Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Лабораторная работа № 15

Вариант № 14

Выполнил: студент гр. 981064 Шичко П.В.

Минск 2020

Вариант № 14

Задание № 1.

Ввести 10-15 целых чисел и построить из них В-дерево порядка 5. Реализовать операции поиска, вставки и удаления элементов В-дерева.

Код программы:

class Program  
{  
 public class Node<TK, TP>  
 {  
 public List<Node<TK, TP>> Children { get; set; }  
 public List<Entry<TK, TP>> Entries { get; set; } *// - значения в листе* private int degree;  
  
 public Node(int degree)  
 {  
 this.degree = degree;  
 Children = new List<Node<TK, TP>>(degree);  
 Entries = new List<Entry<TK, TP>>(degree);  
 }  
  
 public bool IsLeaf => Children.Count == 0; *// - это узел?* public bool HasReachedMaxEntries => Entries.Count == (2 \* degree) - 1;  
  
 public bool HasReachedMinEntries => Entries.Count == degree - 1;  
 }  
  
 public class Entry<TK, TP> : IEquatable<Entry<TK, TP>> *// - узлы листа* {  
 public TK Key { get; set; } *// - ключ* public TP Pointer { get; set; } *// - указатель* public bool Equals(Entry<TK, TP> other) *// - функция эквивалентности значений ключей* {  
 return Key.Equals(other.Key) && Pointer.Equals(other.Pointer);  
 }  
 }  
  
 public class BTree<TK, TP> where TK : IComparable<TK>  
 {  
 public BTree(int degree)  
 {  
 if (degree < 2)  
 {  
 throw new ArgumentException("BTree degree must be at least 2", "degree");  
 }  
  
 Root = new Node<TK, TP>(degree);  
 Degree = degree;  
 Height = 1;  
 }  
  
 public Node<TK, TP> Root { get; private set; } *// - корень* public int Degree { get; private set; } *// - степень* public int Height { get; private set; } *// - высота* public Entry<TK, TP> Search(TK key)  
 {  
 return SearchInternal(Root, key); *// - возвращает узел (вызов приватной функции поиска)* }  
  
 public void Insert(TK newKey, TP newPointer)  
 {  
 if (!Root.HasReachedMaxEntries) *// - если есть место в корне* {  
 InsertNonFull(Root, newKey, newPointer); *//вставка* return;  
 }  
  
 *// Если родительский узел также был заполнен – то нам опять приходится разбивать.  
 // И так далее до корня (если разбивается корень – то появляется новый корень и глубина дерева увеличивается).  
 // Вставить ключ в уже заполненный лист невозможно => необходима операция разбиения узла на 2* var oldRoot = Root;  
 Root = new Node<TK, TP>(Degree); *// - создаем новый узел* Root.Children.Add(oldRoot); *// - перемещаем старый корень* SplitChild(Root, 0, oldRoot); *// - расчепляем узел на 2* InsertNonFull(Root, newKey, newPointer); *// - добалвяем ключ* Height++; *// - если разбивается корень – то появляется новый корень и глубина дерева увеличивается* }  
  
 *// Удаление по ключу (внешний)* public void Delete(TK keyToDelete)  
 {  
 DeleteInternal(Root, keyToDelete);  
  
 if (Root.Entries.Count == 0 && !Root.IsLeaf)  
 {  
 Root = Root.Children.Single();  
 Height--;  
 }  
 }  
  
 *// Метод удаления* private void DeleteInternal(Node<TK, TP> node, TK keyToDelete)  
 {  
 var i = node.Entries.TakeWhile(entry => keyToDelete.CompareTo(entry.Key) > 0)  
 .Count(); *// - ищем позицию  
  
 // Нашли ключ в узле - удаляем из него* if (i < node.Entries.Count && node.Entries[i].Key.CompareTo(keyToDelete) == 0)  
 {  
 DeleteKeyFromNode(node, keyToDelete, i);  
 return;  
 }  
  
 *// Иначе удаление из поддерева* if (!node.IsLeaf)  
 {  
 DeleteKeyFromSubtree(node, keyToDelete, i);  
 }  
 }  
  
 private void DeleteKeyFromSubtree(Node<TK, TP> parentNode, TK keyToDelete, int subtreeIndexInNode)  
 {  
 var childNode = parentNode.Children[subtreeIndexInNode];  
  
 *//Если удаление происходит из листа,  
 //то необходимо проверить, сколько ключей находится в нем.  
 //если существует соседний лист (находящийся рядом с ним и имеющий такого же родителя), который содержит больше t-1 ключа,  
 //то выберем ключ из этого соседа, который является разделителем между оставшимися ключами узла-соседа и исходного узла* if (childNode.HasReachedMinEntries)  
 {  
 var leftIndex = subtreeIndexInNode - 1;  
 var leftSibling =  
 subtreeIndexInNode > 0 ? parentNode.Children[leftIndex] : null; *// - сосед слева* var rightIndex = subtreeIndexInNode + 1;  
 var rightSibling = subtreeIndexInNode < parentNode.Children.Count - 1 *// - сосед справа* ? parentNode.Children[rightIndex]  
 : null;  
  
