IFT2015 :: hiver 2020

Cours de structures de données, Université de Montréal

JANVIER 18, 2020JANVIER 18, 2020 *by* CSUROSM

Liste chaînée: diapos et exercices

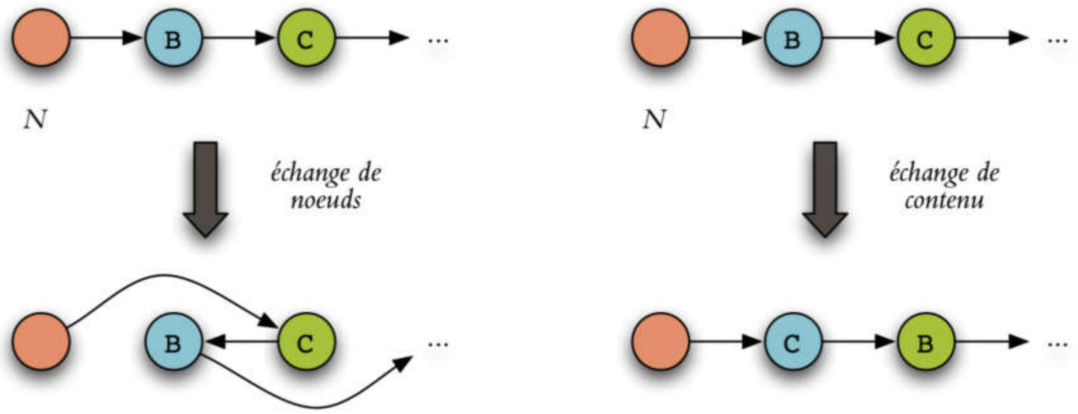
- EXERCICES, NOTES DE COURS
- LAISSER UN COMMENTAIRE

Liste chaînée: 04prez-chaine.pdf (diapos) (https://ift2015h20code.files.wordpress.com/2020/01/ift2015h20-04prez-chaine.pdf).

Exercices

Sauf si autrement spécifié, les exercices suivants représentent une liste comme un ensemble de noeuds: chaque noeud x (sauf le noeud terminal x=`null`) contient les variables x.`next` (prochain élément) et x.`data` (contenu: élément stocké).

L.1 Échange d'éléments



- Développer le code pour `exchange`(N) qui échange deux noeuds suivant N.
- Développer le code pour `exchangeData`(N) qui échange le contenu des deux noeuds suivant N.

L.2 Inversion de liste

Proposer des algorithmes pour renverser une liste chaînée

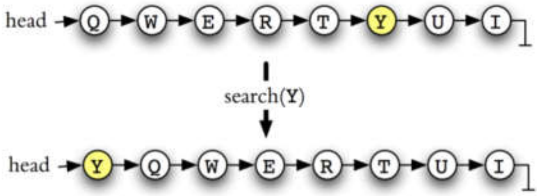
- par itération (en un seul parcours), ou
- par récursion.

L.3 Concaténation

Montrer comment faire la concaténation de deux listes circulaires.

L.4 Move to front

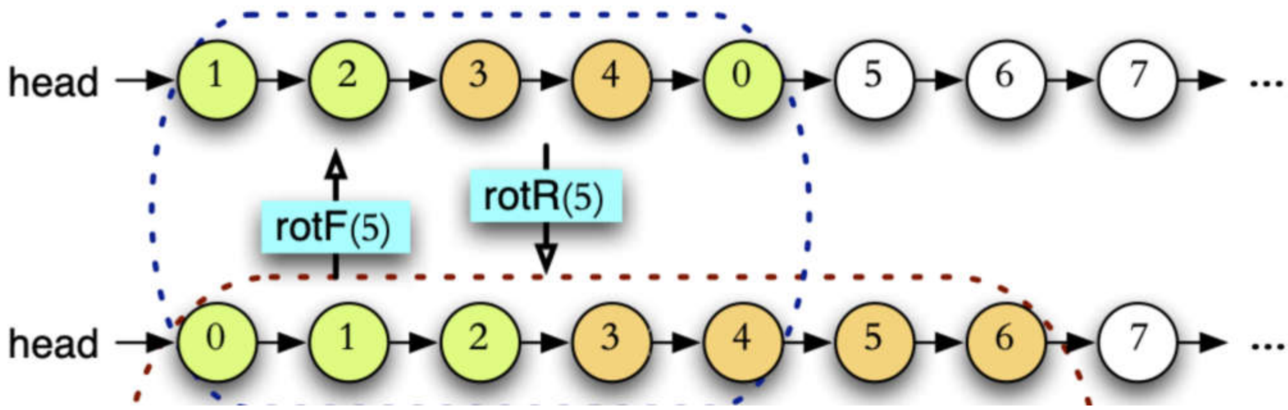
La heuristique MTF (*move-to-front*) déplace l'élément trouvé à la tête. Lors d'une recherche infructueuse, la liste ne change pas. La heuristique est utile quand on cherche des éléments avec des fréquences différentes car les éléments souvent recherchés se trouvent vers le début de la liste pendant une série d'appels.

