Приднестровский Государственный Университет им. Т.Г. Шевченко

Бендерский Политехнический Филиал

Кафедра «Промышленность и информационные технологии»

**Практическая работа №7**

По УП 03.01 «Учебная практика»

ТЕМА**:** «Обратное проектирование алгоритма»

Выполнил:

студент 3 курса, БК22АР52ИС2 группы,

дневной формы обучения, специальность:

«Информационные системы и программирование»

Урыта Эдуард Игоревич

Руководитель учебной практики:

Лазовский Александр Валерьевич

Бендеры, 2025 г.

**Обратное проектирование алгоритма.**

**Что такое обратное проектирование?**

**Обратное проектирование алгоритма *(Reverse Engineering)*** — это процесс анализа программы или её поведения с целью восстановления исходной логики работы **без доступа к исходному коду**. Этот метод позволяет ***"заглянуть внутрь"*** алгоритма, понять, как он работает, какие операции выполняет, и воссоздать его структуру.

Чаще всего этот процесс выполняется по поведению алгоритма, входным и выходным данным, либо по скомпилированному *(исполняемому)* файлу.

**Зачем применяется?**

Обратное проектирование используется в разных сферах:

* **Информационная безопасность** — исследование вредоносных программ *(вирусов, троянов).*
* **Восстановление старого *ПО*** — если исходный код утерян, но нужно продолжать поддержку программы.
* **Изучение закрытых алгоритмов** — *например,* *сжатия, кодирования, шифрования.*
* **Оптимизация и аудит** — выявление слабых мест в чужих или собственных решениях.
* **Подготовка к олимпиадам и хакатонам** — *умение* ***«читать»*** *алгоритм по поведению* — *ценный навык.*

**Этапы обратного проектирования.**

## 1. Сбор информации.

Это первый этап, на котором мы собираем все возможные данные, чтобы понять, как работает алгоритм.

**Входные и выходные данные:**

* ***Что делаем:*** Анализируем, какие данные принимает программа на вход и какие результаты она выдает на выход. Это поможет понять зависимость между входом и выходом, выявить закономерности в обработке данных.
* ***Пример:*** Если на вход подаются числа, а на выходе — отсортированные, скорее всего, это алгоритм сортировки.

**Логи:**

* ***Что делаем:*** Если есть логи работы программы *(журналы)*, анализируем их, чтобы понять, какие действия выполняет программа на разных этапах. Логи часто содержат подробности о процессе выполнения *(например, успешное завершение или ошибки)*.
* ***Пример:*** В логах могут быть записи вроде: *Start processing, Error in row 5, File loaded successfully.*

**Поведение при разных условиях:**

* ***Что делаем:*** Запускаем программу в разных условиях, с разными входами. Это может быть проверка на корректность при разных данных *(например, пустой список, максимальные/минимальные значения и т. д.).*
* ***Пример:*** Если программа должна обрабатывать списки разного размера, смотрим, как она себя ведет с малыми, средними и большими объемами данных.

**Ошибки, задержки и ответы:**

* ***Что делаем:*** Обращаем внимание на ошибки и задержки, анализируем их причину. Иногда ошибки или замедления могут дать подсказки о том, как устроен алгоритм или где он может быть уязвим.
* ***Пример:*** Если программа выдает ошибку при делении на ноль, это может означать, что она не учитывает такие случаи в логике.

## 2. Формирование гипотез.

На этом этапе мы формируем предположения о том, что делает алгоритм и какие структуры данных используются.

**Что делает алгоритм?**

* ***Что делаем:*** Основываясь на собранной информации, пытаемся понять общую задачу, которую решает алгоритм.
* ***Пример:*** Если на вход подается массив чисел, а на выходе — отсортированный массив, гипотеза может быть, что алгоритм сортирует.

**Какие структуры данных могут использоваться?**

* ***Что делаем:*** Определяем, какие структуры данных могут быть задействованы в решении задачи. Это могут быть *массивы, списки, стеки, очереди, деревья и т. д.*
* ***Пример:*** Если алгоритм работает с деревом, возможно, это бинарное дерево поиска.

**Есть ли известные паттерны?**

* ***Что делаем:*** Проверяем, использует ли алгоритм известные паттерны *(например, сортировка, бинарный поиск, хеширование)*. Это помогает сузить круг возможных решений.
* ***Пример:*** Если программа сортирует данные, проверяем, использует ли она быструю сортировку, сортировку слиянием или другие алгоритмы.

## 3. Анализ повторяющихся закономерностей.

На этом этапе мы исследуем, как алгоритм реагирует на разные входные данные.

**Сравнение поведения на разных входах:**

* ***Что делаем:*** Подаем на вход разные данные и анализируем поведение алгоритма. Это может включать в себя тестирование граничных случаев.
* ***Пример:*** Проверяем, как алгоритм работает с пустым массивом, с массивом из одного элемента, с массивом с одинаковыми значениями и так далее.

**Построение таблиц вход–выход:**

* ***Что делаем:*** Строим таблицы, в которых отображаем соответствие входных данных и выходных результатов. Это помогает выявить закономерности и зависимости.
* ***Пример:*** Вход: **[5, 3, 8]**, Выход: **[3, 5, 8]** — *для алгоритма сортировки*.

**Нахождение инвариантов:**

* ***Что делаем:*** Ищем инварианты — это такие данные или состояния программы, которые не изменяются на протяжении выполнения алгоритма. Это может помочь понять, какие операции выполняются.
* ***Пример:*** В алгоритме сортировки инвариантом может быть тот факт, что после каждой итерации списка остаются отсортированными элементы.

## 4. Построение модели.

После того как гипотезы сформированы, мы начинаем строить модель алгоритма.

**Сначала — *псевдокод, блок-схемы, логические правила:***

* ***Что делаем:*** Описываем алгоритм в абстрактной форме: *с помощью псевдокода, блок-схем или логических выражений.*
* ***Пример:*** Если алгоритм сортировки, *можно написать псевдокод для быстрой сортировки или сортировки слиянием*.

**Затем — *возможная реализация на языке программирования:***

* ***Что делаем:*** Преобразуем *псевдокод в реальную программу, используя язык программирования.* Это помогает проверить, насколько точно мы восстановили алгоритм.
* ***Пример:*** Реализуем алгоритм на ***Python, C++***, ***Java*** *или* ***любом другом языке.***

## 5. Тестирование.

Последний этап — проверка, работает ли наша модель и соответствует ли она реальному алгоритму.

**Сравнение результата своего алгоритма с реальным:**

* ***Что делаем:***Сравниваем результаты работы нашего восстановленного алгоритма с результатами оригинального.
* ***Пример:*** Проверяем, дает ли наша сортировка тот же результат, что и оригинал.

**Анализ ошибок и корректировка модели:**

* ***Что делаем:*** Если результаты не совпадают, ищем ошибки в логике и корректируем модель.
* ***Пример:*** Если результаты отличаются, возможно, мы не учли какие-то нюансы в алгоритме *(например, не обработали все случаи данных).*

**Вывод: *Обратное проектирование алгоритма*** — это мощный инструмент, который требует внимательности и системного подхода. С помощью этапов, описанных выше, можно не только понять, как работает алгоритм, но и восстановить его, улучшить и адаптировать для новых условий.