题1

给你一个数组 nums 。
数组「动态和」的计算公式为：runningSum[i] = sum(nums[0]…nums[i]) 。
请返回 nums 的动态和。

示例 1：

- 输入：nums = [1,2,3,4]
- 输出：[1,3,6,10]
- 解释：动态和计算过程为 [1, 1+2, 1+2+3, 1+2+3+4] 。

示例 2：

- 输入：nums = [1,1,1,1,1]
- 输出：[1,2,3,4,5]
- 解释：动态和计算过程为 [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1] 。

示例 3：

- 输入：nums = [3,1,2,10,1]
- 输出：[3,4,6,16,17]

提示：

- 1 <= nums.length <= 1000
- -10^6 <= nums[i] <= 10^6

函数形式：
public int[] runningSum(int[] nums)

题2

给你一个整数数组 nums 。
如果一组数字 (i,j) 满足 nums[i] == nums[j] 且 i < j ， 就可以认为这是一组 好数对 。
返回好数对的数目。

示例 1：

- 输入： nums = [1,2,3,1,1,3]
- 输出： 4
- 解释： 有 4 组好数对，分别是 (0,3), (0,4), (3,4), (2,5) ， 下标从 0 开始

示例 2：

- 输入： nums = [1,1,1,1]
- 输出： 6
- 解释： 数组中的每组数字都是好数对

示例 3：

- 输入：nums = [1,2,3]
- 输出：0

提示：

- 1 <= nums.length <= 100
- 1 <= nums[i] <= 100

函数形式：
public int numIdenticalPairs(int[] nums)

题3

给你一份旅游线路图，该线路图中的旅行线路用数组paths表示，其中 paths[i] = [cityAi, cityBi] 表示该线路将会从 cityAi直接前往cityBi 。请你找出这次旅行的终点站，即没有任何可以通往其他城市的线路的城市。
题目数据保证线路图会形成一条不存在循环的线路，因此只会有一个旅行终点站。

示例 1：

- 输入：paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]
- 输出："Sao Paulo"
- 解释：从 "London" 出发，最后抵达终点站 "Sao Paulo" 。本次旅行的路线是 "London" -> "New York" -> "Lima" -> "Sao Paulo" 。

示例 2：

- 输入：paths = [["B","C"],["D","B"],["C","A"]]
- 输出："A"
- 解释：所有可能的线路是：
  "D" -> "B" -> "C" -> "A".
  "B" -> "C" -> "A".
  "C" -> "A".
  "A".
  显然，旅行终点站是 "A" 。

示例 3：

- 输入：paths = [["A","Z"]]
- 输出："Z"

提示：
1 <= paths.length <= 100
paths[i].length == 2
1 <= cityAi.length, cityBi.length <= 10
cityAi != cityBi
所有字符串均由大小写英文字母和空格字符组成。

题4

有效括号字符串为空 ("")、"(" + A + ")" 或 A + B，其中 A 和 B 都是有效的括号字符串，+ 代表字符串的连接。例如，""，"()"，"(())()" 和 "(()(()))" 都是有效的括号字符串。
如果有效字符串S非空，且不存在将其拆分为 S = A+B 的方法，我们称其为原语（primitive），其中 A和B都是非空有效括号字符串。
给出一个非空有效字符串S，考虑将其进行原语化分解，使得：S = P_1 + P_2 + ... + P_k，其中P_i是有效括号字符串原语。
对S进行原语化分解，删除分解中每个原语字符串的最外层括号，返回S。

示例 1：

- 输入："(()())(())"
- 输出："()()()"
- 解释：
  输入字符串为 "(()())(())"，原语化分解得到 "(()())" + "(())"，
  删除每个部分中的最外层括号后得到 "()()" + "()" = "()()()"。

示例 2：

- 输入：`"(()())(())(()(()))"`
- 输出：`"()()()()(())"`
- 解释：
  输入字符串为 `"(()())(())(()(()))"`，原语化分解得到 `"(()())"` + `"(())"` + `"(()(()))"`，
  删除每个部分中的最外层括号后得到 `"()()"` + `"()"` + `"()(())"` = `"()()()()(())"`。

示例 3：

- 输入："()()"
- 输出：""
- 解释：
  输入字符串为 "()()"，原语化分解得到 "()" + "()"，
  删除每个部分中的最外层括号后得到 "" + "" = ""。