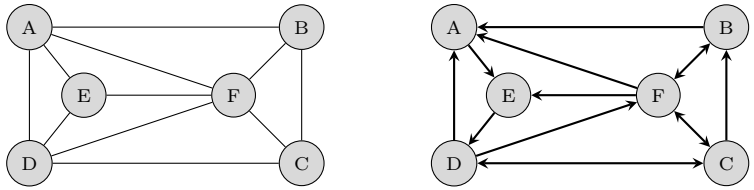
Informatique - TD 7 | Algorithme de Dijkstra

Le but de ce TD est d'étudier l'agorithme de Dijkstra qui permet de trouver le chemin le plus court entre deux points dans un graphe de chemins pondérés. Les graphes sont des objets très utilisés en informatique (ainsi qu'en mathématiques) afin d'étudier des domaines très variés (réseau informatique, relations dans les réseaux sociaux, arbres de probabilités, etc. ...)

Définition et vocabulaire des graphes :

Un graphe est donné par un ensemble de sommets (ou nœuds) et un ensemble d'arrêtes (ou arcs) qui relient les sommets comme sur les deux exemples ci-dessous :

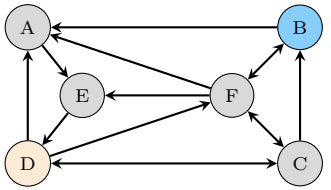


Ici, les sommets sont les points $A, B, \dots, F$. Attention à la différence entre ces deux graphes :

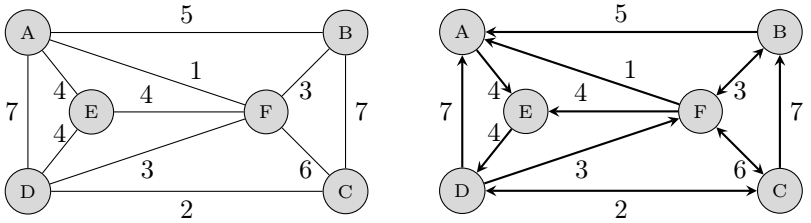
- À gauche, les arrêtes ne sont pas orientées. Par exemple, on peut aller indifféremment de A à B ou de B à A. *On pourrait imaginer que ce graphe schématise des personnes $A, B, \dots, F$ reliées ou non sur les réseaux sociaux.*

- À droite, on attribue un sens aux arrêtes. Par exemple, on peut aller de B à A, mais pas de A à B, alors que l'on peut aller de D à C et réciproquement. Dans ce cas, le graphe est dit *orienté*. *On pourrait imaginer que les sommets désignent là aussi des personnes $A, B, \dots, F$ présentes sur un même réseau. Les flèches orientées schématiseraient le fait d'avoir fait une demande d'ami. Par exemple, F a fait un demande à A, mais A n'a pas répondu (ou a refusé), tandis que F et C sont amis et communiquent entre eux.*

Deux sommets sont dits *adjacents* s'ils sont reliés par une arrête. Une *chaîne* est une suite de sommets reliés par des arrêtes (où on se préoccupe du sens des flèches si le graphe est orienté.) Le début de la chaîne s'appelle *point d'entrée* et le sommet final s'appelle *point de sortie*. Par exemple, sur le graphe ci-dessous, si on désigne B le point d'entrée, la chaîne $BFED$ est une chaîne de point de sortie D. En revanche, $BAFD$ ne peut être une chaîne car on ne peut pas aller directement de A à F.

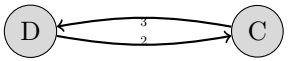


On peut également mettre des *poids* sur les arrêtes comme sur les graphes ci-dessous. On dit alors que c'est un graphe *pondéré*. *Dans ce cas, on peut par exemple imaginer que les sommets désignent des villes et que les arrêtes désignent des routes (éventuellement à sens unique si le graphe est orienté) de poids le poids attribué à l'arrête.*



Dans ce cas, on appelle *poids d'une chaîne* la somme totale des poids des arrêtes qui le composent. Par exemple, la chaîne $BFED$ est de poids $3 + 4 + 4 = 11$.

