Le Compte est Bon

Description de l'algorithme employé

Introduction

La recherche des solutions du jeu « Le Compte est Bon » est assez aisée pour un joueur humain, du moins pour les solution « évidentes », ce critère subjectif étant laissé à l'appréciation de chacun. Pour une recherche automatique il en va autrement, surtout si on demande au programme de rechercher toutes les solutions possibles, ou la (ou les) meilleures solution (s) approchée(s) dans les cas où il n'y aurait pas de solution.

Il serait intéressant (et judicieux ?) de pouvoir imiter le raisonnement humain mais cette tâche me semble difficile à mettre en place dans un premier temps aussi je me contenterai de profiter au maximum des possibilités de la machine en termes de rapidité pour explorer de façon brute l'espace des solutions. Bien entendu cette approche marchera pour un petit nombre de plaques (6 dans le cas du jeu télévisé) mais pourrait s'avérer peu efficace dans la généralisation du problème avec N_{GRAND} plaquettes.

Voyons maintenant comment modéliser le problème et partir à la recherche des solutions. Les données sont les suivantes : nous avons N = 6 nombres de départ, une valeur cible C et on cherche à former à partir des 4 opérations élémentaires {+, -,× ,÷}une suite de calculs menant de ces N nombres jusqu'au résultat C.

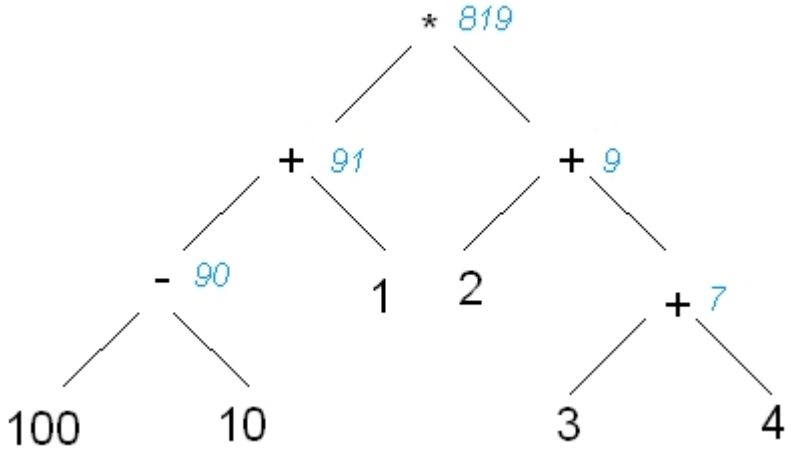
Par exemple, à partir des nombres 100, 2,,1, 10, 4 et 3, former 819. Nous pouvons faire les calculs de l'exemple (1) :

100 - 10 = 90
90 + 1 = 91
2 + 3 = 5
5 + 4 = 9
91 × 9 = 819 (1)

Cette représentation, avec les calculs intermédiaires à la droite est très commode si on utilise le papier et le crayon. Dans l'ordinateur, on va procéder autrement. La suite de calculs (1) est équivalente au calcul parenthésé :

((100 – 10) + 1) × ((2+3) + 4) = 819 (2)

Les expressions parenthésées se représentent très commodément sous forme d'arbre : ainsi le calcul (2) peut se représenter ainsi :



Dans cet arbre, les feuilles sont les nombres sur lesquels on bâtit le raisonnement, les noeuds internes sont les opérations que l'on effectue. La lecture de cet arbre se fait en partant des feuilles

(en bas) en remontant jusque la racine. Les étiquettes sur les noeuds (en caractères bleu-italique) sont les résultats des calculs intermédiaires.

L'algorithme présenté ici va donc consister en la création de tous les arbres possibles utilisant les six nombres donnés au départ ainsi que les quatres opérations arithmétiques élémentaires.

Les règles dui jeu « Le Compte est Bon » imposent des contraintes pour le calcul : la soustraction doit toujours retourner une valeur positive, et la division doit rester entière et de reste nul. Enfin on peut proposer des solutions qui utilisent moins de nombres (5 par exemple). Si on ne trouve pas de solution, on peut éventuellement proposer une solution approchée.

Voyons maintenant comment représenter les expressions mathématiques, et les solutions potentielles. Ensuite nous verrons comment passer d'une solution à une autre, puis enfin comment égrener l'ensemble des solutions possibles.

