Programmation dynamique

I. Introduction, les nombres de Catalan

Les nombres de Catalan sont définis par :

$$C_n = 1 \qquad \forall\, n \geq 1,\ C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

On peut les calculer naïvement de la façon suivante :

```
let rec catalan_naif n =
  if n = 0 then
    1
  else
    let s = ref 0 in
    for k = 0 to n - 1 do
      s := !s + (catalan_naif k * catalan_naif (n - 1 - k))
    done;
    !s
;;
```

La complexité c_n pour calculer C_n vérifie :

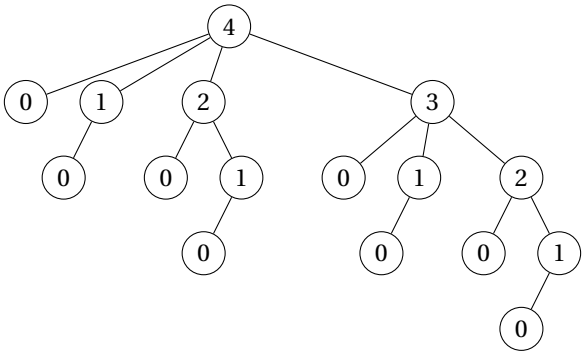
$$c_0 = 1 \qquad \forall\, n \geq 1,\ c_n = 2\sum_{k=0}^{n-1} c_k + \Theta(n)$$

Exercice 1

1. En posant $s_n = \sum_{k=0}^{n} c_k$, prouver que $s_n = 3s_{n-1} + \Theta(n)$.

2. En déduire que $s_n = \Theta\left(3^n\right)$ puis que $c_n = \mathrm{O}\left(3^n\right)$.

Le problème, c'est qu'on passe son temps à recalculer des valeurs que l'on a déjà calculées. Schématiquement, on peut symboliser les appels récursifs ainsi (ici, pour calculer C_4) :



On peut remédier à ce problème, en stockant dans un tableau les différents C_k pour k allant de 0 à n :

```
let catalan n =
  let t = Array.make (n + 1) 0 in
  t.(0) <- 1;
  for i = 1 to n do
    for j = 0 to i - 1 do
      t.(i) <- t.(i) + (t.(j) * t.(i - 1 - j))
    done
  done;
  t.(n)
;;
```

Question 2 Quelle est la complexité de cet algorithme ?

II. Principe de la programmation dynamique

On a vu précédemment la méthode « diviser pour régner » qui tire son efficacité du fait que l'on peut diviser une entrée de taille n en deux entrées **indépendantes** de taille $n/2$ et que l'on peut obtenir, pour un coût raisonnable, la réponse au problème initial à partir des réponses aux sous-problèmes.

Il n'est cependant pas toujours possible de décomposer un problème en sous-problèmes indépendants. Dans ce cas, il y a un risque de redondance dans la résolution. Dans l'exemple précédent, pour calculer C_5, on a besoin (entre autres) des valeurs de C_3 et de C_4. Or, pour calculer C_4, on a aussi besoin de la valeur de C_3. On risque donc de la calculer plusieurs fois.

Pour y remédier, on peut
- ⋆ stocker les résultats des appels récursifs pour ne jamais le recalculer ;
- ⋆ réfléchir à l'ordre de calcul de ces valeurs.

C'est ce que l'on fait avec `catalan`.

III. Exemples

1. Un peu de mathématiques

Question 3 Écrire une fonction `fibo : int -> int` qui calcule efficacement le n-ème terme de la suite de Fibonacci définie par

$$F_0 = 0, \quad F_1 = 1, \qquad \forall\, n \in \mathbf{N},\ F_{n+2} = F_{n+1} + F_n$$

Quelle est la complexité spatiale de la fonction ?

Question 4 Écrire une fonction `binom : int -> int -> int` qui calcule efficacement les coefficients binomiaux **avec des additions uniquement**.

Quelle est la complexité spatiale de la fonction ?

2. Les tours-relais

On vous donne un tableau $[\,|\,p_0\,,\,p_1\,,\,\ldots\,,\,p_{n-1}\,|\,]$ de populations de villes, et vous voulez construire des tours-relais pour le téléphone pour couvrir le maximum de population possible. Problème, vous ne pouvez pas construire deux tours dans des villes voisines.

⋆ Quelle est la population maximale que vous pouvez couvrir ?
⋆ Où construire les tours pour cela ?

Par exemple, pour les tableaux `[|10; 20|]` et `[|99; 100; 99|]`, quel est l'optimum ? Et pour

`[|14; 22; 13; 25; 30; 11; 9|]` ?

Pour une liste de population fixée, on note s_k la population maximale que l'on peut couvrir en ne considérant que les k premières villes.

Question 5 Déterminer une relation de récurrence vérifiée par les s_k.

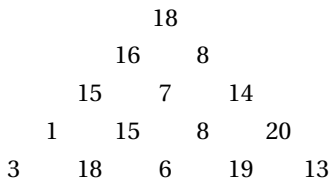
Question 6 En déduire un algorithme efficace pour calculer la population maximale pouvant être couverte, que vous traduirez en une fonction

`couverture : int array -> int`

Question 7 Comment modifier l'algorithme pour avoir la liste des villes à couvrir ?

3. Chemin dans un triangle

On se donne un triangle d'entier comme ci-dessous :



et on cherche à déterminer le chemin partant du sommet et descendant jusqu'à la base qui maximise la somme des nombres rencontrés.

Question 8 Combien y a-t-il de tels chemins ?

Un tel triangle sera représenté en OCaml par un élément `p` de type

```
int array array
```

de taille n tel que `p.(`i`)` est de taille $i+1$. Ainsi, la représentation du triangle précédent est :

```
[|
  [| 18 |];
  [| 16; 8 |];
  [| 15; 15; 14 |];
  [| 1; 15; 8; 20 |];
  [| 3; 18; 6; 19; 13 |];
|]
```

On note $o(l,c)$ le chemin de plus forte somme partant du sommet et arrivant à la c-ème valeurs de la ligne l.

Question 9 Déterminer une relation de récurrence sur les $o(l,c)$.

Question 10 En déduire une fonction `chemin_max : int array array -> int` qui calcule et renvoie la valeur maximale d'un chemin du sommet jusqu'à la base.

Question 11 Quelle est la complexité temporelle de la solution obtenue ? Est-ce bien quadratique ?

Question 12 Quelle est la complexité spatiale de la solution obtenue ? Est-ce bien linéaire ?

Question 13 Comment modifier le programme pour qu'il renvoie le chemin à suivre ?