【哈希表】-跳房子I

题目描述与示例

题目描述

跳房子，也叫跳飞机，是一种世界性的儿童游戏。 游戏参与者需要分多个回合按顺序跳到第 `1` 格直到房子的最后一格

跳房子的过程中，可以向前跳，也可以向后跳。

假设房子的总格数是 `count` ，小红每回合可能连续跳的步教都放在数组 `steps` 中，请问数组中是否有一种步数的组合，可以让小红**两个回合**跳到最后一格? 如果有，请输出**索引和最小的步数组合**。

注意:

- 数组中的步数可以重复，但数组中的元素不能重复使用。
- 提供的数据保证存在满足题目要求的组合，且索引和最小的步数组合是**唯一的**。

输入描述

第一行输入为每回合可能连续跳的步数，它是整数数组类型。

第二行输入为房子总格数 `count` ，它是 `int` 整数类型。

输出描述

返回索引和最小的满足要求的步数组合(顺序保持 `steps` 中原有顺序)

备注

- `count ≤ 1000`
- `0 ≤ steps.length ≤ 5000`
- `-100000000 ≤ steps ≤ 100000000`

示例一

输入

```
1 [1,4,5,2]
2 7
```

输出

```
1 [5,2]
```

说明

示例二

输入

```
1 [-1,2,4,9,6]
2 8
```

输出

```
1 [-1,9]
```

解题思路

> 注意，本题和LeetCode1、两数之和几乎完全一致。区别在于需要输出索引和最小的两个数字。

本题关于哈希表的部分已经强调过很多遍了，因此不再花费篇幅进行介绍。

所以本篇题解主要介绍一些同学们在这一道题里面**比较困惑的一些细节处理上的问题**。

输入与输出

本题的输入其他很多题目不同的地方在于，输入的字符串虽然能够表示一个数组，但是其前后包含了中括号 `"[]"`

如果我们想要根据逗号 `","` 对原字符串进行切割的话，会得到第一个元素和最后一个元素分别包含 `"[` 和 `"]"` 。

为了避免这种情况，我们可以对 `input()` 得到的字符串利用切片进行掐头去尾的操作，即

```
1 s = input()[1:-1]
```

再结合内置函数 `map()` 和方法 `split()` ，我们可以得到数字列表 `nums`

```
1 nums = list(map(int, input()[1:-1].split(",")))
```

同理，本题答案的输出也需要包含中括号 `"[]"` 。

假设最终答案数组 `ans = [0, 1]` ，如果我们直接输出 `ans` ，即

```
1 print(ans)
```

会得到以下输出

```
1 [0, 1]
```

注意到逗号 `","` 的后面的包含空格 `" "` 的。但这是错误的输出结果，会得0分！

仔细观察题目要求的输出，逗号后面不应该包含空格。正确的输出应该是

```
1 [0,1]
```

为了能够正确得到上述结果，我们不得不使用格式化字符串的方式进行输出，即

```
print(f"[{ans[0]},{ans[1]}]")
```

这再一次提醒我们（在其他题目中也有所体现），**ACM模式的判题机制是相当严格的**。

题目要求你的答案输出和期望答案输出**完全一致**。

多一个空格少一个空格，中英文标点符号不分，都会**导致整道题目直接得0分**。

索引和最小值的初始值设置

本题和LeetCode1、两数之和最大的区别在于，本题可能存在多组和为 `target` 的数对。

题目要求我们需要输出**索引和最小的两个数字**。所以我们必须在原版题目的基础上加上这个过程。

首先我们考虑一个非常简单的问题。

如果要求我们在不使用内置函数 `min()` 和 `max()` 的前提下，找到一个数组 `nums` 中的最小值，我们会如何做？

这个问题非常简单，我们可以这样完成：

1. 设置一个初始值 `ans` 表示数组中可能的最小值。
2. 用循环遍历整个数组 `nums` ，如果发现某个数字 `num` 比初始值更小的时候，则我们将 `ans` 替换为这个数字。
3. 退出循环后，最终的 `ans` 即为答案。

```
nums = [4, 1, 3, 2]
ans = nums[0]         # 或者ans = math.inf

for num in nums:
    if ans > num:
        ans = num
```

