10 Parcours d'arbres

July 26, 2024

1 Implémentation à l'aide de listes imbriquées

[20]:
```
def ajoute(li:list, val)-> None:
    """
    Ajoute un élement de valeur val à la fin de la liste chaînée li.
    """
    if len(li) == 0: # liste vide
        li.append(val)
    elif len(li) == 1: # dernier élément
        li.append([val])
    else: # cas général : appel récursif
        ajoute(li[1], val)
```

[21]:
```
li = []
ajoute(li, 1)
```

[22]:
```
li
```

[23]:
```
li = [1, [2, [3]]]
```

[24]:
```
ajoute(li, 4)
```

[25]:
```
print(li)
```

```
[1, [2, [3, [4]]]]
```

[26]:
```
def affiche(li:list, ch='[')->None:
    """
    Affiche la liste chaînée li.
    """
    if len(li) == 0:
        print('[]')
    elif len(li) == 1:
        print(ch + str(li[0]) + ']')
    else:
        affiche(li[1], ch + str(li[0]) + ', ')
```

[27]:
```
affiche(li)
```

```
[1, 2, 3, 4]
```

2 Parcours en profondeur (DFS)

[31]:
```python
def DFS(T:list)->None:
    """
    Affiche les noeuds de l'arbre T dans l'ordre de découverte
    d'un parcours en profondeur.
    """
    print(T[0])
    if len(T) != 1:
        for n in T[1:]:
            DFS(n)
```

[32]:
```python
T = [1, [2, [3], [4]], [5]]
```

[33]:
```python
DFS(T)
```

```
1
2
3
4
5
```

3 Parcours en largeur (BFS)

[10]:
```python
def enfiler(f:list, el)->None:
    """
    Ajoute un élément à la file f.
    """
    f.append(el)
```

[11]:
```python
def defiler(f:list)->int:
    """
    Supprime et retourne le premier élément de la file f.
    """
    return f.pop(0)
```

[12]:
```python
def BFS(T:list)->None:
    """
    Affiche les noeuds de l'arbre T dans l'ordre de découverte
    d'un parcours en largeur.
    """
    f = []
    enfiler(f, T)
    while f != []:
```

```
        T = defiler(f) # traitement du prochain élément de la file
        print(T[0])
        for n in T[1:]: # on met tous les descendants à la file
            enfiler(f, n)
```

[13]:
```
BFS(T)
```

```
1
2
5
3
4
```

4 DFS avec une pile

[14]:
```
def empiler(p:list, el)->None:
    """
    Ajoute l'élément el au sommet de la pile p.
    """
    p.append(el)
```

[15]:
```
def depiler(p:list):
    """
    Supprime et retourne le sommet de la pile p.
    """
    return p.pop()
```

[16]:
```
def DFS(T:list)->None:
    """
    Affiche les noeuds de l'arbre T dans l'ordre de découverte
    d'un parcours en profondeur.
    """
    p = []
    empiler(p, T)
    while p != []:
        T = depiler(p)
        print(T[0])
        for n in reversed(T[1:]): # reversed est nécessaire pour que ce soit le␣
 ↪sous-noeud de gauche en premier
            empiler(p, n)
```

[17]:
```
DFS(T)
```

```
1
2
3
4
```

5

[ ]: