Structures de données	Chap 04	Algorithmique des graphes

4.1 Parcours en largeur

Principe du parcours en largeur

On souhaite explorer un graphe à partir d'un sommet de départ, en visitant les sommets **par cercles successifs** autour de ce point de départ.
Pour organiser cette exploration, on utilise :

- une **file** : les sommets sont traités dans l'ordre où ils sont découverts ;
- un ensemble de **sommets marqués** : dès qu'un sommet est découvert, on le marque pour éviter de le visiter plusieurs fois.

À chaque étape :

1. on retire le premier sommet de la file ;
2. on marque tous ses voisins ;
3. chaque voisin encore non marqué est marqué puis ajouté à la fin de la file.

Algorithme — Parcours en largeur d'un graphe

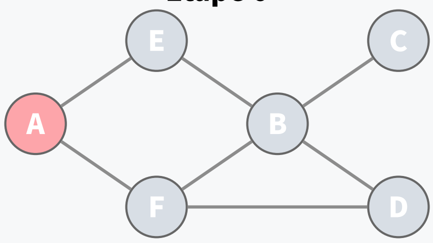
Input : Un graphe G donné par une liste de voisins, un sommet de départ s
Output : Les sommets de G visités dans l'ordre du parcours
Créer une file vide F
Créer un ensemble vide M (sommets marqués)
Ajouter s à la file F
Ajouter s à l'ensemble M

Tant que la file F n'est pas vide :
    Retirer le premier sommet u de la file F
    Traiter le sommet u (effectuer ce qu'on souhaite sur u)
    Pour chaque voisin v de u :
        Si $v \notin M$ alors
            Ajouter v à l'ensemble M
            Ajouter v à la fin de la file F

4.1.1 Exemple sur le graphe de la Terre du Milieu

Schéma — Le parcours en largeur depuis le sommet A sur le graphe de la Terre du Milieu

Étape 0

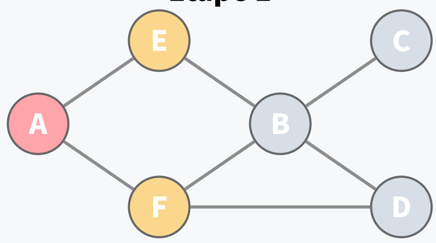


File : [A]

Marqués : {A}

Traité : —

Étape 1

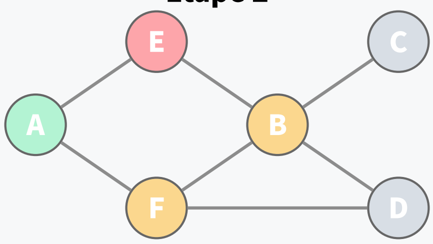


File : [E, F]

Marqués : {A,E,F}

Traité : A

Étape 2

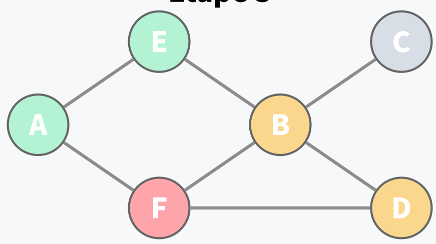


File : [F, B]

Marqués : {A,E,F,B}

Traité : E

Étape 3

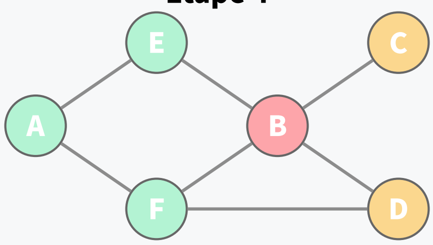


File : [B, D]

Marqués : {A,E,F,B,D}

Traité : F

Étape 4

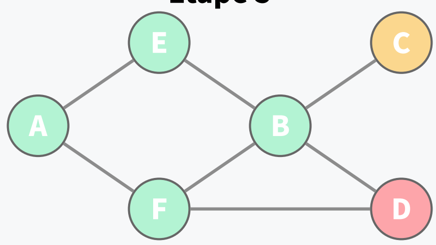


File : [D, C]

