Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ Фундаментальные науки

КАФЕДРА Прикладная математика

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К УЧЕБНО-ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ

НА ТЕМУ:

Задача об ограниченном ранце

Студент ФН2-42Б (Группа) _____ (Подпись, дата) Г. А. Швецов (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) А. Ю. Попов (И. О. Фамилия)

2022 г.

Оглавление

Введение

Классическая задача о рюкзаке (о загрузке) известна очень давно, и формулируется следующим образом. Пусть есть n разных предметов, каждый предмет имеет вес w_i и полезность c_i, так же имеется максимальный вес W, который можно положить в рюкзак. Требуется собрать такой набор предметов, чтобы полезность их была наибольшей, а суммарный вес не превышал W. Несмотря на такую "бытовую" формулировку, решение данной задачи находит гораздо более широкое применение.

Задача о загрузке (или задача о рюкзаке) и различные ее модификации широко применяются на практике в прикладной математике, криптографии, экономике, логистике, для нахождения решения оптимальной загрузки различных транспортных средств: самолетов, кораблей, железнодорожных вагонов и т.д. Приведем в качестве примера задачу, которая представляет собой переиначенную задачу о ранце, чтобы показать, что алгоритмы для решения данной проблемы могут быть полезны не только в упаковке багажа:

Технология *DVS*[1] позволяет снижать напряжение на процессоре и добиваться экономии электроэнергии за счет увеличения времени выполнения задачи.

Пусть процессор поддерживает два уровня напряжения — $U_L < U_H$. Есть набор из n задач, каждая из которых имеет энергоемкость и время выполнения для обоих режимов, т.е. для $i = 1, \ldots, n$ даны энергоемкости c_i^H и c_i^L и длительности t_i^H и t_i^L.

Нужно выполнить все задачи на одном процессоре за время не более заданного T, при этом добиться минимального энергопотребления.

Рассматриваемая нами задача является *NP*-полной, то есть для нее не существует полиномиального алгоритма, решающего ее за разумное время, в этом и заключается основная проблема данной задачи. Решения *NP*-полных задач обычно делят на два типа: приближенные, работающие быстро, но далеко не всегда решающие задачу наилучшим образом, и точные, выполняемые сильно дольше, но гарантированно выдающие лучший результат.

Существует несколько модификаций задачи о ранце. Ниже приведены некоторые из них.

1) Каждый предмет можно брать только один раз.
2) Каждый предмет можно брать сколько угодно раз.
3) Каждый предмет можно брать определенное количество раз.
4) Можно брать дробную часть предмета.

[1] DVS (от англ. Dynamic Voltage Scaling) — динамическое изменение напряжения.

Цель данной работы — реализовать решение для задачи 3) двумя точными алгоритмами, а именно методом ветвей и границ и динамическим программированием, на языке программирования C++ и в системе компьютерной алгебры Wolfram Mathematica. Также исследовать время выполнения и сложность алгоритмов.

1. Постановка задачи

Математически задачу об ограниченном рюкзаке можно представить следующим образом:

$$\begin{gathered} \sum_{i=1}^{n} c_i x_i \to \max, \\ \sum_{i=1}^{n} w_i x_i \leqslant W, \\ x_i \in \{0, 1, \ldots, k_i\}, \end{gathered} \tag{1}$$

где c_i, w_i и k_i — стоимость, вес и максимальное количество предмета i соответственно; W — максимальная грузоподъемность ранца; n — количество предметов.

Очевидно, что сложность алгоритма, осуществляющего полный перебор, т.е. проверяющего всевозможные наборы предметов, пропорциональна количеству этих возможные наборов. Таким образом он имеет сложность $O(K^n)$, где $K = \max\limits_{i=1,\ldots,n} k_i$, что очень неэффективно. Поэтому, как и во многих других задачах, есть методы решения задачи, которые выполняются быстрее обычного перебора.

2. Метод ветвей и границ

Метод ветвей и границ (МВГ) относится к числу основных методов решения задач оптимизации. Этот метод основан на древовидной декомпозиции исходной задачи на подзадачи.

