

BinarySearchTree.java

```
1 package dataStructures;
2
3 public class BinarySearchTree<K extends Comparable<K>, V>
4     implements
5         OrderedDictionary<K, V> {
6
7     static final long serialVersionUID = 0L;
8
9     // The root of the tree.
10    protected BSTNode<K, V> root;
11
12    // Number of entries in the tree.
13    protected int currentSize;
14
15    protected static class PathStep<K, V> {
16
17        // The parent of the node.
18        public BSTNode<K, V> parent;
19
20        // The node is the left or the right child of parent.
21        public boolean isLeftChild;
22
23        public PathStep(BSTNode<K, V> theParent, boolean toTheLeft)
24        {
25            parent = theParent;
26            isLeftChild = toTheLeft;
27        }
28
29        public void set(BSTNode<K, V> newParent, boolean toTheLeft)
30        {
31            parent = newParent;
32            isLeftChild = toTheLeft;
33        }
34
35        public BinarySearchTree() {
36            root = null;
37            currentSize = 0;
38        }
39
40        public void add(K key, V value) {
41            if (root == null) {
42                root = new BSTNode<K, V>(key, value);
43                currentSize++;
44            } else {
45                add(root, key, value);
46            }
47        }
48
49        private void add(BSTNode<K, V> node, K key, V value) {
50            if (isLeftChild) {
51                if (node.left == null) {
52                    node.left = new BSTNode<K, V>(key, value);
53                    currentSize++;
54                } else {
55                    add(node.left, key, value);
56                }
57            } else {
58                if (node.right == null) {
59                    node.right = new BSTNode<K, V>(key, value);
60                    currentSize++;
61                } else {
62                    add(node.right, key, value);
63                }
64            }
65        }
66
67        public V get(K key) {
68            return get(root, key);
69        }
70
71        private V get(BSTNode<K, V> node, K key) {
72            if (node == null) {
73                return null;
74            }
75            if (node.key.equals(key)) {
76                return node.value;
77            } else if (key.compareTo(node.key) < 0) {
78                return get(node.left, key);
79            } else {
80                return get(node.right, key);
81            }
82        }
83
84        public void remove(K key) {
85            remove(root, key);
86        }
87
88        private void remove(BSTNode<K, V> node, K key) {
89            if (node == null) {
90                return;
91            }
92            if (node.key.equals(key)) {
93                if (node.left == null && node.right == null) {
94                    node = null;
95                } else if (node.left == null) {
96                    node = node.right;
97                } else if (node.right == null) {
98                    node = node.left;
99                } else {
100                    BSTNode<K, V> temp = node.right;
101                    while (temp.left != null) {
102                        temp = temp.left;
103                    }
104                    temp.left = node.left;
105                    node = temp;
106                }
107            } else if (key.compareTo(node.key) < 0) {
108                remove(node.left, key);
109            } else {
110                remove(node.right, key);
111            }
112        }
113
114        public void print() {
115            print(root);
116        }
117
118        private void print(BSTNode<K, V> node) {
119            if (node == null) {
120                return;
121            }
122            System.out.println("Key: " + node.key + ", Value: " + node.value);
123            print(node.left);
124            print(node.right);
125        }
126
127        public int size() {
128            return currentSize;
129        }
130
131        public void clear() {
132            clear(root);
133        }
134
135        private void clear(BSTNode<K, V> node) {
136            if (node == null) {
137                return;
138            }
139            clear(node.left);
140            clear(node.right);
141            node = null;
142        }
143
144        public void printInOrder() {
145            printInOrder(root);
146        }
147
148        private void printInOrder(BSTNode<K, V> node) {
149            if (node == null) {
150                return;
151            }
152            printInOrder(node.left);
153            System.out.println("Key: " + node.key + ", Value: " + node.value);
154            printInOrder(node.right);
155        }
156
157        public void printPreOrder() {
158            printPreOrder(root);
159        }
160
161        private void printPreOrder(BSTNode<K, V> node) {
162            if (node == null) {
163                return;
164            }
165            System.out.println("Key: " + node.key + ", Value: " + node.value);
166            printPreOrder(node.left);
167            printPreOrder(node.right);
168        }
169
170        public void printPostOrder() {
171            printPostOrder(root);
172        }
173
174        private void printPostOrder(BSTNode<K, V> node) {
175            if (node == null) {
176                return;
177            }
178            printPostOrder(node.left);
179            printPostOrder(node.right);
180            System.out.println("Key: " + node.key + ", Value: " + node.value);
181        }
182
183        public void printLevelOrder() {
184            printLevelOrder();
185        }
186
187        private void printLevelOrder() {
188            if (root == null) {
189                return;
190            }
191            Queue<BSTNode<K, V>> queue = new Queue<BSTNode<K, V>>();
192            queue.add(root);
193            while (!queue.isEmpty()) {
194                BSTNode<K, V> current = queue.remove();
195                System.out.println("Key: " + current.key + ", Value: " + current.value);
196                if (current.left != null) {
197                    queue.add(current.left);
198                }
199                if (current.right != null) {
200                    queue.add(current.right);
201                }
202            }
203        }
204
205        public void printDepthOrder() {
206            printDepthOrder();
