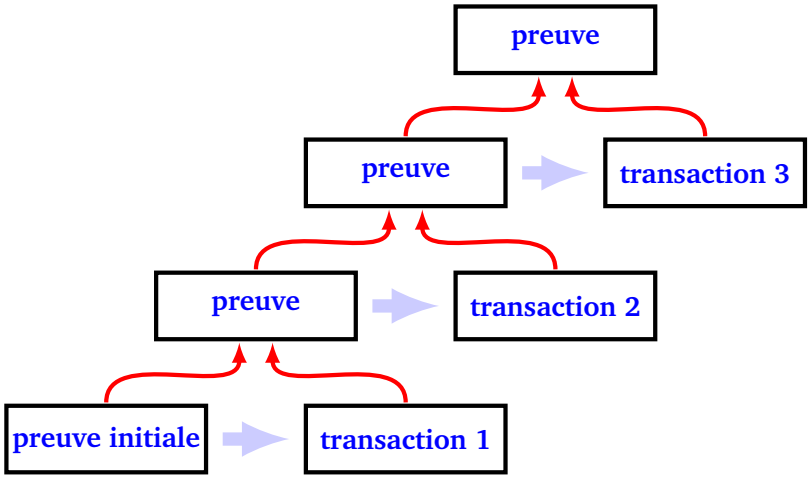
Bitcoin

- La monnaie *bitcoin* est une monnaie dématérialisée.
- Les transactions sont enregistrées dans un grand livre de compte appelé *blockchain*.
- Imaginons un groupe d'amis qui souhaitent partager les dépenses du groupe de façon la plus simple possible.
- Au départ tout le monde dispose de 1000 *bitcoins* et on note au fur et à mesure les dépenses et les recettes de chacun.
- On note sur le livre de compte la liste des dépenses/recettes, par exemple :
  — « Amir a dépensé 100 *bitcoins* »
  — « Barbara a reçu 45 *bitcoins* »
  — etc.
- Il suffit de parcourir tout le livre pour savoir combien chacun a reçu ou dépensé depuis le début.

Blockchain

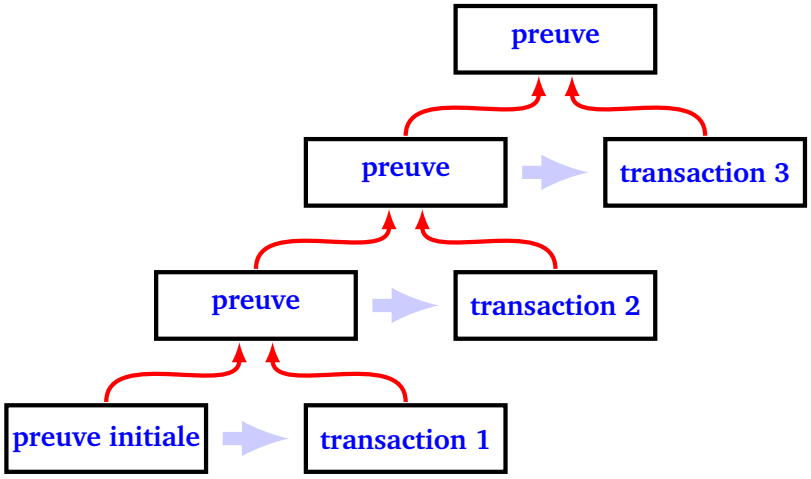
Pour éviter que quelqu'un ne vienne truquer le livre de compte, après chaque transaction on ajoute dans le livre une certification construite à partir d'une preuve de travail.

1. On commence par une preuve de travail quelconque. Pour nous ce sera `[0,0,0,0,0,0]`.
2. On écrit la première transaction (par exemple `"Amir -100"`).
3. On calcule et on écrit dans le livre une preuve de travail, qui va servir de certificat. C'est une liste (par exemple `[56,42,10,98,2,34]`) obtenue après beaucoup de calculs prenant en compte la transaction précédente et la précédente preuve de travail.
4. À chaque nouvelle transaction (par exemple `"Barbara +45"`), quelqu'un calcule une preuve de travail pour la dernière transaction associée à la précédente preuve. On écrit la transaction, puis la preuve de travail.



