09 - Graphes

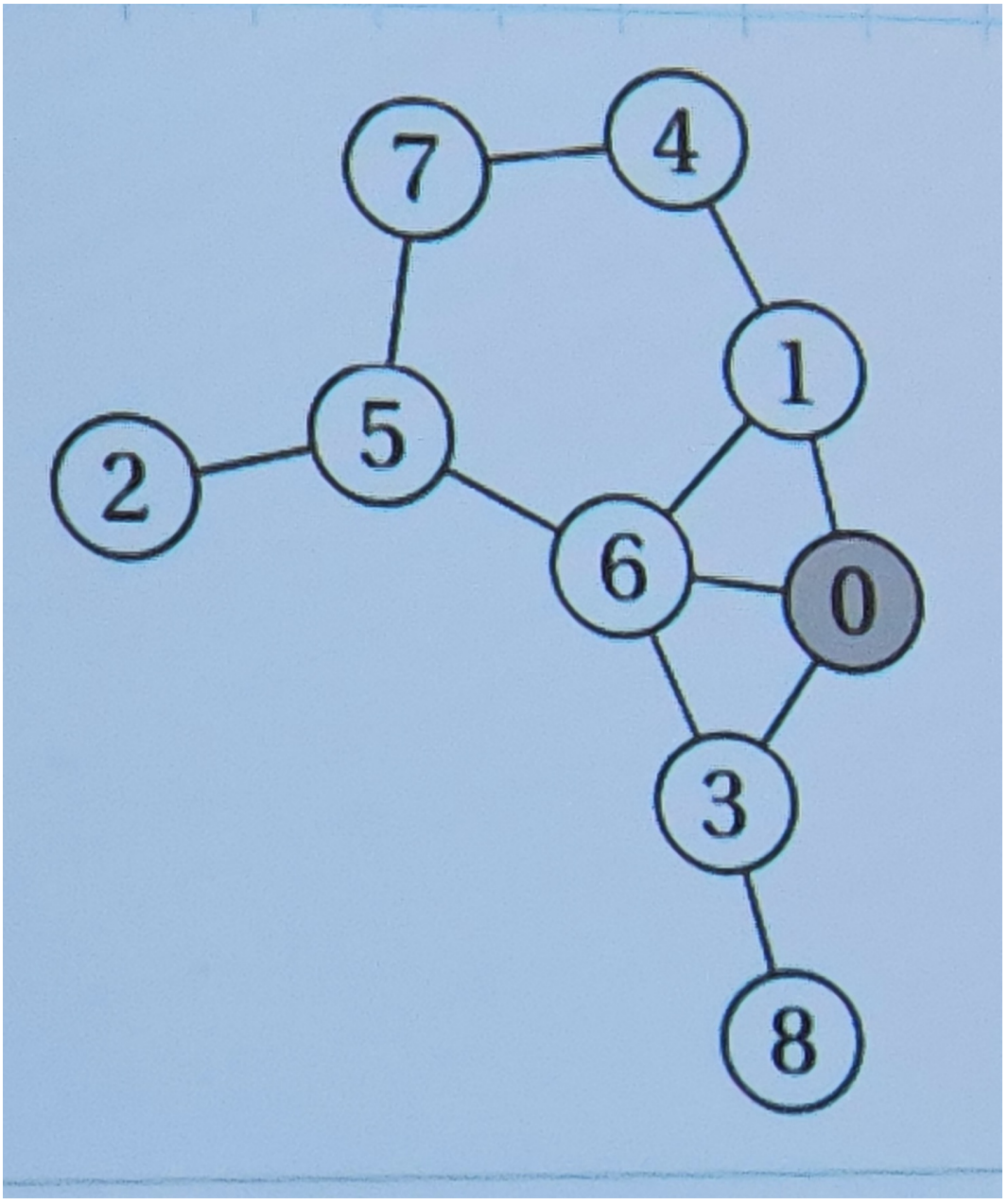
4) Parcours de Graphes

> 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.




```
def parcours_largeur(graph, depart):
    resultat = []
    deja_visites = [False] * len(graph)
    a_traiter = []

    a_traiter.append(depart)
    deja_visites[depart] = True

    while a_traiter:
```

```
        noeud_courant = a_traiter.pop(0)
        resultat.append(noeud_courant)

        for voisin in graph[noeud_courant]:
            if not deja_visites[voisin]:
                a_traiter.append(voisin)
                deja_visites[voisin] = True

    return resultat

# Exemple d'utilisation
# Supposons que le graphe soit représenté par
# une liste d'adjacence
# graph = {0: [1, 2], 1: [3, 4], 2: [5], 3: [], 4: [6], 5:
# [], 6: []}
# On appelle la fonction parcours_largeur avec le graphe
# et le nœud de départ, par exemple, 0
# resultat = parcours_largeur(graph, 0)
# print(resultat)
# Sortie: [0, 1, 2, 3, 4, 5, 6]

graph = {0: [1, 2], 1: [3, 4], 2: [5], 3: [], 4: [6],\
 5: [], 6: []}
print(parcours_largeur(graph, 0))
```