Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №1**

по «Алгоритмам и структурам данных»

Базовые задачи / Timus

Выполнил:

Студент группы P32081

Васильченко Роман

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

# Yandex

#### Задача A «Агроном-любитель»

Пояснение к примененному алгоритму:

Задача сводится к тому, что мы должны найти наибольшую подпоследовательность, где нет последовательности из 3 элементов. В самом алгоритме я сохраняю максимальное повторение и если оно равно трем или последний элемент, то проверяем с максимальным и записываем макс длину. После таких процедур уменьшаем curr\_size и делаем дальше.

Сам код работает за O(N+N) = O(2N) = O(N)

Память: O(N)

*//  
// Created by rossilman on 25.02.23.  
//*#include <iostream>  
#include <vector>  
  
using namespace std;  
  
int main() {  
 int n;  
 cin >> n;  
 vector<int> array(n);  
  
 for (int i = 0; i < n; i++)  
 cin >> array[i];  
  
 int max\_size = 0, curr\_size = 0, last = 1, count = 1;  
  
 for (int i = 0; i < n; i++) {  
 curr\_size++;  
  
 if (i >= n - 1) {  
 if (curr\_size > max\_size) {  
 max\_size = curr\_size;  
 last = i + 1;  
 }  
 } else {  
 if (array[i] != array[i + 1])  
 count = 1;  
 else  
 count++;  
  
 if (count == 3) {  
 count = (array[i] == array[i + 1]) ? 2 : 1;  
 if (curr\_size > max\_size) {  
 max\_size = curr\_size;  
 last = i + 1;  
 }  
 curr\_size = 1;  
 }  
 }  
 }  
  
 cout << last + 1 - max\_size << " " << last << endl;  
 return 0;  
}

#### Задача B «Зоопарк Глеба»

Пояснение к примененному алгоритму:

Данная задача была самой сложной из-за ее непонятности в условиях и пришлось перебирать различные определения задачи, чтобы прийти к правильному решению. Данное решение сводится к тому, что мы записываем для себя по очереди животных в порядке 1,2,3,…, а с ловушками проворачиваем операцию i – count, а это означает, что например если ловушка по индексу 3, а животных было 2, то эта ловушка будет второй. По такому смыслу мы можем определить порядок ловушек и что в них будет лежать. Далее просто проходимся по списку и берем либо самый дальний, либо следующий (Как скобочная последовательность)

Код работает за O(N + N/2) = O(N)

Память: O(N + N/2) = O(N)

#include <bits/stdc++.h>  
using namespace std;  
  
struct animal\_trap {  
 int number;  
 char letter;  
};  
  
bool letterEqualAndDifferentCase(char &l1, char &l2) {  
 return abs(l1 - l2) == 32;  
}  
  
int main() {  
 string s;  
 cin >> s;  
  
 int count = 0;  
 list<animal\_trap> animal\_list;  
  
 for(int i = 0; i < s.size(); i++){  
 if(isupper(s[i])) animal\_list.push\_back({i - count, s[i]});  
 else animal\_list.push\_back({++count, s[i]});  
 }  
  
 vector<int> results(s.size() / 2);  
  
 while(!animal\_list.empty()){  
 auto it = ++animal\_list.begin();  
 animal\_trap animal{}, trap{};  
 if(letterEqualAndDifferentCase(animal\_list.front().letter, animal\_list.back().letter)) {  
 if(isupper(animal\_list.front().letter)) {  
 trap = animal\_list.front();  
 animal = animal\_list.back();  
 } else {  
 animal = animal\_list.front();  
 trap = animal\_list.back();  
 }  
 animal\_list.pop\_front();  
 animal\_list.pop\_back();  
  
 results[trap.number] = animal.number;  
 } else if(letterEqualAndDifferentCase(animal\_list.front().letter, (\*it).letter)) {  
 if(isupper(animal\_list.front().letter)) {  
 trap = animal\_list.front();  
 animal = (\*it);  
 } else {  
 animal = animal\_list.front();  
 trap = (\*it);  
 }  
 animal\_list.pop\_front();  
 animal\_list.pop\_front();  
 results[trap.number] = animal.number;  
 } else {  
 break;  
 }  
 }  
  
 if(!animal\_list.empty()) {  
 cout << "Impossible\n";  
 } else {  
 cout << "Possible\n";  
 for(int result : results){  
 cout << result << " ";  
 }  
 cout << "\n";  
 }  
 return 0;  
  
}

#### Задача C «Конфигурационный файл»

Пояснение к примененному алгоритму:

Сложность задачи в том, как записывать изменения, чтобы не случился memory limit. Для этой задачи используется map variables в качестве переменной для хранения значений, где values это вектор, который позволяет хранить прошлые состояния переменной, а history\_of\_variables назван немного неправильно, так как он больше определяет позицию внутри скобок. Сам код работы достаточно простой. Мы берем следующую строку и проверяем, что есть =, а если нету, то либо открываем новую скобку, либо закрываем. В этом случае просто добавляем или удаляем переменные. В случае, если = , то проверяем, что цифра или другая переменная и записываем значения через push\_back, чтобы сохранить прошлые значения.

