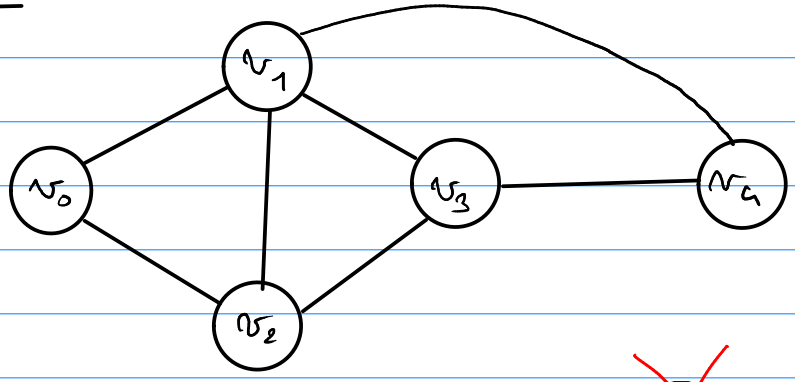
Chap IV - Graphes

1 - Définitions

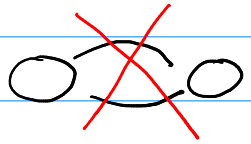
Def: Un graphe $G = (V, E)$ est donné par l'ensemble de ses
- sommets $V = \{v_0, \dots, v_{N-1}\}$
- arêtes $E = \{e_0, \dots, e_{K-1}\}$ où $e_i = [s_i, t_i] \in V^2$

On dit que $v, w \in V$ sont voisins si $[v, w] = [w, v] \in E$

Exemple:

$G =$ 

$E = \{[v_0, v_1], [v_0, v_2], [v_1, v_2], [v_1, v_3], [v_1, v_4], [v_2, v_3], [v_3, v_4]\}$

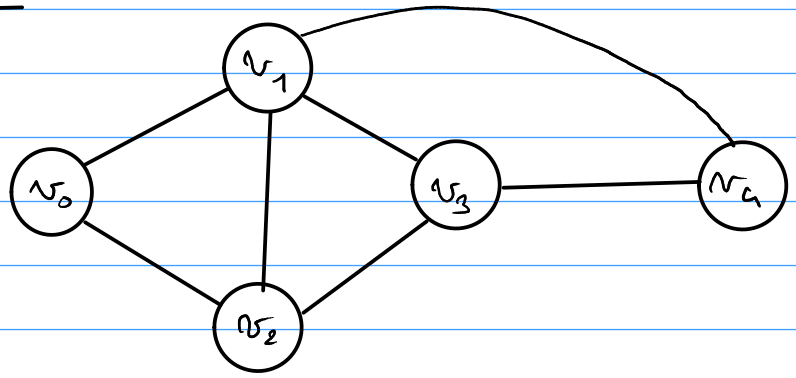
Ici les graphes sont simples  sans boucles  et non orienté $\bigcirc \not\to \bigcirc$

2. Matrices d'adjacences

Def: Soit $G = (V, E)$ un graphe à N sommets $v_0, \dots, v_{N-1}$.
La matrice d'adjacence de G est la matrice A telle que
- A est de taille $N \times N$
- $A_{i,j} = \begin{cases} 1 \text{ si } v_i \text{ et } v_j \text{ sont voisins } ([v_i, v_j] \in E \\ 0 \text{ si non} \end{cases}$

Exemple :

$G =$



$A =$

	v_0	v_1	v_2	v_3	v_4
v_0	0	1	1	0	0
v_1	1	0	1	1	1
v_2	1	1	0	1	0
v_3	0	1	1	0	1
v_4	0	1	0	1	0

