Delete and Earn

Given an array `nums` of integers, you can perform operations on the array.

In each operation, you pick any `nums[i]` and delete it to earn `nums[i]` points. After, you must delete **every** element equal to `nums[i] - 1` or `nums[i] + 1`.

You start with 0 points. Return the maximum number of points you can earn by applying such operations.

Example 1:

```
Input: nums = [3, 4, 2]
Output: 6
Explanation:
Delete 4 to earn 4 points, consequently 3 is also deleted.
Then, delete 2 to earn 2 points. 6 total points are earned.
```

Example 2:

```
Input: nums = [2, 2, 3, 3, 3, 4]
Output: 9
Explanation:
Delete 3 to earn 3 points, deleting both 2's and the 4.
Then, delete 3 again to earn 3 points, and 3 again to earn 3 points.
9 total points are earned.
```

Note:

- The length of `nums` is at most `20000`.
- Each element `nums[i]` is an integer in the range `[1, 10000]`.

Solution 1

This question can be reduced to the House Robbers question also on LeetCode. Please have a look at it if you haven't seen it before.

Observations:

- The order of `nums` does not matter.
- Once we decide that we want a `num`, we can add all the occurrences of `num` into the total.

We first transform the `nums` array into a `points` array that sums up the total number of points for that particular value. A value of `x` will be assigned to index `x` in `points`.

`nums`: `[2, 2, 3, 3, 3, 4]` (2 appears 2 times, 3 appears 3 times, 4 appears once)
`points`: `[0, 0, 4, 9, 4]` <- This is the gold in each house!

The condition that we cannot pick adjacent values is similar to the House Robber question that we cannot rob adjacent houses. Simply pass `points` into the `rob` function for a quick win 🅰🅰!

- *Yangshun*

```python
class Solution(object):
    def rob(self, nums):
        prev = curr = 0
        for value in nums:
            prev, curr = curr, max(prev + value, curr)
        return curr

    def deleteAndEarn(self, nums):
        points = [0] * 10001
        for num in nums:
            points[num] += num
        return self.rob(points)
```

When `rob` is used directly, it is just 6 lines:

```python
class Solution(object):
    def deleteAndEarn(self, nums):
        points, prev, curr = [0] * 10001, 0, 0
        for num in nums:
            points[num] += num
        for value in points:
            prev, curr = curr, max(prev + value, curr)
        return curr
```

Suggested by @ManuelP, it can be further shortened into 4 lines if you use `collections.Counter` and modify the `rob` function:

```python
class Solution(object):
    def deleteAndEarn(self, nums):
        points, prev, curr = collections.Counter(nums), 0, 0
        for value in range(10001):
            prev, curr = curr, max(prev + value * points[value], curr)
        return curr
```

written by yangshun original link here

Solution 2

1. If we sort all the numbers into `buckets` indexed by these numbers, this is essentially asking you to repetitively take an bucket while giving up the 2 buckets next to it. (the range of these numbers is [1, 10000])
2. The optimal final result can be derived by keep updating 2 variables `skip_i`, `take_i`, which stands for:
   `skip_i` : the best result for sub-problem of first `(i+1)` buckets from `0` to `i`, while you **skip** the `i` th bucket.
   `take_i` : the best result for sub-problem of first `(i+1)` buckets from `0` to `i`, while you **take** the `i` th bucket.
3. DP formula:
   `take[i] = skip[i-1] + values[i];`
   `skip[i] = Math.max(skip[i-1], take[i-1]);`
   `take[i]` can only be derived from: if you skipped the `[i-1]` th bucket, and you take bucket[i].
   `skip[i]` through, can be derived from either `take[i-1]` or `skip[i-1]`, whatever the bigger;

```
/**
 * for numbers from [1 - 10000], each has a total sum sums[i]; if you earn sums[i],
you cannot earn sums[i-1] and sums[i+1]
 * kind of like house robbing. you cannot rob 2 connected houses.
 *
 */
```

Java

```java
class Solution {
    public int deleteAndEarn(int[] nums) {
        int n = 10001;
        int[] values = new int[n];
        for (int num : nums)
            values[num] += num;

        int take = 0, skip = 0;
        for (int i = 0; i < n; i++) {
            int takei = skip + values[i];
            int skipi = Math.max(skip, take);
            take = takei;
            skip = skipi;
        }
        return Math.max(take, skip);
    }
}
```

C++

```cpp
class Solution {
public:
    int deleteAndEarn(vector<int>& nums) {
        int n = 10001;
        vector<int> values(n, 0);
        for (int num : nums)
            values[num] += num;

        int take = 0, skip = 0;
        for (int i = 0; i < n; i++) {
            int takei = skip + values[i];
            int skipi = max(skip, take);
            take = takei;
            skip = skipi;
        }
        return max(take, skip);
    }
};
```

written by alexander original link here

Solution 3

Time: O(M+N)
Space: O(N)

M: the length of input array
N: the range of the value of each int element

```
public int deleteAndEarn(int[] nums) {
    int[] count = new int[10001];
    for(int n : nums){
        count[n] += n;
    }
    int[] dp = new int[10003];
    for(int i = 10000; i >= 0; i--) {
        dp[i] = Math.max(count[i] + dp[i + 2], dp[i + 1]);
    }
    return dp[0];
}
```

written by luckman original link here