207        }
208
209        private void printDepthOrder() {
210            if (root == null) {
211                return;
212            }
213            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
214            stack.push(root);
215            while (!stack.isEmpty()) {
216                BSTNode<K, V> current = stack.pop();
217                System.out.println("Key: " + current.key + ", Value: " + current.value);
218                if (current.right != null) {
219                    stack.push(current.right);
220                }
221                if (current.left != null) {
222                    stack.push(current.left);
223                }
224            }
225        }
226
227        public void printDepthOrderLeftFirst() {
228            printDepthOrderLeftFirst();
229        }
230
231        private void printDepthOrderLeftFirst() {
232            if (root == null) {
233                return;
234            }
235            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
236            stack.push(root);
237            while (!stack.isEmpty()) {
238                BSTNode<K, V> current = stack.pop();
239                System.out.println("Key: " + current.key + ", Value: " + current.value);
240                if (current.left != null) {
241                    stack.push(current.left);
242                }
243                if (current.right != null) {
244                    stack.push(current.right);
245                }
246            }
247        }
248
249        public void printDepthOrderRightFirst() {
250            printDepthOrderRightFirst();
251        }
252
253        private void printDepthOrderRightFirst() {
254            if (root == null) {
255                return;
256            }
257            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
258            stack.push(root);
259            while (!stack.isEmpty()) {
260                BSTNode<K, V> current = stack.pop();
261                System.out.println("Key: " + current.key + ", Value: " + current.value);
262                if (current.right != null) {
263                    stack.push(current.right);
264                }
265                if (current.left != null) {
266                    stack.push(current.left);
267                }
268            }
269        }
270
271        public void printDepthOrderLeftFirst() {
272            printDepthOrderLeftFirst();
273        }
274
275        private void printDepthOrderLeftFirst() {
276            if (root == null) {
277                return;
278            }
279            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
280            stack.push(root);
281            while (!stack.isEmpty()) {
282                BSTNode<K, V> current = stack.pop();
283                System.out.println("Key: " + current.key + ", Value: " + current.value);
284                if (current.left != null) {
285                    stack.push(current.left);
286                }
287                if (current.right != null) {
288                    stack.push(current.right);
289                }
290            }
291        }
292
293        public void printDepthOrderRightFirst() {
294            printDepthOrderRightFirst();
295        }
296
297        private void printDepthOrderRightFirst() {
298            if (root == null) {
299                return;
300            }
301            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
302            stack.push(root);
303            while (!stack.isEmpty()) {
304                BSTNode<K, V> current = stack.pop();
305                System.out.println("Key: " + current.key + ", Value: " + current.value);
306                if (current.right != null) {
307                    stack.push(current.right);
308                }
309                if (current.left != null) {
310                    stack.push(current.left);
311                }
312            }
313        }
314
315        public void printDepthOrderLeftFirst() {
316            printDepthOrderLeftFirst();
317        }
318
319        private void printDepthOrderLeftFirst() {
320            if (root == null) {
321                return;
322            }
323            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
324            stack.push(root);
325            while (!stack.isEmpty()) {
326                BSTNode<K, V> current = stack.pop();
327                System.out.println("Key: " + current.key + ", Value: " + current.value);
328                if (current.left != null) {
329                    stack.push(current.left);
330                }
331                if (current.right != null) {
332                    stack.push(current.right);
333                }
334            }
335        }
336
337        public void printDepthOrderRightFirst() {
338            printDepthOrderRightFirst();
339        }
340
341        private void printDepthOrderRightFirst() {
342            if (root == null) {
343                return;
344            }
345            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
346            stack.push(root);
347            while (!stack.isEmpty()) {
348                BSTNode<K, V> current = stack.pop();
349                System.out.println("Key: " + current.key + ", Value: " + current.value);
350                if (current.right != null) {
351                    stack.push(current.right);
352                }
353                if (current.left != null) {
354                    stack.push(current.left);
355                }
356            }
357        }
358
359        public void printDepthOrderLeftFirst() {
360            printDepthOrderLeftFirst();
361        }
362
363        private void printDepthOrderLeftFirst() {
364            if (root == null) {
365                return;
366            }
367            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
368            stack.push(root);
369            while (!stack.isEmpty()) {
370                BSTNode<K, V> current = stack.pop();
371                System.out.println("Key: " + current.key + ", Value: " + current.value);
372                if (current.left != null) {
373                    stack.push(current.left);
374                }
375                if (current.right != null) {
376                    stack.push(current.right);
377                }
378            }
379        }
380
