## Робота №3

*Дерева пошуку*

*Бінарне дерево BinTree* означається рекурсивно як множина вузлів, що є порожньою (*Empty*) або (*Node v tl tr*) складається з виділеної вузла (кореня), що містить значення *v* типу а та двох бінарних дерев – ліве *tl*  і праве *tr..*

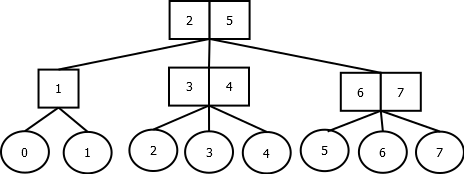
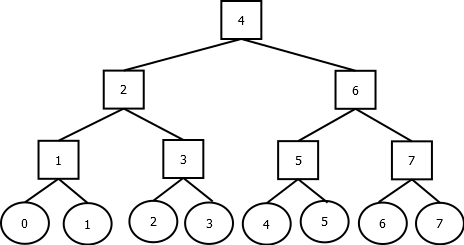
Бінарне дерево *BinTree*, кожне піддерево (*NodeM v tl tr*) якого зі значенням *v*, лівим *tl* та правим *tr* синами має властивість: якщо вузол *u* належить *tl*, то його значення *u < v*, а якщо належить *tr*, то його значення *u > v*, називається  *бінарним деревом пошуку*. Зауважимо, що бінарне дерево пошуку містить в своїх вузлах різні значення. Бінарне дерево пошуку часто використовують для ефективного представлення множин зі значеннями типу a.

Означення бінарного дерева в Haskell:

***data*** BinTree a = EmptyB

| Node a (BinTree a) (BinTree a)

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

*2-3-деревом* (збалансоване дерево) називається дерево, у якого кожний вузол, що не являється листком, має двох або трьох синів, а довжини всіх шляхів з кореня в листки однакові. Порожнє дерево і дерево з одним листком також являються 2-3-деревами. Вважається, що вся інформація - записи з ключами зберігаються в листках 2-3-дерева, а внутрішні вузли містять ключі для ефективності роботи з 2-3-деревом. Для спрощення, ми вважаємо, в подальшому, що запис складається лише з одного ключа.

*Мал.1. Два еквівалентні 2-3-дерева tr1 і tr2, що задають множину {0,1,2,3,4,5,6,7}*

Означення 2-3 дерева, у якого запис містить лише ключ:

***data*** Tree23 a = Leaf a

| Node2 (Tree23 a) a (Tree23 a)

| Node3 (Tree23 a) a (Tree23 a) a (Tree23 a)

| Empty -- порожнє 2-3-дерево!!!

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

2-3-дерево задовольняє властивостям:

* 2-3 дерево має три різні види вершин
  + *Leaf k* : листок, що містить одне значення *k*
  + *Node2 tl x tr* : 2-вузол, що містить ліве *tl* і праве *tr* піддерева та значення *x*.
    - Для всіх значень *v* з піддерева tl *v<= x.*
    - Значення *x* - мінімальне в піддереві *tr*
  + *Node3 tl x tm y tr* : 3-вузол, що містить ліве tl, середнє tm і праве tr піддерева та значення x і y.
    - Для всіх значеннь *v* з піддерева tl *v<= x.*
    - Значення *x* - мінімальне в піддереві *tm*
    - Для всіх значеннь *v* з піддерева tm *v<= y*.
    - Значення *y* - мінімальне в піддереві *tr*
* Всі листки знаходяться на одному рівні

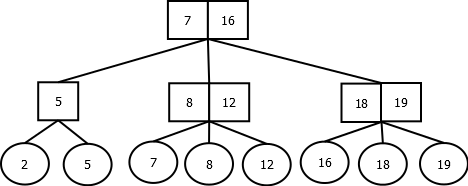
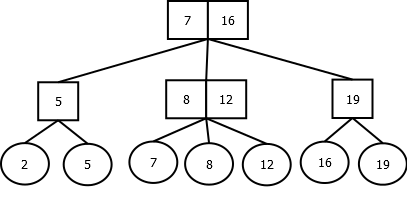
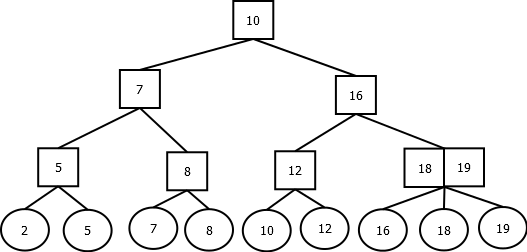
Одна і та ж множина записів може представлятися різними 2-3-деревами. На Мал. 1 показано два різні 2-3-дерева, що задають одну і туж множину натуральних чисел від 0 до 7.

