Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

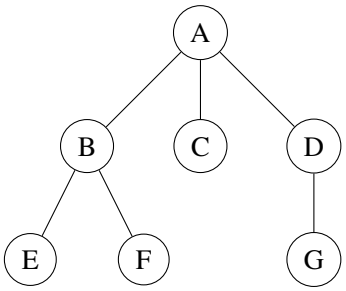
ISN

Algorithmique

Arbre & Graphe en Python

Arbre

Soit l'arbre suivant :



On va le représenter en Python sous forme de listes imbriquées :

```
arbre = ['A',[
            ['B',[
                  ['E',[]],
                  ['F',[]]
                 ]
            ], # fin de B
            ['C', []], # fin de C
            ['D',[
                  ['G',[]]
                 ]
             ] # fin de D
         ]
      ]
```

C-à-d que :

- ⋆ *chaque sous-arbre sera sous forme d'une liste :*
  `[` *sommet* `,` `[` *liste sous arbre* `] ]`
- ⋆ *une feuille est notée sous la forme :*
  `[` *feuille* `,` `[]]`

Une fois cette notation adoptée, les algorithmes de parcours s'écrivent très simplement :

Parcours récursif en profondeur d'abord

```
def ParcoursArbreRecursif(racine):
    sommet = racine[0]
    enfants = racine[1]
    print sommet
    for noeud in enfants:
        ParcoursArbreRecursif(noeud)

ParcoursArbreRecursif(arbre)
```

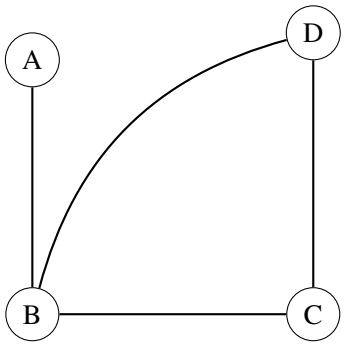
Parcours itératif en largeur

```
def ParcoursArbreIteratif(racine):
    while file_sommets:
        arbre_courant = file_sommets.pop(0)
        sommet_courant = arbre_courant[0]
        enfants_courant = arbre_courant[1]
        print sommet_courant
        for enfant in enfants_courant:
            if enfant:
                file_sommets.append(enfant)

file_sommets = [arbre]
ParcoursArbreIteratif(arbre)
```

Graphes

Soit le graphe suivant :



On va le représenter en Python sous forme d'un dictionnaire Python :

```
1 dico_graph = { 'A' : ['B'], 'B' : ['A','C','D'],
'C':['B','D'], 'D':['B','C']}
```

C-à-d que :

- ⋆ *chaque association du dictionnaire est une paire (sommet, liste d'adjacence) ;*
- ⋆ *les clés sont les noms des sommets ce qui permet un accès direct au travers du dictionnaire.*

Une fois cette notation adoptée, les algorithmes de parcours s'écrivent très simplement :

Parcours récursif en profondeur d'abord

```
1 def TraverserDepuisRecursive(graphe,sommet):
2    sommets_visites.append(sommet)
3    print sommet
4    for voisin in graphe[sommet]:
5       if (not voisin in sommets_visites):
6          TraverserDepuisRecursive(graphe, voisin)

8 sommets_visites = []
9 TraverserDepuisRecursive(dico_graph, 'A')
```

Parcours itératif en largeur

```
1 def TraverserDepuisIterative(graphe,sommet):
2    sommets_a_visiter = [sommet]
3    while sommets_a_visiter:
4       sommet_courant = sommets_a_visiter.pop(0)
5       if (not sommet_courant in sommets_visites):
6          sommets_visites.append(sommet_courant)
7          print sommet_courant
8          sommets_a_visiter.extend(graphe[sommet_courant])

10 sommets_visites = []
11 TraverserDepuisIterative(dico_graph, 'A')
```