BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2026

NUMÉRIQUE ET SCIENCES INFORMATIQUES

ÉPREUVE DU MERCREDI 17 JUIN 2026

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 15 pages numérotées de 1/15 à 15/15.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

Exercice 1 (6 points)

Cet exercice porte sur les bases de données relationnelles, les requêtes SQL et la programmation objet. Cet exercice comporte deux parties indépendantes.

Partie A

Dans cette partie, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT`, `FROM`, `WHERE` (avec les opérateurs logiques `AND` et `OR`) et `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT` et `DELETE` ;
- faire des alias avec `AS` ;
- affiner les recherches à l'aide de `DISTINCT` et `ORDER BY` ;
- réaliser des agrégations à l'aide de `COUNT`, `AVG`, `MAX`, `MIN` et `SUM`.

Une agence de location d'espaces de travail (coworking) a créé une base de données comprenant les relations suivantes :

- `salle` (`id_salle`, `intitule`, `type`, `nb_places`, `tarif_jour`)
- `reservation` (`id`, `id_client`, `id_salle`, `date`, `duree`)
- `client` (`id_client`, `societe`, `num_tel`)

On donne ci-dessous des extraits des relations précédentes :

salle				
id_salle	intitule	type	nb_places	tarif_jour
1	B1	Bureau	3	140
2	Ravel	Amphi	90	2300
3	RA	Réunion	10	350
4	B2	Bureau	4	155
5	RB	Réunion	8	300

reservation				
id	id_client	id_salle	date	duree
1	2	3	2024-03-24	1
2	1	1	2023-12-17	1
3	2	3	2024-04-16	2
4	1	4	2024-02-05	1

client		
id_client	societe	num_tel
1	Dupont&co.	0812121212
2	FleursJ	0950505050

L'attribut `id_salle` de la relation `salle`, l'attribut `id` de la relation `reservation` et l'attribut `id_client` de la relation `client` sont des clés primaires.

1. Donner, en justifiant, les clés étrangères de la relation `reservation`.

2. Donner le résultat de la requête suivante, quand on l'applique aux extraits donnés des tables.

```
SELECT intitule FROM salle WHERE nb_places > 5
    ORDER BY tarif_jour;
```

3. Écrire une requête permettant d'obtenir la liste sans doublon des `id_salle` des salles réservées depuis janvier 2024. On pourra utiliser le comparateur `>` pour les dates.

Un nouveau client de nom de société 'EN' et de numéro de téléphone '0144624055' a réservé le bureau d'identifiant 4 pour une journée le 12 juin 2024.

4. Écrire une suite de requêtes permettant d'ajouter ce client et sa réservation. On attribuera la valeur 3 à `id_client` et la valeur 5 à `id`.

La salle 'RAVEL' a été modifiée, elle peut maintenant contenir 95 personnes.

5. Écrire la requête permettant d'enregistrer cette modification dans la base de données précédente.

6. Écrire une requête permettant d'obtenir le temps total de réservation de la salle d'identifiant 6.

7. Écrire une requête permettant d'obtenir la liste sans doublon des noms de société ayant réservé une salle de réunion.

Partie B

On implémente maintenant ces données sous la forme d'une classe en Python donnée ci-dessous. Pour simplifier le code, on considère qu'il n'y a plus de date de réservation mais simplement un état de disponibilité. Lorsqu'une salle est disponible, l'attribut `occupant` prend pour valeur la chaîne vide.

```
 1 class Salle:
 2     def __init__(self, intitule, type_salle, nb_places,
 3                  tarif_jour):
 4         self.intitule = intitule
 5         self.type_salle = type_salle
 6         self.nb_places = nb_places
 7         self.tarif = tarif_jour
 8         self.dispo = True
 9         self.occupant = ""
10     def reserver(self, client):
11         if self.dispo:
12             self.dispo = False
13             self.occupant = client
14             return self.tarif
15         return False
```

8. Donner un exemple d'attribut et un exemple de méthode de la classe `Salle`.

9. Écrire la méthode `liberer` qui modifie les attributs d'une salle de manière à la rendre disponible et sans occupant (chaîne de caractères vide).

