知乎 首发于 数据结构及算法

数据结构---图

Mekeater
莫凯特，❤剪辑，吉他，写代码，计算机硕士

关注他

14 人赞同了该文章

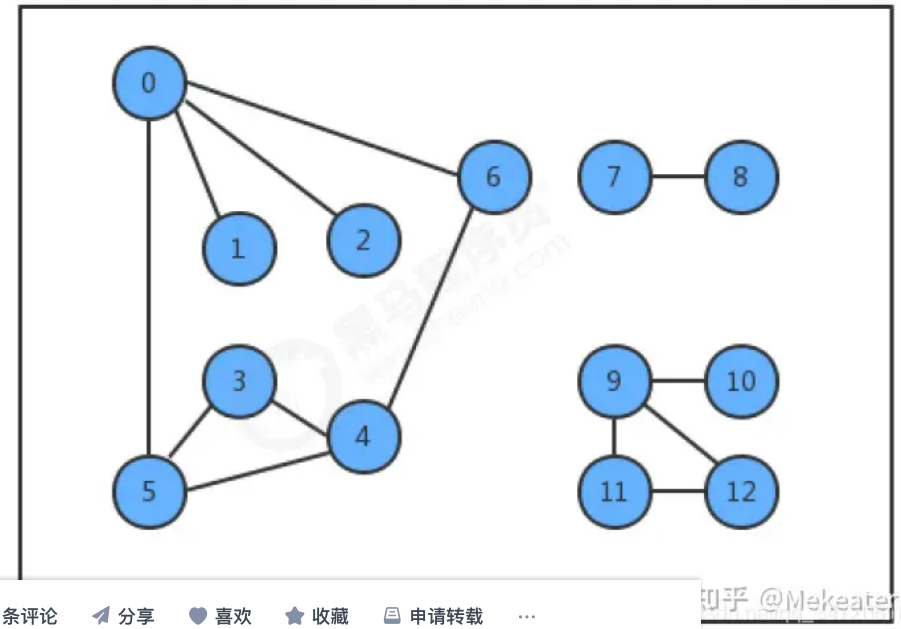
前言

数据结构分为逻辑结构和物理结构，从逻辑结构上，数据结构分为集合结构（数据元素之间没有关系）、线性结构（数据元素之间存在一对一的关系）、树形结构（数据元素之间存储一对多的关旭）和图形结构（数据元素之间存在多对多的关系）；从物理结构上，数据结构分为顺序结构（把数据元素放到地址连续的存储单元里面）和链式结构（把数据元素存放在任意的存储单元里面，这组存储单元可以是连续的也可以是不连续的）。本文讲解数据结构中最后一种结构------图形结构。

