Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

РЕФЕРАТ

По дисциплине «Основы алгоритмизации и программирования»

На тему «Динамическое программирование»

Выполнила:

Студентка 1 курса 7 группы

Яскевич Валерия Александровна

Проверил:

Белодед Николай Иванович

2023, Минск

**Оглавление**

[**Введение** 3](#_Toc162571256)

[**Основные принципы программирования** 4](#_Toc162571257)

[**Реализация динамического программирования в C++** 5](#_Toc162571258)

[**Сравнение с другими методами** 8](#_Toc162571259)

[**Заключение** 9](#_Toc162571260)

[**Список литературы** 10](#_Toc162571261)

# **Введение**

Динамическое программирование – это метод оптимизации, который в основном направлен на улучшение производительности рекурсивных алгоритмов. Когда мы сталкиваемся с задачами, где одни и те же подзадачи решаются многократно для одних и тех же входных данных, мы можем применить динамическое программирование для ускорения процесса. Основная идея состоит в том, чтобы запоминать результаты уже решенных подзадач, чтобы избежать повторных вычислений в будущем. Это простое улучшение позволяет сократить временную сложность алгоритма с экспоненциальной до полиномиальной. Например, если рассмотреть простой рекурсивный алгоритм для вычисления чисел Фибоначчи, то временная сложность будет экспоненциальной. Однако, применение динамического программирования и сохранение результатов уже вычисленных чисел Фибоначчи позволит снизить временную сложность до линейной.

Динамическое программирование – это метод решения задач в информатике, который предполагает разбиение сложной задачи на более простые подзадачи и последующее сохранение результатов этих подзадач для того, чтобы избежать повторных вычислений. Этот подход находит широкое применение в различных областях, включая алгоритмы, оптимизацию, искусственный интеллект и другие.

Вот области, где динамическое программирование является важным инструментом:

1.**Алгоритмы**: Динамическое программирование используется для решения различных алгоритмических задач, таких как нахождение кратчайших путей в графах (например, алгоритм Флойда-Уоршелла), оптимального распределения ресурсов (например, задача о рюкзаке), нахождение наибольшей общей подпоследовательности и т.д.

2.**Оптимизация**: Динамическое программирование широко применяется для решения задач оптимизации, таких как оптимальное управление запасами, управление производственными процессами, оптимизация расписания и т.д. В этих случаях динамическое программирование помогает найти наилучший способ распределения ресурсов или времени с учетом ограничений и целей.

3.**Искусственный** **интеллект**: В области искусственного интеллекта динамическое программирование используется для решения задач планирования, обучения с подкреплением, построения деревьев принятия решений и других. Например, алгоритмы обучения с подкреплением, такие как Q-learning или методы динамического программирования, используются для обучения агентов в средах с наградой.

Динамическое программирование является мощным методом решения задач, который может существенно улучшить эффективность и точность различных алгоритмов и моделей в различных областях науки и техники.

# **Основные принципы программирования**

Динамическое программирование представляет собой метод решения задач, включающий два основных принципа: понятие оптимальной подструктуры и мемоизацию. Первый принцип заключается в том, что оптимальное решение для всей задачи может быть получено из оптимальных решений ее подзадач. Иными словами, если задача имеет оптимальную подструктуру, то решение ее можно построить путем комбинирования оптимальных решений более простых подзадач:

int fibonacci(int n) {

if (n <= 1)

return n;

return fibonacci(n - 1) + fibonacci(n - 2);

}

Второй принцип, мемоизация, предполагает сохранение результатов вычислений для избежания повторных вычислений в будущем. Когда значение функции вычисляется для конкретных входных данных, результат сохраняется в памяти. При повторном запросе к этой же функции для тех же входных данных результат извлекается из памяти, что уменьшает время выполнения программы. Мемоизация является важным элементом оптимизации при применении динамического программирования:

#include <iostream>

#include <vector>

std::vector<int> memo(100, -1); // Массив для хранения рассчитанных значений

int fibonacci(int n) {

if (n <= 1)

return n;

if (memo[n] != -1)

return memo[n];

memo[n] = fibonacci(n - 1) + fibonacci(n - 2);

return memo[n];

}

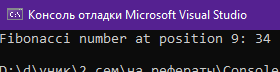
int main() {

int n = 9;

std::cout << "Fibonacci number at position " << n << ": " << fibonacci(n) << std::endl;

return 0;

}



Эти два принципа в совокупности обеспечивают эффективное решение задач с использованием динамического программирования. Понимание оптимальной подструктуры позволяет разбивать сложные задачи на более простые подзадачи, а мемоизация помогает избежать повторных вычислений и ускоряет выполнение алгоритма. Комбинация этих принципов делает динамическое программирование мощным инструментом для решения широкого спектра задач, включая алгоритмические, задачи оптимизации и задачи искусственного интеллекта.

