

## Devoir surveillé d'informatique

### ⚠ Consignes

- Les programmes demandés doivent être écrits en C ou en OCaml suivant l'exercice. Dans le cas du C, on suppose que les bibliothèques standards usuelles (`<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`) sont déjà importées.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veuillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

### □ Exercice 1 : Base de données et SQL

🎓 CAPES NSI 2021, épreuve 1

On s'intéresse dans cette partie à un site Internet d'échange de supports de cours entre enseignants de MP2I/MPI. Chaque personne désirant proposer ou récupérer du contenu doit commencer par se créer un compte sur ce site et peut ensuite accéder à du contenu ou en proposer.

Ce site repose sur une base de données contenant en particulier une table, nommée **ressources**. Elle possède un enregistrement par document téléversé sur le site. Ses attributs sont :

- **id**, un identifiant numérique, unique pour chaque ressource ;
- **owner**, le pseudo de la personne ayant créé la ressource ;
- **annee**, l'année de publication de la ressource ;
- **titre**, une chaîne de caractères décrivant la ressource ;
- **type**, chaîne de caractères pouvant être cours, ds, tp ou td.

Voici un extrait de cette table :

| id  | owner     | annee | titre                     | type  |
|-----|-----------|-------|---------------------------|-------|
| 4   | dknuth    | 2020  | Machine à décalage        | cours |
| 13  | alovelace | 2022  | Intelligence artificielle | td    |
| ... | ...       | ...   | ...                       | ...   |

- Q1**– Écrire une requête SQL permettant de connaître tous les titres des ressources déposées par « jclarke » classées par année de publication croissante.
- Q2**– Écrire une requête SQL permettant de connaître le nombre total de ressources de type cours présentes sur le site.
- Q3**– Que fait la requête suivante : ?

```
SELECT R.owner
FROM Ressources AS R
WHERE R.type = 'td'
GROUP BY R.owner
ORDER BY COUNT(*) DESC
LIMIT 3
```

Cette base de données contient également une table **utilisateurs** qui contient les informations sur les utilisateurs du site. Elle possède un enregistrement par utilisateur. Ses attributs sont :

- **nom**, le nom de l'utilisateur (clé primaire) ;
- **mdp**, le mot de passe de l'utilisateur.
- **email**, l'adresse email de l'utilisateur.
- **naissance**, l'année de naissance de l'utilisateur.

Voici un extrait de cette table :

| nom    | mdp       | email              | naissance |
|--------|-----------|--------------------|-----------|
| dknuth | chepas123 | dknuth@bigboss.com | 1938      |
| ...    | ...       | ...                | ...       |

L'attribut **owner** de la table **ressources** est une clé étrangère qui référence l'attribut **nom** de la table **utilisateurs**.

**Q4**– Ecrire une requête SQL permettant de lister toutes les ressources déposées par des utilisateurs nés après 2000.

**Q5**– Ecrire une requête SQL permettant de lister tous les utilisateurs n'ayant déposé aucune ressource.

□ **Exercice 2 : Logique et calcul des propositions**

CCP 2015

De nombreux travaux sont réalisés en Intelligence Artificielle pour construire un programme qui imite le raisonnement humain et soit capable de réussir le test de Turing, c'est-à-dire qu'il ne puisse pas être distingué d'un être humain dans une conversation à l'aveugle. Vous êtes chargé(e)s de vérifier la correction des réponses données par un tel programme lors des tests de bon fonctionnement. Dans le scénario de test considéré, le comportement attendu est le respect de la règle suivante : pour chaque question, le programme répondra par trois affirmations et parmi ces affirmations une seule sera correcte (et donc les deux autres fausses).

Nous noterons  $A_1$ ,  $A_2$  et  $A_3$  les propositions associées aux affirmations effectuées par le programme.

**Q6**– Représenter le comportement attendu sous la forme d'une formule du calcul des propositions qui dépend de  $A_1$ ,  $A_2$  et  $A_3$ .

■ **Partie I : Premier cas**

Vous demandez au programme : « *Quels éléments doivent contenir les aliments que je dois consommer pour préserver ma santé ?* »

Il répond les affirmations suivantes :

$A_1$  : Consommez au moins des aliments qui contiennent des glucides mais pas de lipides !

$A_2$  : Si vous consommez des aliments qui contiennent des glucides, alors ne consommez pas d'aliments qui contiennent des lipides !