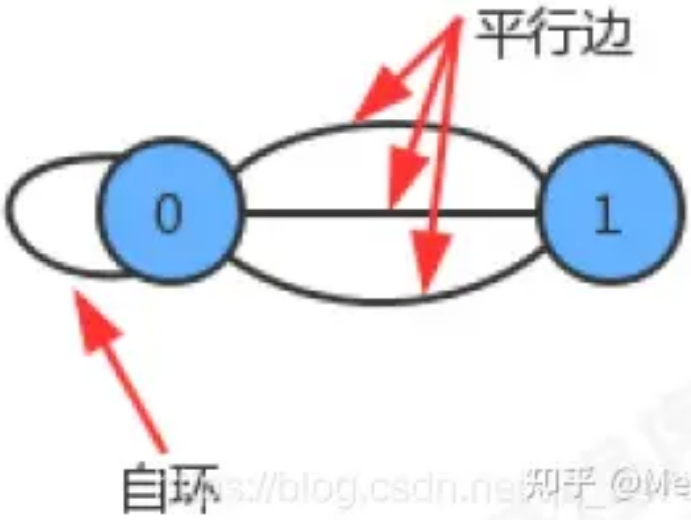
图的入门及无向图的实现

1. 图的相关概念

- 定义：图是由一组顶点和一组能够将两个顶点相连的边组成的



1. 自环：即一条连接一个顶点和其自身的边；
2. 平行边：连接同一对顶点的两条边；



- 图的分类

按照连接两个顶点的边的不同，可以把图分为以下两种：

无向图：边仅仅连接两个顶点，没有其他含义；

有向图：边不仅连接两个顶点，并且具有方向；

2. 图的相关术语

相邻顶点：当两个顶点通过一条边相连时，我们称这两个顶点是相邻的，并且称这条边依附于这两个顶点。

度：某个顶点的度就是依附于该顶点的边的个数

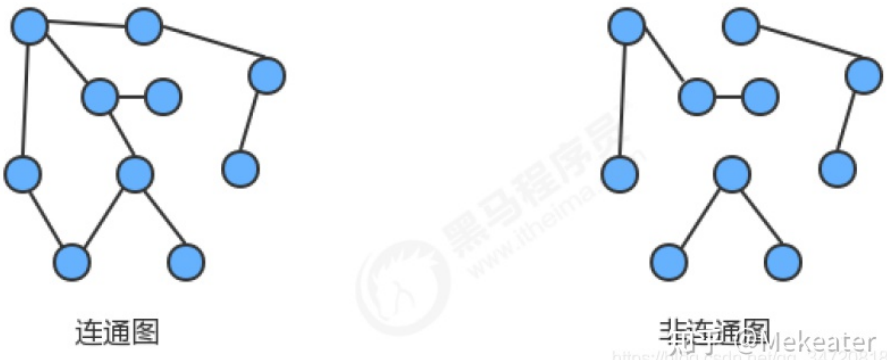
子图：是一幅图的所有边的子集(包含这些边依附的顶点)组成的图；

路径：是由边顺序连接的一系列的顶点组成

环：是一条至少含有一条边且终点和起点相同的路径

连通图：如果图中任意一个顶点都存在一条路径到达另外一个顶点，那么这幅图就称之为连通图。

连通子图：一个非连通图由若干连通的部分组成，每一个连通的部分都可以称为该图的连通子图



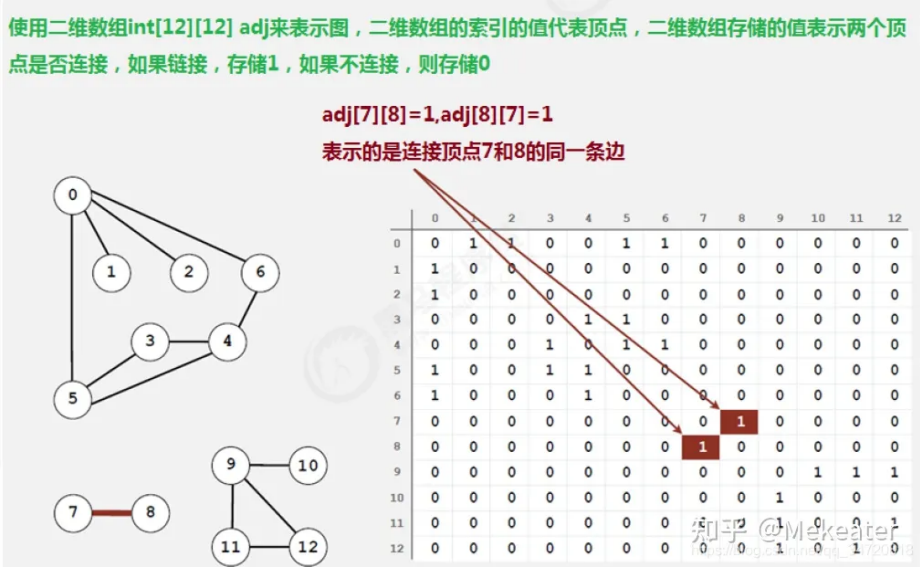
1. 图中所有的顶点；
2. 所有连接顶点的边；

常见图的存储结构有两种：邻接矩阵和邻接表

3.1 邻接矩阵

1.使用一个V*V的二维数组int[V][V] adj,把索引的值看做是顶点；

2.如果顶点v和顶点w相连，我们只需要将adj[v][w]和adj[w][v]的值设置为1,否则设置为0即可。

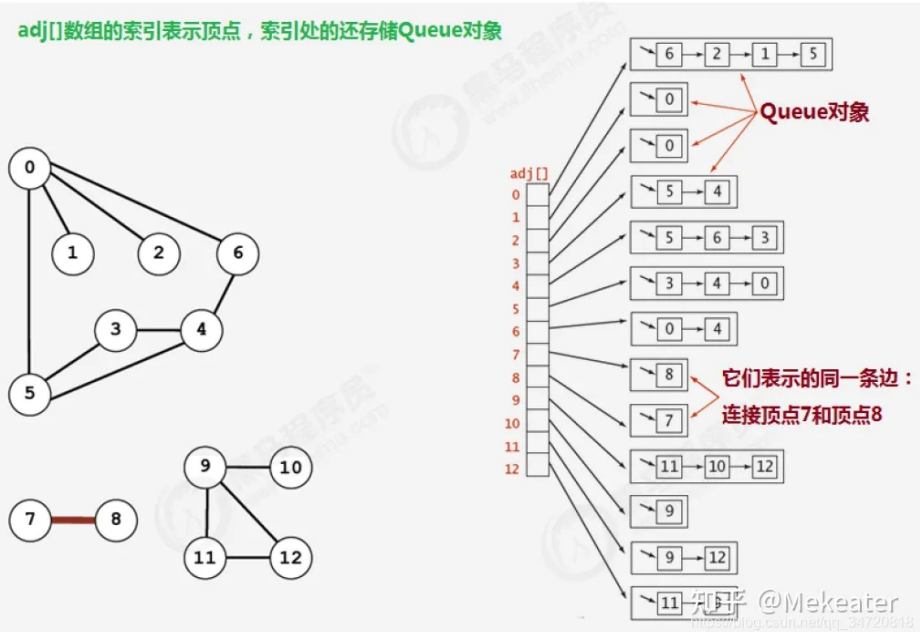


很明显，邻接矩阵这种存储方式的空间复杂度是V^2的，如果我们处理的问题规模比较大的话，内存空间极有可能不够用。

3.2 邻接表

1.使用一个大小为V的数组 Queue[V] adj，把索引看做是顶点；

2.每个索引处adj[v]存储了一个队列，该队列中存储的是所有与该顶点相邻的其他顶点。



很明显，邻接表的空间并不是是线性级别的，所以后面我们一直采用邻接表这种存储形式来表示

图的表示，只需要表示清楚图中顶点，及与依赖于该顶点的边即可。由邻接表的形式来表示图，可以由Queue[]数组的索引表示顶点，而每个数组的值，即队列表示依赖于顶点的边的另一个顶点。

