DS 6

Informatique MP2I

Julien Reichert

Questions de cours

Question de cours 1 : Expliquer la différence entre `>` et `|` en bash. Donner un exemple d'utilisation de chaque.

Question de cours 2 : Suivant le cas, que puis-je faire avec un fichier qui a pour code de permissions 754 ?

Question de cours 3 : Énoncer et prouver le théorème d'interpolation de Craig.

Question de cours 4 : Définir la notion de forme normale conjonctive et donner un exemple.

Question de cours 5 : Prouver que $(\sim)$ est un système complet, en notant $\sim$ l'opérateur « NON ET ».

Question de cours 6 : Soient φ et ψ deux formules. La loi de Peirce donne un lien entre $(\varphi \Rightarrow \psi) \Rightarrow \varphi$ et φ. Lequel (avec preuve) ?

Question de cours 7 : Définir les notions de clé primaire et de clé étrangère, exemples à l'appui.

Question de cours 8 : À quelle condition les requêtes `SELECT COUNT(*) FROM tbl` et `SELECT COUNT(id) FROM tbl` peuvent-elles donner des résultats différents ?

Question de cours 9 : Commenter les requêtes ci-après, avec un schéma intuitif. Dans l'idée, on souhaite savoir **tous les identifiants** qui maximisent une valeur. Si aucune requête ne convient, proposer une version correcte.

```
SELECT id FROM tbl WHERE MAX(valeur);

SELECT id FROM tbl HAVING MAX(valeur);

SELECT id FROM tbl WHERE valeur = MAX(valeur);

SELECT id FROM tbl ORDER BY valeur DESC LIMIT 1;
```

Question de cours 10 : Décoder la séquence suivante qui a été compressée (en fournissant ici les codes de sortie sous forme de liste en OCaml) avec l'algorithme de Lempel-Ziv-Welch. On rappelle que l'espace a pour code `32` et que les capitales ont pour code les nombres de `65` à `90` en respectant l'ordre alphabétique. Voir l'annexe pour plus de détails.

```
[78; 69; 86; 69; 82; 32; 71; 79; 78; 78; 65; 261; 73; 258; 32; 89; 79; 85; 32; 85; 80; 32;
256; 258; 260; 262; 264; 266; 76; 69; 84; 270; 272; 32; 68; 263]
```

