Лабораторная работа № 7 по курсу дискретного анализа: динамическое программирование

Выполнил студент группы 08-208 МАИ *Скворцов Александр.*

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.

2. Вариант 6: Задана матрица натуральных чисел A размерности $n \times m$. 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

Чтобы решить задачу методом динамического программирования, необходимо:

1. Описать структуру оптимального решения.

2. Составить рекурсивное решение для нахождения оптимального решения.

3. Вычисленить значение, соответствующего оптимальному решению, методом восходящего анализа.

4. Непосредственное найти оптимальное решение из полученной на предыдущих этапах информации.

Тогда решение поставленной задачи можно описать следующим образом:

- Так как из текущей клетки можно перейти в любую из 3-х соседних, стоящих в строке с номером на единицу больше, то в клетку (i, j) можно попасть только из клеток $(i-1, j-1)$, $(i-1, j)$ и $(i-1, j+1)$. Тогда если нам изветны кратчайшие пути уровня i, то кратчайшай путь в каждую из клеток уровня $i+1$ равен кратчайшему пути из трех соседних клеток уровня i плюс шраф данной клетки. То есть мы нашли оптимальное решение, причем оно рекурсивное.

- Тогда решение для нижней строки можно найти, постепенно вычисляя путь для каждой из строк, начиная с верхней. То есть мы можем найти оптимальное решение методом восходящего анализа.

Описание программы

Программа состоит из одного файла da7.cpp, все вычисления производятся в int main(), никаких собственных функций или типов данных не используется.
Использованные переменные:

- std::vector<std::vector<int>> a — матрица штрафов;
- std::vector<std::vector<long long int>> b — матрица наименьшего штрафа для каждой из клеток;
- std::vector<std::vector<char>> c — матрица кратчайшего пути для каждой из клеток;
- std::stack<int> path — хранит итоговый путь из верхней строки в нижнюю;
- const int LEFT = 0, const int CENTER = 1,const int RIGHT = 2 — константы, для хранения путей.

Дневник отладки

№	Ответ чекера	Причина ошибки
1	Wrong answer at test 01.t	программа выводила лишний пробел
2	Wrong answer at test 06.t	программа выводила неверный путь
3	Wrong answer at test 02.t	программа выводила неверный путь
4	Wrong answer at test 06.t	отправлен неправильный файл
5, 6	Wrong answer at test 11.t	неверный тип матрицы b — int, вместо long long int

Тест производительности

В алгоритме осуществляется цикл по уровням (n итераций), на каждом из которых проодит цикл по элементам (m итераций), операции на элементе производятся за константу (выбор наименьшего из трех элементов и сложение). Таким образом, сложность алгоритма $O(nm)$.

Количество выделяемой памяти также пропорционально $n \times m$, так создается три матрицы размером n на m и один стек размером n.

Размерность матрицы	Время работы	Потребляемая память
200×200	36	616.960
400×400	140	2.492.384
800×800	576	8.462.864
1600×1600	2279	33.483.840
400×600	218	3.232.384

Таким образом, время работы и количество использованной памяти согласуются с теоретическими оценками.

Выводы

Динамическое программирование — не конкретный алгоритм, а метод создания алгоритмов. Его особенностью и условием применимости является пересечение подзадач, поэтому для предотвращения повторных вычислений используется так называемая мемоизация — сохранение ранее вычисленных результатов. Именно поэтому решение находят восходящим способом.

Свое примение динамика находит в различных задачах оптимизации, в нахождении минимумов, максимумов и т.п. Такой способ намного эффективнее перебора, но, к сожалению, нахождение им конечного алгоритма — не всегда тривиальная задача.