МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"
РТУ МИРЭА

Институт искусственного интеллекта

Кафедра высшей математики

КУРСОВАЯ РАБОТА

по дисциплине

«Алгоритмы и теория сложности»

Тема курсовой работы

«Задача о рюкзаке. Вычисление максимальной суммы значений предметов (Knapsack)»

Студент группы КМБО-03-22 *Лыков Д.С.*

Руководитель курсовой работы *Драгилева И.П.*

Работа представлена к защите «__»_____2024 г.

(подпись студента)

«Допущен(ы) к защите» «__»_____2024 г.

(подпись руководителя)

МОСКВА – 2024

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"
РТУ МИРЭА

Институт искусственного интеллекта

Кафедра высшей математики

Утверждаю
Заведующий
кафедрой_____Шатина А.В.

«_2_» _декабря__2024г.

ЗАДАНИЕ
на выполнение курсовой работы
по дисциплине «Алгоритмы и теория сложности»

Студент Лыков Д.С. Группа *КМБО-03-22.*

1. **Тема: «Задача о рюкзаке. Вычисление максимальной суммы значений предметов (Knapsack)»**

2. Исходные данные: тестовые примеры и наборы данных для испытаний

3. Перечень вопросов, подлежащих обработке, и обязательного графического материала:

1) Алгоритм(ы) решения задачи на языке высокого уровня.
2) Получение и сравнение количественных результатов при различных исходных данных.
3) Асимптотические оценки временной сложности.

4. Срок представления к защите курсовой работы: до «20» декабря 2024 г.

Задание на курсовую работу выдал «1» октября 2024г. _____ (_____)
Задание на курсовую работу получил «___»_____2024г. _____ (_____)

Оглавление

Введение

Задача о рюкзаке является одной из классических задач комбинаторной оптимизации и находит широкое применение в различных областях, таких как логистика, экономика, управление запасами и планирование ресурсов. Суть задачи состоит в выборе множества предметов с определённой стоимостью и весом таким образом, чтобы максимизировать суммарную стоимость предметов, находящихся в рюкзаке, не превышая при этом его весовую вместимость.

Эта задача принадлежит к классу NP-полных задач, что делает её вычислительно сложной при увеличении количества предметов. На практике для её решения используют различные методы: от простого полного перебора до сложных алгоритмов динамического программирования и эвристических подходов.

Цель данной курсовой работы – рассмотреть основные подходы к решению задачи о рюкзаке и реализовать алгоритмы, которые позволяют эффективно находить максимальную стоимость предметов в условиях ограниченной вместимости рюкзака. В рамках работы будет проведен сравнительный анализ различных алгоритмов, а также исследованы особенности их применения на практике.

Постановка задачи

Дано:

N – кол-во предметов, C – емкость рюкзака,

$0 \leq v_1, v_2, v_3, \ldots, v_n$ – стоимости предметов

$0 \leq s_1, s_2, s_3, \ldots, s_n$ – размеры предметов

Найти: $S \subseteq \{1,2, \ldots, n\}: \sum_{i \in S} v_i \rightarrow max$ при $\sum_{i \in S} s_i \leq C$

Предлагается *проверить решение задачи* для двух наборах данных:

- Тестовый пример (ограничение по весу: 10000, кол-во предметов: 100)
- Усложненный набор данных (ограничение по весу: 2.000.000, кол-во предметов: 2000)

В усложненном наборе данных достаточно большое ограничение по весу и большое кол-во предметов. Для решения этой задачи потребуется применить оптимизационные подходы, такие как:

- Использование динамического программирования
- Рекурсивный подход с вычислением подзадач и кэшированием (мемоизацией) промежуточных результатов для устранения избыточных вычислений,
- Выбор подходящих структур данных для хранения и быстрого поиска решений подзадач.

Также необходимо оценить время выполнения алгоритма и привести *асимптотическую оценку* временной сложности разработанного решения, обосновав её.

Описание алгоритмов

1) Алгоритм полного перебора.

Алгоритм полного перебора представляет собой метод решения задачи о рюкзаке путём проверки всех возможных подмножеств предметов.

Описание работы алгоритма:

1. Перебираются все возможные комбинации предметов разного размера.
2. Для каждой комбинации вычисляется её суммарный вес и суммарная ценность.
3. Если суммарный вес комбинации не превышает вместимость рюкзака C, алгоритм проверяет, превышает ли её ценность текущую максимальную ценность. Если да, то обновляет максимальную ценность.
4. После проверки всех возможных комбинаций алгоритм возвращает максимальную найденную ценность.

Knapsack_bruteforce

Вход: значения предметов $v_1, v_2, v_3, \dots, v_n$, размеры предметов $s_1, s_2, s_3, \dots, s_n$, емкость ранца C, кол-во элементов N.