# **Реализация динамического программирования в C++**

Реализация динамического программирования в C++ имеет свои особенности, связанные с языком программирования и спецификой его стандартной библиотеки. Вот некоторые особенности и примеры:

1. **Использование стандартной библиотеки**: в C++ для реализации динамического программирования часто используются контейнеры из стандартной библиотеки, такие как `std::vector` или `std::array`, для хранения результатов подзадач и промежуточных данных.

2. **Указатели и ссылки**: для эффективной работы с динамическим программированием в C++ часто используются указатели и ссылки для передачи данных между функциями и для управления памятью.

3. **Оператор new/delete**: для динамического выделения и освобождения памяти при работе с динамическим программированием в C++ часто используются операторы `new` и `delete` для создания и удаления динамических структур данных.

Нужно отметить несколько задач с использованием динамического программирования:

**Нахождение наибольшей общей подпоследовательности:**

#include <iostream>

#include <vector>

#include <algorithm>

// Функция для нахождения длины наибольшей общей подпоследовательности двух строк

int longestCommonSubsequence(const std::string& s1, const std::string& s2) {

// Получаем длины строк

int m = s1.length();

int n = s2.length();

// Создаем двумерный вектор для хранения результатов подзадач

std::vector<std::vector<int>> dp(m + 1, std::vector<int>(n + 1, 0));

// Заполняем двумерный вектор с использованием динамического программирования

for (int i = 1; i <= m; ++i) {

for (int j = 1; j <= n; ++j) {

if (s1[i - 1] == s2[j - 1]) // Если символы в строках совпадают

dp[i][j] = dp[i - 1][j - 1] + 1; // Увеличиваем длину общей подпоследовательности

else

dp[i][j] = std::max(dp[i - 1][j], dp[i][j - 1]); // В противном случае берем максимум из предыдущих значений

}

}

// Возвращаем длину наибольшей общей подпоследовательности

return dp[m][n];

}

int main() {

// Входные строки

std::string s1 = "ABCDGH";

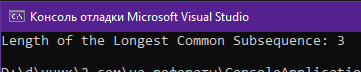
std::string s2 = "AEDFHR";

// Вызываем функцию нахождения длины наибольшей общей подпоследовательности

std::cout << "Length of the Longest Common Subsequence: " << longestCommonSubsequence(s1, s2) << std::endl;

return 0; // Возвращаем успешный статус завершения программы

}



**Рюкзак:**

#include <iostream>

#include <vector>

#include <algorithm>

// Функция для решения задачи о рюкзаке

int knapsack(int W, const std::vector<int>& weights, const std::vector<int>& values) {

int n = weights.size(); // Получаем количество предметов

std::vector<std::vector<int>> dp(n + 1, std::vector<int>(W + 1, 0)); // Создаем двумерный вектор для хранения результатов подзадач

// Заполняем двумерный вектор с использованием динамического программирования

for (int i = 1; i <= n; ++i) {

for (int w = 1; w <= W; ++w) {

if (weights[i - 1] <= w) // Если текущий предмет помещается в рюкзак

dp[i][w] = std::max(dp[i - 1][w], values[i - 1] + dp[i - 1][w - weights[i - 1]]); // Берем максимум между включением и исключением текущего предмета

else

dp[i][w] = dp[i - 1][w]; // Если текущий предмет не помещается в рюкзак, берем значение из предыдущей строки

}

}

// Возвращаем максимальное значение в рюкзаке

return dp[n][W];

}

int main() {

// Заданные значения предметов и их весов

std::vector<int> values = { 60, 100, 120 };

std::vector<int> weights = { 10, 20, 30 };

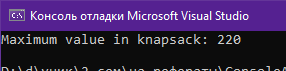
int W = 50; // Вместимость рюкзака

// Вызываем функцию для решения задачи о рюкзаке и выводим результат

std::cout << "Maximum value in knapsack: " << knapsack(W, weights, values) << std::endl;

return 0;

}



**Числа Фибоначчи:**

#include <iostream>

#include <vector>

// Функция для вычисления числа Фибоначчи на заданной позиции

int fibonacci(int n) {

std::vector<int> dp(n + 1, 0); // Создаем вектор для хранения значений чисел Фибоначчи

dp[0] = 0; // Первое число Фибоначчи

dp[1] = 1; // Второе число Фибоначчи

// Заполняем вектор значениями чисел Фибоначчи с использованием динамического программирования

for (int i = 2; i <= n; ++i) {

dp[i] = dp[i - 1] + dp[i - 2]; // Каждое последующее число Фибоначчи равно сумме двух предыдущих

}

return dp[n]; // Возвращаем число Фибоначчи на заданной позиции

}

// Главная функция программы

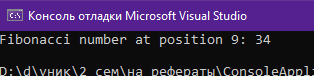
int main() {

int n = 9; // Заданная позиция числа Фибоначчи

std::cout << "Fibonacci number at position " << n << ": " << fibonacci(n) << std::endl; // Выводим результат

return 0; // Возвращаем успешный статус завершения программы

}



# **Сравнение с другими методами**

**Жадный алгоритм**

Жадный алгоритм для задачи о рюкзаке выбирает предметы с наибольшим соотношением стоимость/вес и добавляет их в рюкзак до тех пор, пока вес рюкзака не превысит его вместимость. Этот процесс повторяется до тех пор, пока не будут рассмотрены все предметы.

