DM d'option informatique - Le compte est bon (Renardo) - MPSI

À rendre au plus tard le 12/05/2024 à 20h à l'adresse mail habituelle.

Dans le « jeu du compte est bon », on dispose d'un tableau `tab0` composé d'entiers (non nécessairement distincts) appartenant à $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$. L'objectif est d'obtenir un résultat aussi proche que possible d'un entier noté `but` à l'aide des opérations `+, -, *, /` en respectant les règles suivantes :

→ Chaque entier de `tab0` peut être utilisé au plus une fois.

→ L'opération `-` ne peut pas être utilisée si le résultat est négatif ou nul.

→ L'opération `/` ne peut pas être utilisée si la division ne tombe pas juste.

```
Resultat le plus proche: 966
Nombre de calculs: 4
Calculs:
75 - 6 = 69
2 + 10 = 12
12 + 2 = 14
14 * 69 = 966
```

1. Écrire une fonction « `le_compte_est_bon : int -> int array -> unit` » qui prend en entrée `but` ainsi que `tab0`, et affiche une solution au jeu du compte est bon. Par exemple, avec `tab0 = [|2; 6; 75; 50; 10; 2|)` et `but = 967`, votre fonction pourra afficher le texte ci-dessus.

   Faites d'autres tests en utilisant un solveur en ligne pour vérifier que votre fonction trouve le résultat le plus proche avec le moins de calculs possibles (par exemple `https://www.dcode.fr/compte-est-bon`). Si besoin, on pourra lire les indications ci-dessous.

Indications (essayez de résoudre l'exercice sans lire ce qui suit). La méthode décrite ici n'est bien sûr pas l'unique façon de faire. Dans la suite, la lettre N sera réservée exclusivement pour faire référence à un élément du tableau `tab0` ou bien à un entier obtenu à partir de `tab0` et des opérations `+, -, *, /`. On définit de nouveaux types :

```
type operation =
  | Add  of int * int
  | Sous of int * int
  | Mult of int * int
  | Div  of int * int;;
```

```
type historique = operation list;;
```

```
type valeurs_N = {
    tab: int array;
    histo: historique};;
```

```
type resultat = {
    valeur: int;
    nb_restants: int;
    histo: historique};;
```

L'idée est de générer exhaustivement tous les entiers N atteignables à partir du tableau `tab0` et d'enregistrer celui qui est le plus proche de `but` dans une variable « `best_res: resultat ref` ». Le type `resultat` possède trois champs :

→ Le champ `valeur` contient le nombre N qu'on a réussi à obtenir.

→ Le champ `nb_restants` contient le nombre d'entiers encore utilisables.

→ Le champ `histo` contient l'historique des opérations dont on a eu besoin pour atteindre N.

Dans l'exemple de la question 1, la valeur de `!best_res` à la fin de l'exécution est donnée ci-dessous. En particulier, le champ `nb_restants` vaut `2` puisque les deux nombres pouvant encore être utilisés sont `966` et `50`.

```
{valeur = 966; nb_restants = 2;
 histo = [Mult (14, 69); Add (12, 2); Add (2, 10); Sous (75, 6)]}
```

Au cours de l'exécution, on manipulera une variable « `val_N: valeurs_N` » telle que `val_N.tab` est le tableau des nombres N encore utilisables et `val_N.histo` est l'historique des opérations dont on s'est déjà servis. Pour l'exemple de la question 1, au début de l'exécution, la variable `val_N` vaut :

```
{tab = [|2; 6; 75; 50; 10; 2|]; histo = []}
```

À un autre moment de l'exécution, le programme aura calculé `50/2 = 25` puis `25*75 = 1875`. La variable `val_N` vaudra alors (l'ordre des éléments du tableau n'a pas d'importance) :

```
{tab = [|1875; 6; 10; 2|]; histo = [Mult (25, 75); Div (50, 2)]}
```

Voici le principe général de l'algorithme : à chaque étape, on note `n` la taille de `val_N.tab` et pour tout couple `(i1, i2)` $\in [\![0; n-1]\!]^2$ tels que `i1 < i2`, on note `n1 = val_N.tab.(i1)` et `n2 = val_N.tab.(i2)`. Soit `n3` le résultat de l'une des opérations suivantes (pour une soustraction, le résultat doit être positif ; pour une division, le résultat doit être entier) :

`(n1 + n2)`, `(n1 - n2)`, `(n2 - n1)`, `(n1 * n2)`, `(n1 / n2)`, `(n2 / n1)`.