В процессе дальнейшего изложения мы будем употреблять термин рекорд, означающий наилучшее найденное значение целевой функции (функции подлежащей оптимизации) к данному шагу алгоритма. Предполагается, что рекорд, также как и целевая функция, принимает действительные значения.

Различные варианты МВГ имеют специфичные для решаемой задачи особенности, но при этом подчиняются следующей общей схеме [1]:

Данные: рекорд, список подзадач.

Шаг 1. В список подзадач помещается исходная задача.

Шаг 2. Если список подзадач пуст, то завершить алгоритм. В противном случае из списка выбирается и удаляется подзадача P.

Шаг 3. Вычисляется значение целевой функции, и при необходимости обновляется значение рекорда. Для задачи P проверяется выполнимость условия отсева. Если подзадача P удовлетворяет условию отсева, то осуществляется переход к шагу 2.

Шаг 4. Задача P подвергается декомпозиции. Полученные в результате подзадачи помещаются в список подзадач. Перейти к шагу 2.

Т.е. конкретный вариант метода ветвей и границ определяется следующими тремя функциями: выбором подзадачи из списка; условием отсева; правилом декомпозиции, определяющим, каким образом разбивается очередная подзадача.

Опишем данные функции для задачи об ограниченном ранце.

Для начала запишем линейную задачу релаксации (т.е. задачу с ослабленным условием), соответствующей задаче (1):

$$\begin{gathered}\sum_{i=1}^{n} c_i x_i \to \max, \\ \sum_{i=1}^{n} w_i x_i \leqslant W, \\ 0 \leqslant x_i \leqslant k_i.\end{gathered} \tag{2}$$

Оптимум (оптимальное значение) задачи (2) не меньше оптимума исходной задачи (1). Задача релаксации представляет собой одномерную задачу линейного программирования и может быть решена *методом Данцига*[1] следующим образом. Переменные нумеруются в порядке возрастания удельной стоимости:

$$\frac{c_1}{w_1} \geqslant \ldots \geqslant \frac{c_n}{w_n}.$$

Сначала определяется номер s дробной переменной по следующему правилу:

$$s = \min\left\{ j \in \{1, \ldots, n\} : \sum_{i=1}^{j} k_i w_i > W \right\}.$$

Таким образом решение состоит из предметов $i = 1, \ldots, (s-1)$ в количестве k_i и предмета s в количестве

$$K = \frac{W - \sum\limits_{i=1}^{s-1} k_i w_i}{w_s},$$

[1] Джордж Бернард Да́нциг (англ. George Bernard Dantzig; 8 ноября 1914 – 13 мая 2005) — американский математик, основоположник линейного программирования, известен как разработчик алгоритма, применяемого в решениях задач симплекс-методом.

а стоимость итогового набора задачи (2)

$$S = \sum_{i=1}^{s-1} k_i c_i + K \cdot c_s.$$

Данный алгоритм имеет сложность $O(n)$. На основании данного решения введем функцию $S(i, w)$, решающую аналогичную задачу для предметов $j = i, \ldots, n$ и вместимости рюкзака w, т.е. S, определенное выше, для задачи (2) есть значение $S(1, W)$.

Теперь сформулируем, что будем понимать под подзадачей в рамках загрузки рюкзака. Пусть мы зафиксировали количество у первых m предметов. Данный набор имеет стоимость C' и вес W'. Тогда под подзадачей будем называть следущее:

$$\begin{gathered} C' + \sum_{i=m+1}^{n} c_i x_i \to \max, \\ \sum_{i=m+1}^{n} w_i x_i \leqslant W - W', \\ x_i \in \{0, 1, \ldots, k_i\} \text{ при } i = (m+1), \ldots, n. \end{gathered} \tag{3}$$

Теперь мы готовы, наконец, сформулировать три функции, характеризующие метод ветвей и границ.

Правило декомпозиции. Исходную подзадачу (3) будем разбивать на $(k_{m+1} + 1)$ подзадач, фиксируя в каждой новой такой задаче предмет $(m + 1)$ в количестве $0, 1, 2, \ldots, k_{m+1}$ штук. Отметим, что если в подзадаче (3) присутствует всего один предмет n, то нет смысла разбивать ее на подзадачи, потому что мы можем сразу дать на нее ответ.