Выход: максимальное суммарное значение подмножества $S \subseteq 1,2, \dots, n$, где $\sum_{i \in S} s_t \leq C$.

```
max_profit := 0
for j = 1 to N + 1
    for подмножества c (size = j) to
        total_weight := sum(s_j)
        total_value = sum(v_j)

        if total_weight <= C:
            max_profit := max(max_profit, total_value)
```

Псевдокод 1. Псевдокод алгоритма полного перебора.

2) Алгоритм динамического программирования.

Идея алгоритма динамического программирования заключается в том, чтобы на каждом шаге решать подзадачи — вычислять оптимальное решение для каждого веса от 0 до C с использованием только первых i предметов. Используя значения, вычисленные на предыдущих шагах, которые мы храним в двумерном массиве, мы можем построить оптимальное решение для всей задачи, избегая повторных вычислений.

Описание работы алгоритма:

Пусть $A(k, s)$ - максимальная стоимость предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые k предметов, то есть $\{n_1, n_2, \dots, n_k\}$, назовем этот набор допустимых предметов для $A(k, s)$.

$$A(k, 0) = 0$$

$$A(0, s) = 0$$

Найдем $A(k, s)$. Возможны 2 варианта:

1. Если предмет k не попал в рюкзак. Тогда $A(k, s)$ равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов $\{n_1, n_2 \dots, n_{k-1}\}$, то есть $A(k, s) = A(k-1, s)$.
2. Если k попал в рюкзак. Тогда $A(k, s)$ равно максимальной стоимости рюкзака, где вес s уменьшаем на вес k-ого предмета (w_k) и набор допустимых предметов $\{n_1, n_2 \dots, n_{k-1}\}$ плюс стоимость k (v_k), то есть $A(k-1, s-w_k) + v_k$, где v_k – значение k-го предмета (его стоимость).

Таким образом: $A(k, s) = max(A(k-1, s), A(k-1, s-w_k) + v_k)$, *где* v_k *– значение k-го предмета (его стоимость),* w_k *– вес k-го предмета.*

Стоимость искомого набора равна $A(N, \mathrm{C})$, *так как нужно найти максимальную стоимость рюкзака, где все предметы допустимы и вместимость рюкзака C.*

Knapsack_dynamic

Вход: значения предметов $v_1, v_2, v_3, \dots, v_n$, размеры предметов $s_1, s_2, s_3, \dots, s_n$, емкость ранца C (все положительные целые числа), кол-во элементов N.

Выход: максимальное суммарное значение подмножества $S \subseteq 1,2, \dots, n$, где $\sum_{i \in S} s_t \leq C$.

```
// решения подзадач (проиндексированы от 0)
A = (N + 1) × (C + 1) двумерный массив
// базовый случай (i = 0)
for c = 0 to C do
    A[0][c] = 0
// систематически решить все подзадачи
for i = 1 to N do
    for c = 0 to C do
        // используем рекуррентное отношение (в описании)
        if s_i > c then
            A[i][c] := A[i − 1][c]
        else
            A[i][c] := max{A[i − 1][c], A[i − 1][c − s_i] + v_i}

return A[N][C]
```

Псевдокод 2. Псевдокод алгоритма динамического программирования.

3) имеРекурсивный алгоритм с мемоизацией (через словарь).

Алгоритм решает задачу о рюкзаке, рекурсивно проверяя два возможных случая для каждого предмета: *включение предмета в рюкзак* (если позволяет вместимость) или *исключение его*. Результаты подзадач сохраняются в словаре (или мемо), чтобы при

повторном обращении к той же подзадаче не вычислять её заново, что ускоряет процесс и снижает временные затраты.

Идея алгоритма: на каждом этапе выбираем максимальную суммарную ценность, рассматривая только подмножество предметов до текущего предмета. Рекурсия продолжается, пока не достигнет базы — либо индекс предмета равен нулю (нет доступных предметов), либо вместимость рюкзака равна нулю.

Описание работы алгоритма:

1. **Инициализация:** используется словарь memo для хранения уже вычисленных значений подзадач.
2. **Базовый случай:** если нет оставшихся предметов или вместимость рюкзака равна 0, максимальная ценность — 0.
3. **Кэширование:** если решение подзадачи уже вычислено, оно берётся из memo, что позволяет избежать повторных вычислений.
4. **Рекурсивное вычисление**: если текущий предмет может поместиться в рюкзак, алгоритм рассматривает два варианта — включить или исключить предмет, выбирая вариант с максимальной ценностью.
5. **Возврат результата:** после завершения всех подзадач функция возвращает максимальную ценность для набора предметов и вместимости рюкзака

Knapsack_recursive

Вход: значения предметов $v_1, v_2, v_3, \dots, v_n$, размеры предметов $s_1, s_2, s_3, \dots, s_n$, емкость ранца C (все положительные целые числа), кол-во элементов N.

