Eléments de solution du TD 2

Alain Cournier

Exercice 1 Question 1 : Idées

- Soit L la liste de sommets indiquant le chemin.
- Si la liste L est vide nous sommes en présence d'un chemin (C'est le chemin vide).
- Si L ne contient qu'un élément la réponse est vrai (le chemin se résume à un sommet)
- Si L contient au moins deux éléments soit x le premier et y le second il faut tester si xy est un arc du graphe et si L privée de son premier élément est un chemin. Si la réponse est deux fois oui on répond vrai dans le cas contraire on réponds faux

Exercice 1 Question 1 : Fonction

Fonction TesteChemin

Données : G : Un graphe, L : Une liste de sommets;

Résultat Booléen

Variable x : sommet

DébutCode

    Si TestListeVide(L) alors renvoyer (Vrai)
    Sinon x← Premier(L);
        Si TestListeVide (Suite(L)) alors renvoyer(Vrai)
        Sinon renvoyer (appartient (Premier(Suite(L)), G[x]) et (TesteChemin(G, Suite(L))

FinCode

Exercice 1 Question 1 : Complexité

- Le coût est lié au test d'appartenance
- Si on a une représentation par matrice : O(Longueur(L))
- Si on a une représentarion par liste d'adjacence O(n*Longueur(L))

Exercice 1 Question 2 : Idées

- Pour être un chemin élémentaire il faut d'abord être un chemin
- De plus il ne faut pas avoir de répétition dans la liste L

Exercice 1 Question 2 : Fonction

Fonction TesteCheminElémentaire

Données : G : Un graphe, L : Une liste de sommets;

Résultat Booléen

Variable

DébutCode

    Renvoyer (TesteChemin (G,L) et (SansDoublons(Trier(L)))

FinCode

Exercice 1 Question 2 : Fonction

Fonction SansDoublons

Données : L : Une liste triée de sommets;

Résultat Booléen

DébutCode

    Si TestListeVide(L) alors Renvoyer(Vrai)

    Sinon

        Si Test ListeVide(Suite(L)) alors Renvoyer(Vrai)

        Sinon Renvoyer((Premier(L) ≠ Premier(Suite(L))) et SansDoublons(Suite(L)))

FinCode

Exercice 1 Question 2 : Complexité

- Le coût est lié au test de chemin et à la fonction sans doublons
- TestChemin
  - Si on a une représentation par matrice : O(Longueur(L))
  - Si on a une représentation par listes d'adjacence O(n*Longueur(L))
- SansDoublons : O(|L| log |L|)
- D'où un coût global
- Si on a une représentation par matrice : O(|L| log |L|)
- Si on a une représentation par listes d'adjacence O(n*Longueur(L))

Exercice 1 Question 3 : Idées

- Pour être un chemin hamiltonnien il faut d'abord être un chemin élémentaire
- De plus il faut |L| = |X|

Exercice 1 Question 3 : Fonction

Fonction TesteCheminHamiltonnien

Données : G = (X,U) : Un graphe, L : Une liste de sommets;

Résultat Booléen

Variable

DébutCode

    Renvoyer (TesteCheminElémentaire (G,L) et (Longueur(L)=|X|)

FinCode

Exercice 1 Question 3 : Complexité

- Le coût est lié au test de chemin et à la fonction TestCheminElementaire
- D'où un coût global
- Si on a une représentation par matrice : O(|L| log |L|)
- Si on a une représentation par listes d'adjacence O(n*Longueur(L))

Exercice 2

- 1,0,2,3,4,5,8,7,6,9 : Parcours (Il est même en largeur)
- 0,4,8,6,9,1,2,5,7,3 : Parcours en profondeur
- 1,2,5,7,3,6,9,0,4,8 : n’est pas un parcours 6 ne peut pas être avant 0

Exercice 3

- L’idée est de lancer un parcours (VisiteGraph) à partir de notre sommet.
- Au retour Exploré contient les descendants de notre sommet.

Fonction Exercice 3 (entête)

- Fonction CalculeDesc
  - Donnée : G = (X,U) : Un graphe; x un sommet de G
  - Résultat : Ensemble de sommets
  - Variables :
    - Exploré : ensemble de sommets
    - N°Emp, N°Dep : entiers
    - TEmp, TDep : tableaux d'entiers indicés par X

Fonction Exercice 3 (Code)

- DébutCode
  - Exploré ← {}; N°Emp ← 1; N°Dep ← 1;
  - VisitGraphProf(G,x,Exploré,N°Emp,N°Dep,TEmp, TDep)
  - Renvoyer(Exploré)
- FinCode

Exercice 3 (Complexité)

- La complexité est identique à celle du parcours en profondeur

Exercice 4

- L’ordre inverse de l’ordre de dépilement donne une extension linéaire du graphe G.
- Or l’ordre de dépilement se calcule en utilisant un algorithme de parcours en profondeur (Voir cours)
- Il suffit donc d’écrire un algorithme qui inverse l’ordre de dépilement des éléments dans le tableau

Exercice 4

Algo InverseTableau

Donnée/Résultat : T Un tableau d'entier indicé de IBas(T) à IHaut(T);

Variable i, j : indices; n : entier

DébutCode

    i ← IBas(T); j ← IHaut(T); n ← IHaut - IBas + 1;

    TantQue i ≤ j faire

        T[i] ← n+1-T[i]; i ← i + 1;

FinCode

Fonction Exercice 4 (entête)

- Fonction ExtLinéaire
  - Donnée : G = (X,U) : Un graphe
  - Résultat : tableaux d'entiers indicés par X
  - Variables :
    - Exploré : ensemble de sommets
    - x : un sommet de G
    - N°Emp, N°Dep : entiers
    - TEmp, TDep : tableaux d'entiers indicés par X

Fonction Exercice 4 (Code)

