Лабораторная работа № 7 по курсу дискретного анализа: Динамическое программирование

Выполнил студент группы 08-208 МАИ *Левштанов Денис*.

Условие

1. 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.

2. Обход матрицы

   Задана матрица натуральных чисел A размерности n*m. Из текущей клетки можно перейти в любую из 3-х соседних, стоящих в строке с номером на единицу больше, при этом за каждый проход через клетку (i, j) взымается штраф $A_{i,j}$. Необходимо пройти из какой-нибудь клетки верхней строки до любой клетки нижней, набрав при проходе по клеткам минимальный штраф.

Метод решения

Считывается размерность матрицы, и сама матрица. Начиная со второй строки для каждого её элемента высчитывается наименьший штраф путем взятия минимума из трех предыдущих шагов. После обхода всех матрицы ищется минимальный элемент последней строки и выводится как ответ. Затем начиная с него восстанавливается путь, по которому это значение было получено.

Описание программы

1. main.h(Считывание данных, управление программой):

   size_t n, m - размерность матрицы

   int64_t** arr - двумерный массив с которой происходит работа

   int64_t minWeight - минимальное значение в текущей строке

   int64_t minIndex - индекс минимального значения

   int64_t* path - массив, в котором хранится путь

Дневник отладки

Программа не проходила 11 тест из-за переполнения 32-битного int, решено использованием типа int64_t.

Тест производительности

1. Time of working 100*100: 0.005
2. Time of working 500*500: 0.132
   (В 25 раз больше элементов, время работы дольше в 26,4 раза)
3. Time of working 1000*1000: 0.56
   (В 4 раза больше элементов, время работы дольше в 4,2 раза)

Выводы

Задача может быть решена напрямую с использованием рекурсии, но в этом случаем сложность будет экспоненциальна ($O(m * 3^n)$), так как некоторые значения высчитываются много раз, что неприемлемо долго. Можно заметить, что для получения значения на текущем этапе нужно знать значения трех элементов с предыдущего этапа, то есть эту программу можно разбить на меньшие задачи, а это значит, что здесь применимо динамическое программирование.

Мы можем искать значения ячеек начиная со второй строки, так как значения первой строки нам известно. Тем самым мы продвигаемся вниз по матрице и высчитываем каждый элемент только один раз и для хранения всех значений мы используем исходную матрицу. Тем самым временная сложность и объем затрачиваемой памяти равны O(m*n). Если нам еще нужно получить путь, то дополнительно требуется массив размером n и один обход от последней строки к первой, который занимает O(n) времени. В итоге общая сложность - O(m*n + n).