381        public void printDepthOrderRightFirst() {
382            printDepthOrderRightFirst();
383        }
384
385        private void printDepthOrderRightFirst() {
386            if (root == null) {
387                return;
388            }
389            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
390            stack.push(root);
391            while (!stack.isEmpty()) {
392                BSTNode<K, V> current = stack.pop();
393                System.out.println("Key: " + current.key + ", Value: " + current.value);
394                if (current.right != null) {
395                    stack.push(current.right);
396                }
397                if (current.left != null) {
398                    stack.push(current.left);
399                }
400            }
401        }
402
403        public void printDepthOrderLeftFirst() {
404            printDepthOrderLeftFirst();
405        }
406
407        private void printDepthOrderLeftFirst() {
408            if (root == null) {
409                return;
410            }
411            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
412            stack.push(root);
413            while (!stack.isEmpty()) {
414                BSTNode<K, V> current = stack.pop();
415                System.out.println("Key: " + current.key + ", Value: " + current.value);
416                if (current.left != null) {
417                    stack.push(current.left);
418                }
419                if (current.right != null) {
420                    stack.push(current.right);
421                }
422            }
423        }
424
425        public void printDepthOrderRightFirst() {
426            printDepthOrderRightFirst();
427        }
428
429        private void printDepthOrderRightFirst() {
430            if (root == null) {
431                return;
432            }
433            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
434            stack.push(root);
435            while (!stack.isEmpty()) {
436                BSTNode<K, V> current = stack.pop();
437                System.out.println("Key: " + current.key + ", Value: " + current.value);
438                if (current.right != null) {
439                    stack.push(current.right);
440                }
441                if (current.left != null) {
442                    stack.push(current.left);
443                }
444            }
445        }
446
447        public void printDepthOrderLeftFirst() {
448            printDepthOrderLeftFirst();
449        }
450
451        private void printDepthOrderLeftFirst() {
452            if (root == null) {
453                return;
454            }
455            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
456            stack.push(root);
457            while (!stack.isEmpty()) {
458                BSTNode<K, V> current = stack.pop();
459                System.out.println("Key: " + current.key + ", Value: " + current.value);
460                if (current.left != null) {
461                    stack.push(current.left);
462                }
463                if (current.right != null) {
464                    stack.push(current.right);
465                }
466            }
467        }
468
469        public void printDepthOrderRightFirst() {
470            printDepthOrderRightFirst();
471        }
472
473        private void printDepthOrderRightFirst() {
474            if (root == null) {
475                return;
476            }
477            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
478            stack.push(root);
479            while (!stack.isEmpty()) {
480                BSTNode<K, V> current = stack.pop();
481                System.out.println("Key: " + current.key + ", Value: " + current.value);
482                if (current.right != null) {
483                    stack.push(current.right);
484                }
485                if (current.left != null) {
486                    stack.push(current.left);
487                }
488            }
489        }
490
491        public void printDepthOrderLeftFirst() {
492            printDepthOrderLeftFirst();
493        }
494
495        private void printDepthOrderLeftFirst() {
496            if (root == null) {
497                return;
498            }
499            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
500            stack.push(root);
501            while (!stack.isEmpty()) {
502                BSTNode<K, V> current = stack.pop();
503                System.out.println("Key: " + current.key + ", Value: " + current.value);
504                if (current.left != null) {
505                    stack.push(current.left);
506                }
507                if (current.right != null) {
508                    stack.push(current.right);
509                }
510            }
511        }
512
513        public void printDepthOrderRightFirst() {
514            printDepthOrderRightFirst();
515        }
516
517        private void printDepthOrderRightFirst() {
518            if (root == null) {
519                return;
520            }
521            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
522            stack.push(root);
523            while (!stack.isEmpty()) {
524                BSTNode<K, V> current = stack.pop();
525                System.out.println("Key: " + current.key + ", Value: " + current.value);
526                if (current.right != null) {
527                    stack.push(current.right);
528                }
529                if (current.left != null) {
530                    stack.push(current.left);
531                }
532            }
533        }
534
535        public void printDepthOrderLeftFirst() {
536            printDepthOrderLeftFirst();
537        }
538
539        private void printDepthOrderLeftFirst() {
540            if (root == null) {
541                return;
542            }
543            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
544            stack.push(root);
545            while (!stack.isEmpty()) {
546                BSTNode<K, V> current = stack.pop();
547                System.out.println("Key: " + current.key + ", Value: " + current.value);
548                if (current.left != null) {
549                    stack.push(current.left);
550                }
551                if (current.right != null) {
