عنوان: **درخت‌ها و گراف‌ها قسمت اول**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۰۲ ۱:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Data Structures, Tree

در این مقاله یکی از ساختارهای داده را به نام ساختارهای درختی و گراف‌ها معرفی کردیم و در این مقاله قصد داریم این نوع ساختار را بیشتر بررسی نماییم. این ساختارها برای بسیاری از برنامه‌های مدرن و امروزی بسیار مهم هستند. هر کدام از این ساختارهای داده به حل یکی از مشکلات دنیای واقعی می‌پردازند. در این مقاله قصد داریم به مزایا و معایب هر کدام از این ساختارها اشاره کنیم و اینکه کی و کجا بهتر است از کدام ساختار استفاده گردد. تمرکز ما بر **درخت هایی دودویی** ، **درخت‌های جست و جوی دو دویی** و **درخت‌های جست و جوی دو دویی متوازن** خواهد بود. همچنین ما به تشریح گراف و انواع آن خواهیم پرداخت. اینکه چگونه آن را در حافظه نمایش دهیم و اینکه گراف‌ها در کجای زندگی واقعی ما یا فناوری‌های کامپیوتری استفاده می‌شوند.

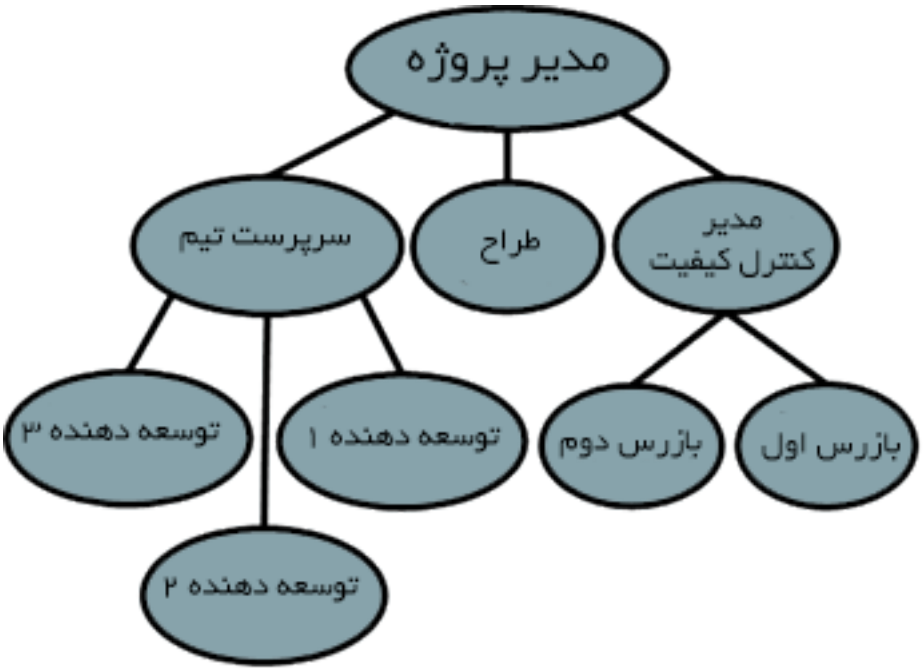
ساختار درختی

در بسیاری از مواقع ما با گروهی از اشیاء یا داده‌هایی سر و کار داریم که هر کدام از آن‌ها به گروهی دیگر مرتبط هستند. در این حالت از ساختار خطی نمی‌توانیم برای توصیف این ارتباط استفاده کنیم. پس بهترین ساختار برای نشان دادن این ارتباط **ساختار شاخه ای Branched Structure** است.

یک ساختار درختی یا یک ساختار شاخه‌ای شامل المان‌هایی به اسم گره Node است. هر گره می‌تواند به یک یا چند گره دیگر متصل باشد و گاهی اوقات این اتصالات مشابه یک *سلسه مراتب hierarchically* می‌شوند.

درخت‌ها در برنامه نویسی جایگاه ویژه‌ای دارند به طوری که استفاده‌ی از آن‌ها در بسیاری از برنامه‌ها وجود دارد و بسیاری از مثال‌های واقعی پیرامون ما را پشتیبانی می‌کنند.

