```java
\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)
        throw new IllegalArgumentException("i is out of bound.");

      if(j < 0 || j >= arr.length)
        throw new IllegalArgumentException("j is out of bound.");

      Object temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
    }

    // O(n)
    private static int sum(int n){

      if(n < 0)
        throw new IllegalArgumentException("n should be greater or equal to zero.");

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

    private static void reverse(Object[] arr){

      int n = arr.length;
      for(int i = 0 ; i < n / 2 ; i ++ )
        swap(arr, i, n - 1 - i);
    }

    // O(n^2) Time Complexity
    private 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);
      }
    }

    // O(n) Time Complexity
    private static void printInformation(int n){

      for( int i = 1 ; i <= n ; i ++ )
        for( int j = 1 ; j <= 30 ; j ++ )
          System.out.println("Class " + i + " - " + "No. " + j);
    }

    // O(logn) Time Complexity
    private static int binarySearch(Comparable[] arr, int n, int target){

      int l = 0, r = n-1;
      while( l <= r ){
        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;
    }

    private static String intToString(int num){

      StringBuilder s = new StringBuilder("");
```

```java
        String sign = "+";
        if(num < 0){
            num = -num;
            sign = "-";
        }

        while(num != 0){
            s.append(Character.getNumericValue('0') + num % 10);
            num /= 10;
        }

        if(s.length() == 0)
            s.append('0');

        s.reverse();
        if(sign == "-")
            return sign + s.toString();
        else
            return s.toString();
    }


    // O(nlogn)
    private static void hello(int n){

        for( int sz = 1 ; sz < n ; sz += sz )
            for( int i = 1 ; i < n ; i ++ )
                System.out.println("Hello, Algorithm!");
    }


    // O(sqrt(n)) Time Complexity
    private static boolean isPrime(int num){

        for(int x = 2 ; x*x <= num ; x ++)
            if( num % x == 0 )
                return false;
        return true;
    }

    private static boolean isPrime2(int num){

        if( num <= 1 ) return false;
        if( num == 2 ) return true;
        if( num % 2 == 0 ) return false;

        for(int x = 3 ; x * x <= num ; x += 2)
            if( num%x == 0 )
                return false;

        return true;
    }

    public static void main(String[] args) {

        System.out.println(intToString(123));
        System.out.println(intToString(0));
        System.out.println(intToString(-123));

        System.out.println();

        if(isPrime2(137)) System.out.println("137 is a prime.");
        else System.out.println("137 is not a prime.");

        if(isPrime2(121)) System.out.println("121 is a prime.");
        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 {

    public static void main(String[] args) {

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

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

            long startTime = System.currentTimeMillis();
            MyAlgorithmTester.binarySearch(arr, n, 0);
            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\Main4.java

```java
/**
 * Created by liuyubobobo.
 */
public class Main4 {

    public static void main(String[] args) {

        // 数据规模倍乘测试mergeSort
        // O(nlogn)
        System.out.println("Test for Merge Sort:");
        for( int i = 10 ; i <= 26 ; i ++ ){

            int n = (int)Math.pow(2,i);
            Integer[] arr = MyUtil.generateRandomArray(n, 0, 1<<30);

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

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

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

```java
/**
 * Created by liuyubobobo.
 */
public class MyAlgorithmTester {

    private MyAlgorithmTester(){}

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

        int l = 0, r = n - 1;
        while(l <= r){

            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){

        if(i < 0 || i >= arr.length)
            throw new IllegalArgumentException("i is out of bound");

        if(j < 0 || j >= arr.length)
            throw new IllegalArgumentException("j is out of bound");

        Object t = arr[i];
        arr[i] = arr[j];
        arr[j] = t;
    }
}


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

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

    private MyUtil(){}

    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;
    }
}


\02-Time-Complexity\05-Recursion-Time-Complexity\src\Main.java

public class Main {

    // binarySearch
    private static int binarySearch(Comparable[] arr, int l, int r, int target){

        if(l > r)
            return -1;

        int mid = l + (r - l) / 2;
        if(arr[mid].compareTo(target) == 0)
            return mid;
        else if(arr[mid].compareTo(target) > 0)
            return binarySearch(arr, l, mid - 1, target);
        else
            return binarySearch(arr, mid + 1, r, target);

    }

    // sum
    private static int sum(int n){

        assert n >= 0;

        if(n == 0)
            return 0;
        return n + sum(n - 1);
    }

    // pow2
    private static double pow(double x, int n){

        assert n >= 0;

        if(n == 0)
            return 1.0;

        double t = pow(x, n / 2);
        if(n % 2 == 1)
            return x * t * t;

        return t * t;
    }

    public static void main(String[] args) {

        System.out.println(sum(100));
        System.out.println(pow(2, 10));
    }
}


\02-Time-Complexity\05-Recursion-Time-Complexity\src\Main2.java

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

    // f
    private static int f(int n){

        assert( n >= 0 );

        if(n == 0)
            return 1;

        return f(n - 1) + f(n - 1);
    }

    /*
    // mergeSort
    private static void mergeSort(Comparable[] arr, int l, int r){

        if(l >= r)
            return;

        int mid = (l+r)/2;
        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);
        merge(arr, l, mid, r);
    }
    */

    public static void main(String[] args) {

```

```java
        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;
    }

    // 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);
            }
            for(int num = 0 ; num < n ; num ++){
                vec.pop_back();
            }
            long endTime = System.currentTimeMillis();

            System.out.print(2 * n + " operations: \t");
            System.out.println((endTime - startTime) + " ms");
        }
    }
}


\03-Using-Array\01-Binary-Search\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 - 1; // 在[l...r]的范围里寻找target
        while(l <= r){    // 当 l == r时,区间[l...r]依然是有效的
            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 - 1;  // target在[l...mid-1]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!");
        long endTime = System.currentTimeMillis();

        System.out.println("Binary Search test complete.");
        System.out.println("Time cost: " + (endTime - startTime) + " ms");
    }
}
```

```java

\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!");
        long endTime = System.currentTimeMillis();

        System.out.println("Binary Search 2 test complete.");
        System.out.println("Time cost: " + (endTime - startTime) + " ms");
    }
}


\03-Using-Array\02-Binary-Search-II\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\03-Move-Zeroes\src\Solution.java

import java.util.*;

// 283. Move Zeroes
// https://leetcode.com/problems/move-zeroes/description/
// 时间复杂度: O(n)
// 空间复杂度: O(n)
class Solution {
    public void moveZeroes(int[] nums) {

        ArrayList<Integer> nonZeroElements = new ArrayList<Integer>();

        // 将vec中所有非0元素放入nonZeroElements中
        for(int i = 0 ; i < nums.length ; i ++)
            if(nums[i] != 0)
                nonZeroElements.add(nums[i]);

        // 将nonZeroElements中的所有元素依次放入到nums开始的位置
        for(int i = 0 ; i < nonZeroElements.size() ; i ++)
            nums[i] = nonZeroElements.get(i);

        // 将nums剩余的位置放置为0
        for(int i = nonZeroElements.size() ; i < nums.length ; i ++)
            nums[i] = 0;
    }
```

```java

    public static void main(String args[]){

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

        (new Solution()).moveZeroes(arr);

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

\03-Using-Array\04-Move-Zeroes-II\src\Solution1.java

import java.util.*;

// 283. Move Zeroes
// https://leetcode.com/problems/move-zeroes/description/
// 时间复杂度: O(n)
// 空间复杂度: O(n)
class Solution1 {

    public void moveZeroes(int[] nums) {

        ArrayList<Integer> nonZeroElements = new ArrayList<Integer>();

        // 将vec中所有非0元素放入nonZeroElements中
        for (int i = 0; i < nums.length; i++)
            if (nums[i] != 0)
                nonZeroElements.add(nums[i]);

        // 将nonZeroElements中的所有元素依次放入到nums开始的位置
        for (int i = 0; i < nonZeroElements.size(); i++)
            nums[i] = nonZeroElements.get(i);

        // 将nums剩余的位置放置为0
        for (int i = nonZeroElements.size(); i < nums.length; i++)
            nums[i] = 0;
    }


    public static void main(String args[]){

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

        (new Solution1()).moveZeroes(arr);

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

\03-Using-Array\04-Move-Zeroes-II\src\Solution2.java

// 283. Move Zeroes
// 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 Solution2 {
    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
        for(int i = 0 ; i < nums.length ; i ++)
            if( nums[i] != 0 )
                nums[k++] = nums[i];

        // 将nums剩余的位置放置为0
        for(int i = k ; i < nums.length ; i ++)
            nums[i] = 0;
    }

    public static void main(String args[]){

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

        (new Solution2()).moveZeroes(arr);

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

\03-Using-Array\04-Move-Zeroes-II\src\Solution3.java

// 283. Move Zeroes
// 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 Solution3 {
    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)
                swap(nums, k++, i);
    }

    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 Solution3()).moveZeroes(arr);

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

\03-Using-Array\04-Move-Zeroes-II\src\Solution4.java

// 283. Move Zeroes
// 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;

        // 小练习: 自学编写计数排序算法
    }

    public 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, 2, 2, 1, 1, 0};
        (new Solution1()).sortColors(nums);
        printArr(nums);
    }
}


\03-Using-Array\05-Sort-Colors\src\Solution2.java

// 75. Sort Colors
// https://leetcode.com/problems/sort-colors/description/
//
// 三路快速排序的思想
// 对整个数组只遍历了一遍
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution2 {

    public void sortColors(int[] nums) {

        int zero = -1;          // [0...zero] == 0
        int two = nums.length;  // [two...n-1] == 2
        for(int i = 0 ; i < two ; ){
            if(nums[i] == 1)
                i ++;
            else if (nums[i] == 2)
                swap(nums, i, --two);
            else{ // nums[i] == 0
                assert nums[i] == 0;
                swap(nums, ++zero, i++);
            }
        }
    }

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

    public 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, 2, 2, 1, 1, 0};
        (new Solution2()).sortColors(nums);
        printArr(nums);
    }
}

```

```

\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");

        int l = 0, r = numbers.length - 1;
        while(l < r){

            if(numbers[l] + numbers[r] == target){
                int[] res = {l+1, r+1};
                return res;
            }
            else if(numbers[l] + numbers[r] < target)
                l ++;
            else // numbers[l] + numbers[r] > target
                r --;
        }

        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 Solution3()).twoSum(nums, target));
    }
}


\03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution1.java

// 209. Minimum Size Subarray Sum
// https://leetcode.com/problems/minimum-size-subarray-sum/description/
//
// 暴力解法
// 该方法在 Leetcode 中会超时！
// 时间复杂度: O(n^3)
// 空间复杂度: O(1)
public class Solution1 {

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

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

        int res = nums.length + 1;
        for(int l = 0 ; l < nums.length ; l ++)
            for(int r = l ; r < nums.length ; r ++){
                int sum = 0;
                for(int i = l ; i <= r ; i ++)
                    sum += nums[i];
                if(sum >= s)
                    res = Math.min(res, r - l + 1);
            }

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

```java

        return res;
    }

    public static void main(String[] args) {

        int[] nums = {2, 3, 1, 2, 4, 3};
        int s = 7;
        System.out.println((new Solution1()).minSubArrayLen(s, nums));
    }
}


\03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution2.java

// 209. Minimum Size Subarray Sum
// https://leetcode.com/problems/minimum-size-subarray-sum/description/
//
// 优化暴力解
// 时间复杂度: O(n^2)
// 空间复杂度: O(n)
public class Solution2 {

    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 ; l ++)
            for(int r = l ; r < nums.length ; r ++){
                // 使用sums[r+1] - sums[l] 快速获得nums[l...r]的和
                if(sums[r+1] - sums[l] >= s)
                    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 Solution2()).minSubArrayLen(s, nums));
    }
}


