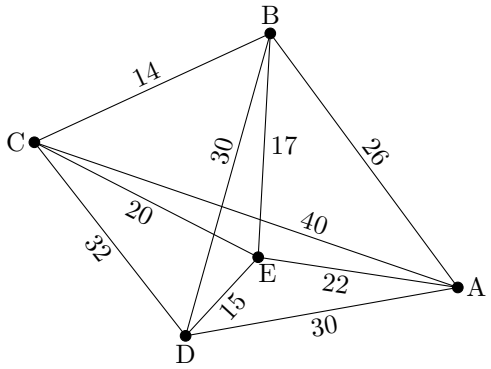
7 Algorithmes efficaces

En théorie des graphes, nous avons déjà vu ce qu'est un algorithme, à savoir la donnée d'une série d'instructions qui permettent, pas à pas, d'obtenir la solution d'un problème donné.

Dans la pratique, la découverte d'un algorithme pour résoudre un problème n'est pas suffisante. Il faut encore se poser la question de son efficacité. Pour mieux comprendre ce que cela veut dire, revenons au problème du voyageur de commerce et supposons que celui-ci, habitant la ville A, dispose de la carte routière suivante avec les distances entre les villes (il doit revenir en A).



7.1 La manière la plus évidente de résoudre ce problème est d'employer la « force brute » :

1) faire la liste de tous les itinéraires possibles ;
2) calculer la longueur de chacun d'entre eux ;
3) sélectionner celui qui est le plus court.

7.2

1) Combien y a-t-il d'itinéraires possibles dans l'exercice précédent ?
2) En supposant que le graphe soit complet, combien y a-t-il d'itinéraires possibles pour
   (a) 6 villes ?
   (b) 7 villes ?
   (c) n villes ?
3) Imaginons qu'un ordinateur très rapide puisse décrire un itinéraire et calculer sa longueur en 10 microsecondes. Combien de temps lui faudrait-il, avec 20 villes, pour déterminer le trajet le plus court en utilisant la méthode de la « force brute » ?

La technique de la « force brute » ou autrement dit de l'énumération exhaustive peut être très efficace sur de petits exemples, mais quand les données sont d'une certaine importance, elle devient complètement impraticable. À part le fait d'exiger qu'un algorithme ait chacune de ses étapes bien définies et fournisse la solution en un nombre fini d'étapes, il faut donc encore ajouter des considérations d'ordre pratique. Est-ce que le problème, pour des données

de taille modérée, peut être résolu par tel ou tel algorithme en un temps raisonnable ? Répondre à cette question implique de dénombrer chacune des opérations qui sont faites à chaque étape.

Par exemple, dans l'algorithme de Fleury, on peut vérifier que ce nombre est proportionnel à n où n est le nombre d'arêtes. Pour l'algorithme de connexion minimale, ce nombre est à peu près proportionnel à n^2 où n est le nombre de sommets.

Dans ces cas ou dans d'autres où le temps utilisé pour faire fonctionner l'algorithme est un polynôme en n, où n représente un certain paramètre associé à la quantité des données implémentées dans l'ordinateur, on parle d'algorithme **en temps polynomial**.

Il existe des algorithmes dont le temps de fonctionnement est de l'ordre d'une puissance de n. C'est le cas par exemple lorsqu'il faut examiner toutes les parties d'un ensemble à n éléments, puisqu'un tel ensemble a 2^n parties. On dit alors que ces algorithmes sont **en temps exponentiel** ; on les appelle aussi algorithmes **gloutons**.

7.3 Imaginons que l'on utilise un ordinateur capable d'effectuer 1000 opérations à la seconde. Compléter le tableau suivant qui donne les temps approximatifs nécessaires pour effectuer divers algorithmes, les uns en temps polynomial (d'ordre n et n^3), les autres en temps exponentiel (d'ordre 2^n et 3^n).

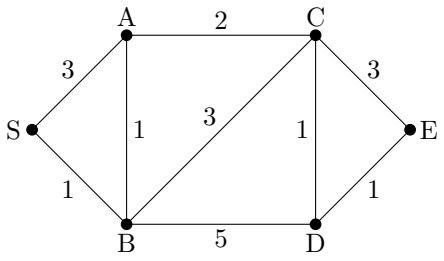
	$n = 10$	$n = 50$	$n = 100$
n			
n^3			
2^n			
3^n			

L'exercice 7.3 montre pourquoi l'on considère les algorithmes en temps polynomial comme efficaces, alors que les algorithmes en temps exponentiel sont considérés de peu d'utilité sauf dans les exemples de petite taille.

Algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme en temps polynomial (d'ordre n^2) qui permet de trouver quelle est la longueur du plus court chemin qui joint deux villes déterminées lorsqu'on connaît la longueur des routes.