- DébutCode
  - Exploré ← {}; N°Emp ← 1; N°Dep ← 1;
  - Pour tout x ϵ X faire
    - VisitGraphProf(G,x,Exploré,N°Emp,N°Dep,TEmp, TDep)
  - FinPour
  - InverseTableau(TDep); Renvoyer(TDep)
- FinCode

Exercice 4 (Complexité)

- La complexité est identique à celle du parcours en profondeur

Exercice 4 : Application du cours

- Soit xy un arc du graphe,
- Il faut étudier deux cas :
- Cas 1 : TDep[y] > TDep[x] et TEmp[y] < TEmp[x]
  - y a été empilé avant x et y sera dépilé après x : alors l'arc xy ferme un circuit.
  - Or notre graphe est sans circuits
- Cas 2 : TDep[y] > TDep[x] et TEmp[y] > TEmp[x]
  - Ce cas est exclu pour le parcours en profondeur
- Donc dans un graphe sans circuits si xy est un arc du graphe on a TDep[y] < TDep[x]

Exercice 5 : Idées

- On fait un parcours en profondeur pour récupérer un numéro d'empilement et de dépilement de chaque sommet.
- On initialise un Tableau T (Pour les opérations d'Union Find) et une Pile
- On prend ensuite chaque sommet x dans l'ordre d'empilement pour effectuer un traitement :
  - On empile x dans P
  - Tq Sommet de la pile est le suivant dans l'ordre de dépilement alors il faut faire un traitement
    - P' reçoit une copie de P (on ne veut pas détruire P)
    - On cherche dans la pile t le voisin de sommet(P) ayant un numéro de dépilement maximal
    - Tq Sommet(P') <> t faire Union(T, x, Sommet(P')) Depiler(P') finTq
    - Dépiler(P)
- Le tableau T contient alors les composantes fortement connexes

Exercice 5 : Algo (Entête)

Algo CFC

Donnée G=(X,U) : Un graphe

Résultat T : Un tableau de sommets indicé par des sommets

Variables

NumEmp, NumDep : Tableau d'entier indicé par les sommet

OrdreEmp, OrdreDep : Tableau de sommets indicé par des entiers [1..|X|]

P, P' : piles

x,y deux sommets

Variables du parcours en profondeur

Exercice 5 : Algo (Code)

DébutCode

Exploré ← {}; N°Emp ← 1; N°Dep ← 1;

Pour tout x ϵ X faire

    VisitGraphProf(G,x,Exploré,N°Emp,N°Dep,TEmp,TDep); T[x] ← x

FinPour

DepCour ← 1; P ← PileVide(); P’ ← PileVide()

Pour tout x ϵ X faire

    OrdreEmp[NumEmp[x]]←x; OrdreDep[NumDep[x]]←x

FinPour Suite→

Exercice 5 : Algo (CodeSuite)

```
Pour i allant de 1 à |X| faire
    Empiler(OrdreEmp[i],P);
    TantQue non(TestPileVide(P)) etalors Sommet(P) = OrdreDep[DepCour] faire
        P' ← P; DepCour ← DepCour + 1; x ← Sommet(P); Dépiler(P)
        y ← ChoisirSucc(G, x, NumEmp, NumDep)
        Tq Sommet(P') <> y faire Union(T, x, Sommet(P')); Depiler(P') finTq
        Union(T, x, y)
    Fintq
FinPour
// T contient une représentation de la partition en CFC
FinCode
```

Exercice 5

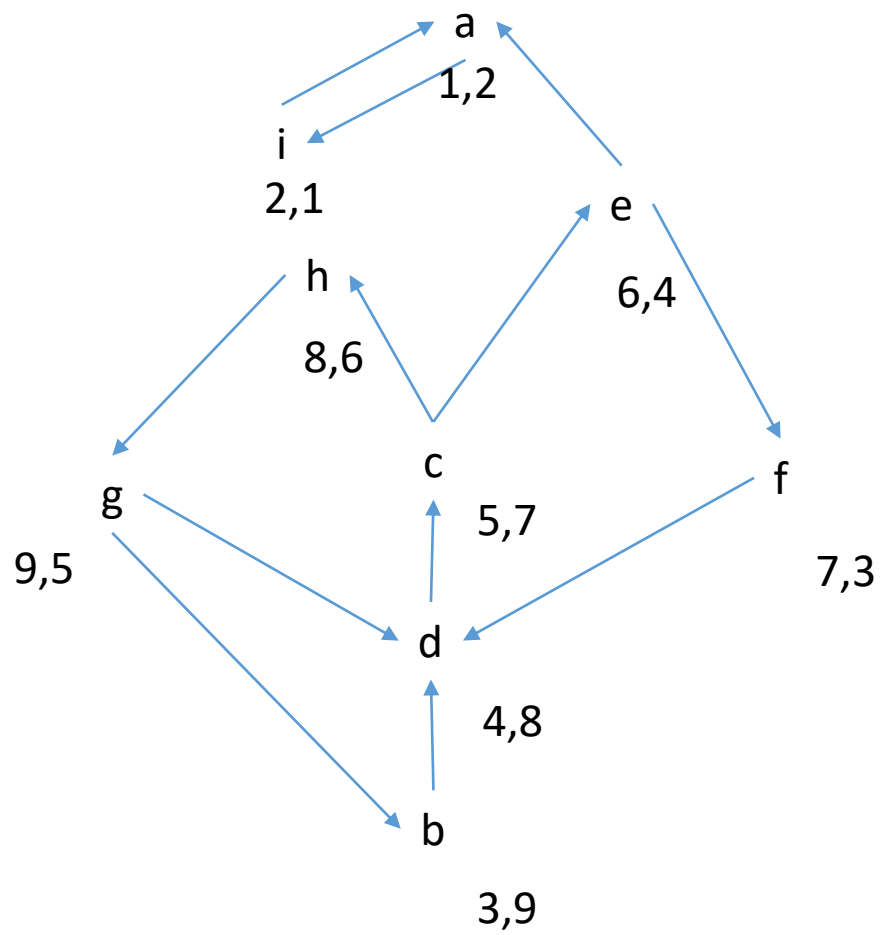
```
Fonction ChoisirSucc
    Données
        G : Un graphe       x : Sommet          NumEmp, NumDep : Tableaux
    Résultat : Sommet
    Variable y, z : Sommets
    DébutCode
        y ← x;
        Pour tout z dans Succ(x) faire
            Si TDep[z] > TDep[x] et TEmp[z] < TEmp[x] et TDep[z] > TDep[y] alors y ←z finsi
        FinPour
        Renvoyer(y)
    FinCode
```

