# 定义

图（Graph）是由顶点的有穷非空集合和顶点之间变的集合组成，通常表示为：G(V,E)，其中，G表示一个图，V是图G中顶点的集合，E是图G中边的集合。

注：图形结构的数据元素是多对多的关系。

图与线性表和非线性表（树）的一些区别：

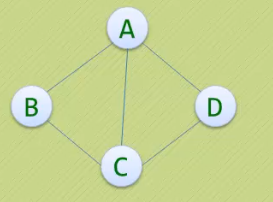
1. 线性表中我们把数据元素叫元素，树中叫结点，在图中数据元素我们称之为顶点（vertex）；
2. 线性表可以没有数据元素，称为空表，树中可以没有结点，称为空树，而图结构强调顶点集合V要有穷非空；
3. 在线性表中，相邻的数据元素之间具有线性关系，树结构中，相邻两层的节点具有层次关系，而图结构中，任意两点之间都可能有关系，顶点之间的逻辑关系用边来表示，边集可以是空的。

## 顶点

## 边

### 无向边

无向边：若顶点Vi到Vj之间的边没有方向，则称这条边为无向边（Edge），用无序偶(Vi,Vj)来表示。



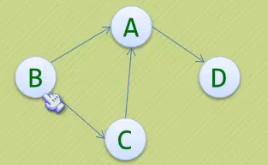
上图G1是一个无向图，G1={V1,E1}，其中：

V1={A,B,C,D}

E1={(A,B),(B,C),(C,D),(D,A),(A,C)}

### 有向边

有向边：若从顶点Vi到Vj的边有方向，则称这条边为有向边，也称为弧（Arc），用有序偶<Vi,Vj>来表示，Vi称为弧尾，Vj称为弧头。



上图G2是一个无向图，G2={V2,E2}，其中：

V2={A,B,C,D}

E2={<B,A>,<B,C>,<C,A>,<A,D>}

### 权

有些图的边或弧带有与它相关的数字，这种与图的边或弧相关的数叫做权（weight），带权的图通常称为网（network）。

## 顶点与边

### 邻接点

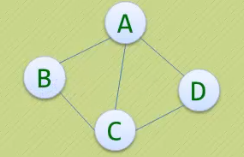
对于无向图G=(V,E)，如果边(V1,V2)∈E，则称顶点V1和V2互为邻接点（Adjacent），即V1和V2相邻接。

边(V1,V2)依附(incident)于顶点V1和V2，或者说边(V1,V2)与顶点V1和V2相关联。

对于有向图G=(V,E)，如果有<V1,V2>∈E，则称顶点V1邻接到顶点V2，顶点V2邻接自顶点V1。

### 度

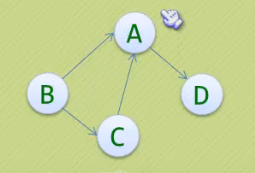
顶点V的度(Degree)是和V相关联的边的数目，记为TD(V)，如下，顶点A与B互为邻接点，边(A,B)依附于顶点A与B上，顶点A的度为3。



#### 入度

#### 出度

以顶点V为头的弧的数目称为V的入度（InDegree），记为ID(V)，以V为尾的弧的数目称为V的出度（OutDegree），记为OD(V)，因此顶点V的度为TD(V)=ID(V)+OD(V)。



顶点A的入度为2，出度为1，所以顶点A的度是3。

### 路径

无向图G=(V,E)中从顶点V1到顶点V2的路径（path）。

路径的长度是路径上的边或弧的数目。

## 子图

假如有两个图G1=(V1,E1)和G2(V2,E2)，如果V2∈V1，E2∈E1，则称G2位G1的子图（subgraph）。

# 分类

## 简单图

在图结构中，若不存在顶点到其本身的边，且同一条边不重复出现，则称这样的图为简单图。

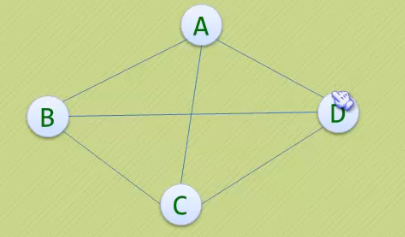
## 无向图

### 连通图

在无向图G中，如果从顶点V1到顶点V2有路径，则称V1和V2是连通的，如果对于图中任意两个顶点Vi和Vj都是连通的，则称G是连通图（ConnectedGraph）。

