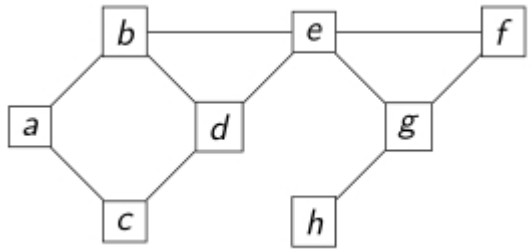
NSI – Graphes – Découverte du parcours en largeur




```
g = {
      'A': ['B', 'C'],
      'B': ['A', 'D', 'E'],
      'C': ['A', 'D'],
      'D': ['B', 'C', 'E'],
      'E': ['B', 'D', 'F', 'G'],
      'F': ['E', 'G'],
      'G': ['E', 'F', 'H'],
      'H': ['G']}
```

Philosophie du parcours en largeur (en anglais Breadth First Search) :

→ Exploration du graphe à partir d'un sommet
→ Détermination de tous les sommets accessibles à partie de ce sommet

→ Le parcours en largeur permet de déterminer la distance entre le sommet de départ et tous les sommets accessibles, c'est à dire le plus petit nombre d'arcs pour relier deux sommets. Il permet aussi de trouver le chemin le plus court d'un sommet source du graphe jusqu'à un autre sommet.

→ Le parcours en largeur consiste à lister les sommets joignables à partir du sommet source, pas « cercles concentriques », c'est à dire en écrivant les sommets par ordre croissant de distance à partir de cette source

→ Pratiquement, on part du sommet source, on note tous les sommets les plus proches, puis on poursuit l'exploration en repartant des premiers sommets découverts.

→ Il n'existe pas forcément un seul parcours en largeur : pour un sommet source, on peut trouver plusieurs parcours en largeur possible, qui respectent la philosophie de l'exploration « du plus proche au plus éloigné ».

→ Selon ce que l'on cherche à obtenir, on pourra compléter le parcours avec :

- la distance du sommet source ;
- le sommet duquel on vient pour la rencontre d'un sommet

→ Pratiquement, pour programmer un parcours en largeur, on utilise une FILE (premier arrivé, premier sorti) , dans laquelle on enfile les nouveaux sommets découverts, et dans laquelle on pioche les sommets à partir desquels on poursuit l'exploration…

ALGORITHMIQUE POUR LE PARCOURS EN D'UN GRAPHE :

PARCOURS EN LARGEUR SIMPLE D'UN GRAPHE

```
données :

graphe : un graphe, sous forme de liste d'adjacence
s : le sommet du graphe à partir on commence le parcours

initialisation :

parcours : variable de type list qui contiendra le parcours en largeur
file : variable de type list qui contiendra les sommets découverts

on place s dans parcours
on place s dans la file

tant qu'il y a des éléments dans la file :
     on défile le premier nœud arrivé
     pour chaque suivant de ce noeud :
          si le suivant n'est pas dans le parcours :
               on ajoute ce suivant au parcours
               on ajoute ce suivant à la file

Quand cette boucle est achevée, la variable parcours contient le parcours
en largeur du graphe à partir de la source s
```

1) Programmer l'algorithme de parcours en largeur décrit ci-dessus.
On pourra appeler cette fonction ***parcours_largeur_simple(graphe, s)*** .

Tester cette fonction, on particulier voici quelques exemples obtenus :

```
print(parcours_largeur_simple(g,'A'))

>>> ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

print(parcours_largeur_simple(g,'E'))

>>> ['E', 'B', 'D', 'F', 'G', 'A', 'C', 'H']

print(parcours_largeur_simple(g,'G'))

>>> ['G', 'E', 'F', 'H', 'B', 'D', 'A', 'C']
```

UTILISER UN PARCOURS EN LARGEUR POUR TROUVER
LA DISTANCE ENTRE DEUX SOMMETS D'UN GRAPHE

Définition : la **distance** entre deux sommets d'un graphe est le nombre minimal d'arêtes (ou d'arc s'il est orienté) pour aller d'un sommet de départ d à un sommet final f.

