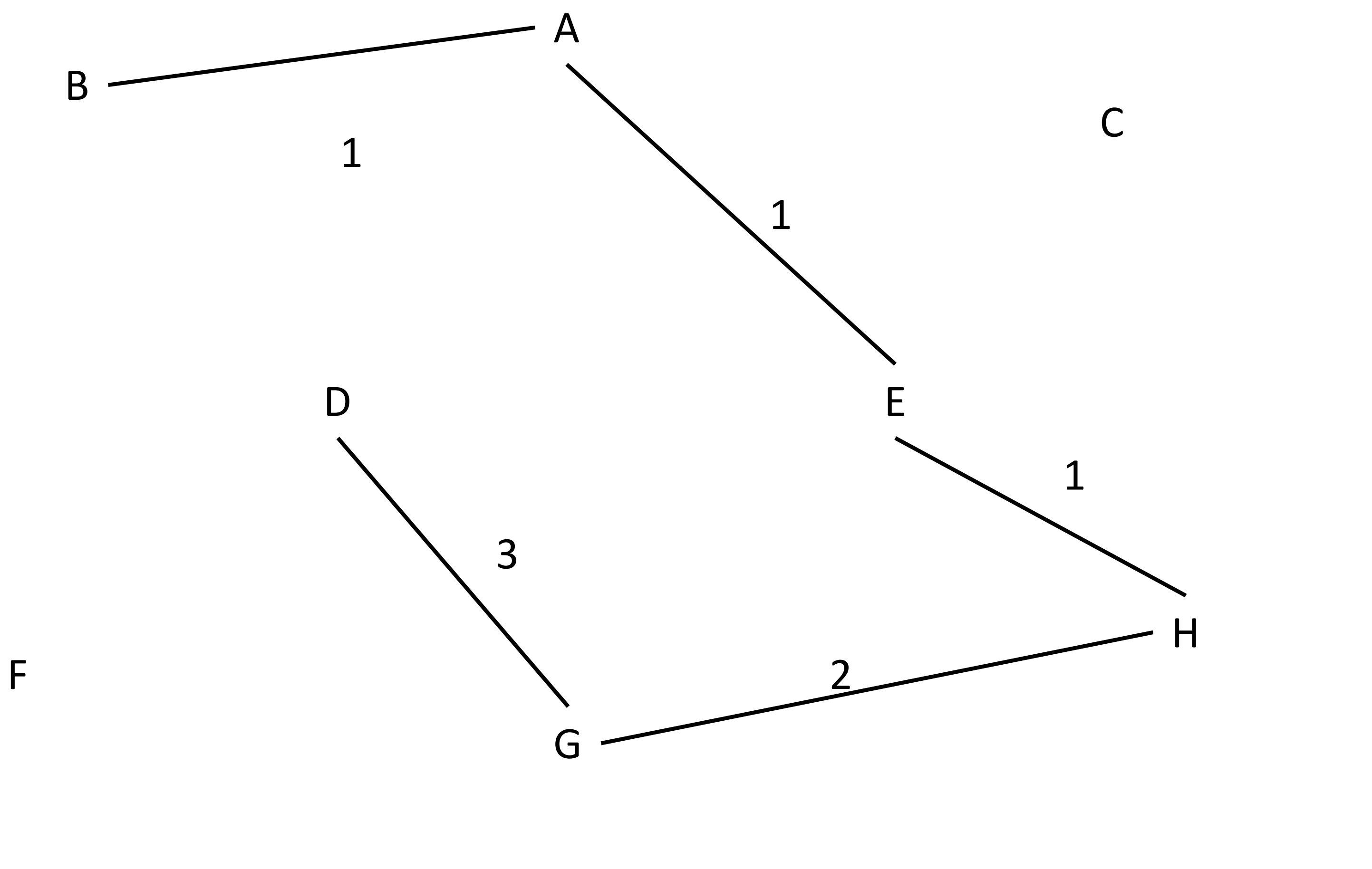
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



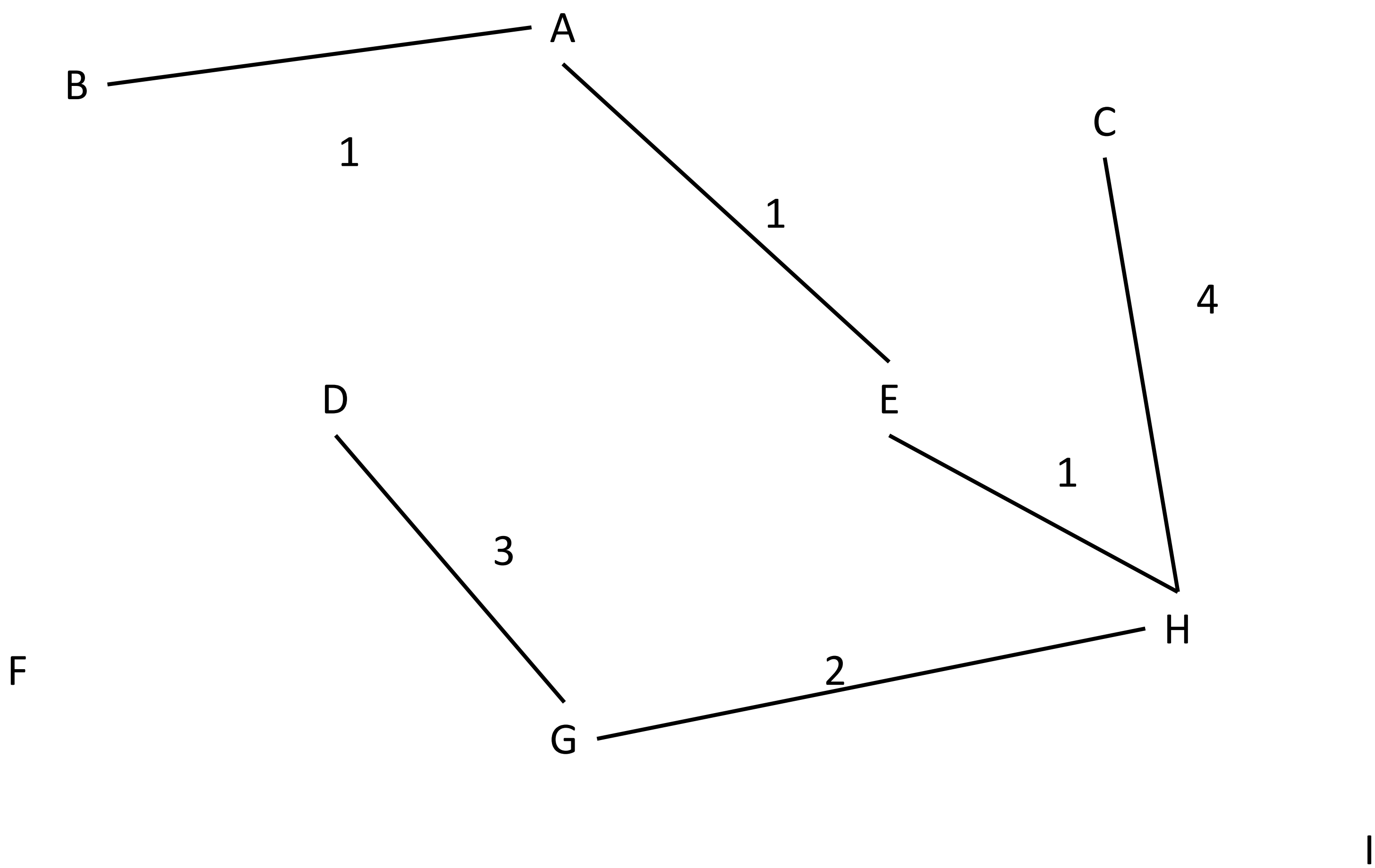
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



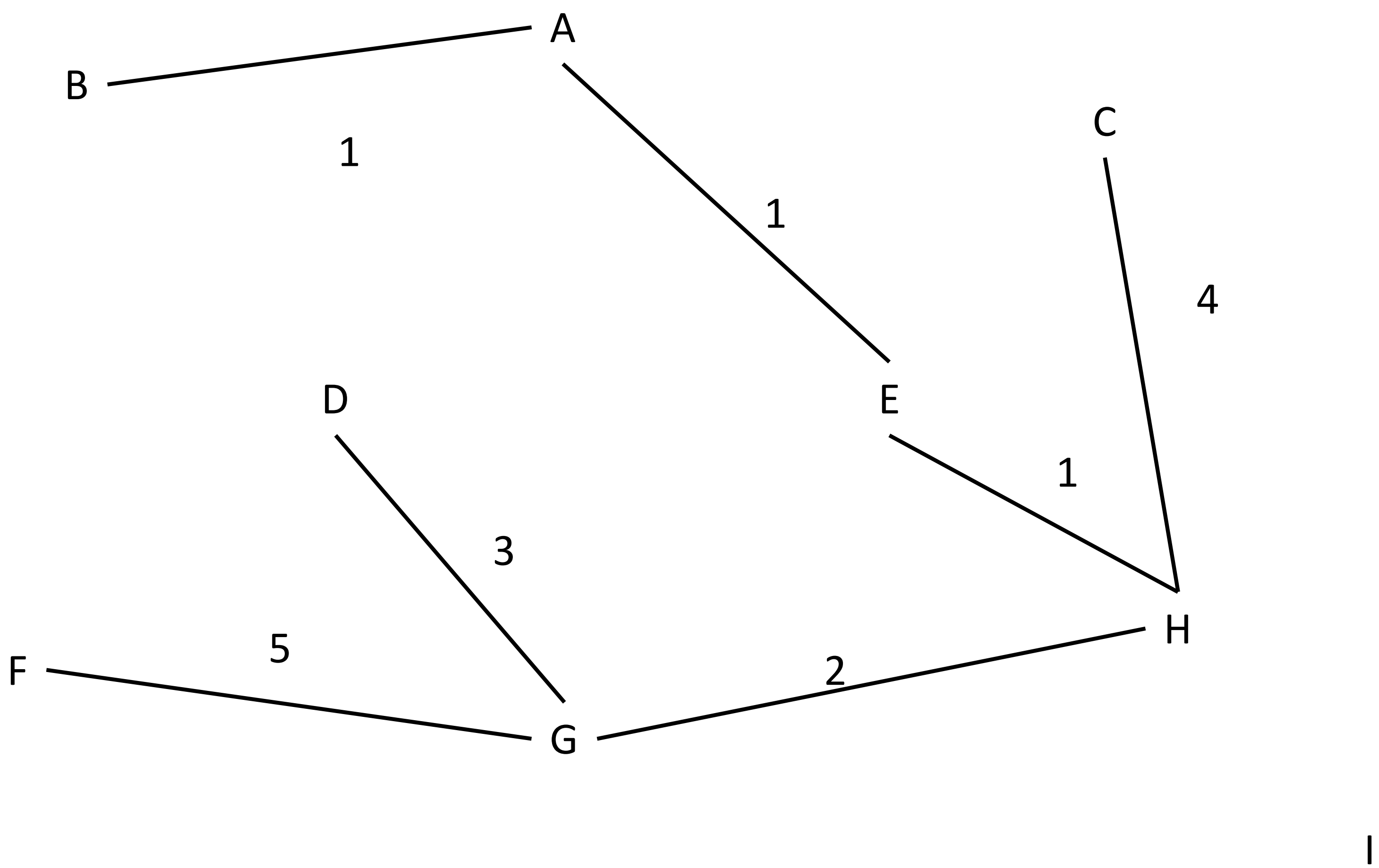
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



