Exercices : solutions de la série #9

Solution à l'exercice 0.

a. Le travail est $\Theta(n \lg n)$: chaque itération génère un travail $\Theta(n)$, puisque n invocations sont créées et que chacune font un travail $\Theta(1)$. Notons toutefois que, dans certains cas, le travail consiste simplement à constater que le pointeur est `null` et, donc, à ne rien faire. Non, ce n'est pas optimal.

b. Le travail est, encore une fois, $\Theta(n \lg n)$.

Plus précisément, on a que le nombre de processeurs actifs à l'itération i (pour $i = 1, \ldots, \lg n$) est le suivant (le nombre de processeurs inactifs double à chaque itération, avec un seul processeur inactif au début, à savoir celui pour le dernier noeud) :

$$n - 2^{i-1}$$

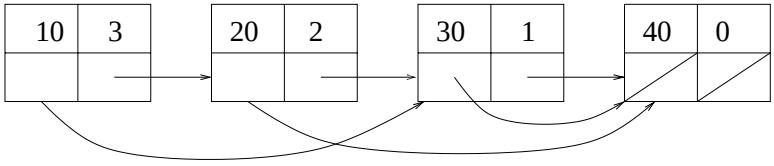
Le nombre de processeurs actifs sur l'ensemble des itérations est donc le suivant :

$$\begin{aligned}
\sum_{i=1}^{\lg n} n - 2^{i-1} &= n \lg n - \sum_{i=1}^{\lg n} 2^{i-1} \\
&= n \lg n - \frac{1}{2} \sum_{i=1}^{\lg n} 2^i \\
&= n \lg n - \frac{1}{2}[\frac{2^{\lg n+1} - 1}{2-1} - 1] \\
&= n \lg n - \frac{1}{2}[2n - 2] \\
&= n \lg n - (n-1) \\
&\in \Theta(n \lg n)
\end{aligned}$$

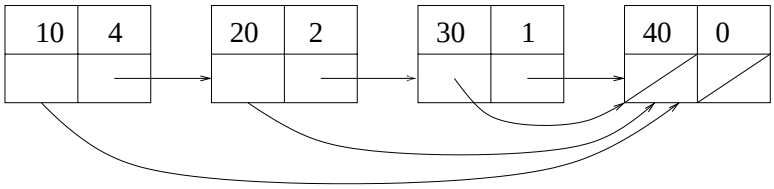
Solution à l'exercice 1.

Supposons, par exemple, que l'invocation de `sauterPointeur` pour les divers noeuds se fasse de façon entrelacée, c'est-à-dire que l'exécution d'une invocation se fasse de telle façon que certaines écritures sont effectuées avant que toutes les écritures aient été effectuées.

Plus précisément, supposons que l'invocation pour le deuxième noeud ait le temps de lire les champs du noeud suivant et de mettre à jour son champ `rang`, mais pas son champ `suivantSaut`, et ce avant que l'activation pour le premier noeud ait eu le temps de *lire* les champs de son suivant (donc de lire des champs qui ont été partiellement modifiés). L'état de la liste serait alors le suivant :



Après l'itération suivante, l'état de la liste serait alors le suivant, où on remarque que le rang associé au premier noeud n'est pas correct (il devrait être égal à 3) :



Solution à l'exercice 2.

```
cap Barriere barriere = create Barriere( n );

procedure sauterPointeur( PtNoeud pt )
{
  if (pt^.suivantSaut != null) {
    # On commence par lire les donnees.
    int rangSuiv = pt^.suivantSaut^.rang;
    PtNoeud suivSuiv = pt^.suivantSaut^.suivantSaut;

    # On attend que les lectures soient completees.
    barriere.attendre();

    # On effectue les mises a jour.
    pt^.rang += rangSuiv;
    pt^.suivantSaut = suivSuiv;
  } else {
    barriere.attendre();
  }
}
```

Solution à l'exercice 3.

```
procedure verifierSiOu( bool Xi, ref bool r )
{ if (Xi) { r = true; } }

procedure calculerOu( bool X[*], int n ) returns bool r
{
  r = false;
  co [i = 1 to n]
    verifierSiOu( X[i], r );
  oc
}
```

N'importe quel mode ferait l'affaire, puisque si plusieurs écritures sont effectuées, elles seront toutes identiques (`true`).

Le coût de l'algorithme est $\Theta(n)$, donc est optimal.