\03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution3.java

// 209. Minimum Size Subarray Sum
// https://leetcode.com/problems/minimum-size-subarray-sum/description/
//
// 滑动窗口的思路
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution3 {
    public int minSubArrayLen(int s, int[] nums) {

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

        int l = 0 , r = -1; // nums[l...r]为我们的滑动窗口
        int sum = 0;
        int res = nums.length + 1;

        while(l < nums.length){   // 窗口的左边界在数组范围内,则循环继续

            if(r + 1 < nums.length && sum < s)
                sum += nums[++r];
            else // r已经到头 或者 sum >= s
                sum -= nums[l++];

            if(sum >= s)
                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 Solution3()).minSubArrayLen(s, nums));
    }
}


\03-Using-Array\07-Minimum-Size-Subarray-Sum\src\Solution4.java

// 209. Minimum Size Subarray Sum
// https://leetcode.com/problems/minimum-size-subarray-sum/description/
//
// 另外一个滑动窗口的实现, 仅供参考
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution4 {

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

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

        int l = 0 , r = -1; // [l...r]为我们的窗口
        int sum = 0;
        int res = nums.length + 1;

        while(r + 1 < nums.length){   // 窗口的右边界无法继续扩展了, 则循环继续

            while(r + 1 < nums.length && sum < s)
                sum += nums[++r];

            if(sum >= s)
                res = Math.min(res, r - l + 1);

            while(l < nums.length && sum >= s){
                sum -= nums[l++];
                if(sum >= s)
                    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());
        testPerformace("Solution3", "lengthOfLongestSubstring", s.toString());
        testPerformace("Solution4", "lengthOfLongestSubstring", s.toString());
        testPerformace("Solution5", "lengthOfLongestSubstring", s.toString());

    }
}


\03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Solution1.java

// 3. Longest Substring Without Repeating Characters
// https://leetcode.com/problems/longest-substring-without-repeating-characters/description/
//
// 滑动窗口
// 时间复杂度: O(len(s))
// 空间复杂度: O(len(charset))
class Solution1 {
    public int lengthOfLongestSubstring(String s) {

        int[] freq = new int[256];

        int l = 0, r = -1; //滑动窗口为s[l...r]
        int res = 0;

        // 整个循环从 l == 0; r == -1 这个空窗口开始
        // 到l == s.size(); r == s.size()-1 这个空窗口截止
        // 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
        while(l < s.length()){

            if(r + 1 < s.length() && freq[s.charAt(r+1)] == 0)
                freq[s.charAt(++r)] ++;
            else    //r已经到头 || freq[s[r+1]] == 1
                freq[s.charAt(l++)] --;

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

        return res;
    }

    public static void main(String[] args) {

        System.out.println((new Solution1()).lengthOfLongestSubstring( "abcabcbb" ));
        System.out.println((new Solution1()).lengthOfLongestSubstring( "bbbbb" ));
        System.out.println((new Solution1()).lengthOfLongestSubstring( "pwwkew" ));
        System.out.println((new Solution1()).lengthOfLongestSubstring( "" ));
    }
}

\03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Solution2.java

// 3. Longest Substring Without Repeating Characters
// https://leetcode.com/problems/longest-substring-without-repeating-characters/description/
//
// 滑动窗口
// 时间复杂度: O(len(s))
// 空间复杂度: O(len(charset))
public class Solution2 {
    public int lengthOfLongestSubstring(String s) {

        int[] freq = new int[256];

        int l = 0, r = -1; //滑动窗口为s[l...r]
        int res = 0;

        // 在这里, 循环中止的条件可以是 r + 1 < s.length(), 想想看为什么?
        // 感谢课程QQ群 @千千 指出 :)
        while( r + 1 < s.length() ){

            if( r + 1 < s.length() && freq[s.charAt(r+1)] == 0 )
                freq[s.charAt(++r)] ++;
            else    //freq[s[r+1]] == 1
                freq[s.charAt(l++)] --;

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

        return res;
    }

    public static void main(String[] args) {
        System.out.println((new Solution2()).lengthOfLongestSubstring( "abcabcbb" ));
        System.out.println((new Solution2()).lengthOfLongestSubstring( "bbbbb" ));
        System.out.println((new Solution2()).lengthOfLongestSubstring( "pwwkew" ));
        System.out.println((new Solution2()).lengthOfLongestSubstring( "" ));
    }
}


\03-Using-Array\08-Longest-Substring-Without-Repeating-Characters\src\Solution3.java

// 3. Longest Substring Without Repeating Characters
// https://leetcode.com/problems/longest-substring-without-repeating-characters/description/
//
// 滑动窗口的另一个实现, 仅做参考
// 时间复杂度: O(len(s))
// 空间复杂度: O(len(charset))
public class Solution3 {
    public int lengthOfLongestSubstring(String s) {

        int[] freq = new int[256];

        int l = 0, r = -1; //滑动窗口为s[l...r]
        int res = 0;

        while(r + 1 < s.length()){

            while(r + 1 < s.length() && freq[s.charAt(r+1)] == 0)
                freq[s.charAt(++r)] ++;

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

            if(r + 1 < s.length()){
                freq[s.charAt(++r)] ++;
                assert(freq[s.charAt(r)] == 2);
                while(l <= r && freq[s.charAt(r)] == 2)
                    freq[s.charAt(l++)] --;
            }
        }

        return res;
    }

    public static void main(String[] args) {

        System.out.println((new Solution3()).lengthOfLongestSubstring( "abcabcbb" ));
        System.out.println((new Solution3()).lengthOfLongestSubstring( "bbbbb" ));
        System.out.println((new Solution3()).lengthOfLongestSubstring( "pwwkew" ));
        System.out.println((new Solution3()).lengthOfLongestSubstring( "" ));
    }
}
```

```
\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 + " ");
        System.out.println();
    }

    public static void main(String[] args) {

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


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

/// 让我们来测试使用Java中的TreeMap:)

import java.util.TreeMap;

public class Main {

    public static void main(String[] args) {

        TreeMap<Integer, Integer> myMap = new TreeMap<Integer, Integer>();
        if(myMap.containsKey(42))
            System.out.println("Element 42 is in the map");
        else
            System.out.println("Can not find element 42");

        System.out.println(myMap.get(42)); // 输出 null

        // Java不存在C++中默认的访问key即添加默认(key, value)的行为
        // 以下代码仍然无法找到42
        if(myMap.containsKey(42))
            System.out.println("Element 42 is in the map");
        else
            System.out.println("Can not find element 42");

        myMap.put(42, 0);
        myMap.put(42, myMap.get(42) + 1);
        System.out.println(myMap.get(42)); // 输出 1
        if(myMap.containsKey(42))
            System.out.println("Element 42 is in the map");
        else
            System.out.println("Can not find element 42");

        myMap.put(42, myMap.get(42) - 1);
        System.out.println(myMap.get(42)); // 输出 0

        // 注意：key对应的值为0，不代表key不存在
        if(myMap.containsKey(42))
            System.out.println("Element 42 is in the map");
        else
            System.out.println("Can not find element 42");

        // 注意： 也不可以为key对应的值设置null来删除一个key
        myMap.put(42, null);
        if(myMap.containsKey(42))
            System.out.println("Element 42 is in the map");
        else
            System.out.println("Can not find element 42");

        // 使用remove删除一个key
        myMap.remove(42);
        if(myMap.containsKey(42))
            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 {
```

```java

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

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

        HashSet<Integer> resultSet = new HashSet<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 + " ");
        System.out.println();
    }

    public static void main(String[] args) {

        int[] nums1 = {1, 2, 2, 1};
        int[] nums2 = {2, 2};
        int[] res = (new Solution349()).intersection(nums1, nums2);
        printArr(res);
    }
}

\04-Using-Hash-Table\03-More-About-Set-And-Map\src\Solution350.java

import java.util.HashMap;
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(len(nums1)+len(nums2))
// 空间复杂度: O(len(nums1))
public class Solution350 {

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

        HashMap<Integer, Integer> record = new HashMap<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 Solution350()).intersect(nums1, nums2);
        printArr(res);
    }
}
```

\04-Using-Hash-Table\04-Two-Sum\src\Solution.java

```
import java.util.HashMap;

// 1. Two Sum
// https://leetcode.com/problems/two-sum/description/
// 时间复杂度：O(n)
// 空间复杂度：O(n)
public class Solution {

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

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

            int complement = target - nums[i];
            if(record.containsKey(complement)){
                int[] res = {i, record.get(complement)};
                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\04-Two-Sum\src\Solution2.java

```
import java.util.HashMap;

// 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

```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);
        }

        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 Solution2()).fourSumCount(a, b, c, d));
    }
}


\04-Using-Hash-Table\06-Number-of-Boomerangs\src\Solution.java

import java.util.HashMap;

// 447. Number of Boomerangs
// https://leetcode.com/problems/number-of-boomerangs/description/
// 时间复杂度: O(n^2)
// 空间复杂度: O(n)
public class Solution {

    public int numberOfBoomerangs(int[][] points) {

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

            // record中存储 点i 到所有其他点的距离出现的频次
            HashMap<Integer, Integer> record = new HashMap<Integer, Integer>();
            for(int j = 0 ; j < points.length ; j ++)
                if(j != i){
                    // 计算距离时不进行开根运算, 以保证精度
                    int dis = dis(points[i], points[j]);
                    if(record.containsKey(dis))
                        record.put(dis, record.get(dis) + 1);
                    else
                        record.put(dis, 1);
            }

            for(Integer dis: record.keySet())
                res += record.get(dis) * (record.get(dis) - 1);
        }

        return res;
    }

    private int dis(int[] pa, int pb[]){
        return (pa[0] - pb[0]) * (pa[0] - pb[0]) +
               (pa[1] - pb[1]) * (pa[1] - pb[1]);
    }

    public static void main(String[] args) {

        int[][] points = {{0, 0}, {1, 0}, {2, 0}};
        System.out.println((new Solution()).numberOfBoomerangs(points));
    }
}


\04-Using-Hash-Table\07-Contains-Duplicate-II\src\Solution.java

import java.util.HashSet;

```

```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

```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]);
        }

        return false;
    }

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

    public static void main(String[] args) {

        int[] nums = {-2147483648, -2147483647};
        int k = 3;
        int t = 3;
        printBool((new Solution()).containsNearbyAlmostDuplicate(nums, k, t));
    }
}


\05-About-Linked-List\01-Reverse-Linked-List\src\Solution1.java

// 206. Reverse Linked List
// https://leetcode.com/problems/reverse-linked-list/description/
// 时间复杂度: O(n)
// 空间复杂度: O(1)
public class Solution1 {

    // Definition for singly-linked list.
    public class ListNode {
        int val;
        ListNode next;
        ListNode(int x) { val = x; }
    }

    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;
    }
}


\05-About-Linked-List\01-Reverse-Linked-List\src\Solution2.java

// 206. Reverse Linked List
// https://leetcode.com/problems/reverse-linked-list/description/
//
// 递归的方式反转链表
// 时间复杂度: O(n)
// 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：）
public class Solution2 {

    // Definition for singly-linked list.
    public class ListNode {
        int val;
        ListNode next;
        ListNode(int x) { val = x; }
    }

```

```java
    public ListNode reverseList(ListNode head) {

        // 递归终止条件
        if(head == null|| head.next == null)
            return head;

        ListNode rhead = reverseList(head.next);

        // head->next此刻指向head后面的链表的尾节点
        // head->next->next = head把head节点放在了尾部
        head.next.next = head;
        head.next = null;

        return rhead;
    }
}


\05-About-Linked-List\02-Test-Your-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){

        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\02-Test-Your-Linked-List\src\Solution.java

// 206. Reverse Linked List
// https://leetcode.com/problems/reverse-linked-list/description/
// 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

