Documentations complètes
Python3 : https://docs.python.org/3/
Nympy : https://docs.scipy.org/doc/
Matplotlib : http://matplotlib.org/
SQL : http://sql.sh/

1 Révisions

Sommaire

1.1 Algorithmique

1.1.1 Autour des nombres premiers

Exercice 1.1. Écrire une fonction `is_prime(n: int)->bool` qui teste la primalité de l'entier `n` et retourne un booléen.

Solution.

Exercice 1.2. Écrire une fonction `next_prime(n: int)->int` qui retourne le plus petit entier premier supérieur à `n`.

Solution.

Exercice 1.3. Écrire une fonction `n_first_prime(n: int)->list[int]` qui prend comme argument un entier `n` et retourne la liste des `n` premiers nombres premiers.

Solution.

Exercice 1.4. Écrire une procédure `n_first_prime_bis(n: int)->None` qui prend comme argument un entier `n` et inscrit dans un fichier texte, dont le nom est `n_first_prime.txt`, la liste des `n` premiers nombres premiers à raison d'un nombre par ligne. Par exemple `n_first_prime_bis(20)` créera un fichier nommé `20_first_prime`.

Solution.

Exercice 1.5. On dispose du fichier `10000_first_prime.txt`, créé grâce à la procédure précédente. Écrire une fonction `selection` qui recherche dans ce fichier tous les nombres premiers de cinq chiffres dont le chiffre des milliers est un 8 et celui des unités un 7, et qui retourne le résultat sous forme d'une liste de nombres.

Solution.

Exercice 1.6. Écrire une fonction `decomposition(n: int)->list` qui retourne la décomposition primaire de l'entier `n`.
Exemple : `decomposition(71944)` retourne `[(2, 3), (17, 1), (23, 2)]`.

Solution.

Exercice 1.7. Utiliser la fonction précédente pour écrire une fonction `nb_of_divisors(n: int)->int` qui retourne le nombre de diviseurs de l'entier `n`. Puis une fonction `sum_of_divisors(n: int)->int`, qui calcule la somme des diviseurs de `n`

Solution.

1.1.2 Algorithmes de recherche

Exercice 1.8. Écrire une fonction `search_word(string: str, word:str)->int` qui retourne la première position d'apparition du mot `word` dans la chaîne `string`, s'il apparaît et `-1` sinon.

Solution.

Exercice 1.9. On suppose que `t` est un tableau de valeurs.
Écrire une fonction `maximum(t: list)->int` qui retourne la première position du maximum des éléments de `t`.
Comment la modifier pour qu'elle renvoie la position du dernier maximum ?

Solution.

Exercice 1.10. On suppose que `t` est un tableau trié de valeurs distinctes.
Écrire une fonction `dicho(t: list,x: int)->int` qui retourne l'unique indice `i`, s'il existe, tel que `t[i]` $\leq$ `x` $<$ `t[i+1]`, et sinon qui retourne `-1`.

Solution.

Exercice 1.11. Écrire une fonction `dichotomy(f,a,b,epsilon)` qui retourne une valeur approchée du zéro de la fonction strictement monotone et continue f sur l'intervalle $[a, b]$, s'il existe et `False` sinon.

Solution.

1.2 Parcours de graphe

Exercice 1.12. On considère le graphe G suivant :

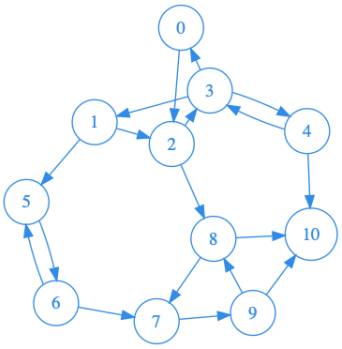
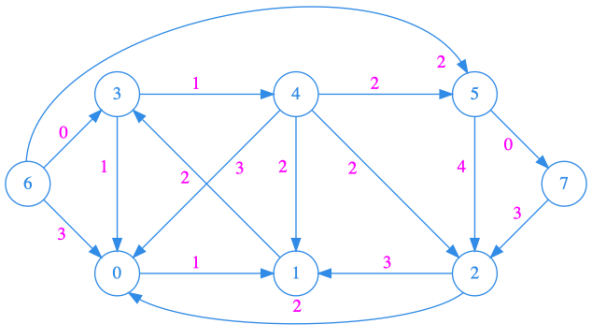


FIGURE 1.1 – Graphe G

1. Faire à la main le parcours profondeur du graphe de la figure 1, dessiner la forêt couvrante du parcours en notant les différents types d'arcs :
   - arc couvrant
   - arc retour
   - arc croisé
   - arc avant
2. Pour chaque sommet donner l'ordre préfixe et suffixe lors du parcours.
3. De combien d'arbres est constituée la forêt ? Que peut-on en déduire ?

