Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3233

Гуменник П. О.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Оглавление

Задача A. Агроном-любитель

Код:

```cpp
#include <iostream>
#include <vector>
using namespace std;

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

    int max_start = 0, max_end = 0, max_len = 0;
    int start = 0;

    for (int i = 2; i < n; i++) {
        if (flowers[i] == flowers[i - 1] && flowers[i] == flowers[i - 2]) {
            if (i - start > max_len) {
                max_len = i - start;
                max_start = start;
                max_end = i;
            }
            start = i - 1;
        }
    }

    if (n - start > max_len) {
        max_len = n - start;
        max_start = start;
        max_end = n;
    }

    cout << (max_start + 1) << " " << (max_end) << endl;

    return 0;
}

```

Пояснение к примененному алгоритму:

1. **Обработка грядки**: Основная логика алгоритма заключается в итерации по грядке начиная с третьего цветка и проверке на наличие трех одинаковых подряд идущих цветков. Если такая последовательность найдена, алгоритм обновляет максимальную длину сегмента без трех подряд идущих одинаковых цветков, если текущий сегмент длиннее предыдущего максимума. Этот подход гарантирует, что все участки грядки будут проверены, а длиннейший подходящий участок будет найден.
2. **Проверка последнего сегмента**: После завершения основного цикла, алгоритм проверяет, не остался ли непроверенным последний участок грядки. Это необходимо, потому что последний участок может не содержать трех одинаковых цветков подряд и при этом быть длиннее ранее найденных участков. Эта проверка гарантирует, что все возможные участки грядки были рассмотрены.
3. **Корректность результата**: Алгоритм всегда поддерживает текущий максимальный участок без трех подряд идущих одинаковых цветков, обновляя его при нахождении более длинного подходящего участка. Поскольку алгоритм проверяет каждый возможный участок и корректно обновляет максимальный участок, он гарантированно найдет и вернет наибольший участок, удовлетворяющий условиям задачи.

Общее время выполнения составляет O(n)

Задача: B.Зоопарк Глеба

Код:

```cpp
#include <iostream>
#include <stack>
#include <vector>
using namespace std;

int main() {
    string s;
    cin >> s;
    int c = 0;
    stack<pair<char, int>> stk; // Стек для пар символ-индекс
    vector<int> solution(s.size(), -1); // Инициализируем вектор для хранения решения

    for (int i = 0; i < s.size(); i++) {
        if (islower(s[i])) { // Если это животное
            c++;
            if (!stk.empty() && toupper(s[i]) == stk.top().first) {
                // Животное встретило свою ловушку
                solution[stk.top().second] = c; // Сохраняем индекс животного (индексы начинаются с 1)
                stk.pop(); // Удаляем ловушку из стека
            } else {
                // Добавляем животное в стек
                stk.push({s[i], c});
            }
        } else { // Если это ловушка
            if (!stk.empty() && tolower(s[i]) == stk.top().first) {
                // Ловушка встретила свое животное
                solution[i] = stk.top().second;
                stk.pop(); // Удаляем животное из стека
            } else {
                // Добавляем ловушку в стек
                stk.push({s[i], i}); // Используем текущий индекс в строке как индекс ловушки в solution
            }
        }
    }

    if (!stk.empty()) { // Если стек не пуст, задача невозможна
        cout << "Impossible\n";
    } else {
        cout << "Possible\n";
        for (int index : solution) {
            if (index != -1) { // Выводим индексы, если они были установлены
                cout << index << " ";
            }
        }
        cout << endl;
    }

    return 0;
}

```

Пояснение к примененному алгоритму:

Гарантия непересекающихся путей: Пути не пересекаются, если каждое животное идет прямо к своей ловушке без обхода других животных или ловушек, которые оно встретило бы на своем пути. В момент, когда животное и ловушка встречаются и удаляются из стека, мы знаем, что между их позициями в исходной строке (круговом порядке) нет других несопоставленных элементов, потому что иначе они бы не были удалены. Это означает, что путь от животного к его ловушке чист, и поскольку каждая пара удаляется из стека таким образом, ни один путь не может пересечься с другим.

Таким образом, если алгоритм завершается с пустым стеком, это значит, что все животные были сопоставлены со своими ловушками без пересечения путей. Если стек не пуст, это значит, что существуют животные и ловушки, которые не могут быть сопоставлены без пересечения путей других пар.

Гарантия корректности индексов: Когда встречается животное (строчная буква), его индекс (начиная с 1) сохраняется в стеке вместе с символом. Если это животное сразу встречает свою ловушку (прописная буква того же символа), индекс животного сохраняется в массиве решений (`solution`) на позиции, соответствующей индексу ловушки в исходной строке.

Если встречается ловушка (прописная буква), и в стеке уже есть соответствующее ей животное (строчная буква того же символа), это означает, что данное животное может быть поймано в эту ловушку без пересечения путей с другими. В этом случае индекс животного (который был сохранен, когда животное было добавлено в стек) записывается в массив решений (`solution`) на позиции этой ловушки.

Массив `solution` инициализируется значениями -1, что означает, что для данной позиции ловушки еще не найдено соответствующее животное. Когда для ловушки находится животное, соответствующее место в массиве `solution` обновляется индексом этого животного. После прохода по всей строке и если задача возможна (стек пуст), массив `solution` будет содержать индексы животных для каждой ловушки, следуя порядку, в котором ловушки были представлены в исходной строке.