Si les routes sont à sens unique, il ne serait pas non plus choquant d'avoir des flèches "aller-retour" n'ayant pas le même poids, comme dans l'exemple ci-dessous, ou des boucles qui relient un point à lui même, mais ces cas sont exlcus de notre étude actuelle. On dit que le graphe est *simple*.



Pour décrire l'ensemble des poids des chemins, on utilise couramment soit les matrices appelées *matrice d'adjacence*, soit des dictionnaires.
Concernant les matrices d'adjacences (notée ici Adj) , on rappelle qu'elles sont construites de la manière suivante :
Adj est une matrice carrée de taille $n \times n$ où n est appelé *ordre* du graphe est désigne le nombre de sommets. Ensuite, on a

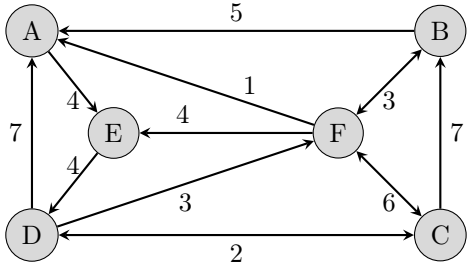
$$Adj = (a_{i,j})_{1 \leqslant i,j \leqslant n}$$

où $a_{i,j}$ est le poids de **l'arrête** allant du sommet numéro i jusqu'au point numéro j. Sur notre graphe (**??**), si on numérote les points $A, B, \dots, F$ par respectivement $1, \dots, 6$, on a $n = 6$ puis par exemple

$$a_{1,1} = 0, \quad a_{1,2} = 0, \quad a_{2,1} = 5, \quad a_{1,5} = 4, \quad \dots$$

Exercice 1:

Considérons le graphe simple, orienté et pondéré ci-dessous :



(1)

Remplir complètement la matrice *Adj* entamée ci-dessous pour notre exemple de graphe (**??**)

$$Adj = \begin{pmatrix} 0 & 0 & ? & ? & 4 & ? \\ 5 & ? & \dots & & & \\ \vdots & & & & & \end{pmatrix}$$

Concerant maintenant la description des graphes par un dictionnaire, voici comment on prpcède classiquement :

On définit un dictionnaire (appelé ici `G`) de clés (`G.keys()`) l'ensemble des noms des sommets :

```
G={'A': ... , 'B': ... , 'C': ... etc }
```

Chaque valeur de `G` (`G.values()`) associé à une clé "sommet" est un dictionnaire des "enfants" de "clé", qui donne le poids de l'arrête, en particulier :

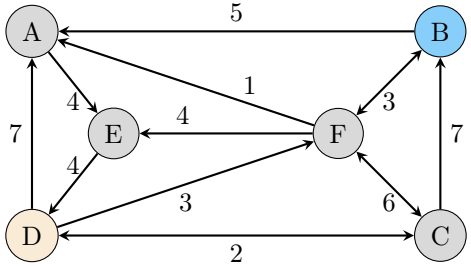
```
G={'A': {'E':4} , 'B': ... , 'C': {'B':7,'F':6}, etc }
```

Exercice 2:

1. Construire le graphe `G` sous forme de dictionnaire pour l'exemple de l'exercice précédent.
2. Vérifier ensuite à l'aide de l'appel de `G.keys()` que vous avez bien défini l'ensemble des clés comme l'ensemble des sommets.
3. Afin de comprendre comment manipuler ce dictionnaire, demandez à Python d'affichez le poids de l'arrête $E \to D$ grâce à `G` et vérifiez que vous obtenez bien 4.

Construction de l'algorithme de Dikjstra sur un exemple :

Considérons le graphe (**??**) pour lequel nous avons choisi un point d'entrée et de sortie.



(2)

Le but de l'algorithme de Dijsktra est de déterminer le chemin de distance minimale entre le point d'entrée et le point de sortie, ici B et D. Naïvement, on pourrait se dire qu'il suffit de parcourir l'ensemble de toutes les possibilités de chemin pour ensuite choisir le plus court. Plusieurs problèmes s'annoncent de manière générale. Supposons que le graphe ait n sommets :

— Pour chaque sommet, il y a potentiellement $n-1$ autres sommets reliés. Ainsi, si on regarde l'ensemble des chemins avec k sommets, il y a alors $(n-1)^k$ possibilités potentielles de chemins. Prenons un exemple simple. Si on a $n = 20$ sommets (ce qui est très peu !!), on peut tout à fait imaginer avoir des chemins de poids $k = n - 1$ à étudier, ce qui donnerait

1 978 419 655 660 313 589 123 979 possibilités de chemins à construire !