Два 2-3-дерева,що задають одну і туж множину записів, називаються еквівалентними.

Зауважимо, що 2-3-дерево може мати декілька записів з однаковими ключами і вони можуть задавати мультимножини.

Щоб знайти запис з ключом v, проходять дерево від кореня до відповідного листка. Проходячи внутрішній вузол *Node2 tl x tr* або *Node3 tl x tm y tr*, порівнюють значення ключа *v* з ключами *x* і *y*, що визначити в якого сина *tl*, *tm* або *tr*, продовжити пошук. Наприклад, якщо для вузла *Node3 tl x tm y t* виконується умова *x <= v <y*, то пошук потрібно продовжити у сина *tm*.

Вставка нового запису з ключом проходить в три кроки:

* *Пошук.* Як при пошуку виконується прохід від кореня дерева до внутрішнього вузла сином якого повинен стати запис з ключом . (Такі внутрішні вузли, сини яких є листки дерева, називаються термінальними.)
* *Вставка.* Виконати вставку нового запису, при цьому якщо
  + Вузол має два листка, то створюється новий термінальний вузол з трьома листками, котрий повертається як результат.
  + Вузол має три листки, то створюються два нових термінальних вузла, котрі повертаються як результат і виконується коригування структури дерева при поверненні до кореня дерева.
* *Коригування.* Виконати коригування 2-3-дерева, повертаючись тим же шляхом до кореня:
  + Якщо в результаті вставки утворився додатковий вузол, то потрібно відновити баланс по висоті, що може привести до збільшення висоти 2-3-дерева.

*Мал.2. Вставка в 2-3-дерево нових записів 18 і 10*

Для реалізації операції вставки нового значення в можна використати функції:

* *isTerminal* :: (Ord a) => Tree23 a -> Bool

*isTerminal tr* – предикат , що перевіряє чи являється корінь 2-3-дерева термінальним вузлом.

* *insTerm* :: (Ord a) => a -> Tree23 a -> (Tree23 a, Maybe (a, Tree23 a))

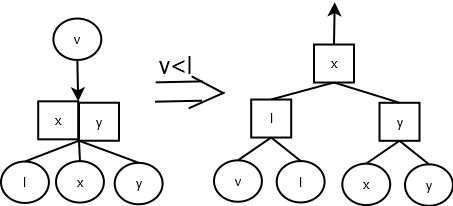
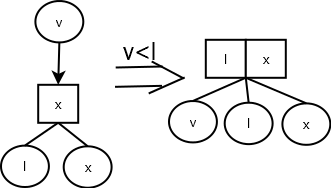
*insTerm v tr* – додає в 2-3-дерево *tr,* корінь якого термінальний вузол, значення *v*, повертаючи:

* + *(t1, Nothing) –* нове розширене дерево *t1*
  + *(t1, Just (w,t2))* – два нових дерева *t1* і *t2*, *w* - найменше значення в дереві *t2*
  + На Мал. 3 показано два варіанта (з 7) вставки в термінальний вузол
    - *insTerm v (Node2 (Leaf l) x (Leaf x)*) =

(Node3 (Leaf v) l (Leaf l) x (Leaf x), Nothing)

* + - *insTerm v (Node3 (Leaf l) x (Leaf x) y (Leaf y))* =

(Node2 (Leaf v) l (Leaf l), Just (x, Node2 (Leaf x) y (Leaf y))

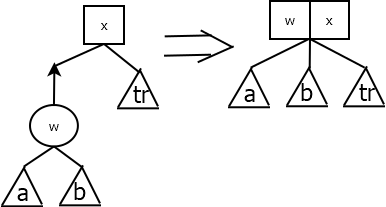
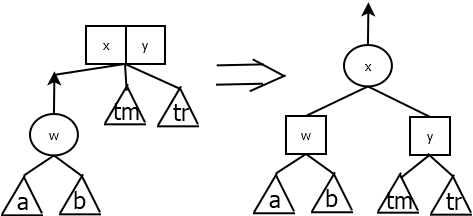


*Мал. 3. Приклади вставки значення в термінальний вузол з двома і трьома листками.*