 if (leftSibling != null && leftSibling.Entries.Count > Degree - 1  
 ) *// - если сосед слева не пуст и содержит больше t-1 ключа* {  
 *//Перемещаем соседа в родительский узел и один узел-разделитель из родительского перемещаем в исходный узел* childNode.Entries.Insert(0,  
 parentNode.Entries[subtreeIndexInNode]); *// - перемещение разделителя* parentNode.Entries[subtreeIndexInNode] =  
 leftSibling.Entries.Last(); *// - вставка соседа в родительский* leftSibling.Entries.RemoveAt(leftSibling.Entries.Count - 1); *// - удаление из соседа* if (!leftSibling.IsLeaf)  
 {  
 childNode.Children.Insert(0, leftSibling.Children.Last());  
 leftSibling.Children.RemoveAt(leftSibling.Children.Count - 1);  
 }  
 }  
 else if (rightSibling != null && rightSibling.Entries.Count > Degree - 1  
 ) *// - если сосед справа не пуст и содержит больше t-1 ключа* {  
 *// Перемещаем соседа в родительский узел и один узел-разделитель из родительского перемещаем в исходный узел* childNode.Entries.Add(parentNode.Entries[subtreeIndexInNode]); *// - перемещение разделителя* parentNode.Entries[subtreeIndexInNode] =  
 rightSibling.Entries.First(); *// - вставка соседа в родительский* rightSibling.Entries.RemoveAt(0); *// - удаление из соседа* if (!rightSibling.IsLeaf) *// - если это не корень, мы выполняем аналогичную процедуру с ним* {  
 childNode.Children.Add(rightSibling.Children.First());  
 rightSibling.Children.RemoveAt(0);  
 }  
 }  
 else  
 {  
 *//Если же все соседи нашего узла имеют по t-1 ключу. То мы объединяем его с каким-либо соседом, удаляем нужный ключ.  
 //И тот ключ из узла-родителя, который был разделителем для  
 //этих двух «бывших» соседей, переместим в наш новообразовавшийся узел (очевидно, он будет в нем медианой).* if (leftSibling != null)  
 {  
 childNode.Entries.Insert(0, parentNode.Entries[subtreeIndexInNode]);  
 var oldEntries = childNode.Entries;  
 childNode.Entries = leftSibling.Entries;  
 childNode.Entries.AddRange(oldEntries);  
 if (!leftSibling.IsLeaf)  
 {  
 var oldChildren = childNode.Children;  
 childNode.Children = leftSibling.Children;  
 childNode.Children.AddRange(oldChildren);  
 }  
  
 parentNode.Children.RemoveAt(leftIndex);  
 parentNode.Entries.RemoveAt(subtreeIndexInNode);  
 }  
 else  
 {  
 childNode.Entries.Add(parentNode.Entries[subtreeIndexInNode]);  
 childNode.Entries.AddRange(rightSibling.Entries);  
 if (!rightSibling.IsLeaf)  
 {  
 childNode.Children.AddRange(rightSibling.Children);  
 }  
  
 parentNode.Children.RemoveAt(rightIndex);  
 parentNode.Entries.RemoveAt(subtreeIndexInNode);  
 }  
 }  
 }  
  
  
 *// Если больше t-1, то просто удаляем и больше ничего делать не нужно.* DeleteInternal(childNode, keyToDelete);  
 }  
  
 private void DeleteKeyFromNode(Node<TK, TP> node, TK keyToDelete, int keyIndexInNode)  
 {  
 *//Если корень одновременно является листом, то есть в дереве всего один узел,  
 //мы просто удаляем ключ из этого узла.* if (node.IsLeaf)  
 {  
 node.Entries.RemoveAt(keyIndexInNode);  
 return;  
 }  
  
 *// Предшествующий потомок* var predecessorChild = node.Children[keyIndexInNode];  
 if (predecessorChild.Entries.Count >= Degree  
 ) *// - если в нем количество вхождений больше или равно мин. степени* {  
 var predecessor = DeletePredecessor(predecessorChild); *// - удаляем предшественника* node.Entries[keyIndexInNode] = predecessor; *// - перемешаем его в исходный узел* }  
 else  
 {  
 *// Преемник* var successorChild = node.Children[keyIndexInNode + 1];  
 if (successorChild.Entries.Count >= Degree  
 ) *// - если в нем количество вхождений больше или равно мин. степени* {  
 var successor = DeleteSuccessor(predecessorChild); *// - удаляем преемника* node.Entries[keyIndexInNode] = successor; *// - перемешаем его в исходный узел* }  
 else  
 {  
 *// Иначе добавляем в предшественника все значение исходного узла и преемника* predecessorChild.Entries.Add(node.Entries[keyIndexInNode]);  
 predecessorChild.Entries.AddRange(successorChild.Entries);  
 predecessorChild.Children.AddRange(successorChild.Children);  
  
 node.Entries.RemoveAt(keyIndexInNode);  
 node.Children.RemoveAt(keyIndexInNode + 1);  
  
 DeleteInternal(predecessorChild, keyToDelete); *// - удаление предшественника* }  
 }  
 }

}