НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Алгоритмы и структуры данных»

Контрольное домашнее задание №1

Исследование алгоритмов решения задачи о рюкзаке

Выполнил: Ярных Роман,
студент группы БПИ141.

Преподаватели: Ульянов Михаил Васильевич,
д.т.н., профессор департамента ПИ ФКН НИУ ВШЭ;
Ахметсафина Римма Закиевна,
к.т.н., доцент департамента ПИ ФКН НИУ ВШЭ;

Москва 2015

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

Имеются m предметов с номерами от 0 до m-1, для каждого из которых известна масса в килограммах wj и стоимость sj (j = 0,1,...,m-1). Определить, какие предметы необходимо положить в рюкзак, чтобы их общая масса не превышала W килограммов, а общая стоимость S была максимальной. Каждый предмет можно использовать только один раз. Вывести номера предметов, их стоимость и вес, общий вес рюкзака и общую стоимость рюкзака.

Решить задачу о рюкзаке методами Перебор с возвратом итерационный Перебор с возвратом рекурсивный Динамическое программирование Жадный алгоритм

Измерить время решения задачи каждым методом для одного и того же веса рюкзака и разного количества предметов (от 5 до 20).

Для корректного измерения времени вызвать каждый метод решения 20 раз, найти среднее время для каждого количества предметов.

Входной файл input_knap.txt, все данные – положительные целые числа. По строкам:

1. NT – количество тестов
2. W - максимальный вес рюкзака (1 <= W <= 50)
3. N - количество предметов (1 <= N <= 20)
4. веса предметов (через пробел)
5. стоимости предметов (через пробел)

Строки 2-5 повторяются NT раз.

Выходной файл output_knap.txt, содержит для каждого из NT тестов результаты решения всеми методами в виде:

- номер теста;
- наименование метода, среднее время расчета (в тактах или в наносекундах, указать единицы измерения);
- вес рюкзака, стоимость рюкзака;
- номера предметов в рюкзаке, их вес и стоимость Эти же данные необходимо вывести в консоль.

2. ИНСТРУМЕНТЫ РАЗРАБОТКИ И ИЗМЕРЕНИЯ ВРЕМЕНИ, ИСПОЛЬЗОВАННЫЕ В ПРОЕКТЕ

2.1 Инструменты разработки

При разработки проекта были использованы следующие инструменты программирования и отладки:

- компилятор gcc (g++) языка программирования C++11;
- среда разработки Jetbrains Clion;
- отладчик gdb;
- утилита сборки проектов CMake;
- операционная система OS X 10.9.5;
- среда разработки Xcode 5;
- инструменты разработчика Xcode Command Tools;
- виртуальная машина JVM.

2.2 Инструменты измерения времени

Для измерения времени использовалась библиотека C++11, лежащая в заголовочном файле <chrono>. В данном заголовочном файле декларированы классы std::chrono::time_point (класс, инкапсулирующий точку отсчета времени и хранящий информацию о текущем времени), std::chrono::high_resolution_clock (класс, реализующий функцию подсчета времени). Чтобы отделить функционал программы, отвечающий за измерение времени, от других частей приложения, был разработан отдельный класс TimeCounter, инкапсулирующий необходимые операции над временем. TimeCounter содержит следующие члены:

- статический метод static void TimeCounter::enter(), который при запуске устанавливает точку отсчета времени в вызывающем контексте;
- статический метод static long long int TimeCounter::leave(), отвечающий за подсчет прошедших наносекунд с момента начала работы текущего контекста и до ее завершения.

3. ОПИСАНИЕ ПЛАНА ЭКСПЕРИМЕНТА ПО ИЗМЕРЕНИЮ ВРЕМЕНИ

В рамках исследования зависимости производительности работы методов решения задачи о рюкзаке был разработан план измерения времени. Измерения должны быть произведены в отношении следующих алгоритмов (методов):

— итеративный перебор с возвратом с использованием стека;
— рекурсивный перебор с возвратом;
— жадный алгоритм;
— упаковка рюкзака с использованием динамического программирования.

Для того чтобы получить наиболее качественные оценки производительности, было решено на вход подавать 250 тестов, в каждом из которых содержится по 10 тестов для одного и того же числа предметов от 1 до 25 единиц. При каждом вызове метода для решения задачи о рюкзаке в текущем контексте начинался отсчет времени с помощью запуска метода TimeCounter::enter(), реализованного в классе TimeCounter. По окончании работы алгоритма вызывался другой метод TimeCounter::leave(), который возвращает количество наносекунд прошедших с момента начала работы алгоритма до ее окончания. Данные измерения записываются в выходные файлы output_knap.txt, который также является и решением исходной задачи, и специальный файл time_table.csv, служащий для дальнейшей обработки полученных результатов.