Preuve de travail

- Une preuve de travail est la résolution d'un problème difficile, mais où il est facile de vérifier que la solution obtenue est correcte.
- Comme les sudokus par exemple : il suffit de dix secondes pour vérifier qu'une grille est remplie correctement, par contre il a fallu plus de dix minutes pour le résoudre.



Fonction de hachage

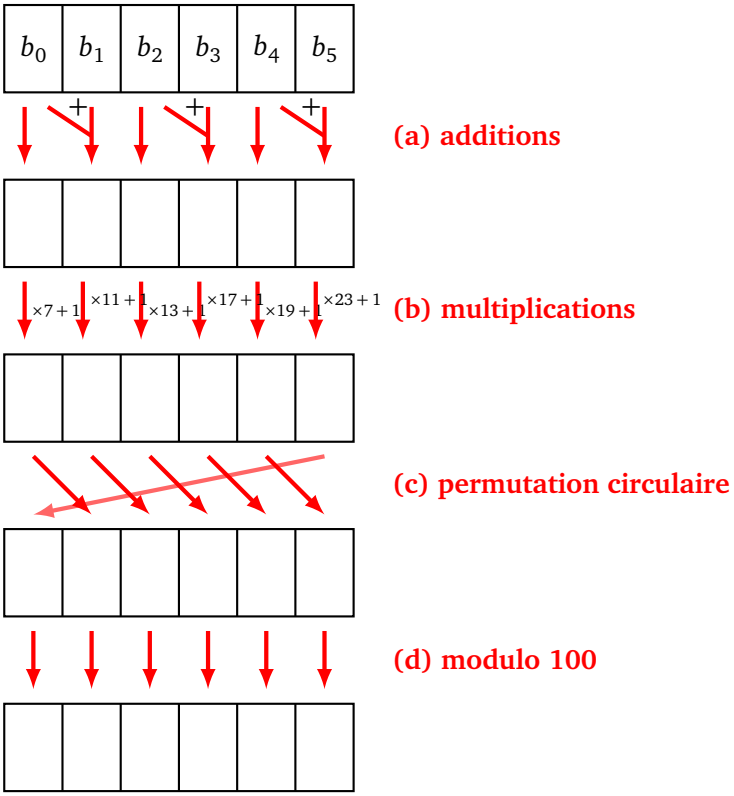
- À partir d'un long message nous calculons une courte empreinte.
- Il est difficile de trouver deux messages différents ayant la même empreinte.
- Ici notre message est une liste d'entiers (entre 0 et 99) de longueur un multiple quelconque de $N = 6$.
- Son empreinte (ou *hash*) sera une liste de longueur $N = 6$.
- Exemple : `[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]` a pour empreinte :
  `[10, 0, 58, 28, 0, 90]`
- Exemple : `[1, 1, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]` a pour empreinte :
  `[25, 14, 29, 1, 19, 6]`

L'idée est de mélanger les nombres par bloc de $N = 6$ entiers, puis de combiner ce bloc au suivant et de recommencer, jusqu'à obtenir un seul bloc.

Un tour

Pour un bloc $[b_0, b_1, b_2, b_3, b_4, b_5]$ de taille $N = 6$, *faire un tour* consiste à faire les opérations suivantes :

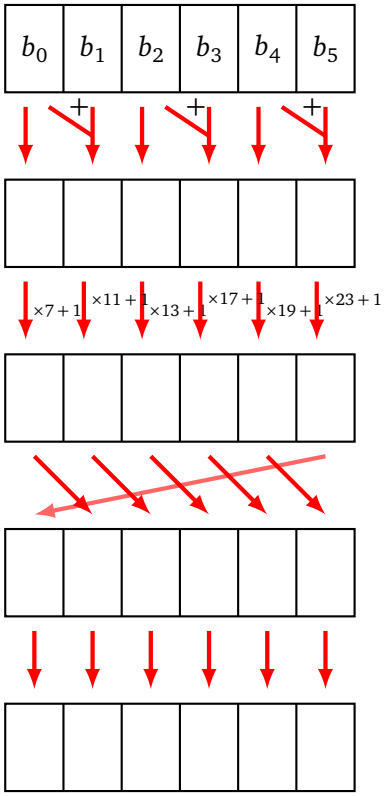
1. On additionne certains entiers :
$$[b'_0, b'_1, b'_2, b'_3, b'_4, b'_5] = [b_0, b_1 + b_0, b_2, b_3 + b_2, b_4, b_5 + b_4]$$
2. On multiplie ces entiers par des nombres premiers (dans l'ordre $7, 11, 13, 17, 19, 23$) et on rajoute 1 :
$$[b''_0, b''_1, b''_2, b''_3, b''_4, b''_5] = [7 \times b'_0 + 1, 11 \times b'_1 + 1, 13 \times b'_2 + 1, 17 \times b'_3 + 1, 19 \times b'_4 + 1, 23 \times b'_5 + 1]$$
3. On effectue une permutation circulaire (le dernier passe devant) :
$$[b'''_0, b'''_1, b'''_2, b'''_3, b'''_4, b'''_5] = [b''_5, b''_0, b''_1, b''_2, b''_3, b''_4]$$
4. On réduit chaque entier modulo 100 afin d'obtenir des entiers entre 0 et 99.



