Commentaires DS n°5

Exercice : traversée de rivière

- orthographe : leng**th**
- trop d'erreurs de syntaxe OCaml : accès à élément d'un tableau, `==` au lieu de `=`, `ref` mal placé, oubli du `in`, oubli du `;`, confusion entre `@` et `::` ...
- trop d'erreurs "classiques" sur OCaml : confusion liste/tableau, vouloir modifier une variable non mutable (il faut une référence), vouloir renvoyer une valeur à l'intérieur d'une boucle for (c'est impossible ! : on utilise une référence)
- **bien lire l'énoncé** (surtout lorsqu'il est compliqué), pour éviter les contre-sens. En particulier, il n'y a qu'un seul 0 dans le tableau ; un 1 peut sauter par-dessus un 1 ou un 2
- **ne pas perdre de vue l'idée générale** de l'exercice : en particulier l'enchaînement des questions, la réutilisation des fonctions précédentes
- respecter les consignes de l'énoncé : type à utiliser, spécification de fonction
- de manière générale, pour répondre à une question de programmation, il faut :
  - bien comprendre l'énoncé (et la spécification de la fonction)
  - trouver un algorithme efficace et simple à coder
  - l'implémenter correctement en OCaml
- **Q1**. `caillou_vide` : on parcourt le tableau avec une boucle for (une boucle while alourdit considérablement le code); on utilise un référence pour stocker la position trouvée. (cette fonction sera très utile pour répondre simplement aux questions suivante)
- **Q2**. `echange` : c'est un simple échange de deux valeurs, mais dans une copie du tableau initial (cette copie sera bien utile pour Q5)
- **Q3**. `randonneurG_avance`. utiliser la fonction `caillou_vide` est très pratique. Réaliser un parcours est possible, mais plus long à coder. Attention à ne pas regarder en dehors du tableau
- **Q4**. `randonneurG_saute`. idem. On peut sauter au dessu-s d'un 1 ou d'un 2
- **Q5**. `mouvement_chemin`. On remplit une liste avec les chemins possibles après un mouvement (4 chemins au plus). Comme on veut modifier la liste, on doit utiliser une référence (type `list ref`)
- **Q6**. `passage`. La question difficile de l'exercice. Voir explications du corrigé

Problème : graphes eulériens en C

- les graphes eulériens sont HP, mais très classiques. Ici on implémente la construction du cycle eulérien dans le cas où tous les degrés du graphe sont pairs
- plusieurs questions de raisonnement sur les graphes (dont plusieurs ont été vus en exercice). Les questions de raisonnement sur les graphes sont souvent délicates ... c'est bien d'essayer, mais ne pas perdre trop de temps sur ces questions
- les correcteurs n'aiment pas les copies abordent uniquement les questions de programmation, ou uniquement les questions de démonstration ... ça sent l'impasse chez le candidat ...
- la représentation du graphe est originale : c'est une matrice d'"adjacence" contenant les arcs que l'on peut emprunter lors du parcours dans le graphe.
- dans l'énoncé, l'étoile des pointeurs est notée "à droite" `edge *e;`
- dans l'énoncé, `&a` signifie l'adresse de la variable `a` (ceci devrait être su), mais il n'était pas utile d'utiliser `&` dans le code à écrire)

- encore une fois **bien lire l'énoncé**.
- encore une fois, essayer de **ne pas perdre de vue l'idée générale** de l'exercice : en particulier l'enchaînement des questions, la réutilisation des fonctions précédentes .
- ici, on supprime les arcs de la matrice pour former des **listes circulaires doublement chaînées** ...
- ... et les questions de raisonnement nous indiquent que s'il reste un sommet sur le graphe avec un arc non utilisé, on peut repartir de cet arc pour créer un nouveau cycle, puis joindre les 2 pour créer un grand cycle, (et itérer)
- Le code des fonctions fournies est tout à fait compréhensible, même s'il n'est pas indispensable de le comprendre pour répondre aux questions ...
- **Q1**. exemples de graphes. Pour le graphe eulérien, attention à ce que le chemin à suivre compréhensible par le correcteur
- **Q2**. démonstration : cycle eulérien $\Rightarrow$ ... Déjà vu en cours, mais il y a d'autres démonstrations possibles
- **Q3**. `is_eulerian`. question plutôt facile, qui demande d'avoir bien compris la structure de données.
- **Q4**. démonstration, mêmes remarques que pour Q2
- **Q5**. `round_trip`. on avance dans le graphe (en connectant les arcs suivis) jusqu'à retomber sur le sommet de départ. Petit piège, on demande de créer un cycle : le premier et le dernier arc doivent être connectés !
- **Q6**. `find_vertex_with_edges`. on parcourt jusqu'à trouver un sommet auquel il reste un arc (s'il en existe un). Re-petit piège , on parcourt un cycle : la condition d'arrêt n'est pas de trouver le pointeur NULL (il n'y en a pas !) mais de retomber sur le sommet de départ
- **Q7**. `eulerian_cycle`. question de synthèse, assez facile si on a bien compris l'énoncé
- **Q8**. calcul de complexité non trivial : il faut détailler le nombre et le coût des différents appels
- **Q9**. démonstration, un joli raisonnement (non vu en cours)
- **Q10**. exemple final. Il faut décrire correctement le graphe, de donner le nombre de sommets avec un. Re-re-petit piège : il faut compter l'extérieur comme une pièce