Exercice 5 : Algo (Complexité)

Cette version de l'algorithme est quadratique : $O(n^2)$ (Recopie de la pile n fois)

Toutefois, une version en $O(n+m\ \alpha(n,m))$ a été donnée par Tarjan (Document présent dans le moodle)

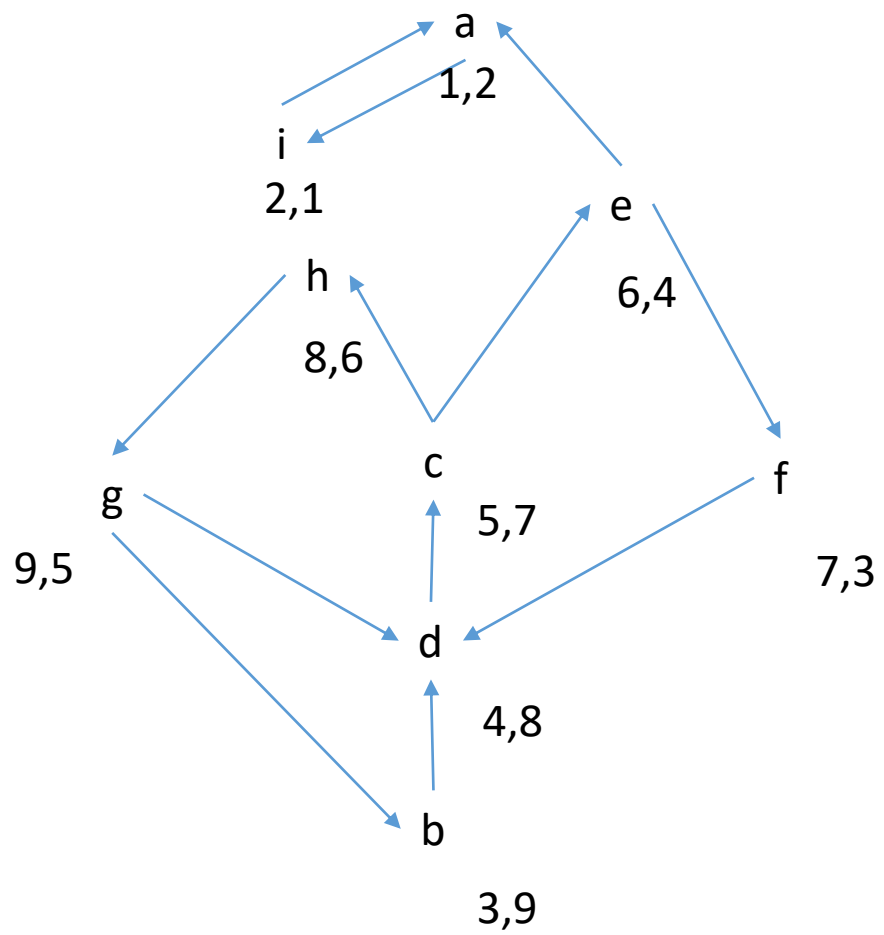
Exercice 5 : Exemple



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

P	P’