$A_3$  : Ne consommez aucun aliment qui contient des lipides

Nous noterons  $G$ , respectivement  $L$ , les variables propositionnelles qui correspondent au fait de consommer des aliments qui contiennent des glucides, respectivement des lipides.

**Q7**– Exprimer  $A_1$ ,  $A_2$  et  $A_3$  sous la forme de formules du calcul des propositions. Ces formules peuvent dépendre des variables  $G$  et  $L$ .

**Q8**– En utilisant le calcul des propositions (résolution avec les formules de De Morgan), déterminer ce que doivent contenir les aliments que vous devrez consommer pour préserver votre santé.

■ **Partie II : Deuxième cas**

Vous demandez au programme : « *Quelles activités dois-je pratiquer si je veux préserver ma santé ?* »

Suite à une coupure de courant, la dernière information est interrompue.

$A_1$  : Si vous faites des activités sportives alors prenez du repos !

$A_2$  : Si vous ne faites pas d'activités intellectuelles, alors ne prenez pas de repos !

$A_3$  : Prenez du repos ou faites des activités ... !

Nous noterons  $S$ ,  $I$  et  $R$  les variables propositionnelles qui correspondent au fait de faire des activités sportives, des activités intellectuelles, et de prendre du repos.

**Q9**– Exprimer  $A_1$ ,  $A_2$  et  $A_3$  sous la forme de formules du calcul des propositions. Ces formules peuvent dépendre des variables  $S$ ,  $I$  et  $R$ . Pour  $A_3$ , on indiquera des pointillés pour signifier que la réponse a été interrompue.

**Q10**– En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer quelle(s) activités vous devez pratiquer pour préserver votre santé. L'information  $A_3$  étant incomplète ou pourra envisager les deux cas suivants : elle se termine par "sportives" ou "intellectuelles" et réaliser la table de vérité dans chacun des cas.

□ **Exercice 3** : Implémentation d'un tableau associatif

CCMP 2025, MPI

Les fonctions demandées dans cet exercice doivent être écrites en langage C.

**Définitions** : Soit  $A$  un arbre binaire de recherche, on notera avec une lettre minuscule (par exemple  $x$  ou  $y$ ) un noeud de  $A$  et avec une lettre majuscule (par exemple  $T_1$ ,  $T_2$  ou  $T_3$ ) un sous arbre de  $A$ . Soit  $x$  un noeud, on notera  $x_g$  et  $x_d$  respectivement le fils gauche et le fils droit de  $x$ . La hauteur d'un arbre  $x$  sera noté  $h(x)$  et pour arbre de racine  $x$ , on définit la valeur d'équilibre de  $x$  noté  $e(x)$  comme la différence entre la hauteur du sous arbre gauche et celle du sous arbre droit, c'est à dire :

$$e(x) = h(x_g) - h(x_d)$$

On définit un noeud d'un arbre binaire de recherche par la structure suivante :

```

1 struct abrNoeud_s
2 {
3     char *clef;
4     int valeur;
5     struct abrNoeud_s *fils_gauche;
6     struct abrNoeud_s *fils_droit;
7 };
8 typedef struct abrNoeud_s abrNoeud;
```

On utilisera la valeur du champ `clef` pour ordonner les noeuds de l'arbre en utilisant l'ordre alphabétique.

