# 题目

这里有一个非负整数数组 arr，你最开始位于该数组的起始下标 start 处。当你位于下标 i 处时，你可以跳到 i + arr[i] 或者 i - arr[i]。

请你判断自己是否能够跳到对应元素值为 0 的 任一 下标处。

注意，不管是什么情况下，你都无法跳到数组之外。

示例 1：

输入：arr = [4,2,3,0,3,1,2], start = 5

输出：true

解释：

到达值为 0 的下标 3 有以下可能方案：

下标 5 -> 下标 4 -> 下标 1 -> 下标 3

下标 5 -> 下标 6 -> 下标 4 -> 下标 1 -> 下标 3

示例 2：

输入：arr = [4,2,3,0,3,1,2], start = 0

输出：true

解释：

到达值为 0 的下标 3 有以下可能方案：

下标 0 -> 下标 4 -> 下标 1 -> 下标 3

示例 3：

输入：arr = [3,0,2,1,2], start = 2

输出：false

解释：无法到达值为 0 的下标 1 处。

提示：

1 <= arr.length <= 5 \* 10^4

0 <= arr[i] < arr.length

0 <= start < arr.length

# 分析

要判断能否从起始下标start跳到数组中任一值为0的下标，核心思路是搜索可达路径，可通过广度优先搜索（BFS）或深度优先搜索（DFS）实现，避免重复访问已检查的下标以提高效率。

解题思路

1、问题分析：

从start出发，每次可跳至i + arr[i]或i - arr[i]（需确保跳后下标在数组范围内）。目标是判断是否存在一条路径到达值为0的下标。

关键：避免无限循环（如下标i和j互相可达时，需标记已访问的下标）。

2、BFS 思路：

- 使用队列存储待访问的下标，初始将start入队。

- 使用布尔数组visited标记已访问的下标，防止重复访问。

- 每次从队列中取出一个下标i：

若arr[i] == 0，返回true（找到目标）。

计算可能的跳转下标next1 = i + arr[i]和next2 = i - arr[i]。

对每个有效的跳转下标（在数组范围内且未访问过），标记为已访问并加入队列。

- 若队列为空仍未找到目标，返回false。

代码实现（BFS 方法）：

class Solution {

public:

bool canReach(vector<int>& arr, int start) {

int n = arr.size();

vector<bool> visited(n, false); // 标记已访问的下标

queue<int> q; // 存储待访问的下标

// 初始化：将起始下标入队并标记为已访问

q.push(start);

visited[start] = true;

while (!q.empty()) {

int i = q.front();

q.pop();

// 若当前下标值为0，直接返回true

if (arr[i] == 0) {

return true;

}

// 计算两个可能的跳转下标

int next1 = i + arr[i];

int next2 = i - arr[i];

// 检查next1是否有效且未访问，若有效则加入队列

if (next1 < n && !visited[next1]) {

visited[next1] = true;

q.push(next1);

}

// 检查next2是否有效且未访问，若有效则加入队列

if (next2 >= 0 && !visited[next2]) {

visited[next2] = true;

q.push(next2);

}

}

// 队列为空，未找到目标

return false;

}

};

代码解释

1、初始化：

- visited数组用于记录已访问的下标，避免重复处理（如循环路径）。

- 队列q初始存入start，表示从起始位置开始搜索。

2、BFS 循环：

- 每次取出队首下标i，先检查arr[i]是否为0，若是则直接返回true。

- 计算两个跳转方向的新下标next1和next2，分别对应i + arr[i]和i - arr[i]。

- 对每个新下标，检查是否在数组范围内（0 ≤ next < n）且未被访问过，若满足则标记为已访问并加入队列，等待后续处理。

3、终止条件：

- 若在循环中找到值为0的下标，返回true。

- 若队列空了仍未找到，说明所有可达路径都已检查，返回false。

复杂度分析

- 时间复杂度：O(n)，其中n是数组长度。每个下标最多被访问一次（标记为visited后不再入队），队列操作的总次数为O(n)。

- 空间复杂度：O(n)。visited数组和队列的空间开销均为O(n)（最坏情况下所有下标入队）。