Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3230

Асташин С.С.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2022

Задача A «Агроном-любитель»

Пояснение к примененному алгоритму:

Поскольку алгоритм содержит очень много ветвлений, описывать каждое бессмысленно – получится пересказ кода. Вместо этого опишу лишь основную идею алгоритма.

По условию задачи, нас не устраивают последовательности вида *N…N*, где количество *N* – больше или равно 3. Предположим, что количество *N* равно 3. Например, рассмотрим последовательность *12333*: в ответ мы возьмём *1233*, то есть от начала последовательности до середины *NNN*. Если у нас после *NNN* есть ещё какая-нибудь *KKK*, например, *12333455*5, то ответ будет *33455*, то есть между серединами *NNN* и *KKK*. Ещё возможен вариант *11123* (ответ *1123*), тут берём от середины *NNN* до конца. Если же n больше, чем 3, то введём понятие двух середин (может не совсем корректное название, но всё же). Объясню на примере (середины выделены): *12**2**222**2**23*. При этом, чтобы избежать лишних проверок в коде, для *NNN* будем считать, что середин тоже две, только они совпадают.

Задача сводится к тому, чтобы разбить нашу последовательность на куски и взять в ответ тот, который имеет наибольшую длину. Если у нас содержится k последовательностей вида *N…N*, то:

- Кусок 1 – от начала последовательности до первой середины $N_1…N_1$;
- Куски со 2 до k – между второй серединой $N_{i-1}…N_{i-1}$ и первой серединой $N_i…N_i$;
- Кусок k+1 – от второй середины $N_k…N_k$ до конца последовательности.

Например, для *123333455567899999990* получим следующие куски: *1233*, *33455*, *5567899*, *990*. В данном случае ответ будет *5567899*.

Задача B «Зоопарк Глеба»

Пояснение к примененному алгоритму:

Для начала создаём `struct element` – элемент зоопарка, у которого есть `value`, `is_trap` и `id` (причём `id` у ловушек и животных – разные). Заводим `map<int,int> trap_map`, в которой будем хранить по ключу (`id` ловушки) `id` животного в ней, и `stack<struct element> line`.

Читаем ввод посимвольно, создаём элемент `tmp` с введённым `value`, определяем, ловушка введена или нет, в зависимости от этого выставляем `id` и `is_trap`. Если наш стек не пустой и буква вершины стека совпадает с буквой в `tmp` (`abs(line.top().value-tmp.value)==32`), то сохраняем в мапу `id` ловушки и животного в ней, выполняем `pop` на стеке; иначе добавляем на него `tmp`. Если в конце наш стек пустой – решение есть.

Задача C «Конфигурационный файл»

Пояснение к примененному алгоритму:

Заводим `map<string,stack<string>*> vars_and_values`, в которой будем хранить по ключу (имени переменной из «конфигурационного файла») её значения на разных уровнях вложенности в виде стека, и `stack<list<string>*> changes`, в котором будем для каждого уровня вложенности хранить список переменных, которые на нём изменились (по сути можно было использовать вместо `list` обычный `stack`, `list` тут только для удобства итерации).

Читаем ввод построчно. Если введена “{“, то увеличиваем уровень вложенности (`changes.push`), если введена “}”, то для стека каждой переменной в `vars_and_values`, которая изменилась на данном уровне, делаем `pop`, потом уменьшаем уровень вложенности (`changes.pop`).

Если введена строка с переменными, то выделяем левую и правую части. В список изменений добавляем `left`. Если `left==right`, сразу выводим значение `left` и идём на следующую итерацию. В ином случае, если в мапе отсутствует ключ `left` (и ключ `right`, если `right` – переменная), то добавляем их туда, а на их стеки кладём 0, выводим значение `right`, если это переменная.

Задача D «Профессор Хаос»

Пояснение к примененному алгоритму:

Достаточно примитивный алгоритм, просто в цикле с `i<=k` изменяем `a` по формуле из условия (`a=a*b-c`), только нужно предварительно проверить несколько условий, после которых нет смысла дальше крутить цикл:

1. `a>=(c+d)/b` – если это выполняется, дальше `a` всегда будет `d`
2. `a<=c/b` – если это выполняется, эксперимент окончен по условию

Кроме того, после изменения надо проверить `b==double(c)/a+1` – на случай, если `a` не изменилось, тогда дальше `a` всегда будет `a`, можно выходить.

Оценка сложности: во всех задачах O(n).