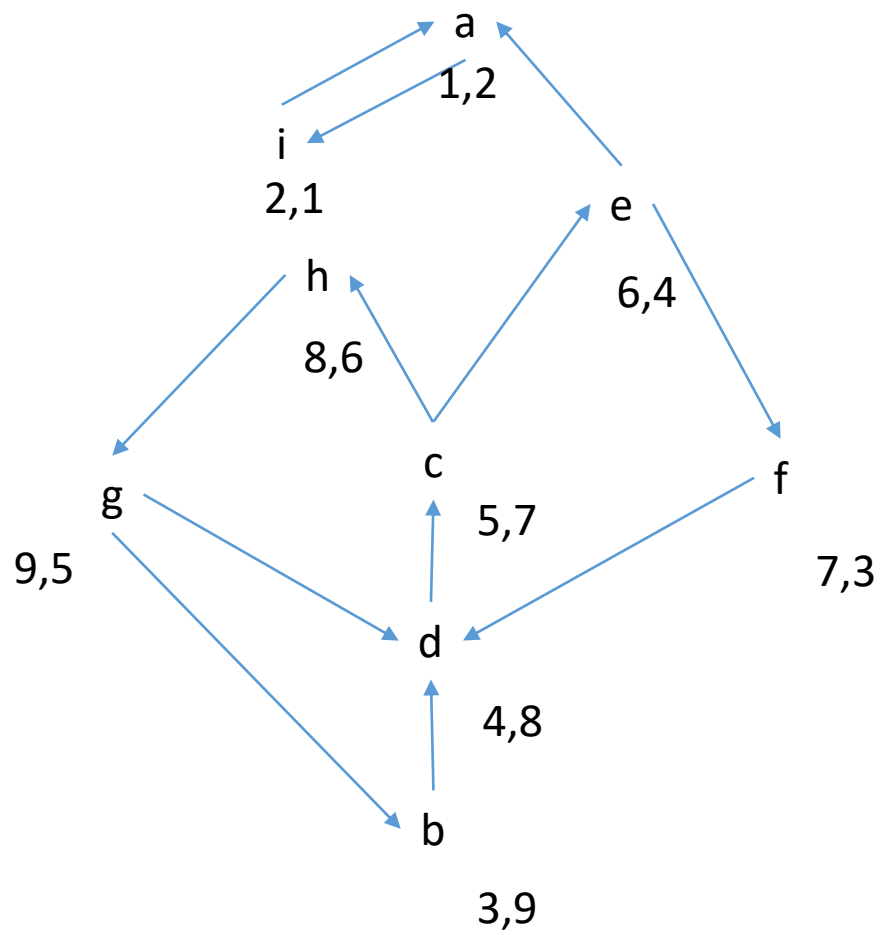
Exercice 5 : Exemple



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

P	P’
i	
a	

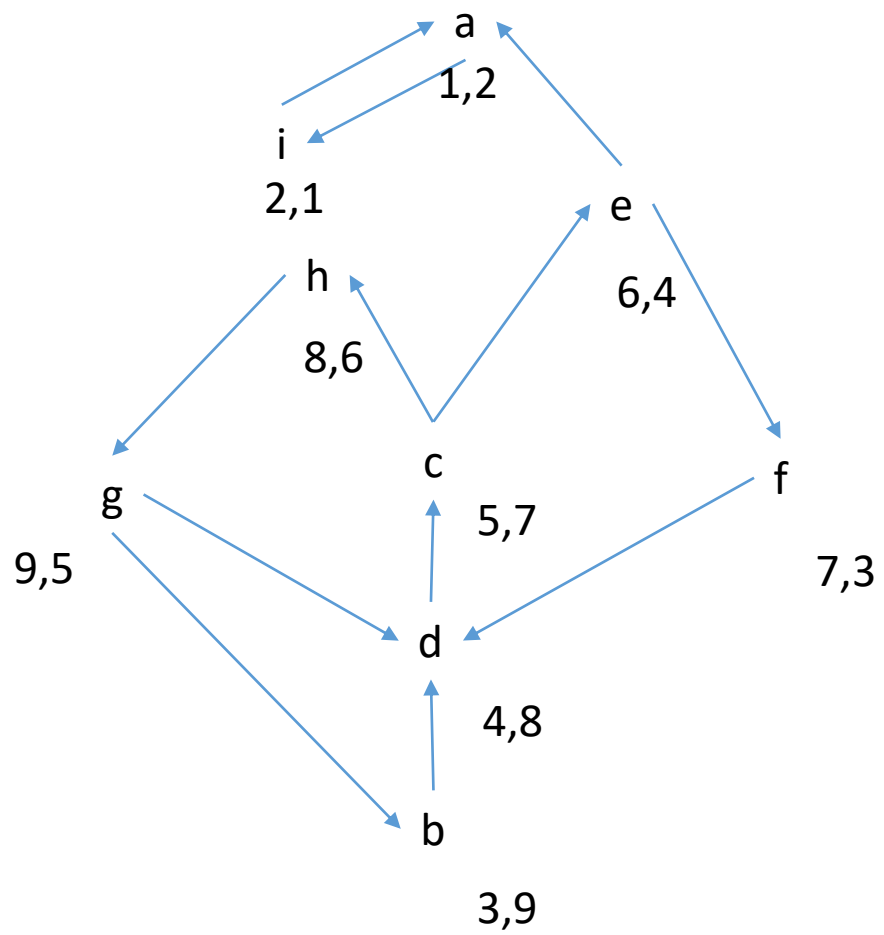
Exercice 5 : Exemple



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

P	P’
i	i