Le code suivant crée et modifie une liste de salles :

```
1 S1 = Salle("B1", "Bureau", 3, 140)
2 S2 = Salle("RAVEL", "Amphi", 90, 2300)
3 S3 = Salle("RA", "Réunion", 10, 350)
4 S4 = Salle("B2", "Bureau", 4, 155)
5 S5 = Salle("RB", "Réunion", 8, 300)
6 S1.reserver("Dupont&co")
7 S2.reserver("FleursJ")
8 liste_salles = [S1,S2,S3,S4,S5]
```

10. Recopier et compléter les lignes 3, 4 et 5 du code de la fonction `salles_dispos` qui prend en paramètre une liste de salles et renvoie la liste des intitulés de celles qui sont disponibles.

```
1 def salles_dispos(liste_salles):
2     res = []
3     for ...:
4         if ...:
5             res.append(...)
6     return res
```

On crée maintenant une classe `Seminaire` ayant pour attributs une liste de salles réservées, un prix total et un nom de client.

```
 1 class Seminaire:
 2     def __init__(self, client):
 3         self.liste_salles = []
 4         self.prix_total = 0
 5         self.client = client
 6     def ajouter_reservation(self, salle):
 7         if salle.dispo:
 8             ...
 9             ...
10             return True
11         return False
```

11. Recopier et compléter la méthode `ajouter_reservation` qui ajoute `salle` à la liste des salles réservées, puis réserve cette salle en modifiant le prix total. Elle renvoie `True` si la salle a pu être réservée car elle était libre, et `False` sinon.

Exercice 2 (6 points)

Cet exercice porte sur les protocoles de routage et les graphes.

Partie A

Un réseau informatique, composé de six routeurs et des liens reliant ces routeurs, est représenté ci-dessous. Le coût d'un lien est inversement proportionnel au débit de la liaison. Ces coûts sont indiqués au-dessus des liaisons et les adresses IP des sous-réseaux sont indiquées en dessous. Le "/24" situé à la suite des adresses IP des sous-réseaux signifie que les identifiants réseaux sont codés sur 24 bits.

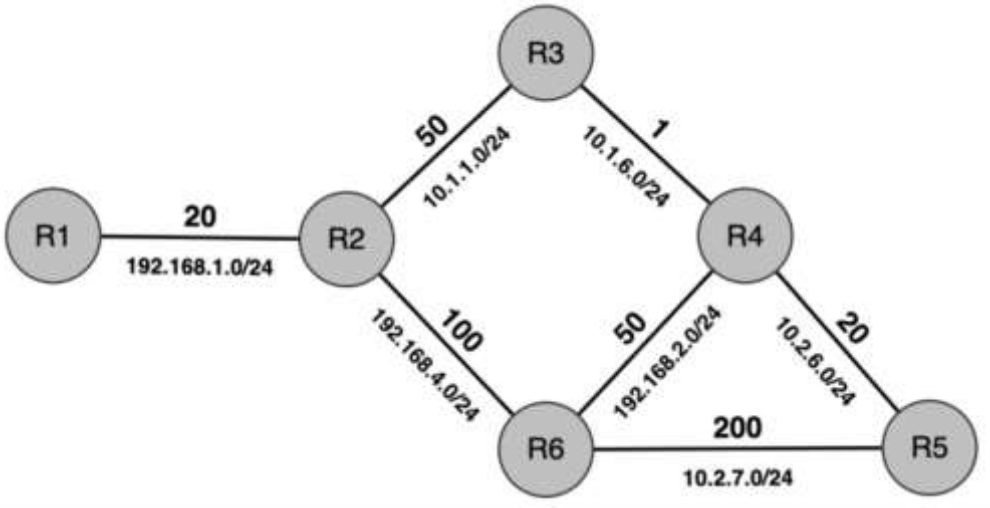


Figure 1. Graphe représentant le réseau

Le tableau ci-dessous donne les adresses IP des routeurs selon leurs liaisons aux différents sous-réseaux.

