Лабораторная работа № 7 по курсу дискрeтного анализа: динамическое программирование

Выполнил студент группы М8О-308Б-20 МАИ *Зубко Дмитрий*.

# Условие

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

# Метод решения

# Считываем матрицу, заменяем 1 на 0, а 1 на 0 с помощью операции (el + 1) % 2, чтобы было удобнее считать в дальнейшем. Представляем нашу матрицу в виде гистограмм. Для i = 0 гистограмма равно исходной строке в матрице. Для 1 <= i <= n – 1 считаем гистограмму так: если matrix[i][j] = 1, то matrix[i][j] = matrix[i][j] + matrix[i – 1][j], то есть увеличиваем высоту. Затем считаем максимальную площадь для i-ой гистограммы. Создаем стек с парой (i, )=(0, -1), где i – абцисса, – высота. Вносим эту пару, чтобы нулевой прямоугольник никогда не был извлечен из стека, а обработка дополнительного прямоугольника с высотой 0 в конце вытолкнет из стека все имеющиеся прямоугольники кроме нулевого. Проходим по столбцам (прямоугольникам) в гистограмме. Если высота столбца больше, чем высота последнего столбца в стеке, то добавляем столбец в стек. Если нет, то вынимаем из стека столбцы, пока их высота больше или равна высоте текущего столбца, и вычисляем площадь прямоугольника, по формуле: h\_prev \* (i – x), h\_prev и x – высота и абцисса вынутого столбца. Добавляем в стек столбец с высотой histogram[i – 1], либо если мы вынимали что-то из стека – значение (j), где j – абцисса последнего вынутого столбца.

# Описание программы

В моей программе один файл main.cpp. Структура Node описывает прямоугольник в гистограмме, x – абцисса, height – высота. Функция find\_max\_hist считает максимальную площадь прямоугольника в гистограмме.

#include <iostream>

#include <algorithm>

#include <vector>

#include <stack>

#include <string>

struct Node {

int x;

int height;

Node(int x, int height) : x(x), height(height) {};

};

int find\_max\_hist(const std::vector<int>& hist) {

std::stack<Node> s;

s.push(Node(0, -1));

int max\_area = 0;

int area;

int n = int(hist.size());

for (int i = 1; i <= n + 1; ++i) {

int h = i <= n ? hist[i - 1] : 0;

int x = i;

int h\_prev;

while (h <= s.top().height) {

x = s.top().x;

h\_prev = s.top().height;

s.pop();

area = h\_prev \* (i - x);

max\_area = std::max(area, max\_area);

}

s.push(Node(x, h));

}

return max\_area;

}

int main() {

int n, m;

std::cin >> n >> m;

std::vector<std::vector<int>> matrix(n, std::vector<int>(m));

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

std::string line;

std::cin >> line;

for (int j = 0; j < m; ++j) {

int value = line[j] - 48;

matrix[i][j] = (value + 1) % 2;

}

}

int max\_area = find\_max\_hist(matrix[0]);

for (int i = 1; i < n; ++i) {

for (int j = 0; j < m; ++j) {

if (matrix[i][j] != 0) {

matrix[i][j] += matrix[i - 1][j];

}

}

max\_area = std::max(max\_area, find\_max\_hist(matrix[i]));

}

std::cout << max\_area << std::endl;

}

# Дневник отладки

В первой посылке я неправильно обрабатывал ввод. Эту проблему я исправил тем, что начал считывать строку и проходить по ней, вместо обычного ввода чисел.

# Тест производительности

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

Время работы алгоритма для чисел N, M:

1) N =100, M = 100 => 0m0.011s

2) N = 100, M = 1000 => 0m0.008s

3) N = 1000, M = 10000 => 0m0.038s

4) N = 10000, M = 10000 => 0m0.287s

# Недочёты

Программа работает корректно только для правильных входных данных.

# Выводы

# Особенность данной лабораторной работы в том, что она не на какой-то алгоритм, а на подход к программированию. Раньше я знал метод динамического программирования, но благодаря данной лабораторной работе улучшил свои знания. Решил задачу поиска прямоугольника из 0 максимальной площади со сложность О(m \* n), где m и n – размер матрицы.