Общее время выполнения составляет O(n)

Задача C. Конфигурационный файл

Код:

```cpp
#include <iostream>
#include <map>
#include <stack>
#include <vector>
#include <string>
using namespace std;

int main() {
    map<string, stack<int>> variables; // Словарь для хранения стеков значений переменных
    stack<vector<string>> modifiedVariables; // Стек для хранения измененных переменных в к
    string line;

    while (getline(cin, line)) { // Чтение строки из входного потока
        if (line == "{") { // Начало нового блока
            modifiedVariables.push(vector<string>());
        } else if (line == "}") { // Конец текущего блока
            if (!modifiedVariables.empty()) {
                vector<string> &vars = modifiedVariables.top(); // Получаем список переменн
                for (string &var : vars) {
                    if (!variables[var].empty()) {
                        variables[var].pop();
                    }
                }
                modifiedVariables.pop();
            }
        } else { // Обработка присваиваний
            size_t pos = line.find('=');
            string var1 = line.substr(0, pos);
            string var2 = line.substr(pos + 1);

            if (isdigit(var2[0]) || var2[0] == '-') { // Присваивание числового значения
                int num = stoi(var2);
                if (variables[var1].empty() || num != variables[var1].top()) {
                    variables[var1].push(num);
                    if (!modifiedVariables.empty()) {
                        modifiedVariables.top().push_back(var1);
                    }
                }
            } else { // Присваивание значения другой переменной
                int val = 0;
                if (!variables[var2].empty()) {
                    val = variables[var2].top();
                }
                if (variables[var1].empty() || val != variables[var1].top()) {
                    variables[var1].push(val);
                    if (!modifiedVariables.empty()) {
                        modifiedVariables.top().push_back(var1);
                    }
                }
                cout << val << endl; // Выводим присвоенное значение
            }
        }
    }

    return 0;
}

```

Пояснение к примененному алгоритму:

1. Управление областью видимости переменных:

Парсер использует стек значений для каждой переменной и стек для отслеживания измененных переменных в каждом блоке. Это позволяет корректно управлять областью видимости переменных:

- **Стек значений переменных**: Когда переменной присваивается новое значение, это значение помещается на вершину стека. При выходе из блока значение с вершины стека удаляется, восстанавливая предыдущее значение переменной. Таким образом, изменения значения переменной действительны только внутри текущего блока, что соответствует требованиям задачи.

- **Стек измененных переменных**: Для каждого блока создается список переменных, значения которых изменены в этом блоке. При выходе из блока для каждой такой переменной удаляется текущее значение, возвращая ее к предыдущему состоянию. Это обеспечивает корректность области видимости переменных и соответствие требованиям задачи.

2. Присваивание значений:

- **Присваивание числового значения**: При присваивании переменной числового значения это значение корректно заносится в стек переменной. Поскольку новое значение добавляется на вершину стека, парсер корректно обрабатывает текущее значение переменной в соответствии с контекстом выполнения.
- **Присваивание значения другой переменной**: Парсер проверяет текущее значение второй переменной (из вершины ее стека) и присваивает это значение первой переменной. Это соответствует требованиям задачи, позволяя переменным получать значения друг от друга в соответствии с их текущей областью видимости.

3. Вывод присваиваемых значений:

Когда происходит присваивание значения одной переменной другой, парсер корректно выводит присваиваемое значение. Это позволяет отлаживать присваивания и проверять корректность обработки переменных.

Парсер работает за константное время для каждой строки

Задача D. Профессор Хаос

Код:

```cpp
#include <iostream>
using namespace std;

int main() {
    long long a, b, c, d, k;
    cin >> a >> b >> c >> d >> k;
    long long a_start;

    for (long long day = 1; day <= k; day++) {
        a_start = a;
        // Умножаем количество бактерий
        a = a * b;

        // Используем c бактерий для экспериментов
        if (a <= c) {
            cout << 0 << endl;
            return 0;
        }
        a -= c;

        // Ограничиваем количество бактерий максимум d
        if (a > d) {
            a = d;
        }

        // Если процесс стабилизировался (нет изменений в количестве бактерий), выхо
        if (a == a_start) break;
    }

    cout << a << endl;
    return 0;
}
```

Пояснение к примененному алгоритму:

В целом, алгоритм последовательно выполняет действия согласно условию задачи. В процессе реализации используется следующая оптимизация:

1. **Если значение `a` не изменилось, происходит выход из цикла**: Это правильно, так как если представить каждую итерацию как функцию от состояния системы (количество бактерий `a`, параметры `b`, `c`, `d`), то эта функция детерминированная – на одних и тех же входных данных она дает одинаковые выходные. Это означает, что если значение `a` не изменилось после итерации, оно не изменится и на последующих итерациях, так как параметры `b`, `c` и `d` статичны, а значит дальнейшие итерации цикла не приведут к изменениям в `a`.

Время выполнения в худшем случае:

В худшем случае, если оптимизация не сработает, алгоритм будет выполняться для каждого из `k` дней эксперимента. Так как внутри цикла все операции (умножение, вычитание, сравнение) выполняются за константное время, общее время выполнения алгоритма в худшем случае составит O(k), где `k` — к о л и ч е с т в о