水壶问题 题解

> 计算学部十大打卡活动——“龙舞编程新春会”编程打卡（1月28日）
>
> 题目为力扣 365. 水壶问题

题目

有两个水壶，容量分别为 `jug1Capacity` 和 `jug2Capacity` 升。水的供应是无限的。确定是否有可能使用这两个壶准确得到 `targetCapacity` 升。

如果可以得到 `targetCapacity` 升水，最后请用以上水壶中的一或两个来盛放取得的 `targetCapacity` 升水。

你可以：

- 装满任意一个水壶
- 清空任意一个水壶
- 从一个水壶向另外一个水壶倒水，直到装满或者倒空

示例 1:

```
输入: jug1Capacity = 3, jug2Capacity = 5, targetCapacity = 4
输出: true
解释: 来自著名的 "Die Hard"
```

示例 2:

```
输入: jug1Capacity = 2, jug2Capacity = 6, targetCapacity = 5
输出: false
```

示例 3:

```
输入: jug1Capacity = 1, jug2Capacity = 2, targetCapacity = 3
输出: true
```

提示:

- `1 <= jug1Capacity, jug2Capacity, targetCapacity <= 106`

题解

DFS

观察题目可知，在任意一个时刻，我们可以且仅可以采取以下6种操作：

1. 把 X 壶灌满；
2. 把 Y 壶灌满；
3. 把 X 壶倒空；
4. 把 Y 壶倒空。
5. 把 X 壶的水灌进 Y 壶，直至灌满或倒空；
6. 把 Y 壶的水灌进 X 壶，直至灌满或倒空；

因此，本题可以使用**深度优先搜索**来解决，并将问题的状态设定为**<X 壶中的水量, Y 壶中的水量>**。搜索过程中我们用哈希表存储所有已经搜索过的状态，避免重复搜索或进入死循环。

```
class Solution {
public:
    bool ans = false;
    map<pair<int, int>, int> hash;
    void dfs(int x, int y, const int cx, const int cy, const int target)
    {
        if (!ans && !hash[make_pair(x, y)]) {
            hash[make_pair(x, y)] = 1;
            if (x == target || y == target || x + y == target) {
                ans = true;
            }
            else {  // 6种情况
                dfs(cx, y, cx, cy, target);
                dfs(x, cy, cx, cy, target);
                dfs(0, y, cx, cy, target);
                dfs(x, 0, cx, cy, target);
                dfs(x - min(x, cy - y), y + min(x, cy - y), cx, cy, target);
                dfs(x + min(y, cx - x), y - min(y, cx - x), cx, cy, target);
            }
        }
        return;
    }
    bool canMeasureWater(int jug1Capacity, int jug2Capacity, int targetCapacity)
{
        dfs(0, 0, jug1Capacity, jug2Capacity, targetCapacity);
        return ans;
    }
};
```

用复杂度带 log 的 `map<pair<int, int>, int>` 会超时，因此写一个 `pair<int, int>` 的哈希函数，用 `unordered_map` 或者 `unordered_set` 即可 $O(1)$ 的检测当前状态是否已被搜索过。

注意不能直接用 `unordered_map<pair<int, int>, int>`，因为 $C++$ 没有给 `pair<int, int>` 做哈希函数，需要自己写一个。