## Робота №07

*Дерева пошуку*

Зображення, що містить картинка

Автоматично згенерований опис*Бінарне дерево з повтореннями* *BinTreeM* означається рекурсивно як множина вузлів, що є порожньою (*EmptyM*) або (*NodeM v k tl tr*) складається з виділеної вузла (кореня), що містить значення *v* типу а*,* коефіцієнт його повторення - ціле значення *k>0* та два бінарних дерева з повтореннями – ліве *tl*  і праве *tr..*

Бінарне дерево з повтореннями *BinTreeM*, кожне піддерево (*NodeM v k tl tr*) якого зі значенням *v,*коефіцієнтом повторення *k>0*, лівим *tl* та правим *tr* синами має властивість: якщо вузол *u* належить *tl*, то його значення *u < v*, а якщо належить *tr*, то його значення *u > v*, називається  *бінарним деревом пошуку з повтореннями*.

Бінарне дерево пошуку з повтореннями часто використовують для ефективного представлення мультимножин зі значеннями типу a.

Означення бінарного дерева з повтореннями в Haskell:

***data*** BinTreeM a = EmptyM

| NodeM a Int (BinTreeM a) (BinTreeM a)

***deriving*** (Show, Eq)

Бінарне дерево пошуку з повтореннями, що на малюнку вище, представляє мультимножину {‘a’, ‘b’, ‘b’, ‘c’, ‘c’} і його можна записати як

*bm = NodeM ‘b’ 2 (NodeM ‘a’ 1 EmptyM EmptyM) (NodeM ‘c’ 2 EmptyM EmptyM)*

B-дерево *Btree* порядку t>=2 задовольняє властивостям

* Кожний вузол, крім корня, має n ключів (t-1) <= n <= (2\*t-1).

Внутрішній вузол (не листок) має (n+1) сина.

* Якщо дерево не порожнє, то корінь має n ключів 1 <= n <= (2\*t-1) і

(n+1) сина, якщо висота дерева більше 0.

* Кожний внутрішній вузол, що має ключі k(1), …, k(n), має (n+1) сина. Син 1 <= i <= (n+1) містить ключі з інтервалу [k(i-1), k(i)], k(0) = -∞, k(n+1) = +∞.
* Ключі в кожному вузлі впорядковані по неспаданню.
* Всі листки знаходяться на одному рівні.

Кожне B-дерево (Btree a) має три характеристики BInform: висота дерева (hB:Int), найменший (minB:a) і найбільший ключі (maxB:a), котрі є в дереві. Характеристика B-дерева використовується при перевірці коректності B-дерева.

В- дерево завжди збалансоване по висоті, тому вони дозволяють ефективно реалізувати операції пошук, додавання і вилучення елементу (ключа).

***data*** BinTreeM a = EmptyM | NodeM a (BinTreeM a) (BinTreeM a)

***deriving*** (Show, Eq)

***data*** Btree a = Node [a] [Btree a] ***deriving*** (Show, Eq)

***data*** BInform a = BInform {hB::Int, minB :: a, maxB :: a}

***deriving*** (Show, Eq)

Зауважимо, що B-дерево може мати декілька однакових ключів, задаючи мультимножину. Одна і та ж мультимножина ключів може представлятися різними B-деревами.

На Мал. 1 та Мал.2 показано два різні B-дерева (t=3 => 2 ≤ n ≤ 5), що задають одну і туж мультимножину з 17 елементів - *{A,C,D,E,E,G,H,K,L,M,M,P, R, S, U, U,W}*.

Два B-дерева, що задають одну і туж множину записів, називаються еквівалентними.

Зображення, що містить предмет, годинник, наручний годинник

Автоматично згенерований описЗображення, що містить годинник

Автоматично згенерований опис

Новий елемент (ключ) вставляється завжди в листок дерева, котрий знаходиться при проходженні дерева від кореня вниз. Але на відміну від інших видів збалансованих дерев (23-дерева, AVL-дерева, червоно-чорні дерева), котрі відновлюють баланс (перебудовуються) при «оберненому» ходу, в В-деревах ця процедура виконується в процесі пошуку необхідного листка.

