МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине

«АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

Базовые задачи

Выполнил:
Студент группы P3218
Рамеев Тимур
Ильгизович
Преподаватель:
Косяков М. С.
Тараканов Д. С.

Санкт-Петербург, 2024

Содержание

Агроном-любитель

Исходный код

```cpp
#include <iostream>

using namespace std;

int main() {
    int n;
    int previous, preprevious;
    int arr[200005];
    int currentLength = 2, index, maxLength = -1;

    cin >> n;
    for (int i = 0; i < n; i++){
        cin >> arr[i];
    }

    if (n > 2){
        previous = arr[n - 2];
        preprevious = arr[n - 1];
        index = n - 1;
    } else{
        cout << 1 << " " << n;
        return 0;
    }

    for (int i = n - 3; i >= 0; i--){
        if (preprevious == previous and previous == arr[i]){
            currentLength = 2;
        } else {
            currentLength++;
        }
        preprevious = previous;
        previous = arr[i];
        if (maxLength <= currentLength){
            index = i + currentLength - 1;
            maxLength = currentLength;
        }
    }

    cout << index - maxLength + 2 << " " << index + 1;

    return 0;
}
```

Пояснение к примененному алгоритму

1. Получаем на вход в программу количество цветов на грядке (n)
2. Проходимся в цикле длинною n, на каждой итерации записывая текущий цветок в массив (arr[])
3. **В блоке проверяем количество цветов:**
   - Если их 1 или 2, то смело выводим 1-ый и последний индексы, так как их меньше 3. На этом программа заканчивается
   - Если их больше двух, то настраиваем предыдущий, пред предыдущий и текущий индексы для обхода массива цветов, начиная с конца
4. **Непосредственно цикл:**
   - Если замечаем три одинаковых цветка, идущих последовательно, меняем текущую длину на 2 (так как до этого шага мы доходим только тогда, когда n > 2, можно смело утверждать, что минимальная длина, которая может быть это 2)
   - Если три последовательно идущих одинаковых цветка не замечаем, инкрементируем счетчик текущей длины
   - Таким образом продвигаемся по массиву с цветами, заменяя на каждом шаге preprevious на previous и previous на текущий элемент массива.
   - Если находим строку, которая больше или равна чем текущий maxLength (максимальная длина найденной строки), заменяем maxLength на currentLength (текущая длина), а также меняем значение переменной index (которая хранит конечный индекс максимально длинной последовательности). Заменить index очень просто, достаточно прибавить к текущему номеру элемента (i для a[i]) (который указывает на начало новой максимально длинной строки) текущий счетчик длины и отнять единицу (так как длина больше разницы индексов на единицу)
5. **Вывод результата:**
   - Таким образом в переменной index у нас будет конец самой длинной подходящей последовательности цветов, а в переменной maxLength – длинна этой последовательности
   - Чтобы получить начальный индекс последовательности, нужно отнять от конечного индекса длину, затем прибавить единицу, так как длина больше разницы индексов на 1, а также еще раз прибавить единицу, чтобы перевести нумерацию индексов (начинающуюся с 0) на привычную человеку (начинающуюся с единицы), по этой же причине мы прибавляем единицу и к конечному индексу. На этом программа заканчивается

Зоопарк Глеба

Исходный код

```
#include <iostream>
#include <map>
#include <vector>

using namespace std;

int main() {
    string str;

    cin >> str;

    size_t len = str.length();

    map<int, char> strMap;
    map<int, int> animalMap;
    map<int, int> resultMap;

    int animal_key = 1;

    if(str.empty()){
        cout << "Possible";
        return 0;
    }

    for(int i = 0; i < str.size(); i++) {
        strMap[i] = str[i];
        if(islower(str[i])){
            animalMap[i] = animal_key++;
        }
    }


    int previous = 0;
    for(int current_symbol = 1; current_symbol < len; current_symbol++){
        if (strMap.find(previous)->second + 32 == strMap.find(current_symbol)->second){
// Ловушка до животного
            resultMap[previous] = animalMap.find(current_symbol) -> second;
            strMap.erase(previous);
            strMap.erase(current_symbol);
            if(strMap.lower_bound(previous) != strMap.begin()){
                previous = (--strMap.lower_bound(previous))->first;
            } else{
                current_symbol += 1;
                previous = current_symbol;
            }
        } else if (strMap.find(previous)->second == strMap.find(current_symbol)->second
+ 32){ // Ловушка после животного
            resultMap[current_symbol] = animalMap.find(previous) -> second;
            strMap.erase(previous);
            strMap.erase(current_symbol);
            if(strMap.lower_bound(previous) != strMap.begin()){
                previous = (--strMap.lower_bound(previous))->first;
            } else{
                current_symbol += 1;
                previous = current_symbol;
```

```
            }
        } else{
            previous = current_symbol;
        }
    }
    if (resultMap.size() != str.length() / 2){
        cout << "Impossible";
        return 0;
    }
    cout << "Possible\n";
    for(auto & iter : resultMap){
        cout << iter.second << " ";
    }

    return 0;
}
```

