Spis treści

Rozdział 1

Algorytmy i struktury danych

1.1 Paradygmat i przykłady programowania generycznego (rodzajowego).

Парадигма программирования с использованием обобщений

Программирование с использованием обобщений — это парадигма программирования, которая позволяет писать код, который может работать с различными типами данных. Это делает код более переиспользуемым и безопасным в работе с типами.

Пример использования обобщений

```
public class Box<T> {
    private T content;

    public void setContent(T content) {
        this.content = content;
    }

    public T getContent() {
        return this.content;
    }
}
```

В этом примере, ‘Box‘ — это обобщённый класс, который может хранить

объекты любого типа. ‘T‘ — это типовой параметр, который будет заменён на реальный тип при использовании класса ‘Box‘.

```
Box<String> stringBox = new Box<>();
stringBox.setContent("Hello, world!");
System.out.println(stringBox.getContent()); // Output: Hello, world!

Box<Integer> integerBox = new Box<>();
integerBox.setContent(123);
System.out.println(integerBox.getContent()); // Output: 123
```

Здесь ‘Box<String>‘ и ‘Box<Integer>‘ — это два разных типа, созданных на основе обобщённого класса ‘Box‘.

1.2 Algorytmy sortowania.

Сортировка - это фундаментальная задача в информатике, которая заключается в упорядочении набора данных на основе определенного критерия. Важно отметить, что эффективность алгоритма сортировки может сильно варьироваться в зависимости от его характеристик и типа входных данных.

Bubble Sort

Сортировка пузырьком работает, перебирая список от начала до конца, сравнивая пары соседних элементов и меняя их местами, если они находятся в неправильном порядке. Этот процесс повторяется, пока весь список не будет отсортирован.

```
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1]) {
                    // swap arr[j+1] and arr[j]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
    }

```

Insertion Sort

Алгоритм сортировки вставками работает, делая в каждой итерации список частично отсортированным. В каждом шаге алгоритма выбирается один из элементов и вставляется на правильное место в уже отсортированной части списка.

```
1   public static void insertionSort(int[] arr) {
2       int n = arr.length;
3       for (int i = 1; i < n; ++i) {
4           int key = arr[i];
5           int j = i - 1;
6
7           while (j >= 0 && arr[j] > key) {
8               arr[j + 1] = arr[j];
9               j = j - 1;
10          }
11          arr[j + 1] = key;
12      }
13  }
14
```

Selection Sort

Сортировка выбором работает, повторно находя минимальный (или максимальный, в зависимости от порядка сортировки) элемент из нераспределённой части списка и помещая его в конец отсортированной части. Этот процесс продолжается, пока все элементы не будут отсортированы.

```
1   public static void selectionSort(int[] arr) {
2       int n = arr.length;
3
4       for (int i = 0; i < n-1; i++) {
5           int min_idx = i;
6           for (int j = i+1; j < n; j++)
7               if (arr[j] < arr[min_idx])
8                   min_idx = j;
9
10          // Swap the found minimum element with the first element
11          int temp = arr[min_idx];
12          arr[min_idx] = arr[i];
13          arr[i] = temp;
```

```
        }
    }

```

Quick Sort

Быстрая сортировка — это эффективный алгоритм сортировки, который работает по принципу "разделяй и властвуй". В этом алгоритме выбирается "опорный"элемент, и остальные элементы распределяются так, чтобы все элементы, меньшие опорного, шли до него, а все элементы, большие опорного, шли после.

```
    public static void quickSort(int[] arr, int begin, int end) {
        if (begin < end) {
            int partitionIndex = partition(arr, begin, end);

            quickSort(arr, begin, partitionIndex-1);
            quickSort(arr, partitionIndex+1, end);
        }
    }

    private static int partition(int[] arr, int begin, int end) {
        int pivot = arr[end];
        int i = (begin-1);

        for (int j = begin; j < end; j++) {
            if (arr[j] <= pivot) {
                i++;

                int swapTemp = arr[i];
                arr[i] = arr[j];
                arr[j] = swapTemp;
            }
        }

        int swapTemp = arr[i+1];
        arr[i+1] = arr[end];
        arr[end] = swapTemp;

        return i+1;
    }

```

Merge Sort

Сортировка слиянием также работает по принципу "разделяй и властвуй". В этом алгоритме список разбивается на две половины, каждая из которых сортируется отдельно, а затем две отсортированные половины объединяются в один список.

```
1   public static void mergeSort(int[] arr, int begin, int end) {
2       if (begin < end) {
3           int mid = (begin + end) / 2;
4
5           mergeSort(arr, begin, mid);
6           mergeSort(arr , mid+1, end);
7
8           merge(arr, begin, mid, end);
9       }
10  }
11
12  private static void merge(int[] arr, int begin, int mid, int end) {
13      // merging code here...
14  }
15
```

1.3 Strategia „dziel i zwyciężaj” budowania algorytmów.

"Разделяй и властвуй"— это методология, используемая в проектировании алгоритмов. Она основана на рекурсивном разделении задачи на подзадачи, которые являются меньшими версиями исходной задачи. Эти подзадачи решаются индивидуально, а затем их решения комбинируются для получения ответа на исходную задачу. Стратегия состоит из следующих основных шагов:

"Разделяй и властвуй это методология, используемая в проектировании алгоритмов. Она включает рекурсивное разбиение задачи на подзадачи, решение этих подзадач и комбинирование их решений для получения решения исходной задачи. Стратегия состоит из следующих шагов:

1. Разделение: Задача разбивается на подзадачи.

2. Властвование: Подзадачи решаются. Если подзадачи все еще слишком

большие, они могут быть разбиты еще больше.

3. Комбинирование: Решения подзадач объединяются, чтобы получить окончательное решение.

Примеры алгоритмов, которые используют эту стратегию, включают быструю сортировку, сортировку слиянием, алгоритмы для нахождения минимального остовного дерева и алгоритмы быстрого возведения в степень.

```
public static double power(double a, int n) {
    if (n == 0) {
        return 1;
    } else if (n % 2 == 0) {
        double temp = power(a, n / 2);
        return temp * temp;
    } else {
        return a * power(a, n - 1);
    }
}
```

В этом примере, функция ‘power‘ рекурсивно вызывает себя, уменьшая степень ‘n‘ с каждым вызовом. Это уменьшение степени и есть "разделение"проблемы. "Властвование"происходит при выполнении умножения в каждом вызове функции, а "комбинирование"решений подзадач происходит автоматически, так как каждый рекурсивный вызов возвращает результат, который умножается на результат другого вызова (или на основание ‘a‘, в случае нечетной степени).

1.4 Algorytmy typu zachłannego.

Жадные алгоритмы — это класс алгоритмов оптимизации, которые решают задачу пошаговым выбором локально оптимального решения в каждый момент времени с надеждой, что набор этих локальных решений приведет к глобальному оптимальному решению.

Основная идея жадного алгоритма заключается в принятии решения, которое кажется наиболее оптимальным в данный момент, и полном игнорировании последствий этих решений для последующих шагов.

1.5 Algorytmy z nawrotami.

Алгоритмы с возвратами, или "backtracking"алгоритмы, представляют собой стратегию решения проблем, которая включает обход дерева возможных решений для задачи. Этот обход может быть описан следующими шагами:

1. Выбор опции из набора возможных решений.
2. Проверка, приведет ли выбор этой опции к решению проблемы. Если да, то решение найдено.
3. Если выбор не привел к решению, возвращение к шагу 1 и выбор следующей опции.
4. Если все опции были проверены и ни одна из них не привела к решению, возвращение на шаг назад и выбор следующей опции на этом уровне.

Этот подход используется в решении многих задач, включая задачу о восьми ферзях, задачу о гамильтоновом цикле, задачу о раскраске графа и решение головоломок, таких как "Судоку".

1.6 Grafy, drzewa, kopce – charakterystyka i przykłady zastosowania.

Графы

Граф - это структура данных, которая состоит из вершин, некоторые из которых соединены ребрами. Графы могут быть направленными (где каждое ребро направлено от одной вершины к другой) или ненаправленными.

Примеры использования графов включают моделирование сетей (таких как интернет или социальные сети), анализ связности и пути в географии или транспортной сети и многое другое.

Деревья

Дерево - это специальный тип графа, который не содержит циклов. Дерево имеет корневую вершину и каждая вершина имеет ноль или более дочерних вершин.

Деревья широко используются в компьютерных науках. Например, деревья используются для представления иерархических отношений (как в системах управления файлами), для обработки и анализа данных (в бинарных деревьях поиска, синтаксических деревьях в компиляторах), в машинном обучении и т.д.

Кучи

Куча - это специальный вид бинарного дерева (обычно полного бинарного дерева), где каждая вершина имеет значение, которое больше или равно (в случае максимальной кучи) или меньше или равно (в случае минимальной кучи) значений своих дочерних вершин.

Кучи часто используются для создания эффективных алгоритмов сортировки, таких как heapsort, а также для создания очередей с приоритетами, которые используются во многих алгоритмах, таких как алгоритм Дейкстры для нахождения кратчайшего пути в графе.

1.7 Definicja i klasy złożoności obliczeniowej – czasowej i pamięciowej.

Временная сложность

Временная сложность алгоритма определяет количество операций, которое алгоритму потребуется выполнить, в зависимости от размера входных данных. Она обычно выражается с использованием "Big O notation", которая описывает верхнюю границу временной сложности в худшем случае.

Для анализа временной сложности алгоритма нужно сначала определить, что считать "операцией". В простейшем случае это может быть любая простая операция, такая как сложение, умножение, сравнение и т.д. Затем нужно посчитать, сколько таких операций будет выполнено в худшем случае для входных данных определенного размера.

Например, простой алгоритм поиска элемента в неотсортированном массиве будет иметь временную сложность O(n), где n - это размер массива. Это потому, что в худшем случае нам придется просмотреть весь массив, чтобы найти нужный элемент.

Сложность по памяти

Сложность по памяти алгоритма определяет количество памяти, которое алгоритму потребуется занять, в зависимости от размера входных данных. Как и временная сложность, сложность по памяти также обычно оценивается асимптотически и выражается с использованием нотации O большого.

Для анализа сложности по памяти нужно посчитать, сколько памяти требуется для хранения входных данных и любых дополнительных структур данных, которые используются в процессе выполнения алгоритма.

