

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технологический университет «СТАНКИН» (ФГБОУ ВО «МГТУ «СТАНКИН»)

Институт информационных систем и технологий

Кафедра информационных систем

**09.03.02 «Информационные системы и технологии»**

**Реферат**

по дисциплине **«Проектирование информационных систем»**

Тема: **«**Понятие вычислительной сложности алгоритма. Связанные показатели качества программных средств. Понятие обратного проектирования (реинжиниринга). Назначение и варианты использования**»**

Студент

группы ИДБ-16-07

Махмудов Б.Н.

подпись

Руководитель

старший преподаватель

подпись

Овчинников П.Е.

**ОГЛАВЛЕНИЕ**

[Понятие вычислительной сложности алгоритма. Связанные показатели качества программных средств 3](#_Toc27925194)

[Временная и пространственная сложности 4](#_Toc27925195)

[Асимптотическая сложность 5](#_Toc27925196)

[Классы сложности 7](#_Toc27925197)

Понятие вычислительной сложности алгоритма. Связанные показатели качества программных средств

Вычислительная сложность — понятие в информатике и теории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных. Раздел, изучающий вычислительную сложность, называется теорией сложности вычислений. Объём работы обычно измеряется абстрактными понятиями времени и пространства, называемыми вычислительными ресурсами. Время определяется количеством элементарных шагов, необходимых для решения задачи, тогда как пространство определяется объёмом памяти или места на носителе данных. Таким образом, в этой области предпринимается попытка ответить на центральный вопрос разработки алгоритмов: «как изменится время исполнения и объём занятой памяти в зависимости от размера входа?». Здесь под размером входа понимается длина описания данных задачи в битах (например, в задаче коммивояжёра длина входа почти пропорциональна количеству городов и дорог между ними), а под размером выхода — длина описания решения задачи (наилучшего маршрута в задаче коммивояжера).

В частности, теория сложности вычислений определяет NP-полные задачи, которые недетерминированная машина Тьюринга может решить за полиномиальное время, тогда как для детерминированной машины Тьюринга полиномиальный алгоритм неизвестен. Обычно это сложные задачи оптимизации, например, задача коммивояжёра.

С теоретической информатикой тесно связаны такие области как анализ алгоритмов и теория вычислимости. Связующим звеном между теоретической информатикой и алгоритмическим анализом является тот факт, что их формирование посвящено анализу необходимого количества ресурсов определённых алгоритмов решения задач, тогда как более общим вопросом является возможность использования алгоритмов для подобных задач. Конкретизируясь, попытаемся классифицировать проблемы, которые могут или не могут быть решены при помощи ограниченных ресурсов. Сильное ограничение доступных ресурсов отличает теорию вычислительной сложности от вычислительной теории, последняя отвечает на вопрос какие задачи, в принципе, могут быть решены алгоритмически.

## Временная и пространственная сложности

Теория сложности вычислений возникла из потребности сравнивать быстродействие алгоритмов, чётко описывать их поведение (время исполнения и объём необходимой памяти) в зависимости от размера входа.

Количество элементарных операций, затраченных алгоритмом для решения конкретного экземпляра задачи, зависит не только от размера входных данных, но и от самих данных. Например, количество операций алгоритма сортировки вставками значительно меньше в случае, если входные данные уже отсортированы. Чтобы избежать подобных трудностей, рассматривают понятие временной сложности алгоритма в худшем случае.

Временная сложность алгоритма (в худшем случае) — это функция от размера входных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера.

Аналогично понятию временной сложности в худшем случае определяется понятие временная сложность алгоритма в наилучшем случае. Также рассматривают понятие среднее время работы алгоритма, то есть математическое ожидание времени работы алгоритма. Иногда говорят просто: «Временная сложность алгоритма» или «Время работы алгоритма», имея в виду временную сложность алгоритма в худшем, наилучшем или среднем случае (в зависимости от контекста).

По аналогии с временной сложностью, определяют пространственную сложность алгоритма, только здесь говорят не о количестве элементарных операций, а об объёме используемой памяти.

## Асимптотическая сложность

Несмотря на то, что функция временной сложности алгоритма в некоторых случаях может быть определена точно, в большинстве случаев искать точное её значение бессмысленно. Дело в том, что во-первых, точное значение временной сложности зависит от определения элементарных операций (например, сложность можно измерять в количестве арифметических операций, битовых операций или операций на машине Тьюринга), а во-вторых, при увеличении размера входных данных вклад постоянных множителей и слагаемых низших порядков, фигурирующих в выражении для точного времени работы, становится крайне незначительным.

Рассмотрение входных данных большого размера и оценка порядка роста времени работы алгоритма, приводят к понятию асимптотической сложности алгоритма. При этом алгоритм с меньшей асимптотической сложностью является более эффективным для всех входных данных, за исключением лишь, возможно, данных малого размера.

