作业3-编程

# 问题1

## 问题重述

给定一个非负整数 n，计算各位数字都不同的数字 x 的个数，其中 0 ≤ x < 。要求使用**动态规划**来解决问题。

## 算法设计

我们设置状态**dp[i]**，表示长度为i的数字中含重复位数的数字总数。初始条件：

即长度为1的数字中，不存在位数相同情况的数字。

针对状态转移方程：我们注意到，i位数字可以被分为前i-1位数字和最后1位数字。如果i位数字中存在重复数字，等价于以下两种情况：

1. 前i-1位数字中有重复数字，则最后一位可以取0-9中的任意值；
2. 前i-1为数字中没有重复数字，则最后一位取前i位数字中的某一个。

由于表示前i-1位数字中存在位数相同情况的数字总数，并且长度为i-1的数字总数为，则长度为i-1的数字中不含重复位数的数字总数为。

因此，转移方程为：

需要注意我们最终求得的是长度为1-n中不含重复数字的个数，即：

## 算法代码

int n;

cin **>>** n;

if (n == 0) {

    cout **<<** 1;

    return 0;

}

**vector**<int> **dp**(n + 1, 0);

dp**[**1**]** = 0;

for (int i = 2; i <= n; ++i) {

    dp**[**i**]** = 10 \* dp**[**i - 1**]** + (9 \* **pow**(10, i - 2) - dp**[**i - 1**]**) \* (i - 1);

}

int sums = 0;

for (int i = 1; i <= n; ++i) {

    sums += dp**[**i**]**;

}

cout **<<** **pow**(10, n) - sums;

## 算法分析

通过主循环分析，循环次数为n，即时间复杂度为O(n)。

在动态规划过程中，建立了长度为n的数组，因此空间复杂度为O(n)。

## 运行截图

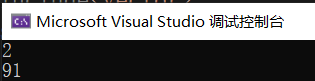
**1.5.1 函数测试**

**测试用例1**

输入：2

预期输出：91

实际输出：

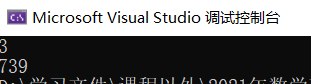


**测试用例2**

输入： 3

预期输出： 739

实际输出：



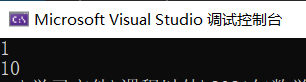
**1.5.2 边界测试**

**测试用例1**

输入：1

预期输出： 10

实际输出：

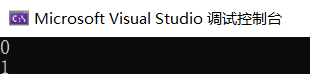


**测试用例2**

输入： 0

预期输出：1

实际输出：



# 问题2

## 问题重述

给定一个区间的集合，其中每个区间。找到需要移除区间的最小数量，使剩余区间互不重叠。

## 算法设计

通过分析，本问题可以通过贪心算法来解决。

贪心策略如下：**当出现区间重叠情况时，我们尽可能给后面的区间留出尽可能多的区间空余，因此应当优先保留结束时间最早的那个区间**。

因此，我们首先将原区间按照结束时间的先后进行排序。

当排序完成之后，我们只需要从头到尾遍历每一个区间，查看它是否与前一个区间重叠。在这一过程中，我们通过变量end记录上一个区间的结束时间。如果不重叠，则可以直接保留，并更新结束时间end；如果重叠，根据贪心策略保留前一个区间即可，无需更新结束时间end。

下面**证明通过上述贪心策略可以得到最优解**。

设原区间为，通过贪心策略我们选择留下了一些区间，记为A。设A不是最优解，即存在一个最优解B，它的区间总数大于A的区间总数。

假设A和B中前k-1个选择相同，第k个选择不同。

对于A而言，A的第k个选择为；对于B而言，B的第k个选择为。

根据我们的贪心策略可以知道，。

我们构造。

1. 很显然，O是这个问题的一个解，也即O中的区间没有重叠：

在O中，前的区间都和A中相同，因此前一部分没有重叠；同时，由于并且不会和O的后半部分发生重叠，因此也不会和后半部分发生重叠。

1. O是一个最优解，因为它的区间总数和B相同。

通过上述步骤，我们找到一个最优解O，它和A共同具有的区间为k个，与假设的前提k-1个相矛盾。

因此，通过贪心策略得到的结果A是最优解。证毕。

## 算法代码

int **removeOverlap**(**vector**<**vector**<int>>& intervals) {

    if (intervals.**empty**()) {

        return 0;

    }

*//按照结束时间排序*

**sort**(intervals.**begin**(), intervals.**end**(), [](const **vector**<int>& a, const **vector**<int>& b) {

        return a**[**1**]** < b**[**1**]**;

    });

*//记录上一个时间段的结束时间*

    int end = intervals**[**0**][**1**]**;

    int removeNum = 0;

    for (auto i = intervals.**begin**() **+** 1; i **!=** intervals.**end**(); **++**i) {

        if ((**\***i)**[**0**]** < end) {

            removeNum++;

        }

        else {

            end = (**\***i)**[**1**]**;

        }

    }

    return removeNum;

}

## 算法分析

在解决该问题时，我们先对区间进行了排序，之后遍历每一个区间。因此，本算法的时间复杂度为O(nlogn)。

本算法中，仅仅声明了常数变量，因此空间复杂度为O(n)。

## 运行截图

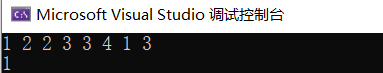
**2.5.1 函数测试**

**测试用例1**

输入：[[1,2],[2,3],[3,4],[1,3]]

预期输出：1

实际输出：

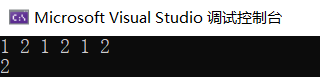


**测试用例2**

输入： [[1,2],[1,2],[1,2]]

预期输出： 2

实际输出：



**2.5.2 边界测试**

输入：[[1,2],[2,3]]

预期输出： 0

实际输出：

