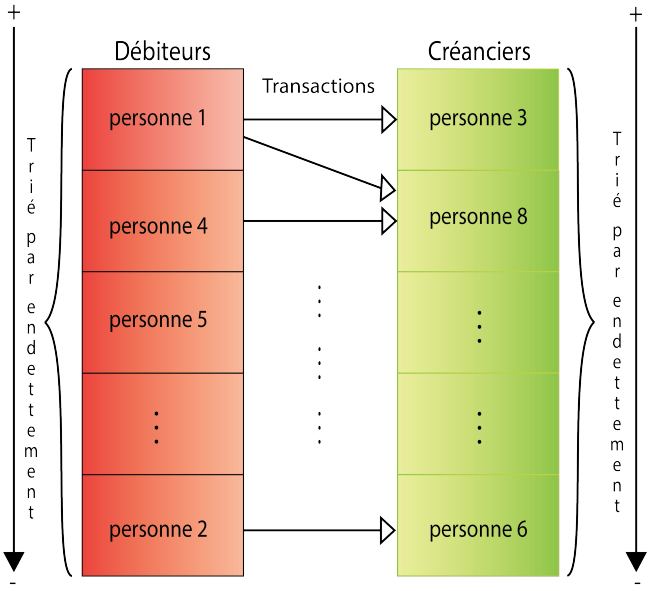
Mini-projet 2 - Conception dalgorithme : Troycount

1. Explication de l'algorithme



2. Choix d'implémentation

Afin d'implémenter l'arbre syntaxique abstrait, nous avons utilisé une structure chainée. Ce type d'implémentation a été préféré à un tableau dynamique car la complexité spatial avec un tableau est de $\Theta(2^n)$ alors qu'avec une structure chainée la complexité spacial est de $\Theta(n)$.

De plus, une structure chainée était un choix obligatoire vu la contrainte d'utiliser des méthodes récursives.

3. Complexité des différentes méthodes

Soit n, le nombre d'opérations et de nombres : n représente donc la taille du problème à résoudre.

3.1 TreeBuilderIF.build()

Complexité temporelle en $\Theta(n)$ car :

1. L'expression sera toujours parcourue entièrement 1 fois, à l'aide d'une boucle.
2. La boucle principale ne contient que des opérations de complexité $\Theta(1)$.

On en déduit donc que le temps d'exécution augmentera de façon linéaire par rapport à la taille du problème.

3.2 ExprIF.getReducedTree()

Complexité temporelle en $\mathcal{O}(\log(n))$ car :

1. Complexité temporelle en $\mathcal{O}(h)$, où h est la hauteur de larbre
2. L'arbre est équilibré donc h $\in \Theta(\log_2(n))$

Donc,

— Meilleur cas : $\Theta(1)$ (on tombe sur la racine)
— Pire cas : $\Theta(\log(n))$
— En général : $\mathcal{O}(\log(n))$

3.3 ExprIF.toString()

Complexité temporelle en $\mathcal{O}(n)$ car on passe, dans le pire des cas, 3x par chaque noeud de l'arbre (parcours d'Euler). On a donc une complexité en $\mathcal{O}(3n)$. On peut "simplifier" le 3 car il s'agit d'une constante. On obtient donc $\mathcal{O}(n)$

— Meilleur cas : $\Theta(1)$ (L'arbre est entièrement simplifié)
— Pire cas : $\Theta(n)$
— En général : $\mathcal{O}(n)$

4. Difficultés rencontrées

— Quelques difficultés à implémenter une méthode récursive
— Choix d'implémentations

Annexe

Troycount.balance()

```
1  public Transaction[] balance() {
2      Person[] persons = generateTabPersons();
3      Arrays.sort(persons);
4
5      int firstSplitIndex = 0;
6      int secondSplitIndex = 0;
7
8      for (int i = 0; persons[i].getBalance() >= 0; i++) {
9          secondSplitIndex++;
10         if (persons[i].getBalance() > 0) {
11                 firstSplitIndex++;
12         }
13      }
14
15     Person[] debtors = Arrays.copyOfRange(persons, 0, firstSplitIndex);
16     Person[] creditors = Arrays.copyOfRange(persons, secondSplitIndex,
           persons.length);
17
18     reverse(creditors);
19
20     Stack<Transaction> transactions = new Stack<Transaction>();
21     int i = 0, j = 0;
22
23     while (i < debtors.length && j < creditors.length) {
24         if (debtors[i].getBalance() <= -creditors[j].getBalance()){
25             double transactionAmount = debtors[i].getBalance();
26             Transaction t = new Transaction(debtors[i].getId(),
27                                             creditors[j].getId(),
28                                             transactionAmount);
29             transactions.add(t);
30             System.out.println(t);
31             creditors[j].balanceAdd(transactionAmount);
32             debtors[i].balanceSub(transactionAmount);
33             i++;
34         } else {
35             double transactionAmount = -creditors[j].getBalance();
36             Transaction t = new Transaction(debtors[i].getId(),
37                                             creditors[j].getId(),
38                                             transactionAmount);
39             transactions.add(t);
40             System.out.println(t);
41             creditors[j].balanceAdd(transactionAmount);
42             debtors[j].balanceSub(transactionAmount);
43             j++;
44                 }
45         }
46     return transactions.toArray(new Transaction[transactions.size()]);
47 }
```

Troycount.generateTabPerson()

```
1  public Person[] generateTabPersons() {
2     Person[] persons = new Person[group_size];
3     for (int i = 0; i < group_size; i++) {
4        int personID = i + 1;
5        persons[i] = new Person(personID);
6     }
7     for (Spending aSpending  :spendings) {
8        int debitedPerson = aSpending.get_paid_by()
9        int[] chargedPerson = aSpending.get_paid_for();
10       int chargedPersons = chargedPerson.length;
11       double spendingAmount = aSpending.get_amount();
12
13       for (int person  :chargedPerson) {
14          double fixedCharge = aSpending.get_fixed_charges(person);
15          spendingAmount -= fixedCharge;
16       }
17
18       for (int person  :chargedPerson) {
19          persons[person - 1].balanceAdd(spendingAmount / chargedPersons);
20       }
21
22       persons[debitedPerson - 1].balanceSub(spendingAmount);
23    }
24    return persons;
25 }
```