a	a

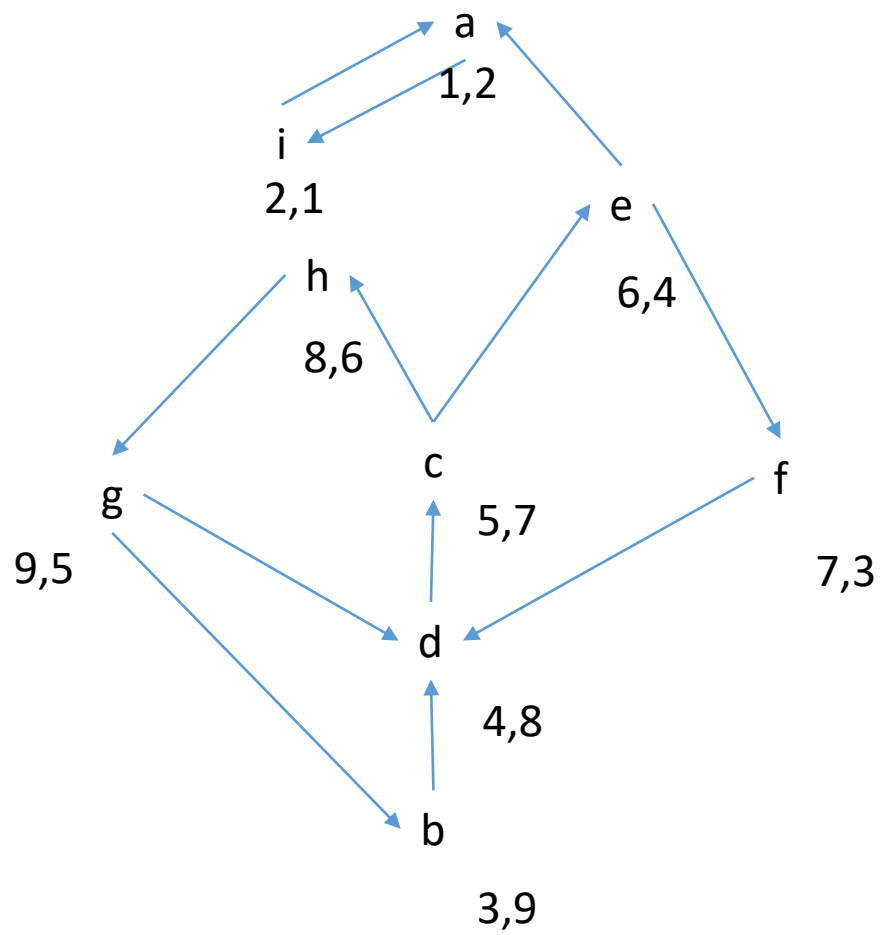
Exercice 5 : Exemple



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

P	P’
i	
a	a

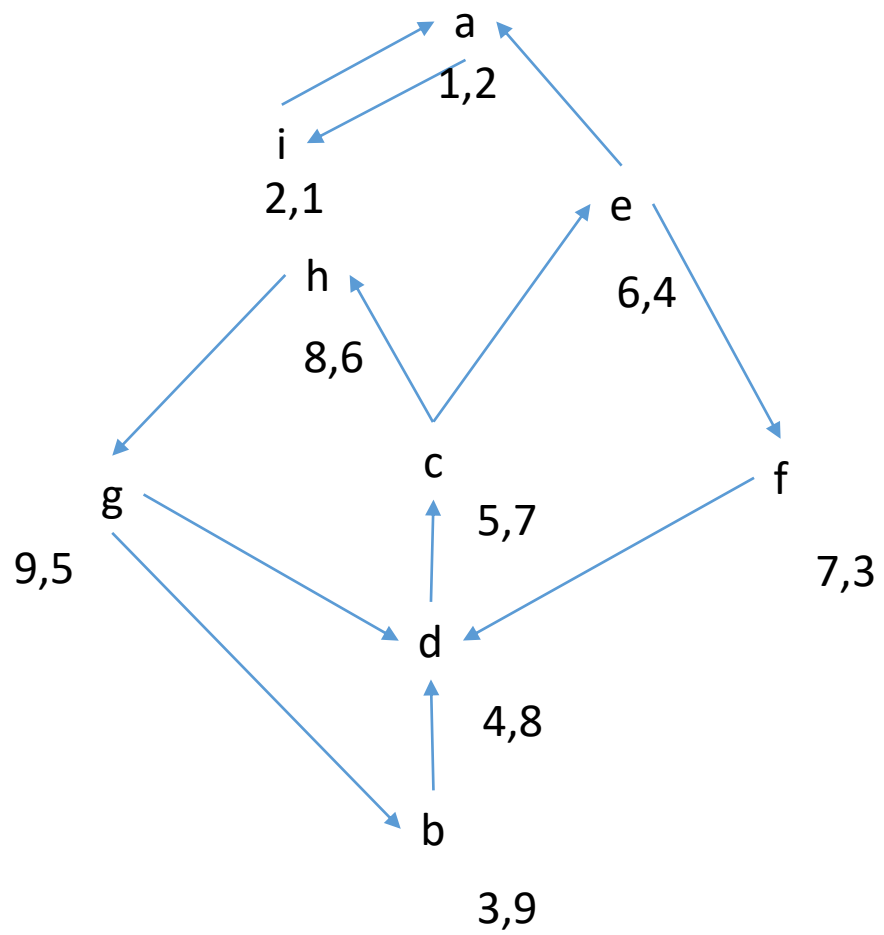
Exercice 5 : Exemple



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

P	P’