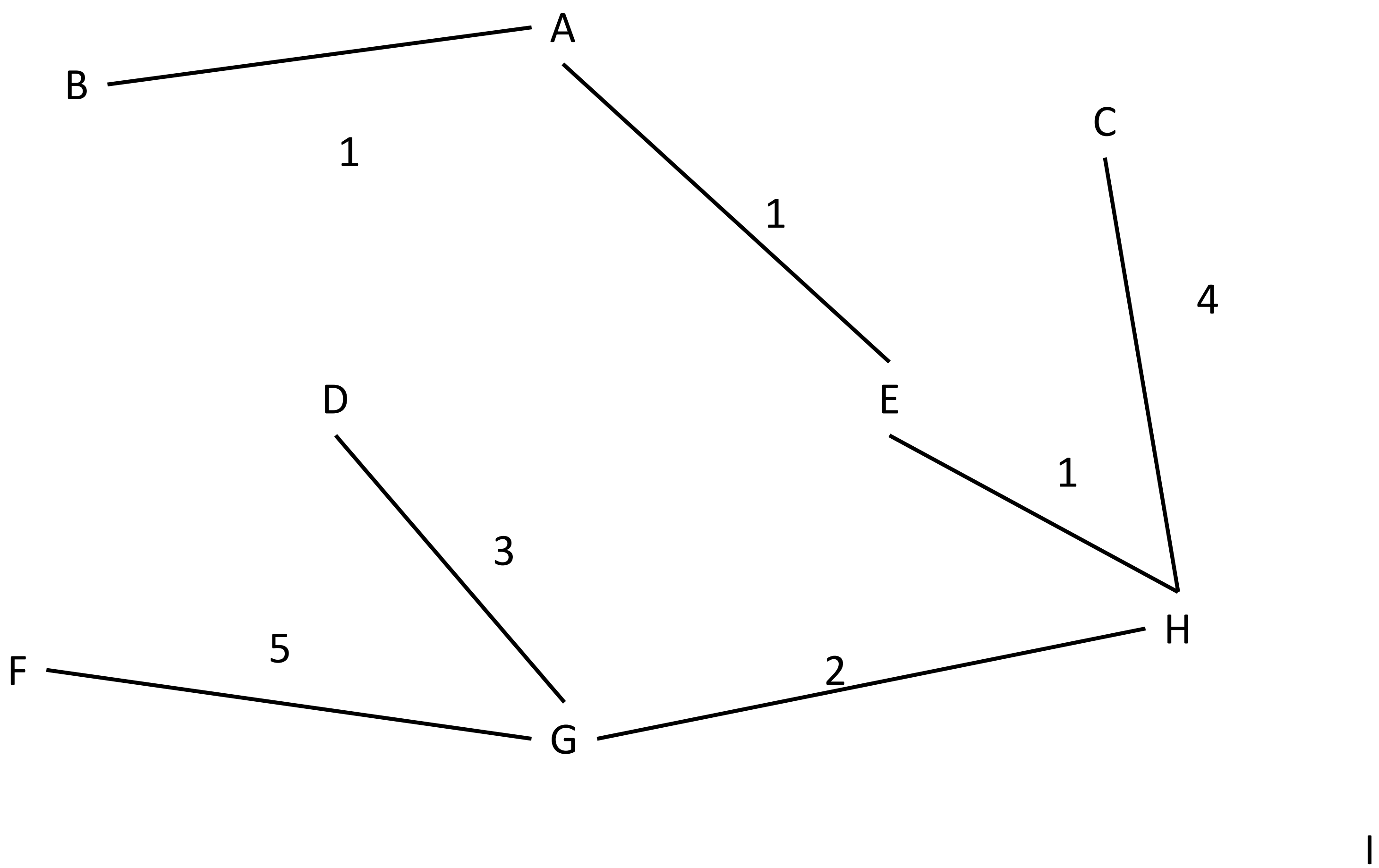
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



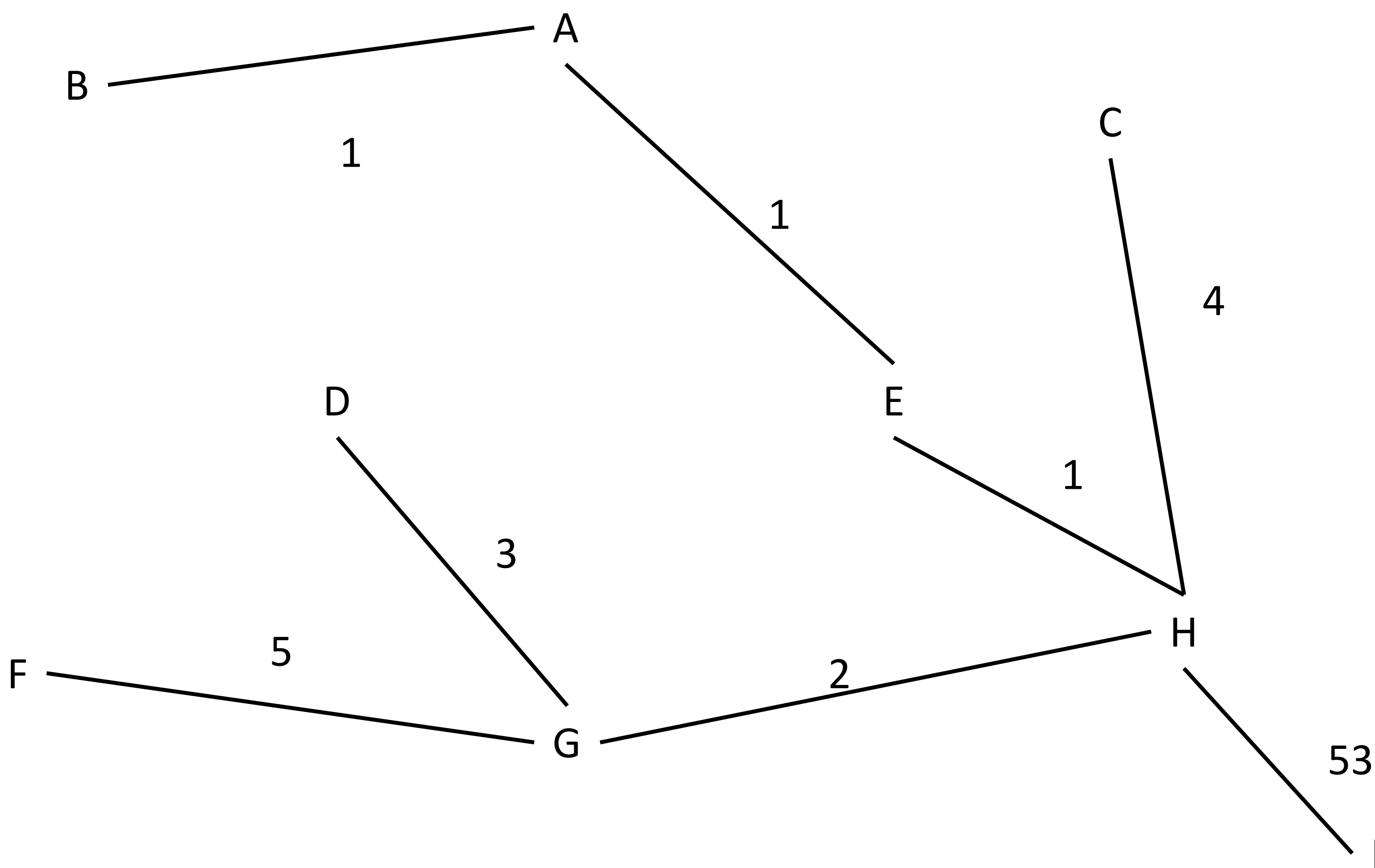
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



Algorithme de Kruskal V1 (Entête)

- Donnée :
  - G = (X, U, V) un graphe pondéré
- Résultat : ACPM = (X, A, V) un graphe pondéré
  - Initialement A = {};
- Variables :
  - L : Liste de couple (arête, poids);

Algorithme de Kruskal (Code)

- Début
  - L ← ListeVide()
  - Pour chaque sommet x de X faire
    - Pour chaque voisin y de x faire
      - Inserer (xy, v(xy)) dans L // v(xy) désigne le poids de l'arête
    - FinPour
  - FinPour
  - Trier(L) // Dans l'ordre croissant
  - Tant que non(TestListeVide (L)) faire
    - Soit (ab, p) = Premier(L)
    - Si non (a ϵ CC(ACPM,b)) alors
      - Inserer ab dans A
    - FinSi
    - L ← Suite(L)
  - FinTant Que
- Fin

Kruskal

- La difficulté de cet algorithme réside dans la gestion des composantes connexes de ACPM
- En effet l’arête uv est introduite dans ACPM si et seulement si la composante connexe de u ne contient pas v.
- Mais l’introduction de cette arête dans ACPM change les composantes connexes de ce graphe.
- Peut-on éviter d’avoir tout à recalculer à chaque étape ?

Comment gérer les composantes connexes

- Méthode simple : utiliser un parcours pour calculer une composante connexe.
- Ainsi pour calculer la composante connexe de a dans le graphe ACPM on peut exécuter la suite d'instructions
  - Exploré ← {}
  - VisiteGraphe (ACPM, a, Exploré)
  - Tester si b ϵ Exploré
- C'est cher (O(nm)) mais ça fonctionne

Comment gérer les composantes connexes : Idées

- Nous allons gérer une structure de données permettant d'associer à chaque composante connexe un représentant et un seul en gérant les trois opérations suivantes
- Initialisation (le point de départ)
- Find x : Recherche le représentant de la composante connexe de x
- Union x,y : fusionne la composante connexe de x et de y en une unique composante connexe

Illustration

- Partant d’un graphe contenant les sommets a, b, c, d, e, f, g, h, i, j et aucune arête.
- Nous allons ajouter des arêtes une à une dans ce graphe et donc fusionner des composantes connexes

