Binary Search Tree

Outline

- Binary Search Tree
- Implementation
- Operations
- Example

Binary Search Tree

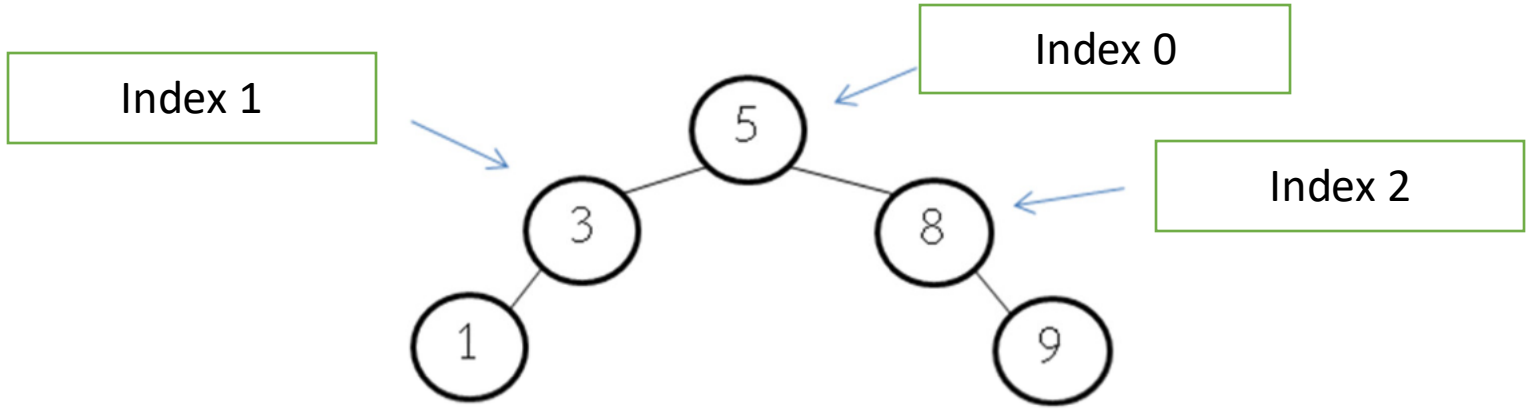
The placement of each element in the binary tree must satisfy the binary search property:

- The value of the key of an element is greater than the value of the key of any element in its left subtree,
- And less than the value of the key of any element in its right subtree.

