```
\02-Time-Complexity\02-Time-Complexity-Basic\src\Main.java

public class Main {

    public static void main(String[] args) {

        // 数据规模每次增大10倍进行测试
        // 有兴趣的同学也可以试验一下数据规模每次增大2倍哦:)
        for( int x = 1 ; x <= 9 ; x ++ ){

            int n = (int)Math.pow(10, x);

            long startTime = System.currentTimeMillis();

            long sum = 0;
            for( int i = 0 ; i < n ; i ++ )
                sum += i;

            long endTime = System.currentTimeMillis();

            System.out.println("sum = " + sum);
            System.out.println("10^" + x + " : " + (endTime - startTime) + " ms");
            System.out.println("");
        }
    }
}


\02-Time-Complexity\02-Time-Complexity-Basic\src\Main2.java

public class Main2 {

    private static int sum1(int n){

        assert n >= 0;
        int ret = 0;
        for( int i = 0 ; i <= n ; i ++ )
            ret += i;
        return ret;
    }

    private static int sum2(int n){

        assert n >= 0;
        if( n == 0 )
            return 0;

        return n + sum2(n-1);
    }

    public static void main(String[] args) {

        System.out.println(sum1(10000));
        System.out.println(sum2(10000));
    }
}


\02-Time-Complexity\03-Common-Code-for-Time-Complexity\src\Main.java

public class Main {

    // O(1)
    private static void swap(Object[] arr, int i, int j){

        if(i < 0 || i >= arr.length)
```

```
67              throw new IllegalArgumentException("i is out of bound.");
68
69          if(j < 0 || j >= arr.length)
70              throw new IllegalArgumentException("j is out of bound.");
71
72          Object temp = arr[i];
73          arr[i] = arr[j];
74          arr[j] = temp;
75      }
76
77      // O(n)
78      private static int sum(int n){
79
80          if(n < 0)
81              throw new IllegalArgumentException("n should be greater or equal to zero.");
82
83          int ret = 0;
84          for(int i = 0 ; i <= n ; i ++)
85              ret += i;
86          return ret;
87      }
88
89      private static void reverse(Object[] arr){
90
91          int n = arr.length;
92          for(int i = 0 ; i < n / 2 ; i ++ )
93              swap(arr, i, n - 1 - i);
94      }
95
96      // O(n^2) Time Complexity
97      private static void selectionSort(Comparable[] arr, int n){
98
99          for(int i = 0 ; i < n ; i ++){
100             int minIndex = i;
101             for(int j = i + 1 ; j < n ; j ++)
102                 if(arr[j].compareTo(arr[minIndex]) < 0)
103                     minIndex = j;
104
105             swap(arr, i, minIndex);
106         }
107     }
108
109     // O(n) Time Complexity
110     private static void printInformation(int n){
111
112         for( int i = 1 ; i <= n ; i ++ )
113             for( int j = 1 ; j <= 30 ; j ++ )
114                 System.out.println("Class " + i + " - " + "No. " + j);
115     }
116
117     // O(logn) Time Complexity
118     private static int binarySearch(Comparable[] arr, int n, int target){
119
120         int l = 0, r = n-1;
121         while( l <= r ){
122             int mid = l + (r-l)/2;
123             if(arr[mid].compareTo(target) == 0) return mid;
124             if(arr[mid].compareTo(target) > 0) r = mid - 1;
125             else l = mid + 1;
126         }
127         return -1;
128     }
129
130     private static String intToString(int num){
131
132         StringBuilder s = new StringBuilder("");
```

```
133             String sign = "+";
134             if(num < 0){
135                 num = -num;
136                 sign = "-";
137             }
138 
139             while(num != 0){
140                 s.append(Character.getNumericValue('0') + num % 10);
141                 num /= 10;
142             }
143 
144             if(s.length() == 0)
145                 s.append('0');
146 
147             s.reverse();
148             if(sign == "-")
149                 return sign + s.toString();
150             else
151                 return s.toString();
152         }
153 
154 
155         // O(nlogn)
156         private static void hello(int n){
157 
158             for( int sz = 1 ; sz < n ; sz += sz )
159                 for( int i = 1 ; i < n ; i ++ )
160                     System.out.println("Hello, Algorithm!");
161         }
162 
163 
164         // O(sqrt(n)) Time Complexity
165         private static boolean isPrime(int num){
166 
167             for(int x = 2 ; x*x <= num ; x ++)
168                 if( num % x == 0 )
169                     return false;
170             return true;
171         }
172 
173         private static boolean isPrime2(int num){
174 
175             if( num <= 1 ) return false;
176             if( num == 2 ) return true;
177             if( num % 2 == 0 ) return false;
178 
179             for(int x = 3 ; x * x <= num ; x += 2)
180                 if( num%x == 0 )
181                     return false;
182 
183             return true;
184         }
185 
186         public static void main(String[] args) {
187 
188             System.out.println(intToString(123));
189             System.out.println(intToString(0));
190             System.out.println(intToString(-123));
191 
192             System.out.println();
193 
194             if(isPrime2(137)) System.out.println("137 is a prime.");
195             else System.out.println("137 is not a prime.");
196 
197             if(isPrime2(121)) System.out.println("121 is a prime.");
198             else System.out.println("121 is not a prime.");
```

```
    }
}


\02-Time-Complexity\04-Time-Complexity-Experiments\src\Main.java

/**
 * Created by liuyubobobo.
 */
public class Main {

    public static void main(String[] args) {

        // 数据规模倍乘测试findMax
        // O(n)
        System.out.println("Test for findMax:");
        for( int i = 10 ; i <= 28 ; i ++ ){

            int n = (int)Math.pow(2, i);
            Integer[] arr = MyUtil.generateRandomArray(n, 0, 100000000);

            long startTime = System.currentTimeMillis();
            Integer maxValue = MyAlgorithmTester.findMax(arr, n);
            long endTime = System.currentTimeMillis();

            System.out.print("data size 2^" + i + " = " + n + "\t");
            System.out.println("Time cost: " + (endTime - startTime) + " ms");
        }
    }
}


\02-Time-Complexity\04-Time-Complexity-Experiments\src\Main2.java

/**
 * Created by liuyubobobo.
 */
public class Main2 {

    public static void main(String[] args) {

        // 数据规模倍乘测试selectionSort
        // O(n^2)
        System.out.println("Test for Selection Sort:");
        for( int i = 10 ; i <= 16 ; i ++ ){

            int n = (int)Math.pow(2,i);
            Integer[] arr = MyUtil.generateRandomArray(n, 0, 100000000);

            long startTime = System.currentTimeMillis();
            MyAlgorithmTester.selectionSort(arr, n);
            long endTime = System.currentTimeMillis();

            System.out.print("data size 2^" + i + " = " + n + "\t");
            System.out.println("Time cost: " + (endTime - startTime) + " ms");
        }
    }
}


\02-Time-Complexity\04-Time-Complexity-Experiments\src\Main3.java

/**
 * Created by liuyubobobo.
 */
public class Main3 {
```

```
265
266         public static void main(String[] args) {
267
268             // 数据规模倍乘测试binarySearch
269             // O(logn)
270             System.out.println("Test for Binary Search:");
271             for(int i = 10 ; i <= 28 ; i ++){
272
273                 int n = (int)Math.pow(2, i);
274                 Integer[] arr = MyUtil.generateOrderedArray(n);
275
276                 long startTime = System.currentTimeMillis();
277                 MyAlgorithmTester.binarySearch(arr, n, 0);
278                 long endTime = System.currentTimeMillis();
279
280                 System.out.print("data size 2^" + i + " = " + n + "\t");
281                 System.out.println("Time cost: " + (endTime - startTime) + " ms");
282             }
283         }
284     }
285
286
287     \02-Time-Complexity\04-Time-Complexity-Experiments\src\Main4.java
288
289     /**
290      * Created by liuyubobobo.
291      */
292     public class Main4 {
293
294         public static void main(String[] args) {
295
296             // 数据规模倍乘测试mergeSort
297             // O(nlogn)
298             System.out.println("Test for Merge Sort:");
299             for( int i = 10 ; i <= 26 ; i ++ ){
300
301                 int n = (int)Math.pow(2,i);
302                 Integer[] arr = MyUtil.generateRandomArray(n, 0, 1<<30);
303
304                 long startTime = System.currentTimeMillis();
305                 MyAlgorithmTester.mergeSort(arr, n);
306                 long endTime = System.currentTimeMillis();
307
308                 System.out.print("data size 2^" + i + " = " + n + "\t");
309                 System.out.println("Time cost: " + (endTime - startTime) + " s");
310             }
311         }
312     }
313
314
315     \02-Time-Complexity\04-Time-Complexity-Experiments\src\MyAlgorithmTester.java
316
317     /**
318      * Created by liuyubobobo.
319      */
320     public class MyAlgorithmTester {
321
322         private MyAlgorithmTester(){}
323
324         // O(logN)
325         public static int binarySearch(Comparable arr[], int n, Comparable target){
326
327             int l = 0, r = n - 1;
328             while(l <= r){
329
330                 int mid = l + (r - l) / 2;
```

```
            if(arr[mid].compareTo(target) == 0) return mid;
            if(arr[mid].compareTo(target) > 0 ) r = mid - 1;
            else l = mid + 1;
        }

        return -1;
    }

    // O(N)
    public static Integer findMax(Integer[] arr, int n){

        assert n > 0;

        Integer res = arr[0];
        for(int i = 1 ; i < n ; i ++)
            if(arr[i]> res)
                res = arr[i];
        return res;
    }

    // O(NlogN)
    public static void mergeSort(Comparable[] arr, int n ){

        Comparable[] aux = new Comparable[n];
        for(int i = 0 ; i < n ; i ++)
            aux[i] = arr[i];

        for(int sz = 1; sz < n ; sz += sz)
            for(int i = 0 ; i < n ; i += sz+sz)
                merge(arr, i, i + sz - 1, Math.min(i + sz + sz - 1, n - 1), aux);

        return;
    }

    private static void merge(Comparable[] arr, int l, int mid, int r, Comparable[] aux){

        for(int i = l ; i <= r ; i ++)
            aux[i] = arr[i];

        int i = l, j = mid + 1;
        for( int k = l ; k <= r; k ++ ){

            if(i > mid)   { arr[k] = aux[j]; j ++;}
            else if(j > r){ arr[k] = aux[i]; i ++;}
            else if(aux[i].compareTo(aux[j]) < 0){ arr[k] = aux[i]; i ++;}
            else          { arr[k] = aux[j]; j ++;}
        }
    }

    // O(N^2)
    public static void selectionSort(Comparable[] arr, int n ){

        for(int i = 0 ; i < n ; i ++){
            int minIndex = i;
            for( int j = i + 1 ; j < n ; j ++ )
                if(arr[j].compareTo(arr[minIndex]) < 0)
                    minIndex = j;

            swap(arr, i, minIndex);
        }

        return;
    }

    private static void swap(Object[] arr, int i, int j){

```

```
397             if(i < 0 || i >= arr.length)
398                 throw new IllegalArgumentException("i is out of bound");
399 
400             if(j < 0 || j >= arr.length)
401                 throw new IllegalArgumentException("j is out of bound");
402 
403             Object t = arr[i];
404             arr[i] = arr[j];
405             arr[j] = t;
406         }
407     }
408 
409 
410     \02-Time-Complexity\04-Time-Complexity-Experiments\src\MyUtil.java
411 
412     /**
413      * Created by liuyubobobo.
414      */
415     public class MyUtil {
416 
417         private MyUtil(){}
418 
419         public static Integer[] generateRandomArray(int n, int rangeL, int rangeR) {
420 
421             assert n > 0 && rangeL <= rangeR;
422 
423             Integer[] arr = new Integer[n];
424             for (int i = 0; i < n; i++)
425                 arr[i] = (int)(Math.random() * (rangeR - rangeL + 1)) + rangeL;
426             return arr;
427         }
428 
429         public static Integer[] generateOrderedArray(int n) {
430 
431             assert n > 0;
432 
433             Integer[] arr = new Integer[n];
434 
435             for (int i = 0; i < n; i++)
436                 arr[i] = i;
437             return arr;
438         }
439     }
440 
441 
442     \02-Time-Complexity\05-Recursion-Time-Complexity\src\Main.java
443 
444     public class Main {
445 
446         // binarySearch
447         private static int binarySearch(Comparable[] arr, int l, int r, int target){
448 
449             if(l > r)
450                 return -1;
451 
452             int mid = l + (r - l) / 2;
453             if(arr[mid].compareTo(target) == 0)
454                 return mid;
455             else if(arr[mid].compareTo(target) > 0)
456                 return binarySearch(arr, l, mid - 1, target);
457             else
458                 return binarySearch(arr, mid + 1, r, target);
459 
460         }
461 
462         // sum
```

```
463         private static int sum(int n){
464 
465             assert n >= 0;
466 
467             if(n == 0)
468                 return 0;
469             return n + sum(n - 1);
470         }
471 
472         // pow2
473         private static double pow(double x, int n){
474 
475             assert n >= 0;
476 
477             if(n == 0)
478                 return 1.0;
479 
480             double t = pow(x, n / 2);
481             if(n % 2 == 1)
482                 return x * t * t;
483 
484             return t * t;
485         }
486 
487         public static void main(String[] args) {
488 
489             System.out.println(sum(100));
490             System.out.println(pow(2, 10));
491         }
492     }
493 
494 
495     \02-Time-Complexity\05-Recursion-Time-Complexity\src\Main2.java
496 
497     /**
498      * Created by liuyubobobo.
499      */
500     public class Main2 {
501 
502         // f
503         private static int f(int n){
504 
505             assert( n >= 0 );
506 
507             if(n == 0)
508                 return 1;
509 
510             return f(n - 1) + f(n - 1);
511         }
512 
513         /*
514         // mergeSort
515         private static void mergeSort(Comparable[] arr, int l, int r){
516 
517             if(l >= r)
518                 return;
519 
520             int mid = (l+r)/2;
521             mergeSort(arr, l, mid);
522             mergeSort(arr, mid + 1, r);
523             merge(arr, l, mid, r);
524         }
525         */
526 
527         public static void main(String[] args) {
528 
```

```
            System.out.println(f(10));
        }
    }


    \02-Time-Complexity\06-Amortized-Time\src\MyVector.java

    import java.util.Arrays;

    /**
     * Created by liuyubobobo.
     */
    public class MyVector<Item> {

        private Item[] data;
        private int size;       // 存储数组中的元素个数
        private int capacity;   // 存储数组中可以容纳的最大的元素个数

        public MyVector(){
            data = (Item[])new Object[100];
            size = 0;
            capacity = 100;
        }

        // 平均复杂度为 O(1)
        public void push_back(Item e){

            if(size == capacity)
                resize(2 * capacity);

            data[size++] = e;
        }

        // 平均复杂度为 O(1)
        public Item pop_back(){

            if(size <= 0)
                throw new IllegalArgumentException("can not pop back for empty vector.");

            size --;
            return data[size];
        }

        // 复杂度为 O(n)
        private void resize(int newCapacity){

            assert newCapacity >= size;
            Item[] newData = (Item[])new Object[newCapacity];
            for(int i = 0 ; i < size ; i ++)
                newData[i] = data[i];

            data = newData;
            capacity = newCapacity;
        }

        // 注意：Java语言由于JVM内部机制的因素，测量的性能时间有可能是跳跃不稳定的。
        public static void main(String[] args) {

            for( int i = 10 ; i <= 26 ; i ++ ){

                int n = (int)Math.pow(2,i);

                long startTime = System.currentTimeMillis();
                MyVector<Integer> vec = new MyVector<Integer>();
                for(int num = 0 ; num < n ; num ++){
                    vec.push_back(num);
```

```
            }
            long endTime = System.currentTimeMillis();

            System.out.print(n + " operations: \t");
            System.out.println((endTime - startTime) + " ms");
        }
    }
}


\02-Time-Complexity\07-Amortized-Time-2\src\MyVector.java

import java.util.Arrays;

/**
 * Created by liuyubobobo.
 */
public class MyVector<Item> {

    private Item[] data;
    private int size;       // 存储数组中的元素个数
    private int capacity;   // 存储数组中可以容纳的最大的元素个数

    public MyVector(){
        data = (Item[])new Object[100];
        size = 0;
        capacity = 100;
    }

    // 平均复杂度为 O(1)
    public void push_back(Item e){

        if(size == capacity)
            resize(2 * capacity);

        data[size++] = e;
    }

    // 平均复杂度为 O(1)
    public Item pop_back(){

        if(size <= 0)
            throw new IllegalArgumentException("can not pop back for empty vector.");

        Item ret = data[size-1];
        size --;

        // 在size达到静态数组最大容量的1/4时才进行resize
        // resize的容量是当前最大容量的1/2
        // 防止复杂度的震荡
        if(size == capacity / 4)
            resize(capacity / 2);

        return ret;
    }

    // 复杂度为 O(n)
    private void resize(int newCapacity){

        assert newCapacity >= size;
        Item[] newData = (Item[])new Object[newCapacity];
        for(int i = 0 ; i < size ; i ++)
            newData[i] = data[i];

        data = newData;
        capacity = newCapacity;
```

```
661         }
662
663         // 注意：Java语言由于JVM内部机制的因素，测量的性能时间有可能是跳跃不稳定的。
664         public static void main(String[] args) {
665
666             for( int i = 10 ; i <= 26 ; i ++ ){
667
668                 int n = (int)Math.pow(2,i);
669
670                 long startTime = System.currentTimeMillis();
671                 MyVector<Integer> vec = new MyVector<Integer>();
672                 for(int num = 0 ; num < n ; num ++){
673                     vec.push_back(num);
674                 }
675                 for(int num = 0 ; num < n ; num ++){
676                     vec.pop_back();
677                 }
678                 long endTime = System.currentTimeMillis();
679
680                 System.out.print(2 * n + " operations: \t");
681                 System.out.println((endTime - startTime) + " ms");
682             }
683         }
684     }
685
686
687     \03-Using-Array\01-Binary-Search\src\BinarySearch.java
688
689     /**
690      * Created by liuyubobobo.
691      */
692     public class BinarySearch {
693
694         private BinarySearch(){}
695
696         public static int binarySearch(Comparable[] arr, int n, Comparable target){
697
698             int l = 0, r = n - 1; // 在[l...r]的范围里寻找target
699             while(l <= r){    // 当 l == r时,区间[l...r]依然是有效的
700                 int mid = l + (r - l) / 2;
701                 if(arr[mid].compareTo(target) == 0) return mid;
702                 if(target.compareTo(arr[mid]) > 0)
703                     l = mid + 1;  // target在[mid+1...r]中; [l...mid]一定没有target
704                 else    // target < arr[mid]
705                     r = mid - 1;  // target在[l...mid-1]中; [mid...r]一定没有target
706             }
707
708             return -1;
709         }
710
711         public static void main(String[] args) {
712
713             int n = (int)Math.pow(10, 7);
714             Integer data[] = Util.generateOrderedArray(n);
715
716             long startTime = System.currentTimeMillis();
717             for(int i = 0 ; i < n ; i ++)
718                 if(i != binarySearch(data, n, i))
719                     throw new IllegalStateException("find i failed!");
720             long endTime = System.currentTimeMillis();
721
722             System.out.println("Binary Search test complete.");
723             System.out.println("Time cost: " + (endTime - startTime) + " ms");
724         }
725     }
726
```

```
\03-Using-Array\01-Binary-Search\src\Util.java

/**
 * Created by liuyubobobo.
 */
public class Util {

    private Util(){}

    public static Integer[] generateRandomArray(int n, int rangeL, int rangeR) {

        assert n > 0 && rangeL <= rangeR;

        Integer[] arr = new Integer[n];
        for (int i = 0; i < n; i++)
            arr[i] = (int)(Math.random() * (rangeR - rangeL + 1)) + rangeL;
        return arr;
    }

    public static Integer[] generateOrderedArray(int n) {

        assert n > 0;

        Integer[] arr = new Integer[n];

        for (int i = 0; i < n; i++)
            arr[i] = i;
        return arr;
    }
}


\03-Using-Array\02-Binary-Search-II\src\BinarySearch.java

/**
 * Created by liuyubobobo.
 */
public class BinarySearch {

    private BinarySearch(){}

    public static int binarySearch(Comparable[] arr, int n, Comparable target){

        int l = 0, r = n; // 在[l...r)的范围里寻找target
        while(l < r){    // 当 l == r 时, 区间[l...r)是一个无效区间
            int mid = l + (r - l) / 2;
            if(arr[mid].compareTo(target) == 0) return mid;
            if(target.compareTo(arr[mid]) > 0)
                l = mid + 1;  // target在[mid+1...r)中; [l...mid]一定没有target
            else    // target < arr[mid]
                r = mid;  // target在[l...mid)中; [mid...r)一定没有target
        }

        return -1;
    }

    public static void main(String[] args) {

        int n = (int)Math.pow(10, 7);
        Integer data[] = Util.generateOrderedArray(n);

        long startTime = System.currentTimeMillis();
        for(int i = 0 ; i < n ; i ++)
            if(i != binarySearch(data, n, i))
                throw new IllegalStateException("find i failed!");
```

```
793             long endTime = System.currentTimeMillis();
794
795             System.out.println("Binary Search 2 test complete.");
796             System.out.println("Time cost: " + (endTime - startTime) + " ms");
797         }
798     }
799
800
801     \03-Using-Array\02-Binary-Search-II\src\Util.java
802
803     /**
804      * Created by liuyubobobo.
805      */
806     public class Util {
807
808         private Util(){}
809
810         public static Integer[] generateRandomArray(int n, int rangeL, int rangeR) {
811
812             assert n > 0 && rangeL <= rangeR;
813
814             Integer[] arr = new Integer[n];
815             for (int i = 0; i < n; i++)
816                 arr[i] = (int)(Math.random() * (rangeR - rangeL + 1)) + rangeL;
817             return arr;
818         }
819
820         public static Integer[] generateOrderedArray(int n) {
821
822             assert n > 0;
823
824             Integer[] arr = new Integer[n];
825
826             for (int i = 0; i < n; i++)
827                 arr[i] = i;
828             return arr;
829         }
830     }
831
832
833     \03-Using-Array\03-Move-Zeroes\src\Solution.java
834
835     import java.util.*;
836
837     // 283. Move Zeroes
838     // https://leetcode.com/problems/move-zeroes/description/
839     // 时间复杂度: O(n)
840     // 空间复杂度: O(n)
841     class Solution {
842         public void moveZeroes(int[] nums) {
843
844             ArrayList<Integer> nonZeroElements = new ArrayList<Integer>();
845
846             // 将vec中所有非0元素放入nonZeroElements中
847             for(int i = 0 ; i < nums.length ; i ++)
848                 if(nums[i] != 0)
849                     nonZeroElements.add(nums[i]);
850
851             // 将nonZeroElements中的所有元素依次放入到nums开始的位置
852             for(int i = 0 ; i < nonZeroElements.size() ; i ++)
853                 nums[i] = nonZeroElements.get(i);
854
855             // 将nums剩余的位置放置为0
856             for(int i = nonZeroElements.size() ; i < nums.length ; i ++)
857                 nums[i] = 0;
858         }
```

```
859
860         public static void main(String args[]){
861
862             int[] arr = {0, 1, 0, 3, 12};
863
864             (new Solution()).moveZeroes(arr);
865
866             for(int i = 0 ; i < arr.length ; i ++)
867                 System.out.print(arr[i] + " ");
868             System.out.println();
869         }
870     }
871
872     \03-Using-Array\04-Move-Zeroes-II\src\Solution1.java
873
874     import java.util.*;
875
876     // 283. Move Zeroes
877     // https://leetcode.com/problems/move-zeroes/description/
878     // 时间复杂度: O(n)
879     // 空间复杂度: O(n)
880     class Solution1 {
881
882         public void moveZeroes(int[] nums) {
883
884             ArrayList<Integer> nonZeroElements = new ArrayList<Integer>();
885
886             // 将vec中所有非0元素放入nonZeroElements中
887             for (int i = 0; i < nums.length; i++)
888                 if (nums[i] != 0)
889                     nonZeroElements.add(nums[i]);
890
891             // 将nonZeroElements中的所有元素依次放入到nums开始的位置
892             for (int i = 0; i < nonZeroElements.size(); i++)
893                 nums[i] = nonZeroElements.get(i);
894
895             // 将nums剩余的位置放置为0
896             for (int i = nonZeroElements.size(); i < nums.length; i++)
897                 nums[i] = 0;
898         }
899
900
901         public static void main(String args[]){
902
903             int[] arr = {0, 1, 0, 3, 12};
904
905             (new Solution1()).moveZeroes(arr);
906
907             for(int i = 0 ; i < arr.length ; i ++)
908                 System.out.print(arr[i] + " ");
909             System.out.println();
910         }
911     }
912
913     \03-Using-Array\04-Move-Zeroes-II\src\Solution2.java
914
915     // 283. Move Zeroes
916     // https://leetcode.com/problems/move-zeroes/description/
917     //
918     // 原地(in place)解决该问题
919     // 时间复杂度: O(n)
920     // 空间复杂度: O(1)
921     class Solution2 {
922         public void moveZeroes(int[] nums) {
923
924             int k = 0; // nums中, [0...k)的元素均为非0元素
```

```
925
926             // 遍历到第i个元素后,保证[0...i]中所有非0元素
927             // 都按照顺序排列在[0...k)中
928             for(int i = 0 ; i < nums.length ; i ++)
929                 if( nums[i] != 0 )
930                     nums[k++] = nums[i];
931
932             // 将nums剩余的位置放置为0
933             for(int i = k ; i < nums.length ; i ++)
934                 nums[i] = 0;
935         }
936
937         public static void main(String args[]){
938
939             int[] arr = {0, 1, 0, 3, 12};
940
941             (new Solution2()).moveZeroes(arr);
942
943             for(int i = 0 ; i < arr.length ; i ++)
944                 System.out.print(arr[i] + " ");
945             System.out.println();
946         }
947     }
948
949     \03-Using-Array\04-Move-Zeroes-II\src\Solution3.java
950
951     // 283. Move Zeroes
952     // https://leetcode.com/problems/move-zeroes/description/
953     //
954     // 原地(in place)解决该问题
955     // 时间复杂度: O(n)
956     // 空间复杂度: O(1)
957     class Solution3 {
958         public void moveZeroes(int[] nums) {
959
960             int k = 0; // nums中, [0...k)的元素均为非0元素
961
962             // 遍历到第i个元素后,保证[0...i]中所有非0元素
963             // 都按照顺序排列在[0...k)中
964             // 同时, [k...i] 为 0
965             for(int i = 0 ; i < nums.length ; i ++)
966                 if(nums[i] != 0)
967                     swap(nums, k++, i);
968         }
969
970         private void swap(int[] nums, int i, int j){
971             int t = nums[i];
972             nums[i] = nums[j];
973             nums[j] = t;
974         }
975
976         public static void main(String args[]){
977
978             int[] arr = {0, 1, 0, 3, 12};
979
980             (new Solution3()).moveZeroes(arr);
981
982             for(int i = 0 ; i < arr.length ; i ++)
983                 System.out.print(arr[i] + " ");
984             System.out.println();
985         }
986     }
987
988     \03-Using-Array\04-Move-Zeroes-II\src\Solution4.java
989
990     // 283. Move Zeroes
```

```java
// https://leetcode.com/problems/move-zeroes/description/
//
// 原地(in place)解决该问题
// 时间复杂度: O(n)
// 空间复杂度: O(1)
class Solution4 {

    public void moveZeroes(int[] nums) {

        int k = 0; // nums中, [0...k)的元素均为非0元素

        // 遍历到第i个元素后,保证[0...i]中所有非0元素
        // 都按照顺序排列在[0...k)中
        // 同时, [k...i] 为 0
        for(int i = 0 ; i < nums.length ; i ++)
            if(nums[i] != 0)
                if(k != i)
                    swap(nums, k++, i);
                else
                    k ++;
    }

    private void swap(int[] nums, int i, int j){
        int t = nums[i];
        nums[i] = nums[j];
        nums[j] = t;
    }

    public static void main(String args[]){

        int[] arr = {0, 1, 0, 3, 12};

        (new Solution4()).moveZeroes(arr);

        for(int i = 0 ; i < arr.length ; i ++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
}

\03-Using-Array\05-Sort-Colors\src\Solution1.java

// 75. Sort Colors
// https://leetcode.com/problems/sort-colors/description/
//
// 计数排序的思路
// 对整个数组遍历了两遍
// 时间复杂度: O(n)
// 空间复杂度: O(k), k为元素的取值范围
public class Solution1 {

    public void sortColors(int[] nums) {

        int[] count = {0, 0, 0};    // 存放0, 1, 2三个元素的频率
        for(int i = 0 ; i < nums.length ; i ++){
            assert nums[i] >= 0 && nums[i] <= 2;
            count[nums[i]] ++;
        }

        int index = 0;
        for(int i = 0 ; i < count[0] ; i ++)
            nums[index++] = 0;
        for(int i = 0 ; i < count[1] ; i ++)
            nums[index++] = 1;
        for(int i = 0 ; i < count[2] ; i ++)
            nums[index++] = 2;
```

```
1057
1058            // 小练习: 自学编写计数排序算法
1059        }
1060
1061        public static void printArr(int[] nums){
1062            for(int num: nums)
1063                System.out.print(num + " ");
1064            System.out.println();
1065        }
1066
1067        public static void main(String[] args) {
1068
1069            int[] nums = {2, 2, 2, 1, 1, 0};
1070            (new Solution1()).sortColors(nums);
1071            printArr(nums);
1072        }
1073    }
1074
1075
1076    \03-Using-Array\05-Sort-Colors\src\Solution2.java
1077
1078    // 75. Sort Colors
1079    // https://leetcode.com/problems/sort-colors/description/
1080    //
1081    // 三路快速排序的思想
1082    // 对整个数组只遍历了一遍
1083    // 时间复杂度: O(n)
1084    // 空间复杂度: O(1)
1085    public class Solution2 {
1086
1087        public void sortColors(int[] nums) {
1088
1089            int zero = -1;          // [0...zero] == 0
1090            int two = nums.length;  // [two...n-1] == 2
1091            for(int i = 0 ; i < two ; ){
1092                if(nums[i] == 1)
1093                    i ++;
1094                else if (nums[i] == 2)
1095                    swap(nums, i, --two);
1096                else{ // nums[i] == 0
1097                    assert nums[i] == 0;
1098                    swap(nums, ++zero, i++);
1099                }
1100            }
1101        }
1102
1103        private void swap(int[] nums, int i, int j){
1104            int t = nums[i];
1105            nums[i]= nums[j];
1106            nums[j] = t;
1107        }
1108
1109        public static void printArr(int[] nums){
1110            for(int num: nums)
1111                System.out.print(num + " ");
1112            System.out.println();
1113        }
1114
1115        public static void main(String[] args) {
1116
1117            int[] nums = {2, 2, 2, 1, 1, 0};
1118            (new Solution2()).sortColors(nums);
1119            printArr(nums);
1120        }
1121    }
1122
```

```
\03-Using-Array\06-Two-Sum-II\src\Solution1.java

// 167. Two Sum II - Input array is sorted
// https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/description/
//
// 暴力枚举法
// 时间复杂度: O(n^2)
// 空间复杂度: O(1)
public class Solution1 {

    public int[] twoSum(int[] numbers, int target) {

        if(numbers.length < 2 /*|| !isSorted(numbers)*/)
            throw new IllegalArgumentException("Illegal argument numbers");

        for(int i = 0 ; i < numbers.length ; i ++)
            for(int j = i+1 ; j < numbers.length ; j ++)
                if(numbers[i] + numbers[j] == target){
                    int[] res = {i+1, j+1};
                    return res;
                }

        throw new IllegalStateException("The input has no solution");
    }

    private boolean isSorted(int[] numbers){
        for(int i = 1 ; i < numbers.length ; i ++)
            if(numbers[i] < numbers[i-1])
                return false;
        return true;
    }

    private static void printArr(int[] nums){
        for(int num: nums)
            System.out.print(num + " ");
        System.out.println();
    }

    public static void main(String[] args) {

        int[] nums = {2, 7, 11, 15};
        int target = 9;
        printArr((new Solution1()).twoSum(nums, target));
    }
}


\03-Using-Array\06-Two-Sum-II\src\Solution2.java

// 167. Two Sum II - Input array is sorted
// https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/description/
//
// 二分搜索法
// 时间复杂度: O(nlogn)
// 空间复杂度: O(1)
public class Solution2 {

    public int[] twoSum(int[] numbers, int target) {

        if(numbers.length < 2 /*|| !isSorted(numbers)*/)
            throw new IllegalArgumentException("Illegal argument numbers");

        for(int i = 0 ; i < numbers.length - 1 ; i ++){
            int j = binarySearch(numbers, i+1, numbers.length-1, target - numbers[i]);
            if(j != -1){
```

```
                    int[] res = {i+1, j+1};
                    return res;
                }
            }

            throw new IllegalStateException("The input has no solution");
        }

        private int binarySearch(int[] nums, int l, int r, int target){

            if(l < 0 || l > nums.length)
                throw new IllegalArgumentException("l is out of bound");

            if(r < 0 || r > nums.length)
                throw new IllegalArgumentException("r is out of bound");

            while(l <= r){
                int mid = l + (r - l)/2;
                if(nums[mid] == target)
                    return mid;
                if(target > nums[mid])
                    l = mid + 1;
                else
                    r = mid - 1;
            }

            return -1;
        }

        private boolean isSorted(int[] numbers){
            for(int i = 1 ; i < numbers.length ; i ++)
                if(numbers[i] < numbers[i-1])
                    return false;
            return true;
        }

        private static void printArr(int[] nums){
            for(int num: nums)
                System.out.print(num + " ");
            System.out.println();
        }

        public static void main(String[] args) {

            int[] nums = {2, 7, 11, 15};
            int target = 9;
            printArr((new Solution2()).twoSum(nums, target));
        }
    }


    \03-Using-Array\06-Two-Sum-II\src\Solution3.java

    // 167. Two Sum II - Input array is sorted
    // https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/description/
    //
    // 对撞指针
    // 时间复杂度: O(n)
    // 空间复杂度: O(1)
    public class Solution3 {

        public int[] twoSum(int[] numbers, int target) {

            if(numbers.length < 2 /*|| !isSorted(numbers)*/)
                throw new IllegalArgumentException("Illegal argument numbers");

```

```
1255            int l = 0, r = numbers.length - 1;
1256            while(l < r){
1257
1258                if(numbers[l] + numbers[r] == target){
1259                    int[] res = {l+1, r+1};
1260                    return res;
1261                }
1262                else if(numbers[l] + numbers[r] < target)
1263                    l ++;
1264                else // numbers[l] + numbers[r] > target
1265                    r --;
1266            }
1267
1268            throw new IllegalStateException("The input has no solution");
1269        }
1270
1271        private boolean isSorted(int[] numbers){
1272            for(int i = 1 ; i < numbers.length ; i ++)
1273                if(numbers[i] < numbers[i-1])
1274                    return false;
1275            return true;
1276        }
1277
1278        private static void printArr(int[] nums){
1279            for(int num: nums)
1280                System.out.print(num + " ");
1281            System.out.println();
1282        }
1283
1284        public static void main(String[] args) {
1285
1286            int[] nums = {2, 7, 11, 15};
1287            int target = 9;
1288            printArr((new Solution3()).twoSum(nums, target));
1289        }
1290    }
1291
1292
1293    \03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution1.java
1294
1295    // 209. Minimum Size Subarray Sum
1296    // https://leetcode.com/problems/minimum-size-subarray-sum/description/
1297    //
1298    // 暴力解法
1299    // 该方法在 Leetcode 中会超时!
1300    // 时间复杂度: O(n^3)
1301    // 空间复杂度: O(1)
1302    public class Solution1 {
1303
1304        public int minSubArrayLen(int s, int[] nums) {
1305
1306            if(s <= 0 || nums == null)
1307                throw new IllegalArgumentException("Illigal Arguments");
1308
1309            int res = nums.length + 1;
1310            for(int l = 0 ; l < nums.length ; l ++)
1311                for(int r = l ; r < nums.length ; r ++){
1312                    int sum = 0;
1313                    for(int i = l ; i <= r ; i ++)
1314                        sum += nums[i];
1315                    if(sum >= s)
1316                        res = Math.min(res, r - l + 1);
1317                }
1318
1319            if(res == nums.length + 1)
1320                return 0;
```

```
1321
1322            return res;
1323        }
1324
1325        public static void main(String[] args) {
1326
1327            int[] nums = {2, 3, 1, 2, 4, 3};
1328            int s = 7;
1329            System.out.println((new Solution1()).minSubArrayLen(s, nums));
1330        }
1331    }
1332
1333
1334    \03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution2.java
1335
1336    // 209. Minimum Size Subarray Sum
1337    // https://leetcode.com/problems/minimum-size-subarray-sum/description/
1338    //
1339    // 优化暴力解
1340    // 时间复杂度: O(n^2)
1341    // 空间复杂度: O(n)
1342    public class Solution2 {
1343
1344        public int minSubArrayLen(int s, int[] nums) {
1345
1346            if(s <= 0 || nums == null)
1347                throw new IllegalArgumentException("Illigal Arguments");
1348
1349            // sums[i]存放nums[0...i-1]的和
1350            int[] sums = new int[nums.length + 1];
1351            sums[0] = 0;
1352            for(int i = 1 ; i <= nums.length ; i ++)
1353                sums[i] = sums[i-1] + nums[i-1];
1354
1355            int res = nums.length + 1;
1356            for(int l = 0 ; l < nums.length ; l ++)
1357                for(int r = l ; r < nums.length ; r ++){
1358                    // 使用sums[r+1] - sums[l] 快速获得nums[l...r]的和
1359                    if(sums[r+1] - sums[l] >= s)
1360                        res = Math.min(res, r - l + 1);
1361                }
1362
1363            if(res == nums.length + 1)
1364                return 0;
1365
1366            return res;
1367        }
1368
1369        public static void main(String[] args) {
1370
1371            int[] nums = {2, 3, 1, 2, 4, 3};
1372            int s = 7;
1373            System.out.println((new Solution2()).minSubArrayLen(s, nums));
1374        }
1375    }
1376
1377
1378    \03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution3.java
1379
1380    // 209. Minimum Size Subarray Sum
1381    // https://leetcode.com/problems/minimum-size-subarray-sum/description/
1382    //
1383    // 滑动窗口的思路
1384    // 时间复杂度: O(n)
1385    // 空间复杂度: O(1)
1386    public class Solution3 {
```

```
1387
1388        public int minSubArrayLen(int s, int[] nums) {
1389
1390            if(s <= 0 || nums == null)
1391                throw new IllegalArgumentException("Illigal Arguments");
1392
1393            int l = 0 , r = -1; // nums[l...r]为我们的滑动窗口
1394            int sum = 0;
1395            int res = nums.length + 1;
1396
1397            while(l < nums.length){   // 窗口的左边界在数组范围内,则循环继续
1398
1399                if(r + 1 < nums.length && sum < s)
1400                    sum += nums[++r];
1401                else // r已经到头 或者 sum >= s
1402                    sum -= nums[l++];
1403
1404                if(sum >= s)
1405                    res = Math.min(res, r - l + 1);
1406            }
1407
1408            if(res == nums.length + 1)
1409                return 0;
1410            return res;
1411        }
1412
1413        public static void main(String[] args) {
1414
1415            int[] nums = {2, 3, 1, 2, 4, 3};
1416            int s = 7;
1417            System.out.println((new Solution3()).minSubArrayLen(s, nums));
1418        }
1419    }
1420
1421
1422    \03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution4.java
1423
1424    // 209. Minimum Size Subarray Sum
1425    // https://leetcode.com/problems/minimum-size-subarray-sum/description/
1426    //
1427    // 另外一个滑动窗口的实现, 仅供参考
1428    // 时间复杂度: O(n)
1429    // 空间复杂度: O(1)
1430    public class Solution4 {
1431
1432        public int minSubArrayLen(int s, int[] nums) {
1433
1434            if(s <= 0 || nums == null)
1435                throw new IllegalArgumentException("Illigal Arguments");
1436
1437            int l = 0 , r = -1; // [l...r]为我们的窗口
1438            int sum = 0;
1439            int res = nums.length + 1;
1440
1441            while(r + 1 < nums.length){   // 窗口的右边界无法继续扩展了, 则循环继续
1442
1443                while(r + 1 < nums.length && sum < s)
1444                    sum += nums[++r];
1445
1446                if(sum >= s)
1447                    res = Math.min(res, r - l + 1);
1448
1449                while(l < nums.length && sum >= s){
1450                    sum -= nums[l++];
1451                    if(sum >= s)
1452                        res = Math.min(res, r - l + 1);
```

```
            }
        }

        if(res == nums.length + 1)
            return 0;
        return res;
    }

    public static void main(String[] args) {

        int[] nums = {2, 3, 1, 2, 4, 3};
        int s = 7;
        System.out.println((new Solution4()).minSubArrayLen(s, nums));
    }
}


\03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution5.java

// 209. Minimum Size Subarray Sum
// https://leetcode.com/problems/minimum-size-subarray-sum/description/
//
// 二分搜索
// 扩展 Solution2 的方法。对于每一个l, 可以使用二分搜索法搜索r
//
// 时间复杂度: O(nlogn)
// 空间复杂度: O(n)
public class Solution5 {

    public int minSubArrayLen(int s, int[] nums) {

        if(s <= 0 || nums == null)
            throw new IllegalArgumentException("Illigal Arguments");

        // sums[i]存放nums[0...i-1]的和
        int[] sums = new int[nums.length + 1];
        sums[0] = 0;
        for(int i = 1 ; i <= nums.length ; i ++)
            sums[i] = sums[i-1] + nums[i-1];

        int res = nums.length + 1;
        for(int l = 0 ; l < nums.length - 1 ; l ++){
            // Java类库中没有内置的lowerBound方法,
            // 我们需要自己实现一个基于二分搜索的lowerBound:)
            int r = lowerBound(sums, sums[l] + s);
            if(r != sums.length){
                res = Math.min(res, r - l);
            }
        }

        if(res == nums.length + 1)
            return 0;
        return res;
    }

    // 在有序数组nums中寻找大于等于target的最小值
    // 如果没有（nums数组中所有值都小于target），则返回nums.length
    private int lowerBound(int[] nums, int target){

        if(nums == null /*|| !isSorted(nums)*/)
            throw new IllegalArgumentException("Illegal argument nums in lowerBound.");

        int l = 0, r = nums.length; // 在nums[l...r)的范围里寻找解
        while(l != r){
            int mid = l + (r - l) / 2;
            if(nums[mid] >= target)
```

```
                r = mid;
            else
                l = mid + 1;
        }

        return l;
    }

    private boolean isSorted(int[] nums){
        for(int i = 1 ; i < nums.length ; i ++)
            if(nums[i] < nums[i-1])
                return false;
        return true;
    }

    public static void main(String[] args) {

        int[] nums = {2, 3, 1, 2, 4, 3};
        int s = 7;
        System.out.println((new Solution5()).minSubArrayLen(s, nums));
    }
}


\03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Main.java

import java.lang.reflect.Method;
import java.lang.Class;

// 比较这个工程中 Solution1, Solution2, Solution3, Solution4 和 Solution5 的算法运行效率
public class Main {

    public static void testPerformace(String algoClassName, String algoName, String s){

        try{
            Class algoClass = Class.forName(algoClassName);
            Object solution = algoClass.newInstance();

            // 通过排序函数的Class对象获得排序方法
            Method algoMethod = algoClass.getMethod(algoName, String.class);

            long startTime = System.currentTimeMillis();
            // 调用算法
            Object resObj = algoMethod.invoke(solution, s);
            long endTime = System.currentTimeMillis();

            int res = (Integer)resObj;
            System.out.print(algoClassName + " : res = " + res + " ");
            System.out.println("Time = " + (endTime-startTime) + " ms" );
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {

        int n = 10000000;

        StringBuilder s = new StringBuilder(n);
        for(int i = 0 ; i < n ; i ++)
            s.append((char)(Math.random()*95 + 32));

        System.out.println("Test: 10,000,000 length of completely random string:");
        testPerformace("Solution1", "lengthOfLongestSubstring", s.toString());
        testPerformace("Solution2", "lengthOfLongestSubstring", s.toString());
```

```
1585            testPerformace("Solution3", "lengthOfLongestSubstring", s.toString());
1586            testPerformace("Solution4", "lengthOfLongestSubstring", s.toString());
1587            testPerformace("Solution5", "lengthOfLongestSubstring", s.toString());
1588
1589        }
1590    }
1591
1592
1593    \03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Solution1.java
1594
1595    // 3. Longest Substring Without Repeating Characters
1596    // https://leetcode.com/problems/longest-substring-without-repeating-characters/description/
1597    //
1598    // 滑动窗口
1599    // 时间复杂度: O(len(s))
1600    // 空间复杂度: O(len(charset))
1601    class Solution1 {
1602        public int lengthOfLongestSubstring(String s) {
1603
1604            int[] freq = new int[256];
1605
1606            int l = 0, r = -1; //滑动窗口为s[l...r]
1607            int res = 0;
1608
1609            // 整个循环从 l == 0; r == -1 这个空窗口开始
1610            // 到l == s.size(); r == s.size()-1 这个空窗口截止
1611            // 在每次循环里逐渐改变窗口, 维护freq, 并记录当前窗口中是否找到了一个新的最优值
1612            while(l < s.length()){
1613
1614                if(r + 1 < s.length() && freq[s.charAt(r+1)] == 0)
1615                    freq[s.charAt(++r)] ++;
1616                else    //r已经到头 || freq[s[r+1]] == 1
1617                    freq[s.charAt(l++)] --;
1618
1619                res = Math.max(res, r-l+1);
1620            }
1621
1622            return res;
1623        }
1624
1625        public static void main(String[] args) {
1626
1627            System.out.println((new Solution1()).lengthOfLongestSubstring( "abcabcbb" ));
1628            System.out.println((new Solution1()).lengthOfLongestSubstring( "bbbbb" ));
1629            System.out.println((new Solution1()).lengthOfLongestSubstring( "pwwkew" ));
1630            System.out.println((new Solution1()).lengthOfLongestSubstring( "" ));
1631        }
1632    }
1633
1634    \03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Solution2.java
1635
1636    // 3. Longest Substring Without Repeating Characters
1637    // https://leetcode.com/problems/longest-substring-without-repeating-characters/description/
1638    //
1639    // 滑动窗口
1640    // 时间复杂度: O(len(s))
1641    // 空间复杂度: O(len(charset))
1642    public class Solution2 {
1643        public int lengthOfLongestSubstring(String s) {
1644
1645            int[] freq = new int[256];
1646
1647            int l = 0, r = -1; //滑动窗口为s[l...r]
1648            int res = 0;
1649
1650            // 在这里, 循环中止的条件可以是 r + 1 < s.length(), 想想看为什么?
```

```
1651            // 感谢课程QQ群 @千千 指出 :)
1652            while( r + 1 < s.length() ){
1653
1654                if( r + 1 < s.length() && freq[s.charAt(r+1)] == 0 )
1655                    freq[s.charAt(++r)] ++;
1656                else    //freq[s[r+1]] == 1
1657                    freq[s.charAt(l++)] --;
1658
1659                res = Math.max(res, r-l+1);
1660            }
1661
1662            return res;
1663        }
1664
1665        public static void main(String[] args) {
1666            System.out.println((new Solution2()).lengthOfLongestSubstring( "abcabcbb" ));
1667            System.out.println((new Solution2()).lengthOfLongestSubstring( "bbbbb" ));
1668            System.out.println((new Solution2()).lengthOfLongestSubstring( "pwwkew" ));
1669            System.out.println((new Solution2()).lengthOfLongestSubstring( "" ));
1670        }
1671    }
1672
1673
1674    \03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Solution3.java
1675
1676    // 3. Longest Substring Without Repeating Characters
1677    // https://leetcode.com/problems/longest-substring-without-repeating-characters/description/
1678    //
1679    // 滑动窗口的另一个实现，仅做参考
1680    // 时间复杂度: O(len(s))
1681    // 空间复杂度: O(len(charset))
1682    public class Solution3 {
1683        public int lengthOfLongestSubstring(String s) {
1684
1685            int[] freq = new int[256];
1686
1687            int l = 0, r = -1; //滑动窗口为s[l...r]
1688            int res = 0;
1689
1690            while(r + 1 < s.length()){
1691
1692                while(r + 1 < s.length() && freq[s.charAt(r+1)] == 0)
1693                    freq[s.charAt(++r)] ++;
1694
1695                res = Math.max(res, r - l + 1);
1696
1697                if(r + 1 < s.length()){
1698                    freq[s.charAt(++r)] ++;
1699                    assert(freq[s.charAt(r)] == 2);
1700                    while(l <= r && freq[s.charAt(r)] == 2)
1701                        freq[s.charAt(l++)] --;
1702                }
1703            }
1704
1705            return res;
1706        }
1707
1708        public static void main(String[] args) {
1709
1710            System.out.println((new Solution3()).lengthOfLongestSubstring( "abcabcbb" ));
1711            System.out.println((new Solution3()).lengthOfLongestSubstring( "bbbbb" ));
1712            System.out.println((new Solution3()).lengthOfLongestSubstring( "pwwkew" ));
1713            System.out.println((new Solution3()).lengthOfLongestSubstring( "" ));
1714        }
1715    }
1716
```

```

\03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Solution4.java

// 3. Longest Substring Without Repeating Characters
// https://leetcode.com/problems/longest-substring-without-repeating-characters/description/
//
// 课程问答区 @yatkun 提出的方法,
// l每次可以向前跳跃, 而不仅仅是+1
// 但代价是, 为了获得这个跳跃的位置, 每次需要遍历整个窗口的字符串
//
// 时间复杂度: O(len(s)*len(charset))
// 空间复杂度: O(1)
public class Solution4{

    public int lengthOfLongestSubstring(String s) {

        int l = 0, r = 0; //滑动窗口为s[l...r]
        int res = 0;

        while(r < s.length()){

            int index = isDuplicateChar(s, l, r);

            // 如果s[r]之前出现过
            // l可以直接跳到s[r+1]之前出现的位置 + 1的地方
            if(index != -1)
                l = index + 1;

            res = Math.max(res, r-l+1);
            r ++;
        }

        return res;
    }

    // 查看s[l...r-1]之间是否存在s[r]
    // 若存在,返回相应的索引, 否则返回-1
    private int isDuplicateChar(String s, int l, int r){
        for(int i = l ; i < r ; i ++)
            if(s.charAt(i) == s.charAt(r))
                return i;
        return -1;
    }

    public static void main(String[] args) {

        System.out.println((new Solution4()).lengthOfLongestSubstring( "abcabcbb" ));
        System.out.println((new Solution4()).lengthOfLongestSubstring( "bbbbb" ));
        System.out.println((new Solution4()).lengthOfLongestSubstring( "pwwkew" ));
        System.out.println((new Solution4()).lengthOfLongestSubstring( "" ));
    }
}


\03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Solution5.java

// 3. Longest Substring Without Repeating Characters
// https://leetcode.com/problems/longest-substring-without-repeating-characters/description/
//
// 滑动窗口
// 其中使用last[c]保存字符c上一次出现的位置, 用于在右边界发现重复字符时, 快速移动左边界
// 使用这种方法, 时间复杂度依然为O(n), 但是只需要动r指针, 实际上对整个s只遍历了一次
// 相较而言, 之前的方法, 需要移动l和r两个指针, 相对于对s遍历了两次

import java.util.Arrays;