```
// 无向图 数据结构（邻接表的思想进行实现）
public class Graph {
    private final int V;//记录顶点数量
    private int E;//记录边数量
    private Queue<Integer>[] adj; //每个顶点的邻接表

    public Graph(int v) {
        //初始化顶点数量
        V = v;
        //初始化边的数量
        E=0;
        //初始每个顶点的邻接表
        adj=new Queue[V];
        for (int i = 0; i < adj.length; i++) {
            adj[i]=new Queue<Integer>();
        }
    }
    //获取图中顶点的数量
    public int V()
    {
        return V;
    }
    //获取图中边的数量
    public int E()
    {
        return E;
    }
    //向图中添加一条边 v-w
    public void addEdge(int v,int w)
    {
        //1.将w添加到顶点v的邻接表中
        adj[v].enqueue(w);
        //2.将v添加到顶点w的邻接表中
        adj[w].enqueue(v);
        //3.边的个数+1
        E++;
    }
    //获取和顶点v相邻的所有顶点
    public Queue<Integer> adj(int v)
    {
        return adj[v];
    }
}
```

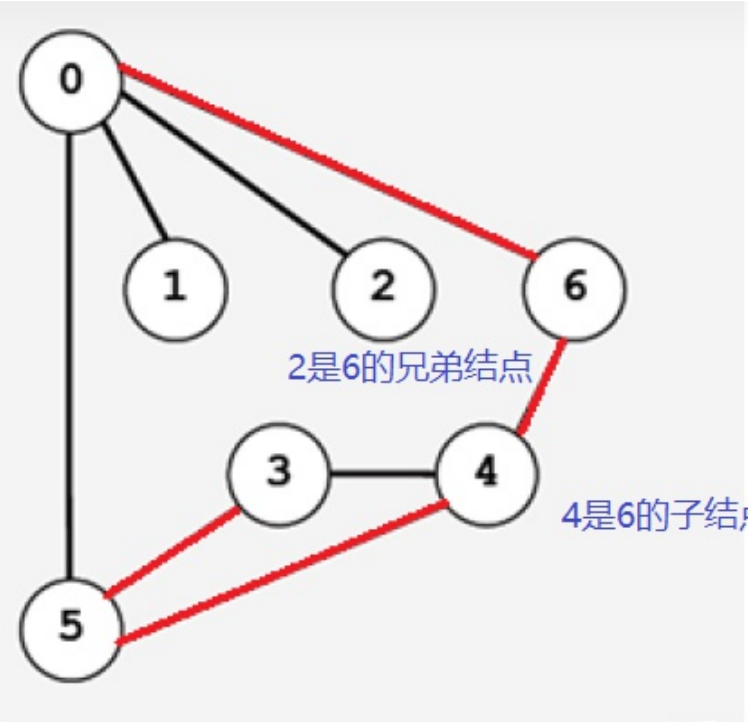
图的搜索算法

在很多情况下，我们需要遍历图，得到图的一些性质，例如，找出图中与指定的顶点相连的所有顶点，或者判定某个顶点与指定顶点是否相通，是非常常见的需求。本节讲解图的深度优先搜索和广度优先搜索两种搜索的思想以及代码实现。