Question de cours 11 : On donne ci-après la fonction d'insertion dans un arbre bicolore en OCaml (le type est celui des ABR, le booléen indique si le nœud est noir). Dessiner (inutile d'écrire l'expression associée) les sept arbres obtenus par l'insertion des entiers de 0 à 6 dans l'ordre et à partir de l'arbre vide, en mettant en lumière le résultat final de chaque insertion et en faisant apparaître les étapes intermédiaires utiles (avec commentaires)

```
type 'a abr = Vide | Noeud of 'a abr * 'a * 'a abr;;

let rotation_droite a = match a with
| Vide -> failwith "Arbre vide"
| Noeud(Vide, _, _) -> failwith "Fils gauche vide"
| Noeud(Noeud(a1, n2, a2), n1, a3) -> Noeud(a1, n2, Noeud(a2, n1, a3));;

let rotation_gauche a = match a with
| Vide -> failwith "Arbre vide"
| Noeud(_, _, Vide) -> failwith "Fils droit vide"
| Noeud(a1, n2, Noeud(a2, n1, a3)) -> Noeud(Noeud(a1, n2, a2), n1, a3);;

let corrige_gauche a = match a with
| Vide -> failwith "Ce cas ne se produit pas"
| Noeud(Noeud(gg, (gx, false), gd), (x, true), Noeud(dg, (dx, false), dd)) ->
  Noeud(Noeud(gg, (gx, true), gd), (x, false), Noeud(dg, (dx, true), dd))
| Noeud(Noeud(gg, (gx, false), Noeud(gdg, (gdx, false), gdd)), (x, true), d) ->
  let g2 = rotation_gauche (Noeud(gg, (gx, false), Noeud(gdg, (gdx, false), gdd))) in
  begin
    match rotation_droite (Noeud(g2, (x, true), d)) with
    | Noeud(Noeud(gbisg, (gbisx, couleur), gbisd), (xbis, rouge), Noeud(dbisg, (dbisx, noir), dbisd)) ->
      Noeud(Noeud(gbisg, (gbisx, couleur), gbisd), (xbis, true), Noeud(dbisg, (dbisx, false), dbisd))
    | _ -> failwith "Ce cas ne se produit pas"
  end
| Noeud(Noeud(Noeud(ggg, (ggx, false), ggd), (gx, false), gd), (x, true), d) ->
  begin
    match rotation_droite a with
    | Noeud(Noeud(gbisg, (gbisx, couleur), gbisd), (xbis, rouge), Noeud(dbisg, (dbisx, noir), dbisd)) ->
      Noeud(Noeud(gbisg, (gbisx, couleur), gbisd), (xbis, true), Noeud(dbisg, (dbisx, false), dbisd))
    | _ -> failwith "Ce cas ne se produit pas"
  end
| _ -> a;;

let corrige_droite a = match a with
| Vide -> failwith "Ce cas ne se produit pas"
| Noeud(Noeud(gg, (gx, false), gd), (x, true), Noeud(dg, (dx, false), dd)) ->
  Noeud(Noeud(gg, (gx, true), gd), (x, false), Noeud(dg, (dx, true), dd))
| Noeud(g, (x, true), Noeud(Noeud(dgg, (dgx, false), dgd), (dx, false), dd)) ->
  let d2 = rotation_droite (Noeud(Noeud(dgg, (dgx, false), dgd), (dx, false), dd)) in
  begin
    match rotation_gauche (Noeud(g, (x, true), d2)) with
    | Noeud(Noeud(gbisg, (gbisx, noir), gbisd), (xbis, rouge), Noeud(dbisg, (dbisx, couleur), dbisd)) ->
      Noeud(Noeud(gbisg, (gbisx, false), gbisd), (xbis, true), Noeud(dbisg, (dbisx, couleur), dbisd))
    | _ -> failwith "Ce cas ne se produit pas"
  end
| Noeud(g, (x, true), Noeud(dg, (dx, false), Noeud(ddg, (ddx, false), ddd))) ->
  begin
    match rotation_gauche a with
    | Noeud(Noeud(gbisg, (gbisx, noir), gbisd), (xbis, rouge), Noeud(dbisg, (dbisx, couleur), dbisd)) ->
      Noeud(Noeud(gbisg, (gbisx, false), gbisd), (xbis, true), Noeud(dbisg, (dbisx, couleur), dbisd))
    | _ -> failwith "Ce cas ne se produit pas"
  end
| _ -> a;;

let rec insertion_bicolore_aux elt a = match a with
| Vide -> Noeud(Vide, (elt, false), Vide)
| Noeud(g, (x, c), d) when elt < x -> corrige_gauche(Noeud(insertion_bicolore_aux elt g, (x, c), d))
| Noeud(g, (x, c), d) -> corrige_droite(Noeud(g, (x, c), insertion_bicolore_aux elt d));;

let insertion_bicolore elt a = match insertion_bicolore_aux elt a with
| Vide -> failwith "Cas impossible"
| Noeud(g, (x, _), d) -> Noeud(g, (x, true), d);;
```

Exercices

Sauf mention explicite du contraire, le langage est au choix **parmi OCaml et C** pour tous les exercices demandant d'écrire un programme. Les recommandations ne correspondent qu'au choix fait pour l'élaboration du corrigé. Cependant, **si un exercice est traité dans les deux langages, aucune des deux versions ne sera corrigée**. Pour changer d'avis alors qu'une partie sur un des langages a déjà été commencée au propre, cette partie sera à barrer de manière bien visible et sera ignorée.

Consigne : tout exercice prenant en argument ou renvoyant une « séquence » prendra en argument / renverra un tableau en C (ainsi que sa taille, comme d'habitude), mais en OCaml le choix est laissé entre tableau et liste.