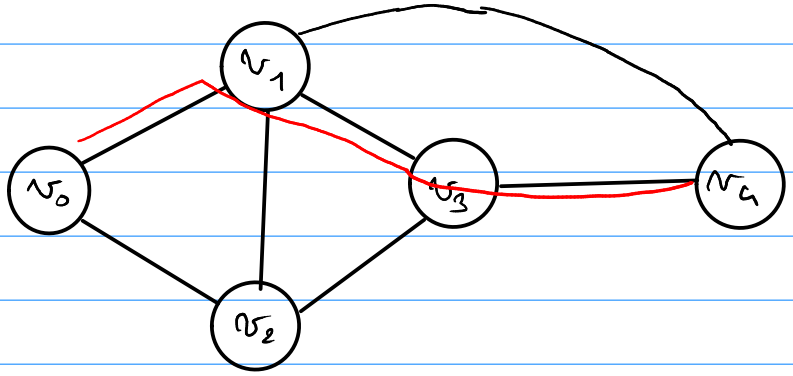
→ Représentation de graphe en Python

→ Il manque le nom des sommets, on a seulement leur ordre.

Proposition

- A est symétrique $[v_i, v_j] \in E \Leftrightarrow [v_j, v_i] \in E$
- la somme des coeffs de la ligne (colonne) i est le nombre d'arêtes attachées à v_i = nombre de voisins de v_i
- la somme des coeffs de A vaut $2K$
  ↳ faux si le graphe contient des boucles

Def Soit $G = (V, E)$ un graphe. Un chemin de longueur ℓ sur G est une suite $(e_1, \dots, e_\ell)$ d'arête de G telle que ces arêtes se suivent $\cdot \overset{e_1}{\text{—}} \cdot \overset{e_2}{\text{—}} \cdots \overset{e_\ell}{\text{—}} \cdot$



Chemin : $([v_0, v_1], [v_1, v_3], [v_3, v_4])$

: $v_0 — v_1 — v_3 — v_4$

→ de longueur 3

Notation Soit A une matrice carré de $\mathcal{M}_N(\mathbb{Z})$ et $\ell \in \mathbb{N}$.
On note $A_{i,j}(\ell)$ le coefficient (i,j) de A^ℓ

$$A_{i,j}(2) = \text{le coeff } (i,j) \text{ de } A^2 = \sum_{k=0}^{n-1} A_{i,k}(1) \times A_{k,j}(1)$$

Proposition Soit $G=(V,E)$ un graphe à N sommets et de matrice d'adjacence A. Pour tout $\ell \in \mathbb{N}$, le nombre de chemins dans G reliant v_i à v_j est $A_{i,j}(\ell)$.

Démonstration Notons $C_{i,j}(\ell) = \{\text{chemins de longueur } \ell \text{ reliant } v_i \text{ à } v_j\}$
$= \{ v_i \xrightarrow{\ell} v_j \}$

Montrons par récurrence sur ℓ :

H_ℓ : " $\forall i,j$ $\operatorname{card} C_{i,j}(\ell) = A_{i,j}(\ell)$ "

Init : Pour $\ell = 0$ on a

$$\operatorname{card} C_{i,j}(0) = \begin{cases} 0 & \text{si } i \neq j \\ 1 & \text{si } i = j \end{cases}$$

et alors $A^0 = I_n$ et donc $A_{i,j}(0) = \begin{cases} 0 & \text{si } i \neq j \\ 1 & \text{si } i = j \end{cases}$ donc H_0 vraie.

Hérédité : Supposons H_ℓ vraie pour $\ell \in \mathbb{N}$ et montrons $H_{\ell+1}$

Soit i et j dans $\{0, \dots, N-1\}$

Si $v_i \xrightarrow{\ell+1} v_j \in C_{i,j}(\ell+1)$ alors $v_i \xrightarrow{\ell} v_k \xrightarrow{1} v_j$

D'où $\operatorname{card} C_{i,j}(\ell+1) = \sum_{k=0}^{N-1} \operatorname{card} C_{i,k}(\ell) \times \operatorname{card} C_{k,j}(1)$

