Practicum vrijdag: een BST

Collapse context

Vorige week implementeerden we onze eigen vector. Nu implementeren we onze eigen map. De map houdt keys en values bij, dus een mogelijke implementatie is via een binaire zoekboom.

Your answer passed the tests! Your score is 100.0%. [Submission #675083e705e5868b279f3b44 (**2024-12-04 16:31:35**)]

Question 1: De datastructuren

✓ Perfect

Maak een struct `User` met velden `firstname` en `lastname` en een struct `BST` met een pointer naar een User (`value`), een `key` (een integer), een `leftTree` en een `rightTree` die telkens pointers zijn naar een `BST`. Initialiseer elke waarde van de BST op een goede startwaarde.

```
1  struct User
2  {
3      string firstname;
4      string lastname;
5  };
6
7  struct BST {
8      User* value;
9      int key;
10     BST* leftTree = nullptr;
11     BST* rightTree = nullptr;
12 };
```

Submit

Question 2: createBST

✓ Perfect

Schrijf een functie `createBST` die een null pointer naar een BST teruggeeft. We steken dat in een aparte functie zodat, als we ooit beslissen om dat anders te doen (bv wel een new doen op een BST, maar met een veld om de size bij te houden en te initialiseren op 0), alle code blijft werken.

```
1  BST* createBST() {
2      return nullptr;
3  }
```

Submit

Question 3: isEmpty

✓ Perfect

Schrijf een functie `isEmpty` die een pointer naar een BST meekrijgt en een boolean teruggeeft of de BST leeg is. Een BST is leeg als de pointer de nullpt is.

```
bool isEmpty(BST* bst)
{
    return bst == nullptr;
}
```

Submit

Question 4: destroy

✓ Perfect

Schrijf een functie `destroy` die een pointer naar een BST meekijgt en al het geheugen vrijgeeft zonder memory leaks.

```
void destroy(BST* bst)
{
    if (bst == nullptr)
    {
        return;
    }
    destroy(bst->rightTree);
    destroy(bst->leftTree);
    delete bst->value;
    delete bst;
}
```

Submit

Question 5: save

✓ Perfect

Schrijf een functie `save` die een BST omzet naar een Python dictionary (en die als string teruggeeft). Bijvoorbeeld

```
{'root': 15,'left':{'root': 5},'right':{'root': 20,'right':{'root': 25}}}
```

Merk op dat je dus enkel de key telkens weergeeft. Je doet dat door een preorder traversal te doen: je bezoekt eerst de root van een deelboom, dan de linkerdeelboom en dan de rechterdeelboom.

```
string save(BST *bst)
{
    // print root.key
    // print the left child tree
    // print the right child tree
    string result = "";
    if (bst == nullptr)
    {
        return "";
    }
```

```
    result += "{\'root\':";
    result += to_string(bst->key);

    if (bst->leftTree != nullptr)
    {
        result += ",\'left\':";
        result += save(bst->leftTree);
    }

    if (bst->rightTree != nullptr)
    {
        result += ",\'right\':";
        result += save(bst->rightTree);
    }

    result += "}";

    return result;
}
```

Submit

Question 6: insert

✓ Perfect

Schrijf een functie `insert` die een pointer naar een BST krijgt, een key en een value (een pointer naar een User). De functie voegt het key, value paar toe op de juiste plaats in de boom. Dat gebeurt altijd in een blad. Als de key al bestaat, wordt de value overschreven.

```
void insert(BST*& bst, int key, User* value)
{
    if (bst == nullptr)
    {
        bst = new BST;
        bst->key = key;
        bst->value = value;
        bst->leftTree = nullptr;
        bst->rightTree = nullptr;
        return;
    }

    if (bst->key == key)
    {
        bst->value = value;
        return;
    }

    if (key > bst->key)
    {
        insert(bst->rightTree, key, value);
    }
    else if (key < bst->key)
    {
        insert(bst->leftTree, key, value);
    }
}
```

Submit

Question 7: visit

✓ Perfect

Schrijf een functie `visit` die een pointer naar een User en een key krijgt en die op het scherm afdrukt, gescheiden door spaties:

```
1 Tom Hofkens
```

```
1  void visit(User* user, int key)
2  {
3      cout << key << ' ' << user->firstname << ' ' << user->lastname << endl;
4  }
```

Submit

Question 8: find

✓ Perfect

Schrijf een functie `find` die een pointer naar een BST krijgt en een key. De functie geeft een pointer naar een User terug indien die gevonden kan worden a.h.v. de key en een nullptr als die het niet vindt.

```
1  User* find(BST* bst, int key)
2  {
3      if (bst == nullptr)
4      {
5          return nullptr;
6      }
7
8      if (bst->key == key)
9      {
10         return bst->value;
11     }
12
13     if (key > bst->key)
14     {
15         find(bst->rightTree, key);
16     }
17     else
18     {
19         find(bst->leftTree, key);
20     }
21 }
```