Initialisation

	a	b	c	d	e	f	g	h	i	j
T	a	b	c	d	e	f	g	h	i	j

c d

b e

a f

g

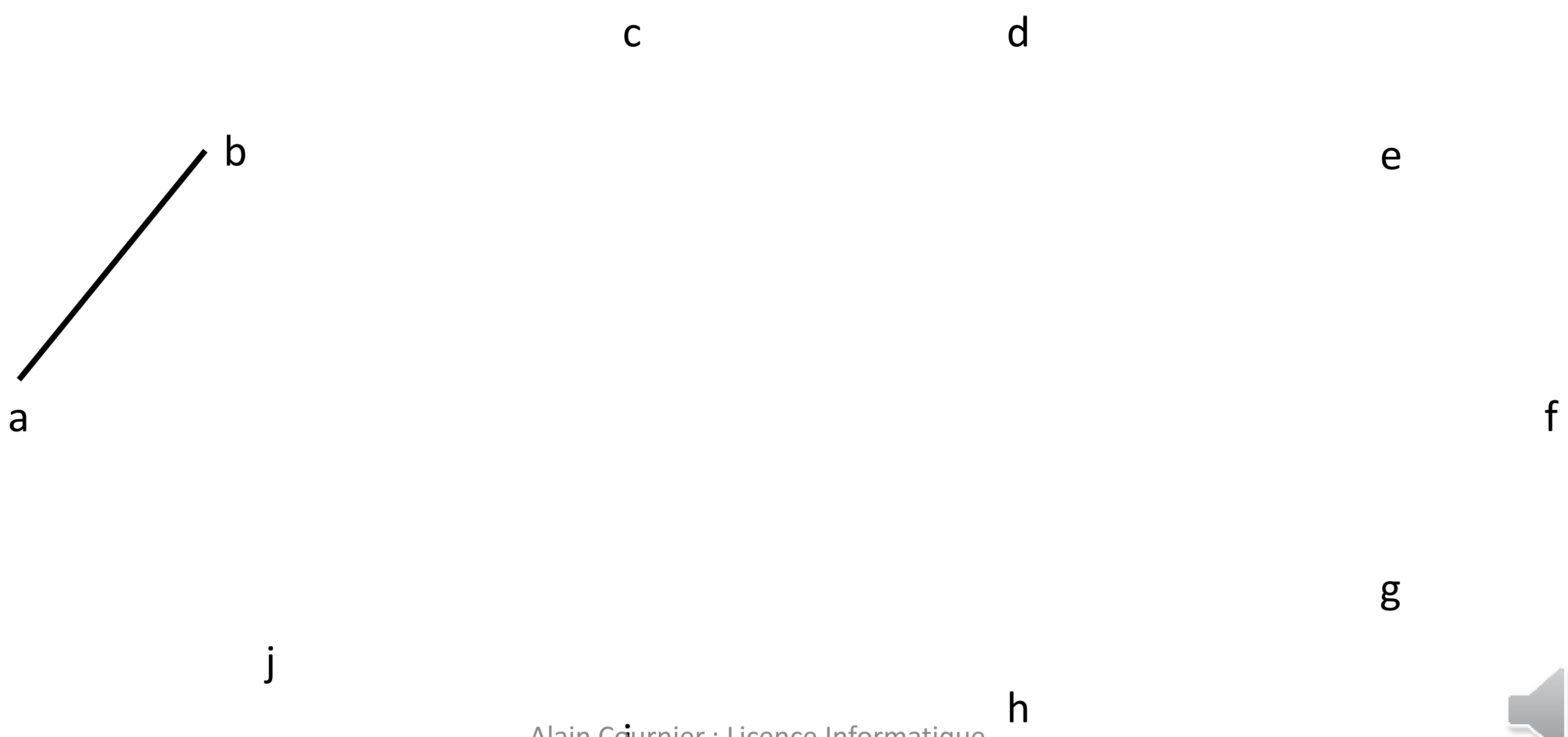
j

h

i

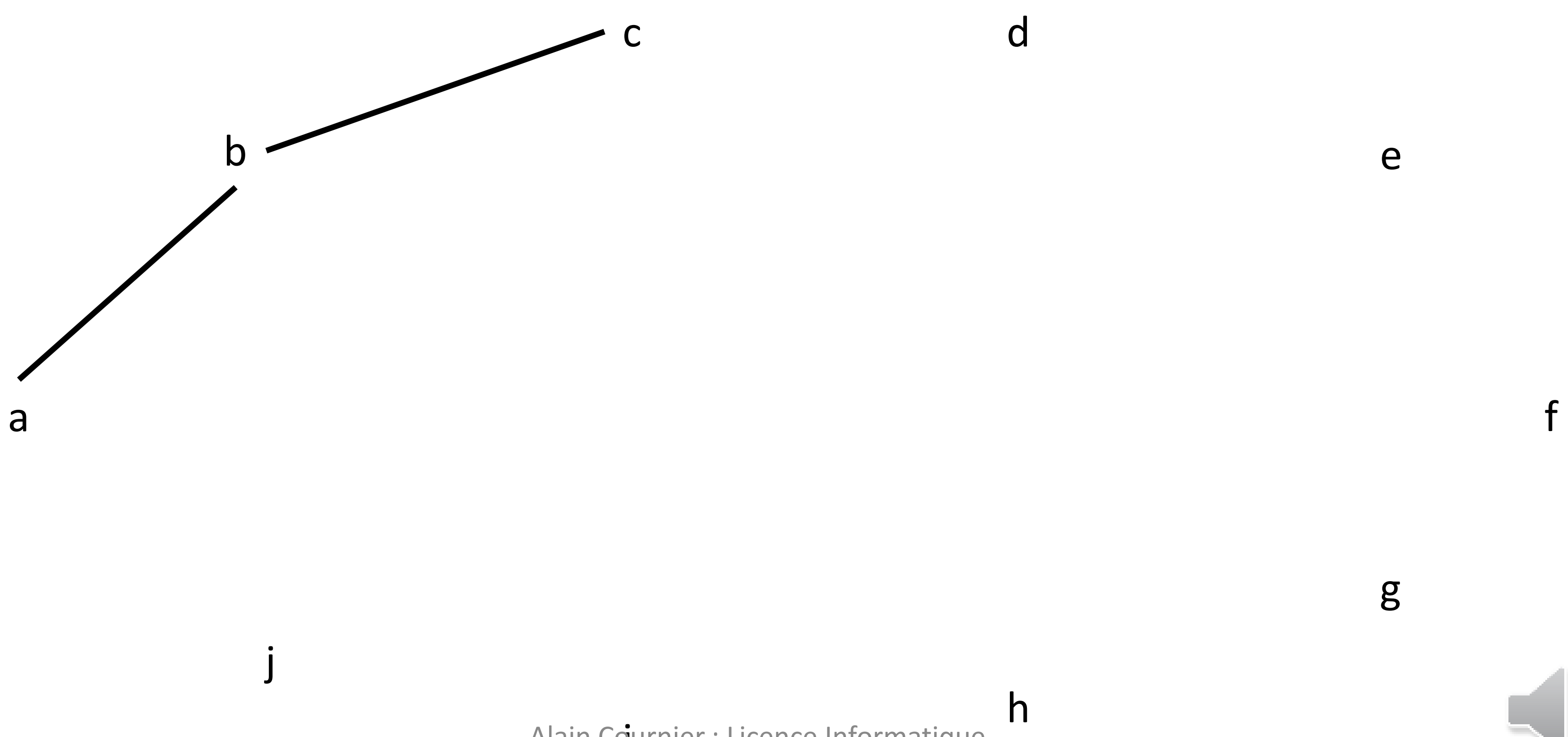
Ajouter ab

	a	b	c	d	e	f	g	h	i	j
T	b	b	c	d	e	f	g	h	i	j



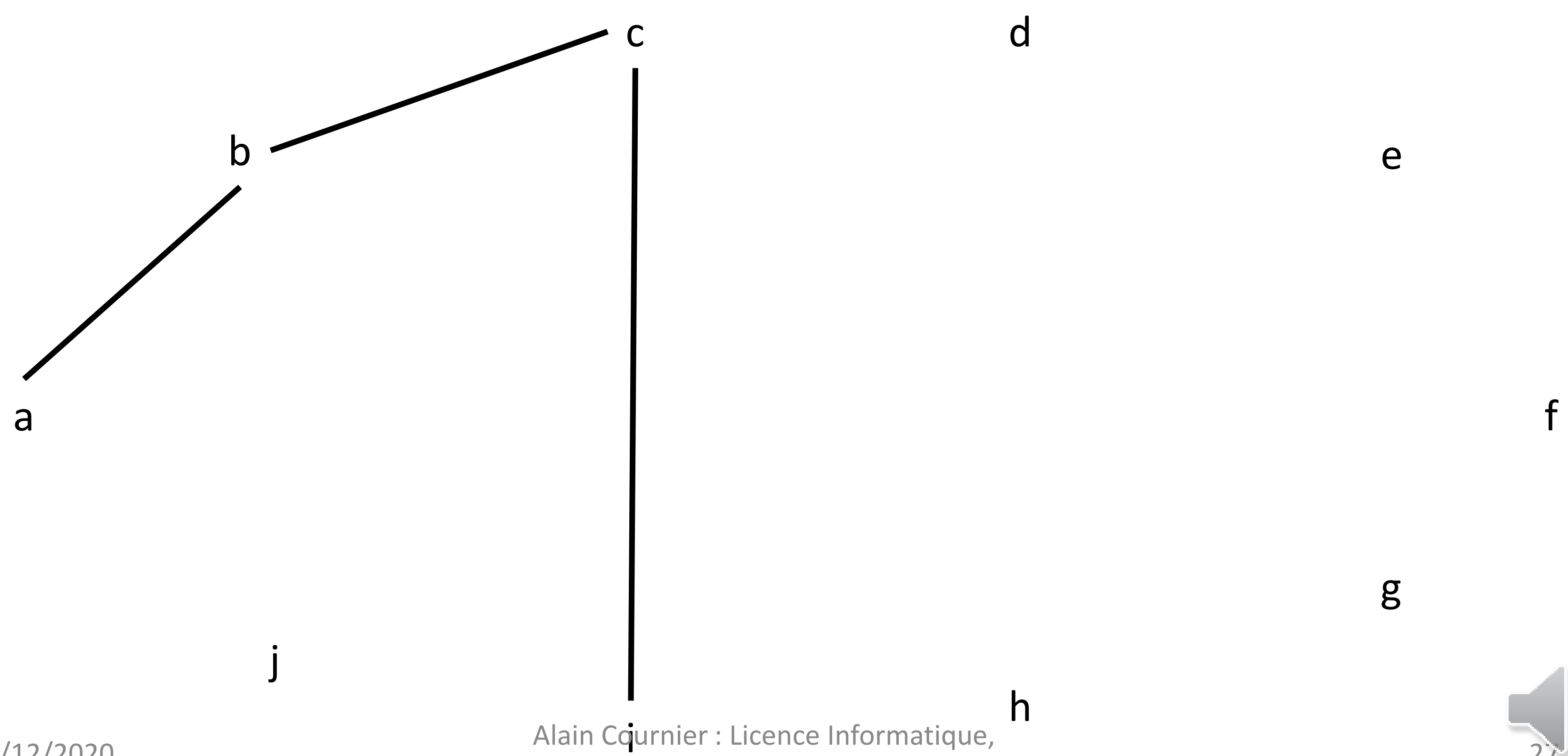
Ajouter bc

	a	b	c	d	e	f	g	h	i	j
T	b	c	c	d	e	f	g	h	i	j



Ajouter ci

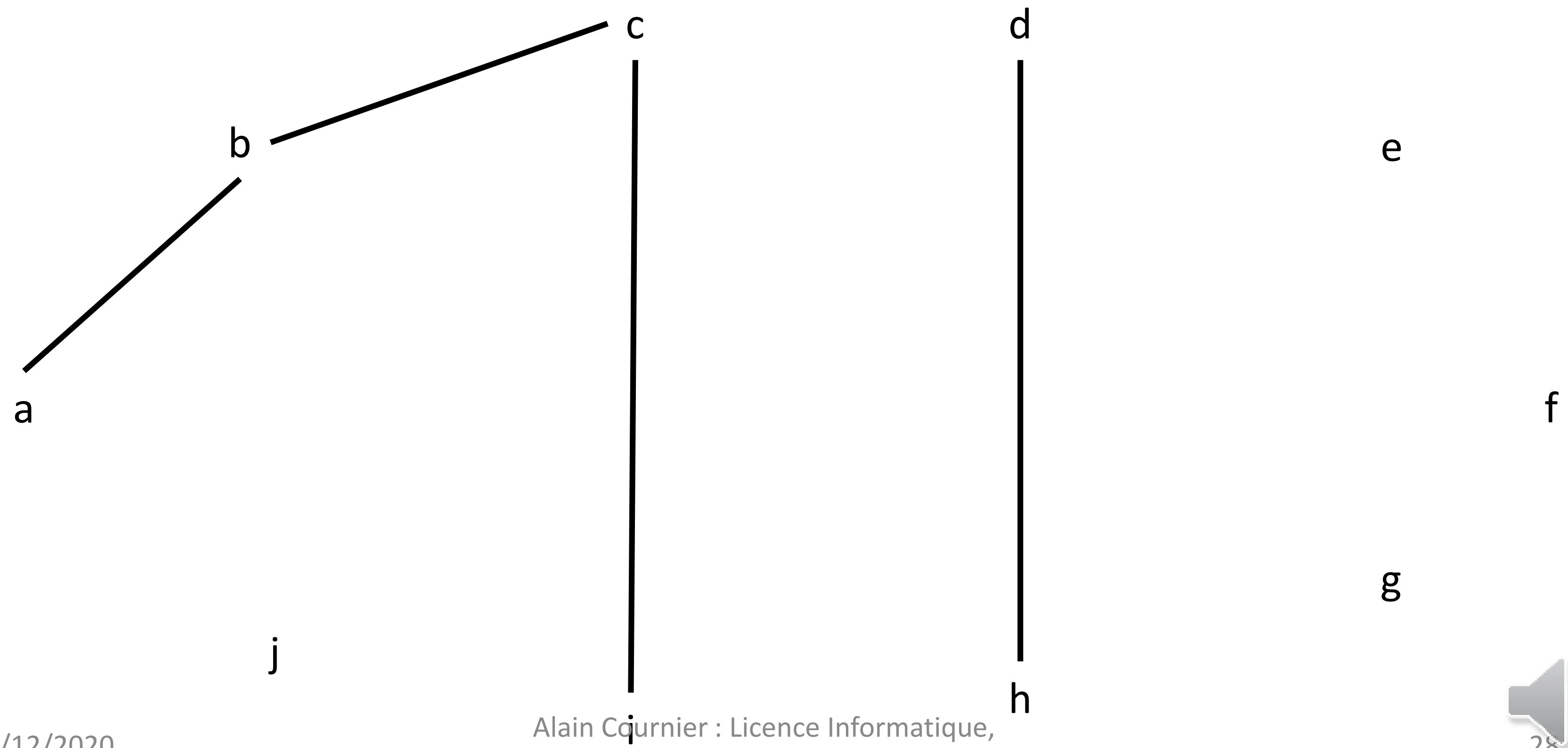
T	a	b	c	d	e	f	g	h	i	j
	b	c	i	d	e	f	g	h	i	j



Ajouter dh

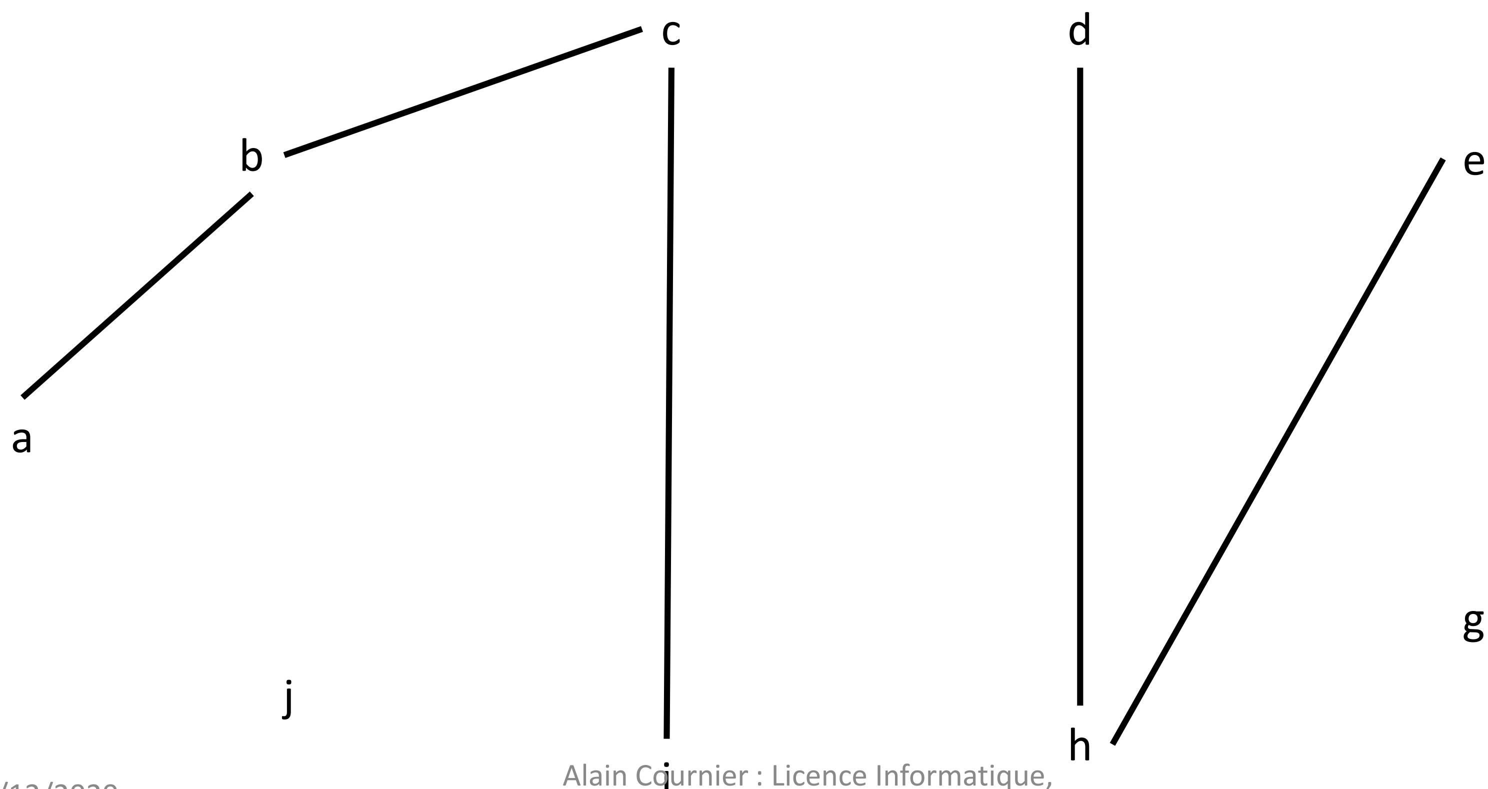
T

a	b	c	d	e	f	g	h	i	j
b	c	i	h	e	f	g	h	i	j



Ajouter he

	a	b	c	d	e	f	g	h	i	j
T	b	c	i	h	e	f	g	e	i	j



Qui est le représentant de la composante connexe de a ?

