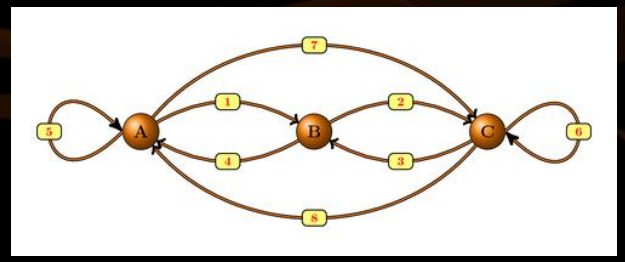
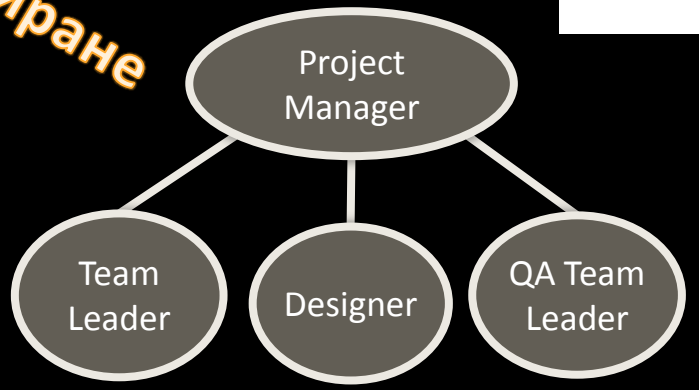
Други структури от данни

хеш таблици, дървета и графи





Учителски екип

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/

https://github.com/BG-IT-Edu/School-Programming/tree/main/Courses/Applied-Programmer/Programming-Fundamentals

Съдържание

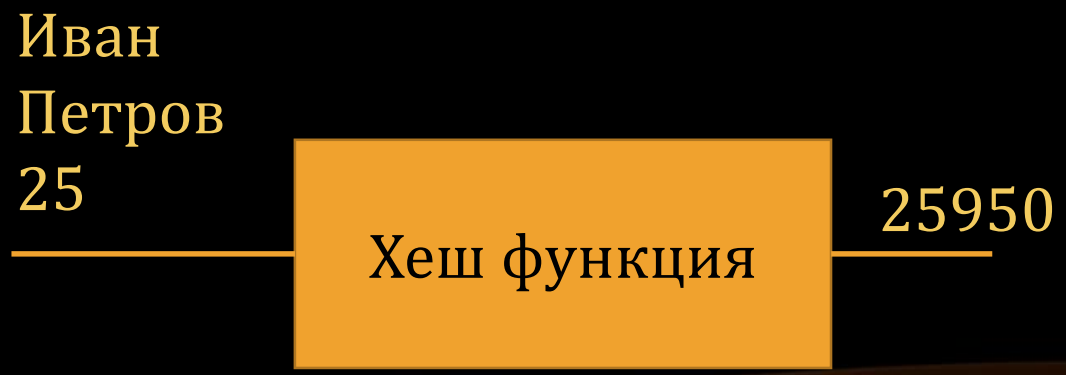
- Хеш таблици
  - Хеширащи функции
  - Управление на колизии
- Дървета
  - Подредени двоични дървета
  - Реализация на двоично дърво
- Графи и представяне на графи

Хеш таблици

Хеширащи функции

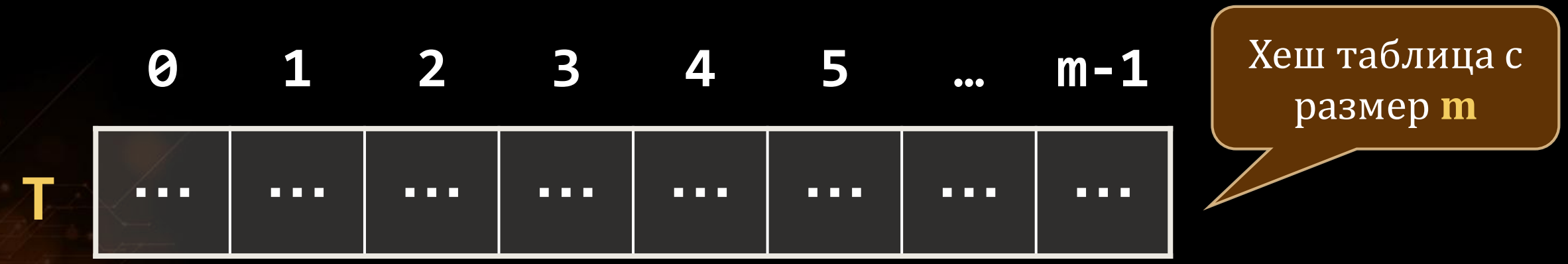
- Хеширащите функции конвертират ключ от произволен тип до стойност от целочислен тип


```
class Person
{
  string firstName;
  string lastName;
  int age;
}
```



Хеш таблица

- Хеш таблица е стандартен масив, който съдържа набор от наредени двойки {ключ, стойност}
- Чрез хешираща функция се определя кой ключ на коя позиция в масива да се съхрани
- Тази техниката се нарича хеширане



Модулна аритметика и хеш таблици

- Имаме масив с размер 10
- Въвеждаме “Pesho”

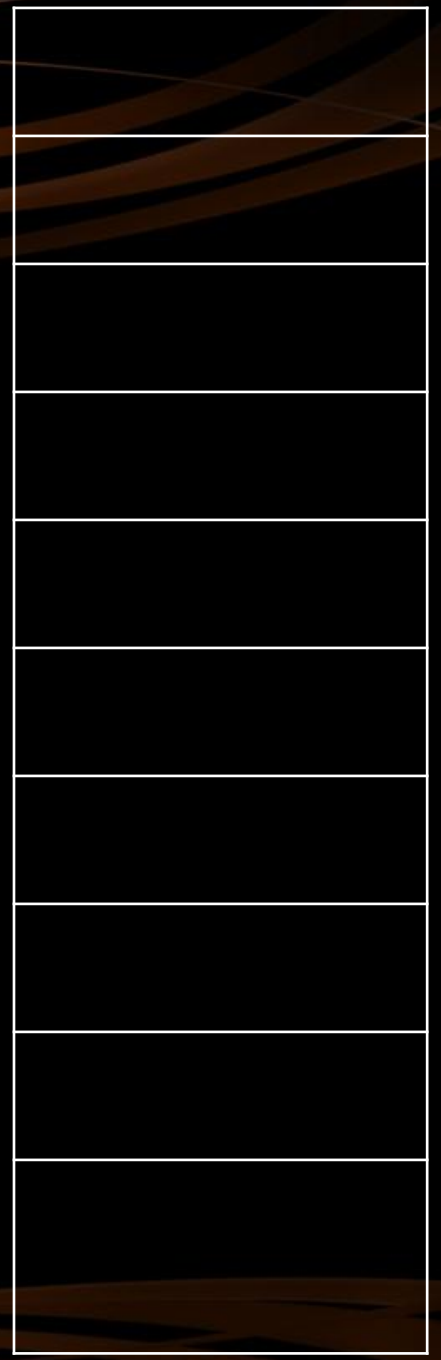
511 е извън размера на хеш таблицата

Pesho → Хеш функция → 511

- Използваме остатъка от делението за да извлечем валидна позиция:

GetHashCode( ) / Array.Length

511 % 10 = 1



Работа с хеш таблица

stamat

Hash Function % 10

Работа с хеш таблица

mitko

Hash Function % 10

Работа с хеш таблица

ivan

Hash Function % 10

Работа с хеш таблица

gosho

Hash Function % 10

Работа с хеш таблица

maria

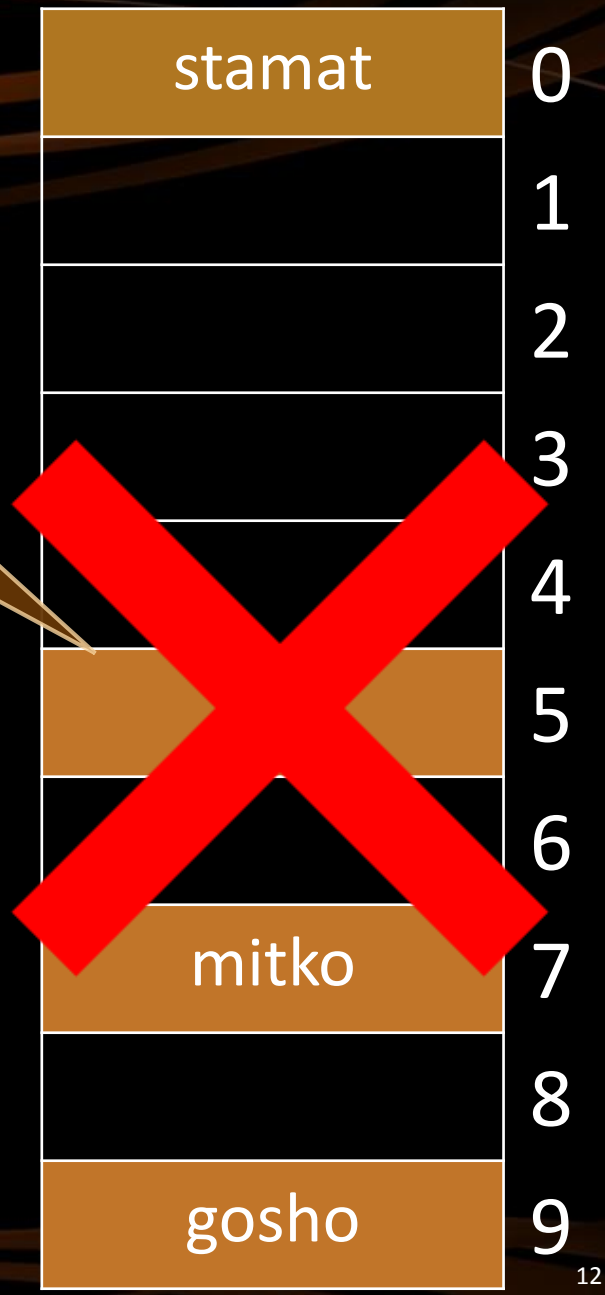
Hash Function % 10

Работа с хеш таблица

- Колизия настъпва, когато х … н
  и същ хеш за различни клю…
- При малко колизии, бърз… …ра
- Стратегии за разрешаване …
  - Свързване на елементите в колизия в списък
  - Използване на други свободни клетки от таблицата
  - Cuckoo хеширане и други...