Même pour un ordinateur c'est assez long. En effet, considérons le programme extrêmement simple suivant :

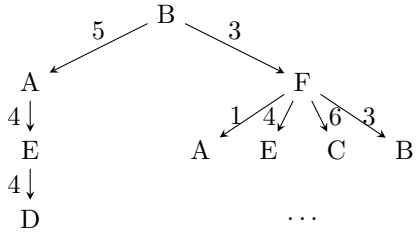
```
1  a=1
2  for i in range(19**j):
3      a=a+1
```

à puissance d'ordinateur égal :
pour $j = 5$, il met $0,22$ secondes à s'exécuter
pour $j = 6$, il met $4,26\,s$.
Pour $j = 7$, il met $82,2\,s \simeq 1\,m\,22\,s$.
Pour $j = 8$, il met $4468.13\,s \simeq 1h\,14\,m\,28\,s$.
Je vous laisse imaginer le temps d'exécution si vous rajoutez du "vrai" programme à l'intérieur et si vous prenez $j = 19\dots$

— Étant donné qu'on peut peut-être repasser plusieurs fois par les mêmes points dans un chemin, on ne sait pas vraiment quelle sera le nombre de sommets que l'on va devoir parcourir avant d'arriver au but et à quel moment on aura en effet parcouru l'ensemble de toutes les possibilités.

On va donc essayer d'optimiser un peu le principe de parcourt afin de limiter le nombre d'études à faire et ainsi de laisser la possibilité au programme de se terminer en un temps raisonnable.

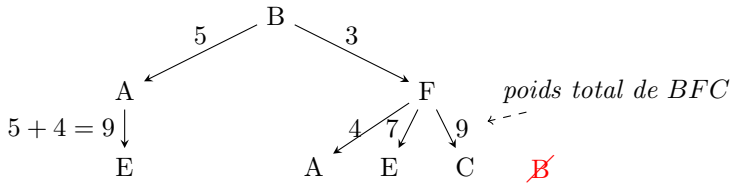
Simplifions progressivement la démarche jusqu'à arriver au principe de l'agorithme de Dikjstra avec le point d'entrée B et le point de sortie D :
- on part de B (étape 0) qu'on appelle ici "parent" et on fait la liste des possibilités de chemin partant de B. *(B aurait donc ici deux "enfants" A et F.)*
- on prend tous les sommets sur lesquels on est arrivés (étape 1). Chacun devient maintenant "parent" à tour de rôle et on recommence pour chacun d'entre eux, et ainsi de suite ...*(Par exemple, le parent A a ici 1 enfant : E. Le parent F a 4 enfants : A, E, C, B. etc...)*



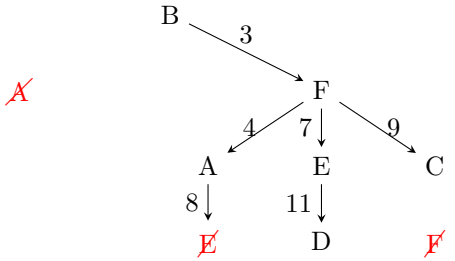
Mais on est confronté à plusieurs problèmes :
— quand s'arrête-t-on ? La première fois que l'on rencontre D ? Non : en effet, il se pourrait que ce ne soit pas le chemin le plus court !
— Que fait-on si on retombe sur le point de départ ? *(par exemple, sur la troisième ligne ?)*
— Que fait-on si on rencontre plusieurs fois le même point dans l'arbre ci-dessus ? *(par exemple E ou A)*

À ces trois questions, on a des solutions :
— Afin de savoir quand s'arrêter, on va garder en mémoire le poids total des chemins parcourus au lieu du poids de chaque arrête.
— Quand on retombe sur un sommet S déjà obtenu dans cette étape ou dans une des étapes précédentes, on étudie le poids de tous les chemins possibles de B à S. On garde celui (ou ceux) qui ont la longueur minimale et on élimine les autres. *(Ils ne servent à rien puisqu'on sait déjà qu'on peut faire mieux.)*
— Si on retombe sur le sommet de départ, on élimine ce chemin car il nous fait tourner en rond.

