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: Е. А. Айрапетова
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

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: У вас есть рюкзак, вместимостью m, а так же n предметов, у каждого из которых есть вес и стоимость w_i, c_i.

Необходимо выбрать такое подмножество I из них, чтобы: $\Sigma_{i\epsilon I}\ w_i \leq m.(\Sigma_{iIci}c_i) * |I|$ является максимальной из всех возможных. $|I|$ – мощность множества I.

1 Описание

Основная идея динамического программирования заключается в том, что сложная задача разбивается на более простые и решение сложной задачи состоит из решений более простых задач [1].

Задача о рюкзаке является известной NP-полной задачей, которая при некоторых ограничениях решается за полиномиальное время с помощью метода динамического программирования.

Для моего варианта задания $dp_{i,j,k}$ — максимальная стоимость j вещей из первых i, таких, что их суммарный вес не превышает k. То есть алгоритм будет перебирать количество предметов, которые будут в рюкзаке.

Пусть существует оптимальное решение в $dp_{i,j,kw,j1}$, тогда $dp_{i+1,j+1,k} = max(dp_{i,j,kwj1} + c_{j+1}, dp_{i+1,j,k})$. В рекуррентной формуле рассматривается два варианта: взять вещь $j + 1$ или нет.

Такое решение имеет $n^2 * m$ состояния, в каждое можно перейти из двух других. Временная сложность алгоритма $O(n^2m)$.

В памяти будет храниться только dp_i, dp_{i+1} и битовые множества предметов, которые оптимальны для решения подзадачи. Пространственная сложность такого подхода $O(n * m)$.

2 Исходный код

Для хрения множества предметов опишем матрицы $setCurr$ и $setPrev$, а для dp_{j+1} и dp_j - матрицы $dpPrev$ и $dpCurr$.

```
#include <bitset>
#include <iostream>
#include <vector>
const size_t MAX_N = 100;

int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector<int> w(n);
    std::vector<long long> c(n);
    for (int i = 0; i < n; ++i) {
        std::cin >> w[i] >> c[i];
    }

    std::vector< std::vector< long long > > dpPrev(n + 1, std::vector<long long>(m + 1)
        );
    std::vector< std::vector< long long > > dpCur(n + 1, std::vector<long long>(m + 1))
        ;
    std::vector< std::vector< std::bitset<MAX_N> > > setPrev(n + 1, std::vector< std::
        bitset<MAX_N> >(m + 1));
    std::vector< std::vector< std::bitset<MAX_N> > > setCur(n + 1, std::vector< std::
        bitset<MAX_N> >(m + 1));
    long long ans = 0;
    std::bitset<MAX_N> res;
    for (int j = 1; j < n + 1; ++j) {
        for (int k = 1; k < m + 1; ++k) {
            dpPrev[j][k] = dpPrev[j - 1][k];
            setPrev[j][k] = setPrev[j - 1][k];
            if (c[j - 1] > dpPrev[j][k] and k - w[j - 1] == 0) {
                dpPrev[j][k] = c[j - 1];
                setPrev[j][k] = 0;
                setPrev[j][k][j - 1] = 1;
            }
            if (dpPrev[j][k] > ans) {
                ans = dpPrev[j][k];
                res = setPrev[j][k];
            }
        }
    }
    for (long long i = 2; i < n + 1; ++i) {
        for (int j = 1; j < n + 1; ++j) {
            for (int k = 1; k < m + 1; ++k) {
                dpCur[j][k] = dpCur[j - 1][k];
                setCur[j][k] = setCur[j - 1][k];
```

```
                if (k - w[j - 1] > 0 and dpPrev[j - 1][k - w[j - 1]] > 0) {
                    if (i * (c[j - 1] + dpPrev[j - 1][k - w[j - 1]] / (i - 1)) > dpCur[j
                        ][k]) {
                        dpCur[j][k] = i * (c[j - 1] + dpPrev[j - 1][k - w[j - 1]] / (i -
                            1));
                        setCur[j][k] = setPrev[j - 1][k - w[j - 1]];
                        setCur[j][k][j - 1] = 1;
                    }
                }
                if (dpCur[j][k] > ans) {
                    ans = dpCur[j][k];
                    res = setCur[j][k];
                }
            }
        }
        std::swap(dpCur, dpPrev);
        std::swap(setCur, setPrev);
    }
    std::cout << ans << '\n';
    for (int i = 0; i < n; ++i) {
        if (res[i]) {
            std::cout << i + 1 << ' ';
        }
    }
    std::cout << '\n';
    return 0;
}
```

3 Консоль

```
jane@Evgenia:/mnt/c/Files/ДА/ЛР7$ g++ main.cpp
jane@Evgenia:/mnt/c/Files/ДА/ЛР7$ cat test1.txt
3 6
2 1
5 4
4 2
jane@Evgenia:/mnt/c/Files/ДА/ЛР7$ cat test2.txt
14 41
2 60
6 25
10 56
8 4
7 81
4 40
10 56
7 2
8 32
2 25
6 22
9 5
9 95
8 6
jane@Evgenia:/mnt/c/Files/ДА/ЛР7$ ./a.out <test1.txt
6
1 3
jane@Evgenia:/mnt/c/Files/ДА/ЛР7$ ./a.out <test2.txt
2674
1 2 3 5 6 10 13
```

4 Тест производительности

Тест производительности представляет из себя тестрирование программы на разных тестах за время.

```
jane@Evgenia:/mnt/c/Files/ДА/ЛР7$ ./benchmark <test1.txt
6
1 3
input 2 ms
jane@Evgenia:/mnt/c/Files/ДА/ЛР7$ ./benchmark <test2.txt
2674
1 2 3 5 6 10 13
input 4 ms
```

Видно, что алгоритм работает довольно быстро, что соответствует временной сложности $O(n * m)$.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я научилось работать с динамическим программированием, разделяя одну большую задачу на более простые. Также для оптимизации памяти я работала с std::bitset, которая по эффективности не уступает std::vector<bool>, которая будет работать по похожей схеме.

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.))

[2] *NP-полная задача — Википедия.*
URL: `http://ru.wikipedia.org/wiki/NP-полная_задача` (дата обращения: 16.05.2021).