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: Т. А. Габдуллин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №7

Задача: При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом; оценить время выполнения алгоритма и объём 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.

Вариант: 2. Пустой прямоугольник.

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

Входные данные: в первой строке заданы $1 \leq n, m \leq 500$. В следующих n строках записаны по m символов 0 или 1 - элементы прямоугольника.

Выходные данные: необходимо вывести одно число – максимальную площадь прямоугольника из одних нулей.

1 Метод решения

Задача о нахождении наибольшей нулевой матрицы являтся одной из классических задач динамического программирования. Искомая нулевая подматрица ограничена со всех четырёх сторон границами поля (единицами). Следовательно пропустить возножные решения не получится. Для реализации требуется завести одномерный масив dp, в котором для каждого **j** столбца будет хранится информация о наибольшем номере строки **i**, если в позиции $matrix[i][j]$ содержится единица, иначе -1. Значение -1 вскоре потребуется для выведения формулы. Пользуясь значением $dp[i]$, мы сразу получаем номер верхней строки нулевой подматрицы. Осталось теперь определить оптимальные левую и правую границы нулевой подматрицы, это значит максимально раздвинуть эту подматрицу влево и вправо от j-го столбца.

Для расширения левой границы необходимо найти индекс такого столбца **x**, для которого будет выполняться условие $dp[j] < dp[x]$. Логично, что x – ближайший такой столбец слева для j. Если индекса x, удовлетворяющему условию нет в матрице, то смело предполагаем, что дальше матрицу расширить нельзя. Правая граница подматрицы определяется аналогично – ближайший справа от j индекс такой, что $dp[j] < dp[y]$, иначе m (дальше расширить матрицу нельзя). Площадь нулевой подматрицы подсчитывается с помощью формулы: $(i - dp[j]) \cdot (y - x - 1)$

Для линейной сложности вычисления индексов следует использовать структуру stack. Алгоритм взят с сайта *e-maxx.ru*.

2 Листинг

```
#include <bits/stdc++.h>
using namespace std;

int16_t size_n, size_m;

int32_t Max_Square_in_Line(vector<int16_t> &column)
{
    stack<int16_t> stack;
    int32_t max_count = 0;

    int16_t i = 0;
    while(i < size_m || !stack.empty())
    {
        if(stack.empty() || (column[stack.top()] <= column[i] && i < size_m))
        {
            stack.push(i);
            i++;
        }
        else
        {
            int16_t last_elem = stack.top();
            stack.pop();
            int32_t current_count = column[last_elem] * (stack.empty() ? i : i - stack.
                top() - 1);

            if (current_count > max_count)
            {
                max_count = current_count;
            }
        }
    }
    return max_count;
}
void Read_Rectangle(vector<vector<int8_t> > &rectan) {
    rectan.resize(size_n);
    for (int i = 0; i < size_n; ++i) {
        rectan[i].resize(size_m);
        for (int j = 0; j < size_m; ++j) {
            cin >> rectan[i][j];
        }
    }
}
int main()
{
    cin >> size_n >> size_m;
    vector<vector<int8_t> > rectangle;
    Read_Rectangle(rectangle);
```

```
47
48      vector<int16_t> column_count(size_m);
49      int32_t max_sqr = 0;
50
51      for (int i = 0; i < size_n; ++i)
52      {
53          for (int j = 0; j < size_m; ++j)
54          {
55              if (rectangle[i][j] == '0')
56              {
57                  column_count[j]++;
58              }
59              else
60              {
61                  column_count[j] = 0;
62              }
63          }
64          int32_t current_sqr = Max_Square_in_Line(column_count);
65          if (current_sqr > max_sqr)
66          {
67              max_sqr = current_sqr;
68          }
69      }
70
71      cout << max_sqr << endl;
72
73      return 0;
74  }
```

3 Выводы

Динамическое программирование применяется в огромной сфере деятельностей, в которых присутствуют процессы, которые можно разбить на ряд одинаковых небольших этапов по любому из параметров(будь то время, сумма, расстояние, температура и *т.д.*)
Задачи на динамическое программирование очень часто находят применение в различных областях, например, таких, как олимпиадные контесты *opencup* и *codeforces*. В виду спектра применений, ДП - один из важнейших разделов программирования, для которого крайне желательна хорошая база решенных задач разной сложности.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 2-е издание. — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))(дата обращения: 11.03.2018).

[2] *Понятие динамического программирования*
URL:`http://fb.ru/article/44641/dinamicheskoe-programmirovanie-osnovnyie-printsipyi`
(дата обращения: 12.03.2018).

[3] *Описание алгоритма о поиске максимальной нулевой подматрицы*
URL:`http://www.e-maxx-ru.1gb.ru/algo/maximum`$_z ero_s ubmatrix$(: 12.03.2018).