贪心：55. 跳跃游戏

代码逻辑

cpp 复制

```cpp
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int left = 0, right = 0, maxPos = 0;
        int n = nums.size();

        // 循环直到 left 超过 right 或者 maxPos 达到或超过最后一个位置
        while (left <= right) {
            // 如果已经可以到达或超过最后一个位置，返回 true
            if (maxPos >= n - 1) return true;

            // 从 left 到 right 范围内，更新 maxPos
            for (int i = left; i <= right; i++) {
                maxPos = max(maxPos, nums[i] + i);
            }

            // 更新下一轮的跳跃范围
            left = right + 1;
            right = maxPos;
        }

        // 如果跳出循环仍未到达最后位置，返回 false
        return false;
    }
};
```

代码解释

- left 和 right 用于定义当前的跳跃范围。
- maxPos 表示当前可以到达的最远位置。
- 每次循环从 left 到 right，更新 maxPos。
- 如果 maxPos 已经达到了最后一个位置或超过了它，则可以跳到最后，返回 true。
- 如果跳出循环后 maxPos 仍未达到最后位置，返回 false。

举例说明

让我们用一个例子来逐步分析代码的执行过程。

假设输入为 nums = [2, 3, 1, 1, 4]。我们的目标是确定是否能从数组的第一个位置跳到最后一个位置。

初始状态

- left = 0
- right = 0
- maxPos = 0
- 数组长度 n = 5

第一次循环

1. **检查是否可以到达终点**：
   - maxPos >= n - 1 (即 0 >= 4)，不成立，因此继续循环。
2. **更新** maxPos：
   - for (int i = left; i <= right; i++)，也就是 i = 0
   - 更新 maxPos = max(0, nums[0] + 0) = max(0, 2) = 2
   - 此时 maxPos = 2，表示可以跳到第 2 个位置。
3. **更新** left **和** right：
   - 更新 left = right + 1 = 1
   - 更新 right = maxPos = 2
   - 当前跳跃范围变为 [1, 2]

第二次循环

1. **检查是否可以到达终点**：
   - maxPos >= n - 1 (即 2 >= 4)，不成立，因此继续循环。
2. **更新** maxPos：

- for (int i = left; i <= right; i++)，即 i = 1 和 i = 2
- 当 i = 1 时：maxPos = max(2, nums[1] + 1) = max(2, 3 + 1) = 4
- 当 i = 2 时：maxPos = max(4, nums[2] + 2) = max(4, 1 + 2) = 4
- 最终 maxPos = 4，表示可以跳到第 4 个位置。

3. **更新** left **和** right：
   - 更新 left = right + 1 = 3

```cpp
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int left = 0, right = 0,maxPos = 0;
        int n = nums.size();
        while(left <= right)
        {
            if(maxPos >= n-1) return true;
            for(int i = left ;i <= right;i++)
            {
                maxPos = max(maxPos,nums[i] + i);
            }
            left = right + 1;
            right = maxPos;
        }
        return false;
    }
};
```

拓扑排序：LCR 114. 火星词典

```cpp
class Solution {
     unordered_map<char,unordered_set<char>> edges;
     unordered_map<char,int> in;
     bool check;
public:
   void add(string& s1, string& s2) {
        int n = min(s1.size(), s2.size());
        int i = 0;
        for (; i < n; i++) {
            if (s1[i] != s2[i]) {
                char a = s1[i];
                char b = s2[i];
                if (!edges[a].count(b)) { // 避免重复插入相同边
                    edges[a].insert(b);
                    in[b]++;
                }
                return; // 找到第一个不同的字符即可确定顺序
            }
        }
        // 检查不合法的前缀情况：s2 是 s1 的前缀且 s1 长于 s2
        if (s1.size() > s2.size()) check = true;
    }
    string alienOrder(vector<string>& words) {
        //1.入度
        for(auto& s: words)
        {
            for(auto& ch : s)
            {
                in[ch] = 0;
            }
        }

        int n = words.size();
        for(int i = 0; i < n;i++)
        {
            for(int j = i + 1;j < n;j++)
            {
                add(words[i], words[j]);
                if(check) return "";
```

```
            }
        }

        //2. 拓扑排序
        queue<char> q;
        for(auto& [a,b] : in)
        {
            if(b == 0) q.push(a);
        }
        string ret;
        while(q.size())
        {
            char t = q.front();
            q.pop();
            ret += t;
            for(char ch : edges[t])
            {
                if(--in[ch] == 0) q.push(ch);
            }
        }
        for(auto& [a,b] : in)
        {
            if(b != 0 ) return "";
        }
        return ret;
    }


};
```