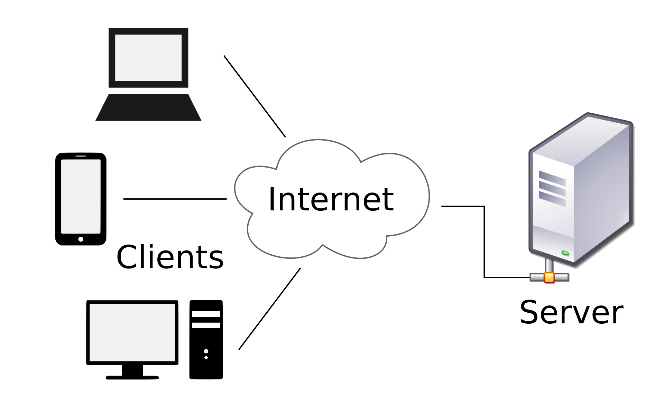
# Изпитна тема No 14: Интернет програмиране

* Мрежови протоколи (IP, TCP, UDP).
* Видове HTTP заявки (GET / POST / PUT / DELETE / PATCH).
* Клиент-сървърна комуникация. Основни тагове в HTML.
* Селектори и основни правила в CSS. Създаване на семантични страници.
* Създаване на адаптивно (responsive) оформление на страници.
* Увод в JavaScript – работа с обекти и събития.
* Принципи и манипулиране на DOM.

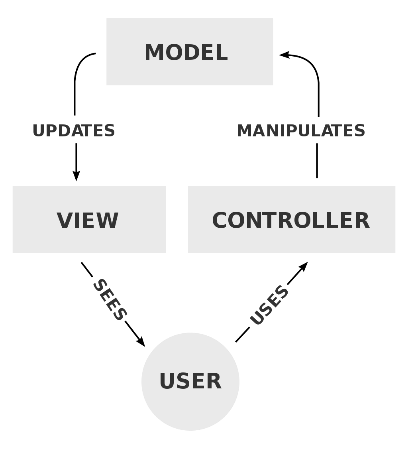
Дидактически материали: Компютър с подходяща версия на интегрирана среда за разработка от изучаваните и уеб браузър. Например, Visual Studio, Visual Studio Code, SublimeText, Atom и др.

1. Обяснява и диференцира различните протоколи.
   1. IP, Internet Protocol, е network-layer протокол който осъществява връзка м-ду апликации, поддържа TCP и UDP. То е предназначен да описва адреси и връзки м-ду компютри в една мрежа.
   2. TCP, Transmission Control Protocol e transport-layer протокол, първо се осъществява връзка м-ду двата края, и се пращат пакети последователно. TCP връзки са едно към едно. Предназначен е за употреба в апликации, нуждаещ се от надеждна връзка, за сметка на скорост, загубени пакети се пре-пращат.
   3. UPD е transport-layer протокол, пакети те се пращат към дестинацията без да се осъществи предварителна връзка. UDP връзките могат да бъдат едно към едно или чрез Broadcasting, едно към много. Предназначен е да предостави информация по най-бързия начин, независими как, от къде или дали дестинацията е готов да приеме данни, UDP връзките не са много надеждни и често губят пакети, които по стандартна имплементация, не се пращат на ново.
2. Дефинира понятието HTTP заявка, прави изводи за различните HTTP методи и избира метод за конкретна ситуация.
   1. HTTP заявка е процес за искане на информация или пращане на информация към даден Уеб Сървър. Това е основния начин на комуникация в модерния уеб.
   2. GET заявка е за искане на информация. Пример за това би било искането на информация за дадена страница в сайт.
   3. POST заявка е за изпращане на нова информация. Това би било пращането на съобщения или коментар в блог или чат апликация.
   4. PUT заявка е за пращане на бинарни ресурси, като файлове. Това би било качването на снимка в социална медия.
   5. DELETE заявка е за изтриване на даден ресурс. Пример за това би било изтриването на коментар от социална медия.
   6. PATCH заявка е за промяна на даден ресурс. Това би било промяната на текста на даден блог пост.