Implementation

- Array Implementation
- Linked List Implementation

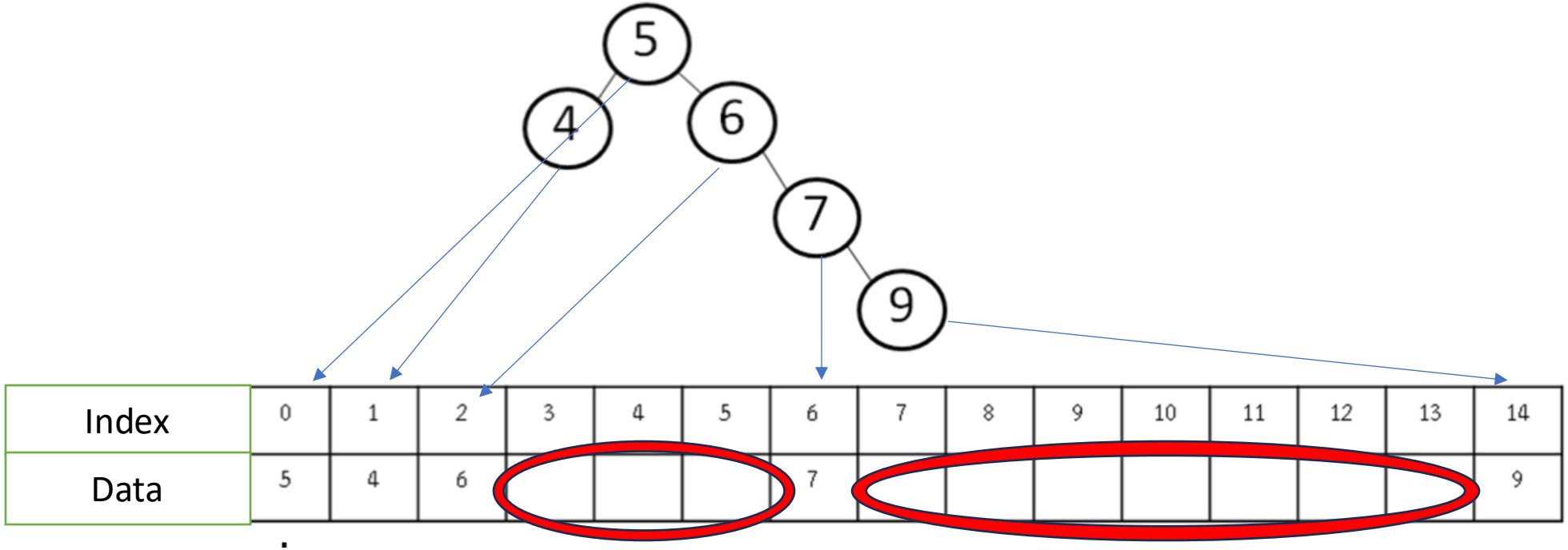
Array Implementation



Index	0	1	2	3	4	5	6
Data	5	3	8	1			9

Example

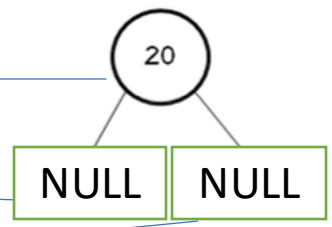
Array Implementation

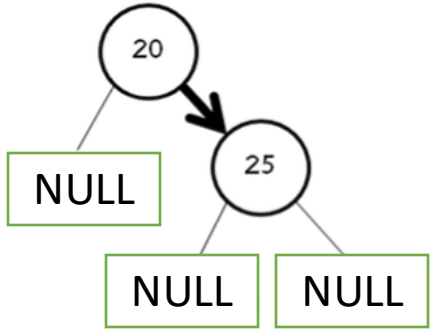


Example

Linked list Implementation

```
Class Node{
        int value;
        Node* left;
        Node* right;
};
```