Par exemple, considérons la situation suivante où les sommets représentent des villes, les arêtes des routes et les nombres des distances.



On cherche à déterminer le plus court chemin allant de S à E.

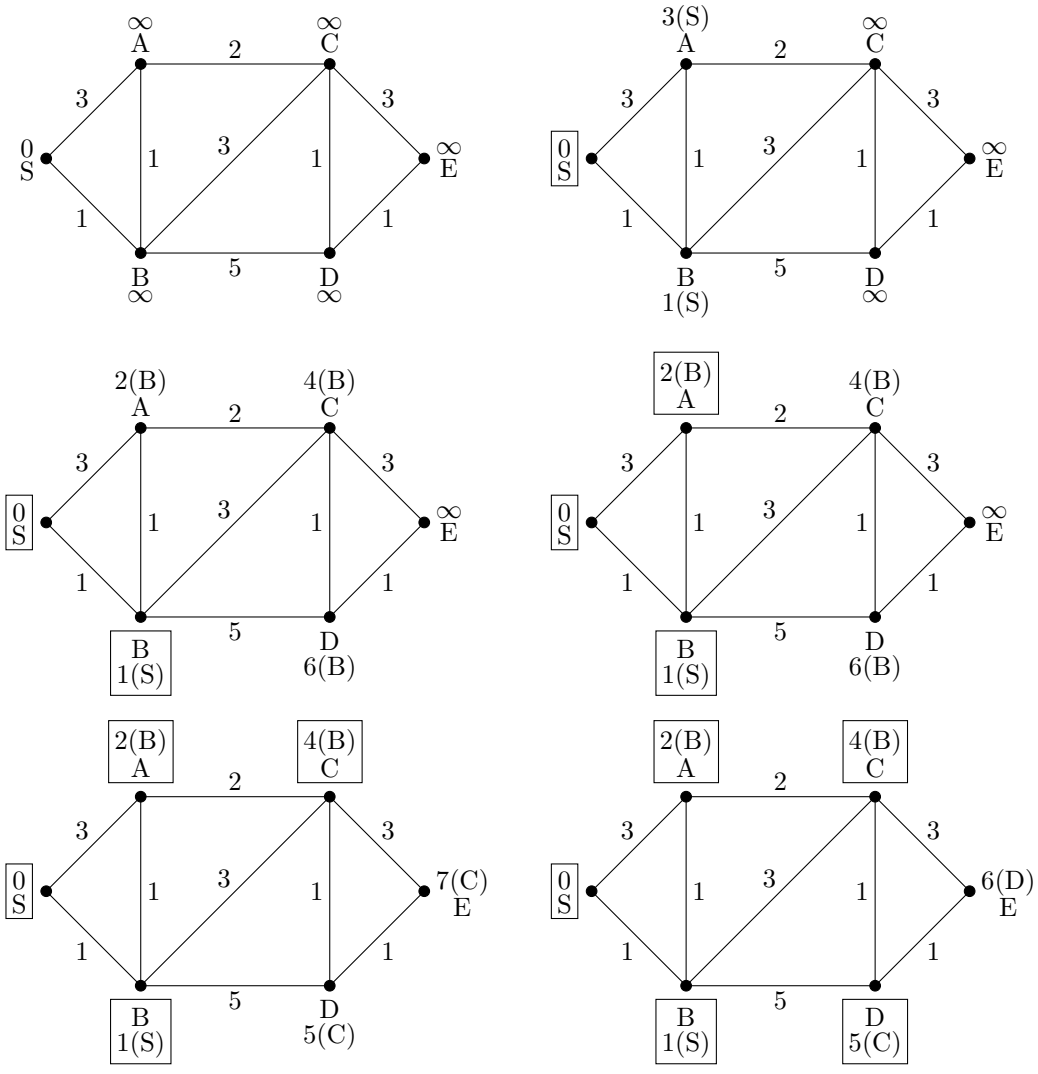
L'idée générale de l'algorithme de Dijkstra est de parcourir le graphe en allant de S à E en attribuant de proche en proche à chaque sommet un poids qui est égal à la plus petite des distances entre S et ce sommet.

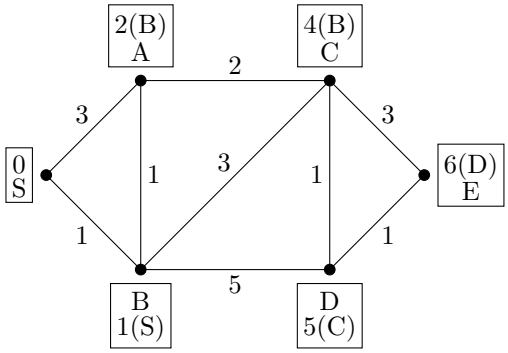
Plus précisément, on procède comme suit.