- Q11**– Nous nous intéressons aux valeurs du champ `clef` de la structure ci-dessus. Expliquer comment une chaîne de caractères est représentée en mémoire en C. Combien d'octets sont nécessaires pour représenter la chaîne *arbre* par exemple ?
- Q12**– Sachant que le code ASCII du caractère 'a' est 97 quelle est sa représentation en binaire ? Quelle est sa représentation en hexadécimal ?
- Q13**– Sans utiliser les fonctions de la librairie `string.h`, écrire une fonction en C, `copie_chaine` qui renvoie une copie de la chaîne de caractères passée en paramètre. La signature de la fonction est :
- ```
char* copie_chaine(char* s).
```
- On supposera que la chaîne de caractères passée en paramètre n'est jamais NULL.
- Q14**– Ecrire une fonction en C, `abr_creer_noeud` qui crée un noeud à partir d'une clef, d'une valeur et de deux sous arbres et qui renvoie ce nouveau noeud. La signature de la fonction est :
- ```
abrNoeud* abr_creer_noeud(char* clef, int valeur, abrNoeud* fg, abrNoeud* fd).
```
- Le champ `clef` du noeud créé doit contenir une *copie* de la chaîne de caractères passée en paramètre. Par contre, il n'est pas nécessaire de dupliquer les sous-arbres passés en paramètre.
- Q15**– Ecrire une fonction, `hauteur` qui renvoie la hauteur de l'arbre binaire de recherche passé en paramètre. La signature de la fonction est : `int hauteur(abrNoeud* a)`.
- Q16**– Ecrire une fonction, `equilibre` qui renvoie la valeur d'équilibre de l'arbre binaire de recherche passé en paramètre. La signature de la fonction est : `int equilibre(abrNoeud* a)`.
- Q17**– Pour un arbre de taille  $N$ , quelle est la complexité de la fonction `equilibre` ? Sans proposer de code, expliquer quelles seraient les modifications à apporter à la structure de données et/ou à l'algorithme pour améliorer cette complexité
- Q18**– Ecrire une fonction récursive `abr_rechercher` qui, étant donné un arbre binaire de recherche dont le noeud racine est passé en paramètre ainsi qu'une clef, renvoie la valeur associée à cette dernière. On considérera que la clef donnée est toujours dans l'arbre binaire de recherche et on rappelle qu'en C l'opérateur `==` permet de comparer des entiers ou des pointeurs mais pas de comparer le contenu de deux chaînes de caractères. La signature de la fonction est : `int abr_rechercher(abrNoeud* a, char* clef)`.
- Q19**– Une autre implémentation possible d'un tableau associatif utilise une table de hachage. Expliquer succinctement le principe de cette implémentation.
- Q20**– Ecrire une fonction `hachage` qui prend en argument une chaîne de caractères et renvoie la somme des codes ASCII de cette chaîne pondérée par la position de chaque caractère. C'est à dire que le code ASCII du premier caractère est multiplié par 1, le code du second par deux et ainsi de suite. Par exemple le

hachage de la chaîne "abc" renvoie  $1 \times 97 + 2 \times 98 + 3 \times 99 = 590$  (on rappelle que le code ASCII de 'a' est 97, celui de 'b' 98, ...). La signature de la fonction est `int hachage(char *s)`.

- Q21**– Expliquer rapidement pourquoi il serait plus pertinent d'utiliser un type entier non signé pour la valeur renvoyée par la fonction `hachage`.
- Q22**– Donner pour la fonction de hachage précédente, deux chaînes de caractères de longueur 2 et écrites uniquement avec des minuscules qui entrent en collision.

□ **Exercice 4** : *Problème du sac à dos*

Les fonctions demandées dans cet exercice doivent être écrites en OCaml et on *s'interdit* dans cet exercice l'utilisation des aspects impératifs du langage (boucles et références notamment).

On dispose d'un sac à dos pouvant contenir un poids maximal noté  $P$  et de  $n$  objets ayant chacun un poids  $(p_i)_{0 \leq i \leq n-1}$  et une valeur  $(v_i)_{0 \leq i \leq n-1}$ . On cherche à remplir le sac à dos de manière à maximiser la valeur totale des objets contenus dans le sac sans dépasser le poids maximal  $P$ . Par exemple, si on dispose des objets suivants :

- un objet de poids  $p_0 = 4$  et de valeur  $v_0 = 20$ ,
- un objet de poids  $p_1 = 5$  et de valeur  $v_1 = 28$ ,
- un objet de poids  $p_2 = 6$  et de valeur  $v_2 = 36$ ,
- un objet de poids  $p_3 = 7$  et de valeur  $v_3 = 50$ ,

et qu'on suppose que le poids maximal du sac est 10 alors un choix possible serait de prendre l'objet 3, aucun autre objet ne rentre alors dans le sac et la valeur du sac est de 50 avec un poids de 7. Une autre possibilité plus intéressante serait de choisir les objets 0 et 2, la valeur totale serait alors de 56 et le poids du sac de 10.