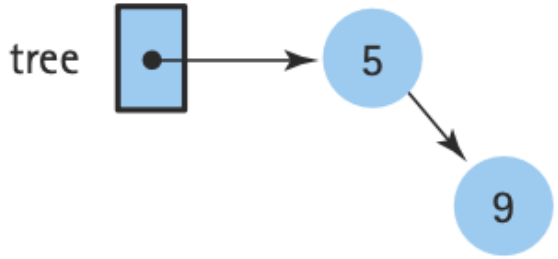
Example

Binary Search Tree - Insert

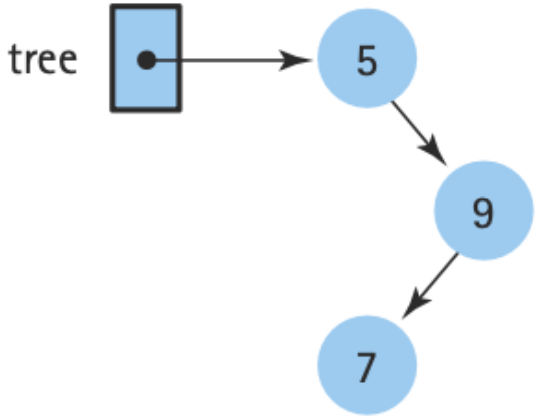
(a) tree

(b) Insert 5

(c) Insert 9

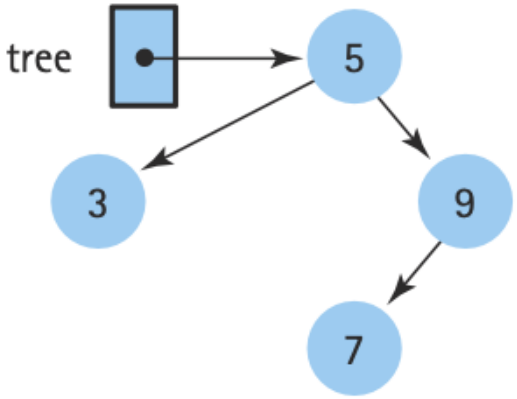


(d) Insert 7

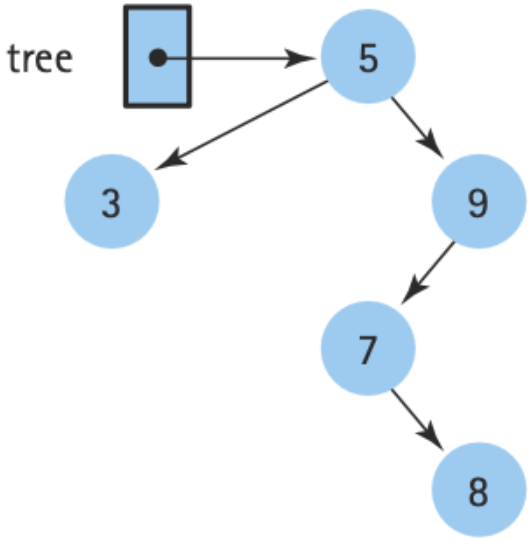


Binary Search Tree - Insert

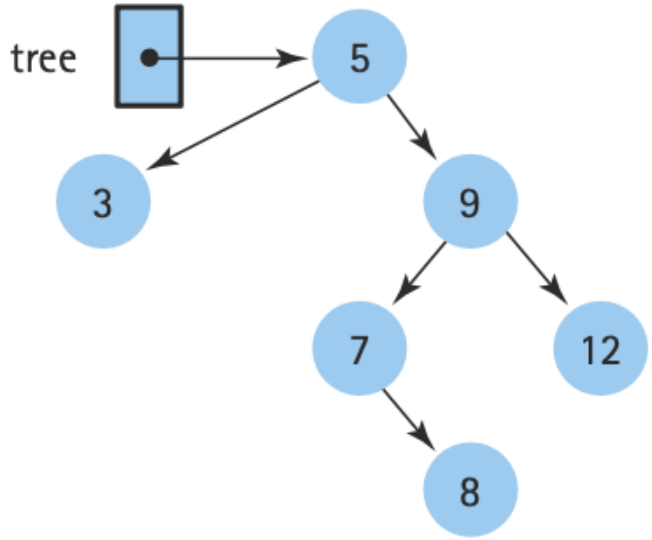
(e) Insert 3



(f) Insert 8

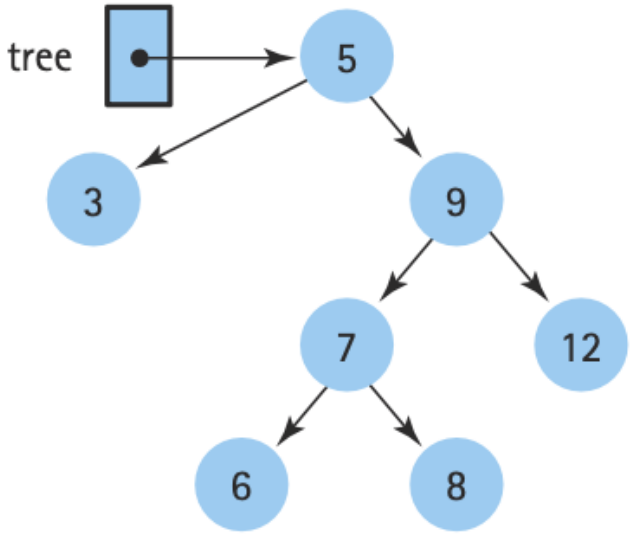


(g) Insert 12

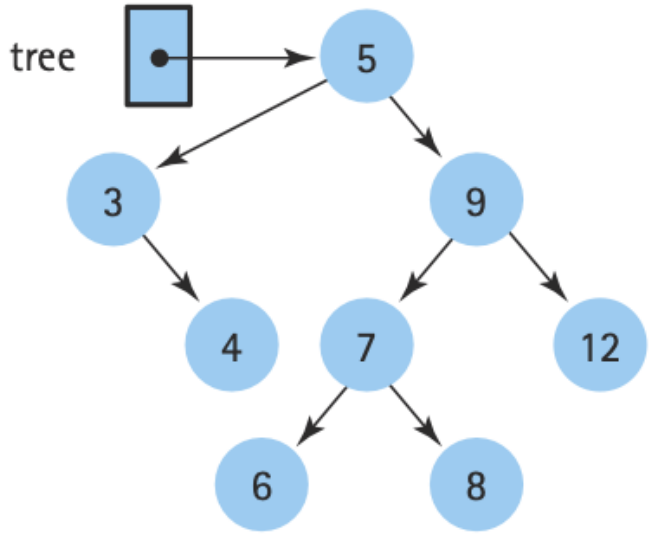


Binary Search Tree - Insert

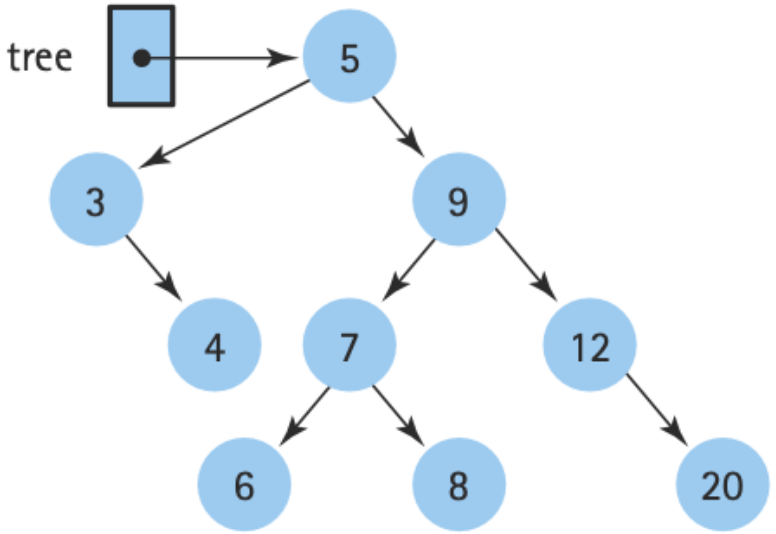
(h) Insert 6



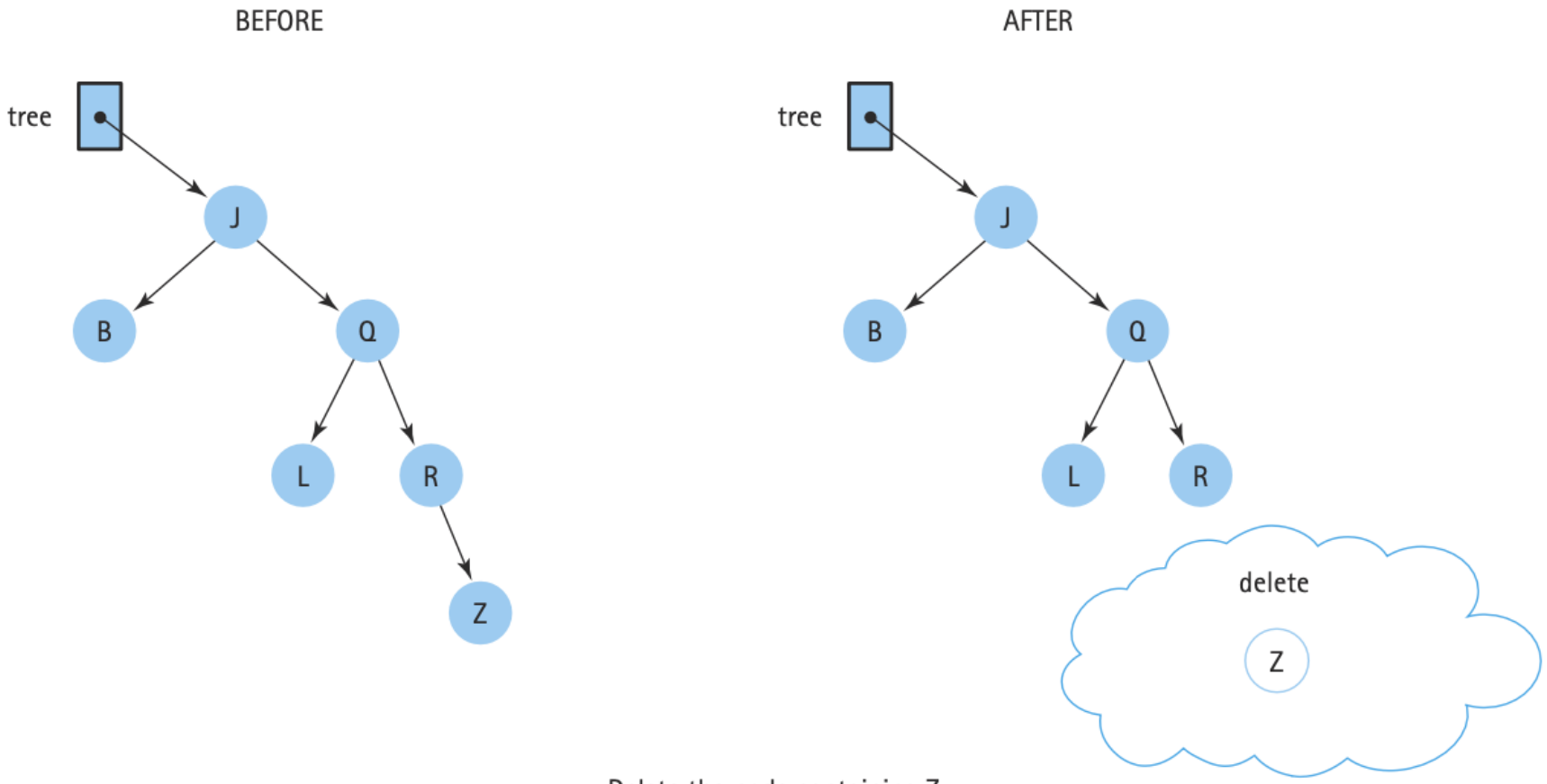
(i) Insert 4



(j) Insert 20



Binary Search Tree – Delete a leaf node



Delete the node containing Z

Figure 8.11 *Deleting a leaf node*

Binary Search Tree – Delete a node with one child

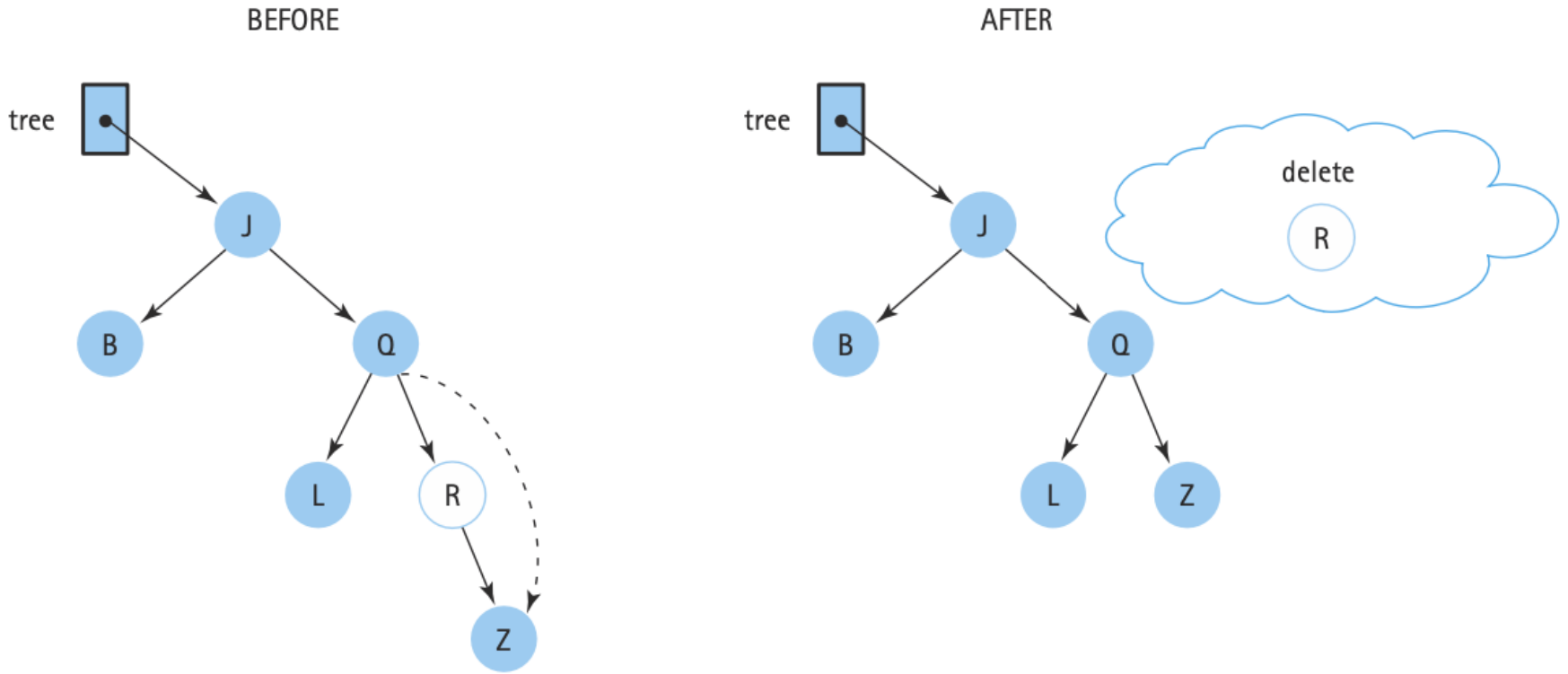


Figure 8.12 *Deleting a node with one child*

Binary Search Tree – Delete a node with 2 children

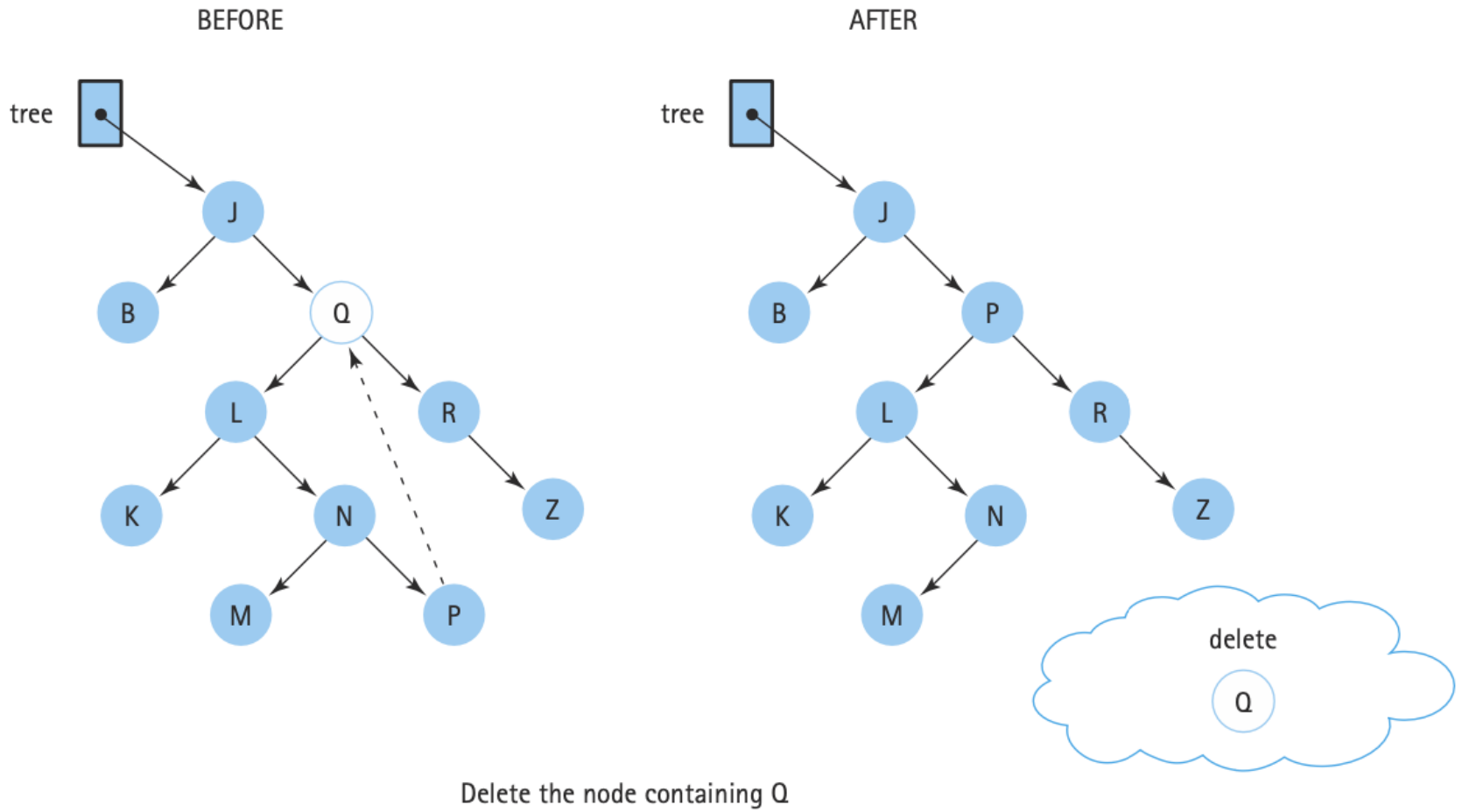


Figure 8.13 *Deleting a node with two children*

Binary Search Tree – Delete

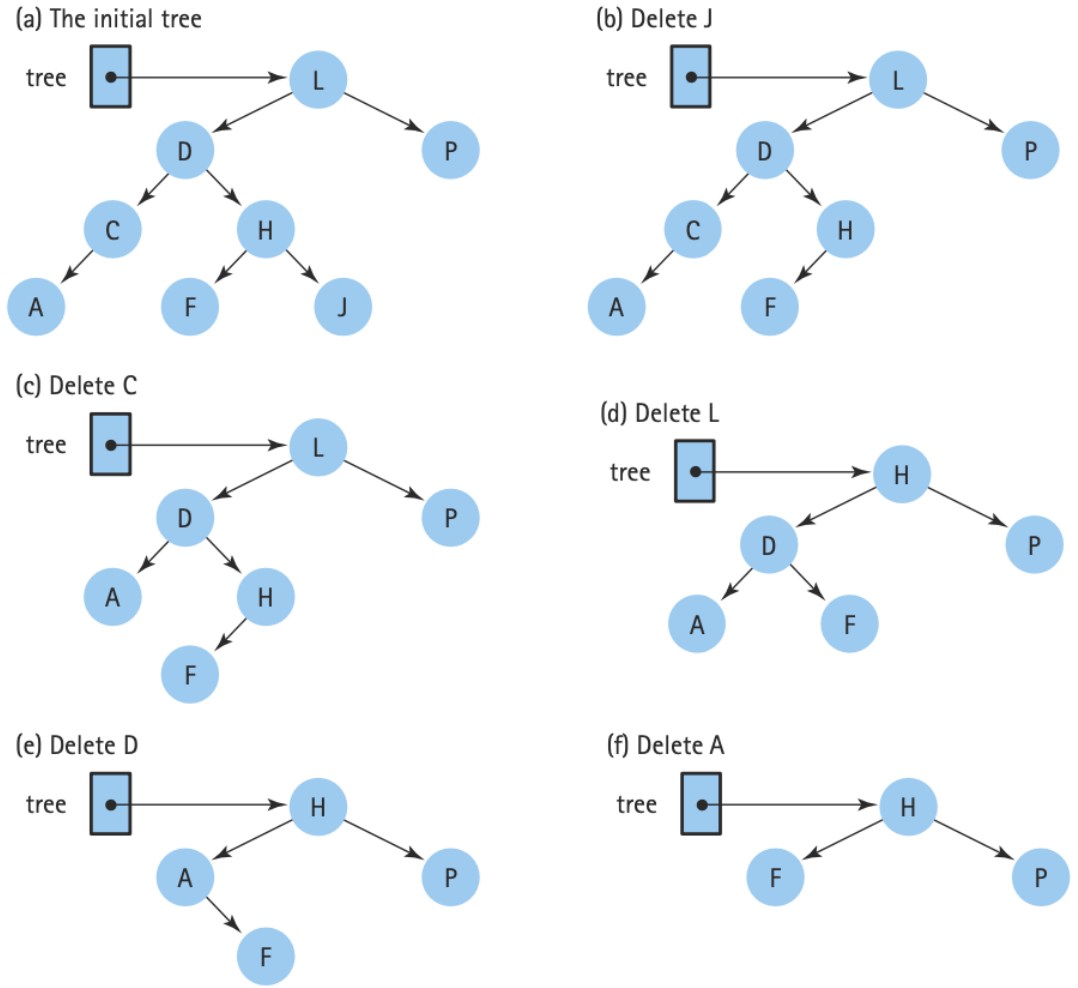


Figure 8.14 *Deletions from a binary search tree*

Example

```
1   #include<bits/stdc++.h>
