ASD2 : Labo 1

Encodage de Huffman – Compresse() 1/5

```
vector<bool> ArbreHuffmanBranche::compresse(char caractere) {

   vector<bool> x(0);
   vector<bool> total(0);

   if (not gauche or not droite) throw SousArbreVide();

   //si le caractère est présent dans la partie droite
   if (droite->contient(caractere)) {
      x = droite->compresse(caractere);

      //total à la taille de x en plus d'une case pour y ajouter le bit courant
      total.resize(1 + x.size());

      total.at(0) = true;

      //si le vecteur retourner est vide (en dessous il y a la feuille)
      if (total.size() == 1)
         return total;
      else {
         for (int i = 0; i < x.size(); i++)
            total.at(i + 1) = x.at(i);
         return total;
      }
      //si le caractère est dans la partie gauche
      ...
   } else {
      throw ProblemeEncodage();
   }
}
```

ASD2 : Labo 1

Encodage de Huffman – Compresse() 2/5

```
vector<bool> ArbreHuffmanBranche::compresse(char caractere) {

   vector<bool> x(0);
   vector<bool> total(0);

   if (not gauche or not droite) throw SousArbreVide();

   //si le caractère est présent dans la partie droite
   if (droite->contient(caractere)) {
      x = droite->compresse(caractere);

      //total à la taille de x en plus d'une case pour y ajouter le bit courant
      total.resize(1 + x.size());

      total.at(0) = true;

      //si le vecteur retourner est vide (en dessous il y a la feuille)
      if (total.size() == 1)
         return total;
      else {
         for (int i = 0; i < x.size(); i++)
            total.at(i + 1) = x.at(i);
         return total;
      }
      //si le caractère est dans la partie gauche
      ...
   } else {
      throw ProblemeEncodage();
   }
}
```

x contient tous les bits suivants.

Il suffit d'y ajouter le courant

```
x.insert(x.begin(), true);
return x;
```

ASD2 : Labo 1

Encodage de Huffman – Compresse() 3/5

```
std::vector<bool> compresse(char c) {
        …
        std::vector<bool> tmp;

        if(gauche->contient(c))  {
                tmp = gauche->compresse(c);
                tmp.insert(tmp.begin(), true);
                return tmp;
        }
        else if(droite->contient(c)) {
                tmp = droite->compresse(c);
                tmp.insert(tmp.begin(), false);
                return tmp;
        }
        else {
                 throw ProblemeEncodage();
        }
}
```

ASD2 : Labo 1

Encodage de Huffman – Compresse() 4/5

```
std::vector<bool> compresse(char c) {
        …
        std::vector<bool> tmp;

        if(gauche->contient(c))  {
                tmp = gauche->compresse(c);
                tmp.insert(tmp.begin(), true);
                return tmp;
        }
        else if(droite->contient(c)) {
                tmp = droite->compresse(c);
                tmp.insert(tmp.begin(), false);
                return tmp;
        }
        else {
                 throw ProblemeEncodage();
        }
}
```

Création et initialisation d'un vecteur vide.

On remplace le vecteur par celui retourné par la méthode compresse() de la récursion.

Le vecteur créé en début de méthode et jamais utilisé est supprimé.

ASD2 : Labo 1

Encodage de Huffman – Compresse() 5/5

```
std::vector<bool> compresse(char c) {
        …
        if(gauche->contient(c))  {
                std::vector<bool> tmp = gauche->compresse(c);
                tmp.insert(tmp.begin(), true);
                return tmp;
        }
        else if(droite->contient(c)) {
                std::vector<bool> tmp = droite->compresse(c);
                tmp.insert(tmp.begin(), false);
                return tmp;
        }
        else {
                 throw ProblemeEncodage();
        }
}
```