Submit

Question 9: preorder

✓ Perfect

Schrijf een functie `preorder` die een pointer naar een BST krijgt en de visit functie toepast op elke value in een preorder manier (dus eerst telkens de parent, dan zijn linkerkind en dan pas het rechterkind).

```
1  void preorder(BST* bst)
2  {
3      if (bst == nullptr)
4      {
5          return;
6      }
7      visit(bst->value, bst->key);
8      preorder(bst->leftTree);
9      preorder(bst->rightTree);
10 }
```

Submit

Question 10: inorder

✓ Perfect

Schrijf een functie `inorder` die een pointer naar een BST krijgt en de visit functie toepast op elke value in een inorder manier (dus eerst telkens het linkerkind, dan de parent en dan pas het rechterkind).

```
1  void inorder(BST* bst)
2  {
3      if (bst == nullptr)
4      {
5          return;
6      }
7      inorder(bst->leftTree);
8      visit(bst->value, bst->key);
9      inorder(bst->rightTree);
10 }
```

Submit

Question 11: postorder

✓ Perfect

Schrijf een functie `postorder` die een pointer naar een BST krijgt en de visit functie toepast op elke value in een postorder manier (dus eerst telkens het linkerkind, dan het rechterkind en dan pas de parent).

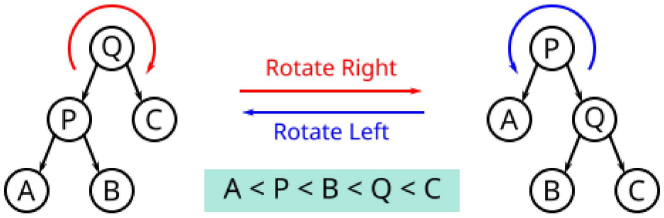
```
1  void postorder(BST* bst)
2  {
3      if (bst == nullptr)
4      {
5          return;
6      }
7      postorder(bst->leftTree);
8      postorder(bst->rightTree);
9      visit(bst->value, bst->key);
10 }
```

Submit

Question 12: rotateLeft

✓ Perfect

Om BST's te balanceren kan je rotaties gebruiken. Rotaties gebeuren als volgt.



Schrijf een functie `rotateLeft` die een pointer naar een BST meekrijgt en die naar links roteert indien mogelijk. Als het gelukt is, wordt de pointer naar de nieuwe root teruggegeven. Als het niet kan, dan geef je de oorspronkelijke BST terug.

```
1  BST* rotateLeft(BST* bst)
2  {
3      if (bst->rightTree == nullptr || bst == nullptr)
4      {
5          return bst;
6      }
7
8      BST* newRoot = bst->rightTree;
9
10     bst->rightTree = newRoot->leftTree;
11     newRoot->leftTree = bst;
12
13     return newRoot;
14 }
```

Submit

Question 13: rotateRight

✓ Perfect

Schrijf een functie `rotateRight` die een pointer naar een BST meekrijgt en die naar rechts roteert indien mogelijk. Als het gelukt is, wordt de pointer naar de nieuwe root teruggegeven. Als het niet kan, dan geef je de oorspronkelijke BST terug.

```
1  BST* rotateRight(BST* bst)
2  {
3      if (bst->leftTree == nullptr || bst == nullptr)
4      {
5          return bst;
6      }
7
8      BST* newRoot = bst->leftTree;
9
10     bst->leftTree = newRoot->rightTree;
11     newRoot->rightTree = bst;
12
13     return newRoot;
```

```
14 }
```

Submit

Question 14: balance

✓ Perfect

Schrijf een functie `balance` om een boom (pointer naar BST) die enkel bestaat uit rechterkinderen of enkel uit linkerkinderen te balanceren. Voor een boom met n knopen, doe je dat door n/2 keer naar links te roteren als de initiële boom enkel bestaat uit rechterkinderen. Analoog voor de andere soort. Je herhaalt dat algoritme vervolgens op de linkerdeelboom en op de rechterdeelboom.

```
1  BST *balance(BST *&bst)
2  {
3      int height = 1;
4      BST *tree = bst;
5
6      if (bst == nullptr)
7      {
8          return bst;
9      }
10
11     if (bst->rightTree == nullptr)
12     {
13         while (tree->leftTree != nullptr)
14         {
15             height++;
16             tree = tree->leftTree;
17         }
18
19         for (int i = 0; i < height / 2; i++)
20         {
21             bst = rotateRight(bst);
22         }
23     }
24     else if (bst->leftTree == nullptr)
25     {
26         while (tree->rightTree != nullptr)
27         {
28             height++;
29             tree = tree->rightTree;
30         }
31
32         for (int i = 0; i < height / 2; i++)
33         {
34             bst = rotateLeft(bst);
35         }
36     }
37
38     bst->leftTree = balance(bst->leftTree);
39     bst->rightTree = balance(bst->rightTree);
40
41     return bst;
42 }
```

Submit