Marqués : {A,E,F,B,D,C}

Traité : B

Étape 5

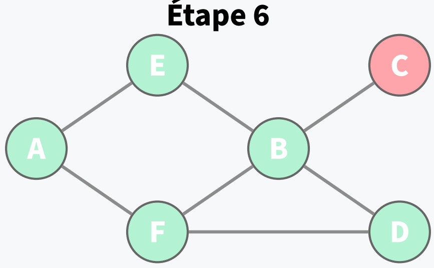


File : [C]

Marqués : {A,E,F,B,D,C}

Traité : D

Étape 6



File : [ ]

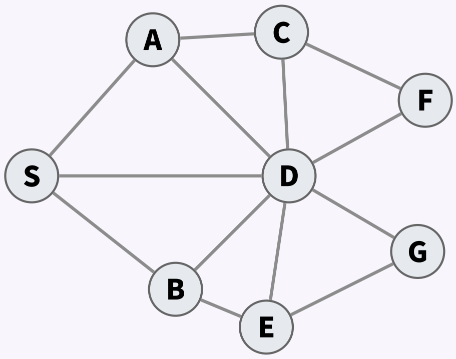
Marqués : {A,E,F,B,D,C}

Traité : C

4.1.2 Exercice : Parcours en largeur avec tableau à compléter

Exercice 1 — Parcours en largeur à la main : tableau de suivi

On considère le graphe non orienté suivant. On lance un parcours en largeur depuis le sommet S.



Convention : quand on parcourt les voisins d'un sommet, on les considère dans l'ordre alphabétique.

1) Compléter la table de suivi du parcours en largeur ci-dessous.

Étape	Sommet retiré	Nouveaux sommets ajoutés	File après ajout	Marqués
0	—-		[S]	{S}
1				
2				
3				
4				
5				
6				
7				
8				
9				

4.2 Parcours en profondeur

Principe du parcours en profondeur

On souhaite explorer un graphe à partir d'un sommet de départ, en visitant les sommets en allant **le plus loin possible** avant de revenir en arrière.

Pour organiser cette exploration, on utilise :

- une **pile** : le dernier sommet découvert est traité en priorité ;
- un ensemble de **sommets marqués** : dès qu'un sommet est découvert, on le marque pour éviter les visites multiples.

Le principe est le suivant :

1. on empile le sommet de départ ;
2. tant que la pile n'est pas vide :
   - on dépile un sommet ;
   - on explore immédiatement l'un de ses voisins non marqués ;
   - si un voisin non marqué existe, on l'empile et on continue depuis lui ;
   - sinon, on revient en arrière.

Ce parcours correspond naturellement à une exploration **récursive**.

Algorithme — Parcours en profondeur d'un graphe

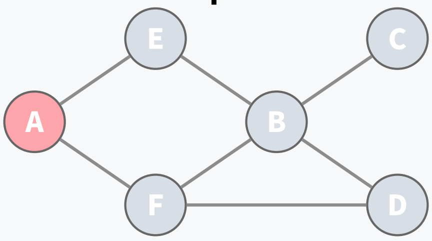
Input : Un graphe G donné par une liste de voisins, un sommet de départ s
Output : Les sommets de G visités dans l'ordre du parcours
Créer une pile vide P
Créer un ensemble vide M (sommets marqués)
Empiler s dans la pile P
Ajouter s à l'ensemble M

Tant que la pile P n'est pas vide :
    Dépiler le sommet u de la pile P
    Traiter le sommet u
    Pour chaque voisin v de u (dans l'ordre choisi) :
        Si $v \notin M$ alors
            Ajouter v à l'ensemble M
            Empiler v dans la pile P
            Quitter la boucle des voisins

4.2.1 Exemple sur le graphe de la Terre du Milieu

Schéma — Le parcours en profondeur depuis le sommet A sur le graphe TdM

Étape 0

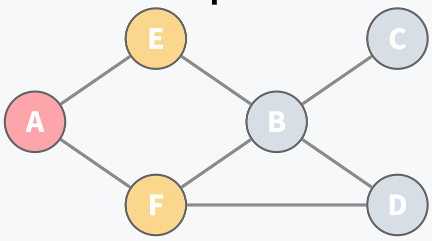


Pile : [A]