4. ОПИСАНИЕ АЛГОРИТМОВ

4.1 Описание задачи о рюкзаке и методов ее решения

Задача упаковки рюкзака (англ. Knapsack problems) является одной из классических NP-задач комбинаторной оптимизации. В общем виде задачу можно сформулировать так: из заданного множества предметов со свойствами «стоимость» и «вес» требуется отобрать подмножество с максимальной полной стоимостью, соблюдая при этом ограничение на суммарный вес:

$$\sum_{i=1}^{n} p_i x_i \to max \Leftrightarrow \sum_{i=1}^{n} w_i x_i \leq W\,, x_i \in \{0,1\}, \forall i \in [1,n] \cup \mathbb{N}$$

Существует несколько видов задачи о рюкзаке. Отличаются они друг от друга требованиями к рюкзаку, предметам, которые нужно упаковать, и способу их выбора. В данном исследовании решается так называемая задача «рюкзак 0-1» (англ. 0-1 Knapsack problem), которая требует не более одного вхождения одного и того же предмета в рюкзак. Для решения данной задачи существует множество алгоритмов и методов: полный перебор, методов ветвей и границ, жадный алгоритм, динамическое программирование и генетические алгоритмы. В рамках данного исследования мы реализовали четыре алгоритма решения задачи о ранце:

— итеративный перебор с возвратом;
— рекурсивный перебор с возвратом;
— жадный алгоритм;
— динамическое программирование.

4.2 Описание итеративного перебора с возвратом

Сложность обычного полного перебора для задачи о рюкзаке в общем виде есть $O(N!)$. Для задачи «рюкзак 0-1» сложность перебора в худшем случае оценивается как $O(2^N)$. В данном исследовании использовался улучшенный вариант алгоритма – итеративный перебор с возвратом с использованием стека. Сложность итеративного перебора в худшем случае совпадает со сложностью обычного полного перебора, но в лучшем случае оценивает как $O(2^{N/2})$. В основе алгоритма лежит принцип отбрасывания заведомо плохих решений. На каждой итерации мы пытаемся построить лучший набор предметов, в котором суммарная стоимость будет наибольшей среди всех ранее просмотренных наборов при условии, что суммарный вес не будет превышать максимальный вес рюкзака. Если какой-то предмет из текущего набора не может влезть в рюкзак, то мы пропускаем данный набор и переходим к следующему. Также все плохие решения мы запоминаем и на следующих итерациях пропускаем. В случае, когда суммарный вес всех предметов, которые будут упакованы, будет меньше, чем вес рюкзака,

итеративный перебор с возвратом скатывается в полный, а время выполнения деградирует до сложности $O(2^N)$. Так как алгоритм использует для решения стек, который представляет собой виртуальный рюкзак, то для работы требуется $O(N)$ дополнительной памяти.

4.2 Описание рекурсивного перебора с возвратом

Рекурсивный алгоритм перебора с возвратом схож с итеративным, но отличается от него тем, что использует рекурсивные вызовы метода. Каждая ветка вызова проверяет, можно ли доложить следующий предмет. После того как дочерняя ветка обработана, она возвращает своему корню максимальную стоимость и набор. Корень выбирает один из двух наборов, проверяя, какой из них содержит наиболее дорогие элементы, и пытается в него доложить новый предмет. Сложность алгоритма можно оценить как $O(2^N)$. С точки зрения практической реализации рекурсивный перебор хуже, чем итеративный, так как каждый вызов рекурсивной функции дорогостоящий, требующий много памяти.

4.3 Описание жадного алгоритма

Среди перечисленных алгоритмов жадный алгоритм является самым быстрым. Однако у жадного алгоритма есть существенный недостаток – он не всегда дает правильный результат. При некоторых входных данных метод, реализующий жадный алгоритм, можно скатится в локальный оптимум, когда есть рядом более глубокий оптимум. Временная сложность в худшем, среднем и лучшем случае оценивается как $O(N\log N)$. Важно заметить, что устойчивость алгоритма зависит от алгоритма сортировки. Если на практике использовать некоторые алгоритмы сортировки такие, как быстрая сортировка Хоара и т.п., то сложность жадного алгоритма может скатится в $O(N^2)$, поэтому важно подобрать хорошую устойчивую сортировку, которая работает в худшем, среднем и лучшем случае одинаково.

Согласно жадному алгоритму предметы сортируются по убыванию стоимости единицы веса каждого. В рюкзак последовательно складываются самые дорогие за единицу веса предметы из тех, что помещаются внутри. Причем, как было сказано выше, данный алгоритм является приближенным, так как не всегда обеспечивает оптимального решения.

4.4 Описание метода динамического программирования

Последнее место в списке реализованных алгоритмов занимает алгоритм основан на методе динамического программирования. Суть динамического программирования сводится к тому, что мы делим задачу на меньшие подзадачи, после чего объединяем решения подзадач в одно общее решение. Метод динамического программирования сверху – это простое запоминание результатов решения тех подзадач, которые могут повторно встретиться в дальнейшем.