Adresse du sous-réseau	Adresses IP des routeurs			
192.168.1.0/24	R1	192.168.1.1	R2	192.168.1.4
10.1.1.0/24	R2	10.1.1.1	R3	10.1.1.2
192.168.4.0/24	R2	192.168.4.2	R6	192.168.4.1
10.1.6.0/24	R3	10.1.6.1	R4	10.1.6.2
10.2.6.0/24	R4	10.2.6.1	R5	10.2.6.2
192.168.2.0/24	R4	192.168.2.2	R6	192.168.2.1
10.2.7.0/24	R5	10.2.7.3	R6	10.2.7.1

1. Expliquer pourquoi le routeur R2 ne peut pas être associé à l'adresse IP 192.168.2.3 dans le sous-réseau qui le lie au routeur R1.

Le premier protocole de routage utilisé est le protocole RIP (*Routing Information Protocol*). Dans celui-ci, la métrique de la table de routage correspond au nombre de routeurs à traverser pour atteindre la destination.

2. Recopier et compléter la table de routage du routeur R4 ci-dessous avec les adresses IP des destinations et des passerelles (ou routeurs suivants) selon le protocole RIP suite à l'échange avec l'ensemble des routeurs. Une destination correspond à un sous-réseau.

Table de routage de R4		
destination	passerelle	distance
10.1.6.0/24	connecté	0
10.2.6.0/24	connecté	0
192.168.2.0/24	connecté	0
10.1.1.0/24	10.1.6.1	1

Les routeurs R3 et R4 sont reliés par la technologie 5G dont le débit est de 10 Gbit/s.

3. Calculer le débit entre les routeurs R1 et R2. On rappelle que le coût d'un lien est inversement proportionnel au débit de la liaison.

4. Indiquer le chemin à suivre pour transmettre des données de R1 à R5 selon le protocole RIP.

La liaison R2-R6 tombe en panne.

5. Indiquer le chemin à suivre pour transmettre des données de R1 à R5, toujours selon le protocole RIP.

La liaison R2-R6 est rétablie. Le protocole de routage OSPF (*Open Shortest Path First*) est alors utilisé. Dans ce protocole, la métrique de la table de routage correspond à la somme des coûts des liaisons traversées.

6. Indiquer le chemin à suivre pour transmettre des données de R1 à R5 selon le protocole OSPF et comparer avec le chemin obtenu précédemment selon le protocole RIP.

Le réseau évolue et de nouveaux routeurs sont implantés. Pour des raisons de lisibilité, le suffixe /24 a été omis. Le tableau ci-après récapitule les mises à jour des adresses IP des routeurs.

Routeur	Adresses IP			
R1	192.168.1.1	114.156.4.3	178.115.8.2	
R2	192.168.1.4	10.1.1.1	192.168.4.2	
R3	10.1.1.2	10.1.6.1	192.162.2.2	
R4	10.1.6.2	10.2.6.1	192.168.2.2	
R5	10.2.6.2	10.2.7.3	12.14.8.2	
R6	192.168.2.1	10.2.7.1	10.7.2.2	192.168.4.1
R7	114.156.4.2	12.14.8.1	192.162.2.8	
R8	10.7.2.3	178.115.8.3		

7. Représenter le réseau informatique mis à jour à l'aide d'un graphe.

Partie B

Un réseau routier, composé de six villes, est représenté ci-dessous.