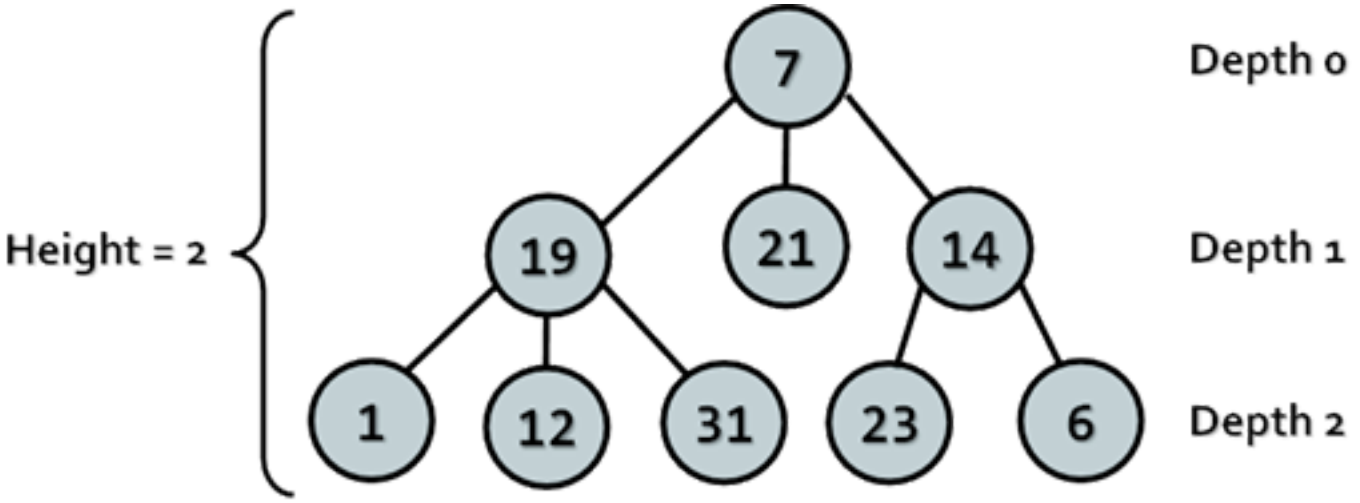
در نمودار زیر مثالی وجود دارد که در آن یک تیم نرم افزاری نمایش داده شده‌است. در اینجا هر یک از بخش‌ها وظایف و مسئولیت‌هایی را بر دوش خود دارند که این مسئولیت‌ها به صورت سلسله مراتبی در تصویر زیر نمایش داده شده‌اند.



ما در ساختار بالا متوجه می‌شویم که چه بخشی زیر مجموعه‌ی چه بخشی است و سمت بالاتر هر بخش چیست. برای مثال ما متوجه شدیم که مدیر توسعه دهندگان، "سرپرست تیم" است که خود نیز مادون "مدیر پروژه" است و این را نیز متوجه می‌شویم که مثلا توسعه دهنده‌ی شماره یک هیچ مادونی ندارد و مدیر پروژه در راس همه است و هیچ مدیر دیگری بالای سر او قرار ندارد.

اصطلاحات درخت

برای اینکه بیشتر متوجه روابط بین اشیا در این ساختار بشویم، به شکل زیر خوب دقت کنید:



در شکل بالا دایره‌هایی برای هر بخش از اطلاعت کشیده شده و ارتباط هر کدام از آن‌ها از طریق یک خط برقرار شده است. اعداد داخل هر دایره تکراری نیست و همه منحصر به فرد هستند. پس وقتی از اعداد اسم ببریم متوجه می‌شویم که در مورد چه چیزی صحبت می‌کنیم.

در شکل بالا به هر یک از دایره‌ها یک **گره Node** می‌گویند و به هر خط ارتباط دهنده بین گره‌ها **لبه Edge** گفته می‌شود. گره‌های 19 و 21 و 14 زیر گره‌های گره 7 محسوب می‌شوند. گره‌هایی که به صورت مستقیم به زیر گره‌های خودشان اشاره می‌کنند را **گره‌های والد Parent** می‌گویند و زیرگره‌های 7 را گره‌های **فرزند ChildNodes** . پس با این حساب می‌توانیم بگوییم گره‌های 1 و 12 و 31 را هم فرزند گره 19 هستند و گره 19 والد آن هاست. همچنین گره‌های یک والد را مثل 19 و 21 و 14 که والد مشترک دارند، گره‌های **خواهر و برادر** یا **حتی همنژاد Sibling** می‌گوییم. همچنین ارتباط بین گره 7 و گره‌های سطح دوم و الی آخر یعنی 1 و 12 و 31 و 23 و 6 را که والد بودن آن به صورت غیر مستقیم است را **جد** یا **ancestor** می‌نامیم و نوه‌ها و نتیجه‌های آن‌ها را **نسل descendants** .

