实验四　动态规划

May 26, 2022

1　前言

动态规划（dynamic programming）已经成为计算机科学中重要的算法设计范型。1957 年, Richard Bellman 在描述一类优化控制问题时创造了这个名字。那时，这个名字更多地用于描述问题，而不是解问题的技巧。规划（programming）的含义意味着一系列的决策，而动态（dynamic）的含义则传递着这样一种思想，就是所做决策可能依赖于当前状态，而与此前所做决策无关。此方法的主要特点是通过采用表格技术，用多项式算法代替指数算法。

动态规划典型的应用领域是组合优化问题，在这类问题中，可能有许多可行解（feasible solution），每个解对应一个值，我们想要找出一个具有最优值的解，称这个解为问题的一个最优解，可能有多个解都能达到这个最优值。本质上，动态规划计算所有子问题的解，计算的过程从小问题到大问题，并将计算结果存储在一张表中，此方法的优点在于，一旦一个子问题被解决，就存储其结果，此后遇到同样的子问题，就不再重复计算。在过去五十多年的进程中, 动态规划在运筹学、控制论、管理科学等领域的发展中, 都发挥了无可比拟的领军作用, 成为解决数学建模问题最常用的优化方法之一。但是, 作为一个重要的最优化方法, 它又存在着很多亟待解决的问题而显得很不完善。因此, 在运用这个方法的过程中, 人们一直致力于不断完善应用动态规划的条件以及动态规划问题的求解方法。诸如确定型一维动态规划的解析法、计算法等算法; 确定型多维动态规划的拉格朗日乘子、逐次迭代、策略与函数空间近似、多项式逼近、超曲面搜索等算法; 随机动态规划的基本算法等。

2018 年数学建模竞赛问题 B 智能 RGV 的动态调度策略问题，该问题对动态规划方法有着深入的应用，对参赛人员也提出了更高的要求，需要我们加以完善和优化。学习动态规划，这是一种解决棘手问题的方法，将问题分为小问题，并先着手解决这些小问题，聪明的你一定跃跃欲试，那还等什么，一起来感受动态规划的魅力吧!

2 实验项目结构

- find_maximum_subarray 题目一 寻找最大子序列问题
  - include
    - util.hpp 常用函数头文件
    - Solution.hpp 待完成
  - data
  - main.cpp 主程序代码
- longest_common_subsequence 题目二 最长公共子序列问题
  - include
    - util.hpp 常用函数头文件
    - Solution.hpp 待完成
  - data
  - main.cpp 主程序代码

请注意，每次修改完代码之后，需要重新编译运行 main.cpp，如果直接执行上次编译好的 main.exe 或 main，新的修改将不会生效。

3 实验内容

3.1 寻找最大子序列问题

给定一个长度为 N 的数组 A,其任意连续子序列可以表示为 $A_i, A_{i+1}, \cdots, A_j$，其中 $0 \leq i \leq j \leq N-1$。最大子序列是指所有的子序列中和最大的那一个，注意子序列不能为空。为了降低难度，你只需要输出它的和，即：

$$result = \max_{0 \leq i \leq j \leq N-1} \sum_{k=i}^{j} A[k]$$

例如：[-2, 11, -4, 13, -5, -2] 的答案为 20。

请使用动态规划思想完成 Solution.hpp 的实现。

```
class Solution {
public:
    long long find_maximum_subarray(vector<int> &A) {
        // 请在这里完成你的代码
    }
};
```

测试数据范围：$N \leq 10^5$, $|A[i]| \leq 10^9$。

3.2 最长公共子序列问题

给定两个字符串 s, t，求出它们的最长公共子序列长度。

```
class Solution {
    // 用于调试输出二维数组
    void print_table(vector<vector<int>> &table) {
        debug(table);
    }
public:
    int lcs(string s, string t, vector<vector<int>> &c,
            vector<vector<int>> &b) {
        // 注意，string 类型是 C++ 的字符串类型，可以通过 s[0] 来访问 s
            的第一个字符
        int n = s.size(), m = t.size();
        c.resize(n + 1);
        b.resize(n + 1);
        for(int i = 0; i <= n; i++) {
          c[i].resize(m + 1, 0);
          b[i].resize(m + 1, 0);
        }
        // 以上是数组初始化操作，请在下面完成你的代码
    }

    // print_lcs 只用于自己调试输出最长公共子序列，不做正确性检测
    void print_lcs(vector<vector<int>> &b, string &s, int i, int j) {
        // 请在这里完成你的代码
    }
};
```

main.cpp 的 main 函数中配置了简单的样例测试，你可以在写代码时首先观察这一组数据能否输出正确结果。调试成功后，再取消掉最后 test() 的注释进行所有测试数据的运行检测。

```
int main(){
    string s = "AGCTAG";
    string t = "ACTCC";
    int n = s.size(), m = t.size();
    vector<vector<int>> b, c;

    Solution sol;
    // 求解 LCS 长度
```

```
    int res = sol.lcs(s, t, c, b);
    cout << BLUE << "[test case]" << END << endl;
    cout << "s: " << s << endl;
    cout << "t: " << t << endl;
    cout << "Length of LCS: " << res << endl;
    cout << "LCS is: ";
    sol.print_lcs(b, s, n, m);
    cout << endl;

    // test();

    return 0;
}
```

4 实验思考

1. 在寻找最大子序列问题中，对比分治算法与动态规划算法在时空复杂度上的不同。
2. 分析动态规划方法求解最长公共子序列问题的时空复杂度。
3. 请解释最长公共子序列问题中，$c[i][j]$ 的 i 和 j 分别是什么含义，$c[i][j]$ 的值又是什么含义。

5 拓展实验

给定一个仅包含 0 和 1 、大小为 $rows \times cols$ 的二维二进制矩阵，找出只包含 1 的最大矩形，并返回其面积。

例如下图所示的 01 矩阵，最大全 1 子矩阵面积为 6：

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

数据范围：$1 \leq row, cols \leq 200$

你可以在这里完成本拓展题目：https://leetcode.cn/problems/maximal-rectangle/

提示：为了通过本题测试，你需要使用不超过 $O(n^3)$ 时间复杂度的算法。可以尝试枚举子矩阵的上边界与下边界，然后将问题转换为你熟悉的题目。