Выбор подзадачи из списка. В основном, выбор подзадачи можно сделать произвольным, но мы будем пользоваться логикой жадных алгоритмов. Чтобы задача релаксации выполнялась за $O(n)$ необходимо, чтобы предметы изначально были отсортированы по убыванию удельной стоимости, поэтому будем предполагать, что предметы изначально отсортированы. Таким образом можно предположить, что нам выгодней взять как можно больше первых предметов. Поэтому после декомпозиции мы сначала будем рассматривать задачу при условии, что мы взяли предмет $(m+1)$ в количестве k_{m+1}, потом $(k_{m+1} - 1)$ и т.д.

Условие отсева. Подзадача не подлежит дальнейшей декомпозиции если:

1) подзадача не имеет решения, что возможно только при условии $W' < 0$;
2) оптимум задачи (2) не превосходит значения рекорда (тогда, как было отмечено выше, оптимум задачи (3) тоже не превосходит значения рекорда).

Сложность алгоритма метода ветвей и границ составляет, как и при использовании перебора, $O(K^n)$, где $K = \max\limits_{i=1,\dots,n} k_i$, т.к. в худшем случае условие отсева не сработает ни разу и мы будем вынуждены рассмотреть все возможные варианты.

3. Реализация метода ветвей и границ

3.1. Особенности реализации на языке C++

Стандартная библиотека C++ предоставляет удобный контейнер для хранения и обработки различных структур данных — std::vector<T>. Для удобства представления предметов рюкзака создана структура Item в отдельном заголовочном файле Item.h. При инициализации она получает имя предмета (name), его количество (count), вес (weight) и стоимость (cost).

Решение исходной задачи методом перебора представлено для пользователя функцией BranchAndBoundMethod::problemSolve, принимающей на вход вектор предметов items и вместимость рюкзака max_weight. Задача данной функции подготовить данные для основной функции МВГ find_bests, а именно отсортировать список предметов items по убыванию удельной стоимости.

Функция find_bests, принимающая на вход вместимость рюкзака и два итератора: на первый и последний рассматриваемые предметы. В целом она повторяет расписанный ход действий, описанный в предыдущем разделе. Для оценки сверху, т.е. условия отсева, реализована функция find_best_greedily, так же использующая два итератора и вместимость в качестве входных данных и реализующая метод Данцига.

Также в процессе работы были реализованы вспомогательные функции для отладки и работы с файлами: generateItems — генерирует случайный набор предметов, FileToVector — читает из файла массив, VectorToFile — соответственно записывает массив в файл.

3.2. Особенности реализации на Wolfram Mathematica

Встроенные возможности системы Wolfram Mathematica значительно упрощают решение данной задачи, в особенности при обработке данных. Для работы с файлами использовались в основном такие команды, как SetDirectory (задание директории) и Import, для работы с List — команды AppendTo, MapIndexed и SortBy (добавление в конец, преобразование с учетом индекса и сортировка по правилу соответственно), для замера времени — Timing. Кроме того, использовались программные блоки (Block) для реализации алгоритма с созданием временных переменных.

Также были реализованы пользовательские функции findBestGreedly и findBest, с помощью которых осуществляются оценка сверху и поиск решения соответственно. Метод ветвей и границ реализован в виде пользовательской функции babMethod, принимающей на вход вместимость рюкзака (maxWeight) и список предметов (items).

4. Метод динамического программирования

Словосочетание "динамическое программирование" впервые было использовано в 1940-х годах *Ричардом Беллманом*[1] для описания процесса нахождения решения задачи, где ответ на одну задачу может быть получен только после решения задачи, "предшествующей" ей. В 1953 году он уточнил это определение до современного. Вклад Беллмана в динамическое программирование был увековечен в названии уравнения Беллмана, центрального результата теории динамического программирования, который переформулирует оптимизационную задачу в рекурсивной форме.

Слово "программирование" в словосочетании "динамическое программирование" в действительности к "традиционному" программированию (написанию кода) почти никакого отношения не имеет. Слово "программа" в данном контексте скорее означает оптимальную последовательность действий для получения решения задачи. К примеру, определенное расписание событий на выставке иногда называют программой.

