双向动态规划

最近为了面试刷了不少题，总结出一类动态规划题型的模板

**第一部分：双向动态规划使用的方法和情景**

通过以下一道题为例

#### [剑指 Offer 63. 股票的最大利润](https://leetcode-cn.com/problems/gu-piao-de-zui-da-li-run-lcof/)

<https://leetcode-cn.com/problems/gu-piao-de-zui-da-li-run-lcof/>

**假设把某股票的价格按照时间先后顺序存储在数组中，请问买卖该股票一次可能获得的最大利润是多少？**

示例 1:

输入: [7,1,5,3,6,4]

输出: 5

解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。

分析：对于每一个结点i，如果我们知道了结点i之前股票的最低价格，以及结点i之后股票的最高价格，那么我们遍历所有结点，便可以得到买卖一次股票可获得的最大利润

具体解法：

(1)定义dpleft[n]，其中每一个结点dpleft[i]代表第i天及以前股票的最低价格

(2)求解dpleft[i]=min(dpleft[i-1],prices[i])

(3)同理定义dpright[n]，dpright[i]=max(dpright[i+1],pricesi])，代表第i天及以后股票的最高价格

(4)则包含第i天能得到的最大利润=dpright[i]-dpleft[i-1]

代码:

class Solution {

public:

int maxProfit(vector<int>& prices) {

int n = prices.size();

if (n == 0) return 0;

int dpleft[n];

dpleft[0] = prices[0];

for (int i = 1; i < n; i++)

{

dpleft[i] = min(dpleft[i - 1], prices[i]);

}

int dpright[n];

dpright[n - 1] = prices[n - 1];

for (int i = n - 2; i >= 0; i--)

{

dpright[i] = max(dpright[i + 1], prices[i]);

}

int res = 0;

for (int i = 1; i < n; i++)

{

res = max(res, dpright[i] - dpleft[i - 1]);

}

return res;

}

};

你可能会觉得在这题中创建dp数组没有必要，那么请看下题

#### [剑指 Offer 66. 构建乘积数组](https://leetcode-cn.com/problems/gou-jian-cheng-ji-shu-zu-lcof/)

<https://leetcode-cn.com/problems/gou-jian-cheng-ji-shu-zu-lcof/>

**给定一个数组 A[0,1,…,n-1]，请构建一个数组 B[0,1,…,n-1]，其中 B[i] 的值是数组 A 中除了下标 i 以外的元素的积, 即 B[i]=A[0]×A[1]×…×A[i-1]×A[i+1]×…×A[n-1]。不能使用除法。**

示例:

输入: [1,2,3,4,5]

输出: [120,60,40,30,24]

分析：对于每一个结点i，如果我们知道了结点i之前所有数的乘积，以及结点i之后所有数的乘积，那么相乘便可以得到A中除了下标 i 以外的元素的积

具体解法：

(1)定义dpleft[i]=dpleft[i-1]\*A[i]代表下表为i及之前所有数的乘积

(2)定义dpright[i]=dpright[i+1]\*A[i]代表下表为i及之后所有数的乘积

(3)B[i]=dpleft[i-1]\*dpright[i+1]

代码:

class Solution {

public:

vector<int> constructArr(vector<int>& a) {

int len = a.size();

if (len == 0) return vector<int>();

int dpleft[len];

dpleft[0] = a[0];

for (int i = 1; i < len; i++)

{

dpleft[i] = dpleft[i - 1] \* a[i];

}

int dpright[len];

dpright[len - 1] = a[len - 1];

for (int i = len - 2; i >= 0; i--)

{

dpright[i] = dpright[i + 1] \* a[i];

}

vector<int> res(len);

res[0] = dpright[1];

res[len - 1] = dpleft[len - 2];

for (int i = 1; i < len - 1; i++)

{

res[i] = dpleft[i - 1] \* dpright[i + 1];

}

return res;

}

};

#### [42. 接雨水](https://leetcode-cn.com/problems/trapping-rain-water/)

<https://leetcode-cn.com/problems/trapping-rain-water/>

**给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。**



示例 1：

输入：height = [0,1,0,2,1,0,1,3,2,1,2,1]

输出：6

解释：上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。

分析：对于每一个结点i，其可接的水为i结点左右两侧最高柱子的较小值-i结点柱子的高度，

当i结点左右两侧最高柱子的较小值<i结点柱子高度时，则可接的水为0

具体解法：

(1)定义dpleft[i]=max(dpleft[i-1],height[i])代表i结点及以前柱子最高高度

(2)定义dpright[i]=max(dpright[i+1],height[i])代表i结点及以后柱子最高高度

(3)则每一个结点可接水的值=max(min(dpleft[i-1],dpright[i+1])-height[i],0)

