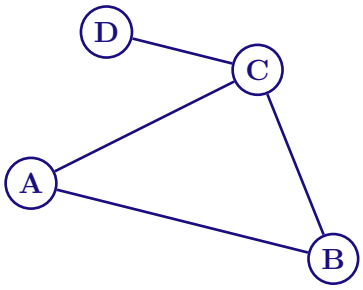
TP : Parcourir un graphe en Python

Parcours en profondeur

On donne ci-dessous un graphe G :



Principe

Le parcours en profondeur consiste à parcourir les sommets reliés d'un graphe. Une pile contient les différents sommets voisins et une liste contient les sommets visités.

La liste des sommets visités donne le parcours du graphe en profondeur lorsque la pile est vide.

En python

La fonction **parcours_profondeur** a deux paramètres : le graphe **g** et le sommet de départ **s**.

La fonction renvoie la liste des sommets visités **vus**.

```
1  def parcours_profondeur(g,s):
2      vus=[]
3      p=creer_pile()
4      p.empiler(s)
5      while not p.est_vide():
6          print(p,vus)
7          s=p.depiler()
8          if s in vus:
9              continue
10         vus.append(s)
11         for v in g.voisins(s):
12             p.empiler(v)
13     return vus
```

1) Donner un parcours en profondeur possible du graphe G en partant du sommet **A**.

2) La fonction en Python contient l'instruction **continue** en ligne 7. Quelle est l'utilité de cette instruction ?

3) On applique la fonction avec le graphe G en partant du sommet **A**.
Donner les différents états de la pile et de la liste vus à chaque tour de boucle.

4) **a)** Récupérer sur l'ENT les scripts Python **parcours.py**, **graphe.py** et **pile.py** puis éditer le fichier **parcours.py**.

b) Recopier la fonction parcours_profondeur puis vérifier son bon fonctionnement

5) On donne le graphe **H** défini par son dictionnaire d'adjacence :

```
H={'A':{'B','C'},\
   'B':{'D','E'},\
   'C':{'F'},\
   'D':{'B','E'},\
   'E':{'B','D'},\
   'F':{'C'}
  }
```

Déterminer avec la fonction python un parcours en profondeur du graphe **H** en partant du sommet **A**.

6) On donne le graphe **K** défini par son dictionnaire d'adjacence :

```
K={'A':{'E','C'},\
   'B':{'D','E','I'},\
   'C':{'F','L'},\
   'D':{'B','E','I'},\
   'E':{'B','D'},\
   'F':{'L'},\
   'G':{'J','H'},\
   'H':{'J','G'},\
   'I':{'B','D'},\
   'J':{'H','G'},\
   'L':{'C','F'}
  }
```

Déterminer avec la fonction python un parcours en profondeur du graphe **K** en partant du sommet **A**. Puis du sommet **G**. Que peut-on en déduire ?

7) Écrire une fonction **chemin** qui a pour paramètres le graphe **g** et deux sommets **s** et **t** du graphe et renvoie un chemin entre les deux sommets **s** et **t** s'il existe.

Parcours en largeur

Principe

Le parcours en largeur a pour but de donner le plus court chemin entre deux sommets du graphe s'il existe.

On se donne un sommet de départ puis on détermine tous les sommets distants de 1, puis tous les sommets distants de 2, etc, jusqu'à avoir visité tous les sommets du graphe relié au sommet de départ.

En python

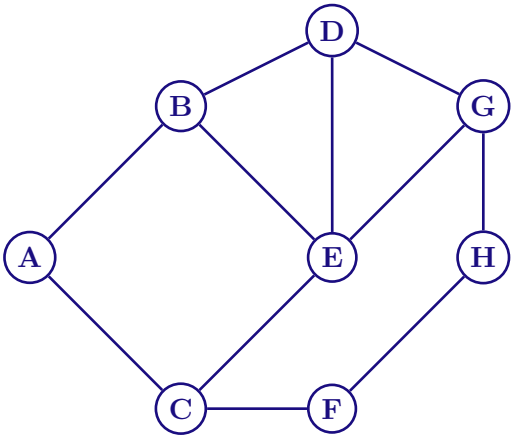
La fonction **parcours_largeur** a deux paramètres : le graphe **g** et le sommet de départ **s**.