Выход: максимальное суммарное значение подмножества $S \subseteq 1,2, \dots, n,$ где $\sum_{i \in S} s_t \leq C$.

```
memo = пустой словарь (для кэширования)

// рекурсивная функция
function knapsack_recursive(C, items, index):
    if index == 0 or C == 0 then:
        return 0

    // если результат уже есть в кэше, вернуть его
    if (index, C) in memo then:
        return memo[(index, C)]

    value := items[index - 1].value
    weight := items[index - 1].weight

    if weight > C then:
        memo[(index, C)] := knapsack_recursive(C, items, index - 1)
    else:
        memo[(index, C)] := max(
            knapsack_recursive(C, items, index - 1),  // не берём предмет
            value + knapsack_recursive(C - weight, items, index - 1)  // берём предмет
        )
```

```
    return memo[(index, C)]

// основная функция
function knapsack(capacity, items):
    return knapsack_recursive(capacity, items, N)
```

Псевдокод 3. Псевдокод рекурсивного алгоритма с мемоизацией.

4) Алгоритм ветвления и отсечения на основе дерева решений (Branch and Bound).

Метод ветвления и отсечения (Branch and Bound) применяется для решения задачи о рюкзаке, когда набор предметов слишком велик, и простой перебор всех комбинаций потребовал бы слишком много времени. Этот метод использует дерево решений, в котором каждый узел представляет вариант, включить предмет в рюкзак или нет. В этом дереве каждый узел проверяется на потенциальный максимум ценности, и узлы, которые не могут привести к лучшему решению, отсеиваются.

Рассмотрим работу алгоритма на 2 примерах.

- Пример 1 (тренировочный).

Дано: $C = 10, N = 4, E = \{(10,2),(5,3),(15,5),(7,7)\}$.

Найти: максимальное суммарное значение.

Решение:

Для начала необходимо упорядочить элементы из множества E в порядке убывания отношений $\frac{\text{значение}}{\text{вес}}$. Для наших данных:

$$\frac{10}{2} = 5;\ \frac{5}{3} = 1{,}67;\ \frac{15}{5} = 3;\ \frac{7}{7} = 1;\ \Rightarrow\ E_{sort} = \{(10,2),(15,5),(5,3),(7,7)\}.$$

Теперь начинаем строить дерево решений.

Создаем начальный узел (или корень дерева решений) с уровнем (level = -1) со значением и весом, равными 0 *(этот узел является базовым состоянием, где никаких предметов ещё не добавлено в рюкзак и является отправной точкой для построения дерева решений).*

Будем формировать дочерние узлы в дереве, исходя из предиката: добавляем или не добавляем следующий элемент (бинарная классификация) – левый потомок будет соответствовать узлу, в который добавили элемент, правый – не добавили. Для каждого узла мы будем высчитывать верхнюю оценку по значениям (она может быть дробная) так, чтобы вес узла не превосходил максимального веса ранца. По максимальному значению верхней оценки мы будем выбирать узел для дальнейшего разбиения.

Замечание: подход с верхней оценкой позволяет не рассматривать заведомо неоптимальные пути.

Итак, в корень дерева мы поместили элемент (0, 0). Его верхняя оценка равна $10 + 15 + 5 = 30$.

Левый потомок корня будет равен (10, 2) с верхней оценкой такой же, как у корня дерева, а правый будет равен (0, 0), так как мы не добавили в него первый элемент, с верхней оценкой:

$15 + 5 + 7\frac{(10-8)}{7} = 22$. (Почему она такая? – первый элемент (10, 2) мы не берем для расчета верхней оценки, а последний помещается в рюкзак не весь, а только его часть).

Теперь сравниваем верхнюю оценку двух потомков корня и выбираем максимальную (в данном случае левый потомок).

К левому потомку добавляем следующий элемент. Тогда его левый потомок равен (25, 7) с верхней оценкой, как у своего предка, а правый (10, 2) с оценкой 20. Выбираем левого.

Таким образом, проделываем операцию до конца, пока не достигнем максимальной ценности в листе, равной верхней оценки этого листа.

На рисунке 1 показано итоговое дерево решений для этой задачи.

