Автореферат

Наименование Алгоритмизация

Автор А.В.Михалькевич

Специальность Алгоритмизация для Web,

Анотация

В дисциплине Алгоримизация для web рассматриваются основы и синтаксис языков PHP и JavaScript. И особенности их использования в разработке web-приложений.

Anotation in English

The Algorithmization for the Web discipline covers the basics and syntax of the PHP and JavaScript languages. And the features of their use in the development of web applications.

Ключевые слова php, js, javascript, algoritm, алгорим, алгоритмизация, github

Количество символов 39540

Содержание

Введение

Алгоритмизация – процесс превращения задачи в последовательность строго определенных действий

1 Основы алгоритмизации

Этапы

Они функционируют через ряд четко определенных этапов, каждый из которых способствует достижению конечной цели:

1. Ввод. На первом этапе необходимо определить входы, которые будет использовать алгоритм. Входы — это данные, с которыми будет работать алгоритм. Это может быть что угодно — от одного значения до сложной структуры данных.
2. Обработка. Это основной этап, на котором алгоритм выполняет операции над входными данными с помощью ряда вычислительных шагов. Для эффективной обработки данных на этом этапе используются логические и арифметические вычисления. В фазе обработки часто присутствуют важнейшие подэтапы:

2.1. Принятие решений. В различные моменты обработки данных необходимо принимать решения, основываясь на определенных условиях. Этот подэтап включает в себя управление потоком алгоритма на основе условных операторов, что приводит к различным путям в алгоритме.
2.2. Циклирование. Во многих алгоритмах определенные шаги необходимо повторять несколько раз, пока не будет выполнено определенное условие. Петля позволяет алгоритму выполнять одни и те же шаги многократно, оптимизируя процесс и экономя время.

3. Выход. После обработки входных данных с помощью различных вычислительных и условных шагов алгоритм выдает выходной сигнал. Этот выход является результатом работы алгоритма и используется для решения проблемы или выполнения поставленной задачи.
4. Окончание. У алгоритма должна быть определенная точка остановки, чтобы он не работал бесконечно. Когда все шаги успешно выполнены и получен результат, алгоритм достигает точки завершения.

Пример

1. Вход. Алгоритм получает данные о температуре от датчика, расположенного в доме.
2. Обработка.
2.1. Принятие решения. Алгоритм принимает решение о состоянии системы отопления на основе полученных данных о температуре:
Если температура ниже определенного нижнего порога, он включает систему отопления.
Если температура выше определенного верхнего порога, он выключает систему отопления.
Если температура находится между двумя пороговыми значениями, сохраняется текущее состояние системы отопления.
2.2. Циклическая работа. Алгоритм проверяет данные о температуре каждую секунду, чтобы решить, нужно ли предпринимать какие-либо действия.
3. Выход. В данном сценарии выходом может быть состояние системы отопления в любой момент времени (включена, выключена или не изменена) и любая корректировка температуры в доме. Однако не каждый алгоритм должен выдавать наблюдаемый результат, так как некоторые из них могут работать в фоновом режиме для поддержания определенного состояния или условий.
4. Окончание. У этого алгоритма нет фиксированной точки завершения, поскольку он продолжает работать до тех пор, пока система отопления активна, или пока кто-то не выключит систему отопления на панели управления.

Задачи алгоритмизации

1. Необходимы для эффективного решения сложных задач. Одна большая задача может быть разбита на несколько более простых подзадач.
2. Они помогают автоматизировать процессы и сделать их более надежными, быстрыми и простыми в исполнении.
3. Алгоритм должен определить, в каком порядке выполнять шаги для достижения цели. Это включает в себя определение условий, при которых выполнять тот или иной шаг.
4. Позволяют компьютерам выполнять задачи, которые человеку было бы сложно или невозможно решить вручную.
5. Выбор операций и ввод-выводных данных.