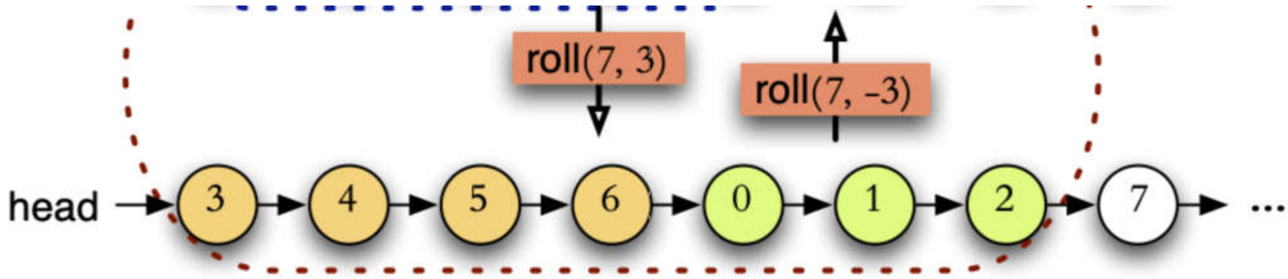


Donner une opération `search`(head, y) qui performe la recherche séquentielle selon la heuristique MTF.

L.5 Rotations

(https://elgoog.im/doabarrelroll/)On veut une structure qui supporte des rotations d'éléments sur une liste. Soit $(x_0, x_1, \ldots, x_{\ell-1})$ l'ordre des noeuds sur une liste. Les opérations suivantes changent l'ordre des noeuds au début de la liste avec $n \leq \ell$. En une **rotation avant** (rotF), on avance les noeuds $x_1, \ldots, x_{n-1}$ vers la tête et on place x_0 après x_{n-1}. En une **rotation arrière** (rotR), les noeuds $x_0, \ldots, x_{n-2}$ reculent vers la queue et x_{n-1} se place à la tête. Le **décalage circulaire** (roll) correspond à multiples rotations avant ou arrière. L'opération $\text{roll}(n, j)$ correspond à j rotations avant si $j > 0$, ou $(-j)$ rotations arrière si $j < 0$. On peut rétablir l'ordre original par l'opération inverse $\text{roll}(n, -j) = \text{roll}(n, n-j)$.





[(https://ift2015h20code.files.wordpress.com/2020/01/liste-roll.png)](https://ift2015h20code.files.wordpress.com/2020/01/liste-roll.png)

Montrer comment implanter les opérations rotF(n), rotR(n), et roll(n, j) sur une liste simplement chaînée.

Indice: Il n'est pas nécessaire de performer plusieurs rotations pour implanter roll. Identifiez plutôt les noeuds où .next doit changer. Pour une solution récursive, inclure un argument pour dénoter le début de la liste: on appellera, p.e., head $\leftarrow$ rotF(head, 10).

Liste ou tableau: comparez le code de cette implémentation avec le décalage circulaire d'un tableau (v. [exercices sur les tableaux (https://ift2015h20.code.blog/2020/01/13/tableaux-exercices/#rotation)](https://ift2015h20.code.blog/2020/01/13/tableaux-exercices/#rotation)).

L.6 Comment battre les cartes? 🏆

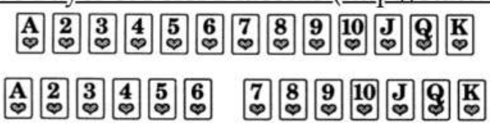
Le but de battre les cartes est d'obtenir un ordre aléatoire dans le paquet. Mathématiquement, on veut une permutation aléatoire uniformement distribuée. L'algorithme suivant construit une telle permutation.

```
RndPerm(n) // contruit permutation aléatoire π[0..n-1]
{
    initialiser π[0..n-1]
    for (i ← 0,1,...,n-1) { π[i] ← i }
    for (i ← 0,1,...,n-2)
    {
        j ← Random(i, i + 1, . . . , n - 1) // une des valeurs i..n-1 au hasard
        échanger π[i] ↔ π[j]
    }
}
```

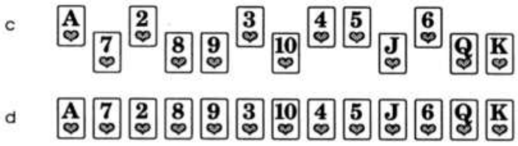
La fonction Random est un générateur de nombres pseudoaléatoires pour choisir l'indice j uniformement distribué parmi $i, i+1, \ldots, n-1$.

- Implémenter la logique de RndPerm pour calculer la permutation aléatoire d'une liste simplement chaînée. Plus précisement, donnez le pseudocode pour une opération ListPerm(x, n) qui retourne la tête d'une liste avec n noeuds (par exemple, n cartes de jeux) après une permutation aléatoire de distribution uniforme. L'argument x est la tête de la liste initiale avec n noeuds.
- Proposer un algorithme qui implante le battage par feuilleter (le «riffle shuffle», illustré ci-dessous) à l'aide de listes simplement chaînées. Analysez le temps de calcul de l'algorithme. Pour comparer à ListPerm, notez qu'il suffit de refaire le battage $c \cdot \lg n$ fois pour convergence à la distribution uniforme [[Dave Bayer et Persi Diaconis (http://statweb.stanford.edu/~cgates/PERSI/papers/bayer92.pdf)](http://statweb.stanford.edu/~cgates/PERSI/papers/bayer92.pdf), 1992].

a
b



(En particulier, sept itérations suffisent pour 52 cartes.) À l'entrée (a), on a une liste $\mathrm{x} = (x_1, x_2, \ldots, x_n)$ et un argument $k \in \{0, 1, \ldots, n\}$. L'algorithme (b) découpe la liste en $A = (x$ … $)$ et $B = (x$ … $)$ (les listes



liste en $A = (x_1, \ldots, x_k)$ et $B = (x_{k+1}, \ldots, x_n)$ (les listes peuvent être vide), (c) entrelace les deux listes au hasard, (d) et ainsi construit la fusion aléatoire des deux listes, où l'ordre original entre les éléments de la même sous-liste reste le même.