Corrigé DS 2017
d'Algorithmique Avancée

Exercice 1

1.

	A	B	C	D	E	F	G	H
A	.	.	.	2	.	1	.	.
B	.	.	3	4	3	.	.	.
C	.	3	.	.	1	.	.	.
D	2	4	.	.	2	2	1	.
E	.	3	1	2	.	.	5	2
F	1	.	.	2	.	.	.	.
G	.	.	.	1	5	.	.	.
H	.	.	.	.	2	.	.	.

ou

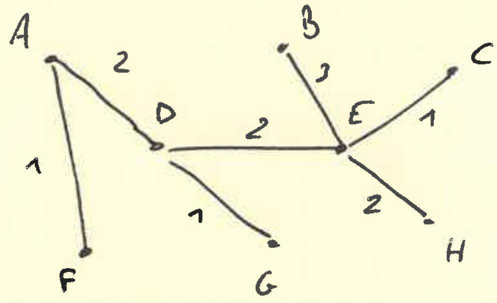
	A	B	C	D	E	F	G	H
A	.	.	.	1	.	1	.	.
B	.	.	1	1	1	.	.	.
C	.	1	.	.	1	.	.	.
D	1	1	.	.	1	1	1	.
E	.	1	1	1	.	.	1	1
F	1	.	.	1	.	.	.	.
G	.	.	.	1	1	.	.	.
H	.	.	.	.	1	.	.	.

les . représentants des 0.

2. On construit un ensemble de sommets découverts, en partant par exemple de A. A chaque étape, on considère toutes les arêtes entre les sommets découverts et les non-découverts, et on sélectionne celle de poids minimal.

Etape	Arêtes considérées	Arête ajouté et sommet découvert
0		A
1	(AD) (AF)	(AF), F
2	(AD) (FD)	(AD), D
3	(DB) (DE) (DG)	(DG), G
4	(DB) (DE) (GE)	(DE), E
5	(DB) (EB) (EC) (EH)	(EC), C
6	(DB) (EB) (CB) (EH)	(EH), H
7	(DB) (EB) (CB)	(EB), B

L'arbre construit est



qui est de poids 12

<u>Remarque</u> : Il y a non-unicité suivant le choix fait en cas d'ex-aequos.

Exercice 2

1. On reprend le déroulement de l'algorithme de Dijkstra tel que vu en cours : à chaque étape, on propose une distance pour les sommets qui peuvent être ajoutés à la structure existante, et on entoure celui qui est sélectionné, c'est-à-dire celui qui a la proposition la plus petite.

u	a	b	c	d	e	p1	p2	p3
0	8	5	∞	∞	∞	∞	∞	∞
0	8	5	11	∞	8	∞	∞	∞
0	8	5	11	∞	8	17	13	14
0	8	5	11	15	8	17	13	14
0	8	5	11	15	8	17	13	14
0	8	5	11	15	8	17	13	14
0	8	5	11	15	8	17	13	14
0	8	5	11					
0	8	5	11	15	8	17	13	14

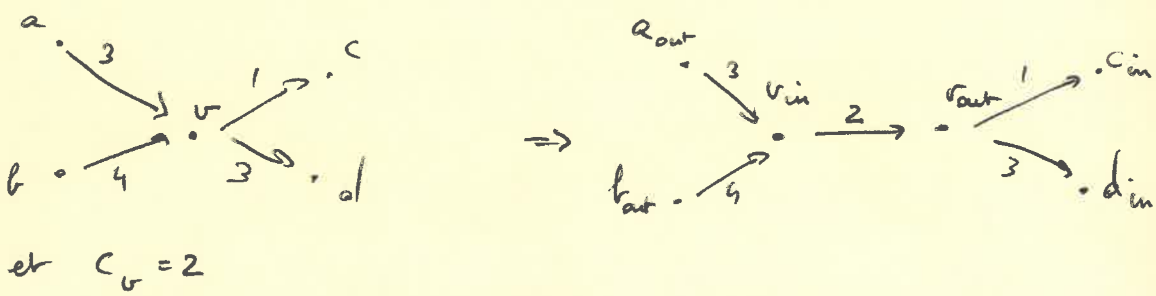
2. Deux solutions :

a) on modifie le code Dijkstra dans le sens suivant : quand un sommet v peut être ajouté à la solution courante depuis un sommet u, on calcule la proposition de coût pour l'ajout de v en prenant le coût de u auquel on ajoute le poids de l'arête uv et c_v.

b) on modifie l'instance en éclatant chaque sommet v et deux sommets v_{in} et v_{out} tel que

- v_{in} et v_{out} sont reliés par une arête de coût c_v
- les arêtes arrivant en v arrivent en v_{in}
- les arêtes partant de v partent de v_{out}

Ex :



A chaque trajet dans le graphe initial correspond un trajet dans le nouveau graphe traversant toutes les arêtes $v_{in} - v_{out}$ des sommets empruntés. On peut donc appliquer Dijkstra tel quel à ce nouveau graphe.