a	b	c	d	e	f	g	h	i	j
b	c	i	h	e	f	g	e	i	j

1. C'est le même que le représentant de b
2. C'est donc le même que le représentant de c
3. C'est donc le même que le représentant de i
4. i est le représentant de la composante connexe
5. a, b, c et i ont tous 4 le même représentant

Opération Find (Entête)

- Fonction Find
  - Donnée :
    - x un sommet
  - Donnée/Résultat :
    - T un tableau de sommets indicé par les sommets
  - Résultat : un sommet

Opération Find (Code)

- DébutCode
  - Si T[x] = x alors renvoyer (x)
  - Sinon
    - T[x] ← Find (T[x], T)
    - Renvoyer (T[x])
  - FinSi
- FinCode

Opération Union (Entête)

- Algorithme Union
  - Donnée :
    - x et y : deux sommets
  - Donnée/Résultat :
    - T un tableau de sommets indicé par les sommets
  - Variables :
    - Repx, Repy : deux sommets

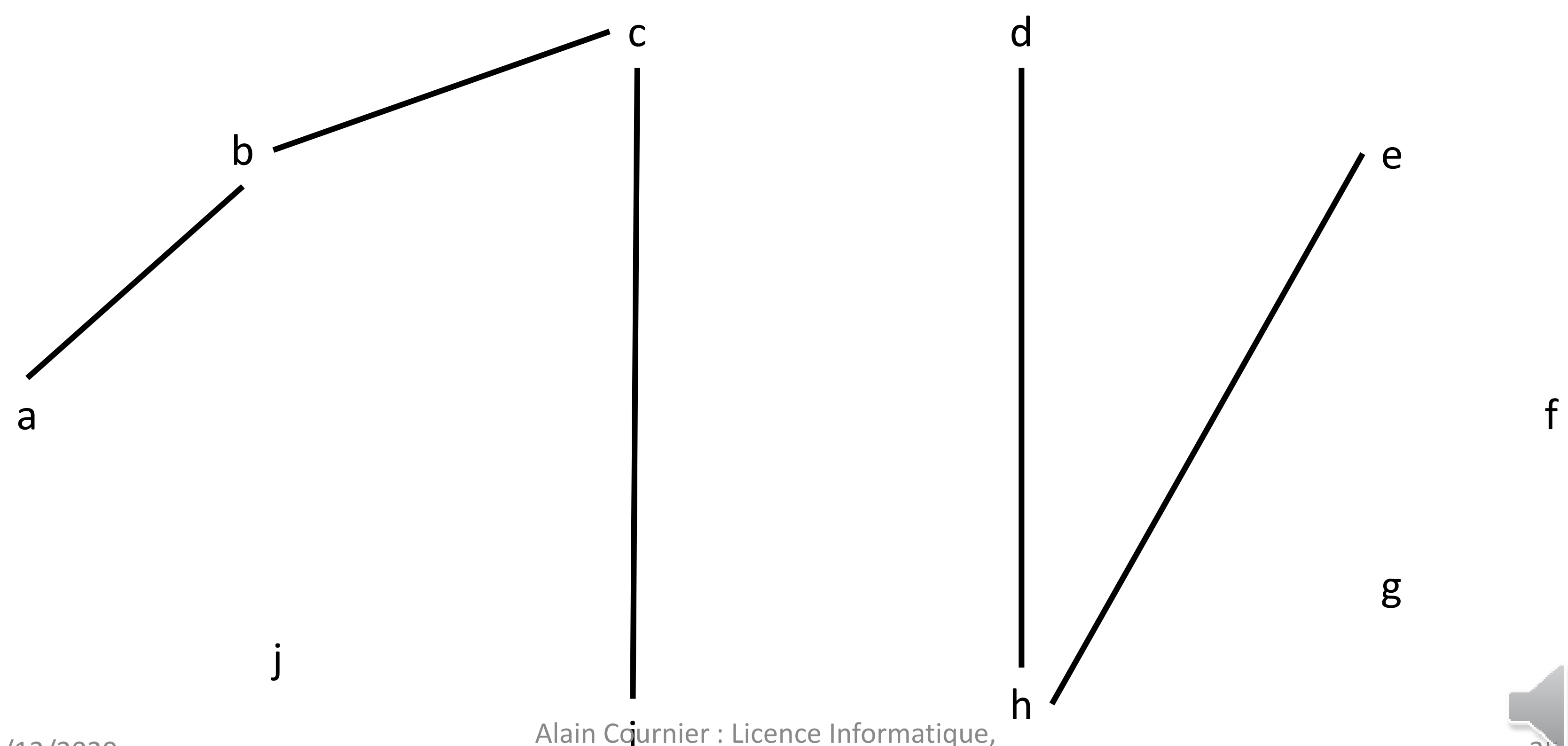
Opération Union (Code)

- DébutCode
  - Repx ← Find(x,T);
  - Repy ← Find(y,T)
  - Si Repx = Repy alors
    - ne rien faire
  - Sinon
    - T[Repx] ← Repy
  - FinSi
- FinCode