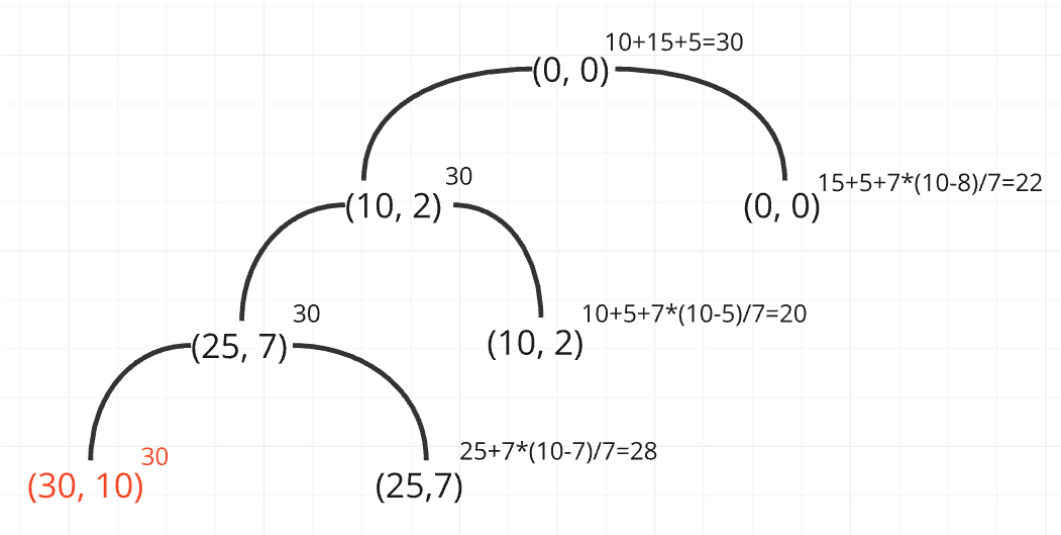

Рисунок 1. Дерево решений для примера 1.

- Пример 2 (усложненный).

Дано: $C = 10, N = 4, E = \{(10,4), (12,5), (8,3), (11,4)\}$.

Найти: максимальное суммарное значение.

Решение:

$$\frac{10}{4} = 2{,}5;\ \frac{12}{5} = 2{,}4;\ \frac{8}{3} = 2{,}66;\ \frac{11}{4} = 2{,}75;\ \Rightarrow\ E_{sort} = \{(11,4), (8,3), (10,4), (12,5)\}.$$

В данном примере все отношения очень близки по значениям, что усложнит наше дерево, сделав его более сбалансированным. Дерево решений для примера 2 изображено на рисунке 2.

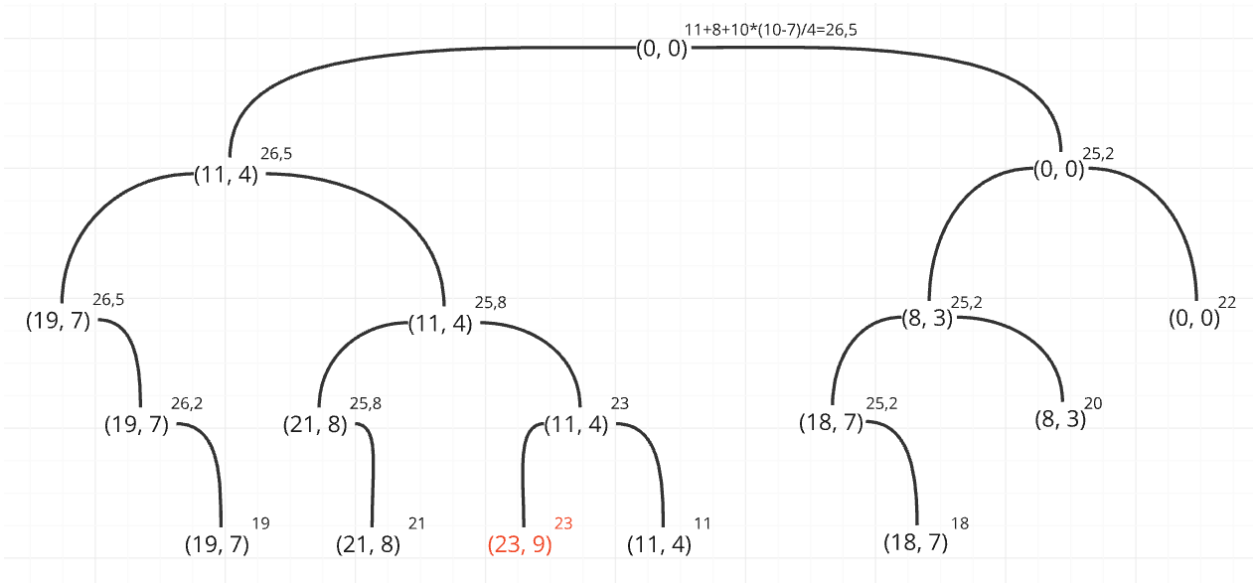

Рисунок 2. Дерево решения для примера 2.

Доказательство корректности

1) Алгоритм полного перебора.

Пусть S – множество всех возможных подмножеств предметов, тогда $S_{valid} \subseteq S$ – множество подмножеств, для которых выполняется условие $\sum_{i \in S_{valid}} weight_i \leq capacity$.