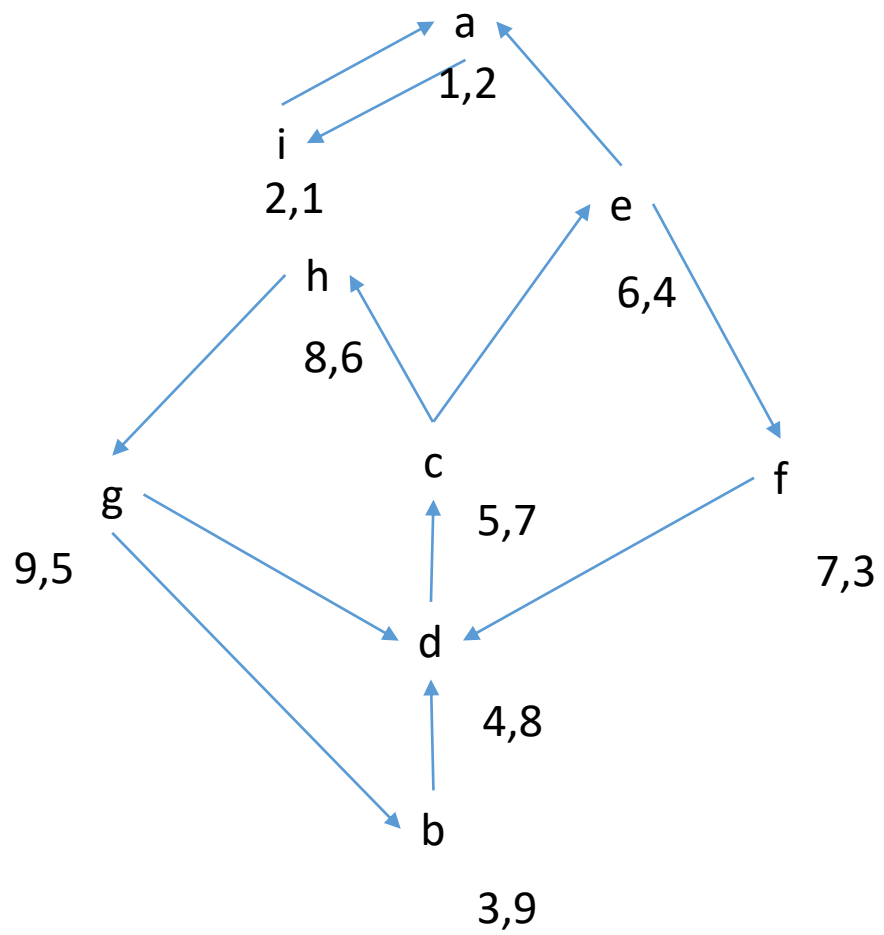
Exercice 5 : Exemple



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

P	P’
f	
e	
c	
d	
b	

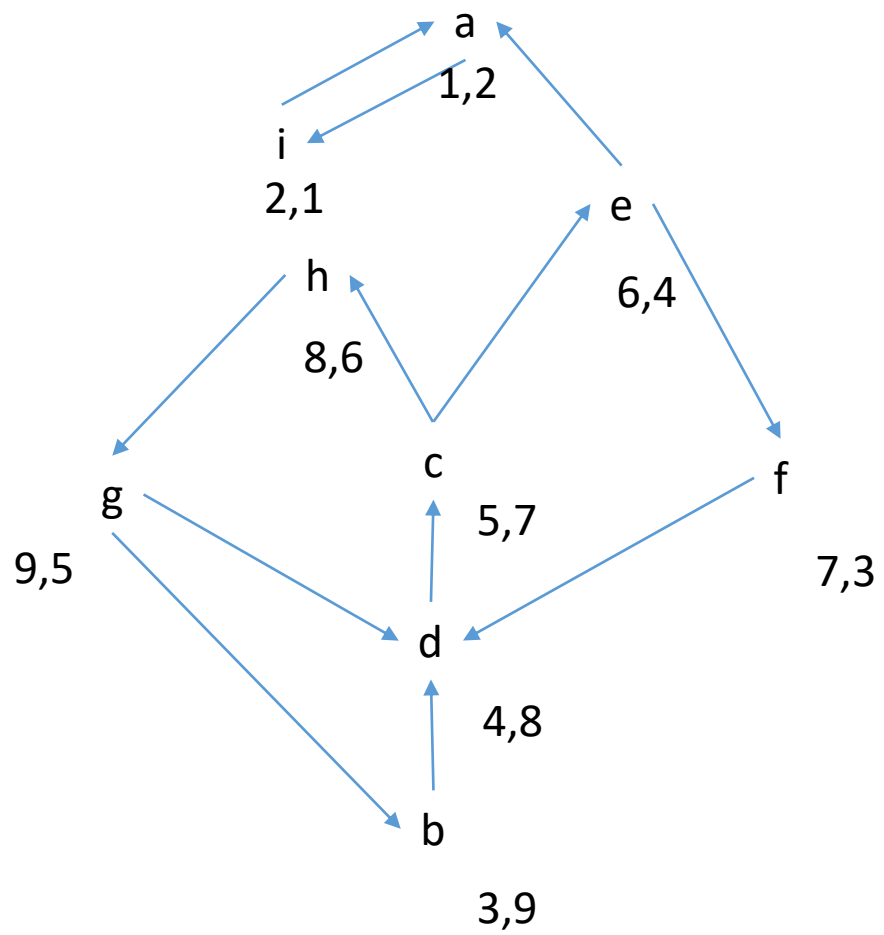
Exercice 5 : Exemple



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

P	P’
f	f
e	e
c	c
d	d
b	b

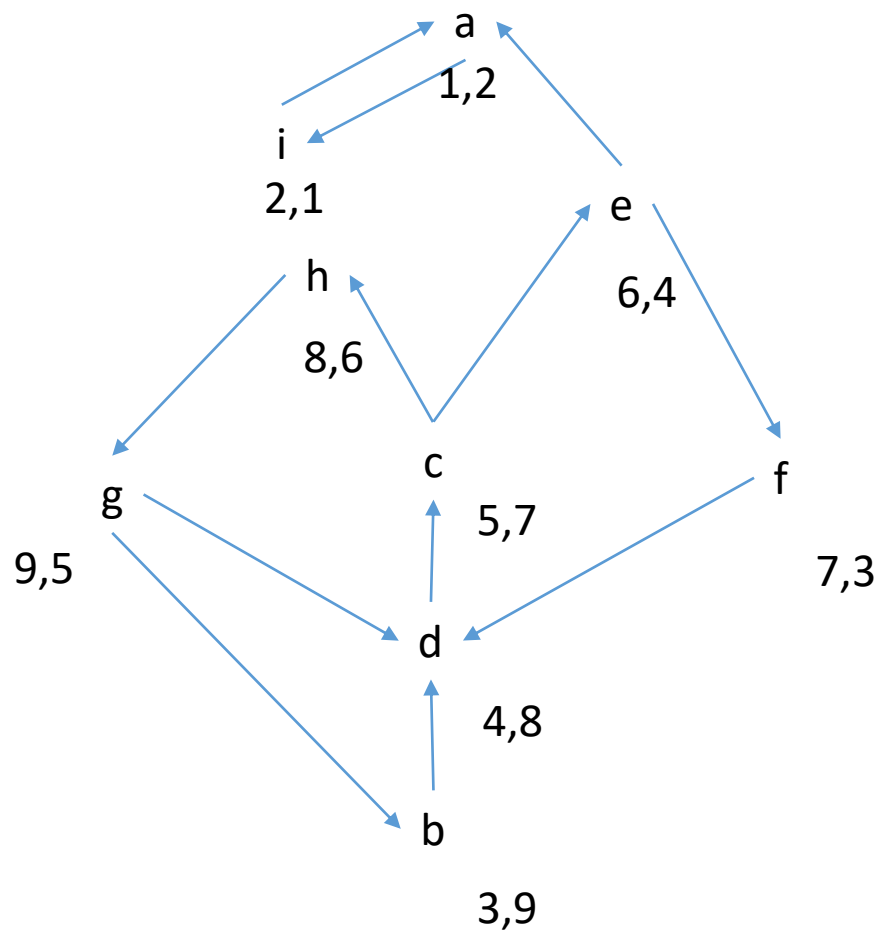
Exercice 5 : Exemple



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