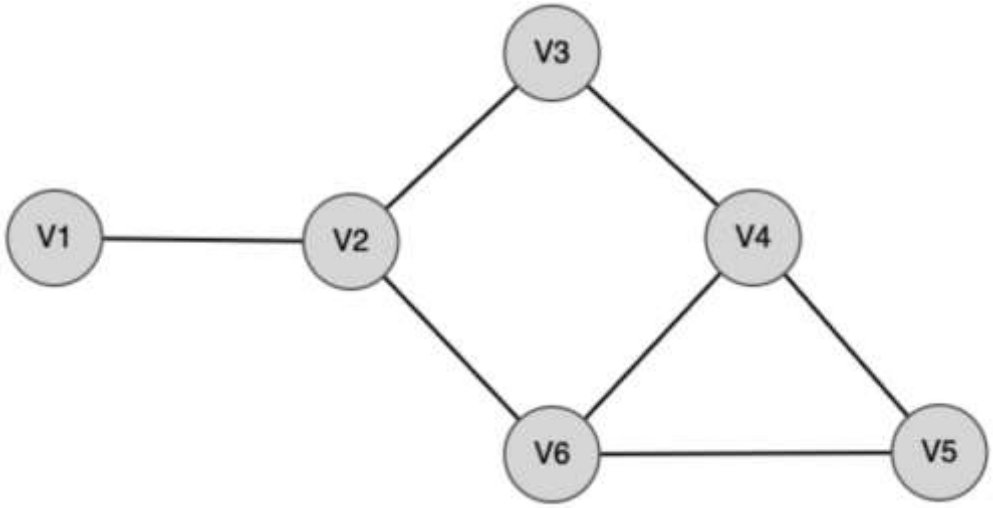


Figure 2. Schéma du réseau routier

Un automobiliste souhaite partir de la ville V1 pour rejoindre la ville V5. Le GPS lui propose deux itinéraires :

- l'itinéraire A minimisant le temps de parcours ;
- l'itinéraire B minimisant le nombre de villes traversées.

8. Pour ces deux itinéraires, indiquer celui qui correspond au protocole RIP et celui qui correspond au protocole OSPF d'un réseau informatique.

Le réseau, modélisé par un graphe, est implémenté dans la suite par un dictionnaire de listes d'adjacence où chaque sommet du graphe est associé à la liste de ses voisins. Le réseau routier de la figure 2 est donc implémenté de la façon suivante :

```
route = {
    'V1': ['V2'],
    'V2': ['V1', 'V3', 'V6'],
    'V3': ['V2', 'V4'],
    'V4': ['V3', 'V5', 'V6'],
    'V5': ['V4', 'V6'],
    'V6': ['V2', 'V4', 'V5']
}
```

9. Écrire la liste des sommets, dans l'ordre de leur visite, lors du parcours en largeur du graphe du réseau routier de la figure 2 depuis le sommet `'V3'`.

En supposant que toutes les arêtes ont le même poids, la fonction `itineraire_court`, ci-après, renvoie le plus court chemin entre deux sommets d'un graphe.

```
def itineraire_court(graphe, depart, arrivee):
    deja_visites = []
    a_visiter = []
    a_visiter.append(depart)
    precedent = {depart: None}
    while a_visiter != []:
        sommet = a_visiter.pop(0)
        if sommet not in deja_visites:
            deja_visites.append(sommet)
        for voisin in graphe[sommet]:
            if voisin not in deja_visites:
                a_visiter.append(voisin)
                precedent[voisin] = sommet
            if voisin == arrivee:
                a_visiter = []
                break
    if arrivee not in precedent:
        return []
    chemin = []
    ville = arrivee
    while ville != None:
        chemin.append(ville)
        ville = precedent[ville]
    chemin.reverse()
    return chemin
```

L'instruction `break` permet d'interrompre l'exécution d'une boucle et d'exécuter la suite du script. La méthode `reverse` des listes en Python inverse l'ordre des éléments d'une liste sans en créer une nouvelle. On rappelle que la méthode `pop` des listes Python enlève et renvoie l'élément de la liste situé à l'index indiqué.

Par exemple, `ma_liste.pop(4)` enlève et renvoie l'élément de la liste `ma_liste` situé à l'index 4.

L'instruction suivante est saisie puis exécutée.

```
>>> itineraire_court(route, 'V1', 'V5')
```

10. Compléter l'évolution du contenu de la variable `precedent` à chaque tour de la boucle `while`, donné ci-dessous, suite à l'exécution de l'instruction saisie.

    Tour n°1 : `{'V1': None}`
    Tour n°2 : `{'V1': None, 'V2': 'V1'}`
    Tour n°3 : `{'V1': None, 'V2': 'V1', 'V3': 'V2'}`
    …

11. Écrire ce que renvoie la fonction `itineraire_court` suite à l'exécution de l'instruction saisie.

Exercice 3 (8 points)

Cet exercice traite de tableaux, de dictionnaires, de récursivité et de programmation dynamique.

Une scierie possède un stock de planches. Chacune des planches du stock a une longueur entière (en mètre) comprise entre 1 m et 10 m et un prix entier (en euro) qui dépend de cette longueur. Le propriétaire de la scierie souhaite estimer la valeur marchande de son stock.