2   using namespace std;
3   class node
4   {
5       public:
6           node *right;
7           node *left;
8           int value = 0;
9           node(int v)
10          {
11              value = v;
12              left = NULL;
13              right = NULL;
14          }
15  };
16  class tree
17  {
18      public:
19      node *r;
20          tree (int value)
21          {
22              r = new node(value);
23          }
24          bool search(int value)
25          {
26              node *n = r;
27              while(true)
28              {
29              if( n->value > value && n->left != NULL )
30              {
31                  n = n->left;
32              }
33              else if( n->value < value && n->right != NULL )
34              {
35                  n = n->right;
36              }
37              else if( n->value == value )
38              {
39                  return true;
40              }
41              else
42              {
43                  return false;
44              }
45          }
46      }
```

Example

```
47  void add_node(int value)
48  {
49      node *n = r;
50      while(true)
51      {
52          if( n->value > value && n->left != NULL)
53          {
54              n = n->left;
55          }
56          else if( n->value <= value && n->right != NULL)
57          {
58              n = n->right;
59          }
60          else if( n->value > value && n->left == NULL)
61          {
62              n->left  = new node(value);
63              break;
64          }
65          else if( n->value <= value && n->right == NULL)
66          {
67              n->right = new node(value);
68              break;
69          }
70          else
71          {
72              break;
73          }
74      }
75  }
```

```
76  void preorder_traversal (node *n)
77  {
78      if( n != NULL )
79      {
80          cout<<n->value<<",";
81          preorder_traversal (n->left);
82          preorder_traversal (n->right);
83      }
84  }
85  void inorder_traversal (node *n)
86  {
87      if( n != NULL )
88      {
89          inorder_traversal (n->left);
90          cout<<n->value<<",";
91          inorder_traversal (n->right);
92      }
93  }
94  void postorder_traversal (node *n)
95  {
96      if( n != NULL )
97      {
98          postorder_traversal(n->left);
99          postorder_traversal(n->right);