Après Ajouter he

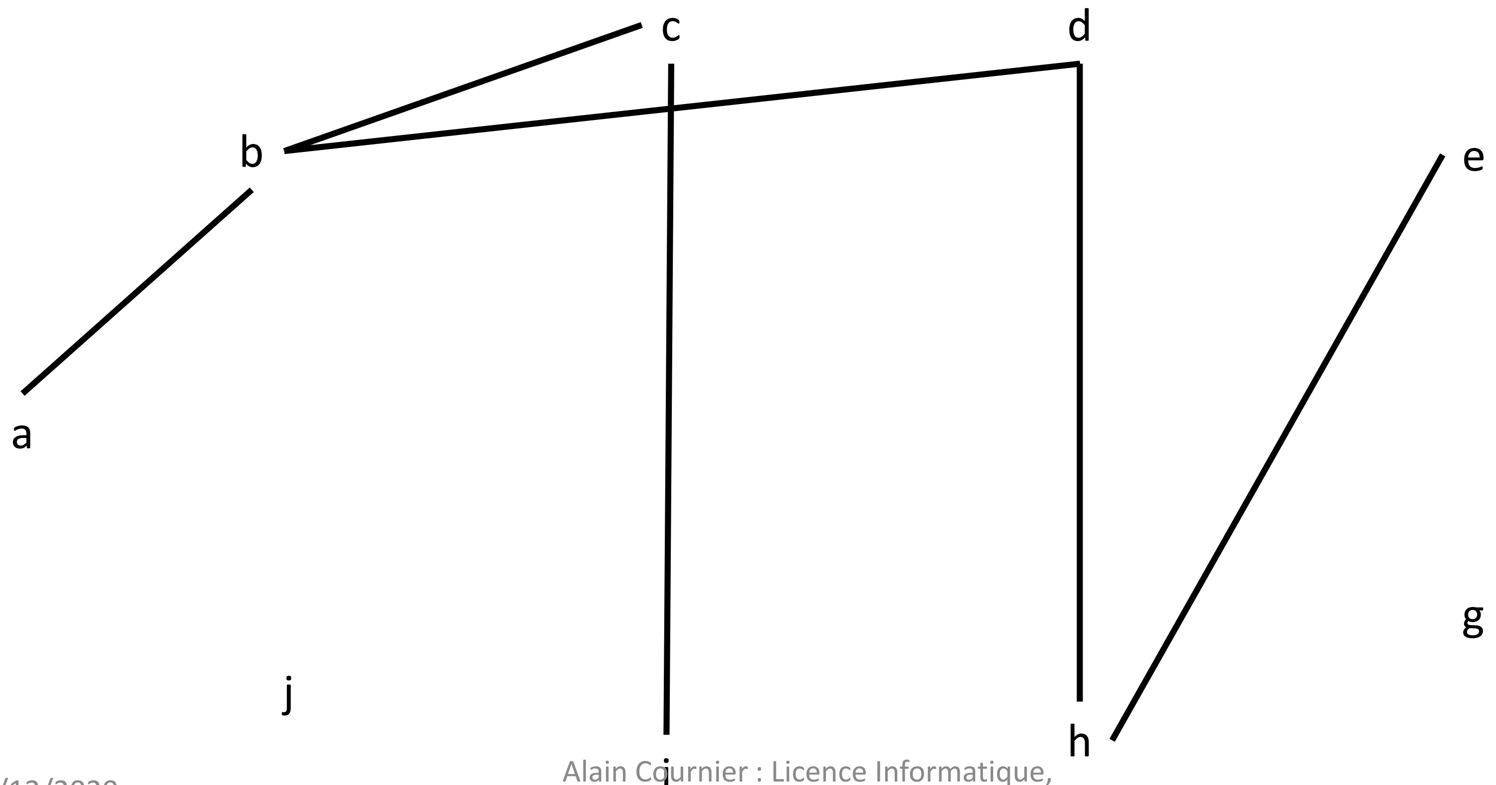
T

a	b	c	d	e	f	g	h	i	j
b	c	i	h	e	f	g	e	i	j



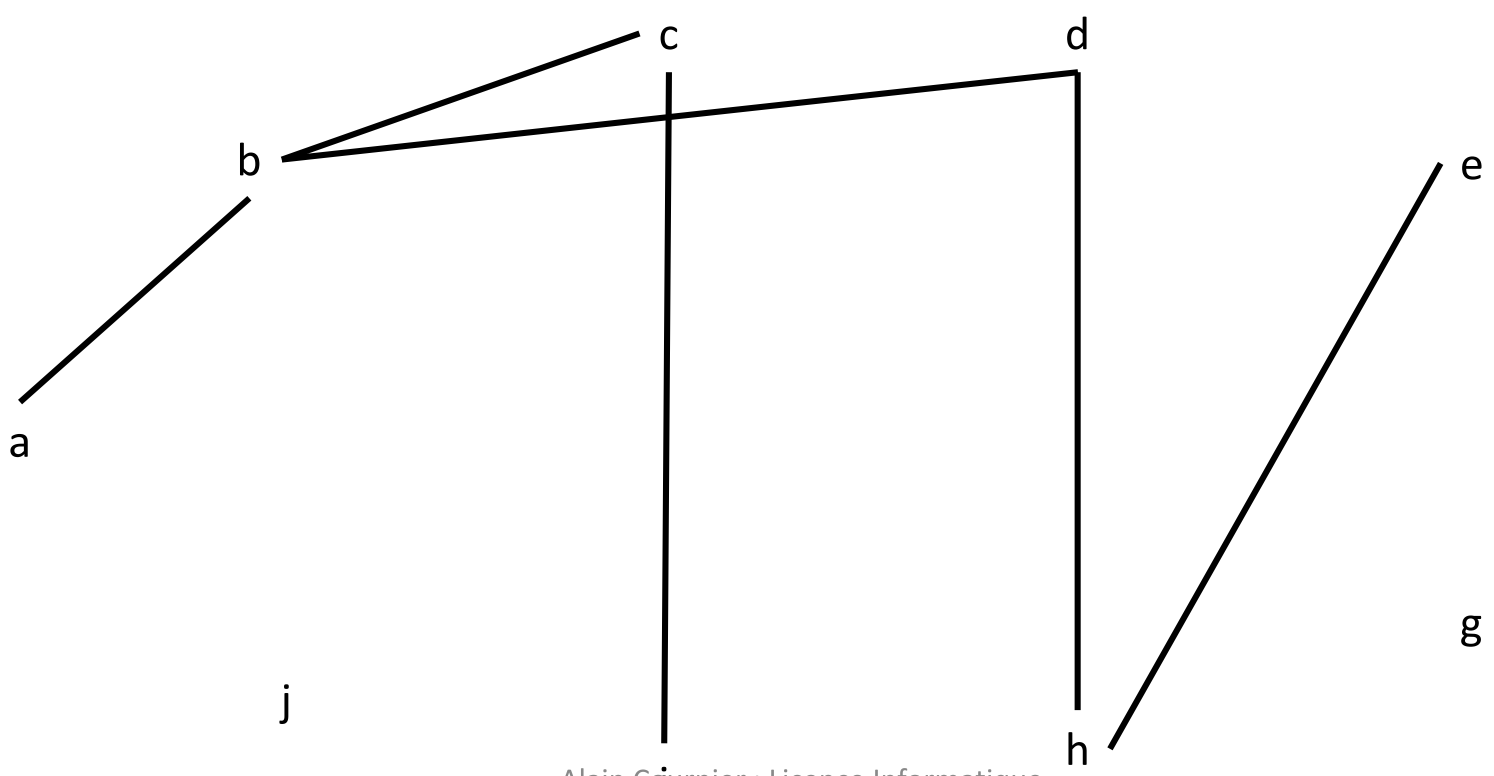
Ajouter bd

T	a	b	c	d	e	f	g	h	i	j
	b	c	i	h	e	f	g	e	i	j



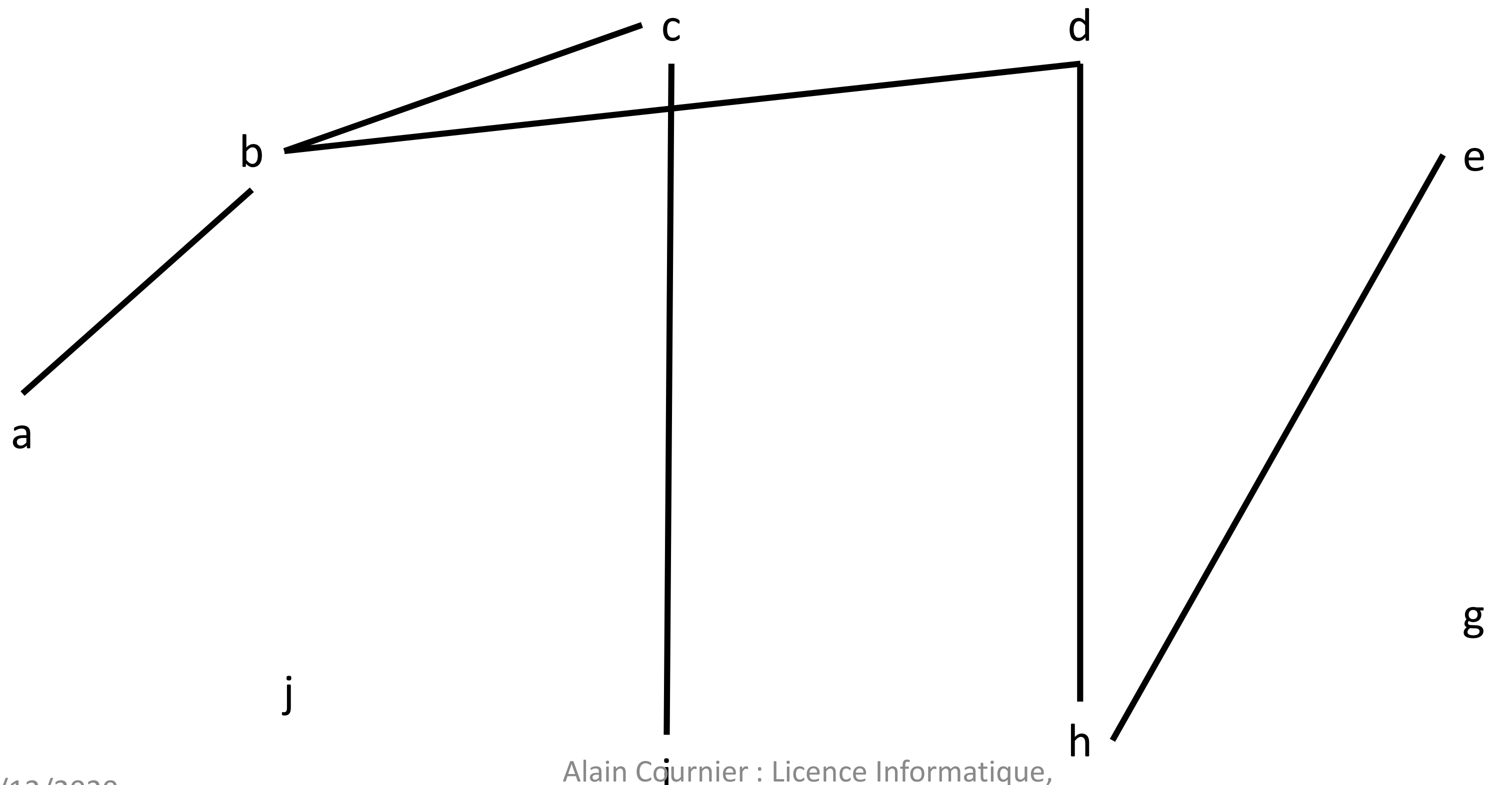
Find(b, T) = i

T	a	b	c	d	e	f	g	h	i	j
	b	i	i	h	e	f	g	e	i	j



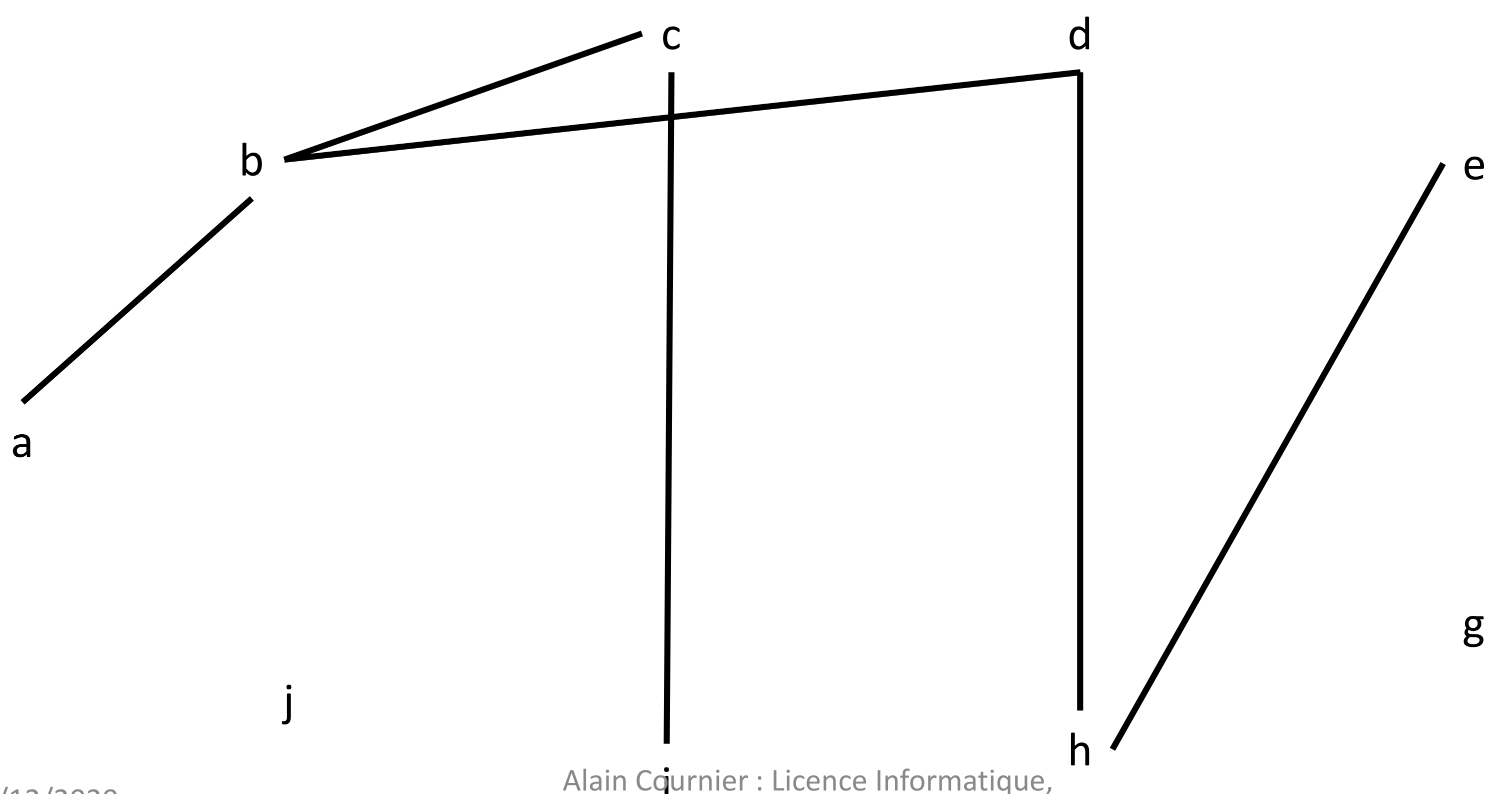
Find(d, T) = e

	a	b	c	d	e	f	g	h	i	j
T	b	i	i	e	e	f	g	e	i	j

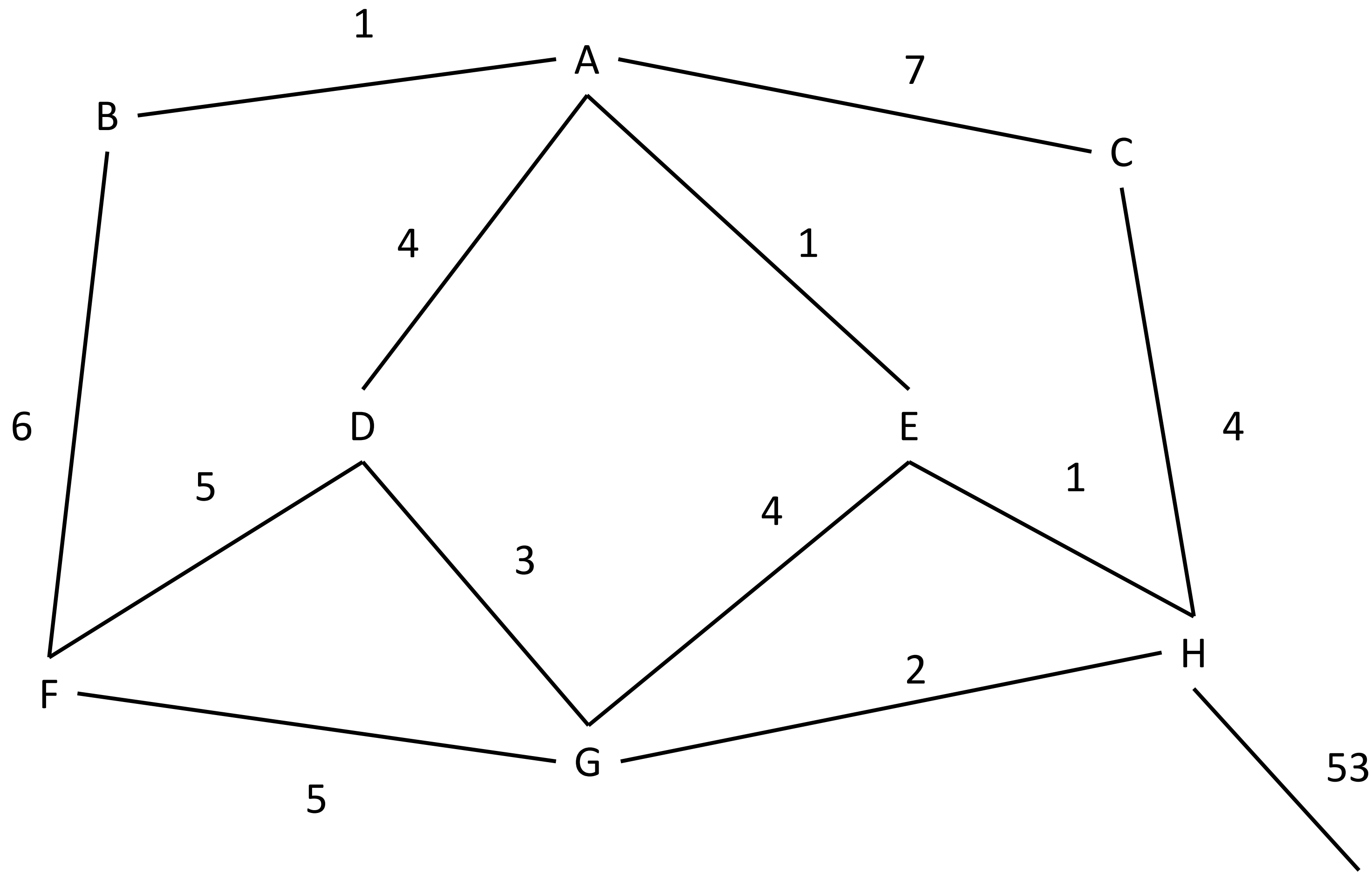


T[i] ← e

T	a	b	c	d	e	f	g	h	i	j
	b	i	i	e	e	f	g	e	e	j