* Вузол В-дерева порядку t повний, якщо він містить (2\*t-1) ключа. Такий вузол можна розділити на два мінімальні вузли, котрі мають по (t-1) ключа, виділивши один з ключів – «медіану» окремо.
* Новий елемент *завжди* вставляється в *неповний* вузол В-дерево. Тому при пошуку листка кожний повний вузол, що зустрівся *завжди* ділиться на два мінімальні і медіану.
* Оскільки вставка виконується завжди в неповний вузол, то ситуація поділу одного з його піддерев на два НЕ приводить до переповнення.

На Мал.3 показана подібна ситуація з В-деревом порядку 3. В неповний вузол з ключами – G,M,P,X потрібно вставити новий ключ Q.

* Зображення, що містить годинник, предмет

  Автоматично згенерований описВузол, корінь піддерева, в який необхідно вставити ключ, повний з ключами R,S,T,U,V.
* Перш ніж виконувати, «спуску» вниз до необхідного листка де буде вставлено новий ключ, повний вузол розділяють на два мінімальні вузли з ключами R,S і U,V та ключ-медіану T.
* Змінюють неповний вузол з ключами – G,M,P,X.
* Після цієї зміни продовжується «спуск» вниз в пошуку листка. Але вже в *неповний* вузол.

Зображення, що містить предмет, годинник

Автоматично згенерований описСхожим чином поступають і на початку виконання вставки нового ключа. Якщо вузол-корінь В-дерева повний (Мал.4), то його ділять на два неповних, збільшуючи висоту В-дерева, і лише після цього виконують вставку.

Для реалізації пошуку ключа та вставки нового ключа в В-дерево, можна попередньо реалізувати наступні допоміжні функції:

* isFull :: Ord a => Int -> Btree a -> Bool

Перевіряє – чи повний вузел?

* insertKey :: Ord a => a -> [a] -> [a]

Виконує вставку у впорядкований список нового значення. Нове значення – ЗАВЖДИ вставляється у вузол-листок

* position :: Ord a => a -> [a] -> Int

posision v xs – виконує пошук у впорядкованому списку xs номеру i першого елементу такого, що v ≤ xs!!i.

* decomposeNodeB :: Ord a => a -> [a] -> [Btree a] -> ([a],[a],[Btree a],Btree a,[Btree a])

(kl1,kl2,tl1,bt,tl2) = decomposeNodeB u kl tl

Розбиття списку ключів kl та піддерев tl внутрішнього вузла В-дерева на піддерево bt, в листок якого потрібно вставити новий ключ v, списки ключів kl1 і kl2 - зліва та справа від bt, списки підерев tl1 і tl2 – зліва та справа від bt.

* splitAtB :: Ord a => Int -> Btree a -> (Btree a, a, Btree a)

(bt1, km, bt2) = splitAtB t bt@(NodeB kl tl)

Розбиття повного вузла (NodeB kl tl) В-дерева bt порядку t на два В-дерева bt1 і bt2 та ключа-медіани km.

На малюнках Мал.5 – Мал.9 показана послідовність перебудови початкового В-дерева (Мал.5) при послідовних вставках нових елементів - B (Мал.6), Q (Мал.7), L (Мал.8) і F (Мал.9).

Зображення, що містить предмет, годинник

Автоматично згенерований опис

Зображення, що містить предмет

Автоматично згенерований опис

Зображення, що містить годинник, предмет

Автоматично згенерований опис

Зображення, що містить предмет, годинник

Автоматично згенерований опис

Зображення, що містить предмет, годинник

Автоматично згенерований опис

На основі допоміжного файлу, котрий включає визначення типів і даних для тестування, створити файл, в якому надати визначення наступних функцій.

