1 Exponentiation rapide

Voici une fonction OCaml réalisant l'exponentiation rapide :

```
let rec exprap = fun x n \rightarrow
  if n = 0 then 1
  else
    let y = (exprap x (n/2)) in
    if (n/2)*2 = n then y*y
    else x*y*y
```

Notons $M(n)$ le nombre de multiplications. Si `n` $= 0$, la fonction effectue 0 multiplications. Si `n` $= 1$, on fait 2 multiplications (`x*y*y`). Sinon, posons `n` $= 2^p$ (avec $p \in \mathbb{N}^*$). Dans ce cas on fait 1 multiplications (`y*y`). On a donc :

$$M(2^p) = \begin{cases} 2 \text{ si } p = 0 \\ 1 + M(2^{p-1}) \text{ sinon} \end{cases}$$

C'est une suite arithmétique, que l'on résout : $M(2^p) = p+2$. La complexité est donc logarithmique en $n = 2^p$, c'est à dire linéaire en la taille de n.

2 Comptaire binaire : suite de INCR par la méthode du potentiel

Convention de nommages, rappelables par le colleur :

On considère une suite arbitrairement longue $o_1, ..., o_k$ d'opérations INCR commençant sur le compteur vide. On considère qu'un flip coûte exactement 1 et le reste 0. On note c_i les coûts respectifs des o_i, et on l'on veut créer des coûts amortis $\hat{c}_i$.

On note L le nombre de bits du compteur.

Faire un très joli schéma d'un compteur binaire à L bits.

<u>Remarque initiale commune aux deux méthodes :</u>

Découpons le coût d'un INCR en deux parties : chaque INCR flippe exactement un bit de 0 à 1 (*car on ignore l'overflow, qui en flippe 0*). En effet, un tel flip ne propage pas de retenue plus avant, et termine donc l'incrémentation. Ainsi, si un INCR flippe c_i bits, $c_i - 1$ flips sont de 1 à 0 et un est de 0 à 1.

<u>Méthode du potentiel :</u>

Prenons comme potentiel $\Phi_i =$ « Nombre de bits à 1 » .

Alors $\Phi_0 = 0$ car l'état initial est le compteur vide. Par définition de Φ_i, on a bien $\forall i, \Phi_i \geq 0 = \Phi_0$.

On a $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$. Mais d'après l'analyse précédente du comportement de INCR, $\Phi_i = \Phi_{i-1} - (c_i - 1) + 1$. Soit $\hat{c}_i = 2$.

On a donc prouvé par la méthode du potentiel que le coût amorti d'un INCR au suite d'une suite arbitrairement longue de INCR est constant.

<u>Méthode du comptable :</u>

Prépayons le coût d'un flip $1 \to 0$ lors du flip $0 \to 1$. Initialement, tous les bits sont à 0. Les états successifs d'un bit sont donc $0 \to 1 \to 0 \to 1 \to 0...$.

En payant 2 lors de chaque flip $0 \to 1$, on peut alors payer 0 lors du flip $1 \to 0$ qui le suit. Il s'ensuit de la remarque initiale que chaque INCR coûte 2, soit $\hat{c}_i = 2$.

On a donc prouvé par la méthode du comptable que le coût amorti d'un INCR au suite d'une suite arbitrairement longue de INCR est constant.

3 Files par tableaux circulaires

Faire un joli schéma. Il est important que sur ce schéma apparaissent `sortie` (l'indice du prochain élément défilé) et `entree` (l'indice de l'autre extrémité de la file; on pensera à préciser s'il est inclus ou exclu), ainsi que le sens de parcours.

Code : cf `https://nuage04.apps.education.fr/index.php/s/EQX2QtdN2kJTW2X` . Dossier TP/09-tabCircC/solution/ . J'y utilise un booléen pour distinguer la file vide de la file pleine; mais c'est un détail qui n'est pas attendu.

4 Retour sur trace

On représentera un ensemble par une liste. J'appelle `ens` la liste contenant l'ensemble, `target` l'entier-cible, et `subsetsum` la fonction récursive.

Le critère de rejet est que lorsque les entiers sont tous positifs, une target négative est impossible.

```
let rec subsetsum ens t =
  if t = 0 then true
  else if t < 0 then false
  else match ens with
  | x :: ens' -> subsetsum ens' (t-x) || subsetsum ens' t
  | [] -> false
```

Pour plus d'explications, cf TP SubsetSum.