Marqués : {A}

Traité : —

Étape 1

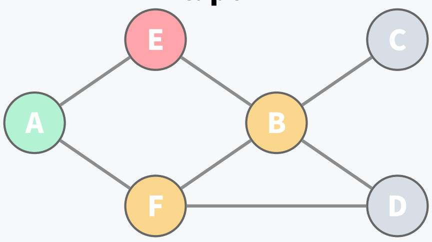


Pile : [E, F]

Marqués : {A,E,F}

Traité : A

Étape 2

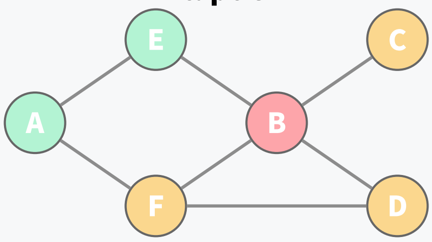


Pile : [B, F]

Marqués : {A,E,F,B}

Traité : E

Étape 3

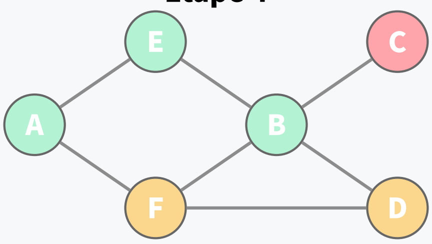


Pile : [C, D, F]

Marqués : {A,E,F,B,C,D}

Traité : B

Étape 4

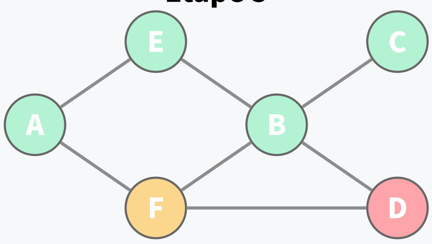


Pile : [D, F]

Marqués : {A,E,F,B,C,D}

Traité : C

Étape 5

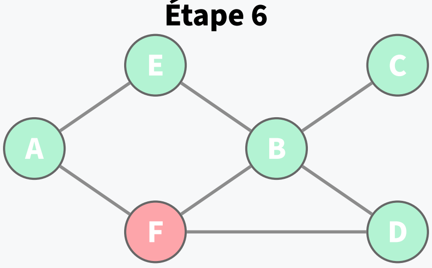


Pile : [F]

Marqués : {A,E,F,B,C,D}

Traité : D

Étape 6



Pile : [ ]

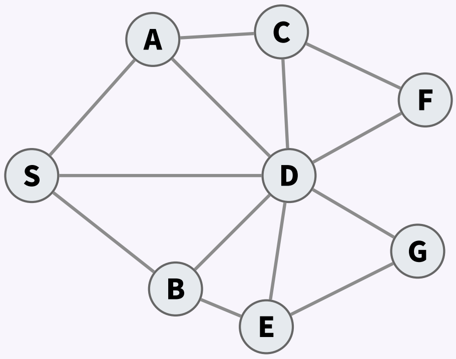
Marqués : {A,E,F,B,C,D}

Traité : F

4.2.2 Exercice : Parcours en profondeur avec tableau à compléter

Exercice 2 — Parcours en profondeur à la main

On considère le même graphe que précédemment.On lance un parcours en profondeur depuis le sommet S.



Convention : les voisins sont explorés dans l'ordre alphabétique.

1) Compléter la table de suivi du parcours en profondeur.

Étape	Sommet dépilé	Sommets empilés	Pile après empilement	Marqués
0	—-		[S]	{S}
1				
2				
3				
4				
5				
6				
7				
8				

4.3 Algorithme de Dijkstra

Principe de l'algorithme de Dijkstra

Dans un **graphe pondéré**, chaque arête possède un **poids** (distance, coût, durée, etc.).
L'objectif de l'algorithme de Dijkstra est de calculer :

les plus courtes distances depuis un sommet de départ

et, grâce à un tableau de **prédécesseurs**, de pouvoir **retracer un plus court chemin**.

