Correction du DS 6

Julien Reichert

La correction des questions de cours étant notamment dans le cours ou dans les corrigés de TD, elle ne sera pas donnée ici...

Problème 0

Exercice P0.1

```
int** redirection(char* motif)
{
  int n = strlen(motif);
  int** res = malloc(n * sizeof(int*));
  for (int i = 0 ; i < n ; i += 1)
  {
    res[i] = malloc(256 * sizeof(int));
    for (int j = 0 ; j < 256 ; j += 1) res[i][j] = (i+1);
  }
  for (int i = 0 ; i < n ; i += 1)
  {
    for (int j = i ; j < n ; j += 1) res[j][(int) motif[i]] = j-i; // Dont le 0 quand j = i.
    // On remarque que des 0 peuvent effacer des valeurs venant d'une occurrence antérieure.
  }
  return res;
}
```

Exercice P0.2

```
int hachage (char* s, int taille)
{
  int final = 0;
  for (int i = 0 ; i < taille ; i += 1)
  {
    final = (final * 31 + (int) s[i]) % 97;
  }
  return final;
}
```

Exercice P0.3

On note `k` la taille du motif et `bpkm1` le résidu modulo `97` de `31` élevé à la puissance `k-1`.

```
int hachage_suivant (int x, char* s, int k, int bpkm1, int ind)
{
  return ((x - s[ind] * bpkm1) * 31 + s[ind + k]) % 97;
}
```

Exercice P0.4

```
int bkmr(char* s, char* m) {
  int ns = strlen(s); int nm = strlen(m);
  if (nm > ns) return -1;
  int hm = hachage(m, nm); int hs = hachagebis(s, nm);
  int bpkm1 = 1;
  for (int i = 1 ; i < nm ; i += 1) bpkm1 = (bpkm1 * 31) % 97;
  int prochain = 0;
  int** tableau = redirection(m);
  for (int i = 0 ; i < ns - nm ; i++)
  {
    if (i >= prochain && hm == hs)
    {
      int j = nm-1;
      while (j >= 0 && s[i + j] == m[j]) j -= 1;
      if (j < 0)
      {
        for (int k = 0 ; k < nm ; k += 1) free(tableau[k]);
        free(tableau);
        return i;
      }
      prochain = i + tableau[j][s[i+j]];
    }
    hs = (hachage_suivant(hs, s, b, nm, bpkm1, i) + 97) % 97; // Risque de valeur négative sinon.
  }
  for (int k = 0 ; k < nm ; k += 1) free(tableau[k]);
  free(tableau);
  return -1;
}
```

Problème 1

Exercice P1.1

```
SELECT MAX(prix) FROM OBJETS
```

Exercice P1.2

```
SELECT AVG(prix) FROM OBJETS JOIN CADEAUX ON id_objet = cadeau
```

Exercice P1.3

```
SELECT MIN(prix) FROM OBJETS JOIN CADEAUX ON id_objet = cadeau WHERE satisfaction
```

Exercice P1.4

Version tenant compte des égalités quand même :

```
SELECT categorie FROM
( SELECT categorie, SUM(prix) AS prix_total FROM OBJETS JOIN CADEAUX ON id_objet = cadeau
  GROUP BY categorie )
WHERE prix_total =
( SELECT SUM(prix) FROM OBJETS JOIN CADEAUX ON id_objet = cadeau
  GROUP BY categorie
  ORDER BY SUM(prix) DESC LIMIT 1 )
```

Exercice P1.5

```
SELECT COUNT(*) FROM CADEAUX
JOIN PARTICIPANTS AS P1 ON participant = P1.id_participant
JOIN PARTICIPANTS AS P2 ON destinataire = P2.id_participant
WHERE P1.genre='F' AND P2.genre='M'
```