Voici alors ce que ça donne après deux étapes parcourues (on a retiré le chemin qui rejoignait B)



Mais on voit à ce stade que
— on retombe sur le point A déjà obtenu à la ligne 1, de surcroît avec un poids total inférieur. Le chemin BA de poids 5 est donc moins optimal que le chemin BFA de poids 4. Pour arriver en A, on n'a donc aucun intérêt à suivre le chemin BA. On l'élimine (et on ne le continue donc pas). On ne garde que BFA. On continue maintenant sur la troisième et quatrième ligne en ne gardant systématiquement que les chemins les plus courts pour arriver à un même point :



À ce stade, on est certain d'avoir terminé car il ne reste qu'une seule possibilité et que tous les autres chemins se sont soldés :
- soit par un sommet atteint différemment par un chemin de poids plus grand
- soit par un sommet déjà atteint précédemment sur la même branche (et donc éliminé car le chemin forme une boucle inutile)

La méthode de Dijsktra otpimise encore légèrement ce procédé. De plus, afin de pouvoir un peu plus tard traduire informatiquement toutes ces données, nous allons pour l'instant résumer notre algorithme dans deux tableaux :
- celui "des poids", constitué ici de 7 colonnes : une pour chaque sommet +1 colonne pour nous aider (simples humains) à remplir le tableau,
- et celui "des chaînes ou chemins", ou plus exactement "des prédécesseurs" :

tableau des poids

A	B	C	D	E	F	Parent
⋮	⋮	⋮	⋮	⋮	⋮	⋮

tableau des prédécesseurs

A	B	C	D	E	F

Chaque ligne du tableau des poids modélisera **une étape** du trajet. À chaque étape :
— le point noté dans la colonne «Parent» est celui est celui duquel on fait partir nos branches dans l'étape considérée *(cf arbre ci-dessus)*
— les cases en en-tête (grises) seront les sommets du graphe,
— le tableau sera rempli par des nombres qui désigneront des distances (poids total) à partir du point d'entrée

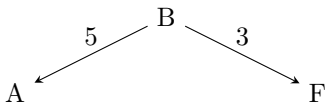
Voyons immédiatement comment faire sur l'exemple :

On commence avec notre sommet de départ, appelé «Entrée» (dans notre exemple, c'est "B"). Nous avons comme but d'aller jusqu'à l'emplacement d'arrivée finale,

appelé «Sortie» (dans notre exemple : "D").

Étape 1 : *Première ligne du tableau des poids*

On va représenter dans le tableau des poids l'arbre



Représentation du point d'entrée dans le tableau :

— On note "B" dans la première ligne de la case "Parent"
— On met 0 dans la case "B". *(le poids de B à B est nul et ceci signifira qu'on s'interdit de repasser par le point "B" pour éviter les cycles.)*

tableau des poids

A	B	C	D	E	F	Parent
	0					B

Remplissage du reste de la ligne :

On rentre ensuite tous les poids des chemins possibles partant de "B" *(enfants du parent B)* :

tableau des poids

A	B	C	D	E	F	Parent
5	0				3	B

car : $\begin{cases} B \to A \text{ est de poids } 5 \\ B \to F \text{ est de poids } 3 \end{cases}$

Les autres sommets non joignables à partir de B sont représentés par ∞ :

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B

Dans le tableau des prédécesseurs :
Les deux sommets atteints sont A et F. Dans la chaîne qui arrive à A, le sommet qui arrive avant lui (son prédécesseur) est B. De même pour F. On note donc B comme prédécésseur dans les cases A et F. B quant à lui n'a pas de prédecesseur.

tableau des prédécesseurs

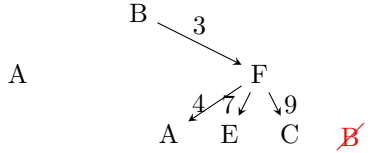
A	B	C	D	E	F
B					B

Doit-on continuer ?