代码:

class Solution {

public:

int trap(vector<int>& height) {

int len = height.size();

if (len <= 2) return 0;

int dpleft[len];

dpleft[0] = height[0];

for (int i = 1; i < len; i++)

{

dpleft[i] = max(dpleft[i - 1], height[i]);

}

int dpright[len];

dpright[len - 1] = height[len - 1];

for (int i = len - 2; i >= 0; i--)

{

dpright[i] = max(dpright[i + 1], height[i]);

}

int res = 0;

for (int i = 1; i < len - 1; i++)

{

res = res + max(min(dpleft[i - 1], dpright[i + 1]) - height[i], 0);

}

return res;

}

};

#### 美团3.11笔试

**最大子段和是一个经典问题，即对于一个数组找出其和最大的子数组。**

**现在允许你在求解该问题之前翻转这个数组的连续一段（如翻转{1,2,3,4,5,6}的第三个到第五个元素组成的子数组得到的是{1,2,5,4,3,6}），则翻转后该数组的最大子段和最大能达到多少？**

示例 1：

输入：nums = [3,-5,2]

输出：5

解释：翻转[-5,2]得到序列[3,2,-5]，最大子段和=5，也可以翻转[3,-5]，得到序列[-5,3,2]，最大子段和=5

分析：此题乍一看很难有思路，但还是可以转换成双向动态规划问题

对于每一个结点i，如果可以知道i之前最大子段和以及i之后的最大子段和，那么结果便为这两者相加。

可以证明经过一次翻转一定能将这两个最大子段拼合在一起

具体做法：

(1)首先通过求最大子段和方法求出以i结点为结尾的子段的最大和（必须包含i），得到数组A，具体可见**[剑指 Offer 42. 连续子数组的最大和](https://leetcode-cn.com/problems/lian-xu-zi-shu-zu-de-zui-da-he-lcof/)**

https://leetcode-cn.com/problems/lian-xu-zi-shu-zu-de-zui-da-he-lcof/

1. 定义dpleft[i]=max(dpleft[i-1],A[i])代表i结点及之前子段的最大和（不一定包含i）
2. 使用(1)的方式从后往前求数组的最大子段和B
3. 定义dpright[i]=max(dpright[i+1],B[i])代表i结点及之后子段的最大和（不一定包含i）
4. 则对于每个结点i，其能得到翻转一次后的最大子段和为dpright[i]+dpleft[i-1]

代码：

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> maxSubArray(vector<int>& nums) { //求最大子段和

vector<int> dp(nums.size());

dp[0] = nums[0];

for (int i = 1; i < nums.size(); i++)

{

if (dp[i - 1] > 0)

{

dp[i] = nums[i] + dp[i - 1];

}

else

{

dp[i] = nums[i];

}

}

return dp;

}

int maxRevertSubArray(vector<int>& nums) {

int len = nums.size();

vector<int> A = maxSubArray(nums);

int dpleft[len];

dpleft[0] = A[0];

for (int i = 1; i < len; i++)

{

dpleft[i] = max(dpleft[i - 1], A[i]);

}

reverse(nums.begin(), nums.end());

vector<int> B = maxSubArray(nums);

reverse(B.begin(), B.end());

int dpright[len];

dpright[len - 1] = B[len - 1];

for (int i = len - 2; i >= 0; i--)

{

dpright[i] = max(dpright[i + 1], B[i]);

}

int res = 0;

for (int i = 1; i < len; i++)

{

res = max(dpright[i] + dpleft[i - 1], res);

}

return res;

}

int main()

{

vector<int> nums = { 4,-1,2,-3,5,6 };

cout << maxRevertSubArray(nums) << endl;

return 0;

}

**第二部分：优化占用空间**

双向动态规划更多是处于理解的目的，而两次动态规划并非都是必要的，第二次动态规划dpright往往可以用一个变量代替,从而达到节省大小为n的空间的目的

#### 以[剑指 Offer 66. 构建乘积数组](https://leetcode-cn.com/problems/gou-jian-cheng-ji-shu-zu-lcof/)为例，可以在从后向前遍历数组A时，通过一个变量代替dpright记录i之后所有数的乘积

代码：

class Solution {

public:

vector<int> constructArr(vector<int>& a) {

int len = a.size();

if (len == 0) return vector<int>();

int dpleft[len];

dpleft[0] = a[0];

for (int i = 1; i < len; i++)

