09 - Graphes

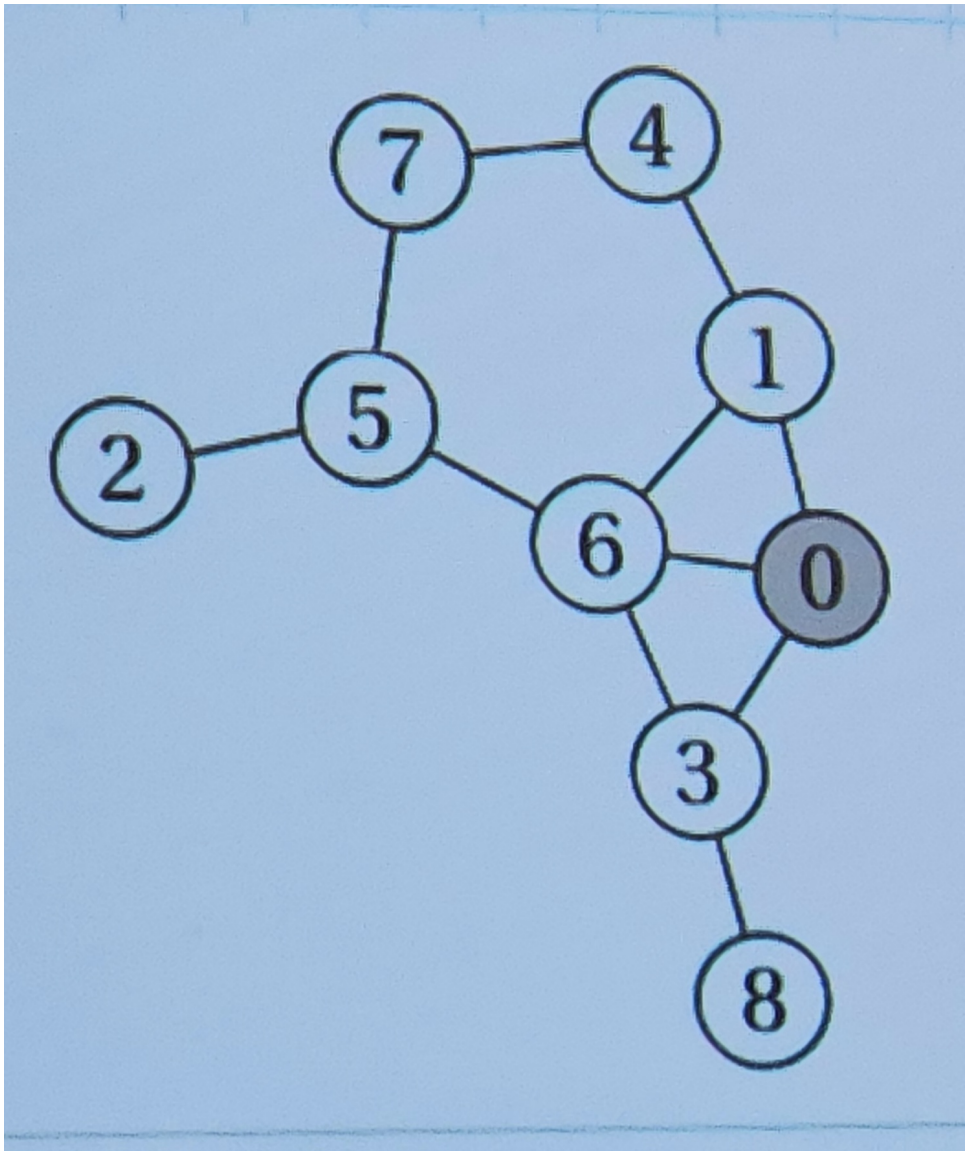
4) Parcours de Graphes

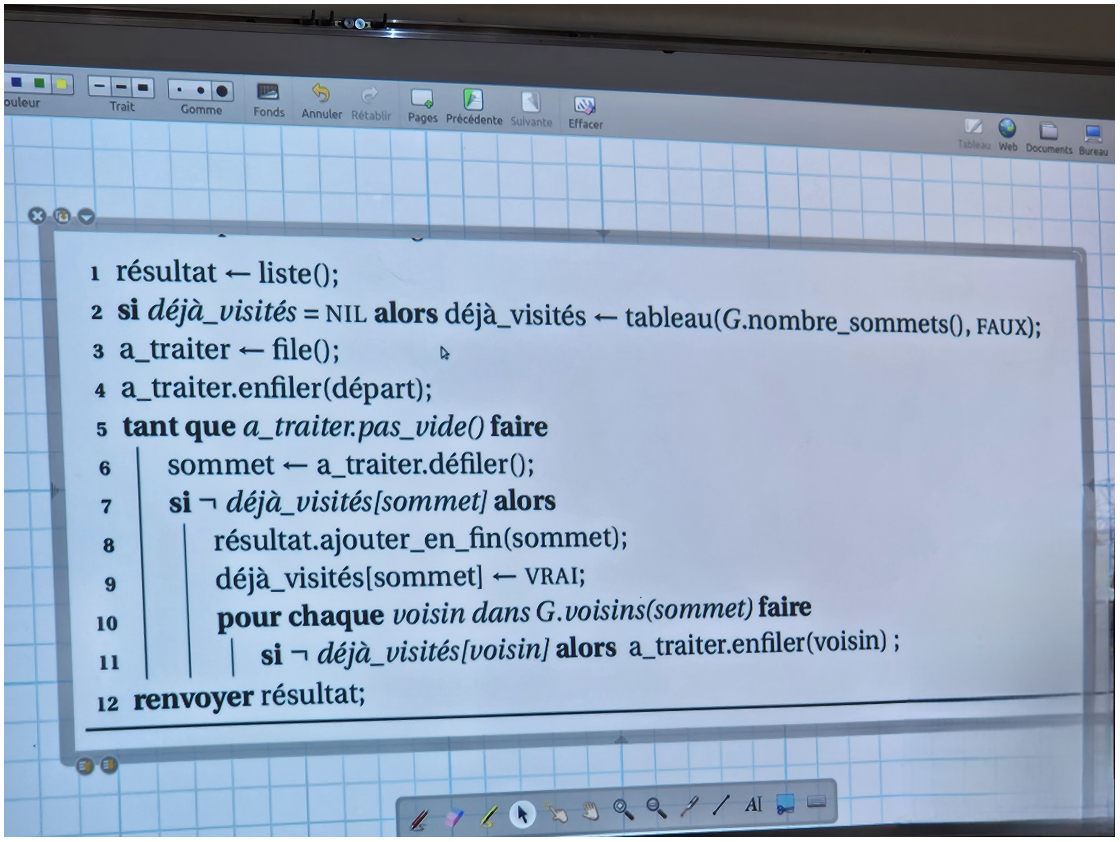
> Les algorithmes de calculs ou de recherche dans des graphes entrent en général dans deux catégories : Parcours en largeur ou en profondeur.

A) Parcours en largeur (Breadth First Search)

> Les parcours en largeur d'un graphe G partant d'un sommet A privilégie l'exploration des voisins de A puis, seulement lorsque tous les voisins ont été visités, les voisins des voisins.

Il s'agit d'un parcours ou la priorité est la proximité.


7
4
1
5
2
6
0
3
8




```
def parcours_largeur(G, v): #vu en cours
    visites = deque(v)
    a_visiter = deque(voisins(G, v))
    while a_visiter:
        #pour une file utiliser popleft() pour
                #retirer le premier élément
        sommet = a_visiter.popleft()
        if sommet not in visites:
            visites.append(sommet)
            for voisin in voisins(G, sommet):
                if voisin not in visites:
                    a_visiter.append(voisin)
    return visites

def parcours_largeur(graph, depart):
    """
    Effectue un parcours en largeur
        sur un graphe à partir d'un
```

```
        nœud de départ.
    Args:
        graph (list): Le graphe représenté
                sous forme de liste d'adjacence.
        depart (int): Le nœud de départ du parcours.
    Returns:
        list: Une liste contenant les nœuds
                visités dans l'ordre
                du parcours en largeur.
    """
    resultat = []  # Liste pour stocker les
        #nœuds visités
    deja_visites = [False] * len(graph)  # Liste pour
        #marquer les nœuds déjà visités
