Weekly Discussion

Tong Wei

Department of Computer Science
Nanjing University

September 25, 2016

Outline

Problem Description

Given a non negative integer number **num**. For every numbers **i** in the range $\mathbf{0} \leq \mathbf{i} \leq \mathbf{num}$ calculate the number of 1's in their binary representation and return them as an array.

Example:
For num = 5 you should return [0,1,1,2,1,2].

Solution

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**.
- for (int j = 0; j < 32; ++ j) if ((1<<j) & i) {}
- Can we do better?

Solution

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**.
- for (int j = 0; j < 32; ++ j) if ((1<<j) & i) {}
- Can we do better?

Solution

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**.
- for (int j = 0; j < 32; ++ j) if ((1<<j) & i) {}
- Can we do better?

Solution

- What can we get if the number of 1's in number i's binary representation is already known.

Example

- ans[i << 1] = ans[i]
- ans[(i<<1) + 1] = ans[i] + 1

Solution

- What can we get if the number of 1's in number i's binary representation is already known.

Example

- ans[i << 1] = ans[i]
- ans[(i<<1) + 1] = ans[i] + 1

Solution

```
1       vector<int> countBits(int num) {
2           vector<int> ans(num + 1, 0);
3           for (int i = 0; i <= num/2; ++ i) {
4               if ((i<<1) <= num) {
5                   ans[i<<1] = ans[i];
6               }
7               if ((i<<1) + 1 <= num) {
8                   ans[(i<<1) | 1] = ans[i] + 1;
9               }
10          }
11          return ans;
12      }
```

Problem Description

Given a set of distinct positive integers, find the largest subset such that every pair (S_i, S_j) of elements in this subset satisfies: $S_i \% S_j = 0$ or $S_j \% S_i = 0$.
If there are multiple solutions, return any subset is fine.

Sample 1
num: [1, 2, 3] Result: [1, 2] (of course, [1,3] will also be ok)
Sample 2
nums: [1,2,4,8] Result: [1,2,4,8]

Solution

- How to check if a given set satisfies above constraints?
- What's the running time complexity?

Solution

- How to check if a given set satisfies above constraints?
- What's the running time complexity?

Solution

```
1  def largestDivisibleSubset(self, nums):
2      S = {-1: set()}
3      for x in sorted(nums):
4          S[x] = max((S[d] for d in S if x % d
               == 0), key=len) | {x}
5      return list(max(S.values(), key=len))
```

Problem Description

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

Sample Input:
[2,3,-2,4]

Sample Output:
6 (2 * 3)

Solution

- Enumerate every possible subarray. How?
- Time Complexity: $O(n\hat{}3)$.
- Can we do better?

Solution

- Enumerate every possible subarray. How?
- Time Complexity: $O(n\hat{}3)$.
- Can we do better?

Solution

- Enumerate every possible subarray. How?
- Time Complexity: $O(n\hat{}3)$.
- Can we do better?

Solution

Observation: It's evident that for any L and R with L <= R, we have prod[L...R] = prefix[R] / prefix[L - 1].

- So how to implement?
- Time Complexity: $O(n\hat{}2)$.
- Can we do better?

Solution

Observation: It's evident that for any L and R with L <= R, we have prod[L...R] = prefix[R] / prefix[L - 1].

- So how to implement?
- Time Complexity: $O(n\hat{}2)$.
- Can we do better?

Solution

Observation: It's evident that for any L and R with L <= R, we have prod[L...R] = prefix[R] / prefix[L - 1].

- So how to implement?
- Time Complexity: $O(n\hat{}2)$.
- Can we do better?

Solution

Observation: Eliminate useless enumeration by .....

- So how to improve?
- Time Complexity: $O(n)$.
- Can we do better?
- Maybe NO.

Solution

Observation: Eliminate useless enumeration by .....

- So how to improve?
- Time Complexity: $O(n)$.
- Can we do better?
- Maybe NO.

Solution

Observation: Eliminate useless enumeration by .....

- So how to improve?
- Time Complexity: $O(n)$.
- Can we do better?
- Maybe NO.

Solution

Observation: Eliminate useless enumeration by .....

- So how to improve?
- Time Complexity: $O(n)$.
- Can we do better?
- Maybe NO.

Solution

```
1  int maxProduct(int A[], int n) {
2  // store the result that is the max we have
       found so far
3       int r = A[0];
4  // imax/imin stores the max/min product of
       subarray that ends with the current number
5       for (int i = 1, imax = r, imin = r; i < n
            ; i++) {
6            if (A[i] < 0) swap(imax, imin);
7            imax = max(A[i], imax * A[i]);
8            imin = min(A[i], imin * A[i]);
9            r = max(r, imax);
10      }
11      return r;
12 }
```

The End

Thank you!