552                    stack.push(current.right);
553                }
554            }
555        }
556
557        public void printDepthOrderRightFirst() {
558            printDepthOrderRightFirst();
559        }
560
561        private void printDepthOrderRightFirst() {
562            if (root == null) {
563                return;
564            }
565            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
566            stack.push(root);
567            while (!stack.isEmpty()) {
568                BSTNode<K, V> current = stack.pop();
569                System.out.println("Key: " + current.key + ", Value: " + current.value);
570                if (current.right != null) {
571                    stack.push(current.right);
572                }
573                if (current.left != null) {
574                    stack.push(current.left);
575                }
576            }
577        }
578
579        public void printDepthOrderLeftFirst() {
580            printDepthOrderLeftFirst();
581        }
582
583        private void printDepthOrderLeftFirst() {
584            if (root == null) {
585                return;
586            }
587            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
588            stack.push(root);
589            while (!stack.isEmpty()) {
590                BSTNode<K, V> current = stack.pop();
591                System.out.println("Key: " + current.key + ", Value: " + current.value);
592                if (current.left != null) {
593                    stack.push(current.left);
594                }
595                if (current.right != null) {
596                    stack.push(current.right);
597                }
598            }
599        }
600
601        public void printDepthOrderRightFirst() {
602            printDepthOrderRightFirst();
603        }
604
605        private void printDepthOrderRightFirst() {
606            if (root == null) {
607                return;
608            }
609            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
610            stack.push(root);
611            while (!stack.isEmpty()) {
612                BSTNode<K, V> current = stack.pop();
613                System.out.println("Key: " + current.key + ", Value: " + current.value);
614                if (current.right != null) {
615                    stack.push(current.right);
616                }
617                if (current.left != null) {
618                    stack.push(current.left);
619                }
620            }
621        }
622
623        public void printDepthOrderLeftFirst() {
624            printDepthOrderLeftFirst();
625        }
626
627        private void printDepthOrderLeftFirst() {
628            if (root == null) {
629                return;
630            }
631            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
632            stack.push(root);
633            while (!stack.isEmpty()) {
634                BSTNode<K, V> current = stack.pop();
635                System.out.println("Key: " + current.key + ", Value: " + current.value);
636                if (current.left != null) {
637                    stack.push(current.left);
638                }
639                if (current.right != null) {
640                    stack.push(current.right);
641                }
642            }
643        }
644
645        public void printDepthOrderRightFirst() {
646            printDepthOrderRightFirst();
647        }
648
649        private void printDepthOrderRightFirst() {
650            if (root == null) {
651                return;
652            }
653            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
654            stack.push(root);
655            while (!stack.isEmpty()) {
656                BSTNode<K, V> current = stack.pop();
657                System.out.println("Key: " + current.key + ", Value: " + current.value);
658                if (current.right != null) {
659                    stack.push(current.right);
660                }
661                if (current.left != null) {
662                    stack.push(current.left);
663                }
664            }
665        }
666
667        public void printDepthOrderLeftFirst() {
668            printDepthOrderLeftFirst();
669        }
670
671        private void printDepthOrderLeftFirst() {
672            if (root == null) {
673                return;
674            }
675            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
676            stack.push(root);
677            while (!stack.isEmpty()) {
678                BSTNode<K, V> current = stack.pop();
679                System.out.println("Key: " + current.key + ", Value: " + current.value);
680                if (current.left != null) {
681                    stack.push(current.left);
682                }
683                if (current.right != null) {
684                    stack.push(current.right);
685                }
686            }
687        }
688
689        public void printDepthOrderRightFirst() {
690            printDepthOrderRightFirst();
691        }
692
693        private void printDepthOrderRightFirst() {
694            if (root == null) {
695                return;
696            }
697            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
698            stack.push(root);
699            while (!stack.isEmpty()) {
700                BSTNode<K, V> current = stack.pop();
701                System.out.println("Key: " + current.key + ", Value: " + current.value);
702                if (current.right != null) {
703                    stack.push(current.right);
704                }
705                if (current.left != null) {
706                    stack.push(current.left);
707                }
708            }
709        }
710
711        public void printDepthOrderLeftFirst() {
712            printDepthOrderLeftFirst();
713        }
714
715        private void printDepthOrderLeftFirst() {
716            if (root == null) {
717                return;
718            }
719            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
720            stack.push(root);
721            while (!stack.isEmpty()) {
722                BSTNode<K, V> current = stack.pop();
723                System.out.println("Key: " + current.key + ", Value: " + current.value);
724                if (current.left != null) {
725                    stack.push(current.left);
