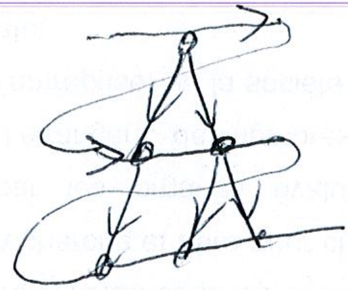
Parcours en largeur d'un graphe simple

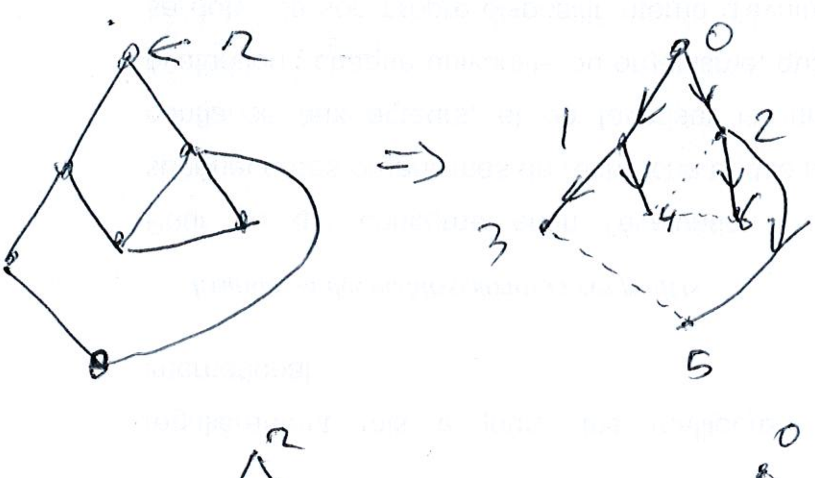
(anglais : ~~depth~~ breadth-first traversal)

Dans le cas d'une arborescence, on visite les sommets par ordre de profondeur



On peut le générali-ser à un graphe orienté ou non.

On choisit un sommet r, et l'algorithme construit un arbre enraciné en r qui est un graphe partiel du graphe de départ, maximal. Pour chaque sommet, il donne le père dans l'arbre, et la distance de ~~ce sommet à r~~ r à ce sommet.



n° = ordre dans le parcours

l'ordre des flèches doit être respecté

L'algorithme utilise 3 "couleurs" pour distinguer le statut des sommets :

blanc : pas encore visité

gris : visité, ayant des voisins non visités

noir : visité, dont les voisins le sont aussi.

Il utilise une file FIFO.

Initialisation

$\forall s \in S \setminus \{r\}$ | couleur(s) := blanc;
père(s) := NIL;
$d(s) := \infty$;

couleur(r) := gris;
père(r) := NIL;
$d(r) := 0$;
file vide
mettre r dans la file.

Corps

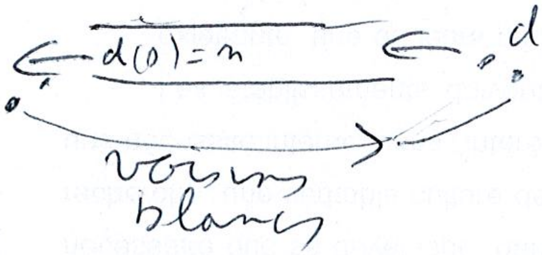
Tant que la file est non-vide :
sortir le sommet s en tête de file;
pour chaque sommet v adjacent à s : (orienté : $s \to v$; non : $s - v$)
si couleur(v) == blanc alors
couleur(v) := gris;
père(v) := s;
$d(v) := d(s) + 1$;
mettre v dans la file;
couleur(s) := noir;

Remarques sur l'algorithme.

– chaque sommet passe une seule fois dans la file. Il est blanc avant d'y entrer, gris à l'intérieur, et noir après être sorti.

– Les sommets passent dans la file dans l'ordre de $d(s)$.

