

解题思路一：

动态规划：

保留每个解，最后一个解是最优解。设置dp数组来存储解（即当前能得到的最大值），观察数组可以发现，第一个解一定是dp[0]=nums[0],第二个解一定是dp[1]=max(dp[0],nums[1]);从第三个解开始，一定是前一个解的值与前一个解的前一项与当前价值之和的较大值，即有方程dp[i]=max(dp[i-1],dp[i+2]+nums[i]),最后输出dp[nums.size()-1]即是所求的最大值

代码：

class Solution {

public:

int rob(vector<int>& nums) {

if (nums.empty()) {

return 0;

}

if (nums.size() == 1) {

return nums[0];

}

//主要方法是找到dp的方程

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

dp[0] = nums[0];

dp[1] = max(nums[0], nums[1]);

for (int i = 2; i < nums.size(); i++) {

dp[i] = max(dp[i - 1], dp[i - 2] + nums[i]);

}

return dp[nums.size() - 1];

}

};

解题思路二：

滑动窗口：

在解题思路一的基础上优化，不必保留每个解，每次只保存两个解，即当前i前两个解，每次比较选取最大值级可

class Solution {

public:

int rob(vector<int>& nums) {

if (nums.empty()) {

return 0;

}

int size = nums.size();

if (size == 1) {

return nums[0];

}

//解的初始化

int first = nums[0], second = max(nums[0], nums[1]);

//只保留前两个最大解

for (int i = 2; i < size; i++) {

int temp = second;

second = max(first + nums[i], second);

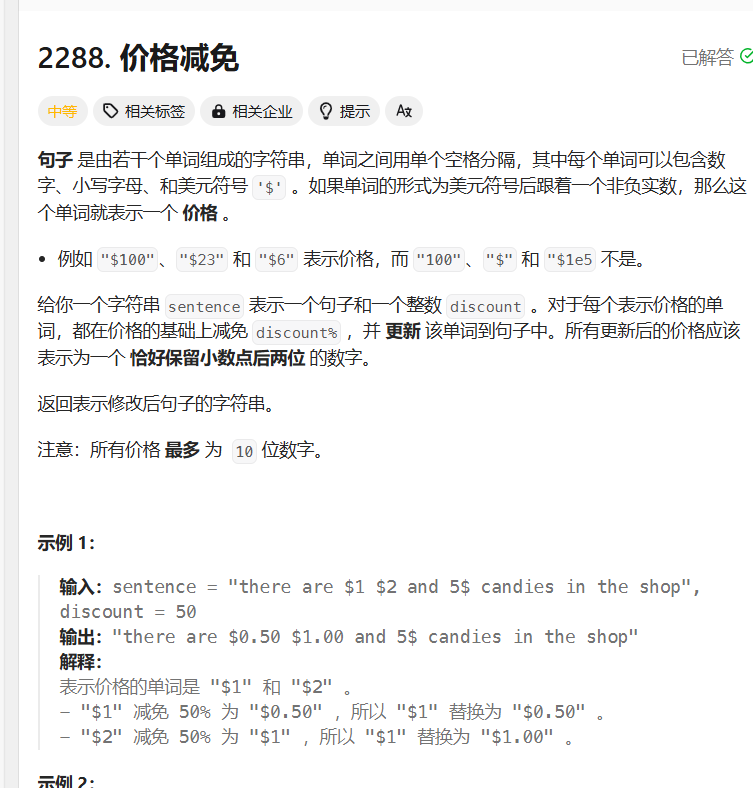
first = temp;

}

return second;

}

};



解题思路一：

按空格分割然后一一替换，先裁成每个单词在一起，对每个单词进行处理，满足要求的根据输入的折扣进行计算，然后更新单词，每个单词更新结束后，重新写入字符串，每个单词之间用空格隔开

代码：

#include <iostream>

#include <sstream>

#include <vector>

#include <iomanip>

using namespace std;

class Solution {

public:

string discountPrices(string sentence, int discount) {

istringstream iss(sentence);

vector<string> words;

string word;

// 分割句子成单词，并存储到words向量中

while (iss >> word) {

words.push\_back(word);

}

// 处理每个单词

for (string& w : words) {

// 检查单词是否以$开头且长度大于1

if (w[0] == '$' && w.size() > 1) {

bool isPrice = true;

// 检查$符号后面的字符是否全是数字或小数点

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

if (!isdigit(w[i]) && w[i] != '.') {

isPrice = false;

break;

}

}

if (isPrice) {

// 转换为价格并计算折扣后的价格

double price = stod(w.substr(1)); // 将字符串转换为double

double discountedPrice = price \* (1 - discount / 100.0); // 计算折扣后的价格

// 格式化为两位小数的字符串

ostringstream oss;

oss << fixed << setprecision(2) << discountedPrice; // 保留两位小数

w = "$" + oss.str(); // 重新组合成带$符号的字符串

}

}

}

// 重新组合句子

ostringstream result;

for (int i = 0; i < words.size(); ++i) {

if (i > 0) result << " "; // 在单词之间添加空格

result << words[i];

}

return result.str(); // 返回处理后的句子

}

};

int main() {

Solution solution;

string sentence = "there are $1 $2 and 5$ candies in the shop";

int discount = 50;

string result = solution.discountPrices(sentence, discount);

cout << result << endl; // 输出处理后的句子

return 0;

}

解题思路二：

模拟：

对每一种情况进行模拟，对于不满足要求的字符正常遍历存储，当满足要求时，当前字符到最后一个连续数字全部用long保存后使用double升级得到折扣结果，这个结果可以用sprintf写入字符串，然后使存储原字符串的ans拼接这段结果即可

代码：

class Solution {

public:

string discountPrices(string sentence, int discount) {

string ans;

for (int i = 0; i < sentence.size(); i++) {

// 非$之后的数字一律先放进数组

ans.push\_back(sentence[i]);

if (sentence[i] == '$') {

// 如果$前一个不是空格

if (i - 1 >= 0 && sentence[i - 1] != ' ')

continue;

// 如果$后一个不是数字

if (i + 1 >= sentence.size() || sentence[i + 1] > '9' ||

sentence[i + 1] < '0') {

continue;

}

// 记录当前下标

int tmp = i;

// 从下一个开始，找到全部连续数字

long num = 0;

i++;

while (i < sentence.size() && sentence[i] >= '0' &&

sentence[i] <= '9') {

num = num \* 10 + sentence[i] - '0';

i++;

}

// 如果数字的结尾不是空格，则返回

if (i < sentence.size() && sentence[i] != ' ') {

i = tmp;

continue;

}

// 将数字打折转换成字符串返回

double ret = num \* (1.0 - discount / 100.0);

char str[40];

sprintf(str, "%.2f", ret);

ans += string(str);

// 由于之前循环里多加了一次，减回来

i--;

}

}

return ans;

}