Initialisation Tous les sommets sont non marqués et ont un poids provisoire infini, sauf le sommet de départ qui a un poids nul.

Itérations Tant qu'il existe un sommet non marqué :
- choisir le sommet T qui possède le plus petit poids provisoire ;
- fixer définitivement le poids de T et marquer le sommet T ;
- pour chaque sommet U non marqué et voisin de T :
  - calculer la somme s du poids de T et du poids de l'arête reliant T à U ;
  - si s est inférieur au poids provisoire de U, affecter s à U comme nouveau poids provisoire et le noter s(T) pour indiquer ainsi la provenance de cette dernière affectation, sinon conserver le poids provisoire.

Voici l'application de cet algorithme au graphe précédent :



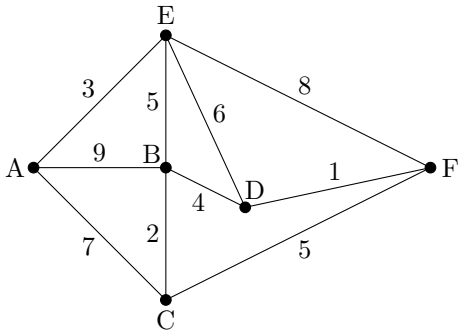


Le plus court chemin a un poids 6.
Il se lit à l'envers : EDCBS.

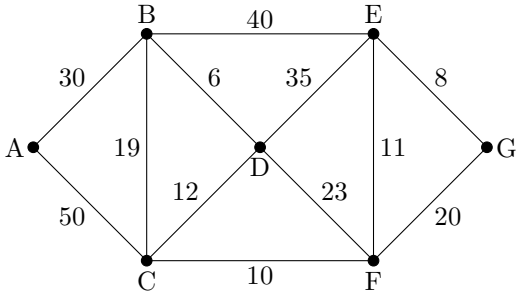
On peut aussi représenter les différentes étapes de l'algorithme exécutées sur cet exemple par un tableau :

S	A	B	C	D	E	sommets marqués
0						
0	3(S)	1(S)				S
	2(B)	1(S)	4(B)	6(B)		S, B
	2(B)		4(B)	6(B)		S, B, A
			4(B)	5(C)	7(C)	S, B, A, C
				5(C)	6(D)	S, B, A, C, D
					6(D)	S, B, A, C, D, E

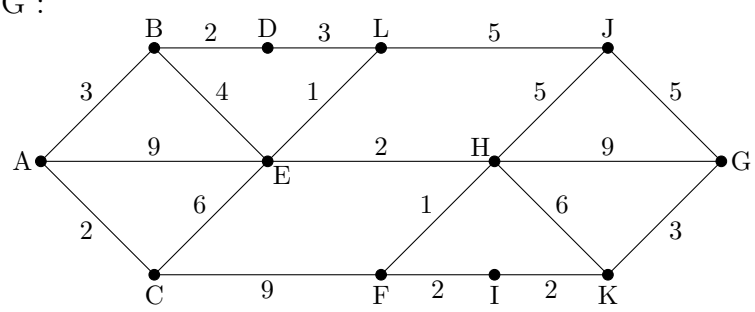
7.4 Utiliser l'algorithme de Dijkstra pour trouver le chemin de poids minimal entre A et F :



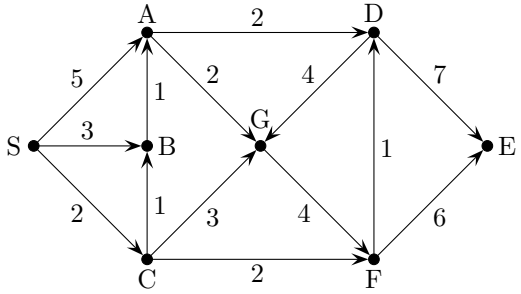
7.5 Utiliser l'algorithme de Dijkstra pour trouver le chemin de poids minimal entre A et G :



7.6 Utiliser l'algorithme de Dijkstra pour trouver le chemin de poids minimal entre A et G :



7.7 Utiliser l'algorithme de Dijkstra pour trouver le chemin de poids minimal entre S et E, en prenant garde au fait qu'il s'agit d'un graphe orienté.



Le problème du postier

Un postier cherche à distribuer ses lettres en parcourant la plus petite distance tout en retournant à son point de départ. Il doit passer évidemment par chaque rue au moins une fois en évitant autant que possible de repasser par un tronçon déjà parcouru.