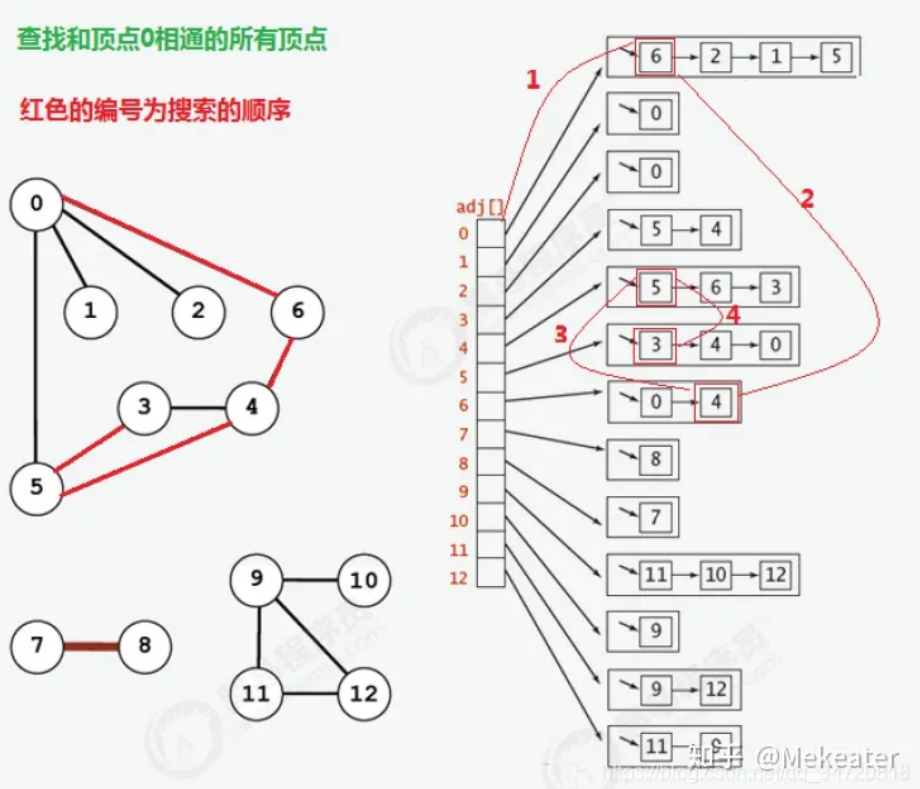
1. 深度优先搜索

1.1 搜索思路

在搜索时如果遇到一个结点既有子结点，又有兄弟结点，那么先找子结点，再找兄弟结点。下图以结点6为例，说明什么是子结点，什么是兄弟结点。**简单来说就是一条路走到底，直到这条路走不下去了，再到这条路的起点尝试走其它路。**



下图是一个顶点的深度优先搜索顺序



1.2 代码实现

```
public class DepthFirstSearch {
    private boolean[] marked;//索引代表顶点，值表示当前顶点是否已经被搜索
    private int count;//记录有多少个顶点与s顶点相通

    //构造深度优先搜索对象，使用深度优先搜索找出G图中s顶点的所有相通顶点
```