Например, простой алгоритм сортировки вставками имеет сложность по памяти O(1), потому что он сортирует массив на месте и не требует дополнительной памяти, независимо от размера входного массива.

Rozdział 2

Bazy danych

2.1 Normalizacja baz danych – pierwsza, druga i trzecia postać normalna.

Нормализация базы данных - это процесс проектирования структуры базы данных с целью уменьшить дублирование данных и избегать проблем с добавлением, удалением и модификацией данных. В этом процессе используются правила или "формы", известные как нормальные формы.

Первая нормальная форма (1NF):

- Все колонки должны быть атомарными, то есть каждое значение в колонке должно быть неделимым.
- Каждая колонка должна иметь уникальное имя.
- Все строки должны быть уникальными, не должно быть дубликатов.

Вторая нормальная форма (2NF):

- Таблица уже находится в 1NF.
- Все колонки, не являющиеся ключами, полностью зависят от первичного ключа. Иначе говоря, не должно быть такой ситуации, когда значение колонки, не являющейся ключом, зависит от одной части составного ключа.

Третья нормальная форма (3NF):

- Таблица уже находится в 2NF.
- Не существует транзитивных зависимостей. То есть ни одна неключевая колонка не должна зависеть от других неключевых колонок. Все зависимости должны исходить только от ключа.

Имеются и более высокие уровни нормализации, такие как BCNF, 4NF, 5NF и 6NF, каждый из которых решает определенные проблемы в проектировании баз данных. Однако первые три нормальные формы наиболее важны и часто используются для большинства приложений баз данных

2.2 Modele baz danych (logiczny, relacyjny, fizyczny).

relacyjny

W terminologii matematycznej - baza danych jest zbiorem relacji. Stąd historycznie pochodzi nazwa relacyjny model danych i relacyjna baza danych. W matematyce definiuje się relację jako podzbiór iloczynu kartezjańskiego zbiorów wartości. Reprezentacją relacji jest dwuwymiarowa tabela złożona z kolumn i wierszy.

Założenia modelu relacyjnego:

- Liczba kolumn/atrybutów/pól (synonimy) jest z góry ustalona.
- Z każdą kolumną jest związana jej nazwa (np. FirstName) oraz dziedzina (np.TEXT(20)), określająca zbiór wartości, jakie mogą wystąpić w kolumnie.
- Na przecięciu wiersza/krotki/rekordu (synonimy) i kolumny znajduje się pojedyncza (atomowa) wartość należąca do dziedziny kolumny
- Wiersz reprezentuje jeden rekord informacji np. osobę.
- W modelu relacyjnym abstrahujemy od kolejności wierszy (rekordów) i kolumn (pól w rekordzie).

Klucz główny: dla każdej tabeli musi być określony klucz główny, będący jedno-

znacznym identyfikatorem. Może to być jedna lub więcej kolumn, w których wartości jednoznacznie identyfikują cały wiersz. Klucz główny w tabeli może być tylko jeden. Klucz jednoznaczny ma te same właściwości co klucz główny, ale ich może być w tabeli więcej niż jeden. Klucz obcy - jedna lub więcej kolumn, których wartości występują również jako klucz główny/jednoznaczny w tej samej/innej tabeli i są interpretowane jako wskaźniki do wierszy w tej drugiej tabeli

Логическая модель базы данных описывает данные так, как они видны пользователям. Это абстракция, которая помогает упростить взаимодействие пользователя с базой данных. В логической модели определяются сущности (таблицы), их атрибуты (поля) и связи между сущностями.

Например, в базе данных компании мы можем определить сущности "Сотрудники", "Отделы"и "Проекты". Сущность "Сотрудники"может иметь атрибуты "Имя", "Фамилия", "Дата рождения", "Позиция", и т.д. Сущность "Отделы"может включать "Название отдела", "Местоположение", и т.д. Затем, связи между этими сущностями определяют, как они взаимодействуют друг с другом, например, один отдел может включать множество сотрудников. реляцийная это прокаченная логическая модель

Физическая модель базы данных, с другой стороны, описывает, как данные фактически хранятся в системе, включая информацию о физическом хранении и доступе к данным. Физическая модель базы данных может включать следующие детали: Способ хранения данных: это может быть последовательное хранение, индексное хранение, хеширование или другие методы. Детали файловой системы: в каких файлах и на каких дисках хранятся данные, как они организованы и так далее. Методы индексации: используются ли индексы для ускорения поиска данных, какие индексы используются, как они структурированы и так далее. Компрессия данных: используется ли сжатие данных для экономии места, какой метод сжатия используется и так далее. Шифрование данных: если данные шифруются для защиты, как это делается, какой алгоритм шифрования используется и так далее.

Физическая модель базы данных — это подробное представление о том,

как данные хранятся в компьютерной памяти или на диске. Она описывает структуру хранения, методы доступа, пути обращения к данным и так далее. Физическая модель обычно не доступна для пользователей базы данных, она служит для оптимизации работы СУБД (системы управления базами данных).

2.3 Rodzaje zapytań w języku SQL.

SQL (Structured Query Language) является стандартным языком для работы с реляционными базами данных. Он позволяет выполнять различные виды запросов для манипулирования и извлечения данных. В SQL есть несколько основных типов запросов:

Запросы выборки данных (SELECT):

```
SELECT column1, column2, ...
FROM table_name;
```

Эти запросы используются для извлечения данных из базы. Вы можете выбрать одну или несколько колонок, и применять различные условия и операции сортировки.

Запросы вставки данных (INSERT INTO):

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

С помощью этих запросов можно добавить новые строки в таблицу.

Запросы обновления данных (UPDATE):

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

UPDATE позволяет изменять значения в уже существующих строках таблицы.

Запросы удаления данных (DELETE):

```
DELETE FROM table_name WHERE condition;
```

DELETE используется для удаления строк из таблицы.

Запросы создания новой таблицы (CREATE TABLE):

```
CREATE TABLE table_name (
column1 datatype,
column2 datatype,
column3 datatype,
....
);
```

CREATE TABLE создает новую таблицу с заданными именами столбцов и типами данных.

Запросы изменения структуры таблицы (ALTER TABLE):

```
ALTER TABLE table_name
ADD column_name datatype;
```

ALTER TABLE позволяет добавлять, удалять или изменять столбцы в существующей таблице.

2.4 Funkcje w języku SQL.

В SQL имеются встроенные функции, которые позволяют выполнять расчеты, трансформации и манипуляции с данными. Они могут быть разделены на следующие категории:

Агрегатные функции (Aggregate Functions) - функции, работающие с множеством строк и возвращающие одно значение:

- AVG() - вычисляет среднее значение набора чисел.
- COUNT() - подсчитывает количество строк в выборке.
- MAX() - находит максимальное значение в наборе данных.
- MIN() - находит минимальное значение в наборе данных.

- SUM() - суммирует значения в наборе чисел.

Функции для работы со строками (String Functions) - функции, предназначенные для манипулирования строками:

- CONCAT() - соединяет две или более строк в одну.
- LENGTH() - возвращает длину строки.
- LOWER() и UPPER() - переводит строку в нижний или верхний регистр соответственно.
- TRIM() - удаляет пробелы из начала и конца строки.

Функции NULL-значений (NULL Functions) - функции, предназначенные для работы с NULL-значениями:

- ISNULL() - проверяет, является ли значение NULL.
- COALESCE() - возвращает первое не-NULL значение из списка.

2.5 Transakcje w bazach danych.

Транзакция в контексте баз данных — это логическая единица работы, которая может включать одну или несколько операций над данными (например, чтение, вставка, обновление, удаление).

Транзакции важны для обеспечения надежности и целостности данных. Они обеспечивают выполнение следующих четырех свойств, известных как свойства ACID:

Атомарность (Atomicity): Это свойство гарантирует, что транзакция считается одной неделимой единицей работы. Это означает, что либо все операции в транзакции выполняются успешно, либо, если хотя бы одна операция не удается, не выполняется ни одна операция. Согласованность (Consistency): Согласованность обеспечивает, что транзакция приводит базу данных из одного согласованного состояния в другое. После транзакции все ограничения целостности в базе данных должны быть сохранены. Изолированность (Isolation): Это свойство

гарантирует, что параллельные транзакции не влияют друг на друга. Результат параллельного выполнения транзакций должен быть таким же, как если бы эти транзакции выполнялись последовательно. Долговечность (Durability): После того, как транзакция успешно завершена, ее результаты становятся постоянными и остаются в базе данных даже в случае сбоя системы. В SQL транзакции управляются с помощью команд BEGIN TRANSACTION, COMMIT и ROLLBACK. Например:

BEGIN TRANSACTION; INSERT INTO table1 (column1) VALUES ('value1'); UPDATE table2 SET column2 = 'value2' WHERE column3 = 'value3'; COMMIT;

Если в процессе транзакции что-то идет не так, можно использовать ROLLBACK, чтобы отменить все изменения, внесенные в рамках этой транзакции.

Rozdział 3

ai

3.1 Metody uczenia maszynowego.

Методы машинного обучения обычно делятся на три основные категории: обучение с учителем, обучение без учителя и обучение с подкреплением.

Обучение с учителем (Supervised Learning):

- В этом подходе модель обучается на основе обучающего набора данных, который содержит входные данные и соответствующие им ожидаемые выходные данные (метки классов или значения).
- Задача модели состоит в том, чтобы научиться прогнозировать выходные данные на основе входных данных.
- Примеры алгоритмов обучения с учителем включают линейную и логистическую регрессию, машины опорных векторов (SVM), и нейронные сети.

Обучение без учителя (Unsupervised Learning):

- Здесь модель обучается на основе набора данных, который содержит только входные данные, и нет конкретных ожидаемых выходных данных.
- Задача модели состоит в том, чтобы найти скрытые структуры или закономерности в данных, такие как группы, паттерны или правила.

- Примеры алгоритмов обучения без учителя включают K-средних, иерархическую кластеризацию и методы снижения размерности, такие как анализ основных компонентов (PCA).

Обучение с подкреплением (Reinforcement Learning):

- В этом подходе агент (модель) обучается, взаимодействуя со средой. Агент получает награды или штрафы (подкрепления) на основе своих действий, и цель состоит в том, чтобы максимизировать суммарную награду.
- Обучение с подкреплением часто используется в областях, где требуется обучение последовательности действий, таких как игры, навигация и управление роботами.