Par récurrence, on sort les sommets de $d(s)=n$ et on insère les voisins avec $d(s)=n+1$



– Pour tout s, $d(s)$ est la distance de r à s. Par récurrence :

$d(r)=0$ OK

$s \neq r$. $d(s) = d(\text{père}(s))+1$

par hypothèse d'induction $d(\text{père}(s)) = \text{distance}(r, \text{père}(s))$

or s est voisin (sortant) de $\text{père}(s)$

donc $\text{distance}(r,s) \leq \text{distance}(r, \text{père}(s)) + 1$

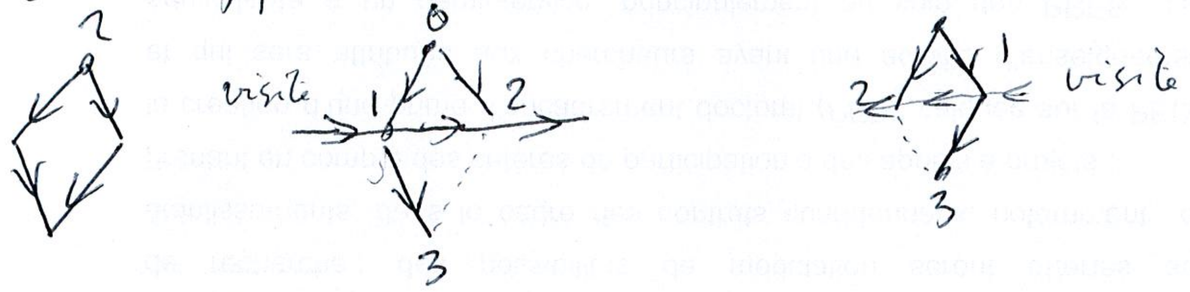
$= d(\text{père}(s))+1 = d(s)$

Mais si on avait $\text{distance}(r,s) < d(s)$, il y aurait t avec $d(t) < d(s)-1$ tel que s est voisin de t. Mais alors t est dans la file avant $\text{père}(s)$, donc s serait inséré dans la

file lors de la sortie de t, avec $\text{père}(s) = t$
et $d(s) = d(t) + 1$ . . X.

Donc $\text{distance}(r, s) = d(s)$

– L'ordre de visite des voisins d'un sommet influe sur le résultat : les pères peuvent changer, donc l'arborescence sera différente



– On a un chemin le plus court possible de r à s en faisant par récurrence chemin le plus court de r à $\text{père}(s)$, puis s

Ce chemin est obtenu en sens inverse comme la suite $s, \text{père}(s), \text{père}(\text{père}(s)), \ldots$
$\ldots, \overbrace{\text{père}(\ldots}^{d(s)\times}(s)\ldots) = r$

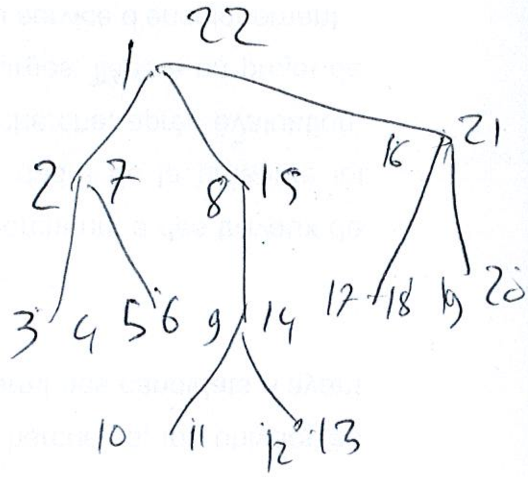
Ce n'est pas nécessairement l'unique plus court chemin (cf. supra)

– Dans le cas d'un graphe non orienté, l'arborescence construite sera un arbre couvrant de la composante connexe du graphe contenant 1.