    a_traiter = []  # File pour stocker
        #les nœuds à traiter
    a_traiter.append(depart)  # Ajoute le nœud
        #de départ à la file
    deja_visites[depart] = True  # Marque le nœud
        #de départ comme visité
    while a_traiter:
        # Retire le premier nœud de la file
        noeud_courant = a_traiter.pop(0)
        resultat.append(noeud_courant)  # Ajoute le
                #nœud courant à
                #la liste des nœuds visités
        # Parcourt les voisins du nœud courant
        for voisin in graph[noeud_courant]:
            if not deja_visites[voisin]:
                a_traiter.append(voisin)  # Ajoute le
                                #voisin à la file
                deja_visites[voisin] = True  # Marque le
                                #voisin comme visité
        return resultat


def parcours_profondeur(G, v):
    visites = deque(v)
```

```
    a_visiter = deque(voisins(G, v))
    while a_visiter:
        sommet = a_visiter.pop()
        if sommet not in visites:
            visites.append(sommet)
            for voisin in voisins(G, sommet):
                if voisin not in visites:
                    a_visiter.append(voisin)
    return visites
```

Une des applications de graphes : Coloration de graphe pour créer des EDT

coloration propre : Chaque sommet à une couleur différente de celles de ses voisins

pour un EDT le nombre de couleurs corresponds au nombre d'horaires, l'objectif est d'en avoir le minimum.

https://www.youtube.com/watch?v=CUe7LC3CdH8&ab_channel=ÀladécouvertedesgraphesÀladécouvertedesgraphes

https://www.youtube.com/watch?v=w3LFOAw-J1I&ab_channel=ÀladécouvertedesgraphesÀladécouvertedesgraphes