POLYTECH LILLE

Université Lille1 Sciences et Technologies

Département IMA / 3A (S6) **Structures de données 2010/2011**

Devoir surveillé - 2h

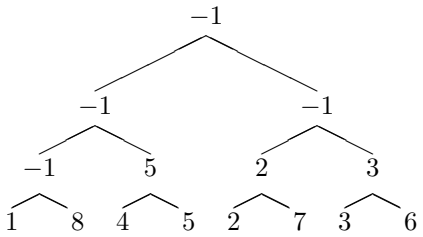
Machines et livres interdits, traducteurs, documents de cours et de TD autorisés

A – Exercice : Arbres Binaires pour un tournoi (5pt/20min)

Les algorithmes de cette partie seront écrits en pseudo-code algorithmique. Ne pas passer plus d'une demi-heure !

Les matchs d'un tournoi de tennis à $N = 2^p$ joueurs peuvent être décrits à l'aide d'un arbre binaire complet dont les noeuds sont des entiers (les joueurs sont numérotés de 1 à N). Les numéros des joueurs sont rangés initialement dans les feuilles de l'arbre, ce qui détermine le tableau des matchs à jouer. Les noeuds internes contiennent soit la valeur -1 pour un match non encore effectué soit le numéro du joueur ayant remporté le match disputé entre les deux joueurs stockés dans les deux noeuds fils.

Voici un exemple :



Dans cet exemple, 8 joueurs sont numérotés de 1 à 8 :

- Le joueur 2 a remporté le match disputé avec le joueur 7
- Le joueur 3 a remporté le match disputé avec le joueur 6
- Le joueur 2 doit rencontrer le joueur 3, le match n'a pas encore eu lieu

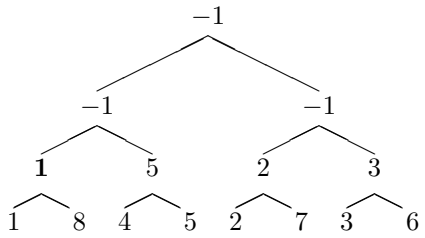
1. Faire les déclarations nécessaires pour représenter de tels arbres. Écrire un algorithme qui, étant donné l'arbre binaire d'un tournoi, affiche les matchs disputés et leur vainqueur. Pour l'arbre ci-dessus, l'affichage produit par exemple :

   ```
   match entre 4 et 5 : vainqueur 5
   match entre 2 et 7 : vainqueur 2
   match entre 3 et 6 : vainqueur 3
   ```

2. Écrire un algorithme permettant de stocker le résultat d'un match dans l'arbre du tournoi. Les données sont :
   - `j1` et `j2` : les numéros des 2 joueurs disputant le match
   - `j`, le numéro du joueur ayant remporté le match
   - l'arbre, bien sûr !

   L'algorithme doit donc ranger la valeur `j` dans le noeud **pas forcément une feuille !** de l'arbre ayant pour fils gauche `j1` et pour fils droit `j2`. On suppose que l'arbre est correctement construit et que les données de l'algorithme sont correctes (il existe dans l'arbre, un noeud ayant `j1` pour fils gauche et `j2` pour fils droit).

   Par exemple, avec `j1=1, j2=8, j=1`, l'algorithme doit modifier l'arbre de l'exemple de la façon suivante :

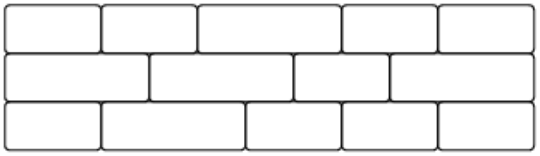


B – Problème de BTP (le reste)

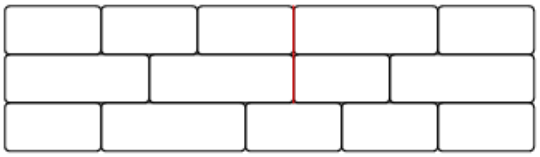
Les algorithmes de cette partie seront écrits en C

D'après Projet Euler 215 et Archives de 421, Polytechnique

On cherche à construire un mur à partir de briques horizontales de taille 2x1 et 3x1. Un mur est correctement construit si la jointure verticale entre deux briques ne se trouve jamais immédiatement au dessus d'une autre jointure verticale. Ainsi le mur suivant (de hauteur 3 et de longueur 11) est correctement construit :



Mais pas celui là :



L'objectif de ce problème est de dénombrer le nombre de façons différentes de construire un mur de longueur ℓ et de hauteur h.