L'algorithme manipule deux informations :

— $dist[v]$: meilleure distance connue (provisoire) de s vers v ;

— $pred[v]$: le **sommet précédent** sur le meilleur chemin connu vers v.

À chaque étape :

1. on choisit le sommet non visité avec la plus petite distance $dist$;
2. on « fixe » sa distance (elle devient définitive) ;
3. on tente d'améliorer les distances de ses voisins (c'est la **relaxation**) : si on trouve un chemin plus court vers un voisin v, on met à jour

$$dist[v] \leftarrow dist[u] + poids(u, v) \quad \text{et} \quad pred[v] \leftarrow u.$$

Attention

Dijkstra fonctionne si tous les poids sont **positifs ou nuls**. Avec des poids négatifs, l'algorithme peut se tromper.

Algorithme — Dijkstra

Input : Un graphe pondéré g (liste d'adjacence), un sommet de départ s
Output : $dist$ (distances minimales), $pred$ (prédécesseurs pour reconstruire les chemins)

$dist \leftarrow$ dictionnaire des distances (initialisées à $+\infty$)
$pred \leftarrow$ dictionnaire des prédécesseurs (initialisés à $\varnothing$)
$dist[s] \leftarrow 0$
$sommets\_visites \leftarrow [\,]$

Tant que tous les sommets n'ont pas été visités :
  $sommet\_courant \leftarrow None$
  $dist\_min \leftarrow +\infty$

  Pour tous les sommets $sommet$ de g :
    Si $sommet$ n'est pas visité **et** $dist[sommet] < dist\_min$ alors
      $sommet\_courant \leftarrow sommet$
      $dist\_min \leftarrow dist[sommet]$

  Pour tous les sommets adjacents $voisin$ de $g[sommet\_courant]$:
    $nouvelle\_dist \leftarrow dist[sommet\_courant] + g[sommet\_courant][voisin]$
    Si $nouvelle\_dist < dist[voisin]$ alors
      $dist[voisin] \leftarrow nouvelle\_dist$
      $pred[voisin] \leftarrow sommet\_courant$

  Ajouter $sommet\_courant$ à $sommets\_visites$

Renvoyer $dist$ et $pred$

Reconstruire un chemin

Pour retrouver un plus court chemin de s vers un sommet t :

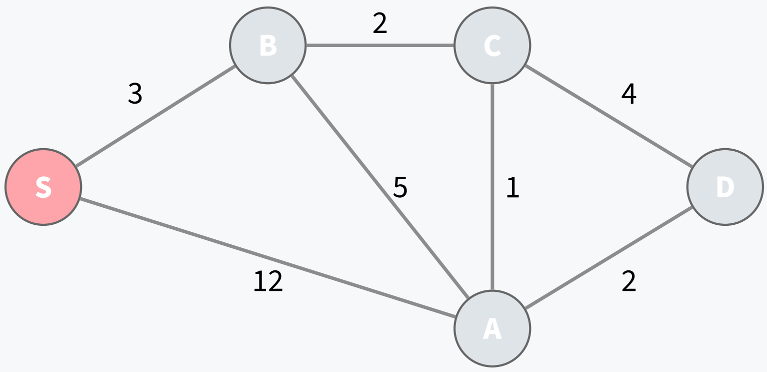
- on part de t ;
- on remonte grâce à $pred[t]$, puis $pred[pred[t]]$, etc. ;
- on s'arrête quand on arrive à s (ou quand on rencontre $\varnothing$ si t est inaccessible).

Le chemin obtenu est à l'envers : on l'inverse pour obtenir le chemin de s vers t.

