Оглавление

[Введение 2](#_Toc1718063303)

[Глава 1. Теоретическая часть 3](#_Toc1967731011)

[1.1 Метод динамического программирования 3](#_Toc342497955)

[1.2 Метод ветвей и границ 3](#_Toc800326650)

[Глава 2. Задача. Описание алгоритма решения. 5](#_Toc1721797657)

[2.1 Метод динамического программирования 5](#_Toc815076601)

[2.2 Метод ветвей и границ 7](#_Toc836806419)

[Глава 3. Решение задачи. 8](#_Toc1733967278)

[3.1 Описание класса Knapsack 8](#_Toc2095277813)

[3.2 Решение методом динамического программирования 9](#_Toc308752595)

[3.3 Решение методом ветвей и границ 11](#_Toc259144158)

[3.3.1 Определение узлов и построение связей 13](#_Toc651463580)

[Глава 4. Результаты запуска кода. 14](#_Toc782094473)

[4.1 Динамическое программирование 14](#_Toc1108356817)

[4.1.1 Датасет №1 14](#_Toc1371613690)

[4.1.2 Датасет №2 16](#_Toc1060391286)

[4.2 Ветви и границы 17](#_Toc317912468)

[4.2.1 Датасет №1 17](#_Toc458341469)

[4.2.2 Датасет №2 19](#_Toc831667645)

[Заключение. 22](#_Toc331513207)

[Список литературы 23](#_Toc221994282)

## **Введение**

Теория сложности вычислений — подраздел теоретической информатики, занимающейся исследованием сложности алгоритмов для решения задач на основе формально определённых моделей вычислительных устройств.

Задача о рюкзаке (*англ. Knapsack problem*) — дано *N* предметов,

предмет имеет массу  > 0 и стоимость  >0. Необходимо выбрать из этих предметов такой набор, чтобы суммарная масса не превосходила заданной величины *W* (вместимость рюкзака), а суммарная стоимость была максимальна.

# **Глава 1. Теоретическая часть**

## **Метод динамического программирования**

Динамическое программирование в теории управления и теории вычислительных систем — способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В этом случае время вычислений, по сравнению с «наивными» методами, можно значительно сократить.

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.

## **Метод ветвей и границ**

Метод ветвей и границ — общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. Метод является развитием метода полного перебора, в отличие от последнего — с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Общая идея метода может быть описана на примере поиска минимума функции f ( x ) на множестве допустимых значений переменной x . Функция и переменная x xxxмогут быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Процедура ветвления состоит в разбиении множества допустимых значений переменной на подмножества меньших размеров. Процедуру можно рекурсивно применять к подмножествам. Полученные подмножества образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подмножества множества значений переменной .

Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти допустимых значений переменной .

В основе метода ветвей и границ лежит следующая идея: если нижняя граница значений функции на подобласти дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти , то может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно минимальную из полученных верхних оценок записывают в глобальную переменную .  
Любой узел дерева поиска, нижняя граница которого больше значения , может быть исключён из дальнейшего рассмотрения.

Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

## **Глава 2. Задача. Описание алгоритма решения.**

В рамках курсовой работы требуется реализовать задачу о рюкзаке, а именно найти множество предметов с максимально возможной суммой значений, при условии, что суммарный размер предметов не превосходит заданного .

## **Метод динамического программирования**

Пусть есть максимальная стоимость предметов, которые можно уложить в рюкзак вместимости , если можно использовать только первые  предметов, то есть , назовем этот набор допустимых предметов для .

Найдем . Возможны 2 варианта:

1. Если предмет не попал в рюкзак. Тогда равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов , то есть
2. Если предмет попал в рюкзак. Тогда равно максимальной стоимости рюкзака, где вес уменьшаем на вес -ого предмета и набор допустимых предметов плюс стоимость , то есть

То есть:

Стоимость искомого набора равна , так как нужно найти максимальную стоимость рюкзака, где все предметы допустимы и вместимость рюкзака .

*Восстановим набор предметов, входящих в рюкзак*

Будем определять, входит ли предмет в искомый набор.  
Начинаем с элемента , где . Для этого сравниваем со следующими значениями:

1. Максимальная стоимость рюкзака с такой же вместимостью и набором допустимых предметов , то есть
2. Максимальная стоимость рюкзака с вместимостью на меньше и набором допустимых предметов плюс стоимость , то есть

Заметим, что при построении выбирали максимум из этих значений и записывали в . Тогда будем сравнивать с , если равны, тогда не входит в искомый набор.

Таким образом, хотя сложность этого алгоритма — является экспоненциальной от длины входа, при ограниченных размерах , алгоритм может быть полезен и эффективен.

## **Метод ветвей и границ**

Метод ветвей и границ в отличии от полного перебора каждый предмет может быть добавлен или пропущен — каждая ситуация дает отдельный узел в дереве решения.