Алгоритм проходит через все подмножества S и проверяет для каждого подмножества $s \in S$:

Если $s \in S_{valid}$, вычисляется его общая ценность. Если она превышает текущее максимальное значение, максимальное значение обновляется.

Так как алгоритм проверяет каждое подмножество и обновляет max_profit, если найдено подмножество с более высокой ценностью, то в конце работы max_profit содержит максимальную возможную ценность среди всех допустимых решений, что и является целью задачи.

2) Алгоритм динамического программирования (индуктивное доказательство).

Для начальных условий имеем:

- Если нет предметов ($i = 0$), то максимальная ценность рюкзака равна 0, независимо от вместимости w: $pack[0][w] = 0$.
- Если рюкзак имеет вместимость 0 ($w = \mathbf{0}$), то ценность тоже равна 0: $pack[i][0] = 0$.

Эти начальные значения являются основой для всех дальнейших вычислений.

Докажем, что $pack[i][w]$ действительно максимизирует ценность рюкзака при рассматриваемых условиях.

База индукции: для $i = 0$ и $w = \mathbf{0}$, как было задано, $pack[0][w] = 0$. Это верно, так как при отсутствии предметов или вместимости рюкзака максимальная ценность всегда равна нулю.

Индуктивное предположение: предположим, что для всех значений $i' < i$ и вместимостей $w' \leq C$ алгоритм корректно вычисляет $pack[i'][w']$ как максимальную суммарную ценность для подмножества предметов с учетом текущей вместимости w'.

Индуктивный шаг: рассмотрим *i*-й предмет и вместимость *w*:

- Если $weight_i > w$, алгоритм устанавливает $pack[i][w] = pack[i-1][w]$ поскольку текущий предмет нельзя добавить, что соответствует максимуму возможной ценности.
- Если $weight_i \leq w$, алгоритм выбирает максимум из:
  - $pack[i-1][w]$ (ценность без включения текущего предмета) и
  - $pack[i-1][w - weight_i] + value_i$ (ценность с включением текущего предмета).

Обе подзадачи $pack[i-1][w]$ и $pack[i-1][w-weight_i]$ уже оптимально решены по индуктивному предположению. Следовательно, $pack[i][w]$ выбирает максимальное из доступных решений.

После того как таблица полностью заполнена, итоговое решение для исходной задачи — это $pack[n][C]$, где учтены все n предметов и вся вместимость C рюкзака. Значение в этой ячейке — максимальная ценность, которую можно получить, не превышая заданную вместимость.

3) Рекурсивный алгоритм с мемоизацией (индуктивное доказательство).

Если индекс текущего предмета ($i = 0$) или оставшаяся вместимость рюкзака ($w = 0$), то оптимальная ценность равна 0, поскольку добавлять больше предметов невозможно.

То есть: $knapsack(0, w) = 0$ для любого w. Это соответствует базовым условиям, как и в алгоритме динамического программирования: если нет доступных предметов или рюкзак полностью заполнен, ценность остаётся 0.

База индукции: для $i = 0$ или $w = 0$ значение функции равно нулю, что соответствует оптимальной ценности рюкзака в этих случаях.

Индуктивное предположение: предположим, что для всех $i' < i$ и $w' \leq C$, значение функции $knapsack(i', w')$ вычисляется оптимально и сохраняется в $memo[(i', w')]$.

Индуктивный шаг: рассмотрим *i*-й предмет и вместимость *w*:

1. Если $weight_i > w$, то оптимальная ценность для $knapsack(i, w)$ берётся из $knapsack(i-1, w)$, как в индуктивном предположении.
2. Если $weight_i \leq w$, оптимальная ценность берётся из максимума значений $knapsack(i-1, w)$ и $value_i + knapsack(i-1, w-weighti)$, где обе подзадачи вычислены корректно по индуктивному предположению.

Завершив рекурсивное построение с мемоизацией, конечный результат $knapsack(n, C)$ содержит максимальную ценность, которую можно получить, используя n предметов при вместимости C. Это значение соответствует оптимальному решению исходной задачи, так как каждый возможный выбор был проверен через рекурсивное разветвление и сохранение оптимальных подзадач.

4) Алгоритм ветвления и отсечения на основе дерева решений.

Корректность алгоритма можно показать, доказав два ключевых утверждения:

1. Алгоритм всегда находит оптимальное решение, так как он не отсеивает потенциально лучшие варианты.
2. Верхняя оценка корректно ограничивает максимальную ценность, возможную по выбранному пути.

Доказательство первого утверждения:

Алгоритм использует стратегию, известную как *метод ветвей и границ*. Основной принцип заключается в том, что алгоритм проверяет все возможные варианты размещения предметов, отсеивая ветви, которые не могут привести к оптимальному решению.