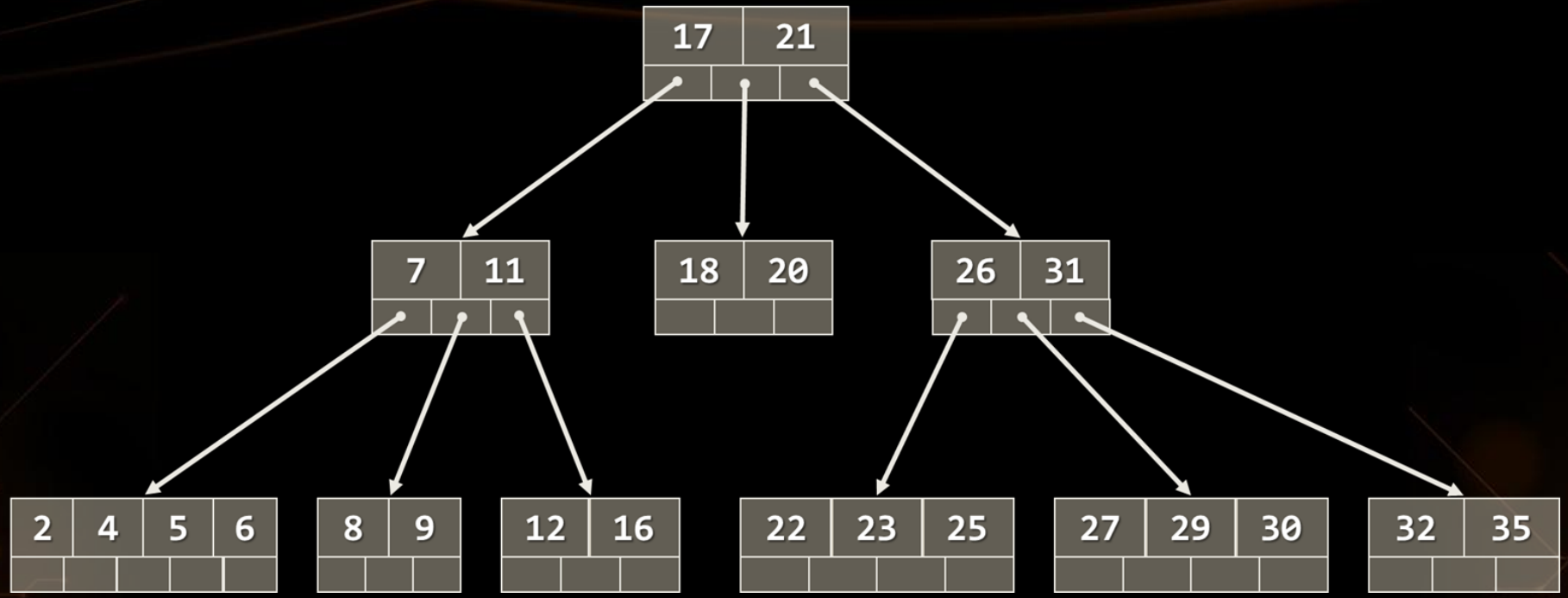
Колизия

Hash Function % 10



Хеширащи функции

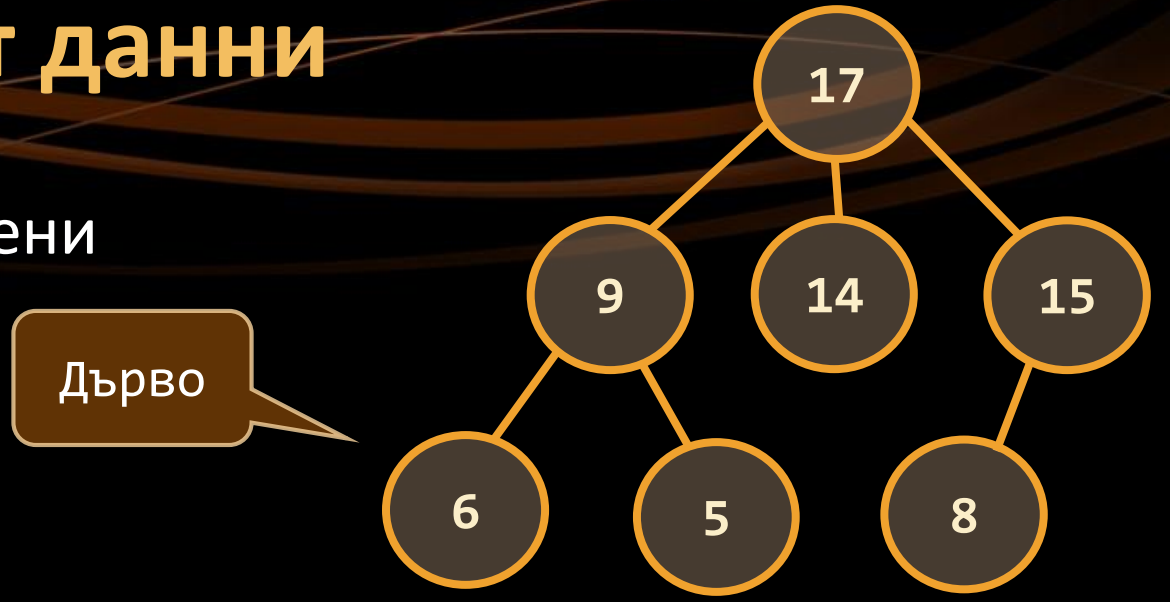
- Перфектно хеширане
  - Перфектно хешираща функция е тази, която прави 1:1 съответствие и свързва всеки ключ към уникално цяло число в рамките на конкретен интервал
  - В повечето случаи перфектното хеширане не е възможно
- Свойства на добрата хешираща функция
  - Консистентност - еднакви ключове => един и същ хеш
  - Ефективност – бързи при изчисляването на хеш-а
  - Равномерност - хешовете, произведени от хеширащата функция трябва да се равномерно разпределени