ریشه Root : به گره‌ای می‌گوییم که هیچ والدی ندارد و خودش در واقع اولین والد محسوب می‌شود؛ مثل گره 7.

برگ Leaf : به گره‌هایی که هیچ فرزندی ندارند، برگ می‌گوییم. مثال گره‌های 1 و12 و 31 و 23 و 6

گره‌های داخلی Internal Nodes : گره هایی که نه برگ هستند و نه ریشه. یعنی حداقل یک فرزند دارند و خودشان یک گره فرزند محسوب می‌شوند؛ مثل گره‌های 19 و 14.

مسیر Path : راه رسیدن از یک گره به گره دیگر را مسیر می‌گویند. مثلا گره‌های 1 و 19 و 7 و 21 به ترتیب یک مسیر را تشکیل می‌دهند ولی گره‌های 1 و 19 و 23 از آن جا که هیچ جور اتصالی بین آن‌ها نیست، مسیری را تشکیل نمی‌دهند.

طول مسیر Length of Path : به تعداد لبه‌های یک مسیر، طول مسیر می‌گویند که می‌توان از تعداد گره‌ها -1 نیز آن را به دست آورد. برای نمونه : مسیر 1 و19 و 7 و 21 طول مسیرشان 3 هست.

عمق Depth : طول مسیر یک گره از ریشه تا آن گره را عمق درخت می‌گویند. عمق یک ریشه همیشه صفر است و برای مثال در درخت بالا، گره 19 در عمق یک است و برای گره 23 عمق آن 2 خواهد بود.

تعریف خود درخت Tree : درخت یک ساختار داده برگشتی recursive است که شامل گره‌ها و لبه‌ها، برای اتصال گره‌ها به

یکدیگر است.

جملات زیر در مورد درخت صدق می‌کند:

هر گره می‌تواند فرزند نداشته باشد یا به هر تعداد که می‌خواهد فرزند داشته باشد.
هر گره یک والد دارد و تنها گره‌ای که والد ندارد، گره ریشه است (البته اگر درخت خالی باشد هیچ گره ای وجود ندارد).
همه گره‌ها از ریشه قابل دسترسی هستند و برای دسترسی به گره مورد نظر باید از ریشه تا آن گره، مسیری را طی کرد.
ار تفاع درخت Height: به حداکثر عمق یک درخت، ارتفاع درخت می‌گویند. **درجه گره Degree** : به تعداد گره‌های فرزند یک گره، درجه آن گره می‌گویند. در درخت بالا درجه گره‌های 7 و 19 سه است. درجه گره 14 دو است و درجه برگ‌ها صفر است. **ضریب انشعاب Branching Factor:** به حداکثر درجه یک گره در یک درخت، ضریب انشعاب آن درخت گویند.