Exercice P1.6

```
SELECT COUNT(*) FROM PARTICIPANTS
JOIN CADEAUX AS C1 ON C1.participant = id_participant
JOIN CADEAUX AS C2 ON C2.destinataire = id_participant
JOIN OBJETS AS O1 ON O1.id_objet = C1.cadeau
JOIN OBJETS AS O2 ON O2.id_objet = C2.cadeau
WHERE O1.categorie = O2.categorie
```

Exercice P1.7

```
SELECT MIN(-o_m_r) AS max_recu_moins_offert, MAX(o_m_r) AS max_offert_moins_recu FROM
( SELECT O1.prix - O2.prix AS o_m_r FROM PARTICIPANTS
  JOIN CADEAUX AS C1 ON C1.participant = id_participant
  JOIN OBJETS AS O1 ON O1.id_objet = C1.cadeau
  JOIN CADEAUX AS C2 ON C2.destinataire = id_participant
  JOIN OBJETS AS O2 ON O2.id_objet = C2.cadeau )
```

Problème 2

Exercice P2.1

La formule, qu'on note φ_i, est $\bigvee_{j=1}^{k_i} v_{i,s_{i,j}}$.

Exercice P2.2

On note $J = \{i \in [|1;n|] \mid \exists j \in [|1;k_i|], d = s_{i,j}\}$.

En ce qui concerne le fait de voir au moins un spectacle à la date d, la formule qu'on note ψ'_d est $\psi'_d = \bigvee_{i \in J} v_{i,d}$, et la formule vraie si, et seulement si, je suis allé voir au plus un spectacle à la date d est $\psi''_d = \bigwedge_{a,b \in J, a \neq b} \neg v_{a,d} \vee \neg v_{b,d}$.

Exercice P2.3

Dans le même style que dans la P2.2, on crée $\varphi'_i = \bigwedge_{1 \leq d < d' \leq k} \neg v_{i,d} \vee \neg v_{i,d'}$.

La formule attendue est $\bigwedge_{i=1}^{n} \varphi_i \wedge \bigwedge_{i=1}^{n} \varphi'_i \wedge \bigwedge_{d=1}^{k} \psi''_d$.

Exercice P2.4

Le problème auquel on pense est MAX-SAT, mais puisqu'il faut prendre en compte toutes les clauses excluant un conflit de date, il n'est pas garanti que chaque clause non satisfaite corresponde à un spectacle qui n'est pas vu.

Une variante de MAX-SAT comptant le nombre de clauses satisfaites en imposant qu'un sous-ensemble soit intégralement satisfait conviendrait.

Dans l'état actuel des choses, on dispose tout de même d'une borne : si m clauses sont insatisfaites, il y a au plus m spectacles manqués (et a priori au moins de l'ordre de $\sqrt{m}$, car dans le pire des cas on a une affectation des variables qui donne lieu à avoir k spectacles le même jour, ce qui fait que $\binom{k}{2}$ clauses sont insatisfaites).

Problème 3

Exercice P3.1

La réponse est non, et on peut imaginer une illustration par un contre-exemple minimaliste : sur une grille de 2 cases de haut et 2 cases de large, les quatre indices sont forcément identiques. Pour peu que ces indices ne soient pas tous 0 ni 4, il n'y a pas unicité de la solution car la solution est alors invariante par rotation et symétrie.

Dans le cas général, une case sans indice dans son voisinage peut être noire ou blanche sans que cela n'impacte le moindre indice.

Exercice P3.2

La réponse est non, et le même contre-exemple s'applique avec par exemple une seule case à noircir sur les quatre, en mettant l'indice 1 partout.