Дървовидни структури от данни

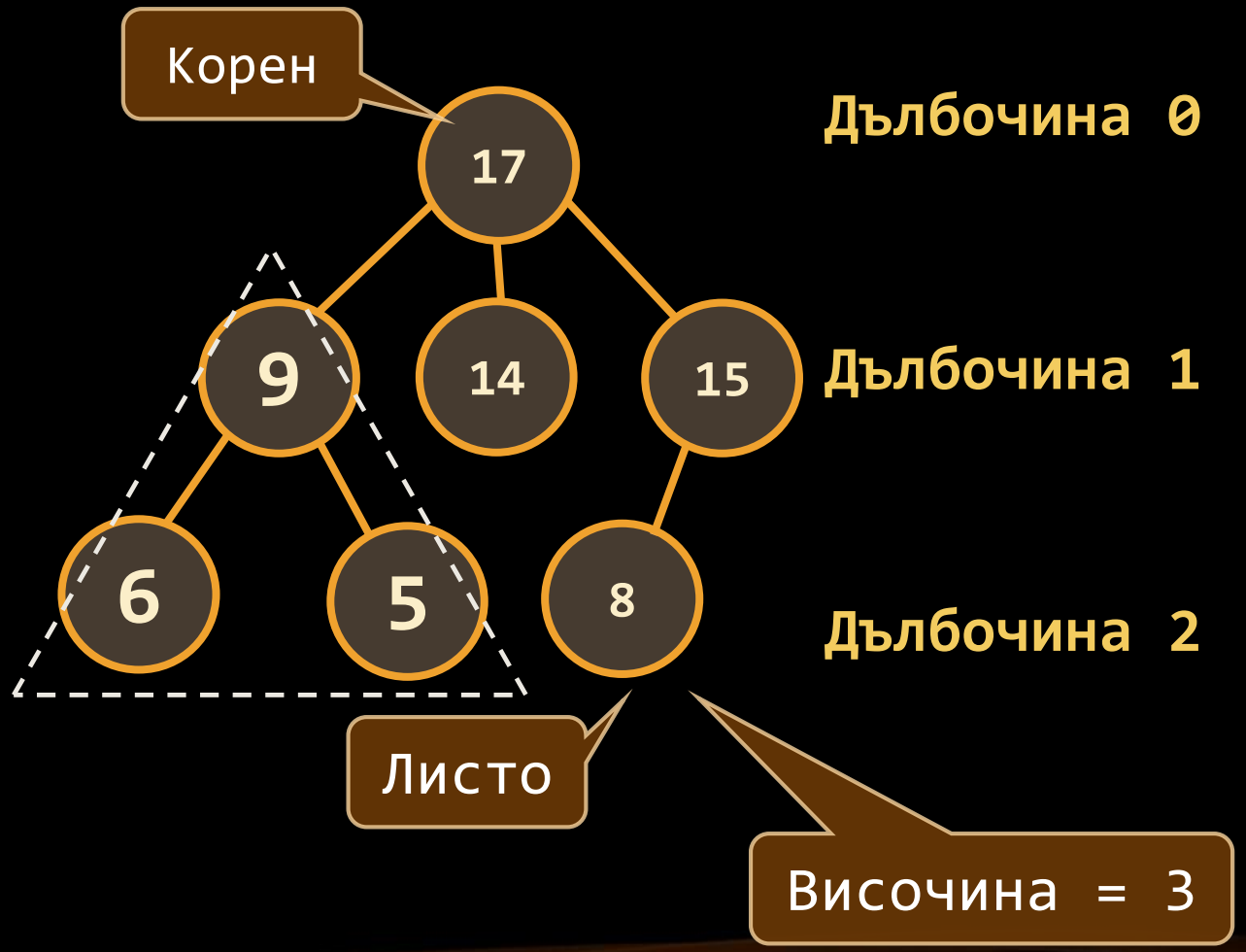
Дървовидни структури от данни

- Дървовидните структури са разклонени йерархични структури от данни
- Изградени са от възли (върхове)
- Всеки възел е свързан с други възли (разклонения на дървото) чрез ребра
- Върховете може да са:
  - Родител
  - Наследник
- Върхът без родител се нарича “корен”
  - Всяко дърво има само един корен
- Връх без наследници се нарича “листо”



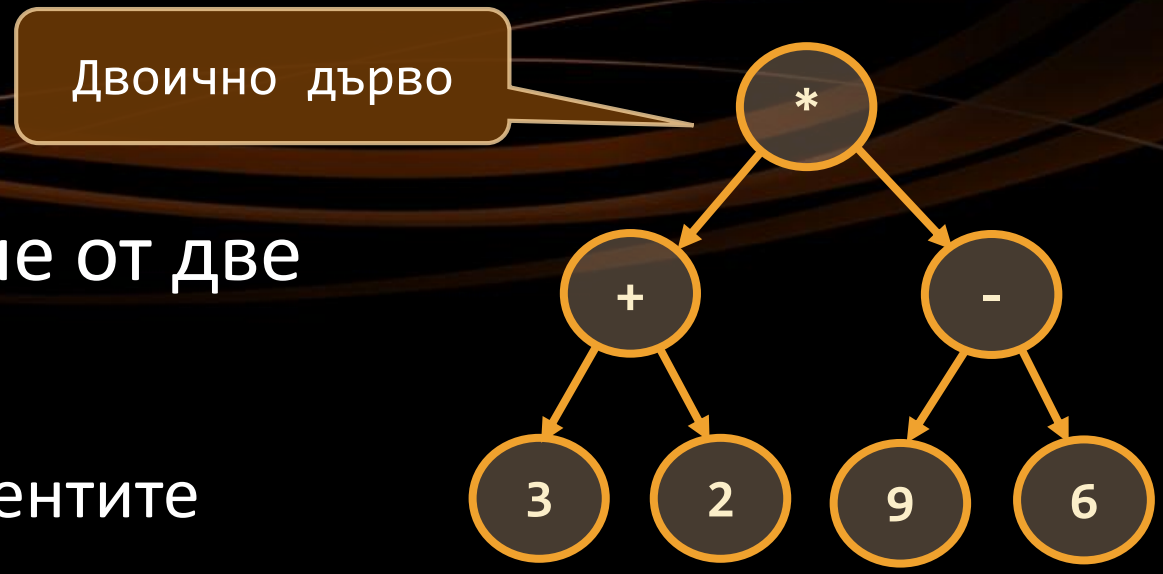
Дървовидни структури от данни – терминология

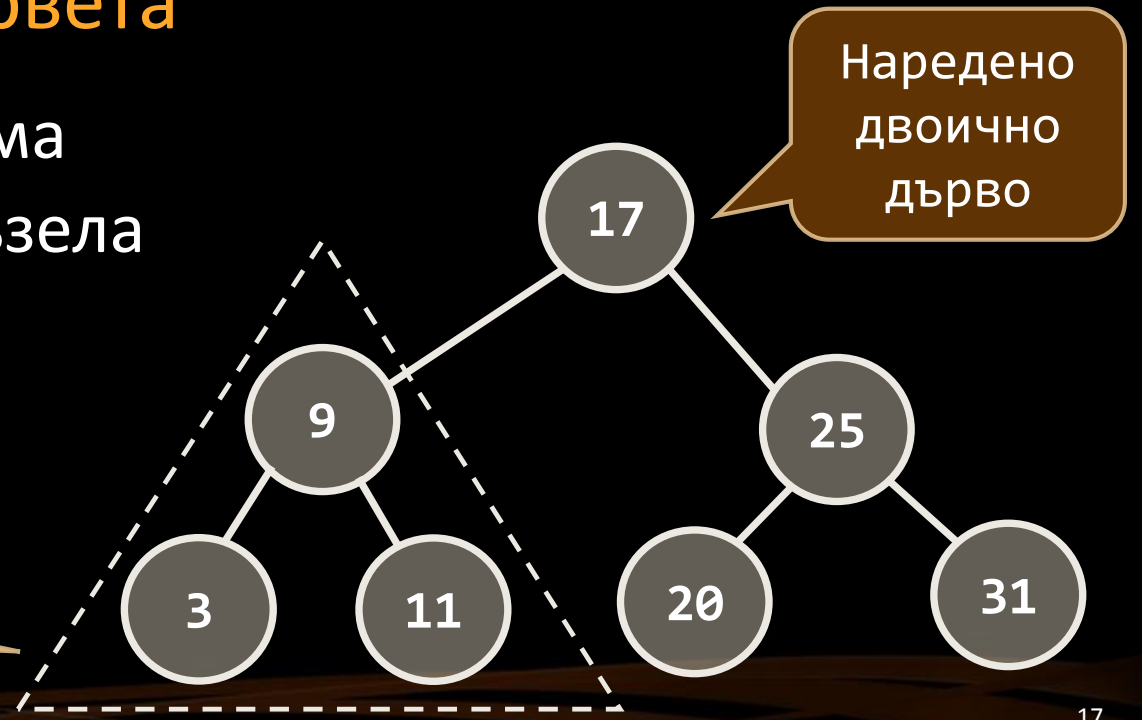
- Възел, ребро
- Корен, родител, дете, брат
- Дълбочина, височина
- Под-дърво
- Вътрешен възел, листо
- Предшественик, наследник

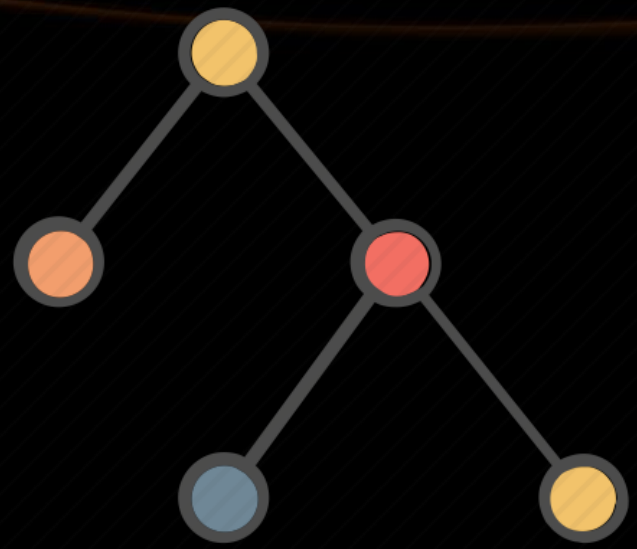


Двоични дървета

- Двоични дървета - имат не повече от две разклонения
  - няма правила за подредба на елементите
- Наредени (сортирани) двоични дървета
  - Лявото разклонение на всеки възел има по-малка стойност от стойността на възела
  - Дясното разклонение на всеки възел има по-голяма стойност от стойността на възела.