Parcours en profondeur

(anglais : depth-first traversal)

Dans le cas d'une arborescence : on descend, en donnant une priorité sur la branche la plus à gauche possible



2 nombres : 1° et 2° passage sur le sommet

~~On l'a~~ Exemple ordre de succession ~~Cont~~ dans une monarchie

On va le généraliser à un graphe orienté ou non

Contrairement au parcours en largeur, on ne se limite pas à une racine

On va l'appliquer successivement jusqu'à obtenir une forêt couvrante

Pour chaque sommet s on construit deux nombres $d(s)$ et $f(s)$ (début et fin) qui donnent le temps du 1e passage (descendant) et du 2e (montant) sur ce sommet. 44

A nouveau, on associera à chaque sommet un père dans son arborescence, et une couleur

blanc : avant le 1e passage dessus
gris : entre les 2 passages
noir : après le 2e passage.

$d(s)$ est déterminé quand s passe de blanc à gris, et $f(s)$ est déterminé quand il passe de gris à noir. $[d(s) < f(s)]$

Algorithme de construction de la forêt

Initialisation

$\forall s \in S$ | couleur(s) := blanc;
| père(s) := NIL

temps := 0

Corps

tant qu'il reste des sommets blancs :
prendre s tel que couleur(s) == blanc
faire visite(s)

↳ fonction construisant l'arborescence de racine s.

Visite (s) Initialisation

couleur(s) := gris

temps++

d(s) := temps

Corps

pour chaque v adjacent à s (orienté $s \to v$, non $s - v$)

si couleur(v) == blanc alors

| père(v) := s;
| Visite(v).

couleur(s) := noir;

temps++;

f(s) := temps.

Remarques sur l'algorithme ($d(u) < f(u)$ $\forall u \in S$)

Les intervalles $[d(u), f(u)]$ sont soit disjoints, soit imbriqués.

4 cas $d(u) < d(v) < f(v) < f(u)$

v descendant de u dans son arborescence

$d(v) < d(u) < f(u) < f(v)$

u descendant de v

$d(u) < f(u) < d(v) < f(v)$

$d(v) < f(v) < d(u) < f(u)$

soit sur 2 arborescentes différentes, soit sur une même mais sans relation de descendance.

Tri topologique

Soit G un graphe orienté sans circuit simple. Un tri topologique de G est un ordonnancement temporel (linéaire) des sommets tel que pour 2 sommets u, v, si (u, v) est une arête, alors u est avant v.

Donc on numérote les sommets $s_1, \dots, s_n$ de sorte que pour $(s_i, s_j) \in A$ on a nécessairement $i < j$.

Application : ordonnancement de tâches liées par une relation de priorité : certaines tâches devant être effectuées avant d'autres, dans quel ordre les effectuera-t-on ? p. ex. dans les systèmes d'exploitation des ordinateurs.

