**Приложение №9**

|  |  |
| --- | --- |
| **Дата** | 18.12.2023 |
| **Тема** | 1. Разработка программы по выделенным программным модулям.  2. Анализ, разработка и реализация алгоритма |
| **Задания** | 1. Разработка программы по выделенным программным модулям.  2. Анализ, разработка и реализация алгоритма |

**1. Разработка программы по выделенным программным модулям**

В качестве модульной структуры программы принято использовать древовидную структуру, включая деревья со сросшимися ветвями. В узлах дерева размещаются модули, а стрелки показывают подчиненность модулей (т.е. в тексте модуля, из которого она исходит, имеется ссылка на модуль, в который она входит)

Модульная структура программы должна включать и совокупность спецификаций модулей, образующих эту программу. *Спецификация*программного модуля содержит:

* синтаксическую спецификацию его входов, позволяющую построить обращение к нему на используемом языке программирования
* функциональную спецификацию модуля (описание всех функций, выполняемых этим модулем).

Существуют разные методы разработки структуры программы. Обычно используют 2 метода:

* Метод восходящей разработки
* метод нисходящей разработки

Метод восходящей разработки. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня, в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в принципе в таком же (восходящем) порядке, в каком велось их программирование.

На первый взгляд такой порядок разработки программы кажется вполне естественным: каждый модуль при программировании выражается через уже запрограммированные непосредственно подчиненные модули, а при тестировании использует уже отлаженные модули. Не рекомендуется, т.к.:

* для программирования какого-либо модуля совсем не требуется текстов используемых им модулей - для этого достаточно, чтобы каждый используемый модуль был лишь специфицирован (в объеме, позволяющем построить правильное обращение к нему), а для тестирования его возможно используемые модули заменять их имитаторами (заглушками).
* каждая программа в какой-то степени подчиняется некоторым внутренним для нее, но глобальным для ее модулей соображениям (принципам реализации, предположениям, структурам данных и т.п.), что определяет ее концептуальную целостность и формируется в процессе ее разработки. При восходящей разработке эта глобальная информация для модулей нижних уровней еще не ясна в полном объеме, поэтому очень часто приходится их перепрограммировать.
* при восходящем тестировании для каждого модуля (кроме головного) приходится создавать ведущую программу (модуль), которая должна подготовить для тестируемого модуля необходимое состояние информационной среды и произвести требуемое обращение к нему. Это приводит к большому объему "отладочного" программирования и в то же время не дает никакой гарантии, что тестирование модулей производилось именно в тех условиях, в которых они будут выполняться в рабочей программе.

Метод *нисходящей разработки*.

Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модуля самого верхнего уровня (головного), переходя к программированию какого-либо другого модуля только в том случае, если уже запрограммирован модуль, который к нему обращается. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же (нисходящем) порядке.

Положительные стороны

При таком порядке разработки программы вся необходимая глобальная информация формируется своевременно, т.е. ликвидируется весьма неприятный источник просчетов при программировании модулей.

Существенно облегчается и тестирование модулей, производимое при нисходящем тестировании программы. Первым тестируется головной модуль программы, который представляет всю тестируемую программу и поэтому тестируется при "естественном" состоянии информационной среды, при котором начинает выполняться эта программа. При этом все модули, к которым может обращаться головной, заменяются на их имитаторы. И*митатор модуля -*простой программный фрагмент, сигнализирующий, о самом факте обращения к имитируемому модулю с необходимой для правильной работы программы обработкой значений его входных параметров и с выдачей, если это необходимо, заранее запасенного подходящего результата. После завершения тестирования и отладки головного и любого последующего модуля производится переход к тестированию одного из модулей, которые в данный момент представлены имитаторами, если таковые имеются. Для этого имитатор выбранного для тестирования модуля заменяется на сам этот модуль и добавляются имитаторы тех модулей, к которым может обращаться выбранный для тестирования модуль. При этом каждый такой модуль будет тестироваться при "естественных" состояниях информационной среды, возникающих к моменту обращения к этому модулю при выполнении тестируемой программы. Таким образом большой объем "отладочного" программирования заменяется программированием достаточно простых имитаторов используемых в программе модулей. Кроме того, имитаторы удобно использовать для подыгрывания процессу подбора тестов путем задания нужных результатов, выдаваемых имитаторами.

**2. Анализ, разработка и реализация алгоритма:**

**Что такое анализ?**  
Анализируя алгоритм, можно получить представление о том, сколько времени займет решение данной задачи при помощи данного алгоритма. Одну и ту же задачу можно решить с помощью различных алгоритмов. Анализ алгоритмов дает нам инструмент для выбора алгоритма.  
Результат анализа алгоритмов — не формула для точного количества секунд или компьютерных циклов, которые потребует конкретный алгоритм. Нужно понимать, что разница между алгоритмом, который делает N + 5 операций, и тем, который делает N + 250 операций, становится незаметной, как только N становится очень большим.  
**Классы входных данных**  
При анализе алгоритма выбор входных данных может существенно повлиять на его выполнение. Скажем, некоторые алгоритмы сортировки могут работать очень быстро, если входной список уже отсортирован, тогда как другие алгоритмы покажут весьма скромный результат на таком списке. А вот на случайном списке результат может оказаться противоположным. Поэтому не будем ограничиваться анализом поведения алгоритмов на одном входном наборе данных. Практически нужно искать такие данные, которые обеспечивают как самое быстрое, так и самое медленное выполнение алгоритма. Кроме того, полезно оценивать и среднюю эффективность алгоритма на всех возможных наборах данных.  
**Наилучший случай**  
Время выполнения алгоритма в наилучшем случае очень часто оказывается маленьким или просто постоянным, поэтому подобный анализ проводится редко.  
**Наихудший случай**  
Анализ наихудшего случая чрезвычайно важен, поскольку он позволяет представить максимальное время работы алгоритма. При анализе наихудшего случая необходимо найти входные данные, на которых алгоритм будет выполнять больше всего работы.  
**Средний случай**  
Анализ среднего случая является самым сложным, поскольку он требует учета множества разнообразных деталей. В основе анализа лежит определение различных групп, на которые следует разбить возможные входные наборы данных. На втором шаге определяется вероятность, с которой входной набор данных принадлежит каждой группе. На третьем шаге подсчитывается время работы алгоритма на данных из каждой группы. Время работы алгоритма на всех входных данных одной группы должно быть одинаковым, в противном случае группу следует подразбить.

**Разработка алгоритма**— особый метод для создания математического способа решения проблемы.

**Разработка алгоритма** — это отождествление и объединение во множество решений теорий [исследования операций](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B9), например [динамическое программирование](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [разделяй и властвуй](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D1%8F%D0%B9_%D0%B8_%D0%B2%D0%BB%D0%B0%D1%81%D1%82%D0%B2%D1%83%D0%B9_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)). Методиками разработки и реализации разработки алгоритма будут шаблоны[[1]](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2#cite_note-1), такие как [шаблонные методы](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) и [декораторы](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), использование [структуры данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), а также имя и сортировка списков. Сейчас использование разработки алгоритма можно найти в поисковых процессах сканирования Интернета, маршрутизации пакетов и кэшировании.

Одним из наиболее важных качеств алгоритма является его эффективность по времени выполнения и по используемой памяти.

**Выводы:** Разработал программы по выделенным программным модулям. Проанализировал, разработал, и реализовал алгоритм.

Студент: / Ульянов Никита Анатольевич