726                }
727                if (current.right != null) {
728                    stack.push(current.right);
729                }
730            }
731        }
732
733        public void printDepthOrderRightFirst() {
734            printDepthOrderRightFirst();
735        }
736
737        private void printDepthOrderRightFirst() {
738            if (root == null) {
739                return;
740            }
741            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
742            stack.push(root);
743            while (!stack.isEmpty()) {
744                BSTNode<K, V> current = stack.pop();
745                System.out.println("Key: " + current.key + ", Value: " + current.value);
746                if (current.right != null) {
747                    stack.push(current.right);
748                }
749                if (current.left != null) {
750                    stack.push(current.left);
751                }
752            }
753        }
754
755        public void printDepthOrderLeftFirst() {
756            printDepthOrderLeftFirst();
757        }
758
759        private void printDepthOrderLeftFirst() {
760            if (root == null) {
761                return;
762            }
763            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
764            stack.push(root);
765            while (!stack.isEmpty()) {
766                BSTNode<K, V> current = stack.pop();
767                System.out.println("Key: " + current.key + ", Value: " + current.value);
768                if (current.left != null) {
769                    stack.push(current.left);
770                }
771                if (current.right != null) {
772                    stack.push(current.right);
773                }
774            }
775        }
776
777        public void printDepthOrderRightFirst() {
778            printDepthOrderRightFirst();
779        }
780
781        private void printDepthOrderRightFirst() {
782            if (root == null) {
783                return;
784            }
785            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
786            stack.push(root);
787            while (!stack.isEmpty()) {
788                BSTNode<K, V> current = stack.pop();
789                System.out.println("Key: " + current.key + ", Value: " + current.value);
790                if (current.right != null) {
791                    stack.push(current.right);
792                }
793                if (current.left != null) {
794                    stack.push(current.left);
795                }
796            }
797        }
798
799        public void printDepthOrderLeftFirst() {
800            printDepthOrderLeftFirst();
801        }
802
803        private void printDepthOrderLeftFirst() {
804            if (root == null) {
805                return;
806            }
807            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
808            stack.push(root);
809            while (!stack.isEmpty()) {
810                BSTNode<K, V> current = stack.pop();
811                System.out.println("Key: " + current.key + ", Value: " + current.value);
812                if (current.left != null) {
813                    stack.push(current.left);
814                }
815                if (current.right != null) {
816                    stack.push(current.right);
817                }
818            }
819        }
820
821        public void printDepthOrderRightFirst() {
822            printDepthOrderRightFirst();
823        }
824
825        private void printDepthOrderRightFirst() {
826            if (root == null) {
827                return;
828            }
829            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
830            stack.push(root);
831            while (!stack.isEmpty()) {
832                BSTNode<K, V> current = stack.pop();
833                System.out.println("Key: " + current.key + ", Value: " + current.value);
834                if (current.right != null) {
835                    stack.push(current.right);
836                }
837                if (current.left != null) {
838                    stack.push(current.left);
839                }
840            }
841        }
842
843        public void printDepthOrderLeftFirst() {
844            printDepthOrderLeftFirst();
845        }
846
847        private void printDepthOrderLeftFirst() {
848            if (root == null) {
849                return;
850            }
851            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
852            stack.push(root);
853            while (!stack.isEmpty()) {
854                BSTNode<K, V> current = stack.pop();
855                System.out.println("Key: " + current.key + ", Value: " + current.value);
856                if (current.left != null) {
857                    stack.push(current.left);
858                }
859                if (current.right != null) {
860                    stack.push(current.right);
861                }
862            }
863        }
864
865        public void printDepthOrderRightFirst() {
866            printDepthOrderRightFirst();
867        }
868
869        private void printDepthOrderRightFirst() {
870            if (root == null) {
871                return;
872            }
873            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
874            stack.push(root);
875            while (!stack.isEmpty()) {
876                BSTNode<K, V> current = stack.pop();
877                System.out.println("Key: " + current.key + ", Value: " + current.value);
878                if (current.right != null) {
879                    stack.push(current.right);
880                }
881                if (current.left != null) {
882                    stack.push(current.left);
883                }
884            }
885        }
886
887        public void printDepthOrderLeftFirst() {
888            printDepthOrderLeftFirst();
889        }
890
891        private void printDepthOrderLeftFirst() {
892            if (root == null) {
893                return;
894            }
895            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
896            stack.push(root);
897            while (!stack.isEmpty()) {
898                BSTNode<K, V> current = stack.pop();
899                System.out.println("Key: " + current.key + ", Value: " + current.value);
