

题目：

234. 回文链表

给你一个单链表的头节点 `head`，请你判断该链表是否为回文链表。如果是，返回 `true`；否则，返回 `false`。

示例 1:

输入: `head = [1,2,2,1]`
输出: `true`

示例 2:

输入: `head = [1,2]`
输出: `false`

提示:

- 链表中节点数目在范围 $[1, 10^5]$ 内

```
1 /**  
2  * Definition for singly-linked list.  
3  * struct ListNode {  
4  *     int val;  
5  *     ListNode *next;  
6  *     ListNode() : val(0), next(nullptr) {}  
7  *     ListNode(int x) : val(x), next(nullptr) {}  
8  *     ListNode(int x, ListNode *next) : val(x), next(next) {}  
9  * };  
10 */  
11 class Solution {  
12 public:  
13     bool isPalindrome(ListNode* head) {  
14         queue<ListNode*> wait;  
15         stack<ListNode*> another;  
16         if (!head) return true; // 空链表返回 true  
17         while (head != nullptr) // 非空链表入栈和入队  
18         {  
19             wait.push(head);  
20             another.push(head);  
21             head = head->next;  
22         }  
23         int len = another.size(); // 获取比较次数  
24         while (len) // 开始比较  
25         {  
26             if (wait.front()->val == another.top()->val)  
27             {  
28                 wait.pop();  
29                 another.pop();  
30                 len--;  
31             }  
32         }  
33         return true;  
34     }  
35 };
```

解题思路一：

栈和队列：

使用栈和队列分别存储遍历结果，由于栈的特性是先进后出，队列的特性是先进先出，这时比较栈和队列的栈顶元素与队首元素的数据域的值，如果相等继续比较，不等直接退出，相等继续比较的前提是比较次数等于栈或队列的长度

//方法 1，使用辅助队列

```
class Solution {  
public:  
    bool isPalindrome(ListNode* head) {  
        queue<ListNode*> wait;  
        stack<ListNode*> another;  
        if (!head) return true; // 空链表返回 true  
        while (head != nullptr) // 非空链表入栈和入队  
        {  
            wait.push(head);  
            another.push(head);  
            head = head->next;  
        }  
        int len = another.size(); // 获取比较次数  
        while (len) // 开始比较  
        {  
            if (wait.front()->val == another.top()->val)  
            {  
                wait.pop();  
                another.pop();  
                len--;  
            }  
        }  
        return true;  
    }  
};
```

```

        wait.pop();
        another.pop();
        len--;
    }
    else
        return false;
}
return true;
}
};

```

解题思路二：

使用两个辅助栈：

思路与一大致相同，不同点是入栈过后的栈得出栈然后进入另外一个栈，然后两个栈开始比较，一个栈是正序，一个栈是倒序，这次我们存入元素，不存入指针

```

class Solution {
public:
    bool isPalindrome(ListNode* head) {
        stack<int> mystack1;
        stack<int> mystack2;
        stack<int> mystack3;
        if (!head) return true;
        //正序
        while (head)
        {
            mystack1.push(head->val);
            head = head->next;
        }
        mystack2 = mystack1;
        //倒序
        while (!mystack2.empty())
        {
            mystack3.push(mystack2.top());
            mystack2.pop();
        }
        int len = mystack1.size();
        while (len)
        {
            if (mystack1.top() == mystack3.top())

```

```

        {
            mystack1.pop();
            mystack3.pop();
            len--;
        }
        else
            return false;
    }

    return true;
}

};

```

解题思路三：

使用双端队列：

遍历链表元素入队，每次比较队首和队尾元素，相同时队首队尾都出队，继续下次比较，直到只剩下一个（奇数个元素链表）或者 0 个（偶数个元素的链表），比较过程中有一次不等就退出比较，返回 false

```

class Solution {
public:
    bool isPalindrome(ListNode* head) {
        if (!head) return true;
        deque<int> wait;
        while (head)
        {
            wait.push_front(head->val);
            head = head->next;
        }
        while (wait.size() > 1)
        {
            if (wait.front() != wait.back())
                return false;
            wait.pop_front();
            wait.pop_back();
        }

        return true;
    }
};

```

解题思路四：

快慢指针加翻转链表：

使用快慢指针来确定链表的中间节点，快指针的速度是慢指针的两倍，当快指针走完时，慢指针刚好走到一半，慢指针的下一个节点作为翻转链表的起始节点。翻转后，两个链表开始逐一比较，直到后半的翻转链表走完或者出现不等的情况