Для решения задачи о ранце с помощью динамического программирования мы строим матрицу A размером $(N+1)\times(W+1)$, которую заполняем следующим образом: $A(0,w)=0;\ A(n,0)=0,$ где $\forall w \in \overline{0..W}$ & $\forall n \in \overline{0..N}$. Пусть $A(n,w)$ есть максимальная стоимости предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые n предметов, то есть $\{e_1, e_2, e_3, \dots, e_n\}$ назовем этот набор допустимых предметов для $A(n,w)$. Найдем $A(n,w)$. Возможны 2 варианта:

1. Если предмет n не попал в рюкзак. Тогда $A(n,w)$ равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов $\{e_1, e_2, e_3, \dots, e_n\}$, то есть $A(n,w)=A(n-1,w)$;
2. Если n попал в рюкзак. Тогда $A(n,w)$ равно максимальной стоимости рюкзака, где вес w уменьшаем на вес n-ого предмета и набор допустимых предметов $\{e_1, e_2, e_3, \dots, e_n\}$ плюс стоимость n, то есть $A(n-1,w-w_n)+p_n$.

То есть:
1. $A(n,w)=A(n-1,w)$;
2. $A(n,w)=A(n-1,w-w_n)+p_n$.

Выберем из этих двух значений максимальное:
$A(n,w)=\max(A(n-1,w);\ A(n-1,w-w_n)+p_n)$
Стоимость искомого набора равна $A(N,W)$, так как нужно найти максимальную стоимость рюкзака, где все предметы допустимы и вместимость рюкзака W.

Будем определять, входит ли предмет в искомый набор. Начинаем с элемента $A(i,w)$, где $i=N$, $w=W$. Для этого сравниваем $A(i,w)$ со следующими значениями:

1. Максимальная стоимость рюкзака с такой же вместимостью и набором допустимых предметов $\{e_1, e_2, e_3, \dots, e_{i-1}\}$, то есть $A(i-1,w)$;
2. Максимальная стоимость рюкзака с вместимостью на w_i меньше и набором допустимых предметов $\{e_1, e_2, e_3, \dots, e_{i-1}\}$ плюс стоимость p_i, то есть $A(i-1,w-w_i)+p_i$.

Заметим, что при построении A мы выбирали максимум из этих значений и записывали в $A(i,w)$. Тогда будем сравнивать $A(i,w)$ с $A(i-1,w)$, если равны, тогда предмет не входит в искомый набор, иначе входит.

5. ПОЛУЧЕННЫЕ ДАННЫЕ ИЗМЕРЕНИЙ

Все исходные измерения времени были записаны в файл time_table.csv, в котором есть столбцы “Номер теста”, “Количество предметов”, “Перебор с возвратом”, “Рекурсивный перебор”, “Жадный алгоритм”, “Динамическое программирование”. Каждая строка содержит время работы методов в наносекундах, причем для каждого набора из n предметов предусмотрено по 10 тестов, что в общей сложности дает 250 тестов для наборов от 1 до 25 предметов. На основе полученных результатов были вычислены средние величины времени работы для каждого из исходных наборов предметов:

Тест	Backtracking (нс)	Recursive (нс)	Greedy (нс)	Dynamic (нс)
1	14227,8	2765,9	2461,5	4960,2
2	4338,8	2362,7	1972,5	8984,3
3	5021,4	4534,5	3186,5	13145
4	6187,3	8445,2	2985,8	16096,9
5	8217,4	15768,9	3643,3	17680,9
6	11290,9	29250,6	3719,2	23782,9
7	15310,6	48497,4	3833,4	23088,3
8	20628,4	91253,1	3942,9	27124,7
9	37378,2	193649,7	4888	35914,7
10	47409,2	294084,5	4733,9	36424,3
11	42127,7	317121,9	4755,2	30354,9
12	90207,2	748393,4	4937,1	43909,4
13	166611,8182	1515134,818	5951,363636	55976,36364
14	219071,4	2079028,8	6417,3	60201,1
15	248887,9	2874727,9	6841,6	53498,6
16	330605,1	3798391,4	7088,3	59984,7
17	355550,2	4185340,2	6470,3	55186,3
18	422151,9	5607089,1	23786,5	56766,3
19	337419,9	5017616,3	8917,9	61068,7
20	1441356,3	18388162,8	11500,8	102621,8
21	956617,8	15807834,9	10038,5	70454
22	1849105,2	32737135,1	12052,4	86626
23	2026438,6	40016203,9	11988,5	97812
24	2080225	41141286,9	10078,7	71402,4
25	2269266	45499861,2	11192,4	84759,3

Таблица №1. Зависимость среднего времени работы от числа предметов

Для каждого из методов на основе составленной таблицы (таблица №1) мы вычислили среднее время работы, дисперсию, среднеквадратичное отклонение, 95%- и 90%-доверительный интервал для математического ожидания:

Параметр	Backtracking	Recursive	Greedy	Dynamic
среднее значение	520226,0807	8816957,645	7095,354545	47912,96255
дисперсия	5,79561E+11	2,16027E+14	22115478,68	784285115
среднее квадратичное отклонение	761289,1342	14697849,09	4702,709717	28005,09088
доверительный интервал (95%)	520226,0807 ±298419,857	8816957,645 ±5761450,973	7095,354545 ±1843,428335	47912,96255 ±10977,7939
доверительный интервал (90%)	520226,0807 ±250441,8387	8816957,645 ±4835162,076	7095,354545 ±1547,053827	47912,96255 ±9212,855061

Таблица №2. Статистические оценки полученных данных

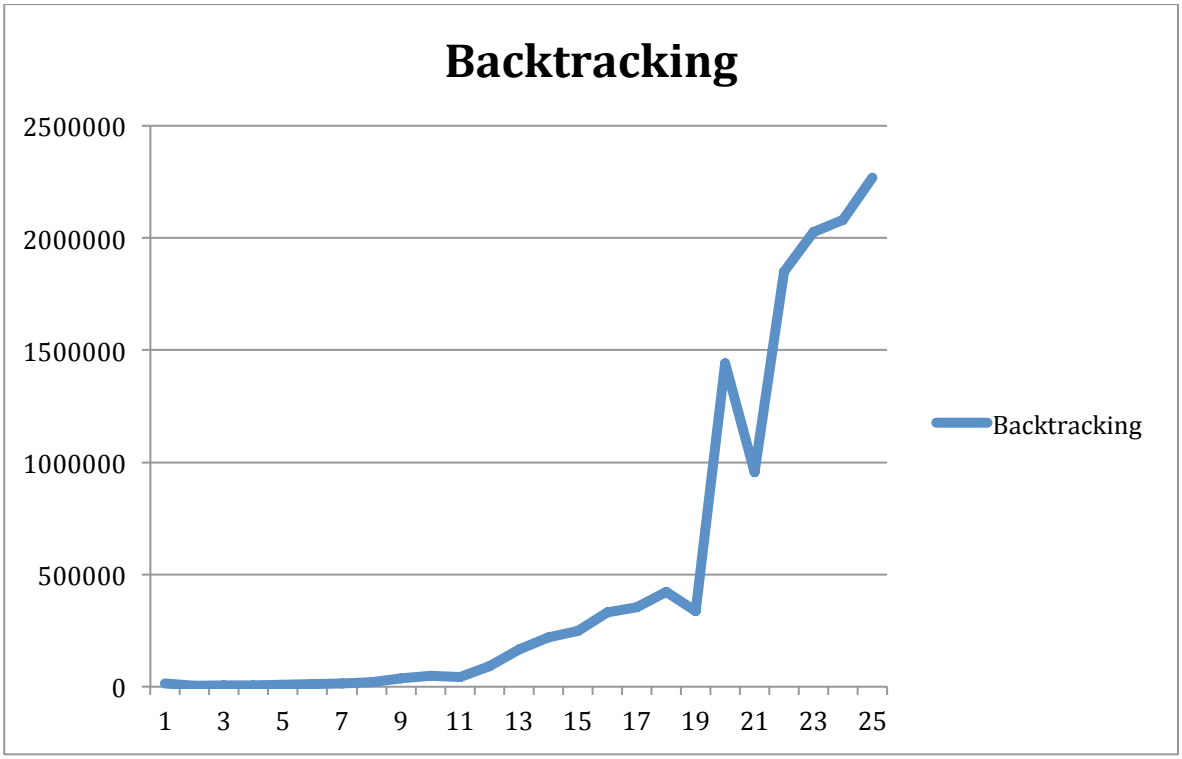

Рис. 1 – График зависимости среднего времени работы полного перебора с возвратом от числа предметов