Exercice 1.13. On considère le graphe suivant. Appliquer l'algorithme de Djikstra depuis le sommet 6.



1.3 Corrections

Corrigé 1.1 Le code :

```
def is_prime(n: int)->bool:
    test = True
    if n in [0,1] or (n != 2 and n % 2 == 0):
        test = False
    k = 2
    while k*k <= n and test:
        if n % k == 0:
            test = False
        k += 1
    return test
```

Corrigé 1.2 Le code :

```
def next_prime(n: int)->int:
    p = n + 1
    while not is_prime(p):
        p += 1
    return p
```

Corrigé 1.3 Le code :

```
def n_first_prime(n: int)->list[int]:
    first_prime = [2]
    range_prep = range(n-1)
    for k in range_prep:
        first_prime.append(next_prime(first_prime[-1]))
    return first_prime
```

Corrigé 1.4 Le code :

```
 # Une version avec open et close
def n_first_prime_bis(n):
    file = open(str(n) + "_first_prime.txt","w")
    first_prime = n_first_prime(n)
    for p in first_prime:
        file.write(str(p) + "\n")
    file.close()
# Une version avec with
def n_first_prime_bis2(n):
    with open(str(n) + "_first_prime.txt","w") as file:
        first_prime = n_first_prime(n)
        for p in first_prime:
            file.write(str(p) + "\n")
```

Corrigé 1.5 Le code :

```
def selection():
    file = open("10000_first_prime.txt","r")
    selection_prime = []
    for p in file:
        if len(p.rstrip("\n")) == 5 and p[1] == "8" and p[4] == "7":
            selection_prime.append(int(p))

    file.close()
    return selection_prime
```

Corrigé 1.6 Le code :

```
def decomposition(n: int)->list:
    facteurs = []
    p=2
    while n > 1:
        s=0
        while n % p == 0:
            s += 1
            n //= p
        if s > 0:
            facteurs.append((p, s))
        p += 1
    return facteurs
```

Corrigé 1.7 Le code :

```
def nb_of_divisors(n: int)->int:
    res = 1
    for (p, k) in decomposition(n):
        res *= k + 1
    return res

def sum_of_divisors(n: int)->int:
    res = 1
    for (p, k) in decomposition(n):
        res *= (p**(k+1) -1) // (p - 1)
    return res
```

Corrigé 1.8 Le code :

```
def search_word(string,word):
    n = len(string)
    p = len(word)
    if p > n:
        return -1
    else:
        position = 0
        while position + p < n + 1 and sting[position:position + p] != word:
            position += 1
        if position + p == n+1:
            return -1
        else:
            return position
```

Corrigé 1.9 Le code :

```
def maximum(t: list)->int:
    maxi = t[0]
    n = len(t)
    range_prep = range(n)
    for k in range_prep:
        if t[k] > maxi:
            maxi = t[k]
    return maxi
```

Corrigé 1.10 Le code :

```
def dicho(x, t):
    left, right = 0, len(t) - 1
    if x < t[left] or x > t[right]:
        return -1
    else:
        while left + 1 < right:
            k = (left + right) // 2
            if t[k] <= x:
                left = k
            else:
                right = k
        return left
```

Corrigé 1.11 Le code :

```
def dichotomy(f, a, b, epsilon):
    if f(a)*f(b) > 0:
        return False
    else:
        left, right = a, b
        while right - left > epsilon:
            id_film = (left + right)/2
            if f(a)*f(id_film) <= 0:
                right = id_film
            else:
                left = id_film
        return right
```

Corrigé 1.12 1. Les différents arc du parcours :

0 -> 2 : arc couvrant
2 -> 3 : arc couvrant
3 -> 0 : arc retour
3 -> 1 : arc couvrant
1 -> 2 : arc retour
1 -> 5 : arc couvrant
5 -> 6 : arc couvrant
6 -> 5 : arc retour
6 -> 7 : arc couvrant
7 -> 9 : arc couvrant
9 -> 8 : arc couvrant
8 -> 7 : arc retour
8 -> 10 : arc couvrant
9 -> 10 : arc en avant
3 -> 4 : arc couvrant
4 -> 3 : arc retour

4 -> 10 : arc croisé
2 -> 8 : arc en avant

2. Les ordres préfixe et suffixe : [1, 4, 2, 3, 11, 5, 6, 7, 9, 8, 10], [11, 7, 10, 9, 8, 6, 5, 4, 2, 3, 1]
3. Le graphe est connexe.

Corrigé 1.13 Tableau des distances.

Sommet	0	1	2	3	4	5	6	7
dist	1	2	3	0	1	2	0	2
pred	3	0	4	6	3	6	-1	5