```

```java
// 时间复杂度: O(len(s))
// 空间复杂度: O(len(charset))
public class Solution5 {

    public int lengthOfLongestSubstring(String s) {

        int[] last = new int[256];
        Arrays.fill(last, -1);

        int l = 0, r = -1; //滑动窗口为s[l...r]
        int res = 0;
        while(r + 1 < s.length()){

            r ++;
            if(last[s.charAt(r)] != -1)
                l = Math.max(l, last[s.charAt(r)] + 1);

            res = Math.max(res, r - l + 1);
            last[s.charAt(r)] = r;
        }

        return res;
    }

    public static void main(String[] args) {

        System.out.println((new Solution5()).lengthOfLongestSubstring( "abcabcbb" ));
        System.out.println((new Solution5()).lengthOfLongestSubstring( "bbbbb" ));
        System.out.println((new Solution5()).lengthOfLongestSubstring( "pwwkew" ));
        System.out.println((new Solution5()).lengthOfLongestSubstring( "" ));
    }
}


\04-Using-Hash-Table\01-Intersection-of-Two-Arrays\src\Solution.java

import java.util.TreeSet;

// 349. Intersection of Two Arrays
// https://leetcode.com/problems/intersection-of-two-arrays/description/
// 时间复杂度: O(nlogn)
// 空间复杂度: O(n)
public class Solution {

    public int[] intersection(int[] nums1, int[] nums2) {

        TreeSet<Integer> record = new TreeSet<Integer>();
        for(int num: nums1)
            record.add(num);

        TreeSet<Integer> resultSet = new TreeSet<Integer>();
        for(int num: nums2)
            if(record.contains(num))
                resultSet.add(num);

        int[] res = new int[resultSet.size()];
        int index = 0;
        for(Integer num: resultSet)
            res[index++] = num;

        return res;
    }

    private static void printArr(int[] arr){
        for(int e: arr)
            System.out.print(e + " ");
```

```
1849            System.out.println();
1850        }
1851
1852        public static void main(String[] args) {
1853
1854            int[] nums1 = {1, 2, 2, 1};
1855            int[] nums2 = {2, 2};
1856            int[] res = (new Solution()).intersection(nums1, nums2);
1857            printArr(res);
1858        }
1859    }
1860
1861
1862    \04-Using-Hash-Table\02-Intersection-of-Two-Arrays-II\src\Main.java
1863
1864    /// 让我们来测试使用Java中的TreeMap:)
1865
1866    import java.util.TreeMap;
1867
1868    public class Main {
1869
1870        public static void main(String[] args) {
1871
1872            TreeMap<Integer, Integer> myMap = new TreeMap<Integer, Integer>();
1873            if(myMap.containsKey(42))
1874                System.out.println("Element 42 is in the map");
1875            else
1876                System.out.println("Can not find element 42");
1877
1878            System.out.println(myMap.get(42)); // 输出 null
1879
1880            // Java不存在C++中默认的访问key即添加默认(key, value)的行为
1881            // 以下代码仍然无法找到42
1882            if(myMap.containsKey(42))
1883                System.out.println("Element 42 is in the map");
1884            else
1885                System.out.println("Can not find element 42");
1886
1887            myMap.put(42, 0);
1888            myMap.put(42, myMap.get(42) + 1);
1889            System.out.println(myMap.get(42)); // 输出 1
1890            if(myMap.containsKey(42))
1891                System.out.println("Element 42 is in the map");
1892            else
1893                System.out.println("Can not find element 42");
1894
1895            myMap.put(42, myMap.get(42) - 1);
1896            System.out.println(myMap.get(42)); // 输出 0
1897
1898            // 注意: key对应的值为0, 不代表key不存在
1899            if(myMap.containsKey(42))
1900                System.out.println("Element 42 is in the map");
1901            else
1902                System.out.println("Can not find element 42");
1903
1904            // 注意:  也不可以为key对应的值设置null来删除一个key
1905            myMap.put(42, null);
1906            if(myMap.containsKey(42))
1907                System.out.println("Element 42 is in the map");
1908            else
1909                System.out.println("Can not find element 42");
1910
1911            // 使用remove删除一个key
1912            myMap.remove(42);
1913            if(myMap.containsKey(42))
1914                System.out.println("Element 42 is in the map");
```

```
            else
                System.out.println("Can not find element 42");
        }
    }


    \04-Using-Hash-Table\02-Intersection-of-Two-Arrays-II\src\Solution.java

    import java.util.TreeMap;
    import java.util.ArrayList;

    // 350. Intersection of Two Arrays II
    // https://leetcode.com/problems/intersection-of-two-arrays-ii/description/
    // 时间复杂度: O(nlogn)
    // 空间复杂度: O(n)
    public class Solution {

        public int[] intersect(int[] nums1, int[] nums2) {

            TreeMap<Integer, Integer> record = new TreeMap<Integer, Integer>();
            for(int num: nums1)
                if(!record.containsKey(num))
                    record.put(num, 1);
                else
                    record.put(num, record.get(num) + 1);

            ArrayList<Integer> result = new ArrayList<Integer>();
            for(int num: nums2)
                if(record.containsKey(num) && record.get(num) > 0){
                    result.add(num);
                    record.put(num, record.get(num) - 1);
                }

            int[] ret = new int[result.size()];
            int index = 0;
            for(Integer num: result)
                ret[index++] = num;

            return ret;
        }

        private static void printArr(int[] arr){
            for(int e: arr)
                System.out.print(e + " ");
            System.out.println();
        }

        public static void main(String[] args) {

            int[] nums1 = {1, 2, 2, 1};
            int[] nums2 = {2, 2};
            int[] res = (new Solution()).intersect(nums1, nums2);
            printArr(res);
        }
    }


    \04-Using-Hash-Table\03-More-About-Set-And-Map\src\Solution349.java

    import java.util.HashSet;

    // 349. Intersection of Two Arrays
    // https://leetcode.com/problems/intersection-of-two-arrays/description/
    // 时间复杂度: O(len(nums1)+len(nums2))
    // 空间复杂度: O(len(nums1))
    public class Solution349 {
```

```
1981
1982        public int[] intersection(int[] nums1, int[] nums2) {
1983
1984            HashSet<Integer> record = new HashSet<Integer>();
1985            for(int num: nums1)
1986                record.add(num);
1987
1988            HashSet<Integer> resultSet = new HashSet<Integer>();
1989            for(int num: nums2)
1990                if(record.contains(num))
1991                    resultSet.add(num);
1992
1993            int[] res = new int[resultSet.size()];
1994            int index = 0;
1995            for(Integer num: resultSet)
1996                res[index++] = num;
1997
1998            return res;
1999        }
2000
2001        private static void printArr(int[] arr){
2002            for(int e: arr)
2003                System.out.print(e + " ");
2004            System.out.println();
2005        }
2006
2007        public static void main(String[] args) {
2008
2009            int[] nums1 = {1, 2, 2, 1};
2010            int[] nums2 = {2, 2};
2011            int[] res = (new Solution349()).intersection(nums1, nums2);
2012            printArr(res);
2013        }
2014    }
2015
2016    \04-Using-Hash-Table\03-More-About-Set-And-Map\src\Solution350.java
2017
2018    import java.util.HashMap;
2019    import java.util.ArrayList;
2020
2021    // 350. Intersection of Two Arrays II
2022    // https://leetcode.com/problems/intersection-of-two-arrays-ii/description/
2023    // 时间复杂度: O(len(nums1)+len(nums2))
2024    // 空间复杂度: O(len(nums1))
2025    public class Solution350 {
2026
2027        public int[] intersect(int[] nums1, int[] nums2) {
2028
2029            HashMap<Integer, Integer> record = new HashMap<Integer, Integer>();
2030            for(int num: nums1)
2031                if(!record.containsKey(num))
2032                    record.put(num, 1);
2033                else
2034                    record.put(num, record.get(num) + 1);
2035
2036            ArrayList<Integer> result = new ArrayList<Integer>();
2037            for(int num: nums2)
2038                if(record.containsKey(num) && record.get(num) > 0){
2039                    result.add(num);
2040                    record.put(num, record.get(num) - 1);
2041                }
2042
2043            int[] ret = new int[result.size()];
2044            int index = 0;
2045            for(Integer num: result)
2046                ret[index++] = num;
```

```
2047
2048            return ret;
2049        }
2050
2051        private static void printArr(int[] arr){
2052            for(int e: arr)
2053                System.out.print(e + " ");
2054            System.out.println();
2055        }
2056
2057        public static void main(String[] args) {
2058
2059            int[] nums1 = {1, 2, 2, 1};
2060            int[] nums2 = {2, 2};
2061            int[] res = (new Solution350()).intersect(nums1, nums2);
2062            printArr(res);
2063        }
2064    }
2065
2066    \04-Using-Hash-Table\04-Two-Sum\src\Solution.java
2067
2068    import java.util.HashMap;
2069
2070    // 1. Two Sum
2071    // https://leetcode.com/problems/two-sum/description/
2072    // 时间复杂度: O(n)
2073    // 空间复杂度: O(n)
2074    public class Solution {
2075
2076        public int[] twoSum(int[] nums, int target) {
2077
2078            HashMap<Integer, Integer> record = new HashMap<Integer, Integer>();
2079            for(int i = 0 ; i < nums.length; i ++){
2080
2081                int complement = target - nums[i];
2082                if(record.containsKey(complement)){
2083                    int[] res = {i, record.get(complement)};
2084                    return res;
2085                }
2086
2087                record.put(nums[i], i);
2088            }
2089
2090            throw new IllegalStateException("the input has no solution");
2091        }
2092
2093        private static void printArr(int[] nums){
2094            for(int num: nums)
2095                System.out.print(num + " ");
2096            System.out.println();
2097        }
2098
2099        public static void main(String[] args) {
2100
2101            int[] nums = {0,4,3,0};
2102            int target = 0;
2103            printArr((new Solution()).twoSum(nums, target));
2104        }
2105    }
2106
2107
2108    \04-Using-Hash-Table\04-Two-Sum\src\Solution2.java
2109
2110    import java.util.HashMap;
2111
2112    // 1. Two Sum
```

```java
// https://leetcode.com/problems/two-sum/description/
//
// 感谢课程中的 @Charles_Zhang 提出:
// 由于题目中只要求求出唯一的一个解。因此可以在最初的时候遍历整个数组，将数组中的每个数字的索引放在map中。
// 此时，record中记录的永远是每一个数字最后出现的位置。
// 而对于 target = 2*a的情况，如果nums中有两个或两个以上a,
// 我们在扫描时会先看到第一个a，而从record中拿到的是最后一个a :)
//
// 时间复杂度: O(n)
// 空间复杂度: O(n)
public class Solution2 {

    public int[] twoSum(int[] nums, int target) {

        HashMap<Integer, Integer> record = new HashMap<Integer, Integer>();
        for(int i = 0 ; i < nums.length ; i ++)
            record.put(nums[i], i);

        for(int i = 0 ; i < nums.length; i ++){

            if(record.containsKey(target - nums[i]))
                if(record.get(target - nums[i]) != i){
                    int[] res = {i, record.get(target - nums[i])};
                    return res;
                }

            record.put(nums[i], i);
        }

        throw new IllegalStateException("the input has no solution");
    }

    private static void printArr(int[] nums){
        for(int num: nums)
            System.out.print(num + " ");
        System.out.println();
    }

    public static void main(String[] args) {

        int[] nums = {0,4,3,0};
        int target = 0;
        printArr((new Solution()).twoSum(nums, target));
    }
}


\04-Using-Hash-Table\05-4Sum-II\src\Solution1.java

import java.util.HashMap;

// 454. 4Sum II
// https://leetcode.com/problems/4sum-ii/description/
// 时间复杂度: O(n^2)
// 空间复杂度: O(n^2)
public class Solution1 {

    public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {

        if(A == null || B == null || C == null || D == null)
            throw new IllegalArgumentException("Illegal argument");

        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for(int i = 0 ; i < C.length ; i ++)
            for(int j = 0 ; j < D.length ; j ++){
                int sum = C[i] + D[j];
```

```
                if(map.containsKey(sum))
                    map.put(sum, map.get(sum) + 1);
                else
                    map.put(sum, 1);
            }

        int res = 0;
        for(int i = 0 ; i < A.length ; i ++)
            for(int j = 0 ; j < B.length ; j ++)
                if(map.containsKey(-A[i]-B[j]))
                    res += map.get(-A[i]-B[j]);

        return res;
    }

    public static void main(String[] args) {

        int[] a = {1, 2};
        int[] b = {-2, -1};
        int[] c = {-1, 2};
        int[] d = {0, 2};
        System.out.println((new Solution1()).fourSumCount(a, b, c, d));
    }
}


\04-Using-Hash-Table\05-4Sum-II\src\Solution2.java

import java.util.HashMap;

// 454. 4Sum II
// https://leetcode.com/problems/4sum-ii/description/
// 时间复杂度: O(n^2)
// 空间复杂度: O(n^2)
public class Solution2 {

    public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {

        if(A == null || B == null || C == null || D == null)
            throw new IllegalArgumentException("Illegal argument");

        HashMap<Integer, Integer> mapAB = new HashMap<Integer, Integer>();
        for(int i = 0 ; i < A.length ; i ++)
            for(int j = 0 ; j < B.length ; j ++){
                int sum = A[i] + B[j];
                if(mapAB.containsKey(sum))
                    mapAB.put(sum, mapAB.get(sum) + 1);
                else
                    mapAB.put(sum, 1);
            }

        HashMap<Integer, Integer> mapCD = new HashMap<Integer, Integer>();
        for(int i = 0 ; i < C.length ; i ++)
            for(int j = 0 ; j < D.length ; j ++){
                int sum = C[i] + D[j];
                if(mapCD.containsKey(sum))
                    mapCD.put(sum, mapCD.get(sum) + 1);
                else
                    mapCD.put(sum, 1);
            }

        int res = 0;
        for(Integer sumab: mapAB.keySet()){
            if(mapCD.containsKey(-sumab))
                res += mapAB.get(sumab) * mapCD.get(-sumab);
        }
```

```
2245
2246            return res;
2247        }
2248
2249        public static void main(String[] args) {
2250
2251            int[] a = {1, 2};
2252            int[] b = {-2, -1};
2253            int[] c = {-1, 2};
2254            int[] d = {0, 2};
2255            System.out.println((new Solution2()).fourSumCount(a, b, c, d));
2256        }
2257    }
2258
2259
2260    \04-Using-Hash-Table\06-Number-of-Boomerangs\src\Solution.java
2261
2262    import java.util.HashMap;
2263
2264    // 447. Number of Boomerangs
2265    // https://leetcode.com/problems/number-of-boomerangs/description/
2266    // 时间复杂度: O(n^2)
2267    // 空间复杂度: O(n)
2268    public class Solution {
2269
2270        public int numberOfBoomerangs(int[][] points) {
2271
2272            int res = 0;
2273            for( int i = 0 ; i < points.length ; i ++ ){
2274
2275                // record中存储 点i 到所有其他点的距离出现的频次
2276                HashMap<Integer, Integer> record = new HashMap<Integer, Integer>();
2277                for(int j = 0 ; j < points.length ; j ++)
2278                    if(j != i){
2279                        // 计算距离时不进行开根运算, 以保证精度
2280                        int dis = dis(points[i], points[j]);
2281                        if(record.containsKey(dis))
2282                            record.put(dis, record.get(dis) + 1);
2283                        else
2284                            record.put(dis, 1);
2285                    }
2286
2287                for(Integer dis: record.keySet())
2288                    res += record.get(dis) * (record.get(dis) - 1);
2289            }
2290
2291            return res;
2292        }
2293
2294        private int dis(int[] pa, int pb[]){
2295            return (pa[0] - pb[0]) * (pa[0] - pb[0]) +
2296                   (pa[1] - pb[1]) * (pa[1] - pb[1]);
2297        }
2298
2299        public static void main(String[] args) {
2300
2301            int[][] points = {{0, 0}, {1, 0}, {2, 0}};
2302            System.out.println((new Solution()).numberOfBoomerangs(points));
2303        }
2304    }
2305
2306
2307    \04-Using-Hash-Table\07-Contains-Duplicate-II\src\Solution.java
2308
2309    import java.util.HashSet;
2310
```

```java
// 219. Contains Duplicate II
// https://leetcode.com/problems/contains-duplicate-ii/description/
// 时间复杂度: O(n)
// 空间复杂度: O(k)
public class Solution {

    public boolean containsNearbyDuplicate(int[] nums, int k) {

        if(nums == null || nums.length <= 1)
            return false;

        if(k <= 0)
            return false;

        HashSet<Integer> record = new HashSet<Integer>();
        for(int i = 0 ; i < nums.length; i ++){
            if(record.contains(nums[i]))
                return true;

            record.add(nums[i]);
            if(record.size() == k + 1)
                record.remove(nums[i-k]);
        }

        return false;
    }

    private static void printBool(boolean b){
        System.out.println(b ? "True" : "False");
    }

    public static void main(String[] args) {

        int[] nums = {1, 2, 1};
        int k = 1;
        printBool((new Solution()).containsNearbyDuplicate(nums, k));
    }
}


\04-Using-Hash-Table\08-Contains-Duplicate-III\src\Solution.java

import java.util.TreeSet;

// 220. Contains Duplicate III
// https://leetcode.com/problems/contains-duplicate-iii/description/
// 时间复杂度: O(nlogk)
// 空间复杂度: O(k)
public class Solution {

    public boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t) {

        // 这个问题的测试数据在使用int进行加减运算时会溢出
        // 所以使用long long
        TreeSet<Long> record = new TreeSet<Long>();
        for(int i = 0 ; i < nums.length ; i ++){

            if(record.ceiling((long)nums[i] - (long)t) != null &&
                    record.ceiling((long)nums[i] - (long)t) <= (long)nums[i] + (long)t)
                return true;

            record.add((long)nums[i]);

            if(record.size() == k + 1)
                record.remove((long)nums[i-k]);
        }
```

```
2377
2378            return false;
2379        }
2380
2381        private static void printBool(boolean b){
2382            System.out.println(b ? "True" : "False");
2383        }
2384
2385        public static void main(String[] args) {
2386
2387            int[] nums = {-2147483648, -2147483647};
2388            int k = 3;
2389            int t = 3;
2390            printBool((new Solution()).containsNearbyAlmostDuplicate(nums, k, t));
2391        }
2392    }
2393
2394
2395    \05-About-Linked-List\01-Reverse-Linked-List\src\Solution1.java
2396
2397    // 206. Reverse Linked List
2398    // https://leetcode.com/problems/reverse-linked-list/description/
2399    // 时间复杂度: O(n)
2400    // 空间复杂度: O(1)
2401    public class Solution1 {
2402
2403        // Definition for singly-linked list.
2404        public class ListNode {
2405            int val;
2406            ListNode next;
2407            ListNode(int x) { val = x; }
2408        }
2409
2410        public ListNode reverseList(ListNode head) {
2411
2412            ListNode pre = null;
2413            ListNode cur = head;
2414            while(cur != null){
2415                ListNode next = cur.next;
2416                cur.next = pre;
2417                pre = cur;
2418                cur = next;
2419            }
2420
2421            return pre;
2422        }
2423    }
2424
2425
2426    \05-About-Linked-List\01-Reverse-Linked-List\src\Solution2.java
2427
2428    // 206. Reverse Linked List
2429    // https://leetcode.com/problems/reverse-linked-list/description/
2430    //
2431    // 递归的方式反转链表
2432    // 时间复杂度: O(n)
2433    // 空间复杂度: O(n) - 注意，递归是占用空间的，占用空间的大小和递归深度成正比：）
2434    public class Solution2 {
2435
2436        // Definition for singly-linked list.
2437        public class ListNode {
2438            int val;
2439            ListNode next;
2440            ListNode(int x) { val = x; }
2441        }
2442
```

```
2443        public ListNode reverseList(ListNode head) {
2444
2445            // 递归终止条件
2446            if(head == null|| head.next == null)
2447                return head;
2448
2449            ListNode rhead = reverseList(head.next);
2450
2451            // head->next此刻指向head后面的链表的尾节点
2452            // head->next->next = head把head节点放在了尾部
2453            head.next.next = head;
2454            head.next = null;
2455
2456            return rhead;
2457        }
2458    }
2459
2460
2461    \05-About-Linked-List\02-Test-Your-Linked-List\src\ListNode.java
2462
2463    // Definition for singly-linked list.
2464    // 在Java版本中，我们将LinkedList相关的测试辅助函数写在ListNode里
2465    public class ListNode {
2466
2467        public int val;
2468        public ListNode next = null;
2469
2470        public ListNode(int x) {
2471            val = x;
2472        }
2473
2474        // 根据n个元素的数组arr创建一个链表
2475        // 使用arr为参数，创建另外一个ListNode的构造函数
2476        public ListNode (int[] arr){
2477
2478            if(arr == null || arr.length == 0)
2479                throw new IllegalArgumentException("arr can not be empty");
2480
2481            this.val = arr[0];
2482            ListNode curNode = this;
2483            for(int i = 1 ; i < arr.length ; i ++){
2484                curNode.next = new ListNode(arr[i]);
2485                curNode = curNode.next;
2486            }
2487        }
2488
2489        // 返回以当前ListNode为头结点的链表信息字符串
2490        @Override
2491        public String toString(){
2492
2493            StringBuilder s = new StringBuilder("");
2494            ListNode curNode = this;
2495            while(curNode != null){
2496                s.append(Integer.toString(curNode.val));
2497                s.append(" -> ");
2498                curNode = curNode.next;
2499            }
2500            s.append("NULL");
2501            return s.toString();
2502        }
2503    }
2504
2505    \05-About-Linked-List\02-Test-Your-Linked-List\src\Solution.java
2506
2507    // 206. Reverse Linked List
2508    // https://leetcode.com/problems/reverse-linked-list/description/
```

```java
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution {

    public ListNode reverseList(ListNode head) {

        ListNode pre = null;
        ListNode cur = head;
        while(cur != null){
            ListNode next = cur.next;
            cur.next = pre;
            pre = cur;
            cur = next;
        }

        return pre;
    }

    public static void main(String[] args) {

        int[] nums = {1, 2, 3, 4, 5};
        ListNode head = new ListNode(nums);
        System.out.println(head);

        ListNode head2 = (new Solution()).reverseList(head);
        System.out.println(head2);
    }
}


\05-About-Linked-List\03-Remove-Linked-List-Elements\src\ListNode.java

// Definition for singly-linked list.
// 在Java版本中，我们将LinkedList相关的测试辅助函数写在ListNode里
public class ListNode {

    public int val;
    public ListNode next = null;

    public ListNode(int x) {
        val = x;
    }

    // 根据n个元素的数组arr创建一个链表
    // 使用arr为参数，创建另外一个ListNode的构造函数
    public ListNode (int[] arr){

        if(arr == null || arr.length == 0)
            throw new IllegalArgumentException("arr can not be empty");

        this.val = arr[0];
        ListNode curNode = this;
        for(int i = 1 ; i < arr.length ; i ++){
            curNode.next = new ListNode(arr[i]);
            curNode = curNode.next;
        }
    }

    // 返回以当前ListNode为头结点的链表信息字符串
    @Override
    public String toString(){

        StringBuilder s = new StringBuilder("");
        ListNode curNode = this;
        while(curNode != null){
            s.append(Integer.toString(curNode.val));
```

```
            s.append(" -> ");
            curNode = curNode.next;
        }
        s.append("NULL");
        return s.toString();
    }
}

\05-About-Linked-List\03-Remove-Linked-List-Elements\src\Solution1.java

// 203. Remove Linked List Elements
// https://leetcode.com/problems/remove-linked-list-elements/description/
// 不使用虚拟头结点
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution1 {

    public ListNode removeElements(ListNode head, int val) {

        // 需要对头结点进行特殊处理
        while(head != null && head.val == val){
            ListNode node = head;
            head = head.next;
        }

        if(head == null)
            return head;

        ListNode cur = head;
        while(cur.next != null){
            if(cur.next.val == val){
                ListNode delNode = cur.next;
                cur.next = delNode.next;
            }
            else
                cur = cur.next;
        }

        return head;
    }

    public static void main(String[] args) {

        int[] arr = {1, 2, 6, 3, 4, 5, 6};
        int val = 6;

        ListNode head = new ListNode(arr);
        System.out.println(head);

        (new Solution1()).removeElements(head, val);
        System.out.println(head);
    }
}


\05-About-Linked-List\03-Remove-Linked-List-Elements\src\Solution2.java

// 203. Remove Linked List Elements
// https://leetcode.com/problems/remove-linked-list-elements/description/
// 使用虚拟头结点
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution2 {

    public ListNode removeElements(ListNode head, int val) {

```

```
2641            // 创建虚拟头结点
2642            ListNode dummyHead = new ListNode(0);
2643            dummyHead.next = head;
2644
2645            ListNode cur = dummyHead;
2646            while(cur.next != null){
2647                if(cur.next.val == val ){
2648                    ListNode delNode = cur.next;
2649                    cur.next = delNode.next;
2650                }
2651                else
2652                    cur = cur.next;
2653            }
2654
2655            return dummyHead.next;
2656        }
2657
2658        public static void main(String[] args) {
2659
2660            int[] arr = {1, 2, 6, 3, 4, 5, 6};
2661            int val = 6;
2662
2663            ListNode head = new ListNode(arr);
2664            System.out.println(head);
2665
2666            (new Solution1()).removeElements(head, val);
2667            System.out.println(head);
2668        }
2669    }
2670
2671
2672    \05-About-Linked-List\04-Swap-Nodes-in-Pairs\src\ListNode.java
2673
2674    // Definition for singly-linked list.
2675    // 在Java版本中，我们将LinkedList相关的测试辅助函数写在ListNode里
2676    public class ListNode {
2677
2678        public int val;
2679        public ListNode next = null;
2680
2681        public ListNode(int x) {
2682            val = x;
2683        }
2684
2685        // 根据n个元素的数组arr创建一个链表
2686        // 使用arr为参数，创建另外一个ListNode的构造函数
2687        public ListNode (int[] arr){
2688
2689            if(arr == null || arr.length == 0)
2690                throw new IllegalArgumentException("arr can not be empty");
2691
2692            this.val = arr[0];
2693            ListNode curNode = this;
2694            for(int i = 1 ; i < arr.length ; i ++){
2695                curNode.next = new ListNode(arr[i]);
2696                curNode = curNode.next;
2697            }
2698        }
2699
2700        // 返回以当前ListNode为头结点的链表信息字符串
2701        @Override
2702        public String toString(){
2703
2704            StringBuilder s = new StringBuilder("");
2705            ListNode curNode = this;
2706            while(curNode != null){
```

```
            s.append(Integer.toString(curNode.val));
            s.append(" -> ");
            curNode = curNode.next;
        }
        s.append("NULL");
        return s.toString();
    }
}

\05-About-Linked-List\04-Swap-Nodes-in-Pairs\src\Solution.java

// 24. Swap Nodes in Pairs
// https://leetcode.com/problems/swap-nodes-in-pairs/description/
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution {

    public ListNode swapPairs(ListNode head) {

        ListNode dummyHead = new ListNode(0);
        dummyHead.next = head;

        ListNode p = dummyHead;
        while(p.next != null && p.next.next != null ){
            ListNode node1 = p.next;
            ListNode node2 = node1.next;
            ListNode next = node2.next;
            node2.next = node1;
            node1.next = next;
            p.next = node2;
            p = node1;
        }

        return dummyHead.next;
    }

    public static void main(String[] args) {

        int[] arr = {1, 2, 3, 4};

        ListNode head = new ListNode(arr);
        System.out.println(head);

        head = (new Solution()).swapPairs(head);
        System.out.println(head);
    }
}


\05-About-Linked-List\05-Delete-Node-in-a-Linked-List\src\ListNode.java

// Definition for singly-linked list.
// 在Java版本中，我们将LinkedList相关的测试辅助函数写在ListNode里
public class ListNode {

    public int val;
    public ListNode next = null;

    public ListNode(int x) {
        val = x;
    }

    // 根据n个元素的数组arr创建一个链表
    // 使用arr为参数，创建另外一个ListNode的构造函数
    public ListNode (int[] arr){