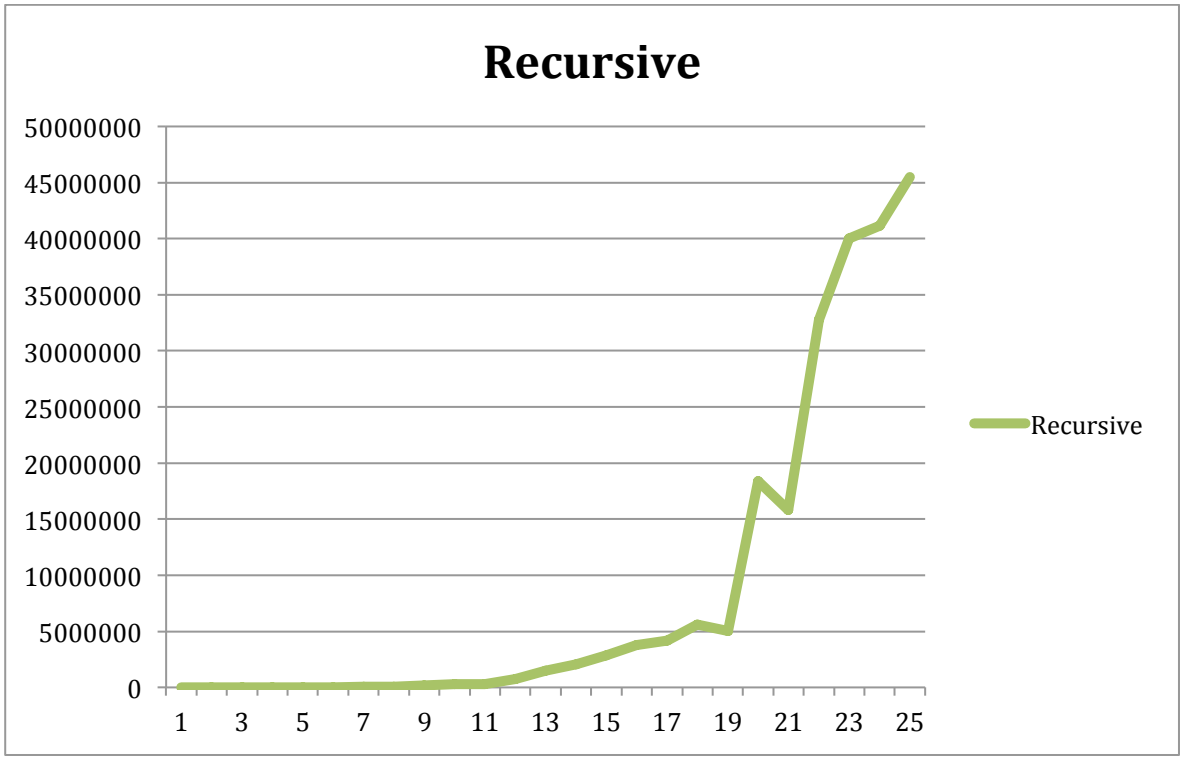

Рис. 2 – График зависимости среднего времени работы рекурсивного перебора с возвратом от числа предметов

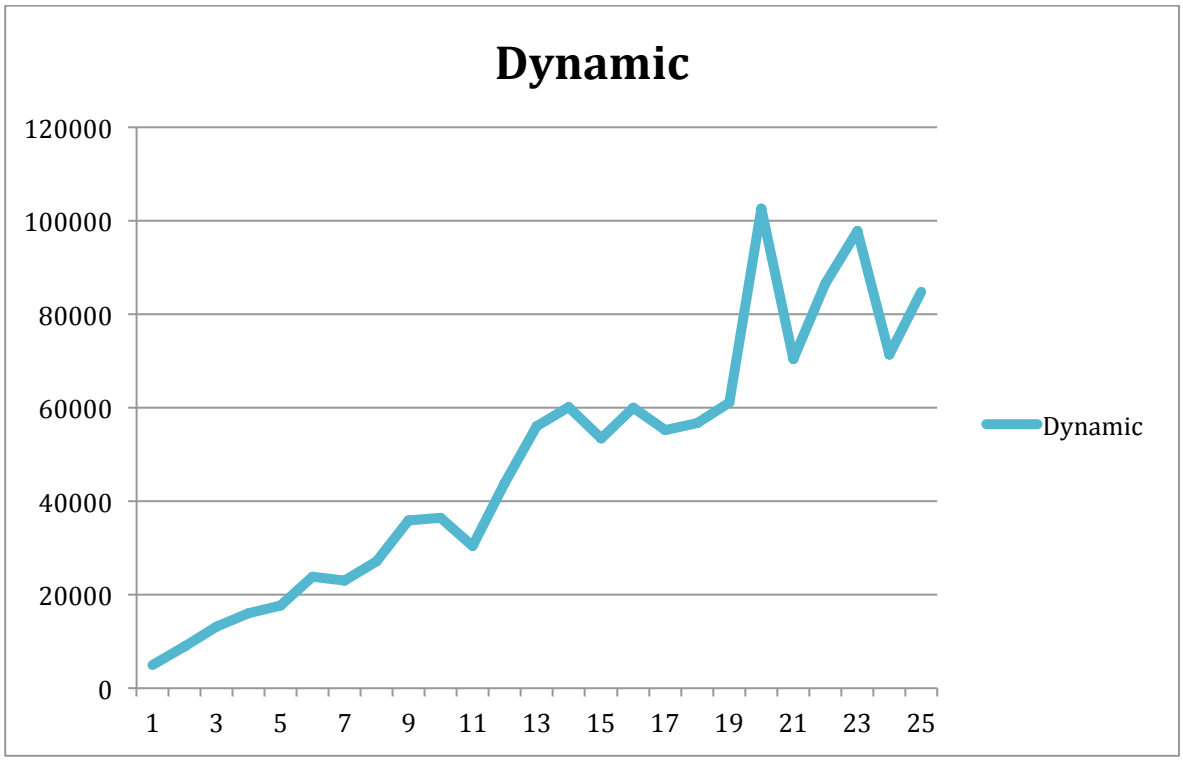

Рис. 3 – График зависимости среднего времени работы динамического программирования от числа предметов

