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: А. В. Барсов
Преподаватель: Н. С. Капралов
Группа: М8О-307Б-19

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №7

Задача: 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:

Вариант №2:

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 $1 \leq n \leq 500$ и $1 \leq m \leq 500$. 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

Тривиальный алгоритм, — перебирающий искомую подматрицу, — даже при самой хорошей реализации будет работать $O(n^2m^2)$. Но мы опишем алгоритм, работающий за O (n m), т.е. за линейное относительно размеров матрицы время.

Для устранения неоднозначностей сразу заметим, что n равно числу строк матрицы a, соответственно, m — это число столбцов. Элементы матрицы будем нумеровать в 0-индексации, т.е. в обозначении a[i][j] индексы i и j пробегают диапазоны i = 0 ... n-1, j = 0 ... m-1.

Шаг 1: Вспомогательная динамика Сначала посчитаем следующую вспомогательную динамику: d[i][j] — ближайшая сверху единица для элемента a[i][j]. Формально говоря, d[i][j] равно наибольшему номеру строки (среди строк диапазоне от -1 до i), в которой в j-ом столбце стоит единица. В частности, если такой строки нет, то d[i][j] полагается равным -1 (это можно понимать как то, что вся матрица как будто ограничена снаружи единицами).

Эту динамику легко считать двигаясь по матрице сверху вниз: пусть мы стоит в i-ой строке, и известно значение динамики для предыдущей строки. Тогда достаточно скопировать эти значения в динамику для текущей строки, изменив только те элементы, в которых в матрице стоят единицы. Понятно, что тогда даже не требуется хранить всю прямоугольную матрицу динамики, а достаточно только одного массива размера m:

```
vector<int> d (m, -1);
for (int i=0; i<n; ++i) {
  for (int j=0; j<m; ++j)
    if (a[i][j] == 1)
      d[j] = i;
}
```

Шаг 2: Решение задачи найти такой индекс k_1, для которого будет $d[i][k_1] > d[i][j]$, и при этом k_1 — ближайший такой слева для индекса j. Понятно, что тогда $k_1 + 1$ даёт номер левого столбца искомой нулевой подматрицы. Если такого индекса вообще нет, то положить $k_1 = -1$ (это означает, что мы смогли расширить текущую нулевую подматрицу влево до упора — до границы всей матрицы a).

Симметрично можно определить индекс k_2 для правой границы: это ближайший справа от j индекс такой, что $d[i][k_2] > d[i][j]$ (либо m, если такого индекса нет).

Итак, индексы k_1 и k_2, если мы научимся эффективно их искать, дадут нам всю необходимую информацию о текущей нулевой подматрице. В частности, её площадь будет равна$(i - d[i][j]) \cdot (k_2 - k_1 - 1)$.

Добиться асимптотики $O(1)$ можно с помощью стека (stack) следующим образом. Научимся сначала искать индекс k_1, и сохранять его значение для каждого индекса j внутри текущей строки i в динамике $d_1[i][j]$. Для этого будем просматривать все

столбцы j слева направо, и заведём такой стек, в котором всегда будут лежать только те столбцы, в которых значение динамики $d[][]$ строго больше $d[i][j]$. Понятно, что при переходе от столбца j к следующему столбцу j+1 требуется обновить содержимое этого стека. Утверждается, что требуется сначала положить в стек столбец j (поскольку для него стек "хороший"), а затем, пока на вершине стека лежит неподходящий элемент (т.е. у которого значение $d \leq d[i][j+1]$), — доставать этот элемент. Легко понять, что удалять из стека достаточно только из его вершины, и ни из каких других его мест (потому что стек будет содержать возрастающую по d последовательность столбцов).

2 Консоль

```
[artem@IdeaPad solution]$ make
g++ -pedantic -Wall -std=c++11 -Werror -Wno-sign-compare -O2 -lm -c main.cpp
g++ -pedantic -Wall -std=c++11 -Werror -Wno-sign-compare -O2 -lm main.o -o
solution
[artem@IdeaPad solution]$ ./solution
4 5
01011
10001
01000
11011
4
[artem@IdeaPad solution]$ python generator.py | ./solution
18
[artem@IdeaPad solution]$ python generator.py | ./solution
22
[artem@IdeaPad solution]$ python generator.py | ./solution
20
[artem@IdeaPad solution]$ python generator.py | ./solution
18
```

3 Листинг прграммы

Обход динамики реализован довольно просто, а главное - работает быстро, т.е. за линейное от размеров матрицы время.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        if (a[i][j] == '1') d[j] = i;
    }
    while (!st.empty()) st.pop();
    for (int j = 0; j < m; j++) {
        while (!st.empty() && d[st.top()] <= d[j]) st.pop();
        d1[j] = (!st.empty() ? st.top() : -1);
        st.push(j);
    }
    while (!st.empty()) st.pop();
    for (int j = m-1; j >= 0; j--) {
        while (!st.empty() && d[st.top()] <= d[j]) st.pop();
        d2[j] = (!st.empty() ? st.top() : m);
        st.push(j);
    }
    for (int j = 0; j < m; j++)
        ans = max(ans, (i - d[j]) * (d2[j] - d1[j] - 1));
}
```

Для генерации случайнх тестов был реализован скрипт генератор generator.py:

```
import random
n, m = 500, 500
print(n, m)
for i in range(n):
    for j in range(m):
        print(random.choice(('0', '1')), end='')
    print()
```

4 Выводы

Выполнив данную лабораторную работу по курсу «Дискретный анализ», я попрактиковал технику динамического программирования, которая помимо того, что очень часто встречается в задачах олимпиадного программирования, так и нередко бывает полезной для реализации и оптимизации более сложных алгоритмов. Тема динамического программирования очень широка и зачастую трудно уловить, что перед тобой задача именно на применение динамики, поэтому необхожимо изучать больше разновидностей задач динамического программирования и больше практиковаться.

Список литературы

[1] Дональд Кнут *Искусство программирования.* Перевод и издатели: В. Штонда, Г. Петриковец, А. Орлович.

[2] *Справочник по алгоритмам и структурам данных.*
URL: `https://e-maxx.ru/` .