Принципы

1. Детерминированность (должен быть определен и однозначен). Для каждого входного значения должны быть предусмотрены определенные выходные значения, и алгоритм должен дать одинаковый результат для одних и тех же входных данных.
2. Ремонтопригодность. Удобство обслуживания заключается в том, насколько легко алгоритм может быть обновлен или модифицирован. Поддерживаемый алгоритм позволяет плавно обновлять и изменять его, обеспечивая сохранение актуальности и функциональности в течение долгого времени.
3. Имеет конечное количество шагов.
4. Эффективность — важнейший аспект хорошего алгоритма. Она подразумевает оптимальное использование вычислительных ресурсов, включая время и память. Эффективный алгоритм выполняет задачи быстро, экономя время и энергию.
5. Корректность. В нем не должно быть ошибок и багов, чтобы обеспечить надежную работу.
6. Хороший алгоритм должен быть разработан с учетом требований безопасности, обеспечивая защиту конфиденциальных данных и противодействуя атакам злоумышленников.
7. Стабильность очень важна; она гарантирует, что алгоритм будет надежно и стабильно работать в различных условиях, сохраняя свою точность и надежность с течением времени, даже при изменении входных данных.
8. Алгоритм должен быть разделен на логические блоки или модули, которые могут быть понятными и повторно используемыми. Это позволяет упростить разработку, тестирование и сопровождение алгоритма.

Блок-схема

В алгоритмизации понятие блок-схема используется для графического представления структуры алгоритма. Она помогает визуализировать последовательность выполнения операций и принимаемых решений в алгоритме.

Блок-схема состоит из блоков, соединенных линиями. Каждый блок представляет отдельную операцию или решение, а линии показывают порядок и направление выполнения операций. В блок-схеме можно использовать различные типы блоков, такие как блоки начала и конца, блоки ввода данных, блоки вывода результатов, блоки выполнения операций и блоки условных операторов.

Блок-схема с линейной логикой

Вопросы: Понятие алгоритма Особенности языка программирования JavaScript Отличия и области применения языков программирования PHP и JavaScript Классификация алгоритмов Пример линейных алгоритмов в JavaScript Разветвляющиеся алгоритмы в JavaScript Циклические алгоритмы в JavaScript Основы синтаксиса JavaScript Типы данных в программировании Операторы и выражения в JavaScript Переменные и области видимости в JavaScript Асинхронное программирование в JavaScript Типы строк в PHP Основные объекты JavaScript Функциональное программирование Протокол http Http-запросы Тернарный оператор в PHP и JavaScript Объявление и вызов функций Возвращаемое значение функции Передача аргументов по ссылке в PHP

2 Синтаксис PHP и шаблонизация проекта

Синтаксис PHP

Алгоритмизация в PHP

Установка и настройка сервера

Переменные

Строки, функции, управляющие конструкции

Циклы

Условия

Функция require()

Вывод переменных

Шаблонизация проекта

1. Шаблон главной страницы, файл index.html переименовываем в index.php

2. В папку с проектом добавляем папку templates, содержащую файлы top.php, bottom.php

Структура проекта:

/media/css

/templates/top.php

/bottom.php

index.php

3. Содержимое файла index.php

```
require_once('templates/top.php');
//меняющаяся часть шаблона
require_once('templates/bottom.php')
```

Файлы top.php и bottom.php содержат неизменную часть шаблона. Изначально, это в основном html-код.

Вопросы: Особенности языка программирования PHP Разветвляющиеся алгоритмы в PHP Пример линейных алгоритмов в PHP Циклические алгоритмы в PHP Основы синтаксиса PHP Операторы и выражения в PHP Суперглобальные переменные в PHP Синхронное программирование в PHP

3 Массивы, работа с данными

Методы и функции работы с массивами,

Способы сохранения данных

Сохранение в файл, в базу, в суперглобальную переменную на стороне сервера и клиента

Вопросы: Массивы и функции обработки массивов Массивы в JavaScript, свойства и методы для работы с массивами

4 Функция: определение, виды. Возврат значений из функции.

Фу́нкция в программировании, или подпрограмма — фрагмент программного кода, к которому можно обратиться из другого места программы.

Функции в PHP

Приведём пример синтаксиса для описания функций:

```
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Пример функции.\n";
    return $retval;
}
```

Функция принимает информацию в виде списка аргументов — выражений через запятую. Аргументы вычисляются слева направо перед фактическим вызовом функции (*энергичное* вычисление).

PHP поддерживает передачу аргументов по значению (по умолчанию), передачу аргументов по ссылке, и значения по умолчанию. Списки аргументов переменной длины и именованные аргументы также поддерживаются.

