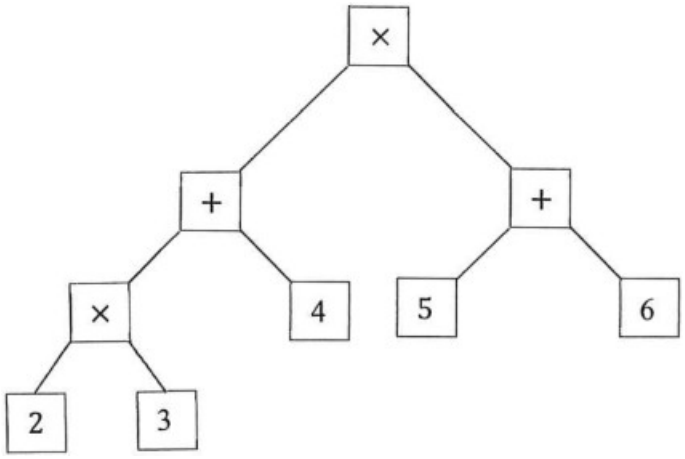
Proposition de correction

Exercice 1

Q1

((2 x 3) + (4 x 5)) x 2

Q2



Q3.a

2 3 x 4 5 x + 2 x

Q3.b

Proposition 3

Q4

```
def evalue(arbre):
    f = suffixe(arbre, File())
    p = Pile()
    while len(f):
        elt = f.defile()
        if elt == '+' or elt == 'x':
            d = p.depile()
            g = p.depile()
            p.empile(ops(elt, d, g))
        else:
            p.empile(elt)
    return p.depile()
```

Exercice 2

Partie A

Q1

Igo → R6 → R4 → serveur

serveur → R4 → R5 → R1 → Baduk

Baduk → R2 → R4 → serveur

serveur → R4 → R6 → Igo

Q2

R3		
Destination	**Passerelle**	**Métrique**
R1, R5, R6	-	1
R2	R1	2
R4	R6	2

Partie B

Q3.a & b

Il faut d'abord peupler la table Tournoi avant la table Partie à cause de la clef étrangère tournoi.

Q4.a

3

Q4.b

totalise le nombre de parties dans lesquelles à joué le joueur d'identifiant 4.

Q5

```
SELECT nom
FROM Tournoi
where pays = 'Japon'
ORDER BY nom ASC
```

Q6

```
SELECT DISTINCT Joueur.nom
FROM Joueur, Partie
WHERE Partie.idjnoir = Joueur.idjoueur
AND Partie.jour = '2016-03-15'
ORDER BY Joueur.nom ASC
```

Q7

renvoie les noms des joueurs qui ont participé au tournoi Meijin

EXERCICE 3

Q1.a

1 → noire

Q1.b

```
def creer_goban_vide(n):
    assert n==9 or n==13 or n==19, "valeur de n non permise"
    return [[0] *n  for _ in range(n)]
```

Q1.c

lève l'exception AssertionError en indiquant « valeur de n non permise »

Q2

```
def libertes_pierre(go, pi, pj):
    libertes = []
    n = len(go)
    for i, j in [(pi+1, pj), (pi-1, pj), (pi, pj+1), (pi, pj-1)]:
        if est_valide(i, p, n) and go[i][p] == 0:
            libertes += [(i, j)]
    return libertes
```

Q3

```
def nb_liberte_chaine(go, chaine):
    n = len(go)
    marquage = [[False for j in range(n)] for i in range(n)]
    nb libertes = 0
    for pos in chaine:
        pi = pos [0]
        pj = pos [1]
        for i, j in libertespierre(go, pi, pj):
            if marquage[i][j] == False:
                marquage[i][j] = True
                nblibertes += 1
    return nblibertes
```

Q4

```
def supprime_prisonniers(go : list, chaine : list):
    """
    @brief    supprime les pierres prisonnières d'un goban
    @param go -- un goban
    @param chaine -- une chaîne de pierres
    @return le nombre de pierres prisonnières
    """
```

```
    prisonniers = 0
    if nb_liberte_chaine(go, chaine) == 0:
        for i, j in chaine:
            go[i][j] = 0
        prisonniers = len(chaine)
    return prisonniers
```

Q5

```
def cherche chaine(go, pi, pj, chaine):
    n = len(go)
    chaine.append((pi, pj))
    couleur = go[pi][pj]
    for i, j in [(pi+1, pj), (pi-1, pj), (pi, pj+1), (pi, pj-1)]:
        if est valide(i, j, n) and go[i][j] == couleur and couleur != 0:
            cherche_ chaine(go, i, j, chaine)
    return chaine
```