На первом шаге добавляется корневой узел дерева решений, представляющий пустой рюкзак, с ценностью и весом, равными нулю, и вычисленной верхней оценкой (текущая максимальная ценность равна 0).

Алгоритм *всегда извлекает узел с наибольшей верхней оценкой*. Это гарантирует, что рассматривается наиболее перспективный путь. Если на каком-то узле верхняя оценка оказывается ниже текущей максимальной ценности max_profit, этот узел отсеивается, так как он не может улучшить текущее лучшее решение.

Для каждого узла алгоритм рассматривает два случая: добавление текущего предмета в рюкзак и исключение его. Для каждого нового узла рассчитываются:

- **новая ценность** рюкзака,
- **новый вес**,
- **верхняя оценка**.

Таким образом, алгоритм исследует все возможные комбинации включения и исключения предметов, что **позволяет не упустить оптимальное решение.**

Мы обновляем максимальную ценность, исходя из ценности нового узла. Поскольку алгоритм рассматривает узлы в *порядке убывания верхней оценки*, всегда будет представлять собой **наибольшую найденную ценность**, возможную для текущего состояния рюкзака.

Доказательство второго утверждения:

Пусть текущий узел на уровне i имеет вес W и ценность V. Рассмотрим оставшуюся вместимость $C - W$ и предметы с индексами $j \geq i + 1$, отсортированные по убыванию $\frac{v_j}{w_j}$ (отношение ценности к весу).

Тогда верхняя оценка *bound* определяется как:

$$bound = V + \sum_{j=i+1}^{k} v_j + \frac{(C - W - \sum_{j=i+1}^{k} w_j) \cdot v_{k+1}}{w_{k+1}}$$

где *k* — индекс последнего полностью добавленного предмета.

Эта верхняя оценка является корректной, так как она включает максимально возможную ценность при текущих условиях, но не превышает её. Если верхняя оценка оказывается меньше текущей *max_profit*, то никакие дальнейшие узлы, порожденные от данного узла, не смогут превысить *max_profit*, и узел можно отбросить.

Таким образом, *алгоритм ветвления и отсечения* является корректным.

Асимптотическая оценка временной сложности алгоритмов

1) Алгоритм полного перебора.

Из линейной алгебры и дискретной математики мы знаем, что мощность всех подмножеств множества равняется 2^n, где n – кол-во элементов множества.

В алгоритме полного перебора мы генерируем все возможные множества, а после высчитываем суммарную стоимость по всем элементам подмножества. Соответственно, **асимптотическая сложность алгоритма полного перебора составляет $\boldsymbol{O(2^n)}$**, где n — количество предметов.

2) Алгоритм динамического программирования.

При использовании динамического программирования в задаче о рюкзаке нам необходимо построить двумерный массив **pack** размером $(n+1) \times (C+1)$, где n – кол-во элементов, C – размер рюкзака.

Каждая ячейка массива $pack[i][w]$ хранит **максимальную ценность** подмножества предметов от 1 до i и вычисляется за $O(1)$ (либо копирует значение, либо вычисляет максимум), которое можно вместить в рюкзак ёмкостью w, поэтому нам необходимо заполнить каждую ячейку массива (кроме 1-ой строки и 1-го столбца).

Всего ячеек без первой строки и первого столбца $n \times C$, поэтому **асимптотическая сложность алгоритма динамического программирования $\boldsymbol{O(n \cdot C)}$**.

3) Рекурсивный алгоритм с мемоизацией.

Рекурсивный алгоритм с мемоизацией для задачи о рюкзаке также имеет **асимптотическую сложность $\boldsymbol{O(n \cdot C)}$,** аналогично подходу с динамическим программированием.

Поскольку каждый предмет может быть либо включён, либо исключён при определённой вместимости рюкзака, **возможное количество различных состояний** ограничено комбинациями (i, w), где:

- i принимает значения от 0 до n,
- w принимает значения от 0 до C.

Таким образом, количество возможных подзадач (или уникальных записей в **memo**) составляет $\boldsymbol{O(n \cdot C)}$.

4) Алгоритм ветвления и отсечения на основе дерева решений.

Оценить асимптотическую сложность данного алгоритма довольно сложно. В примере 2, рассмотренным выше, мы заметили, что все $\frac{v_j}{w_j}$ очень близки по значениям, что усложняет наше дерево, делая его более сбалансированным. Это заставляет нас обрабатывать почти все возможные комбинации предметов, что приводит к экспоненциальной сложности (если на каждом уровне не удаётся отсеивать узлы, и алгоритм вынужден рассматривать все подмножества предметов) – $O(2^n)$.

Однако, на практике отношения $\frac{v_j}{w_j}$ не так часто близки по значениям, поэтому оценку в виде экспоненциальной сложности можно рассматривать, как **худший случай** для данного алгоритма.