Algorithme : faire un parcours en profondeur et ordonner les sommets par

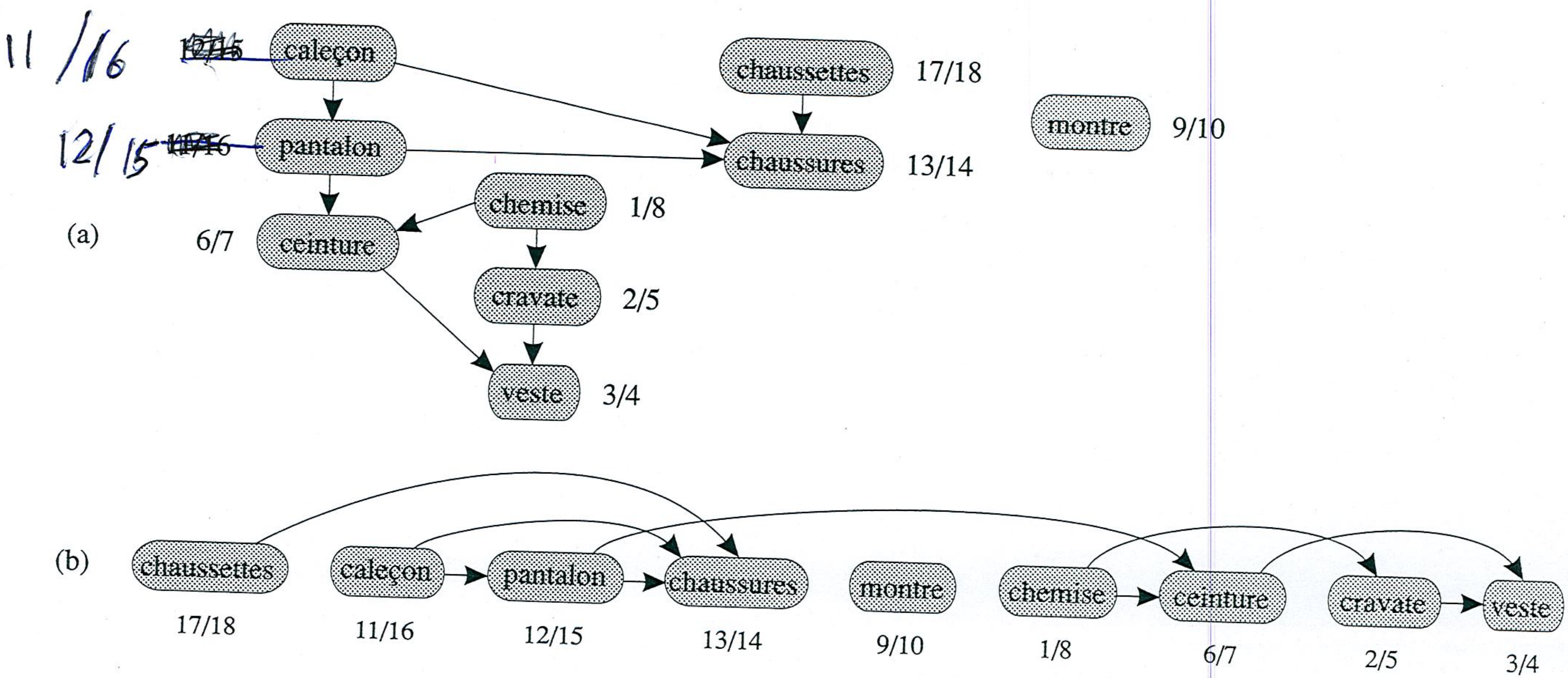


Figure 23.7 : **(a)** Le savant Cosinus trie topologiquement ses vêtements quand il s'habille. Chaque arc (u, v) signifie que le vêtement u doit être enfilé avant le vêtement v. Les dates de découverte et de fin de traitement lors d'un parcours en profondeur sont données à côté de chaque sommet. **(b)** Le même graphe trié topologiquement. Ses sommets sont ordonnés de gauche à droite par ordre décroissant de leur date de fin de traitement. On notera que tous les arcs sont orientés de gauche à droite.

aucun ordre linéaire n'est possible.) Le tri topologique d'un graphe peut être vu comme

[47]

valeur décroissante de f

c.-à-d. : chaque fois qu'un sommet u est terminé (au temps $f(u)$), on place u en tête de liste.

justification soit $u \to v$

si $d(u) < d(v)$, v sera dans l'arborescence sous u, soit directement comme fils, soit comme descendant. Donc $d(u)<d(v)<f(v)<f(u)$

si $d(v) < d(u)$, u ne peut pas être dans l'arborescence sous v, sinon on aurait un circuit, donc cette arborescence se termine sans passer par u, donc $d(v)<f(v) < d(u) < f(u)$

Ainsi $f(v) < f(u)$ dans les deux cas.

Détection de composantes fortement connexes d'un graphe orienté.