Важно отметить, что это обобщения и многие алгоритмы машинного обучения могут быть классифицированы в более чем одну категорию. Кроме того, существуют и другие типы машинного обучения, такие как полу-надежное обучение (semi-supervised learning) и обучение с активным использованием примеров (active learning), которые являются гибридами этих основных типов.

3.2 Budowa sieci neuronowych

Нейронная сеть состоит из нескольких слоёв, каждый из которых содержит набор нейронов. Каждый нейрон в свою очередь связан с нейронами в следующем слое через весовые коэффициенты.

Нейрон

Нейрон — это базовая единица в нейронной сети. Это функция, которая принимает вектор входных данных $x = (x_1, x_2, \ldots, x_n)$ и весовых коэффициентов $w = (w_1, w_2, \ldots, w_n)$, и возвращает выходное значение y. Математически это можно записать так:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.1}$$

где b - это смещение (bias), а f - функция активации, которая может быть различной (например, сигмоидальной, ReLU, гиперболического тангенса и т.д.).

Слои

В нейронной сети различают следующие основные типы слоёв:

- Входной слой: этот слой получает входные данные.
- Скрытые слои: эти слои обрабатывают входные данные. В них происходит большая часть вычислений нейронной сети.
- Выходной слой: этот слой возвращает окончательный результат работы нейронной сети.

Веса между нейронами различных слоев обычно оптимизируются с помощью метода обратного распространения ошибки (backpropagation), чтобы минимизировать ошибку между предсказанными и реальными значениями на тренировочных данных.

Rozdział 4

BSK

4.1 Funkcje skrótu (mieszające) i ich zastosowania.

Функции хэширования, или как они иногда называются, функции сокращения, являются важными инструментами в компьютерной науке и используются в различных областях, включая криптографию, поиск данных и обнаружение ошибок. Эти функции берут входные данные и преобразуют их в обычно более короткий, фиксированный размер выходных данных, называемый хешем. Хеш-функции должны быть детерминированными, что означает, что один и тот же вход всегда будет давать один и тот же выход.

Вот некоторые из применений хеш-функций:

1. Структуры данных: Хеш-функции используются в структурах данных, таких как хеш-таблицы, для быстрого доступа к данным.
2. Криптография: В криптографии хеш-функции используются для создания цифровых подписей, проверки целостности данных и сохранения паролей.
3. Кэширование: Хеш-функции могут использоваться для кэширования данных, когда нужно быстро проверить, содержится ли значение в наборе данных.
4. Обнаружение ошибок: Хеш-функции могут использоваться для обнаружения ошибок при передаче данных. Если хеш отправленных данных не

соответствует хешу принятых данных, это указывает на ошибку в данных.

4.2 Atrybuty bezpieczeństwa informacji.

Безопасность информации часто описывается с помощью трех основных атрибутов, известных как "Триада CIA Конфиденциальность, Целостность и Доступность.

- Конфиденциальность (Confidentiality): Этот атрибут гарантирует, что информация доступна только для тех лиц, которые имеют право ее просматривать. На практике конфиденциальность может быть обеспечена с помощью различных методов, включая использование паролей, шифрования и контроля доступа.
- Целостность (Integrity): Целостность означает сохранность и точность информации. Этот атрибут гарантирует, что информация не была изменена без авторизации, что означает защиту от несанкционированного или случайного изменения. Целостность может быть обеспечена с помощью контрольных сумм, хэш-функций и цифровых подписей.
- Доступность (Availability): Доступность означает, что информация и соответствующие ресурсы доступны тем лицам, которым они нужны, когда они нужны. Недоступность системы может быть результатом атаки или неполадок оборудования. Доступность обычно обеспечивается через резервное копирование, отказоустойчивость и обслуживание оборудования.

Важно отметить, что для обеспечения безопасности информации необходимо обеспечить все три атрибута. Например, система может быть конфиденциальной и доступной, но если информация в ней может быть изменена несанкционированно, система не может считаться безопасной.

4.3 Modele dystrybucji kluczy kryptograficznych.

Модели распределения ключей криптографии являются важным аспектом в обеспечении безопасного обмена информацией. Вот некоторые основные модели:

- Симметричное шифрование (Private Key Cryptography): В этой модели используется один и тот же ключ для шифрования и дешифрования сообщений. Проблема с этим подходом состоит в том, что безопасное распределение этого секретного ключа между обеими сторонами может быть сложной задачей, поскольку его необходимо передать через безопасный канал.

- Асимметричное шифрование (Public Key Cryptography): В этой модели используется пара ключей: открытый ключ для шифрования и приватный ключ для дешифрования сообщений. Открытый ключ может быть свободно распространяем, а приватный ключ должен оставаться в тайне. Преимуществом этого подхода является то, что секретный ключ никогда не передается, что уменьшает риск его перехвата.

- Инфраструктура открытых ключей (Public Key Infrastructure, PKI): Эта модель использует асимметричное шифрование, но также добавляет третью сторону, известную как сертификационный центр (CA), который выпускает цифровые сертификаты, подтверждающие владение парой ключей.

- Diffie-Hellman key exchange: Этот протокол позволяет двум сторонам сгенерировать секретный ключ через незащищенный канал без передачи самого ключа. Это достигается за счет создания временных ключей, которые затем используются для создания общего секретного ключа.

- Key Distribution Center (KDC): В этой модели существует третья доверенная сторона, которая хранит пары симметричных ключей для каждой стороны. Когда две стороны хотят общаться, они обращаются к KDC за симметричными ключами.

Выбор подходящей модели распределения ключей зависит от многих факторов, включая уровень безопасности, который необходим для конкретной задачи, доступность каналов связи и возможность использовать доверенные третьи стороны.

4.4 Rodzaje zagrożeń oraz ochrona aplikacji sieciowych

Приложения сети подвержены множеству угроз, и для обеспечения их безопасности необходимо учесть множество аспектов. Вот некоторые из основных типов угроз, а также способы их защиты:

- Атаки на отказ в обслуживании (DoS и DDoS): Эти атаки направлены на перегрузку системы или сети таким образом, что нормальные пользователи не могут получить доступ к сервису. Защита от таких атак может включать мониторинг трафика, ограничение количества запросов от одного источника или использование специализированных сервисов для защиты от DDoS.
- Прослушивание (Eavesdropping)/ Перехват (Interception): Злоумышленники могут попытаться перехватить и прочитать незашифрованные данные, передаваемые через сеть. Для защиты можно использовать шифрование данных, транслируемых через сеть, а также использовать безопасные протоколы передачи данных, такие как HTTPS.
- Внедрение кода и SQL-инъекции: Злоумышленники могут попытаться внедрить вредоносный код в приложение или использовать SQL-инъекции для доступа к базам данных. Защита включает в себя валидацию и очистку всех входных данных, использование параметризованных запросов и регулярное обновление и патчинг приложений и систем.
- Спуфинг и фишинг: Атакующие могут попытаться подделать свою идентичность или идентичность доверенного веб-сайта, чтобы получить чувствительную информацию. Защита от этих атак включает в себя обучение пользователей, использование анти-фишинговых фильтров и сертификатов SSL.
- Взлом паролей: Злоумышленники могут попытаться угадать пароли пользователей. Защита включает использование сильных паролей, ограничение числа попыток ввода пароля, использование двухфакторной аутентификации и хэширование паролей при хранении.

Обеспечение безопасности сетевого приложения - это непрерывный процесс,

который включает мониторинг угроз, реагирование на инциденты безопасности, обновление и улучшение мер безопасности по мере развития новых угроз.

4.5 Charakterystyka kryptografii symetrycznej oraz asymetrycznej.

Криптография с симметричным ключом, также известная как секретная ключевая криптография, использует один и тот же ключ для шифрования и дешифрования данных. Это значит, что обе стороны (отправитель и получатель) должны знать ключ, чтобы безопасно обмениваться информацией. Примерами симметричного шифрования являются DES, 3DES, AES и Blowfish. Преимуществом является скорость работы таких алгоритмов. Однако есть проблема с безопасной передачей ключа между сторонами.

```
1    Cipher cipher = Cipher.getInstance("AES");
2    SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
3    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
4    byte[] encryptedText = cipher.doFinal(plainText.getBytes());
5
```

Криптография с асимметричным ключом, также известная как криптография с открытым ключом, использует пару ключей: открытый ключ для шифрования и закрытый ключ для дешифрования. Это значит, что ключ, используемый для шифрования данных, отличается от ключа, используемого для их дешифрования. Примеры асимметричного шифрования включают RSA, DSA и ECC. Преимущество состоит в том, что открытый ключ можно свободно распространять, не беспокоясь о его перехвате. Однако такие алгоритмы работают медленнее по сравнению с симметричными.

```
1    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
2    keyPairGenerator.initialize(2048);
3    KeyPair keyPair = keyPairGenerator.generateKeyPair();
4
5    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
6    cipher.init(Cipher.ENCRYPT_MODE, keyPair.getPublic());
7
8    byte[] encryptedText = cipher.doFinal(plainText.getBytes());
```

```
9
```

Важно отметить, что на практике часто используется комбинация обоих подходов. Например, в протоколе SSL/TLS сначала используется асимметричное шифрование для обмена секретным ключом, который затем используется для симметричного шифрования сеанса связи.

Rozdział 5

IO MMMM OAOAO

5.1 Standardowe metodyki procesu wytwórczego oprogramowania.

Стандартные методологии разработки программного обеспечения предлагают структурированный подход к планированию, разработке, тестированию и управлению процессом разработки ПО. Вот некоторые из наиболее известных методологий:

- Waterfall: Это линейная и последовательная модель, где каждая стадия начинается только после завершения предыдущей. Этапы включают сбор требований, проектирование, реализацию, тестирование, развертывание и поддержку.
- Agile: Agile представляет собой гибкую методологию, которая обеспечивает быструю разработку и поставку функциональности через инкрементные "спринты". Она подчеркивает коммуникацию, обратную связь и адаптивность к изменениям.
- Scrum: Scrum - это форма Agile, которая включает в себя короткие фазы разработки, называемые "спринтами". Каждый спринт продолжается от одной до четырех недель и включает планирование, разработку, ревью и ретроспективу.

- Lean: Lean разработка основана на принципах бережливого производства и фокусируется на устранении отходов, повышении эффективности и поставке максимальной ценности для клиента.

