

```

1  #include <iostream>
2  using namespace std;
3
4  class Node {
5      int *keys;
6      int t;
7      Node **C;
8      int n;
9      bool leaf;
10
11  public:
12      Node(bool _leaf);
13      void traverse();
14      Node *search(int k);
15      int findKey(int k);
16      void insertNonFull(int k);
17      void splitChild(int i, Node *y);
18      void remove(int k);
19      void removeFromLeaf(int idx);
20      void removeFromNonLeaf(int idx);
21      int getPred(int idx);
22      int getSucc(int idx);
23      void fill(int idx);
24      void borrowFromPrev(int idx);
25      void borrowFromNext(int idx);
26      void merge(int idx);
27      friend class Tree;
28 };
29
30 class Tree {
31     Node *root;
32     int t;
33
34  public:
35      Tree() {
36          root = NULL;
37          t = 2;
38      }
39
40      void traverse() {
41          if (root != NULL)
42              root->traverse();
43      }
44
45      Node *search(int k) {
46          return (root == NULL) ? NULL : root->search(k);
47      }
48
49      void insert(int k);
50      void remove(int k);
51 };
52
53 Node::Node(bool leaf1) {
54     t = 2;
55     leaf = leaf1;
56     keys = new int[2 * t - 1];
57     C = new Node *[2 * t];
58     n = 0;
59 }
60
61 int Node::findKey(int k) {
62     int idx = 0;
63     while (idx < n && keys[idx] < k)
64         ++idx;
65     return idx;
66 }
67
68 void Node::remove(int k) {
69     int idx = findKey(k);
70     if (idx < n && keys[idx] == k) {
71         if (leaf)
72             removeFromLeaf(idx);
73         else
74             removeFromNonLeaf(idx);
75     } else {
76         if (leaf) {
77             cout << "The key " << k << " is does not exist in the tree\n";
78             return;
79         }
80         bool flag = ((idx == n) ? true : false);
81         if (C[idx]->n < t)
82             fill(idx);
83         if (flag && idx > n)
84             C[idx - 1]->remove(k);
85         else
86             C[idx]->remove(k);
87     }
88     return;
89 }
90
91 void Node::removeFromLeaf(int idx) {
92     for (int i = idx + 1; i < n; ++i)
93         keys[i - 1] = keys[i];
94     n--;
95     return;
96 }
97
98 void Node::removeFromNonLeaf(int idx) {
99     int k = keys[idx];
100    if (C[idx]->n >= t) {
101        int pred = getPred(idx);
102        keys[idx] = pred;
103        C[idx]->remove(pred);
104    } else if (C[idx + 1]->n >= t) {
105        int succ = getSucc(idx);
106        keys[idx] = succ;
107        C[idx + 1]->remove(succ);
108    } else {
109        merge(idx);
110        C[idx]->remove(k);
111    }
112    return;
113 }
114
115 int Node::getPred(int idx) {
116     Node *cur = C[idx];
117     while (!cur->leaf)
118         cur = cur->C[cur->n];
119     return cur->keys[cur->n - 1];
120 }
121
122 int Node::getSucc(int idx) {
123     Node *cur = C[idx + 1];
124     while (!cur->leaf)
125         cur = cur->C[0];
126     return cur->keys[0];
127 }