On rappelle la documentation du module `Hashtbl` :

- `Hashtbl.create n` crée une table de hachage avec `n` places pour commencer, mais en adaptant si besoin (donc on devine `n` sans qu'il n'y ait de risque si l'estimation est mauvaise) ;
- `Hashtbl.add th cle valeur` ajoute une association à la table de hachage, en masquant une éventuelle clé déjà existante (l'autre valeur sera de nouveau accessible en cas de retrait de ce qui l'a masqué) ;
- `Hashtbl.find th cle` détermine la valeur associée à la clé dans la table de hachage, en déclenchant l'erreur `Not_found` si la clé est absente ;
- `Hashtbl.mem th cle` détermine si la clé est présente dans la table de hachage ;
- `Hashtbl.remove th cle` retire une occurrence de la clé dans la table de hachage s'il y en a une (sinon la fonction n'a pas d'effet) ;
- `Hashtbl.replace th cle valeur` remplace la valeur associée à la clé dans la table de hachage par une nouvelle valeur (une éventuelle valeur masquée n'est pas impactée) en ajoutant la clé si elle n'y était pas encore.
- `Hashtbl.find_opt th cle` agit comme la fonction `find`, mais retourne une option, donc si la clé est absente aucune exception n'est levée, c'est simplement le cas où `None` est renvoyé ;
- `Hashtbl.iter f th` appelle la fonction fournie, prenant des clés et des valeurs (dans cet ordre) en argument, à tous les éléments de la table de hachage, sans contrôle sur l'ordre, sachant que si des clés sont masquées par d'autres clés identiques, elles subiront aussi la fonction (et on sait que ce sera dans l'ordre inverse de leur apparition dans la table de hachage).

Exercice 1 [OCaml recommandé] : On considère une séquence de couples, dont les éléments représentent dans cet ordre une personne et un objet. La séquence décrit en quelque sorte qui a reçu quel objet dans l'ordre chronologique. Il n'est pas garanti que chaque personne soit associée à chaque objet, en pratique. Pour chaque objet, la dernière personne à l'avoir reçue marque un point, l'avant-dernière deux points, et ainsi de suite à concurrence du nombre de personnes l'ayant reçu. Chaque personne a donc un score total pour les objets qu'il a reçus. Écrire une fonction prenant en argument une telle séquence et renvoie une séquence contenant toutes les personnes et leur score, dans l'ordre décroissant du score (la gestion des égalités peut être quelconque).

Pour l'exercice précédent, l'algorithme de tri à écrire explicitement fera partie de la notation.

Exercice 2 : À la suite de la première question de cours, écrire une commande (éventuellement en plusieurs fois) qui recopie dans un fichier appelé `j.txt` et placé dans le dossier parent toutes les lignes contenant la lettre j (on acceptera une version sensible à la casse ou non) du fichier `texte.txt` dans le dossier courant. On se servira des commandes découvertes dans le TP 0.

Exercice 3 : Écrire une fonction prenant en argument un tableau de tableaux de flottants vu comme une matrice carrée (inutile de vérifier que les dimensions sont correctes) et retournant son déterminant.

Exercice 4 : Écrire une fonction prenant en argument deux entiers `n` et `k` et retournant le nombre de surjections d'un ensemble à `n` éléments dans un ensemble à `k` éléments.

Pour l'exercice précédent, la méthode consistant à engendrer toutes les applications possibles et de compter les surjections est autorisée. On peut aussi se servir de la formule $\sum_{i=0}^{k} \binom{k}{i} u_i^{(n)} = k^n$ en notant $u_k^{(n)}$ la valeur à calculer. Dans ce cas, prouver la formule fera partie de la notation.

Exercice 5 : Déterminer ce que fait la fonction suivante.

```
let f g =
  let dans_la_liste = Hashtbl.create 8 in
  let aux1 sommet =
    if Hashtbl.mem dans_la_liste sommet then [] else begin
      Hashtbl.add dans_la_liste sommet 0;
      let rec aux2 accu s =
        let rec mouline accu2 l = match l with
        | [] -> s::accu2
        | t::reste when Hashtbl.mem dans_la_liste t -> mouline accu2 reste
        | t::reste -> Hashtbl.add dans_la_liste t 0;
          let accu3 = aux2 accu2 t in mouline accu3 reste
        in mouline accu g.(s)
      in aux2 [] sommet end
  in let res = ref [] in
  for i = 0 to Array.length g - 1 do res := aux1 i @ !res done; !res;;
```

Problème 1

On considère la base de données suivante, utilisée par une auto-école afin de gérer les sessions d'exercices au code. Les tables sont les suivantes, avec des attributs intuitifs (et l'identifiant de séance dans l'ordre chronologique) :

