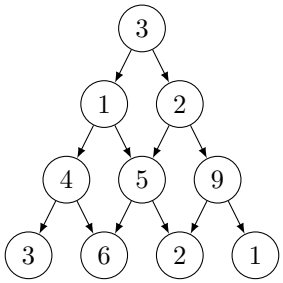
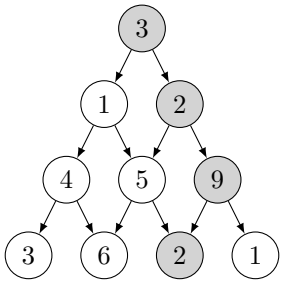
Sujet 0 - 2024 - sujet 2 (corrigé)

Exercice 1 (Listes, récursivité et programmation dynamique)

1. On a la pyramide suivante :



2. Le conduit suivant est un conduit de score de confiance maximal (il est de $3+2+9+2=16$).



3. On a les conduits suivants : $2-5-2$; $2-5-3$; $2-1-3$ et $2-1-9$
4. Pour représenter les conduits dans une pyramide de trois niveaux, on peut utiliser un nombre écrit en binaire : 11 (à gauche puis à gauche) ; 10 (à droite puis à gauche) ; 01 (à gauche puis à droite) et 00 (à droite puis à droite).
Pour représenter les conduit pour une pyramide de n niveaux, on utilisera un nombre de $n-1$ bits. Avec $n-1$ bits, on peut écrire 2^{n-1} nombres, donc on peut donc modéliser 2^{n-1} conduits. Il existe donc 2^{n-1} conduits dans une pyramide de hauteur n.
5. Pour une pyramide de hauteur 5 on a $2^4=16$ possibilités et pour une pyramide de hauteur 6 on a $2^5=32$ possibilités. A chaque fois que l'on augmente la hauteur d'une unité, on double le nombre de conduits possibles (croissance exponentielle). Il va donc être très rapidement impossible de dénombrer toutes les possibilités de conduits.
6. La fonction suivante convient :

```
def score_max(i, j, p):
    if i == len(p) - 1:
        return p[len(p) - 1][j]
    else :
        return p[i][j] + max(score_max(i+1,j,p), score_max(i+1,j+1,p))
```

7. La fonction suivante convient :

```
def pyramide_nulle(n):
    return [[0]*i for i in range(1,n+1)]

# ou

def pyramide_nulle(n):
    tab = []
    for i in range(1,n+1):
        t = []
        for j in range(i):
            t.append(0)
        tab.append(t)
    return tab
```

8. La fonction suivante convient :

```
def prog_dyn(p):
    n = len(p)
    s = pyramide_nulle(n)
    # remplissage du dernier niveau
    for j in range(n) :
        s[n-1][j] = p[n-1][j]
    for i in range(n-2, -1,-1) :
        for j in range(i+1) :
            s[i][j] = p[i][j]+max(s[i+1][j], s[i+1][j+1])
    # renvoie du score maximal
    return s[0][0]
```

9. Le coût est quadratique car on peut observer la présence de deux boucles imbriquées dans la fonction `prog_dyn`, ce qui est souvent le marqueur d'un coût quadratique. Si on veut être plus rigoureux, on peut aussi remarquer que pour une pyramide de hauteur n, on a un nombre d'opérations élémentaires égal à

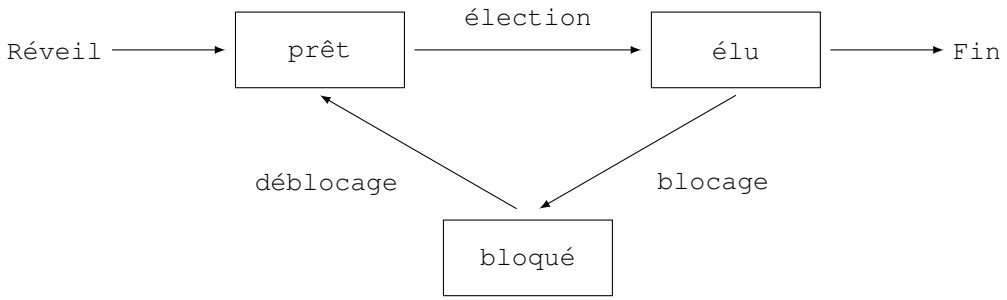
$$1 + 2 + 3 + ... + n - 1 + n = \frac{n(n+1)}{2},$$

soit, en utilisant la notation O, une complexité en $O(n^2)$.

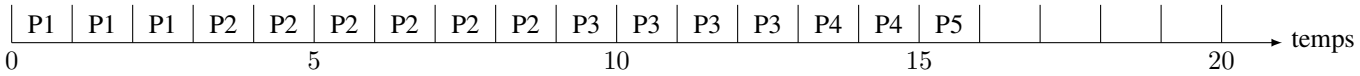
10. On pourrait de nouveau utiliser la programmation dynamique, mais cette fois en utilisant l'approche descendante (top-down). Cela nous permettrait d'éviter des appels récursifs quand les calculs ont déjà été faits.