4.3.1 Exemple détaillé

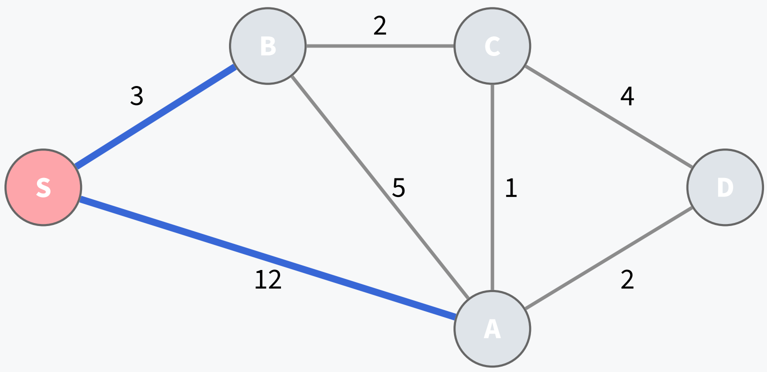
Schéma — Dijkstra : pas à pas

Étape 0



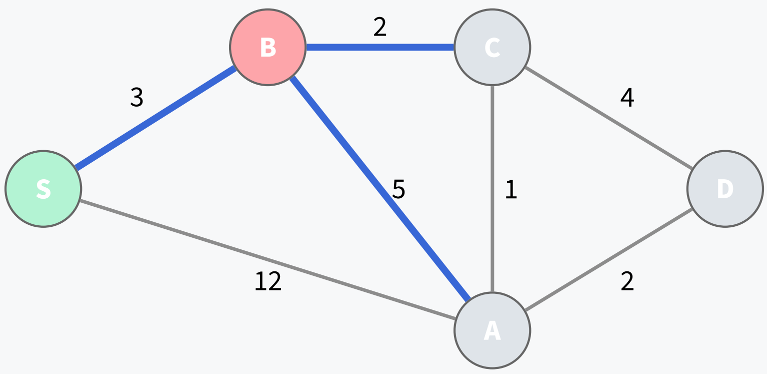
Étape	**Traité**	$d(S)$	$pred$	$d(B)$	$pred$	$d(C)$	$pred$	$d(A)$	$pred$	$d(D)$	$pred$
0	—	0	$\varnothing$	∞	$\varnothing$	∞	$\varnothing$	∞	$\varnothing$	∞	$\varnothing$

Étape 1



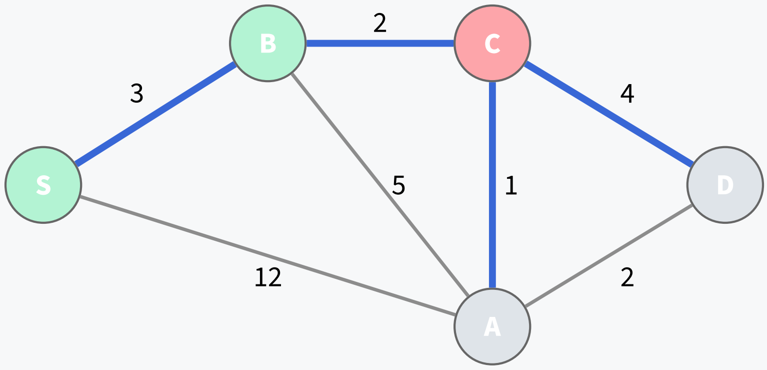
Étape	**Traité**	$d(S)$	$pred$	$d(B)$	$pred$	$d(C)$	$pred$	$d(A)$	$pred$	$d(D)$	$pred$
1	S	0	$\varnothing$	3	S	∞	$\varnothing$	12	S	∞	$\varnothing$

