Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №1**

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3233

Анисимов М. Д.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

**Yandex contest**

**Задача А «Агроном-любитель»**

Пояснение к применённому алгоритму:

Задача сводится к тому, чтобы найти наибольшую последовательность элементов, где нет 3 одинаковых элементов. Алгоритм поочерёдно проходится по всем элементам при помощи переменных start и final (Номер первого и последнего цветка на участке без 3 одинаковых элементов). Для сохранения самого лучшего длинного участка используются переменные best\_start и best\_final. Пока программа не найдёт 3 одинаковых элемента, переменная final инкрементируется. Программа проверяет является ли участок start-final длиннее best\_start-best\_final. Если да – start-final становится best\_start-best\_final, если нет, best\_start-best\_final сохраняет своё значение, обход продолжается до конца последовательности. Если же программа нашла 3 одинаковых элемента, то ищется наибольшее расстояние между отрезками start-final и best\_start-best\_final, потом start и final зануляются и происходит новый поиск длинного отрезка.

Код работает за O(n + n) = O(n)

Память: O(n)

Код программы:

#include <iostream>  
using namespace std;  
void solutionFunc(const int size, int flowers[]) {  
 int start = 0;  
 int final = 0;  
 int best\_final = 0;  
 int best\_start = 0;  
  
 for (int i = 0; i < size; i++) {  
 if (size == 1) {  
 best\_start = 1;  
 best\_final = 1;  
 } else if (flowers[i] == flowers[i + 1] && flowers[i] == flowers[i + 2]) {  
 if (i == 0) {  
 start = 1;  
 final = 2;  
 } else if (start != 0 && final != 0) {  
 final = i + 2;  
 }  
 if (best\_final - best\_start >= final - start && best\_start < start) {  
  
 best\_start = min(best\_start, start);  
 best\_final = min(best\_final, final);  
 } else {  
 best\_start = max(best\_start, start);  
 best\_final = max(best\_final, final);  
 }  
 start = 0;  
 final = 0;  
 } else {  
  
 if (start == 0) {  
 start = i + 1;  
 final = start;  
 }  
  
 if (best\_final - best\_start >= final - start && best\_start < start) {  
 best\_start = min(best\_start, start);  
 best\_final = min(best\_final, final);  
 } else {  
 best\_start = max(best\_start, start);  
 best\_final = max(best\_final, final);  
 }  
 final++;  
  
 }  
 }  
 cout << best\_start << " " << best\_final;  
}  
int main() {  
 int size = 0;  
  
 cin >> size;  
 int flowers[size];  
 for (int i = 0; i < size; i++) {  
 cin >> flowers[i];  
 }  
 solutionFunc(size, flowers);  
 return 0;  
}

**Задача B «Зоопарк Глеба»**

Пояснение к применённому алгоритму:

Алгоритм данной задачи заключается в том, что программа проходится циклом по строчке и запоминает порядковые номера ловушек и животных (Для этого в первую половину двусторонней очереди animals\_traps\_counter вносятся номера животных, а во вторую половину номера ловушек). Если же в строчке обнаружена нужная пара «животное – ловушка» (Для этого есть вспомогательный вектор elements, который хранит на вершине стека последний найденный без пары элемент) - программа заносит в result\_map порядковый номер животного, ключом которого является порядковый номер нужной ловушки. Когда программа полностью прошлась по строке, она выводит результат через цикл

Код работает за O(n + n + n/2) = O(2.5n) = O(n)

Память: O(n)

Код программы:

#include <iostream>  
#include <string>  
#include <stack>  
#include <map>  
#include <vector>  
  
using namespace std;  
  
void zooFunc(map<int, int> traps\_animals) {  
 deque<int> animals\_traps\_counter;  
 vector<int> elements;  
 map<int, int> result\_map;  
 int counter\_animals = 0;  
 int counter\_traps = 0;  
  
 for (int i = 0; i < traps\_animals.size(); i++) {  
 if (traps\_animals[i] >= 65 && traps\_animals[i] <= 90) {  
 animals\_traps\_counter.push\_back(counter\_traps);  
 counter\_traps++;  
 } else {  
 counter\_animals++;  
 animals\_traps\_counter.push\_front(counter\_animals);  
 }  
  
 if (!elements.empty()) {  
 if (traps\_animals[i] - elements.back() == 32 || traps\_animals[i] - elements.back() == -32) {  
 result\_map[animals\_traps\_counter.back()] = animals\_traps\_counter.front();  
 elements.pop\_back();  
 animals\_traps\_counter.pop\_back();  
 animals\_traps\_counter.pop\_front();  
 } else  
 elements.push\_back(traps\_animals[i]);  
 } else  
 elements.push\_back(traps\_animals[i]);  
 }  
 if (!elements.empty())  
 cout << "Impossible" << endl;  
 else {  
 cout << "Possible" << "\n";  
 for (int i = 0; i < result\_map.size(); i++)  
 cout << result\_map[i] << " ";  
 }  
  
}  
int main() {  
  
 map<int, int> traps\_animals;  
 string line;  
 cin >> line;  
 const char \*charArray = line.c\_str();  
 for (int i = 0; i < line.size(); i++) {  
 traps\_animals[i] = charArray[i];  
 }  
 if (!traps\_animals.empty())  
 zooFunc(traps\_animals);  
 return 0;  
}

**Задача С «Конфигурационный файл»**

Пояснение к применённому алгоритму:

Алгоритм проходится по текстовому файлу и определяет, что находится в считываемой строке, при помощи регулярных выражений. При каждом открытии новой фигурной скобки в стек all\_values вносится список букв, которые инициализировали внутри данной фигурной скобки, а сами значения букв вносятся в previous\_buffer\_number. Если скобка закрывается, из стека all\_values удаляется верхний список элементов, то есть все инициализации закрытой скобки забываются, и все присвоенные внутри значения удаляются из previous\_buffer\_number. В случаях, когда программа находит строчку «переменная» = «переменная», происходит вывод на экран присвоенного значения

Код работает за: O(n\*m)

Память: O(n)

Код программы:

#include <iostream>  
#include <list>  
#include <fstream>  
#include <map>  
#include <stack>  
#include <regex>  
#include <unordered\_map>  
  
using namespace std;  
  
int main() {  
 smatch match;  
 string buff1;  
 string buff2;  
 unordered\_map<string, stack<int>> previous\_buffer\_number;  
 stack<list<string>> all\_values;  
 list<string> first\_list\_of\_elements;  
 all\_values.push(first\_list\_of\_elements);  
 static const auto regex\_open\_brackets = regex("^[{]");  
 static const auto regex\_letters = regex("([a-zA-Z]{1,})=([a-zA-Z]{1,})");  
 static const auto regex\_numbers = regex("([a-zA-Z]{1,})=([-]?[0-9]{1,})");  
 static const auto regex\_close\_brackets = regex("^[}]");  
 string enterLine;  
 ifstream in;  
 in.open("C:\\example.txt");  
 if (in.is\_open()) {  
 while (getline(in, enterLine)) {  
 if (regex\_match(enterLine, regex\_open\_brackets)) {  
 all\_values.push(first\_list\_of\_elements);  
 continue;  
 } else if (regex\_match(enterLine, regex\_close\_brackets)) {  
 for (const auto &name\_of\_list : all\_values.top()) {  
 previous\_buffer\_number[name\_of\_list].pop();  
 }  
 all\_values.pop();  
 continue;  
 }  
 if (regex\_match(enterLine, regex\_letters)) {  
  
 if (regex\_search(enterLine, match, regex\_letters)) {  
 buff1 = match.str(1);  
 buff2 = match.str(2);  
 if (previous\_buffer\_number[buff1].empty())  
 previous\_buffer\_number[buff1].push(0);  
 if (previous\_buffer\_number[buff2].empty())  
 previous\_buffer\_number[buff2].push(0);  
 previous\_buffer\_number[buff1].push(previous\_buffer\_number[buff2].top());  
 }  
 cout << previous\_buffer\_number[buff1].top() << endl;  
 all\_values.top().push\_front(buff1);  
 continue;  
 } else if (regex\_match(enterLine, match, regex\_numbers)) {  
 buff1 = match.str(1);  
 buff2 = match.str(2);  
 if (previous\_buffer\_number[buff1].empty())  
 previous\_buffer\_number[buff1].push(0);  
 previous\_buffer\_number[buff1].push(stoi(buff2));  
 all\_values.top().push\_front(buff1);  
 }  
 }  
 }  
 in.close();  
  
 return 0;  
}

**Задача D «Профессор Хаос»**

Пояснение к применённому алгоритму:

Данный алгоритм считает количество бактерий после каждого дня и находит минимальное число между получившимся количеством бактерий и максимально возможным числом бактерий в контейнере. После этого он проверяет изменения: если количество бактерий не меняется или оно стало больше или равно допустимому количеству бактерий в контейнере, то можно сразу выводить результат, так как число не меняется и считать дальше нет смысла. В ином случае считаем до тех пор, пока не получим количество бактерий после k-го дня. Также в моём алгоритме идёт проверка на корректность введённых данных с клавиатуры. Если они некорректные, то по стандарту выводится 0

Код работает за: O(n + n) = O(n)

Память: O(n)

Код программы:

#include <iostream>  
#include <map>  
#include <list>  
#include <sstream>  
#include <math.h>  
  
using namespace std;  
void bacteriaFunc(map<int, long long int> bacterias) {  
 long long int total\_bacteria\_number = 0;  
 long long int result = -1;  
  
 for (int i = 0; i < bacterias[4]; i++) {  
  
 if (i == 0)  
 total\_bacteria\_number = bacterias[0] \* bacterias[1] - bacterias[2];  
 else  
 total\_bacteria\_number = total\_bacteria\_number \* bacterias[1] - bacterias[2];  
  
 result = min(total\_bacteria\_number, bacterias[3]);  
  
 if (result <= 0 || bacterias[3] < 1 || bacterias[4] > pow(10, 18)  
 || (bacterias[3] > 1000 && total\_bacteria\_number != bacterias[0])) {  
 result = 0;  
 break;  
 } else if (result == bacterias[3] || result == bacterias[0]) {  
 break;  
 }  
  
 }  
 cout << result;  
}  
int main() {  
 string input;  
 getline(cin, input);  
  
 map<int, long long int> numbersMap;  
 stringstream ss(input);  
 int counter = 0;  
 long long int num = 0;  
 while (ss >> num) {  
  
 numbersMap[counter] = num;  
 counter++;  
 }  
  
 if (num > pow(10, 18) || num == 9223372036854775807) {  
 cout << 0;  
 return 0;  
 }  
 bacteriaFunc(numbersMap);  
 return 0;  
}