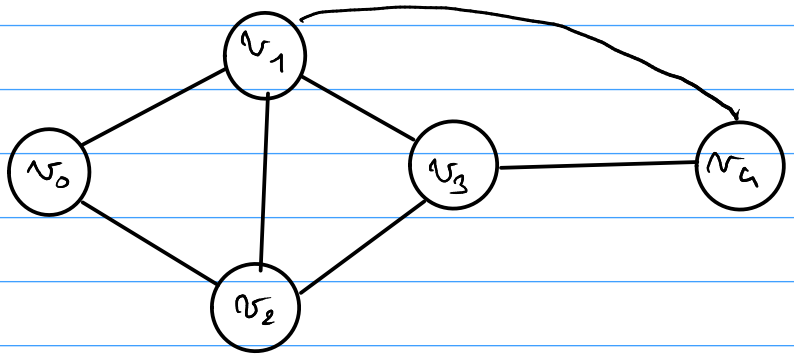
$$= \sum_{k=0}^{N-1} A_{i,k}(\ell) \times A_{k,j}(1) \quad \text{(H.R.)}$$

$$= (A^\ell \times A)_{i,j} = A_{i,j}(\ell+1)$$

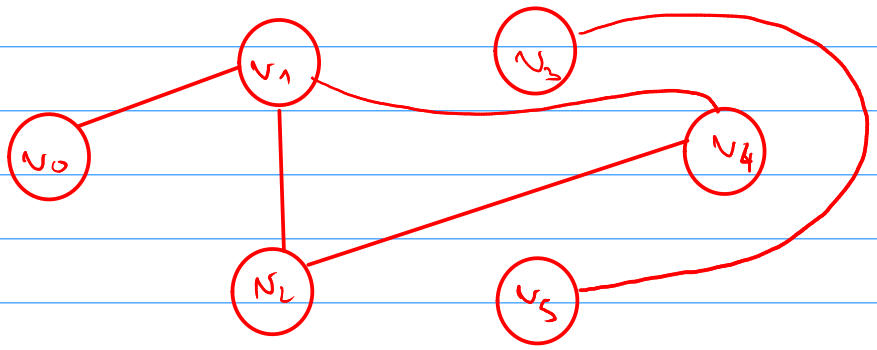
D'où $H_{\ell+1}$. □

3. Connexité

Def Un graphe $G=(V,E)$ est connexe si pour tout couple $(v,w) \in V^2$ il existe un chemin reliant v à w



← graphe connexe



← graphe pas connexe car pas de chemin de v_0 à v_5

Question Comment tester algorithmiquement qu'un graphe est connexe ?

→ En déterminant tous les sommets accessibles à partir d'un sommet donné

→ parcours d'un graphe

Il existe deux types de parcours

- le parcours en largeur (avec une file)
- le parcours en profondeur (avec une pile)

4. Parcours en largeur (avec une file)

Def Une file est une structure de donnée basée sur le principe premier entré, premier sorti (FIFO : first In, first out)

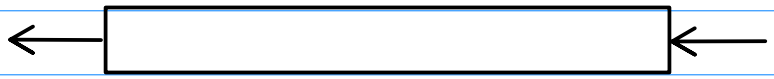
→ peut être représentée par un tableau

↝ Similaire à une file d'attente.

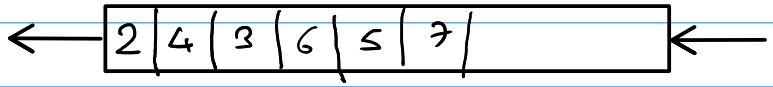
Les primitives d'une file sont :

- enfiler : ajouter un nouvel élément
- défiler : retirer un élément
- est_vide : tester si la file est vide