```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) {

        // 创建虚拟头结点
        ListNode dummyHead = new ListNode(0);
        dummyHead.next = head;

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

        return dummyHead.next;
    }

    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\04-Swap-Nodes-in-Pairs\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){
```

```java
            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){

        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;
        }
    }

    ListNode findNode(int x){

        ListNode curNode = this;
        while(curNode != null){
            if(curNode.val == x)
                return curNode;
            curNode = curNode.next;
        }
        return null;
    }

    // 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\05-Delete-Node-in-a-Linked-List\src\Solution.java

// 237. Delete Node in a Linked List
// https://leetcode.com/problems/delete-node-in-a-linked-list/description/
// 时间复杂度: O(1)
// 空间复杂度: O(1)
public class Solution {

    public void deleteNode(ListNode node) {

        // 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
        // 我们检查node.next, 如果为null则抛出异常, 确保了node不是尾节点
        if(node == null || node.next == null)
            throw new IllegalArgumentException("node should be valid and can not be the tail node.");

        node.val = node.next.val;
        node.next = node.next.next;
    }

    public static void main(String[] args) {

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

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

        ListNode node2 = head.findNode(2);
        (new Solution()).deleteNode(node2);
        System.out.println(head);
    }
}
```

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

```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;
        }
    }

    ListNode findNode(int x){

        ListNode curNode = this;
        while(curNode != null){
            if(curNode.val == x)
                return curNode;
            curNode = curNode.next;
        }
        return null;
    }

    // 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\06-Remove-Nth-Node-From-End-of-List\src\Solution1.java

```java
// 19. Remove Nth Node From End of List
// https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/
//
// 先记录链表总长度
// 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

// 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;

```

```java
        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 Solution2()).removeNthFromEnd(head, 2);
        System.out.println(head);
    }
}
```

\06-Stack-and-Queue\01-Valid-Parentheses\src\Solution.java

```java
import java.util.Stack;

// 20. Valid Parentheses
// https://leetcode.com/problems/valid-parentheses/description/
// 时间复杂度: O(n)
// 空间复杂度: O(n)
public class Solution {

    public boolean isValid(String s) {

        Stack<Character> stack = new Stack<Character>();
        for( int i = 0 ; i < s.length() ; i ++ )
            if( s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt(i) == '[')
                stack.push(s.charAt(i));
            else{

                if( stack.size() == 0 )
                    return false;

                Character c = stack.pop();

                Character match;
                if( s.charAt(i) == ')' )
                    match = '(';
                else if( s.charAt(i) == ']' )
                    match = '[';
                else{
                    assert s.charAt(i) == '}';
                    match = '{';
                }

                if(c != match)
                    return false;
            }

        if( stack.size() != 0 )
            return false;

        return true;
    }

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

    public static void main(String[] args) {

        printBool((new Solution()).isValid("()"));
        printBool((new Solution()).isValid("()[]{}"));
        printBool((new Solution()).isValid("(]"));
        printBool((new Solution()).isValid("([)]"));
    }
}


\06-Stack-and-Queue\02-Recursion-and-Stack\src\Solution094.java

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

/// 94. Binary Tree Inorder Traversal
/// https://leetcode.com/problems/binary-tree-inorder-traversal/solution/
/// 二叉树的中序遍历
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(h), h为树的高度
public class Solution094 {

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

    public List<Integer> inorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        inorderTraversal(root, res);
        return res;
    }

    private void inorderTraversal(TreeNode node, List<Integer> list){
        if(node != null){
            inorderTraversal(node.left, list);
            list.add(node.val);
            inorderTraversal(node.right, list);
        }
    }
}


\06-Stack-and-Queue\02-Recursion-and-Stack\src\Solution144.java

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

/// 144. Binary Tree Preorder Traversal
/// https://leetcode.com/problems/binary-tree-preorder-traversal/description/
/// 二叉树的前序遍历
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(h), h为树的高度
public class Solution144 {

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

    public List<Integer> preorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        preorderTraversal(root, res);
        return res;
    }

    private void preorderTraversal(TreeNode node, List<Integer> list){
        if(node != null){
            list.add(node.val);
            preorderTraversal(node.left, list);
            preorderTraversal(node.right, list);
        }
    }
}


\06-Stack-and-Queue\02-Recursion-and-Stack\src\Solution145.java

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

/// 145. Binary Tree Postorder Traversal
/// https://leetcode.com/problems/binary-tree-postorder-traversal/description/
/// 二叉树的后序遍历
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(h), h为树的高度
public class Solution145 {

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

    public List<Integer> postorderTraversal(TreeNode root) {

        ArrayList<Integer> res = new ArrayList<Integer>();
        postorderTraversal(root, res);
        return res;
    }

    private void postorderTraversal(TreeNode node, List<Integer> list){
        if(node != null){
            postorderTraversal(node.left, list);
            postorderTraversal(node.right, list);
            list.add(node.val);
        }
    }
}


\06-Stack-and-Queue\03-Non-Recursive-Implementation-of-a-Recursive-Algorithm\src\Solution094.java

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

/// 94. Binary Tree Inorder Traversal
/// https://leetcode.com/problems/binary-tree-inorder-traversal/solution/
/// 非递归二叉树的中序遍历
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(h), h为树的高度
public class Solution094 {

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

    private class Command{
        String s;   // go, print
        TreeNode node;
        Command(String s, TreeNode node){
            this.s = s;
            this.node = node;
        }
    };

    public List<Integer> inorderTraversal(TreeNode root) {

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

        Stack<Command> stack = new Stack<Command>();
        stack.push(new Command("go", root));
        while(!stack.empty()){
            Command command = stack.pop();

            if(command.s.equals("print"))
                res.add(command.node.val);
            else{
                assert command.s.equals("go");
                if(command.node.right != null)
                    stack.push(new Command("go",command.node.right));
                stack.push(new Command("print", command.node));
                if(command.node.left != null)
                    stack.push(new Command("go",command.node.left));
            }
        }
        return res;
    }

}


\06-Stack-and-Queue\03-Non-Recursive-Implementation-of-a-Recursive-Algorithm\src\Solution144.java

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

/// 144. Binary Tree Preorder Traversal
/// https://leetcode.com/problems/binary-tree-preorder-traversal/description/
/// 非递归二叉树的前序遍历
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(h), h为树的高度
public class Solution144 {

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

    private class Command{
        String s;   // go, print
        TreeNode node;
        Command(String s, TreeNode node){
            this.s = s;
            this.node = node;
        }
    };

    public List<Integer> preorderTraversal(TreeNode root) {

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

        Stack<Command> stack = new Stack<Command>();
        stack.push(new Command("go", root));
        while(!stack.empty()){
            Command command = stack.pop();

            if(command.s.equals("print"))
                res.add(command.node.val);
            else{
                assert command.s.equals("go");
                if(command.node.right != null)
                    stack.push(new Command("go",command.node.right));
                if(command.node.left != null)
                    stack.push(new Command("go",command.node.left));
                stack.push(new Command("print", command.node));
            }
        }
        return res;
    }

}


\06-Stack-and-Queue\03-Non-Recursive-Implementation-of-a-Recursive-Algorithm\src\Solution145.java

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

/// 145. Binary Tree Postorder Traversal
/// https://leetcode.com/problems/binary-tree-postorder-traversal/description/
/// 非递归的二叉树的后序遍历
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(h), h为树的高度
public class Solution145 {

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

    private class Command{
        String s;   // go, print
        TreeNode node;
        Command(String s, TreeNode node){
            this.s = s;
            this.node = node;
        }
    };

    public List<Integer> postorderTraversal(TreeNode root) {
```

```java

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

        Stack<Command> stack = new Stack<Command>();
        stack.push(new Command("go", root));
        while(!stack.empty()){
            Command command = stack.pop();

            if(command.s.equals("print"))
                res.add(command.node.val);
            else{
                assert command.s.equals("go");
                stack.push(new Command("print", command.node));
                if(command.node.right != null)
                    stack.push(new Command("go",command.node.right));
                if(command.node.left != null)
                    stack.push(new Command("go",command.node.left));
            }
        }
        return res;
    }

}


\06-Stack-and-Queue\04-Binary-Tree-Level-Order-Traversal\src\Solution.java

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

/// 102. Binary Tree Level Order Traversal
/// https://leetcode.com/problems/binary-tree-level-order-traversal/description/
/// 二叉树的层序遍历
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(n)
class Solution {

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

    public List<List<Integer>> levelOrder(TreeNode root) {

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

        // 我们使用LinkedList来做为我们的先入先出的队列
        LinkedList<Pair<TreeNode, Integer>> queue = new LinkedList<Pair<TreeNode, Integer>>();
        queue.addLast(new Pair<TreeNode, Integer>(root, 0));

        while(!queue.isEmpty()){

            Pair<TreeNode, Integer> front = queue.removeFirst();
            TreeNode node = front.getKey();
            int level = front.getValue();

            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();

            if(num == 0)
                return step;

            for(int i = 1 ; num - i*i >= 0 ; i ++){
                int a = num - i*i;
                if(!visited[a]){
                    if(a == 0) return step + 1;
                    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 Solution3()).numSquares(12));
        System.out.println((new Solution3()).numSquares(13));
    }
}


\06-Stack-and-Queue\06-Priority-Queue\src\Main.java

import java.util.Comparator;
import java.util.PriorityQueue;
import java.util.Random;

public class Main {

    public static void main(String[] args) {

        // 默认的PriorityQueue, 底层是最小堆
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();

        for(int i = 0 ; i < 10 ; i ++){
            int num = (int)(Math.random() * 100);
            pq.add(num);
            System.out.println("insert " + num + " in priority queue.");
        }

        while (!pq.isEmpty())
            System.out.print(pq.poll() + " ");

        System.out.println();
        System.out.println();


        // 使用lambda表达式，创建底层是最大堆的PriorityQueue
        PriorityQueue<Integer> pq2 = new PriorityQueue<Integer>(10, (a, b) -> b - a);

        for(int i = 0 ; i < 10 ; i ++){
            int num = (int)(Math.random() * 100);
            pq2.add(num);
            System.out.println("insert " + num + " in priority queue.");
        }

        while (!pq2.isEmpty())
            System.out.print(pq2.poll() + " ");

        System.out.println();
        System.out.println();


```

```java
        // 使用自定义的Comparator，创建个性化的PriorityQueue
        // 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
        // 同理，上一个例子也可以使用自定义的Comparator的方式完成
        class myCmp implements Comparator<Integer>{
            @Override
            public int compare(Integer a, Integer b){
                if(a%10 != b%10)
                    return a%10 - b%10;
                return a - b;
            }
        }
        PriorityQueue<Integer> pq3 = new PriorityQueue<Integer>(10, new myCmp());

        for(int i = 0 ; i < 10 ; i ++){
            int num = (int)(Math.random() * 100);
            pq3.add(num);
            System.out.println("insert " + num + " in priority queue.");
        }

        while (!pq3.isEmpty())
            System.out.print(pq3.poll() + " ");

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

\06-Stack-and-Queue\07-Top-K-Frequent-Elements\src\Solution.java

```java
import java.util.*;
import java.util.HashMap;

import javafx.util.Pair;

// 347. Top K Frequent Elements
// https://leetcode.com/problems/top-k-frequent-elements/description/
// 时间复杂度: O(nlogk)
// 空间复杂度: O(n + k)
class Solution {

    private class PairComparator implements Comparator<Pair<Integer, Integer>>{

        @Override
        public int compare(Pair<Integer, Integer> p1, Pair<Integer, Integer> p2){
            if(p1.getKey() != p2.getKey())
                return p1.getKey() - p2.getKey();
            return p1.getValue() - p2.getValue();
        }
    }

    public List<Integer> topKFrequent(int[] nums, int k) {

        if(k <= 0)
            throw new IllegalArgumentException("k should be greater than 0");