В алгоритме полного перебора использовался поиск в глубину, однако метод ветвей и границ применяет поиск в ширину, сравнивая узлы, находящиеся на одном ярусе в дереве поиска. На основании результатов сравнения, некоторые узлы могут быть отброшены как неперспективные.

Функции сравнения узлов называют **граничными функциями**. В задаче о рюкзаке граничные функции могут опираться лишь на объем и цену предметов уже уложенных в рюкзак и всех остальных предметов.  
Предметы вне рюкзака удобно хранить упорядоченными по убыванию цены, тогда граничная функция стоимости может быть записана следующим образом:

Предметы для добавления в рюкзак выбираются в порядке уменьшения их стоимости до тех пор, пока ранец не заполнится.  
PossibleWeight — это объем предметов, которые могут быть полностью помещены в рюкзак.  
Граничная функция (costBound) вычисляется как сумма предметов уже уложенных в рюкзак на предыдущих шагах (currentCost), суммы цен предметов, которые могут быть помещены в рюкзак целиком, а также стоимость части предмета, который полностью в рюкзак не поместился.

Добавление предмета считается неперспективным если его добавление:

* приводит к превышению предельного размена мешка;
* не приводит к улучшению наилучшей границы стоимости, найденной к текущему шагу.

На каждом шаге из очереди (обеспечивающей поиск в ширину) выбирается узел, для которого рассматриваются варианты добавления или пропуска всех необработанных предметов. Для каждого случая рассчитывается оценочная граничная функция, в очередь добавляются лишь те узлы, которые способны улучшить результат.

Оценка сложности —

# Глава 3. Решение задачи.

## **Описание класса Knapsack**

Код 1. Реализация хранения данных о предмете.  
 Item.cs

public class Item  
{  
 public Item(string name, int value, int weight)  
 {  
 Name = name;  
 Value = value;  
 Weight = weight;  
 Console.ForegroundColor = ConsoleColor.*Yellow*;  
 Console.*WriteLine*("Added new item:" +  
 $"\nName: {Name}" +  
 $"\nValue: {Value}" +  
 $"\nWeight: {Weight}");  
 Console.ForegroundColor = ConsoleColor.*White*;  
 }

public int Ratio => Value / Weight;

}

Код 2. Интерфейс общего решения.  
 IKnapsackSolution.cs

public interface IKnapsackSolution  
{  
 string Approach { get; set; }  
  
 IList<Item> Items { get; set; }  
  
 int TotalWeight { get; set; }  
  
 int Value { get; set; }  
}

## **Решение методом динамического программирования**

Код 3. Поле и конструктор решения.   
DynamicProgrammingSolver.cs

private int[,] table;  
  
public DynamicProgrammingSolver(IList<Item> items, int capacity)  
 : base(items, capacity){}

Код 4. Входная точка в метод решения DynamicProgrammingSolver.cs

public override KnapsackSolution Solve()  
{  
 FillTable();  
 var solution = TakeItems();  
 solution.Approach = "Dynamic Programming";  
  
 return solution;  
}

Код 5. Заполнение таблицы решения.   
DynamicProgrammingSolver.cs

private void FillTable()  
{  
 table = new int[Items.Count + 1, Capacity + 1];  
 for (var row = 1; row <= Items.Count; row++)  
 {  
 var item = Items[row - 1];  
 for (var col = 0; col <= Capacity; col++)  
 if (item.Weight > col)  
 table[row, col] = table[row - 1, col];  
 else  
 table[row, col] = Math.*Max*(table[row - 1, col], table[row - 1, col - item.Weight] + item.Value);  
 }  
}

Код 6. Решение задачи рюкзак по средствам востановления набора предметов.  
DynamicProgrammingSolver.cs

private KnapsackSolution TakeItems()  
{  
 var best = new KnapsackSolution { Items = new List<Item>() };  
 for (int row = Items.Count, col = Capacity; row > 0; row--)  
 if (table[row, col] != table[row - 1, col])  
 {  
 best.Items.Add(Items[row - 1]);  
 col -= Items[row - 1].Weight;  
 }  
  
 best.TotalWeight = GetWeight(best.Items);  
 best.Value = GetValue(best.Items);  
 return best;  
}

## **Решение методом ветвей и границ**

Код 7. Конструктор решения.   
BranchAndBoundSolver.cs

public BranchAndBoundSolver(IList<Item> items, int capacity)  
 : base(items, capacity){}