Код работает за O(N \* (str\_line + N + N)) = O(N^2)

Память: O(N)

*//#include <bits/stdc++.h>*#include <iostream>  
#include <vector>  
#include <string>  
#include <map>  
#define **endl** "\n"  
using namespace std;  
  
bool is\_num(string &str){  
 if((str)[0] != '-' && !isdigit((str)[0])) return false;  
 for(int i = 1; i < (str).size(); ++i){  
 if(!isdigit((str)[i])) return false;  
 }  
 return true;  
}  
  
int main() {  
  
 ios\_base::sync\_with\_stdio(false);  
 cin.tie(nullptr);  
  
 freopen("output.txt","w",**stdout**);  
 freopen("input.txt","r",**stdin**);  
  
 map <string, vector <int> > variables;  
 vector <vector <string> > history\_of\_variables;  
 history\_of\_variables.emplace\_back();  
  
 string str;  
 while(cin >> str) {  
  
 int equation\_symbol = str.find('=');  
  
  
 if (equation\_symbol >= 0) {  
 string write\_to\_variable = str.substr(0, equation\_symbol);  
 string read\_from\_variable = str.substr(equation\_symbol + 1);  
 history\_of\_variables.back().push\_back(write\_to\_variable);  
  
  
 if (is\_num(read\_from\_variable)) {  
 variables[write\_to\_variable].push\_back(stoi(read\_from\_variable));  
 continue;  
 }  
  
  
 if (variables.find(read\_from\_variable) == variables.end() || variables[read\_from\_variable].empty()) {  
 variables[write\_to\_variable].push\_back(0);  
 } else variables[write\_to\_variable].push\_back(variables[read\_from\_variable].back());  
  
 cout << variables[write\_to\_variable].back() << **endl**;  
 continue;  
 }  
  
 if (str[0] == '{') {  
 history\_of\_variables.emplace\_back();  
 }  
  
  
 else {  
 vector <string> last\_vec = history\_of\_variables.back();  
 history\_of\_variables.pop\_back();  
  
 for (equation\_symbol = 0; equation\_symbol < last\_vec.size(); equation\_symbol++) {  
 variables[last\_vec.at(equation\_symbol)].pop\_back();  
 }  
 }  
 }  
  
 return 0;  
}

#### Задача D «Профессор Хаос»

Пояснение к примененному алгоритму:

Задача возможно решается за O(1), но я не нашел решения и поэтому оно такое. В данном случае мы можем заметить, что k слишком большой и компьютер никогда не посчитает это за выделенное время. Поэтому используем некоторые хитрости. Во первых если количество бактерий стало равно 0, то оно никогда не станет больше нуля. Также и с d, так как мы не можем поместить больше d и всегда оно будет = d. Можно сделать вывод, что если количество бактерий уменьшается, то всегда будет уменьшаться и наоборот. Также если оно стало зацикливаться, то всегда будет зациклено. А начальные if были сделаны, так как были тесты, которые не проходили и пришлось просто напросто их расписать вне цикла.

#include <iostream>

#define endl "\n"

using namespace std;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(nullptr);

int a, b, c, d;

long k;

cin >> a >> b >> c >> d >> k;

int loop = a;

if (a \* b - c <= 0) {

cout << 0 << endl;

return 0;

} else if (a \* b - c >= d) {

cout << d << endl;

return 0;

} else if (a \* b - c == a) {

cout << a << endl;

return 0;

}

for (long i = 0; i < k; ++i) {

a = a \* b - c;

if (a <= 0) {

a = 0;

break;

}

if (a >= d) {

a = d;

}

if (a == loop) {

break;

}

loop = a;

}

cout << a << "\n";

}

# Timus

#### Задача №1005 «Куча камней»

Пояснение к примененному алгоритму:

Задача является простым примером динамического программирования, где мы используем рекурсию для выискивания наименьшей разницы. Тут просто записывается minDiff, как наименьшее значение, которое возможное, а если остались камни, то рекурсивно выберем наименьшее.

Работает за O(2^N)

Память занимает O(N)

#include <bits/stdc++.h>

using namespace std;

int n = 20, minDiff = INT\_MAX;

vector<int> w(n);

void calculateSize(int i, int stone\_heap1, int stone\_heap2) {

if (i >= n)

minDiff = (minDiff > abs(stone\_heap1 - stone\_heap2))

? abs(stone\_heap1 - stone\_heap2)

: minDiff;

else {

calculateSize(i + 1, stone\_heap1 + w[i], stone\_heap2);

calculateSize(i + 1, stone\_heap1, stone\_heap2 + w[i]);

}

}

int main() {

cin >> n;

for (int i = 0; i < n; i++) {

cin >> w[i];

}

calculateSize(0, 0, 0);

cout << minDiff;

}

#### Задача № 1296 «Гиперпереход»

Пояснение к примененному алгоритму:

Самая простая задача из всех 6. Тут просто пробуем добавить к нашему число другое и если отрицательное получается, то 0, а потом просто сравниваем с максимумом.

//

// Created by roman on 24.02.2023.

//

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n;

cin >> n;

vector<int> p(n);

for (int i = 0; i < n; i++)

cin >> p[i];

int tmp = 0, res = 0;

for (int i = 0; i < n; i++) {

tmp += p[i];

tmp = (tmp > 0) ? tmp : 0;

res = (tmp > res) ? tmp : res;

}

cout << res << "\n";

return 0;

}