Жадный алгоритм обычно имеет линейную временную сложность O(n log n), где n - количество предметов. Однако он не всегда даёт оптимальное решение для задачи о рюкзаке. В некоторых случаях он может пропустить комбинацию предметов, которая дает наилучшую общую стоимость.

**Рекурсивный подход без применения динамического программирования**

Рекурсивный подход к решению задачи о рюкзаке заключается в применении обратной рекурсии: на каждом шаге рекурсии мы принимаем решение, включить текущий предмет в рюкзак или нет, и вызываем рекурсивно функцию для оставшихся предметов с учетом измененной вместимости рюкзака.

Этот подход прост в реализации, но может привести к большому количеству повторных вычислений, так как одни и те же подзадачи могут быть вычислены многократно. Это приводит к экспоненциальной временной сложности и рискам переполнения стека вызовов.

**Сравнение:**

Динамическое программирование, жадные алгоритмы и рекурсивный подход представляют собой различные стратегии решения задач и имеют свои сильные и слабые стороны, особенно при решении задачи о рюкзаке.

Динамическое программирование обладает оптимальностью и эффективной временной сложностью. Этот метод разбивает сложную задачу на более простые подзадачи и сохраняет результаты для последующего использования. В задаче о рюкзаке это позволяет эффективно находить оптимальное решение, минимизируя время выполнения программы. Однако динамическое программирование требует дополнительной памяти для хранения результатов подзадач, что может быть нежелательным в случае ограниченных ресурсов.

Жадные алгоритмы могут быть полезны, когда требуется быстрое решение, и оптимальность не является первоочередным критерием. Они принимают локально оптимальное решение на каждом шаге, надеясь, что это приведет к глобально оптимальному решению. В случае задачи о рюкзаке жадные алгоритмы могут быть быстрее, чем динамическое программирование, но могут давать недопустимо низкое качество решения.

Рекурсивный подход без применения динамического программирования\*\* часто непрактичен из-за своей высокой временной сложности. Рекурсивная реализация может привести к большому количеству повторных вычислений, что делает ее неэффективной даже для небольших входных данных. В случае задачи о рюкзаке рекурсивный подход может привести к экспоненциальной временной сложности, что делает его непрактичным для больших размеров рюкзака или множества предметов.

Таким образом, хотя каждый метод имеет свои преимущества и недостатки, динамическое программирование остается предпочтительным выбором для задачи о рюкзаке благодаря своей оптимальности и эффективной временной сложности, особенно при работе с большими наборами данных.

# **Заключение**

Динамическое программирование представляет собой мощный метод решения задач в информатике и других областях. Его преимущества и ограничения были внимательно рассмотрены в анализе.

Преимущества динамического программирования включают его оптимальность, которая обеспечивается благодаря применению оптимальной подструктуры. Это означает, что метод гарантирует нахождение оптимального решения для многих задач. Кроме того, правильное применение динамического программирования способно значительно сократить время выполнения задач, превращая экспоненциальные временные сложности в полиномиальные. Еще одним преимуществом является возможность повторного использования результатов с помощью мемоизации, что помогает избежать повторных вычислений и ускоряет процесс решения задач.

Однако динамическое программирование имеет и свои ограничения. Например, оно не всегда применимо, так как требует определенной структуры задачи для эффективной работы. Кроме того, для сохранения результатов подзадач может потребоваться дополнительное пространство памяти, что может ограничить его использование в некоторых приложениях. Сложность реализации также может стать проблемой, поскольку некоторые задачи могут быть сложны для реализации с применением динамического программирования, а неправильное применение может привести к ошибкам или неэффективным решениям.

Дальнейшие исследования в области динамического программирования могут быть направлены на оптимизацию методов и разработку новых подходов для улучшения эффективности и расширения областей применения. Кроме того, его использование в машинном обучении и искусственном интеллекте представляет большой потенциал для развития, и исследования в этой области могут помочь расширить его применение и повысить производительность алгоритмов. Развитие алгоритмов для параллельных и распределенных систем также может быть важным направлением исследований, с учетом роста интереса к таким вычислительным платформам. В целом, динамическое программирование остается важным исследовательским направлением с широким спектром приложений и потенциалом для дальнейших улучшений и инноваций.

# **Список литературы**

1. <https://habr.com/ru/articles/113108/>
2. <https://neerc.ifmo.ru/wiki/index.php?title=Динамическое_программирование>
3. <https://апо.рф/материалы/динамическое-программирование/>