Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3233

Фамилия И.О.

Шикунов Максим Евгеньевич

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача №A «Агроном»

Код:

```cpp
#include <iostream>

using namespace std;

int main() {
    long start {1}, finish, n, x, y, z, maxStart {1}, maxFinish {1};
    cin >> n;
    for (long i {1}; i <= n; i++) {
        cin >> x;
        if ((x == y && x == z) || i == n) {
            if (i != n || (x == y && x == z && i == n)) {
                finish = i - 1;
            } else {
                finish = i;
            }
            if (maxFinish - maxStart < finish - start) {
                maxStart = start;
                maxFinish = finish;
            }
            start = finish;
        }
        z = y;
        y = x;
    }

    cout << maxStart << " " << maxFinish;
    return 0;
}
```

Пояснение к примененному алгоритму:

Так, в задаче нельзя, чтобы на фото было три цветка подряд одинаковых. Значит, надо пройтись по всем цветкам, запоминая предыдущие и сравнивать их при каждой итерации. Ну и если они равны, то начинать счетчик с этого места.
Сложность O(n)

Задача №B «Зоопарк Глеба»

Код:

```
#include <iostream>
#include <cstring>
#include <stack>
#include <map>

using namespace std;

int main() {
    char charArray[100001];
    cin.getline(charArray, 100001);
    long n = strlen(charArray);
    long countAnimal {1}, countTraps {0}, count {0};
    stack<long> animalStack, trapStack;
    stack<char> myStack;
    map<long, long> resMap;
    bool check;

    for (long i {0}; i < n; i ++) {
        check = false;
        if (isupper(charArray[i])) {
            trapStack.push(countTraps);
            countTraps++;
            if (!myStack.empty() && islower(myStack.top()) &&
toupper(myStack.top()) == charArray[i]) {
                check = true;
            }
        } else {
            animalStack.push(countAnimal);
            countAnimal++;
            if (!myStack.empty() && isupper(myStack.top()) &&
tolower(myStack.top()) == charArray[i]) {
                check = true;
            }
        }
        if (check) {
            resMap[trapStack.top()] = animalStack.top();
            trapStack.pop();
            animalStack.pop();
            myStack.pop();
            count++;
        } else {
            myStack.push(charArray[i]);
        }
    }

    if (count == n / 2) {
        cout << "Possible" << endl;
        for (long i {0}; i < n / 2; i++) {
            cout << resMap[i] << " ";
        }
    } else {
        cout << "Impossible";
    }
    return 0;
}
```

Пояснение к примененному алгоритму:

Идея алгоритма заключается в том, что если записывать буквы друг за другом и проверять, а они пара ловушка-зверь и удалять ее, то в итоге если такая комбинация возможна, то у нас они все удалятся из нашего массива, иначе это невозможно.

Сложность: O(n)

Задача №C «Конфигурационный файл»

Код:

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

int main() {
    map<string, long> massive;
    stack<pair<string, long>> myVector;
    stack<long> countVector;
    bool check {false};
    long count {0};
    string input;
    while (getline(cin, input)) {
        if (input == "{") {
            countVector.push(count);
            check = true;
            count = 0;
        } else if (input == "}") {
            long j {0};
            if (count != 0) {
                j = count;
                for (long i{0}; i < j; i++) {
                    pair<string, int> pairElements = myVector.top();
                    myVector.pop();
                    massive[pairElements.first] = pairElements.second;
                }
            }
            count = countVector.top();
            countVector.pop();
        } else {
            size_t signPos = input.find('=');
            string name = input.substr(0, signPos);
            if (check) {
                myVector.push(pair(name, massive[name]));
                count++;
            }
            try {
                long value = stol(input.substr(signPos + 1, input.length()));
                massive[name] = value;
            } catch (const invalid_argument& e) {
                string valueName = input.substr(signPos + 1, input.length());
                long value = massive[valueName];
                massive[name] = value;
                cout << value << endl;
            }
        }
```

