DS 3

Informatique MP2I

Julien Reichert

Questions de cours

Question de cours A1 : Soit un programme dont le calcul de complexité donne la relation $c_n = 2c_{n/2} + 2n + 3$. Exprimer alors c_n en fonction de n à l'aide des notations de Landau. Donner des exemples classiques d'algorithmes ayant cette complexité asymptotique (sans écrire de programme associé).

Question de cours A2 : Décrire la structure de données abstraite de liste chaînée (sans écrire d'implémentation).

Question de cours A3 : Donner deux implémentations possibles de la structure de file en expliquant les algorithmes réalisant les opérations élémentaires (sans écrire de programme).

Questions de cours en OCaml

Question de cours B1 : Soit le type suivant : `type carte = {valeur : int , couleur : int};;`. Écrire une fonction qui prend en argument un entier et qui retourne la carte associée avec pour valeur le quotient de l'entier dans la division par `4` et pour couleur le reste dans la même division. Il faudra vérifier au préalable que l'entier est entre `0` et `31` inclus.

Question de cours B2 : Écrire une expression qui déclenche une exception (pas une erreur de syntaxe, attention), quelle que soit la méthode.

Question de cours B3 : Écrire une fonction qui prend en argument un entier `n` et un flottant `x` et qui détermine si `x` est strictement compris entre `n` et `n+1`.

Questions de cours en C

Question de cours C1 : Soit la boucle `for (int i = 0 ; condition(i) ; maj(&i)) corps(i);`. Écrire un code rigoureusement équivalent qui serait placé dans le même contexte que cette boucle mais en utilisant exclusivement `while`. Les fonctions `condition`, `maj` et `corps` sont à reprendre telles quelles.

Question de cours C2 : Compléter le code suivant, où `aux` est une chaîne de caractères auxiliaire contenant `n` vrais caractères, à gérer correctement.

```
char f(int t[], int n)
{
  // Créer ici aux correctement.
  /* Ici un code qui travaille sur aux et qui ne nous intéresse pas */
  // Compléter ci-après pour que la fonction renvoie le premier caractère de aux.
}
```

Question de cours C3 : Quelle est l'option de compilation qu'il est particulièrement recommandé d'ajouter et qui permet de voir « tous » les avertissements ?

Exercices en OCaml

On rappelle à toute fin utile des fonctions du module `Hashtbl` avec des informations sur leur spécification :

- `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 : Écrire une fonction prenant en argument un tableau de tableaux d'entiers ayant les propriétés suivantes : le nombre de tableaux, noté `n`, est non nul, chacun de ces `n` tableaux est de la même taille `m` également non nulle, et les `m` fois `n` éléments sont les premiers entiers naturels en un exemplaire chacun, et retournant le couple `(mini, maxi)` tel que `mini` soit la plus courte distance entre les indices de deux entiers consécutifs, mesurée en tant que différence en ligne plus différence en colonnes, et `maxi` soit la plus longue telle distance.

Par exemple, pour le tableau `[| [|6; 5|] ; [|0; 3|] ; [|7; 1|] ; [|2; 4|] |]`, la réponse sera `(1, 3)`, en réalisant la distance minimale entre `5` et `6` et la distance maximale entre `4` et `5`.

Exercice 2 : Même exercice mais sans l'hypothèse sur les éléments du tableau (qui ne seront en particulier plus forcément entiers), dans ce cas les distances devront concerner deux éléments égaux à des indices différents, en retournant au choix une erreur ou le couple `(-1, -1)` si tous les éléments sont différents deux à deux.

Exercice 3 : Écrire une fonction qui prend en argument deux entiers, matérialisant la fortune de deux personnes, et qui retourne un couple déterminant le nombre d'enchères que chaque personne peut remporter, selon le protocole suivant : tant que les deux personnes ont une fortune différente, la plus riche enchérit une valeur égale à la fortune de la moins riche plus un pour remporter une enchère, et une fois les deux fortunes égales, les enchères s'arrêtent.

Par exemple, avec `19` et `12` comme arguments, la première personne remporte une enchère en dépensant `13`, il lui reste alors une fortune de `6`, puis la deuxième personne remporte une enchère en dépensant `7`, ce qui fait qu'il lui reste une fortune de `5`, suite à quoi la première personne dépense `6` pour remporter une troisième enchère, ce qui fait qu'il ne lui reste rien, permettant à la deuxième personne de remporter cinq enchères de suite en dépensant `1`. La réponse est alors `(2, 6)`.

Exercices en C

Exercice 4 : Écrire une fonction de même principe que les fonctions `map` de Python et OCaml. Concrètement, à partir d'une fonction et un tableau, il faut renvoyer l'image par la fonction de tous les éléments du tableau dans le même ordre. Toutes les données manipulées seront des entiers. Il faudra également écrire une fonction `main` appelant la fonction écrite.

