LeetCode Answers

Keaton Clark

May 2, 2022

Contents

Part I

Easy

Chapter 1

1.Two Sum

1.1 Description

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.
You may assume that each input would have exactly one solution, and you may not use the same element twice.
You can return the answer in any order.

1.2 Results

Runtime: 7 ms, faster than 97.35% of C++ online submissions for Two Sum.
Memory Usage: 11 MB, less than 29.34% of C++ online submissions for Two Sum.

1.3 Attempt 1

It's finals week so I wanted to do a easier one. Just use an unordered map for constant lookup.

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> map;
        map.reserve(nums.size());
        map[nums[0]] = 0;
        for (int i = 1; i < nums.size(); i++) {
            if (map.find(target-nums[i]) != map.end()){
                return {map[target - nums[i]], i};
            }
            map[nums[i]] = i;
        }
        return {};
    A}
};
```

Chapter 2

191. Number of 1 bits

2.1 Description

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).
Note:
Note that in some languages, such as Java, there is no unsigned integer type. In this case, the input will be given as a signed integer type. It should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned. In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in Example 3, the input represents the signed integer. -3.

2.2 Results

Runtime: 0 ms, faster than 100.00% of C++ online submissions for Number of 1 Bits. **Memory Usage:** 6 MB, less than 48.80% of C++ online submissions for Number of 1 Bits.

2.3 Attempt 1

Easy fun bit manipulation. Saves space by not creating a variable in a for loop.

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int out = 0;
        while (n) {
            if (n & 0b1)
                ++out;
            n >>= 1;
        }
        return out;
    }
};
```

Chapter 3

905. Sort Array By Parity

3.1 Description

Given an integer array nums, move all the even integers at the beginning of the array followed by all the odd integers. Return any array that satisfies this condition.

3.2 Results

Runtime: 8 ms, faster than 88.01% of C++ online submissions for Sort Array By Parity.
16.1 MB, less than 83.25% of C++ online submissions for Sort Array By Parity.
Memory Usage:

3.3 Attempt 1

Because it doesn't need to be in any specific order you can sort in place to preserve space. Have a pointer to the front and end and swap everytime the end pointer comes across an even number

```
class Solution {
public:
    vector<int> sortArrayByParity(vector<int>& nums)
    {
        int i=0,j=nums.size()-1;
        while(i<=j)
        {
            if(nums[j]%2==0)
                swap(nums[i++],nums[j]);
            else
                j--;
        }
        return nums;
    }
};
```

Part II

Medium

Chapter 4

2. Add Two numbers

4.1 Description

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.
You may assume the two numbers do not contain any leading zero, except the number 0 itself.

4.2 Results

Runtime: 47 ms, faster than 55.66% of C++ online submissions for Add Two Numbers.
Memory Usage: 71.4 MB, less than 50.94% of C++ online submissions for Add Two Numbers.

4.3 Attempt 1 - 2022-04-23

Worked great until I realized that the numbers can be up to $9 * 10^{100}$. Turn each list into ints, sums them, and uses a recursive function to convert it back into a reversed linked list.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode *addTwoNumbers(ListNode* l1, ListNode* l2) {
        int place = 1;
        int s1 = 0, s2 = 0;
        while(l1) {
            s1 += l1->val * place;
            place *= 10;
            l1 = l1->next;
        }
        place = 1;
        while(l2) {
            s2 += l2->val * place;
            place *= 10;
            l2 = l2->next;
        }
        int sum = s1 + s2;
        return createNodes(sum, nullptr);
    }
    ListNode *createNodes(int x, ListNode *curr) {
        if (x >= 10)
            curr = createNodes(x / 10, curr);
        return new ListNode(x % 10, curr);
    }
};
```

4.4 Attempt 2 - 2022-04-23

The correct way to do it. Sums them just like an adder in a cpu would. Sums each digit and has a carry digit to keep track of overflows.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */

class Solution {
public:
    ListNode *addTwoNumbers(ListNode* l1, ListNode* l2) {
        int sum = l1->val + l2->val;
        int carry = (sum > 9) ? 1 : 0;
        auto output = new ListNode();
        auto curr = output;
        l1 = l1->next;
        l2 = l2->next;
        if (carry == 1) {
            curr->val = sum % 10;
        } else {
            curr->val = sum;
        }
        while (l1 && l2) {
            sum = l1->val + l2->val + carry;
            carry = (sum > 9) ? 1 : 0;
            curr->next = new ListNode();
            curr = curr->next;
            if (carry == 1) {
                curr->val = sum % 10;
            } else {
                curr->val = sum;
            }
            l1 = l1->next;
            l2 = l2->next;
        }
        while (l1) {
            sum = l1->val + carry;
            carry = (sum > 9) ? 1 : 0;
            curr->next = new ListNode((carry) ? sum % 10 : sum);
            curr = curr->next;
            l1 = l1->next;
        }
        while (l2) {
            sum = l2->val + carry;
            carry = (sum > 9) ? 1 : 0;
            curr->next = new ListNode((carry) ? sum % 10 : sum);
            curr = curr->next;
            l2 = l2->next;
        }
        if (carry) {
            curr->next = new ListNode(carry);
        }
        return output;
    }

};
```

Chapter 5

7. Reverse Integer

5.1 Description

Given a signed 32-bit integer x, return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.
Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

5.2 Results

Runtime: 0 ms, faster than 100.00% of C++ online submissions for Reverse Integer.
Memory Usage: Memory Usage: 5.7 MB, less than 95.97% of C++ online submissions for Reverse Integer.

