# Baccalauréat Général

## Session 2022

Épreuve : **Numérique et sciences informatiques**

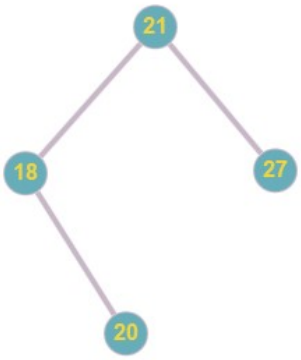
Durée de l'épreuve : 3 heures 30

Coefficient : 16

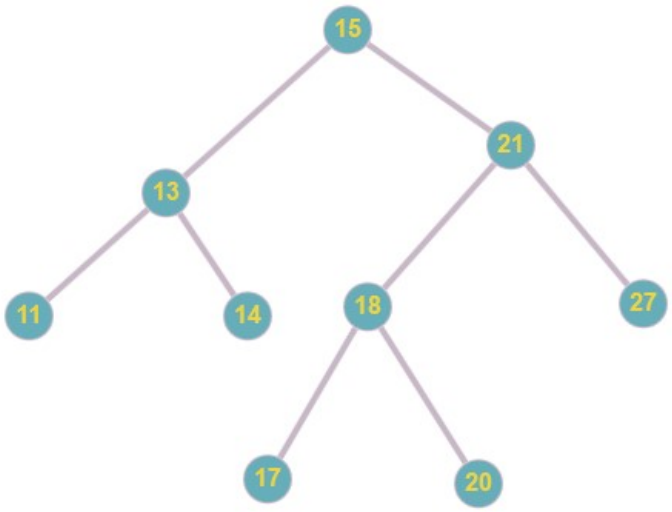
PROPOSITION DE CORRIGÉ

## Exercice I :

### *Question 1*




a) La taille de l'arbre est de 8
b) La hauteur de l'arbre est de 4
c) Le sous-arbre droit du nœud de valeur 15 est :
d) Dans un arbre binaire de recherche les valeurs situées dans le sous–arbre gauche sont inférieures (ou égales) à la valeur de la racine et les nœuds du sous-arbre droit sont supérieures (ou égales) à la valeur de la racine.
e) L'ajout de 17 :15<17 à droite puis 21>17 à gauche puis 18>17 à gauche.




### *Question 2*

a) La bonne réponse est C :
abr=Noeud(Noeud(None,13,None),15,Noeud(None,21,None)
)
b) La fonction ins est :

```
def ins(v, abr):
    if abr is None:
        return Noeud(None,v,None)
    if v>abr.valeur:
        return Noeud(abr.gauche,abr.valeur,ins(v,abr.droit))
    elif v<abr.valeur:
        return Noeud(ins(v,abr.gauche),abr.valeur,abr.droit)
    else:
        return abr
```

## *Question 3*

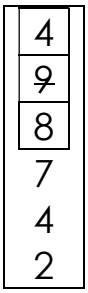
a) Il y aura 17 appels de la fonction nb-sup pour l'arbre de l'exemple 1.
En effet, on appelle une première fois la fonction, puis 2 fois pour chacun des 8 nœuds qui composent l'arbre soit un total de 17.

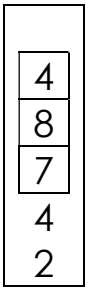
b) Il y a exploration des 2 sous–arbres alors que la valeur plus grande se situera toujours à droite. Pour améliorer la fonction on peut ignorer les sous-arbres gauches qui auront automatiquement des valeurs inférieures à la valeur cherchée.

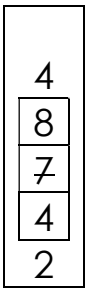
```
    else:
        if abr.valeur<= v:
            return 1+nb_sup(v,abr.gauche) + nb_sup(v,abr.droit)
        else:
            return nb_sup(v,abr.droit)
```

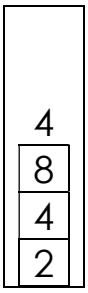
**Exercice II :**

1) a. Premier parcours de la pile :

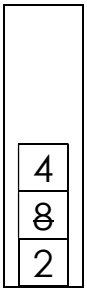


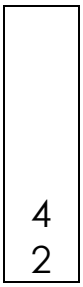




Dernier parcours




b. Il s'agit de la pile B.

