

## 1 Minimisation d'échange d'argent

«Écrire un programme qui détermine les remboursements à effectuer entre des personnes se devant mutuellement de l'argent, en minimisant le nombre d'échanges. »

C'est un problème courant : dans un groupe chacun a payé une partie des dépenses du groupe. L'une a payé le restaurant pour d'autres, l'un a payé les courses pour une autre, etc. À la fin, chacun doit régler ses dettes envers les autres et souhaite effectuer un nombre minimum de virements à ses comparses. Un algorithme glouton existe pour déterminer qui doit *in fine* verser combien à qui.

Il consiste à commencer par calculer le **solde** net de chaque participant, c'est-à-dire la somme de ce qui lui est dû moins la somme de ce qu'il doit.

Ensuite, à chaque étape il faut trouver la personne **devant le plus** d'argent et celle **attendant le plus** d'argent. Si ces 2 montants sont nuls, plus personne ne doit rien et plus personne n'attend rien : le problème est réglé. Sinon, le montant du versement à effectuer est le minimum des deux en **valeur absolue**. On crédite et débite les comptes des 2 personnes, on affiche qui verse combien à qui.

**Q 1.1.** Prenons un exemple concret et nommons les personnes  $P_0$ ,  $P_1$ , etc.

- $P_0$  doit 100 à  $P_1$ .
- $P_0$  doit 50 à  $P_2$ .
- $P_2$  doit 75 à  $P_0$ .
- $P_3$  doit 20 à  $P_2$ .
- $P_3$  doit 40 à  $P_1$ .

Représentez ces dettes par un schéma.

**Q 1.2.** Imaginez, avec votre chargé de TD, comment représenter «informatiquement » (par une structure de données) ce schéma.

**Q 1.3.** À partir de la description informelle des traitements, quelles sont les 2 grandes étapes de l'algorithme ?

**Q 1.4.** Esquissez l'algorithme de la fonction qui implémente la **seconde** étape identifiée en Q1.3. Quels sont ses domaines d'entrée et de sortie ?

**Q 1.5.** Esquissez l'algorithme qui implémente la **première** étape identifiée en Q1.3. Quels sont ses domaines d'entrée et de sortie ?

**Q 1.6.** Dans l'esquisse issue de Q1.4, il est question de trouver les personnes ayant le solde le plus débiteur et le solde le plus créditeur. Proposez une solution.

## 2 Fraction égyptienne

«Écrire un programme qui transforme une fraction inférieure ou égale à 1 en une fraction égyptienne. »

Les Égyptiens antiques n'utilisaient que des fractions dont le numérateur était 1. On qualifie une telle fraction de *unitaire*. Il est possible d'écrire n'importe quelle fraction inférieure ou égale à 1 sous forme d'additions de fractions unitaires de dénominateurs **tous différents**.

L'algorithme glouton de Fibonacci permet de déterminer les fractions à additionner pour représenter une fraction  $\frac{a}{b}$ .

À chaque étape il cherche la plus grande fraction unitaire  $\frac{1}{n}$  **strictement inférieure** à  $\frac{a}{b}$  et récurse sur  $\frac{a}{b} - \frac{1}{n}$  tant que ce «reste» n'est pas une fraction unitaire (ou ne se réduit pas trivialement à une fraction unitaire).

Les fractions  $\frac{1}{n}$  trouvées ainsi que le dernier «reste» sont celles dont la somme représente la fraction initiale.

**Q 2.1.** Reformulez cet algorithme de manière plus structurée.

**Q 2.2.** Comment trouver la plus grande fraction unitaire inférieure à  $\frac{a}{b}$  ?

### 3 Implémentation : minimisation d'échange d'argent

**Q 3.1.** Écrivez la fonction `compute_amounts` qui implémente la **première** étape identifiée en Q1.3 et dont vous avez esquissé l'algorithme en Q1.5.

**Q 3.2.** Écrivez la fonction `find_min_max_indices` dont vous avez esquissé l'algorithme en Q1.6.

**Q 3.3.** Écrivez la fonction `compute_flow` qui implante l'algorithme esquissé en Q4.

**Q 3.4.** Écrivez la fonction principale `main` qui orchestre tout les traitements. Vous pourrez décrire en dur un graphe représentant un jeu de données (par exemple celui donné dans l'énoncé). Pour vous éviter l'écriture des valeurs à mettre dans chaque case de la matrice, dans le fichier `cflow_skel.c` vous sont données les quelques lignes de code le faisant. Attention il vous faut quand même allouer la matrice `debts` avant !

Attention, comme discuté en Q1.2, vous n'avez pas d'autre choix que l'allocation dynamique pour créer votre graphe puisque c'est un tableau à 2 dimensions de taille *a priori* inconnue à la compilation.

### 4 Implémentation : fraction égyptienne

**Q 4.1.** Écrivez une fonction qui prend en argument une fraction et affiche la suite de fractions unitaires composant la fraction égyptienne associée.

**Attention** : votre fonction devra travailler avec des **long int** et non de simples **int**.

**Q 4.2.** Écrivez le `main`, recevant sur la ligne de commande la fraction (numérateur et dénominateur), qui lance le calcul de la fraction égyptienne. Il faudra s'assurer que la fraction est bien **inférieure ou égale** à 1.

**Pour rappel** : la conversion chaîne  $\rightarrow$  **long int** se fait avec la fonction `atol` qui nécessite l'inclusion de `stdlib.h`.

**Q 4.3.** Testez votre programme avec  $\frac{7}{11}$ . Calculez ce que vaut  $\frac{1}{2} + \frac{1}{11} + \frac{1}{22}$ . Que constatez-vous ?

**Q 4.4.** Testez votre programme avec  $\frac{4}{65}$ . Calculez ce que vaut  $\frac{1}{26} + \frac{1}{65} + \frac{1}{130}$ . Que constatez-vous ?