```

```
2773            if(arr == null || arr.length == 0)
2774                throw new IllegalArgumentException("arr can not be empty");
2775
2776            this.val = arr[0];
2777            ListNode curNode = this;
2778            for(int i = 1 ; i < arr.length ; i ++){
2779                curNode.next = new ListNode(arr[i]);
2780                curNode = curNode.next;
2781            }
2782        }
2783
2784        ListNode findNode(int x){
2785
2786            ListNode curNode = this;
2787            while(curNode != null){
2788                if(curNode.val == x)
2789                    return curNode;
2790                curNode = curNode.next;
2791            }
2792            return null;
2793        }
2794
2795        // 返回以当前ListNode为头结点的链表信息字符串
2796        @Override
2797        public String toString(){
2798
2799            StringBuilder s = new StringBuilder("");
2800            ListNode curNode = this;
2801            while(curNode != null){
2802                s.append(Integer.toString(curNode.val));
2803                s.append(" -> ");
2804                curNode = curNode.next;
2805            }
2806            s.append("NULL");
2807            return s.toString();
2808        }
2809    }
2810
2811    \05-About-Linked-List\05-Delete-Node-in-a-Linked-List\src\Solution.java
2812
2813    // 237. Delete Node in a Linked List
2814    // https://leetcode.com/problems/delete-node-in-a-linked-list/description/
2815    // 时间复杂度: O(1)
2816    // 空间复杂度: O(1)
2817    public class Solution {
2818
2819        public void deleteNode(ListNode node) {
2820
2821            // 注意: 这个方法对尾节点不适用。题目中要求了给定的node不是尾节点
2822            // 我们检查node.next, 如果为null则抛出异常, 确保了node不是尾节点
2823            if(node == null || node.next == null)
2824                throw new IllegalArgumentException("node should be valid and can not be the tail node.");
2825
2826            node.val = node.next.val;
2827            node.next = node.next.next;
2828        }
2829
2830        public static void main(String[] args) {
2831
2832            int[] arr = {1, 2, 3, 4};
2833
2834            ListNode head = new ListNode(arr);
2835            System.out.println(head);
2836
2837            ListNode node2 = head.findNode(2);
2838            (new Solution()).deleteNode(node2);
```

```
2839            System.out.println(head);
2840        }
2841    }
2842
2843
2844    \05-About-Linked-List\06-Remove-Nth-Node-From-End-of-List\src\ListNode.java
2845
2846    // Definition for singly-linked list.
2847    // 在Java版本中，我们将LinkedList相关的测试辅助函数写在ListNode里
2848    public class ListNode {
2849
2850        public int val;
2851        public ListNode next = null;
2852
2853        public ListNode(int x) {
2854            val = x;
2855        }
2856
2857        // 根据n个元素的数组arr创建一个链表
2858        // 使用arr为参数，创建另外一个ListNode的构造函数
2859        public ListNode (int[] arr){
2860
2861            if(arr == null || arr.length == 0)
2862                throw new IllegalArgumentException("arr can not be empty");
2863
2864            this.val = arr[0];
2865            ListNode curNode = this;
2866            for(int i = 1 ; i < arr.length ; i ++){
2867                curNode.next = new ListNode(arr[i]);
2868                curNode = curNode.next;
2869            }
2870        }
2871
2872        ListNode findNode(int x){
2873
2874            ListNode curNode = this;
2875            while(curNode != null){
2876                if(curNode.val == x)
2877                    return curNode;
2878                curNode = curNode.next;
2879            }
2880            return null;
2881        }
2882
2883        // 返回以当前ListNode为头结点的链表信息字符串
2884        @Override
2885        public String toString(){
2886
2887            StringBuilder s = new StringBuilder("");
2888            ListNode curNode = this;
2889            while(curNode != null){
2890                s.append(Integer.toString(curNode.val));
2891                s.append(" -> ");
2892                curNode = curNode.next;
2893            }
2894            s.append("NULL");
2895            return s.toString();
2896        }
2897    }
2898
2899    \05-About-Linked-List\06-Remove-Nth-Node-From-End-of-List\src\Solution1.java
2900
2901    // 19. Remove Nth Node From End of List
2902    // https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/
2903    //
2904    // 先记录链表总长度
```

```java
// 需要对链表进行两次遍历
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution1 {

    public ListNode removeNthFromEnd(ListNode head, int n) {

        ListNode dummyHead = new ListNode(0);
        dummyHead.next = head;

        int length = 0;
        for(ListNode cur = dummyHead.next ; cur != null ; cur = cur.next)
            length ++;

        int k = length - n;
        assert k >= 0;
        ListNode cur = dummyHead;
        for(int i = 0 ; i < k ; i ++)
            cur = cur.next;

        cur.next = cur.next.next;

        return dummyHead.next;
    }

    public static void main(String[] args) {

        int arr[] = {1, 2, 3, 4, 5};
        ListNode head = new ListNode(arr);
        System.out.println(head);

        head = (new Solution1()).removeNthFromEnd(head, 2);
        System.out.println(head);
    }
}
```

\05-About-Linked-List\06-Remove-Nth-Node-From-End-of-List\src\Solution2.java

```java
// 19. Remove Nth Node From End of List
// https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/
//
// 使用双指针, 对链表只遍历了一遍
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution2 {

    public ListNode removeNthFromEnd(ListNode head, int n) {

        ListNode dummyHead = new ListNode(0);
        dummyHead.next = head;

        ListNode p = dummyHead;
        ListNode q = dummyHead;
        for( int i = 0 ; i < n + 1 ; i ++ ){
            assert q != null;
            q = q.next;
        }

        while(q != null){
            p = p.next;
            q = q.next;
        }

        p.next = p.next.next;

```

```
2971            return dummyHead.next;
2972        }
2973
2974        public static void main(String[] args) {
2975
2976            int arr[] = {1, 2, 3, 4, 5};
2977            ListNode head = new ListNode(arr);
2978            System.out.println(head);
2979
2980            head = (new Solution2()).removeNthFromEnd(head, 2);
2981            System.out.println(head);
2982        }
2983    }
2984
2985
2986    \06-Stack-and-Queue\01-Valid-Parentheses\src\Solution.java
2987
2988    import java.util.Stack;
2989
2990    // 20. Valid Parentheses
2991    // https://leetcode.com/problems/valid-parentheses/description/
2992    // 时间复杂度: O(n)
2993    // 空间复杂度: O(n)
2994    public class Solution {
2995
2996        public boolean isValid(String s) {
2997
2998            Stack<Character> stack = new Stack<Character>();
2999            for( int i = 0 ; i < s.length() ; i ++ )
3000                if( s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt(i) == '[')
3001                    stack.push(s.charAt(i));
3002                else{
3003
3004                    if( stack.size() == 0 )
3005                        return false;
3006
3007                    Character c = stack.pop();
3008
3009                    Character match;
3010                    if( s.charAt(i) == ')' )
3011                        match = '(';
3012                    else if( s.charAt(i) == ']' )
3013                        match = '[';
3014                    else{
3015                        assert s.charAt(i) == '}';
3016                        match = '{';
3017                    }
3018
3019                    if(c != match)
3020                        return false;
3021                }
3022
3023            if( stack.size() != 0 )
3024                return false;
3025
3026            return true;
3027        }
3028
3029        private static void printBool(boolean b){
3030            System.out.println(b ? "True" : "False");
3031        }
3032
3033        public static void main(String[] args) {
3034
3035            printBool((new Solution()).isValid("()"));
3036            printBool((new Solution()).isValid("()[]{}"));
```

```
3037            printBool((new Solution()).isValid("(]"));
3038            printBool((new Solution()).isValid("([)]"));
3039        }
3040    }
3041
3042
3043    \06-Stack-and-Queue\02-Recursion-and-Stack\src\Solution094.java
3044
3045    import java.util.ArrayList;
3046    import java.util.List;
3047
3048    /// 94. Binary Tree Inorder Traversal
3049    /// https://leetcode.com/problems/binary-tree-inorder-traversal/solution/
3050    /// 二叉树的中序遍历
3051    /// 时间复杂度: O(n), n为树的节点个数
3052    /// 空间复杂度: O(h), h为树的高度
3053    public class Solution094 {
3054
3055        // Definition for a binary tree node.
3056        public class TreeNode {
3057            int val;
3058            TreeNode left;
3059            TreeNode right;
3060            TreeNode(int x) { val = x; }
3061        }
3062
3063        public List<Integer> inorderTraversal(TreeNode root) {
3064
3065            ArrayList<Integer> res = new ArrayList<Integer>();
3066            inorderTraversal(root, res);
3067            return res;
3068        }
3069
3070        private void inorderTraversal(TreeNode node, List<Integer> list){
3071            if(node != null){
3072                inorderTraversal(node.left, list);
3073                list.add(node.val);
3074                inorderTraversal(node.right, list);
3075            }
3076        }
3077    }
3078
3079
3080    \06-Stack-and-Queue\02-Recursion-and-Stack\src\Solution144.java
3081
3082    import java.util.ArrayList;
3083    import java.util.List;
3084
3085    /// 144. Binary Tree Preorder Traversal
3086    /// https://leetcode.com/problems/binary-tree-preorder-traversal/description/
3087    /// 二叉树的前序遍历
3088    /// 时间复杂度: O(n), n为树的节点个数
3089    /// 空间复杂度: O(h), h为树的高度
3090    public class Solution144 {
3091
3092        // Definition for a binary tree node.
3093        public class TreeNode {
3094            int val;
3095            TreeNode left;
3096            TreeNode right;
3097            TreeNode(int x) { val = x; }
3098        }
3099
3100        public List<Integer> preorderTraversal(TreeNode root) {
3101
3102            ArrayList<Integer> res = new ArrayList<Integer>();
```

```
3103            preorderTraversal(root, res);
3104            return res;
3105        }
3106
3107        private void preorderTraversal(TreeNode node, List<Integer> list){
3108            if(node != null){
3109                list.add(node.val);
3110                preorderTraversal(node.left, list);
3111                preorderTraversal(node.right, list);
3112            }
3113        }
3114    }
3115
3116
3117    \06-Stack-and-Queue\02-Recursion-and-Stack\src\Solution145.java
3118
3119    import java.util.ArrayList;
3120    import java.util.List;
3121
3122    /// 145. Binary Tree Postorder Traversal
3123    /// https://leetcode.com/problems/binary-tree-postorder-traversal/description/
3124    /// 二叉树的后序遍历
3125    /// 时间复杂度: O(n), n为树的节点个数
3126    /// 空间复杂度: O(h), h为树的高度
3127    public class Solution145 {
3128
3129        // Definition for a binary tree node.
3130        public class TreeNode {
3131            int val;
3132            TreeNode left;
3133            TreeNode right;
3134            TreeNode(int x) { val = x; }
3135        }
3136
3137        public List<Integer> postorderTraversal(TreeNode root) {
3138
3139            ArrayList<Integer> res = new ArrayList<Integer>();
3140            postorderTraversal(root, res);
3141            return res;
3142        }
3143
3144        private void postorderTraversal(TreeNode node, List<Integer> list){
3145            if(node != null){
3146                postorderTraversal(node.left, list);
3147                postorderTraversal(node.right, list);
3148                list.add(node.val);
3149            }
3150        }
3151    }
3152
3153
3154    \06-Stack-and-Queue\03-Non-Recursive-Implementation-of-a-Recursive-Algorithm\src\Solution094.java
3155
3156    import java.util.ArrayList;
3157    import java.util.List;
3158    import java.util.Stack;
3159
3160    /// 94. Binary Tree Inorder Traversal
3161    /// https://leetcode.com/problems/binary-tree-inorder-traversal/solution/
3162    /// 非递归二叉树的中序遍历
3163    /// 时间复杂度: O(n), n为树的节点个数
3164    /// 空间复杂度: O(h), h为树的高度
3165    public class Solution094 {
3166
3167        // Definition for a binary tree node.
3168        public class TreeNode {
```

```
3169            int val;
3170            TreeNode left;
3171            TreeNode right;
3172            TreeNode(int x) { val = x; }
3173        }
3174
3175        private class Command{
3176            String s;   // go, print
3177            TreeNode node;
3178            Command(String s, TreeNode node){
3179                this.s = s;
3180                this.node = node;
3181            }
3182        };
3183
3184        public List<Integer> inorderTraversal(TreeNode root) {
3185
3186            ArrayList<Integer> res = new ArrayList<Integer>();
3187            if(root == null)
3188                return res;
3189
3190            Stack<Command> stack = new Stack<Command>();
3191            stack.push(new Command("go", root));
3192            while(!stack.empty()){
3193                Command command = stack.pop();
3194
3195                if(command.s.equals("print"))
3196                    res.add(command.node.val);
3197                else{
3198                    assert command.s.equals("go");
3199                    if(command.node.right != null)
3200                        stack.push(new Command("go",command.node.right));
3201                    stack.push(new Command("print", command.node));
3202                    if(command.node.left != null)
3203                        stack.push(new Command("go",command.node.left));
3204                }
3205            }
3206            return res;
3207        }
3208
3209    }
3210
3211
3212    \06-Stack-and-Queue\03-Non-Recursive-Implementation-of-a-Recursive-Algorithm\src\Solution144.java
3213
3214    import java.util.ArrayList;
3215    import java.util.List;
3216    import java.util.Stack;
3217
3218    /// 144. Binary Tree Preorder Traversal
3219    /// https://leetcode.com/problems/binary-tree-preorder-traversal/description/
3220    /// 非递归二叉树的前序遍历
3221    /// 时间复杂度: O(n), n为树的节点个数
3222    /// 空间复杂度: O(h), h为树的高度
3223    public class Solution144 {
3224
3225        // Definition for a binary tree node.
3226        public class TreeNode {
3227            int val;
3228            TreeNode left;
3229            TreeNode right;
3230            TreeNode(int x) { val = x; }
3231        }
3232
3233        private class Command{
3234            String s;   // go, print
```

```
3235            TreeNode node;
3236            Command(String s, TreeNode node){
3237                this.s = s;
3238                this.node = node;
3239            }
3240        };
3241
3242        public List<Integer> preorderTraversal(TreeNode root) {
3243
3244            ArrayList<Integer> res = new ArrayList<Integer>();
3245            if(root == null)
3246                return res;
3247
3248            Stack<Command> stack = new Stack<Command>();
3249            stack.push(new Command("go", root));
3250            while(!stack.empty()){
3251                Command command = stack.pop();
3252
3253                if(command.s.equals("print"))
3254                    res.add(command.node.val);
3255                else{
3256                    assert command.s.equals("go");
3257                    if(command.node.right != null)
3258                        stack.push(new Command("go",command.node.right));
3259                    if(command.node.left != null)
3260                        stack.push(new Command("go",command.node.left));
3261                    stack.push(new Command("print", command.node));
3262                }
3263            }
3264            return res;
3265        }
3266
3267    }
3268
3269
3270    \06-Stack-and-Queue\03-Non-Recursive-Implementation-of-a-Recursive-Algorithm\src\Solution145.java
3271
3272    import java.util.ArrayList;
3273    import java.util.List;
3274    import java.util.Stack;
3275
3276    /// 145. Binary Tree Postorder Traversal
3277    /// https://leetcode.com/problems/binary-tree-postorder-traversal/description/
3278    /// 非递归的二叉树的后序遍历
3279    /// 时间复杂度: O(n), n为树的节点个数
3280    /// 空间复杂度: O(h), h为树的高度
3281    public class Solution145 {
3282
3283        // Definition for a binary tree node.
3284        public class TreeNode {
3285            int val;
3286            TreeNode left;
3287            TreeNode right;
3288            TreeNode(int x) { val = x; }
3289        }
3290
3291        private class Command{
3292            String s;   // go, print
3293            TreeNode node;
3294            Command(String s, TreeNode node){
3295                this.s = s;
3296                this.node = node;
3297            }
3298        };
3299
3300        public List<Integer> postorderTraversal(TreeNode root) {
```

```
3301
3302            ArrayList<Integer> res = new ArrayList<Integer>();
3303            if(root == null)
3304                return res;
3305
3306            Stack<Command> stack = new Stack<Command>();
3307            stack.push(new Command("go", root));
3308            while(!stack.empty()){
3309                Command command = stack.pop();
3310
3311                if(command.s.equals("print"))
3312                    res.add(command.node.val);
3313                else{
3314                    assert command.s.equals("go");
3315                    stack.push(new Command("print", command.node));
3316                    if(command.node.right != null)
3317                        stack.push(new Command("go",command.node.right));
3318                    if(command.node.left != null)
3319                        stack.push(new Command("go",command.node.left));
3320                }
3321            }
3322            return res;
3323        }
3324
3325    }
3326
3327
3328    \06-Stack-and-Queue\04-Binary-Tree-Level-Order-Traversal\src\Solution.java
3329
3330    import java.util.ArrayList;
3331    import java.util.List;
3332    import java.util.LinkedList;
3333    import javafx.util.Pair;
3334
3335    /// 102. Binary Tree Level Order Traversal
3336    /// https://leetcode.com/problems/binary-tree-level-order-traversal/description/
3337    /// 二叉树的层序遍历
3338    /// 时间复杂度: O(n), n为树的节点个数
3339    /// 空间复杂度: O(n)
3340    class Solution {
3341
3342        // Definition for a binary tree node.
3343        public class TreeNode {
3344            int val;
3345            TreeNode left;
3346            TreeNode right;
3347            TreeNode(int x) { val = x; }
3348        }
3349
3350        public List<List<Integer>> levelOrder(TreeNode root) {
3351
3352            ArrayList<List<Integer>> res = new ArrayList<List<Integer>>();
3353            if(root == null)
3354                return res;
3355
3356            // 我们使用LinkedList来做为我们的先入先出的队列
3357            LinkedList<Pair<TreeNode, Integer>> queue = new LinkedList<Pair<TreeNode, Integer>>();
3358            queue.addLast(new Pair<TreeNode, Integer>(root, 0));
3359
3360            while(!queue.isEmpty()){
3361
3362                Pair<TreeNode, Integer> front = queue.removeFirst();
3363                TreeNode node = front.getKey();
3364                int level = front.getValue();
3365
3366                if(level == res.size())
```

```
                res.add(new ArrayList<Integer>());
            assert level < res.size();

            res.get(level).add(node.val);
            if(node.left != null)
                queue.addLast(new Pair<TreeNode, Integer>(node.left, level + 1));
            if(node.right != null)
                queue.addLast(new Pair<TreeNode, Integer>(node.right, level + 1));
        }

        return res;
    }
}


\06-Stack-and-Queue\05-Perfect-Squares\src\Solution1.java

import java.util.LinkedList;
import javafx.util.Pair;

// 279. Perfect Squares
// https://leetcode.com/problems/perfect-squares/description/
// 该方法会导致 Time Limit Exceeded 或者 Memory Limit Exceeded
//
// 时间复杂度: O(2^n)
// 空间复杂度: O(2^n)
public class Solution1 {

    public int numSquares(int n) {

        LinkedList<Pair<Integer, Integer>> queue = new LinkedList<Pair<Integer, Integer>>();
        queue.addLast(new Pair<Integer, Integer>(n, 0));

        while(!queue.isEmpty()){
            Pair<Integer, Integer> front = queue.removeFirst();
            int num = front.getKey();
            int step = front.getValue();

            if(num == 0)
                return step;

            for(int i = 1 ; num - i*i >= 0 ; i ++)
                queue.addLast(new Pair(num - i * i, step + 1));
        }

        throw new IllegalStateException("No Solution.");
    }

    public static void main(String[] args) {

        System.out.println((new Solution1()).numSquares(12));
        System.out.println((new Solution1()).numSquares(13));
    }
}


\06-Stack-and-Queue\05-Perfect-Squares\src\Solution2.java

import java.util.LinkedList;
import javafx.util.Pair;

// 279. Perfect Squares
// https://leetcode.com/problems/perfect-squares/description/
// 使用visited数组,记录每一个入队元素
//
// 时间复杂度: O(n)
```

```java
// 空间复杂度: O(n)
public class Solution2 {

    public int numSquares(int n) {

        LinkedList<Pair<Integer, Integer>> queue = new LinkedList<Pair<Integer, Integer>>();
        queue.addLast(new Pair<Integer, Integer>(n, 0));

        boolean[] visited = new boolean[n+1];
        visited[n] = true;

        while(!queue.isEmpty()){
            Pair<Integer, Integer> front = queue.removeFirst();
            int num = front.getKey();
            int step = front.getValue();

            if(num == 0)
                return step;

            for(int i = 1 ; num - i*i >= 0 ; i ++)
                if(!visited[num - i * i]){
                    queue.addLast(new Pair(num - i * i, step + 1));
                    visited[num - i * i] = true;
                }
        }

        throw new IllegalStateException("No Solution.");
    }

    public static void main(String[] args) {

        System.out.println((new Solution2()).numSquares(12));
        System.out.println((new Solution2()).numSquares(13));
    }
}


\06-Stack-and-Queue\05-Perfect-Squares\src\Solution3.java

import java.util.LinkedList;
import javafx.util.Pair;

// 279. Perfect Squares
// https://leetcode.com/problems/perfect-squares/description/
// 进一步优化
//
// 时间复杂度: O(n)
// 空间复杂度: O(n)
public class Solution3 {

    public int numSquares(int n) {

        if(n == 0)
            return 0;

        LinkedList<Pair<Integer, Integer>> queue = new LinkedList<Pair<Integer, Integer>>();
        queue.addLast(new Pair<Integer, Integer>(n, 0));

        boolean[] visited = new boolean[n+1];
        visited[n] = true;

        while(!queue.isEmpty()){
            Pair<Integer, Integer> front = queue.removeFirst();
            int num = front.getKey();
            int step = front.getValue();

