Лабораторная 4. Слуцкий Никита, гр. 053501

Задача 1 (5 баллов). Число называется красивым, если оно имеет ровно три различных делителя (включая себя и единицу). Необходимо найти все красивые числа на промежутке от L до R (2 <= L, R <= 1000000000).

Решение:

1. Нетрудно заметить, что свойствами так называемых простых чисел обладают только квадраты простых чисел.
2. Необходимо определить нижнюю и верхнюю границу интервала поиска простых чисел. Они будут равны корню из левой и правой грани исходного интервала соответственно.
3. Необходимо пройтись циклом по полученному интервалу [sqrt(L) .. sqrt(R)] и проверить каждое входящее в него число на простоту.
4. Если число удовлетворяет проверке, выводим это число в квадрате
5. Проверка на простоту занимает в худшем случае sqrt(X) операций, где X –проверяемое число. Таким образом сложность алгоритма составляет: (sqrt(R) – sqrt(L)) * sqrt(X). Если взять X, скажем, для худшего случая (X = R), то сложность можно определить как (sqrt(R) – sqrt(L)) * sqrt(*R*)

2) Задача 2 (5 баллов). Дано натуральное число N (1 <= L <= R <= 1000000000). Необходимо найти минимальное число M, такое, что оно больше чем N и совпадает с числом, полученным его переворачиванием. Например, для числа 1220 ответом будет являться число 1221 (оно выглядит как 1221 если читать с обеих сторон).

Решение:

1. Исходное число разбиваю на цифры и сохраняю их в массиве.
2. Отражаю левую часть массива относительно центра (size() / 2), тем самым затирая правую часть => создаю искусственно палиндром.
3. Создаю число на основании полученных цифр и сравниваю его с входным числом: если оно строго больше, вывожу его в качестве ответа. Иначе: см. пункт 5
4. В массиве с цифрами первый элемент слева от центрального (или центральный, это зависит от (не)чётности длины массива) увеличиваю на 1 (как-то надо учитывать, что цифра может быть равна 9). И провожу снова пункт 2. Создаю число и вывожу в ответ.

5. Сложность алгоритма: длина числа (при разбиении на цифры) + полдлины числа (при отражении) + длина числа (при конвертировании массива цифр в число) + полдлины числа (при отражении) + длина числа (при конвертировании массива цифр в число). В худшем длина составляет 10 символов. Поэтому сложность O(40)

3) Задача 3 (10 баллов). Дана строка из N круглых открывающихся и закрывающихся скобок (N натуральное четное, N <= 1000000). Необходимо сказать, какое минимальное количество символов необходимо добавить, чтобы строка стала правильной скобочной последовательностью.

Решение:

1. В данной задаче основной выбранной структурой для выбранного алгоритма является stack.
2. Совершаю проход по символам строки. В цикле смотрю:
   - если stack пустой: добавляю в него символ
   - иначе: смотрю на текущий верхний элемент нашего stack и на текущий рассматриваемый символ нашей строки.
     - Если верхний элемент stack – открывающая скобка, а текущий рассматриваемый элемент строки – закрывающая: удаляю верхний элемент из stack.
     - Иначе: добавляю рассматриваемую строку в верх stack
3. Пояснение к пункту 2:

```
for (short counter = 0; counter < input.size(); counter++)
{
        if (MAIN.empty())
        {
            MAIN.push(input[counter]);
        }
        else
        {
            if (MAIN.top() == '(' && input[i] == ')')
                MAIN.pop();
            else
                MAIN.push(input[i]);
        }
}
```

4. После прохода по строке смотрю: если stack пустой, то последовательность правильная и ответом является 0. Иначе: см. пункт 5
5. Stack не пустой. Циклом очищаю stack, считая при этом количество закрывающих и открывающих скобок. Ответом на задачу будет разность (по модулю) между количеством открывшихся и закрывшихся скобок.
6. Бонус: Решение можно сделать универсальным для сколько угодно большого количества РАЗЛИЧНЫХ скобок. В начало программы можно

добавить константную строку (или константный массив строк, если вдруг одна скобка может занимать более 2-ух знаков), где на чётных местах стоят открывающие скобки, а на нечётных – закрывающие. Пример такого массива в моей программе:

```
const std::string TEMPLATE = "()[]{}<>";
```

Принцип остаётся прежним, только на этапе пункта 2 необходимо будет искать каждый текущий элемент строки и верхний элемент stack в нашей константной строке, смотреть, соседи ли они и является ли один чётным, а другой – нечётным. Также придётся усложнить пункт 5, чтобы смочь хранить открывающиеся и закрывающиеся скобки разных типов.

7. Сложность базового решения (с (и)) составляет: (N + N) = 2N (если я правильно понимаю, что просмотр верхнего элемента stack и удаление верхнего элемента stack работают за O(1). Итого финальная сложность: N