3. Обяснява и представя графично клиент-сървърната комуникация.
   1. 
   2. Клиенти се свързват с централен сървър чрез локална или интернет връзка, и комуникират директно и само със сървъра.
   3. Комуникация към други услуги или клиенти се осъществява чрез посредничество на сървъра.
4. Различава смисъла на употребата и необходимостта от HTML, CSS и JavaScript.
   1. HTML представлява структурата на една страница, без нея няма как да се структурира информацията. Това би било скелета и мускулите на един човек.
   2. CSS описва визуалната му презентация, без нея всичко би било чист текст. Това би било кожата и косата на един човек.
   3. JavaScript добавя интерактивност и интелигентност на страницата, без нея няма как да има интерактивност. Това би било мозъка и нервната система на един човек.
5. Дефинира и използва коректно HTML тагове.
   1. Прекалено много са, тук да се ползва онлайн ресурс.
6. Задава свойства на HTML компонентите чрез CSS.
   1. Компонента се избира чрез един от следните начини:
      1. Името на тага
      2. Class – Класа даден на компонента при създаване или по-късно чрез JavaScript.
      3. Id - Идентификатора даден на компонента при създаване или по-късно чрез JavaScript.
      4. Релативно местоположение в документа – Описват се взаимоотношенията на компоненти чрез техните имена, класове и идентификатори.
7. Описва и демонстрира употребата на семантични елементи за създаване на семантична страница.
   1. Семантични елементи и страници са документи или частни на документа, описани подробно по начин разбираем за машини/автоматизирани системи.
   2. Предназначени са автоматизация на комуникация и улеснение за намиране на информация.
   3. Създаването му изисква описването на значителните елементи на една страница в семантична форма (съдържание, композиция, предназначение, произход и т.н.).
   4. Пример за това би било страница със снимки на различни цветя, при правилно семантични описани елементи, търсенето на снимки за рози би довел даден потребител в тази страница, ако има снимки на рози и са описани чрез семантични тагове.
8. Обяснява и демонстрира начините за създаване на адаптивен (responsive) дизайн.
   1. Използват се модерни методологии за структуриране, като FlexBox или CSS Grids.
   2. Използват се CSS Media Queries за да се определи формата и резолюцията на екрана и се използват различни стилове според това.
9. Работа с обекти и събития в JavaScript. Ученикът пише код и модифицира съществуващ такъв спрямо конкретна задача.
10. Демонстрира манипулирането на DOM.

