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: Р. С. Лисин
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б-20

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №7

Задача: 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 $\mathbf{0 < n, m < 501}$. В последующих 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

Согласно [1], динамическое программирование - способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В этом случае время вычислений, по сравнению с «наивными» методами, можно значительно сократить.

Ключевая идея в динамическом программировании достаточно проста. Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи (подзадачи), после чего объединить решения подзадач в одно общее решение. Часто многие из этих подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений. Это особенно полезно в случаях, когда число повторяющихся подзадач экспоненциально велико.

Данная задача как раз решается с помощью динамического программирования. Но найти прямоугольник наибольшей площади в другом прямоугольнике нетривиальная задача. Пойти во все стороны, поддерживая какие-то значения, не получится (так получилось бы, если бы мы искали квадрат максимальной площади). Идея решения заключается в сведении этой задачи к другой задаче - нахождение максимальной площади прямоугольника в гистограмме. Для этого нам нужно пробежаться по нашему двумерному массиву, в котором мы храним исходный прямоугольник из нулей и единиц, по столбцам сверху вниз, насчитывая двумерный массив высот. Для каждого столбца делаем так: если мы увидели 0, тогда записываем 1 в массив высот и сохраняем это значение. Если дальше снова 0, тогда мы уже ставим 2 в массив высот, то есть прибавляем 1 к прошлому шагу и так далее. А если встретилась единица, тогда заполняем эту позицию 0 в массиве высот и обнуляем наш счётчик (память), то есть когда мы встретим в следующий раз 0, то напишем 1 в массив высот.

Следующий шаг - решение задачи нахождения максимальной площади прямоугольника в гистограмме n раз, потому что каждая строка двумерного массива высот это гистограмма. Эта задача решается таким образом. Используя стек, добавляем туда каждый новый прямоугольник. Если высота следующего прямоугольника меньше, чем высота предыдущего, тогда достаём прямоугольники из стека, попутно считая их общую площадь, пока не попадётся прямоугольник, у которого высота меньше или равна высоте текущего прямоугольника. Также нам нужно поддерживать максимум среди всех найденных площадей за всё время. Это и будет ответом на задачу. Ещё доставать все прямоугольники нужно, если числа в строке массива высот закончились.

Сложность алгоритма составляет $O(n * m)$ по времени и памяти, так как мы n раз решаем линейную задачу.

2 Исходный код

Изначально я считываю числа в строку и заполняю двумерный массив из символов $matrix$ - исходный прямоугольник. По нему я насчитываю массив высот h и решаю задачу по описанию, которое я привёл выше.

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

using namespace std;

int FindMaxRectangleSquare(const vector<vector<char>>& matrix) {
    if (matrix.size() == 0) {
        return 0;
    }
    vector<vector<int>> h(matrix.size(), vector<int>(matrix[0].size()));
    for (int i = 0; i < matrix[0].size(); i++) {
        int count = 0;
        for (int j = 0; j < matrix.size(); j++){
            if (matrix[j][i] == '0') {
                count++;
            }
            else {
                count = 0;
            }
            h[j][i] = count;
        }
    }
    int max_area = 0;
    for (int i = 0; i < h.size(); i++) {
        stack<int> s;
        h[i].push_back(0);
        for (int j = 0; j < h[i].size(); j++) {
            while (!s.empty() && h[i][s.top()] >= h[i][j]) {
                int cur_idx = s.top();
                s.pop();
                int width = s.empty() ? j : j - s.top() - 1;
                max_area = max(max_area, h[i][cur_idx] * width);
            }
            s.push(j);
        }
    }
    return max_area;
}

int main() {
    int n, m;
    cin >> n >> m;
```

```
    vector<vector<char>> v(n, vector<char>(m));
    for (int i = 0; i < n; i++) {
        string row;
        cin >> row;
        for (int j = 0; j < m; j++) {
            v[i][j] = row[j];
        }
    }
    cout << FindMaxRectangleSquare(v);
}
```

3 Консоль

```
roma@DESKTOP-JD58QU2:~/Diskran/lab7$ g++ -pedantic -Wall -std=c++11 -Werror
-Wno-sign-compare -O2 -lm main.cpp
roma@DESKTOP-JD58QU2:~/Diskran/lab7$ ./a.out
4 5
01011
10001
01000
11011
4
roma@DESKTOP-JD58QU2:~/Diskran/lab7$ ./a.out
4 4
1110
1100
1000
0000
6
```

4 Тест производительности

Убедимся, что построенный алгоритм действительно имеет сложность $O(n*m)$. Для этого замерим время работы программы на нескольких тестах: с площадями исходного прямоугольника 100, 1000, 10000.

```
roma@DESKTOP-JD58QU2:~/Diskran/lab7$ ./a.out <test1e2
Time: 0.000460 s
roma@DESKTOP-JD58QU2:~/Diskran/lab7$ ./a.out <test1e3
Time: 0.005608 s
roma@DESKTOP-JD58QU2:~/Diskran/lab7$ ./a.out <test1e4
Time: 0.0011082 s
```

Видно, что время работы программы возрастает прямо пропорционально объему входных данных, значит сложность программы действительно равна $O(n*m)$.

5 Выводы

Выполнив данную лабораторную работу, я применял методы динамического программирования. Этот способ решения задач на самом деле помогает решать сложные задачи, используя принцип - разделяй и властвуй. Динамическое программирование пригодится мне в решении задач оптимизации.

Список литературы

[1] *Динамическое программирование*
URL: `https://en.wikipedia.org/wiki/Dynamic_programming` (дата обращения: 23.10.2022).