پیاده سازی درخت

برای پیاده سازی یک درخت، از دو کلاس یکی جهت ساخت گره که حاوی اطلاعات است <TreeNode<T و دیگری جهت ایجاد درخت اصلی به همراه کلیه متدها و خاصیت هایش <Tree<T کمک می‌گیریم.

```
public class TreeNode<T>
{
    // شامل مقدار گره است
    private T value;

    // مشخص می‌کند که آیا گره والد دارد یا خیر
    private bool hasParent;

    // در صورت داشتن فرزند ، لیست فرزندان را شامل می‌شود
    private List<TreeNode<T>> children;

    /// <summary>سازنده کلاس </summary>
    /// <param name="value">مقدار گره</param>
    public TreeNode(T value)
    {
        if (value == null)
        {
            throw new ArgumentNullException(
                "Cannot insert null value!");
        }
        this.value = value;
        this.children = new List<TreeNode<T>>();
    }

    /// <summary>خاصیتی جهت مقداردهی گره</summary>
    public T Value
    {
        get
        {
            return this.value;
        }
        set
        {
            this.value = value;
        }
    }

    /// <summary>تعداد گره‌های فرزند را بر میگرداند</summary>
    public int ChildrenCount
    {
        get
        {
            return this.children.Count;
        }
    }

    /// <summary>به گره یک فرزند اضافه می‌کند</summary>
    /// <param name="child">آرگومان این متد یک گره است که قرار است به فرزندی گره فعلی در آید</param>
    public void AddChild(TreeNode<T> child)
    {
        if (child == null)
        {
            throw new ArgumentNullException(
                "Cannot insert null value!");
        }

        if (child.hasParent)
```

```
        {
            throw new ArgumentException(
                "The node already has a parent!");
        }

        child.hasParent = true;
        this.children.Add(child);
    }

    /// <summary>
    /// گره ای که اندیس آن داده شده است بازگردانده می‌شود
    /// </summary>
    /// <param name="index">اندیس گره</param>
    /// <returns>گره بازگشتی</returns>
    public TreeNode<T> GetChild(int index)
    {
        return this.children[index];
    }
}

/// <summary>این کلاس ساختار درخت را به کمک کلاس گره‌ها که در بالا تعریف کردیم میسازد</summary>
/// <typeparam name="T">نوع مقادیری که قرار است داخل درخت ذخیره شوند</typeparam>
public class Tree<T>
{
    // گره ریشه
    private TreeNode<T> root;

    /// <summary>سازنده کلاس</summary>
    /// <param name="value">مقدار گره اول که همان ریشه می‌شود</param>
    public Tree(T value)
    {
        if (value == null)
        {
            throw new ArgumentNullException(
                "Cannot insert null value!");
        }

        this.root = new TreeNode<T>(value);
    }

    /// <summary>سازنده دیگر برای کلاس درخت</summary>
    /// <param name="value">مقدار گره ریشه مثل سازنده اول</param>
    /// <param name="children">آرایه ای از گره‌ها که فرزند گره ریشه می‌شوند</param>
    public Tree(T value, params Tree<T>[] children)
        : this(value)
    {
        foreach (Tree<T> child in children)
        {
            this.root.AddChild(child.root);
        }
    }

    /// <summary>
    /// ریشه را بر میگرداند ، اگر ریشه ای نباشد نال بر میگرداند
    /// </summary>
    public TreeNode<T> Root
    {
        get
        {
            return this.root;
        }
    }

    /// <summary> پیمودن عرضی و نمایش درخت با الگوریتم دی اف اس</summary>
    /// <param name="root">ریشه (گره ابتدایی) درختی که قرار است پیمایش از آن شروع شود</param>
    /// <param name="spaces">یک کاراکتر جهت جداسازی مقادیر هر گره</param>
    private void PrintDFS(TreeNode<T> root, string spaces)
    {
        if (this.root == null)
        {
            return;
        }

        Console.WriteLine(spaces + root.Value);

        TreeNode<T> child = null;
        for (int i = 0; i < root.ChildrenCount; i++)
        {
            child = root.GetChild(i);
            PrintDFS(child, spaces + "   ");
        }
```

```
    }

    /// <summary>متد پیمایش درخت به صورت عمومی که تابع خصوصی که در بالا توضیح دادیم را صدا
می‌زند</summary>
    public void TraverseDFS()
    {
        this.PrintDFS(this.root, string.Empty);
    }
}

/// <summary>
/// کد استفاده از ساختار درخت
/// </summary>
public static class TreeExample
{
    static void Main()
    {
        // Create the tree from the sample
        Tree<int> tree =
            new Tree<int>(7,
                new Tree<int>(19,
                    new Tree<int>(1),
                    new Tree<int>(12),
                    new Tree<int>(31)),
                new Tree<int>(21),
                new Tree<int>(14,
                    new Tree<int>(23),
                    new Tree<int>(6))
            );

        // پیمایش درخت با الگوریتم دی اف اس یا عمقی
        tree.TraverseDFS();

        // خروجی
        // 7
        //       19
        //        1
        //        12
        //        31
        //       21
        //       14
        //        23
        //        6
    }
}
```