        // 统计每个元素出现的频率
        HashMap<Integer, Integer> freq = new HashMap<Integer, Integer>();
        for(int i = 0 ; i < nums.length ; i ++)
            if(freq.containsKey(nums[i]))
                freq.put(nums[i], freq.get(nums[i]) + 1);
            else
                freq.put(nums[i], 1);

        if(k > freq.size())
            throw new IllegalArgumentException("k should be less than the number of unique numbers in nums");

        // 扫描freq,维护当前出现频率最高的k个元素
        // 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
        PriorityQueue<Pair<Integer, Integer>> pq = new PriorityQueue<Pair<Integer, Integer>>(new PairComparator
        for(Integer num: freq.keySet()){
            int numFreq = freq.get(num);
            if(pq.size() == k){
                if(numFreq > pq.peek().getKey()){
                    pq.poll();
                    pq.add(new Pair(numFreq, num));
                }
            }
            else
                pq.add(new Pair(numFreq, num));
        }

        ArrayList<Integer> res = new ArrayList<Integer>();
        while(!pq.isEmpty())
            res.add(pq.poll().getValue());

        return res;
    }

    private static void printList(List<Integer> nums){
        for(Integer num: nums)
            System.out.print(num + " ");
        System.out.println();
    }

    public static void main(String[] args) {

        int[] nums = {1, 1, 1, 2, 2, 3};
        int k = 2;
        printList((new Solution()).topKFrequent(nums, k));
    }
}


\06-Stack-and-Queue\Optional-01-Classic-Non-Recursive-Preorder-Traversal\src\Solution1.java

/// Source : https://leetcode.com/problems/binary-tree-preorder-traversal/description/
/// Author : liuyubobobo
/// Time   : 2017-11-17

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

// 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 Solution1 {

    public List<Integer> preorderTraversal(TreeNode root) {

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

        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);
        while(!stack.empty()){
            TreeNode curNode = stack.pop();
            res.add(curNode.val);

            if(curNode.right != null)
```

```java
                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>();
        if(root == null)
            return res;

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


\06-Stack-and-Queue\Optional-01-Classic-Non-Recursive-Preorder-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-02-Classic-Non-Recursive-Inorder-Traversal\src\Solution1.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;

// Classic Non-Recursive algorithm for inorder 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 Solution1 {

    public List<Integer> inorderTraversal(TreeNode root) {

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

        Stack<TreeNode> stack = new Stack<>();
        TreeNode cur = root;
        while(cur != null || !stack.empty()){

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

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


\06-Stack-and-Queue\Optional-02-Classic-Non-Recursive-Inorder-Traversal\src\Solution2.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;

// Another Classic Non-Recursive algorithm for inorder 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> inorderTraversal(TreeNode root) {

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

        Stack<TreeNode> stack = new Stack<>();
        TreeNode cur = root;
        while(cur != null || !stack.empty()){

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


\06-Stack-and-Queue\Optional-02-Classic-Non-Recursive-Inorder-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-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution1.java

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

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

// Non-Recursive
// Using a tag to record whether the node has been visited
```

```
//
// 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 Solution1 {

    private class TagNode{
        TreeNode node;
        boolean isFirst;
        TagNode(TreeNode node){
            this.node = node;
            this.isFirst = false;
        }
    };

    public List<Integer> postorderTraversal(TreeNode root) {

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

        Stack<TagNode> stack = new Stack<>();
        TreeNode cur = root;
        while(cur != null || !stack.empty()){

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

            TagNode tagNode = stack.pop();
            cur = tagNode.node;
            if(tagNode.isFirst == false){
                tagNode.isFirst = true;
                stack.push(tagNode);
                cur = cur.right;
            }
            else{
                res.add(cur.val);
                cur = null;
            }
        }
        return res;
    }
}


\06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution2.java

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

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

// Non-Recursive
// Using two stacks, Reverse Preorder Traversal!
//
// Time Complexity: O(n)
// Space Complexity: O(n)
public class Solution2 {

    public List<Integer> postorderTraversal(TreeNode root) {

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

```java
        if(root == null)
            return res;

        Stack<TreeNode> stack = new Stack<>();
        Stack<Integer> output = new Stack<>();

        stack.push(root);
        while(!stack.empty()){

            TreeNode cur = stack.pop();
            output.push(cur.val);

            if(cur.left != null)
                stack.push(cur.left);
            if(cur.right != null)
                stack.push(cur.right);
        }

        while(!output.empty())
            res.add(output.pop());
        return res;
    }
}


\06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution3.java

/// Source : https://leetcode.com/problems/binary-tree-postorder-traversal/description/
/// Author : liuyubobobo
/// Time   : 2018-07-03

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

// Non-Recursive
// Using two stacks, Reverse Preorder Traversal!
//
// Time Complexity: O(n)
// Space Complexity: O(n)
public class Solution3 {

    public List<Integer> postorderTraversal(TreeNode root){

        Stack<TreeNode> stack = new Stack<>();
        LinkedList<TreeNode> output = new LinkedList<>();

        TreeNode p = root;
        while(p != null || !stack.isEmpty()){
            if(p != null){
                stack.push(p);
                output.push(p);
                p = p.right;
            }
            else{
                p = stack.pop();
                p = p.left;
            }
        }

        ArrayList<Integer> res = new ArrayList<>();
        while(!output.isEmpty())
            res.add(output.pop().val);
        return res;
    }
```

```java
}


\06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution4.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;

// Non-Recursive
// Using a pre pointer to record the last visted node
//
// Time Complexity: O(n)
// Space Complexity: O(h)
public class Solution4 {

    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;

        stack.push(root);
        while(!stack.empty()){

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


\06-Stack-and-Queue\Optional-03-Classic-Non-Recursive-Postorder-Traversal\src\Solution5.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 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){
```

```java
                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

```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

```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
```

```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-05-Word-Ladder\src\Solution.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;

/// BFS
/// Time Complexity: O(n*n)
/// Space Complexity: O(n)
public class Solution {

    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));

        // bfs
        LinkedList<Integer> q = new LinkedList<>();
        int[] step = new int[n];

        q.addLast(begin);
        step[begin] = 1;
        while(!q.isEmpty()){

            int cur = q.removeFirst();

            for(int i = 0 ; i < n ; i ++)
                if(step[i] == 0 && g[cur][i]){
                    if(i == end)
                        return step[cur] + 1;
                    step[i] = step[cur] + 1;
                    q.addLast(i);
                }
        }

        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\Solution2.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.HashSet;
import javafx.util.Pair;

/// BFS
/// Using set to store all the words and erase visited word eagerly.
/// Time Complexity: O(n*n)
/// Space Complexity: O(n)
public class Solution2 {

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

        HashSet<String> wordSet = new HashSet<>();
        for(String word: wordList)
            wordSet.add(word);

        // bfs
        LinkedList<Pair<String, Integer>> q = new LinkedList<>();
        q.addLast(new Pair<>(beginWord, 1));
        wordSet.remove(beginWord);

        HashSet<String> visited = new HashSet<>();

```

```java
        while(!q.isEmpty()){

            String curWord = q.getFirst().getKey();
            int curStep = q.getFirst().getValue();
            q.removeFirst();

            visited.clear();
            for(String word: wordSet){
                if(similar(word, curWord)){
                    if(word.equals(endWord))
                        return curStep + 1;
                    q.addLast(new Pair<>(word, curStep + 1));
                    visited.add(word);
                }
            }

            for(String word: visited)
                wordSet.remove(word);
        }

        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\Solution3.java

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

```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)
```

```java
                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

```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;

        // 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));

        // 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
    }
}


\07-Binary-Tree-and-Recursion\01-Maximum-Depth-of-Binary-Tree\src\Solution.java

// 104. Maximum Depth of Binary Tree
// https://leetcode.com/problems/maximum-depth-of-binary-tree/description/
// 时间复杂度: O(n), n是树中的节点个数
// 空间复杂度: O(h), h是树的高度
class Solution {

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

    public int maxDepth(TreeNode root) {

        if(root == null)
            return 0;

        return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));
    }
}


\07-Binary-Tree-and-Recursion\02-Invert-Binary-Tree\src\Solution.java

/// 226. Invert Binary Tree
/// https://leetcode.com/problems/invert-binary-tree/description/
/// 时间复杂度: O(n), n为树中节点个数
/// 空间复杂度: O(h), h为树的高度
public class Solution {

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

    public TreeNode invertTree(TreeNode root) {

        if(root == null)
            return null;

        TreeNode left = invertTree(root.left);
        TreeNode right = invertTree(root.right);

        root.left = right;
        root.right = left;

        return root;
    }
}


\07-Binary-Tree-and-Recursion\03-Path-Sum\src\Solution.java

/// 112. Path Sum
/// https://leetcode.com/problems/path-sum/description/
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(h), h为树的高度
class Solution {

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

    public boolean hasPathSum(TreeNode root, int sum) {

        if(root == null)
            return false;

        if(root.left == null && root.right == null)
            return sum == root.val;

        return hasPathSum(root.left, sum - root.val)
                || hasPathSum(root.right, sum - root.val);
    }
}

\07-Binary-Tree-and-Recursion\04-Binary-Tree-Paths\src\Solution.java

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

/// 257. Binary Tree Paths
/// https://leetcode.com/problems/binary-tree-paths/description/
/// 时间复杂度: O(n), n为树中的节点个数
/// 空间复杂度: O(h), h为树的高度
public class Solution {

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

    public List<String> binaryTreePaths(TreeNode root) {

        ArrayList<String> res = new ArrayList<String>();

        if(root == null)
            return res;

        if(root.left == null && root.right == null){
            res.add(Integer.toString(root.val));
            return res;
        }

        List<String> leftPaths = binaryTreePaths(root.left);
        for(String s: leftPaths){
            StringBuilder sb = new StringBuilder(Integer.toString(root.val));
            sb.append("->");
            sb.append(s);
            res.add(sb.toString());
        }

        List<String> rightPaths = binaryTreePaths(root.right);
        for(String s: rightPaths) {
            StringBuilder sb = new StringBuilder(Integer.toString(root.val));
            sb.append("->");
            sb.append(s);
            res.add(sb.toString());
        }

        return res;
    }
}


\07-Binary-Tree-and-Recursion\05-Path-Sum-III\src\Solution.java

/// 437. Path Sum III
/// https://leetcode.com/problems/path-sum-iii/description/
/// 时间复杂度: O(n), n为树的节点个数
/// 空间复杂度: O(h), h为树的高度
class Solution {

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

    // 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
    public int pathSum(TreeNode root, int sum) {

        if(root == null)
            return 0;

        return findPath(root, sum)
                + pathSum(root.left , sum)
                + pathSum(root.right , sum);
    }

    // 在以node为根节点的二叉树中,寻找包含node的路径,和为sum
    // 返回这样的路径个数
    private int findPath(TreeNode node, int num){

        if(node == null)
            return 0;

        int res = 0;
        if(node.val == num)
            res += 1;

        res += findPath(node.left , num - node.val);
        res += findPath(node.right , num - node.val);

        return res;
    }

    public static void main(String[] args) {

        // 手动创建Leetcode题页上的测试用例。
        // 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:)

        /*****************
         * 测试用例:
         *
         *       10
         *      /  \
         *     5   -3
         *    / \    \
         *   3   2   11
         *  / \   \
         * 3  -2   1
         *****************/
        TreeNode node1 = new TreeNode(3);
        TreeNode node2 = new TreeNode(-2);

        TreeNode node3 = new TreeNode(3);
        node3.left = node1;
        node3.right = node2;

        TreeNode node4 = new TreeNode(1);
        TreeNode node5 = new TreeNode(2);
        node5.right = node4;

        TreeNode node6 = new TreeNode(5);
        node6.left = node3;
        node6.right = node5;

        TreeNode node7 = new TreeNode(11);
        TreeNode node8 = new TreeNode(-3);
        node8.right = node7;

        TreeNode node9 = new TreeNode(10);
        node9.left = node6;
        node9.right = node8;

        System.out.println((new Solution()).pathSum(node9, 8));
    }
}
```