# Изпитна тема No 15: Интернет програмиране

* MVC модел. Бази данни и ORM технологии. Реализиране на CRUD операции.
* Създаване на шаблонни изгледи.
* Автентикация и авторизация.
* Управление на сесии и бисквитки.
* Често срещани уязвимости в сигурността на уеб приложенията (SQL инжекция, XSS атака, CRSF, LFI и RFI, DDoS атака, MITM атака).
* Принципи на REST API.
* Работа с REST API в JSON/XML формат.
* Използване на AJAX в REST API.

Дидактически материали: Компютър с подходяща версия на интегрирана среда за разработка от изучаваните, уеб браузър и сървърен софтуер. Наличие на подходяща версия на СУБД от изучаваните.

1. Обяснява и представя графично MVC модела.
   1. 
   2. MVC модела е предназначен за създаването на модерен потребителски софтуер с модулярни части. Потребителя интерактува с контролер, който манипулира модели, резултата се показва на потребителя под формата на изглед, генериран от данните на модела.
2. Описва същността и демонстрира употребата на ORM технологиите.
   1. ORM, Object Relational Model, е методология за управление на база от данни от програмен код. То работи като се престава данните от базата от данни като обекти в език за програмиране, най-често, обектно ориентиран език.
   2. Операциите извършени в езика за програмиране се превръщат в заявки към базата от данни, най често под формата на SQL заявки.
3. Реализира CRUD операции.
   1. CRUD. Create, Read, Update, Delete операции са основните начини за четене и манипулиране на данни в една база от данни.
4. Дефинира и създава шаблонен изглед.
   1. Шаблонен изглед е крайния резултат което се показва от потребителя, шаблон което показва данни от даден модел.
5. Различава автентикацията и авторизацията.
   1. Автентикация определя кой е потребителя. Осъществява се чрез потребителски имена и пароли, хардуерни автентикатори и др.
   2. Авторизация определя към какво има достъп, даден потребител, кои страници може да посети и какви действия може да извърши в рамките на текущите си права. Осъществява се чрез потребителски роли и системи за права.
6. Диференцира сесиите от бисквитките.
   1. Сесиите са информация, запазена от сървъра, относно действията на потребителя. Тази информация е достъпна вътрешно на сървъра, за определено време, то е временно и най-често изтича до час след приключването на потребителя.
   2. Бисквитките са информация, запазено от браузъра на клиента, и се пращат на сървъра със повторни заявки, те могат да се нагласят да изтекат след определено време или да се безсрочни.
7. Обяснява и дава пример за различните уязвимости в сигурността на уеб приложенията. Прави изводи за предотвратяването им.
   1. Brute Force атаки, са нападения към потребителски пароли, пробата на безбройни комбинации от пароли над дадено потребителско име, с цел, да се разгадае паролата. Могат да се спрат като се ограничи броя на опити, ползването на CAPTCHA системи или системи без потребителски пароли.
   2. MITM атаки, Man in the Middle, инфраструктурна атака който работи когато нападателя играе като посредник м-ду потребител и мрежата към сървъра. Когато връзката е не-криптирана, позволява кражбата на информация. Спира се като се ползва HTTPS.
   3. SQL Injection атаки се уязвимости причинени от употребата на SQL заявки без санитация на входните данни от потребител/нападател, могат да бъдат спрени чрез употребата на ORM система или чрез санитиране на входните данни към софтуерната система.
   4. И други.
8. Обяснява основните принципи на REST API.
   1. REST, REstful State Transfer, е стил за софтуерна архитектура, предназначен като ограничителни правила към стандартни HTTP методи. REST има за цел да създаде едновидни методологии за комуникация между системи, и да осъществи стандартизирани начини за интеркомуниакция.
   2. Основни принципи биват:
      1. Ползват се **Униформени интерфейси**.
      2. Базиран на **Клиент-Сървър архитектури**.
      3. Всички комуникации да съдържат всичката информация нужна за дадена задача.
      4. Всички отговори да се маркират дали могат да се кешират или не.
      5. Системите да съществуват в отделни нива, независими от други нива, с което на интеракатуват.
9. Възпроизвежда и обяснява код за работа с REST API в JSON/XML формат.
10. Възпроизвежда и обяснява код за използване на AJAX в REST API.

# Изпитна тема No 16: Софтуерно инженерство

* Етапи в разработката на софтуер. Софтуерни изисквания и документация.
* Случаи на употреба (use cases) и Потребителски истории (user stories).
* Спецификация на изискванията.
* Методологии за разработка на софтуер - Waterfall и Agile.
* Scrum.
* Scrum артефакти, екипи, роли и събития.
* Инструменти за управление на екип (Jira или други изучавани).
* Работа със системи за сорс-контрол (git, svn или друга изучавана).
* Работа с чужд код. Преглед на чужд код (code review).
* Видове софтуерно тестване.