```
    }
    return 0;
}
```

Пояснение к примененному алгоритму:

Тут мы все наши переменные храним в map’е, если у нас начинается блок, то мы запоминаем сколько у нас было изменений на данном этапе и во время блока записываем все изменения как пары в вектор, тем самым, при окончании блока мы знаем сколько пар нам над взять из нашего вектора и изменить в мапе.

Сложность: O(n^2)

Задача №D «Профессор Хаос»

Код:

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

using namespace std;

int main() {
    long long a, b, c, d;
    long long k;
    double res;
    cin >> a >> b >> c >> d >> k;
    if (k > 1000) {
        if ((a * b - c) > a) {
            res = static_cast<double>(d);
        } else if ((a * b - c) < a) {
            res = 0;
        } else {
            res = static_cast<double>(a);
        }
    } else {
        if ((a * b - c) > d) {
            a = d;
            k--;
        }
        res = b == 1 ? a - c * k : a + ((a * b - c) - a) * (1 + (b * (pow(b, k -
1) - 1)) / (b - 1));
    }
    if (res >= static_cast<double>(d)) {
        cout << d;
    } else if (res <= 0) {
        cout << 0;
    } else {
        cout << res;
    }
    return 0;
}
```

Пояснение к примененному алгоритму:

Вывел формулу для вычисления результат на какой-либо день, если расписывать каждый последующий результат, то мы будем получать такую вот формулу: a1+ kf) * (1 +), где a1 - стартовое число бактерий, kf - насколько увеличивается

бактерии в первый день и суммы чисел от b^0 до b^(k-1).

Сложность: O(1)

Задача №1296 «Гиперпереход»

Код:

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

int main() {
    long n, k;
    long long sumi {0}, maxi {0};
    cin >> n;
    for (long i {0}; i < n; i++) {
        cin >> k;
        if (k >= 0) {
            sumi += k;
        } else {
            if ((sumi + k) > 0) {
                maxi = max(sumi, maxi);
                sumi += k;
            } else {
                maxi = max(sumi, maxi);
                sumi = 0;
            }
        }
    }
    maxi = max(sumi, maxi);
    cout << maxi;
    return 0;
}
```

Пояснение к примененному алгоритму:

Задача сводится просто к тому, что мы должны найти такую последовательность чисел, сумма которых будет максимальной. Следовательно, когда число больше нуля, то мы просто его берем в учет, а если число меньше нуля, то мы проверяем, а вместе с ним сумма будет положительно или нет. Если положительно, то идем вместе с этим числом, а если нет, то начинаем новую цепочку. Также каждый раз сравнивая число с максимальной перед этим.

Сложность: O(n)

Задача №1401 «Игроки»

Код:

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;
long long count = 1;

void fill(int n, int x, int y, vector<vector<long long>>& board, bool check, int
xLeftCorn, int yLeftCorn) {
    int half = pow(2, n - 1);
    if (n == 1 && !check) {
        board[xLeftCorn + x - 1][yLeftCorn + y - 1] = 0;
        fill(1, x, y, board, true, xLeftCorn, yLeftCorn);
        return;
    }
    if (x <= half && y <= half) {
        board[xLeftCorn + half - 1][yLeftCorn + half] = count;
        board[xLeftCorn + half][yLeftCorn + half - 1] = count;
        board[xLeftCorn + half][yLeftCorn + half] = count;
        count++;
        if (!check) {
            fill(n - 1, x, y, board, false, xLeftCorn, yLeftCorn);
            fill(n - 1, half, 1, board, true, xLeftCorn, yLeftCorn + half);
            fill(n - 1, 1, half, board, true, xLeftCorn + half, yLeftCorn);
            fill(n - 1, 1, 1, board, true, xLeftCorn + half, yLeftCorn + half);
        } else {
            if (n != 1) {
                fill(n - 1, x, y, board, true, xLeftCorn, yLeftCorn);
                fill(n - 1, half, 1, board, true, xLeftCorn, yLeftCorn + half);
                fill(n - 1, 1, half, board, true, xLeftCorn + half, yLeftCorn);
                fill(n - 1, 1, 1, board, true, xLeftCorn + half, yLeftCorn +
half);
            } else {
                return;
            }
        }
    } else if (x <= half && y > half) {
        board[xLeftCorn + half - 1][yLeftCorn + half - 1] = count;
        board[xLeftCorn + half][yLeftCorn + half - 1] = count;
        board[xLeftCorn + half][yLeftCorn + half] = count;
        count++;
        if (!check) {
            fill(n - 1, x, y - half, board, false, xLeftCorn, yLeftCorn + half);
            fill(n - 1, half, half, board, true, xLeftCorn, yLeftCorn);
            fill(n - 1, 1, half, board, true, xLeftCorn + half, yLeftCorn);
            fill(n - 1, 1, 1, board, true, xLeftCorn + half, yLeftCorn + half);
        } else {
            if (n != 1) {
                fill(n - 1, x, y - half, board, true, xLeftCorn, yLeftCorn +
half);
                fill(n - 1, half, half, board, true, xLeftCorn, yLeftCorn);
                fill(n - 1, 1, half, board, true, xLeftCorn + half, yLeftCorn);
                fill(n - 1, 1, 1, board, true, xLeftCorn + half, yLeftCorn +
half);
            } else {
                return;
            }
        }
```

```
    } else if(x > half && y <= half) {
        board[xLeftCorn + half - 1][yLeftCorn + half - 1] = count;
        board[xLeftCorn + half - 1][yLeftCorn + half] = count;
        board[xLeftCorn + half][yLeftCorn + half] = count;
        count++;
        if (!check) {
            fill(n - 1, x - half, y, board, false, xLeftCorn + half, yLeftCorn);
            fill(n - 1, half, half, board, true, xLeftCorn, yLeftCorn);
            fill(n - 1, half, 1, board, true, xLeftCorn, yLeftCorn + half);
            fill(n - 1, 1, 1, board, true, xLeftCorn + half, yLeftCorn + half);
        } else {
            if (n != 1) {
                fill(n - 1, x - half, y, board, true, xLeftCorn + half,
yLeftCorn);
                fill(n - 1, half, half, board, true, xLeftCorn, yLeftCorn);
                fill(n - 1, half, 1, board, true, xLeftCorn, yLeftCorn + half);
                fill(n - 1, 1, 1, board, true, xLeftCorn + half, yLeftCorn +
half);
            } else {
                return;
            }
        }
    } else {
        board[xLeftCorn + half - 1][yLeftCorn + half - 1] = count;
        board[xLeftCorn + half - 1][yLeftCorn + half] = count;
        board[xLeftCorn + half][yLeftCorn + half - 1] = count;
        count++;
        if (!check) {
            fill(n - 1, x - half, y - half, board, false, xLeftCorn + half,
yLeftCorn + half);
            fill(n - 1, half, half, board, true, xLeftCorn, yLeftCorn);
            fill(n - 1, half, 1, board, true,  xLeftCorn, yLeftCorn + half);
            fill(n - 1, 1, half, board, true, xLeftCorn + half, yLeftCorn);
        } else {
            if (n != 1) {
                fill(n - 1, x - half, y - half, board, true, xLeftCorn + half,
yLeftCorn + half);
                fill(n - 1, half, half, board, true, xLeftCorn, yLeftCorn);
                fill(n - 1, half, 1, board, true,  xLeftCorn, yLeftCorn + half);
                fill(n - 1, 1, half, board, true, xLeftCorn + half, yLeftCorn);
            } else {
                return;
            }
        }
    }
}

int main() {
    int n, x, y;
    cin >> n >> x >> y;

    vector<vector<long long>> board(1 << n, vector<long long>(pow(2, n), -1));

    fill(n, x, y, board, false, 0, 0);

    for (int i = 0; i < pow(2, n); i++) {
        for (int j = 0; j < pow(2, n); j++) {
            cout << board[i][j] << " ";
        }
        cout << endl;
    }
```

```
    return 0;
}
```

Пояснение к примененному алгоритму:

В данной задаче мы рекурсивно перебираем все квадраты от 2^n до 2^1 сторонами и заполняем их. Если заполнили числами квадрат со сторонами 2^1 то выходим из метода. Если в метод передается в check false, то мы проверяем в какой из четырех частей квадрата находится выколотая часть, в тех частях, где ее нету мы заполняем углы прилежащие к центру квадрата и вызываем методы этих квадратов с check = true, а где выколотая часть находится, мы ничего не заполняем и вызываем метод с check = false. Это все работает, потому что любой квадрат со сторонами 2 можно закрасить, если в нем уже заполнен один квадрат.

Сложность: O(4^n)