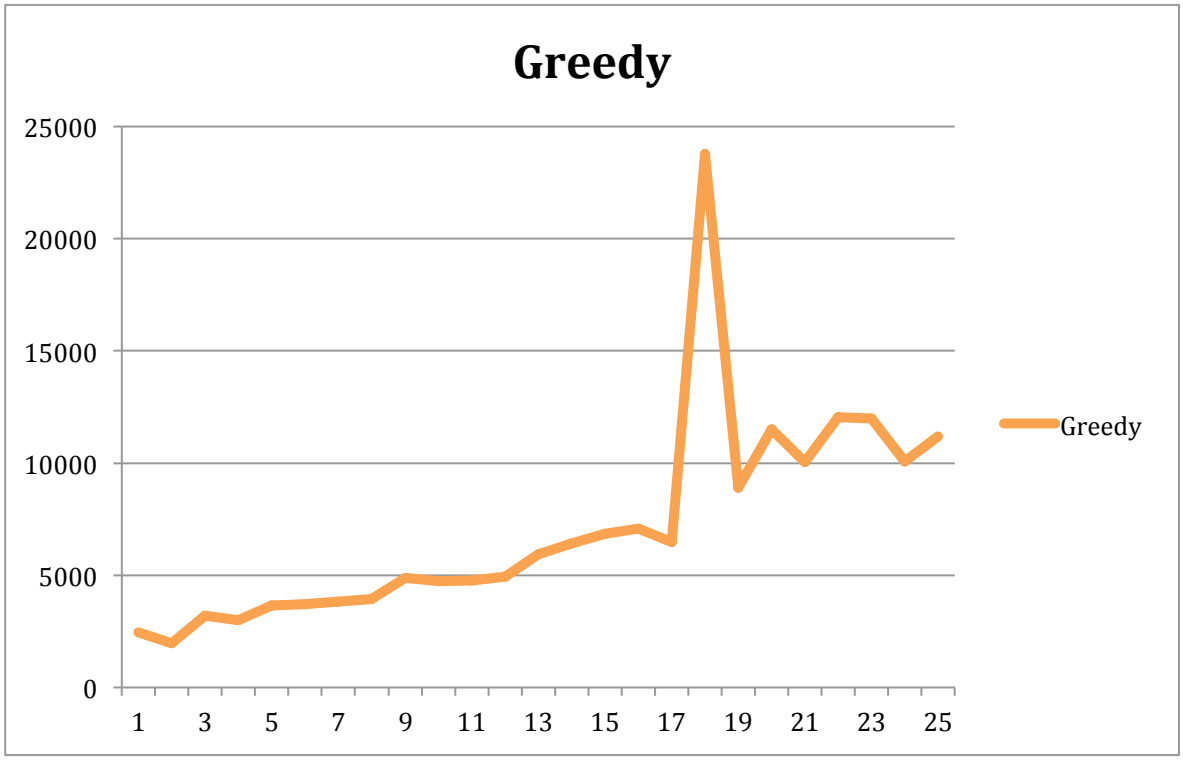

Рис. 4 – График зависимости среднего времени работы жадного алгоритма от числа предметов