Le prototype est fourni :

```
int* map(int (*f)(int), int tab[], int taille)
```

Exercice 5 : Écrire une fonction prenant en argument deux chaînes de caractères forcément de même taille et composés de lettres capitales uniquement (inutile de vérifier ces deux propriétés) ainsi que deux pointeurs d'entiers et qui mute le premier pointeur pour que l'entier pointé devienne le nombre d'indices où les deux mots ont la même lettre, et le deuxième pointeur pour que l'entier pointé devienne le nombre d'autres lettres présentes dans les deux mots à des indices différents, sachant que si par exemple une lettre apparaît trois fois dans un des mots et deux fois dans l'autre à des indices toujours différents, ceci comptera pour deux. Il faudra également écrire une fonction `main` appelant la fonction écrite (les chaînes `ECREME` et `RECURE` sont imposées) et annoncer le résultat.

Exercice 6 : Expliquer ce que fait la fonction suivante, en donner la complexité en temps et en espace et corriger le problème associé (inutile de réécrire la fonction en entier, il suffit de bien localiser la correction à apporter) :

```
int f(int t[], int taille)
{
  assert (taille > 0);
  int *a = malloc(taille * sizeof(int));
  a[0] = 0;
  int na = 1;
  for (int i = 0 ; i < taille ; i += 1)
  {
    if (t[i] > t[a[0]])
    {
      a[0] = i;
      na = 1;
    }
    else if (t[i] == t[a[0]])
    {
      a[na] = i;
      na += 1;
    }
  }
  int reponse = a[(na-1)/2];
  return reponse;
}
```

Problème

Ce problème constitue une version discrète d'exercices pouvant être posés en analyse sur la notion de continuité. Bien entendu cette notion ne s'appliquera pas dans notre contexte.

On considèrera dans les deux langages des tableaux d'entiers avec d'éventuelles propriétés supplémentaires **valables uniquement pour la question où ces propriétés sont énoncées**.

L'indexation respectera la syntaxe des langages respectifs dans les parties correspondantes, et on écrira `t(i)` pour la partie théorique indépendamment du langage.

Dans les énoncés, on désignera toujours par `n` la taille du tableau.

Question P1 : Supposons le tableau décroissant. À quelle condition y a-t-il un indice à partir duquel `t(i) < i` sera toujours vrai ?

Question P2 : Supposons que les éléments du tableau soient des entiers entre `0` et `n-1` avec répétitions possibles. Est-il garanti qu'il existe `i` et `j` (potentiellement égaux) tels que `t(i) = j` et `t(j) = i` ?

Question P3 : Même question si on exclut **en plus** les répétitions, le tableau pouvant alors représenter une bijection de l'ensemble des entiers de `0` à `n-1` dans lui-même.

Programmation en OCaml

Question P4 : Supposons le tableau décroissant. Écrire une fonction déterminant le premier indice à partir duquel `t.(i) < i` est toujours vrai (comportement au choix si un tel indice n'existe pas). La complexité devra être logarithmique en n (pas besoin de le prouver), pas de point sinon.

Question P5 : Écrire une fonction déterminant deux indices `i` et `j` tels que `t.(i)` soit égal à `j` et `t.(j)` soit égal à `i`. S'il n'en existe pas, le comportement est au choix entre déclencher une erreur et retourner le couple `(-1, -1)`. La complexité devra être au plus linéaire en `n` (pas besoin de le prouver), la moitié des points en dépend.

Question P6 : Même question en supposant **en plus** le tableau strictement décroissant. La complexité devra être logarithmique, tous les points en dépendent. Gros bonus si on arrive à le faire sans avoir besoin de la décroissance stricte (ce n'est pas garanti qu'une solution existe).

Question P7 : Supposons que les éléments du tableau soient des nombres de `0` à `n-1` sans répétition. Le tableau représente ainsi une bijection, et il existe alors un entier naturel non nul `k` tel que la composée de `k` occurrences de cette bijection soit l'identité. Écrire une fonction calculant le plus petit `k` correspondant. Il n'y a pas d'objectif de complexité cette fois.

Programmation en C

Toutes les questions de cette partie sont associées à un calcul de complexité en temps dans le pire des cas une fois le programme écrit, ainsi que, pour les deux dernières, une preuve qu'une solution existe forcément (en parlant de l'unicité) ou un contre-exemple sans tenir compte du programme (voire sans l'avoir écrit).

Question P8 : Supposons que les éléments du tableau soient des entiers entre `0` et `n-1` avec répétitions possibles. Écrire une fonction déterminant un indice `i` tel que `t[i]` soit égal à `i`. S'il n'en existe pas, le comportement est au choix entre déclencher une erreur ou retourner la valeur `-1`.

Question P9 : Supposons **en plus** le tableau croissant. Qu'en est-il de l'existence et de l'unicité d'une solution ? Réécrire un programme si on peut optimiser la complexité.

Question P10 : Même question en supposant **à la place de croissant** le tableau décroissant.