```

```
3499                if(num == 0)
3500                    return step;
3501
3502                for(int i = 1 ; num - i*i >= 0 ; i ++){
3503                    int a = num - i*i;
3504                    if(!visited[a]){
3505                        if(a == 0) return step + 1;
3506                        queue.addLast(new Pair(num - i * i, step + 1));
3507                        visited[num - i * i] = true;
3508                    }
3509                }
3510            }
3511
3512            throw new IllegalStateException("No Solution.");
3513        }
3514
3515        public static void main(String[] args) {
3516
3517            System.out.println((new Solution3()).numSquares(12));
3518            System.out.println((new Solution3()).numSquares(13));
3519        }
3520    }
3521
3522
3523    \06-Stack-and-Queue\06-Priority-Queue\src\Main.java
3524
3525    import java.util.Comparator;
3526    import java.util.PriorityQueue;
3527    import java.util.Random;
3528
3529    public class Main {
3530
3531        public static void main(String[] args) {
3532
3533            // 默认的PriorityQueue, 底层是最小堆
3534            PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
3535
3536            for(int i = 0 ; i < 10 ; i ++){
3537                int num = (int)(Math.random() * 100);
3538                pq.add(num);
3539                System.out.println("insert " + num + " in priority queue.");
3540            }
3541
3542            while (!pq.isEmpty())
3543                System.out.print(pq.poll() + " ");
3544
3545            System.out.println();
3546            System.out.println();
3547
3548
3549            // 使用lambda表达式, 创建底层是最大堆的PriorityQueue
3550            PriorityQueue<Integer> pq2 = new PriorityQueue<Integer>(10, (a, b) -> b - a);
3551
3552            for(int i = 0 ; i < 10 ; i ++){
3553                int num = (int)(Math.random() * 100);
3554                pq2.add(num);
3555                System.out.println("insert " + num + " in priority queue.");
3556            }
3557
3558            while (!pq2.isEmpty())
3559                System.out.print(pq2.poll() + " ");
3560
3561            System.out.println();
3562            System.out.println();
3563
3564
```

```
3565            // 使用自定义的Comparator，创建个性化的PriorityQueue
3566            // 注意：也可以使用lambda表达式。在这里只是为了演示PriorityQueue的不同用法
3567            // 同理，上一个例子也可以使用自定义的Comparator的方式完成
3568            class myCmp implements Comparator<Integer>{
3569                @Override
3570                public int compare(Integer a, Integer b){
3571                    if(a%10 != b%10)
3572                        return a%10 - b%10;
3573                    return a - b;
3574                }
3575            }
3576            PriorityQueue<Integer> pq3 = new PriorityQueue<Integer>(10, new myCmp());
3577
3578            for(int i = 0 ; i < 10 ; i ++){
3579                int num = (int)(Math.random() * 100);
3580                pq3.add(num);
3581                System.out.println("insert " + num + " in priority queue.");
3582            }
3583
3584            while (!pq3.isEmpty())
3585                System.out.print(pq3.poll() + " ");
3586
3587            System.out.println();
3588            System.out.println();
3589        }
3590    }
3591
3592
3593    \06-Stack-and-Queue\07-Top-K-Frequent-Elements\src\Solution.java
3594
3595    import java.util.*;
3596    import java.util.HashMap;
3597
3598    import javafx.util.Pair;
3599
3600    // 347. Top K Frequent Elements
3601    // https://leetcode.com/problems/top-k-frequent-elements/description/
3602    // 时间复杂度: O(nlogk)
3603    // 空间复杂度: O(n + k)
3604    class Solution {
3605
3606        private class PairComparator implements Comparator<Pair<Integer, Integer>>{
3607
3608            @Override
3609            public int compare(Pair<Integer, Integer> p1, Pair<Integer, Integer> p2){
3610                if(p1.getKey() != p2.getKey())
3611                    return p1.getKey() - p2.getKey();
3612                return p1.getValue() - p2.getValue();
3613            }
3614        }
3615
3616        public List<Integer> topKFrequent(int[] nums, int k) {
3617
3618            if(k <= 0)
3619                throw new IllegalArgumentException("k should be greater than 0");
3620
3621            // 统计每个元素出现的频率
3622            HashMap<Integer, Integer> freq = new HashMap<Integer, Integer>();
3623            for(int i = 0 ; i < nums.length ; i ++)
3624                if(freq.containsKey(nums[i]))
3625                    freq.put(nums[i], freq.get(nums[i]) + 1);
3626                else
3627                    freq.put(nums[i], 1);
3628
3629            if(k > freq.size())
3630                throw new IllegalArgumentException("k should be less than the number of unique numbers in nums");
```

```
3631
3632            // 扫描freq,维护当前出现频率最高的k个元素
3633            // 在优先队列中,按照频率排序,所以数据对是 (频率,元素) 的形式
3634            PriorityQueue<Pair<Integer, Integer>> pq = new PriorityQueue<Pair<Integer, Integer>>(new PairComparator
3635            for(Integer num: freq.keySet()){
3636                int numFreq = freq.get(num);
3637                if(pq.size() == k){
3638                    if(numFreq > pq.peek().getKey()){
3639                        pq.poll();
3640                        pq.add(new Pair(numFreq, num));
3641                    }
3642                }
3643                else
3644                    pq.add(new Pair(numFreq, num));
3645            }
3646
3647            ArrayList<Integer> res = new ArrayList<Integer>();
3648            while(!pq.isEmpty())
3649                res.add(pq.poll().getValue());
3650
3651            return res;
3652        }
3653
3654        private static void printList(List<Integer> nums){
3655            for(Integer num: nums)
3656                System.out.print(num + " ");
3657            System.out.println();
3658        }
3659
3660        public static void main(String[] args) {
3661
3662            int[] nums = {1, 1, 1, 2, 2, 3};
3663            int k = 2;
3664            printList((new Solution()).topKFrequent(nums, k));
3665        }
3666    }
3667
3668
3669    \06-Stack-and-Queue\Optional-01-Classic-Non-Recursive-Preorder-Traversal\src\Solution1.java
3670
3671    /// Source : https://leetcode.com/problems/binary-tree-preorder-traversal/description/
3672    /// Author : liuyubobobo
3673    /// Time   : 2017-11-17
3674
3675    import java.util.ArrayList;
3676    import java.util.List;
3677    import java.util.Stack;
3678
3679    // Classic Non-Recursive algorithm for preorder traversal
3680    // Time Complexity: O(n), n is the node number in the tree
3681    // Space Complexity: O(h), h is the height of the tree
3682    public class Solution1 {
3683
3684        public List<Integer> preorderTraversal(TreeNode root) {
3685
3686            ArrayList<Integer> res = new ArrayList<Integer>();
3687            if(root == null)
3688                return res;
3689
3690            Stack<TreeNode> stack = new Stack<TreeNode>();
3691            stack.push(root);
3692            while(!stack.empty()){
3693                TreeNode curNode = stack.pop();
3694                res.add(curNode.val);
3695
3696                if(curNode.right != null)
```

```
                stack.push(curNode.right);
            if(curNode.left != null)
                stack.push(curNode.left);
        }
        return res;
    }

}


\06-Stack-and-Queue\Optional-01-Classic-Non-Recursive-Preorder-Traversal\src\Solution2.java

/// Source : https://leetcode.com/problems/binary-tree-preorder-traversal/description/
/// Author : liuyubobobo
/// Time   : 2018-05-30

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

// Another Classic Non-Recursive algorithm for preorder traversal
// Time Complexity: O(n), n is the node number in the tree
// Space Complexity: O(h), h is the height of the tree
public class Solution2 {

    public List<Integer> preorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        if(root == null)
            return res;

        Stack<TreeNode> stack = new Stack<TreeNode>();
        TreeNode cur = root;
        while(cur != null || !stack.isEmpty()){
            while(cur != null){
                res.add(cur.val);
                stack.push(cur);
                cur = cur.left;
            }

            cur = stack.pop();
            cur = cur.right;
        }
        return res;
    }
}


\06-Stack-and-Queue\Optional-01-Classic-Non-Recursive-Preorder-Traversal\src\Solution3.java

/// Source : https://leetcode.com/problems/binary-tree-preorder-traversal/description/
/// Author : liuyubobobo
/// Time   : 2018-05-30

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

// Another Classic Non-Recursive algorithm for preorder traversal
// Time Complexity: O(n), n is the node number in the tree
// Space Complexity: O(h), h is the height of the tree
public class Solution3 {

    public List<Integer> preorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
```

```
3763            if(root == null)
3764                return res;
3765
3766            Stack<TreeNode> stack = new Stack<TreeNode>();
3767            TreeNode cur = root;
3768            while(cur != null || !stack.isEmpty()){
3769                if(cur != null){
3770                    res.add(cur.val);
3771                    stack.push(cur);
3772                    cur = cur.left;
3773                }
3774                else{
3775                    cur = stack.pop();
3776                    cur = cur.right;
3777                }
3778            }
3779            return res;
3780        }
3781    }
3782
3783
3784    \06-Stack-and-Queue\Optional-01-Classic-Non-Recursive-Preorder-Traversal\src\TreeNode.java
3785
3786    // Definition for a binary tree node.
3787    public class TreeNode {
3788        int val;
3789        TreeNode left;
3790        TreeNode right;
3791        TreeNode(int x) { val = x; }
3792    }
3793
3794    \06-Stack-and-Queue\Optional-02-Classic-Non-Recursive-Inorder-Traversal\src\Solution1.java
3795
3796    /// Source : https://leetcode.com/problems/binary-tree-inorder-traversal/solution/
3797    /// Author : liuyubobobo
3798    /// Time   : 2018-05-30
3799
3800    import java.util.ArrayList;
3801    import java.util.List;
3802    import java.util.Stack;
3803
3804    // Classic Non-Recursive algorithm for inorder traversal
3805    // Time Complexity: O(n), n is the node number in the tree
3806    // Space Complexity: O(h), h is the height of the tree
3807    public class Solution1 {
3808
3809        public List<Integer> inorderTraversal(TreeNode root) {
3810
3811            ArrayList<Integer> res = new ArrayList<Integer>();
3812            if(root == null)
3813                return res;
3814
3815            Stack<TreeNode> stack = new Stack<>();
3816            TreeNode cur = root;
3817            while(cur != null || !stack.empty()){
3818
3819                while(cur != null){
3820                    stack.push(cur);
3821                    cur = cur.left;
3822                }
3823
3824                cur = stack.pop();
3825                res.add(cur.val);
3826                cur = cur.right;
3827            }
3828            return res;
```

```
3829        }
3830    }
3831
3832
3833    \06-Stack-and-Queue\Optional-02-Classic-Non-Recursive-Inorder-Traversal\src\Solution2.java
3834
3835    /// Source : https://leetcode.com/problems/binary-tree-inorder-traversal/solution/
3836    /// Author : liuyubobobo
3837    /// Time   : 2018-05-30
3838
3839    import java.util.ArrayList;
3840    import java.util.List;
3841    import java.util.Stack;
3842
3843    // Another Classic Non-Recursive algorithm for inorder traversal
3844    // Time Complexity: O(n), n is the node number in the tree
3845    // Space Complexity: O(h), h is the height of the tree
3846    public class Solution2 {
3847
3848        public List<Integer> inorderTraversal(TreeNode root) {
3849
3850            ArrayList<Integer> res = new ArrayList<Integer>();
3851            if(root == null)
3852                return res;
3853
3854            Stack<TreeNode> stack = new Stack<>();
3855            TreeNode cur = root;
3856            while(cur != null || !stack.empty()){
3857
3858                if(cur != null){
3859                    stack.push(cur);
3860                    cur = cur.left;
3861                }
3862                else{
3863                    cur = stack.pop();
3864                    res.add(cur.val);
3865                    cur = cur.right;
3866                }
3867            }
3868            return res;
3869        }
3870    }
3871
3872
3873    \06-Stack-and-Queue\Optional-02-Classic-Non-Recursive-Inorder-Traversal\src\TreeNode.java
3874
3875    // Definition for a binary tree node.
3876    public class TreeNode {
3877        int val;
3878        TreeNode left;
3879        TreeNode right;
3880        TreeNode(int x) { val = x; }
3881    }
3882
3883    \06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution1.java
3884
3885    /// Source : https://leetcode.com/problems/binary-tree-postorder-traversal/description/
3886    /// Author : liuyubobobo
3887    /// Time   : 2018-05-30
3888
3889    import java.util.ArrayList;
3890    import java.util.List;
3891    import java.util.Stack;
3892
3893    // Non-Recursive
3894    // Using a tag to record whether the node has been visited
```

```
3895    //
3896    // Time Complexity: O(n), n is the node number in the tree
3897    // Space Complexity: O(h), h is the height of the tree
3898    public class Solution1 {
3899
3900        private class TagNode{
3901            TreeNode node;
3902            boolean isFirst;
3903            TagNode(TreeNode node){
3904                this.node = node;
3905                this.isFirst = false;
3906            }
3907        };
3908
3909        public List<Integer> postorderTraversal(TreeNode root) {
3910
3911            ArrayList<Integer> res = new ArrayList<Integer>();
3912            if(root == null)
3913                return res;
3914
3915            Stack<TagNode> stack = new Stack<>();
3916            TreeNode cur = root;
3917            while(cur != null || !stack.empty()){
3918
3919                while(cur != null){
3920                    stack.push(new TagNode(cur));
3921                    cur = cur.left;
3922                }
3923
3924                TagNode tagNode = stack.pop();
3925                cur = tagNode.node;
3926                if(tagNode.isFirst == false){
3927                    tagNode.isFirst = true;
3928                    stack.push(tagNode);
3929                    cur = cur.right;
3930                }
3931                else{
3932                    res.add(cur.val);
3933                    cur = null;
3934                }
3935            }
3936            return res;
3937        }
3938    }
3939
3940
3941    \06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution2.java
3942
3943    /// Source : https://leetcode.com/problems/binary-tree-postorder-traversal/description/
3944    /// Author : liuyubobobo
3945    /// Time   : 2018-05-30
3946
3947    import java.util.ArrayList;
3948    import java.util.List;
3949    import java.util.Stack;
3950
3951    // Non-Recursive
3952    // Using two stacks, Reverse Preorder Traversal!
3953    //
3954    // Time Complexity: O(n)
3955    // Space Complexity: O(n)
3956    public class Solution2 {
3957
3958        public List<Integer> postorderTraversal(TreeNode root) {
3959
3960            ArrayList<Integer> res = new ArrayList<Integer>();
```

```
3961            if(root == null)
3962                return res;
3963
3964            Stack<TreeNode> stack = new Stack<>();
3965            Stack<Integer> output = new Stack<>();
3966
3967            stack.push(root);
3968            while(!stack.empty()){
3969
3970                TreeNode cur = stack.pop();
3971                output.push(cur.val);
3972
3973                if(cur.left != null)
3974                    stack.push(cur.left);
3975                if(cur.right != null)
3976                    stack.push(cur.right);
3977            }
3978
3979            while(!output.empty())
3980                res.add(output.pop());
3981            return res;
3982        }
3983    }
3984
3985
3986    \06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution3.java
3987
3988    /// Source : https://leetcode.com/problems/binary-tree-postorder-traversal/description/
3989    /// Author : liuyubobobo
3990    /// Time   : 2018-07-03
3991
3992    import java.util.ArrayList;
3993    import java.util.List;
3994    import java.util.Stack;
3995    import java.util.LinkedList;
3996
3997    // Non-Recursive
3998    // Using two stacks, Reverse Preorder Traversal!
3999    //
4000    // Time Complexity: O(n)
4001    // Space Complexity: O(n)
4002    public class Solution3 {
4003
4004        public List<Integer> postorderTraversal(TreeNode root){
4005
4006            Stack<TreeNode> stack = new Stack<>();
4007            LinkedList<TreeNode> output = new LinkedList<>();
4008
4009            TreeNode p = root;
4010            while(p != null || !stack.isEmpty()){
4011                if(p != null){
4012                    stack.push(p);
4013                    output.push(p);
4014                    p = p.right;
4015                }
4016                else{
4017                    p = stack.pop();
4018                    p = p.left;
4019                }
4020            }
4021
4022            ArrayList<Integer> res = new ArrayList<>();
4023            while(!output.isEmpty())
4024                res.add(output.pop().val);
4025            return res;
4026        }
```

```
4027    }
4028
4029
4030    \06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution4.java
4031
4032    /// Source : https://leetcode.com/problems/binary-tree-postorder-traversal/description/
4033    /// Author : liuyubobobo
4034    /// Time   : 2018-05-31
4035
4036    import java.util.ArrayList;
4037    import java.util.List;
4038    import java.util.Stack;
4039
4040    // Non-Recursive
4041    // Using a pre pointer to record the last visted node
4042    //
4043    // Time Complexity: O(n)
4044    // Space Complexity: O(h)
4045    public class Solution4 {
4046
4047        public List<Integer> postorderTraversal(TreeNode root) {
4048
4049            ArrayList<Integer> res = new ArrayList<Integer>();
4050            if(root == null)
4051                return res;
4052
4053            Stack<TreeNode> stack = new Stack<>();
4054            TreeNode pre = null;
4055
4056            stack.push(root);
4057            while(!stack.empty()){
4058
4059                TreeNode cur = stack.pop();
4060                if((cur.left == null && cur.right == null) ||
4061                        (pre != null && pre == cur.left && cur.right == null) ||
4062                        (pre != null && pre == cur.right)){
4063                    res.add(cur.val);
4064                    pre = cur;
4065                }
4066                else{
4067                    stack.push(cur);
4068                    if(cur.right != null)
4069                        stack.push(cur.right);
4070                    if(cur.left != null)
4071                        stack.push(cur.left);
4072                }
4073            }
4074            return res;
4075        }
4076    }
4077
4078
4079    \06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution5.java
4080
4081    /// Source : https://leetcode.com/problems/binary-tree-postorder-traversal/description/
4082    /// Author : liuyubobobo
4083    /// Time   : 2018-05-31
4084
4085    import java.util.ArrayList;
4086    import java.util.List;
4087    import java.util.Stack;
4088
4089    // Classic Non-Recursive
4090    // Using a pre pointer to record the last visted node
4091    //
4092    // Time Complexity: O(n)
```

```
// Space Complexity: O(h)
public class Solution5 {

    public List<Integer> postorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        if(root == null)
            return res;

        Stack<TreeNode> stack = new Stack<>();
        TreeNode pre = null;
        TreeNode cur = root;

        while(cur != null || !stack.empty()){

            while(cur != null){
                stack.push(cur);
                cur = cur.left;
            }

            cur = stack.pop();
            if(cur.right == null || pre == cur.right){
                res.add(cur.val);
                pre = cur;
                cur = null;
            }
            else{
                stack.push(cur);
                cur = cur.right;
            }
        }
        return res;
    }
}


\06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution6.java

/// Source : https://leetcode.com/problems/binary-tree-postorder-traversal/description/
/// Author : liuyubobobo
/// Time   : 2018-05-31

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

// Classic Non-Recursive
// Using a pre pointer to record the last visted node
//
// Time Complexity: O(n)
// Space Complexity: O(h)
public class Solution6 {

    public List<Integer> postorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        if(root == null)
            return res;

        Stack<TreeNode> stack = new Stack<>();
        TreeNode pre = null;
        TreeNode cur = root;

        while(cur != null || !stack.empty()){

            if(cur != null){
```

```
                stack.push(cur);
                cur = cur.left;
            }
            else{
                cur = stack.pop();
                if(cur.right == null || pre == cur.right){
                    res.add(cur.val);
                    pre = cur;
                    cur = null;
                }
                else{
                    stack.push(cur);
                    cur = cur.right;
                }
            }
        }
        return res;
    }
}


\06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\TreeNode.java

// Definition for a binary tree node.
public class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

\06-Stack-and-Queue\Optional-04-Binary-Tree-Morris-Traversal\src\InorderSolution.java

/// Source : https://leetcode.com/problems/binary-tree-inorder-traversal/solution/
/// Author : liuyubobobo
/// Time   : 2018-05-30

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

// Inorder Morris Traversal
// Time Complexity: O(n), n is the node number in the tree
// Space Complexity: O(1)
public class InorderSolution {

    public List<Integer> inorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        if(root == null)
            return res;

        TreeNode cur = root;
        while(cur != null){

            if(cur.left == null){
                res.add(cur.val);
                cur = cur.right;
            }
            else{
                TreeNode prev = cur.left;
                while(prev.right != null && prev.right != cur)
                    prev = prev.right;

                if(prev.right == null){
                    prev.right = cur;
```

```
                    cur = cur.left;
                }
                else{
                    prev.right = null;
                    res.add(cur.val);
                    cur = cur.right;
                }
            }
        }
        return res;
    }
}


\06-Stack-and-Queue\Optional-04-Binary-Tree-Morris-Traversal\src\PostorderSolution.java

/// Source : https://leetcode.com/problems/binary-tree-postorder-traversal/description/
/// Author : liuyubobobo
/// Time   : 2018-05-31

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Stack;

// Morris PostOrder Traversal
//
// Time Complexity: O(n)
// Space Complexity: O(1)
public class PostorderSolution {

    public List<Integer> postorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        if(root == null)
            return res;

        TreeNode dummyRoot = new TreeNode(-1);
        dummyRoot.left = root;

        TreeNode cur = dummyRoot;
        while(cur != null){
            if(cur.left == null)
                cur = cur.right;
            else{
                TreeNode pre = cur.left;
                while(pre.right != null && pre.right != cur)
                    pre = pre.right;

                if(pre.right == null){
                    pre.right = cur;
                    cur = cur.left;
                }
                else{
                    pre.right = null;
                    reverseTraversal(cur.left, res);
                    cur = cur.right;
                }
            }
        }
        return res;
    }

    private void reverseTraversal(TreeNode node, ArrayList<Integer> res){
        int start = res.size();
        while(node != null){
```

```
            res.add(node.val);
            node = node.right;
        }

        int i = start, j = res.size() - 1;
        while(i < j){
            Integer t = res.get(i);
            res.set(i, res.get(j));
            res.set(j, t);

            i ++;
            j --;
        }
    }
}


\06-Stack-and-Queue\Optional-04-Binary-Tree-Morris-Traversal\src\PreorderSolution.java

/// Source : https://leetcode.com/problems/binary-tree-preorder-traversal/description/
/// Author : liuyubobobo
/// Time   : 2018-05-29

import java.util.ArrayList;
import java.util.List;

// PreOrder Morris Traversal
// Time Complexity: O(n), n is the node number in the tree
// Space Complexity: O(1)
public class PreorderSolution {

    public List<Integer> preorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        if(root == null)
            return res;

        TreeNode cur = root;
        while(cur != null){
            if(cur.left == null){
                res.add(cur.val);
                cur = cur.right;
            }
            else{
                TreeNode prev = cur.left;
                while(prev.right != null && prev.right != cur)
                    prev = prev.right;

                if(prev.right == null){
                    res.add(cur.val);
                    prev.right = cur;
                    cur = cur.left;
                }
                else{
                    prev.right = null;
                    cur = cur.right;
                }
            }
        }

        return res;
    }
}


\06-Stack-and-Queue\Optional-04-Binary-Tree-Morris-Traversal\src\TreeNode.java
```

```
4357
4358    // Definition for a binary tree node.
4359    public class TreeNode {
4360        int val;
4361        TreeNode left;
4362        TreeNode right;
4363        TreeNode(int x) { val = x; }
4364    }
4365
4366    \06-Stack-and-Queue\Optional-05-Word-Ladder\src\Solution.java
4367
4368    /// Source : https://leetcode.com/problems/word-ladder/description/
4369    /// Author : liuyubobobo
4370    /// Time   : 2018-03-27
4371
4372    import java.util.ArrayList;
4373    import java.util.Arrays;
4374    import java.util.List;
4375    import java.util.LinkedList;
4376
4377    /// BFS
4378    /// Time Complexity: O(n*n)
4379    /// Space Complexity: O(n)
4380    public class Solution {
4381
4382        public int ladderLength(String beginWord, String endWord, List<String> wordList) {
4383
4384            int end = wordList.indexOf(endWord);
4385            if(end == -1)
4386                return 0;
4387
4388            if(!wordList.contains(beginWord))
4389                wordList.add(beginWord);
4390            int begin = wordList.indexOf(beginWord);
4391
4392            int n = wordList.size();
4393            boolean[][] g = new boolean[n][n];
4394            for(int i = 0 ; i < n ; i ++)
4395                for(int j = 0 ; j < i ; j ++)
4396                    g[j][i] = g[i][j] = similar(wordList.get(i), wordList.get(j));
4397
4398            // bfs
4399            LinkedList<Integer> q = new LinkedList<>();
4400            int[] step = new int[n];
4401
4402            q.addLast(begin);
4403            step[begin] = 1;
4404            while(!q.isEmpty()){
4405
4406                int cur = q.removeFirst();
4407
4408                for(int i = 0 ; i < n ; i ++)
4409                    if(step[i] == 0 && g[cur][i]){
4410                        if(i == end)
4411                            return step[cur] + 1;
4412                        step[i] = step[cur] + 1;
4413                        q.addLast(i);
4414                    }
4415            }
4416
4417            return 0;
4418        }
4419
4420        private boolean similar(String word1, String word2){
4421
4422            if(word1.length() != word2.length() || word1.equals(word2))
```

```
4423                throw new IllegalArgumentException();
4424
4425            int diff = 0;
4426            for(int i = 0 ; i < word1.length() ; i ++)
4427                if(word1.charAt(i) != word2.charAt(i)){
4428                    diff ++;
4429                    if(diff > 1)
4430                        return false;
4431                }
4432            return true;
4433        }
4434
4435        public static void main(String[] args) {
4436
4437            ArrayList<String> wordList1 = new ArrayList<String>(
4438                    Arrays.asList("hot","dot","dog","lot","log","cog"));
4439            String beginWord1 = "hit";
4440            String endWord1 = "cog";
4441            System.out.println((new Solution()).ladderLength(beginWord1, endWord1, wordList1));
4442
4443            // 5
4444
4445            // ---
4446
4447            ArrayList<String> wordList2 = new ArrayList<String>(
4448                    Arrays.asList("a","b","c"));
4449            String beginWord2 = "a";
4450            String endWord2 = "c";
4451            System.out.println((new Solution()).ladderLength(beginWord2, endWord2, wordList2));
4452            // 2
4453        }
4454    }
4455
4456
4457    \06-Stack-and-Queue\Optional-05-Word-Ladder\src\Solution2.java
4458
4459    /// Source : https://leetcode.com/problems/word-ladder/description/
4460    /// Author : liuyubobobo
4461    /// Time   : 2018-03-27
4462
4463    import java.util.ArrayList;
4464    import java.util.Arrays;
4465    import java.util.List;
4466    import java.util.LinkedList;
4467    import java.util.HashSet;
4468    import javafx.util.Pair;
4469
4470    /// BFS
4471    /// Using set to store all the words and erase visited word eagerly.
4472    /// Time Complexity: O(n*n)
4473    /// Space Complexity: O(n)
4474    public class Solution2 {
4475
4476        public int ladderLength(String beginWord, String endWord, List<String> wordList) {
4477
4478            HashSet<String> wordSet = new HashSet<>();
4479            for(String word: wordList)
4480                wordSet.add(word);
4481
4482            // bfs
4483            LinkedList<Pair<String, Integer>> q = new LinkedList<>();
4484            q.addLast(new Pair<>(beginWord, 1));
4485            wordSet.remove(beginWord);
4486
4487            HashSet<String> visited = new HashSet<>();
4488
```

```
4489            while(!q.isEmpty()){
4490
4491                String curWord = q.getFirst().getKey();
4492                int curStep = q.getFirst().getValue();
4493                q.removeFirst();
4494
4495                visited.clear();
4496                for(String word: wordSet){
4497                    if(similar(word, curWord)){
4498                        if(word.equals(endWord))
4499                            return curStep + 1;
4500                        q.addLast(new Pair<>(word, curStep + 1));
4501                        visited.add(word);
4502                    }
4503                }
4504
4505                for(String word: visited)
4506                    wordSet.remove(word);
4507            }
4508
4509            return 0;
4510        }
4511
4512        private boolean similar(String word1, String word2){
4513
4514            if(word1.length() != word2.length() || word1.equals(word2))
4515                throw new IllegalArgumentException();
4516
4517            int diff = 0;
4518            for(int i = 0 ; i < word1.length() ; i ++)
4519                if(word1.charAt(i) != word2.charAt(i)){
4520                    diff ++;
4521                    if(diff > 1)
4522                        return false;
4523                }
4524            return true;
4525        }
4526
4527        public static void main(String[] args) {
4528
4529            ArrayList<String> wordList1 = new ArrayList<String>(
4530                    Arrays.asList("hot","dot","dog","lot","log","cog"));
4531            String beginWord1 = "hit";
4532            String endWord1 = "cog";
4533            System.out.println((new Solution()).ladderLength(beginWord1, endWord1, wordList1));
4534
4535            // 5
4536
4537            // ---
4538
4539            ArrayList<String> wordList2 = new ArrayList<String>(
4540                    Arrays.asList("a","b","c"));
4541            String beginWord2 = "a";
4542            String endWord2 = "c";
4543            System.out.println((new Solution()).ladderLength(beginWord2, endWord2, wordList2));
4544            // 2
4545        }
4546    }
4547
4548
4549    \06-Stack-and-Queue\Optional-05-Word-Ladder\src\Solution3.java
4550
4551    /// Source : https://leetcode.com/problems/word-ladder/description/
4552    /// Author : liuyubobobo
4553    /// Time   : 2018-03-27
4554
```

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.LinkedList;

/// Bi-directional BFS
/// Time Complexity: O(n*n)
/// Space Complexity: O(n)
public class Solution3 {

    public int ladderLength(String beginWord, String endWord, List<String> wordList) {

        int end = wordList.indexOf(endWord);
        if(end == -1)
            return 0;

        if(!wordList.contains(beginWord))
            wordList.add(beginWord);
        int begin = wordList.indexOf(beginWord);

        int n = wordList.size();
        boolean[][] g = new boolean[n][n];
        for(int i = 0 ; i < n ; i ++)
            for(int j = 0 ; j < i ; j ++)
                g[j][i] = g[i][j] = similar(wordList.get(i), wordList.get(j));


        // bi-derectional-bfs
        LinkedList<Integer> qStart = new LinkedList<>();
        LinkedList<Integer> qEnd = new LinkedList<>();

        int[] stepStart = new int[n];
        int[] stepEnd = new int[n];

        qStart.addLast(begin);
        stepStart[begin] = 1;

        qEnd.addLast(end);
        stepEnd[end] = 1;

        while(!qStart.isEmpty() && !qEnd.isEmpty()){

            int curStart = qStart.removeFirst();
            int curEnd = qEnd.removeFirst();

            for(int i = 0 ; i < n ; i ++) {
                if (stepStart[i] == 0 && g[curStart][i]) {
                    stepStart[i] = stepStart[curStart] + 1;
                    qStart.addLast(i);
                }
            }

            for(int i = 0 ; i < n ; i ++){
                if(stepEnd[i] == 0 && g[curEnd][i]){
                    stepEnd[i] = stepEnd[curEnd] + 1;
                    qEnd.addLast(i);
                }
            }

            // check intersection
            int res = Integer.MAX_VALUE;
            for(int i = 0 ; i < n ; i ++)
                if(stepStart[i] != 0 && stepEnd[i] != 0)
                    res = Integer.min(res, stepStart[i] + stepEnd[i] - 1);

            if(res != Integer.MAX_VALUE)
```

```
                    return res;
            }

            return 0;
        }

        private boolean similar(String word1, String word2){

            if(word1.length() != word2.length() || word1.equals(word2))
                throw new IllegalArgumentException();

            int diff = 0;
            for(int i = 0 ; i < word1.length() ; i ++)
                if(word1.charAt(i) != word2.charAt(i)){
                    diff ++;
                    if(diff > 1)
                        return false;
                }
            return true;
        }

        public static void main(String[] args) {

            ArrayList<String> wordList1 = new ArrayList<String>(
                    Arrays.asList("hot","dot","dog","lot","log","cog"));
            String beginWord1 = "hit";
            String endWord1 = "cog";
            System.out.println((new Solution()).ladderLength(beginWord1, endWord1, wordList1));

            // 5

            // ---

            ArrayList<String> wordList2 = new ArrayList<String>(
                    Arrays.asList("a","b","c"));
            String beginWord2 = "a";
            String endWord2 = "c";
            System.out.println((new Solution()).ladderLength(beginWord2, endWord2, wordList2));
            // 2
        }
    }


    \06-Stack-and-Queue\Optional-05-Word-Ladder\src\Solution4.java

    /// Source : https://leetcode.com/problems/word-ladder/description/
    /// Author : liuyubobobo
    /// Time   : 2018-03-27

    import java.util.ArrayList;
    import java.util.Arrays;
    import java.util.List;
    import java.util.LinkedList;
    import java.util.HashMap;

    /// Bi-directional BFS
    /// No need to calculate all pairs similarity
    /// Time Complexity: O(n*n)
    /// Space Complexity: O(n)
    public class Solution4 {

        public int ladderLength(String beginWord, String endWord, List<String> wordList) {

            if(!wordList.contains(endWord))
                return 0;

```

```java
        // bi-derectional-bfs
        LinkedList<String> qStart = new LinkedList<>();
        LinkedList<String> qEnd = new LinkedList<>();

        HashMap<String, Integer> stepStart = new HashMap<>();
        HashMap<String, Integer> stepEnd = new HashMap<>();

        qStart.addLast(beginWord);
        stepStart.put(beginWord, 1);

        qEnd.addLast(endWord);
        stepEnd.put(endWord, 1);

        while(!qStart.isEmpty() && !qEnd.isEmpty()){

            String curStartWord = qStart.removeFirst();
            String curEndWord = qEnd.removeFirst();
            for(String word: wordList){
                if(!stepStart.containsKey(word) && similar(word, curStartWord)){
                    stepStart.put(word, stepStart.get(curStartWord) + 1);
                    qStart.addLast(word);
                }

                if(!stepEnd.containsKey(word) && similar(word, curEndWord)){
                    stepEnd.put(word, stepEnd.get(curEndWord) + 1);
                    qEnd.addLast(word);
                }
            }