Код 8. Решение задачи рюкзак.   
BranchAndBoundSolver.cs

public override KnapsackSolution Solve()  
{  
 Items = Items.OrderByDescending(i => i.Ratio).ToList();  
 var best = new Node();  
 var root = new Node();  
 root.ComputeBound(Items, Capacity);  
 var queue = new PriorityQueue<Node>();  
 queue.Enqueue(root);  
  
 while (queue.Count != 0){  
 var node = queue.Dequeue();  
 if (node.Bound > best.Value && node.Height < Items.Count - 1){  
 var with = new Node(node);  
 var item = Items[node.Height];  
 with.Weight += item.Weight;  
 if (with.Weight <= Capacity){  
 with.TakenItems.Add(Items[node.Height]);  
 with.Value += item.Value;  
 with.ComputeBound(Items, Capacity);  
 if (with.Value > best.Value) best = with;  
 if (with.Bound > best.Value) queue.Enqueue(with);  
 }  
 var without = new Node(node);  
 without.ComputeBound(Items, Capacity);  
 if (without.Bound > best.Value) queue.Enqueue(without);  
 }  
 }  
 var solution = new KnapsackSolution{  
 Value = best.Value,  
 TotalWeight = best.Weight,  
 Items = best.TakenItems,  
 Approach = "Best-First Search with Branch and Bound"  
 };  
 return solution;  
}

### Определение узлов и построение связей

Код 9. Конструктор узла.  
Node.cs

public void ComputeBound(IList<Item> items, int capacity)  
{  
 double w = Weight;  
 Bound = Value;  
 var index = Height;  
 Item currentItem;  
  
 do  
 {  
 currentItem = items[index];  
 if (w + currentItem.Weight > capacity) break;  
  
 w += currentItem.Weight;  
 Bound += currentItem.Value;  
 index++;  
 } while (index < items.Count);  
  
 Bound += (capacity - w) \* currentItem.Ratio;  
}

}

public class Node : IComparable<Node>  
{  
 public Node()  
 {  
 TakenItems = new List<Item>();  
 }  
  
 public Node(Node parent)  
 {  
 Height = parent.Height + 1;  
 TakenItems = new List<Item>(parent.TakenItems);  
 Bound = parent.Bound;  
 Value = parent.Value;  
 Weight = parent.Weight;  
 }

Код 10. Создание связей на основе входных данных.  
Node.cs

# Глава 4. Результаты запуска кода.

## Динамическое программирование

### Датасет №1

Датасет состоящий из 100 элементов,

DataSize: 100

MaxCapacity: 10000

Max Capacity is 10000

Dynamic Programming | value: 2493893, total weight: 9976

Products:

99 - weight: 153, value: 54680,

98 - weight: 12, value: 46043,

92 - weight: 386, value: 99248,

91 - weight: 540, value: 95273,

90 - weight: 218, value: 97386,

88 - weight: 160, value: 36551,

82 - weight: 95, value: 35239,

79 - weight: 127, value: 48906,

77 - weight: 224, value: 83455,

76 - weight: 320, value: 49096,

75 - weight: 99, value: 97973,

74 - weight: 7, value: 37336,

71 - weight: 58, value: 14739,

68 - weight: 713, value: 85696,

65 - weight: 366, value: 97943,

61 - weight: 9, value: 41595,

58 - weight: 400, value: 98143,

57 - weight: 299, value: 53636,

56 - weight: 176, value: 59746,

53 - weight: 398, value: 53178,

49 - weight: 145, value: 54439,

45 - weight: 124, value: 82189,

42 - weight: 196, value: 46518,

41 - weight: 502, value: 75630,

40 - weight: 95, value: 24116,

36 - weight: 14, value: 28636,

31 - weight: 25, value: 28550,

28 - weight: 197, value: 88125,

25 - weight: 154, value: 35002,

24 - weight: 358, value: 40195,

23 - weight: 92, value: 25229,

18 - weight: 150, value: 80980,

15 - weight: 827, value: 99098,

14 - weight: 336, value: 51817,

13 - weight: 279, value: 93100,

10 - weight: 170, value: 44304,

9 - weight: 613, value: 78841,

8 - weight: 730, value: 82328,

6 - weight: 43, value: 84493,

5 - weight: 166, value: 64441

Elapsed = 00:00:00.0347789

### Датасет №2

Данное решение не подходит для данного датасета, так как размер данных слишком велик и происходит переполнение стека.