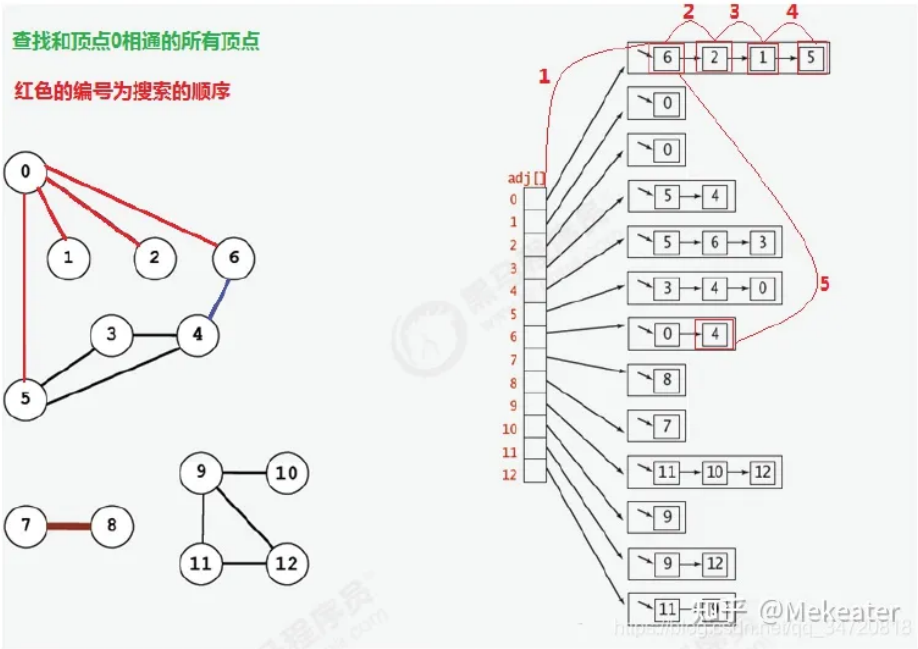
```
        //2.初始化G图中多少个顶点与顶点s相通，初始情况下count为0
        count=0;
        //3.调用深度优先搜索方法，更新count的值和marked的值
        dfs(G,s);
    }
    //使用深度优先搜索找出G图中v顶点的所有相通顶点
    private void dfs(Graph G, int v)
    {
        //1.将当前顶点标记为已搜索
        marked[v]=true;
        //2.搜索当前顶点的邻接表，若邻接表中的顶点不为0，如果邻接表中的顶点未被搜索，则递归
        for (Integer w : G.adj(v)) {
            if (!marked[w])
                dfs(G,w);
        }
        //3.每次深度优先搜索后，count+1
        count++;
    }
    //判断w顶点与s顶点是否相通
    public boolean marked(int w)
    {
        //如果相通，则一定被搜索过
        return marked[w];
    }
    //获取与顶点s相通的所有顶点的总数
    public int count()
    {
        return count;
    }
```

2. 广度优先搜索

2.1 搜索思路

如果遇到一个结点既有子结点，又有兄弟结点，那么先找兄弟结点，然后找子结点。**简单来说就是先把所有可能的路都走一步看看。**

下图是一个顶点的广度优先搜索顺序



```
    private int count;//与顶点相通的所有顶点个数
    private Queue<Integer> waitSearch;//辅助队列，用来存储待搜索的顶点

    //构造广度优先搜索对象，使用广度优先搜索找出G图中s顶点的所有相邻顶点
    public BreadthFirstSearch(Graph G,int s)
    {
        this.marked=new boolean[G.V()];
        this.count=0;
        this.waitSearch=new Queue<Integer>();
        BFS(G,s);
    }

    //广度优先搜索图G中与顶点V相通的所有顶点
    public void BFS(Graph G,int V)
    {
        //1.将顶点V标记为已搜索
        marked[V]=true;
        //2.将顶点V入队
        waitSearch.enqueue(V);
        //3.通过循环，如果队列不为空，则从队列中弹出一个待搜索的顶点，然后递归调用待搜索顶点
        while (!waitSearch.isempty())
        {
            Integer wait = waitSearch.dequeue();
            for (Integer w : G.adj(wait)) {
                if(!marked(w))
                    BFS(G,w);
            }
        }

        //4.每次搜素都会找到一个相通的顶点，因此将相通的顶点+1
        count++;
    }

    //获取图G中与顶点V相通的所有顶点的个数
    public int count()
    {
        return count;
    }

    //判断w顶点与s顶点是否相通
    public boolean marked(int w)
    {
        return marked[w];
    }
}
```

有没有发现，广度优先搜索其实和树的层序遍历思路一样，而且实现都需要一个辅助队列。

上一篇文章

Mekeater：数据结构---树（彻底理解递归算法）
27 赞同 · 1 评论 文章

后记

之所以在每日算法题中间穿插【树和图】这两篇数据结构，是因为后面我们将做和图相关的题目，因此要有一个基本了解，好好琢磨这两篇文章，相信对你的递归理解，及对数据结构的深入理解都会有质的提升！

知乎 首发于 数据结构及算法

数据结构_Mekeater的博客-CSDN博客
blog.csdn.net/qq_34720818/category_861...

编辑于 2021-12-14 09:53

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

数据结构 搜索算法 算法与数据结构

发布一条带图评论吧

1 条评论 默认 最新

哎哈哈 ...
看完您的图和树后，理解得更深入了！
08-18 回复 喜欢

文章被以下专栏收录

数据结构及算法
一起刷算法，一起感受算法之美!

推荐阅读

数据结构——图

翻斗花园胡图图

图解数据结构（开篇）

作者：天行参加了 lucifer 的
学算法活动，不知不觉中已经
有余。从盲目地做到有目的、
路地去做。 在 lucifer 的 91 i
中，从基础到进阶到专题，在
月中，经历了基础篇的...

lucif... 发表于力