Partant du bloc $[0,1,2,3,4,5]$, on a donc successivement :

1. additions : $[0,1,2,5,4,9]$
2. multiplications : $[7\times 0+1, 11\times 1+1, 13\times 2+1, 17\times 5+1, 19\times 4+1, 23\times 9+1] = [1,12,27,86,77,208]$
3. permutation : $[208,1,12,27,86,77]$
4. réduction modulo 100 : $[8,1,12,27,86,77]$

Vérifie que le bloc $[1,1,2,3,4,5]$ est transformé en $[8,8,23,27,86,77]$.



Dix tours

Pour bien mélanger chaque bloc, itère dix fois les opérations précédentes. Après 10 tours :

- le bloc $[0, 1, 2, 3, 4, 5]$ devient $[98, 95, 86, 55, 66, 75]$,
- le bloc $[1, 1, 2, 3, 4, 5]$ devient $[18, 74, 4, 42, 77, 42]$.

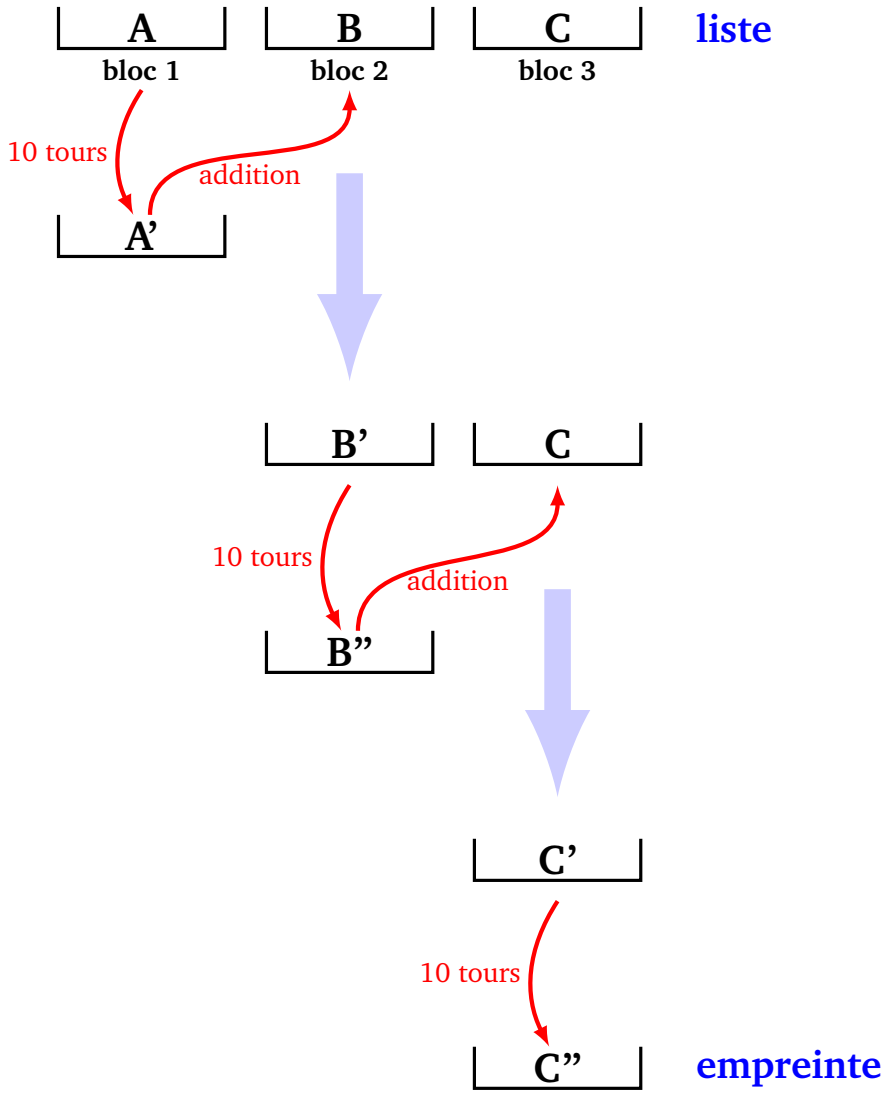
Deux blocs proches sont transformés en deux blocs très différents !