Exemple : File vide

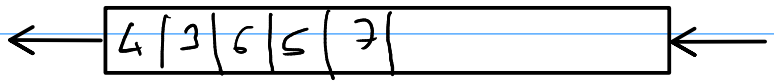


- On enfile les entiers 2, 4, 3, 6, 5 et 7

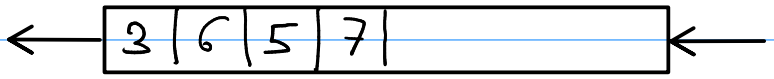


- On défile 3 fois

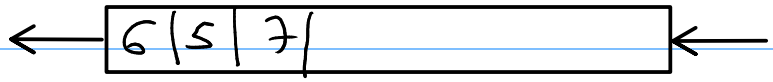
2



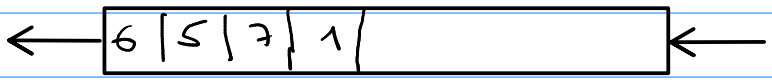
4



3



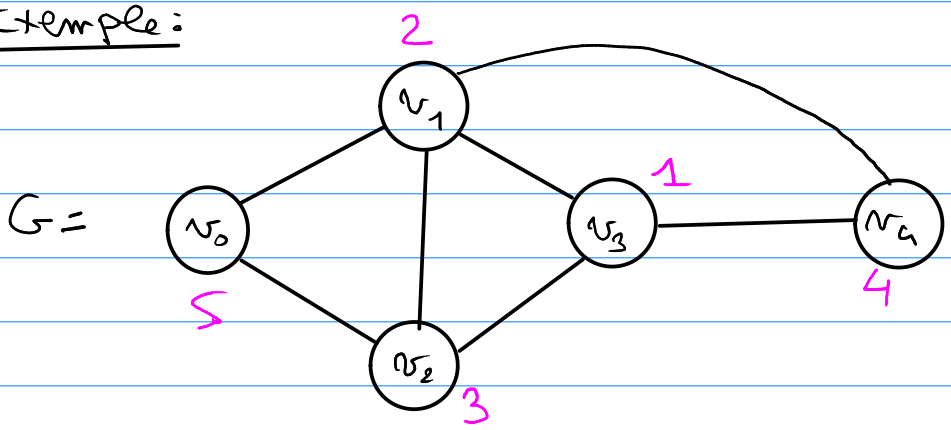
- On enfile 1



Principe parcours en largeur à partir d'un sommet $v \in V$ $(G = (V, E))$

1. F = file vide
2. F.enfile(v) enfile v à la file F
3. Tant que F est non vide :
   w = F.defile()
   Pour tout w' voisin de w faire :
   si w' non déjà visité faire :
   F.enfile(w')

Exemple :



Rq : On parcours d'abord les sommets à distance 1 de v_3 puis ceux à distance 2, etc.

Parcours en largeur à partir de v_3 .

1) F = ← | . | ←

2) F = ← | v_3 | ←

3) v_3 ← | . | ← On enfile les voisins de v_3 non déjà visités ← | v_1 | v_2 | v_4 . | ←

v_1 ← | v_2 | v_4 | . | ← Voisins de v_1 pas visités ← | v_2 | v_4 | v_0 . | ←

v_2 ← | v_4 | v_0 | . | ← Voisins de v_2 pas visités ← | v_4 | v_0 | . | ←

v_4 ← | v_0 | . | ← Voisins de v_4 pas visités ← | v_0 | . | ←

v_0 ← | . | ← Voisins de v_0 non visités ← | . | ←

Bilan : Un parcours en largeur de G à partir de v_3 est :

v_3, v_1, v_2, v_4 et v_0

5. Parcours en profondeur (à l'aide d'une pile)

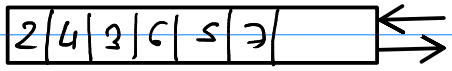
Déf : Une pile est une structure de donnée basée sur le principe dernier arrivé, premier sorti (LIFO : Last In, First Out)

Les primitives d'une pile sont :

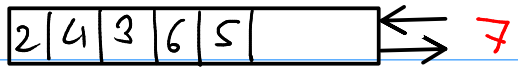
- empiler ajoute un élément
- dépiler retire un élément
- est-vide teste si la pile est vide

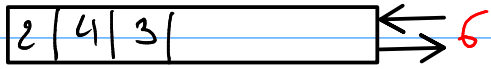
Exemple : Pile vide

On empile 2, 4, 3, 6, 5 et 7 :