Exercice 3

1. Supposons que la racine est bleue.
Les sommets du premier niveau étant voisins de la racine, ils sont forcément tous rouges.
Les sommets du deuxième niveau étant tous voisins d'un sommet du premier niveau, ils sont forcément bleus.

On montre ainsi par récurrence que tous les sommets de niveau pair sont bleus et tous les sommets de niveau impair sont rouges (ou inversement si la racine est rouge).

2. On applique un DFS avec la modification suivante : Chaque fois qu'on inspecte un voisin déjà visité du sommet courant, on vérifie que la parité de son niveau est différente de la parité du niveau du sommet courant. Il suffit pour cela de stocker une variable avec le niveau et d'ajouter un test if dans le code.

Si le DFS parcourt tout le graphe sans découvrir d'arêtes entre deux voisins de même parité de niveau, il est 2-colorable, sinon il ne l'est pas.

La première affirmation est évidente puisqu'il suffit alors de colorier suivant la parité du niveau pour obtenir une 2-coloration valide. La deuxième est vraie car sinon on contredirait la question 1.

Au final, l'algorithme fait un nombre constant d'opérations supplémentaires par arête (calculer le niveau d'un sommet adjacent et/ou tester si les deux sommets adjacents sont de même parité de niveau).

Le coût reste donc en $\Theta(m)$, comme pour le DFS.

Exercice 4

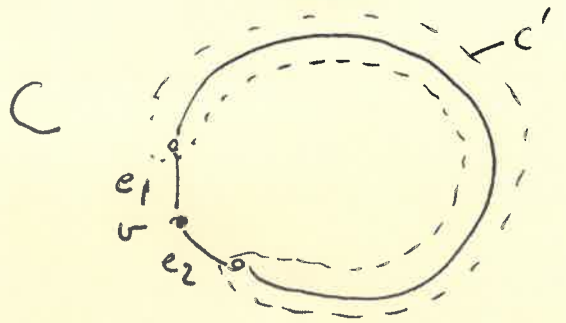
1. Soit G un graphe non orienté valué, de fonction de poids w. Il faut déterminer le parcours passant au moins une fois par chaque sommet qui est de poids minimal.

2. Pour construire H, il faut déterminer le plus court chemin entre toute paire de sommet. Il faut donc lancer Dijkstra sur chacun des sommets (on obtient ainsi le plus court chemin ~~du~~ sommet de départ vers tous les autres).

3. Soit C le trajet optimal. D'après les affirmations (a) et (b), C correspond à un cycle hamiltonien sur H. Il peut donc être découpé en

$$C = \{e_1, e_2\} \cup C'$$

où e_1 et e_2 sont les arêtes de C incidentes à v et d'un chemin passant par tous les sommets de H_v



C' est connexe, sans cycle et couvre H_v, c'est donc un arbre couvrant de H_v. Par conséquent,

$$w(C') \geq w(T_v)$$

De plus, $w(e_1) + w(e_2) \geq c(v)$ par définition de $c(v)$

donc

$$w(C) = w(e_1) + w(e_2) + w(C')$$

entraîne

$$w(C) \geq w(T_v) + c(v).$$

4. L'algorithme est le suivant.

- Pour tout v de G
  - Dijkstra (v)
  - Stockage des distances obtenues

  } construction de H

- Pour tout v de H
  - Calcul de $c(v)$
  - Prim sur H_v pour le calcul de T_v
  - Calcul de $w(T_v) + c(v)$

  } borne inférieure liée à v

- Prendre le maximum des bornes inférieures obtenues.

On obtient une complexité en

$$n \times (\Theta(m \log m) + n) + n \times (f + \Theta(m \log m) + cte) + \Theta(n)$$

où f est la complexité du calcul de $c(v)$.

Si on détermine $c(v)$ en essayant toutes les paires, $f = \Theta(n^2)$ et l'algorithme général est cubique.

Cependant, $c(v)$ est la somme du plus petit et du deuxième plus petit coefficient du vecteur des poids entre v et les autres sommets. Chercher le minimum, puis le supprimer et chercher le minimum suivant coûte

$$\Theta(n) + 1 + \Theta(n) = \Theta(n) \quad \text{opérations}$$

donc

$$f = \Theta(n).$$

La complexité totale est donc en

$$\Theta(n\, m \log m)$$