Передача массива в функцию:

```
function takes_array($input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

Передача аргументов по ссылке

По умолчанию аргументы в функцию передаются по значению (это означает, что если вы измените значение аргумента внутри функции, то вне её значение всё равно останется

прежним). Если вы хотите разрешить функции модифицировать свои аргументы, вы должны передавать их по ссылке.

Перед именем аргумента в описании функции указывают амперсанд &, если требуется, чтобы аргумент передавался по ссылке:

```
function add_some_extra(&$string)
{
    $string .= 'и кое-что ещё.';
}

$str = 'Это строка, ';
add_some_extra($str);
echo $str;    // Выведет «Это строка, и кое-что ещё.»
```

Значения аргументов по умолчанию

Функция умеет определять для аргументов значения по умолчанию, в этом помогает синтаксис, который похож на синтаксис присваивания значения переменной. Функция присвоит параметру значение по умолчанию, только если в параметр не передали аргумент; обратите внимание, что функция не присваивает параметру значение по умолчанию при передаче аргумента со значением null.

```
function makecoffee($type = "капучино")
{
    return "Готовим чашку $type.\n";
}

echo makecoffee();
echo makecoffee(null);
echo makecoffee("эспрессо");
```

Подробнее - https://www.php.net/manual/ru/functions.arguments.php

5 Массивы $_POST и $_GET в PHP. Обработка форм. Загрузка и перемещение файлов

Формы — это часть языка HTML. Формы нужны для передачи данных от клиента на сервер. Чаще всего формы используются для регистрации пользователей, заполнения анкет, оформления заказа в интернет магазине, и так далее.

Через формы можно отправлять как простую текстовую информацию, так и файлы.

Большую часть времени программирования на PHP вы будете так или иначе работать с формами и данными из них.

HTML описывает то, из каких элементов состоит форма, и как она выглядит. Но без принимающей стороны, то есть сервера, который принимает эти данные и обрабатывает их нужным образом, создавать формы нет никакого смысла.

PHP содержит множество средств для работы с формами. Это позволяет очень просто решать типичные задачи, которые часто возникают в веб-программировании:

Регистрация и аутентификация пользователя;
Отправка комментариев на форумах и социальных сетях;
Оформление заказов.

Практически любой современный сайт содержит как минимум несколько разных HTML-форм.

Подробнее о формах HTML - https://www.w3schools.com/html/html_forms.asp

Это очень простая форма, состоящая из трёх полей и одной кнопки отправки.

Почти весь приведённый код описывает внешний вид и содержание формы, но следует обратить внимание на два атрибута тега `form`, которые нужны для указания на способ обработки данных:

method — этот атрибут используется для определения метода HTTP, который будет использован для передачи данных на сервер. Вы уже знакомы с HTTP-методом GET, предписывающим серверу просто вернуть определённый документ.

Метод POST сообщает о намерении передать на сервер некоторую информацию, что, впрочем, не отменяет последующее получение контента.

action — содержит адрес PHP-скрипта, который должен обработать эту форму.

После нажатия на кнопку отправки (type="submit"), браузер выполняет POST запрос со введёнными данными на адрес, указанный в атрибуте action.

Обработка формы

После отправки формы управление передаётся PHP-скрипту, который должен получить переданные данные, выполнить с ними какие-либо действия (например, сохранить в базе данных) и показать результат.

Результатом может быть какое-нибудь сообщение об успешном завершении операции, например, «ваши данные успешно отправлены».

Поэтому требуется в первую очередь научиться получать данные из формы в сценарии.

В PHP это делается легко — все данные из формы находятся в глобальном ассоциативном массиве `$_POST`. Этот массив всегда будет неявно присутствовать в сценарии, если он был загружен по методу POST.

Каждое поле из формы будет находиться в массиве, где ключом будет значение атрибута name, а значением содержимое поля. Например, чтобы вывести из формы всю информацию на экран, можно написать такой сценарий:

```
        if (isset($_POST)) {
                print("Имя: " . $_POST['name']);
                print("
Email: " . $_POST['email']);
                print("
Сообщение: " . $_POST['message']);
        }
```

Функция isset служит для определения, существует ли переданная ей переменная. Так мы проверяем, что сценарий загружен методом POST, то есть была отправлена форма.

Как правило, после обработки формы в PHP, сценарий должен переадресовать пользователя на другую страницу. Это связано с тем, что если форма была отправлена через метод POST, то после обновления страницы данные будут отправлены ещё раз, а это, в большинстве случаев, нежелательное поведение.

Отправка файлов

Кроме текстовой информации, существует возможность отправлять на сервер файлы любых типов. Пример формы для загрузки файла:

`action="form.php" enctype="multipart/form-data"`

Два важных отличия в сравнении с формой без отправки файла:

> Добавился новый атрибут enctype, который всегда должен иметь значение `multipart/form-data`. Если его не будет, то файл не отправится.
> Сам файл загружается при помощи поля с типом *file*: `input type="submit" name="send" value="Отправить файл"`.

В PHP загруженный файл будет доступен в другом специальном массиве — `$_FILES`.

В PHP загруженный файл будет доступен в другом специальном массиве — $_FILES.

