Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

Студент: И. О. Ильин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №7

Задача:

Вариант №2

При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом; оценить время выполнения алгоритма и объем затрачиваемой оперативной памяти. Перед выполнением задания необходимо обосновать применимость метода динамического программирования.

Разработать программу на языке C или C++, реализующую построенный алгоритм. Формат входных и выходных данных описан в варианте задания:

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

Формат входных данных

В первой строке заданы 1 n 500 и 1 m 500. В следующих n строках записаны по m символов 0 или 1 — элементы прямоугольника.

Формат результата

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

1 Описание

Основная идея динамического программирования заключается в том, что для для решения большой сложной задачи мы разбиваем ее на более простые подзадачи, которые умеем решать [?].

Для решения поставленной задачи воспользуемся следующим алгоритмом.

Будем производить поэлементный обход матрицы, для каждого элемента которой будем искать площадь такого прямоугольника, что если элемент является 0, то он лежит внутри верхней стороны прямоугольника (или правой, в зависимости от системы координат). Для этого необходимо хранить количество подряд идущих 0 по столбцу (строке) до текущего элемента, а также индексы левой и правой границы, которые являются крайними точками этой стороны прямоугольника. Выбрав максимум из данных площадей, мы найдём ответ на поставленную задачу.

Оценим сложность данного алгоритма.

Пусть n — количество строк, m — количество столбцов. В цикле по строкам мы проходим каждый элемент в текущем столбце два раза (для нахождения правой и левой границ), тогда сложность составит $O(n \cdot 2 \cdot m) = O(n \cdot m)$. При этом, для каждого элемента матрицы мы храним 4 числа.

2 Исходный код

Объявим заголовочный файл *dp.hpp* с необходимыми конструкторами нашего класса и полями. Также объявим метод вычисления максимальной площади прямоугольника *findMaxRectangleArea*, а также метод для получения матрицы значений максимальных площадей для каждого элемента матрицы *getAreas*.

```
#pragma once
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>

namespace NMyDP {

class TDPMatrix {
    public:
        TDPMatrix(size_t rows, size_t cols);

        int findMaxRectangleArea();
        std::vector<std::vector<int> > getAreas();

        friend std::istream& operator>>(std::istream &is, TDPMatrix & rhs);

    private:

        struct dpCell {
            int maxArea; // .
            int height; //
            size_t rightBound; //   ,   .
            size_t leftBound;
        };

        std::vector<std::vector<char> > matrix;
        std::vector<std::vector<dpCell> > dpMatrix;

        int fillDpMatrix();
};

} // namespace NMyDP
```

В dp.cpp напишем реализации методов.

```
#include "dp.hpp"

#ifdef DEBUG
#include <iomanip>
#endif

namespace NMyDP {
```

```
8
9   TDPMatrix::TDPMatrix(size_t rows, size_t cols)
10      : matrix(rows, std::vector<char>(cols)),
11      dpMatrix(rows, std::vector<dpCell>(cols))
12  {}
13
14  std::istream& operator>>(std::istream &is, TDPMatrix & rhs) {
15      size_t rowsCount = rhs.matrix.size();
16      size_t colsCount = rhs.matrix[0].size();
17      for (size_t i = 0; i < rowsCount; ++i) {
18          std::string row;
19          is >> row;
20          for (size_t j = 0; j < colsCount; ++j) {
21              rhs.matrix[i][j] = row[j];
22          }
23      }
24      return is;
25  }
26
27  int TDPMatrix::findMaxRectangleArea() {
28      return fillDpMatrix();
29  }
30
31  std::vector<std::vector<int> > TDPMatrix::getAreas() {
32      fillDpMatrix();
33      size_t rows = matrix.size();
34      size_t cols = matrix[0].size();
35      std::vector<std::vector<int> > result (
36          rows,
37          std::vector<int>(cols)
38      );
39
40      for (size_t i = 0; i < rows; ++i) {
41          for (size_t j = 0; j < cols; ++j) {
42              result[i][j] = dpMatrix[i][j].maxArea;
43          }
44      }
45
46      return result;
47  }
48
49  int TDPMatrix::fillDpMatrix() {
50      int result = 0;
51      size_t rows = matrix.size();
52      size_t cols = matrix[0].size();
53
54      for (int i = 0; i < rows; ++i) {
55
56          //   .          .
```

```
57            size_t currLeftBound = 0;
58            for (int j = 0; j < cols; ++j) {
59                if (i > 0) {
60                    dpMatrix[i][j] = dpMatrix[i - 1][j];
61                    dpMatrix[i][j].maxArea = -1;
62                } else {
63                    dpMatrix[i][j].height = 0;
64                    dpMatrix[i][j].leftBound = 0;
65                    dpMatrix[i][j].rightBound = cols - 1;
66                    dpMatrix[i][j].maxArea = 0;
67                }
68
69                if (matrix[i][j] == '0') {
70                    ++dpMatrix[i][j].height;
71                    dpMatrix[i][j].leftBound = std::max(dpMatrix[i][j].leftBound,
                          currLeftBound);
72                } else {
73                    dpMatrix[i][j].height = 0;
74                    dpMatrix[i][j].leftBound = 0;
75                    currLeftBound = j + 1;
76                }
77            }
78
79            //
80            size_t currRightBound = cols - 1;
81            for (int j = cols - 1; j >= 0; --j) {
82                if (matrix[i][j] == '0') {
83                    dpMatrix[i][j].rightBound = std::min(dpMatrix[i][j].rightBound ,
                          currRightBound);
84                } else {
85                    dpMatrix[i][j].rightBound = cols - 1;
86                    currRightBound = j - 1;
87                }
88            }
89
90            //
91            for (int j = 0; j < cols; ++j) {
92                dpCell& curr = dpMatrix[i][j];
93                curr.maxArea = curr.height * (curr.rightBound - curr.leftBound + 1);
94                result = std::max(result, curr.maxArea);
95            }
96        }
97
98    #ifdef DEBUG
99        std::cout << "Areas:" << std::endl;
100       for (int i = 0; i < rows; ++i) {
101           for (int j = 0; j < cols; ++j) {
102               std::cout << "[" << i << "][" << j << "] " <<std::setw(3) << dpMatrix[i
                      ][j].maxArea << "\t";
```

```
            }
            std::cout << std::endl;
        }

        for (auto row : dpMatrix) {
            for (auto el : row) {
                std::cout << std::setw(2) << el.height << " (";
                std::cout << std::setw(2) << el.leftBound << ", ";
                std::cout << std::setw(2) << el.rightBound << ") ";
                std::cout << std::setw(2) << el.maxArea << "\t";
            }
            std::cout << std::endl << std::endl;
        }
    #endif

        return result;
    }

} //namespace NMyDP
```

