Archax

2022 Nov04

Question 1

You are given an integer array prices where `prices[i]` is the price of a given stock on the i-th day. On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day. Find and return the maximum profit you can achieve.

Example 1 : prices = [7,1,5,3,6,4], output is 7

- buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4
- buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3
- total profit is 4 + 3 = 7

Example 2 : prices = [1,2,3,4,5], output is 4

- buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4
- total profit is 4

Example 3 : prices = [7,6,4,3,1], output is 0

- there is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0

My solution

```
int max_profit(vector<int>& prices)
{
    std::vector<int> cashT0;
    cashT0.push_back(0);
    cashT0.push_back(-prices[0]);
    for(int t=1; t!=prices.size(); ++t)
    {
        std::vector<int> cashT1(2, 0);
        cashT1[0] = std::max(cashT0[0], cashT0[1]+prices[t]);
        cashT1[1] = std::max(cashT0[1], cashT0[0]-prices[t]);
        cashT0 = std::move(cashT1);
    }
    return cashT0[0];
}

std::vector<int> prices1{7,1,5,3,6,4};      std::cout << "\n" << maxProfit(prices1);
std::vector<int> prices2{1,2,3,4,5};        std::cout << "\n" << maxProfit(prices2);
std::vector<int> prices3{7,6,4,3,1};        std::cout << "\n" << maxProfit(prices3);
```

Simpler solution

As we can buy today and sell the next day, there is no transaction cost, no multiple shares, hence the logic can be simple :

```
int max_profit(vector<int>& prices)
{
    int sum = 0;
    for (int i=1; i<prices.size(); ++i)
    {
        if (prices[i] > prices[i-1])
        {
            sum += prices[i] - prices[i-1];
        }
    }
    return sum;
}
```

Question 2
Given an unsorted array of signed integers, there exists duplicated numbers, find the minimum missing positive number with *O(N)* time and *O(1)* space.

Examples :

- `[4,3,2,1]` returns `5`
- `[5,4,3,2]` returns `1`
- `[8,7,6,5,4]` returns `1`
- `[5,4,-3,2,1]` returns `3`
- `[3,8,2,6,-1,4,-3,0,1,5]` returns `7`
- `[3,8,2,6,2,4,3,0,1,5]` returns `7`

My solution in O(NlogN) and O(1) space

```
std::int32_t min_missing_pos_num_sort(std::vector<std::int32_t>& vec)
{
    std::sort(vec.begin(), vec.end());
    std::int32_t ans = 1;
    for(std::uint32_t n=0; n!=vec.size(); ++n)
    {
        if (vec[n] <= 0)          continue;
        if (vec[n] >  ans)        return ans;
        if (vec[n] == ans)        ++ans;
    //  if (vec[n] == ans-1)      duplicated number, we should proceed
    }
    return vec.size()+1;
}
```

My solution in O(N) time and O(N) space

```
std::int32_t min_missing_pos_num_flag(const std::vector<std::int32_t>& vec)
{
    std::vector<bool> flags(vec.size(), false);
    for(const auto& x:vec)
    {
        if (x <= 0)          continue;
        if (x > vec.size()) continue;
        flags[x-1] = true;
    }
    for(std::uint32_t n=0; n!=flags.size(); ++n)
    {
        if (!flags[n]) return n+1;
    }
    return vec.size()+1;
}
```

My solution in O(N) time and O(1) space (but changing the input, assume all numbers within +/-BOUND)

```
static const std::int32_t BOUND = 1024;
void mark_visited(std::int32_t& value)
{
    if (value <= -BOUND)          return; // it is marked as visited before
    if (value >  +BOUND)          return; // it is marked as visited before
    if (value <= 0)               value -= BOUND;
    else                          value += BOUND;
}

auto get_value_and_flag(const std::int32_t value)
{
    if     (value <= -BOUND)     return std::make_pair(value+BOUND, true);
    else if(value >  +BOUND)     return std::make_pair(value-BOUND, true);
    else                         return std::make_pair(value,      false);
}
```

```
std::int32_t min_missing_pos_num_bounded(std::vector<std::int32_t>& vec)
{
    // step 1 : mark visited
    for(const auto& x:vec)
    {
        auto [value,flag] = get_value_and_flag(x); // flag is not used in this case
        if (value > 0 && value <= vec.size())
        {
            mark_visited(vec[value-1]);
        }
    }

    // step 2 : check if visited
    for(std::int32_t n=0; n!=vec.size(); ++n)
    {
        auto [value,flag] = get_value_and_flag(vec[n]); // value is not used in this case
        if (!flag) return n+1;
    }
    return vec.size()+1;
}
```

My solution in O(N) time and O(1) space without any assumption

```
// Given contiguous positive integer starting from 1 with missing numbers,
// special sorting is defined, such that
// vec[0] = 1
// vec[1] = 2
// ...
// vec[n] = n+1
//
// with some missing numbers in between
// vec[m] <= 0 or
// vec[m] >  vec.size()
//
// This sorting can be done in O(N), simply by placing vec[n] in vec[vec[n]-1].
// For example : 1,2,3,-2,5,6,0,8,2,10,11,99,13,14,15 where ...
//  4 is missing and replaced by -2
//  7 is missing and replaced by  0
//  9 is missing and replaced by  2 (i.e. 2 is duplicated)
// 12 is missing and replaced by 99

void special_sort(std::vector<std::int32_t>& vec)
{
    for(std::uint32_t n=0; n!=vec.size(); ++n)
    {
        bool duplicated = false;

        // for valid numbers, put them in correct place, using swap (since there is no extra space)
        while(vec[n]>=1 && vec[n]<=vec.size() && !duplicated)
        {
            if (vec[n] != vec[vec[n]-1])
            {
                std::swap(vec[n], vec[vec[n]-1]);
            }
            else
            {
                duplicated = true;
            }
        }
    }
}

std::int32_t min_missing_pos_num_special_sort(std::vector<std::int32_t>& vec)
{
    special_sort(vec);                          // O(N)
    for(std::uint32_t n=0; n!=vec.size(); ++n)  // O(N)
    {
        if (vec[n] != n+1) return n+1;
    }
    return vec.size()+1;
}
```