- DevOps: DevOps является практикой, которая сосредоточена на совместной работе разработчиков и операционных команд с целью улучшить коммуникацию, сотрудничество и интеграцию, а также ускорить поставку ПО.

Важно понимать, что нет одной "лучшей"методологии для каждого проекта или организации. Подход, который следует выбирать, зависит от многих факторов, включая природу проекта, размер и навыки команды, предпочтения клиента, и т.д.

5.2 Metodyki zwinne – SCRUM.

Scrum является подходом к управлению проектами, который подчеркивает гибкость, быструю итерацию и коллективную ответственность. Он был первоначально разработан для проектов по разработке программного обеспечения, но впоследствии был адаптирован для различных видов командных проектов. Вот основные компоненты методологии Scrum:

- Роли:
  - Product Owner (Владелец продукта): Ответственный за определение и приоритизацию элементов бэклога продукта.
  - Scrum Master: Обеспечивает следование принципам и практикам Scrum. Служит в качестве моста между командой и внешними стейкхолдерами.
  - Scrum Team (Команда): Группа, которая разрабатывает продукт и вместе принимает решения.
- Артефакты:
  - Product Backlog (Бэклог продукта): Список требований к продукту, упорядоченных по приоритетам.

  - Sprint Backlog (Бэклог спринта): Подмножество бэклога продукта, выбранное для реализации в течение следующего спринта.
  - Increment (Инкремент продукта): Совокупность всех элементов бэклога продукта, реализованных в течение спринта.

- События:
  - Sprint Planning (Планирование спринта): Встреча, на которой команда определяет, что будет разработано в течение следующего спринта.
  - Daily Scrum (Ежедневный стоячий митинг): Короткая встреча для обсуждения прогресса и планирования работы на день.
  - Sprint Review (Ревью спринта): Встреча, на которой команда показывает то, что было разработано в течение спринта.
  - Sprint Retrospective (Ретроспектива спринта): Встреча после ревью спринта, где команда обсуждает, что хорошо прошло, что плохо, и как можно улучшить следующий спринт.

Основная идея Scrum - создать небольшие, самоорганизующиеся команды, которые могут быстро реагировать на изменения и динамически планировать и проводить работу.

5.3 Testowanie oprogramowania.

Тестирование программного обеспечения — это процесс оценки функциональности программного продукта для обнаружения различий между существующими и требуемыми условиями (то есть дефектами/ошибками/багами) и для понимания рисков использования программного продукта.

Вот некоторые ключевые аспекты тестирования программного обеспечения:

- Функциональное тестирование: Это вид тестирования, в котором ПО проверяется на соответствие его функциональным требованиям.

- Тестирование производительности: Включает в себя тестирование на нагрузку, стресс-тестирование и тестирование стабильности, чтобы гарантировать, что ПО сможет работать под высокой нагрузкой и в течение продолжительного времени.
- Регрессионное тестирование: Это вид тестирования, проводимого для убеждения, что внесенные изменения в код не влияют на уже существующую функциональность.
- Модульное тестирование: Здесь отдельные модули программного обеспечения проверяются независимо друг от друга.
- Интеграционное тестирование: Тестирует, как различные модули работают вместе, обнаруживая проблемы во взаимодействии.
- Системное тестирование: Проверяет полную систему для проверки того, что весь продукт работает правильно и в соответствии со спецификациями.
- Приемочное тестирование: Конечный этап тестирования, который определяет, будет ли система принята или нет. Это может включать в себя тестирование пользователя (UAT), когда реальные пользователи проверяют систему на соответствие их требованиям.

Тестирование ПО — это критически важная часть процесса разработки ПО, которая требует планирования, проектирования, построения тестов и анализа результатов.

5.4 Diagramy UML.

Unified Modeling Language (UML) является стандартным языком для определения и визуализации модели системы программного обеспечения. Он представляет собой набор инструментов для записи и анализа систем и используется для общения об архитектуре и дизайне программных систем. UML включает в себя несколько видов диаграмм, каждая из которых отображает различные аспекты системы:

- Диаграмма классов: Используется для визуализации статической структуры системы, показывая классы, их атрибуты и методы, а также отношения между ними.
- Диаграмма последовательности: Используется для иллюстрации, как объекты взаимодействуют во времени. Она показывает, как сообщения передаются между объектами в течение определенного сценария.
- Диаграмма состояний: Используется для описания поведения системы, отображая состояния объекта и переходы между ними.
- Диаграмма случаев использования: Используется для представления функциональности системы с точки зрения внешних акторов и их взаимодействия со системой.
- Диаграмма активности: Показывает поток управления от одной деятельности к другой в системе. Они очень похожи на диаграммы потоков данных и могут быть использованы для моделирования бизнес-процессов или рабочего процесса системы.

С помощью этих диаграмм UML может быть использован для визуализации, спецификации, конструирования и документирования артефактов системы программного обеспечения.

Rozdział 6

Math

6.1 Wektory i macierze – definicje i podstawowe operacje.

6.1.1 Векторы

Вектор – это элемент векторного пространства, что является обобщением понятия “направленного отрезка”, известного из геометрии. Векторы обычно используются для представления величин, имеющих и размер, и направление.

$$\vec{v} = (v_1, v_2, ..., v_n)$$

6.1.2 Матрицы

Матрица – это прямоугольная таблица чисел, символов или выражений, упорядоченная по строкам и столбцам. Числа, символы или выражения называются элементами матрицы.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

6.1.3 Основные операции

С векторами и матрицами можно выполнять различные операции.

Сложение и вычитание векторов

$$\vec{v} + \vec{u} = (v_1 + u_1, v_2 + u_2, ..., v_n + u_n)$$

$$\vec{v} - \vec{u} = (v_1 - u_1, v_2 - u_2, ..., v_n - u_n)$$

Умножение вектора на скаляр

$$\alpha\vec{v} = (\alpha v_1, \alpha v_2, ..., \alpha v_n)$$

Скалярное произведение векторов

$$\vec{v} \cdot \vec{u} = v_1 u_1 + v_2 u_2 + \dots + v_n u_n$$

Сложение и вычитание матриц

$$A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \dots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \dots & a_{mn} + b_{mn} \end{bmatrix}$$

$$A - B = \begin{bmatrix} a_{11} - b_{11} & a_{12} - b_{12} & \dots & a_{1n} - b_{1n} \\ a_{21} - b_{21} & a_{22} - b_{22} & \dots & a_{2n} - b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} - b_{m1} & a_{m2} - b_{m2} & \dots & a_{mn} - b_{mn} \end{bmatrix}$$

Умножение матрицы на скаляр

$$\alpha A = \begin{bmatrix} \alpha a_{11} & \alpha a_{12} & \dots & \alpha a_{1n} \\ \alpha a_{21} & \alpha a_{22} & \dots & \alpha a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha a_{m1} & \alpha a_{m2} & \dots & \alpha a_{mn} \end{bmatrix}$$

Умножение матриц

Пусть A – матрица размера $m \times n$ и B – матрица размера $n \times p$, тогда произведение AB будет матрицей размера $m \times p$.

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

6.2 Definicja funkcji obliczalnej (częściowo rekurencyjnej).

Функция называется частично рекурсивной (или вычислимой), если существует алгоритм, который сможет вычислить её значение для любого допустимого аргумента за конечное количество времени.

Определение функции как частично рекурсивной связано с понятием машин Тьюринга, где машина Тьюринга, обладающая этим свойством, способна вычислить функцию для любого данного ввода, либо продолжать бесконечно, если функция не определена для этого ввода.

Функция называется тотально рекурсивной или полностью вычислимой, если она частично рекурсивная и определена для каждого ввода. То есть, машина

Тьюринга, обрабатывающая такую функцию, гарантированно остановится для любого ввода.

Это является основой для определения понятия разрешимости и полуразрешимости проблем в теории вычислимости.

6.3 Maszyna Turinga jako model procesów obliczalnych.

Машина Тьюринга – это теоретическая модель вычислений, которая была предложена английским математиком Аланом Тьюрингом в 1936 году. Машина Тьюринга представляет собой простую, но мощную модель, способную вычислять любые задачи, которые можно описать алгоритмически.

Машина Тьюринга состоит из бесконечной ленты, разделенной на ячейки, и головки чтения/записи, которая может двигаться влево или вправо по ленте. Каждая ячейка может содержать один символ из конечного алфавита. Машина работает по определенным правилам, которые определяют, что она должна делать в зависимости от текущего состояния и символа, который она читает на ленте.

Говорят, что функция вычислима на машине Тьюринга, если существует программа для этой машины, которая может вычислить функцию для любого допустимого ввода и остановится с правильным результатом.

Подобные простые машины могут казаться очень далекими от современных компьютеров, но каждый компьютер, с которым мы сталкиваемся в повседневной жизни, от мобильных телефонов до мощных суперкомпьютеров, может быть смоделирован машиной Тьюринга. Это приводит нас к утверждению, которое называется тезисом Чёрча-Тьюринга, оно гласит, что любая функция, которая может быть вычислена физической машиной, может быть вычислена машиной Тьюринга.

6.4 Zagadnienia nierostrzygalne w kontekście obliczalności.

В области теории вычислимости, неразрешимые задачи – это такие задачи, которые невозможно решить с помощью какого-либо алгоритма. Другими словами, не существует машины Тьюринга, которая могла бы решить эти задачи для всех возможных входных данных.

Самый известный пример неразрешимой задачи – это проблема остановки (halting problem). Эта задача состоит в определении, остановится ли определенная машина Тьюринга при работе с заданным вводом или будет работать бесконечно. Алан Тьюринг доказал, что эта проблема неразрешима в общем случае.

Неразрешимые проблемы не ограничиваются только проблемой остановки. Существует множество других неразрешимых проблем, многие из которых возникают в области теории чисел и формальных систем. Например, десятая проблема Гильберта, которая спрашивает о наличии общего алгоритма для определения, существуют ли целочисленные решения уравнения Диофанта. Эта проблема была признана неразрешимой Юрием Матиясевичем в 1970 году.