Алгоритм ветвей и границ выигрывает за счёт отсечения (отказа от неперспективных узлов) и очереди с приоритетом, которая обрабатывает узлы с наиболее высокими верхними оценками. Это позволяет в среднем обрабатывать только часть пространства состояний, что значительно снижает время выполнения по сравнению с полным перебором.

Средняя сложность становится ближе к:

$$O(k \cdot logk)$$

где k — количество узлов, которые реально добавляются и извлекаются из очереди.

Результаты запуска программы

Теперь перейдем к оценке работы этих алгоритмов на тестовой и усложненной выборках, а также проведем сравнение времени их выполнения.

В тестовом наборе данных: ограничение по весу: 10000, кол-во предметов: 100.

В усложненном наборе данных: ограничение по весу: 2.000.000, кол-во предметов: 2000.

Каждый файл описывает экземпляр задачи о рюкзаке и имеет формат:

[размер_рюкзака] [количество_предметов]

[значение_1] [вес_1]

[значение_2] [вес_2]

В таблице 1 приведено сравнение всех алгоритмов по времени выполнения с вычисленным значением на обоих наборах данных.

Алгоритм	Время выполнения на тестовом наборе данных	Время выполнения на усложненном наборе данных	Вычисленный результат на тестовом наборе данных	Вычисленный результат на усложненном наборе данных
Полный перебор	3 ч. 34 мин.	>6 часов	2493893	-
Динамическое программирование	0.317 с.	20 мин. 42 с.	2493893	4243395
Рекурсивный с мемоизацией	1.174 с.	15.918 с.	2493893	4243395
Ветвление и отсечение на основе дерева решений	0.002 с.	0.011 с.	2493893	4243395

Таблица 1. Сравнение всех алгоритмов для вычисления максимальной суммы значений предметов (Knapsack) в задаче о рюкзаке.

Заключение

В ходе курсовой работы были рассмотрены 4 алгоритма решения задачи о рюкзаке (вычисления максимальной суммы значений предметов (knapsack)):

- Алгоритм полного перебора (сложность $O(2^n)$),
- Алгоритм динамического программирования (сложность $O(n \cdot C)$)
- Рекурсивный алгоритм с мемоизацией (сложность $O(n \cdot C)$)
- Алгоритм ветвления и отсечения на основе дерева решений ($O(k \cdot logk)$).

Мной были доказаны корректности данных алгоритмов и выведена их асимптотическая оценка.

Алгоритмы были протестированы на тестовом и усложненном наборе данных, где лучшим по времени выполнения оказался **алгоритм ветвления и отсечения на основе дерева решений,** выполнив задачу на тестовом наборе за 0.002 с., на усложненном за 0.023 с..

Список литературы

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - 3-е изд. - СПб: Питер, 2024. - 288 с.
2. Рафгарден Т. Совершенный алгоритм. Основы. - 2-е изд. - СПб: Питер, 2023. - 256 с.
3. Задача о рюкзаке // Яндекс Образование URL: https://education.yandex.ru/handbook/algorithms/article/zadacha-o-ryukzake (дата обращения: 07.11.2024).
4. Скиена С.С. Алгоритмы. Руководство по разработке. - 2-е изд. - М.: БХБ, 2018. - 720 с.

Приложения

```
    start_time = time.time()
    max_value = knapsack_pack(C, data)
    end_time = time.time()

    print(f'Время выполнения: {end_time - start_time}')
    print(f'Максимальная сумма значений предметов: {max_value}')
```

✓ 0.3s

```
Время выполнения: 0.31743597984313965
Максимальная сумма значений предметов: 2493893
```

Рисунок 3. Результат выполнения алгоритма динамического программирования на тестовых данных.

```
    start_time = time.time()
    max_value = knapsack(C, data)
    end_time = time.time()

    print(f'Время выполнения: {end_time - start_time}')
    print(f'Максимальная сумма значений предметов: {max_value}')
```

✓ 1.1s

```
Время выполнения: 1.1746985912322998
Максимальная сумма значений предметов: 2493893
```

Рисунок 4. Результат выполнения рекурсивного алгоритма с мемоизацией на тестовых данных.

```
    start_time = time.time()
    max_value = knapsack_branch_and_bound(C, data)
    end_time = time.time()

    print(f'Время выполнения на данных: {end_time - start_time}')
    print(f'Максимальная сумма значений предметов на данных: {max_value}')
```

✓ 0.0s

```
Время выполнения на данных: 0.0024743080139160156
Максимальная сумма значений предметов на данных: 2493893
```

Рисунок 5. Результат выполнения алгоритма ветвления и отсечения на тестовых данных.