Dans toute la suite de l'exercice on supposera que les poids et les valeurs des objets ainsi que le poids maximal du sac sont des entiers (type `int`), et que donc un objet être représenté par un couple (poids, valeurs) `int*int` et une liste d'objets par le type `(int*int) list` de OCaml. Et on définit le type suivant afin de représenter un problème du sac à dos :

```
1 type pbsac =
2   {
3     pmax : int;
4     objets : (int*int) list
5   }
```

Ainsi, le problème donné en exemple ci-dessus correspond à

```
1 let ex =
2   {
3     pmax = 10;
4     objets = [(7, 35); (4, 18); (5, 20)];
5   }
```

On propose de représenter un choix d'objets par une liste de booléens. Si le  $i$ -ème élément de la liste vaut `true` alors l'objet  $i$  est choisi, sinon l'objet  $i$  n'est pas choisi. Par exemple, pour les objets précédents, le choix de prendre uniquement l'objet 3 serait représenté par la liste `[false; false; false; true]` et le choix de prendre les objets 0 et 2 serait représenté par la liste `[true; false; true; false]`.

■ **Partie I** : Approche par recherche exhaustive

La recherche exhaustive consiste à énumérer tous les choix possibles d'objet et à calculer la valeur ainsi que le poids pour chaque choix. Parmi toutes les possibilités, on retient alors le choix qui maximise la valeur du sac sans dépasser le poids maximal.

- Q23**– Justifier rapidement que le nombre possible de choix d'objet est  $2^n$ .
- Q24**– Ecrire une fonction `poids_valeur : (int*int) list -> bool list -> int*int` qui prend en arguments, une liste d'objets ainsi qu'un choix d'objet (sous la forme indiquée ci-dessus) et qui renvoie le

poids et la valeur du sac correspondant à ce choix. Par exemple avec le problème du sac à dos `ex` définie ci-dessus, `poids_valeur(ex.objets, [true; false; true; false])` doit renvoyer (10, 56). On supposera que la liste d'objets et le choix d'objet sont de même taille.

- Q25**– Sans le programmer, donner un algorithme permettant de générer tous les choix possibles d'objets.
- Q26**– Donner la complexité d'une méthode qui pour chaque choix possible d'objet calculerait la valeur du sac ainsi que son poids et renverrait le choix optimal.


### ■ Partie II : Stratégie gloutonne

On considère la stratégie gloutonne suivante : on trie les objets par ordre décroissant de leur rapport valeur/poids et on les prend dans cet ordre jusqu'à ce que le poids maximal soit atteint.

- Q27**– Montrer à l'aide d'un contre exemple que cette stratégie ne permet pas toujours d'obtenir la valeur maximale.
- Q28**– La documentation de la fonction `List.sort : ('a -> 'a -> int) -> 'a list -> 'a list` de la librairie standard d'OCaml indique que cette fonction trie une liste d'éléments en utilisant l'ordre défini par la fonction de comparaison passée en paramètre. Cette dernière renvoie 0 lorsque les deux éléments sont égaux, un entier positif lorsque le premier élément est plus grand et un entier négatif sinon. Cette documentation indique aussi que l'algorithme utilisé est le *tri fusion*. En déduire une fonction `tri : (int*int) list -> (int*int) list` qui trie une liste d'objets par ordre décroissant de leur rapport valeur/poids.
- Q29**– Ecrire une fonction `glouton : pbsac -> int`, qui prend en arguments un problème du sac à dos et qui renvoie la valeur maximale que l'on peut obtenir en appliquant la stratégie gloutonne.
- Q30**– Donner la complexité de la fonction `glouton` en fonction du nombre d'objets  $n$ .

### ■ Partie III : Approche par programmation dynamique

On propose de résoudre le problème du sac à dos par programmation dynamique. Pour tout  $i \in \llbracket 0; n \rrbracket$ , on note  $V(i, p)$  la valeur maximale que l'on peut obtenir avec les objets à partir de celui d'indice  $i$  et un poids maximal  $p$ . Par exemple s'il y a 5 objets (numérotés de 0 à 4),  $v(2, 10)$  est le poids maximal atteint pour un sac de poids maximal 10 en ne considérant que les objets 2, 3 et 4.