900                if (current.left != null) {
901                    stack.push(current.left);
902                }
903                if (current.right != null) {
904                    stack.push(current.right);
905                }
906            }
907        }
908
909        public void printDepthOrderRightFirst() {
910            printDepthOrderRightFirst();
911        }
912
913        private void printDepthOrderRightFirst() {
914            if (root == null) {
915                return;
916            }
917            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
918            stack.push(root);
919            while (!stack.isEmpty()) {
920                BSTNode<K, V> current = stack.pop();
921                System.out.println("Key: " + current.key + ", Value: " + current.value);
922                if (current.right != null) {
923                    stack.push(current.right);
924                }
925                if (current.left != null) {
926                    stack.push(current.left);
927                }
928            }
929        }
930
931        public void printDepthOrderLeftFirst() {
932            printDepthOrderLeftFirst();
933        }
934
935        private void printDepthOrderLeftFirst() {
936            if (root == null) {
937                return;
938            }
939            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
940            stack.push(root);
941            while (!stack.isEmpty()) {
942                BSTNode<K, V> current = stack.pop();
943                System.out.println("Key: " + current.key + ", Value: " + current.value);
944                if (current.left != null) {
945                    stack.push(current.left);
946                }
947                if (current.right != null) {
948                    stack.push(current.right);
949                }
950            }
951        }
952
953        public void printDepthOrderRightFirst() {
954            printDepthOrderRightFirst();
955        }
956
957        private void printDepthOrderRightFirst() {
958            if (root == null) {
959                return;
960            }
961            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
962            stack.push(root);
963            while (!stack.isEmpty()) {
964                BSTNode<K, V> current = stack.pop();
965                System.out.println("Key: " + current.key + ", Value: " + current.value);
966                if (current.right != null) {
967                    stack.push(current.right);
968                }
969                if (current.left != null) {
970                    stack.push(current.left);
971                }
972            }
973        }
974
975        public void printDepthOrderLeftFirst() {
976            printDepthOrderLeftFirst();
977        }
978
979        private void printDepthOrderLeftFirst() {
980            if (root == null) {
981                return;
982            }
983            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
984            stack.push(root);
985            while (!stack.isEmpty()) {
986                BSTNode<K, V> current = stack.pop();
987                System.out.println("Key: " + current.key + ", Value: " + current.value);
988                if (current.left != null) {
989                    stack.push(current.left);
990                }
991                if (current.right != null) {
992                    stack.push(current.right);
993                }
994            }
995        }
996
997        public void printDepthOrderRightFirst() {
998            printDepthOrderRightFirst();
999        }
1000
1001        private void printDepthOrderRightFirst() {
1002            if (root == null) {
1003                return;
1004            }
1005            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1006            stack.push(root);
1007            while (!stack.isEmpty()) {
1008                BSTNode<K, V> current = stack.pop();
1009                System.out.println("Key: " + current.key + ", Value: " + current.value);
1010                if (current.right != null) {
1011                    stack.push(current.right);
1012                }
1013                if (current.left != null) {
1014                    stack.push(current.left);
1015                }
1016            }
1017        }
1018
1019        public void printDepthOrderLeftFirst() {
1020            printDepthOrderLeftFirst();
1021        }
1022
1023        private void printDepthOrderLeftFirst() {
1024            if (root == null) {
1025                return;
1026            }
1027            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1028            stack.push(root);
1029            while (!stack.isEmpty()) {
1030                BSTNode<K, V> current = stack.pop();
1031                System.out.println("Key: " + current.key + ", Value: " + current.value);
1032                if (current.left != null) {
1033                    stack.push(current.left);
1034                }
1035                if (current.right != null) {
1036                    stack.push(current.right);
1037                }
1038            }
1039        }
1040
1041        public void printDepthOrderRightFirst() {
1042            printDepthOrderRightFirst();
1043        }
1044
1045        private void printDepthOrderRightFirst() {
1046            if (root == null) {
1047                return;
1048            }
1049            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1050            stack.push(root);
1051            while (!stack.isEmpty()) {
1052                BSTNode<K, V> current = stack.pop();
1053                System.out.println("Key: " + current.key + ", Value: " + current.value);
1054                if (current.right != null) {
1055                    stack.push(current.right);
1056                }
1057                if (current.left != null) {
1058                    stack.push(current.left);
1059                }
1060            }
1061        }
1062
1063        public void printDepthOrderLeftFirst() {
1064            printDepthOrderLeftFirst();
1065        }
1066
1067        private void printDepthOrderLeftFirst() {
1068            if (root == null) {
1069                return;
1070            }
1071            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1072            stack.push(root);
1073            while (!stack.isEmpty()) {