\07-Binary-Tree-and-Recursion\06-Lowest-Common-Ancestor-of-a-Binary-Search-Tree\src\Solution.java

```java
/// 235. Lowest Common Ancestor of a Binary Search Tree
/// https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/description/
/// 时间复杂度: O(lgn), 其中n为树的节点个数
/// 空间复杂度: O(h), 其中h为树的高度
class Solution {

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

    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

        if(p == null || q == null)
            throw new IllegalArgumentException("p or q can not be null.");

        if(root == null)
            return null;

        if(p.val < root.val && q.val < root.val)
            return lowestCommonAncestor(root.left, p, q);
        if(p.val > root.val && q.val > root.val)
            return lowestCommonAncestor(root.right, p, q);

        assert p.val == root.val || q.val == root.val
                || (root.val - p.val) * (root.val - q.val) < 0;

        return root;
    }
}


\08-Recurion-and-Backstracking\01-02-Letter-Combinations-of-a-Phone-Number\src\Solution.java

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

/// 17. Letter Combinations of a Phone Number
/// https://leetcode.com/problems/letter-combinations-of-a-phone-number/description/
/// 时间复杂度: O(2^len(s))
/// 空间复杂度: O(len(s))
class Solution {

    private String letterMap[] = {
                " ",    //0
                "",     //1
                "abc",  //2
                "def",  //3
                "ghi",  //4
                "jkl",  //5
                "mno",  //6
                "pqrs", //7
                "tuv",  //8
                "wxyz"  //9
    };

    private ArrayList<String> res;

    public List<String> letterCombinations(String digits) {

        res = new ArrayList<String>();
        if(digits.equals(""))
            return res;

        findCombination(digits, 0, "");
        return res;
    }

    // s中保存了此时从digits[0...index-1]翻译得到的一个字母字符串
    // 寻找和digits[index]匹配的字母, 获得digits[0...index]翻译得到的解
    private void findCombination(String digits, int index, String s){

        System.out.println(index + " : " + s);
        if(index == digits.length()){
            res.add(s);
            System.out.println("get " + s + " , return");
            return;
        }

        Character c = digits.charAt(index);
        assert  c.compareTo('0') >= 0 &&
                c.compareTo('9') <= 0 &&
                c.compareTo('1') != 0;
        String letters = letterMap[c - '0'];
```

```java
        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;
    }

    private static void printList(List<Integer> list){
        for(Integer e: list)
            System.out.print(e + " ");
        System.out.println();
    }

    public static void main(String[] args) {

        int[] nums = {1, 2, 3};
        List<List<Integer>> res = (new Solution()).permute(nums);
        for(List<Integer> list: res)
            printList(list);
    }
}


\08-Recurion-and-Backstracking\04-Combinations\src\Solution.java

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

/// 77. Combinations
/// https://leetcode.com/problems/combinations/description/
/// 时间复杂度: O(n^k)
/// 空间复杂度: O(k)
public class Solution {

    private ArrayList<List<Integer>> res;

    public List<List<Integer>> combine(int n, int k) {

        res = new ArrayList<List<Integer>>();
        if(n <= 0 || k <= 0 || k > n)
            return res;

        LinkedList<Integer> c = new LinkedList<Integer>();
        generateCombinations(n, k, 1, c);

        return res;
    }

    // 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
    private void generateCombinations(int n, int k, int start, LinkedList<Integer> c){

        if(c.size() == k){
            res.add((List<Integer>)c.clone());
            return;
        }

        for(int i = start ; i <= n ; i ++){
            c.addLast(i);
            generateCombinations(n, k, i + 1, c);
            c.removeLast();
        }

        return;
    }

    private static void printList(List<Integer> list){
        for(Integer e: list)
            System.out.print(e + " ");
        System.out.println();
    }
```

```java

    public static void main(String[] args) {

        List<List<Integer>> res = (new Solution()).combine(4, 2);
        for(List<Integer> list: res)
            printList(list);
    }
}


```

\08-Recurion-and-Backstracking\05-Combinations-optimized\src\Solution.java

```java

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

/// 77. Combinations
/// https://leetcode.com/problems/combinations/description/
/// 时间复杂度: O(n^k)
/// 空间复杂度: O(k)
public class Solution {

    private ArrayList<List<Integer>> res;

    public List<List<Integer>> combine(int n, int k) {

        res = new ArrayList<List<Integer>>();
        if(n <= 0 || k <= 0 || k > n)
            return res;

        LinkedList<Integer> c = new LinkedList<Integer>();
        generateCombinations(n, k, 1, c);

        return res;
    }

    // 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
    private void generateCombinations(int n, int k, int start, LinkedList<Integer> c){

        if(c.size() == k){
            res.add((List<Integer>)c.clone());
            return;
        }

        // 还有k - c.size()个空位, 所以, [i...n] 中至少要有 k - c.size() 个元素
        // i最多为 n - (k - c.size()) + 1
        for(int i = start ; i <= n - (k - c.size()) + 1 ; i ++){
            c.addLast(i);
            generateCombinations(n, k, i + 1, c);
            c.removeLast();
        }

        return;
    }

    private static void printList(List<Integer> list){
        for(Integer e: list)
            System.out.print(e + " ");
        System.out.println();
    }

    public static void main(String[] args) {

        List<List<Integer>> res = (new Solution()).combine(4, 2);
        for(List<Integer> list: res)
            printList(list);
    }
}


\08-Recurion-and-Backstracking\06-Word-Search\src\Solution.java

/// 79. Word Search
/// Source : https://leetcode.com/problems/word-search/description/
///
/// 回溯法
/// 时间复杂度: O(m*n*m*n)
/// 空间复杂度: O(m*n)
public class Solution {

    private int d[][] = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};
    private int m, n;
    private boolean[][] visited;

    public boolean exist(char[][] board, String word) {

        if(board == null || word == null)
            throw new IllegalArgumentException("board or word can not be null!");

        m = board.length;
        if(m == 0)
            throw new IllegalArgumentException("board can not be empty.");
        n = board[0].length;
        if(n == 0)
            throw new IllegalArgumentException("board can not be empty.");

        visited = new boolean[m][n];
        for(int i = 0 ; i < m ; i ++)
            for(int j = 0 ; j < n ; j ++)
                if(searchWord(board, word, 0, i, j))
                    return true;

        return false;
    }

    private boolean inArea( int x , int y ){
        return x >= 0 && x < m && y >= 0 && y < n;
    }

    // 从board[startx][starty]开始, 寻找word[index...word.size())
    private boolean searchWord(char[][] board, String word, int index,
                               int startx, int starty){

        //assert(inArea(startx,starty));
        if(index == word.length() - 1)
            return board[startx][starty] == word.charAt(index);

        if(board[startx][starty] == word.charAt(index)){
            visited[startx][starty] = true;
            // 从startx, starty出发,向四个方向寻
            for(int i = 0 ; i < 4 ; i ++){
                int newx = startx + d[i][0];
                int newy = starty + d[i][1];
                if(inArea(newx, newy) && !visited[newx][newy] &&
                        searchWord(board, word, index + 1, newx, newy))
                    return true;
            }
            visited[startx][starty] = false;
        }
        return false;
    }

    public static void main(String args[]){

        char[][] b1 = { {'A','B','C','E'},
                        {'S','F','C','S'},
                        {'A','D','E','E'}};

        String words[] = {"ABCCED", "SEE", "ABCB" };
        for(int i = 0 ; i < words.length ; i ++)
            if((new Solution()).exist(b1, words[i]))
                System.out.println("found " + words[i]);
            else
                System.out.println("can not found " + words[i]);

        // ---

        char[][] b2 = {{'A'}};
        if((new Solution()).exist(b2,"AB"))
            System.out.println("found AB");
        else
            System.out.println("can not found AB");
    }
}


\08-Recurion-and-Backstracking\07-Number-of-Islands\src\Solution.java

/// 200. Number of Islands
/// https://leetcode.com/problems/number-of-islands/description/
/// 时间复杂度: O(n*m)
/// 空间复杂度: O(n*m)
class Solution {

    private int d[][] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
    private int m, n;
    private boolean visited[][];

    public int numIslands(char[][] grid) {

        if(grid == null || grid.length == 0 || grid[0].length == 0)
            return 0;

        m = grid.length;
        n = grid[0].length;

        visited = new boolean[m][n];

        int res = 0;
        for(int i = 0 ; i < m ; i ++)
            for(int j = 0 ; j < n ; j ++)
                if(grid[i][j] == '1' && !visited[i][j]){
                    dfs(grid, i, j);
                    res ++;
                }

        return res;
    }

    // 从grid[x][y]的位置开始,进行floodfill
    // 保证(x,y)合法,且grid[x][y]是没有被访问过的陆地
    private void dfs(char[][] grid, int x, int y){

        //assert(inArea(x,y));
        visited[x][y] = true;
        for(int i = 0; i < 4; i ++){
            int newx = x + d[i][0];
            int newy = y + d[i][1];
            if(inArea(newx, newy) && !visited[newx][newy] && grid[newx][newy] == '1')
                dfs(grid, newx, newy);
        }

        return;
    }

    private boolean inArea(int x, int y){
        return x >= 0 && x < m && y >= 0 && y < n;
    }

    public static void main(String[] args) {

        char grid1[][] = {
            {'1','1','1','1','0'},
            {'1','1','0','1','0'},
            {'1','1','0','0','0'},
            {'0','0','0','0','0'}
        };
        System.out.println((new Solution()).numIslands(grid1));
        // 1

        // ---

        char grid2[][] = {
            {'1','1','0','0','0'},
            {'1','1','0','0','0'},
            {'0','0','1','0','0'},
            {'0','0','0','1','1'}
        };
        System.out.println((new Solution()).numIslands(grid2));
        // 3
    }
}


\08-Recurion-and-Backstracking\08-N-Queens\src\Solution.java

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

/// 51. N-Queens
/// https://leetcode.com/problems/n-queens/description/
/// 时间复杂度: O(n^n)
/// 空间复杂度: O(n)
public class Solution {

    private boolean[] col;
    private boolean[] dia1;
    private boolean[] dia2;
    private ArrayList<List<String>> res;

    public List<List<String>> solveNQueens(int n) {

        res = new ArrayList<List<String>>();
        col = new boolean[n];
        dia1 = new boolean[2 * n - 1];
        dia2 = new boolean[2 * n - 1];

        LinkedList<Integer> row = new LinkedList<Integer>();
        putQueen(n, 0, row);

        return res;
    }

    // 尝试在一个n皇后问题中, 摆放第index行的皇后位置
    private void putQueen(int n, int index, LinkedList<Integer> row){

        if(index == n){
            res.add(generateBoard(n, row));
            return;
        }

        for(int i = 0 ; i < n ; i ++)
            // 尝试将第index行的皇后摆放在第i列
            if(!col[i] && !dia1[index + i] && !dia2[index - i + n - 1]){
                row.addLast(i);
                col[i] = true;
                dia1[index + i] = true;
                dia2[index - i + n - 1] = true;
                putQueen(n, index + 1, row);
                col[i] = false;
                dia1[index + i] = false;
                dia2[index - i + n - 1] = false;
                row.removeLast();
            }

        return;
    }

    private List<String> generateBoard(int n, LinkedList<Integer> row){

        assert row.size() == n;

        ArrayList<String> board = new ArrayList<String>();
        for(int i = 0 ; i < n ; i ++){
            char[] charArray = new char[n];
            Arrays.fill(charArray, '.');
            charArray[row.get(i)] = 'Q';
            board.add(new String(charArray));
        }
        return board;
    }

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

    public static void main(String[] args) {

        int n = 4;
        List<List<String>> res = (new Solution()).solveNQueens(n);
        for(List<String> board: res)
            printBoard(board);
    }
}


\09-Dynamic-Programming\01-Fibonacci\src\Solution1.java

// 递归求斐波那契数列
public class Solution1 {

    private int num = 0;

    public int fib( int n ){

        num ++;
```

```java
        if( n == 0 )
            return 0;

        if( n == 1 )
            return 1;

        return fib(n-1) + fib(n-2);
    }

    public int getNum(){
        return num;
    }

    public static void main(String[] args) {

        int n = 42;

        Solution1 solution = new Solution1();
        long startTime = System.currentTimeMillis();
        int res = solution.fib(n);
        long endTime = System.currentTimeMillis();

        System.out.println("fib(" + n + ") = " + res);
        System.out.println("time : " + (endTime - startTime) + " ms");
        System.out.println("run function fib() " + solution.getNum() + " times.");
    }
}


\09-Dynamic-Programming\01-Fibonacci\src\Solution2.java

import java.util.Arrays;

// 记忆化搜索
public class Solution2 {

    private int num = 0;

    public int fib(int n){

        int[] memo = new int[n + 1];
        Arrays.fill(memo, -1);
        return fib(n, memo);
    }

    private int fib(int n, int[] memo){

        num ++;

        if(n == 0)
            return 0;

        if(n == 1)
            return 1;

        if(memo[n] == -1)
            memo[n] = fib(n - 1, memo) + fib(n - 2, memo);

        return memo[n];
    }

    public int getNum(){
        return num;
    }

    public static void main(String[] args) {

        //int n = 42;
        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
                      // 斐波那契额数列是以指数速度上涨的

        Solution2 solution = new Solution2();
        long startTime = System.currentTimeMillis();
        int res = solution.fib(n);
        long endTime = System.currentTimeMillis();

        System.out.println("fib(" + n + ") = " + res);
        System.out.println("time : " + (endTime - startTime) + " ms");
        System.out.println("run function fib() " + solution.getNum() + " times.");
    }
}


\09-Dynamic-Programming\01-Fibonacci\src\Solution3.java

import java.util.Arrays;

// 动态规划
public class Solution3 {

    public int fib(int n){

        int[] memo = new int[n + 1];
        Arrays.fill(memo, -1);

        memo[0] = 0;
        memo[1] = 1;
        for(int i = 2 ; i <= n ; i ++)
            memo[i] = memo[i - 1] + memo[i - 2];

        return memo[n];
    }

    public static void main(String[] args) {

        //int n = 42;
        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
                      // 斐波那契额数列是以指数速度上涨的

        Solution3 solution = new Solution3();
        long startTime = System.currentTimeMillis();
        int res = solution.fib(n);
        long endTime = System.currentTimeMillis();

        System.out.println("fib(" + n + ") = " + res);
        System.out.println("time : " + (endTime - startTime) + " ms");
    }
}


\09-Dynamic-Programming\02-Climbing-Stairs\src\Solution1.java

import java.util.Arrays;

/**
 * Created by liuyubobobo.
 */
public class Solution1 {

    private int[] memo;

    public int climbStairs(int n) {
        memo = new int[n+1];
        Arrays.fill(memo, -1);
        return calcWays(n);
    }

    private int calcWays(int n){

        if(n == 0 || n == 1)
            return 1;

        if(memo[n] == -1)
            memo[n] = calcWays(n - 1) + calcWays(n - 2);

        return memo[n];
    }

    public static void main(String[] args) {

        System.out.println((new Solution1()).climbStairs(10));
    }
}


\09-Dynamic-Programming\02-Climbing-Stairs\src\Solution2.java

/// 70. Climbing Stairs
/// https://leetcode.com/problems/climbing-stairs/description/
/// 动态规划
/// 时间复杂度: O(n)
/// 空间复杂度: O(n)
public class Solution2 {

    public int climbStairs(int n) {

        int[] memo = new int[n + 1];
        memo[0] = 1;
        memo[1] = 1;
        for(int i = 2 ; i <= n ; i ++)
            memo[i] = memo[i - 1] + memo[i - 2];
        return memo[n];
    }

    public static void main(String[] args) {

        System.out.println((new Solution2()).climbStairs(10));
    }
}


\09-Dynamic-Programming\03-Integer-Break\src\Solution1.java

/// 343. Integer Break
/// https://leetcode.com/problems/integer-break/description/
/// 暴力搜索
/// 在Leetcode中提交这个版本的代码会超时! (Time Limit Exceeded)
/// 时间复杂度: O(n^n)
/// 空间复杂度: O(n)
public class Solution1 {

    public int integerBreak(int n) {

        if(n < 1)
            throw new IllegalArgumentException("n should be greater than zero");

        return breakInteger(n);
    }

    // 将n进行分割(至少分割两部分), 可以获得的最大乘积
    private int breakInteger(int n){

        if(n == 1)
            return 1;

        int res = -1;
        for(int i = 1 ; i <= n - 1 ; i ++)
            res = max3(res, i * (n - i), i * breakInteger(n - i));
        return res;
    }

    private int max3(int a, int b, int c){
        return Math.max(a, Math.max(b, c));
    }

    public static void main(String[] args) {

        System.out.println((new Solution1()).integerBreak(2));
        System.out.println((new Solution1()).integerBreak(10));
    }
}


\09-Dynamic-Programming\03-Integer-Break\src\Solution2.java

import java.util.Arrays;

/// 343. Integer Break
/// https://leetcode.com/problems/integer-break/description/
/// 记忆化搜索
/// 时间复杂度: O(n^2)
/// 空间复杂度: O(n)
public class Solution2 {

    private int[] memo;

    public int integerBreak(int n) {

        if(n < 1)
            throw new IllegalArgumentException("n should be greater than zero");

        memo = new int[n+1];
        Arrays.fill(memo, -1);

        return breakInteger(n);
    }

    // 将n进行分割(至少分割两部分), 可以获得的最大乘积
    private int breakInteger(int n){

        if(n == 1)
            return 1;

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

        int res = -1;
        for(int i = 1 ; i <= n - 1 ; i ++)
            res = max3(res, i * (n - i) , i * breakInteger(n - i));
        memo[n] = res;
        return res;
    }