在初始化 `ans` 的时候，我们有两种策略：

- `ans` 越大越好，必须要大过 `nums` 中的最小值。所以可以设置 `ans = inf` 。
- `ans` 是一个可能的答案。数组中的每一个元素都是可能的答案，所以可以设置 `ans = nums[0]` 。

回到这道题本身，我们要找到索引和最小的两个数字，可以设置一个变量 `min_idx_sum` 来表示这个索引和。

那么很多同学困惑的地方在于，下面的代码为什么是这样初始化变量 `min_idx_sum` 的。

```
1 min_idx_sum = len(nums)*2
```

由于后面我们希望能够找到更小的索引和，所以我们可以把 `min_idx_sum` 初始化为一个较大值。

显然，对于长度 `n = len(nums)` 而言的数组，最大的两个索引是 `n-1` 和 `n-2` ，它们的和小于 `n*2` 。

当我们设置 `min_idx_sum` 的初始值为 `len(nums)*2` 时，其实跟设置 `inf` 或者 `len(nums)*2-2` 是一样的。

> PS：在此我也希望大家能够举一反三。
>
> 如果某个题目问的是寻找最大值且元素都是正数的话，那么我们就可以设置 `ans` 初始值为 `0` 或者 `-inf` 了。
>
> 一定要学会变通。

寻找索引和最小值

接下来的任务就是，如何把索引和最小值的问题，加入到经典题目LeetCode1、两数之和中。

如果不加限制，其算法过程如下：

1. 遍历 `nums` 的每一个索引 `i` 和对应的数字 `num`
2. 计算剩余数字 `rest_num = target-num` 是否位于哈希表 `hash_dic` 中，若在则得到答案

3. 将当前数字 `num` 作为 `key` ，当前索引 `i` 作为 `value` ，将键值对 `(num, i)` 储存入哈希表 `hash_dic` 中

其代码如下

```
for i, num in enumerate(nums):
    rest_num = target-num
    if rest_num in hash_dic:
        ans = [rest_num, num]
    hash_dic[num] = i
```

但由于我们希望寻找到的是索引和最小值。假设在原数组中存在两个相同的数字，我们在哈希表中储存的索引 `i` ，必然希望它是更早出现的那个而不是更晚出现的那个。

所以在更新哈希表的时候，需要额外多加一个判断：如果 `num` 已经位于哈希表中了，则不进行更新。

修改后代码如下

```
for i, num in enumerate(nums):
    rest_num = target-num
    if rest_num in hash_dic:
        ans = [rest_num, num]
    if num not in hash_dic:
        hash_dic[num] = i
```

另外，只有当当前两数索引和，小于全局索引和的时候，我们才需要更新答案 `ans` 。

当前数字 `num` 的索引为 `i` ，剩余数字 `rest_num` 的索引为 `hash_dic[rest_num]` 。故修改后代码如下

```
for i, num in enumerate(nums):
    rest_num = target-num
    if rest_num in hash_dic:
        if min_idx_sum > hash_dic[rest_num] + i:
            min_idx_sum = hash_dic[rest_num] + i
            ans = [rest_num, num]
    if num not in hash_dic:
```

```
8           hash_dic[num] = i
```

特别注意，当`min_idx_sum > hash_dic[rest_num] + i`成立后，既要修改答案列表`ans`，也要修改全局的索引最小和`min_idx_sum`为当前更新的`hash_dic[rest_num] + i`。

综上就构成了本题的核心算法逻辑了。

*不能在第一次找到答案就退出

在和同学们交流的过程中，有同学提出了以下解法。

```
1 for i, num in enumerate(nums):
2     rest_num = target-num
3     if rest_num in hash_dic:
4         ans = [rest_num, num]
5         break
6     if num not in hash_dic:
7         hash_dic[num] = i
```