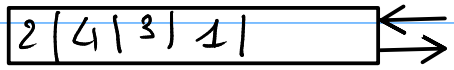


Dépiler 3 fois





On empile 1



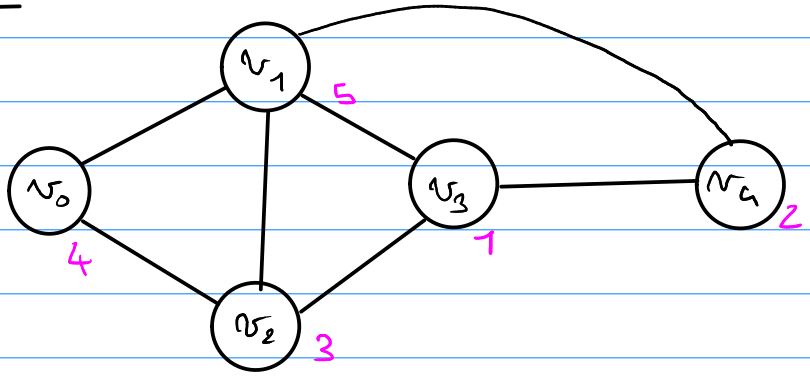
Dépile 1 fois

Principe parcours en profondeur à partir d'un sommet $v \in V$ $(G=(V,E))$

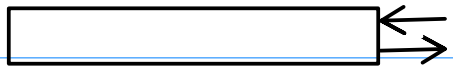
1. P = pile vide
2. P.empile(v)
3. Tant que P est non vide
   w = P.depile()
   Pour tout w' voisin de w faire :
   Si w' non déjà visité faire
   P.empile(w')

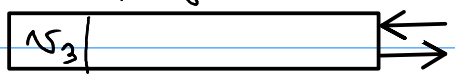
Exemple :

$G =$

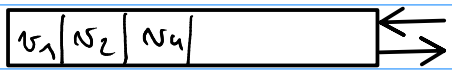


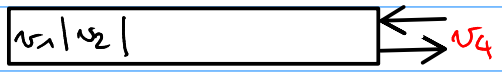
Le parcours en profondeur n'est pas "lié" à la distance

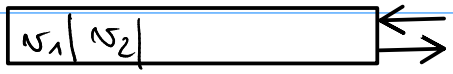
Parcours en profondeur à partir de v_3 : 1) P = 

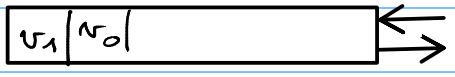
2) P = 

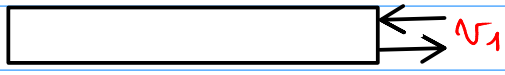
3)