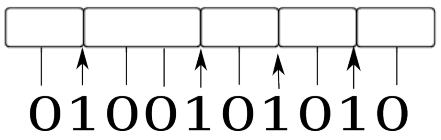
Codage du problème et fonctions faciles

Chaque rangée de briques (une ligne) de longueur ℓ va être codée comme un tableau d'entiers (0 ou 1) de taille $\ell - 1$. Par exemple, la rangée de taille 11 :

est codée dans le tableau de taille 10 :

0	1	0	0	1	0	1	0	1	0

Le $i$ème chiffre du tableau indique si au niveau de la $i$ème unité de longueur il y a ou non une jointure entre deux briques. (1 si il y a une jointure, 0 sinon). On n'inclut **pas** les deux extrémités :



Question 1 Dessiner la rangée de briques correspondant au tableau suivant :

0	1	0	0	1	0	0

De quelle taille est cette rangée ?

Maintenant, on suppose fixée à l'avance la longueur des murs que l'on souhaite compter. Une macro définit la constante `TAILLEMAX` comme étant égale à $\ell - 1$. Les rangées seront donc des tableaux de taille `TAILLEMAX`.

Question 2 Déclarer le type rangée de briques (nommé `Rangee`)

Question 3 Écrire une fonction `void imprimerangee(Rangee rangee)` qui imprime une rangée de briques (sous la forme `011100`).

Deux rangées sont dites **compatibles** si elles peuvent être placées l'une sur l'autre (les jointures ne sont pas au même endroit !).

Question 4 Écrire une fonction `bool sontCompatibles(Rangee r1,Rangee r2)` qui retourne `true` si les deux rangées données sont compatibles, `false` sinon.

Résolution

Pour dénombrer le nombre de façons différentes de construire un mur de longueur ℓ (fixée par l'intermédiaire de la constante `TAILLEMAX`) et de hauteur h (passée en paramètre), on adopte le principe de résolution suivant :

- **Phase 1** Déterminer toutes les combinaisons possibles des briques permettant de construire une rangée de longeur ℓ. Par exemple, il n'y a que deux façons possibles pour les rangées de longueur 5 : une brique de longueur 2, puis une brique de longueur 3 ; et l'inverse. Les codages correspondants sont : `0100` et `0010`. Soit R l'ensemble de ces codages de rangées.
- **Phase 2** Pour chaque rangée r de R, déterminer le nombre de murs de hauteurs h dont la **première rangée de briques** (celle tout en bas) est r. Cette valeur est notée $count(r, h)$.
- **Phase 3** Calculer $\sum_{r \in R} count(r, h)$ la somme des $count(r, h)$ pour toutes les rangées possibles de longueur ℓ. C'est le résultat final attendu.

Pour la phase 1, on suppose écrite une fonction `allrows(int l)` qui retourne la liste de toutes les rangées possibles de longueur ℓ. Par exemple, `allrows(5)` retourne les deux rangées : `0100` et `0010`.

►Dans la suite, l'objectif est d'écrire une version optimisée de l'algorithme récursif de la phase 2.

L'algorithme récursif initial

Pour dénombrer les murs (phase 2 de la résolution), on utilise une fonction récursive `count(r, h)` qui détermine le nombre de murs dont la première rangée de briques est r et dont la hauteur est h. L'algorithme est le suivant :

```
count(r, h) =
   si h vaut 1, renvoyer 1
   sinon -- faire la somme des count(s,h-1) pour toute rangee s compatible avec r
         -- renvoyer ce résultat
```

L'objet de cette section est de comprendre cet algorithme et son implémentation en C, qui est donnée à l'annexe 1.

Question 5 Pourquoi la quantité `count(r,1)` vaut-elle 1 pour tout r ?

Question 6 Écrire la déclaration du type `ListeRangees` : liste chaînée de `Rangee`.

Question 7 On suppose $\ell = 5$. Développer sur votre copie les différentes étapes de calcul (avec les appels récursifs) de l'appel `count(allrows,2)`, `allrows` étant la liste comportant les deux tableaux codant les rangées `0100` et `0010`.

Remarque : la longueur n'est pas passée ici, elle est implicite, on suppose que les rangées de *allrows* et r sont bien de longueurs identiques ($\ell = TAILLEMAX + 1$).

Optimisation avec stockage des valeurs intermédiaires

Sur une entrée $\ell = 30$ et $h = 10$ l'algorithme précédent résout le problème, mais en un temps peu raisonnable. En effet, la fonction `count` est appelée de nombreuses fois pour des mêmes valeurs de (r, h) : par exemple, pour le calcul des murs de longueur 9 et de hauteur 3, une bête impression au début de chaque appel récursif nous montre que l'on appelle deux fois `count(00100100,2)`. On peut donc effectuer ce calcul une fois, puis stocker ce nombre dans une table.

