Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт № 8 информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №7 по курсу**

**«Дискретный анализ»**

**задача о рюкзаке**

Студент: Пермяков Никита Александрович

Группа: М80 – 208Б-19

Вариант: 1

Преподаватель: [*Кухтичев Антон Алексеевич*](https://mai.ru/education/schedule/ppc.php?guid=a3f854e1-f771-11e7-ae95-485b3919ee6d)

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

**Содержание**

1. Постановка задачи
2. Метод и алгоритм решения
3. Описание программы
4. Дневник отладки
5. Тестирование производительности
6. Вывод

**Постановка задачи**

**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 также n предметов, у каждого из которых есть вес wi и стоимость ci . Необходимо выбрать подмножество I. |I| – мощность множества I.

**Метод и алгоритм решения**

Динамическое программирование — метод решения задачи - разбиение на несколько одинаковых подзадач, рекуррентно связанных между собой. Задача о рюкзаке является NP-полной задачей комбинаторной оптимизации, которая не решается за полиномиальное время.

total — 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.

В рекуррентной формуле рассматривается два варианта: взять вещь j + 1 или нет. Решение будет иметь n 2 ∗ m состояния, в каждое можно перейти из двух других.

Так временная сложность алгоритма O(n 2 ∗ m).

Хранение таблицы состояний дорого по памяти, но необходимо для восстановления ответа. Поэтому сохраняем total[i] и total[i+1] и битовые множества предметов, которые оптимальны для решения подзадачи. Сложность O(n ∗ m) по памяти.

**Описание программы**

struct thing - структура хранит параметры предмета

class TrickyBackpack - класс объединяющий методы решения задачи, и поля ответа с маской.

void PutData(struct thing& thing\_item) - забор данных

void GenerateMatrix(struct thing& thing\_item) - генерация матрицы решений

void PrintResponse() - вывод ответа

поля weight, count - заданы начальными условиями задачи

**Дневник отладки**

1-5)Понимание формата входных данных

6-10) Написание представление данных с помощью двух структур - векторов

11-15) Написание представление данных с помощью одной структуры и двух векторов в классе

15-30) Перепись алгоритма подсчета

30-40) Использование bitset - по другому никакак

**Тестирование производительности**

генератор: gen\_tests.py

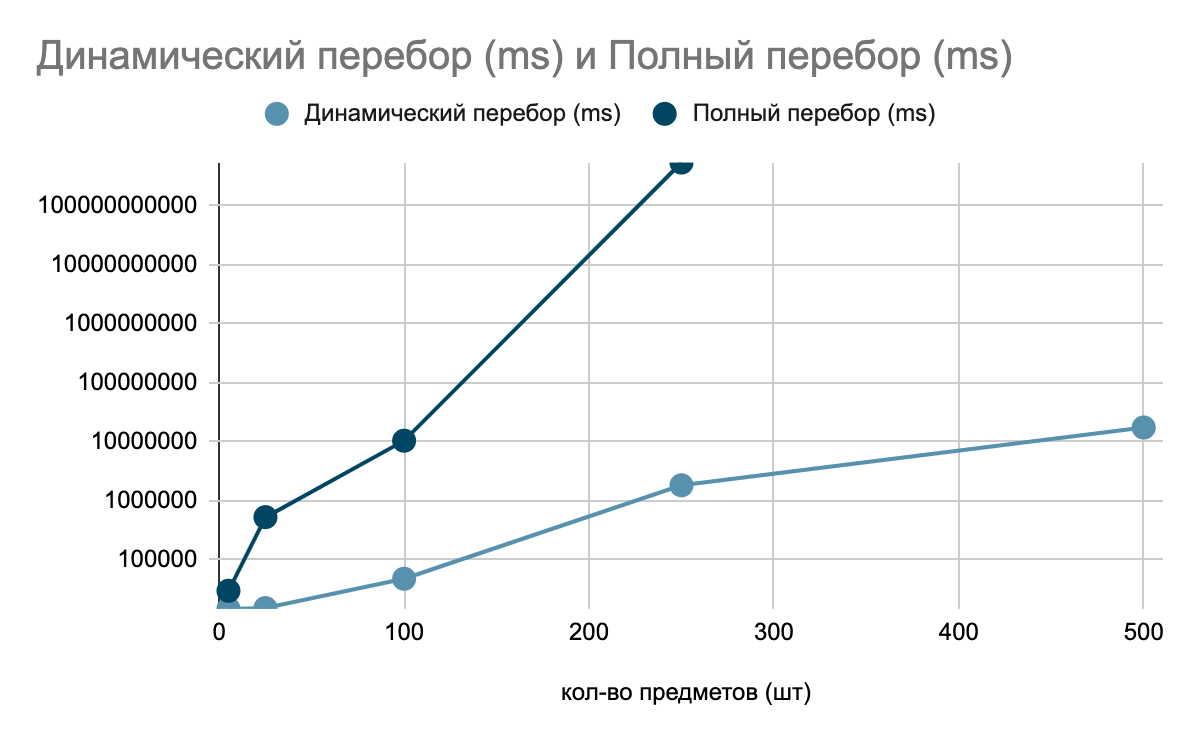
Формат входных данных

В первой строке заданы 1 ≤ n ≤ 100, 1000, 10000, 100000, 1000000 и 1 ≤ m ≤ 5000. В последующих n строках через пробел заданы параметры предметов: wi и ci.

Между методами реализации тестовый набор не менялся

**Таблица 1.** Зависимость величины тестовых данных от времени работы программы, с разделением на два метода реализации

|  |  |  |
| --- | --- | --- |
| кол-во предметов (шт) | Динамический перебор (ms) | Полный перебор (ms) |
| 5 | 14502 | 29433 |
| 25 | 14990 | 523245 |
| 100 | 47092 | 10366599 |
| 250 | 1816575 | 543484923445 |
| 500 | 17357519 | 25303643534676599 |



**Вывод**

В ходе выполнения лабораторной работы были изучены задачи динамического программирования, реализован алгоритм задачи о неограниченном рюкзаке.

Существуют другие методы решения:

Точные:

* Полный перебор O(2N)
* Метод ветвей и границ (отбрасываем заведомо проигрышные варианты)
* Задача о рюкзаке 0-1 (каждый предмет в единственном экземпляре)

Приближенные:

* Жадный алгоритм (отсортировать вещи по их удельной ценности)
* Генетические алгоритмы (функция приспособляемости - отсев лишних)
* Приближенные схемы за полиномиальное время (разбиение на классы)

Освоил на практике std::bitset для уменьшении потребляемой программой памяти. Написал генератор тестовых данных, протестировал производительность в зависимости от объема входных данных и соотношения малых дорогих - больших дешевых предметов.