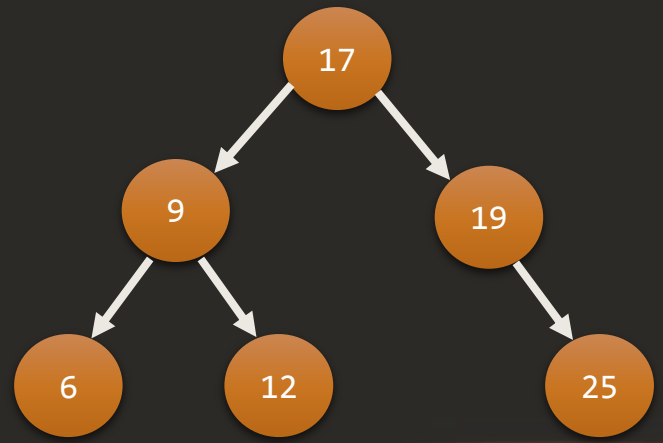


Двоични дървета за търсене

Реализация

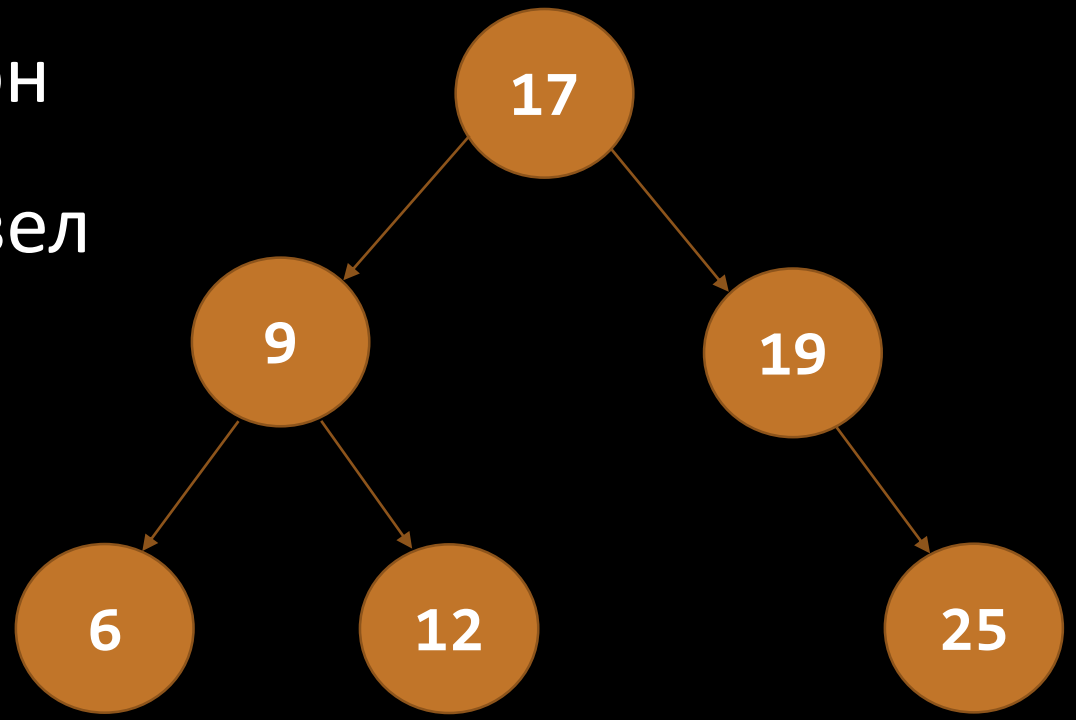
Двоично дърво за търсене - реализация

```
public class BinaryTree<T>
{
    private class Node
    {
        public Node Left { get; set; }
        public Node Right { get; set; }
        public T Item { get; set; }
    }

    private Node Root { get; set; }
    public int Count { get; private set; }
    public void Add(T item)...
    public void Remove(T item)...
    public bool Contains(T item)...
}
```



Двоично дърво за търсене – търсене на елемент

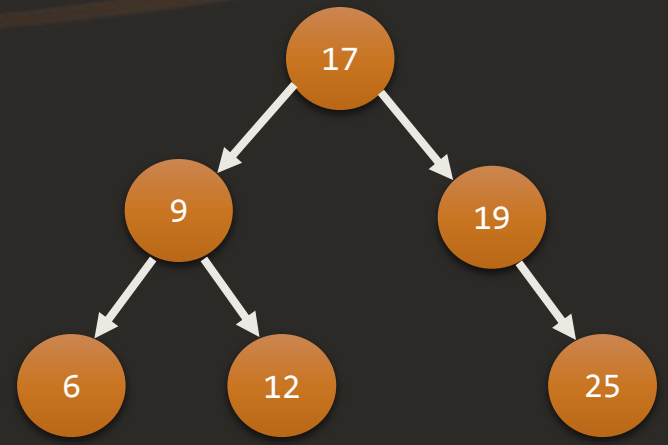
- if node != null
  - if x < node.value -> левия клон
  - else if x > node.value -> десния клон
  - else if x == node.value -> върни възел




```
Търсим 12 -> 17 9 12
Търсим 27 -> 17 19 25 null
```

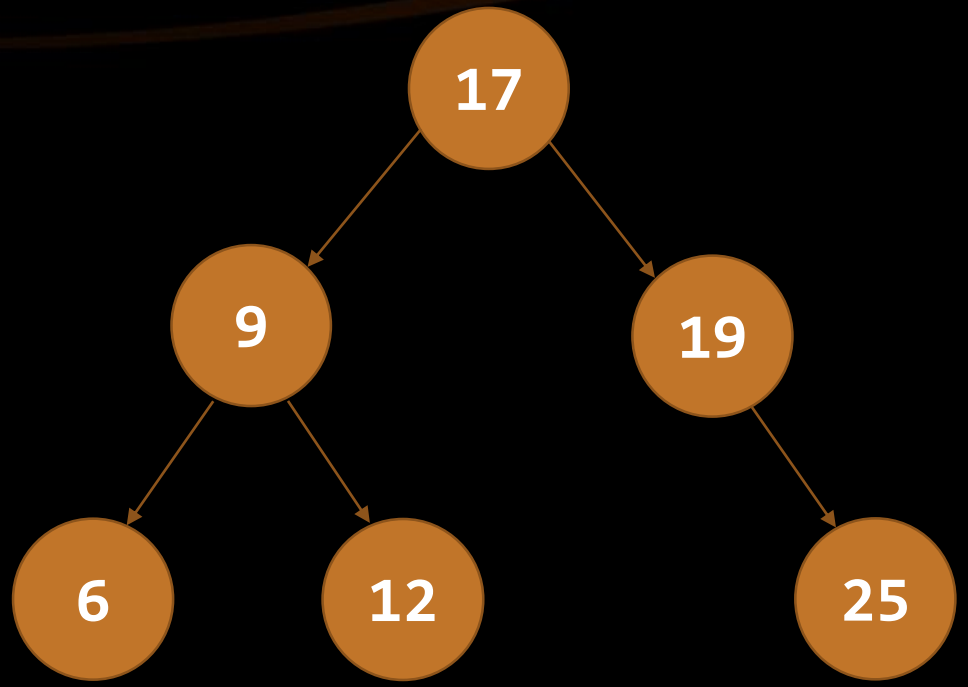
Двоично дърво за търсене - търсене

```
public bool Contains(T item) {
    if (Root == null)
        return false;

    Node iterator = Root;
    while (true)     {
        if (iterator == null)
            return false;
        else if (iterator.Item.CompareTo(item) == 0)
            return true;
        else if (iterator.Item.CompareTo(item) > 0)
            iterator = iterator.Left;
        else if (iterator.Item.CompareTo(item) < 0)
            iterator = iterator.Right;
    }
}
```



Двоично дърво за търсене – добавяне на елемент

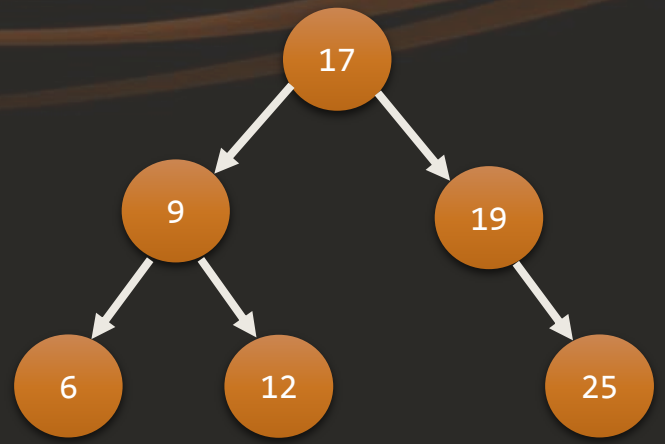
- if node == null -> добави x
- else if x < node.value -> ляв клон
- else if x > node.value -> десен клон
- else -> възела съществува




```
Добавяне 12   17 9 12 return
Добавяне 27   17 19 25 null (добавяне)
```