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {

        // 找到前半部分链表的尾节点并反转后半部分链表
        ListNode* firstHalfEnd = endOfFirstHalf(head);
        ListNode* secondHalfStart = reverseList(firstHalfEnd->next);

        // 判断是否回文
        ListNode* p1 = head;
        ListNode* p2 = secondHalfStart;
        bool result = true;
        while (result && p2 != nullptr) {
            if (p1->val != p2->val) {
                result = false;
            }
            p1 = p1->next;
            p2 = p2->next;
        }

        // 还原链表并返回结果
        firstHalfEnd->next = reverseList(secondHalfStart);
        return result;
    }

    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* curr = head;
        while (curr != nullptr) {
            ListNode* nextTemp = curr->next;
```

```

        curr->next = prev;

        prev = curr;

        curr = nextTemp;

    }

    return prev;
}

ListNode* endOfFirstHalf(ListNode* head) {
    ListNode* fast = head;
    ListNode* slow = head;

    while (fast->next != nullptr && fast->next->next != nullptr) {
        fast = fast->next->next;
        slow = slow->next;
    }

    return slow;
}
};

```

题目：

The screenshot shows a coding platform interface. On the left, the problem description for '739. 每日温度' (739. Daily Temperatures) is visible. It asks to return an array 'answer' where 'answer[i]' is the number of days until a warmer temperature appears. If no warmer temperature exists, the value should be 0. Three examples are provided: Example 1 with input [73, 74, 75, 71, 69, 72, 76, 73] and output [1, 1, 4, 2, 1, 1, 0, 0]; Example 2 with input [30, 40, 50, 60] and output [1, 1, 1, 0]; Example 3 with input [30, 60, 90] and output [1, 1, 0]. Constraints are also listed: 1 ≤ temperatures.length ≤ 10⁵ and 30 ≤ temperatures[i] ≤ 100.

On the right, the C++ code is shown in a '智能模式' (Smart Mode) editor. The code defines a 'Solution' class with a 'dailyTemperatures' method that takes a vector of integers and returns a vector of integers. The method implementation is as follows:

```

1 class Solution {
2 public:
3     vector<int> dailyTemperatures(vector<int>& temperatures) {
4         ...
5     }
6 };

```

Below the code editor, there is a '测试用例' (Test Cases) section. It shows 'Case 1' selected, with the input 'temperatures = [73, 74, 75, 71, 69, 72, 76, 73]' and the expected output.

解题思路一：

单调栈：

维护一个单调递减的栈，这个栈存放温度的下标，当栈为空时，元素直接入栈，当栈非空时，如果当前温度小于栈顶元素对应的温度，该下标入栈，如果当前温度大于栈顶元素温度，栈顶元素出栈，直到栈中所有比当前温度小的元素都出栈过后，当前温度对应下标入栈，每次出栈在 ans 数组中把 ans[index] 置为 i-index（ans 初始值全 0）

```
#include<iostream>
```

```
#include<vector>
```

```

#include<stack>
#include<queue>
#include<deque>

using namespace std;
//方法1 暴力
//方法2 单调栈
class Solution {
public:
    vector<int> dailyTemperatures(vector<int>& temperatures) {
        int n = temperatures.size();//获取长度
        vector<int> ans(n);
        stack<int> s;
        for (int i = 0; i < n; ++i) {
            while (!s.empty() && temperatures[i] > temperatures[s.top()]) {
                int previousIndex = s.top();//当前栈中温度的索引
                ans[previousIndex] = i - previousIndex;
                s.pop();
            }
            s.push(i);
        }
        return ans;
    }
};

```

解题思路二:

暴力:

简单的正向遍历会超出时间限制，所以需要优化，我们从数组中倒数第二个元素开始计算，（因为倒数第一个元素必定没有比它温度更高的天气），如果当前元素大于它后面的元素，且它后面的元素没有更大的元素时，结束循环，得到天数是 $j-i$;如果有更大的元素 $j+=ans[j]$

代码:

```

class Solution {
public:
    vector<int> dailyTemperatures(vector<int>& temperatures) {
        int n = temperatures.size();
        vector<int> ans(n, 0);
        for (int i = n - 2; i >= 0; --i) {
            int j = i + 1;

```

```
while (j < n && temperatures[j] <= temperatures[i]) {  
    // 如果 ans[j] == 0, 说明在 j 之后没有更高的温度  
    if (ans[j] == 0) {  
        j = n; // 结束循环  
    }  
    else {  
        j = j + ans[j]; // 跳到下一个更高温度的位置  
    }  
}  
if (j < n) {  
    ans[i] = j - i;  
}  
}  
return ans;  
}  
};
```