Bilan : Un parcours en profondeur à partir de v_3 est

v_3, v_4, v_2, v_0 et v_1

6. Notation polonaise inversée d'expression arithmétique

Opérations binaires : +, ×, −, /

$a + b \longrightarrow a\ b\ +$

Notat° classique — notation polonaise inversée

Exemple Calculer 1 2 + 4 × 3 +

1 2 + 4 × 3 +

3 4 × 3 +

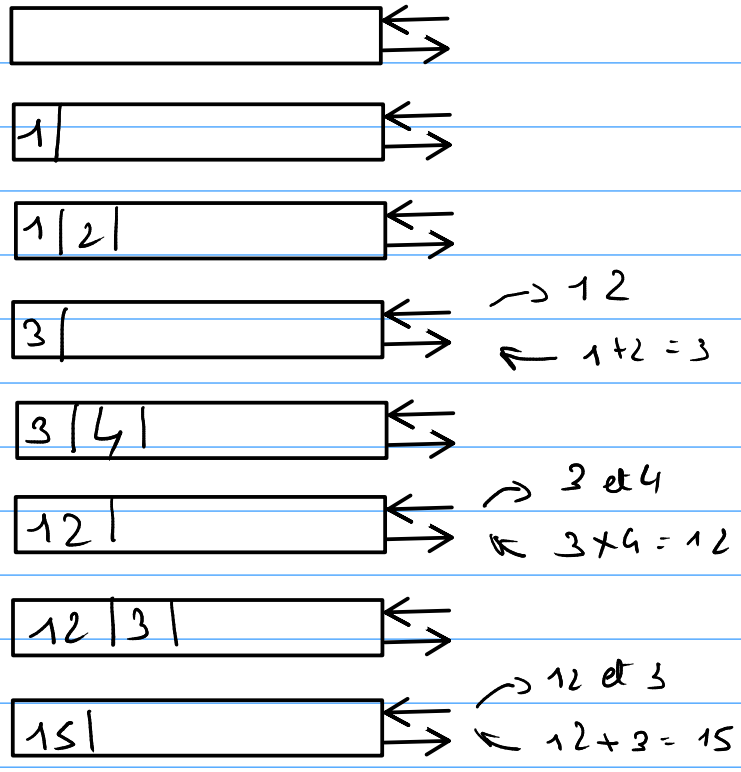
12 3 +

15

Principe algo évaluation expression en notation polonaise inversée

- Lire l'expression de la gauche vers la droite.
- On part d'une pile P vide
- Si on lit un entier on l'empile
- Si on lit une opération on dépile a, on dépile b on empile le résultat de l'opération

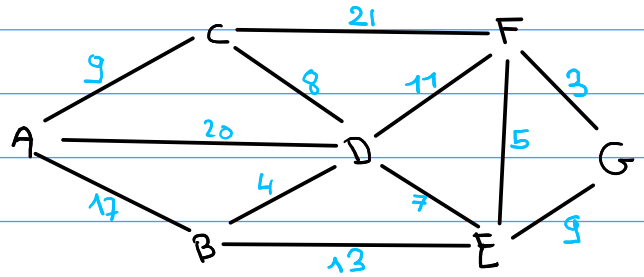
Exemple 1 2 + 4 × 3 +



7 - Dijkstra

Def : Un graphe $G = (V, E)$ est pondéré s'il est muni d'une application $\omega : E \longrightarrow \mathbb{N} \setminus \{0\}$ qui associe un poid $\omega(e)$ à chaque arête $e \in E$

Exemple : $V = \{A, B, C, D, E, F, G\}$



$E = \{[A, B], [A, C], \dots\}$

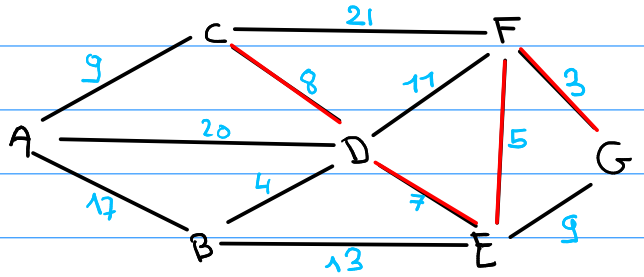
ω : $[A, B] \mapsto 17$

$[A, C] \mapsto 9$

⋮

Def : Soit $c = (e_1, \dots, e_\ell)$ chemin de longueur ℓ sur un graphe pondéré $G = (V, E, \omega)$. Le poid de c est $\omega(c) = \sum_{i=1}^{\ell} \omega(e_i)$

Exemple :



c : $C - D - E - F - G$

$\omega(c) = \omega([C, D]) + \omega([D, E]) + \omega([E, F]) + \omega([F, G])$

$= 8 + 7 + 5 + 3 = 23$

Problème : Soit deux sommet s et t d'un graphe pondéré $G = (V, E, \omega)$.

Déterminer, lorsqu'il existe, le chemin de poid minimal allant de s à t dans G.

Application : Instruments de navigation type GPS

G : graphe du réseau routier

V : points d'intersection

E : portion de route

ω : longueur en kilomètre / durée de parcours des portions

→ Application aux routages réseaux

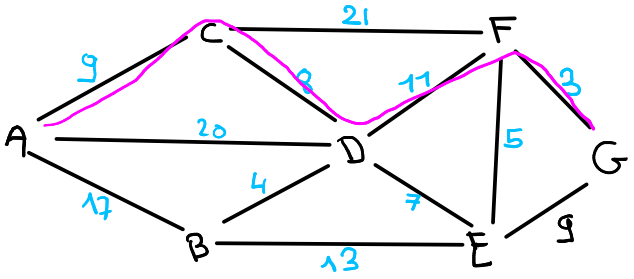
Algorithme de Dijkstra pour déterminer un chemin de poids minimal entre deux sommets d'un graphe pondéré $G=(V, E, \omega)$

Le principe repose sur le parcours de G à l'aide d'une file de priorités.