Дидактически материали: -

1. Описва и обяснява етапите в разработката на софтуер.
   1. Планиране
      1. Събиране на бизнес изисквания
      2. Срещи между заинтересованите страни
      3. Разработчиците трябва добре да се запознаят с изискванията на клиента
      4. Предвиждане на евентуалните проблеми, които могат да възникнат
   2. Анализ
      1. Екипът дефинира детайлно целия проект, като проверява неговата приложимост
      2. Работният процес е разделен на малки задачи, за да може разработчици, тестъри, дизайнери и мениджъри на проекта да дадат оценка за своите задачи
      3. Те оценят дали дадена задача е приложима относно цена, време, функционалност, надеждност и др.
   3. Дизайн
      1. Дефинира се дизайнът на софтуера на база събраните изисквания от фазата “Планиране”
      2. Определяне на системните и хардуерните изисквания за проекта
      3. Създаване на стратегия за тестване
      4. Кое да се тества?
      5. Как да се тества?
      6. Съдържа цялостна софтуерна архитектура на продукта, както и дизайн на базата от данни
   4. Разработка & имплементация
      1. Най-дългият етап в софтуерната разработка
      2. След определяне дизайна на продукта работата се разпределя на малки задачи
      3. Всеки разработчик получава собствени задачи
      4. Всеки разработчик изпълнява своите задачи
   5. Тестване
      1. След като софтуерът е разработен, той се подлага на тестване
      2. Софтуерът не преминава в следващ етап на разработка, ако тестърите не го одобрят
      3. Намерените бъгове и проблеми се документират и софтуерът се връща в етап на разработка
   6. Инсталация и поддръжка
      1. Софтуера се предава на клиентите, за да се инсталира на техните устройства
      2. След това софтуера преминава в етап на поддръжка
      3. Поправят се възникнали грешки
      4. Софтуера се подновява от време на време
2. Посочва и различава методологии за разработка на софтуер. Прави заключения и изводи за значението на методологии за разработка на софтуер.
   1. Waterfall
      1. Всеки етап трябва да приключи преди да започне следващия
      2. Няма припокриване между етапите
      3. Линеарен подход за разработка на софтуер
      4. По принцип резултата на даден етап играе ролята на начало за следващия
      5. Състои се от вече споменатите етапи на софтуерната разработка
   2. Agile (гъвкава методология)
      1. Комбинация от итеративни и инкрементални модели за разработка
      2. С фокус върху задоволяване желанията на клиента чрез бърза доставка на работещи сегменти от продукта
      3. Разбива продукта на малки инкрементални части
      4. Тези части се снабдяват в итерации
      5. Всяка итерация трае средно между 1 и 3 седмици
      6. Всяка итерация включва вече споменатите етапи на разработка на софтуер
      7. След всяка итерация работещият продукт се показва на клиента и заинтересованите страни
      8. Умение да се реагира на промени
      9. Основно се характеризира с адаптивност
   3. Scrum
      1. Най-популярният agile фреймуърк
      2. Фокусиран върху това как да се управляват задачи в среда за разработка базирана върху екипна работа
      3. Използва итеративен и инкрементален подход за разработка
      4. Кратки периоди на итерациите
      5. Сравнително лесна имплементация
      6. Фокус върху бързите и чести доставки на работещ софтуер
      7. Състои се от Scrum екипи и техните роли, събития, артефакти и правила
      8. Всеки компонент във фреймуърка има специфично значение
   4. Kanban
      1. Визуален начин за управление на задачи и работен процес
      2. Използва kanban табло с колони и карти
      3. Всяка карта представлява определена задача
      4. Колоните организират задачите според техния прогрес или етапа на разработка, в който се намират
      5. Разработен от Toyota през 40-те
      6. Лесно е да се види по какви задачи работи всеки един човек в организацията
      7. Мениджърите на екипи могат бързо да назначат задача на даден екип, когато той е без такава
      8. Съвместим с agile
      9. Използван заедно със scrum, образува мощен подход за разработка на софтуер
   5. Непрекъсната интеграция
      1. Практика при разработката на софтуер
      2. Всеки екип трябва подсигурява, че успешен build и последващо тестване са проведени за всяка промяна по кода в софтуерна програма
      3. Концепцията гарантира премахването на проблема с откриване на грешки в по-късен етап от разработката
      4. Гарантира, че промените по кода никога не се правят в изолация
      5. Отговаря на въпросите
         1. Работят ли всички компоненти на софтуера заедно, както би трябвало?”
         2. Минават ли всички тестове след последните промени?
         3. Каква част от кода подлежи на автоматизирано тестване?
3. Обяснява и свързва SCRUM артефакти, събития и роли. Обобщава и прави изводи за SCRUM артефакти, събития и роли.
   1. SCRUM артефактите представляват основните факти за един проект, те най-често се описват като Product Backlog, Sprint Backlog и Product Assessment.
   2. Описва се какво трябва да се направи за проекта, какво трябва да се прави за текущия спринт, и как се разбира да ли проекта е готов.
4. Описва и обобщава съдържанието на софтуерната документация. Прави изводи и заключения за значението на софтуерната документация в работата на програмистите.
5. Различава случаи на употреба (use cases) и потребителски истории (user stories).
   1. Use Case е описание на употребата на една система, под-система или продукт от един потребител, включва потока на действие и информация чрез визуална диаграма.
   2. User Story описва мотивации на един потребител за до постигне даден резултат както и действието което има в предвид за да постигне целта си.
6. Познава и демонстрира инструменти за управление на работата на екип.
   1. Trello, Jira
7. Демонстрира команди за работа със система за сорс-контрол.
   1. Git
8. Демонстрира процеса на преглед на чужд код (code review).
9. Описва и различава видове софтуерно тестване. Обобщава и диференцира употребата на различните видове софтуерно тестване.
   1. Компонентно тестване – Систематично тестването на индивидуални класове и техните методи и валидации.
   2. Интеграционно тестване – Проверка на взаимодействието между различни модули на един проект или система.
   3. Регресивно тестване – Включва проверки за предотвратяване на предишни грешки или загуба на бързодействие в проекта.
   4. И други.