2)

```
def reduire_triplet_au_sommet(p):
    a=depiler(p)
    b=depiler(p)
    c=sommet(p)
    if a%2 != c%2  :
        empiler(p,b)
    empiler(p,a)
```

3) a. Pour être réductible la taille de la pile doit être au Minimum de 3.

b.

```
def parcourir_pile_en_reduisant(p):
    q = creer_pile_vide()
    while taille(p) >= 3:
        reduire_triplet_au_sommet(p)
        e = depiler(p)
        empiler(q,e)
    while not est_vide(q):
        f = depiler(q)
        empiler(p,f)
    return p
```

4)

```
def jouer(p):
    q = parcourir_pile_en_reduisant(p)
    if taille(p)==taille(q):
        return(p)
    else:
        return jouer(q)
```

**Exercice III :**

## *Question 1*

a) adresse réseau : 192.1686.1.0
b) adresse de diffusion (broadcast) : 192.168.1.255
c) 254 machines peuvent être connectées
d) une adresse possible pour une nouvelle machine dans ce réseau est : 192.168.1.06

## *Question 2*

a) les chemins possibles sont :

les routeurs sont nommés par leur lettre : le routeur A est nommé A

- ordinateur1-> SW1 -> A-> E -> D -> SW4-> ordinateur 2
- ordinateur1-> SW1 -> A-> C -> F -> D -> SW4-> ordinateur 2
- ordinateur1-> SW1 -> A-> C-> E -> D -> SW4-> ordinateur 2
- ordinateur1-> SW1 -> A-> B -> C -> F -> SW4-> ordinateur 2
- ordinateur1-> SW1 -> A-> B -> C -> E-> D -> SW4-> ordinateur 2
- ordinateur1-> SW1 -> A-> E -> C -> F-> D -> SW4-> ordinateur 2

b) La possibilité de choisir plusieurs routes différentes permet d'éviter une rupture de la communication en cas de panne ou de coupure d'un élément du réseau(rupture d'une connection, panne d'un routeur,…)

## *Question 3*

a) Les trois routeurs B, C et E sont directement connectés au routeur A

| **Routeur A** | |
|---|---|
| Destination | Passe par |
| B | B |
| C | C |
| D | E |
| E | E |

| | |
|---|---|
| F | C |

b) Pour aller de B à D en utilisant les tables :

B -> C -> E -> D

c) Les connexions B-C et A-E sont coupées, les tables deviennent :

| **Routeur A** | |
|---|---|
| Destination | Destination |
| B | B |
| C | C |
| D | C |
| E | C |
| F | C |

| **Routeur B** | |
|---|---|
| Destination | Destination |
| A | A |
| C | A |
| D | A |
| E | A |
| F | A |

| **Routeur C** | |
|---|---|
| Destination | Destination |
| A | A |
| B | A |
| D | E |
| E | E |
| F | F |

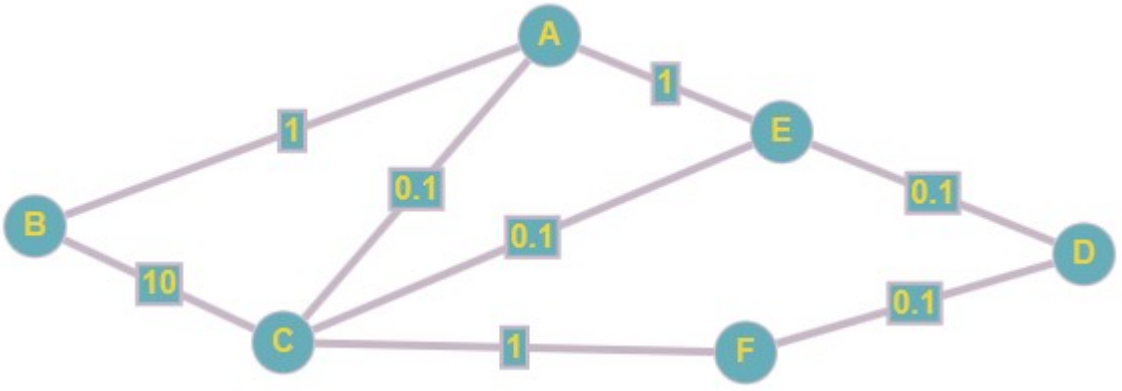
d) Le nopuveau trajet des paquets allant de B à D est le suivant : B -> A->C -> E -> D

