

416. Partition Equal Subset Sum

Medium

Given an integer array `nums`, return `true` if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or `false` otherwise.

```
from typing import List

class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        if sum(nums) % 2 != 0:
            return False

        target = sum(nums) // 2
        check = set()
        check.add(0)

        for i in range(len(nums) - 1, -1, -1):
            next_check = set()
            for t in check:
                if (t + nums[i]) == target:
                    return True
                next_check.add(t + nums[i])
                next_check.add(t)
            check = next_check
        return False
```

Example 1:

Input: `nums = [1,5,11,5]`

Output: `true`

Explanation: The array can be partitioned as `[1, 5, 5]` and `[11]`.

Example 2:

Input: `nums = [1,2,3,5]`

Output: `false`

Explanation: The array cannot be partitioned into equal sum subsets.

Constraints:

`1 <= nums.length <= 200`

`1 <= nums[i] <= 100`

`[1, 5, 11, 5]`
↓ ↓
`[1, 5, 5]` `[11]`

target = 11

check = set()

check → set(0)

1. Loop through the `nums` list
2. The 'check' set() holds the possible combination of numbers that have been previously seen that could sum up to the target.
3. The 'next_check' set() holds new combinations of sums
4. As the values in 'check', `t`, are iterated over, check to see if `nums[i] + t` is equal to target. If so, return `True` (otherwise, if the loop completes, return `False`)

`[1, 5, 11, 5]`

```
for i in range(len(nums) - 1, -1, -1):
```

```
    next_check = set()
```

```
    for t in check:
```

```
        if t + nums[i] == target:
```

```
            return True
```

```
        next_check.add(t + nums[i])
```

```
        next_check.add(t)
```

```
    check = next_check
```