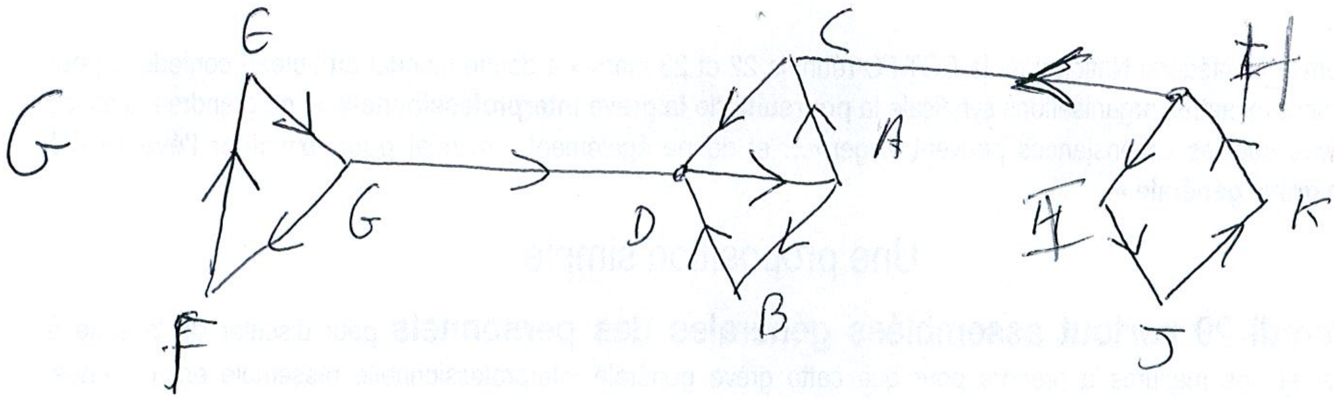
Algorithme :

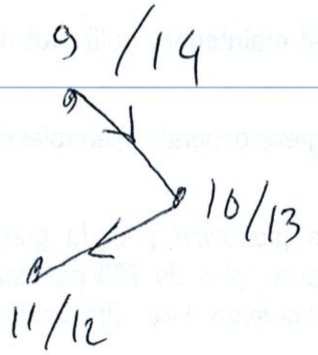
(1) Faire un parcours en profondeur, et retenir pour chaque sommet la valeur de f.

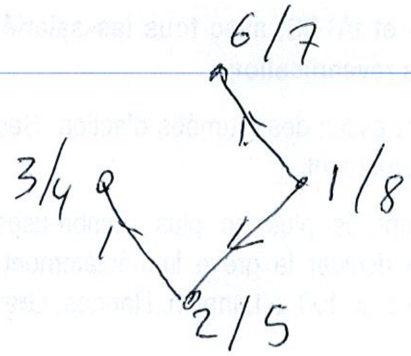
(2) Construire le graphe transposé (où les arcs sont inversés). Concrètement à partir de la liste de $\Gamma^+(s)$, $s \in S$, construire la liste des $\Gamma^-(s)$.

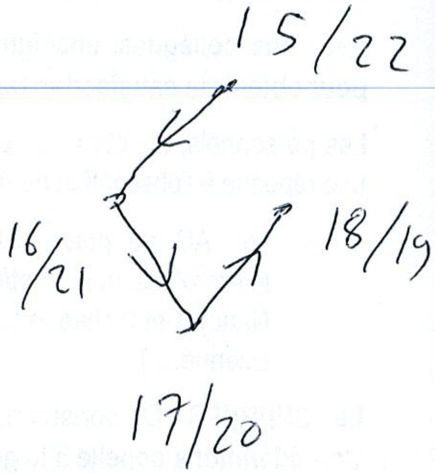
(3) Faire un parcours en profondeur du graphe transposé, où la sélection des racines des arborescences se fait selon l'ordre décroissant des valeurs de f calculées en (1)