```

```

127 }
128
129 void Node::fill(int idx) {
130     if (idx != 0 && C[idx - 1] ->n >= t)
131         borrowFromPrev(idx);
132     else if (idx != n && C[idx + 1] ->n >= t)
133         borrowFromNext(idx);
134     else {
135         if (idx != n)
136             merge(idx);
137         else
138             merge(idx - 1);
139     }
140     return;
141 }
142
143 void Node::borrowFromPrev(int idx) {
144     Node *child = C[idx];
145     Node *sibling = C[idx - 1];
146     for (int i = child->n - 1; i >= 0; --i)
147         child->keys[i + 1] = child->keys[i];
148     if (!child->leaf) {
149         for (int i = child->n; i >= 0; --i)
150             child->C[i + 1] = child->C[i];
151     }
152     child->keys[0] = keys[idx - 1];
153     if (!child->leaf)
154         child->C[0] = sibling->C[sibling->n];
155     keys[idx - 1] = sibling->keys[sibling->n - 1];
156     child->n += 1;
157     sibling->n -= 1;
158     return;
159 }
160
161 void Node::borrowFromNext(int idx) {
162     Node *child = C[idx];
163     Node *sibling = C[idx + 1];
164     child->keys[(child->n)] = keys[idx];
165     if (!(child->leaf))
166         child->C[(child->n) + 1] = sibling->C[0];
167     keys[idx] = sibling->keys[0];
168
169     for (int i = 1; i < sibling->n; ++i)
170         sibling->keys[i - 1] = sibling->keys[i];
171     if (!sibling->leaf) {
172         for (int i = 1; i <= sibling->n; ++i)
173             sibling->C[i - 1] = sibling->C[i];
174     }
175     child->n += 1;
176     sibling->n -= 1;
177     return;
178 }
179
180 void Node::merge(int idx) {
181     Node *child = C[idx];
182     Node *sibling = C[idx + 1];
183     child->keys[t - 1] = keys[idx];
184     for (int i = 0; i < sibling->n; ++i)
185         child->keys[i + t] = sibling->keys[i];
186     if (!child->leaf) {
187         for (int i = 0; i <= sibling->n; ++i)
188             child->C[i + t] = sibling->C[i];
189     }
190     for (int i = idx + 1; i < n; ++i)
191         keys[i - 1] = keys[i];
192     for (int i = idx + 2; i <= n; ++i)
193         C[i - 1] = C[i];
194     child->n += sibling->n + 1;
195     n--;
196     delete (sibling);
197     return;
198 }
199
200 void Tree::insert(int k) {
201     if (root == NULL) {
202         root = new Node(true);
203         root->keys[0] = k;
204         root->n = 1;
205     } else {
206         if (root->n == 2 * t - 1) {
207             Node *s = new Node(false);
208             s->C[0] = root;
209             s->splitChild(0, root);
210             int i = 0;
211             if (s->keys[0] < k)
212                 i++;
213             s->C[i]->insertNonFull(k);
214             root = s;
215         } else
216             root->insertNonFull(k);
217     }
218 }
219
220 void Node::insertNonFull(int k) {
221     int i = n - 1;
222     if (leaf == true) {
223         while (i >= 0 && keys[i] > k) {
224             keys[i + 1] = keys[i];
225             i--;
226         }
227         keys[i + 1] = k;
228         n = n + 1;
229     } else {
230         while (i >= 0 && keys[i] > k)
231             i--;
232         if (C[i + 1]->n == 2 * t - 1) {
233             splitChild(i + 1, C[i + 1]);
234             if (keys[i + 1] < k)
235                 i++;
236         }
237         C[i + 1]->insertNonFull(k);
238     }
239 }

```

```

240
241 void Node::splitChild(int i, Node *y) {
242     Node *z = new Node(y->leaf);
243     z->n = t - 1;
244     for (int j = 0; j < t - 1; j++)
245         z->keys[j] = y->keys[j + t];
246     if (y->leaf == false) {
247         for (int j = 0; j < t; j++)
248             z->C[j] = y->C[j + t];
249     }
250     y->n = t - 1;
251     for (int j = n; j >= i + 1; j--)
252         C[j + 1] = C[j];
253
254     C[i + 1] = z;
255     for (int j = n - 1; j >= i; j--)
256         keys[j + 1] = keys[j];
257
258     keys[i] = y->keys[t - 1];
259     n = n + 1;
260 }
261
262 void Node::traverse() {
263     int i;
264     for (i = 0; i < n; i++) {
265         if (leaf == false)
266             C[i]->traverse();
267         cout << " " << keys[i];
268     }
269     if (leaf == false)
270         C[i]->traverse();
271 }
272
273 Node *Node::search(int k) {
274     int i = 0;
275     while (i < n && k > keys[i])
276         i++;
277     if (keys[i] == k)
278         return this;
279
280     if (leaf == true)
281         return NULL;
282     return C[i]->search(k);
283 }
284
285 void Tree::remove(int k) {
286     if (!root) {
287         cout << "The tree is empty\n";
288         return;
289     }
290
291     root->remove(k);
292     if (root->n == 0) {
293         Node *tmp = root;
294         if (root->leaf)
295             root = NULL;
296         else
297             root = root->C[0];
298
299         delete tmp;
300     }
301     return;
302 }
303
304 int main() {
305     Tree t;
306     cout << "1. Insert\n2. Delete\n3. Display Tree\n4. Exit" << endl;
307     int choice, node;
308     do {
309         cout << "Enter choice: ";
310         cin >> choice;
311         switch (choice) {
312             case 1:
313                 cout << "Enter node: ";
314                 cin >> node;
315                 t.insert(node);
316                 break;
317             case 2:
318                 cout << "Enter node to remove: ";
319                 cin >> node;
320                 t.remove(node);
321                 break;
322             case 3:
323                 cout << "Tree is \n";
324                 t.traverse();
325                 cout << endl;
326                 break;
327             case 4:
328                 return 0;
329             default:
330                 cout << "Enter valid choice!" << endl;
331         }
332     } while (choice != 4);
333     return 0;
334 }

```



```
1. Insert
2. Delete
3. Display Tree
4. Exit
Enter choice: 1
Enter node: 1
Enter choice: 1
Enter node: 5
Enter choice: 1
Enter node: 3
Enter choice: 3
Tree is
 1 3 5
Enter choice: 1
Enter node: 6
Enter choice: 1
Enter node: 4
Enter choice: 1
Enter node: 3
Enter choice: 3
Tree is
 1 3 3 4 5 6
Enter choice: 2
Enter node to remove: 4
Enter choice: 3
Tree is
 1 3 3 5 6
Enter choice: 2
Enter node to remove: 4
The key 4 is does not exist in the tree
Enter choice: 2
Enter node to remove: 3
Enter choice: 3
Tree is
 1 3 5 6
Enter choice: 4
```