Пояснение к примененному алгоритму

1. Записываем в переменную str входные данные программы.
2. Если строка пустая (чисто в теории ведь 0 четный), выводим “Possible” и на этом программа заканчивается.
3. На этом шаге переводим нашу строку в map strMap (ключом является индекс символа, а значением – сам символ), параллельно с этим, если символ является буквой в нижнем регистре, то записываем его в map animalMap (ключом является индекс, а значением – порядковый номер животного)
4. **Непосредственно цикл:**
    - В цикле по strMap мы рассматриваем три случая: когда предыдущий символ – ловушка, а текущий – животное для этой ловушки; когда предыдущий символ – животное, а текущий – ловушка для этого животного; все остальные варианты. Цикл продолжается до тех пор, пока мы не дойдем до конца строки.
    - Если это третий случай, то мы передвигаем счетчик предыдущего символа previous на текущий и идем дальше по циклу.
    - Остальные два случая мы обрабатываем похожим кодом:
    - Добавляем в resultMap (с ключом индекса ловушки, чтобы map отсортировалась в порядке следования ловуше и значением порядкового номера животного, который на эту ловушку попадется). Далее удаляем ловушку и животного из strMap так как они уже нашлись. И вместо того, чтобы начинать цикл сначала, мы меняем указатели previous и current_symbol.
    - Если мы удалили элементы в начале strMap то инкрементируем current_symbol и присваиваем переменной previous значение current_symbol. Затем, итерация заканчивается, с новой итерацией мы увеличим current_symbol на единицу и продолжим обход по strMap.
    - Если же мы удалили элементы не в начале strMap то единственное, что нам нужно сделать, это перевести указатель на предыдущий символ (previous), на пред предыдущий символ в strMap. Делается это легко через итераторы. Затем итеграция заканчивается, и с новой итерацией, когда мы увеличим current_symbol на единицу, мы

по-прежнему будем сравнивать предыдущий символ с текущим. (Просто между ними будет пропасть в виде символов, которые мы уже удалили (если такие есть))

5. **Далее легко проверить количество правильность результата:**

- Если размер resultSize (map, которая содержит номера животных, пойманных в ловушки) не равен, длине строки, деленной на 2, значит в strMap остались животные, которых поймать в ловушки не удалось. Поэтому в таком случае выводим “Impossible”.
- Если размер resultSize равен длине строки, деленной на 2, значит все животные пойманны. Поэтому выводим “Possible” и в цикле выводим resultSize. Так как он уже отсортирован по индексу ловушки, никаких других махинаций нам производить не нужно – для каждой ловушки по порядку выведется соответствующий порядковый номер живоотного.