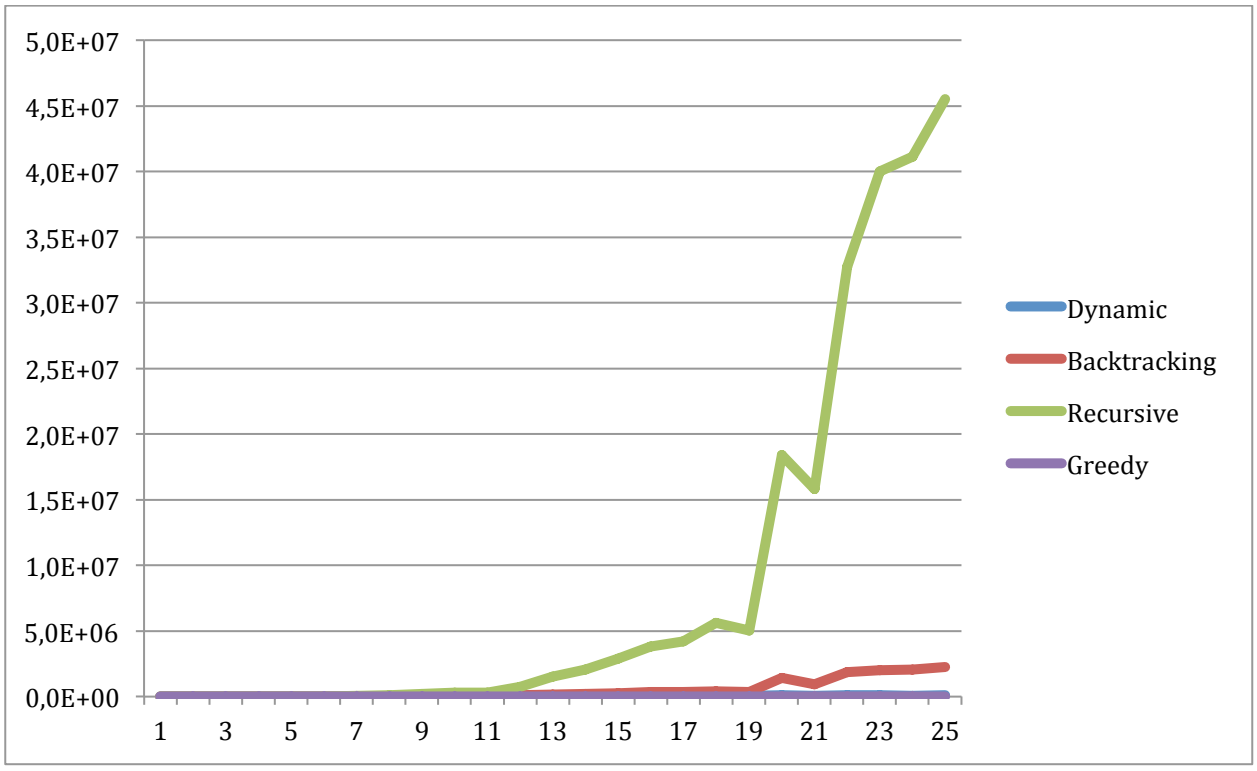

Рис. 5 – Сравнение производительности методов решения задачи о рюкзаке

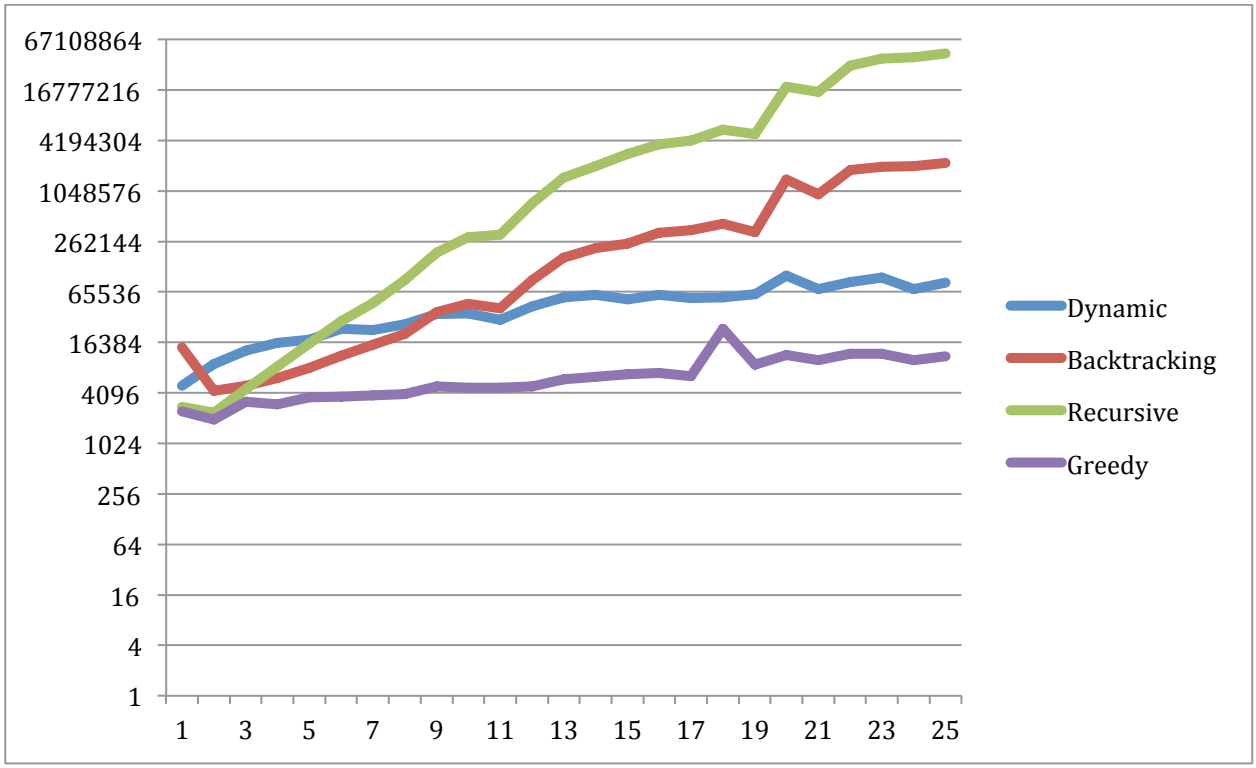

Рис. 6 – Сравнение производительности методов решения задачи о рюкзаке (логарифмическая шкала по основанию 2)

6. СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ

На основе полученных данных были получены статистические оценки работы методов: среднее время работы метода, дисперсия, среднеквадратичное отклонение, 95%- и 90%-доверительный интервалы для среднего времени работы. С некоторой точностью данные результаты измерений равны теоретическим оценкам сложности алгоритмов, решающих задачу о рюкзаке. Для наглядности приведена таблица (Таблица №3), в которой содержится временная сложность в худшем, среднем и лучшем случаем, а также требуемая память. Данная таблица позволяет обобщить результаты исследований и на большие наборы предметов и веса рюкзака. Также важно заметить, что графики зависимости времени от числа предметов аппроксимируют теоретические функции временной сложности.