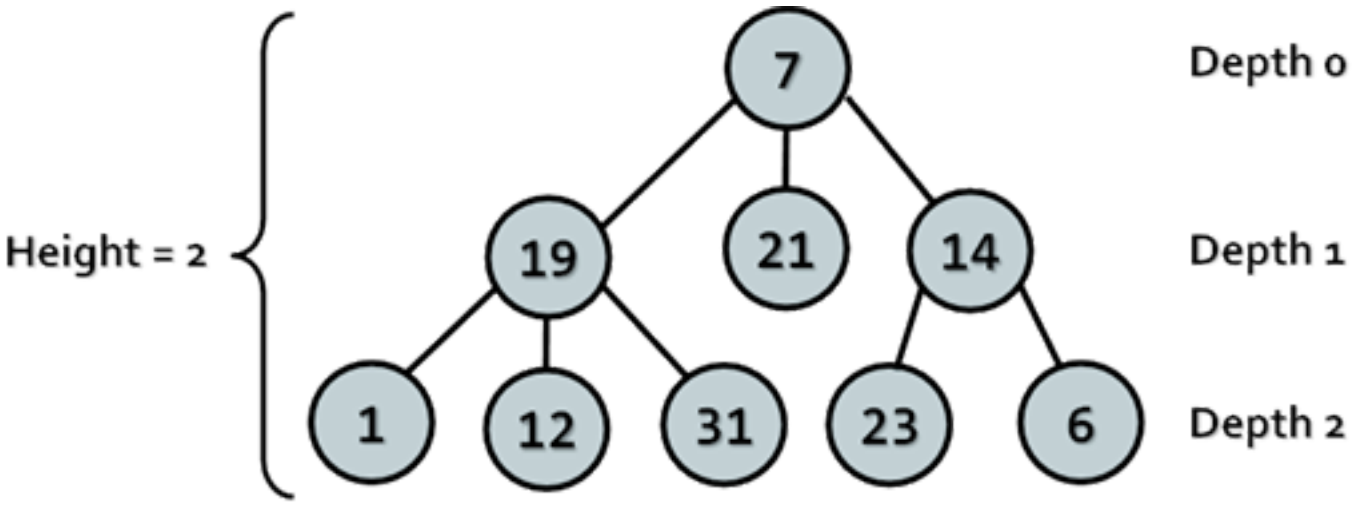
کلاس TreeNode وظیفه‌ی ساخت گره را بر عهده دارد و با هر شیءایی که از این کلاس می‌سازیم، یک گره ایجاد می‌کنیم که با خاصیت Children و متد AddChild آن می‌توانیم هر تعداد گره را که می‌خواهیم به فرزندی آن گره در آوریم که باز خود آن گره می‌تواند در خاصیت Children یک گره دیگر اضافه شود. به این ترتیب با ساخت هر گره و ایجاد رابطه از طریق خاصیت children هر گره درخت شکل می‌گیرد. سپس گره والد در ساختار کلاس درخت Tree قرار می‌گیرد و این کلاس شامل متدهایی است که می‌تواند روی درخت، عملیات پردازشی چون پیمایش درخت را انجام دهد.

پیمایش درخت به روش عمقی DFS (Depth First Search)

هدف از پیمایش درخت ملاقات یا بازبینی (تهیه لیستی از همه گره‌های یک درخت) تنها یکبار هر گره در درخت است. برای این کار الگوریتم‌های زیادی وجود دارند که ما در این مقاله تنها دو روش DFS و BFS را بررسی می‌کنیم.

روش DFS: هر گره‌ای که به تابع بالا بدهید، آن گره برای پیمایش، گره ریشه حساب خواهد شد و پیمایش از آن آغاز می‌گردد. در الگوریتم DFS روش پیمایش بدین گونه است که ما از گره ریشه آغاز کرده و گره ریشه را ملاقات می‌کنیم. سپس گره‌های فرزندش را به دست می‌آوریم و یکی از گره‌ها را انتخاب کرده و دوباره همین مورد را رویش انجام می‌دهیم تا نهایتا به یک برگ برسیم. وقتی که به برگی می‌رسیم یک مرحله به بالا برگشته و این کار را آنقدر تکرار می‌کنیم تا همه‌ی گره‌های آن ریشه یا درخت پیمایش شده باشند.

