1470. 重新排列数组（简单）

1. 题目描述

   给你一个数组 `nums` ，数组中有 `2n` 个元素，按 `[x1,x2,...,xn,y1,y2,...,yn]` 的格式排列。

   请你将数组按 `[x1,y1,x2,y2,...,xn,yn]` 格式重新排列，返回重排后的数组。

   示例 1：

   ```
   输入：nums = [2,5,1,3,4,7], n = 3
   输出：[2,3,5,4,1,7]
   解释：由于 x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 ，所以答案为 [2,3,5,4,1,7]
   ```

   示例 2：

   ```
   输入：nums = [1,2,3,4,4,3,2,1], n = 4
   输出：[1,4,2,3,3,2,4,1]
   ```

   示例 3：

   ```
   输入：nums = [1,1,2,2], n = 2
   输出：[1,2,1,2]
   ```

   提示：

   - `1 <= n <= 500`
   - `nums.length == 2n`
   - `1 <= nums[i] <= 10^3`

2. 比赛实现

```cpp
class Solution {
public:
    vector<int> shuffle(vector<int>& nums, int n) {
        vector<int> ans;
        for(int i = 0; i < n; i++){
            ans.push_back(nums[i]);
            ans.push_back(nums[n+i]);
        }
        return ans;
    }
};
```

1471. 数组中的K个最强值（中等）

1. 题目描述

给你一个整数数组 `arr` 和一个整数 `k` 。

设 `m` 为数组的中位数，只要满足下述两个前提之一，就可以判定 `arr[i]` 的值比 `arr[j]` 的值更强：

- `|arr[i] - m| > |arr[j] - m|`
- `|arr[i] - m| == |arr[j] - m|` ，且 `arr[i] > arr[j]`

请返回由数组中最强的 `k` 个值组成的列表。答案可以以 **任意顺序** 返回。

中位数 是一个有序整数列表中处于中间位置的值。形式上，如果列表的长度为 `n` ，那么中位数就是该有序列表（下标从 0 开始）中位于 `((n - 1) / 2)` 的元素。

- 例如 `arr = [6, -3, 7, 2, 11]` ， `n = 5` ：数组排序后得到 `arr = [-3, 2, 6, 7, 11]` ，数组的中间位置为 `m = ((5 - 1) / 2) = 2` ，中位数 `arr[m]` 的值为 `6` 。
- 例如 `arr = [-7, 22, 17, 3]` ， `n = 4` ：数组排序后得到 `arr = [-7, 3, 17, 22]` ，数组的中间位置为 `m = ((4 - 1) / 2) = 1` ，中位数 `arr[m]` 的值为 `3` 。

示例 1：

```
输入：arr = [1,2,3,4,5], k = 2
输出：[5,1]
解释：中位数为 3，按从强到弱顺序排序后，数组变为 [5,1,4,2,3]。最强的两个元素是 [5, 1]。[1, 5] 也是正确答案。
注意，尽管 |5 - 3| == |1 - 3| ，但是 5 比 1 更强，因为 5 > 1 。
```

示例 2：

```
输入：arr = [1,1,3,5,5], k = 2
输出：[5,5]
解释：中位数为 3，按从强到弱顺序排序后，数组变为 [5,5,1,1,3]。最强的两个元素是 [5, 5]。
```

示例 3：

```
输入：arr = [6,7,11,7,6,8], k = 5
输出：[11,8,6,6,7]
解释：中位数为 7，按从强到弱顺序排序后，数组变为 [11,8,6,6,7,7]。
[11,8,6,6,7] 的任何排列都是正确答案。
```

示例 4：

```
输入：arr = [6,-3,7,2,11], k = 3
输出：[-3,11,2]
```

示例 5：

```
输入：arr = [-7,22,17,3], k = 2
输出：[22,17]
```

提示：

- `1 <= arr.length <= 10^5`
- `-10^5 <= arr[i] <= 10^5`
- `1 <= k <= arr.length`

2. 比赛实现

```
int m;
class Solution {
public:
    static bool cmp(pair<int,int>& a, pair<int,int>& b){
        if(abs(a.first-m) != abs(b.first-m))
            return abs(a.first-m) > abs(b.first-m);
        else
            return a.first > b.first;
    }
    vector<int> getStrongest(vector<int>& arr, int k) {
        vector<pair<int,int>> v;
        int n = arr.size();
        for(int i = 0; i < n; i++)
            v.push_back({arr[i], i});
        sort(v.begin(), v.end());
        m = v[(n-1)/2].first;
        sort(v.begin(), v.end(), cmp);
        vector<int> ans;
        for(int i = 0; i < k; i++)
            ans.push_back(v[i].first);
        return ans;
    }
};
```