Exercice 2 (Système d'exploitation, piles, files et processus)

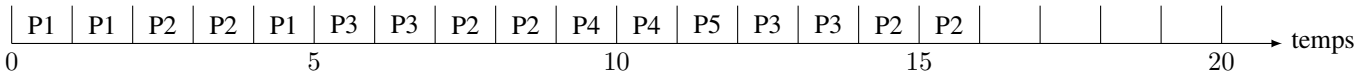
1. Les logiciels peuvent être librement étudiés (code source accessible), copiés, modifiés, diffusés, ce qui n'est pas le cas des logiciels propriétaires
2. Un système d'exploitation est un ensemble de logiciels permettant de faire l'interface entre le matériel (CPU, disque dur...) et l'utilisateur. Le système d'exploitation permet aux autres logiciels d'utiliser les ressources proposées par le matériel sans avoir à se préoccuper en détail de ce matériel.
3. On a le chemin suivant : `/home/elsa/documents/boulot/rapport.odt`
4. On a le chemin suivant : `../max/images/photos_vac/photo_1.jpg`
5. - Le répertoire `documents` contient le fichier `fiche.ods` et le répertoire `boulot`
   - Le répertoire `boulot` contient le fichier `fiche.ods` et le fichier `rapport.odt`.
6. On a le schéma suivant :



7. Un processus peut passer de l'état élu à l'état bloqué, par exemple si à un moment donné de son exécution, il a besoin d'une ressource externe et que cette ressource n'est pas disponible.
8. La pile est une structure de données linéaire de type LIFO.
9. On a le schéma suivant :



10. On a le schéma suivant :



11. - Soient deux processus P1 et P2 et deux ressources R1 et R2. Initialement, les deux ressources sont « libres » (utilisées par aucun processus).
    - Le processus P1 commence son exécution (état élu) : il demande la ressource R1. Il obtient satisfaction puisque R1 est libre, passe donc dans l'état « prêt ».
    - Pendant ce temps, le système a passé P2 à l'état élu : P2 commence son exécution et demande la ressource R2. Celle-ci étant libre, P2 l'obtient et passe donc à l'état « prêt ».
    - Pendant ce temps, P1 est passé à l'état « élu ». Au cours de son exécution, il a besoin de la ressource R2. Celle-ci n'étant pas libre (R2 a été attribuée à P2), P1 passe à l'état « bloqué ».
    - Le système passe alors P2 à l'état « élu », au cours de son exécution, P2 a besoin de la ressource R1 qui n'est pas disponible (toujours attribuée à P1). P2 passe alors à l'état « bloqué ».
    - P1 et P2 sont bien en situation d'interblocage.

Exercice 3 (Dictionnaires, POO et bases de données)

Partie A.

1. `a` vaut `[10, 8, 9, 9, 8, 10, 6, 7, 8, 8]` et `b` vaut `'Fondation'`.
2. La fonction suivante convient :

```
def titre_livre(dico, id_livre):
    for i in range(len(dico['id'])):
        if dico['id'][i] == id_livre :
            return dico['titre'][i]
    return None
```

3. La fonction suivante convient :

```
def note_maxi(dico):
    n_max = 0
    for n in dico['note']:
        if n > n_max:
            n_max = n
    return n_max
```

4. La fonction suivante convient :

```
def livres_note(dico, n):
    tab_titres = []
    for i in range(len(dico['note'])):
        if n == dico['note'][i]:
            tab_titres.append(dico['titre'][i])
    return tab_titres
```

5. La fonction suivante convient :

```
def livre_note_maxi(dico):
    n_max = note_maxi(dico)
    return livres_note(dico, n_max)
```

Partie B.

6. ⋆ Un attribut de la classe `Livre` : `self.id`
   ⋆ Une méthode de la classe `Livre` : `get_id`
7. La fonction suivante convient :

```
def get_note(self):
    return self.note
```

8. Le programme suivant convient :

```
b_r = Livre(8, 'Blade Runner', 'K.Dick', 1968,8)
b = Bibliotheque()
b.ajout_livre(b_r)
```

9. Le code suivant convient :

```
def titre_livre(self, id_livre):
    for livre in self.liste_livre :
        if livre.get_id() == id_livre :
            return livre.get_titre()
    return None
```

Partie C.

10. La clé primaire doit être unique. Un auteur pouvant écrire plusieurs livres, l'attribut `auteur` ne peut donc pas être une clé primaire.

11. On obtient le résultat suivant :

Ubik
Blade Runner

12. La requête suivante convient :

```
SELECT titre FROM livres WHERE auteur = Asimov AND ann_pub > 1950;
```

13. La requête suivante convient :

```
UPDATE livres SET note = 10 WHERE titre = Ubik;
```

14. Manipuler deux tables permet d'éviter la duplication des données. En particulier, elle évite de réécrire l'ensemble des informations personnelles d'un auteur pour chacun de ses livres.
15. `id_auteur` est une clé étrangère. Elle permet d'établir une relation entre la table `auteurs` et la table `livres` (jointure). A chaque valeur de l'attribut `id_auteur` correspond un auteur dans la table `auteurs`.
16. La requête suivante convient :

```
SELECT nom, prenom
FROM auteurs
JOIN livre ON id_auteur = auteurs.id
WHERE ann_pub > 1960;
```

17. Cette requête permet d'obtenir le titre des livres écrits par des auteurs ayant moins de 30 ans au moment de leur publication
18. Ce projet ne respecte pas la protection de la vie privée : divulgation des données personnelles (date de naissance, numéro de téléphone, etc.)