<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}



<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
A	B	C	D	E	F	G	H	I

A

B

C

D

E

H

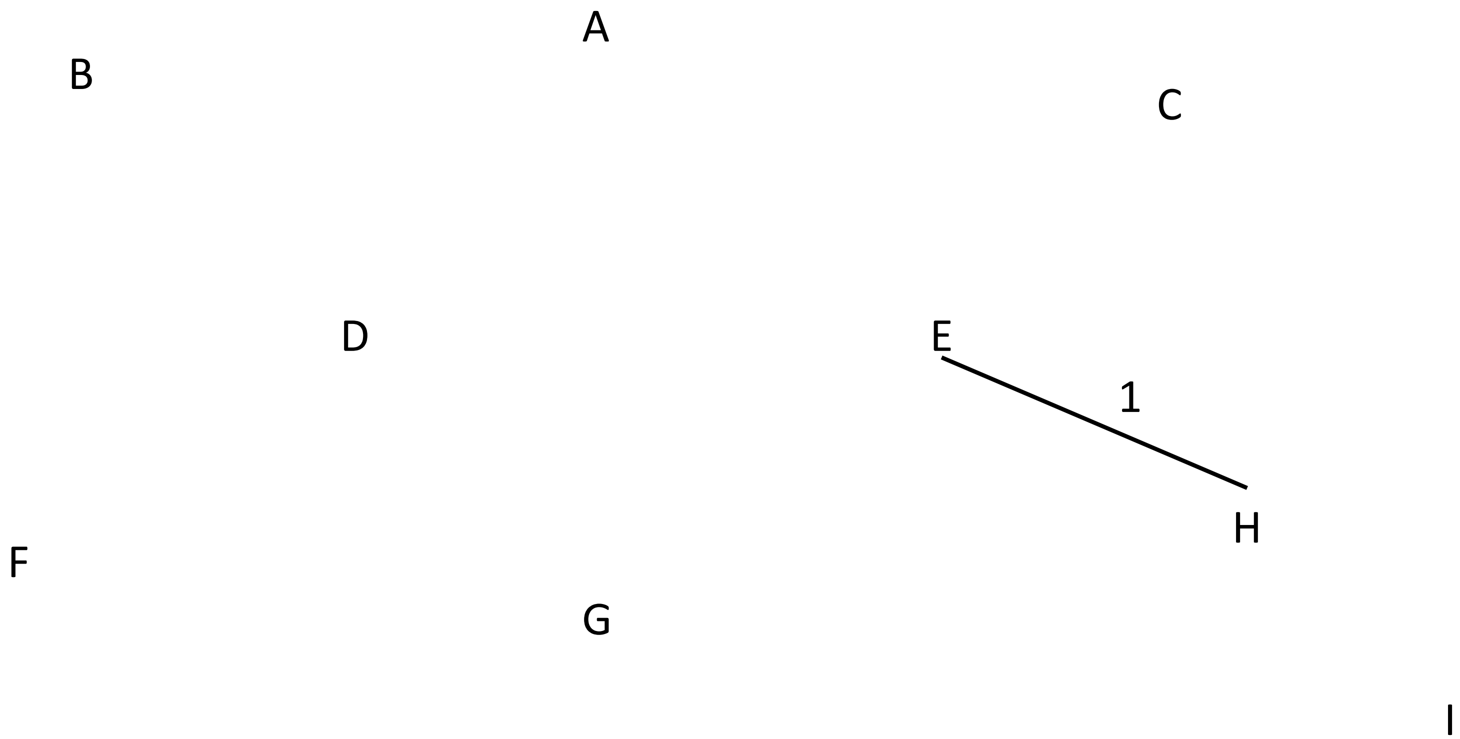
F

G

I

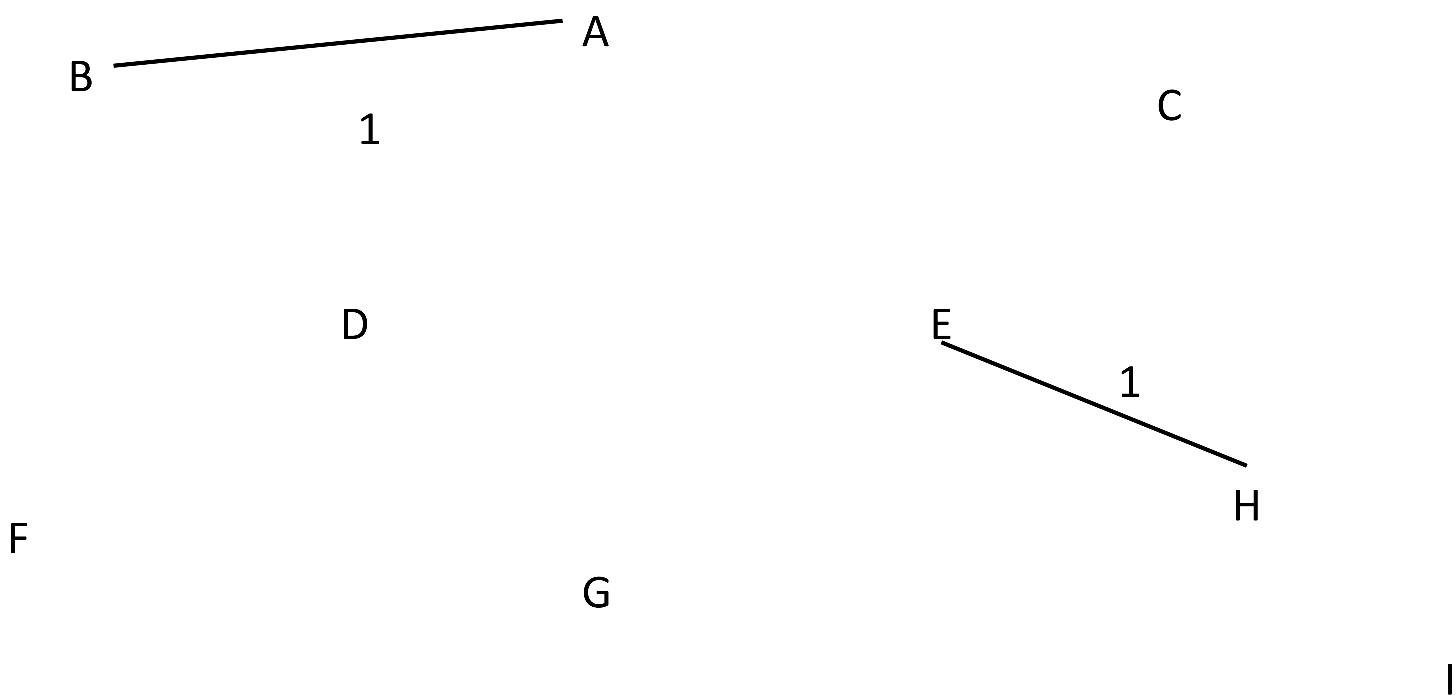
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
A	B	C	D	H	F	G	H	I



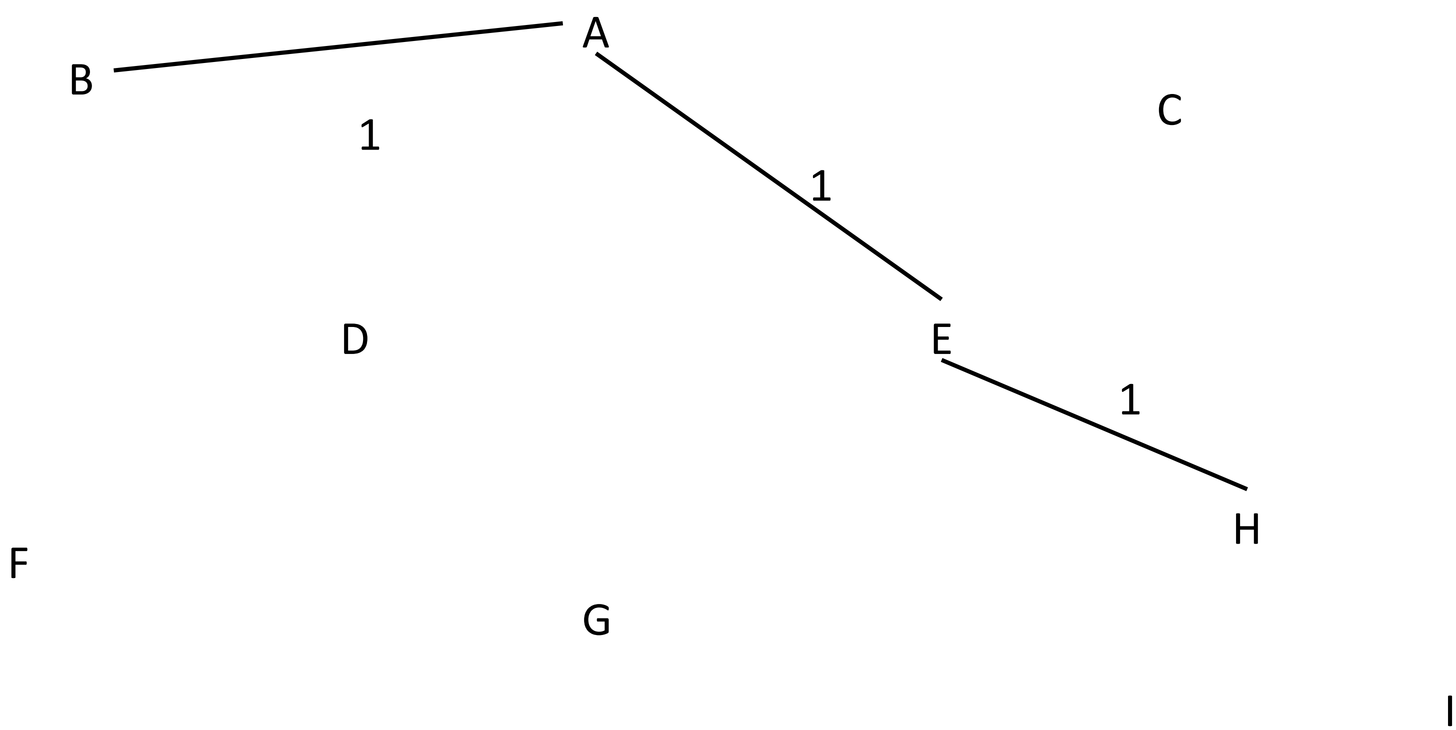
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
B	B	C	D	H	F	G	H	I



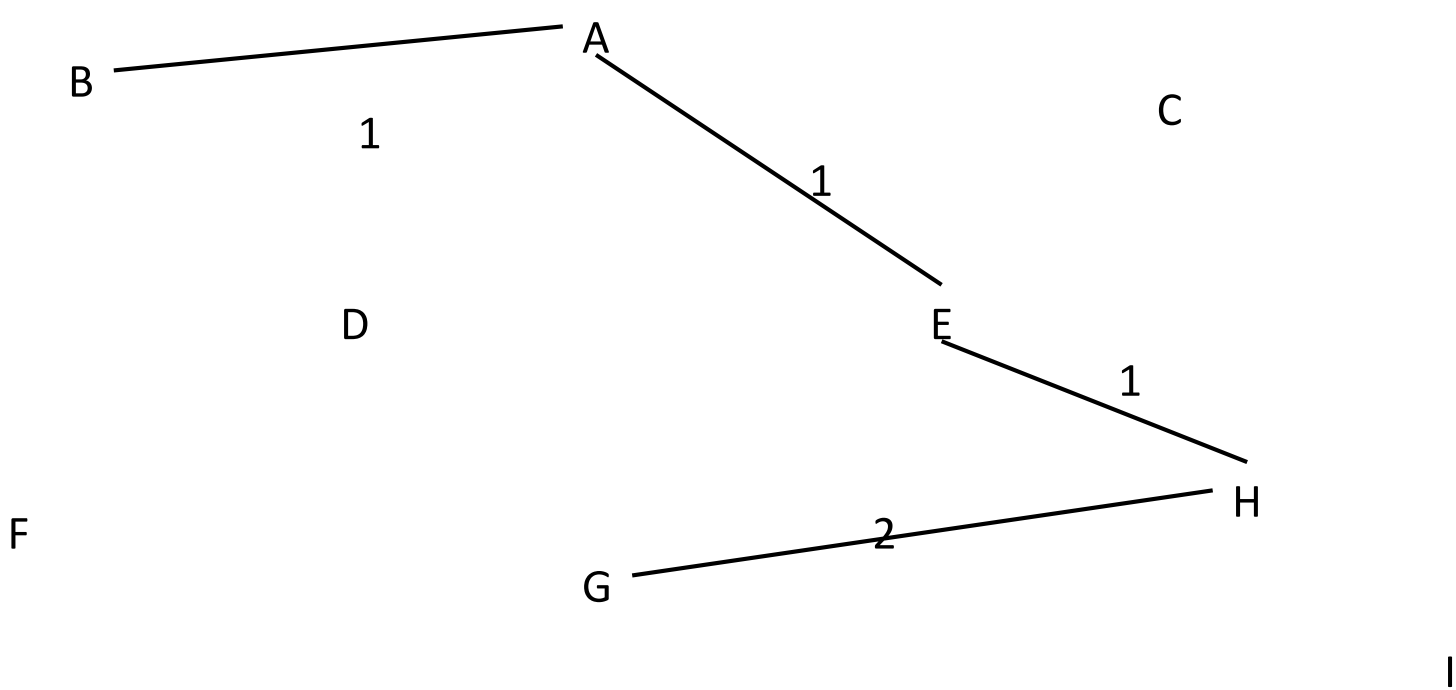
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
B	H	C	D	H	F	G	H	I