1472. 设计浏览器历史记录（中等）

---

1. 题目描述

你有一个只支持单个标签页的 **浏览器**，最开始你浏览的网页是 `homepage`，你可以访问其他的网站 `url`，也可以在浏览历史中后退 `steps` 步或前进 `steps` 步。

请你实现 `BrowserHistory` 类：

- `BrowserHistory(string homepage)`，用 `homepage` 初始化浏览器类。

- `void visit(string url)` 从当前页跳转访问 `url` 对应的页面 。执行此操作会把浏览历史前进的记录全部删除。
- `string back(int steps)` 在浏览历史中后退 `steps` 步。如果你只能在浏览历史中后退至多 `x` 步且 `steps > x` ，那么你只后退 `x` 步。请返回后退 **至多** `steps` 步以后的 `url` 。
- `string forward(int steps)` 在浏览历史中前进 `steps` 步。如果你只能在浏览历史中前进至多 `x` 步且 `steps > x` ，那么你只前进 `x` 步。请返回前进 **至多** `steps` 步以后的 `url` 。

示例：

```
输入：
["BrowserHistory","visit","visit","visit","back","back","forward","visit","forward","back","back"]
[["leetcode.com"],["google.com"],["facebook.com"],["youtube.com"],[1],[1],[1],["linkedin.com"],[2],[2],[7]]
输出：
[null,null,null,null,"facebook.com","google.com","facebook.com",null,"linkedin.com","google.com","leetcode.com"]

解释：
BrowserHistory browserHistory = new BrowserHistory("leetcode.com");
browserHistory.visit("google.com");       // 你原本在浏览 "leetcode.com" 。访问 "google.com"
browserHistory.visit("facebook.com");     // 你原本在浏览 "google.com" 。访问 "facebook.com"
browserHistory.visit("youtube.com");      // 你原本在浏览 "facebook.com" 。访问 "youtube.com"
browserHistory.back(1);                   // 你原本在浏览 "youtube.com" ，后退到 "facebook.com" 并返回 "facebook.com"
browserHistory.back(1);                   // 你原本在浏览 "facebook.com" ，后退到 "google.com" 并返回 "google.com"
browserHistory.forward(1);                // 你原本在浏览 "google.com" ，前进到 "facebook.com" 并返回 "facebook.com"
browserHistory.visit("linkedin.com");     // 你原本在浏览 "facebook.com" 。 访问 "linkedin.com"
browserHistory.forward(2);                // 你原本在浏览 "linkedin.com" ，你无法前进任何步数。
browserHistory.back(2);                   // 你原本在浏览 "linkedin.com" ，后退两步依次先到 "facebook.com" ，然后到 "google.com" ，并返回 "google.com"
browserHistory.back(7);                   // 你原本在浏览 "google.com"，你只能后退一步到 "leetcode.com" ，并返回 "leetcode.com"
```

提示：

- `1 <= homepage.length <= 20`
- `1 <= url.length <= 20`

- `1 <= steps <= 100`
- `homepage` 和 `url` 都只包含 '.' 或者小写英文字母。
- 最多调用 `5000` 次 `visit`， `back` 和 `forward` 函数。

2. 比赛实现

```
class BrowserHistory {
public:
    vector<string> data;
    int idx;
    BrowserHistory(string homepage) {
        data.push_back(homepage);
        idx = 0;
    }

    void visit(string url) {
        if(idx == data.size() - 1){
            data.push_back(url);
            idx++;
        }
        else{
            data[++idx] = url;
            if(idx < data.size() - 1)
                data.erase(data.begin()+idx+1, data.end());
        }
    }

    string back(int steps) {
        idx -= steps;
        idx = max(idx, 0);
        return data[idx];
    }

    string forward(int steps) {
        idx = min(idx+steps, int(data.size())-1);
        return data[idx];
    }
};
```

1473. 给房子涂色III（困难）

---

1. 题目描述

在一个小城市里，有 `m` 个房子排成一排，你需要给每个房子涂上 `n` 种颜色之一（颜色编号为 `1` 到 `n`）。有的房子去年夏天已经涂过颜色了，所以这些房子不需要被重新涂色。

我们将连续相同颜色尽可能多的房子称为一个街区。（比方说 `houses = [1,2,2,3,3,2,1,1]`，它包含 5 个街区 `[{1}, {2,2}, {3,3}, {2}, {1,1}]`。）