```

```java
    private int max3(int a, int b, int c){
        return Math.max(a, Math.max(b, c));
    }

    public static void main(String[] args) {

        System.out.println((new Solution2()).integerBreak(2));
        System.out.println((new Solution2()).integerBreak(10));
    }
}


\09-Dynamic-Programming\03-Integer-Break\src\Solution3.java

/// 343. Integer Break
/// https://leetcode.com/problems/integer-break/description/
/// 动态规划
/// 时间复杂度: O(n^2)
/// 空间复杂度: O(n)
public class Solution3 {

    public int integerBreak(int n) {

        if(n < 1)
            throw new IllegalArgumentException("n should be greater than zero");

        int[] memo = new int[n+1];
        memo[1] = 1;
        for(int i = 2 ; i <= n ; i ++)
            // 求解memo[i]
            for(int j = 1 ; j <= i - 1 ; j ++)
                memo[i] = max3(memo[i], j * (i - j), j * memo[i - j]);

        return memo[n];
    }

    private int max3(int a, int b, int c){
        return Math.max(a, Math.max(b, c));
    }

    public static void main(String[] args) {

        System.out.println((new Solution3()).integerBreak(2));
        System.out.println((new Solution3()).integerBreak(10));
    }
}


\09-Dynamic-Programming\04-House-Robber\src\Solution1.java

import java.util.Arrays;

/// 198. House Robber
/// https://leetcode.com/problems/house-robber/description/
/// 记忆化搜索
/// 时间复杂度: O(n^2)
/// 空间复杂度: O(n)
public class Solution1 {

    // memo[i] 表示考虑抢劫 nums[i...n) 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, 0);
    }

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

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

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

        int res = 0;
        for(int i = index ; i < nums.length ; 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 Solution1()).rob(nums));
    }
}


\09-Dynamic-Programming\04-House-Robber\src\Solution2.java

import java.util.Arrays;

/// 198. House Robber
/// https://leetcode.com/problems/house-robber/description/
/// 动态规划
/// 时间复杂度: O(n^2)
/// 空间复杂度: O(n)
public class Solution2 {

    public int rob(int[] nums) {

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

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

        return memo[0];
    }

    public static void main(String[] args) {

        int nums[] = {2, 1};
        System.out.println((new Solution2()).rob(nums));
    }
}


\09-Dynamic-Programming\04-House-Robber\src\Solution3.java

import java.util.Arrays;

```

```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];
    }

    public static void main(String[] args) {

        int nums[] = {2, 1};
        System.out.println((new Solution4()).rob(nums));
    }
}


\09-Dynamic-Programming\04-House-Robber\src\Solution5.java

import java.util.Arrays;

/// 198. House Robber
/// https://leetcode.com/problems/house-robber/description/
/// 记忆化搜索, 优化状态转移
/// 时间复杂度: O(n)
/// 空间复杂度: O(n)
public class Solution5 {

    // memo[i] 表示考虑抢劫 nums[i...n) 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, 0);
    }

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

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

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

        // 或者当前房子放弃, 从下一个房子开始考虑
        // 或者抢劫当前的房子, 从i+2以后的房子开始考虑
        return memo[index] =
                Math.max(tryRob(nums, index + 1),
                         nums[index] + tryRob(nums, index + 2));
    }

    public static void main(String[] args) {

        int nums[] = {2, 1};
        System.out.println((new Solution5()).rob(nums));
    }
}


\09-Dynamic-Programming\04-House-Robber\src\Solution6.java

import java.util.Arrays;

/// 198. House Robber
/// https://leetcode.com/problems/house-robber/description/
/// 动态规划, 优化状态转移
/// 时间复杂度: O(n)
/// 空间复杂度: O(n)
public class Solution6 {

    public int rob(int[] nums) {

        int n = nums.length;
```

```java
        if(n == 0)
            return 0;

        // memo[i] 表示考虑抢劫 nums[i...n) 所能获得的最大收益
        int[] memo = new int[nums.length];
        memo[n - 1] = nums[n - 1];
        for(int i = n - 2 ; i >= 0 ; i --)
            // 或者当前房子放弃, 从下一个房子开始考虑
            // 或者抢劫当前的房子, 从i+2以后的房子开始考虑
            memo[i] = Math.max(memo[i + 1],
                               nums[i] + (i + 2 < n ? memo[i + 2] : 0));

        return memo[0];
    }

    public static void main(String[] args) {

        int nums[] = {2, 1};
        System.out.println((new Solution6()).rob(nums));
    }
}


\09-Dynamic-Programming\04-House-Robber\src\Solution7.java

import java.util.Arrays;

/// 198. House Robber
/// https://leetcode.com/problems/house-robber/description/
/// 记忆化搜索, 改变状态定义, 优化转移方程
/// 时间复杂度: O(n)
/// 空间复杂度: O(n)
public class Solution7 {

    // 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];

        // 或者当前房子放弃, 考虑[0...index-1]的所有房子
        // 或者抢劫当前的房子, 考虑[0...index-2]的所有房子
        return memo[index] =
                Math.max(tryRob(nums, index - 1),
                         nums[index] + tryRob(nums, index - 2));
    }

    public static void main(String[] args) {

        int nums[] = {2, 1};
        System.out.println((new Solution7()).rob(nums));
    }
}