5.3 Attempt 1

This one was pretty easy but I'm still pretty sure the runtime glitched out but I'm adding it because it makes me feel nice. Simply get the last digit off of x with mod 10, move x one decimal down , check if multiplying your output by ten will overflow and then just append the last digit to your output.

```
class Solution {
public:
    int reverse(int x) {
        int out = 0;
        while (x) {
            int tmp = x % 10;
            x /= 10;
            if(out > INT_MAX/10 || out == INT_MAX/10 && tmp > 7){
                return 0 ;
            }

            if(out < INT_MIN/10 || out == INT_MIN/10 && tmp < -8){
                return 0 ;
            }
            out = out * 10 + tmp;
        }
        return out;
    }
};
```

Chapter 6

43. Multiply Strings

6.1 Description

Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2, also represented as a string.
Note: You must not use any built-in BigInteger library or convert the inputs to integer directly.

6.2 Results

Runtime: 4 ms, faster than 89.14% of C++ online submissions for Multiply Strings.
Memory Usage: 6.8 MB, less than 42.26% of C++ online submissions for Multiply Strings.

6.3 Attempt 1

```
ass Solution {
public:
    string multiply(string num1, string num2) {
        if (num1 == "0" || num2 == "0") return "0";
        vector<int> num(num1.size() + num2.size(), 0);
        for (int i = num1.size() - 1; i >= 0; --i) {
            for (int j = num2.size() - 1; j >= 0; --j) {
                num[i + j + 1] += (num1[i] - '0') * (num2[j] - '0');
                num[i + j] += num[i + j + 1] / 10;
                num[i + j + 1] %= 10;
            }
        }
        int i = 0;
        while (i < num.size() && num[i] == 0) ++i;
        string res = "";
        while (i < num.size()) res.push_back(num[i++] + '0');
        return res;
    }
};
```

Chapter 7

1584. Min Cost to Connect All Points

7.1 Description

You are given an array points representing integer coordinates of some points on a 2D-plane, where points[i] = [xi, yi]. The cost of connecting two points [xi, yi] and [xj, yj] is the manhattan distance between them: —xi - xj— + —yi - yj—, where —val— denotes the absolute value of val.
Return the minimum cost to make all points connected. All points are connected if there is exactly one simple path between any two points.

7.2 Results

Runtime: 636 ms, faster than 50.33% of C++ online submissions for Min Cost to Connect All Points.
Memory Usage: 150.9 MB, less than 19.58% of C++ online submissions for Min Cost to Connect All Points.

7.3 Attempt 1

Interesting problem, not sure exactly where I went wrong. Maybe I need to use the given vector more to conserve memory.

```
class Solution {
public:
    int minCostConnectPoints(vector<vector<int>>& points) {
        int out = 0;
        bool connect[points.size()];
        memset(connect, 0, points.size());
        connect[0] = true;
        std::vector<int> dis(points.size(), INT_MAX);
        int curr = 0;
        for (int i = 1; i < points.size(); i++) {
            int min = 0;
            for (int j = 1; j < points.size(); j++) {
                if (!connect[j]) {
                    int tmp = man(points[curr], points[j]);
                    if (tmp < dis[j]) dis[j] = tmp;
                    if (dis[j] < dis[min]) min = j;
                }
            }
            connect[min] = true;
            curr = min;
            out += dis[min];
        }
        return out;
    }
    int man(vector<int> A, vector<int> B) {
        return abs(A[0] - B[0]) + abs(A[1] - B[1]);
    }
};
```

Part III

Hard

Chapter 8

4. Median of Two Sorted Arrays

8.1 Description

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

8.2 Results

Runtime: 44 ms, faster than 69.41% of C++ online submissions for Median of Two Sorted Arrays.
Memory Usage: 89.7 MB, less than 42.98% of C++ online submissions for Median of Two Sorted Arrays.

8.3 Attempt 1 - 2022-04-24

Just use the given cpp merge functions. This can be further optimized.