Exercice P3.3

La réponse est oui, avec une grille de 3 cases de haut et 3 cases de large où la solution est de noircir toutes les cases sauf les quatre coins. Les indices sont alors 3 dans les coins, 4 aux bords et 5 au centre, donc les contraintes de l'énoncé sont respectées, et la solution est unique par le raisonnement suivant :

- L'indice 5 impose quatre cases blanches sur l'ensemble de la grille.
- Chaque indice 3 exclut d'avoir plus d'une case blanche dans chaque carré de 2 cases de côté (contenant un coin) inclus dans la grille.
- Par conséquent, la case centrale ne peut qu'être noire (sinon ce serait la seule case blanche de la grille d'après ce qui précède).
- De même, les cases orthogonalement adjacentes à la case centrale (autrement dit les cases qui ne sont ni en coin ni au centre) ne peuvent pas être blanche, car sinon cela correspondrait à une case blanche commune à deux carrés et le même problème interviendrait.
- Une fois cinq cases obligatoirement noires trouvées, on peut conclure.

Attention, l'unicité de la solution n'est pas garantie : le même contre-exemple que dans la question précédente s'applique.

Exercice P3.4

```
let demarrage grille =
  let n = Array.length grille in let m = Array.length grille.(0) in (* Pas de blague avec n = 0. *)
  let taille_voisinage i j =
    if n = 1 && m = 1 then 1 (* grille dégénérée *)
    else if n = 1 && (j = 0 || j = m-1) then 2
    else if m = 1 && (i = 0 || i = n-1) then 2
    else if n = 1 || m = 1 then 3
    else if (i = 0 || i = n-1) && (j = 0 || j = m-1) then 4
    else if i = 0 || i = n-1 || j = 0 || j = m-1 then 6
    else 9
  in let rec mouline i j =
    if i = n then []
    else if j = m then mouline (i+1) 0
    else if grille.(i).(j) = taille_voisinage i j || grille.(i).(j) = 0 then (i, j)::(mouline i (j+1))
    else mouline i (j+1)
  in mouline 0 0
```

Exercice P3.5

```
let compte_voisins reso i j =
  let n = Array.length reso in
  let m = Array.length reso.(0) in
  let valeurs = [|0; 0; 0|] in
  for ligne = max 0 (i-1) to min (n-1) (i+1) do
    for colonne = max 0 (j-1) to min (m-1) (j+1) do
      valeurs.(reso.(ligne).(colonne) + 1) <- valeurs.(reso.(ligne).(colonne) + 1) + 1
    done
  done;
  valeurs

let completable grille reso =
  let n = Array.length grille in
  let m = Array.length grille.(0) in
  let rec mouline i j =
    if i = n then []
    else if j = m then mouline (i+1) 0
    else if grille.(i).(j) = -1 then mouline i (j+1)
    else let valeurs = compte_voisins reso i j in
      if valeurs.(0) = 0 then mouline i (j+1) (* Plus rien à faire *)
      if valeurs.(2) = grille.(i).(j) (* Déjà le bon nombre de cases noires. *)
      then (i, j, 0)::(mouline i (j+1))
      else if valeurs.(2) + valeurs.(0) = grille.(i).(j) (* Déjà le bon nombre de cases blanches *)
      then (i, j, 1)::(mouline i (j+1))
      else mouline i (j+1)
  in mouline 0 0
```

Exercice P3.6

```
let miseajour reso todolist =
  let n = Array.length reso in
  let m = Array.length reso.(0) in
  let traite (i, j, b) =
    for l = max 0 (i-1) to min (n-1) (i+1) do
      for c = max 0 (j-1) to min (m-1) (j+1) do
        if reso.(l).(c) = -1 then reso.(l).(c) <- b
      done
    done
  in List.iter traite todolist
```

Exercice P3.7

```
let resoudre_simple grille =
  let n = Array.length grille in
  let m = Array.length grille.(0) in
  let reso = Array.make_matrix n m (-1) in
  let stop = ref false in
  while not !stop do
    match completable grille reso with
    | [] -> stop := true
    | l -> miseajour reso l;
  done;
  reso, Array.for_all (fun tab -> not (Array.mem (-1) tab)) reso
```