Rozdział 7

OOP

7.1 Obiekt i klasa w wybranym języku programowania zorientowanym obiektowo.

Класс

Класс в Java — это шаблон или чертёж, используемый для описания объектов. Класс определяет свойства (атрибуты) и методы, которые будут общими для всех объектов данного класса. Вот пример простого класса в Java:

```
public class Car {
    // class fields
    private String color;
    private int speed;

    // Class methods
    public void setColor(String color) {
        this.color = color;
    }

    public void setSpeed(int speed) {
        this.speed = speed;
    }

    public String getColor() {
        return this.color;
    }
```

```
18
19      public int getSpeed() {
20          return this.speed;
21      }
22  }
```

Объект

Объект в Java — это экземпляр класса. Создание объекта класса называется инстанциацией. Каждый объект имеет свои собственные значения для свойств, определённых в классе. Вот как вы можете создать объект класса ‘Car‘ в Java:

```
1  public class Main {
2      public static void main(String[] args) {
3          // creation obj Car
4          Car myCar = new Car();
5
6          // using methods of Car class to set and get values
7          myCar.setColor("Red");
8          myCar.setSpeed(70);
9          System.out.println("Car color: " + myCar.getColor());
10         System.out.println("Car speed: " + myCar.getSpeed());
11     }
12 }
```

7.2 Hermetyzacja, dziedziczenie i polimorfizm w programowaniu obiektowym.

Герметизация

Герметизация, или инкапсуляция, — это концепция ООП, которая скрывает детали реализации от пользователя. Если метод или переменная объявлены с модификатором private, они не могут быть доступны извне класса.

```
1  public class Example {
2      private String hiddenVar;
3
4      public String getHiddenVar() {
5          return this.hiddenVar;
```

```
    }

    public void setHiddenVar(String value) {
        this.hiddenVar = value;
    }
}
```

Наследование

Наследование — это свойство, которое позволяет одному классу наследовать поля и методы другого. С помощью наследования можно создавать более общие классы, а затем расширять их для создания более специализированных классов.

```
public class Vehicle {
    public void move() {
        System.out.println("Vehicle is moving");
    }
}

public class Car extends Vehicle {
    @Override
    public void move() {
        System.out.println("Car is moving");
    }
}
```

Полиморфизм

Полиморфизм позволяет использовать объекты различных типов с общим интерфейсом. В Java это достигается благодаря использованию интерфейсов и наследования.

```
public interface Animal {
    public void makeSound();
}

public class Cat implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }
}
```

```
11
12 public class Dog implements Animal {
13     @Override
14     public void makeSound() {
15         System.out.println("Woof");
16     }
17 }
```

7.3 Interfejsy i klasy abstrakcyjne w programowaniu obiektowym.

Интерфейсы

Интерфейс в объектно-ориентированном программировании — это контракт, который определяет, какие методы должны быть реализованы в классе. Интерфейсы не содержат деталей реализации методов. В Java это может выглядеть так:

```
1 public interface Animal {
2     void makeSound();
3 }
```

Абстрактные классы

Абстрактный класс — это класс, который не может быть инстанциирован напрямую, и который обычно содержит один или несколько абстрактных методов (методов без реализации). Он служит базой для подклассов, которые должны реализовать все абстрактные методы. Пример абстрактного класса в Java:

```
1 public abstract class Animal {
2     public abstract void makeSound();
3
4     public void eat() {
5         System.out.println("The animal eats");
6     }
7 }
```

В этом примере, ‘makeSound‘ — это абстрактный метод, который должен быть реализован в каждом конкретном подклассе класса ‘Animal‘. В то же

время, ‘eat‘ — это обычный метод с реализацией, и его можно переопределить в подклассе, если требуется.

Rozdział 8

Paradygmaty programowania

8.1 Główne paradygmaty programowania – charakterystyka i przykłady.

Парадигмы программирования — это основные подходы к написанию программ. Каждая парадигма представляет собой уникальную комбинацию принципов, стилей и методологий, которые определяют подход к организации и структурированию кода. Вот некоторые из основных парадигм программирования:

- Процедурное программирование: Ориентировано на выполнение последовательности процедур или подпрограмм для обработки данных. Примеры языков: C, Fortran, COBOL.
- Объектно-ориентированное программирование (ООП): Ориентировано на использование "объектов", которые объединяют данные и функции, работающие с этими данными. ООП также поддерживает наследование, полиморфизм и инкапсуляцию. Примеры языков: Java, C++, C шарп, Python.
- Функциональное программирование: Фокусируется на использовании чистых функций, которые не имеют побочных эффектов. Функциональные программы строятся путем композиции функций. Примеры языков: Haskell, Lisp, Clojure.

- Декларативное программирование: Ориентировано на описание желаемого результата, а не последовательности шагов для его достижения. SQL (язык запросов к базам данных) является примером декларативного языка.
- Логическое программирование: Программы формулируются как наборы логических утверждений, а выполнение программы — это процесс доказательства или отрицания этих утверждений. Примером может служить Prolog.

Большинство современных языков программирования поддерживают несколько парадигм, что позволяет разработчикам выбирать подход, который наиболее подходит для решаемой задачи.

8.2 Gramatyki bezkontekstowe – definicje, charakterystyki i przykłady.

Безконтекстная грамматика – это форма грамматики, используемая в области формальных языков. Она состоит из набора продукционных правил, каждое из которых преобразует один символ (называемый нетерминалом) в последовательность терминалов и/или нетерминалов. Грамматика называется "безконтекстной", потому что левая часть каждого продукционного правила состоит из одного нетерминала, и, следовательно, преобразование можно выполнить вне зависимости от контекста, в котором этот нетерминал появляется.

Простым примером безконтекстной грамматики может быть грамматика, генерирующая все правильно сформированные скобочные выражения. Она может быть определена следующими продукционными правилами:

Безконтекстные грамматики играют важную роль в области компиляции и анализа программ, где они используются для определения синтаксиса языков программирования. Безконтекстные грамматики обеспечивают достаточную мощность для описания синтаксиса большинства языков программирования, в то же время они достаточно просты для анализа с помощью автоматизированных инструментов.

Более сложные типы грамматик, такие как контекстно-зависимые и контекстно-свободные грамматики, могут описывать более сложные структуры, но они также сложнее для анализа и использования.

8.3 Analiza leksykalna, syntaktyczna i semantyczna kodu.

Анализ кода — это процесс преобразования исходного кода в более абстрактное представление. Этот процесс обычно включает три основных этапа: лексический анализ, синтаксический анализ и семантический анализ.

- Лексический анализ: Это первый этап анализа, на котором исходный код разбивается на отдельные лексемы — наименьшие смысловые единицы кода. Лексемы могут включать ключевые слова, идентификаторы, литералы, операторы и другие компоненты синтаксиса языка программирования. Результатом лексического анализа является поток токенов, который используется для дальнейшего анализа.

- Синтаксический анализ: Этот этап анализирует поток токенов, созданный на этапе лексического анализа, и создает структурированное представление кода, называемое деревом разбора. Синтаксический анализ проверяет код на соответствие синтаксическим правилам языка программирования.

- Семантический анализ: На этом этапе проводится проверка кода на соответствие семантическим правилам языка. Это может включать проверку типов, проверку использования переменных и функций, проверку потока управления и т.д. Семантический анализ может обнаружить ошибки, которые не были обнаружены на этапе синтаксического анализа.

Эти этапы анализа обычно выполняются компиляторами и интерпретаторами языков программирования в процессе трансляции исходного кода в машинный код или в процессе его интерпретации.

8.4 Rodzaje błędów w kontekście analizy leksykalnej, syntaktycznej i semantycznej kodu.

В контексте анализа кода можно выделить три основных типа ошибок, соответствующих этапам анализа: лексические, синтаксические и семантические ошибки.

- Лексические ошибки: Эти ошибки возникают, когда программа содержит символы или последовательности символов, которые не могут быть правильно преобразованы в токены. Примеры лексических ошибок включают нераспознанные символы и неправильно сформированные числовые или строковые литералы. Например, в языке программирования Python следующий код вызовет лексическую ошибку из-за неправильного использования символа ':

  print('Hello, world!)

- Синтаксические ошибки: Эти ошибки возникают, когда программа нарушает грамматические правила языка. Это может произойти, например, когда пропущена закрывающая скобка или ключевое слово. В Python следующий код вызовет синтаксическую ошибку из-за пропущенного двоеточия:

  if x > 0
  print("x is positive")

- Семантические ошибки: Эти ошибки возникают, когда программа нарушает правила языка, связанные с поведением программ, такие как правила использования типов или переменных. Например, в Python следующий код вызовет семантическую ошибку из-за использования неинициализированной переменной:

  print(x)

Каждый из этих типов ошибок обычно обнаруживается на соответствующем этапе анализа кода, и их исправление обычно требует изменения кода программы.

8.5 Deklaratywne programowanie funkcyjne: rachunek lambda, monady

Декларативное программирование — это парадигма программирования, которая фокусируется на описании, что программа должна выполнять, вместо того, чтобы описывать, как она это должна делать. Функциональное программирование — это подкатегория декларативного программирования, которая основана на использовании функций и их композиции.

Расчет lambda — это система формальной логики, которая изучает функции и их применение. В контексте функционального программирования, расчет lambda предоставляет основу для понимания функций как объектов первого класса, которые могут быть переданы в другие функции в качестве аргументов или возвращены из других функций в качестве результатов.

Вот простой пример лямбда-выражения в языке программирования Python:

add = lambda x, y: x + y

print(add(5, 3)) выведет 8

Монады — это концепт из категориальной логики, который используется в некоторых функциональных языках программирования для обработки побочных эффектов в чистом функциональном стиле. В языке Haskell, например, монады используются для обработки ввода-вывода, работы с изменяемым состоянием и других операций, которые вне контекста функционального программирования обычно вызывают побочные эффекты.

8.6 Deklaratywne programowanie w logice: klauzule Horne'a, nawracanie.

Декларативное программирование в логике представляет собой подход к программированию, основанный на использовании формальной логики. Программы, написанные в этой парадигме, обычно представляют собой набор логических утверждений, и исполнение программы представляет собой процесс вывода на основе этих утверждений. Один из наиболее известных языков программирования, основанных на логике, — это Prolog.

Клаузулы Хорна — это специальный вид логических формул, который широко используется в программировании на логике. Клаузула Хорна представляет собой дизъюнкцию литералов с не более чем одним недоказанным (положительным) литералом. Программы на Prolog обычно представляют собой набор клаузул Хорна.