Динамическое программирование — способ решения сложных задач путем разбиения их на более простые подзадачи. Ключевая идея в динамическом программировании достаточно проста. Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи (подзадачи), после чего объединить решения подзадач в одно общее решение. Часто многие из этих подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений. Это особенно полезно в случаях, когда число повторяющихся подзадач экспоненциально велико.

Чтобы успешно решить задачу динамикой нужно [2]:

1) состояние динамики: параметры, однозначно задающие подзадачу;
2) значения начальных состояний;
3) переходы между состояниями: формула пересчета;
4) порядок пересчета;

[1] Ричард Эрнест Бе́ллман (26 августа 1920, Нью-Йорк, США – 19 марта 1984, Лос-Анджелес, США) — американский математик, один из ведущих специалистов в области математики и вычислительной техники. Член Национальной инженерной академии США (1977), Национальной академии наук США (1983).

5) положение ответа на задачу: иногда это сумма или, например, максимум из значений нескольких состояний.

Для решения нашей задачи введем двумерный массив dp размером $n+1$ на $W+1$, где в $dp[i, w]$ будем хранить максимально возможную стоимость набора состоящая из первых i предметов при максимальной допустимой нагрузке w. Теперь распишем пункты выше.

1) состояние динамики: $dp[i, w]$;
2) начальные состояния: $dp[0, w]$ и $dp[i, 0]$ равны нулю, т.е. если у нас нет никаких вещей или мы не можем ничего положить в рюкзак, то и максимально возможная стоимость равна нулю;
3) формула пересчета: $dp[i, w] = \max\limits_{x_j=0,1,\ldots,m(i,w)} (x_j \cdot c_i + dp[i-1, w - x_j \cdot w_i])$, где $m(i, w)$ — максимальное количество предмета i, которое влезает в рюкзак вместимости w, ограниченное при этом максимальным количеством k_i, таким образом $m(i, w) = \min\left\{k_i;\ \left\lfloor \frac{w}{w_i} \right\rfloor\right\}$;
4) пересчитывать будем в следующем порядке: для каждого i будем заполнять строку $dp[i, w]$, w изменяется в диапазоне от 1 до W, т.е. постепенно будем добавлять каждый предмет по порядку;
5) ответ будет находится в ячейке $dp[n, W]$, т.к. это именно та задача, которую нам надо решить, для всех n предметов при емкости рюкзака W.

Таким образом, заполнение массива dp можно представить в виде следующего псевдокода [6]:

1: **for** $w = 0$ **to** W **do**
2: $\quad dp[0, w] = 0$
3: **for** $i = 1$ **to** n **do**
4: $\quad$ **for** $w = 1$ **to** W **do**
5: $\quad\quad dp[i, w] = dp[i-1, w]$
6: $\quad\quad$ **for** $x_j = \min\left\{k_i, \left\lfloor \frac{w}{w_i} \right\rfloor\right\}$ **downto 1 do**
7: $\quad\quad\quad dp[i, w] = \max\left\{dp[i, w];\ x_j \cdot c_i + dp[i-1, w - x_j \cdot w_i]\right\}$

Такое решение работает за $O(n\,W^2)$, так как k_i могут быть очень большими, а $w_i = 1$.

Теперь опишем восстановление одного из решений также в виде псевдокода.

1: $w = W$
2: $solution = \{\ \}$
3: **for** $i = n$ **downto** 1 **do**
4: $\quad$ **if** $w = 0$ **then**
5: $\quad\quad$ **break**

6: Находим $mVal = \max\limits_{x_j=0,1,\ldots,m(i,w)} (x_j \cdot c_i + dp[i-1, w - x_j \cdot w_i])$ и $mCnt = x_j$, на котором достигается $mVal$
7: **if** $mCnt \neq 0$ **then**
8: Добавляем в решение *solution* предмет i в количестве $mCnt$ штук
9: $w \mathrel{-}= w_i \cdot mCnt$

Впоследствии выполнения алгоритма итоговое решение будет представлено массивом *solution*. Сложность алгоритма восстановления одного решения составляет $O(n)$, потому что в общем случае мы пройдем по всем n предметам.