**Примеры:**

1. «почистить ковёр пылесосом» требует время, линейно зависящее от его площади (A), то есть на ковёр, площадь которого больше в два раза, уйдет в два раза больше времени. Соответственно, при увеличении площади ковра в сто тысяч раз объём работы увеличивается строго пропорционально в сто тысяч раз, и т. п.
2. «найти имя в телефонной книге» требует всего лишь времени, логарифмически зависящего от количества записей , так как, открыв книгу примерно в середине, мы уменьшаем размер «оставшейся проблемы» вдвое (за счет сортировки имен по алфавиту). Таким образом, в книге объёмом в 1000 страниц любое имя находится не больше, чем за 10 раз (открываний книги). При увеличении объёма страниц до ста тысяч проблема все ещё решается за = 17 заходов.

Необходимо подчеркнуть, что степень роста наихудшего времени выполнения — не единственный или самый важный критерий оценки алгоритмов и программ. Приведем несколько соображений, позволяющих посмотреть на критерий времени выполнения с других точек зрения:

Если создаваемая программа будет использована только несколько раз, тогда стоимость написания и отладки программы будет доминировать в общей стоимости программы, то есть фактическое время выполнения не окажет существенного влияния на общую стоимость. В этом случае следует предпочесть алгоритм, наиболее простой для реализации.

Если программа будет работать только с «малыми» входными данными, то степень роста времени выполнения будет иметь меньшее значение, чем константа, присутствующая в формуле времени выполнения. Вместе с тем и понятие «малости» входных данных зависит от точного времени выполнения конкурирующих алгоритмов. Существуют алгоритмы, такие как алгоритм целочисленного умножения, асимптотически самые эффективные, но которые никогда не используют на практике даже для больших задач, так как их константы пропорциональности значительно превосходят подобные константы других, более простых и менее «эффективных» алгоритмов. Другой пример — фибоначчиевы кучи, несмотря на асимптотическую эффективность, с практической точки зрения программная сложность реализации и большие значения констант в формулах времени работы делают их менее привлекательными, чем обычные бинарные деревья. Известны примеры, когда эффективные алгоритмы требуют таких больших объёмов машинной памяти (без возможности использования более медленных внешних средств хранения), что этот фактор сводит на нет преимущество «эффективности» алгоритма. Таким образом, часто важна не только «сложность по времени», но и «сложность по памяти» (пространственная сложность).

В численных алгоритмах точность и устойчивость алгоритмов не менее важны, чем их временная эффективность

## Классы сложности

Класс сложности — это множество задач распознавания, для решения которых существуют алгоритмы, схожие по вычислительной сложности. Два важных представителя:

**Класс P**

Класс P вмещает все те проблемы, решение которых считается «быстрым», то есть время решения, которых полиномиально зависит от размера входа. Сюда относится сортировка, поиск в массиве, выяснение связности графов и многие другие.

**Класс NP**

Класс NP содержит задачи, которые недетерминированная машина Тьюринга в состоянии решить за полиномиальное количество шагов от размера входных данных. Их решение может быть проверено детерминированной машиной Тьюринга за полиномиальное количество шагов. Следует заметить, что недетерминированная машина Тьюринга является лишь абстрактной моделью, в то время как современные компьютеры соответствуют детерминированной машине Тьюринга с ограниченной памятью. Поскольку детерминированная машина Тьюринга может рассматриваться как специальный случай недетерминированной машины Тьюринга, класс NP включает в себя класс P, а также некоторые проблемы, для решения которых известны лишь алгоритмы, экспоненциально зависящие от размера входа (то есть неэффективные для больших входов). В класс NP входят многие знаменитые проблемы, такие как задача коммивояжёра, задача выполнимости булевых формул, факторизация и др.

Понятие обратного проектирования (реинжиниринга). Назначение и варианты использования

Обратная разработка (обратное проектирование, обратный инжиниринг, реверс-инжиниринг) – исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы; например, чтобы обнаружить недокументированные возможности (в том числе программные закладки), сделать изменение или воспроизвести устройство, программу или иной объект с аналогичными функциями, но без прямого копирования.

Применяется обычно в том случае, если создатель оригинального объекта не предоставил информации о структуре и способе создания (производства) объекта. Правообладатели таких объектов могут заявить, что проведение обратной разработки или использование её результатов нарушает их исключительное право по закону об авторском праве и патентному законодательству.

Исследование и обратная разработка программ обычно осуществляются с целью дальнейшей модификации, копирования, или, например, написания генераторов ключей, алгоритм работы которых получен на основе анализа алгоритма их проверки. Также исследование программ применяется с целью получения некоторых закрытых сведений о внутреннем устройстве программы – о протоколе сетевого обмена с сервером, аппаратным средством, ключом защиты или о взаимодействии с другой программой. Ещё одна область применения получение информации о способах экспортирования данных из многочисленных проприетарных форматов файлов.