            // check intersection
            int res = Integer.MAX_VALUE;
            for(String word: wordList)
                if(stepStart.containsKey(word) && stepEnd.containsKey(word))
                    res = Integer.min(res,
                            stepStart.get(word) + stepEnd.get(word) - 1);

            if(res != Integer.MAX_VALUE)
                return res;
        }

        return 0;
    }

    private boolean similar(String word1, String word2){

        if(word1.length() != word2.length() || word1.equals(word2))
            throw new IllegalArgumentException();

        int diff = 0;
        for(int i = 0 ; i < word1.length() ; i ++)
            if(word1.charAt(i) != word2.charAt(i)){
                diff ++;
                if(diff > 1)
                    return false;
            }
        return true;
    }

    public static void main(String[] args) {

        ArrayList<String> wordList1 = new ArrayList<String>(
                Arrays.asList("hot","dot","dog","lot","log","cog"));
        String beginWord1 = "hit";
        String endWord1 = "cog";
        System.out.println((new Solution()).ladderLength(beginWord1, endWord1, wordList1));

```

```
4753            // 5
4754
4755            // ---
4756
4757            ArrayList<String> wordList2 = new ArrayList<String>(
4758                    Arrays.asList("a","b","c"));
4759            String beginWord2 = "a";
4760            String endWord2 = "c";
4761            System.out.println((new Solution()).ladderLength(beginWord2, endWord2, wordList2));
4762            // 2
4763        }
4764    }
4765
4766
4767    \07-Binary-Tree-and-Recursion\01-Maximum-Depth-of-Binary-Tree\src\Solution.java
4768
4769    // 104. Maximum Depth of Binary Tree
4770    // https://leetcode.com/problems/maximum-depth-of-binary-tree/description/
4771    // 时间复杂度: O(n), n是树中的节点个数
4772    // 空间复杂度: O(h), h是树的高度
4773    class Solution {
4774
4775        // Definition for a binary tree node.
4776        public class TreeNode {
4777            int val;
4778            TreeNode left;
4779            TreeNode right;
4780            TreeNode(int x) { val = x; }
4781        }
4782
4783        public int maxDepth(TreeNode root) {
4784
4785            if(root == null)
4786                return 0;
4787
4788            return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));
4789        }
4790    }
4791
4792
4793    \07-Binary-Tree-and-Recursion\02-Invert-Binary-Tree\src\Solution.java
4794
4795    /// 226. Invert Binary Tree
4796    /// https://leetcode.com/problems/invert-binary-tree/description/
4797    /// 时间复杂度: O(n), n为树中节点个数
4798    /// 空间复杂度: O(h), h为树的高度
4799    public class Solution {
4800
4801        // Definition for a binary tree node.
4802        public class TreeNode {
4803            int val;
4804            TreeNode left;
4805            TreeNode right;
4806            TreeNode(int x) { val = x; }
4807        }
4808
4809        public TreeNode invertTree(TreeNode root) {
4810
4811            if(root == null)
4812                return null;
4813
4814            TreeNode left = invertTree(root.left);
4815            TreeNode right = invertTree(root.right);
4816
4817            root.left = right;
4818            root.right = left;
```

```
4819
4820            return root;
4821        }
4822    }
4823
4824
4825    \07-Binary-Tree-and-Recursion\03-Path-Sum\src\Solution.java
4826
4827    /// 112. Path Sum
4828    /// https://leetcode.com/problems/path-sum/description/
4829    /// 时间复杂度: O(n), n为树的节点个数
4830    /// 空间复杂度: O(h), h为树的高度
4831    class Solution {
4832
4833        // Definition for a binary tree node.
4834        public class TreeNode {
4835            int val;
4836            TreeNode left;
4837            TreeNode right;
4838            TreeNode(int x) { val = x; }
4839        }
4840
4841        public boolean hasPathSum(TreeNode root, int sum) {
4842
4843            if(root == null)
4844                return false;
4845
4846            if(root.left == null && root.right == null)
4847                return sum == root.val;
4848
4849            return hasPathSum(root.left, sum - root.val)
4850                    || hasPathSum(root.right, sum - root.val);
4851        }
4852    }
4853
4854    \07-Binary-Tree-and-Recursion\04-Binary-Tree-Paths\src\Solution.java
4855
4856    import java.util.List;
4857    import java.util.ArrayList;
4858
4859    /// 257. Binary Tree Paths
4860    /// https://leetcode.com/problems/binary-tree-paths/description/
4861    /// 时间复杂度: O(n), n为树中的节点个数
4862    /// 空间复杂度: O(h), h为树的高度
4863    public class Solution {
4864
4865        // Definition for a binary tree node.
4866        public class TreeNode {
4867            int val;
4868            TreeNode left;
4869            TreeNode right;
4870            TreeNode(int x) { val = x; }
4871        }
4872
4873        public List<String> binaryTreePaths(TreeNode root) {
4874
4875            ArrayList<String> res = new ArrayList<String>();
4876
4877            if(root == null)
4878                return res;
4879
4880            if(root.left == null && root.right == null){
4881                res.add(Integer.toString(root.val));
4882                return res;
4883            }
4884
```

```
4885            List<String> leftPaths = binaryTreePaths(root.left);
4886            for(String s: leftPaths){
4887                StringBuilder sb = new StringBuilder(Integer.toString(root.val));
4888                sb.append("->");
4889                sb.append(s);
4890                res.add(sb.toString());
4891            }
4892
4893            List<String> rightPaths = binaryTreePaths(root.right);
4894            for(String s: rightPaths) {
4895                StringBuilder sb = new StringBuilder(Integer.toString(root.val));
4896                sb.append("->");
4897                sb.append(s);
4898                res.add(sb.toString());
4899            }
4900
4901            return res;
4902        }
4903    }
4904
4905
4906    \07-Binary-Tree-and-Recursion\05-Path-Sum-III\src\Solution.java
4907
4908    /// 437. Path Sum III
4909    /// https://leetcode.com/problems/path-sum-iii/description/
4910    /// 时间复杂度: O(n), n为树的节点个数
4911    /// 空间复杂度: O(h), h为树的高度
4912    class Solution {
4913
4914        /// Definition for a binary tree node.
4915        public static class TreeNode {
4916            int val;
4917            TreeNode left;
4918            TreeNode right;
4919            TreeNode(int x) { val = x; }
4920        }
4921
4922        // 在以root为根节点的二叉树中,寻找和为sum的路径,返回这样的路径个数
4923        public int pathSum(TreeNode root, int sum) {
4924
4925            if(root == null)
4926                return 0;
4927
4928            return findPath(root, sum)
4929                    + pathSum(root.left , sum)
4930                    + pathSum(root.right , sum);
4931        }
4932
4933        // 在以node为根节点的二叉树中,寻找包含node的路径,和为sum
4934        // 返回这样的路径个数
4935        private int findPath(TreeNode node, int num){
4936
4937            if(node == null)
4938                return 0;
4939
4940            int res = 0;
4941            if(node.val == num)
4942                res += 1;
4943
4944            res += findPath(node.left , num - node.val);
4945            res += findPath(node.right , num - node.val);
4946
4947            return res;
4948        }
4949
4950        public static void main(String[] args) {
```

```
4951
4952            // 手动创建Leetcode题页上的测试用例。
4953            // 当然，有更好的更智能的创建二叉树的方式，有兴趣的同学可以自行研究编写程序:)
4954
4955            /*****************
4956             * 测试用例:
4957             *
4958             *       10
4959             *      /  \
4960             *     5   -3
4961             *    / \    \
4962             *   3   2   11
4963             *  / \   \
4964             * 3  -2   1
4965             *****************/
4966            TreeNode node1 = new TreeNode(3);
4967            TreeNode node2 = new TreeNode(-2);
4968
4969            TreeNode node3 = new TreeNode(3);
4970            node3.left = node1;
4971            node3.right = node2;
4972
4973            TreeNode node4 = new TreeNode(1);
4974            TreeNode node5 = new TreeNode(2);
4975            node5.right = node4;
4976
4977            TreeNode node6 = new TreeNode(5);
4978            node6.left = node3;
4979            node6.right = node5;
4980
4981            TreeNode node7 = new TreeNode(11);
4982            TreeNode node8 = new TreeNode(-3);
4983            node8.right = node7;
4984
4985            TreeNode node9 = new TreeNode(10);
4986            node9.left = node6;
4987            node9.right = node8;
4988
4989            System.out.println((new Solution()).pathSum(node9, 8));
4990        }
4991    }
4992
4993
4994    \07-Binary-Tree-and-Recursion\06-Lowest-Common-Ancestor-of-a-Binary-Search-Tree\src\Solution.java
4995
4996    /// 235. Lowest Common Ancestor of a Binary Search Tree
4997    /// https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/description/
4998    /// 时间复杂度: O(lgn), 其中n为树的节点个数
4999    /// 空间复杂度: O(h), 其中h为树的高度
5000    class Solution {
5001
5002        // Definition for a binary tree node.
5003        public class TreeNode {
5004            int val;
5005            TreeNode left;
5006            TreeNode right;
5007            TreeNode(int x) { val = x; }
5008        }
5009
5010        public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
5011
5012            if(p == null || q == null)
5013                throw new IllegalArgumentException("p or q can not be null.");
5014
5015            if(root == null)
5016                return null;
```

```
5017
5018            if(p.val < root.val && q.val < root.val)
5019                return lowestCommonAncestor(root.left, p, q);
5020            if(p.val > root.val && q.val > root.val)
5021                return lowestCommonAncestor(root.right, p, q);
5022
5023            assert p.val == root.val || q.val == root.val
5024                    || (root.val - p.val) * (root.val - q.val) < 0;
5025
5026            return root;
5027        }
5028    }
5029
5030
5031    \08-Recurion-and-Backstracking\01-02-Letter-Combinations-of-a-Phone-Number\src\Solution.java
5032
5033    import java.util.List;
5034    import java.util.ArrayList;
5035
5036    /// 17. Letter Combinations of a Phone Number
5037    /// https://leetcode.com/problems/letter-combinations-of-a-phone-number/description/
5038    /// 时间复杂度: O(2^len(s))
5039    /// 空间复杂度: O(len(s))
5040    class Solution {
5041
5042        private String letterMap[] = {
5043                    " ",    //0
5044                    "",     //1
5045                    "abc",  //2
5046                    "def",  //3
5047                    "ghi",  //4
5048                    "jkl",  //5
5049                    "mno",  //6
5050                    "pqrs", //7
5051                    "tuv",  //8
5052                    "wxyz"  //9
5053        };
5054
5055        private ArrayList<String> res;
5056
5057        public List<String> letterCombinations(String digits) {
5058
5059            res = new ArrayList<String>();
5060            if(digits.equals(""))
5061                return res;
5062
5063            findCombination(digits, 0, "");
5064            return res;
5065        }
5066
5067        // s中保存了此时从digits[0...index-1]翻译得到的一个字母字符串
5068        // 寻找和digits[index]匹配的字母, 获得digits[0...index]翻译得到的解
5069        private void findCombination(String digits, int index, String s){
5070
5071            System.out.println(index + " : " + s);
5072            if(index == digits.length()){
5073                res.add(s);
5074                System.out.println("get " + s + " , return");
5075                return;
5076            }
5077
5078            Character c = digits.charAt(index);
5079            assert  c.compareTo('0') >= 0 &&
5080                    c.compareTo('9') <= 0 &&
5081                    c.compareTo('1') != 0;
5082            String letters = letterMap[c - '0'];
```

```
            for(int i = 0 ; i < letters.length() ; i ++){
                System.out.println("digits[" + index + "] = " + c +
                        " , use " + letters.charAt(i));
                findCombination(digits, index+1, s + letters.charAt(i));
            }

            System.out.println("digits[" + index + "] = " + c + " complete, return");

            return;
        }

        private static void printList(List<String> list){
            for(String s: list)
                System.out.println(s);
        }

        public static void main(String[] args) {

            printList((new Solution()).letterCombinations("234"));
        }
    }


    \08-Recurion-and-Backstracking\03-Permutations\src\Solution.java

    import java.util.List;
    import java.util.ArrayList;
    import java.util.LinkedList;

    public class Solution {

        private ArrayList<List<Integer>> res;
        private boolean[] used;

        public List<List<Integer>> permute(int[] nums) {

            res = new ArrayList<List<Integer>>();
            if(nums == null || nums.length == 0)
                return res;

            used = new boolean[nums.length];
            LinkedList<Integer> p = new LinkedList<Integer>();
            generatePermutation(nums, 0, p);

            return res;
        }

        // p中保存了一个有index-1个元素的排列。
        // 向这个排列的末尾添加第index个元素，获得一个有index个元素的排列
        private void generatePermutation(int[] nums, int index, LinkedList<Integer> p){

            if(index == nums.length){
                res.add((LinkedList<Integer>)p.clone());
                return;
            }

            for(int i = 0 ; i < nums.length ; i ++)
                if(!used[i]){
                    used[i] = true;
                    p.addLast(nums[i]);
                    generatePermutation(nums, index + 1, p );
                    p.removeLast();
                    used[i] = false;
                }

            return;
```

```
5149        }
5150
5151        private static void printList(List<Integer> list){
5152            for(Integer e: list)
5153                System.out.print(e + " ");
5154            System.out.println();
5155        }
5156
5157        public static void main(String[] args) {
5158
5159            int[] nums = {1, 2, 3};
5160            List<List<Integer>> res = (new Solution()).permute(nums);
5161            for(List<Integer> list: res)
5162                printList(list);
5163        }
5164    }
5165
5166
5167    \08-Recurion-and-Backstracking\04-Combinations\src\Solution.java
5168
5169    import java.util.List;
5170    import java.util.ArrayList;
5171    import java.util.LinkedList;
5172
5173    /// 77. Combinations
5174    /// https://leetcode.com/problems/combinations/description/
5175    /// 时间复杂度: O(n^k)
5176    /// 空间复杂度: O(k)
5177    public class Solution {
5178
5179        private ArrayList<List<Integer>> res;
5180
5181        public List<List<Integer>> combine(int n, int k) {
5182
5183            res = new ArrayList<List<Integer>>();
5184            if(n <= 0 || k <= 0 || k > n)
5185                return res;
5186
5187            LinkedList<Integer> c = new LinkedList<Integer>();
5188            generateCombinations(n, k, 1, c);
5189
5190            return res;
5191        }
5192
5193        // 求解C(n,k), 当前已经找到的组合存储在c中, 需要从start开始搜索新的元素
5194        private void generateCombinations(int n, int k, int start, LinkedList<Integer> c){
5195
5196            if(c.size() == k){
5197                res.add((List<Integer>)c.clone());
5198                return;
5199            }
5200
5201            for(int i = start ; i <= n ; i ++){
5202                c.addLast(i);
5203                generateCombinations(n, k, i + 1, c);
5204                c.removeLast();
5205            }
5206
5207            return;
5208        }
5209
5210        private static void printList(List<Integer> list){
5211            for(Integer e: list)
5212                System.out.print(e + " ");
5213            System.out.println();
5214        }
```

```
5215
5216        public static void main(String[] args) {
5217
5218            List<List<Integer>> res = (new Solution()).combine(4, 2);
5219            for(List<Integer> list: res)
5220                printList(list);
5221        }
5222    }
5223
5224
5225    \08-Recurion-and-Backstracking\05-Combinations-optimized\src\Solution.java
5226
5227    import java.util.List;
5228    import java.util.ArrayList;
5229    import java.util.LinkedList;
5230
5231    /// 77. Combinations
5232    /// https://leetcode.com/problems/combinations/description/
5233    /// 时间复杂度: O(n^k)
5234    /// 空间复杂度: O(k)
5235    public class Solution {
5236
5237        private ArrayList<List<Integer>> res;
5238
5239        public List<List<Integer>> combine(int n, int k) {
5240
5241            res = new ArrayList<List<Integer>>();
5242            if(n <= 0 || k <= 0 || k > n)
5243                return res;
5244
5245            LinkedList<Integer> c = new LinkedList<Integer>();
5246            generateCombinations(n, k, 1, c);
5247
5248            return res;
5249        }
5250
5251        // 求解C(n,k), 当前已经找到的组合存储在c中, 需要从start开始搜索新的元素
5252        private void generateCombinations(int n, int k, int start, LinkedList<Integer> c){
5253
5254            if(c.size() == k){
5255                res.add((List<Integer>)c.clone());
5256                return;
5257            }
5258
5259            // 还有k - c.size()个空位, 所以, [i...n] 中至少要有 k - c.size() 个元素
5260            // i最多为 n - (k - c.size()) + 1
5261            for(int i = start ; i <= n - (k - c.size()) + 1 ; i ++){
5262                c.addLast(i);
5263                generateCombinations(n, k, i + 1, c);
5264                c.removeLast();
5265            }
5266
5267            return;
5268        }
5269
5270        private static void printList(List<Integer> list){
5271            for(Integer e: list)
5272                System.out.print(e + " ");
5273            System.out.println();
5274        }
5275
5276        public static void main(String[] args) {
5277
5278            List<List<Integer>> res = (new Solution()).combine(4, 2);
5279            for(List<Integer> list: res)
5280                printList(list);
```

```
5281        }
5282    }
5283
5284
5285    \08-Recurion-and-Backstracking\06-Word-Search\src\Solution.java
5286
5287    /// 79. Word Search
5288    /// Source : https://leetcode.com/problems/word-search/description/
5289    ///
5290    /// 回溯法
5291    /// 时间复杂度: O(m*n*m*n)
5292    /// 空间复杂度: O(m*n)
5293    public class Solution {
5294
5295        private int d[][] = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};
5296        private int m, n;
5297        private boolean[][] visited;
5298
5299        public boolean exist(char[][] board, String word) {
5300
5301            if(board == null || word == null)
5302                throw new IllegalArgumentException("board or word can not be null!");
5303
5304            m = board.length;
5305            if(m == 0)
5306                throw new IllegalArgumentException("board can not be empty.");
5307            n = board[0].length;
5308            if(n == 0)
5309                throw new IllegalArgumentException("board can not be empty.");
5310
5311            visited = new boolean[m][n];
5312            for(int i = 0 ; i < m ; i ++)
5313                for(int j = 0 ; j < n ; j ++)
5314                    if(searchWord(board, word, 0, i, j))
5315                        return true;
5316
5317            return false;
5318        }
5319
5320        private boolean inArea( int x , int y ){
5321            return x >= 0 && x < m && y >= 0 && y < n;
5322        }
5323
5324        // 从board[startx][starty]开始, 寻找word[index...word.size())
5325        private boolean searchWord(char[][] board, String word, int index,
5326                                   int startx, int starty){
5327
5328            //assert(inArea(startx,starty));
5329            if(index == word.length() - 1)
5330                return board[startx][starty] == word.charAt(index);
5331
5332            if(board[startx][starty] == word.charAt(index)){
5333                visited[startx][starty] = true;
5334                // 从startx, starty出发,向四个方向寻
5335                for(int i = 0 ; i < 4 ; i ++){
5336                    int newx = startx + d[i][0];
5337                    int newy = starty + d[i][1];
5338                    if(inArea(newx, newy) && !visited[newx][newy] &&
5339                            searchWord(board, word, index + 1, newx, newy))
5340                        return true;
5341                }
5342                visited[startx][starty] = false;
5343            }
5344            return false;
5345        }
5346
```

```
5347        public static void main(String args[]){
5348
5349            char[][] b1 = { {'A','B','C','E'},
5350                            {'S','F','C','S'},
5351                            {'A','D','E','E'}};
5352
5353            String words[] = {"ABCCED", "SEE", "ABCB" };
5354            for(int i = 0 ; i < words.length ; i ++)
5355                if((new Solution()).exist(b1, words[i]))
5356                    System.out.println("found " + words[i]);
5357                else
5358                    System.out.println("can not found " + words[i]);
5359
5360            // ---
5361
5362            char[][] b2 = {{'A'}};
5363            if((new Solution()).exist(b2,"AB"))
5364                System.out.println("found AB");
5365            else
5366                System.out.println("can not found AB");
5367        }
5368    }
5369
5370
5371    \08-Recurion-and-Backstracking\07-Number-of-Islands\src\Solution.java
5372
5373    /// 200. Number of Islands
5374    /// https://leetcode.com/problems/number-of-islands/description/
5375    /// 时间复杂度: O(n*m)
5376    /// 空间复杂度: O(n*m)
5377    class Solution {
5378
5379        private int d[][] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
5380        private int m, n;
5381        private boolean visited[][];
5382
5383        public int numIslands(char[][] grid) {
5384
5385            if(grid == null || grid.length == 0 || grid[0].length == 0)
5386                return 0;
5387
5388            m = grid.length;
5389            n = grid[0].length;
5390
5391            visited = new boolean[m][n];
5392
5393            int res = 0;
5394            for(int i = 0 ; i < m ; i ++)
5395                for(int j = 0 ; j < n ; j ++)
5396                    if(grid[i][j] == '1' && !visited[i][j]){
5397                        dfs(grid, i, j);
5398                        res ++;
5399                    }
5400
5401            return res;
5402        }
5403
5404        // 从grid[x][y]的位置开始,进行floodfill
5405        // 保证(x,y)合法,且grid[x][y]是没有被访问过的陆地
5406        private void dfs(char[][] grid, int x, int y){
5407
5408            //assert(inArea(x,y));
5409            visited[x][y] = true;
5410            for(int i = 0; i < 4; i ++){
5411                int newx = x + d[i][0];
5412                int newy = y + d[i][1];
```

```
5413                if(inArea(newx, newy) && !visited[newx][newy] && grid[newx][newy] == '1')
5414                    dfs(grid, newx, newy);
5415            }
5416
5417            return;
5418        }
5419
5420        private boolean inArea(int x, int y){
5421            return x >= 0 && x < m && y >= 0 && y < n;
5422        }
5423
5424        public static void main(String[] args) {
5425
5426            char grid1[][] = {
5427                {'1','1','1','1','0'},
5428                {'1','1','0','1','0'},
5429                {'1','1','0','0','0'},
5430                {'0','0','0','0','0'}
5431            };
5432            System.out.println((new Solution()).numIslands(grid1));
5433            // 1
5434
5435            // ---
5436
5437            char grid2[][] = {
5438                {'1','1','0','0','0'},
5439                {'1','1','0','0','0'},
5440                {'0','0','1','0','0'},
5441                {'0','0','0','1','1'}
5442            };
5443            System.out.println((new Solution()).numIslands(grid2));
5444            // 3
5445        }
5446    }
5447
5448
5449    \08-Recurion-and-Backstracking\08-N-Queens\src\Solution.java
5450
5451    import java.util.Arrays;
5452    import java.util.LinkedList;
5453    import java.util.List;
5454    import java.util.ArrayList;
5455
5456    /// 51. N-Queens
5457    /// https://leetcode.com/problems/n-queens/description/
5458    /// 时间复杂度: O(n^n)
5459    /// 空间复杂度: O(n)
5460    public class Solution {
5461
5462        private boolean[] col;
5463        private boolean[] dia1;
5464        private boolean[] dia2;
5465        private ArrayList<List<String>> res;
5466
5467        public List<List<String>> solveNQueens(int n) {
5468
5469            res = new ArrayList<List<String>>();
5470            col = new boolean[n];
5471            dia1 = new boolean[2 * n - 1];
5472            dia2 = new boolean[2 * n - 1];
5473
5474            LinkedList<Integer> row = new LinkedList<Integer>();
5475            putQueen(n, 0, row);
5476
5477            return res;
5478        }
```

```
5479
5480        // 尝试在一个n皇后问题中, 摆放第index行的皇后位置
5481        private void putQueen(int n, int index, LinkedList<Integer> row){
5482
5483            if(index == n){
5484                res.add(generateBoard(n, row));
5485                return;
5486            }
5487
5488            for(int i = 0 ; i < n ; i ++)
5489                // 尝试将第index行的皇后摆放在第i列
5490                if(!col[i] && !dia1[index + i] && !dia2[index - i + n - 1]){
5491                    row.addLast(i);
5492                    col[i] = true;
5493                    dia1[index + i] = true;
5494                    dia2[index - i + n - 1] = true;
5495                    putQueen(n, index + 1, row);
5496                    col[i] = false;
5497                    dia1[index + i] = false;
5498                    dia2[index - i + n - 1] = false;
5499                    row.removeLast();
5500                }
5501
5502            return;
5503        }
5504
5505        private List<String> generateBoard(int n, LinkedList<Integer> row){
5506
5507            assert row.size() == n;
5508
5509            ArrayList<String> board = new ArrayList<String>();
5510            for(int i = 0 ; i < n ; i ++){
5511                char[] charArray = new char[n];
5512                Arrays.fill(charArray, '.');
5513                charArray[row.get(i)] = 'Q';
5514                board.add(new String(charArray));
5515            }
5516            return board;
5517        }
5518
5519        private static void printBoard(List<String> board){
5520            for(String s: board)
5521                System.out.println(s);
5522            System.out.println();
5523        }
5524
5525        public static void main(String[] args) {
5526
5527            int n = 4;
5528            List<List<String>> res = (new Solution()).solveNQueens(n);
5529            for(List<String> board: res)
5530                printBoard(board);
5531        }
5532    }
5533
5534
5535    \09-Dynamic-Programming\01-Fibonacci\src\Solution1.java
5536
5537    // 递归求斐波那契数列
5538    public class Solution1 {
5539
5540        private int num = 0;
5541
5542        public int fib( int n ){
5543
5544            num ++;
```

```
5545
5546            if( n == 0 )
5547                return 0;
5548
5549            if( n == 1 )
5550                return 1;
5551
5552            return fib(n-1) + fib(n-2);
5553        }
5554
5555        public int getNum(){
5556            return num;
5557        }
5558
5559        public static void main(String[] args) {
5560
5561            int n = 42;
5562
5563            Solution1 solution = new Solution1();
5564            long startTime = System.currentTimeMillis();
5565            int res = solution.fib(n);
5566            long endTime = System.currentTimeMillis();
5567
5568            System.out.println("fib(" + n + ") = " + res);
5569            System.out.println("time : " + (endTime - startTime) + " ms");
5570            System.out.println("run function fib() " + solution.getNum() + " times.");
5571        }
5572    }
5573
5574
5575    \09-Dynamic-Programming\01-Fibonacci\src\Solution2.java
5576
5577    import java.util.Arrays;
5578
5579    // 记忆化搜索
5580    public class Solution2 {
5581
5582        private int num = 0;
5583
5584        public int fib(int n){
5585
5586            int[] memo = new int[n + 1];
5587            Arrays.fill(memo, -1);
5588            return fib(n, memo);
5589        }
5590
5591        private int fib(int n, int[] memo){
5592
5593            num ++;
5594
5595            if(n == 0)
5596                return 0;
5597
5598            if(n == 1)
5599                return 1;
5600
5601            if(memo[n] == -1)
5602                memo[n] = fib(n - 1, memo) + fib(n - 2, memo);
5603
5604            return memo[n];
5605        }
5606
5607        public int getNum(){
5608            return num;
5609        }
5610
```