ASD2 : Labo 1

Encodage de Huffman – Decompresse() 1/5 – Problème 1

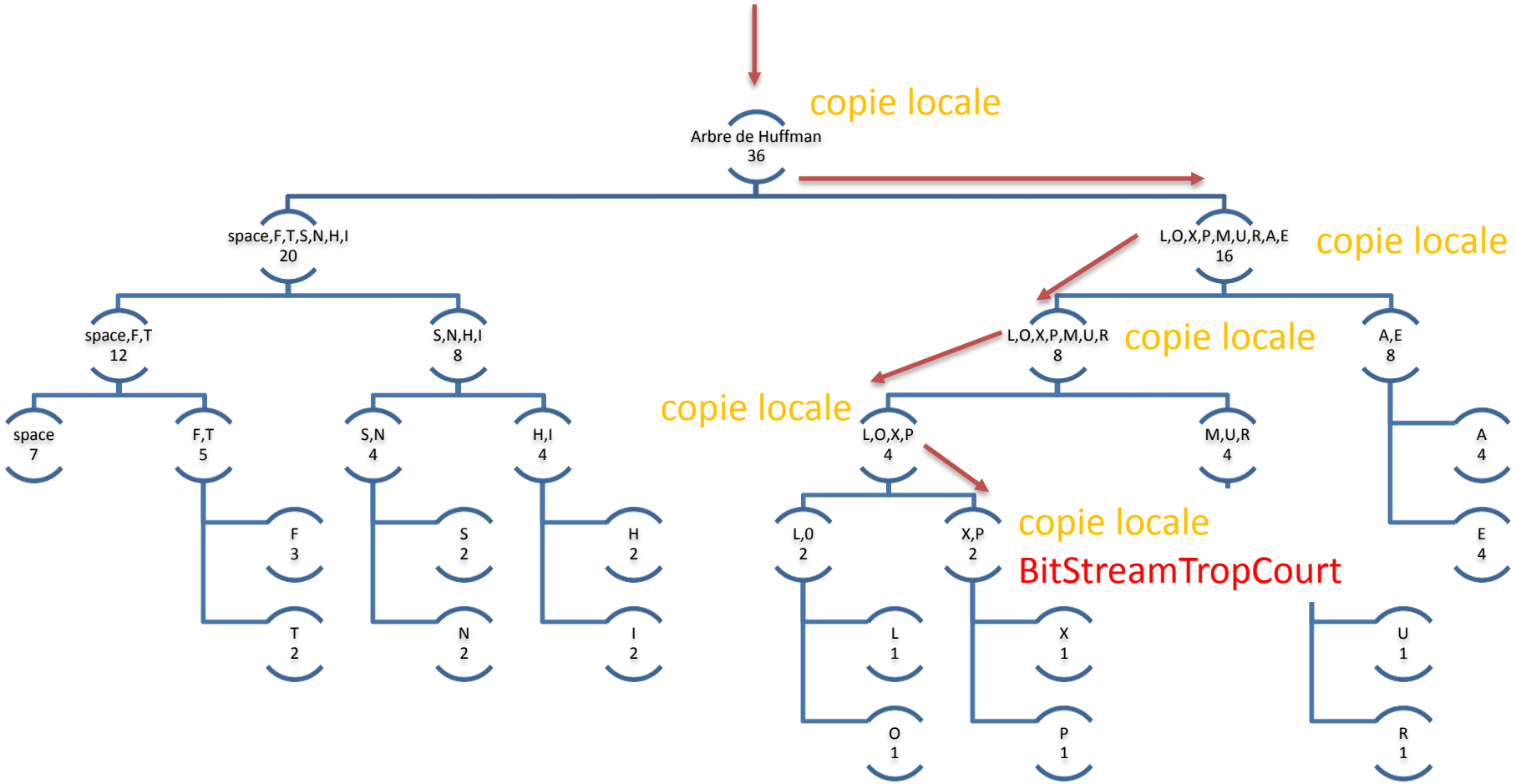
```
char ArbreHuffmanBranche::decompresse(std::vector<bool>* bits) {
        std::vector<bool> temp = *bits;
        if(gauche != nullptr || droite != nullptr){
                try{
                        if(bits->empty()){
                                throw BitStreamTropCourt();
                        } else {
                                if(bits->at(0)){
                                        bits->erase(bits->begin());
                                        return this->gauche->decompresse(bits);
                                } else {
                                        bits->erase(bits->begin());
                                        return this->droite->decompresse(bits);
                                }
                        }
                } catch(BitStreamTropCourt) {
                        *bits = temp;
                        throw BitStreamTropCourt();
                }
        } else {
                throw SousArbreVide();
        }
}
```