Dans le tableau des poids : on voit que le point de Sortie (dans notre exemple : "D") n'est pas encore atteint car on a ∞ dans la case D. (On voit également que le point n'est pas atteint car D n'a pas encore de prédécesseur.)
On continue donc le raisonnement.

Étape 2 : *$2^{ème}$ ligne et suivantes*

Quand une ligne est entièrement remplie, on repère le chemin de poids minimal fait jusque là (ici, en colonne F). Sur l'arbre étudié auparavant, cela revient à ne pas considérer la branche de gauche pour l'instant et à regarder la suite à partir du nouveau parent F :



Notons que la branche de gauche n'est pas totalement oubliée : elle est toujours là. En effet, dans le tableau des prédécesseurs, on sait que le prédécesseur de A est B et que le poids total de cette branche est 5 d'après le tableau des poids.

Afin de clarifier le raisonnement, on entame pour l'instant **une nouvelle ligne sur le tableau des poids** qui symbolise l'étape suivante. **On ne rajoute pas** de ligne au tableau de prédecesseurs.

— Le nouveau point "Parent" (ici : F) est à inscrit dans la colonne "Parent" et on note 0 dans sa colonne (ici : dans la colonne F).
— La (ou les) colonne(s) contenant déjà 0 (ici : "B") restent vides ou à nouveau remplies par 0. (On ne repasse pas deux fois par le même point ; *inutile de tourner en rond ...*)
— Pour un sommet atteint pour la première fois (ici E et C), on inscrit le poids total de la chaîne et les prédecesseurs

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B
		9		7	0	F

tableau des prédécesseurs

A	B	C	D	E	F
B		F		F	B

— Pour un sommet atteint lors d'une étape précédente mais qui ne l'est pas à cette étape là (pas de cas de ce type ici), on recopierait simplement la case de la ligne précédente et on ne toucherait pas au prédécesseur.
— Pour un sommet atteint lors d'une étape précédente mais qui est à nouveau atteint ici (seul cas ici : A) il faut comparer le poids total des chemins pour ne garder que le plus avantageux :
Ici le sommet A est à nouveau atteint à partir de F.
Le poids total de la chaîne BFA est de 4, alors que la poids total de l'autre chemin possible BA était de 5. Le chemin le plus avantageux est celui de longueur 4. On peut donc définitivement oublier la chaîne BA. Pour ce faire, on inscrit 4 dans le tableau des poids et on change le prédécesseur de A par F au lieu de B :

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B
~~5~~ 4		9		7	0	F

tableau des prédécesseurs

A	B	C	D	E	F
~~B~~ F		F		F	B

(le cas où il y a égalité des poids sera discuté plus tard.)

— On continue à noter ∞ pour les points qui ne sont toujours pas atteints. (ici : D)

Résultat final pour l'étape 2 :

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B
~~5~~ 4		9	∞	7	0	F

tableau des prédécesseurs

A	B	C	D	E	F
~~B~~ F		F		F	B

Pour l'instant, on a donc explicité :
- dans la dernière ligne du tableau de gauche : la poids des chemins (il n'y a pas plus court pour l'instant) partant de l'entrée (B) et arrivant aux divers points donnés en en-tête du tableau.
- dans le tableau de droite : les chemins parcourus (à lire à l'envers !). Par exemple, pour aller jusqu'à E, la liste des prédécesseurs successifs est $E \to F$ puis $F \to B$. Le chemin le plus court pour aller en E est donc pour l'instant $B - F - E$.

Le point de sortie (D) n'est toujours pas atteint. On continue. Avant de regarder la suite, essayez donc de remplir vous-même la prochaine ligne.
...
Voyons maintenant ce que vous avez dû obtenir (pour l'instant, tout sauf la colonne E) :

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B
4		9	∞	7	0	F
0		9	∞			A

tableau des prédécesseurs

A	B	C	D	E	F
F		F		F	B

La distance la plus petite est atteinte pour A cette fois-ci. On met donc 0 dans la colonne A et A dans la colonne "Parent".
Le sommet C n'étant pas atteint, la distance la plus courte est toujours de 9 et son prédécesseur n'a pas changé.
Pour le sommet E, on est confronté à un dilemne : deux chemins permettent d'y arriver :
- celui qui a déjà été obtenu dans la ligne précédente de poids 7
- et celui qui part maintenant de A de poids totale

$$\underbrace{4}_{\text{poids du chemin de } B \text{ à } A} + \underbrace{4}_{\text{poids de l'arrête de } A \text{ à } E} = 8.$$