1074                BSTNode<K, V> current = stack.pop();
1075                System.out.println("Key: " + current.key + ", Value: " + current.value);
1076                if (current.left != null) {
1077                    stack.push(current.left);
1078                }
1079                if (current.right != null) {
1080                    stack.push(current.right);
1081                }
1082            }
1083        }
1084
1085        public void printDepthOrderRightFirst() {
1086            printDepthOrderRightFirst();
1087        }
1088
1089        private void printDepthOrderRightFirst() {
1090            if (root == null) {
1091                return;
1092            }
1093            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1094            stack.push(root);
1095            while (!stack.isEmpty()) {
1096                BSTNode<K, V> current = stack.pop();
1097                System.out.println("Key: " + current.key + ", Value: " + current.value);
1098                if (current.right != null) {
1099                    stack.push(current.right);
1100                }
1101                if (current.left != null) {
1102                    stack.push(current.left);
1103                }
1104            }
1105        }
1106
1107        public void printDepthOrderLeftFirst() {
1108            printDepthOrderLeftFirst();
1109        }
1110
1111        private void printDepthOrderLeftFirst() {
1112            if (root == null) {
1113                return;
1114            }
1115            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1116            stack.push(root);
1117            while (!stack.isEmpty()) {
1118                BSTNode<K, V> current = stack.pop();
1119                System.out.println("Key: " + current.key + ", Value: " + current.value);
1120                if (current.left != null) {
1121                    stack.push(current.left);
1122                }
1123                if (current.right != null) {
1124                    stack.push(current.right);
1125                }
1126            }
1127        }
1128
1129        public void printDepthOrderRightFirst() {
1130            printDepthOrderRightFirst();
1131        }
1132
1133        private void printDepthOrderRightFirst() {
1134            if (root == null) {
1135                return;
1136            }
1137            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1138            stack.push(root);
1139            while (!stack.isEmpty()) {
1140                BSTNode<K, V> current = stack.pop();
1141                System.out.println("Key: " + current.key + ", Value: " + current.value);
1142                if (current.right != null) {
1143                    stack.push(current.right);
1144                }
1145                if (current.left != null) {
1146                    stack.push(current.left);
1147                }
1148            }
1149        }
1150
1151        public void printDepthOrderLeftFirst() {
1152            printDepthOrderLeftFirst();
1153        }
1154
1155        private void printDepthOrderLeftFirst() {
1156            if (root == null) {
1157                return;
1158            }
1159            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1160            stack.push(root);
1161            while (!stack.isEmpty()) {
1162                BSTNode<K, V> current = stack.pop();
1163                System.out.println("Key: " + current.key + ", Value: " + current.value);
1164                if (current.left != null) {
1165                    stack.push(current.left);
1166                }
1167                if (current.right != null) {
1168                    stack.push(current.right);
1169                }
1170            }
1171        }
1172
1173        public void printDepthOrderRightFirst() {
1174            printDepthOrderRightFirst();
1175        }
1176
1177        private void printDepthOrderRightFirst() {
1178            if (root == null) {
1179                return;
1180            }
1181            Stack<BSTNode<K, V>> stack = new Stack<BSTNode<K, V>>();
1182            stack.push(root);
1183            while (!stack.isEmpty()) {
1184                BSTNode<K, V> current = stack.pop();
1185                System.out.println("Key: " + current.key + ", Value: " + current.value);
1186                if (current.right != null) {
1187                    stack.push(current.right);
1188                }
1189                if (current.left != null) {
1190                    stack.push(current.left);
1191                }
1192            }
1193        }
1194
1195        public void printDepthOrderLeftFirst() {
1196            printDepthOrderLeftFirst();
1197        }
1198
1199        private void printDepthOrderLeftFirst() {
1200            if (root == null) {
1201                return;
1202            }
1203            Stack<BST
```

BinarySearchTree.java

```
39 // Returns true iff the dictionary contains no entries.
40 public boolean isEmpty() {
41     return root == null;
42 }
43
44 // Returns the number of entries in the dictionary.
45 public int size() {
46     return currentSize;
47 }
48
49 // If there is an entry in the dictionary whose key is the
50 // specified key,
51 // returns its value; otherwise, returns null.
52 public V find(K key) {
53     BSTNode<K, V> node = this.findNode(root, key);
54     if (node == null)
55         return null;
56     else
57         return node.getValue();
58 }
59
60 // Returns the node whose key is the specified key;
61 // or null if no such node exists.
62 protected BSTNode<K, V> findNode(BSTNode<K, V> node, K key) {
63     if (node == null)
64         return null;
65     else {
66         int compResult = key.compareTo(node.getKey());
67         if (compResult == 0)
68             return node;
69         else if (compResult < 0)
70             return this.findNode(node.getLeft(), key);
71         else
72             return this.findNode(node.getRight(), key);
73     }
74 }
75
76 // Returns the entry with the smallest key in the dictionary.
77 public Entry<K, V> minEntry() throws EmptyDictionaryException {
78     if (this.isEmpty())
79         throw new EmptyDictionaryException();
```

BinarySearchTree.java

```
79         return this.minNode(root).getEntry();
80     }
81
82
83     // Returns the node with the smallest key
84     // in the tree rooted at the specified node.