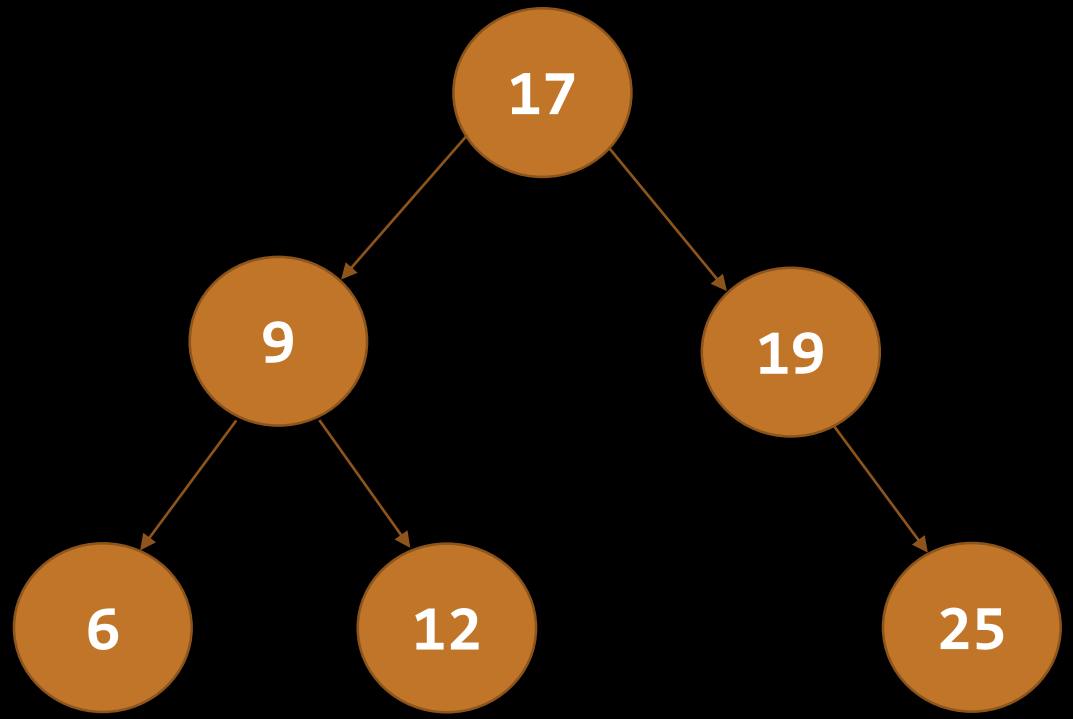
Двоично дърво за търсене – добавяне на елемент

```
public void Add(T item) {
    Node node = new Node();
    node.Item = item;

    if (Root == null) {
        Root = node;
        return;
    }

    Node iterator = Root;
    while (true) {
        if (iterator.Left != null && iterator.Item.CompareTo(item) >= 0)
            iterator = iterator.Left;
        else if (iterator.Right != null && iterator.Item.CompareTo(item) < 0)
            iterator = iterator.Right;
        else break;
    }

    if (iterator.Item.CompareTo(item) >= 0)
        iterator.Left = node;
    else if (iterator.Item.CompareTo(item) < 0)
        iterator.Right = node;
}
```



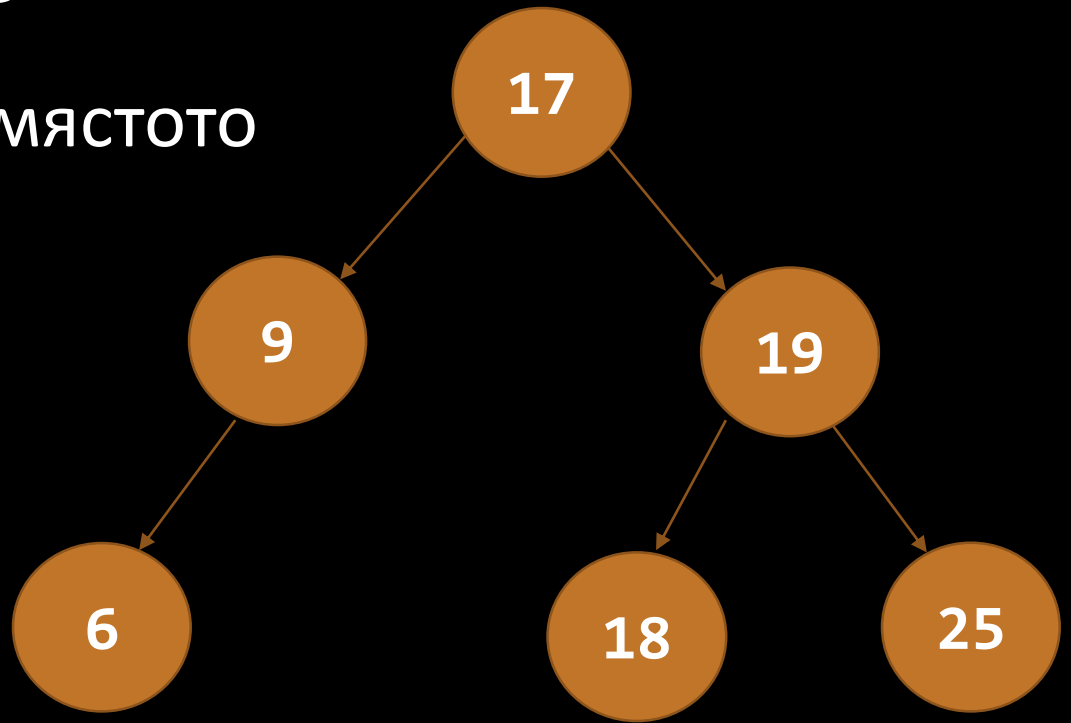
Двоично дърво за търсене – изтриване на елемент

- if node == null -> изход
- else if x is leaf -> премахни
- else if x is not leaf -> подмени
- (3 случая при подмяна на възел)



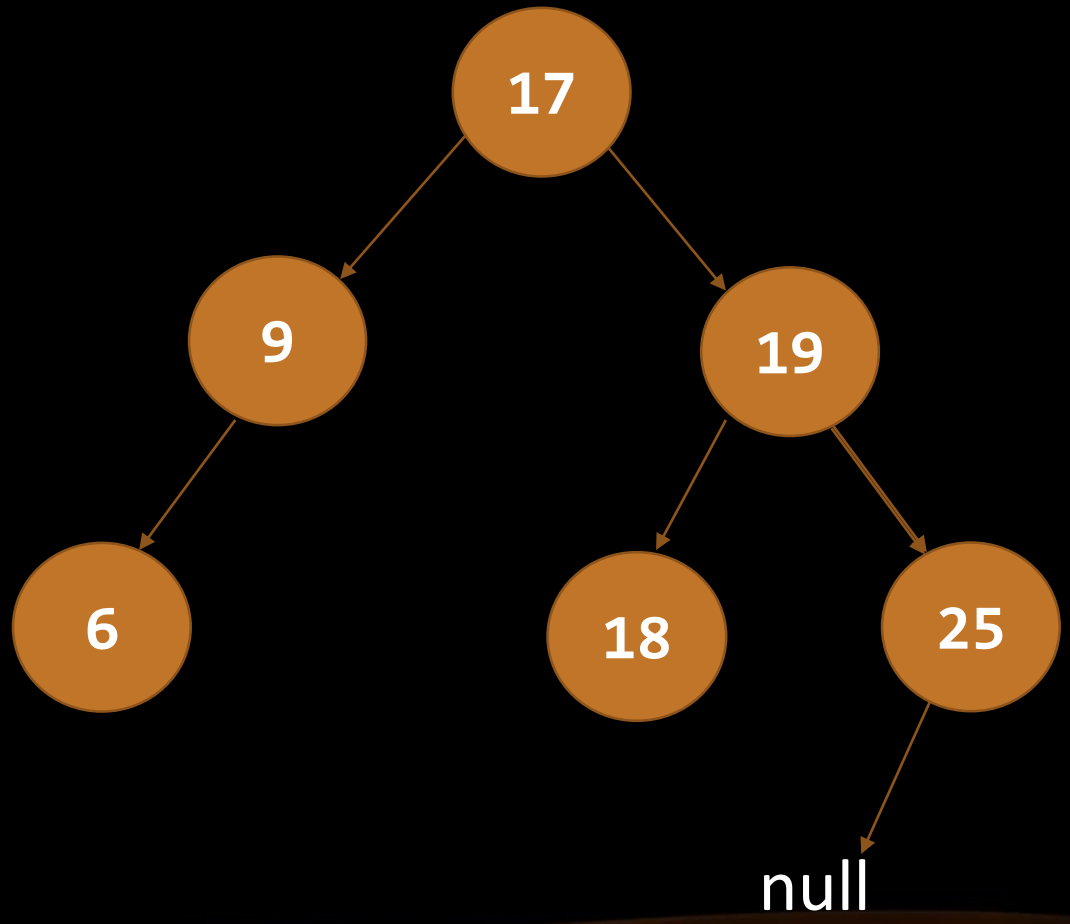
Двоично дърво за търсене - премахване

- На елемент, който няма дясно поддърво – например 9
  - Намираме елемента за премахване
  - Корена на лявото поддърво заема мястото на премахнатия елемент