1. Предикат *isSearch tr*, котрий перевіряє чи являється бінарне дерево з повтореннями *tr* – бінарним деревом пошуку з повтореннями .
   * isSearch (NodeM ‘t’ 0 EmptyM EmptyM) = False
   * isSearch bm == True
2. Предикат *elemSearch tr v,* котрий перевіряє чи містить бінарне дерево пошуку з повтореннями *tr* значення *v*
   * elemSearch bm ‘e’ = True
   * elemSearch bm ‘s’ = False
3. Функція *insSearch tr v,* що вставляє в бінарне дерево пошуку з повтореннями *tr* нове значення *v*.
   * insSearch EmptyM ‘t’ = NodeM ‘t’ 1 EmptyM EmptyM
   * insSearch (NodeM ‘t’ 1 EmptyM EmptyM) ‘t’ = NodeM ‘t’ 2 EmptyM EmptyM
4. Функція *delSearch tr v,* що вилучає з бінарного дерева пошуку *tr*  значення *v*. Якщо дерево не містить значення, то залишається без змін.
   * delSearch (NodeM ‘t’ 1 EmptyM EmptyM) ‘t’ = EmptyM
   * delSearch (NodeM ‘t’ 2 EmptyM EmptyM) ‘t’ = NodeM ‘t’ 1 EmptyM EmptyM
5. Функція *sortList l*, котра сортує список *l*, використовуючи бінарне дерево пошуку з повтореннями:
   * додаючи елементи списку *l*, формується бінарне дерево пошуку з повтореннями, починаючи з порожнього, (можна використати функцію *foldl*)
   * генерується впорядкований список – результат проходження бінарного дерева пошуку з повтореннями в симетричному (централізованому) порядку.
   * sortList “bca” “abc”
   * sortList “Capablanca” “Caaaabclnp”
6. *findBInform tr* – знаходить для даного *tr* типу Btree a його характеристики – дане типу BInform a.
   * findBInform (NodeB “AAFT” []) = BInform 0 ‘A’ ‘T’
   * findBInform tBt1 = BInform 2 ‘A’ ‘W’
7. *isBtree t tr* - предикат, котрий перевіряє чи являється об`єкт *tr* типа Btree B-де-ревом порядка *t.*
   * isBtree 3 (NodeB “b” [NodeB “aac” [], NodeB “bfr” []]) = False
   * isBtree 3 tBt6 = True
8. Предикат *eqBtree tr1 tr2*, котрий перевіряє чи являються два B-дерева *tr1* і *tr2* еквівалентними.
   * eqBtree 3 tBt1 tBt2 = True
   * eqBtree 3 (NodeB “gfgh” []) (NodeB “gfh” []) = False
9. Предикат *elemBtree tr v,* котрий перевіряє чи містить B-дерево *tr* ключ *v.*
   * elemBtree tBt1 ‘G’= True
   * elemBtree tBt1 ‘B’ = False
10. *insBtree t tr v* – функція вставки елемента *v* в B-дерево *tr* порядка *t.*
    * insBtree 3 tBt5 ‘B’ = tBt6
    * insBtree 3 tBt6 ‘Q’ = tBt7

*isSearch*  :: (Ord a) => BinTreeM a -> Bool

*elemSearch*:: (Ord a) => BinTreeM a -> a -> Bool

*insSearch* :: (Ord a) => BinTreeM a -> a -> BinTreeM a

*delSearch* :: (Ord a) => BinTreeM a -> a -> BinTreeM a

*sortList*  :: (Ord a) => [ a] -> [a ]

*findBInform*:: (Bounded a,Ord a) => Btree a -> BInform a

*isBtree* :: (Bounded a, Ord a) => Int -> Btree a -> Bool

*eqBtree*  :: (Bounded a, Ord a) => Btree a -> Btree a -> Bool

*elemBtree* :: (Ord a) => Btree a -> a -> Bool

*insBtree*  :: (Ord a) => Int -> Btree a -> a -> Btree a

*Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. — М.: МЦНМО, 2000. — 960 с

Зауваження:

Назва файлу Family07.hs (Family – прізвище студента). Файл включає модуль Family07 і створюється на основі файла-заготовки HWP07.hs