```
5611        public static void main(String[] args) {
5612
5613            //int n = 42;
5614            int n = 1000; // 注意: 我们使用n = 1000只是为了测试性能, 实际上会溢出
5615                          // 斐波那契额数列是以指数速度上涨的
5616
5617            Solution2 solution = new Solution2();
5618            long startTime = System.currentTimeMillis();
5619            int res = solution.fib(n);
5620            long endTime = System.currentTimeMillis();
5621
5622            System.out.println("fib(" + n + ") = " + res);
5623            System.out.println("time : " + (endTime - startTime) + " ms");
5624            System.out.println("run function fib() " + solution.getNum() + " times.");
5625        }
5626    }
5627
5628
5629    \09-Dynamic-Programming\01-Fibonacci\src\Solution3.java
5630
5631    import java.util.Arrays;
5632
5633    // 动态规划
5634    public class Solution3 {
5635
5636        public int fib(int n){
5637
5638            int[] memo = new int[n + 1];
5639            Arrays.fill(memo, -1);
5640
5641            memo[0] = 0;
5642            memo[1] = 1;
5643            for(int i = 2 ; i <= n ; i ++)
5644                memo[i] = memo[i - 1] + memo[i - 2];
5645
5646            return memo[n];
5647        }
5648
5649        public static void main(String[] args) {
5650
5651            //int n = 42;
5652            int n = 1000; // 注意: 我们使用n = 1000只是为了测试性能, 实际上会溢出
5653                          // 斐波那契额数列是以指数速度上涨的
5654
5655            Solution3 solution = new Solution3();
5656            long startTime = System.currentTimeMillis();
5657            int res = solution.fib(n);
5658            long endTime = System.currentTimeMillis();
5659
5660            System.out.println("fib(" + n + ") = " + res);
5661            System.out.println("time : " + (endTime - startTime) + " ms");
5662        }
5663    }
5664
5665
5666    \09-Dynamic-Programming\02-Climbing-Stairs\src\Solution1.java
5667
5668    import java.util.Arrays;
5669
5670    /**
5671     * Created by liuyubobobo.
5672     */
5673    public class Solution1 {
5674
5675        private int[] memo;
5676
```

```
5677        public int climbStairs(int n) {
5678            memo = new int[n+1];
5679            Arrays.fill(memo, -1);
5680            return calcWays(n);
5681        }
5682
5683        private int calcWays(int n){
5684
5685            if(n == 0 || n == 1)
5686                return 1;
5687
5688            if(memo[n] == -1)
5689                memo[n] = calcWays(n - 1) + calcWays(n - 2);
5690
5691            return memo[n];
5692        }
5693
5694        public static void main(String[] args) {
5695
5696            System.out.println((new Solution1()).climbStairs(10));
5697        }
5698    }
5699
5700
5701    \09-Dynamic-Programming\02-Climbing-Stairs\src\Solution2.java
5702
5703    /// 70. Climbing Stairs
5704    /// https://leetcode.com/problems/climbing-stairs/description/
5705    /// 动态规划
5706    /// 时间复杂度: O(n)
5707    /// 空间复杂度: O(n)
5708    public class Solution2 {
5709
5710        public int climbStairs(int n) {
5711
5712            int[] memo = new int[n + 1];
5713            memo[0] = 1;
5714            memo[1] = 1;
5715            for(int i = 2 ; i <= n ; i ++)
5716                memo[i] = memo[i - 1] + memo[i - 2];
5717            return memo[n];
5718        }
5719
5720        public static void main(String[] args) {
5721
5722            System.out.println((new Solution2()).climbStairs(10));
5723        }
5724    }
5725
5726
5727    \09-Dynamic-Programming\03-Integer-Break\src\Solution1.java
5728
5729    /// 343. Integer Break
5730    /// https://leetcode.com/problems/integer-break/description/
5731    /// 暴力搜索
5732    /// 在Leetcode中提交这个版本的代码会超时! (Time Limit Exceeded)
5733    /// 时间复杂度: O(n^n)
5734    /// 空间复杂度: O(n)
5735    public class Solution1 {
5736
5737        public int integerBreak(int n) {
5738
5739            if(n < 1)
5740                throw new IllegalArgumentException("n should be greater than zero");
5741
5742            return breakInteger(n);
```

```
5743        }
5744
5745        // 将n进行分割(至少分割两部分), 可以获得的最大乘积
5746        private int breakInteger(int n){
5747
5748            if(n == 1)
5749                return 1;
5750
5751            int res = -1;
5752            for(int i = 1 ; i <= n - 1 ; i ++)
5753                res = max3(res, i * (n - i), i * breakInteger(n - i));
5754            return res;
5755        }
5756
5757        private int max3(int a, int b, int c){
5758            return Math.max(a, Math.max(b, c));
5759        }
5760
5761        public static void main(String[] args) {
5762
5763            System.out.println((new Solution1()).integerBreak(2));
5764            System.out.println((new Solution1()).integerBreak(10));
5765        }
5766    }
5767
5768
5769    \09-Dynamic-Programming\03-Integer-Break\src\Solution2.java
5770
5771    import java.util.Arrays;
5772
5773    /// 343. Integer Break
5774    /// https://leetcode.com/problems/integer-break/description/
5775    /// 记忆化搜索
5776    /// 时间复杂度: O(n^2)
5777    /// 空间复杂度: O(n)
5778    public class Solution2 {
5779
5780        private int[] memo;
5781
5782        public int integerBreak(int n) {
5783
5784            if(n < 1)
5785                throw new IllegalArgumentException("n should be greater than zero");
5786
5787            memo = new int[n+1];
5788            Arrays.fill(memo, -1);
5789
5790            return breakInteger(n);
5791        }
5792
5793        // 将n进行分割(至少分割两部分), 可以获得的最大乘积
5794        private int breakInteger(int n){
5795
5796            if(n == 1)
5797                return 1;
5798
5799            if(memo[n] != -1)
5800                return memo[n];
5801
5802            int res = -1;
5803            for(int i = 1 ; i <= n - 1 ; i ++)
5804                res = max3(res, i * (n - i) , i * breakInteger(n - i));
5805            memo[n] = res;
5806            return res;
5807        }
5808
```

```
5809        private int max3(int a, int b, int c){
5810            return Math.max(a, Math.max(b, c));
5811        }
5812
5813        public static void main(String[] args) {
5814
5815            System.out.println((new Solution2()).integerBreak(2));
5816            System.out.println((new Solution2()).integerBreak(10));
5817        }
5818    }
5819
5820
5821    \09-Dynamic-Programming\03-Integer-Break\src\Solution3.java
5822
5823    /// 343. Integer Break
5824    /// https://leetcode.com/problems/integer-break/description/
5825    /// 动态规划
5826    /// 时间复杂度: O(n^2)
5827    /// 空间复杂度: O(n)
5828    public class Solution3 {
5829
5830        public int integerBreak(int n) {
5831
5832            if(n < 1)
5833                throw new IllegalArgumentException("n should be greater than zero");
5834
5835            int[] memo = new int[n+1];
5836            memo[1] = 1;
5837            for(int i = 2 ; i <= n ; i ++)
5838                // 求解memo[i]
5839                for(int j = 1 ; j <= i - 1 ; j ++)
5840                    memo[i] = max3(memo[i], j * (i - j), j * memo[i - j]);
5841
5842            return memo[n];
5843        }
5844
5845        private int max3(int a, int b, int c){
5846            return Math.max(a, Math.max(b, c));
5847        }
5848
5849        public static void main(String[] args) {
5850
5851            System.out.println((new Solution3()).integerBreak(2));
5852            System.out.println((new Solution3()).integerBreak(10));
5853        }
5854    }
5855
5856
5857    \09-Dynamic-Programming\04-House-Robber\src\Solution1.java
5858
5859    import java.util.Arrays;
5860
5861    /// 198. House Robber
5862    /// https://leetcode.com/problems/house-robber/description/
5863    /// 记忆化搜索
5864    /// 时间复杂度: O(n^2)
5865    /// 空间复杂度: O(n)
5866    public class Solution1 {
5867
5868        // memo[i] 表示考虑抢劫 nums[i...n) 所能获得的最大收益
5869        private int[] memo;
5870
5871        public int rob(int[] nums) {
5872            memo = new int[nums.length];
5873            Arrays.fill(memo, -1);
5874            return tryRob(nums, 0);
```

```
5875        }
5876
5877        // 考虑抢劫nums[index...nums.size())这个范围的所有房子
5878        private int tryRob(int[] nums, int index){
5879
5880            if(index >= nums.length)
5881                return 0;
5882
5883            if(memo[index] != -1)
5884                return memo[index];
5885
5886            int res = 0;
5887            for(int i = index ; i < nums.length ; i ++)
5888                res = Math.max(res, nums[i] + tryRob(nums, i + 2));
5889            memo[index] = res;
5890            return res;
5891        }
5892
5893        public static void main(String[] args) {
5894
5895            int nums[] = {2, 1};
5896            System.out.println((new Solution1()).rob(nums));
5897        }
5898    }
5899
5900
5901    \09-Dynamic-Programming\04-House-Robber\src\Solution2.java
5902
5903    import java.util.Arrays;
5904
5905    /// 198. House Robber
5906    /// https://leetcode.com/problems/house-robber/description/
5907    /// 动态规划
5908    /// 时间复杂度: O(n^2)
5909    /// 空间复杂度: O(n)
5910    public class Solution2 {
5911
5912        public int rob(int[] nums) {
5913
5914            int n = nums.length;
5915            if(n == 0)
5916                return 0;
5917
5918            // memo[i] 表示考虑抢劫 nums[i...n) 所能获得的最大收益
5919            int[] memo = new int[nums.length];
5920            memo[n - 1] = nums[n - 1];
5921            for(int i = n - 2 ; i >= 0 ; i --)
5922                for (int j = i; j < n; j++)
5923                    memo[i] = Math.max( memo[i],
5924                                        nums[j] + (j + 2 < n ? memo[j + 2] : 0));
5925
5926            return memo[0];
5927        }
5928
5929        public static void main(String[] args) {
5930
5931            int nums[] = {2, 1};
5932            System.out.println((new Solution2()).rob(nums));
5933        }
5934    }
5935
5936
5937    \09-Dynamic-Programming\04-House-Robber\src\Solution3.java
5938
5939    import java.util.Arrays;
5940
```

```java
/// 198. House Robber
/// https://leetcode.com/problems/house-robber/description/
/// 记忆化搜索, 改变状态定义
/// 时间复杂度: O(n^2)
/// 空间复杂度: O(n)
public class Solution3 {

    // memo[i] 表示考虑抢劫 nums[0...i] 所能获得的最大收益
    private int[] memo;

    public int rob(int[] nums) {
        memo = new int[nums.length];
        Arrays.fill(memo, -1);
        return tryRob(nums, nums.length - 1);
    }

    // 考虑抢劫nums[0...index]这个范围的所有房子
    private int tryRob(int[] nums, int index){

        if(index < 0)
            return 0;

        if(memo[index] != -1)
            return memo[index];

        int res = 0;
        for(int i = 0 ; i <= index ; i ++)
            res = Math.max(res, nums[i] + tryRob(nums, i - 2));
        memo[index] = res;
        return res;
    }

    public static void main(String[] args) {

        int nums[] = {2, 1};
        System.out.println((new Solution3()).rob(nums));
    }
}


\09-Dynamic-Programming\04-House-Robber\src\Solution4.java

/// 198. House Robber
/// https://leetcode.com/problems/house-robber/description/
/// 动态规划, 改变状态定义
/// 时间复杂度: O(n^2)
/// 空间复杂度: O(n)
public class Solution4 {

    public int rob(int[] nums) {

        int n = nums.length;
        if(n == 0)
            return 0;

        // memo[i] 表示考虑抢劫 nums[0...i] 所能获得的最大收益
        int[] memo = new int[nums.length];
        memo[0] = nums[0];
        for(int i = 1 ; i < n ; i ++)
            for (int j = i; j >= 0; j--)
                memo[i] = Math.max(memo[i],
                                   nums[j] + (j - 2 >= 0 ? memo[j - 2] : 0));

        return memo[n-1];
    }

```

```
6007        public static void main(String[] args) {
6008
6009            int nums[] = {2, 1};
6010            System.out.println((new Solution4()).rob(nums));
6011        }
6012    }
6013
6014
6015    \09-Dynamic-Programming\04-House-Robber\src\Solution5.java
6016
6017    import java.util.Arrays;
6018
6019    /// 198. House Robber
6020    /// https://leetcode.com/problems/house-robber/description/
6021    /// 记忆化搜索, 优化状态转移
6022    /// 时间复杂度: O(n)
6023    /// 空间复杂度: O(n)
6024    public class Solution5 {
6025
6026        // memo[i] 表示考虑抢劫 nums[i...n) 所能获得的最大收益
6027        private int[] memo;
6028
6029        public int rob(int[] nums) {
6030            memo = new int[nums.length];
6031            Arrays.fill(memo, -1);
6032            return tryRob(nums, 0);
6033        }
6034
6035        // 考虑抢劫nums[index...nums.size())这个范围的所有房子
6036        private int tryRob(int[] nums, int index){
6037
6038            if(index >= nums.length)
6039                return 0;
6040
6041            if(memo[index] != -1)
6042                return memo[index];
6043
6044            // 或者当前房子放弃, 从下一个房子开始考虑
6045            // 或者抢劫当前的房子, 从i+2以后的房子开始考虑
6046            return memo[index] =
6047                    Math.max(tryRob(nums, index + 1),
6048                             nums[index] + tryRob(nums, index + 2));
6049        }
6050
6051        public static void main(String[] args) {
6052
6053            int nums[] = {2, 1};
6054            System.out.println((new Solution5()).rob(nums));
6055        }
6056    }
6057
6058
6059    \09-Dynamic-Programming\04-House-Robber\src\Solution6.java
6060
6061    import java.util.Arrays;
6062
6063    /// 198. House Robber
6064    /// https://leetcode.com/problems/house-robber/description/
6065    /// 动态规划, 优化状态转移
6066    /// 时间复杂度: O(n)
6067    /// 空间复杂度: O(n)
6068    public class Solution6 {
6069
6070        public int rob(int[] nums) {
6071
6072            int n = nums.length;
```

```
6073            if(n == 0)
6074                return 0;
6075
6076            // memo[i] 表示考虑抢劫 nums[i...n) 所能获得的最大收益
6077            int[] memo = new int[nums.length];
6078            memo[n - 1] = nums[n - 1];
6079            for(int i = n - 2 ; i >= 0 ; i --)
6080                // 或者当前房子放弃, 从下一个房子开始考虑
6081                // 或者抢劫当前的房子, 从i+2以后的房子开始考虑
6082                memo[i] = Math.max(memo[i + 1],
6083                                   nums[i] + (i + 2 < n ? memo[i + 2] : 0));
6084
6085            return memo[0];
6086        }
6087
6088        public static void main(String[] args) {
6089
6090            int nums[] = {2, 1};
6091            System.out.println((new Solution6()).rob(nums));
6092        }
6093    }
6094
6095
6096    \09-Dynamic-Programming\04-House-Robber\src\Solution7.java
6097
6098    import java.util.Arrays;
6099
6100    /// 198. House Robber
6101    /// https://leetcode.com/problems/house-robber/description/
6102    /// 记忆化搜索, 改变状态定义, 优化转移方程
6103    /// 时间复杂度: O(n)
6104    /// 空间复杂度: O(n)
6105    public class Solution7 {
6106
6107        // memo[i] 表示考虑抢劫 nums[0...i] 所能获得的最大收益
6108        private int[] memo;
6109
6110        public int rob(int[] nums) {
6111            memo = new int[nums.length];
6112            Arrays.fill(memo, -1);
6113            return tryRob(nums, nums.length - 1);
6114        }
6115
6116        // 考虑抢劫nums[0...index]这个范围的所有房子
6117        private int tryRob(int[] nums, int index){
6118
6119            if(index < 0)
6120                return 0;
6121
6122            if(memo[index] != -1)
6123                return memo[index];
6124
6125            // 或者当前房子放弃, 考虑[0...index-1]的所有房子
6126            // 或者抢劫当前的房子, 考虑[0...index-2]的所有房子
6127            return memo[index] =
6128                    Math.max(tryRob(nums, index - 1),
6129                             nums[index] + tryRob(nums, index - 2));
6130        }
6131
6132        public static void main(String[] args) {
6133
6134            int nums[] = {2, 1};
6135            System.out.println((new Solution7()).rob(nums));
6136        }
6137    }
6138
```

```
6139
6140    \09-Dynamic-Programming\04-House-Robber\src\Solution8.java
6141
6142    /// 198. House Robber
6143    /// https://leetcode.com/problems/house-robber/description/
6144    /// 动态规划, 改变状态定义, 优化转移方程
6145    /// 时间复杂度: O(n)
6146    /// 空间复杂度: O(n)
6147    public class Solution8 {
6148
6149        public int rob(int[] nums) {
6150
6151            int n = nums.length;
6152            if(n == 0)
6153                return 0;
6154
6155            // memo[i] 表示考虑抢劫 nums[0...i] 所能获得的最大收益
6156            int[] memo = new int[nums.length];
6157            memo[0] = nums[0];
6158            for(int i = 1 ; i < n ; i ++)
6159                memo[i] = Math.max(memo[i - 1],
6160                                   nums[i] + (i - 2 >= 0 ? memo[i - 2] : 0));
6161
6162            return memo[n-1];
6163        }
6164
6165        public static void main(String[] args) {
6166
6167            int nums[] = {2, 1};
6168            System.out.println((new Solution8()).rob(nums));
6169        }
6170    }
6171
6172
6173    \09-Dynamic-Programming\05-0-1-knapsack\src\Solution1.java
6174
6175    /// 背包问题
6176    /// 记忆化搜索
6177    /// 时间复杂度: O(n * C) 其中n为物品个数; C为背包容积
6178    /// 空间复杂度: O(n * C)
6179    public class Solution1 {
6180
6181        private int[][] memo;
6182
6183        public int knapsack01(int[] w, int[] v, int C){
6184
6185            if(w == null || v == null || w.length != v.length)
6186                throw new IllegalArgumentException("Invalid w or v");
6187
6188            if(C < 0)
6189                throw new IllegalArgumentException("C must be greater or equal to zero.");
6190
6191            int n = w.length;
6192            if(n == 0 || C == 0)
6193                return 0;
6194
6195            memo = new int[n][C + 1];
6196            return bestValue(w, v, n - 1, C);
6197        }
6198
6199        // 用 [0...index]的物品,填充容积为c的背包的最大价值
6200        private int bestValue(int[] w, int[] v, int index, int c){
6201
6202            if(c <= 0 || index < 0)
6203                return 0;
6204
```

```
6205            if(memo[index][c] != -1)
6206                return memo[index][c];
6207
6208            int res = bestValue(w, v, index-1, c);
6209            if(c >= w[index])
6210                res = Math.max(res, v[index] + bestValue(w, v, index - 1, c - w[index]));
6211
6212            return memo[index][c] = res;
6213        }
6214
6215        public static void main(String[] args) {
6216
6217        }
6218
6219    }
6220
6221
6222    \09-Dynamic-Programming\05-0-1-knapsack\src\Solution2.java
6223
6224    /// 背包问题
6225    /// 动态规划
6226    /// 时间复杂度: O(n * C) 其中n为物品个数; C为背包容积
6227    /// 空间复杂度: O(n * C)
6228    public class Solution2 {
6229
6230        public int knapsack01(int[] w, int[] v, int C){
6231
6232            if(w == null || v == null || w.length != v.length)
6233                throw new IllegalArgumentException("Invalid w or v");
6234
6235            if(C < 0)
6236                throw new IllegalArgumentException("C must be greater or equal to zero.");
6237
6238            int n = w.length;
6239            if(n == 0 || C == 0)
6240                return 0;
6241
6242            int[][] memo = new int[n][C + 1];
6243
6244            for(int j = 0 ; j <= C ; j ++)
6245                memo[0][j] = (j >= w[0] ? v[0] : 0 );
6246
6247            for(int i = 1 ; i < n ; i ++)
6248                for(int j = 0 ; j <= C ; j ++){
6249                    memo[i][j] = memo[i-1][j];
6250                    if(j >= w[i])
6251                        memo[i][j] = Math.max(memo[i][j], v[i] + memo[i - 1][j - w[i]]);
6252                }
6253
6254            return memo[n - 1][C];
6255        }
6256
6257        public static void main(String[] args) {
6258
6259        }
6260    }
6261
6262
6263    \09-Dynamic-Programming\06-0-1-knapsack-optimized\src\Solution1.java
6264
6265    /// 背包问题
6266    /// 动态规划改进: 滚动数组
6267    /// 时间复杂度: O(n * C) 其中n为物品个数; C为背包容积
6268    /// 空间复杂度: O(C), 实际使用了2*C的额外空间
6269    public class Solution1 {
6270
```

```
6271        public int knapsack01(int[] w, int[] v, int C){
6272
6273            if(w == null || v == null || w.length != v.length)
6274                throw new IllegalArgumentException("Invalid w or v");
6275
6276            if(C < 0)
6277                throw new IllegalArgumentException("C must be greater or equal to zero.");
6278
6279            int n = w.length;
6280            if(n == 0 || C == 0)
6281                return 0;
6282
6283            int[][] memo = new int[2][C + 1];
6284
6285            for(int j = 0 ; j <= C ; j ++)
6286                memo[0][j] = (j >= w[0] ? v[0] : 0);
6287
6288            for(int i = 1 ; i < n ; i ++)
6289                for(int j = 0 ; j <= C ; j ++){
6290                    memo[i % 2][j] = memo[(i-1) % 2][j];
6291                    if(j >= w[i])
6292                        memo[i % 2][j] = Math.max(memo[i % 2][j], v[i] + memo[(i-1) % 2][j - w[i]]);
6293                }
6294
6295            return memo[(n-1) % 2][C];
6296        }
6297
6298        public static void main(String[] args) {
6299
6300        }
6301    }
6302
6303
6304    \09-Dynamic-Programming\06-0-1-knapsack-optimized\src\Solution2.java
6305
6306    /// 背包问题
6307    /// 动态规划改进
6308    /// 时间复杂度: O(n * C) 其中n为物品个数; C为背包容积
6309    /// 空间复杂度: O(C), 只使用了C的额外空间
6310    public class Solution2 {
6311
6312        public int knapsack01(int[] w, int[] v, int C){
6313
6314            if(w == null || v == null || w.length != v.length)
6315                throw new IllegalArgumentException("Invalid w or v");
6316
6317            if(C < 0)
6318                throw new IllegalArgumentException("C must be greater or equal to zero.");
6319
6320            int n = w.length;
6321            if(n == 0 || C == 0)
6322                return 0;
6323
6324            int[] memo = new int[C+1];
6325
6326            for(int j = 0 ; j <= C ; j ++)
6327                memo[j] = (j >= w[0] ? v[0] : 0);
6328
6329            for(int i = 1 ; i < n ; i ++)
6330                for(int j = C ; j >= w[i] ; j --)
6331                    memo[j] = Math.max(memo[j], v[i] + memo[j - w[i]]);
6332
6333            return memo[C];
6334        }
6335
6336        public static void main(String[] args) {
```

```
6337
6338        }
6339    }
6340
6341
6342    \09-Dynamic-Programming\07-Partition-Equal-Subset-Sum\src\Solution1.java
6343
6344    import java.util.Arrays;
6345
6346    /// 416. Partition Equal Subset Sum
6347    /// https://leetcode.com/problems/partition-equal-subset-sum/description/
6348    /// 记忆化搜索
6349    /// 时间复杂度: O(len(nums) * O(sum(nums)))
6350    /// 空间复杂度: O(len(nums) * O(sum(nums)))
6351    public class Solution1 {
6352
6353        // memo[i][c] 表示使用索引为[0...i]的这些元素,是否可以完全填充一个容量为c的背包
6354        // -1 表示为未计算; 0 表示不可以填充; 1 表示可以填充
6355        private int[][] memo;
6356
6357        public boolean canPartition(int[] nums) {
6358
6359            int sum = 0;
6360            for(int i = 0 ; i < nums.length ; i ++){
6361                if(nums[i] <= 0)
6362                    throw new IllegalArgumentException("numbers in nums must be greater than zero.");
6363                sum += nums[i];
6364            }
6365
6366            if(sum % 2 == 1)
6367                return false;
6368
6369            memo = new int[nums.length][sum / 2 + 1];
6370            for(int i = 0 ; i < nums.length ; i ++)
6371                Arrays.fill(memo[i], -1);
6372            return tryPartition(nums, nums.length - 1, sum / 2);
6373        }
6374
6375        // 使用nums[0...index], 是否可以完全填充一个容量为sum的背包
6376        private boolean tryPartition(int[] nums, int index, int sum){
6377
6378            if(sum == 0)
6379                return true;
6380
6381            if(sum < 0 || index < 0)
6382                return false;
6383
6384            if(memo[index][sum] != -1)
6385                return memo[index][sum] == 1;
6386
6387            memo[index][sum] = (tryPartition(nums, index - 1, sum) ||
6388                    tryPartition(nums, index - 1, sum - nums[index])) ? 1 : 0;
6389
6390            return memo[index][sum] == 1;
6391        }
6392
6393        private static void printBool(boolean res){
6394            System.out.println(res ? "True" : "False");
6395        }
6396
6397        public static void main(String[] args) {
6398
6399            int[] nums1 = {1, 5, 11, 5};
6400            printBool((new Solution1()).canPartition(nums1));
6401
6402            int[] nums2 = {1, 2, 3, 5};
```

```
6403            printBool((new Solution1()).canPartition(nums2));
6404        }
6405    }
6406
6407
6408    \09-Dynamic-Programming\07-Partition-Equal-Subset-Sum\src\Solution2.java
6409
6410    import java.util.Arrays;
6411
6412    /// 416. Partition Equal Subset Sum
6413    /// https://leetcode.com/problems/partition-equal-subset-sum/description/
6414    /// 动态规划
6415    /// 时间复杂度: O(len(nums) * O(sum(nums)))
6416    /// 空间复杂度: O(len(nums) * O(sum(nums)))
6417    public class Solution2 {
6418
6419        public boolean canPartition(int[] nums) {
6420
6421            int sum = 0;
6422            for(int i = 0 ; i < nums.length ; i ++){
6423                if(nums[i] <= 0)
6424                    throw new IllegalArgumentException("numbers in nums must be greater than zero.");
6425                sum += nums[i];
6426            }
6427
6428            if(sum % 2 == 1)
6429                return false;
6430
6431            int n = nums.length;
6432            int C = sum / 2;
6433
6434            boolean[] memo = new boolean[C + 1];
6435            for(int i = 0 ; i <= C ; i ++)
6436                memo[i] = (nums[0] == i);
6437
6438            for(int i = 1 ; i < n ; i ++)
6439                for(int j = C; j >= nums[i] ; j --)
6440                    memo[j] = memo[j] || memo[j - nums[i]];
6441
6442            return memo[C];
6443        }
6444
6445        private static void printBool(boolean res){
6446            System.out.println(res ? "True" : "False");
6447        }
6448
6449        public static void main(String[] args) {
6450
6451            int[] nums1 = {1, 5, 11, 5};
6452            printBool((new Solution2()).canPartition(nums1));
6453
6454            int[] nums2 = {1, 2, 3, 5};
6455            printBool((new Solution2()).canPartition(nums2));
6456        }
6457    }
6458
6459
6460    \09-Dynamic-Programming\08-Longest-Increasing-Subsequence\src\Solution1.java
6461
6462    import java.util.Arrays;
6463
6464    /// 300. Longest Increasing Subsequence
6465    /// https://leetcode.com/problems/longest-increasing-subsequence/description/
6466    /// 记忆化搜索
6467    /// 时间复杂度: O(n^2)
6468    /// 空间复杂度: O(n)
```

