Лабораторная работа № 7 по курсу дискретного анализа: динамическое программирование

Выполнил студент группы М8О-308Б-20 МАИ *Борисов Ян*.

Условие

Кратко описывается задача:

1. Задан прямоугольник с высотой n и шириной m, состоящий из нулей и единиц. Найдите в нём прямоугольник наибольшой площади, состоящий из одних нулей.
2. 2

Метод решения

1. Считываем данные и заполняем матрицу.
2. Затем каждую строку из матрицы представляем как высоты гистограммы в массиве, то есть если в ячейке матрицы 0, то в соответствующей ячейке массива увеличиваем значение на 1, иначе обнуляем значение в ячейке массива.
3. На каждом шаге считаем максимальную площадь гистограммы с помощью отдельной функции, которая реализована следующим образом:
    (a) Читаем высоту i-го прямоугольника. Его абсцисса x равна i. Если $i = n + 1$, то его высота равна нулю.
    (b) В конце выталкиваем все прямоугольники из стека кроме первого с высотой -1 и пересчитываем искомую площадь по формуле
    $\mathbf{area = hPrev * (i - x)}$;

Если из функции была возвращена площадь большая, чем текущая, обновляем значение площади.

Описание программы

В моей программе один файл main.cpp:

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

using namespace std;

struct Node{
    int x;
    int Height;
    Node(int x, int Height) : x(x), Height(Height) {};
};

int findMaxArea(const int array[], int n) {
    stack<Node> stack;
    int res = 0;
    stack.push(Node(0,-1));
```

```
    int h;
    int hPrev;
    int area;
    for(int i = 1; i <= n + 1; ++i) {
        if(i <= n) {
            h = array[i - 1];
        }
        else {
            h = 0;
        }
        int x = i;
        while(h <= stack.top().Height) {
            x = stack.top().x;
            hPrev = stack.top().Height;
            stack.pop();
            area = 1LL * hPrev * (i - x);
            if(area > res) {
                res = area;
            }
        }
        stack.push(Node(x, h));
    }
    return res;
}

int main() {
    int n = 0, m = 0;
    std::cin >> n >> m;
    int matrix[n][m];
    std::fill(matrix[0], matrix[0] + m * n, 0);
    std::string inputLine;
    for(int i = 0; i < n; ++i) {
        std::cin >> inputLine;
        for(int j = 0; j < m; ++j) {
            matrix[i][j] = inputLine[j] - 48;
        }
    }
    int array[m];
    int area;
    int newArea;
    for (int j = 0; j < m; ++j) {
        if (matrix[0][j] == 0) {
            array[j] = 1;
        } else {
            array[j] = 0;
        }
    }
    area = findMaxArea(array, m);
    for(int i = 1; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if(matrix[i][j] == 0) {
                ++array[j];
            }
            else {
```

```
                array[j] = 0;
            }
        }
        newArea = findMaxArea(array, m);
        if(area < newArea) {
            area = newArea;
        }
    }
    std::cout << area;
    return 0;
}
```

Дневник отладки

Достаточно долго думал над идеей решения этой задачи, так как первоначально придуманный мной алгоритм выдавал WA.

Тест производительности

Померить время работы кода лабораторной и теста производительности на разных объёмах входных данных. Сравнить результаты. Проверить, что рост времени работы приувеличении объема входных данных согласуется с заявленной сложностью.

Сложность по времени: в алгоритме имеется заполнение и обход матрицы, которые по сложности O $(n * m)$, сам алгоритм поиска площади гистограмм O (n), Итоговая сложность O $(n * m)$.

1) 100 × 100 => 0.01c

2) 100 × 1000 => 0.011c

3) 1000 × 1000 => 0.136c

Из проведенных тестов, можно заметить, что при увеличении n и m в 10 раз, время работы программы тоже увеличивается примерно в 10 раз. Моя асимптотическая оценка подтвердилась.

Недочёты

Не были обнаружены.

Выводы

При выполнении данной лабораторной работы я ознакомился с динамическим программированием и смог решить задачу поиска наибольшего прямоугольника из нулей в матрице.