Конфигурационный файл

Исходный код

```
#include <iostream>
#include <string>
#include <stack>
#include <vector>
#include <unordered_map>

using namespace std;

bool is_number(const string & s)
{
    if(s.empty() || ((!isdigit(s[0])) && (s[0] != '-') && (s[0] != '+'))) return false;
    char * p;
    strtol(s.c_str(), &p, 10);
    return (*p == 0);
}

stack<int> initialize(const int first_elem){
    stack<int> stack;
    stack.push(0);
    stack.push(first_elem);
    return stack;
}

string current_string;
stack<string> changed_on_this_cycle;
string separator = "separator";
unordered_map<string, stack<int>> main_map;

int main(){
    while(cin >> current_string){
        if(current_string == "{"){
            changed_on_this_cycle.push(separator);
            continue;
        }
        if(current_string == "}"){
            while(!changed_on_this_cycle.top().empty()){
                if (changed_on_this_cycle.top() == separator){
                    changed_on_this_cycle.pop();
                    break;
                }
                main_map[changed_on_this_cycle.top()].pop();
                changed_on_this_cycle.pop();
            }
            continue;
        }

        vector<string> current_vector = {current_string.substr(0,
current_string.find('=')),
                                         current_string.substr(current_string.find('=')
+ 1, current_string.length() - 1)};
        if(main_map.find(current_vector[0]) != main_map.end()){
            if(is_number(current_vector[1])) {
                main_map[current_vector[0]].push(stoi(current_vector[1]));
            }
            else if(main_map.find(current_vector[1]) != main_map.end()){
```

```
                main_map[current_vector[0]].push(main_map[current_vector[1]].top());
                cout << main_map[current_vector[1]].top() << endl;
            } else{
                main_map[current_vector[0]].push(0);
                cout << 0 << endl;
            }
        } else{
            if(is_number(current_vector[1])){
                main_map[current_vector[0]] = initialize(stoi(current_vector[1]));
            } else if(main_map.find(current_vector[1]) != main_map.end()){
                main_map[current_vector[0]] =
initialize(main_map[current_vector[1]].top());
                cout << main_map[current_vector[1]].top() << endl;
            } else{
                main_map[current_vector[0]] = initialize(0);
                cout << 0 << endl;
            }
        }
        changed_on_this_cycle.push(current_vector[0]);
    }
    return 0;
}
```

Пояснение к примененному алгоритму

1. **Блок объявления функций:**
   - is_number – функция возвращает true если входную строку можно перевести в число
   - initialize – функция получает на вход числовое значение, затем создает stack содержащий числа, кладет сначала 0(важный момент, потом будет понятно, зачем нужно класть 0), а затем это самое числовое значение, и возвращает этот стек, наполненный нулем и еще каким-то числом
2. **Блок инициализации:**
   - current_string – переменная содержит текущую строку, которая программа получила на текущей итерации
   - main_map – map которая содержит все переменные, присутствующие в программе. Ключом является имя переменной, а значением stack, содержащий числа, этакая история значений переменной, которая периодически добавляется и удаляется
   - changed_on_this_cycle – stack, в который добавляется имя переменной, когда над ней выполняется операция присваивания “=”, а также разделяющая строка
   - Separator – строка, которая добавляется в changed_on_this_cycle когда на вход программе подается строка “{“, служащая для открытия нового блока
3. **Основной цикл.** Цикл длится до тех пор, пока на вход подаются новые строки, и в течение одной итерации происходит обработка строки, которую программа получила на вход. В цикле рассматривается отдельно 4 вида входных строк

4. **Входная строка – “{“**
   - Все что мы делаем в таком случае – это добавляем в общий стек изменений changed_on_this_cycle строку separator, которая, как я уже говорил ранее, показывает начало нового блока
5. **Входная строка – “}”**
   - Такая строка сигнализирует о конце текущего блока. Поэтому, когда мы получаем её на вход, запускается цикл, удаляющий из стека верхний элемент changed_on_this_cycle, пока на вершине стека не будет находиться строка separator. Также во время каждой итерации для каждого элемента, который мы удаляем из цикла changed_on_this_cycle нам нужно найти стек, соответствующий ему в main_map и удалить верхний элемент этого стека. Это происходит потому, что scope этого значения соответствует блоку, который сейчас закрывается, то есть такого значения для данной переменной больше нет. Строка separator также удаляется.
6. **Оставшиеся два случая:**
   - Третий и четвертый случаи относятся к строкам, где происходит присваивание, а отличаются между собой тем, что в одном из них элемент, который стоит слева от знака “=” новый, и тогда перед тем как добавлять значение в его stack его нужно проинициализировать, а в другом, мы просто добавляем новое значение без ненужной нам инициализации
   - Каждый из этих двух случаев, распознает, что находится справа от знака “=”. Если там – неизвестная переменная, то присваивается 0, если число, то присваивается непосредственно число, если же другая, известная нам переменная, то происходит поиск текущего значения этой переменной (то есть верхний элемент стека, соответствующего этой переменной), и уже затем это значение присваивается нашей переменной (под присваиванием я подразумеваю push этого значения на стек элемента)