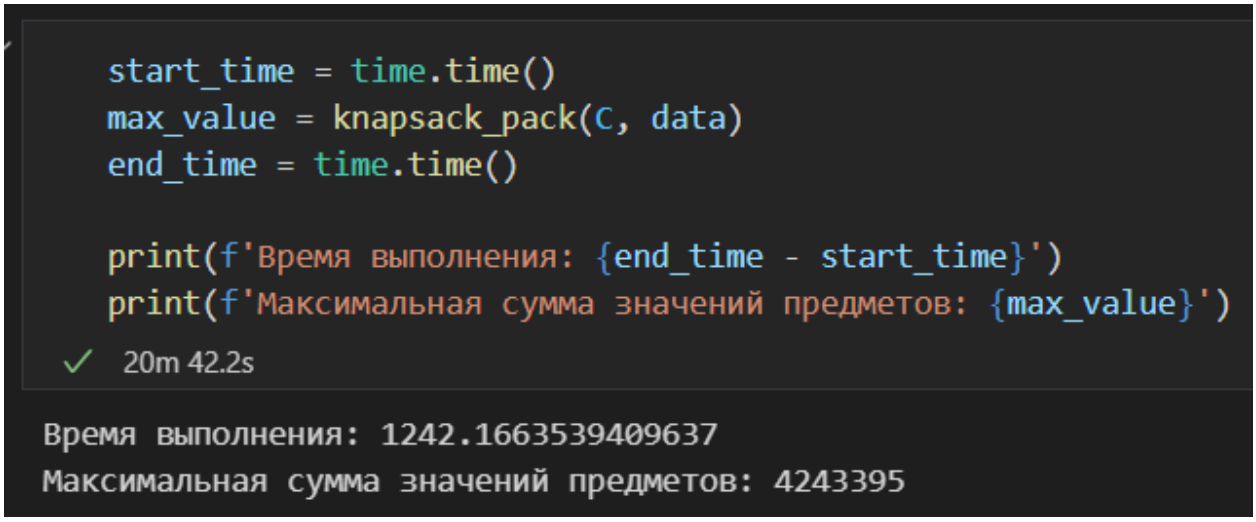

Рисунок 6. Результат выполнения алгоритма динамического программирования на усложненных данных.

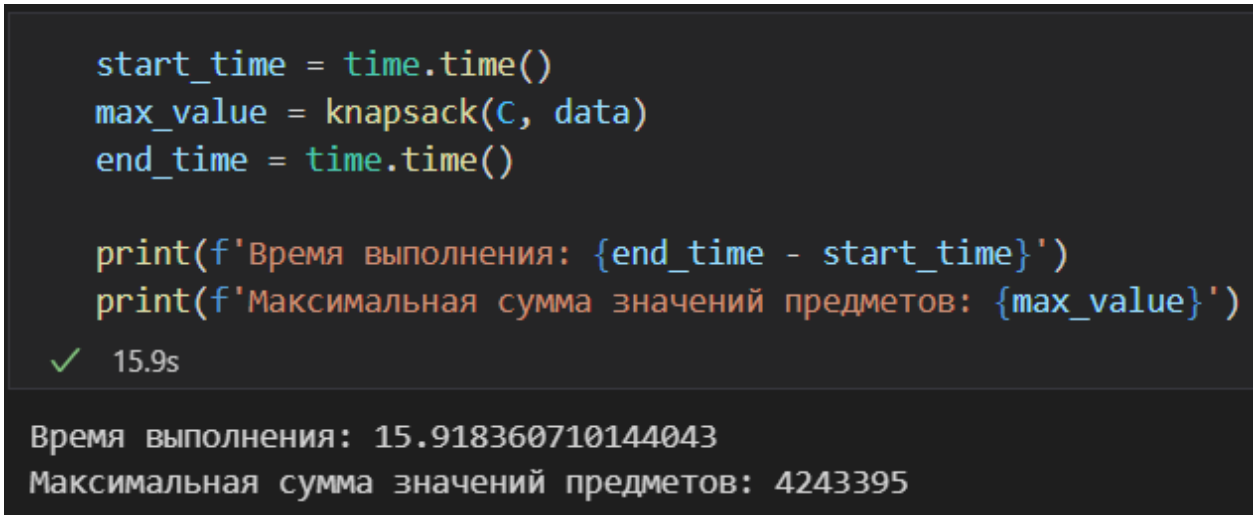

Рисунок 7. Результат выполнения рекурсивного алгоритма с мемоизацией на усложненных данных.

```
    start_time = time.time()
    max_value = knapsack_branch_and_bound(C, data)
    end_time = time.time()

    print(f'Время выполнения на данных: {end_time - start_time}')
    print(f'Максимальная сумма значений предметов на данных: {max_value}')
 ✓  0.0s
Время выполнения на данных: 0.011014938354492188
Максимальная сумма значений предметов на данных: 4243395
```

Рисунок 8. Результат выполнения алгоритма ветвления и отсечения на усложненных данных.