100         cout<<n->value<<",";
101     }
102 }
```

Example

```
103  int depth (int v)
104  {
105     int d = 0;
106     node *n = r;
107     while(true)
108     {
109        if( n->value > v && n->left != NULL )
110        {
111           n = n->left;
112           d++;
113        }
114        else if( n->value < v && n->right != NULL )
115        {
116           n = n->right;
117           d++;
118        }
119        else if( n->value == v )
120        {
121           return d;
122        }
123        else
124        {
125           return -1;
126        }
127     }
128  }
```

```
129  int h = 0, t_h = -1;
130  void h_height (node *n)
131  {
132     if( n != NULL )
133     {
134        t_h++;
135        h_height(n->left);
136        h_height(n->right);
137        if( t_h > h )
138        {
139           h = t_h;
140        }
141        t_h--;
142     }
143  }
144  int height (int v)
145  {
146     node *n = r;
147     while(true)
148     {
149        if( n->value > v && n->left != NULL )
150        {
151           n = n->left;
152        }
```

Example

```
153         else if( n->value < v && n->right != NULL )
154         {
155             n = n->right;
156         }
157         else if( n->value == v )
158         {
159             h = 0, t_h = -1;
160             h_height(n);
161             return h;
162         }
163         else
164         {
165             return -1;
166         }
167     }
168 }
169 void print_level(node *n, int level)
170 {
171     if(level == 0)
172     {
173         cout<<n->value<<",";
174     }
175     else
176     {
177         if( n->left != NULL ) {  print_level(n->left,level-1);   }
178         if( n->right != NULL ){  print_level(n->right,level-1);  }
179     }
180 }
```

```
181 void breadth_first_traversal()
182 {
183     for(int i=0 ; i <= height(r->value) ; i++)
184     {
185         print_level(r,i);
186         cout<<"|";
187     }
188     cout<<endl;
189 }
190 void delete_node_child(node *n)
191 {
192     if( n->right->left == NULL  )
193     {
194         int s = n->right->value;
195         delete_node( s );
196         n->value = s;
197     }
198     else
199     {
200         node *tn = n->right;
201         while(true)
202         {
203             if(tn->left == NULL)
204             {
205                 break;
206             }
```