On utilise dans tout le problème la variable globale `prix` du type `list` telle que pour tout entier `n` compris entre 1 et 10, `prix[n]` est le prix de vente en euro d'une planche de longueur `n` (en mètre).

On suppose ici que la liste `prix` est `prix = [0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30]`. On remarque que l'élément d'indice 0 de `prix` est fixé à 0 mais ne sera jamais utilisé car une planche ne peut mesurer 0 m.

Partie A

Pour gérer son stock, la scierie utilise un dictionnaire `effectif` qui associe à chaque longueur de planche présente dans le stock son effectif, c'est-à-dire le nombre de planches de cette longueur dans le stock.

Par exemple :

- s'il y a 3 planches de 2 m dans le stock alors `effectif[2]` vaut 3 ;
- s'il n'y a aucune planche de 3 m dans le stock alors `effectif` n'a pas de clé égale à 3.

1. Donner une instruction Python qui permet de déclarer et d'initialiser le dictionnaire `effectif` qui correspond à un stock constitué de trois planches de 1 m, de quatre planches de 2 m et d'une planche de 4 m.

2. Donner une expression en Python qui calcule la valeur totale en euro de l'ensemble des planches de 10 m d'un stock représenté par le dictionnaire `effectif` contenant au moins une planche de 10 m. On rappelle que la variable `prix` contient l'ensemble des prix de vente en euro des planches.

3. Donner une expression en Python qui vaut `True` si le stock représenté par le dictionnaire `effectif` contient des planches de 5 m et `False` sinon.

On souhaite disposer d'une fonction `valeur_stock` en Python qui prend en paramètre le dictionnaire `effectif` et qui renvoie la valeur, en euro, du stock représenté par `effectif` si l'on vend toutes les planches stockées dans la scierie selon les prix de la variable globale `prix`.

4. Recopier et compléter les lignes 2, 3 et 4 du code de la fonction `valeur_stock` ci-après.

```
1 def valeur_stock(effectif):
2   valeur_totale = ...
3   for longueur in ... :
4     valeur_totale = valeur_totale + ...
5   return valeur_totale
```

Un camion apporte de nouvelles planches à la scierie. Le stock de la scierie est donc modifié.

5. Écrire une fonction `mise_a_jour` qui prend en paramètre l'effectif du stock déjà présent dans la scierie nommé `effectif` et l'effectif du stock contenu dans le camion nommé `autre_effectif` et qui met à jour le dictionnaire `effectif`.

   Par exemple,

```
effectif = {1: 3, 5: 5}
# le camion apporte deux planches de longueur 1
# et deux planches de longueur 2 :
autre_effectif = {1: 2, 2: 2}
mise_a_jour(effectif, autre_effectif)
# L'expression effectif == {1: 5, 2: 2, 5: 5}
# vaut alors True.
```

Un client achète une planche et on a déjà vérifié qu'il y avait bien une planche de cette longueur dans le stock de la scierie. On souhaite actualiser le dictionnaire représentant le stock à l'aide d'une fonction Python `vendre`.

6. Écrire une fonction `vendre` qui prend en paramètre l'effectif du stock nommé `effectif` et une longueur de planche nommée `longueur` et qui modifie le dictionnaire `effectif` en conséquence. Si le stock de planches de cette longueur devient nul, l'association devra être supprimée du dictionnaire `effectif`. On pourra utiliser `del d[c]` qui supprime la clé `c` du dictionnaire `d`.

   Par exemple :

```
effectif = {1: 3, 5: 1}
longueur = 1
vendre(effectif, longueur)
# L'expression effectif == {1: 2, 5: 1} vaut alors True
longueur = 5
vendre(effectif, longueur)
# L'expression effectif == {1: 2} vaut alors True
```

Partie B

On réalise qu'on peut parfois augmenter la valeur d'une planche en la découpant en planches plus petites (toujours de longueurs entières) et en vendant séparément les morceaux obtenus.

On cherche donc à déterminer la valeur maximale que peut rapporter une planche à l'entreprise, éventuellement en la découpant.