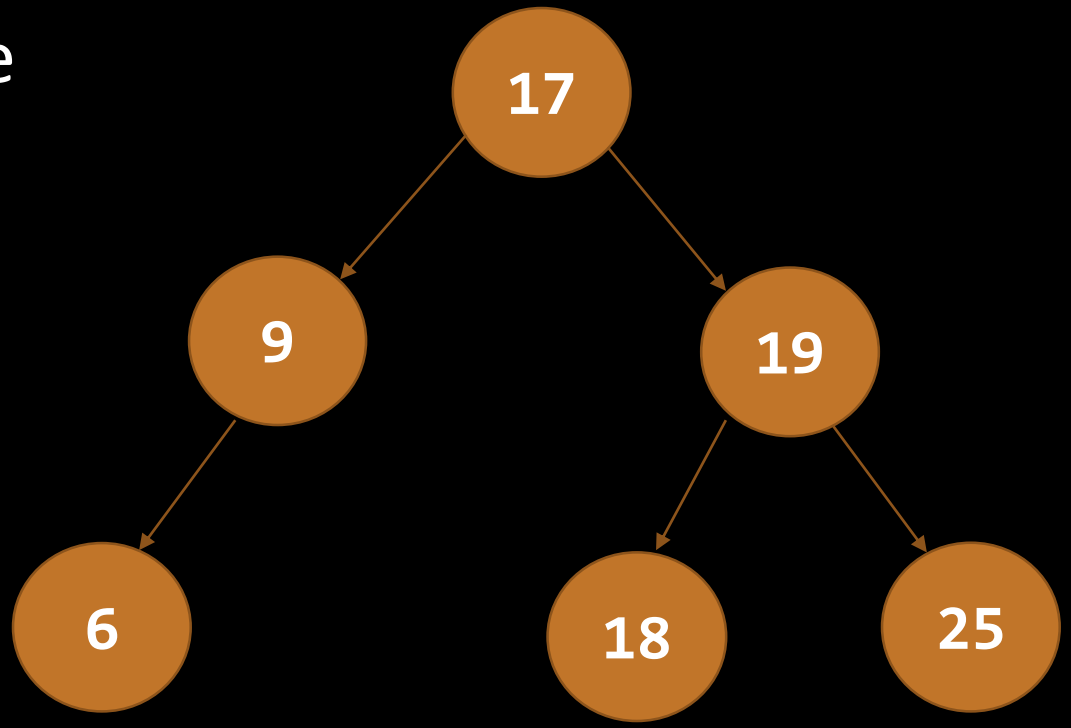
Двоично дърво за търсене - премахване

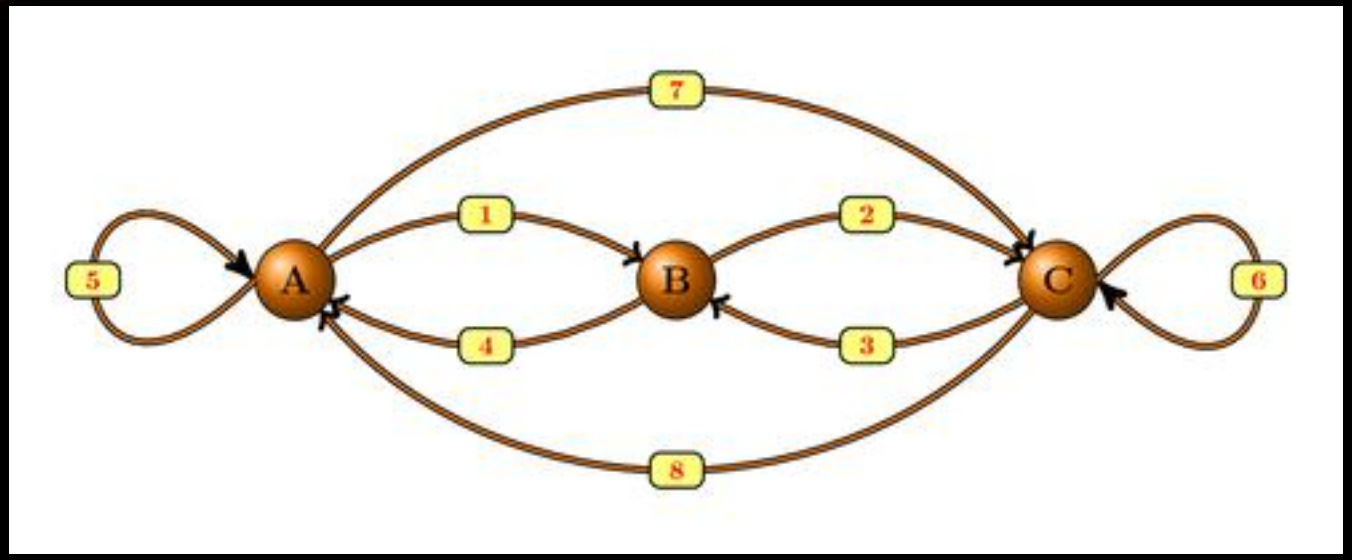
- На елемент, чието дясно поддърво няма ляво поддърво - 19
  - Намираме елемента за премахване
  - Корена на дясното поддърво заема мястото на премахнатия елемент



Двоично дърво за търсене - премахване

- На елемент с ляво и дясно поддърво – например 17
  - Намираме елемента за премахване
  - Намираме най-малкия елемент в лявото разклонение на дясното му поддърво
  - Разменяме двата елемента и извършваме премахването

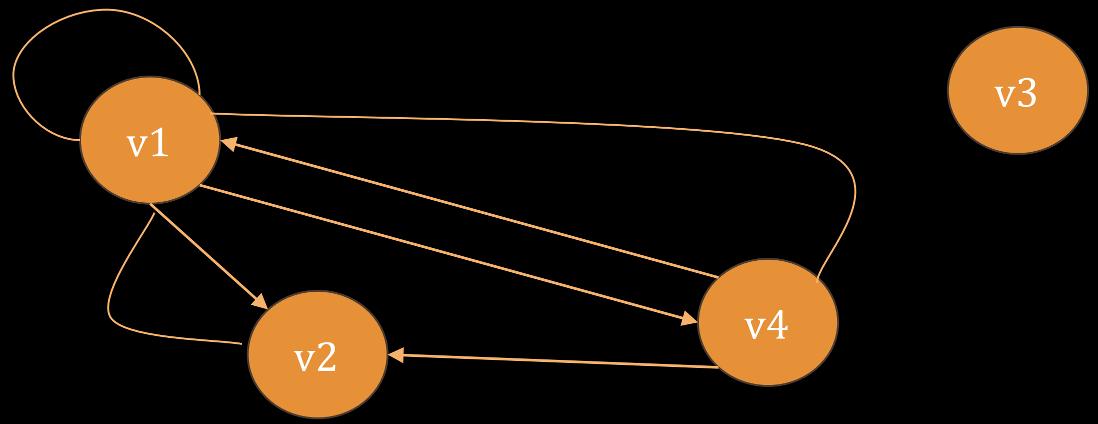


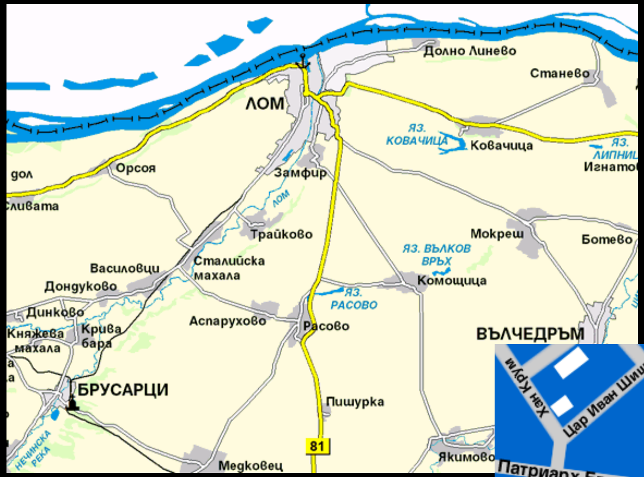


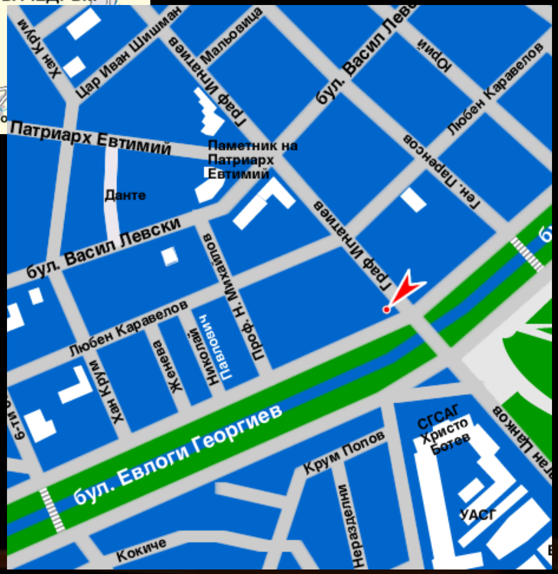
Графи

Графи

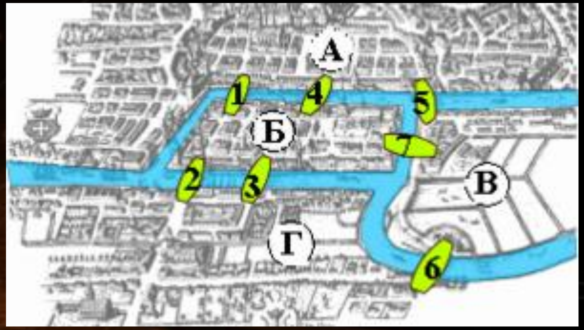
- Нелинейна структура от данни, съдържаща крайно непразно множество от точки, наречени **върхове** (или **възли**), свързани помежду си с линии, наречени **ребра**.