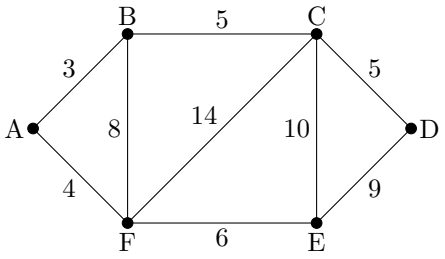
Il y a quelques années, la ville de Zürich commandita une grande étude pour déterminer rationnellement le plan de déneigement de ses rues. C'est, en plus compliqué, un problème semblable à celui du postier. En effet, il est nécessaire en plus de partager habilement la ville en secteurs de manière à occuper chaque véhicule de déneigement.

Le problème peut être reformulé en termes de graphe pondéré où le graphe correspond au réseau des rues et le poids de chaque arête à la longueur de la rue correspondante. Dans cette nouvelle formulation, l'exigence est de trouver un chemin fermé de poids minimal qui inclut chaque arête au moins une fois.

Si le graphe est eulérien, chaque chemin eulérien sera un itinéraire acceptable. Un tel chemin peut être obtenu si nécessaire à l'aide de l'algorithme de Fleury.

Par contre, si le graphe n'est pas eulérien, le problème est beaucoup plus difficile. On connaît cependant un algorithme efficace pour trouver la solution, même s'il est trop compliqué pour être donné ici. La solution générale s'inspire du cas particulier où le graphe est semi-eulérien.

Considérons par exemple le graphe suivant :

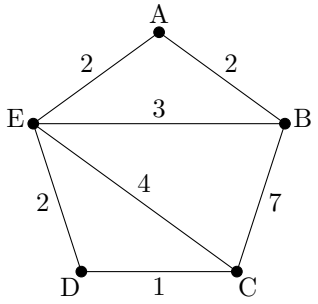


Puisque B et E sont les deux seuls sommets de degré impair, nous pouvons trouver un chemin semi-eulérien allant de B à E en parcourant chaque arête une seule fois.

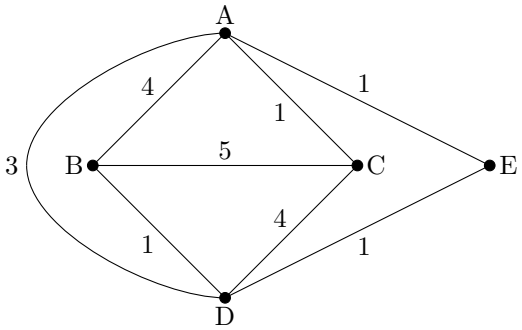
Pour retourner au point de départ en parcourant la plus petite distance possible, nous pouvons trouver le plus court chemin allant de B à E en utilisant l'algorithme de Dijkstra.

La solution du problème du postier est obtenue dans ce cas en joignant le plus court chemin EFAB au chemin semi-eulérien initial BAFBCFECDE. La distance totale vaut donc $13 + 64 = 77$.

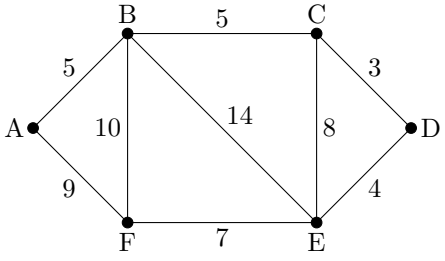
7.8 Résoudre le problème du postier pour le graphe pondéré ci-dessous :



7.9 Résoudre le problème du postier pour le graphe pondéré ci-dessous :



7.10 Résoudre le problème du postier pour le graphe pondéré ci-dessous :



Réponses

7.1

ABCDEA	109	ACBDEA	121	ADBCEA	116	AEBCDA	115
ABCEDA	105	ACBEDA	116	ADBECA	137	AEBDCA	141
ABDECA	131	ACDBEA	141	ADCBEA	115	AECBDA	116
ABDCEA	130	ACDEBA	130	ADCEBA	125	AECDBA	130
ABEDCA	130	ACEBDA	137	ADEBCA	116	AEDBCA	121
ABECDA	125	ACEDBA	131	ADECBA	105	AEDCBA	109

7.2 1) $4! = 24$ 2) $5! = 120$ $6! = 720$ $(n-1)!$ 3) 38 573 années

7.3

	$n = 10$	$n = 50$	$n = 100$
n	0,01 s	0,05 s	0,1 s
n^3	1 s	2 min	16 min
2^n	1 s	35 702 ans	$4 \cdot 10^{19}$ ans
3^n	1 min	$2{,}27 \cdot 10^{13}$ ans	$1{,}63 \cdot 10^{37}$ ans

7.4 AEDF de poids 10

7.5 ABDCFEG de poids 77

7.6 ABEHFIKG de poids 17

7.7 SCFE de poids 10

7.8 BAEDCBECDEB de poids 27

7.9 BACDAEDBCAEDB de poids 24

7.10 CDEFABCEBFEDC de poids 79