* *insNode* :: (Ord a) => a -> Tree23 a -> (Tree23 a, Maybe (a, Tree23 a))

*insNode v tr* – додає в 2-3-дерево *tr,* корінь якого нетермінальний вузол, значення *v*, повертаючи:

* + *(t1, Nothing) –* нове розширене дерево *t1*
  + *(t1, Just (w,t2))* – два нових дерева *t1* і *t2*, *w* - найменше значення в дереві *t2*
  + На Мал. 4 показано два варіанта (з 5) коригування внутрішніх вузлів 2-3-дерева.
    - *insNode v (Node2 tl x tr) : v <x* - в результаті виконання вставки v в дерево tr повернуто (a, Just (w,b)) => ( Node3 a w b y tr, Nothing)
    - *insNode v (Node3 tl x tm y tr) : v <x* - в результаті виконання вставки v в дерево tr повернуто (a, Just (w,b)) => ( Node2 a w b, Just (x, Node2 tm y tr))



*Мал. 4. Приклади коригування 2-3-дерева коли створено два дерева при вставці.*

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

1. Предикат *isSearch tr*, котрий перевіряє чи являється бінарне дерево *tr* – бінарним деревом пошуку. Наприклад:
   * isSearch (Node 9 EmptyB (Node 9 EmptyB EmptyB) ) = False
   * isSearch bt1 = True
2. Предикат *elemSearch tr v,* котрий перевіряє чи містить бінарне дерево пошуку *tr* значення *v*. Наприклад:
   * elemSearch bt1 10 = True
   * elemSearch bt1 11 = False
3. Функція *insSearch tr v,* що вставляє в бінарне дерево пошуку *tr* нове значення *v*. Наприклад:
   * insSearch bt1 6 = bt2
4. Функція *delSearch tr v,* що вилучає з бінарного дерева пошуку *tr*  значення *v*. Якщо дерево не містить значення, то залишається без змін. Наприклад:
   * delSearch bt2 6 = bt1
5. Функція *sortList l*, котра сортує список *l*, що немає однакових елементів, використовуючи бінарне дерево пошуку:
   * додаючи елементи списку *l*, формується бінарне дерево пошуку, починаючи з порожнього, (можна використати функцію *foldl*)
   * генерується впорядкований список – результат проходження бінарного дерева пошуку в симетричному порядку.

Наприклад:

* sortList “algorithm” = “aghilmort”

1. Предикат *isTree23 tr*, котрий перевіряє чи являється об`єкт *tr* типа Tree23 2-3-де-ревом*.* Наприклад:
   * isTree23 (Node2 (Leaf 0) 1 (Leaf 2)) =False
   * isTre23 tr3 = True
2. Предикат *eqTree23 tr1 tr2*, котрий перевіряє чи являються два 2-3-дерева *tr1* і *tr2* еквівалентними. Наприклад:
   * eqTree23 tr1 tr2 = True
3. Предикат *elemTree23 tr v,* котрий перевіряє чи містить 2-3-дерево *tr* значення *v.* Наприклад:
   * elemTree23 tr3 12 = True
   * elemTree23 tr3 13 = False
4. Функція *insTree23 tr v,* що вставляє в 2-3-дерево *tr* нове значення *v*. Можна визначити допоміжні функції *isTerminal*, *insTerm* та *insNode*. Наприклад:
   * insTree23 tr4 10 = tr5

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

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

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

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

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

*isTree23* :: (Ord a) => Tree23 a -> Bool

*eqTree23*  :: (Ord a) => Tree23 a -> Tree23 a -> Bool

*elemTree23* :: (Ord a) => Tree23 a -> a -> Bool

*insTree23* :: (Ord a) => Tree23 a -> a -> Tree23 a

*isTerminal* :: (Ord a) => Tree23 a -> Bool

*insTerm* :: (Ord a) => a -> Tree23 a -> (Tree23 a, Maybe (a, Tree23 a))

*insNode* :: (Ord a) => a -> Tree23 a -> (Tree23 a, Maybe (a, Tree23 a))

*Ахо, Хопкрофт, Ульман.* Структуры данных и алгоритмы. 2000, «Вильямс». Стр.157-167. *Ахо, Хопкрофт, Ульман.* Построение и анализ вычислительных алгоритмов. 1979, «Мир». Стр.168-175

Зауваження:

Назва файлу Family03.hs (Family – прізвище студента). Файл включає модуль Family03 і trстворюється на основі файла-заготовки HWP03.hs