```
 1
 2  // COS30008, Problem Set 4, Problem 1, 2022
 3
 4  #pragma once
 5
 6  #include <stdexcept>
 7  #include <algorithm>
 8
 9  template<typename T>
10  struct BinaryTreeNode
11  {
12      using BNode = BinaryTreeNode<T>; //BNode represents the btree
13      using BTreeNode = BNode*; //BtreeNode represents the pointer
14
15      T key; //data
16      BTreeNode left;
17      BTreeNode right;
18
19      static BNode NIL;
20
21      const T& findMax() const
22      {
23          if ( empty() )
24          {
25              throw std::domain_error( "Empty tree encountered." );
26          }
27
28          return right->empty() ? key : right->findMax(); // if right is empty
              then return the node, otherwise keep going
29      }
30
31      const T& findMin() const
32      {
33          if ( empty() )
34          {
35              throw std::domain_error( "Empty tree encountered." );
36          }
37
38          return left->empty() ? key : left->findMin(); // if left is empty then
              return the node, otherwise keep going
39      }
40
41      bool remove( const T& aKey, BTreeNode aParent )
42      {
43          BTreeNode x = this; // BNode<S>* x
44          BTreeNode y = aParent;
45
46          while ( !x->empty() ) //while current node is not empty
47          {
```

```
48              if ( aKey == x->key ) //if the given key matches the node we break
49              {
50                  break;
51              }
52
53              y = x;                                          // new parent (current node becomes the parent)
54
55              x = aKey < x->key ? x->left : x->right;         //ternary for if the given key is less than current then its set to left, otherwise set to right
56          }
57
58          if ( x->empty() )
59          {
60              return false;                                   // delete failed
61          }
62
63          if ( !x->left->empty() )
64          {
65              const T& lKey = x->left->findMax();         // find max to left
66              x->key = lKey;
67              x->left->remove( lKey, x );
68          }
69          else
70          {
71              if ( !x->right->empty() )
72              {
73                  const T& lKey = x->right->findMin();    // find min to right
74                  x->key = lKey;
75                  x->right->remove( lKey, x );
76              }
77              else
78              {
79                  if ( y != &NIL )                        // y can be NIL
80                  {
81                      if ( y->left == x )
82                      {
83                          y->left = &NIL;
84                      }
85                      else
86                      {
87                          y->right = &NIL;
88                      }
89                  }
90
91                  delete x;                               // free deleted node
92              }
93          }
```

```
94
95          return true;
96      }
97
98      // PS4 starts here
99
100
101     ~BinaryTreeNode()
102     {
103         if (!empty())
104         {
105             if (left != &NIL)
106             {
107                 delete left;
108             }
109             if (right != &NIL)
110             {
111                 delete right;
112             }
113         }
114     }
115
116     const bool empty()
117     {
118         return this == &NIL;
119     }
120
121     const bool leaf()
122      {
123          return left == &NIL && right == &NIL;
124      }
125
126
127     const size_t height()
128     {
129         if (empty())
130         {
131             throw std::domain_error("Empty tree encountered.");
132         }
133
134         if (leaf())
135         {
136             return 0;
137         }
138
139         size_t left_side = 0;
140         try {
141             left_side = this->left->height(); //get the max height of the left
                    subtree recursively(could also do it iteratively)
```

```
142         }
143         catch(domain_error e) {
144             left_side = 0;
145         }
146 
147         size_t right_side = 0;
148         try {
149              right_side = this->right->height(); //get the max height of the
                  right subtree
150         }
151         catch (domain_error e) {
152             right_side = 0;
153         }
154 
155         if (left_side > right_side) //compare which side is higher + 1  and
              return max height
156         {
157             return left_side + 1;
158         }
159         else
160         {
161             return right_side + 1;
162         }
163     }
164 
165     const size_t max(size_t left, size_t right)
166     {
167         if (left > right) {
168             return left;
169         }
170         return right;
171     }
172 
173 
174     bool insert(const T& aKey)
175     {
176         BTreeNode x = this;  //  BTreeNode x == BNode<S>* x
177         BTreeNode y = &NIL;
178 
179         while (!x->empty()) //if root node is NOT empty
180         {
181             y = x;  //copy it into y
182 
183             if (aKey == x->key) // if there is a duplicate key then theres an
                  error
184             {
185                 return false;
186             }
187 
```

```
188             x = aKey < x->key ? x->left : x->right; //otherwise if the key is less than current insert to left otherwise to the right
189         }
190
191         BTreeNode z = new BinaryTreeNode(aKey);
192
193         if (y->empty())
194         {
195             return false;   // if y is empty x was empty, insertion will fail (NIL)
196         }
197         else
198         {
199             if (aKey < y->key)  //if given key is less than current key, insert left
200             {
201                 y->left = z;
202             }
203             else
204             {
205                 y->right = z; //otherwise insert right
206             }
207         }
208         return true;
209     }
210
211     BinaryTreeNode() :
212         key(T()),
213         left(&NIL),
214         right(&NIL)
215     {}
216
217     BinaryTreeNode(const T& aKey):
218         key(aKey),
219         left(&NIL),
220         right(&NIL)
221     {}
222
223     BinaryTreeNode(T&& aKey)
224         :
225         key(std::move(aKey)),
226         left(&NIL),
227         right(&NIL)
228     {}
229 };
230
231 template<typename T>
232 BinaryTreeNode<T> BinaryTreeNode<T>::NIL;
233
```

234
235