Hachage d'une liste

Partant d'une liste de longueur un multiple de $N = 6$, on la découpe en blocs de longueur 6 et on calcule l'empreinte de cette liste selon l'algorithme suivant :

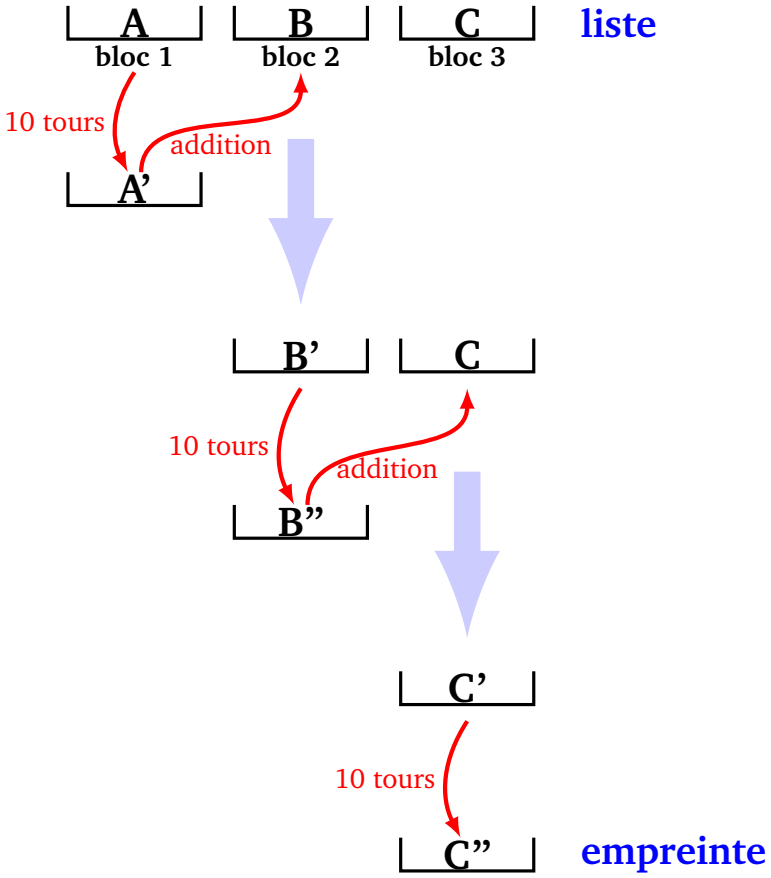
- On extrait le premier bloc de la liste, on effectue 10 tours de mélange.
- On ajoute terme à terme (et modulo 100), le résultat de ce mélange au second bloc.
- On recommence en partant du nouveau second bloc.
- Lorsqu'il ne reste plus qu'un bloc, on effectue 10 tours de mélange, le résultat est l'empreinte de la liste.

Voici le schéma d'une situation avec trois blocs : dans un premier temps il y a trois blocs (A,B,C) ; dans un second temps il ne reste plus que deux bloc (B' et C) ; à la fin il ne reste qu'un bloc (C") : c'est l'empreinte !



Exemple avec la liste $[0, 1, 2, 3, 4, 5, 1, 1, 1, 1, 1, 1, 10, 10, 10, 10, 10, 10]$.