Si `n3` est le meilleur résultat vu jusqu'à présent, on met à jour la variable `best_res`. On recommence ensuite récursivement en supprimant `n1` et `n2` du tableau et en y ajoutant `n3`.

Plus d'indications (essayez de résoudre l'exercice sans lire ce qui suit).

2. Écrire une fonction « `print_res : resultat -> unit` » qui affiche un résultat sous la forme demandée dans la question 1.
3. Écrire une fonction « `nv_res : int -> resultat -> resultat ref -> unit` » qui prend en entrée l'entier `but`, un entier `res` ainsi que la variable `best_res`, et teste si la variable `res` est le meilleur résultat vu jusqu'à présent (dans ce cas, on met à jour la variable `best_res`). "Le meilleur résultat" signifie que le nombre est plus proche de `but` que tous les résultats vus précédemment. En cas d'égalité, on sélectionnera le résultat ayant nécessité le moins de calculs.
4. Écrire une fonction « `ini_best_res : int -> int array -> resultat ref` » qui prend en entrée l'entier `but` ainsi que le tableau `tab0` et initialise la variable `best_res`. Pour cette initialisation, le meilleur résultat correspond à l'entier de `tab0` le plus proche de `but`. Si on reprend l'exemple de la question 1, l'élément de `tab0` le plus proche de `967` est `75` donc la valeur initiale de `!best_res` est :

```
{valeur = 75; nb_restants = 6; histo = []}
```

5. Écrire une fonction « `suppr : 'a array -> int -> 'a array` » qui prend en entrée un tableau `tab` de taille `n` ainsi qu'un indice `i`, et renvoie un tableau de taille `n-1` contenant tous les éléments de `tab` sauf `tab.(i)`.
6. Écrire une fonction « `get_nv_N : int -> int -> (int * operation) array` » qui prend en entrée deux entiers `n1` et `n2` et renvoie un tableau contenant tous les entiers N possibles parmi :

   `(n1 + n2)`, `(n1 - n2)`, `(n2 - n1)`, `(n1 * n2)`, `(n1 / n2)`, `(n2 / n1)`.

   Par exemple :

```
(* (get_nv_N 12 3) vaut: *)
[|(15, Add(12,3)); (36, Mult(12,3)); (9, Sous(12,3)); (4, Div(12,3))|]
```

7. Écrire une fonction « `explorer_LCEB : int -> valeurs_N -> resultat ref -> unit` » qui prend en entrée l'entier `but` ainsi que les variables `val_N` et `best_res`, et suit la procédure suivante :
   - Soit `n` la taille de `val_N.tab`.
   - Pour chaque `i2` $\in [\![0; n-1]\!]$:
     - On note `n2 = val_N.tab.(i2)` et on définit `tab2` le tableau renvoyé par « `suppr val_N.tab i2` ».
     - Pour chaque `i1` $\in [\![0; i2-1]\!]$:
       - On note `n1 = val_N.tab.(i1)`.
       - Pour chaque entier `n3` obtenu avec l'appel à « `get_nv_N n1 n2` » :
         — On teste si `n3` est le meilleur résultat vu jusqu'à présent avec la fonction `nv_res`.
         — On remplace `tab2.(i1)` par `n3` et on fait un appel récursif à la fonction `explorer_LCEB` sur le tableau `tab2`.
       - On remplace `tab2.(i1)` par `n1`.
8. Répondre à la question 1.