La fonction renvoie un dictionnaire **dist** dont les clefs sont les sommets visités et les valeurs sont la distance qui les sépare du sommet de départ.

```
def parcours_largeur(g,s):
    dist={s:0}
    courant={s}
    suivant=set()
    while len(courant)>0:
```

```
        sommet=courant.pop()
        for v in g.voisins(sommet):
            if v not in dist:
                suivant.add(v)
                dist[v]=dist[sommet] + 1
            if len(courant)==0:
                courant,suivant=suivant,set()
    return dist
```

1) On donne ci-dessous un graphe G :



Quel sera le dictionnaire renvoyé par l'appel **parcours_largeur(G,A)** ?

2) Compléter le code de la fonction **parcours_largeur** dans le fichier **parcours.py**.

3) Définir le graphe G par un dictionnaire d'adjacence en Python.

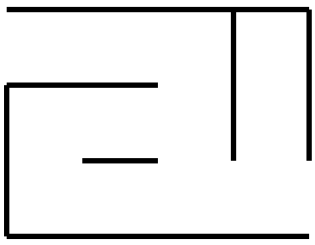
4) Appliquer la fonction de parcours en largeur à ce graphe en Python. Contrôler avec le dictionnaire de la première question.

5) Écrire une fonction **distance** qui prend en arguments un graphe et deux sommets du graphe et renvoie la distance séparant les deux sommets du graphe.

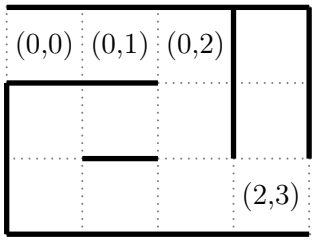
6) Déterminer la distance **AH** du graphe **G**.

Du graphe au labyrinthe

On donne ci-dessous la représentation d'un labyrinthe.



Ce labyrinthe est construit comme une grille dont chaque cellule est repérée par un couple de coordonnées.



On peut alors concevoir un **graphe** tel que :

— Les coordonnées de chaque cellule sont les sommets de notre graphe.

— Les arêtes (arcs) du graphe sont les passages possibles d'une cellule à une autre cellule contigüe.

La réalisation, l'affichage et la résolution des labyrinthes reposent sur des scripts se trouvant dans le fichier **labyrinthe.py** à récupérer sur l'ENT dans le dossier script.

Pour que le programme soit opérationnel, il faut s'assurer d'avoir le module **matplotlib** installé et d'avoir dans le même dossier les fichiers python **graphe.py**, **pile.py** et **parcours.py**.

Il va falloir modifier certains lignes du script pour visualiser vos labyrinthes.

1) Construire le graphe qui donne le labyrinthe ci-dessus.

2) On va modifier le fichier **labyrinthe.py** pour visualiser votre labyrinthe et sa solution :

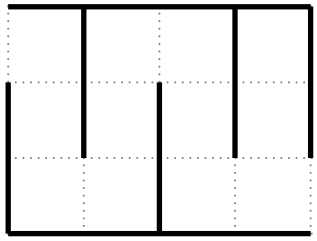
a) modifier la hauteur **N** et la largeur **M** du labyrinthe ;

b) créer votre graphe ; par exemple en donnant la liste des arcs qui le composent dans la variable **arcs**.

c) ensuite exécuter le code et contrôler son exactitude.

3) Modifier votre labyrinthe pour obtenir un seul chemin comme solution.

4) **a)** Créer un graphe pour générer un labyrinthe qui ressemble à la figure suivante :



b) Modifier le fichier **labyrinthe.py** et visualiser le labyrinthe et sa solution.

5) Écrire un script qui génère un labyrinthe ayant la même configuration quel que soit sa taille ($N < 50$ et $M < 50$).