```
        if (isset($_FILES['avatar'])) {
                $file = $_FILES['avatar'];
                print("Загружен файл с именем " . $file['name'] . " и
размером " . $file['size'] . " байт");
        }
```

PHP автоматически сохраняет все загруженные файлы во временную папку на сервере. Но хранить там файлы нельзя, потому что эта директория периодически очищается, и ссылку на такой файл нельзя дать на сайте. Решение здесь только одно — переместить загруженный файл в другую папку. Перемещение файла всегда выполняют сразу после загрузки.

Для начала нужно убедиться, что в рабочей директории проекта существует папка для хранения загруженных файлов. Пусть она называется `uploads`.

Перемещение загруженного файла

Для перемещения файла нужно знать, где он находится сейчас, и адрес папки, в которую он будет переноситься.

С текущим адресом всё крайне просто — он уже находится в массиве `$_FILES`. Новый адрес файла, в свою очередь, состоит из пути к папке и имени файла. Так как папка uploads находится там же, где и текущий сценарий, получить путь к ней можно так: `dirname(__FILE__)`.

Код для перемещения файла в новую папку:

```
        $current_path = $_FILES['avatar']['tmp_name'];
        $filename = $_FILES['avatar']['name'];
        $new_path = dirname(**__FILE__**) . '/' . $filename;
```

```
move_uploaded_file($current_path, $new_path);
```

Функция `move_uploaded_file()`выполняет два действия:

Проверяет, что файл действительно загружен через форму.
Перемещает загруженный файл по новому адресу.

Валидация формы

Валидация формы — это проверка содержимого её полей. Задача такой проверки — убедиться, что необходимые поля заполнены, а значения в них соответствуют ожидаемому формату.

Так, например, при регистрации пользователя на сайте, он должен заполнить поля с адресом электронной почты и придумать себе пароль. Оба поля обязательны к заполнению, но значение из поля email также должно быть корректным email-адресом.

Помимо текстовых значений формы, можно проверять формат и размер загружаемых файлов.

Вопросы: Функции подключения файлов в PHP GET-запрос и область применения POST-запрос и область применения Функции чтения и записи файлов Обработка данных форм в PHP

6 Session и Cookie в PHP

Протокол HTTP является протоколом "без сохранения состояния". Это означает, что когда пользователь открывает сначала одну страницу сайта, а затем переходит на другую страницу этого же сайта, то основываясь только на средствах, предоставляемых протоколом HTTP невозможно установить, что оба запроса относятся к одному пользователю. Необходим механизм, при помощи которого можно было бы отслеживать информацию о пользователе в течение одного сеанса связи с Web-сайтом. В PHP такой механизм реализован при помощи сессий (**session**) и куки (**cookies**).

Сессии и куки предназначены для хранения сведений о пользователях при переходах между несколькими страницами. При использовании сессий данные сохраняются во временных файлах на сервере. Файлы с cookies хранятся на компьютере пользователя, и по запросу отсылаются браузером (browser) серверу.

Использование сессий и куки очень удобно и оправдано в таких приложениях как интернет-магазины, форумы, доски объявлений и блоги когда, во-первых, необходимо сохранять информацию о пользователях на протяжении нескольких станиц, а, во-вторых, своевременно предоставлять пользователю новую информацию.

COOKIES

Для установки Cookies используется функция `SetCookie()`. Для этой функции можно указать шесть параметров, первый из которых является обязательным:

name - задает имя (строк), закрепленное за Cookie;
value - определяет значение переменной (строка);
expire - время "жизни" переменной (целое число). Если данный параметр не указать, то Cookie будут "жить" до конца сессии, то есть до закрытия браузера. Если время указано, то, когда оно

наступит, Cookie самоуничтожится.
path - путь к Cookie (строка);
domain - домен (строка). В качестве значения устанавливается имя хоста, с которого Cookie был установлен;
secure - передача Cookie через защищенное HTTPS-соединение.
Обычно используются только три первые параметра.

Пример установки Cookies:

```
// Устанавливаем Cookie до конца сессии:
SetCookie("Test","Value");

