Exercices INF7440 : série #9

0. Algorithme de sauts de pointeurs et travail

a. Quel est le *travail* effectué par l'algorithme de calcul du rang des éléments d'une liste chaînée par la technique des sauts de pointeurs (algorithme 12, p. 28)? Est-ce optimal?

b. Supposons que l'on modifie l'algorithme comme suit :

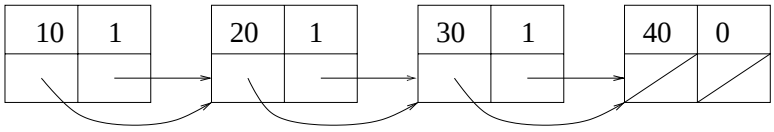
```
for [j = 1 to lg(n)] {
  co [i = 1 to n st noeuds[i]^.suivantSaut != null]
    sauterPointeur( noeuds[i] )
  oc
 }
}
```

Quel est alors le travail effectué par l'algorithme? Est-ce optimal?

1. Algorithme MPD à la PRAM et entrelacement des invocations

L'algorithme 12 (p. 28) du chapitre "Le modèle PRAM" n'est pas correct lorsqu'interprété selon la sémantique du langage MPD. En MPD, les diverses invocations créées par une instruction `co` peuvent s'exécuter dans n'importe quel ordre.

Pour la liste chaînée suivante, dont le contenu est celui immédiatement après l'initialisation, donnez un exemple d'exécution qui pourrait conduire à un résultat incorrect :



2. Saut de pointeur à la SPMD avec barrière de synchronisation

L'exercice précédent nous a montré que la version MPD de calcul des rangs présentée dans les notes de cours (algorithme 12, p. 28) n'était pas correct, et ce à cause du modèle d'exécution SPMD sous-jacent à MPD (lorsqu'une famille d'invocations est créée à partir d'une instruction `co`).

Réécrivez la procédure `sauterPointeur` en utilisant les barrières de synchronisation introduites par le module `Barriere`, et ce de façon à assurer son bon fonctionnement.

Note : Une nouvelle barrière `b` peut être créé et initialisée à l'aide d'une instruction de la forme suivante, où `n` indique le nombre de processus qui doivent se synchroniser à la barrière :

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

Soulignons qu'une telle barrière *peut être réutilisée*, c'est-à-dire qu'elle peut servir à nouveau après que tous les processus sont finalement arrivés et ont été réactivés.

Note : Dans un modèle SPMD utilisant le parallélisme de données, comme c'est le cas ici, il faut bien comprendre *que tous les processeurs sont actifs* à chacune des itérations de saut de pointeur, et ce même si leur pointeur associé est `null`.

3. Disjonction booléenne en temps constant

Concevez un algorithme CRCW permettant de calculer la disjonction (OU) de n valeurs booléennes en temps constant. L'en-tête de la procédure associée serait le suivant :

```
procedure calculerOu( bool X[*], int n ) returns bool leOu
```

Quel mode de résolution des conflits CW peut être utilisé pour cet algorithme? Quel est le coût de cet algorithme? Est-ce optimal?