{

dpleft[i] = dpleft[i - 1] \* a[i];

}

vector<int> res(len);

int right = 1; //替代dpright

for (int i = len - 1; i > 0; i--)

{

res[i] = dpleft[i - 1] \* right;

right = right \* a[i];

}

res[0] = right;

return res;

}

};

双向动态规划

最近为了面试刷了不少题，总结出一类动态规划题型的模板

#第一部分：双向动态规划使用的方法和情景

通过以下一道题为例

##剑指 Offer 63. 股票的最大利润

https://leetcode-cn.com/problems/gu-piao-de-zui-da-li-run-lcof/

假设把某股票的价格按照时间先后顺序存储在数组中，请问买卖该股票一次可能获得的最大利润是多少？

示例 1:

输入: [7,1,5,3,6,4]

输出: 5

解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。

###分析：对于每一个结点i，如果我们知道了结点i之前股票的最低价格，以及结点i之后股票的最高价格，那么我们遍历所有结点，便可以得到买卖一次股票可获得的最大利润

###具体解法：

(1)定义dpleft[n]，其中每一个结点dpleft[i]代表第i天及以前股票的最低价格

(2)求解dpleft[i]=min(dpleft[i-1],prices[i])

(3)同理定义dpright[n]，dpright[i]=max(dpright[i+1],pricesi])，代表第i天及以后股票的最高价格

(4)则包含第i天能得到的最大利润=dpright[i]-dpleft[i-1]

###代码:

class Solution {

public:

int maxProfit(vector<int>& prices) {

int n = prices.size();

if (n == 0) return 0;

int dpleft[n];

dpleft[0] = prices[0];

for (int i = 1; i < n; i++)

{

dpleft[i] = min(dpleft[i - 1], prices[i]);

}

int dpright[n];

dpright[n - 1] = prices[n - 1];

for (int i = n - 2; i >= 0; i--)

{

dpright[i] = max(dpright[i + 1], prices[i]);

}

int res = 0;

for (int i = 1; i < n; i++)

{

res = max(res, dpright[i] - dpleft[i - 1]);

}

return res;

}

};

你可能会觉得在这题中创建dp数组没有必要，那么请看下题

##剑指 Offer 66. 构建乘积数组

https://leetcode-cn.com/problems/gou-jian-cheng-ji-shu-zu-lcof/

给定一个数组 A[0,1,…,n-1]，请构建一个数组 B[0,1,…,n-1]，其中 B[i] 的值是数组 A 中除了下标 i 以外的元素的积, 即 B[i]=A[0]×A[1]×…×A[i-1]×A[i+1]×…×A[n-1]。不能使用除法。

示例:

输入: [1,2,3,4,5]

输出: [120,60,40,30,24]

###分析：对于每一个结点i，如果我们知道了结点i之前所有数的乘积，以及结点i之后所有数的乘积，那么相乘便可以得到A中除了下标 i 以外的元素的积

###具体解法：

(1)定义dpleft[i]=dpleft[i-1]\*A[i]代表下表为i及之前所有数的乘积

(2)定义dpright[i]=dpright[i+1]\*A[i]代表下表为i及之后所有数的乘积

(3)B[i]=dpleft[i-1]\*dpright[i+1]

###代码:

class Solution {

public:

vector<int> constructArr(vector<int>& a) {

int len = a.size();

if (len == 0) return vector<int>();

int dpleft[len];

dpleft[0] = a[0];

for (int i = 1; i < len; i++)

{

dpleft[i] = dpleft[i - 1] \* a[i];

}

int dpright[len];

dpright[len - 1] = a[len - 1];

for (int i = len - 2; i >= 0; i--)

{

dpright[i] = dpright[i + 1] \* a[i];

}

vector<int> res(len);

res[0] = dpright[1];

res[len - 1] = dpleft[len - 2];

for (int i = 1; i < len - 1; i++)

{

res[i] = dpleft[i - 1] \* dpright[i + 1];

}

return res;

}

};

##接雨水

https://leetcode-cn.com/problems/trapping-rain-water/

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

示例 1：

输入：height = [0,1,0,2,1,0,1,3,2,1,2,1]

输出：6

解释：上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。

###分析：对于每一个结点i，其可接的水为i结点左右两侧最高柱子的较小值-i结点柱子的高度，

当i结点左右两侧最高柱子的较小值<i结点柱子高度时，则可接的水为0

###具体解法：

(1)定义dpleft[i]=max(dpleft[i-1],height[i])代表i结点及以前柱子最高高度

(2)定义dpright[i]=max(dpright[i+1],height[i])代表i结点及以后柱子最高高度

(3)则每一个结点可接水的值=max(min(dpleft[i-1],dpright[i+1])-height[i],0)

