课程实验报告

22920202200764 刘本宸

[leetcode300 -----最长上升子序列]

实验任务

1. 选择leetcode300 作为例子，并使用多种不同的方法求解之。
2. 用文字来描述你的算法思路，包括解空间、限界函数、算法主要步骤等（不限于以上方面）。
3. 在PC环境下使用C/C++语言编程实现算法。
4. 记录运行结果，包括输入数据，问题解答及运行时间。
5. 分析算法最坏情况下时间复杂度和空间复杂度。
6. 谈谈实验后的感想，包括关于该问题或类似问题的求解算法的建议。

实验环境

MacBookAir (M1,2020). macOS Monterey 12.3.1

g++ --version :

Apple clang version 13.1.6 (clang-1316.0.21.2.5)
Target: arm64-apple-darwin21.4.0
Thread model: posix

解空间

是子集树的模型，只取最后最长的子序列。所以解空间的大小是2^N， N为输入的数列长度。

解法0------暴力回溯

简述：暴力枚举所有可能的子集数列并且判断正确性。

这种方法直接爆掉了，tle，具体写法就是把下面的记忆化搜索的记忆去掉。与下面基本相同就不写了。

- 时间复杂度 O（N * 2^N）
- 空间复杂度 O（N）

解法1------dfs+memo:

```
class Solution {
public:
    vector<int> counter;
    int n;
    int dfs(vector<int> &nums,int index)
    {
        if(counter[index]!=0){return counter[index];}
        ++counter[index];
        for(int next=index+1;next<n;++next)
```

```
        {
            if(nums[next]>nums[index])
{counter[index]=max(counter[index],1+dfs(nums,next));}
        }
        return counter[index];
    }
    int lengthOfLIS(vector<int>& nums) {
        n=nums.size();
        counter.resize(n,0);
        int res=0;
        for(int i=0;i<n;++i)
        {
            res=max(res,dfs(nums,i));
        }
        return res;
    }
};
```

- 时间复杂度 O(N ^ 2) 空间复杂度O(N)

主要是枚举所有的可能的状态，是一个子集树的模型，但是这个地方注意因为用了备忘录，所以就少计算了很多层。

与下一个解法dp相差不是很多，可以想见多出的时间花在了递归中。

解法2------线性dp：

```
class Solution {
```

```
public:
    int lengthOfLIS(vector<int>& nums) {
        int n = (int)nums.size();
        if (!n) return 0;
        vector<int> dp(n, 0);
        for (int i = 0; i < n; ++i) {
            dp[i] = 1;
            for (int j = 0; j < i; ++j) {
                if (nums[j] < nums[i]) {
                    dp[i] = max(dp[i], dp[j] + 1);
                }
            }
        }
        return *max_element(dp.begin(), dp.end());
    }
};
```

- 时间复杂度 O(N ^ 2) 空间复杂度O(N)

从题目描述可以看出来状态转移方程:

dp[i] = max(dp[i] , dp[j] + 1) for j <--- 0 to i

其中我们的i要从1开始枚举到N。所以最后的时间复杂度是N^2

dp的快速做法

注：这个代码取自leetcode最快的代码实例，并非自己写的

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int max_len = 1;
```

```
        int sz = nums.size();
        int dp[sz];
        dp[0] = 1;
        for (int i=1; i<sz; ++i) {
            dp[i] = 1;
            for (int j=0; j<i; ++j) {
                if (nums[j]<nums[i]) {
                    dp[i] = max(dp[i], dp[j]+1);
                }
            }
            max_len = max(max_len, dp[i]);
        }
        return max_len;
    }
};
```

当然，也没快多少，属于是使用了相同数量级的空间，没啥好优化的。

解法3------贪心+二分：

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int len = 1, n = (int)nums.size();
        if (!n) return 0;
        vector<int> d(n + 1, 0);
        d[len] = nums[0];
        for (int i = 1; i < n; ++i) {
            if (nums[i] > d[len]) {
                d[++len] = nums[i];
            } else {
                int l = 1, r = len, pos = 0;
                while (l <= r) {
```

```
                int mid = (l + r) >> 1;
                if (d[mid] < nums[i]) {
                    pos = mid;
                    l = mid + 1;
                }
                else  r = mid - 1;
            }
            d[pos + 1] = nums[i];
        }
    }
    return len;
}
};
```

- 时间复杂度O(N * lg N) 空间复杂度 O(N)

贪心的思想，核心在于让子序列上升的足够慢。我们维护一个子序列d，每一次只往d的末尾加入可以加入的最小的值（这个值来自于每一个给定的数列中的数），寻找这个插入位置的时候我们可以针对d来进行二分。

为什么可以用二分？d是单调的

结论

类似问题，先从最暴力的开始想，然后想着剪枝，最后变成dp其实就可以了（对我来说）。

最后这种贪心的方法只求以后遇到相同的题目可以想到就行了。巧妙的解法实在是太多了。

后记

算法是真的难，真的费脑子。刷力扣学到的最多的一句话就是：我是傻逼

不过还好都挺过来了。希望以后面试算法岗的时候不会掉链子～

Reference

https://leetcode.cn/submissions/detail/314725177/