```java
public class Solution1 {

    private int[] memo;

    public int lengthOfLIS(int[] nums) {

        if(nums.length == 0)
            return 0;

        memo = new int[nums.length];
        Arrays.fill(memo, -1);
        int res = 1;
        for(int i = 0 ; i < nums.length ; i ++)
            res = Math.max(res, getMaxLength(nums, i));

        return res;
    }

    // 以 nums[index] 为结尾的最长上升子序列的长度
    private int getMaxLength(int[] nums, int index){

        if(memo[index] != -1)
            return memo[index];

        int res = 1;
        for(int i = 0 ; i <= index-1 ; i ++)
            if(nums[index] > nums[i])
                res = Math.max(res, 1 + getMaxLength(nums, i));

        return memo[index] = res;
    }

    public static void main(String[] args) {

        int nums1[] = {10, 9, 2, 5, 3, 7, 101, 18};
        System.out.println((new Solution1()).lengthOfLIS(nums1));
        // 4

        // ---

        int nums2[] = {4, 10, 4, 3, 8, 9};
        System.out.println((new Solution1()).lengthOfLIS(nums2));
        // 3

        // ---

        int nums3[] = {2, 2};
        System.out.println((new Solution1()).lengthOfLIS(nums3));
        // 1

        // ---

        int nums4[] = {1, 3, 6, 7, 9, 4, 10, 5, 6};
        System.out.println((new Solution1()).lengthOfLIS(nums4));
        // 6
    }
}
```

\09-Dynamic-Programming\08-Longest-Increasing-Subsequence\src\Solution2.java

```java
import java.util.Arrays;

/// 300. Longest Increasing Subsequence
/// https://leetcode.com/problems/longest-increasing-subsequence/description/
/// 记忆化搜索
```

```java
/// 时间复杂度: O(n^2)
/// 空间复杂度: O(n)
public class Solution2 {

    public int lengthOfLIS(int[] nums) {

        if(nums.length == 0)
            return 0;

        // memo[i] 表示以 nums[i] 为结尾的最长上升子序列的长度
        int memo[] = new int[nums.length];
        Arrays.fill(memo, 1);
        for(int i = 1 ; i < nums.length ; i ++)
            for(int j = 0 ; j < i ; j ++)
                if(nums[i] > nums[j])
                    memo[i] = Math.max(memo[i], 1 + memo[j]);

        int res = memo[0];
        for(int i = 1 ; i < nums.length ; i ++)
            res = Math.max(res, memo[i]);

        return res;
    }

    public static void main(String[] args) {

        int nums1[] = {10, 9, 2, 5, 3, 7, 101, 18};
        System.out.println((new Solution2()).lengthOfLIS(nums1));
        // 4

        // ---

        int nums2[] = {4, 10, 4, 3, 8, 9};
        System.out.println((new Solution2()).lengthOfLIS(nums2));
        // 3

        // ---

        int nums3[] = {2, 2};
        System.out.println((new Solution2()).lengthOfLIS(nums3));
        // 1

        // ---

        int nums4[] = {1, 3, 6, 7, 9, 4, 10, 5, 6};
        System.out.println((new Solution2()).lengthOfLIS(nums4));
        // 6
    }
}


\09-Dynamic-Programming\09-Longest-Common-Subsequence\src\LCS1.java

import java.util.Arrays;

/// LCS问题
/// 动态规划
/// 时间复杂度: O(len(s1)*len(s2))
/// 空间复杂度: O(len(s1)*len(s2))
public class LCS1 {

    private int[][] memo;

    public String lcs(String s1, String s2){

        if(s1 == null || s2 == null)
```

```
6601                throw new IllegalArgumentException("s1 and s2 can not be null.");
6602
6603            if(s1.length() == 0 || s2.length() == 0)
6604                return "";
6605
6606            memo = new int[s1.length()][s2.length()];
6607            for(int i = 0 ; i < s1.length() ; i ++)
6608                Arrays.fill(memo[i], -1);
6609
6610            lcs(s1, s2, s1.length() - 1, s2.length() - 1);
6611            return getLCS(s1, s2);
6612        }
6613
6614        // 求s1[0...m]和s2[0...n]的最长公共子序列的长度值
6615        private int lcs(String s1, String s2, int m, int n){
6616
6617            if(m < 0 || n < 0)
6618                return 0;
6619
6620            if(memo[m][n] != -1)
6621                return memo[m][n];
6622
6623            int res = 0;
6624            if(s1.charAt(m) == s2.charAt(n))
6625                res = 1 + lcs(s1, s2, m - 1, n - 1);
6626            else
6627                res = Math.max(lcs(s1, s2, m - 1, n),
6628                               lcs(s1, s2, m, n - 1));
6629
6630            memo[m][n] = res;
6631            return res;
6632        }
6633
6634        // 通过memo反向求解s1和s2的最长公共子序列
6635        private String getLCS(String s1, String s2){
6636
6637            int m = s1.length() - 1;
6638            int n = s2.length() - 1;
6639
6640            StringBuilder res = new StringBuilder("");
6641            while(m >= 0 && n >= 0)
6642                if(s1.charAt(m) == s2.charAt(n)){
6643                    res = res.insert(0, s1.charAt(m));
6644                    m --;
6645                    n --;
6646                }
6647                else if(m == 0)
6648                    n --;
6649                else if(n == 0)
6650                    m --;
6651                else{
6652                    if(memo[m-1][n] > memo[m][n-1])
6653                        m --;
6654                    else
6655                        n --;
6656                }
6657
6658            return res.toString();
6659        }
6660
6661        public static void main(String[] args) {
6662
6663            String s1 = "ABCDGH";
6664            String s2 = "AEDFHR";
6665            System.out.println((new LCS1()).lcs(s1, s2));
6666
```

```
6667            s1 = "AAACCGTGAGTTATTCGTTCTAGAA";
6668            s2 = "CACCCCTAAGGTACCTTTGGTTC";
6669            System.out.println((new LCS1()).lcs(s1, s2));
6670        }
6671    }
6672
6673
6674    \09-Dynamic-Programming\09-Longest-Common-Subsequence\src\LCS2.java
6675
6676    /// LCS问题
6677    /// 动态规划
6678    /// 时间复杂度: O(len(s1)*len(s2))
6679    /// 空间复杂度: O(len(s1)*len(s2))
6680    public class LCS2 {
6681
6682        public String lcs(String s1, String s2){
6683
6684            int m = s1.length();
6685            int n = s2.length();
6686
6687            // 对memo的第0行和第0列进行初始化
6688            int[][] memo = new int[m][n];
6689            for(int j = 0 ; j < n ; j ++)
6690                if(s1.charAt(0) == s2.charAt(j)){
6691                    for(int k = j ; k < n ; k ++)
6692                        memo[0][k] = 1;
6693                    break;
6694                }
6695
6696            for(int i = 0 ; i < m ; i ++)
6697                if(s1.charAt(i) == s2.charAt(0)) {
6698                    for(int k = i ; k < m ; k ++)
6699                        memo[k][0] = 1;
6700                    break;
6701                }
6702
6703            // 动态规划的过程
6704            for(int i = 1 ; i < m ; i ++)
6705                for(int j = 1 ; j < n ; j ++)
6706                    if(s1.charAt(i) == s2.charAt(j))
6707                        memo[i][j] = 1 + memo[i-1][j-1];
6708                    else
6709                        memo[i][j] = Math.max(memo[i-1][j], memo[i][j-1]);
6710
6711            // 通过memo反向求解s1和s2的最长公共子序列
6712            m = s1.length() - 1;
6713            n = s2.length() - 1;
6714            StringBuilder res = new StringBuilder("");
6715            while(m >= 0 && n >= 0)
6716                if(s1.charAt(m) == s2.charAt(n)){
6717                    res.insert(0, s1.charAt(m));
6718                    m --;
6719                    n --;
6720                }
6721                else if(m == 0)
6722                    n --;
6723                else if(n == 0)
6724                    m --;
6725                else{
6726                    if(memo[m-1][n] > memo[m][n-1])
6727                        m --;
6728                    else
6729                        n --;
6730                }
6731
6732            return res.toString();
```

```
6733        }
6734
6735        public static void main(String[] args) {
6736
6737            String s1 = "ABCDGH";
6738            String s2 = "AEDFHR";
6739            System.out.println((new LCS2()).lcs(s1, s2));
6740
6741            s1 = "AAACCGTGAGTTATTCGTTCTAGAA";
6742            s2 = "CACCCCTAAGGTACCTTTGGTTC";
6743            System.out.println((new LCS2()).lcs(s1, s2));
6744        }
6745    }
6746
6747
6748    \09-Dynamic-Programming\09-Longest-Common-Subsequence\src\LCS3.java
6749
6750    /// LCS问题
6751    /// 动态规划, 躲避边界条件
6752    /// 时间复杂度: O(len(s1)*len(s2))
6753    /// 空间复杂度: O(len(s1)*len(s2))
6754    public class LCS3 {
6755
6756        public String lcs(String s1, String s2){
6757
6758            int m = s1.length();
6759            int n = s2.length();
6760
6761            // memo 是 (m + 1) * (n + 1) 的动态规划表格
6762            // memo[i][j] 表示s1的前i个字符和s2前j个字符的最长公共子序列的长度
6763            // 其中memo[0][j] 表示s1取空字符串时, 和s2的前j个字符作比较
6764            // memo[i][0] 表示s2取空字符串时, 和s1的前i个字符作比较
6765            // 所以, memo[0][j] 和 memo[i][0] 均取0
6766            // 我们不需要对memo进行单独的边界条件处理 :-)
6767            int[][] memo = new int[m + 1][n + 1];
6768
6769            // 动态规划的过程
6770            // 注意, 由于动态规划状态的转变, 下面的i和j可以取到m和n
6771            for(int i = 1 ; i <= m ; i ++)
6772                for(int j = 1 ; j <= n ; j ++)
6773                    if(s1.charAt(i - 1) == s2.charAt(j - 1))
6774                        memo[i][j] = 1 + memo[i - 1][j - 1];
6775                    else
6776                        memo[i][j] = Math.max(memo[i - 1][j], memo[i][j - 1]);
6777
6778            // 通过memo反向求解s1和s2的最长公共子序列
6779            m = s1.length();
6780            n = s2.length();
6781            StringBuilder res = new StringBuilder("");
6782            while(m > 0 && n > 0)
6783                if(s1.charAt(m - 1) == s2.charAt(n - 1)){
6784                    res.insert(0, s1.charAt(m - 1));
6785                    m --;
6786                    n --;
6787                }
6788                else if(memo[m - 1][n] > memo[m][n - 1])
6789                    m --;
6790                else
6791                    n --;
6792
6793            return res.toString();
6794        }
6795
6796        public static void main(String[] args) {
6797
6798            String s1 = "ABCDGH";
```

```
6799            String s2 = "AEDFHR";
6800            System.out.println((new LCS3()).lcs(s1, s2));
6801
6802            s1 = "AAACCGTGAGTTATTCGTTCTAGAA";
6803            s2 = "CACCCCTAAGGTACCTTTGGTTC";
6804            System.out.println((new LCS3()).lcs(s1, s2));
6805        }
6806    }
6807
6808
6809    \09-Dynamic-Programming\Optional-01-More-about-Fibonacci\src\Solution1.java
6810
6811    /// 70. Climbing Stairs
6812    /// https://leetcode.com/problems/climbing-stairs/description/
6813    ///
6814    /// 在这一章的学习中, 我们看到了, 70号问题本质就是求斐波那契数
6815    /// 只不过 climbStairs(n) 的答案, 对应第 n+1 个斐波那契数
6816    /// 其中 f0 = 0, f(1) = 1, f(2) = 1, f(3) = 2...
6817    /// 首先, 我们可以非常简单的使用O(1)的空间求出斐波那契数
6818    /// 这个对空间的优化和我们在这个课程中所介绍的背包问题的空间优化, 其实是类似的思想
6819    /// 我们对背包问题的空间优化, 从O(n^2)优化到了O(n)
6820    /// 我们对斐波那契问题的优化,可以从O(n)优化到O(1)
6821    /// 依靠的依然是, 求第n个斐波那契数, 我们只需要n-1和n-2两个斐波那契数,
6822    /// 更小的斐波那契数不需要一直保存。
6823    ///
6824    /// 时间复杂度: O(n)
6825    /// 空间复杂度: O(1)
6826    public class Solution1 {
6827
6828        public int climbStairs(int n) {
6829
6830            if(n <= 0)
6831                throw new IllegalArgumentException("n must be greater than zero");
6832
6833            if(n == 1)
6834                return 1;
6835
6836            int prev = 1, cur = 1;
6837            for(int i = 3 ; i <= n + 1; i ++){
6838                int f = cur + prev;
6839                prev = cur;
6840                cur = f;
6841            }
6842            return cur;
6843        }
6844
6845        public static void main(String[] args) {
6846
6847            System.out.println((new Solution1()).climbStairs(10));
6848        }
6849    }
6850
6851
6852    \09-Dynamic-Programming\Optional-01-More-about-Fibonacci\src\Solution2.java
6853
6854    /// 70. Climbing Stairs
6855    /// https://leetcode.com/problems/climbing-stairs/description/
6856    ///
6857    /// 斐波那契数可以根据一个特殊矩阵的幂的形式求出。
6858    /// | F(n+1) F(n)   | = | 1 1 |^n
6859    /// | F(n)   F(n-1) |   | 1 0 |
6860    /// 幂运算可以使用分治法, 优化为O(logn)的复杂度
6861    /// 具体该方法的证明, 有兴趣的同学可以自行在互联网上搜索学习。
6862    ///
6863    /// 时间复杂度: O(logn)
6864    /// 空间复杂度: O(1)
```

```
6865    public class Solution2 {
6866
6867        public int climbStairs(int n) {
6868
6869            if(n <= 0)
6870                throw new IllegalArgumentException("n must be greater than zero");
6871
6872            if(n == 1)
6873                return 1;
6874
6875            int[][] base = {{1, 1}, {1, 0}};
6876            return matrix_pow(base, n)[0][0];
6877        }
6878
6879        private int[][] matrix_pow(int[][] m, int n){
6880
6881            if(n == 1)
6882                return m;
6883
6884            int[][] t = matrix_pow(m, n / 2);
6885            int[][] res = matrix_multiply(t, t);
6886            if(n % 2 == 1)
6887                return matrix_multiply(res, m);
6888            return res;
6889        }
6890
6891        int[][] matrix_multiply(int[][] m1, int[][] m2){
6892            int[][] res = new int[2][2];
6893            res[0][0] = m1[0][0] * m2[0][0] + m1[0][1] * m2[1][0];
6894            res[0][1] = m1[0][0] * m2[0][1] + m1[0][1] * m2[1][1];
6895            res[1][0] = m1[1][0] * m2[0][0] + m1[1][1] * m2[1][0];
6896            res[1][1] = m1[1][0] * m2[0][1] + m1[1][1] * m2[1][1];
6897            return res;
6898        }
6899
6900        public static void main(String[] args) {
6901
6902            System.out.println((new Solution2()).climbStairs(10));
6903        }
6904    }
6905
6906
6907    \09-Dynamic-Programming\Optional-01-More-about-Fibonacci\src\Solution3.java
6908
6909    /// 70. Climbing Stairs
6910    /// https://leetcode.com/problems/climbing-stairs/description/
6911    ///
6912    /// 对于第n个斐波那契数, 可以推导出其公式
6913    /// Fn = 1/sqrt(5) * {[(1+sqrt(5))/2]^n - [(1-sqrt(5))/2]^n}
6914    /// 具体推导过程, 有兴趣的同学可以自行在互联网上搜索学习。
6915    /// 注意: 这个方法的时间复杂度依然是O(logn)的,因为数的幂运算也需要logn的时间
6916    /// 但这个方法快于使用矩阵的幂运算符的方法
6917    ///
6918    /// 时间复杂度: O(logn)
6919    /// 空间复杂度: O(1)
6920    public class Solution3 {
6921
6922        public int climbStairs(int n) {
6923
6924            if(n <= 0)
6925                throw new IllegalArgumentException("n must be greater than zero");
6926
6927            if(n == 1)
6928                return 1;
6929
6930            double sqrt5 = Math.sqrt(5.0);
```

```
6931            return (int)((Math.pow((1 + sqrt5) / 2, n + 1) - Math.pow((1 - sqrt5) / 2, n + 1)) / sqrt5);
6932        }
6933
6934        public static void main(String[] args) {
6935
6936            System.out.println((new Solution3()).climbStairs(10));
6937        }
6938    }
6939
6940
6941    \09-Dynamic-Programming\Optional-02-More-about-LIS\src\Solution.java
6942
6943    import java.util.Arrays;
6944
6945    /// 300. Longest Increasing Subsequence
6946    /// https://leetcode.com/problems/longest-increasing-subsequence/description/
6947    ///
6948    /// 我们这一章介绍的动态规划法求解LIS问题，时间复杂度为O(nlogn)的
6949    /// LIS有一个经典的，同时也非常巧妙的动态规划方法，其时间复杂度为O(nlogn)的
6950    /// 以下为参考代码和简单注释，如果需要更详细的解释，大家可以自行在互联网上搜索学习
6951    /// 通过这个例子，也请大家再体会改变动态规划的状态定义，
6952    /// 带来解决问题方法的重大不同，甚至是时间复杂度数量级上的巨大优化
6953    ///
6954    /// 时间复杂度: O(nlogn)
6955    /// 空间复杂度: O(n)
6956    public class Solution {
6957
6958        public int lengthOfLIS(int[] nums) {
6959
6960            if(nums.length == 0)
6961                return 0;
6962
6963            // dp[i] 表示最长长度为i的递增子序列，最后一个数字的最小值
6964            int dp[] = new int[nums.length + 1];
6965            Arrays.fill(dp, Integer.MIN_VALUE);
6966
6967            int len = 1;
6968            dp[1] = nums[0];
6969            for(int i = 1 ; i < nums.length ; i ++)
6970                if(nums[i] > dp[len]){
6971                    len ++;
6972                    dp[len] = nums[i];
6973                }
6974                else{
6975                    // 我们的dp数组将是一个单调递增的数组，所以可以使用二分查找法
6976                    int index = lowerBound(dp, 0, len, nums[i]);
6977                    if(dp[index] != nums[i])
6978                        dp[index] = Math.min(dp[index], nums[i]);
6979                }
6980
6981            return len;
6982        }
6983
6984        // lowerBound求出arr[l...r]范围里，大于等于target的第一个元素所在的索引
6985        private int lowerBound(int[] arr, int l, int r, int target){
6986
6987            int left = l, right = r + 1;
6988            while(left != right){
6989                int mid = left + (right - left) / 2;
6990                if(arr[mid] >= target)
6991                    right = mid;
6992                else // arr[mid] < target
6993                    left = mid + 1;
6994            }
6995            return left;
6996        }
```

```
6997
6998        public static void main(String[] args) {
6999
7000            int nums1[] = {10, 9, 2, 5, 3, 7, 101, 18};
7001            System.out.println((new Solution()).lengthOfLIS(nums1));
7002            // 4
7003
7004            // ---
7005
7006            int nums2[] = {4, 10, 4, 3, 8, 9};
7007            System.out.println((new Solution()).lengthOfLIS(nums2));
7008            // 3
7009
7010            // ---
7011
7012            int nums3[] = {2, 2};
7013            System.out.println((new Solution()).lengthOfLIS(nums3));
7014            // 1
7015
7016            // ---
7017
7018            int nums4[] = {1, 3, 6, 7, 9, 4, 10, 5, 6};
7019            System.out.println((new Solution()).lengthOfLIS(nums4));
7020            // 6
7021        }
7022    }
7023
7024
7025    \10-Greedy-Algorithms\01-Assign-Cookies\src\Solution.java
7026
7027    import java.util.Arrays;
7028
7029    /// 455. Assign Cookies
7030    /// https://leetcode.com/problems/assign-cookies/description/
7031    /// 先尝试满足最贪心的小朋友
7032    /// 时间复杂度: O(nlogn)
7033    /// 空间复杂度: O(1)
7034    public class Solution {
7035
7036        public int findContentChildren(int[] g, int[] s) {
7037
7038            Arrays.sort(g);
7039            Arrays.sort(s);
7040
7041            int gi = g.length - 1, si = s.length - 1;
7042            int res = 0;
7043            while(gi >= 0 && si >= 0){
7044                if(s[si] >= g[gi]){
7045                    res ++;
7046                    si --;
7047                }
7048                gi --;
7049            }
7050
7051            return res;
7052        }
7053
7054        public static void main(String[] args) {
7055
7056            int g1[] = {1, 2, 3};
7057            int s1[] = {1, 1};
7058            System.out.println((new Solution()).findContentChildren(g1, s1));
7059
7060            int g2[] = {1, 2};
7061            int s2[] = {1, 2, 3};
7062            System.out.println((new Solution()).findContentChildren(g2, s2));
```

```
        }
    }


    \10-Greedy-Algorithms\01-Assign-Cookies\src\Solution2.java

    import java.util.Arrays;

    /// 455. Assign Cookies
    /// https://leetcode.com/problems/assign-cookies/description/
    /// 先尝试满足最不贪心的小朋友
    /// 时间复杂度: O(nlogn)
    /// 空间复杂度: O(1)
    public class Solution2 {

        public int findContentChildren(int[] g, int[] s) {

            Arrays.sort(g);
            Arrays.sort(s);

            int gi = 0, si = 0;
            int res = 0;
            while(gi < g.length && si < s.length){
                if(s[si] >= g[gi]){
                    res ++;
                    gi ++;
                }
                si ++;
            }

            return res;
        }

        public static void main(String[] args) {

            int g1[] = {1, 2, 3};
            int s1[] = {1, 1};
            System.out.println((new Solution2()).findContentChildren(g1, s1));

            int g2[] = {1, 2};
            int s2[] = {1, 2, 3};
            System.out.println((new Solution2()).findContentChildren(g2, s2));
        }
    }


    \10-Greedy-Algorithms\02-Non-overlapping-Intervals\src\Solution1.java

    import java.util.Arrays;
    import java.util.Comparator;

    /// 435. Non-overlapping Intervals
    /// https://leetcode.com/problems/non-overlapping-intervals/description/
    /// 动态规划
    /// 时间复杂度: O(n^2)
    /// 空间复杂度: O(n)
    public class Solution1 {

        // Definition for an interval.
        public static class Interval {
            int start;
            int end;
            Interval() { start = 0; end = 0; }
            Interval(int s, int e) { start = s; end = e; }
        }

```

```
7129        public int eraseOverlapIntervals(Interval[] intervals) {
7130
7131            if(intervals.length == 0)
7132                return 0;
7133
7134            Arrays.sort(intervals, new Comparator<Interval>() {
7135                @Override
7136                public int compare(Interval o1, Interval o2) {
7137                    if(o1.start != o2.start)
7138                        return o1.start - o2.start;
7139                    return o1.end - o2.end;
7140                }
7141            });
7142
7143            // memo[i]表示以intervals[i]为结尾的区间能构成的最长不重叠区间序列
7144            int[] memo = new int[intervals.length];
7145            Arrays.fill(memo, 1);
7146            for(int i = 1 ; i < intervals.length ; i ++)
7147                // memo[i]
7148                for(int j = 0 ; j < i ; j ++)
7149                    if(intervals[i].start >= intervals[j].end)
7150                        memo[i] = Math.max(memo[i], 1 + memo[j]);
7151
7152            int res = 0;
7153            for(int i = 0; i < memo.length ; i ++)
7154                res = Math.max(res, memo[i]);
7155
7156            return intervals.length - res;
7157        }
7158
7159        public static void main(String[] args) {
7160            Interval[] interval1 = {new Interval(1,2),
7161                                    new Interval(2,3),
7162                                    new Interval(3,4),
7163                                    new Interval(1,3)};
7164            System.out.println((new Solution1()).eraseOverlapIntervals(interval1));
7165
7166            Interval[] interval2 = {new Interval(1,2),
7167                                    new Interval(1,2),
7168                                    new Interval(1,2)};
7169            System.out.println((new Solution1()).eraseOverlapIntervals(interval2));
7170
7171            Interval[] interval3 = {new Interval(1,2),
7172                                    new Interval(2,3)};
7173            System.out.println((new Solution1()).eraseOverlapIntervals(interval3));
7174        }
7175    }
7176
7177
7178    \10-Greedy-Algorithms\02-Non-overlapping-Intervals\src\Solution2.java
7179
7180    import java.util.Arrays;
7181    import java.util.Comparator;
7182
7183    /// 435. Non-overlapping Intervals
7184    /// https://leetcode.com/problems/non-overlapping-intervals/description/
7185    /// 贪心算法
7186    /// 时间复杂度: O(n)
7187    /// 空间复杂度: O(n)
7188    public class Solution2 {
7189
7190        // Definition for an interval.
7191        public static class Interval {
7192            int start;
7193            int end;
7194            Interval() { start = 0; end = 0; }
```

```
7195            Interval(int s, int e) { start = s; end = e; }
7196        }
7197
7198        public int eraseOverlapIntervals(Interval[] intervals) {
7199
7200            if(intervals.length == 0)
7201                return 0;
7202
7203            Arrays.sort(intervals, new Comparator<Interval>() {
7204                @Override
7205                public int compare(Interval o1, Interval o2) {
7206                    if(o1.end != o2.end)
7207                        return o1.end - o2.end;
7208                    return o1.start - o2.start;
7209                }
7210            });
7211
7212            int res = 1;
7213            int pre = 0;
7214            for(int i = 1 ; i < intervals.length ; i ++)
7215                if(intervals[i].start >= intervals[pre].end){
7216                    res ++;
7217                    pre = i;
7218                }
7219
7220            return intervals.length - res;
7221        }
7222
7223        public static void main(String[] args) {
7224            Interval[] interval1 = {new Interval(1,2),
7225                    new Interval(2,3),
7226                    new Interval(3,4),
7227                    new Interval(1,3)};
7228            System.out.println((new Solution2()).eraseOverlapIntervals(interval1));
7229
7230            Interval[] interval2 = {new Interval(1,2),
7231                    new Interval(1,2),
7232                    new Interval(1,2)};
7233            System.out.println((new Solution2()).eraseOverlapIntervals(interval2));
7234
7235            Interval[] interval3 = {new Interval(1,2),
7236                    new Interval(2,3)};
7237            System.out.println((new Solution2()).eraseOverlapIntervals(interval3));
7238        }
7239    }
7240
```