Профессор Хаос

Исходный код

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main(){
    long days;
    double result, capacity, start, multiplier, destruction;

    cin >> start >> multiplier >> destruction >> capacity >> days;

    if (multiplier == 1){
        result = start - destruction * days;
        cout << max(0.0, result);
        return 0;
    }

    if (multiplier - destruction / start > 1 && (days >= capacity - start || start >= capacity)){
        cout << capacity;
        return 0;
    }
    if (multiplier - destruction / start < 1 && days >= start){
        cout << 0;
        return 0;
    }
    if (multiplier - destruction / start == 1){
        cout <<  min(start, capacity);
        return 0;
    }

    result = start * pow(multiplier, days) - destruction * (pow(multiplier, days)  - 1) / (multiplier - 1);
    cout << min(max(result, 0.0), capacity);
    return 0;
}
```

Пояснение к примененному алгоритму

1. **Входные параметры:**
    - result – переменная для накопления итогового результата, для дальнейшего выведения его на экран
    - capacity – количество бактерий, которые могут уместиться в хранилище
    - start – переменная, содержащая начальное количество бактерий
    - multiplier – множитель, на который умножается текущее количество бактерий ежедневно
    - destruction – количество бактерий, которые ежедневно уничтожаются

- days – количество дней, для которого нужно провести расчеты

2. Структура программы:

- Структура программы реализована в нескольких блоках if, которые стоят перед непосредственно самим вычислением результата. Они позволяют заранее предсказать результат, не занимаясь большими и тяжелыми расчетами, тем самым оптимизируют код. Таким образом если какой-то из них сработает, то предсказуемый результат выведется на экран, и программа завершится, не доходя до основного блока расчета
- 1-й if. Сработает, если multiplier равен единице. Тогда вычисление результата сводится к разности между стартовым значением и произведением destruction и days
- Следующие 3 if. Они связаны с оценкой некого множителя [X] равного [multiplier – destruction / current_number]. Он отражает то, во сколько раз увеличится количество бактерий в этот день. Логично, что, если этот множитель больше единицы, значит количество бактерий на следующем шаге увеличится, по сравнению с текущим количеством. Более того, нетрудно понять, что если текущее количество увеличится первый раз, то в течение всех оставшихся дней оно будет увеличиваться все больше и больше. Это связано с тем, что current_number находится в знаменателе вычитаемого у множителя, а значит при увеличении current_number множитель будет только увеличиваться. То же самое можно сказать в случае, если множитель меньше единицы – он будет уменьшаться всегда. А когда множитель равен единице, результат не будет меняться вообще
- 2-й if. Сработает, когда множитель [X] больше единицы, а также либо количество дней должно быть больше, чем разница между максимальной вместимостью хранилища и стартовым количеством, либо стартовое количество бактерий должно быть больше вместимости хранилища. Со вторым все понятно – количество бактерий больше вместимости + каждый день количество только растет, куда ему ещё увеличиваться? Результатом будет размер хранилища – capacity. С условием разности могут возникнуть вопросы. Дело в том, что количество бактерий – дискретная величина. А значит, когда у нас множитель больше единицы, количество бактерий хотя бы на единицу, но вырастит. А значит минимальное число, на которое увеличится число бактерий – [days * 1]. Ну и соответственно, если это число больше чем разница между размером хранилища и стартовым значением, то при множителе [X] > 1. Результат точно достигнет capacity, поэтому в этом if capacity и будет выводится на экран
- 3-й if. Рассуждения аналогичны предыдущему абзацу. Когда множитель [X] меньше единицы, достаточным условием будет, что start <= days, так как за такое количество дней количество значений точно сможет достигнуть нуля. Поэтому в этом if будет выводиться 0
- 4-й if. Сработает, когда множитель [X] равен единице. В таком случае, вне зависимости от количества дней, количество бактерий не поменяется, поэтому можно смело выводить стартовое количество бактерий.

- Вычисления. Если все предыдущие блоки пропустили выполнение до текущих строк, то оптимизировать вычисление (предсказать) количества бактерий нельзя. Тогда рассчитаем его. Для этого я использую формулу суммы геометрической прогрессии, через которую можно расписать сумму бактерий, которые уничтожаются на каждом шаге. После вычисления выводим результат.