Remarque : si les sommets ne peuvent pas être reliés, cette distance n'existe pas, donc on notera None dans ce cas…

1) En utilisant le principe du parcours en largeur, écrire la fonction :
parcours_largeur_distance(graphe, s) , qui prend comme argument un graphe et un sommet source s, et qui renvoie sous forme de dictionnaire un parcours en largeur du graphe à partir de s, en la valeur associée à chaque clé est la distance qui sépare ce sommet de s. On suivra le modèle :

```
def parcours_largeur_distance(graphe,s):
    """
    parcours en largeur d'un graphe
    à partir d'un sommet source s
    qui renvoie sous forme de dico
    avec en association le sommet : distance
    """
    parcours = {s : 0}  # le dico du parcours

    return parcours
```

Vérifier les quelques tests ci dessous…

```
print(parcours_largeur_distance(g, "A"))
>>> {'A': 0, 'B': 1, 'C': 1, 'D': 2, 'E': 2, 'F': 3, 'G': 3, 'H': 4}
print(parcours_largeur_distance(g, "E"))
>>> {'E': 0, 'B': 1, 'D': 1, 'F': 1, 'G': 1, 'A': 2, 'C': 2, 'H': 2}
print(parcours_largeur_distance(g, "G"))
>>> {'G': 0, 'E': 1, 'F': 1, 'H': 1, 'B': 2, 'D': 2, 'A': 3, 'C': 3}
```

2) Utiliser la fonction précédente pour écrire la fonction
distance_entre_sommets(graphe, d, f) qui prend comme argument un graphe, un sommet de départ d et un sommet final f, et qui renvoie :
- la distance entre les deux sommets si on peut dans le graphe se rendre de d à f ;
- None si d et f ne peuvent être joints…

Quelques tests :

```
print(distance_entre_sommets(g, 'A', 'H'))
>>> 4
print(distance_entre_sommets(g, 'A', 'I'))
>>> None
```

UTILISER UN PARCOURS EN LARGEUR POUR TROUVER
LE PLUS COURT CHEMIN ENTRE DEUX SOMMETS DANS UN GRAPHE

Le principe du parcours en largeur d'un graphe consiste à lister les sommets atteignables dans l'ordre croissant de distance par rapport à un sommet source. Il peut donc servir à trouver, s'il existe, le chemin le plus court entre deux sommets d'un graphe...

1) En utilisant le principe du parcours en largeur, écrire la fonction **parcours_largeur_provenance(graphe,s)** , qui prend comme un argument un graphe et un sommet source s, et qui renvoie un parcours en largeur du graphe à partir de s sous forme de dictionnaire, où pour chaque sommet on indique comme d'où on vient quand on l'a rencontré pour la première fois.

Le résultat n'est plus une liste mais un dictionnaire sous la forme :

{sommet : provenance}

On suivra le modèle :

```
def parcours_largeur_provenance(graphe,s):
    """
    parcours en largeur d'un graphe
    à partir d'un sommet source s
    qui renvoie sous forme de dico
    avec en association le sommet : distance
    """
    parcours = {s : 0}  # le dico du parcours

    return parcours
```

On pourra vérifier les quelques tests suivants :

```
print(parcours_largeur_provenance(g, "A"))
>>> {'A': None, 'B': 'A', 'C': 'A', 'D': 'B', 'E': 'B', 'F': 'E', 'G':
'E', 'H': 'G'}
print(parcours_largeur_provenance(g, "E"))
>>> {'E': None, 'B': 'E', 'D': 'E', 'F': 'E', 'G': 'E', 'A': 'B', 'C':
'D', 'H': 'G'}
print(parcours_largeur_provenance(g, "G"))
>>> {'G': None, 'E': 'G', 'F': 'G', 'H': 'G', 'B': 'E', 'D': 'E', 'A':
'B', 'C': 'D'}
```

2) Analyser comment utiliser le parcours en largeur sous forme {sommet : provenance} pour trouver le plus court chemin entre deux sommets d'un graphe.

3) Écrire la fonction **plus_court_chemin(graphe, d, f)** qui renvoie la liste des sommets à suivre pour aller selon la plus courte distance de la source d à la destination f.

Cette fonction devra utiliser la fonction **parcours_largeur_provenance** écrite à la question précédente, et renvoyer None si on ne peut pas aller de d à f dans le graphe...

```
print(plus_court_chemin(g, 'A', 'E'))
>>> ['A', 'B', 'E']
```