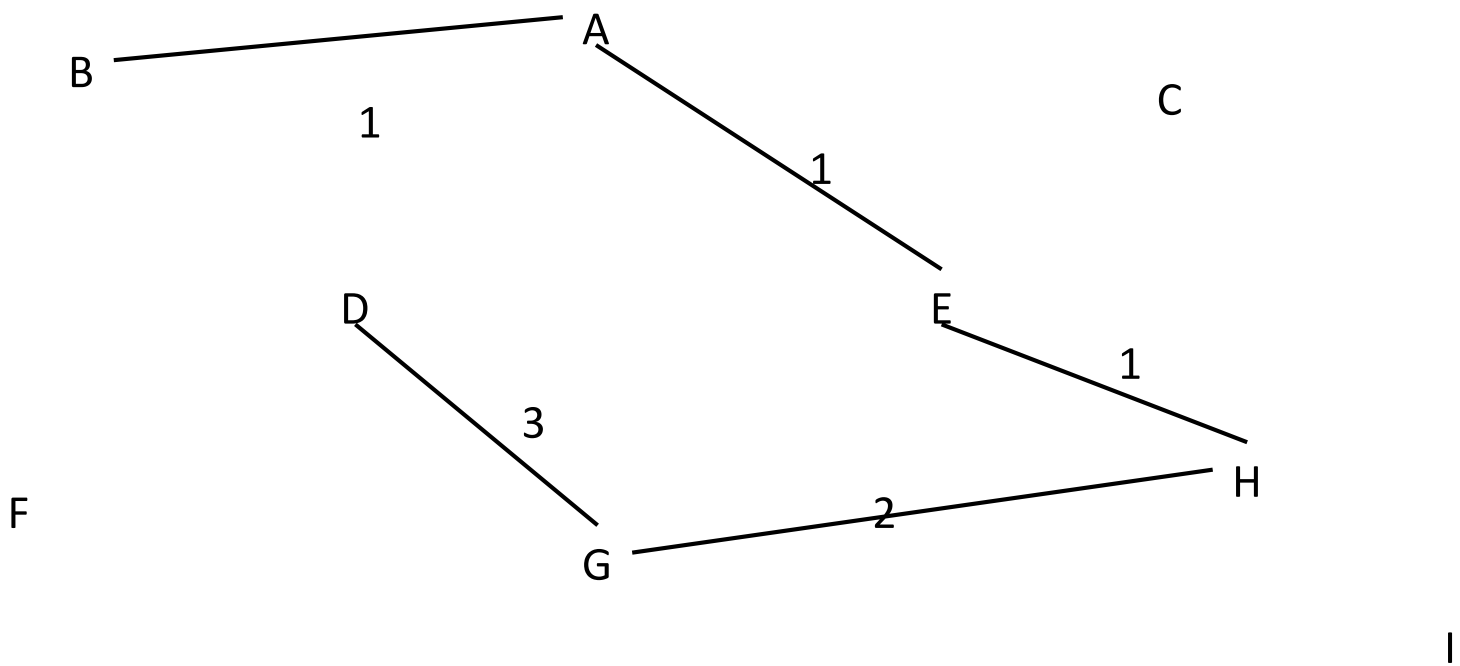
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
B	H	C	D	H	F	G	G	I



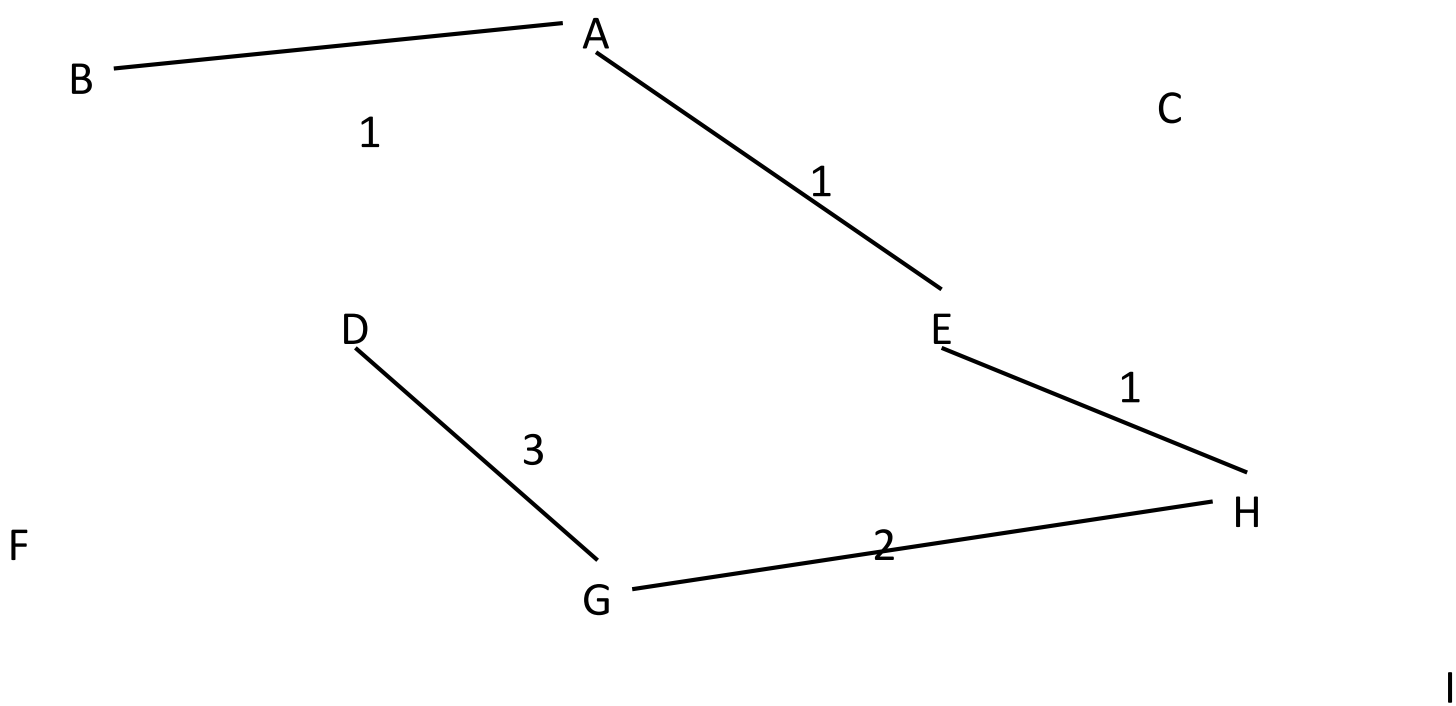
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
B	H	C	D	H	F	D	G	I



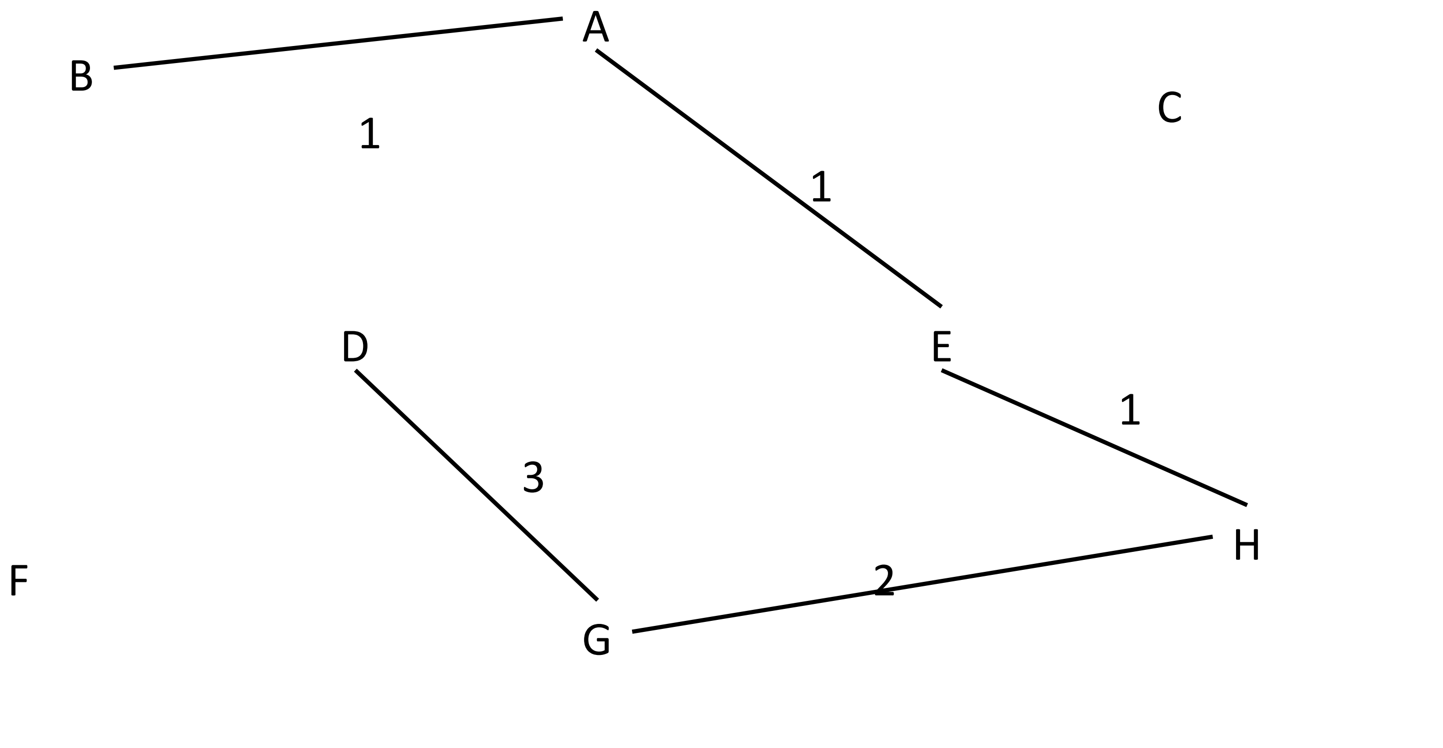
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
D	D	C	D	H	F	D	D	I