Représentation des expressions mathématiques

Les expressions athématiques parenthésées sont représentées à l'aide d'arbres binaires. Les noeuds de cet arbre peuvent être des noeuds **terminaux** (externes = les feuilles) qui portent une valeur, celle de la plaquette tirée au sort dans le jeu. Ou bien ils sont des noeuds **internes**, et sont dans ce cas porteurs d'une opération élémentaire {+, -,× ,÷} qu'ils appliquent aux valeurs de leur noeuds fils. La définition de la classe Tnoeud est essentiellement ceci :(en langage Pascal-objet)

```
type Tnoeud = class(TObject)
   operation : Toperation;
   valeur : int64;
   OperandeGauche : Tnoeud;
   OperandeDroite : Tnoeud;
   // puis les constructeurs et destructeurs de la classe
   ...
end;
```

Le premier champ, operation est un type défini par l'utilisateur et qui vaut (les constantes) ;

```
TOperation = (toVal, toPlus, toMoins, toMult, toDiv);
```

Ce type sert à indiquer quelle opération appliquer sur les noeuds fils OperandeGauche et OperandeDroite, mais également sert à **discriminer** les noeuds-feuilles des noeuds internes : pour cela il suffit d'utiliser la constante **`toVal`** et à ce moment (et dans ce cas seulement) on ne considère plus les fils OperandeGauche et OperandeDroite que l'on laisse à **`nil`** (valeur nulle).

Le second champ est la valeur de ce noeud de type entier très-long, **`int64`** : c'est soit la valeur de la plaquette pour le cas des noeuds **`toVal`**, soit le résultat de l'opération faite à partir des valeurs des noeuds fils dans les autres cas.
Les champs OpérandeGauche et OpérandeFroite valent soit **`nil`**, soit contiennent les références vers les noeuds fils.

Bien sûr tout ceci n'est que la cuisine interne des expressions mathématiques et ne prend son sens qu'une fois défini le niveau supérieur d'arbre-solution, dont voici l'en-tête de classe

```
TArbreSolution = class(TObject)
   Racine : Tnoeud;
   ...
end;
```

Un arbre-solution est donc constitué d'un noeud particulier qu'on appelle la **racine**.

Il dispose des constructeurs suivants :

```
constructor CreerSurValeur(Valeur : Int64);

constructor Cloner(ASource : TArbreSolution);

constructor Combiner(ArbreGauche : TArbreSolution;
                     Operation : TOperation;
                     ArbreDroite : TArbreSolution);
```

qui permettent respectivement :

- ✔ de créer un arbre contenant une seule feuille (en même temps racine) sur une valeur donnée
- ✔ de créer un arbre à partir d'un autre (clonage de la structure)
- ✔ de créer un arbre en **combinant** deux sous arbres existants à l'aide d'une opération à préciser par l'utilisateur.

La méthode de construction par combinaison est l'une des pièces maîtresses de l'algorithme de recherche des solutions de Le Compte est Bon : c'est elle qui permet de fabriquer des solutions élaborées à partir de morceaux de solutions plus simples.

Fig. 2 : Combinaison d'arbres

Dans cet exemple, au départ l'arbre de gauche comporte 3 noeuds (2 feuilles + 1 interne) et sa racine porte la valeur 90, et l'arbre de droite comporte 1 noeud (racine et feuille) de valeur 1. Le résultat de la combinaison de ces deux arbres par l'opération « + » est un arbre à 5 noeuds (3 feuilles + 2 internes) de valeur à la racine 90 + 1 = 91.

Les arbres-solutions disposent en outre des méthodes **`evaluer`** qui renvoie la valeur de la racine, et **`toStringList`** qui sort une liste de chaîne de caractères, trace de la solution calculée, en vue de l' affichage à l'écran pour les besoins de l'utilisateur.

Paires	***0***	***1***	***2***	***3***	***4***	***5***
0	***-***	***0-1***	***0-2***	***0-3***	***0-4***	***0-5***
1	*sym*	***-***	***1-2***	***1-3***	***1-4***	***1-5***
2	*sym*	*sym*	***-***	***2-3***	***2-4***	***2-5***
3	*sym*	*sym*	*sym*	***-***	***3-4***	***3-5***
4	*sym*	*sym*	*sym*	*sym*	***-***	***4-5***
5	*sym*	*sym*	*sym*	*sym*	*sym*	***-***