Def: Une file de priorité est une structure de donnée sur laquelle on peut effectuer les opérations suivantes :

- insérer un élément (enfile)
- tester si la file est vide (est_vide)
- extraire l'élément de poid minimal (ou maximal) (défile)

Exemple: Rechercher un chemin entre A et G de poids minimal



A	B	C	D	E	F	G
0 A	∞	∞	∞	∞	∞	∞
	17 A	9 A	20 A	∞	∞	∞
	17 A		17 C	∞	30 C	∞
			17 C	30 B	30 C	∞
				24 D	28 D	∞
					28 D	33 E
						31 F

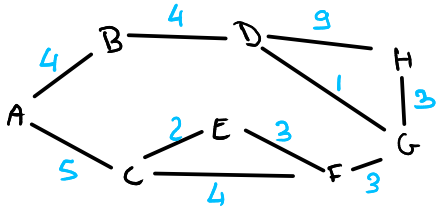
- Parmi les sommets non visités, on considère celui de poid minimal.
- On met à jour ses voisins si on trouve mieux, en précisant d'où on vient
- On stop l'algorithme lorsqu'on considère le sommet d'arrivé

Bilan Le coût minimal pour aller de A à G est 31 et est réalisé par

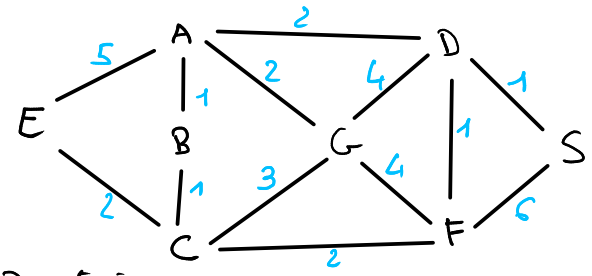
G — F — D — C — A (lecture à l'envers)

A —9— C —8— D —20— F —3— G

Exercices

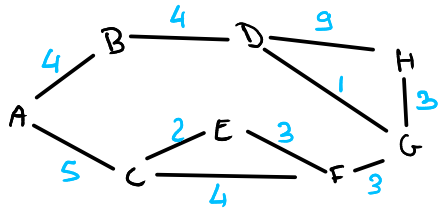


De A à G ABDG



De E à S EAFS

Correction

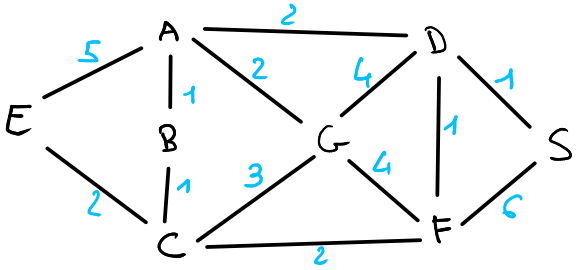


Chemin de A à G

A	B	C	D	E	F	G	H
0A	∞	∞	∞	∞	∞	∞	∞
	4A	5A	∞	∞	∞	∞	∞
		5A	8B	∞	∞	∞	∞
			8B	7C	9C	∞	∞
			8B		9C	∞	∞
					9C	9D	17D
						9D	17D

Bilan : G —1— D —4— B —4— A

⇝ A — B — D — G de coût 9



Chemin de E à S

E	A	B	C	D	F	G	S
0E	∞	∞	∞	∞	∞	∞	∞
	5E	∞	2E	∞	∞	∞	∞
	5E	3C		∞	4C	5C	∞
	4B			∞	4C	5C	∞
				6A	4C	5C	∞
				5F		5C	10F
						5C	6D
							6D

Bilan :

S —1— D —1— F —2— C —2— E

⇝ E — C — F — D — S de coût 6