## Ветви и границы

### Датасет №1

Датасет состоящий из 100 элементов,

Max Capacity is 10000

Best-First Search with Branch and Bound | value: 2493893, total weight: 9976

Products:

74 - weight: 7, value: 37336,

61 - weight: 9, value: 41595,

98 - weight: 12, value: 46043,

36 - weight: 14, value: 28636,

6 - weight: 43, value: 84493,

31 - weight: 25, value: 28550,

75 - weight: 99, value: 97973,

45 - weight: 124, value: 82189,

18 - weight: 150, value: 80980,

28 - weight: 197, value: 88125,

90 - weight: 218, value: 97386,

5 - weight: 166, value: 64441,

79 - weight: 127, value: 48906,

49 - weight: 145, value: 54439,

77 - weight: 224, value: 83455,

82 - weight: 95, value: 35239,

99 - weight: 153, value: 54680,

56 - weight: 176, value: 59746,

13 - weight: 279, value: 93100,

23 - weight: 92, value: 25229,

65 - weight: 366, value: 97943,

10 - weight: 170, value: 44304,

92 - weight: 386, value: 99248,

71 - weight: 58, value: 14739,

40 - weight: 95, value: 24116,

58 - weight: 400, value: 98143,

42 - weight: 196, value: 46518,

88 - weight: 160, value: 36551,

25 - weight: 154, value: 35002,

57 - weight: 299, value: 53636,

91 - weight: 540, value: 95273,

14 - weight: 336, value: 51817,

76 - weight: 320, value: 49096,

41 - weight: 502, value: 75630,

53 - weight: 398, value: 53178,

9 - weight: 613, value: 78841,

68 - weight: 713, value: 85696,

15 - weight: 827, value: 99098,

8 - weight: 730, value: 82328,

24 - weight: 358, value: 40195

Elapsed = 00:00:00.0196440

### Датасет №2

Датасет состоящий из 2000 элементов**,**

DataSize: 2000

MaxCapacity: 2000000

Max Capacity is 2000000

Best-First Search with Branch and Bound | value: 3915650, total weight: 1993544

Products:

365 - weight: 9542, value: 82858,

1548 - weight: 12111, value: 93233,

356 - weight: 11744, value: 74087,

1980 - weight: 12478, value: 83813,

75 - weight: 17616, value: 97973,

467 - weight: 14680, value: 75515,

767 - weight: 8808, value: 47161,

1973 - weight: 8074, value: 41077,

619 - weight: 12111, value: 55642,

667 - weight: 15414, value: 63665,

1057 - weight: 11010, value: 52264,

1820 - weight: 17249, value: 77149,

1853 - weight: 18717, value: 82924,

64 - weight: 21653, value: 71007,

468 - weight: 29360, value: 95020,

498 - weight: 21653, value: 72113,

1342 - weight: 18717, value: 59636,

97 - weight: 23855, value: 61626,

262 - weight: 28993, value: 85007,

308 - weight: 29727, value: 80847,

391 - weight: 26424, value: 59045,

724 - weight: 17983, value: 39055,

945 - weight: 32296, value: 68184,

1008 - weight: 23488, value: 51621,

1163 - weight: 10643, value: 31507,

1190 - weight: 40003, value: 98471,

1531 - weight: 23488, value: 61929,

1607 - weight: 32296, value: 65776,

1713 - weight: 31562, value: 94548,

1831 - weight: 28259, value: 63157,

1905 - weight: 16148, value: 48418,

204 - weight: 86245, value: 92041,

231 - weight: 44774, value: 44911,

266 - weight: 49178, value: 80109,

285 - weight: 30461, value: 49819,

290 - weight: 80006, value: 86541,

303 - weight: 25323, value: 33980,

394 - weight: 58720, value: 95732,

411 - weight: 58353, value: 61421,

417 - weight: 41104, value: 77449,

422 - weight: 44407, value: 81080,

445 - weight: 74134, value: 75252,

485 - weight: 48811, value: 69652,

512 - weight: 55417, value: 66901,

524 - weight: 11010, value: 14879,

533 - weight: 31195, value: 52126,

595 - weight: 40370, value: 66538,

634 - weight: 39269, value: 59830,

641 - weight: 86979, value: 89962,

676 - weight: 78905, value: 97807,

690 - weight: 59821, value: 60596,

774 - weight: 65693, value: 99905,

854 - weight: 83309, value: 87445,

868 - weight: 61656, value: 84687,

871 - weight: 75235, value: 97318,

955 - weight: 23855, value: 39195,

1825 - weight: 13212, value: 16146

Elapsed = 00:00:00.0007373

# Заключение.

В данное работе я продемострировал, оценил и сравнил работу двух алгоримтов решения задачи Рюкзак. Лучше изучил подход к проектирования архитектуры кода и оптимизации обработки данных. Получил опыт в разработке алгоритмов и оценки их сложности.

# Список литературы

1. Совершенный алгоритм. Тим Рафгарден. Тома 1-4. СПб. - Питер, 2022
2. Алгоритмы. С.Дасгупта, Х.Пападимитриу, У.Вазирани. М.- МЦНМО, 2014
3. Алгоритмы. Построение и анализ. Т.Корман, Ч.Лейзерсон и др. Диалектика-Вильямс, М., СПб. - 2020
4. Лекции по алгоритмам и структурам данных. С.Л Бабичев

https://www.babichev.org › books › AlgoBook

1. Грокаем алгоритмы. А.Бхаргава. СПб. - Питер, 2022