1. maxLeaf работи, но може да бъде написан по-добре. За начало махни maxRightLeaf, понеже не я използваш никъде. По-натам – обхождаш едното поддърво два пъти в случая, когато left != nullptr != right, понеже повтаряш едното рекурсивно извикване. Направи си там променливи, в които да запомниш двата макса и после върни по-големия (като го върнеш от променливата, а не от рек.извикване, спестяваш доооста работа). Ще е малко по-красиво също, ако пренаредиш малко случаите – в идеалния вариант са подредени по сложност и в if-овете има == вместо !=. Накрая даже може да спесиш единия if, понеже ако не си в първите 3 случая, със сигурност си в четвъртия.

T maxLeaf(Node<T> \*root) const {

if(root->left == nullptr && root->right == nullptr) {

return root->data;

}

if(root->left == nullptr) {

return maxLeaf(root->right);

}

if(root->right == nullptr) {

return maxLeaf(root->left);

}

// Now it’s sure left != nullptr != right

maxLeft = maxLeaf(root->left);

maxRight = maxLeaf(root->right);

return maxLeft > maxRight ? maxLeft : maxRight;

}

1. В getElement пиши return преди рекурсивните извиквания – освен че трябва да върнеш „стойността“, трябва и да прекъснеш изпълнението на функцията след if-а. Аз бих assert(false)-вал при некоректен път (с неправилни символи и без елемент на позицията).
2. findTrace го правихме на последното упражнение с рекурсия (path\_to\_element). Доста по-неочевидно е как ще стане със стек, затова жокер: за да намериш х, трябва да обходиш цялото дърво, стъпка по стъпка, проверявайки дали не си го открил. Нещо подобно на това също правихме скоро. :)
3. В listLeaves изтърваш root == nullptr.
4. Не е грешно да използваш моя pop, стига да си го дефинирал в кода си, което ти си направил (аз го ползвам, за да пестя време и място, както и за повече четимост – все хубави неща).