Étape 2



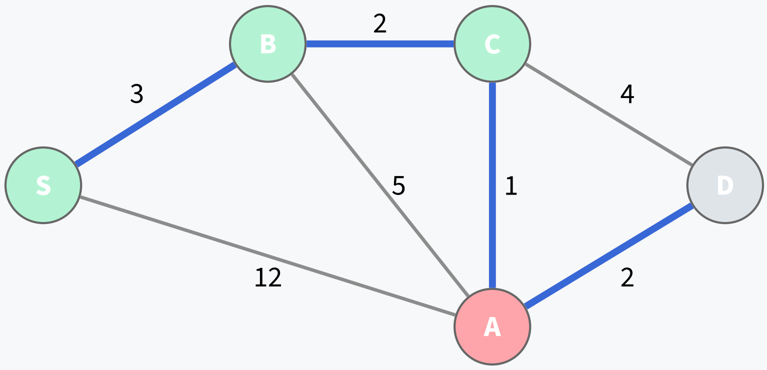
Étape	**Traité**	$d(S)$	$pred$	$d(B)$	$pred$	$d(C)$	$pred$	$d(A)$	$pred$	$d(D)$	$pred$
2	B	0	$\varnothing$	3	S	5	B	**8**	B	∞	$\varnothing$

Étape 3



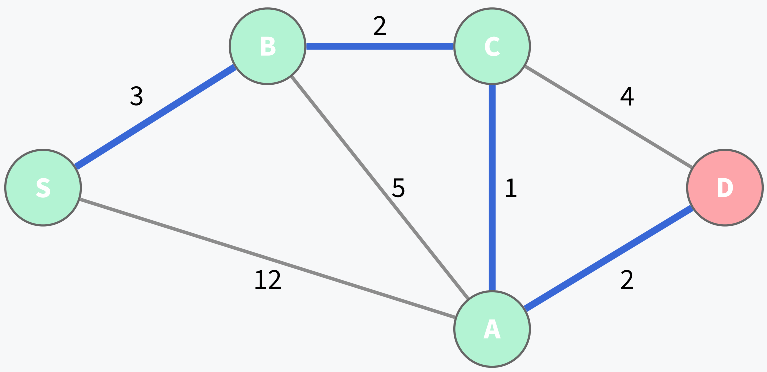
Étape	**Traité**	$d(S)$	$pred$	$d(B)$	$pred$	$d(C)$	$pred$	$d(A)$	$pred$	$d(D)$	$pred$
3	C	0	$\varnothing$	3	S	5	B	**6**	C	9	C

Étape 4



Étape	**Traité**	$d(S)$	$pred$	$d(B)$	$pred$	$d(C)$	$pred$	$d(A)$	$pred$	$d(D)$	$pred$
4	A	0	$\varnothing$	3	S	5	B	6	C	**8**	A

Étape 5



Étape	**Traité**	$d(S)$	$pred$	$d(B)$	$pred$	$d(C)$	$pred$	$d(A)$	$pred$	$d(D)$	$pred$
5	D	0	$\varnothing$	3	S	5	B	6	C	8	A

Retrouver le plus court chemin de A vers S

À la fin de l'étape 3, on a :

$$dist(A) = 6 \quad \text{et} \quad pred(A) = C,\ pred(C) = B,\ pred(B) = S.$$

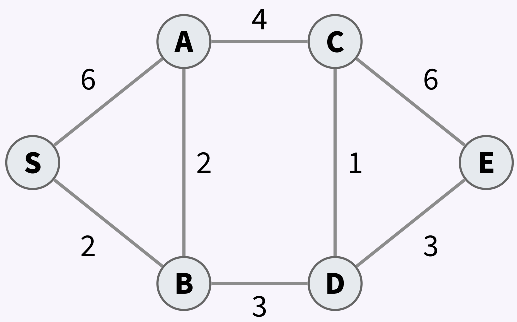
Donc un plus court chemin de S vers A est :

$$S \to B \to C \to A.$$

4.3.2 Application

Exercice 3 — Dijkstra à la main

On considère le graphe pondéré ci-dessous. On applique Dijkstra depuis S.



1) Compléter la table ci-dessous (distances et prédécesseurs).

Étape	**Traité**	$d(S)$	$pred$	$d(B)$	$pred$	$d(C)$	$pred$	$d(A)$	$pred$	$d(D)$	$pred$
0	—	0	$\varnothing$	∞	$\varnothing$	∞	$\varnothing$	∞	$\varnothing$	∞	$\varnothing$
1											
2											
3											
4											
5											

2) Donner un plus court chemin de S vers E