Le chemin le plus court est celui qui existait déjà. On le garde. Nous n'avonc donc cette fois-ci aucune raison de changer le prédécesseur de E. On obtient donc

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B
4		9	∞	7	0	F
0		9	∞	7		A

tableau des prédécesseurs

A	B	C	D	E	F
F		F		F	B

on continue sur la même lancée :

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B
4		∞	∞	7	0	F
0		9	∞	7		A
		9	11	0		E

tableau des prédécesseurs

A	B	C	D	E	F
F		F	E	F	B

Le sommet de sortie D est atteint. Si le chemin correspondant est de poids minimale (ici, 11, non minimal), on peut s'arrêter. Sinon, on continue pour vérifier s'il n'y a pas un autre moyen d'y arriver de manière plus courte :

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B
4		9	∞	7	0	F
0		9	∞	7		A
		9	11	0		E
		0	11			C

tableau des prédécesseurs

A	B	C	D	E	F
F		F	E	F	B

Il ne reste plus qu'une seule colonne. La seule façon de continuer ici est de rajouter une ligne qui ne serait constitué que de cases vides et de 0 (y réfléchir...) On peut donc, en tant qu'humain qui réfléchit, s'arrêter là. *Notons de plus qu'il serait tout à fait possible que certains sommets ne soient jamais atteints par l'algorithme. On aurait donc toujours une ou plusieurs cas ∞ à la fin, ce qui n'est absolument pas un problème.*

On obtient alors la poids du chemin : 11, ainsi que le trajet le plus court de notre algorithme. Attention, celui-ci n'est pas constitué des lettres dans la colonne "Parent" mais bien du chemin constitué par les prédécesseurs, ici :

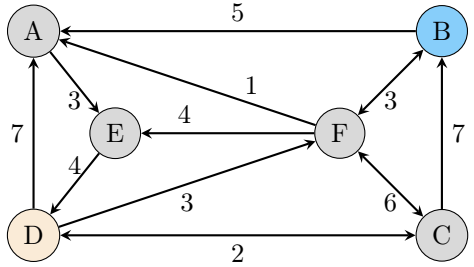
BFED

Il n'en existe pas de plus court que celui-ci, mais il peut éventuellement en exister d'autres de même poids, par exemple ici : $BFCD$.

Cas litigieux :

Il est possible, suivant les cas, d'obtenir deux chemins possibles avec le même poids.

⋆ *Premier cas :* on arrive au même point de deux manières différentes. Par exemple, si la poids de A à E était de 3 et non 4 :



un cas de conscience se pose à la troisième étape. Pour arriver à E on peut venir par le nouveau sommet A avec une poids 7, ou alors arriver par "l'ancien" sommet F avec le même poids. Le tableau des poids ne change pas :

tableau des poids

A	B	C	D	E	F	Parent
5	0	∞	∞	∞	3	B
4		9	∞	7	0	F
0		9	∞	7		A

mais le prédécesseur de E peut être choisi n'importe comment entre A et F. C'est donc le deuxième tableau qui peut **éventuellement** être modifié en changeant le prédécesseur de E :

tableau des prédécesseurs

A	B	C	D	E	F
F		F		~~F~~ A	B

Dans certains cas, on peut éventuellement opter pour celui qui possède le moins de sommets intermédiaires. *(Par exemple, dans des problèmes de minimisation des feux ou de carrefours dans une ville pour éviter la perte de temps.)*

⋆ *Deuxième cas :* **Il y plusieurs "minimum"** : On ne sait pas de quel sommet repartir car il n'y a pas de minimum clair sur une ligne. Par exemple, modifions les poids de la première étape :