On remarque que :

- on ne peut pas découper une planche de 1 m : sa valeur maximale est donc 1€ ;
- une planche 2 m peut être vendue entière pour 5 € ou en deux planches de 1 m pour $1 + 1 = 2$ € : il vaut donc mieux ne pas la découper ;
- une planche de 3 m peut être vendue entière pour 8 € ou en une planche de 2 m et une planche de 1 m pour $1 + 5 = 6$ € ou en trois planches de 1 m pour $1 + 1 + 1 = 3$ € : il vaut donc mieux ne pas la découper.

7. Déterminer, en justifiant, la valeur maximale que peut rapporter une planche de 4 m.

On note `valmax(n)` la valeur maximale d'une planche de longueur `n` mètres pour `n` compris entre 1 m et 10 m en optimisant sa découpe et on propose la formule récursive ci-dessous.

- Si $n \leq 3$, `valmax(n) = prix[n]`.
- Sinon, `valmax(n) = max(prix[n], prix[n-1] + valmax(1), prix[n-2] + valmax(2), ... , prix[1] + valmax(n-1))`.

8. Expliquer le cas d'arrêt de cette formule récursive.

9. Recopier et compléter les quatre lignes ci-dessous.

   - `valmax(1) = prix[...] = ...`
   - `valmax(2) = ... = ...`
   - `valmax(3) = ... = ...`
   - `valmax(4) = max(prix[...], prix[...] + ... , prix[...] + ... , prix[...] + ...) = ...`

10. Expliquer pourquoi, si $n \geq 4$, on a bien : `valmax(n) = max(prix[n], prix[n-1] + valmax(1), prix[n-2] + valmax(2), ... , prix[1] + valmax(n-1))`

On souhaite disposer de la fonction `valmax` qui prend en paramètre un entier `n` et qui renvoie la valeur maximale obtenue pour une planche de longueur `n` selon les prix de la variable globale `prix`.

11. Recopier et compléter les lignes 2 à 7 du code de la fonction `valmax` ci-après.

```
1 def valmax(n):
2   if ... :
3     ...
4   else:
5     maximum = prix[n]
6     for longueur in range(1, ...):
7       maximum = max(maximum, ... + ...)
8     return maximum
```

12. Expliquer pourquoi l'appel de `valmax(7)` effectuera deux appels de `valmax(5)`.

Partie C

Pour éviter les calculs redondants, on décide d'utiliser la programmation dynamique et de mémoriser les résultats de ces calculs au fur et à mesure qu'on les obtient dans un tableau `memo` (type `list` en Python).

On déclare et initialise la variable globale `memo` comme suit.

```
memo = [0, 1, 5, 8, -1, -1, -1, -1, -1, -1, -1]
```

Il faut maintenant coder la fonction `valmax_dynamique` de paramètre `n`, la longueur de la planche (en mètre) dont on cherche la valeur maximale, de façon à ce que :

- si `memo[n]` ne vaut pas -1 alors la valeur maximale a déjà été mémorisée et `memo[n]` vaut cette valeur maximale donc on la renvoie ;
- sinon :
  - on calcule la valeur maximale en utilisant les appels récursifs ;
  - on affecte à `memo[n]` la valeur maximale obtenue.

13. Recopier et compléter les lignes 2 à 8 du code de la fonction ci-après.

```
1 def valmax_dynamique(n):
2   if ... :
3     ...
4   else:
5     maximum = prix[n]
6     for longueur in range(1, ...):
7       maximum = max(maximum, ... + ...)
```

```
8       memo[n] ...
9   return maximum
```

14. Donner en justifiant le nombre d'appels récursifs effectués par l'appel de `valmax_dynamique(6)`, après l'exécution de `valmax_dynamique(5)`.

15. Expliquer ce qui se serait passé si on avait initialisé la variable `memo` avec uniquement des valeurs égales à -1.

On rappelle que, pour gérer son stock, la scierie utilise un dictionnaire `effectif` qui associe à chaque longueur de planche en mètre présente dans le stock son effectif dans le stock. De plus on suppose que la variable `memo` vient d'être initialisée comme suit.

```
memo = [0, 1, 5, 8, -1, -1, -1, -1, -1, -1, -1]
```

16. Écrire une fonction `valeur_max_stock` qui prend en paramètre un dictionnaire `effectif` et qui renvoie la valeur maximale en euro que peut rapporter le stock à l'entreprise en optimisant le découpage de chaque planche afin qu'elle rapporte le maximum.