В итоге алгоритм имеет временную сложность $O(n\,W^2)$, а по памяти $O(n\,W)$, которая требуется на хранение массива *dp*. Также заметим, что если требуется найти только максимальную стоимость набора, не восстанавливая само решение, то потребление памяти можно сократить до $O(W)$, храня только две последние рассматриваемые строчки.

5. Реализация метода динамического программирования

5.1. Особенности реализации на языке C++

Реализация полностью совпадает с псевдокодом алгоритма, указанного в предыдущем разделе. Но благодаря широким возможностям управления памятью в языке программирования C++ удалось увеличить быстродействие и уменьшить потребление памяти. Некоторые вспомогательные функции, например, для работы с файлами и генерации предметов были заимствованы из реализации метода ветвей и границ на языке C++.

5.2. Особенности реализации на Wolfram Mathematica

Как и в случае с МВГ, возможности системы очень полезны при выполнении данной задачи. Встроенные команды позволили значительно сократить объем написанного кода, устранив необходимость в создании большого числа пользовательских функций. Большая часть команд, указанных в описании реализации метода перебора, нашли применение и в данном алгоритме. В процессе выполнения практики были реализованы пользовательские функции, необходимые для данного метода, такие как dynamic и dynFindAns, выполняющие заполнение матрицы *dp* и восстановление решений соответственно.

6. Примеры работы программ

Приведем некоторые примеры решения задачи о рюкзаке на языке программирования C++ и в системе компьютерной алгебры Wolfram Mathematica. Исходные данные и результаты выполнения представлены в таблицах (1), (2) и (3). Названия заголовков таблиц имеют тот же смысл, что и в описании особенностей реализации метода ветвей и границ на языке программирования C++. Первый пример представляет собой задачу ранца 0-1, т.к. количество каждого предмета ограничено единицей. Дополнительно отметим, что задача, описанная в таблице (3), имеет два решения.

Таблица 1. Пример при вместимости рюкзака max_weight = 1500

Исходные данные			
item	count	weight	cost
1	1	200	40
2	1	314	50
3	1	198	100
4	1	500	60
5	1	200	30
6	1	400	45

Решение			
item	count	max_cost	sum_weight
5	1	265	1412
6	1		
2	1		
1	1		
3	1		

Таблица 2. Пример при max_weight = 1000

Исходные данные			
item	count	weight	cost
Apple	4	39	40
Banana	4	27	60
Beer	12	52	10
Book	2	30	10
Camera	1	32	30
Cheese	4	23	30
Chocolate Bar	10	15	60
Compass	1	13	35
Jeans	1	48	10
Map	1	9	150
Notebook	1	22	80
Sandwich	4	50	160
Ski Jacket	1	43	75
Ski Pants	1	42	70
Socks	2	4	50
Sunglasses	1	7	20
Suntan Lotion	1	11	70
T-Shirt	1	24	15
Tin	1	68	45
Towel	1	18	12
Umbrella	1	73	40
Water	1	153	200

Решение			
item	count	max_cost	sum_weight
T-Shirt	1	2535	999
Apple	3		
Cheese	4		
Water	1		
Ski Pants	1		
Ski Jacket	1		
Banana	4		
Compass	1		
Sunglasses	1		
Sandwich	4		
Notebook	1		
Chocolate Bar	10		
Suntan Lotion	1		
Socks	2		
Map	1		

Таблица 3. Пример при max_weight = 1000

Исходные данные			
item	count	weight	cost
1	6	54	8
2	5	63	12
3	4	55	2
4	3	63	7
5	8	97	6
6	6	26	11
7	4	91	16
8	7	42	2
9	5	8	18
10	6	18	3

Решение 1			
item	count	max_cost	sum_weight
10	6	298	983
7	4		
2	5		
6	6		
9	5		
Решение 2			
item	count	max_cost	sum_weight
1	2	298	1000
10	6		
7	3		
2	5		
6	6		
9	5		

Также ниже приведем наглядные графики времени исполнения обоих методов на языке программирования C++ в зависимости от количества предметов (N — количество предметов, t — время в секундах). Предметы генерировались описанной ранее функцией generateItems (count$\leqslant$10, weight$\leqslant$ 50, cost$\leqslant$ 100).