Exercice P3.8

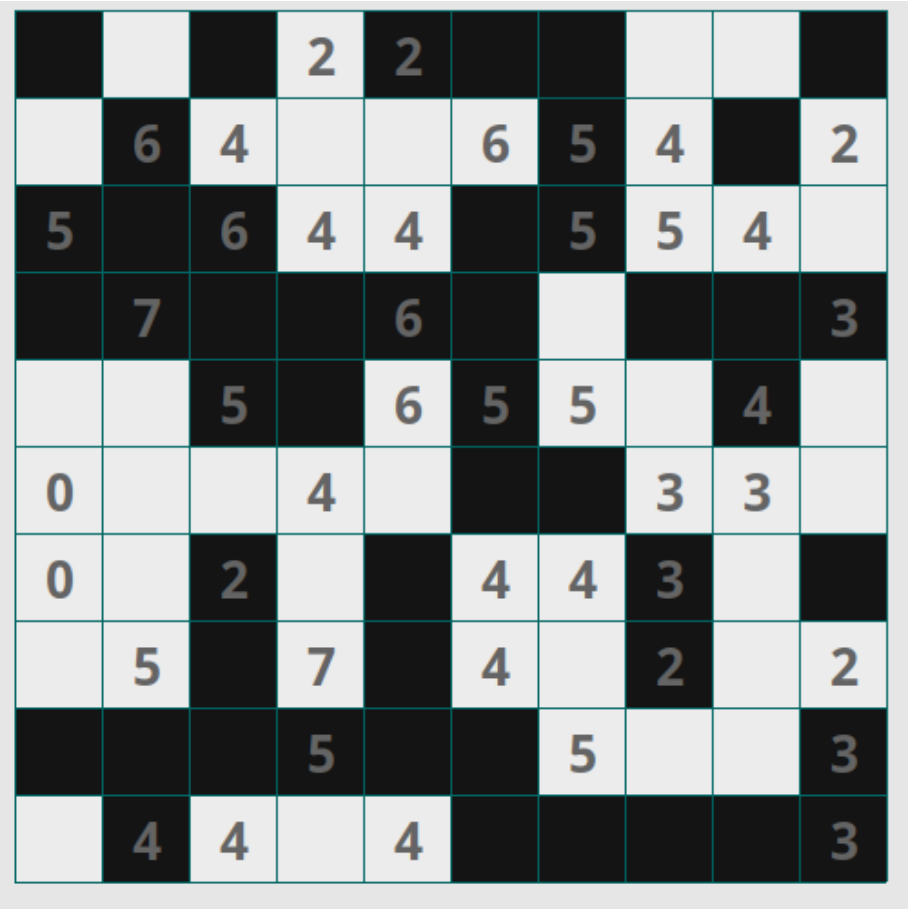
En tant qu'humain, on peut plus facilement identifier des zones d'une certaine taille n dont k cases sont à noircir, ce qui permet de déduire d'autres zones par répercussion à partir d'indices voisins. À terme, on trouve éventuellement des zones à noircir intégralement ou à blanchir intégralement. Il s'agit d'une extension des raisonnements mis en œuvre dans ce qui précède.

Et bien entendu, toute réponse à la question suivante peut correspondre dans cette question en tant qu'algorithme que l'on ferait tourner à la main.

Exercice P3.9

L'exploration exhaustive est toujours un recours dans ce genre de situation, en émettant une hypothèse sur la couleur d'une case (enregistrer quelle hypothèse est faite) et en faisant toutes les déductions qui en découlent (mémoriser dans le cadre de quelle hypothèse chaque déduction a été faite), et dans le cas où une contradiction apparaît, on peut déduire que la dernière hypothèse est erronée, ce qui veut dire dans le cas précis de ce jeu que la case est de l'autre couleur, sans autre choix possible comme dans le cadre du sudoku. En vue de trouver la solution plus rapidement, on pourra choisir (en mode algorithme glouton) une hypothèse particulièrement probable ou au contraire particulièrement improbable (donc une case indéterminée au voisinage d'une case dont toutes les cases noires ont été trouvées sauf une, ou toutes les cases blanches ont été trouvées sauf une).

Exercice P3.10



Exercice P3.11