С развитием Интернета популярные операционные системы и программы всё интенсивнее исследуются на предмет обнаружения в них уязвимостей или т. н. «дыр». В дальнейшем найденные дыры могут использоваться для получения несанкционированного доступа к удалённому компьютеру или компьютерной сети. C другой стороны, обратная разработка применяется при исследовании антивирусными компаниями вредоносного ПО c целью добавления его сигнатур в базы своих продуктов.

Одним из широко известных примеров обратной разработки является исследование BIOS персонального компьютера IBM, ставшее серьёзным шагом на пути развития производства IBM-совместимых компьютеров сторонними производителями. Создание сервера Samba (входящего в состав ОС GNU/Linux и работающего с серверами на базе ОС Windows) также потребовало обратной разработки используемого Microsoft протокола SMB. Создание многих ICQ-клиентов также потребовало обратной разработки протокола ICQ.

Обратная разработка программного обеспечения производится с помощью следующих методик.

Анализ обмена данными, наиболее распространённый в обратной разработке протоколов обмена данными, который производится с помощью анализатора шины и пакетного сниффера для прослушивания шины компьютера и компьютерной сети соответственно.

Дизассемблирование машинного кода программы для получения её листинга на языке ассемблера. Этот способ работает на любой компьютерной программе, но требует достаточно много времени, особенно для неспециалиста.

Декомпиляция машинного или байт-кода программы для создания исходного кода на некотором языке программирования высокого уровня.

В настоящее время под словами «reverse engineering» чаще всего понимается т. н. clean room reverse engineering, то есть процесс, при котором одна группа разработчиков анализирует машинный код программы, составляет алгоритм данной программы на псевдокоде либо, если программа является драйвером какого-либо устройства, составляет исчерпывающие спецификации интересующего устройства. После получения спецификаций другая группа разработчиков пишет собственный драйвер на основе полученных спецификаций или алгоритмов. Такой подход позволяет избежать обвинений в нарушении авторских прав на исходную программу, так как по законам, к примеру в США, попадает под понятие «fair use», то есть добросовестного использования оригинальной программы. Результат обратной разработки редко идентичен оригиналу, что и позволяет избежать ответственности перед законом, особенно при условии контроля отсутствия этой идентичности первой группой разработчиков и отсутствия нарушений торговых марок и патентов.

## Понятие «реинжиниринга ИС»

Сразу следует признать, что в настоящий момент понятие «реинжиниринг ИС» не является повсеместно устоявшимся. Как следствие довольно часто возникает определенная терминологическая путаница. Авторами исследуются одни и те же проблемы, подходы, методы и технологии их решения, однако в качестве базовых понятий, наряду с «реинжинирингом ИС» употребляются «эволюция ИС», «миграция ИС», «модернизация ИС», «реструктуризация ИС».

Нельзя отрицать, что деятельность по миграции ИС имеет определенную специфику (окраску) по отношению к деятельности по модернизации ИС. Однако, принимая во внимание определение реинжиниринга ИС, приводимое в: «Реинжиниринг представляет собой систематическую трансформацию существующей системы с целью улучшения ее характеристик качества, поддерживаемой ею функциональности, понижения стоимости ее сопровождения, вероятности возникновения значимых для заказчика рисков, уменьшения сроков работ по сопровождению системы». Становится очевидным, что и миграция, и модернизация ИС являются частью деятельности по реинжинирингу ИС. Как результат, подходы, методы и технологии миграции, модернизации, эволюции ИС, следует считать частью методологического и инструментально - технологического обеспечения процесса реинжиниринга ИС.

Такой взгляд на реинжиниринг ИС согласуется с таксономией, вводимой в ряде работ. В этих работах авторами делается попытка выстроить систему понятий, соотносимых с данным видом деятельности. Так, в [38] реинжиниринг ИС определяется как «исследование (изучение, обследование) и перестройка исходной системы с целью ее воссоздания в новой форме с последующей реализацией этой новой формы[1]. Далее, в контексте деятельности по реинжинирингу вводятся и определяются такие важные понятия, как:

* прямой инжиниринг (Forward engineering);
* редокументирование (Redocumentation);
* рефакторинг (Refactoring);
* реструктуризация (Restructuring);
* переориентация (Retargeting);
* обратный инжиниринг (обратное проектирование) (Reverse engineering);
* сопровождение программных продуктов (Software maintenance);
* трансляция исходного кода (Source Code Translation);
* и т.д.

Перечисленные понятия раскрывают понятие «реинжиниринг ИС», а соотносимая с ними деятельность рассматривается либо как одна из форм реинжиниринга ИС, либо как подпроцесс процесса реинжиниринга. Определения этих и других понятий приводятся в приложении к данной статье в глоссарии понятий и терминов. При этом в тех случаях, когда проявлялись отличия в определении того или иного понятия, в глоссарий включались альтернативные варианты определения, а при необходимости, и поясняющие определение комментарии.