【第二十七课】动态规划算法优化

一、Leetcode 选题

1.714. 买卖股票的最佳时机含手续费

```
class Solution {
public:
    int maxProfit(vector<int>& prices, int fee) {
        int n = prices.size();
        int dp[n][2];
        dp[0][0] = 0;
        dp[0][1] = -prices[0];
        for (int i = 1; i < n; i++) {
            dp[i][0] = max(dp[i - 1][0], dp[i - 1][1] + prices[i] - fee);
            dp[i][1] = max(dp[i - 1][1], dp[i - 1][0] - prices[i]);
        }
        return max(dp[n - 1][0], dp[n - 1][1]);
    }
};
```

2.213. 打家劫舍 II

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        if (n == 1) return nums[0];
        int dp[n][2];
        dp[0][0] = 0;
        dp[0][1] = nums[0];
        for (int i = 1; i < n; i++) {
            dp[i][0] = max(dp[i - 1][1], dp[i - 1][0]);
            dp[i][1] = dp[i - 1][0] + nums[i];
        }
        int ans1 = dp[n - 1][0];
        dp[0][0] = 0;
        dp[0][1] = 0;
        for (int i = 1; i < n; i++) {
            dp[i][0] = max(dp[i - 1][1], dp[i - 1][0]);
            dp[i][1] = dp[i - 1][0] + nums[i];
        }
        int ans2 = max(dp[n - 1][0], dp[n - 1][1]);
        return max(ans1, ans2);
    }
};
```

3.416. 分割等和子集

```
class Solution {
public:
    bool canPartition(vector<int>& nums) {
        int sum = 0;
        for (auto x : nums) sum += x;
        if (sum % 2) return false;
        int f[sum + 1];
        memset(f, 0, sizeof(f));
        f[0] = 1;
        sum = 0;
        for (auto x : nums) {
            sum += x;
            for (int j = sum; j >= x; j--) {
                f[j] |= f[j - x];
            }
        }
        return f[sum / 2];
    }
};
```

4.474. 一和零

```
class Solution {
public:
    int findMaxForm(vector<string>& strs, int m, int n) {
        int dp[m + 1][n + 1];
        memset(dp, 0, sizeof(dp));
        for (auto x : strs) {
            int cnt0 = 0, cnt1 = 0;
            for (auto y : x) {
                if (y == '0') cnt0 += 1;
                else cnt1 += 1;
            }
            for (int i = m; i >= cnt0; --i) {
                for (int j = n; j >= cnt1; --j) {
                    dp[i][j] = max(dp[i][j], dp[i - cnt0][j - cnt1] + 1);
                }
            }
        }
        return dp[m][n];
    }
};
```

5.494. 目标和

```
class Solution {
public:
    int findTargetSumWays(vector<int>& nums, int target) {
        int n = nums.size(), sum = 0;
        for (auto x : nums) sum += x;
        int buff[2][2 * sum + 5], *f[2] = {buff[0] + sum + 2, buff[1] + sum +
2};
        sum = 0;
        memset(buff, 0, sizeof(buff));
```

```
        f[0][0] = 1;
        for (int i = 1; i <= n; i++) {
            int ind = i % 2;
            int pre_ind = !ind;
            int x = nums[i - 1];
            memset(buff[ind], 0, sizeof(buff[ind]));
            for (int j = -sum; j <= sum; j++) {
                f[ind][j + x] += f[pre_ind][j];
                f[ind][j - x] += f[pre_ind][j];
            }
            sum += x;
        }
        return f[n % 2][target];
    }
};
```

6.322. 零钱兑换

```
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        int dp[amount + 1];
        for (int i = 0; i <= amount; i++) dp[i] = -1;
        dp[0] = 0;
        for (auto x : coins) {
            for (int j = x; j <= amount; j++) {
                if (dp[j - x] == -1) continue;
                if (dp[j] == -1 || dp[j] > dp[j - x] + 1) dp[j] = dp[j - x] + 1;
            }
        }
        return dp[amount];
    }
};
```

7.518. 零钱兑换 II

```
class Solution {
public:
    int change(int amount, vector<int>& coins) {
        int f[amount + 1];
        memset(f, 0, sizeof(f));
        f[0] = 1;
        for (auto x : coins) {
            for (int j = x; j <= amount; j++) {
                f[j] += f[j - x];
            }
        }
        return f[amount];
    }
};
```

8.377. 组合总和 IV

```
class Solution {
public:
    int combinationSum4(vector<int>& nums, int target) {
        unsigned int f[target + 1];
        memset(f, 0, sizeof(f));
        f[0] = 1;
        for (int i = 1; i <= target; i++) {
            for (auto x : nums) {
                if (i < x) continue;
                f[i] += f[i - x];
            }
        }
        return f[target];
    }
};
```

9.382. 链表随机节点

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    /** @param head The linked list's head.
        Note that the head is guaranteed to be not null, so it contains at least
one node. */
    int n;
    ListNode *head;
    Solution(ListNode* head) : head(head), n(0) {
        ListNode *p = head, *q;
        while (p) q = p, p = p->next, n += 1;
        q->next = head;
    }

    /** Returns a random node's value. */
    int getRandom() {
        int x = rand() % n;
        while (x--) head = head->next;
        return head->val;
    }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(head);
 * int param_1 = obj->getRandom();
 */
```

10.462. 最少移动次数使数组元素相等 II

```
class Solution {
public:
    int minMoves2(vector<int>& nums) {
        int n = nums.size();
        nth_element(nums.begin(), nums.begin() + n / 2, nums.end());
        int x0 = nums[n / 2], ans = 0;
        for (auto x : nums) ans += abs(x - x0);
        return ans;
    }
};
```

11.77. 组合

```
class Solution {
public:
    void dfs(
        int i, int n, int cnt, int k,
        vector<int> &buff, vector<vector<int>> &ret
    ) {
        if (cnt == k) {
            ret.push_back(buff);
            return ;
        }
        if (n - i + 1 < k - cnt) return ;
        buff[cnt] = i;
        dfs(i + 1, n, cnt + 1, k, buff, ret);
        dfs(i + 1, n, cnt, k, buff, ret);
        return ;
    }
    vector<vector<int>> combine(int n, int k) {
        vector<int> buff(k);
        vector<vector<int>> ret;
        dfs(1, n, 0, k, buff, ret);
        return ret;
    }
};
```

12.234. 回文链表

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode *reverse(ListNode *head) {
        ListNode ret, *p = head, *q;
        while (p) {
```

```
            q = p->next;
            p->next = ret.next;
            ret.next = p;
            p = q;
        }
        return ret.next;
    }
    bool isPalindrome(ListNode* head) {
        if (head == nullptr) return false;
        ListNode *p = head, *q = head;
        while (q->next && q->next->next) {
            p = p->next;
            q = q->next->next;
        }
        p->next = reverse(p->next);
        q = p->next;
        p = head;
        while (q) {
            if (p->val != q->val) return false;
            p = p->next;
            q = q->next;
        }
        return true;
    }
};
```