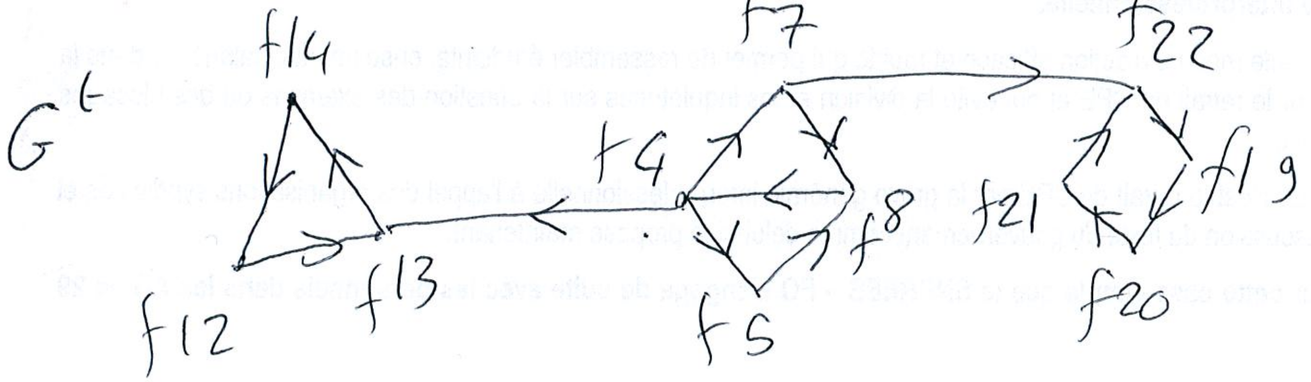
50/22

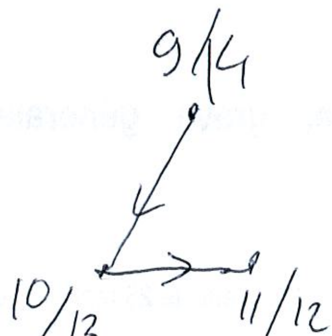

E
G
G
F
C
D
A
B
H
I
K
J

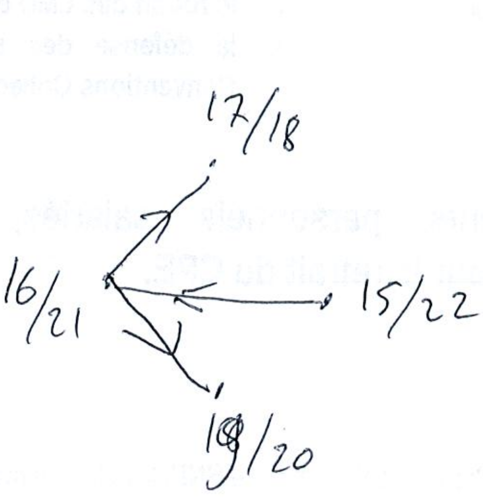

9/14
10/13
11/12

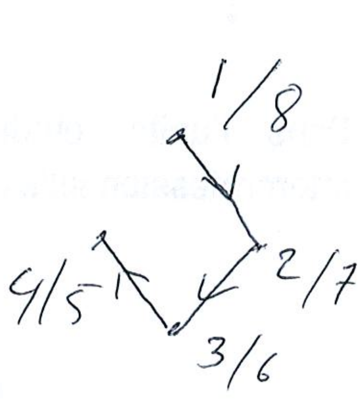

6/7
3/4
1/8
2/5


15/22