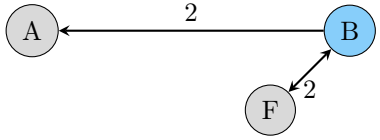


tableau des poids

A	B	C	D	E	F	Parent
2	0	∞	∞	∞	2	B

tableau des prédécesseurs

A	B	C	D	E	F
B					B

Dans ce cas encore, on choisit un des deux sommets (peu importe lequel) comme nouveau point de Parent. Supposons que nous partions ici de A. La ligne suivante serait alors :

tableau des poids

A	B	C	D	E	F	Parent
2	0	∞	∞	∞	2	B
0	0	∞	∞	4	2	A

tableau des prédécesseurs

A	B	C	D	E	F
B				A	B

Mais du coup, le "2" de la case F est toujours là et c'est nécessairement la valeur minimale *(à méditer)*. On prend donc maintenant comme point de Parent le sommet F :

tableau des poids

A	B	C	D	E	F	Parent
2	0	∞	∞	∞	2	B
0	0	∞	∞	4	2	A
0	0	8	∞	4	0	F

tableau des prédécesseurs

A	B	C	D	E	F
B		F		A	B

et au final, nous avons étudié les deux points litigieux A et F !

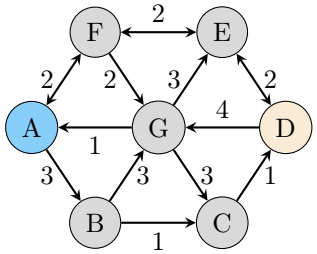
Mise en pratique :

Exercice 3:

1. Refaire entièrement le cas de l'exemple vous même pour voir si vous avez compris.
2. Avec le même graphe pondéré, construire le tableau et déterminer le chemin le plus court
   - **a)** entre C et A.
   - **b)** Entre C et E

Exercice 4:

On considère le graphe de chemins pondérés ci-dessous



Établir à l'aide de l'agorithme de Dijkstra un chemin de poids minimal pour aller de A à D.

On cherche maintenant à faire faire le travail à Python. On peut utiliser des listes et la matrice d'adjacence, mais on propose ici d'utiliser plutôt des dictionnaires (qui accélèrent légèrement le déroulement de l'algorithme.)
Remarquons dans un premier temps que dans l'élaboration du tableau des poids, on n'a besoin uniquement que de la dernière ligne et que le "Parent" n'a pas nécessité d'être retenu d'une étape à l'autre. On propose donc de modéliser le problème par deux dictionnaires `Poids_total` et `Predecesseurs`.

- `Poids_total` représentera la dernière ligne en cours du tabeau des poids (sans le "Parent")
- `Predecesseurs` contiendra le prédecesseur de chaque sommet.

Les deux dictionnaires évolueront à chaque étape.

Il va donc falloir

- les initialiser (exercice 5) ;
- les modifier par la première étape, qui est un peu plus simple que les suivantes (exercice 6) ;
- rechercher le nouveau sommet parent (exercice 7) ;
- les modifier par étapes successives (exercice 8) ;
- déterminer à quel moment on a terminé (exercice 9) ;
- afficher le chemin le plus court (exercice 10) grâce à un tableau des prédecesseurs ;
- et regrouper tout ceci dans une fonction qui déterminer un algorithme général (exercice 11).

Exercice 5: Initialisation

On reprend le graphe `G` des sommets **établi à l'exercice 2**. On rappelle que dans ce cas pour un sommet `S`, `G[S]` désigne le dictionnaire des sommets pondérés avec le poids de l'arrête. Par exemple, `G['C']={'B':7,'F':6}` ou `G['C']={'F':6,'B':7}` (l'ordre des points dans `G['C']` n'a pas d'importance.)

1. Initialiser un dictionnaire `Poids_total` de clés "l'ensemble des sommets de `G`" et de valeurs "∞" pour chaque clé.
   Attention ! On souhaite que cette initialisation fonctionne automatiquement même si on modifie le graphe `G`. On prendra donc soin de faire ceci avec une boucle adéquate !

   (On note que *la bibliothèque nympy permet de définir le "nombre" $+\infty$ par la valeur* ***`np.inf`****. Celui-ci se comporte comme un nombre "classique" dans les calculs et comparaisons.)*
   Le résultat affiché par Python à l'instruction `print(Poids_total)` doit être le suivant :

```
{'A': inf, 'B': inf, 'C': inf, 'D': inf, 'E': inf, 'F': inf}
```

   La valeur ***`inf`*** *symbolise que le sommet n'est pas encore atteint.*

2. De la même façon, initialiser un dictionnaire `Predecesseurs` de clés "l'ensemble des sommets de `G`" et de valeurs "*None*" pour chaque clé.
   Attention !Là aussi, on souhaite que cette initialisation fonctionne automatiquement même si on modifie le graphe `G`. On prendra donc là aussi soin de faire ceci avec une boucle adéquate !