P	P’
f	
e	
c	
d	d
b	b

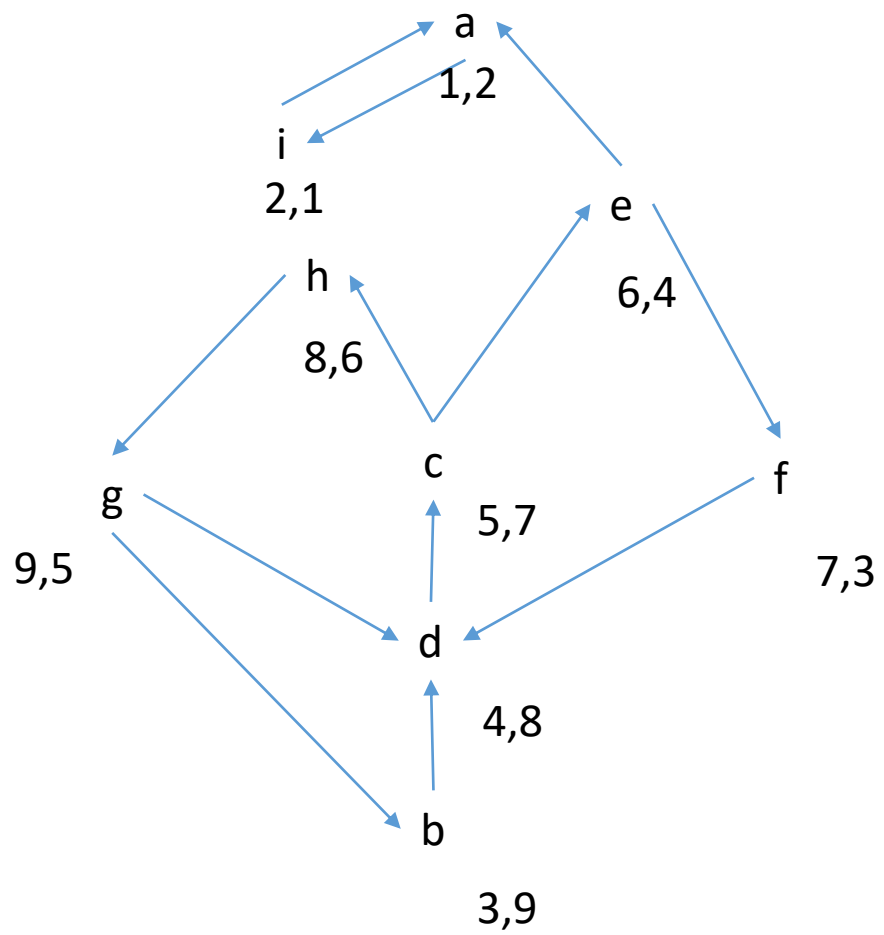
Exercice 5 : Exemple



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

P	P’
c	
d	d
b	b

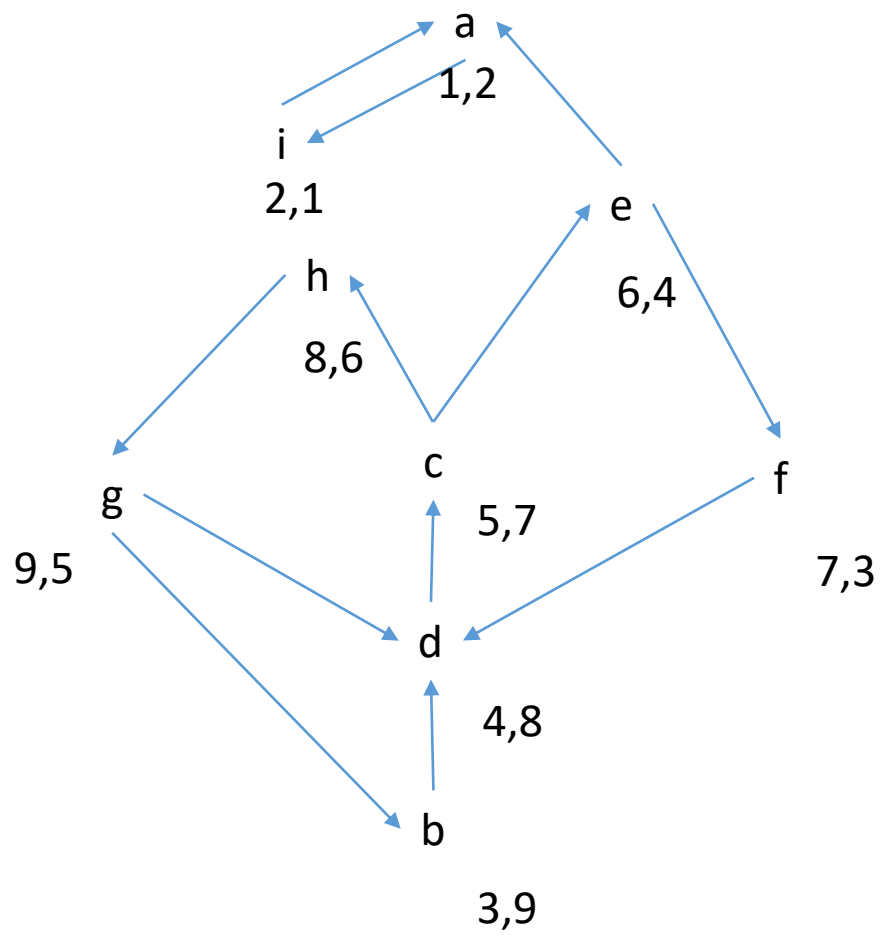
Exercice 5 : Exemple



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

P	P’
g	
h	
c	
d	d
b	b

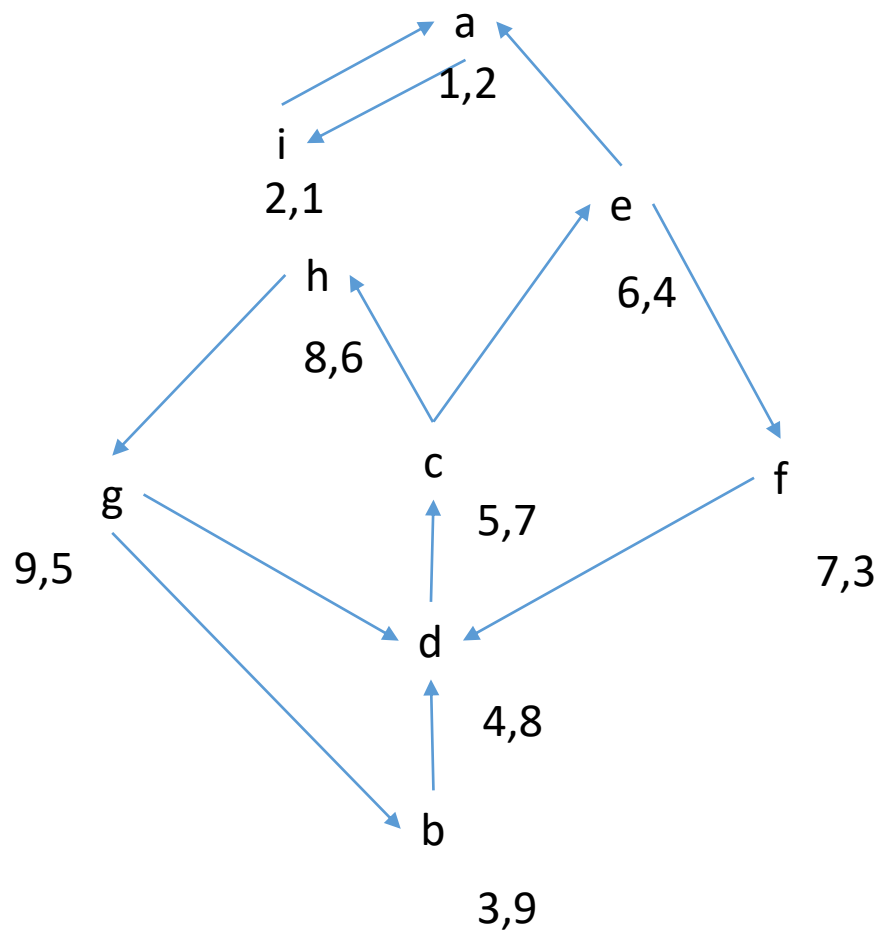
Exercice 5 : Exemple



