Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3212

Куприянов Артур Алексеевич

Преподаватель:

Косяков М.С.

Санкт-Петербург

2020

Все исходники и отчет доступны в хабе: /AppLoidx/vtalgo_2020

905. Sort Array By Parity

Для решения этой задачи я воспользовался двумя указателями (справа и слева).

Суть алгоритма довольно проста, необходимо сдвигать левый указатель к правому, до тех пор, пока не встретим нечетное число. Если встретим нечетное число, то меняем местами значения, на которые указывают левый и правый указатели. Затем двигаем правый индекс влево, так как мы уже точно знаем, что там находится нечетное число.

При этом, если правый указатель указывал уже на нечетное число, то оно сместится на позицию левее, так как левый указатель еще не двигался после свапа и указывает на старое значение указателя справа

859 Buddy Strings

```
class Solution {
public:
    bool buddyStrings(string A, string B) {
        if ( A.size() != B.size() ) {
            return false;
        }

        int freq[26] = {0};
        int first = -1;
        int second = -1;
        bool swap = false;
        for (size_t i = 0; i < A.size(); ++i) {
            if ( A[i] != B[i] ) {
                if ( first == -1 ) {
                    first = i;
                } else if ( second == -1 ) {
                    second = i;
                } else {
                    return false;
                }
            }

            ++freq[A[i] - 'a'];
            if ( freq[A[i] - 'a'] > 1 ) {
                swap = true;
            }
        }
        return (second != -1 && (A[first] == B[second] && A[second] == B[first]))
               || (first == -1 && swap);
    }
};
~
~
                                                                    1,16          All
```

Для начала проверяем соразмерность двух строк.

Далее, создадим переменные: массив частот, два указателя и регистр для хранения возможности обязательного свапа.

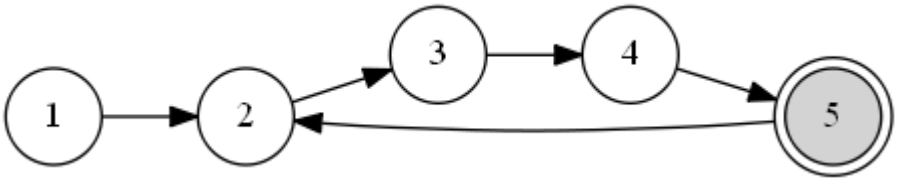
И таким образом, у нас есть условие: у нас есть две буквы, которые мы можем свапнуть и при их свапе символы будут равны, иначе, если различных букв больше двух или одно, то мы не сможем выполнить операцию задачи. Тем не менее, если различных букв вообще нету, то мы проверим условие регистра swap, который принимает значение true, при том условии, что какая-то буква встречается два или более раза.

141. Linked List Cycle

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while(fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if(slow==fast) return true;
        }
        return false;
    }
};
~
~
                                                    1,16          All
```

Здесь можно применить алгоритм Флойда:

Запускаем два указателя, так чтобы у одного скорость обхода была через один, а другой обходил каждый узел. Таким образом, мы можем проверить условие Xi = X2i, тем самым доказать то, что в графе есть цикл.



21. Merge Two Sorted Lists

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(!l1) return l2;
        if(!l2) return l1;

        ListNode* head{nullptr};
        if(l1->val < l2->val) {
            head=l1;
            l1=l1->next;
        } else {
            head=l2;
            l2=l2->next;
        }

        ListNode* prev{head};
        while(l1 && l2) {
            if(l1->val < l2->val)
            {
                prev->next=l1;
                l1=l1->next;
            }
            else
            {
                prev->next=l2;
                l2=l2->next;
            }
            prev=prev->next;
        }

        if(!l1)
            prev->next=l2;
        else
            prev->next=l1;

        return head;
    }
};
                                                                    2,7           All
```

Сначала создадим связанный список, где за начало возьмем минимальный из начал двух входных списков.

Далее пока у одного из списков не закончатся ноды будем сравнивать ноды списков, которые еще не добавили к нашему новому списку и добавлять минимальные. Когда у одного из списков закончатся ноды, просто допишем все оставшиеся ноды другого списка

110. Balanced Binary Tree

```
class Solution {
    int height(TreeNode* root) {
        if (root == nullptr) return 0;
        return max(height(root->left), height(root->right)) + 1;
    }
public:
    bool isBalanced(TreeNode* root) {
        if (root == nullptr) return true;
        return isBalanced(root->left) && isBalanced(root->right) &&
               (abs(height(root->left) - height(root->right)) <= 1);
    }
};
~
~
                                                                  1,16          All
```

Создадим две рекурсивные функции. Один будет вычислять высоту, а другой проходится по всем узлам.

На каждом шаге будем проверять, сбалансированы ли поддеревья ноды, и, если да, то проверим сбалансировано ли поддерево, корнем которой является эта нода

108. Converted Sorted Array to Binary Search Tree

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return buildBST(nums,0,nums.size()-1);
    }
private:
    TreeNode* buildBST(vector<int>& nums,int left,int right) {
        if(left>right) return NULL;

        int mid = left+(right-left)/2;
        TreeNode* current = new TreeNode(nums[mid]);
        current -> left = buildBST(nums,left,mid-1);
        current -> right = buildBST(nums,mid+1,right);
        return current;
    }
};
~
~
                                                    1,15          All
```

Создадим рекурсивную функцию, которая будет строить сначала левую сторону дерева, а потом правую.

Здесь важно выбрать за начало такой узел, который был бы наиболее близким к среднему значению всех элементов входного массива. Таким образом, дерево будет сбалансированным.

Так как входный массив отсортирован, то просто следует выбирать медиану

Тут я бы еще заметил нахождение среднего значения не через (right+left)/2, а так как написано в примере выше, чтобы избежать переполнения значения int. Это очень важно, так как при переполнении мы можем получить не то значение mid. Более того, такую ошибку допустить достаточно просто. Даже в JVM был такой баг, который пофиксили спустя только три года после его находки.

1207. Unique Number of Occurrences

```cpp
class Solution {
public:
    bool uniqueOccurrences(vector<int>& arr) {
        int cnt[2010] = {};
        for(int i = 0; i < arr.size(); i++)
        {
            arr[i] += 1000;     // shift for array indexes

            cnt[arr[i]]++;
        }

        bool occur[2010];
        memset(occur, 0, sizeof(occur));

        for(int i = 0; i < 2010; i++)
        {
            if(cnt[i])
            {
                if(occur[cnt[i]]) return false;
                occur[cnt[i]] = true;
            }
        }

        return true;
    }
};
                                                1,16          All
```

Так как мы знаем ограничения входных данных, то мы можем создать массив, в котором будем хранить количество повторений буквы (частоту)

Затем необходимо будет лишь пройтись по нему и проверить на повторяемость.

997. Find the Town Judge

```
class Solution {
public:
    int findJudge(int N, vector<vector<int>>& trust) {
        if( N == 1 && trust.empty())
            return N;

        int size = trust.size();
        vector<int> rating(N+1, 0);

        for(int i = 0; i<size; i++)
        {
            rating[trust[i][0]]--;
            rating[trust[i][1]]++;
        }

        for(int i = 0 ; i<= N ; i++)
        {
            if( rating[i] == N-1)
                return i;
        }
        return -1;
    }
};
                                                1,15          All
```

Воспользуемся тем свойством, что судья никому не доверяет, а другие все ему доверяют. Следовательно, судье доверяет (N – 1) человек.

Тогда мы можем создать счетчик для каждого человека