Метод решения	**Временная сложность**	**Дополнительная память**
Итеративный перебор с возвратом с использованием стека	в худшем случае $O(2^N)$	$O(N)$ памяти для стека
Рекурсивный перебор с возвратом	в худшем случае $O(2^N)$	$O(1)$
Жадный алгоритм	в худшем, лучшем и среднем случае $O(N\log N)$	$O(N)$
Динамическое программирование	в худшем, лучшем и среднем случае $O(NW)$	$O(NW)$

Таблица №3. Теоретическая временная сложность алгоритмов и требуемая память

Все статистические оценки производительности методов такие, как среднее время, СКО (среднеквадратичное отклонение), 95%- и 90%-доверительный интервалы для среднего времени работы, находятся в таблица №4 и №5.

Метод решения	**Среднее время**	**СКО (среднеквадратичное отклонение)**
Итеративный перебор с возвратом с использованием стека	520226,0807	761289,1342
Рекурсивный перебор с возвратом	8816957,645	14697849,09

Жадный алгоритм	7095,354545	4702,709717
Динамическое программирование	47912,96255	28005,09088

Таблица №4. Статистическая оценка производительности методов

Метод решения	**95%**	**90%**
Итеративный перебор с возвратом с использованием стека	520226,0807 ±298419,857	520226,0807 ±250441,8387
Рекурсивный перебор с возвратом	8816957,645 ±5761450,973	8816957,645 ±4835162,076
Жадный алгоритм	7095,354545 ±1843,428335	7095,354545 ±1547,053827
Динамическое программирование	47912,96255 ±10977,7939	47912,96255 ±9212,855061

Таблица №5. Доверительные интервалы для среднего времени работы

Теперь проведем сравнительный анализ алгоритмов. Как видно из графиков зависимости времени работы методов от количества предметов (рис. 5, рис. 6) в среднем медленнее всех работает рекурсивный перебор с возвратом. Среднее время в наносекундах работы рекурсивного перебора больше среднего времени итеративного перебора, динамического программирования и жадного алгоритма в 17, 184 и 1243 раз соответственно. Данная особенность связана с тем, что сам вызов рекурсивной функции достаточно дорогостоящий для центрального процессора и оперативной памяти, так как при каждом дочернем вызове выделяется отдельный контекст данных.

Среднее время работы итеративного перебора больше среднего времени динамического программирования и жадного алгоритма в 11 и 73 раз соответственно, что немного лучше по сравнению с рекурсивным перебором.

Так или иначе, с некоторой точностью полученные функциональные зависимости производительности методов решения задачи о рюкзаке от числа предметов аппроксимируют теоретические функции сложности. Практически полученная функциональная зависимость от числа предметов для итеративного и рекурсивного перебора ведет себя почти, как экспонента, в то время как динамическое программирование и жадный алгоритм работают за полиномиальное и полилогарифмическое время. С точки зрения высокой скорости работы методов решения задачи о рюкзаке на практике есть смысл использовать только динамическое программирование и жадный алгоритм, так как при больших значениях веса рюкзака и числа предметов

вычисление с помощью перебора на обычных компьютерах не представляется возможным.

Для получения точного решения необходимо использовать динамическое программирование, однако за его использование нужно платить дополнительной памятью. Жадный алгоритм достаточно быстро дает результат, но часто решение оказывается не самым оптимальным.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1) Томас Х. Кормен. Алгоритмы: построение и анализ. / Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн; пер. с англ. И.В. Красикова; под ред. И.В. Красикова. – М.: ООО “И.Д. Вильямс”, 2013.
2) Wikipedia. Knapsack problem [Электронный ресурс] //URL: https://en.wikipedia.org/wiki/Knapsack_problem#Computational_complexity (Дата обращения: 13.12.2015, режим доступа: свободный).
3) Wikipedia. Dynamic programming [Электронный ресурс] //URL: https://en.wikipedia.org/wiki/Dynamic_programming (Дата обращения: 13.12.2015, режим доступа: свободный).
4) Wikipedia. Computational complexity theory [Электронный ресурс] //URL: https://en.wikipedia.org/wiki/Computational_complexity_theory (Дата обращения: 13.12.2015, режим доступа: свободный).
5) Youtube. Dynamic programming: 0/1 Knapsack problem [Электронный ресурс] //URL: https://www.youtube.com/watch?v=PLJHuErj-Tw (Дата обращения: 13.12.2015, режим доступа: свободный).
6) Youtube. The Fractional Knapsack Problem – Greedy Algorithms [Электронный ресурс] //URL: https://www.youtube.com/watch?v=kFUs5VUxO-s (Дата обращения: 13.12.2015, режим доступа: свободный).