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

P	P’
g	g
h	h
c	c
d	d
b	b

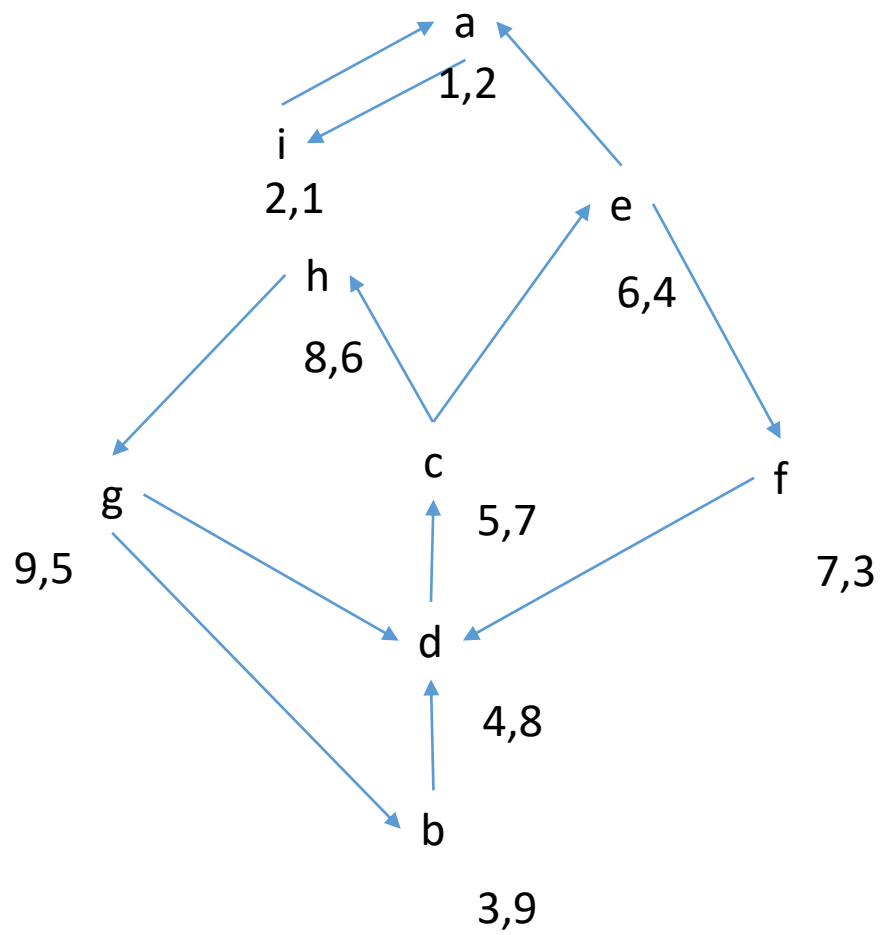
Exercice 5 : Exemple



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

P	P’
g	
h	
c	
d	
b	b

Exercice 5 : Exemple



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

P	P’
	b

Exercice 6

- Cet algorithme calcule lui aussi les CFC du graphe