Навигация (backtracking) — это метод решения задачи, который включает перебор всех возможных решений и возврат назад, когда определенное решение оказывается неприемлемым. В контексте программирования на логике, возврат используется для исследования различных путей вывода, когда пытаются удовлетворить набор логических утверждений. Если определенный путь вывода не приводит к успешному решению, процесс может "вернуться назад"и попробовать другой путь.

Rozdział 9

PAS

9.1 Mechanizm sesji w zarządzaniu stanem aplikacji sieciowej.

Веб-приложения в основном являются "без сохранения состояния"(stateless), что означает, что каждый запрос обрабатывается независимо от других. Это может создать проблемы, когда необходимо сохранить информацию о состоянии пользователя между несколькими запросами. Для решения этой проблемы веб-приложения часто используют механизм сессий.

Сессия - это способ сохранения информации (например, идентификатор пользователя или данные корзины покупок) на сервере между несколькими запросами. Когда пользователь делает первый запрос, сервер создает уникальный идентификатор сессии и отправляет его обратно в браузер в виде cookie. Затем при каждом последующем запросе браузер отправляет этот идентификатор сессии обратно на сервер, позволяя серверу "восстановить"состояние сессии.

Сессии важны для поддержания "состояния"между множеством запросов и являются основой для функциональности, такой как аутентификация пользователя, сохранение содержимого корзины покупок, отслеживание предпочтений пользователя и так далее.

9.2 Mechanizm gniazd – pojęcie, sposób realizacji i zastosowanie

Сокет - это абстракция, используемая для отправки и получения данных между процессами. Процессы могут находиться на одном компьютере или на разных компьютерах, соединенных через сеть.

Способ реализации

Сокеты обычно реализуются в операционной системе как системные вызовы. В Unix-подобных операционных системах, например, предоставляются системные вызовы, такие как socket (для создания нового сокета), connect (для установления соединения с другим сокетом), и send и receive (для отправки и получения данных).

Застосування

Сокеты используются для обмена данными между процессами в сети, и они являются основой для многих сетевых протоколов и технологий, таких как HTTP, FTP, SSH и так далее.

Например, когда вы открываете веб-страницу в браузере, браузер создает сокет и устанавливает соединение с веб-сервером, а затем использует этот сокет для отправки HTTP-запроса на сервер и получения HTTP-ответа от сервера.

9.3 Metody obsługi wielu klientów równolegle w aplikacjach sieciowych.

Многопоточность

Многопоточность - это метод, при котором каждому клиенту назначается отдельный поток в рамках одного процесса. Это позволяет приложению обслу-

живать множество клиентов одновременно, каждый в своем потоке. Этот метод может быть эффективным, но требует аккуратной работы с потоками, включая синхронизацию и управление состоянием.

Многопроцессность

Многопроцессность подразумевает использование отдельного процесса для каждого клиента. Это дает каждому клиенту свой собственный контекст выполнения и пространство памяти, что может упростить управление состоянием, но также может быть более ресурсоемким, чем многопоточность.

Асинхронная обработка

Асинхронная обработка предполагает использование одного или нескольких потоков или процессов, которые могут обрабатывать множество клиентов одновременно, используя асинхронные операции ввода-вывода. Вместо блокировки и ожидания завершения операции ввода-вывода, асинхронное приложение будет продолжать обслуживание других клиентов. Это может быть очень эффективным, особенно для приложений с высокой загрузкой, но также требует от разработчика более сложной модели программирования.

9.4 Pocztowe protokoły warstwy aplikacji.

SMTP (Simple Mail Transfer Protocol)

SMTP - это протокол, используемый для отправки электронной почты от отправителя к получателю и между серверами почты. Он работает над протоколом TCP/IP и использует порт 25. SMTP поддерживает только отправку текстовых сообщений.

POP3 (Post Office Protocol version 3)

POP3 - это протокол, используемый клиентами электронной почты для получения сообщений с почтового сервера. Сообщения скачиваются на локальное устройство и обычно удаляются с сервера. Он работает над протоколом TCP/IP и использует порт 110.

IMAP (Internet Message Access Protocol)

IMAP - это альтернатива POP3, которая предоставляет более сложные функции. В отличие от POP3, IMAP позволяет клиентам электронной почты сохранять сообщения на сервере, организовывать их в папки и синхронизировать структуру папок между несколькими устройствами. Он работает над протоколом TCP/IP и использует порт 143.

9.5 Porównanie HTTP i WebSocket.

HTTP

HTTP (HyperText Transfer Protocol) - это протокол, широко используемый для обмена данными в веб-приложениях. HTTP основан на модели "запрос-ответ", где клиент отправляет запрос на сервер, а сервер затем возвращает ответ.

HTTP не сохраняет состояние между различными запросами, что делает его "без сохранения состояния"(stateless). Кроме того, в HTTP только клиент может инициировать обмен данными, отправляя запрос на сервер.

WebSocket

WebSocket - это протокол, позволяющий двустороннее взаимодействие между клиентом и сервером в реальном времени. В отличие от HTTP, WebSocket позволяет серверу инициировать передачу данных, а также поддерживает постоянное

соединение, что обеспечивает более быструю обработку сообщений, поскольку не требуется переустановка соединения для каждого обмена данными.

WebSocket идеально подходит для приложений, которым требуется быстрое и постоянное взаимодействие между клиентом и сервером, таких как игры в реальном времени, чаты и применения в реальном времени.

Rozdział 10

Podstawy informatyki

10.1 Problemy rekurencyjne i ich rozwiązywanie.

Рекурсия в программировании это техника, при которой функция вызывает саму себя один или несколько раз. Важным моментом при использовании рекурсии является определение "базового случая", который не вызывает сам себя, чтобы предотвратить бесконечную рекурсию.

Рекурсия часто используется для решения задач, которые могут быть разбиты на более мелкие/простые подзадачи того же типа.

Например, факториал числа - это классическая рекурсивная задача. Факториал числа n (обозначается n!) — это произведение всех натуральных чисел от 1 до n включительно. Факториал можно определить рекурсивно, так как факториал n это произведение n на факториал n-1, а факториал 1 равен 1.

```
1 int factorial(int n) {
2 if (n == 1) {
3 return 1;
4 } else {
5 return n * factorial(n - 1);
6 }
7 }
```

Важно отметить, что хотя рекурсия может быть мощным инструментом,

она может также привести к проблемам с производительностью и переполнением стека, если рекурсивные вызовы становятся слишком глубокими. В некоторых случаях эти проблемы можно избежать с помощью техник, таких как мемоизация или преобразование рекурсии в итерацию.

10.2 Pozycyjne systemy liczbowe i konwersje pomiędzy nimi.

Позиционные системы счисления используются для представления чисел, причем важную роль играет позиция каждой цифры. Вот несколько распространенных систем счисления:

- Десятичная система: Использует 10 различных цифр (0-9). Это наиболее распространенная система счисления в повседневной жизни.
- Бинарная система: Использует только две цифры: 0 и 1. Бинарная система широко используется в компьютерах, так как информация в них хранится в двоичной форме.
- Восьмеричная система: Использует восемь цифр (0-7). Восьмеричные числа иногда используются в программировании.
- Шестнадцатеричная система: Использует 16 цифр. Для представления чисел от 10 до 15 используются буквы A-F. Шестнадцатеричная система часто используется в программировании и информатике.

Конвертация между различными системами счисления может быть выполнена с использованием различных методов. Например, для конвертации числа из бинарной системы в десятичную, можно взять каждую цифру, умножить ее на два в степени, равной ее позиции (начиная с нуля справа), и сложить все эти значения. Обратный процесс можно использовать для конвертации из десятичной системы в бинарную.

Во многих языках программирования есть встроенные функции для выполнения таких преобразований. Например, в Java можно использовать Integer.parseInt()

для преобразования бинарной строки в десятичное число, и Integer.toBinaryString() для преобразования десятичного числа в бинарную строку.

10.3 Typ, zmienna, obiekt i zarządzanie pamięcią.

Тип: В программировании тип данных определяет множество значений, которые может принимать переменная, а также набор операций, которые могут быть выполнены над этими значениями. Примеры типов данных включают целые числа (int), числа с плавающей запятой (float), булевы значения (boolean) и строки (String).

Переменная: Переменная это элемент программы, который может хранить значение определенного типа. Имя переменной используется для обращения к этому значению в программе.

Объект: В объектно-ориентированном программировании объект представляет собой экземпляр класса. Объекты могут содержать поля (переменные, связанные с объектом) и методы (функции, которые могут быть выполнены объектом).

Управление памятью: Управление памятью это аспект программирования, который относится к контролю над использованием памяти в компьютерной программе. В некоторых языках программирования, таких как C и C++, программистам приходится вручную выделять и освобождать память. Другие языки, такие как Java и Python, автоматически управляют памятью с помощью механизма под названием "сборка мусора", который автоматически освобождает память, которая больше не используется программой.

10.4 Instrukcje sterujące przepływem programu.

Управляющие инструкции в программировании используются для управления порядком, в котором выполняются инструкции. Они позволяют программам принимать решения и повторять действия. Вот некоторые основные типы управляющих инструкций:

- Условные инструкции (if, switch): Эти инструкции позволяют программе выбирать между различными путями выполнения на основе результатов проверки некоторых условий. Например, в Java оператор if может быть использован так:

```
if (condition) {

} else {

}
```

- Циклы (for, while, do-while): Циклы позволяют повторять блок кода несколько раз. Вот пример цикла for в Java:

```
for (int i = 0; i < 10; i++) {
// loop 10
}
```

- Break и continue: Эти инструкции используются для изменения нормального порядка выполнения циклов. Break прерывает цикл, а continue пропускает оставшуюся часть текущей итерации и переходит к следующей итерации.

- Return: Используется для возвращения значения из функции и немедленного прекращения выполнения этой функции.

Эти управляющие инструкции обеспечивают гибкость и мощь, которые делают программирование полезным для решения широкого спектра задач.

10.5 Kodowanie liczb ze znakiem w systemie U2, generowanie liczby ze znakiem przeciwnym, dodawanie i odejmowanie.

Дополнительный код (U2) используется для представления отрицательных чисел в двоичной системе. В этом коде отрицательные числа кодируются как дополнение до двух их абсолютных значений. Для получения дополнительного кода к двум числа, можно инвертировать все биты (менять 0 на 1 и наоборот),

а затем прибавить 1 к получившемуся числу.