# Изпитна тема No 17: Функционално програмиране

* Функционална парадигма на програмиране.
* Понятие за „странични ефекти“ в програмирането.
* Входно/изходни операции.
* „Състояние“ на програмата.
* Функции и рекурсия.
* Чисти функции.
* Работа със списъци: „глава“ и „опашка“.

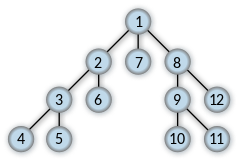
Дидактически материали: Компютър с подходяща версия на интегрирана среда за разработка.

1. Описва функционалната парадигма на програмиране. Различава функционална, процедурна и обектно-ориентирана парадигма. Прави изводи за значението на функционалната парадигма на програмиране.
   1. Функционално програмиране е форма на програмиране където инструкции се описват под формата на функции, които приемат данни, и връщат, нови, преобразовани варианти на тези данни. При тези езици, функциите се третират като съществени, и често се допуска подаването на функции като параметри на други функции, както и да се пазят като променливи. В основата си, това подпомага с избягването на тъй наречени, странични ефекти.
   2. За разлика от процедурни и обектно ориентирани езици, при повечето функционални езици за програмиране, данните не могат да се помнят (мутират), могат само да се създадат нови, модифицирани копия.
   3. Функционалното програмиран намира употреба в среди където мутирането на данни може да води до грешки в изпълнението или резултата на програмата. Те най-вече намират употреба в научни и академични среди, както и в индустриални среди.
2. Дефинира понятието „страничен ефект“ в програмирането.
   1. Странични ефекти са резултати от мутирането на данни от една функция или метод, който в последствие променя действието на друга функция, метод или процедура. Пример за това би било метод който проверява всички числа в даден масив, срещу външна променлива и връща само по-големите, ако външната променлива се промени, един и същ вход може да върни други резултати, според стойността на тази външна променлива.
3. Посочва и демонстрира начини за извършване на входно/изходни операции чрез функционален език за програмиране.
   1. Тези IO операции се правят чрез функциите getLine, putStr и putStrLn.
   2. Четене става по следния начин: input <- getLine, след което input може да се конвертира в друг тип от данни по този начин: let number = (read input::Integer).
   3. Извеждане на информация става с по следния начин: putStrLn (text), ако това което ще се изведе не е текстова информация, трябва да се ползва show, по следния начин: putStrLn (show number)
4. Обяснява понятието „състояние“ на програма.
   1. Състояние в отношение на функционално се отнася за променливите в една програма. Променливите в фун. Езици могат да се зададат само веднъж, след което не могат повече да се променят.
5. Описва и демонстрира създаване на функции.
   1. Функции, при езици като Haskell, най-често се създават чрез писането на името на функцията и нареждането на входните му параметри. Опционално може и да се опише сигнатурата на функцията,
6. Разработва рекурсивни функции чрез функционален език за програмиране.
7. Дефинира понятието „чиста функция“.
   1. Чиста функция е функция, който ще връща един и същ резултат, при едни и същи входни данни. Функция, чиито изход е зависим само и само от входните му параметри. Те се срещат често при функционалното програмиране.
8. Дава пример и обобщава обработката на списъци с функции.
   1. Списъците в един език като Haskell представляват структури от линейни данни, които принципно се обработват чрез рекурсия
9. Различава глава и опашка на списък.
   1. Главата на един списък, в език като Haskell представлява първия елемент от този списък, докато опашката е под-група от всички останали елементи освен първи елемент.
10. Анализира фрагмент/и от код и идентифицира и поправя правилно грешките в написания програмен код, така че да реши поставената задача. Допълва кода, ако и когато това е необходимо.

# Изпитна тема No 18: Алгоритми и структури от данни

* Алчни алгоритми.
* Рекурсия и рекурсивни алгоритми.
* Комбинаторни алгоритми: вариации, комбинации, пермутации.
* Динамично оптимиране.
* Теория на графите.
* Алгоритми върху графи.

Дидактически материали: Компютър с подходяща версия на софтуерното обезпечение и интегрирана среда за разработка от изучаваните. Например, Visual Studio, IntelliJ IDEA, PyCharm, Eclipse и др.