即在第一次找到符合要求的答案时就退出。

同学的理由非常简单，由于遍历是从左到右进行的，那么找到的第一组自然就是索引和最小的情况。

但这是一个比较典型的、过于想当然的思路。

我们可以非常容易地举出反例。考虑例子

```
1 target = 9
2 nums = [1, 5, 6, 3, 8]
```

显然有两组索引`[0, 4]`和`[2, 3]`（注意是索引而非元素本身）能使得`nums`中对应的两个数的和为`target`。

如果是从左往右遍历，我们会先找到索引对`[2, 3]`，但显然`[2, 3]`的和会小于`[0, 4]`。

一旦在找到`[2, 3]`时就提前退出，那么将找不到正确结果`[0, 4]`。

代码

Python

```python
# 题目：2024E-跳房子I
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：哈希表
# 代码看不懂的地方，请直接在群上提问


# 输入步数列表，注意需要去除最开头和最末尾的中括号
nums = list(map(int, input()[1:-1].split(",")))
target = int(input())

# 初始化索引和最小值的取值，这里可以取inf，也可以取nums长度乘2
min_idx_sum = len(nums)*2
# 构建哈希表，储存每一种步数首次出现的下标
hash_dic = dict()
# 初始化答案列表
ans = [0, 0]

# 遍历nums
for i, num in enumerate(nums):
    # 计算target-num的值rest_num
    rest_num = target-num
    # 若rest_num位于哈希表中，说明其曾经出现过
    # rest_num和num相加等于所需要的结果target
    if rest_num in hash_dic:
        # 若此时min_idx_sum大于两者下标和
        # 则更新min_idx_sum和ans
        if min_idx_sum > hash_dic[rest_num] + i:
            min_idx_sum = hash_dic[rest_num] + i
            # 注意题目要求两个元素保持原有顺序
            ans = [rest_num, num]
    # 若num不位哈希表中，说明它第一次出现，记录其下标
    # 由于我们希望两数的下标和尽可能小
    # 故对于重复出现的数字，只记录其第一次出现的下标即可
    if num not in hash_dic:
        hash_dic[num] = i

```

```python

# 输出答案，注意要按照格式输出，逗号后面不带空格，不能直接输出ans
print(f"[{ans[0]},{ans[1]}]")
```

Java

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        input = input.substring(1, input.length() - 1); // Remove square brackets
        String[] numStrings = input.split(",");
        int[] nums = new int[numStrings.length];

        for (int i = 0; i < nums.length; i++) {
            nums[i] = Integer.parseInt(numStrings[i].trim());
        }

        int target = scanner.nextInt();
        int minIdxSum = nums.length * 2;
        HashMap<Integer, Integer> hashDic = new HashMap<>();
        int[] ans = new int[2];

        for (int i = 0; i < nums.length; i++) {
            int num = nums[i];
            int restNum = target - num;

            if (hashDic.containsKey(restNum)) {
                if (minIdxSum > hashDic.get(restNum) + i) {
                    minIdxSum = hashDic.get(restNum) + i;
                    ans[0] = restNum;
                    ans[1] = num;
                }
            }

            if (!hashDic.containsKey(num)) {
                hashDic.put(num, i);
            }
        }

        System.out.println("[" + ans[0] + "," + ans[1] + "]");
    }
```

```
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <sstream>
using namespace std;

int main() {
    string input;
    getline(cin, input);
    input = input.substr(1, input.length() - 2); // Remove square brackets
    istringstream iss(input);
    vector<int> nums;
    string numStr;

    while (getline(iss, numStr, ',')) {
        nums.push_back(stoi(numStr));
    }

    int target;
    cin >> target;
    int minIdxSum = nums.size() * 2;
    unordered_map<int, int> hashDic;
    vector<int> ans(2);

    for (int i = 0; i < nums.size(); i++) {
        int num = nums[i];
        int restNum = target - num;

        if (hashDic.find(restNum) != hashDic.end()) {
            if (minIdxSum > hashDic[restNum] + i) {
                minIdxSum = hashDic[restNum] + i;
                ans[0] = restNum;
                ans[1] = num;
            }
        }

        if (hashDic.find(num) == hashDic.end()) {
            hashDic[num] = i;
        }
    }
```

```
41
42      cout << "[" << ans[0] << "," << ans[1] << "]" << endl;
43
44      return 0;
45  }
46
```

时空复杂度

时间复杂度：`O(N)`。仅需一次遍历数组。

空间复杂度：`O(N)`。哈希表所占空间。