Для примера, преобразуем -5 в двоичной системе в 8-битной форме:

- Запишем 5 в двоичной форме (7-битный код): 0000101
- Инвертируем биты: 1111010
- Прибавим 1: 1111011

Получившийся результат 1111011 — это представление числа -5 в 8-битной форме в системе U2.

Для выполнения операций сложения и вычитания, можно сложить числа как обычно, не обращая внимания на знаки. Если в результате сложения возникает перенос из старшего разряда, его следует отбросить. Если вычитаемое число больше уменьшаемого, результат должен взяться с обратным знаком.

Основной преимуществом кода U2 является его простота использования при выполнении арифметических операций, поскольку используются стандартные процедуры сложения и вычитания, а также удобство обработки знака.

Rozdział 11

Sieci

11.1 Protokoły TCP i UDP – porównanie i zastosowanie.

TCP (Transmission Control Protocol) и UDP (User Datagram Protocol) – это два основных протокола транспортного уровня, используемых в интернете. Они обеспечивают различные методы доставки данных от одного устройства к другому.

11.1.1 TCP

TCP является надежным протоколом, который гарантирует доставку пакетов данных. Он предоставляет функции, такие как контроль потока, контроль перегрузки и повторная передача потерянных пакетов. Это делает TCP идеальным для приложений, которым требуется гарантированная доставка данных, таких как веб-браузеры, электронная почта и файловые передачи.

11.1.2 UDP

В отличие от TCP, UDP является ненадежным протоколом, который не гарантирует доставку пакетов данных. Он просто отправляет пакеты без установления соединения или проверки, что пакеты были доставлены. Это делает UDP более быстрым и эффективным для приложений, которым не требуется гарантированная

доставка, таких как видео и аудио потоки, игры и некоторые виды VoIP (Voice over IP).

11.1.3 Сравнение

Вот несколько ключевых различий между TCP и UDP:

- Надежность: TCP обеспечивает надежность за счет повторной передачи потерянных пакетов и проверки ошибок, в то время как UDP не предоставляет такой гарантии.
- Скорость: UDP обычно быстрее, чем TCP, поскольку он не заботится о повторной передаче потерянных пакетов.
- Соединения: TCP использует принцип соединений, то есть перед передачей данных устанавливается соединение. UDP не требует установления соединения перед отправкой данных.
- Порядок пакетов: TCP гарантирует, что пакеты будут доставлены в том порядке, в котором они были отправлены. UDP не предоставляет такую гарантию.

В целом, выбор между TCP и UDP зависит от приложения и требуемых свойств передачи данных.

11.2 Adresowanie w warstwie Internetu modelu TCP/IP.

В модели TCP/IP адресация в интернет-слое относится к способу, которым устройства идентифицируются и локализуются в сети. Это в основном происходит через протоколы IP-адресации, а именно IPv4 и IPv6.

11.2.1 IPv4

IPv4 (Internet Protocol version 4) является наиболее широко использованным протоколом IP. IPv4-адрес состоит из 32 битов и обычно записывается в виде четырех десятичных чисел, разделенных точками, каждое из которых представляет восьмибитное число от 0 до 255. Например, 192.168.0.1.

11.2.2 IPv6

IPv6 (Internet Protocol version 6) был разработан как решение проблемы исчерпания адресов IPv4. IPv6-адрес состоит из 128 битов и записывается как восемь групп из четырех шестнадцатеричных чисел, разделенных двоеточиями. Например, 2001:0db8:85a3:0000:0000:8a2e:0370:7334.

11.2.3 Сети и подсети

IP-адреса обычно включают идентификатор сети и идентификатор устройства внутри этой сети (хоста). Это позволяет сетевым устройствам маршрутизировать пакеты на правильную сеть и к правильному устройству внутри этой сети. Маска подсети используется для разделения адреса на сетевую и хостовую части.

Важно отметить, что хотя IPv4 и IPv6 используют разные форматы адресов, оба они работают по тому же основному принципу, и оба могут быть использованы в интернет-слое модели TCP/IP.

11.3 Porównanie zadań przełącznika (switcha) i routera.

Переключатели (switches) и маршрутизаторы (routers) являются ключевыми компонентами сетевой инфраструктуры, но они выполняют разные функции и работают на разных уровнях модели OSI.

11.3.1 Переключатель (Switch)

Переключатель – это устройство, работающее на уровне данных (2-й уровень) модели OSI. Он используется для соединения устройств внутри одной локальной сети (LAN). Переключатели могут "изучить"MAC-адреса устройств, подключенных к ним, и направлять трафик непосредственно от отправителя к получателю в пределах одной сети. Это повышает эффективность сети, уменьшая коллизии и увеличивая пропускную способность.

11.3.2 Маршрутизатор (Router)

Маршрутизатор – это устройство, работающее на сетевом уровне (3-й уровень) модели OSI. Он соединяет две или более сетей вместе и управляет трафиком между ними. Маршрутизаторы используют IP-адреса для определения лучшего пути для передачи пакетов между сетями. Они обеспечивают связь между локальными сетями и глобальной сетью (например, Интернетом), а также могут обрабатывать такие задачи, как преобразование адресов (NAT), фильтрация трафика и безопасность сети.

11.3.3 Сравнение

В общем, можно сказать, что переключатели используются для управления трафиком внутри одной сети, а маршрутизаторы – для управления трафиком между разными сетями. Оба они играют важную роль в современной сетевой инфраструктуре.

11.4 Porównanie modelu OSI i TCP/IP.

Модель OSI (Open Systems Interconnection) и модель TCP/IP (Transmission Control Protocol/Internet Protocol) – это две концептуальные модели, которые используются для описания, как сетевые протоколы взаимодействуют и коммуницируют друг с другом.

11.4.1 Модель OSI

Модель OSI состоит из семи слоев: Физический, Канальный, Сетевой, Транспортный, Сеансовый, Представления и Прикладной. Эта модель была разработана Международной организацией по стандартизации (ISO) и представляет собой идеализированное представление о том, как должны взаимодействовать сетевые протоколы.

11.4.2 Модель TCP/IP

Модель TCP/IP, также известная как Интернет-модель, состоит из четырех слоев: Сетевого интерфейса, Интернета, Транспорта и Приложения. Эта модель была разработана в рамках проекта DARPA и является основой современного Интернета.

11.4.3 Сравнение

Вот несколько ключевых отличий между моделями OSI и TCP/IP:

- Количество слоев: Модель OSI состоит из семи слоев, в то время как модель TCP/IP имеет только четыре.
- Абстракция vs реальность: Модель OSI часто рассматривается как идеальная, но искусственная модель, в то время как модель TCP/IP базируется на реальных протоколах, используемых в Интернете.
- Универсальность: Модель OSI стремится быть универсальной моделью для всех видов сетевых протоколов, в то время как модель TCP/IP была разработана специально для семейства протоколов Интернета.

Обе модели играют важную роль в понимании и описании сетевых взаимодействий и протоколов.

11.5 Mechanizm enkapsulacji w modelu OSI.

Инкапсуляция является ключевым процессом в модели OSI, который позволяет данным перемещаться от верхних слоев модели к нижним и обратно. На каждом слое информация, которую нужно передать, оборачивается в капсулу с добавлением заголовка (и, возможно, трейлера), содержащего специфическую для этого слоя информацию.

При передаче данных от источника к получателю процесс инкапсуляции происходит следующим образом:

1. Данные начинают свое путешествие на прикладном слое (седьмой слой) модели OSI на стороне отправителя.
2. На каждом следующем слое к данным добавляется заголовок (и иногда трейлер), содержащий информацию, специфичную для этого слоя. Этот процесс продолжается до тех пор, пока данные не достигнут физического слоя (первого слоя) и не будут переданы по сети.
3. По прибытии данных к получателю процесс инкапсуляции происходит в обратном порядке (это называется декапсуляцией). На каждом слое заголовки (и трейлеры), добавленные на стороне отправителя, удаляются, и данные передаются на следующий слой вверх по стеку, пока не достигнут прикладного слоя.

Этот процесс позволяет каждому слою модели OSI фокусироваться на своей конкретной задаче и обрабатывать специфическую для слоя информацию, не заботясь о деталях, специфических для других слоев.

Rozdział 12

Statystyka

12.1 Основные характеристики описательной и математической статистики

Статистика – это отрасль математики, которая занимается сбором, анализом, интерпретацией, представлением и организацией данных. Она делится на две основные ветви: описательную статистику и инференциальную (или математическую) статистику.

12.1.1 Описательная статистика

Описательная статистика, как следует из названия, занимается описанием или суммированием набора данных. Она включает в себя расчет различных статистических показателей, таких как среднее значение, медиана, мода, стандартное отклонение, дисперсия и коэффициенты корреляции. Описательная статистика также может включать в себя создание графических представлений данных, таких как гистограммы, ящики с усами и диаграммы рассеяния.

12.1.2 Математическая статистика

Математическая (или инференциальная) статистика использует математические методы для обобщения данных и делает выводы или предсказания на основе

набора данных. Инференциальная статистика используется для оценки параметров, проверки статистических гипотез, построения доверительных интервалов и прогнозирования будущих результатов на основе выборки данных.

Важно отметить, что описательная статистика и инференциальная статистика не являются взаимоисключающими и часто используются вместе в статистическом анализе.

Rozdział 13

Systemy operacyjne

13.1 Wielowarstwowa organizacja systemów komputerowych.

13.2 Многоуровневая организация компьютерных систем

13.2.1 Преимущества

Преимущества многоуровневой организации включают следующее:

- Модульность: Слои можно разрабатывать и обновлять независимо друг от друга, что упрощает процесс разработки и обслуживания системы.
- Переносимость: При должном проектировании функции и услуги, предоставляемые одним слоем, могут быть заменены без влияния на другие слои.
- Абстракция: Каждый слой может сосредоточиться на выполнении своих специфических задач, не беспокоясь о деталях выполнения задач на других уровнях.

13.2.2 Примеры

Примеры многоуровневой организации включают модель взаимодействия открытых систем (OSI), которая используется в сетевой коммуникации, и архитектуру систем на основе микросервисов, в которой различные функции приложения

разделяются на независимые службы, которые могут быть разработаны, развернуты и масштабированы независимо друг от друга.

