Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа по курсу "Дискретный анализ" №7

Студент: Клименко В. М.

Преподаватель: Макаров Н. К.

Группа: М8О-203Б-22

Дата: _____

Оценка: _____

Подпись: _____

Содержание

Постановка задачи

Задан прямоугольник с высотой n и шириной m, состоящий из нулей и единиц. Найдите в нем прямоугольник наибольшей площади, состоящий из одних нулей.

Формат ввода

В первой строке заданы $1 \leq n \leq 500$ и $1 \leq m \leq 500$. 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.

Алгоритм решения

Подготовим данные при помощи динамического программирования и решим другую задачу. Для каждой строки рассчитаем массив – количество подряд идущих нулей во всех столбцах, решим задачу поиска наибольшего прямоугольника в гистограмме. Из всех площадей возьмем максимальную.

Площадь наибольшего прямоугольника в гистограмме

1. Заведем стек с индексами высот, в котором верхний элемент – индекс наибольшей высоты на данный момент
2. Пройдемся по всем высотам. Пока в стеке сверху лежит индекс высоты, большей чем настоящей – посчитаем ширину прямоугольника, которая равна разнице индексов настоящей высоты и второму индексу сверху стека минус 1 (т.к. прямоугольник нужной высоты начинается со следующего индекса)
3. Максимальное произведение высоты на ширину – искомая площадь

Временная сложность поиска наибольшего прямоугольника в гистограмме – $O(m)$, пространственная сложность – $O(m)$.

Тогда временная сложность алгоритма исходной задачи – $O(nm)$, пространственная – $O(nm)$.

Исходный код

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



uint32_t maxRectangleInHistogram(const std::vector<uint16_t> &histogram) {
	std::stack<uint16_t> biggestHeightIndecies;

	uint32_t maxSquare = 0;
	const uint16_t width = histogram.size();

	for (uint16_t currentHeightIndex = 0; currentHeightIndex <= width;
++currentHeightIndex) {
	uint16_t currentHeight;
	if (currentHeightIndex != width) {
		currentHeight = histogram[currentHeightIndex];
	} else { // last bar height is always zero
		currentHeight = 0;
	}

	// current height is smaller, so calculate the max square and remove all
heights that are bigger than the current height
	while (!biggestHeightIndecies.empty() &&
histogram[biggestHeightIndecies.top()] >= currentHeight) {
		const uint16_t biggestHeight =
histogram[biggestHeightIndecies.top()];
		biggestHeightIndecies.pop();

		uint16_t currentWidth;
		if (biggestHeightIndecies.empty()) { // this index is the biggest
width so far
		currentWidth = currentHeightIndex;
		} else { // take the last height index that was smaller than the
biggest height + 1
		currentWidth = currentHeightIndex - (biggestHeightIndecies.top() +
1);
		}
```

```
            maxSquare = std::max(maxSquare, (uint32_t) biggestHeight *
(uint32_t) currentWidth);
      }

      biggestHeightIndecies.push(currentHeightIndex);
      }

      return maxSquare;
}


uint32_t maxZeroRectangleSquare(const std::vector<std::vector<bool>> &grid) {
      uint16_t width = grid[0].size(), height = grid.size();
      // histogram is a row of heights of current rectangle
      std::vector<uint16_t> histogram(width, 0);
      uint32_t maxSquare = 0;

      for (uint16_t i = 0; i < height; ++i) {
      for (uint16_t j = 0; j < width; ++j) {
            if (grid[i][j] == false) { // accumulate the height
            histogram[j] += 1;
            } else { // reset the height
            histogram[j] = 0;
            }
      }
      maxSquare = std::max(maxSquare, maxRectangleInHistogram(histogram));
      }

      return maxSquare;
}



int main() {
      uint16_t width, height;
      std::cin >> height >> width;

      std::vector<std::vector<bool>> grid(height, std::vector<bool>(width));

      for (uint16_t i = 0; i < height; ++i) {
      for (uint16_t j = 0; j < width; ++j) {
            char currentElement;
            std::cin >> currentElement;
            grid[i][j] = currentElement - '0';
```

```
        }
        }

        std::cout << maxZeroRectangleSquare(grid) << '\n';

        return 0;
}
```

Тесты

Входные данные:

```
5 5
01010
10101
01010
10101
01010
```

Выходные данные:

```
1
```

Входные данные:

```
2 8
01011110
00000000
```

Выходные данные:

```
8
```

Вывод

В ходе лабораторной работы я научился решать задачи при помощи подготовки данных используя динамическое программирование.