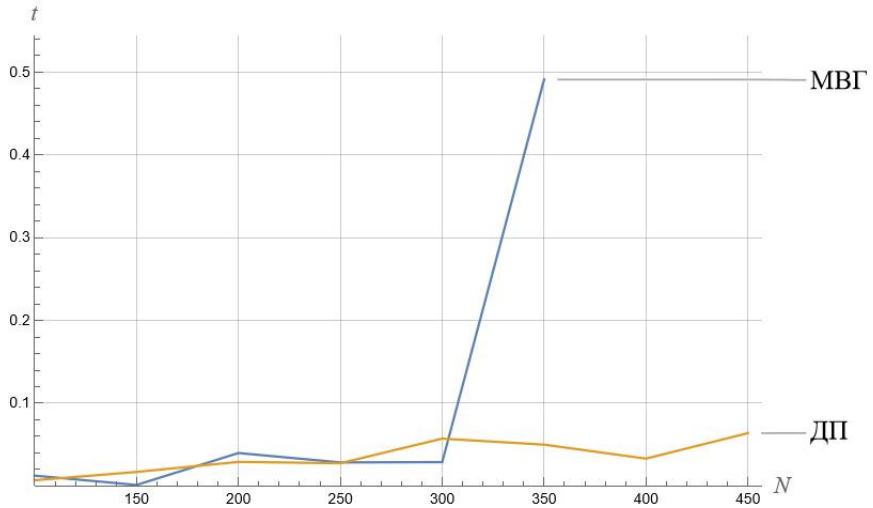

Рис. 1. max_weight=1000

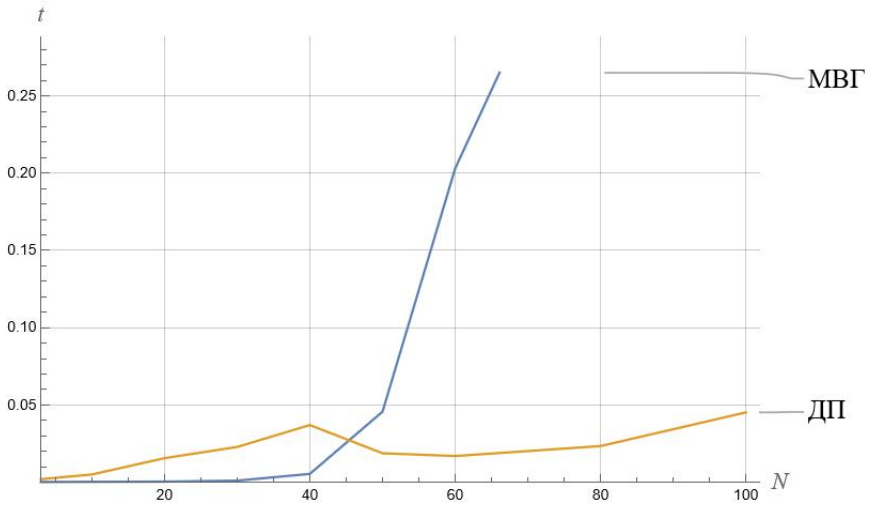

Рис. 2. max_weight=3000

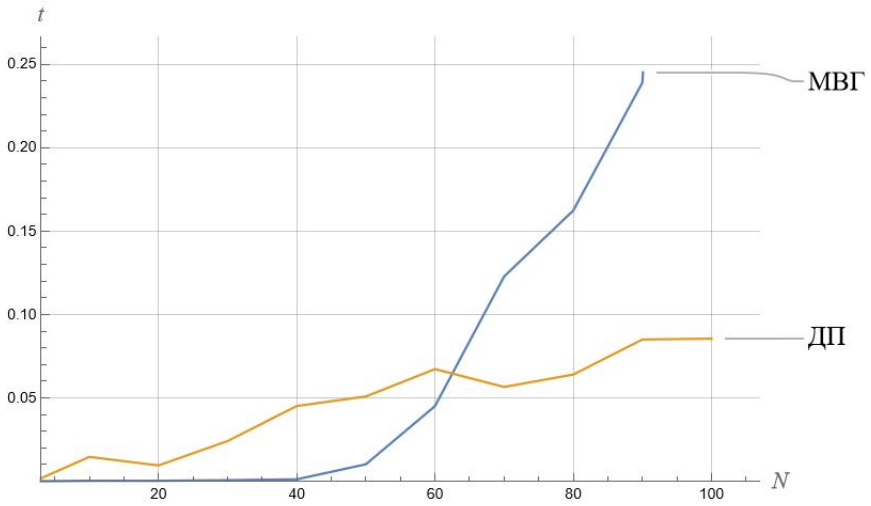

Рис. 3. max_weight=5000

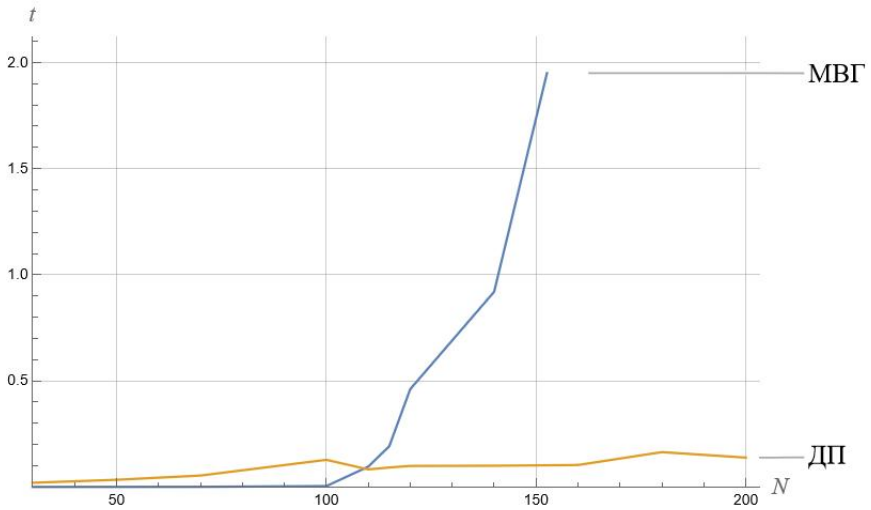

Рис. 4. max_weight=10000

Заключение

В ходе практики были изучены основные возможности языка программирования C++ и системы компьютерной алгебры "Wolfram Mathematica" , закреплены знания и умения, полученные в рамках курсов "Введение в информационные технологии" , "Информационные технологии профессиональной деятельности" . Были изучены методы решения задачи об ограниченном ранце. Также изучены возможности системы компьютерной алгебры "Wolfram Mathematica" в сфере визуализации и построения алгоритмов.

Список использованных источников

1. Р.М. Колпаков, М.А. Посыпкин. Верхняя и нижняя оценки трудоемкости метода ветвей и границ для задачи о ранце, Тр. ИСА РАН, 32, 2008. С. 137—158. URL: http://www.isa.ru/proceedings/images/documents/2008-32/137-158.pdf (дата обращения: 23.04.2022).
2. Всё, что вы хотели знать о динамическом программировании, но боялись спросить // Хабр. [Электронный ресурс] URL: https://habr.com/ru/post/191498/.
3. Задача о рюкзаке // Википедия. Свободная энциклопедия. [Электронный ресурс] URL: https://ru.m.wikipedia.org/wiki/Задача_о_рюкзаке (дата обращения: 22.03.2022).
4. Метод ветвей и границ // Википедия. Свободная энциклопедия. [Электронный ресурс] URL: https://ru.m.wikipedia.org/wiki/Метод_ветвей_и_границ (дата обращения: 15.04.2022).
5. Динамическое программирование // Википедия. Свободная энциклопедия. [Электронный ресурс] URL: https://ru.m.wikipedia.org/wiki/Динамическое_программирование (дата обращения: 25.04.2022).
6. Задача о рюкзаке // Университет ИТМО. [Электронный ресурс] URL: http://neerc.ifmo.ru/wiki/index.php?title=Задача_о_рюкзаке (дата обращения: 28.04.2022).
7. Bounded Knapsack Algorithm [Электронный ресурс] / Givens B. URL: https://www.codeproject.com/Articles/706838/Bounded-Knapsack-Algorithm (дата обращения: 4.05.2022).