- `CLIENTS`, avec les attributs `Id` (entier, clé primaire), `NomPrenom` (chaîne de caractères) et d'autres informations qui ne nous intéressent pas ici ;
- `SEANCES`, avec les attributs `Id_seance` (entier, clé primaire), `Date` (format spécifique ordonné), `Id_client` (entier, clé extérieure référant au client), `NbFautes` (entier) ;
- `REPONSES`, avec les attributs `Id_seance` (entier, clé extérieure référant à la séance), `Id_client` (entier, clé extérieure référant au client), `Question` (entier), `Reponses` (chaîne de caractères) ;
- `SOLUTIONS`, avec les attributs `Id_seance` (entier, clé extérieure référant à la séance), `Question` (entier), `Reponses` (chaîne de caractères) ;

Question P1.1 : Quelle clé primaire peut-on envisager pour les tables `REPONSES` et `SOLUTIONS` ?

Question P1.2 : Au vu de la structure, on peut considérer qu'une table n'était pas nécessaire. Comment procéder pour s'en passer ?

Question P1.3 : Écrire des requêtes permettant de récupérer les informations suivantes :

- Le nombre de clients enregistrés.
- L'ensemble des dates où une séance a eu lieu.
- Le nombre moyen de fautes d'un client précisé (on considère que son identifiant est disponible dans la valeur `n`).
- Le nombre minimal de fautes sur l'ensemble des clients et des séances.
- Le nombre maximal de questions ayant la même solution lors d'une séance précisée (on considère que son identifiant est disponible dans la valeur `s`).
- Le nombre de fautes qu'un client précisé (on considère que son identifiant est disponible dans la valeur `n`) a faites lors d'une séance précisée (de même, l'identifiant sera `s`). Pour cette dernière requête, on s'interdira d'utiliser la table `SEANCES` (l'idée est de faire une requête d'insertion à partir de données élémentaires).

Question P1.4 : Un client est considéré comme prêt à passer l'examen du code s'il a toujours fait moins de cinq fautes lors de ses trois précédentes séances. Écrire une requête qui détermine si c'est le cas d'un client précisé.

Question P1.5 : L'auto-école a décidé d'une offre de lancement promotionnelle pour fêter les deux mois de son activité : au client qui a fait le moins de fautes au total et dont l'assiduité est parfaite, dix heures de conduite sont offertes. Écrire une requête qui a permis de le trouver. On pourra supposer qu'il y a existence et unicité.

Deux énigmes de logique

Énigme 1 : Résoudre l'énigme ci-après.

[Source : magazine TV hebdomadaire allemand]

La séquence d'opérations ci-après est formée de lettres représentant chacune un seul et même chiffre, sans que deux lettres différentes ne représentent le même chiffre.

Le principe est simple : remplacer chaque lettre par le chiffre correspondant, en détaillant au maximum le raisonnement.

$$\begin{array}{cccccccc} A & B & \times & C & = & D & E & C \\ & - & & + & & & & - \\ E & F & - & D & = & & C & A \\ \hline E & B & \times & G & = & D & F & G \end{array}$$

Énigme 2 : Résoudre l'énigme ci-après.

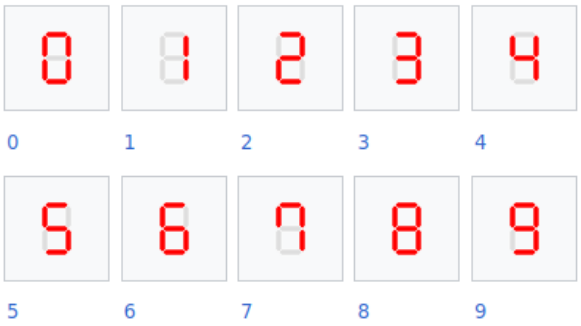
[Source : magazine "der Skatfreund" numéro 1 de 2019]

La grille ci-après est composée de 7 lignes de taille 7. Dans les cellules de chaque ligne et de chaque colonne, il faut placer deux cases noircies ainsi que chaque nombre parmi 2, 3, 4, 10 et 11 (ou les symboles de carte correspondants...) une et une seule fois. Les nombres au début de chaque ligne (resp. colonne) indiquent la somme des nombres qui sont entre les deux cases noircies de la ligne (resp. colonne) en question. La solution est unique (mais ce n'est pas une information à utiliser comme argument). Pour avoir la totalité des points, il faudra expliquer des passages non triviaux du raisonnement.