```

```

\09-Dynamic-Programming\04-House-Robber\src\Solution8.java

/// 198. House Robber
/// https://leetcode.com/problems/house-robber/description/
/// 动态规划, 改变状态定义, 优化转移方程
/// 时间复杂度: O(n)
/// 空间复杂度: O(n)
public class Solution8 {

    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 ++)
            memo[i] = Math.max(memo[i - 1],
                               nums[i] + (i - 2 >= 0 ? memo[i - 2] : 0));

        return memo[n-1];
    }

    public static void main(String[] args) {

        int nums[] = {2, 1};
        System.out.println((new Solution8()).rob(nums));
    }
}


\09-Dynamic-Programming\05-0-1-knapsack\src\Solution1.java

/// 背包问题
/// 记忆化搜索
/// 时间复杂度: O(n * C) 其中n为物品个数; C为背包容积
/// 空间复杂度: O(n * C)
public class Solution1 {

    private int[][] memo;

    public int knapsack01(int[] w, int[] v, int C){

        if(w == null || v == null || w.length != v.length)
            throw new IllegalArgumentException("Invalid w or v");

        if(C < 0)
            throw new IllegalArgumentException("C must be greater or equal to zero.");

        int n = w.length;
        if(n == 0 || C == 0)
            return 0;

        memo = new int[n][C + 1];
        return bestValue(w, v, n - 1, C);
    }

    // 用 [0...index]的物品,填充容积为c的背包的最大价值
    private int bestValue(int[] w, int[] v, int index, int c){

        if(c <= 0 || index < 0)
            return 0;

```

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

        int res = bestValue(w, v, index-1, c);
        if(c >= w[index])
            res = Math.max(res, v[index] + bestValue(w, v, index - 1, c - w[index]));

        return memo[index][c] = res;
    }

    public static void main(String[] args) {

    }

}


\09-Dynamic-Programming\05-0-1-knapsack\src\Solution2.java

/// 背包问题
/// 动态规划
/// 时间复杂度: O(n * C) 其中n为物品个数; C为背包容积
/// 空间复杂度: O(n * C)
public class Solution2 {

    public int knapsack01(int[] w, int[] v, int C){

        if(w == null || v == null || w.length != v.length)
            throw new IllegalArgumentException("Invalid w or v");

        if(C < 0)
            throw new IllegalArgumentException("C must be greater or equal to zero.");

        int n = w.length;
        if(n == 0 || C == 0)
            return 0;

        int[][] memo = new int[n][C + 1];

        for(int j = 0 ; j <= C ; j ++)
            memo[0][j] = (j >= w[0] ? v[0] : 0 );

        for(int i = 1 ; i < n ; i ++)
            for(int j = 0 ; j <= C ; j ++){
                memo[i][j] = memo[i-1][j];
                if(j >= w[i])
                    memo[i][j] = Math.max(memo[i][j], v[i] + memo[i - 1][j - w[i]]);
            }

        return memo[n - 1][C];
    }

    public static void main(String[] args) {

    }
}


\09-Dynamic-Programming\06-0-1-knapsack-optimized\src\Solution1.java

/// 背包问题
/// 动态规划改进: 滚动数组
/// 时间复杂度: O(n * C) 其中n为物品个数; C为背包容积
/// 空间复杂度: O(C), 实际使用了2*C的额外空间
public class Solution1 {

    public int knapsack01(int[] w, int[] v, int C){

        if(w == null || v == null || w.length != v.length)
            throw new IllegalArgumentException("Invalid w or v");

        if(C < 0)
            throw new IllegalArgumentException("C must be greater or equal to zero.");

        int n = w.length;
        if(n == 0 || C == 0)
            return 0;

        int[][] memo = new int[2][C + 1];

        for(int j = 0 ; j <= C ; j ++)
            memo[0][j] = (j >= w[0] ? v[0] : 0);

        for(int i = 1 ; i < n ; i ++)
            for(int j = 0 ; j <= C ; j ++){
                memo[i % 2][j] = memo[(i-1) % 2][j];
                if(j >= w[i])
                    memo[i % 2][j] = Math.max(memo[i % 2][j], v[i] + memo[(i-1) % 2][j - w[i]]);
            }

        return memo[(n-1) % 2][C];
    }

    public static void main(String[] args) {

    }
}


\09-Dynamic-Programming\06-0-1-knapsack-optimized\src\Solution2.java

/// 背包问题
/// 动态规划改进
/// 时间复杂度: O(n * C) 其中n为物品个数; C为背包容积
/// 空间复杂度: O(C), 只使用了C的额外空间
public class Solution2 {

    public int knapsack01(int[] w, int[] v, int C){

        if(w == null || v == null || w.length != v.length)
            throw new IllegalArgumentException("Invalid w or v");

        if(C < 0)
            throw new IllegalArgumentException("C must be greater or equal to zero.");

        int n = w.length;
        if(n == 0 || C == 0)
            return 0;

        int[] memo = new int[C+1];

        for(int j = 0 ; j <= C ; j ++)
            memo[j] = (j >= w[0] ? v[0] : 0);

        for(int i = 1 ; i < n ; i ++)
            for(int j = C ; j >= w[i] ; j --)
                memo[j] = Math.max(memo[j], v[i] + memo[j - w[i]]);

        return memo[C];
    }

    public static void main(String[] args) {
    }
}


\09-Dynamic-Programming\07-Partition-Equal-Subset-Sum\src\Solution1.java

import java.util.Arrays;

/// 416. Partition Equal Subset Sum
/// https://leetcode.com/problems/partition-equal-subset-sum/description/
/// 记忆化搜索
/// 时间复杂度: O(len(nums) * O(sum(nums)))
/// 空间复杂度: O(len(nums) * O(sum(nums)))
public class Solution1 {

    // 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
    // -1 表示为未计算; 0 表示不可以填充; 1 表示可以填充
    private int[][] memo;

    public boolean canPartition(int[] nums) {

        int sum = 0;
        for(int i = 0 ; i < nums.length ; i ++){
            if(nums[i] <= 0)
                throw new IllegalArgumentException("numbers in nums must be greater than zero.");
            sum += nums[i];
        }

        if(sum % 2 == 1)
            return false;

        memo = new int[nums.length][sum / 2 + 1];
        for(int i = 0 ; i < nums.length ; i ++)
            Arrays.fill(memo[i], -1);
        return tryPartition(nums, nums.length - 1, sum / 2);
    }

    // 使用nums[0...index], 是否可以完全填充一个容量为sum的背包
    private boolean tryPartition(int[] nums, int index, int sum){

        if(sum == 0)
            return true;

        if(sum < 0 || index < 0)
            return false;

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

        memo[index][sum] = (tryPartition(nums, index - 1, sum) ||
                tryPartition(nums, index - 1, sum - nums[index])) ? 1 : 0;

        return memo[index][sum] == 1;
    }

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

    public static void main(String[] args) {

        int[] nums1 = {1, 5, 11, 5};
        printBool((new Solution1()).canPartition(nums1));

        int[] nums2 = {1, 2, 3, 5};
```

```java
        printBool((new Solution1()).canPartition(nums2));
    }
}


\09-Dynamic-Programming\07-Partition-Equal-Subset-Sum\src\Solution2.java

import java.util.Arrays;

/// 416. Partition Equal Subset Sum
/// https://leetcode.com/problems/partition-equal-subset-sum/description/
/// 动态规划
/// 时间复杂度: O(len(nums) * O(sum(nums)))
/// 空间复杂度: O(len(nums) * O(sum(nums)))
public class Solution2 {

    public boolean canPartition(int[] nums) {

        int sum = 0;
        for(int i = 0 ; i < nums.length ; i ++){
            if(nums[i] <= 0)
                throw new IllegalArgumentException("numbers in nums must be greater than zero.");
            sum += nums[i];
        }

        if(sum % 2 == 1)
            return false;

        int n = nums.length;
        int C = sum / 2;

        boolean[] memo = new boolean[C + 1];
        for(int i = 0 ; i <= C ; i ++)
            memo[i] = (nums[0] == i);

        for(int i = 1 ; i < n ; i ++)
            for(int j = C; j >= nums[i] ; j --)
                memo[j] = memo[j] || memo[j - nums[i]];

        return memo[C];
    }

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

    public static void main(String[] args) {

        int[] nums1 = {1, 5, 11, 5};
        printBool((new Solution2()).canPartition(nums1));

        int[] nums2 = {1, 2, 3, 5};
        printBool((new Solution2()).canPartition(nums2));
    }
}


\09-Dynamic-Programming\08-Longest-Increasing-Subsequence\src\Solution1.java

import java.util.Arrays;

/// 300. Longest Increasing Subsequence
/// https://leetcode.com/problems/longest-increasing-subsequence/description/
/// 记忆化搜索
/// 时间复杂度: O(n^2)
/// 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
/// 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)
```

```java
            throw new IllegalArgumentException("s1 and s2 can not be null.");

        if(s1.length() == 0 || s2.length() == 0)
            return "";

        memo = new int[s1.length()][s2.length()];
        for(int i = 0 ; i < s1.length() ; i ++)
            Arrays.fill(memo[i], -1);

        lcs(s1, s2, s1.length() - 1, s2.length() - 1);
        return getLCS(s1, s2);
    }

    // 求s1[0...m]和s2[0...n]的最长公共子序列的长度值
    private int lcs(String s1, String s2, int m, int n){

        if(m < 0 || n < 0)
            return 0;

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

        int res = 0;
        if(s1.charAt(m) == s2.charAt(n))
            res = 1 + lcs(s1, s2, m - 1, n - 1);
        else
            res = Math.max(lcs(s1, s2, m - 1, n),
                           lcs(s1, s2, m, n - 1));

        memo[m][n] = res;
        return res;
    }

    // 通过memo反向求解s1和s2的最长公共子序列
    private String getLCS(String s1, String s2){

        int m = s1.length() - 1;
        int n = s2.length() - 1;

        StringBuilder res = new StringBuilder("");
        while(m >= 0 && n >= 0)
            if(s1.charAt(m) == s2.charAt(n)){
                res = res.insert(0, s1.charAt(m));
                m --;
                n --;
            }
            else if(m == 0)
                n --;
            else if(n == 0)
                m --;
            else{
                if(memo[m-1][n] > memo[m][n-1])
                    m --;
                else
                    n --;
            }

        return res.toString();
    }

    public static void main(String[] args) {

        String s1 = "ABCDGH";
        String s2 = "AEDFHR";
        System.out.println((new LCS1()).lcs(s1, s2));

        s1 = "AAACCGTGAGTTATTCGTTCTAGAA";
        s2 = "CACCCCTAAGGTACCTTTGGTTC";
        System.out.println((new LCS1()).lcs(s1, s2));
    }
}


\09-Dynamic-Programming\09-Longest-Common-Subsequence\src\LCS2.java

/// 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 LCS2 {

    public String lcs(String s1, String s2){

        int m = s1.length();
        int n = s2.length();

        // 对memo的第0行和第0列进行初始化
        int[][] memo = new int[m][n];
        for(int j = 0 ; j < n ; j ++)
            if(s1.charAt(0) == s2.charAt(j)){
                for(int k = j ; k < n ; k ++)
                    memo[0][k] = 1;
                break;
            }

        for(int i = 0 ; i < m ; i ++)
            if(s1.charAt(i) == s2.charAt(0)) {
                for(int k = i ; k < m ; k ++)
                    memo[k][0] = 1;
                break;
            }

        // 动态规划的过程
        for(int i = 1 ; i < m ; i ++)
            for(int j = 1 ; j < n ; j ++)
                if(s1.charAt(i) == s2.charAt(j))
                    memo[i][j] = 1 + memo[i-1][j-1];
                else
                    memo[i][j] = Math.max(memo[i-1][j], memo[i][j-1]);