85     // Precondition: node != null.
86     protected BSTNode<K, V> minNode(BSTNode<K, V> node) {
87         if (node.getLeft() == null)
88             return node;
89         else
90             return this.minNode(node.getLeft());
91     }
92
93     // Returns the entry with the largest key in the dictionary.
94     public Entry<K, V> maxEntry() throws EmptyDictionaryException {
95         if (this.isEmpty())
96             throw new EmptyDictionaryException();
97
98         return this.maxNode(root).getEntry();
99     }
100
101    // Returns the node with the largest key
102    // in the tree rooted at the specified node.
103    // Precondition: node != null.
104    protected BSTNode<K, V> maxNode(BSTNode<K, V> node) {
105        if (node.getRight() == null)
106            return node;
107        else
108            return this.maxNode(node.getRight());
109    }
110
111    // Returns the node whose key is the specified key;
112    // or null if no such node exists.
113    // Moreover, stores the last step of the path in lastStep.
114    protected BSTNode<K, V> findNode(K key, PathStep<K, V>
115                                         lastStep) {
116        BSTNode<K, V> node = root;
117        while (node != null) {
118            int compResult = key.compareTo(node.getKey());
119            if (compResult == 0)
```

BinarySearchTree.java

```
119         return node;
120     else if (compResult < 0) {
121         lastStep.set(node, true);
122         node = node.getLeft();
123     } else {
124         lastStep.set(node, false);
125         node = node.getRight();
126     }
127 }
128 return null;
129 }
130
131 // If there is an entry in the dictionary whose key is the
132 // specified key,
133 // replaces its value by the specified value and returns the
134 // old value;
135 // otherwise, inserts the entry (key, value) and returns null.
136 public V insert(K key, V value) {
137     PathStep<K, V> lastStep = new PathStep<K, V>(null, false);
138     BSTNode<K, V> node = this.findNode(key, lastStep);
139     if (node == null) {
140         BSTNode<K, V> newLeaf = new BSTNode<K, V>(key, value);
141         this.linkSubtree(newLeaf, lastStep);
142         currentSize++;
143         return null;
144     } else {
145         V oldValue = node.getValue();
146         node.setValue(value);
147         return oldValue;
148     }
149
150     // Links a new subtree, rooted at the specified node, to the
151     // tree.
152     // The parent of the old subtree is stored in lastStep.
153     protected void linkSubtree(BSTNode<K, V> node, PathStep<K, V>
154     lastStep) {
155         if (lastStep.parent == null)
156             // Change the root of the tree.
157             root = node;
158         else
```

BinarySearchTree.java

```
156     // Change a child of parent.
157     if (lastStep.isLeftChild)
158         lastStep.parent.setLeft(node);
159     else
160         lastStep.parent.setRight(node);
161 }
162
163 // Returns the node with the smallest key
164 // in the tree rooted at the specified node.
165 // Moreover, stores the last step of the path in lastStep.
166 // Precondition: theRoot != null.
167 protected BSTNode<K, V> minNode(BSTNode<K, V> theRoot,
168                                 PathStep<K, V> lastStep) {
169     BSTNode<K, V> node = theRoot;
170     while (node.getLeft() != null) {
171         lastStep.set(node, true);
172         node = node.getLeft();
173     }
174     return node;
175 }
176
177 // If there is an entry in the dictionary whose key is the
178 // specified key,
179 // removes it from the dictionary and returns its value;
180 // otherwise, returns null.
181 public V remove(K key) {
182     PathStep<K, V> lastStep = new PathStep<K, V>(null, false);
183     BSTNode<K, V> node = this.findNode(key, lastStep);
184     if (node == null)
185         return null;
186     else {
187         V oldValue = node.getValue();
188         if (node.getLeft() == null)
189             // The left subtree is empty.
190             this.linkSubtree(node.getRight(), lastStep);
191         else if (node.getRight() == null)
192             // The right subtree is empty.
193             this.linkSubtree(node.getLeft(), lastStep);
194         else {
195             // Node has 2 children. Replace the node's entry
with
```

BinarySearchTree.java

```
195         // the 'minEntry' of the right subtree.
196         lastStep.set(node, false);
197         BSTNode<K, V> minNode =
198             this.minNode(node.getRight(), lastStep);
199             node.setEntry(minNode.getEntry());
200             // Remove the 'minEntry' of the right subtree.
201             this.linkSubtree(minNode.getRight(), lastStep);
202         }
203         currentSize--;
204         return oldValue;
205     }
206
207     // Returns an iterator of the entries in the dictionary
208     // which preserves the key order relation.
209     public Iterator<Entry<K, V>> iterator() {
210         return new BSTInorderIterator<K, V>(root);
211     }
212
213 }
214
```