- Le premier bloc est $[0, 1, 2, 3, 4, 5]$, son mélange à 10 tours est $[98, 95, 86, 55, 66, 75]$.
- On ajoute ce mélange au second bloc $[1, 1, 1, 1, 1, 1]$.
- La liste restante est maintenant $[99, 96, 87, 56, 67, 76, 10, 10, 10, 10, 10, 10]$.
- On recommence. Le nouveau premier bloc est $[99, 96, 87, 56, 67, 76]$, son mélange à 10 tours vaut $[60, 82, 12, 94, 6, 80]$, on l'ajoute au dernier bloc $[10, 10, 10, 10, 10, 10]$ pour obtenir (modulo 100) $[70, 92, 22, 4, 16, 90]$.
- On effectue un dernier mélange à 10 tours pour obtenir l'empreinte : $[77, 91, 5, 91, 89, 99]$.



Preuve de travail - Minage

On va construire un problème compliqué à résoudre, pour lequel, si quelqu'un nous donne la solution, alors il est facile de vérifier qu'elle convient.

Problème à résoudre. On nous donne une liste, il s'agit de trouver un bloc tel que, lorsque qu'on le rajoute à la liste, cela produit un hachage commençant par des zéros. Plus précisément étant donné une liste `liste` et un objectif maximal `Max`, il s'agit de trouver un bloc `preuve` qui, concaténé à la liste puis haché, est plus petit que la liste `Max`, c'est-à-dire :

`hachage(liste + preuve)` plus petit que `Max`

La liste est de longueur quelconque (un multiple de $N = 6$), la preuve est un bloc de longueur N, l'objectif est de trouver une liste commençant par des 0.

Exemple.
soit la `liste = [0,1,2,3,4,5]` et `Max = [0,0,7]`. Quel bloc `preuve` puis-je concaténer à `liste` pour résoudre mon problème ?

- — `preuve = [12, 3, 24, 72, 47, 77]` convient
  — car concaténé à notre liste cela donne `[0,1,2,3,4,5,12, 3, 24, 72, 47, 77]`
  — et le hachage de toute cette liste vaut `[0, 0, 5, 47, 44, 71]`
  — qui commence par `[0,0,5]` plus petit que l'objectif.

- — `preuve = [0, 0, 2, 0, 61, 2]` convient aussi
  — car après concaténation on a `[0,1,2,3,4,5,0, 0, 2, 0, 61, 2]`
  — dont le hachage donne `[0, 0, 3, 12, 58, 92]`.

- `[97, 49, 93, 87, 89, 47]` ne convient pas, car après concaténation puis hachage on obtient `[0, 0, 8, 28, 6, 60]` qui est plus grand que l'objectif voulu.

Vérification (facile).
Programme une fonction `verification_preuve_de_travail(liste,preuve)` qui renvoie vrai si la solution `preuve` proposée convient pour `liste`.

Recherche de solution (difficile).
Programme une fonction `preuve_de_travail(liste)` qui cherche un bloc `preuve` solution à notre problème pour la liste donnée.

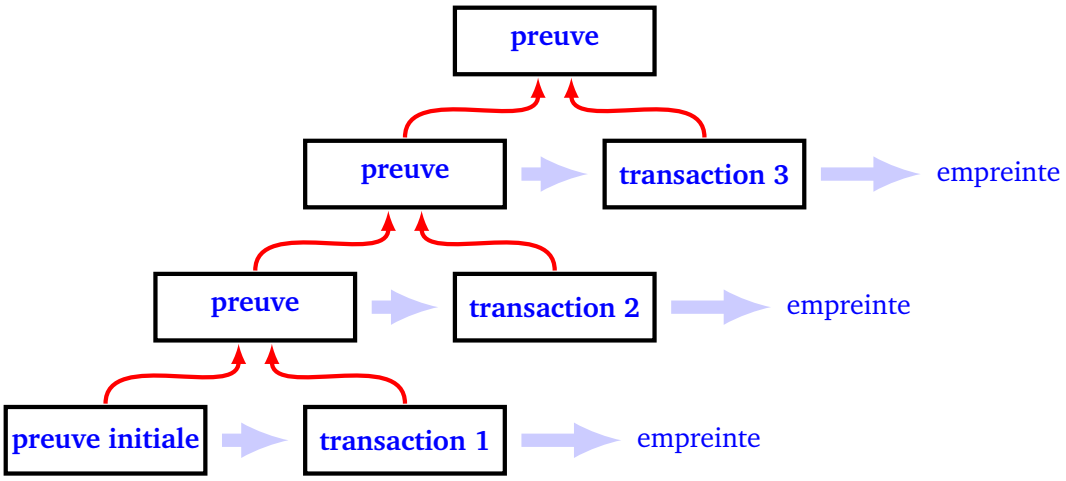
Indications.