Example

```
207             tn = tn->left;
208         }
209         int s = tn->value;
210         delete_node(s);
211         n->value = s;
212     }
213 }
214 void delete_node(int v)
215 {
216     if( search(v) )
217     {
218         if( r->value == v &&  r->left == NULL &&
219             r->right == NULL )
220         {
221             return;
222         }
223         else if( r->value == v && r->left == NULL &&
224                 r->right != NULL  )
225         {
226             r = r->right;
227         }
228         else if( r->value == v && r->left != NULL &&
229                 r->right == NULL )
230         {
231             r = r->left;
232         }
233         else if( r->value == v && r->left != NULL &&
234                 r->right != NULL )
235         {
236             delete_node_child(r);
237         }
```

```
238     else
239     {
240         node *p = r;
241         while(true)
242         {
243         if( p->value > v && p->left != NULL )
244         {
245             node *c = p->left;
246             if( c->value == v )
247             {
248                 if( c->left == NULL &&
249                     c->right == NULL )
250                 {
251                     p->left = NULL;
252                     return;
253                 }
254                 else if( c->left == NULL &&
255                         c->right != NULL )
256                 {
257                     p->left = c->right;
258                     return;
259                 }
260                 else if( c->left != NULL &&
261                         c->right == NULL )
262                 {
263                     p->left = c->left;
264                     return;
265                 }
```

Example

```
266                     else if( c->left != NULL &&
267                             c->right != NULL )
268                     {
269                         delete_node_child(c);
270                         return;
271                     }
272                 }
273                 p = p->left;
274             }
275             else if( p->value < v && p->right != NULL )
276             {
277                 node *c = p->right;
278                 if( c->value == v )
279                 {
280                     if( c->left == NULL &&
281                         c->right == NULL )
282                     {
283                         p->right = NULL;
284                         return;
285                     }
286                     else if( c->left == NULL &&
287                             c->right != NULL )
288                     {
289                         p->right = c->right;
290                         return;
291                     }
292                     else if( c->left != NULL &&
293                             c->right == NULL )
294                     {
295                         p->right = c->left;
296                         return;
297                     }
```

```
298                     else if( c->left != NULL &&
299                             c->right != NULL )
300                     {
301                         delete_node_child(c);
302                         return;
303                     }
304                 }
305                 p = p->right;
306             }
307         }
308     }
309 }
310 };
311 int main()
312 {
313     tree t(20);                 t.breadth_first_traversal();
314     t.add_node( 10 );           t.breadth_first_traversal();
315     t.add_node( 30 );           t.breadth_first_traversal();
316     t.add_node( 5 );            t.breadth_first_traversal();
317     t.add_node( 15 );           t.breadth_first_traversal();
318     t.add_node( 25 );           t.breadth_first_traversal();
319     t.add_node( 35 );           t.breadth_first_traversal();
320     t.add_node( 12 );           t.breadth_first_traversal();
321     t.add_node( 17 );           t.breadth_first_traversal();
322     cout<<t.search(11)<<endl;
323     cout<<t.search(35)<<endl;
324     cout<<t.height(20)<<endl;
325     cout<<t.height(15)<<endl;
326     cout<<t.depth(20)<<endl;
327     cout<<t.depth(15)<<endl;
328
```

Example

```
329     t.inorder_traversal(t.r);       cout<<endl;
330     t.postorder_traversal(t.r);     cout<<endl;
331     t.preorder_traversal(t.r);      cout<<endl;
332     t.delete_node(25);              t.breadth_first_traversal();
333     t.delete_node(30);              t.breadth_first_traversal();
334     t.delete_node(10);              t.breadth_first_traversal();
335     t.delete_node(12);              t.breadth_first_traversal();
336     t.delete_node(15);              t.breadth_first_traversal();
337     t.delete_node(17);              t.breadth_first_traversal();
338     t.delete_node(20);              t.breadth_first_traversal();
339     t.delete_node(20);              t.breadth_first_traversal();
340     t.delete_node(35);              t.breadth_first_traversal();
341     t.delete_node(5);               t.breadth_first_traversal();
342     return 0;
343 }
```

Reference

Allen, W. M. (2007). *Data structures and algorithm analysis in C++*. Pearson Education India.

Nell B. Dale. (2003). *C++ plus data structures*. Jones & Bartlett Learning.

เฉียบวุฒิ รัตนวิลัยสกุล. (2023). โครงสร้างข้อมูล. มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

https://ece.uwaterloo.ca/~dwharder/aads/Lecture_materials/