В main.cpp напишем считывание матрицы и вывод результата.

```
#include <iostream>
#include <vector>
#include "dp.hpp"

int main() {
    std::ios_base::sync_with_stdio(false);

    size_t rows, cols;
    std::cin >> rows >> cols;
    NMyDP::TDPMatrix matrix(rows, cols);
    std::cin >> matrix;

    std::cout << matrix.findMaxRectangleArea() << std::endl;

    return 0;
}
```

3 Консоль

```
MacBook-Pro:da_lab_07 mr-ilin$ ./wrapper.sh
[2021-05-12 17:54:43] [INFO] Compiling...
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -O3 -c main.cpp -o main.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -O3 -c dp.cpp -o dp.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -O3 main.o dp.o -o solution
[2021-05-12 17:54:46] [INFO] Making unittest...
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -O3 -c unit_tests.cpp -o
unit_tests.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -O3 unit_tests.o dp.o -o
unit_test -lgtest -lgtest_main
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from TDPMatrixSuite
[ RUN      ] TDPMatrixSuite.simpleResultTest
[       OK ] TDPMatrixSuite.simpleResultTest (107 ms)
[ RUN      ] TDPMatrixSuite.maxAreasTest
[       OK ] TDPMatrixSuite.maxAreasTest (0 ms)
[ RUN      ] TDPMatrixSuite.maxAreasMatrixTest
[       OK ] TDPMatrixSuite.maxAreasMatrixTest (0 ms)
[----------] 3 tests from TDPMatrixSuite (107 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (107 ms total)
[  PASSED  ] 3 tests.
[2021-05-12 17:54:48] [INFO] Executing tests/t01.t...
OK
[2021-05-12 17:54:49] [INFO] Executing tests/t02.t...
OK
[2021-05-12 17:54:49] [INFO] Executing tests/t03.t...
OK
[2021-05-12 17:54:49] [INFO] Executing tests/t06.t...
OK
[2021-05-12 17:54:49] [INFO] Executing tests/t11.t...
OK
[2021-05-12 17:54:49] [INFO] No failed tests,hooray
MacBook-Pro:da_lab_07 mr-ilin$ cat tests/*.t
4 5
01011
10001
```

```
01000
11011
4 8
10010011
01000001
10000100
101000106 14
10010100010010
01010010001001
10100101010100
10101001010010
01100010011001
00000000000000
4 8
11111100
11111100
11111100
00000000
9 5
11101
11101
11001
11001
11001
10000
10000
00000
```

4 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я познакомился с динамическим программированием, концепции которого могут применяться при написании кода для решения множества задач.

В ходе выполнения данной работы мне пригодились навыки написания юнит тестов, полученные в ходе выполнения предыдущих работ, поскольку изначальный алгоритм был придуман неверно, что обнаружилось в ходе тестирования. При это сами тесты не изменились.

Список литературы

[1] *Динамическое программирование — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Динамическое_программирование` (дата обращения: 15.04.2021).

Список литературы

[1] *Динамическое программирование — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Динамическое_программирование` (дата обращения: 15.04.2021).