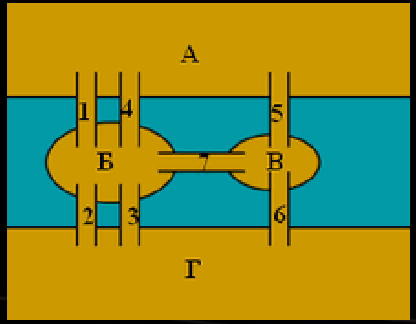


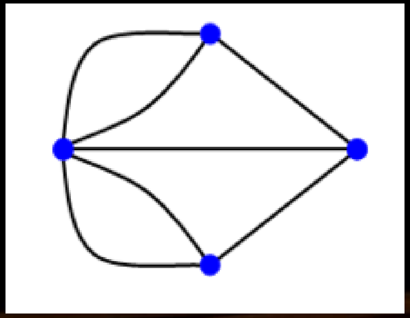




- Леонард Ойлер: задача за "седемте моста на Кьонингсберг"

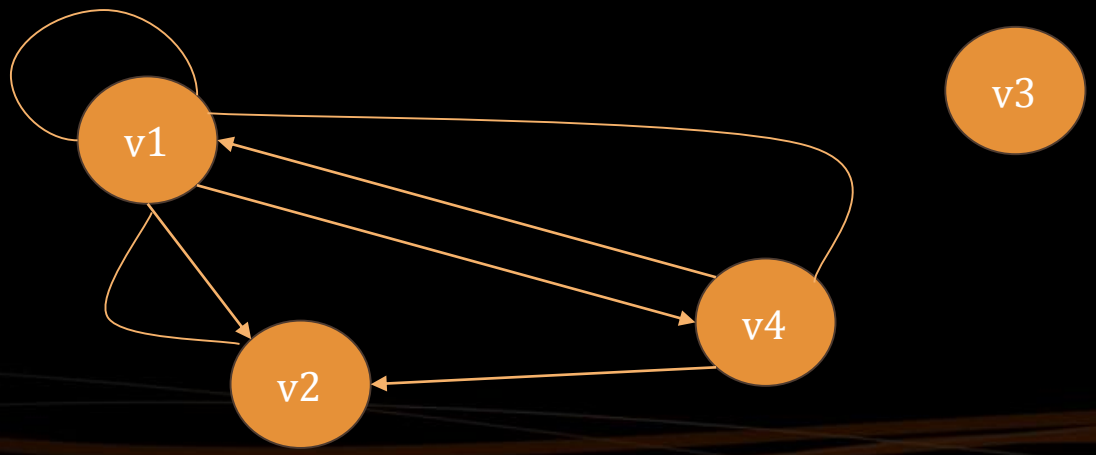






Графи – основни понятия

- степен на връх – броят на ребрата, чрез които даден връх е свързан с другите върхове
- изолиран връх – в който не влизат и не излизат ребра, връх от степен 0
- примка – ребро, чието начало и край съвпадат
- паралелни ребра – когато два върха са свързани с повече от едно ребра

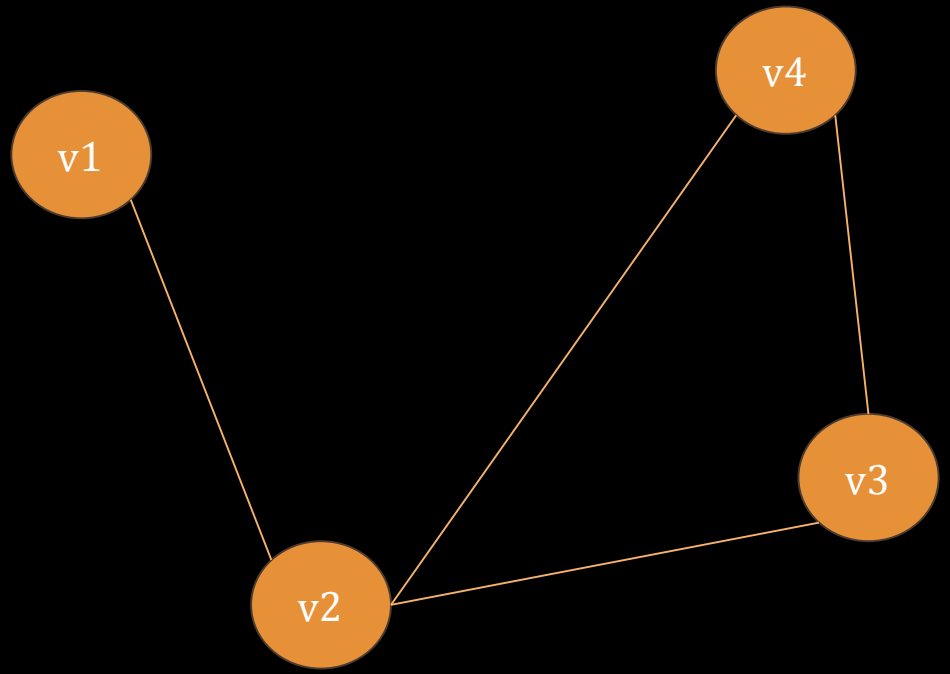


Ориентиран граф

- всяко ребро се определя еднозначно от съответната двойка върхове и има посока (начален и краен връх)

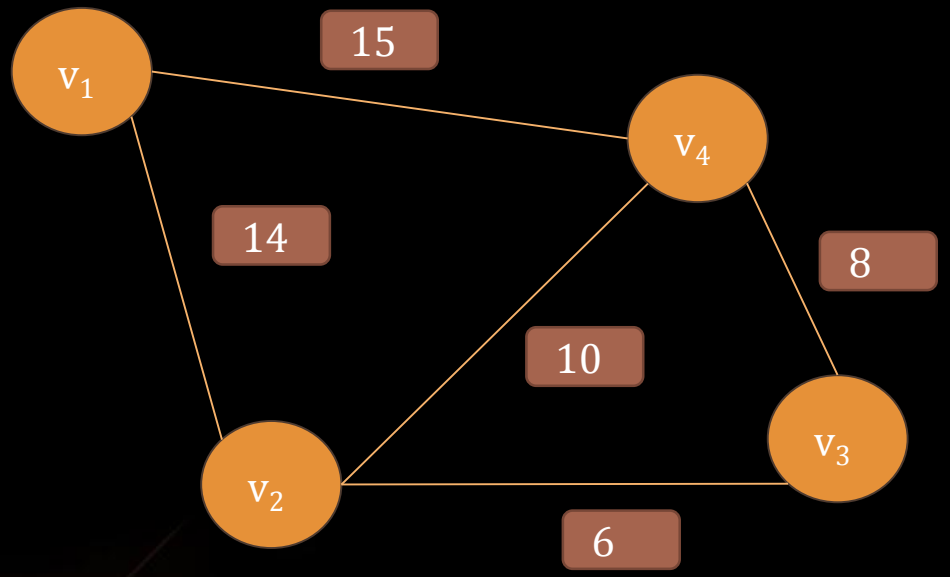
Неориентиран граф

- когато ребрата нямат посока



Претеглен граф

- когато всяко ребро има тегло

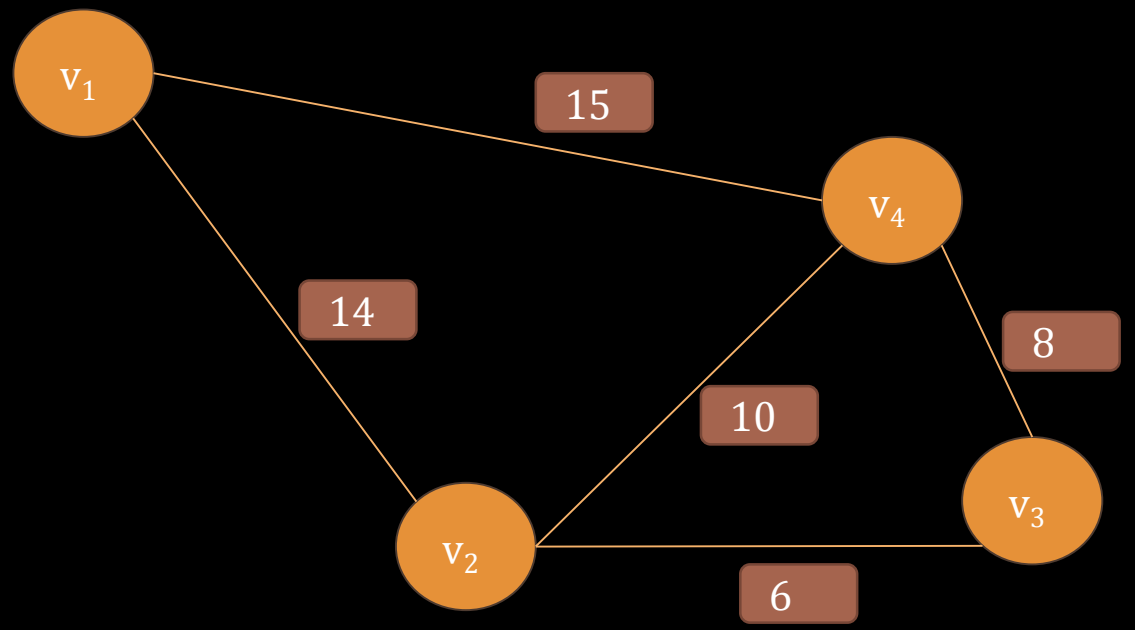


Път в граф

- път – редица от дъги, свързваща два върха
- дължина на пътя – броят на дъгите, които свързват два върха
- прост път – път без повтарящи се дъги
- цикъл – път, чиито начало и край съвпадат
- цикличен граф – ако има поне един цикъл
- свързан граф - ако между всяка двойка върхове съществува път