## Question 4

*Remarque : on suppose que les liaisons coupées précédemment ont été rétablies.*

a) Liaison Ethernet : coût : $10^8$ / $10^7$ = 10
Fast-Ethernet : coût : $10^8$ / $10^8$ = 1
Fibre : coût : $10^8$ / $10^9$ = 0,1

b) On obtient le schéma suivant :



c) La listes des différents routeurs empruntées est :
   - B-A-E-D pour un coût de 2,1
   - B-A-E-C-F-D pour un coût de 3,2
   - B-A-C-F-D pour un coût de 2,2
   - B-A-C-E-D pour un coût de 1,3
   - B-C-F-D pour un coût de 11,1
   - B-C-E-D pour un coût de 10.2
   - B-C-A-E-D pour un coût de 11.2

d) Dans le protocole de routage OSPF, on cherche à minimiser le coût c'est donc le trajet B-A-C-E-D qui sera choisi

**Exercice IV :**

1) a. La requête permet d'afficher les titres de la table morceaux dont l'id de l'interprète est 4 donc 'Hey Jude'.
   b. **SELECT** nom
   **FROM** interpretes
   **WHERE** pays = 'Angleterre' ;
   c. La requête permet d'afficher les attributs titre et annee des entités de la relation morceaux triés par ordre croissant d'année.
   d. **SELECT COUNT** (*)
   **FROM** morceaux;
   e. **SELECT** titre
   **FROM** morceaux **ORDER BY** titre ;
2) a. L'attribut id_interprete de la table morceaux est clé primaire de la table interpretes. Il est donc clé étrangère de la table morceaux.
   b. morceaux(id_morceau *int* (clé primaire), titre *char*, annee *int*, id_interprete *int* (clé étrangère))
   interpretes(id_interprete *int* (clé primaire), nom *char*, pays *char*)
   c. La requête provoque une erreur car l'entité de clé primaire 1 existe déjà dans la relation interpretes.
3) a. **UPDATE** morceaux
   **SET** annee = 1971
   **WHERE** nom = 'Imagine' ;
   b. **INSERT INTO** interpretes

**VALUES** (6,'The Who', 'Angleterre') ;
c. **INSERT INTO** morceaux
**VALUES** (7, 'My Generation', 1965, 6) ;

4) **SELECT** titre
**FROM** morceau
**JOIN** interprete **ON**
morceau.id_interprete=interpretes.id_interprete
**WHERE** interpretes.pays = 'Etats-Unis';

**Exercice V :**

1)

```
1  cellule = Cellule(True,False,True,True)
```

2)

```
 6          for i in range(hauteur):
 7              ligne = []
 8              for j in range(longueur):
 9                  cellule = Cellule(True,True,True,True)
10                  ligne.append(cellule)
```

3)

```
17          if c1_lig - c2_lig == 1 and c1_col == c2_col:
18              cellule1.murs['N'] = False
19              cellule2.murs['S'] = False
```

4)

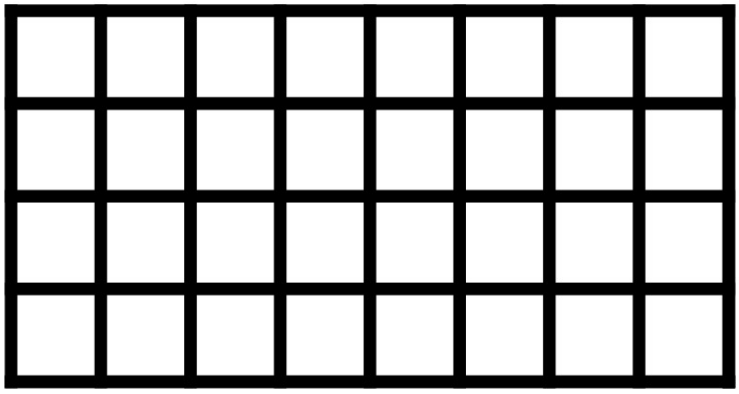
```
20          elif c1_lig == c2_lig and c1_col - c2_col == 1:
21              cellule1.murs['O'] = False
22              cellule2.murs['E'] = False
```

