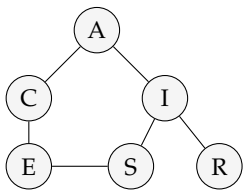


Parcours de graphes

Fiche de synthèse

① REPRÉSENTER UN GRAPHE

- Une **matrice d'adjacence** est un tableau à deux dimensions : la case de la ligne i et de la colonne j indique la présence, ou le poids, de l'arc du sommet i vers le sommet j . Elle occupe n^2 cases pour n sommets.
- Une **liste d'adjacence** donne, pour chaque sommet, la liste de ses voisins. Elle convient mieux quand il y a peu d'arêtes. En Python, on l'écrit avec un dictionnaire dont les clés sont les sommets.



```
G = { "A": [ "C", "I" ],
      "C": [ "A", "E" ],
      "I": [ "A", "R", "S" ],
      "R": [ "I" ],
      "E": [ "C", "S" ],
      "S": [ "E", "I" ] }
```

Un graphe non orienté se traite comme un graphe orienté où chaque arête est un arc dans les deux sens. Pour un graphe pondéré, on range des couples (voisin, poids).

② PARCOURS EN PROFONDEUR

Définition 1 (Parcours en profondeur) Le **parcours en profondeur** avance d'un sommet à un voisin non encore visité tant que c'est possible, puis revient en arrière. Il utilise une **pile**, celle des appels récursifs ou une pile explicite.

```
def profondeur(G, S, deja_vus):
    deja_vus.append(S)          # marquer S
    print(S)                   # traiter S
    for voisin in G[S]:
        if voisin not in deja_vus:
            profondeur(G, voisin, deja_vus)

profondeur(G, "A", [])         # A C E S I R
```

La version itérative empile les voisins et marque un sommet quand elle le dépile :

```
def profondeur_iteratif(G, S):
    deja_vus = []
    pile = [S]
    while pile != []:
        S = pile.pop()
        if S not in deja_vus:
            deja_vus.append(S)
            print(S)
            for voisin in G[S]:
                if voisin not in deja_vus:
```

```
pile.append(voisin)
```

Elle affiche A I S E C R : les voisins sont dépilés dans l'ordre inverse de leur empilement, donc le dernier voisin de A est visité en premier.

③ PARCOURS EN LARGEUR

Définition 2 (Parcours en largeur) Le **parcours en largeur** visite le sommet de départ, puis tous ses voisins, puis les voisins de ses voisins, et ainsi de suite. Il utilise une **file**.

```
def largeur(G, S):
    deja_vus = [S]
    file = [S]
    while file != []:
        S = file.pop(0)
        print(S)
        for voisin in G[S]:
            if voisin not in deja_vus:
                deja_vus.append(voisin)
                file.append(voisin)
```

Depuis A, il affiche A, puis C et I, à une arête de A, puis E, R et S, à deux arêtes.

	Profondeur	Largeur
Structure	pile : dernier entré, premier sorti	file : premier entré, premier sorti
Marquage	au dépilement	à l'enfilement

En largeur, un sommet est atteint par une chaîne qui a le moins d'arêtes possible : le parcours en largeur donne un **plus court chemin en nombre d'arêtes**.

④ DÉTECTER UN CYCLE

On parcourt le graphe en profondeur sans revenir par l'arête d'arrivée. Si l'on dépile un sommet déjà marqué, il a été atteint par deux chemins différents : le graphe possède un cycle.

```
def possede_cycle(G, S):
    deja_vus = []
    pile = [S]
    while pile != []:
        S = pile.pop()
        if S in deja_vus:
            return True
```

```

deja_vus.append(S)
for voisin in G[S]:
    if voisin not in deja_vus:
        pile.append(voisin)
return False

```

⑤ RECHERCHER UN CHEMIN

On parcourt le graphe en largeur en notant, dans un dictionnaire, le **prédécesseur** de chaque sommet atteint, c'est-à-dire le sommet d'où l'on vient. On reconstitue ensuite le chemin en remontant les prédécesseurs depuis l'arrivée jusqu'au départ.

```

def chemin(G, depart, arrivee):
    pred = {depart: None}
    file = [depart]
    while file != []:
        S = file.pop(0)
        for voisin in G[S]:
            if voisin not in pred:
                pred[voisin] = S
                file.append(voisin)
    if arrivee not in pred:
        return []
    R = [arrivee]
    while R[-1] != depart:
        R.append(pred[R[-1]])
    R.reverse()
    return R

```

`chemin(G, "A", "S")` renvoie ['A', 'I', 'S'].

📌 À retenir.

- Un graphe se représente par une matrice ou par un dictionnaire d'adjacence.
- Le parcours en profondeur utilise une pile, le parcours en largeur une file.
- On marque les sommets visités pour ne pas les traiter deux fois.
- Le parcours en largeur trouve un chemin qui a le moins d'arêtes possible.
- Rencontrer un sommet déjà marqué en profondeur révèle un cycle.