   Le résultat affiché par Python à l'instruction `print(Predecesseurs)` doit être le suivant :

```
{'A': None, 'B': None, 'C': None, 'D': None, 'E': None, 'F': None}
```

   La valeur ***`None`*** *symbolise une case vide.*

Exercice 6: Première étape

1. En utilisant la technique d'initialisation mise en place dans l'exercice précédent, constuire une fonction `Etape1(G,Entree)` qui prend en argument le graphe "dictionnaire" `G` et le sommet de départ "`Entree`", et qui rend les dictionnaires `Poids_total` et `Predecesseurs` après la première étape.
2. On vérifiera que pour le graphe donné en exemple et le sommet parent "B", on trouve le résultat

```
{'A': 5, 'B': 0, 'C': inf, 'D': inf, 'E': inf, 'F':
    3} # pour les poids

{'A': 'B', 'B': None, 'C': None, 'D': None, 'E':
   None, 'F': 'B'} # pour les prédécesseurs
```

Exercice 7: Nouveau parent

1. Constuire une fonction `nouveau_parent(Poids)` qui prend en argument un dictionnaire du type `Poids_total`, et qui rend le parent de la ligne suivante, c'est-à-dire un des sommets qui correspond au poids total minimum.
2. On vérifiera que pour le résultat de `Poids_total` obtenu dans l'application de la question précédente, on trouve ici comme nouveau parent `'F'`.

Exercice 8: Etapes successives

1. Constuire une fonction `Etape_suivante(Poids_total,Predecesseurs,G)` qui prend comme argument les dictionnaires `Poids_total` et `Predecesseurs` issus d'une étape $i \geq 1$, et qui rend les dictionnaires `Poids_total` et `Predecesseurs` à l'issue de l'étape suivante $i+1$.
2. On vérifiera que pour le graphe donné en exemple et le sommet Parent B, pour les dictionnaires `Poids_total` et `Predecesseurs` obtenus à l'issue de l'étape 1, on trouve le résultat

```
{'A': 4, 'B': 0, 'C': 9, 'D': inf, 'E': 7, 'F': 3}
   # pour les poids
{'A': 'F', 'B': None, 'C': 'F', 'D': None, 'E': 'F
   ', 'F': 'B'} # pour les précédesseurs
```

Exercice 9: La fin ?

Pour simplifier les vérifications, on observe qu'informatiquement, on peut naïvement continuer jusqu'à ce qu'on n'ait plus que des 0 et des ∞ dans le tableau des poids !

1. Construire alors une fonction qui détermine, à partir du dictionnaire `Poids_total` en vigeur si c'est terminé. Elle devrait idéalement rendre simplement `True` ou `False`.
2. Vérifier qu'à l'issue de la deuxième étape, ce n'est pas terminé.

Exercice 10: Le chemin et son poids

1. À partir d'un dictionnaire des prédécesseurs considéré comme abouti, faire une fonciton qui rend le chemin le plus court obtenu (sous forme de liste ou de texte.)
2. À partir du chemin précédent obtenu et d'un graphe `G`, faire une fonction qui rend le poids total du chemin donné en argument.

Exercice 11:

1. Pour finir, construire une fonction `dijkstra(G,Entree,Sortie)` qui rend le chemin obtenu pour aller de `Entree` à `Sortie` dans un graphe quelconque `G`.
2. Appliquer l'algorithme aux deux graphes évoqués dans cette feuille (ex 1 et ex 4) et vérifiez vos résultats.

Exercice 12: Bonus : Seulement pour ceux qui ont fini le reste

On pourra se demander comment :

1. à partir d'une liste de Sommets, (par exemple `Sommet='ABCDEF'`) et d'une matrice d'adjacence `Adj` comme celle construire dans l'exercice 1, on peut construire le dictionnaire associé au graphe.
2. et au contraire, à partir d'un dictionnaire `G`, contruire la liste des sommets et la matrice d'adjacence.