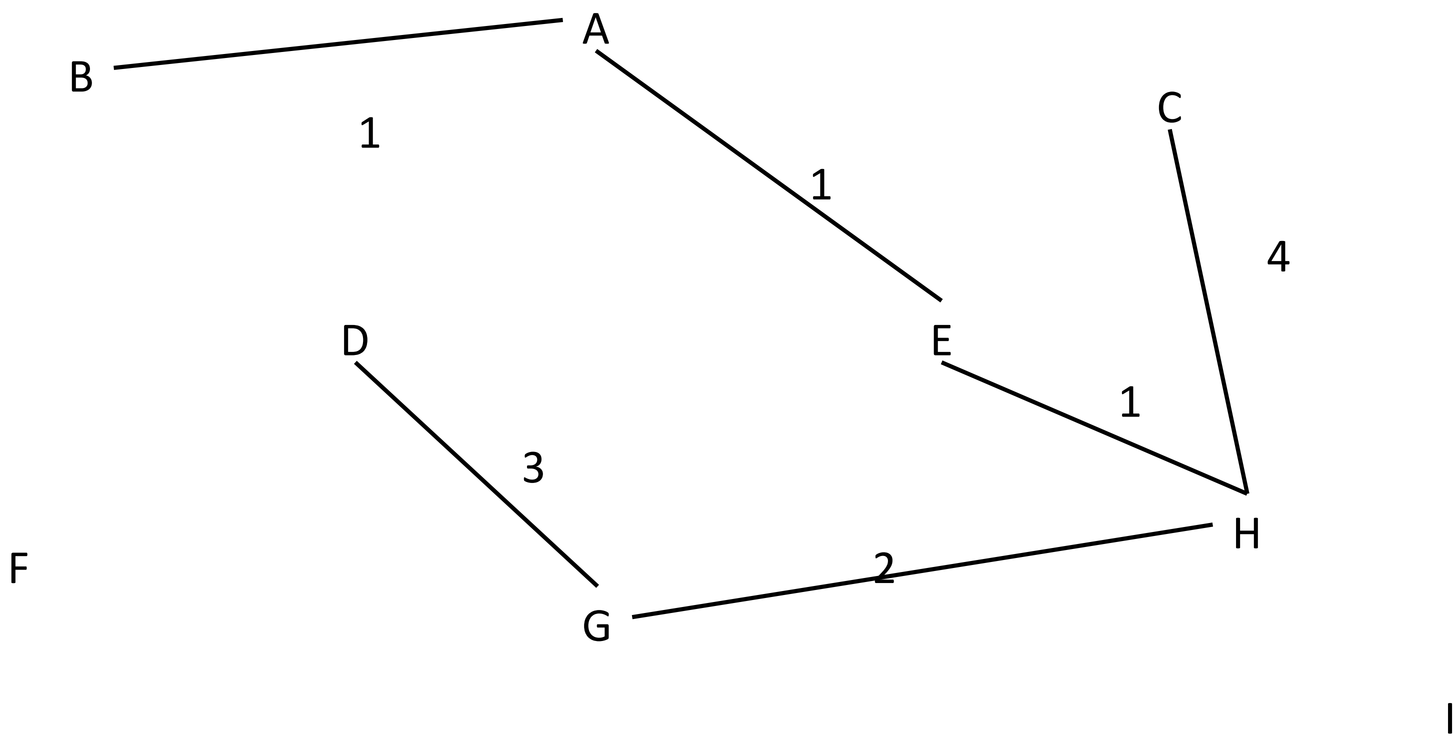
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
D	D	C	D	D	F	D	D	I



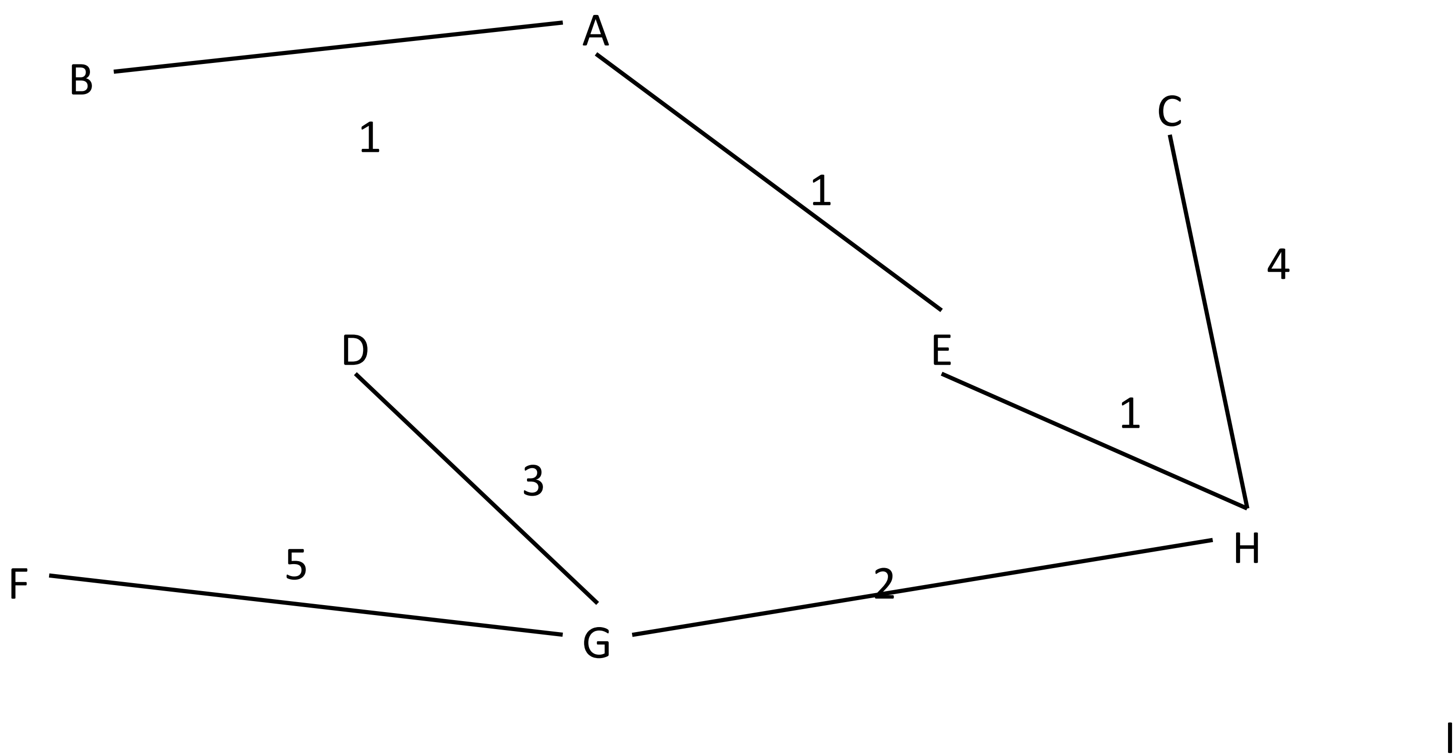
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
D	D	D	D	D	F	D	D	I



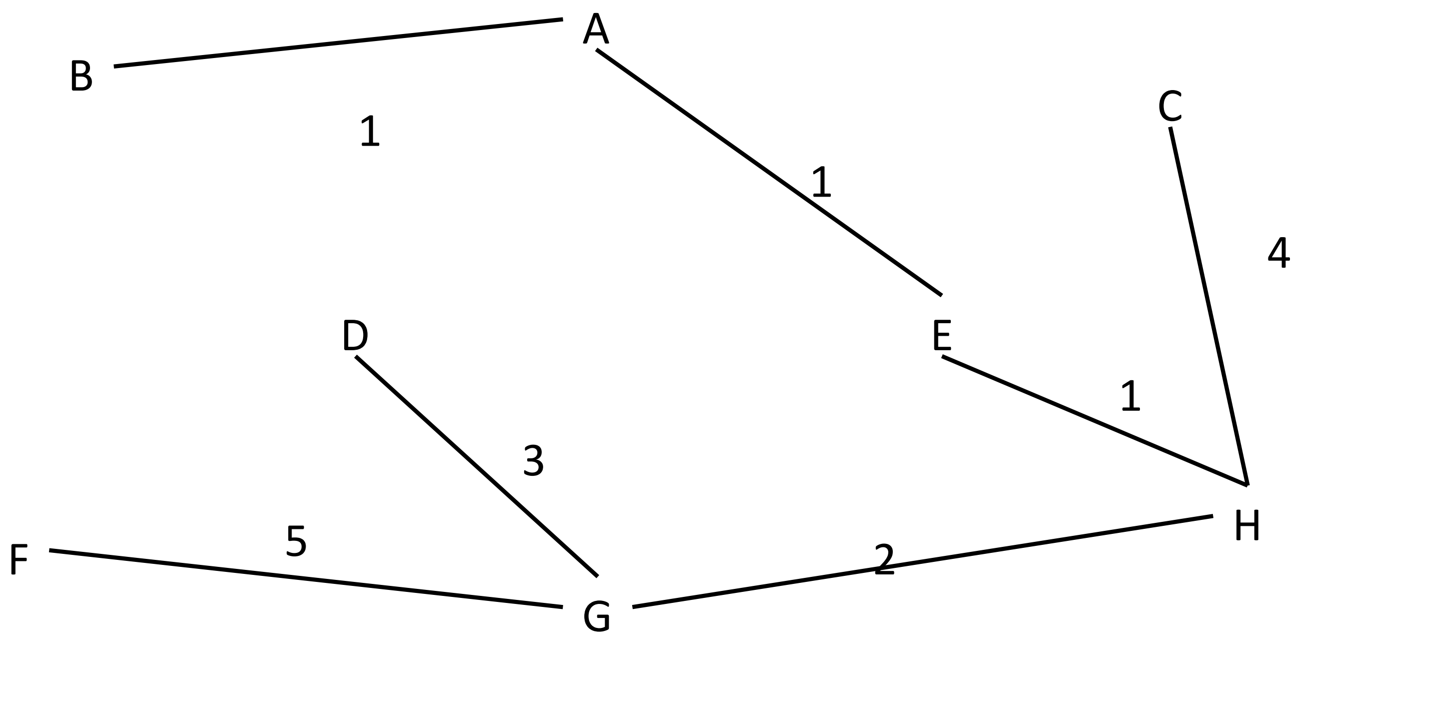
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
D	D	D	F	D	F	D	D	I



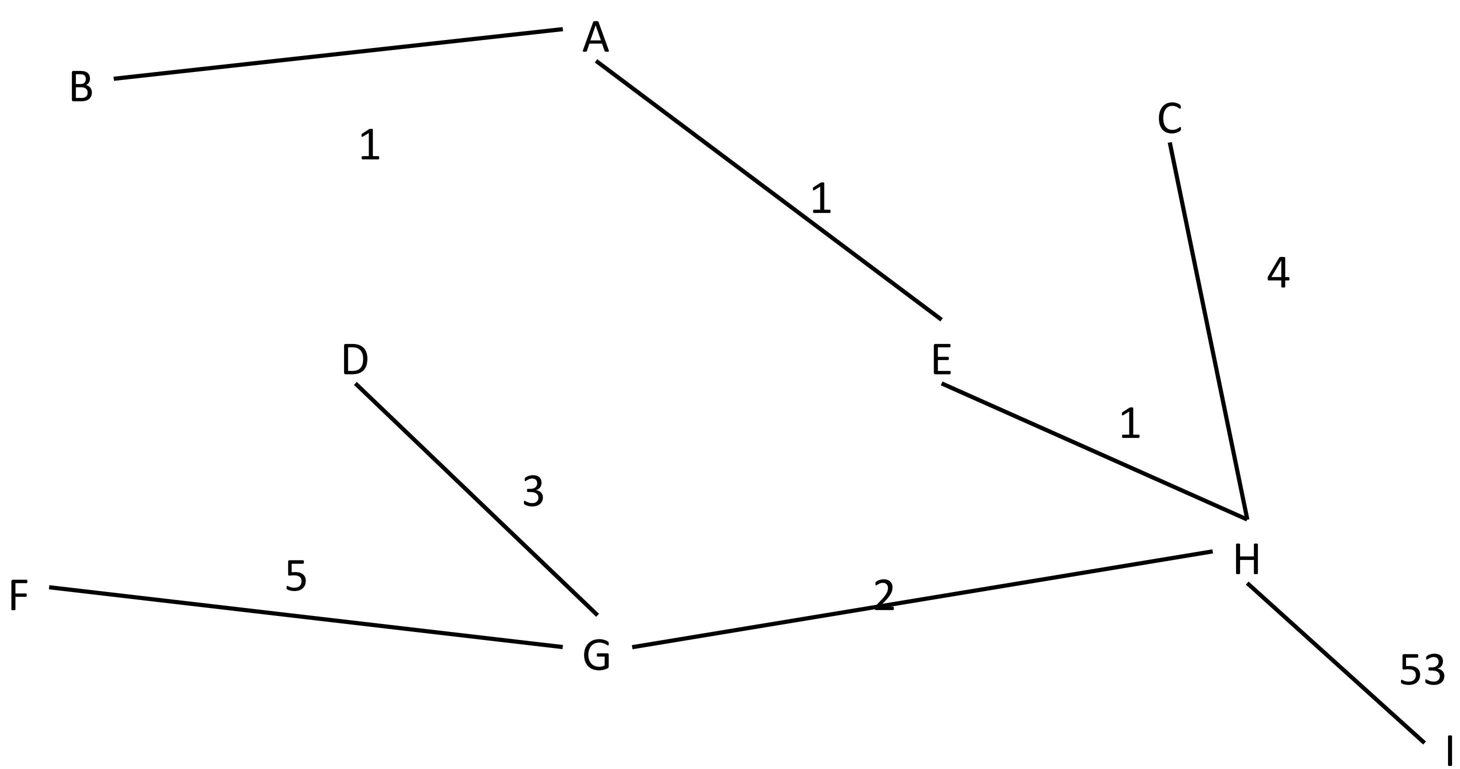
<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
F	F	F	F	D	F	D	D	I



<(EH,1),(AB,1),(AE,1),(HG,2),(GD,3),(AD,4),(EG,4),
(CH,4),(GF,5),(DF,5),(BF,6),(AC,7),(HI,53)}

A	B	C	D	E	F	G	H	I
F	F	F	F	D	I	D	F	I



Complexité

- Coût du tri des arêtes O(m log n)
- Les opérations d'union et de find (n unions et m find) coûtent O(n+m α(n,m))
- n nombre de sommets, m nombre d'arêtes
- α(n,m) est l'inverse de la fonction de Ackerman pour les valeurs courrantes, on considère que α(n,m) < 4