- La méthode la plus simple est de prendre un bloc `preuve` de nombres au hasard et de recommencer jusqu'à trouver une solution.
- Tu peux aussi tester systématiquement tous les blocs en commençant avec `[0,0,0,0,0,0]`, puis `[0,0,0,0,0,1]`... et t'arrêter au premier qui convient.
- Tu ajustes la difficulté du problème en changeant l'objectif : facile avec `Max = [0,0,50]`, moyen avec `Max = [0,0,5]`, difficile avec `Max = [0,0,0]`, trop difficile avec `Max = [0,0,0,0]`.
- Comme il existe plusieurs solutions, tu n'obtiens pas nécessairement la même solution à chaque recherche.

Temps de calcul.

- Compare le temps de calcul d'une simple vérification par rapport au temps de recherche d'une solution.
- Choisis l'objectif `Max` de sorte que la recherche d'une preuve de travail nécessite environ entre 30 et 60 secondes de calculs.
- Pour le *bitcoin*, ceux qui calculent des preuves de travail sont appelés les *mineurs*.
- Le premier qui trouve une preuve gagne une récompense.
- La difficulté du problème est ajustée de sorte que le temps de calcul mis par le gagnant (parmi l'ensemble de tous les mineurs) pour trouver une solution, soit d'environ 10 minutes.

Tes *bitcoins*



Initialisation et ajout d'une transaction.

1. Variable globale `Livre` initialisée à `Livre = [ [0,0,0,0,0,0] ]` .
2. Une *transaction* est une chaîne de caractères comprenant un nom et la somme à ajouter (ou à retrancher) à son compte. Par exemple `"Abel +25"` ou `"Barbara -45"`.

   Programme une fonction `ajout_transaction(transaction)` qui ajoute la chaîne de caractère `transaction` à la liste `Livre`.

   Par exemple après l'initialisation `ajout_transaction("Camille +100")`, `Livre` vaut `[ [0, 0, 0, 0, 0, 0], "Camille +100" ]`.

   Attention, pour pouvoir modifier `Livre` il faut commencer la fonction par : `global Livre`.

Calcul et ajout d'une preuve de travail.

Dès qu'une transaction est ajoutée, il faut calculer et ajouter au livre de compte une preuve de travail. Programme une fonction `minage()`, sans paramètre, qui ajoute une preuve de travail au livre.

Voici comment faire :

- On prend la dernière transaction `transaction`, on la transforme en une liste d'entiers par une fonction `phrase_vers_liste()` (on associe à un caractère sur ordre ASCII/unicode modulo 100).
- On prend la preuve de travail `prec_preuve` située juste avant cette transaction.
- On forme la liste `liste` composée d'abord des éléments de `prec_preuve`, puis des éléments de la liste d'entiers obtenue en convertissant la chaîne `transaction`.
- On calcule une preuve de travail de cette liste.
- On ajoute cette preuve au livre de compte.

Par exemple si le livre se termine par :

```
[3, 1, 4, 1, 5, 9], "Abel +35"
```

alors après calcul de la preuve de travail le livre se termine par exemple par :

```
[3, 1, 4, 1, 5, 9], "Abel +35", [32, 17, 37, 73, 52, 90]
```

On rappelle que la preuve de travail n'est pas unique et qu'en plus elle dépend de l'objectif `Max`.

Vérification du livre de compte.

Une seule personne à la fois ajoute une preuve de travail. Par contre tout le monde a la possibilité de vérifier que la preuve proposée est correcte (et devrait le faire).

Écris une fonction `verification_livre()`, sans paramètre, qui vérifie que la dernière preuve ajoutée au `Livre` est valide.

Exemple.

Écris un livre de compte qui correspond aux données suivantes :

- On prend `Max = [0,0,5]` et au départ `Livre = [ [0,0,0,0,0,0] ]`.
- `"Alfred -100"` (Alfred doit 100 *bitcoins*).
- Barnabé en reçoit 150.
- Chloé gagne 35 *bitcoins*.