16/21
18/19
17/20

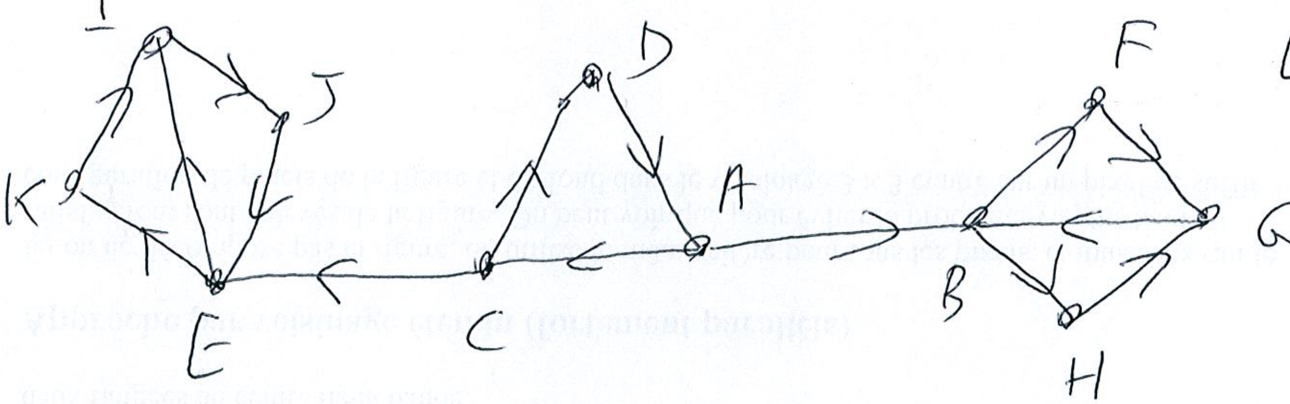

G
f14
f12
f13
f7
f4
f8
f5
f22
f19
f21
f20


9/14
10/13
11/12

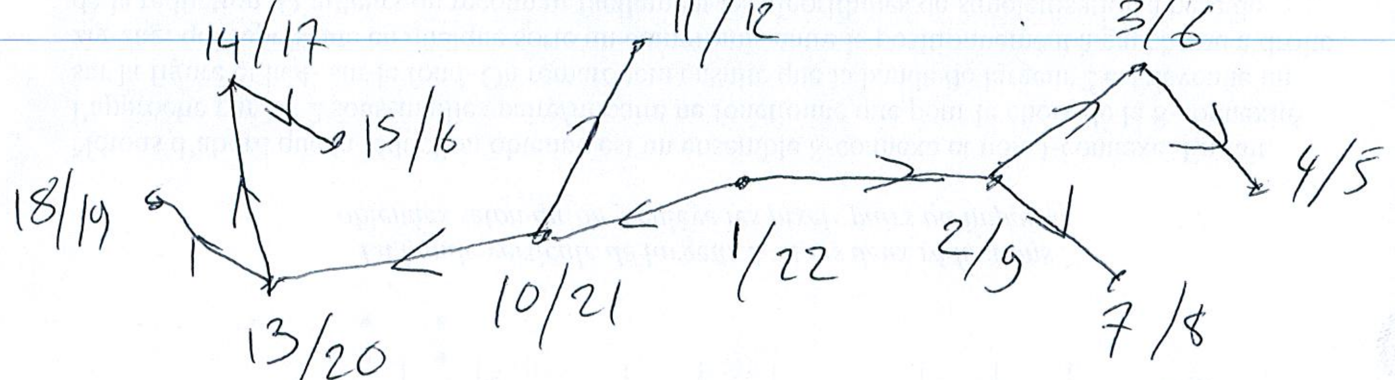

17/18
16/21
15/22
18/20


1/8
4/5
2/7
3/6

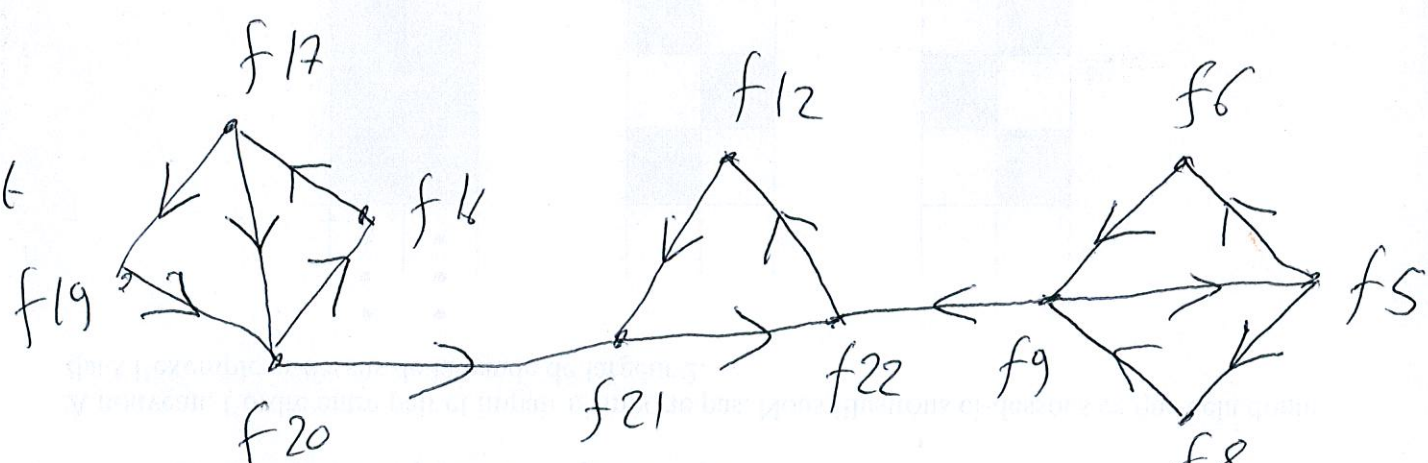