	15	15	28	??	13	13	3
18							
4							
0							
??							
11							
12							
15							

Problème 2

L'affichage à sept segments est une méthode employée dans les appareils numériques pour représenter (au moins) les dix chiffres. Le choix des segments est imposé pour ce problème et donné ci-après.



Les segments sont nommés par les sept premières lettres de l'alphabet selon l'agencement suivant :

$$\begin{array}{ccc} & a & \\ f & & b \\ & g & \\ e & & c \\ & d & \end{array}$$

Chaque lettre correspondra à une variable booléenne étant mise à vrai si, et seulement si, le segment est censé être visible.

Question P2.1 : Donner pour chaque chiffre une formule propositionnelle sous forme de conjonction utilisant les sept variables (dans l'ordre alphabétique si possible) qui est vraie si, et seulement si, l'affichage donne le chiffre en question.

Avec une telle représentation, on utilise sept bits (en pratique un octet) pour représenter l'affichage. Ce n'est pas spécialement du gaspillage, car d'autres symboles peuvent avoir un code ainsi, notamment les lettres de A à F en vue de faire de l'hexadécimal.

Il y a deux méthodes classiques pour passer d'un affichage à sept segments à un entier qui l'encode : écrire $\overline{abcdefg}^2$ ou $\overline{gfedcba}^2$. **On utilisera la première méthode ici**.

Question P2.2 : Donner pour chaque chiffre le code entier associé selon la méthode ci-avant.

Question P2.3 : De manière duale, on peut aussi considérer dix variables de v_0 à v_9 indiquant si c'est le chiffre en question qui est affiché. Quelle formule en forme normale disjonctive reliant ces dix variables est alors nécessairement vraie quand un chiffre est affiché ?

Question P2.4 : Sous les mêmes hypothèses, on pourrait écrire une formule présentant une équivalence entre chaque variable de a à g et une disjonction de variables parmi les dix (conjointe à la formule de la question précédente). En pratique, pour faciliter les choses, on considère quatre variables, $b8$, $b4$, $b2$ et $b1$, de sorte que le chiffre affiché corresponde au nombre $8 \times b8 + 4 \times b4 + 2 \times b2 + b1$ (par abus, on assimile ici les booléens aux entiers selon la méthode usuelle). Dans ce cas, écrire pour v_3, v_6 et v_9 une formule équivalente en se servant des quatre nouvelles variables (inutile de faire les dix...).

Question P2.5 : Écrire finalement pour a une formule équivalente en se servant des quatre mêmes variables.

Question P2.6 : Même exercice pour les six autres variables entre b et g (cet exercice donnera moins de points que le précédent, il s'agit en quelque sorte d'un bonus si on a terminé).

La suite du problème consiste en des exercices de programmation dans un langage au choix.

Question P2.7 : Dans certains cas, un des segments peut être défectueux et rester constamment affiché ou ne jamais l'être. On peut néanmoins tenter de deviner le chiffre à afficher en considérant le « plus proche » au sens où le nombre de segments différents est le plus petit. Écrire une fonction prenant en argument un tableau de sept booléens (représentant de a à g là encore) et renvoyant une séquence listant tous les chiffres minimisant la « distance » à l'affichage qui s'en déduit.

Question P2.8 : Adapter la fonction pour renvoyer un booléen indiquant si le chiffre est identifiable sans ambigüité, en supposant que seul un segment soit défectueux.

On pourra au choix écrire une fonction qui appelle la précédente et fait un post-traitement ou signaler les modifications à apporter à la fonction précédente. Si la fonction précédente n'a pas été faite, il faudra être très précis quant aux adaptations...

Annexe : table ASCII restreinte.

Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P
33	!	49	1	65	A	81	Q
34	"	50	2	66	B	82	R
35	#	51	3	67	C	83	S
36	$	52	4	68	D	84	T
37	%	53	5	69	E	85	U
38	&	54	6	70	F	86	V
39	'	55	7	71	G	87	W
40	(	56	8	72	H	88	X
41	)	57	9	73	I	89	Y
42	*	58	:	74	J	90	Z
43	+	59	;	75	K	91	[
44	,	60	<	76	L	92	\
45	-	61	=	77	M	93	]
46	.	62	>	78	N	94	^
47	/	63	?	79	O	95	_