Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Дискретный анализ" №7

Студент: Кострюков Е.С.

Преподаватель: Макаров Н.К.

Группа: М8О-307Б-22

Дата:

Оценка:

Подпись:

Москва 2024 г.

Оглавление

Цель работы

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

Постановка задачи

Формат ввода

В первой строке заданы *1 <= n <= 500* и *1 <= m <= 500*. В последующих *n* строках записаны по *m* символов 0 или 1 - элементы прямоугольника.

Формат вывода

Необходимо вывести одно число – максимальную площадь прямоугольника из одних нулей.

Общий алгоритм решения

1. Инициализация:

- Вводится двумерная матрица *inputRectangle* размером *n x m*, где каждая ячейка матрицы может быть либо 0, либо 1.
- Создаётся вектор *heights* для хранения "высоты" каждого столбца на текущем уровне (строке).

2. Обработка строк матрицы:

- Для каждой строки матрицы обновляются значения вектора *heights*. Если в текущей ячейке матрицы стоит 0, то соответствующее значение в *heights* увеличивается на 1 (это значит, что прямоугольник может быть продолжен вниз), иначе оно обнуляется.

3. Поиск максимальной площади для текущей строки:

- Для каждого обновлённого вектора *heights* применяется алгоритм нахождения максимальной области в гистограмме с использованием стека: мы входим в цикл *while,* когда текущая высота меньше или равна высоте на вершине стека, чтобы обработать все прямоугольники, которые "закрываются" при появлении этой более низкой высоты, т.к. когда высота на следующем индексе становится меньше текущей, это значит, что закончился диапазон высоты для тех столбцов, которые выше текущего. Чтобы найти максимальные прямоугольники для каждой из этих высот, нам нужно их обработать.

Порядок действий:

1. Удаляем индексы из стека, пока высота по индексу *j* меньше или равна высоте на вершине стека. Это позволяет определить максимальную ширину для текущей высоты из стека.
2. Вычисляем площадь прямоугольника для каждого из этих элементов, пока они остаются в стеке, так как их высота больше текущей. Каждый прямоугольник может «расшириться» вправо до текущего индекса *j*.
3. Определяем ширину для высоты из стека:

- Ширина для текущей высоты равна расстоянию от текущего индекса *j* до индекса, который находится на вершине стека после удаления текущего элемента.
- Это нужно, так как при "падении" высоты мы можем максимально расширить прямоугольник, законченный на этом индексе.

4. Обновление максимальной площади:
- Для каждого шага в поиске максимальной площади обновляется переменная *maxArea*, хранящая максимальную найденную площадь.

5. Вывод результата:
- После того, как все строки матрицы обработаны, возвращается максимальная площадь прямоугольника, которую можно найти в данной матрице.

Основные шаги алгоритма:

1. Для каждой строки обновляем высоты столбцов.
2. Для обновлённых высот находим максимальный прямоугольник с использованием стека.
3. Повторяем для всех строк и выводим максимальную площадь.

Время работы алгоритма составляет $O(n * m)$, где n — количество строк, а m — количество столбцов в матрице.

Реализация

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

using namespace std;

int findMaxArea(vector<vector<int>>& inputRectangle) {
    int n = inputRectangle.size();
    int m = inputRectangle[0].size();

    vector<int> heights(m, 0);
    int maxArea = 0;

    for (int i = 0; i < n; ++i) {
        // Обновляем высоты для текущей строки
        for (int j = 0; j < m; ++j) {
            heights[j] = (inputRectangle[i][j] == 0) ? heights[j] + 1 : 0;
        }

        // Находим максимальную площадь прямоугольника для текущей строки
        stack<int> indexStack;
        for (int j = 0; j <= m; ++j) {
            while (!indexStack.empty() && (j == m || heights[indexStack.top()] >= heights[j])) {
                int currentHeight = heights[indexStack.top()];
                indexStack.pop();
                int width = indexStack.empty() ? j : j - indexStack.top() - 1;
                maxArea = max(maxArea, currentHeight * width);
            }
            indexStack.push(j);
```

```
            }
        }
        return maxArea;
    }

    int main() {
        ios::sync_with_stdio(false);
        cin.tie(0);

        int n, m;
        cin >> n >> m;
        vector<vector<int>> inputRectangle(n, vector<int>(m));

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                char c;
                cin >> c;
                inputRectangle[i][j] = (c == '0') ? 0 : 1;
            }
        }
        cout << findMaxArea(inputRectangle);
    }
```

Пример работы

Input	Output
4 5 01011 10001 01000 11011	4
3 4 0110 0010 1110	3
3 3 000 000 111	6

Вывод

В результате выполнения данной лабораторной работы я лучше понял принцип работы динамического программирования и его применение в задачах поиска оптимальных решений.

Главное, что я усвоил – это метод построения оптимальных решений с использованием накопленных данных. В данной задаче я научился строить массив высот и обновлять его по каждой строке матрицы, что позволяет эффективно находить максимальную площадь прямоугольника, состоящего из одних нулей. Этот подход демонстрирует, как динамическое программирование может свести задачу поиска наибольшей площади к последовательному накоплению информации, упрощая вычисления.

Также я освоил работу со стеком для решения задачи максимальной площади в гистограмме, что помогает найти наибольшую площадь прямоугольника за линейное время. Я понял, как с помощью стека можно хранить промежуточные состояния (индексы предыдущих высот), чтобы корректно вычислять ширину прямоугольника при каждой новой высоте.