همین درخت را در نظر بگیرید:



پیمایش درخت را از گره 7 آغاز می‌کنیم و آن را به عنوان ریشه در نظر می‌گیریم. حتی می‌توانیم پیمایش را از گره مثلا 19 آغاز کنیم و آن را برای پیمایش ریشه در نظر بگیریم ولی ما از همان 7 پیمایش را آغاز می‌کنیم:

ابتدا گره 7 ملاقات شده و آن را می‌نویسیم. سپس فرزندانش را بررسی می‌کنیم که سه فرزند دارد. یکی از فرزندان مثل گره 19 را انتخاب کرده و آن را ملاقات می‌کنیم (با هر بار ملاقات آن را چاپ می‌کنیم) سپس فرزندان آن را بررسی می‌کنیم و یکی از گره‌ها را انتخاب می‌کنیم و ملاقاتش می‌کنیم؛ برای مثال گره 1. از آن جا که گره یک، برگ است و فرزندی ندارد یک مرحله به سمت بالا برمی‌گردیم و برگ‌های 12 و 31 را هم ملاقات می‌کنیم. حالا همه‌ی فرزندان گره 19 را بررسی کردیم، بر می‌گردیم یک مرحله به سمت بالا و گره 21 را ملاقات می‌کنیم و از آنجا که گره 21 برگ است و فرزندی ندارد به بالا باز می‌گردیم و بعد گره 14 و فرزندانش 23 و 6 هم بررسی می‌شوند. پس ترتیب چاپ ما اینگونه می‌شود:

```
7-19-1-12-31-21-14-23-6
```

پیمایش درخت به روش BFS (Breadth First Search)

در این روش (پیمایش سطحی) گره والد ملاقات شده و سپس همه گره‌های فرزندش ملاقات می‌شوند. بعد از آن یک گره انتخاب شده و همین پیمایش مجددا روی آن انجام می‌شود تا آن سطح کاملا پیمایش شده باشد. سپس به همین مرحله برگشته و فرزند بعدی را پیمایش می‌کنیم و الی آخر. نمونه‌ی این پیمایش روی درخت بالا به صورت زیر نمایش داده می‌شود:

```
7-19-21-14-1-12-31-23-6
```

اگر خوب دقت کنید می‌بینید که پیمایش سطحی است و هر سطح به ترتیب ملاقات می‌شود. به این الگوریتم، پیمایش موجی هم می‌گویند. دلیل آن هم این است که مثل سنگی می‌ماند که شما برای ایجاد موج روی دریاچه پرتاب می‌کنید.

برای این پیمایش از صف کمک گرفته می‌شود که مراحل زیر روی صف صورت می‌گیرد:

ریشه وارد صف Q می‌شود.
دو مرحله زیر مرتبا تکرار می‌شوند:

اولین گره صف به نام V را از Q در یافت می‌کنیم و آن را چاپ می‌کنیم.
فرزندان گره V را به صف اضافه می‌کنیم.

این نوع پیمایش، پیاده سازی راحتی دارد و همیشه نزدیک‌ترین گره‌ها به ریشه را می‌خواند و در هر مرحله گره‌هایی که می‌خواند از ریشه دورتر و دورتر می‌شوند.

نظرات خوانندگان

نویسنده: آقا ابراهیم
تاریخ: ۱۳۹۳/۱۲/۰۲ ۲۲:۴۶

سلام. ممنون بابت مطلب تون. کلا چند کاربرد سنگین و روز مره این ساختارهای درختی رو میخواستم.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۰۲ ۲۳:۳۱

همین نظر تو در تویی که الان ذیل مطلب شما ارسال شد یک ساختار درختی هست.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۲/۰۳ ۰:۵۶

کاربرد این موارد زیاد هست و در قسمت‌های بعدی مواردی رو هم نام خواهیم برد
نمونه‌های این مثال مثل دیکشنری‌ها و جست و جوها ، نقشه‌های شهر و مسیریابی و بازی ها
سیستم‌های برق کشی و لوله کشی و .. در بعضی کشورها روی سیستم‌ها نظارت میشه و با ایجاد یک نقص فنی روی نقشه به اونها نشون میده
یا حتی سیستم فایل یا سیستم‌های جست و جو گر
همین موتور گوگل یا حتی موتورهای جدید که با روش خاص از گراف برای جست و جوی‌های مرتبط استفاده میکنن و داده‌ها مرتبط به هم متصل میشن رو میشه نمونه از این موارد دونست
در خیلی از موارد هم شما دارین ازشون استفاده می‌کنین ولی شاید به خاطر قابلیت‌های فریم ورک‌های جدید و پیشرفت زبان‌ها چنان محسوس نبودن