- Q31**– Donner  $V(i, 0)$  pour  $i \in \llbracket 0; n - 1 \rrbracket$  et  $V(n, p)$  pour  $p \in \llbracket 0; P \rrbracket$ .
- Q32**– Donner les relations de récurrence liant  $v(i, p)$  à  $v(i + 1, p)$  et  $v(i + 1, p - p_i)$ .  
 Indication : on pourra considérer deux cas, celui où l'objet  $i$  est pris et celui où il ne l'est (dans ce cas on a nécessairement  $p \geq p_i$ ).
- Q33**– Écrire une fonction récursive `dynamique : pbsac -> int` qui prend en arguments un problème du sac à dos et qui renvoie la valeur maximale que l'on peut obtenir en appliquant la stratégie de programmation dynamique.
- Q34**– Sans la programmer, expliquer une stratégie permettant de mémoriser les résultats déjà calculés de la fonction `dynamique` afin d'éviter de les recalculer.

### □ Exercice 5 : Tableaux autoréférents

 ORAUX CCINP 2024, MPI

Les fonctions demandées dans cet exercice doivent être écrites en langage C.

On dit qu'un tableau `tab` de taille  $n$  est *autoréférent* si pour tout entier  $i \in \llbracket 0; n - 1 \rrbracket$ , `tab[i]` est le nombre d'occurrences de `i` dans `tab`. Par exemple, le tableau `ex={ 1, 2, 1, 0 }` est autoréférent, en effet :

- `ex[0]` = 1 et 0 apparaît bien une fois dans le tableau
- `ex[1]` = 2 et 1 apparaît bien deux fois dans le tableau
- `ex[2]` = 1 et 2 apparaît bien une fois dans le tableau
- `ex[3]` = 0 et 3 n'apparaît pas dans le tableau

L'exercice traite de la recherche par retour sur trace d'un tableau autoréférent de taille donnée.

- Q35**– Justifier rapidement que dans un tableau autoréférent de taille  $n$ , chaque valeur doit être comprise entre 0 et  $n$  et que la somme des éléments du tableau doit être égale à  $n$ .
- Q36**– Montrer que pour  $n \in \llbracket 1; 3 \rrbracket$ , il n'existe aucun tableau auto référent de taille  $n$ .
- Q37**– Déterminer un autre tableau autoréférent de taille 4 que celui donné en exemple.

**Q38–** Soit  $n \geq 7$ , on définit le tableau **tab** de taille **n** par :

- **tab**.(0) =  $n-4$
- **tab**.(1) = 2
- **tab**.(2) = 1
- **tab**.( $n-4$ ) = 1
- **tab**.(*i*) = 0 si  $i \notin \{0, 1, 2, n-4\}$

Prouver que **tab** est autoréférent

**Q39–** Ecrire une fonction **est\_autoreferent** qui prend en argument un tableau d'entiers (et sa taille) et renvoie **true** si ce tableau est autoréférent et **false** sinon. On attend une complexité en  $\mathcal{O}(n)$  où  $n$  est la taille du tableau c'est à dire qu'on veut parcourir une seule fois le tableau. On cherche maintenant à construire un tableau autoréférent de taille **n** en utilisant un algorithme de recherche par retour sur trace (*backtracking*) qui valide une solution partielle construite jusqu'à un index **i** donné, on propose pour cela la fonction récursive suivante qui utilise la fonction de validation partielle **valide** qui sera écrite plus loin et qui essaye d'affecter une valeur valide à l'indice **i** puis de poursuivre la construction du tableau :

```

1  bool autoreferent(int tab[], int n, int i)
2  {
3      if (i == n)
4      {
5          return est_autoreferent(tab,n);
6      }
7      for (int k = ..... )
8      {
9          tab[i] = k;
10         if (valide(tab, n, i))
11         {
12             if (autoreferent(.....))
13             {
14                 return .....;
15             }
16         }
17     }
18     return .....;
19 }
```

**Q40–** Compléter les lignes 7, 12, 14, et 18 de la fonction précédente.

**Q41–** On sait que la somme des éléments d'un tableau autoréférent de taille **n** est **n**. D'autre part, si après avoir affecté la case d'indice **i**, il y déjà strictement plus d'occurrences d'une valeur **k** comprise entre 0 et **i** que la valeur de **t[k]** alors la solution partielle n'est pas valide. En déduire une fonction **valide** de signature **bool valide(int tab[], int n, int i)** qui prend en argument un tableau d'entiers, sa taille et un indice **i** et qui renvoie **true** si la ce tableau peut-être complété en un tableau autoréférent.

**Q42–** Quelle est la réponse à la grande question, la vie, l'univers et le reste ?

⊗ Indication : la réponse est dans la question !