51/23

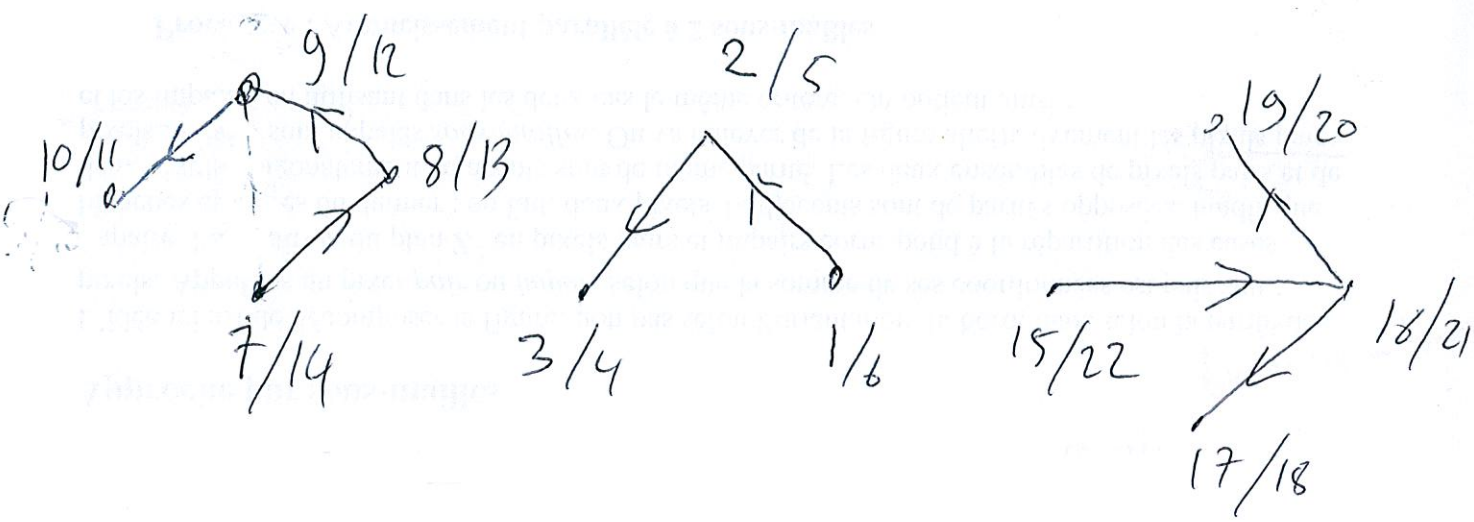

I
J
K
D
A
C
E
F
G
B
H

G


14/17
11/12
3/6
15/16
18/19
4/5
2/9
1/22
10/21
13/20
7/8

G^t


f17
f12
f6
f16
f19
f5
f22
f9
f20
f21
f8


9/12
2/5
19/20
10/11
8/13
7/14
3/4
1/6
15/22
16/21
17/18