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        auto total = nums1.size() + nums2.size();
        vector<int> nums(total);
        merge(nums1.begin(), nums1.end(), nums2.begin(), nums2.end(), nums.begin());
        if (total % 2 == 0) {
            return (nums[total/2] + nums[total/2 - 1]) / 2.0;
        } else {
            return nums[total/2];
        }
    }
};
```

Chapter 9

8. String to Integer (atoi)

9.1 Description

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).
The algorithm for myAtoi(string s) is as follows:
Read in and ignore any leading whitespace. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored. Convert these digits into an integer (i.e. "123" -¿ 123, "0032" -¿ 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2). If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1. Return the integer as the final result.
Note:
Only the space character ' ' is considered a whitespace character. Do not ignore any characters other than the leading whitespace or the rest of the string after the digits.

9.2 Results

Runtime: 7 ms, faster than 32.43% of C++ online submissions for String to Integer (atoi).
Memory Usage: 8.1 MB, less than 5.67% of C++ online submissions for String to Integer (atoi).

9.3 Attempt 1

Wow this problem sucked. There were so many edge cases. I guess that is why it is the second hardest problem on the site. My time complexity wasn't horrible but my space complexity was. I could probably change the stack from holding integers to pointing at the individual chars in the string. I will assuredly be coming back to this problem.

```
#define ISNUM(C) (C >= '0' && C <= '9')
#define ISLET(C) (C >= 'A' && C <= 'z')

class Solution {
public:
    int myAtoi(string s) {
        int ptr = 0;
        int n = s.length();
        bool negative = false;
        stack<int> integers;
        long int out = 0;
        if(ISLET(s[ptr])) return out;
        while (s[ptr] == ' ') ptr++;
        if (s[ptr] == '-') {
            negative = true;
            ptr++;
        } else if (s[ptr] == '+') {
            ptr++;
        }
        if (!(s[ptr] >= '0' && s[ptr] <= '9')) return 0;
        while (ptr < n) {
            if (!ISNUM(s[ptr]) || s[ptr] == '.' || s[ptr] == ' ') {
                break;
            } else {
                integers.push(s[ptr] - 48);
                ptr++;
            }
        }
        long int place = 1;
        long int tmp;
        while (!integers.empty()) {
            out += (integers.top() * place);
            if (out >= 0x80000000) return (negative) ? 0x80000000 : 0x7fffffff;
            if (place > 2147483647) {
                while (!integers.empty()) {
                    if (integers.top() > 0) return (negative) ? 0x80000000 : 0x7fffffff;
                    integers.pop();
                }
                break;
            }
            place *= 10;
            integers.pop();
        }
        return (negative) ? -out : out;
    }
};
```

Chapter 10

23. Merge K Sorted Lists

10.1 Description

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.
Merge all the linked-lists into one sorted linked-list and return it.

10.2 Results

Attempt 1 Runtime: 30 ms, faster than 63.70% of C++ online submissions for Merge k Sorted Lists.
Memory Usage: 13.6 MB, less than 36.67% of C++ online submissions for Merge k Sorted Lists.
Attempt 2 Runtime: 20 ms, faster than 94.42% of C++ online submissions for Merge k Sorted Lists.
Memory Usage: 13.4 MB, less than 45.00% of C++ online submissions for Merge k Sorted Lists.

10.3 Attempt 1

Had to learn priorty queues for this one, that's something they don't teach you in school. Loads each link into the queue which is then sorted as they are added then one by one links the top node of the queue to the next node in the queue.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
using namespace std;
class Compare {
public:
    bool operator() (ListNode* A, ListNode* B) {
        return (A->val > B->val);
    }
};
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.size() == 1) return *lists.begin();
        priority_queue<ListNode*, vector<ListNode*>, Compare> pq;
        for (auto i = lists.begin(); i < lists.end(); i++) {
            auto curr = *i;
            if (!curr) continue;
            do {
                pq.push(curr);
            } while ((curr = curr->next));
        }
        auto output = (pq.empty()) ? nullptr : pq.top();
        while (!pq.empty()) {
            auto curr = pq.top();
            pq.pop();
            curr->next = (pq.empty()) ? nullptr : pq.top();
        }
        return output;
    }
};
```

10.4 Attempt 2

I had so much fun with this I decided to optimize this. Since we know the linked lists are already sorted we can just pull the leading value from each list and everytime we use a value from that list off the queue we pull another value off that corresponding list. This means we only have at most k comparisons per each insertion.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
typedef pair<ListNode*, int> p;
using namespace std;
class Compare {
public:
    bool operator() (p A, p B) {
        return (A.first->val > B.first->val);
    }
};
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.size() == 1) return *lists.begin();
        priority_queue<p, vector<p>, Compare> pq;
        for (int i = 0; i < lists.size(); i++) {
            if (!lists[i]) continue;
            pq.push({lists[i], i});
            lists[i] = lists[i]->next;
        }
        auto output = (pq.empty()) ? nullptr : pq.top().first;
        while (!pq.empty()) {
            auto curr = pq.top();
            pq.pop();
            if (lists[curr.second]) {
                pq.push({lists[curr.second], curr.second});
                lists[curr.second] = lists[curr.second]->next;
            }
            curr.first->next = (pq.empty()) ? nullptr : pq.top().first;
        }
        return output;
    }
};
```