Представяне на граф

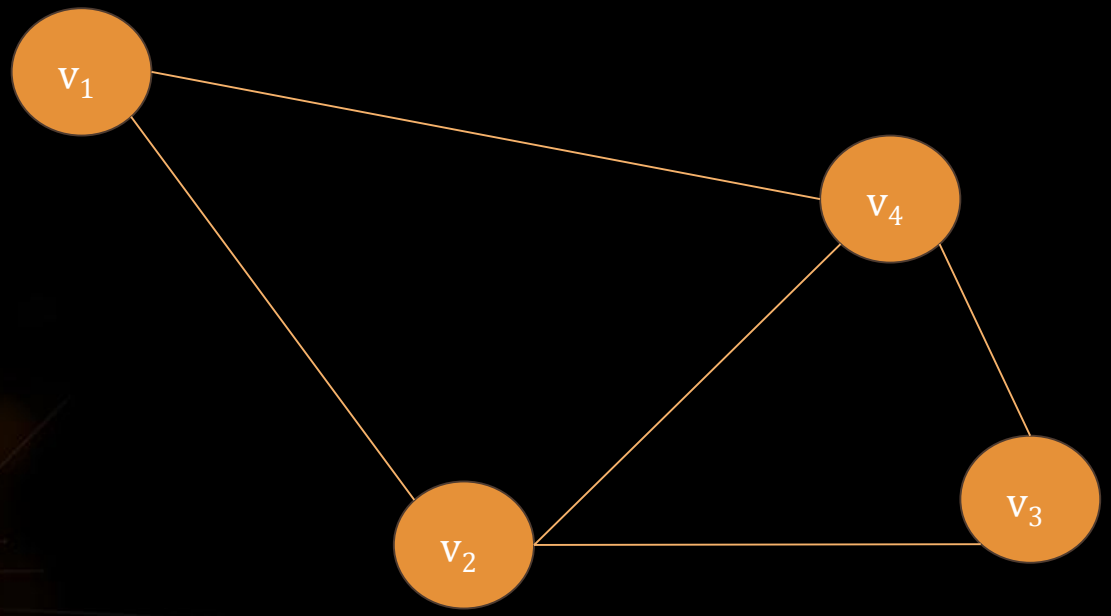
- Списък на съседите
- Всеки връх съдържа списък на своите съседи
- v_1 -> {v_2, v_4 }
- v_2 -> { v_1, v_4, v_3 }
- v_3 -> { v_2, v_4 }
- v_4 -> {v_1, v_2, v_3 }



Представяне на граф

- Матрица на свързаност
- 1 - ако има свързващо ребро
- 0 - ако няма свързващо ребро

връх	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	1
v_3	0	1	0	1
v_4	1	1	1	0



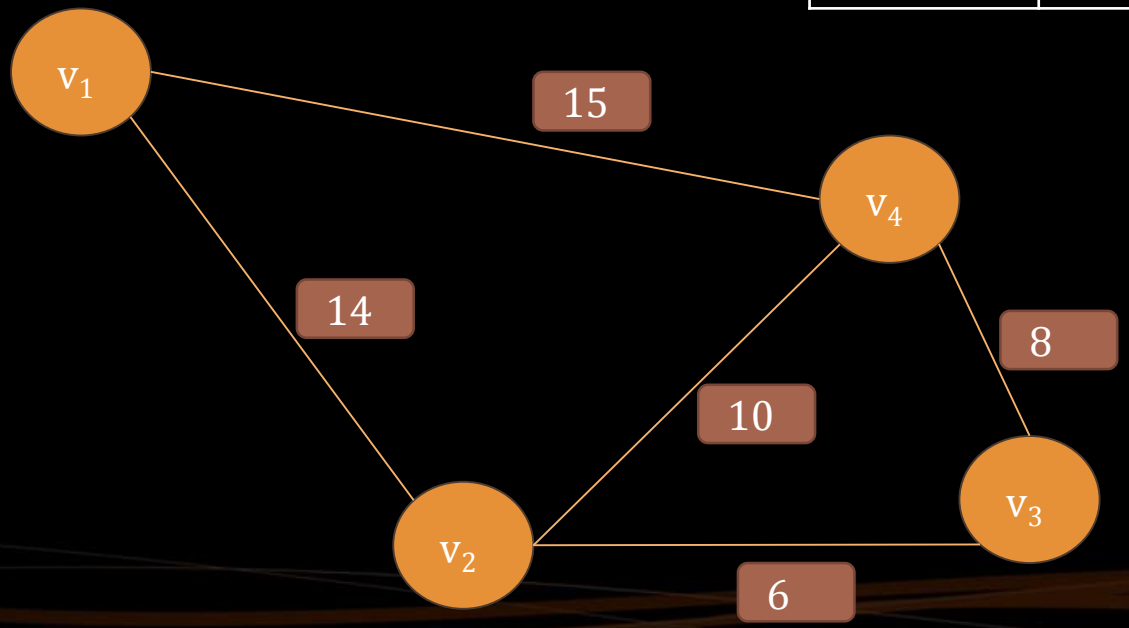
Представяне на граф

- Матрица на свързаност
- Стойността на теглото -

ако има свързващо ребро

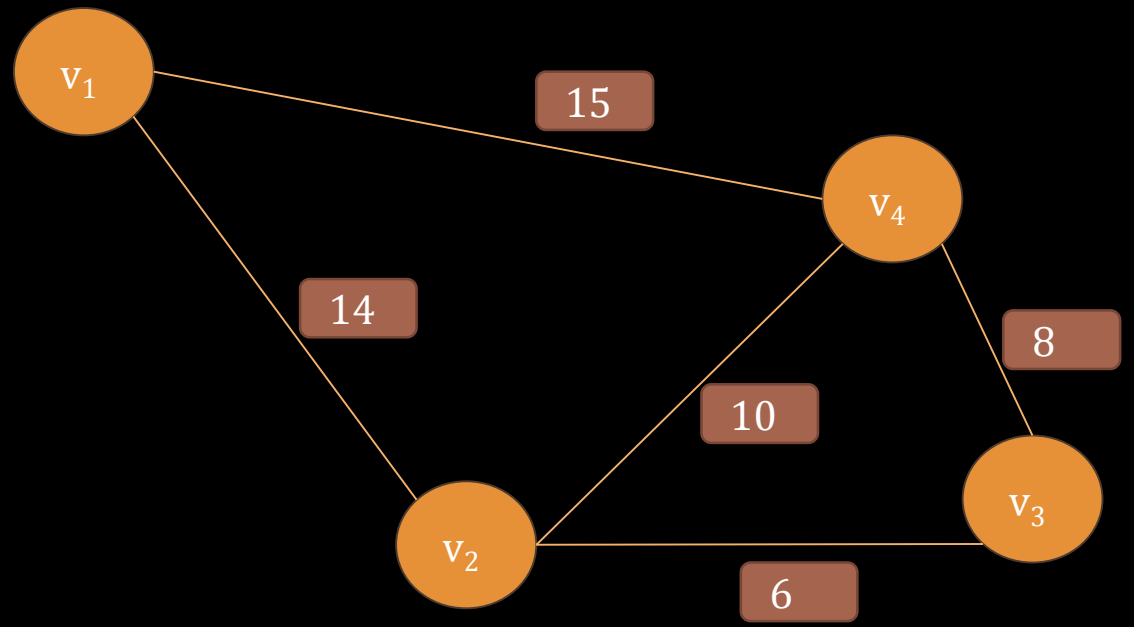
- 0 - ако няма свързващо ребро

връх	v_1	v_2	v_3	v_4
v_1	0	14	0	15
v_2	14	0	6	10
v_3	0	6	0	8
v_4	15	10	8	0



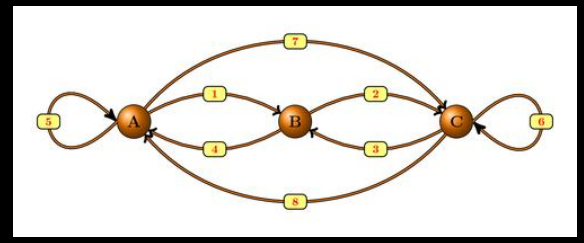
Представяне на граф

- Списък на ребрата
- Изброяват се всички ребра, прекарани в графа
- $\{v_1, v_2\}$
- $\{v_1, v_4\}$
- $\{v_2, v_4\}$
- $\{v_2, v_3\}$
- $\{v_3, v_4\}$



Какво научихме този час?

- Хеш таблици – масив + хеширане
  - Хеширащи функции – преобразува ключ в цяло число (по възможност уникално)
  - Колизии – ако има два ключа с един и същ хеш
- Дървета - разклонени йерархични структури
  - Подредени двоични дървета – всеки възел е с най-много два наследника; левият наследник има по-малка стойност от възела, десният – по-голяма
- Графи - крайно множество от върхове, свързани помежду си с ребра



Други структури от данни

Въпроси?

https://github.com/BG-IT-Edu/School-Programming/tree/main/Courses/Applied-Programmer/Programming-Fundamentals

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**