5)

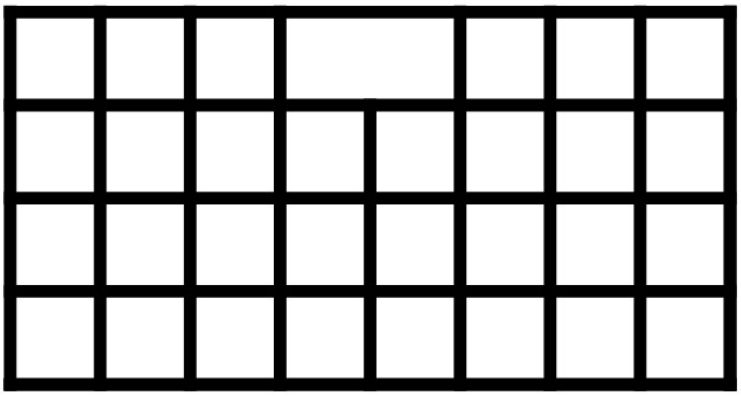
```
25          if haut == 1:
26              for k in range(colonne,colonne+long-1):
27                  self.creer_passage(ligne,k,ligne,k+1)
28          elif long == 1:
29              for k in range(ligne,ligne+haut-1):
30                  self.creer_passage(k,colonne,k+1,colonne)
```

6)

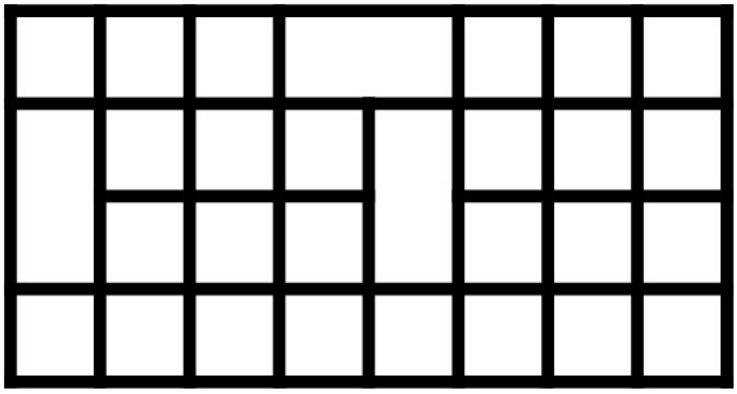
Grille initiale



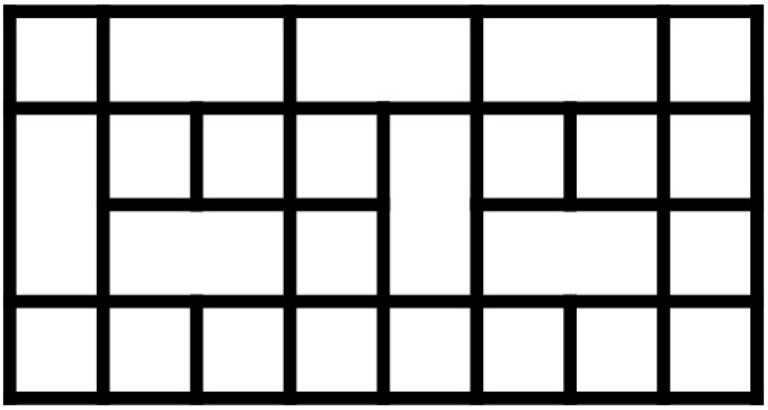
On coupe en deux grilles de 4x4 et on ouvre au nord entre les deux labyrinthes.



Chaque grille de 4x4 est coupée en deux grilles de 4x2 et on ouvre à l'ouest entre les deux labyrinthes obtenus.



Chaque grille de 4x2 est coupée en deux grilles de 2x2 et on ouvre au nord entre les deux labyrinthes obtenus.



Et ainsi de suite jusqu'à obtenir :