        // 通过memo反向求解s1和s2的最长公共子序列
        m = s1.length() - 1;
        n = s2.length() - 1;
        StringBuilder res = new StringBuilder("");
        while(m >= 0 && n >= 0)
            if(s1.charAt(m) == s2.charAt(n)){
                res.insert(0, s1.charAt(m));
                m --;
                n --;
            }
            else if(m == 0)
                n --;
            else if(n == 0)
                m --;
            else{
                if(memo[m-1][n] > memo[m][n-1])
                    m --;
                else
                    n --;
            }

        return res.toString();
```

```java
    }

    public static void main(String[] args) {

        String s1 = "ABCDGH";
        String s2 = "AEDFHR";
        System.out.println((new LCS2()).lcs(s1, s2));

        s1 = "AAACCGTGAGTTATTCGTTCTAGAA";
        s2 = "CACCCCTAAGGTACCTTTGGTTC";
        System.out.println((new LCS2()).lcs(s1, s2));
    }
}


\09-Dynamic-Programming\09-Longest-Common-Subsequence\src\LCS3.java

/// LCS问题
/// 动态规划，躲避边界条件
/// 时间复杂度: O(len(s1)*len(s2))
/// 空间复杂度: O(len(s1)*len(s2))
public class LCS3 {

    public String lcs(String s1, String s2){

        int m = s1.length();
        int n = s2.length();

        // memo 是 (m + 1) * (n + 1) 的动态规划表格
        // 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
        // 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
        // memo[i][0] 表示s2取空字符串时，和s1的前i个字符作比较
        // 所以，memo[0][j] 和 memo[i][0] 均取0
        // 我们不需要对memo进行单独的边界条件处理 :-)
        int[][] memo = new int[m + 1][n + 1];

        // 动态规划的过程
        // 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
        for(int i = 1 ; i <= m ; i ++)
            for(int j = 1 ; j <= n ; j ++)
                if(s1.charAt(i - 1) == s2.charAt(j - 1))
                    memo[i][j] = 1 + memo[i - 1][j - 1];
                else
                    memo[i][j] = Math.max(memo[i - 1][j], memo[i][j - 1]);

        // 通过memo反向求解s1和s2的最长公共子序列
        m = s1.length();
        n = s2.length();
        StringBuilder res = new StringBuilder("");
        while(m > 0 && n > 0)
            if(s1.charAt(m - 1) == s2.charAt(n - 1)){
                res.insert(0, s1.charAt(m - 1));
                m --;
                n --;
            }
            else if(memo[m - 1][n] > memo[m][n - 1])
                m --;
            else
                n --;

        return res.toString();
    }

    public static void main(String[] args) {

        String s1 = "ABCDGH";
        String s2 = "AEDFHR";
        System.out.println((new LCS3()).lcs(s1, s2));

        s1 = "AAACCGTGAGTTATTCGTTCTAGAA";
        s2 = "CACCCCTAAGGTACCTTTGGTTC";
        System.out.println((new LCS3()).lcs(s1, s2));
    }
}


\09-Dynamic-Programming\Optional-01-More-about-Fibonacci\src\Solution1.java

/// 70. Climbing Stairs
/// https://leetcode.com/problems/climbing-stairs/description/
///
/// 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
/// 只不过 climbStairs(n) 的答案，对应第 n+1 个斐波那契数
/// 其中 f0 = 0, f(1) = 1, f(2) = 1, f(3) = 2...
/// 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
/// 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
/// 我们对背包问题的空间优化，从O(n^2)优化到了O(n)
/// 我们对斐波那契问题的优化,可以从O(n)优化到O(1)
/// 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,
/// 更小的斐波那契数不需要一直保存。
///
/// 时间复杂度: O(n)
/// 空间复杂度: O(1)
public class Solution1 {

    public int climbStairs(int n) {

        if(n <= 0)
            throw new IllegalArgumentException("n must be greater than zero");

        if(n == 1)
            return 1;

        int prev = 1, cur = 1;
        for(int i = 3 ; i <= n + 1; i ++){
            int f = cur + prev;
            prev = cur;
            cur = f;
        }
        return cur;
    }

    public static void main(String[] args) {

        System.out.println((new Solution1()).climbStairs(10));
    }
}


\09-Dynamic-Programming\Optional-01-More-about-Fibonacci\src\Solution2.java

/// 70. Climbing Stairs
/// https://leetcode.com/problems/climbing-stairs/description/
///
/// 斐波那契数可以根据一个特殊矩阵的幂的形式求出。
/// | F(n+1) F(n)   | = | 1 1 |^n
/// | F(n)   F(n-1) |   | 1 0 |
/// 幂运算可以使用分治法，优化为O(logn)的复杂度
/// 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。
///
/// 时间复杂度: O(logn)
/// 空间复杂度: O(1)
```

```
public class Solution2 {

    public int climbStairs(int n) {

        if(n <= 0)
            throw new IllegalArgumentException("n must be greater than zero");

        if(n == 1)
            return 1;

        int[][] base = {{1, 1}, {1, 0}};
        return matrix_pow(base, n)[0][0];
    }

    private int[][] matrix_pow(int[][] m, int n){

        if(n == 1)
            return m;

        int[][] t = matrix_pow(m, n / 2);
        int[][] res = matrix_multiply(t, t);
        if(n % 2 == 1)
            return matrix_multiply(res, m);
        return res;
    }

    int[][] matrix_multiply(int[][] m1, int[][] m2){
        int[][] res = new int[2][2];
        res[0][0] = m1[0][0] * m2[0][0] + m1[0][1] * m2[1][0];
        res[0][1] = m1[0][0] * m2[0][1] + m1[0][1] * m2[1][1];
        res[1][0] = m1[1][0] * m2[0][0] + m1[1][1] * m2[1][0];
        res[1][1] = m1[1][0] * m2[0][1] + m1[1][1] * m2[1][1];
        return res;
    }

    public static void main(String[] args) {

        System.out.println((new Solution2()).climbStairs(10));
    }
}


\09-Dynamic-Programming\Optional-01-More-about-Fibonacci\src\Solution3.java

/// 70. Climbing Stairs
/// https://leetcode.com/problems/climbing-stairs/description/
///
/// 对于第n个斐波那契数，可以推导出其公式
/// Fn = 1/sqrt(5) * {[(1+sqrt(5))/2]^n - [(1-sqrt(5))/2]^n}
/// 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。
/// 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
/// 但这个方法快于使用矩阵的幂运算符的方法
///
/// 时间复杂度：O(logn)
/// 空间复杂度：O(1)
public class Solution3 {

    public int climbStairs(int n) {

        if(n <= 0)
            throw new IllegalArgumentException("n must be greater than zero");

        if(n == 1)
            return 1;

        double sqrt5 = Math.sqrt(5.0);
        return (int)((Math.pow((1 + sqrt5) / 2, n + 1) - Math.pow((1 - sqrt5) / 2, n + 1)) / sqrt5);
    }

    public static void main(String[] args) {

        System.out.println((new Solution3()).climbStairs(10));
    }
}


\09-Dynamic-Programming\Optional-02-More-about-LIS\src\Solution.java

import java.util.Arrays;

/// 300. Longest Increasing Subsequence
/// https://leetcode.com/problems/longest-increasing-subsequence/description/
///
/// 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
/// 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
/// 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
/// 通过这个例子, 也请大家再体会改变动态规划的状态定义,
/// 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
///
/// 时间复杂度: O(nlogn)
/// 空间复杂度: O(n)
public class Solution {

    public int lengthOfLIS(int[] nums) {

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

        // 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
        int dp[] = new int[nums.length + 1];
        Arrays.fill(dp, Integer.MIN_VALUE);

        int len = 1;
        dp[1] = nums[0];
        for(int i = 1 ; i < nums.length ; i ++)
            if(nums[i] > dp[len]){
                len ++;
                dp[len] = nums[i];
            }
            else{
                // 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
                int index = lowerBound(dp, 0, len, nums[i]);
                if(dp[index] != nums[i])
                    dp[index] = Math.min(dp[index], nums[i]);
            }

        return len;
    }

    // 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
    private int lowerBound(int[] arr, int l, int r, int target){

        int left = l, right = r + 1;
        while(left != right){
            int mid = left + (right - left) / 2;
            if(arr[mid] >= target)
                right = mid;
            else // arr[mid] < target
                left = mid + 1;
        }
        return left;
    }
```

```java

    public static void main(String[] args) {

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

        // ---

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

        // ---

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

        // ---

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


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

import java.util.Arrays;

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

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

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

        int gi = g.length - 1, si = s.length - 1;
        int res = 0;
        while(gi >= 0 && si >= 0){
            if(s[si] >= g[gi]){
                res ++;
                si --;
            }
            gi --;
        }

        return res;
    }

    public static void main(String[] args) {

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

        int g2[] = {1, 2};
        int s2[] = {1, 2, 3};
        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; }
    }

    public int eraseOverlapIntervals(Interval[] intervals) {

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

        Arrays.sort(intervals, new Comparator<Interval>() {
            @Override
            public int compare(Interval o1, Interval o2) {
                if(o1.start != o2.start)
                    return o1.start - o2.start;
                return o1.end - o2.end;
            }
        });

        // 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
        int[] memo = new int[intervals.length];
        Arrays.fill(memo, 1);
        for(int i = 1 ; i < intervals.length ; i ++)
            // memo[i]
            for(int j = 0 ; j < i ; j ++)
                if(intervals[i].start >= intervals[j].end)
                    memo[i] = Math.max(memo[i], 1 + memo[j]);

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

        return intervals.length - res;
    }

    public static void main(String[] args) {
        Interval[] interval1 = {new Interval(1,2),
                                new Interval(2,3),
                                new Interval(3,4),
                                new Interval(1,3)};
        System.out.println((new Solution1()).eraseOverlapIntervals(interval1));

        Interval[] interval2 = {new Interval(1,2),
                                new Interval(1,2),
                                new Interval(1,2)};
        System.out.println((new Solution1()).eraseOverlapIntervals(interval2));

        Interval[] interval3 = {new Interval(1,2),
                                new Interval(2,3)};
        System.out.println((new Solution1()).eraseOverlapIntervals(interval3));
    }
}


\10-Greedy-Algorithms\02-Non-overlapping-Intervals\src\Solution2.java

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

/// 435. Non-overlapping Intervals
/// https://leetcode.com/problems/non-overlapping-intervals/description/
/// 贪心算法
/// 时间复杂度: O(n)
/// 空间复杂度: O(n)
public class Solution2 {

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

```java
        Interval(int s, int e) { start = s; end = e; }
    }

    public int eraseOverlapIntervals(Interval[] intervals) {

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

        Arrays.sort(intervals, new Comparator<Interval>() {
            @Override
            public int compare(Interval o1, Interval o2) {
                if(o1.end != o2.end)
                    return o1.end - o2.end;
                return o1.start - o2.start;
            }
        });

        int res = 1;
        int pre = 0;
        for(int i = 1 ; i < intervals.length ; i ++)
            if(intervals[i].start >= intervals[pre].end){
                res ++;
                pre = i;
            }

        return intervals.length - res;
    }

    public static void main(String[] args) {
        Interval[] interval1 = {new Interval(1,2),
                new Interval(2,3),
                new Interval(3,4),
                new Interval(1,3)};
        System.out.println((new Solution2()).eraseOverlapIntervals(interval1));

        Interval[] interval2 = {new Interval(1,2),
                new Interval(1,2),
                new Interval(1,2)};
        System.out.println((new Solution2()).eraseOverlapIntervals(interval2));

        Interval[] interval3 = {new Interval(1,2),
                new Interval(2,3)};
        System.out.println((new Solution2()).eraseOverlapIntervals(interval3));
    }
}
```