1. Описва понятието алчен алгоритъм. Дава пример за алчни алгоритми.
   1. Алчни алгоритми са клас от алгоритми които са добре в това да намерят локално оптимизирани решения, като те във всяка дадена стъпка, правят най-доброто потенциално решение до сега, и никога не се връщат назад.
   2. Примери за такъв вид алгоритъм би було, алгоритъма за монетите и алгоритъма за раницата.
2. Описва понятието рекурсия. Дава пример за рекурсивни алгоритми с пълно изчерпване.
   1. Рекурсия е термина което се ползва за клас от методи и функции, които директно (пряко) или индиректно (косвено) извикват себе си.
   2. Пример за такива методи/функции е намиране на Факториела на число, тъй като всяко число на Факториел, се състои от продукта на всички предишни числа на факториел: 5! = 5 \* 4!, 4! = 4\* 3!, и т.н.  
        
      static int Factorial(int input)  
      {  
       return (input < 2) ? 1 : (input \* Factorial(input-1));  
      }
3. Разработва програми за решаването на задачи с помощта на алчни алгоритми.
   1. Пример за това е задачата за монети, задачата изисква видове монети с техните стойности, и желана сума, решението, при всяка стъпка, взима най-голямата възможна монета, което може да си позволи, и го слага към текущата сумата, разликата м-ду желаната сума и текущата е останалата сума.  
      Този алгоритъм приключва когато двете суми са равни.
4. Дефинира понятията от комбинаториката: вариации, комбинации и пермутации. Различава вариации, комбинации и пермутации.
   1. Вариация - Брой уникални избора на K елемента от група с N елемента, където К е по-малко от N. Реда на елементите **НЕ** е от значение.  
      Примерно: Кои 10 от 100 човека ще бъдат избрани да участват в шоу,
   2. Комбинация – Брой уникални подреждания на K елемента от група с N елемента, където К е по-малко от N. Реда на елементите е от значение.  
      Пример: Кои 3 три участника ще спечелят на в състезание (1во, 2ро и 3то място) от 10 участника.
   3. Пермутация – Брой уникални подреждания на N елемента, без повторения.  
      По колко различни начина може да се подредят всички 10 книги.
5. Обяснява техниката на динамичното оптимиране.
   1. В програмирането, Динамично Оптимиране, също се нарича Динамично Програмиране, то е начин на решаване на проблеми като рекурсивно се пречупват на по прости задачи, чиито решение водят во холистично решение, най-често като се изключват потенциални подходи които не водят до оптимално решение, ранно в етапите на рекурсията.
6. Разработва програми за решението на едномерни и двумерни задачи с помощта на техники от динамичното оптимиране (задача за раницата, най-малка подредица, оптимален път в лабиринт и др.).
7. Дефинира основни понятия от теория на графите: граф, ориентиран и неориентиран граф, връх, ребро, път в граф, цикъл в граф.
   1. Граф е структура, състоящ се от точки и връзки помежду им. То най-често се реализира като списък на съседство или матрица на връзки.
   2. Ориентиран граф е граф където връзките м-ду точките на графа са принципно еднопосочни.
   3. Неориентиран граф е граф където връзките м-ду точките на графа са **ВИНАГИ** двупосочни.
   4. Възел, Node е точка/дестинация в графа.
   5. Ребро, Edge, e връзка между две точки в графа.
   6. Тежест, Weight, е цената на изминаване през дадено ребро.
   7. Цикъл, Loop, е ребро който започва и завършва на един и същ възел.
   8. Път в граф е описание на точките и връзките през които трябва да се премине от една точка до друга.
8. Обяснява и представя графично алгоритми за обхождане в ширина и дълбочина на графи (BFS и DFS).
   1. DFS, Depth-First Search, е алгоритъм за търсене в дълбочина за дървовидни структури от данни. При този метод на търсене, рекурсивно се гледат децата на всеки възел, до като има такива, след което се гледат съседите, преди да се връща назад към родителите.  
      
   2. BFS, Breadth-First Search, е алгоритъм за търсене в широчина за дървовидни структури от данни. При този метод на търсене, се започва като добавим началния възел в опашка, и започваме, циклично да обработваме елемента което на върха на опашката. Добавят се всички деца на елемента на върха, обработва се самия елемент, и се преминава на следващия, този алгоритъм продължава до като има елементи в опашката.  
      
9. Анализира фрагмент/и от код и идентифицира и поправя правилно грешките в написания програмен код, така че да реши поставената задача. Допълва кода, ако и когато това е необходимо.