Copie du buffer

Remplace le buffer par la copie

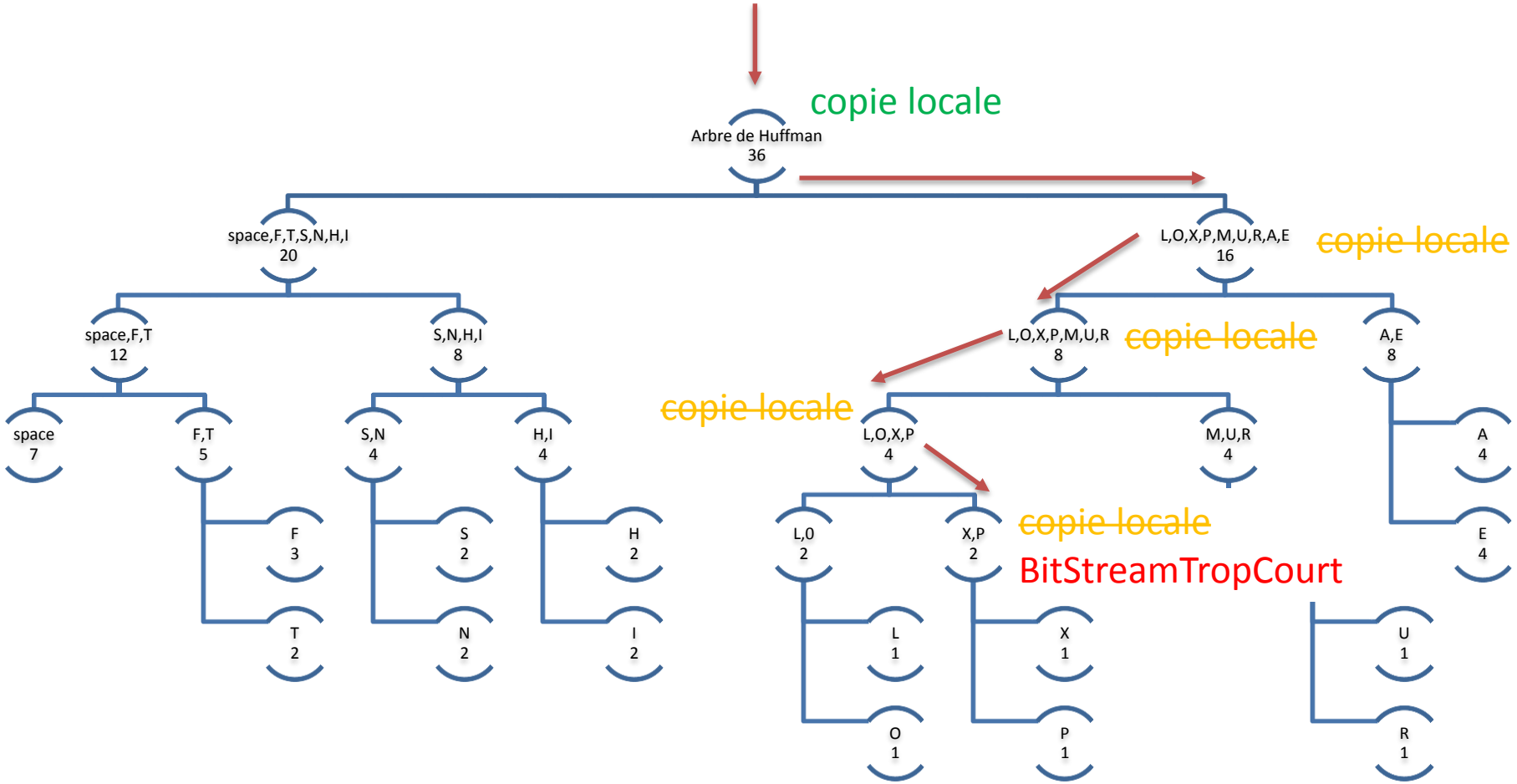
ASD2 : Labo 1

Encodage de Huffman – Decompresse() 2/5 – Problème 1



ASD2 : Labo 1

Encodage de Huffman – Decompresse() 3/5 – Problème 1



ASD2 : Labo 1

Encodage de Huffman – Decompresse() 4/5 – Problème 2

```
char ArbreHuffmanBranche::decompresse(std::vector<bool>* bits) {

   static ArbreHuffmanBranche* sauvegarde; // Sauvegarde de l'arbre courant lors de la levée de l'exception
   static bool restauration = false;
  // Ces champs sont déclarés statique car leurs états doivent être accessible sur n'importe quelle arbre.

   if (bits->size() == 0) { // Plus de bits
      sauvegarde = this;
      restauration = true;
      throw BitStreamTropCourt();
   }

   // Consommation d'un bit
   bool b = bits->front();
   bits->erase(bits->begin());

   if (restauration) {
      restauration = false;

      // On part depuis l'arbre sauvegardé au lieu de recommencer à la racine
      if (b)
         sauvegarde->gauche->decompresse(bits);
      else
         sauvegarde->droite->decompresse(bits);
   } else {
      if (b)
         gauche->decompresse(bits);
      else
         droite->decompresse(bits);
   }
}
```

En cas de buffer vide:

- On passe en mode restauration
- On stock le nœud courant

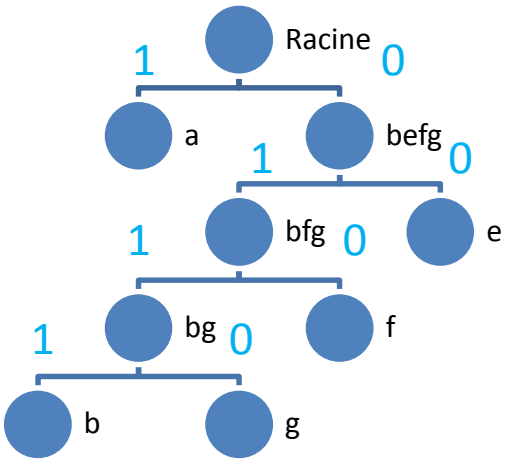
Restauration:
On continue la récursion à partir du nœud sauvegardé

Récursion normale

ASD2 : Labo 1

Encodage de Huffman – Decompresse() 5/5 – Problème 2

- 2 problèmes:
  - On ne respecte plus le contrat, le buffer ne doit pas être modifié en cas d'exception
  - On introduit une notion d'état au décodage



decompresse([1]) → 'a'

decompresse([0;1;1]) → exception

decompresse([1]) → 'b'