// Устанавливаем Cookie на один час после установки:
SetCookie("My_Cookie","Value",time()+3600);
```

При использовании Cookies необходимо иметь в виду, что Cookies должны устанавливаться до первого вывода информации в браузер (например, оператором echo или выводом какой-либо функции). Поэтому желательно устанавливать Cookies в самом начале скрипта. Cookies устанавливаются с помощью определенного заголовка сервера, а если скрипт выводит что-либо, то это означает, что начинается тело документа. В результате Cookies не будут установлены и может быть выведено предупреждение. Для проверки успешности установки Cookies можно использовать такой метод:

```
// Устанавливаем Cookie до конца сессии:
// В случае успешной установки Cookie, функция SetCookie возвращает TRUE:
if (SetCookie("Test","Value")) echo "
```

Cookies успешно установлены!

"; Функция SetCookie() возвращает TRUE в случае успешной установки Cookie. В случае, если Cookie установить не удается SetCookie() возвратит FALSE и возможно, предупреждение (зависит от настроек PHP). Пример неудачной установки Cookie:

```
echo "Hello";
// Функция SetCookie возвратит FALSE:
if (SetCookie("Test","Value")) echo "
```

Cookie успешно установлен!

"; else echo "

Cookie установить не удалось!

"; // Выводит 'Cookie установить не удалось!'.

Cookie установить не удалось, поскольку перед посылкой заголовка Cookie мы вывели в браузер строку "Hello".

Чтение значений Cookies

Получить доступ к Cookies и их значениям достаточно просто. Они хранятся в

суперглобальных массивах и $_COOKIE и $HTTP_COOKIE_VARS.

Доступ к значениям осуществляется по имени установленных Cookies, например:

```
echo $_COOKIE['my_cookie'];
```

СЕССИИ

Сессия - это сеанс, во время которого система идентифицирует зашедшего на сайт пользователя. Для этого надо выдать ему уникальный идентификатор и попросить передавать его с каждым запросом. Идентификатор - это обычная переменная. По умолчанию ее имя - PHPSESSID.

Для того, чтобы иметь доступ к переменным сессии на любых страницах сайта, надо написать только одну строчку в самом начале КАЖДОГО файла, в котором нам нужны сессии:

```
session_start();
```

И далее обращаться к элементам массива $_SESSION:

```
$_SESSION['test']='Hello world!';
```

Например, проверка авторизации будет выглядеть примерно так:

```
session_start();
if ($_SESSION['authorized']<>1) {
header("Location: /auth.php");
exit;
}
```

Удаление переменных из сессии

Если в настройках PHP включена переменная register_globals=off, то достаточно написать

```
unset($_SESSION['var']);
```

Если же нет, то тогда рядом с ней надо написать:

```
session_unregister('var');
```

Сравнение механизмов session и cookie

Сравнение механизмов идентификации пользователей при помощи Cookies и Session

Cookies — это просто пара имя-значение, которые сервер может оставить у клиента (браузера). Наглядно:

1. Приходит клиент, спрашивает у сервера страницу.

2. Сервер по IP определяет геолокацию клиента и в заголовках ответа может установить cookies. Например, выдав заголовок: Set-Cookie: country=russia. При следующем обращении серверу уже не надо проверять IP, он может взять готовое значение из cookies, если они разрешены у клиента.

Сессии — это механизм, который позволяет отличить одного клиента от другого и хранить связанные с ним данные.

Как правило, сессии реализуются используя cookies и идентификаторы сессий. Т.е. сервер со своей стороны создает уникальный идентификатор, например, «1a2b3c» (session_id), а клиента просит его запомнить. Обычно — при помощи cookies, говоря что-то в духе Set-Cookie: PHPSESSID=1a2b3c (где «PHPSESSID» — имя сессии). Со своей стороны сервер где-то в файле хранит различные данные, которые ему приказано связывать с этой сессией.

Если куки у пользователя отключены, то перед отдачей страницы пользователю сервер переписывает все ссылки в ней, добавляя к ним параметр «?PHPSESSID=1a2b3c». Таким образом сервер получает идентификатор сессии.

Фактически механизм Cookies хранит данные о пользователе на стороне клиента, т.е. у самого пользователя, а механизм сессий хранит у пользователя только идентификатор клиента. Сами данные, связанные с этим пользователем, хранятся на сервере.

Вопросы: Работа с файлами в PHP Сохранение данных на сервере Сохранение данных на компьютере пользователя

Заключение

Список использованных источников

Приложения

1. [электронный документ] 6601773f65348_Алгоритмизация.docx
2. [Учебная программа] **Учебная программа** 6601775e5ac64_основы алгоритмизации.docx