Dans cette partie, nous allons donc stocker les valeurs intermédaires $(r, h, count(r, h))$ dans une table de hachage. **On rappelle qu'une table de hachage est un tableau t de listes. Pour stocker un élément, on commence par calculer son indice $hash(el)$ avec $hash$ une fonction de hachage qui retourne un entier, et ensuite on insère l'élément dans la liste qui se trouve à la case $t[hash(el)]$.**

Question 8 Écrire la déclaration d'un type `Triplet` pour stocker des valeurs $(r, h, nbmurs)$ ($nbmurs$ sera le résultat de $count(r, h)$). Écrire une fonction qui permet de tester l'égalité de deux triplets **Attention, r est une rangée**.

Question 9 Déclarer tout ce qu'il faut pour pouvoir parler de listes de Triplets. Écrire une fonction d'ajout en tête d'un Triplet dans une liste de Triplets.

Question 10 Déclarer le type `HashTable` comme étant un tableau (de taille `SIZEHASH`) de listes de Triplets. Écrire une fonction d'initialisation d'un élément de type `HashTable`.

Question 11 Écrire la fonction de hachage qui permettra de stocker le triplet $(r, h, nbmurs)$ à l'intérieur de la liste située à la case de la table de hachage d'indice i calculé par la formule : $i = (nbuns(r) * h) \mod SIZEHASH$. La fonction $nbuns(r)$ désigne le nombre de 1 de la rangée r, vous pouvez utiliser cette fonction sans la réécrire.

►À ce stade, il nous faut faire en sorte de pouvoir :
- rechercher un triplet donné dans une `HashTable` et récupérer le résultat (3ième champ du triplet).
- insérer un triplet si il n'existe pas dans la table.

Question 12 Écrire une ou plusieurs fonctions qui vous permettront de faire ces traitements. Ces fonctions seront utilisées dans la question suivante. **On n'écrira pas que du C, mais on fera également une vraie analyse algorithmique détaillée.**

Question 13 Compléter le code de l'annexe 2 (`count` et `count_all`) pour :
- déclarer et initialiser une `Hashtable` ;
- sauvegarder dans cette table chaque nouveau triplet calculé ;
- avant tout nouveau calcul, chercher si ce calcul a déjà été fait.

Annexe1 Code fourni, ne pas rendre cette feuille!

```
//definition de la constante pour des rangees de longueur 5
#define TAILLEMAX 4

//Fonction auxilliaire avec rangee fixee
int count (ListeRangees allrows, Rangee r, int hauteur)
{
  if (hauteur==1) return 1;
  else
    {
      Cell* pc = allrows;
      int sum = 0;
      while(pc != NULL) {
        if (sontCompatibles(r,(*pc).el))
          {
            sum = sum+ count(allrows,(*pc).el,hauteur-1);
          }
        pc = (*pc).suivant ;
      }
      return sum;
    }
}

//fonction qui compte le nombre total de murs de largeur TAILLEMAX
//et de hauteur h
int count_all (ListeRangees allrows, int hauteur)
{
  Cell* pc = allrows;
  int sum = 0;
  while(pc != NULL) {
    int res = count(allrows,(*pc).el,hauteur);
    sum = sum + res;
    pc = (*pc).suivant ;
  }
  return sum;
}
```

Annexe 2 Code à modifier, feuille à rendre avec votre copie !

NOM :
Numéro de Place :

```
//Fonction auxilliaire avec rangee fixee
int count (ListeRangees allrows, Rangee r, int hauteur, ____________________________)
{
  if (hauteur==1) return 1;
  else {
      //ici ajouter de quoi chercher si count(allrows,r,hauteur) a deja ete calcule




      //si oui :



       //si non :
      else {
        Cell* pc = allrows;
        int sum = 0;
        while(pc != NULL) {
          if (sontCompatibles(r,(*pc).el))
            {
              sum = sum+ count(allrows,(*pc).el,hauteur-1);
            }
          pc = (*pc).suivant ;
        }
        //ajouter de quoi sauvegarder sum ici (nouveau triplet+insertion)







        return sum;
      }//fin du "sinon"
} }

//fonction qui compte le nombre total de murs de largeur TAILLEMAX
//et de hauteur h
int count_all (ListeRangees allrows, int hauteur)
{
  Cell* pc = allrows; int sum = 0;
  //declaration et initialisation hashtable :



  while(pc != NULL) {
    int res = count(allrows,(*pc).el,hauteur,_________________);
    sum = sum + res;
    pc = (*pc).suivant ;
  }
  return sum;
}
```