###代码:

class Solution {

public:

int trap(vector<int>& height) {

int len = height.size();

if (len <= 2) return 0;

int dpleft[len];

dpleft[0] = height[0];

for (int i = 1; i < len; i++)

{

dpleft[i] = max(dpleft[i - 1], height[i]);

}

int dpright[len];

dpright[len - 1] = height[len - 1];

for (int i = len - 2; i >= 0; i--)

{

dpright[i] = max(dpright[i + 1], height[i]);

}

int res = 0;

for (int i = 1; i < len - 1; i++)

{

res = res + max(min(dpleft[i - 1], dpright[i + 1]) - height[i], 0);

}

return res;

}

};

##美团3.11笔试

最大子段和是一个经典问题，即对于一个数组找出其和最大的子数组。

现在允许你在求解该问题之前翻转这个数组的连续一段（如翻转{1,2,3,4,5,6}的第三个到第五个元素组成的子数组得到的是{1,2,5,4,3,6}），则翻转后该数组的最大子段和最大能达到多少？

示例 1：

输入：nums = [3,-5,2]

输出：5

解释：翻转[-5,2]得到序列[3,2,-5]，最大子段和=5，也可以翻转[3,-5]，得到序列[-5,3,2]，最大子段和=5

###分析：此题乍一看很难有思路，但还是可以转换成双向动态规划问题

对于每一个结点i，如果可以知道i之前最大子段和以及i之后的最大子段和，那么结果便为这两者相加。

可以证明经过一次翻转一定能将这两个最大子段拼合在一起

###具体解法：

(1)首先通过求最大子段和方法求出以i结点为结尾的子段的最大和（必须包含i），得到数组A，具体可见剑指 Offer 42. 连续子数组的最大和

https://leetcode-cn.com/problems/lian-xu-zi-shu-zu-de-zui-da-he-lcof/

(2)定义dpleft[i]=max(dpleft[i-1],A[i])代表i结点及之前子段的最大和（不一定包含i）

(3)使用(1)的方式从后往前求数组的最大子段和B

(4)定义dpright[i]=max(dpright[i+1],B[i])代表i结点及之后子段的最大和（不一定包含i）

(5)则对于每个结点i，其能得到翻转一次后的最大子段和为dpright[i]+dpleft[i-1]

###代码：

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> maxSubArray(vector<int>& nums) { //求最大子段和

vector<int> dp(nums.size());

dp[0] = nums[0];

for (int i = 1; i < nums.size(); i++)

{

if (dp[i - 1] > 0)

{

dp[i] = nums[i] + dp[i - 1];

}

else

{

dp[i] = nums[i];

}

}

return dp;

}

int maxRevertSubArray(vector<int>& nums) {

int len = nums.size();

vector<int> A = maxSubArray(nums);

int dpleft[len];

dpleft[0] = A[0];

for (int i = 1; i < len; i++)

{

dpleft[i] = max(dpleft[i - 1], A[i]);

}

reverse(nums.begin(), nums.end());

vector<int> B = maxSubArray(nums);

reverse(B.begin(), B.end());

int dpright[len];

dpright[len - 1] = B[len - 1];

for (int i = len - 2; i >= 0; i--)

{

dpright[i] = max(dpright[i + 1], B[i]);

}

int res = 0;

for (int i = 1; i < len; i++)

{

res = max(dpright[i] + dpleft[i - 1], res);

}

return res;

}

int main()

{

vector<int> nums = { 4,-1,2,-3,5,6 };

cout << maxRevertSubArray(nums) << endl;

return 0;

}

#第二部分：优化占用空间

双向动态规划更多是处于理解的目的，而两次动态规划并非都是必要的，第二次动态规划dpright往往可以用一个变量代替,从而达到节省大小为n的空间的目的

以剑指 Offer 66. 构建乘积数组为例，可以在从后向前遍历数组A时，通过一个变量代替dpright记录i之后所有数的乘积

###代码：

class Solution {

public:

vector<int> constructArr(vector<int>& a) {

int len = a.size();

if (len == 0) return vector<int>();

int dpleft[len];

dpleft[0] = a[0];

for (int i = 1; i < len; i++)

{

dpleft[i] = dpleft[i - 1] \* a[i];

}

vector<int> res(len);

int right = 1; //替代dpright

for (int i = len - 1; i > 0; i--)

{

res[i] = dpleft[i - 1] \* right;

right = right \* a[i];

}

res[0] = right;

return res;

}

};