Split Array with Equal Sum

Given an array with n integers, you need to find if there are triplets (i, j, k) which satisfies following conditions:

1. 0
2. Sum of subarrays (0, i - 1), (i + 1, j - 1), (j + 1, k - 1) and (k + 1, n - 1) should be equal.

where we define that subarray (L, R) represents a slice of the original array starting from the element indexed L to the element indexed R.

Example:

```
Input: [1,2,1,2,1,2,1]
Output: True
Explanation:
i = 1, j = 3, k = 5.
sum(0, i - 1) = sum(0, 0) = 1
sum(i + 1, j - 1) = sum(2, 2) = 1
sum(j + 1, k - 1) = sum(4, 4) = 1
sum(k + 1, n - 1) = sum(6, 6) = 1
```

Note:

1. 1
2. Elements in the given array will be in range [-1,000,000, 1,000,000].

Solution 1

Here j is used for middle cut, i for left cut and k for right cut.
Iterate middle cuts and then find left cuts which divides the first half into two equal quarters, store that quarter sums in the hashset. Then find right cuts which divides the second half into two equal quarters and check if quarter sum is present in the hashset. If yes return true.

```
public class Solution {
    public boolean splitArray(int[] nums) {
        if (nums.length < 7)
            return false;
        int[] sum = new int[nums.length];
        sum[0] = nums[0];
        for (int i = 1; i < nums.length; i++) {
            sum[i] = sum[i - 1] + nums[i];
        }
        for (int j = 3; j < nums.length - 3; j++) {
            HashSet < Integer > set = new HashSet < > ();
            for (int i = 1; i < j - 1; i++) {
                if (sum[i - 1] == sum[j - 1] - sum[i])
                    set.add(sum[i - 1]);
            }
            for (int k = j + 2; k < nums.length - 1; k++) {
                if (sum[nums.length - 1] - sum[k] == sum[k - 1] - sum[j] && set.con
tains(sum[k - 1] - sum[j]))
                    return true;
            }
        }
        return false;
    }
}
```

written by vinod23 original link here

Solution 2

Just think this problem as a DFS. What we need is to search for `3` positions (i, j, k) and see if they divide the array to `4` parts with same summary. Some tricks:

1. Calculate `left` on the fly. Thus at last we don't need to calc summary of the `4th` part.
2. Skip `0` during calculate `target` because adding zero won't change it.

```
public class Solution {
    public boolean splitArray(int[] nums) {
        int sum = 0, target = 0;
        for (int num : nums) sum += num;
        for (int i = 1; i + 5 < nums.length; i++) {
            if (i != 1 && nums[i - 1] == 0  && nums[i] == 0) continue;
            target += nums[i - 1];
            if (dfs(nums, i + 1, target, sum - target - nums[i], 1)) return true;
        }
        return false;
    }

    private boolean dfs(int[] nums, int start, int target, int left, int depth) {
        if (depth == 3) {
            if (left == target) return true;
            return false;
        }

        int sub = 0;
        for (int j = start + 1; j + 5 - depth * 2 < nums.length; j++) {
            sub += nums[j - 1];
            if (sub == target) {
                if (dfs(nums, j + 1, target, left - sub - nums[j], depth + 1)) {
                    return true;
                }
            }
        }

        return false;
    }
}
```

written by shawngao original link here

Solution 3

Let **A** be the array. As in most problems involving querying the sum of contiguous elements of an array, let P[x] = sum(A[:x]) be the prefix sums of A, which can be found in linear time.

Then the sums in question are P[i] = P[j] - P[i+1] = P[k] - P[j+1] = P[-1] - P[k+1]. For every j < k, P[i] = P[-1] - P[k+1] is a necessary requirement to choose i, so let's iterate over those indices first. This gives us the advantage that since we are iterating over a sorted list of candidate indices i, we can break when i >= j.

```
def splitArray(self, A):
    P = [0]
    for x in A: P.append(P[-1] + x)

    N = len(A)
    Pinv = collections.defaultdict(list)
    for i, u in enumerate(P):
        Pinv[u].append(i)

    for j in xrange(1, N-1):
        for k in xrange(j+1, N-1):
            for i in Pinv[P[-1] - P[k+1]]:
                if i >= j: break
                if P[i] == P[j] - P[i+1] == P[k] - P[j+1]:
                    return True
    return False
```

written by awice original link here