13.3 System operacyjny – charakterystyka, zadania, klasyfikacja.

Операционная система (ОС) - это системное программное обеспечение, которое управляет аппаратными ресурсами компьютера и предоставляет услуги для выполнения прикладного программного обеспечения. Она служит в качестве интерфейса между пользователем и аппаратными ресурсами системы.

13.3.1 Характеристики

Операционные системы могут быть характеризованы следующими свойствами:

- Многозадачность: Поддержка выполнения нескольких процессов или потоков в рамках одной системы.
- Многопользовательский режим: Позволяет нескольким пользователям одновременно использовать ресурсы системы.
- Управление памятью: Отслеживает использование памяти и координирует доступ к ней.
- Управление процессами: Управляет выполнением процессов и распределением ресурсов процессора.
- Управление устройствами: Координирует доступ и использование аппаратных ресурсов, таких как диски, принтеры и интерфейсы сети.

13.3.2 Задачи

Основные задачи операционной системы включают:

- Управление ресурсами: Координация и управление доступом к общим ресурсам, таким как процессор, память и устройства ввода-вывода.

- Обеспечение безопасности и доступа: Защита данных и системных ресурсов от несанкционированного доступа и обеспечение контроля доступа.
- Обеспечение удобного интерфейса: Предоставление удобного и эффективного интерфейса для пользователей и прикладных программ.

13.3.3 Классификация

Операционные системы можно классифицировать по различным критериям, включая:

- Тип системы: Настольные (например, Windows, macOS, Linux), мобильные (например, Android, iOS), серверные (например, Windows Server, Unix), встроенные и т.д.
- Архитектура: Монолитная, микроядерная, многоядерная и т.д.

13.4 Procesy i wątki – charakterystyka i problemy

13.4.1 Процессы

Процесс - это экземпляр программы во время выполнения, включающий в себя текущее состояние программы и всю информацию, необходимую для ее выполнения, такую как значения регистров процессора, системные ресурсы и область памяти.

13.4.2 Потоки

Поток - это наименьшая единица обработки, которую можно запланировать и управлять с помощью операционной системы. Поток представляет собой подмножество процесса и делится на область памяти и ресурсы с другими потоками в том же процессе. Потоки могут выполняться параллельно, что позволяет более эффективно использовать многоядерные процессоры.

13.4.3 Проблемы

- Взаимная блокировка: Это состояние, при котором два или более процессов или потоков вечно ждут друг друга, чтобы освободить ресурсы. Это может привести к полному останову работы системы.

- Гонка данных: Это состояние, при котором два или более потока неожиданно взаимодействуют при общем доступе к ресурсу, что приводит к некорректным или непредсказуемым результатам.

- Планирование и контекстные переключения: Планирование процессов и потоков и переключение контекста между ними требуют значительных затрат системных ресурсов.

Многие из этих проблем могут быть смягчены с помощью техник синхронизации, таких как мьютексы, семафоры и мониторы.

13.5 Zarządzanie pamięcią operacyjną w systemie operacyjnym.

Управление оперативной памятью - это одна из основных функций операционной системы. Она включает в себя управление физической и виртуальной памятью, выделение и освобождение памяти для процессов, а также защиту и изоляцию адресных пространств.

13.5.1 Управление физической памятью

Операционная система отслеживает доступную физическую память и отвечает за ее выделение процессам, когда они ее запрашивают. Операционная система также отвечает за освобождение памяти, когда процессы завершаются или когда они больше не нуждаются в выделенных ресурсах.

13.5.2 Управление виртуальной памятью

Операционная система может предоставлять процессам виртуальное адресное пространство, которое может быть больше, чем реально доступная физическая память. Это достигается путем использования подкачки и страничного обмена, при которых неиспользуемые страницы памяти могут быть временно перемещены на диск.

13.5.3 Защита памяти

Операционная система также отвечает за защиту памяти, гарантируя, что процессы не могут читать или записывать в адресное пространство других процессов, если только это явно не разрешено. Это обеспечивает безопасность и стабильность системы.

13.5.4 Сегментация и страничное разделение

Операционная система может использовать сегментацию или страничное разделение для управления памятью. Сегментация заключается в разделении адресного пространства процесса на части, или сегменты, разного размера. Страничное разделение разделяет память на блоки фиксированного размера, или страницы.

13.6 Organizacja systemu plików i pamięci zewnętrznej.

13.6.1 Система файлов

Система файлов - это метод организации и хранения информации на внешнем устройстве хранения, таком как жесткий диск, SSD или съемный накопитель. Она определяет, как файлы и каталоги структурированы и как они связаны между собой. Существуют различные типы файловых систем, включая FAT32, NTFS, HFS+, Ext4 и другие, каждая из которых имеет свои особенности и применение.

13.6.2 Внешняя память

Внешняя память относится к устройствам хранения данных, которые не используются для выполнения программ, но используются для долгосрочного хранения данных. Это включает в себя различные типы устройств, такие как жесткие диски, твердотельные накопители (SSD), оптические диски и съемные накопители.

13.6.3 Организация

Операционная система использует систему файлов для организации данных на внешней памяти. Файлы обычно группируются в каталоги (также называемые "папками") для удобства навигации и организации. Файлы и каталоги могут иметь атрибуты, такие как права доступа, дата создания и владелец, которые контролируют, как файлы могут быть прочитаны, записаны и изменены.

13.6.4 Управление памятью

Операционная система также отвечает за управление внешней памятью. Это включает в себя отслеживание доступного пространства, выделение пространства для файлов, освобождение пространства, когда файлы удаляются, и многое другое. Операционная система также может использовать техники, такие как фрагментация и дефрагментация, чтобы оптимизировать использование пространства и производительность чтения/записи.

Rozdział 14

Systemy wbudowane

14.1 Różnice pomiędzy obsługą zdarzeń w przerwaniach sprzętowych a obsługą zdarzeń w pętli programowej.

14.2 Powody i przykłady stosowania mikrokontrolerów zamiast typowych komputerów

14.3 Podstawowe układy systemu mikroprocesorowego i sposób wymiany informacji pomiędzy nimi.

14.4 Dekoder, multiplekser i demultiplekser: budowa, zasada, działania, przeznaczenie, zastosowanie.

14.5 Budowa i zasada działania generatora obrazu w systemie mikroprocesorowym.

Rozdział 15

XZ CZTO ETO

15.1 Modele reprezentacji wiedzy

Модели представления знаний используются для формализации и структурирования информации и знаний таким образом, чтобы они могли быть обработаны компьютерами. Ниже приведены некоторые общие типы моделей представления знаний.

15.1.1 Семантические сети

Семантические сети – это графическая структура, которая используется для представления знаний в форме сети, где узлы представляют концепции, а дуги представляют отношения между концепциями.

15.1.2 Базы данных

Базы данных обеспечивают структурированный и организованный способ хранения, управления и извлечения информации. Модели данных, такие как реляционная, иерархическая и сетевая, используются для организации данных в базах данных.

15.1.3 Базы знаний

Базы знаний - это тип базы данных, специально разработанный для управления знаниями. Они часто используют формы логики и онтологии для представления сложных структур знаний и отношений.

15.1.4 Экспертные системы

Экспертные системы используют базы знаний в сочетании с набором правил, чтобы моделировать решения, которые могут быть приняты экспертом в определенной области.

15.1.5 Онтологии

Онтологии - это формальные, явные спецификации общих концепций в определенной области. Они используются для представления знаний в структурированной и стандартизированной форме и могут быть использованы для облегчения обмена информацией и интеграции данных.

15.2 Mechanizmy wnioskowań

Механизмы рассуждения – это методы, используемые для вывода новой информации из имеющихся данных и знаний. Они играют ключевую роль в области искусственного интеллекта и науки о данных. Ниже приведены некоторые основные типы механизмов рассуждения.

15.2.1 Дедуктивное рассуждение

Дедуктивное рассуждение – это процесс вывода конкретных выводов из общих принципов или предложений. Если все предположения верны и логика правильна, то вывод должен быть верным.

15.2.2 Индуктивное рассуждение

Индуктивное рассуждение – это процесс формирования общих утверждений на основе конкретных наблюдений или примеров. Оно обычно используется для создания новых теорий или гипотез.

15.2.3 Абдуктивное рассуждение

Абдуктивное рассуждение – это процесс формирования наиболее вероятного объяснения для определенного набора наблюдений. Оно часто используется в области диагностики и обнаружения причинно-следственных связей.

15.2.4 Аналоговое рассуждение

Аналоговое рассуждение - это процесс применения знаний или опыта, полученного в одном контексте, к новому контексту на основе обнаруженных сходств.

15.2.5 Статистическое рассуждение

Статистическое рассуждение - это использование теории вероятностей и статистического анализа для формирования выводов на основе данных.

15.2.6 Машинное обучение

Машинное обучение - это подраздел искусственного интеллекта, который использует механизмы рассуждения для создания моделей, способных обучаться и делать прогнозы на основе данных. Это включает в себя различные типы обучения, такие как обучение с учителем, обучение без учителя и обучение с подкреплением.

“‘latex

15.2.7 Искусственные нейронные сети

Искусственные нейронные сети - это модели, которые имитируют структуру и функцию биологических нейронных сетей. Они способны обучаться и адаптироваться к новым данным, что делает их мощным инструментом для многих задач машинного обучения.

15.3 Sposoby cyfrowej reprezentacji liczby całkowitej i rzeczywistej.

15.3.1 Целые числа

Целые числа обычно представляются в виде двоичных чисел. Вот некоторые из основных систем представления:

- Двоичная система: Числа представлены в виде последовательности двоичных цифр (0 и 1).
- Дополнительный код: Для представления отрицательных чисел обычно используется дополнительный код. В дополнительном коде отрицательное число представляется как "дополнение"соответствующего положительного числа.
- Двоично-десятичная кодировка: В этом случае каждая десятичная цифра представляется своим двоичным эквивалентом. Это обычно используется в финансовых вычислениях, где требуется высокая точность.

15.3.2 Вещественные числа

Вещественные числа обычно представляются в виде чисел с плавающей запятой. Наиболее распространенный стандарт представления чисел с плавающей запятой - это IEEE 754.

- Стандарт IEEE 754: Этот стандарт определяет, как представлять вещественные числа в двоичной форме и как выполнять арифметические операции с

числами с плавающей запятой. Он включает в себя специальные значения, такие как "не число"(NaN), "плюс бесконечность"и "минус бесконечность".