给你一个数组 `houses`，一个 `m * n` 的矩阵 `cost` 和一个整数 `target`，其中：

- `houses[i]`：是第 `i` 个房子的颜色，**0** 表示这个房子还没有被涂色。
- `cost[i][j]`：是将第 `i` 个房子涂成颜色 `j+1` 的花费。

请你返回房子涂色方案的最小总花费，使得每个房子都被涂色后，恰好组成 `target` 个街区。如果没有可用的涂色方案，请返回 **-1**。

示例 1：

```
输入：houses = [0,0,0,0,0], cost = [[1,10],[10,1],[10,1],[1,10],[5,1]],
m = 5, n = 2, target = 3
输出：9
解释：房子涂色方案为 [1,2,2,1,1]
此方案包含 target = 3 个街区，分别是 [{1}, {2,2}, {1,1}]。
涂色的总花费为 (1 + 1 + 1 + 1 + 5) = 9。
```

示例 2：

```
输入：houses = [0,2,1,2,0], cost = [[1,10],[10,1],[10,1],[1,10],[5,1]],
m = 5, n = 2, target = 3
输出：11
解释：有的房子已经被涂色了，在此基础上涂色方案为 [2,2,1,2,2]
此方案包含 target = 3 个街区，分别是 [{2,2}, {1}, {2,2}]。
给第一个和最后一个房子涂色的花费为 (10 + 1) = 11。
```

示例 3：

```
输入：houses = [0,0,0,0,0], cost = [[1,10],[10,1],[1,10],[10,1],[1,10]],
m = 5, n = 2, target = 5
输出：5
```

示例 4：

```
输入：houses = [3,1,2,3], cost = [[1,1,1],[1,1,1],[1,1,1],[1,1,1]], m =
4, n = 3, target = 3
输出：-1
解释：房子已经被涂色并组成了 4 个街区，分别是 [{3},{1},{2},{3}] ，无法形成
target = 3 个街区。
```

提示：

- `m == houses.length == cost.length`
- `n == cost[i].length`
- `1 <= m <= 100`
- `1 <= n <= 20`
- `1 <= target <= m`
- `0 <= houses[i] <= n`
- `1 <= cost[i][j] <= 10^4`

2. 正确解法

三维动态规划，定义dp[i][j][k]为第i个房子颜色为j,且到目前为止，一共有k个街区的最小花费

在扩展状态的时候，如果当前颜色和上一个颜色相同，则街区数不变，否则街区数+1.

```
class Solution {
public:
    int minCost(vector<int>& houses, vector<vector<int>>& cost, int m,
int n, int target) {
        vector<vector<vector<int>>> dp(m , vector<vector<int>>(n,
vector<int>(target + 1, 0x3f3f3f3f)));
        //初始化
        if(houses[0] != 0)
            dp[0][houses[0] - 1][1] = cost[0][houses[0] - 1];
        else
            for(int i = 0; i < n; i ++) dp[0][i][1] = cost[0][i];

        for(int i = 1; i < m; i ++){//依次各个房子
            for(int j = 1; j <= target; j ++){//构成街区数
                if(houses[i] != 0){//已有颜色，不能更改
                    for(int k = 0; k < n; k ++){
                        if(k == houses[i] - 1)
                            dp[i][houses[i] - 1][j] = min(dp[i]
[houses[i] - 1][j],
                                                          dp[i-1][houses[i]
- 1][j] + cost[i][houses[i] -1]);
                        else
                            dp[i][houses[i] - 1][j] = min(dp[i]
[houses[i] - 1][j],
                                                          dp[i-1][k][j - 1]
+ cost[i][houses[i] - 1]);
                    }
                }else{//无颜色
                    for(int k = 0; k < n; k ++){
                        for(int l = 0; l < n; l ++){
                            if(k == l)
                                dp[i][k][j] = min(dp[i][k][j], dp[i-1]
[k][j] + cost[i][k]);
```

```
                            else
                                dp[i][k][j] = min(dp[i][k][j], dp[i-1]
    [l][j - 1] + cost[i][k]);
                        }
                    }
                }
            }
        }
        int ans = 0x3f3f3f3f;
        for(int i = 0; i < n; i ++)
            ans = min(ans, dp[m-1][i][target]);
        if(ans == 0x3f3f3f3f) return -1;
        for(int i = 0; i < m; i ++)//去掉已有涂色的花费
            if(houses[i] != 0) ans -= cost[i][houses[i] - 1];
        return ans;
    }
};
```