### 完全图

在无向图中，如果任意两个顶点之间都存在边，则称该图为无向完全图。含有n个顶点的无向完全图有n\*(n-1)/2条边。



### 生成树

所谓的一个连通图的生成树是一个极小的连通子图，它含有图中全部的n个顶点，但只有足以构成一棵树的n-1条边。

## 有向图

### 强连通图

在有向图G中，如果对于每一对Vi到Vj都存在路径，则称G是强连接图。

有向图中的极大强连通子图称为有向图的强连通分量。

### 完全图

在有向图中，如果任意两个顶点之间都存在方向且互为相反的两条弧，则称该图为有向完全图。含有n个顶点的有向完全图有n\*(n-1)条边。

## 稀疏图

这里的稀疏和稠密是模糊的概念，都是相对而言的，通常认为边或弧数小于n\*logn(n是顶点的个数)的图称为稀疏图，反之称为稠密图。

## 稠密图

# 存储

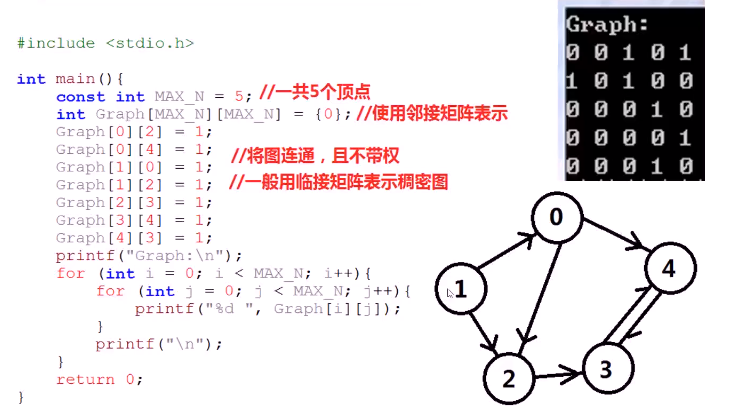
对于线性表而言，是一对一的关系，所以用数组或链表均可简单存放。树结构是一对多的关系，所以我们要将数组和链表的特性组合在一起才能更好地存放。

图是多对多的关系，图上的任何一个顶点都可以被看作是第一个顶点，任一顶点的邻接点之间也不存在次序关系。

因为任意两个顶点之间都可能存在联系，因此无法以数据元素在内存中的物理位置来表示元素之间的关系（内存物理位置是线性的，图的元素关系是平面的）。如果用多重链表来描述是可以做到的，但是纯粹使用多重链表导致的浪费非常大（如果每个顶点的度数相差太大，就会造成巨大的浪费）。

## 邻接矩阵

邻接矩阵一般应用于带权图中，比如最小生成树，最短路径（搜索的话一般用邻接表）。

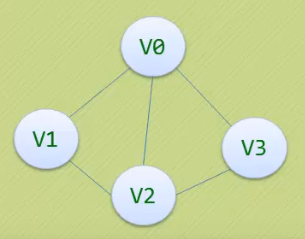


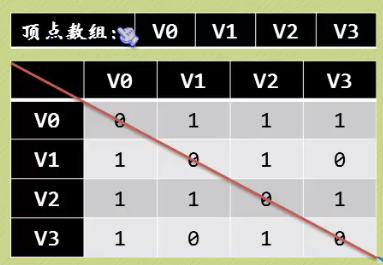
### 无向图

考虑到图是由顶点和边或弧两部分组成，合在一起比较困难，那就很自然考虑到分为两个结构分别存储。

顶点因为不区分大小、主次，所以用一个一维数组来存储是不错的选择。而边或弧由于是顶点与顶点之间的关系，一维数组无法满足需求，我们可以考虑使用一个二维数组来存储。

图的邻接矩阵（Adjacency Matrix）存储方式是用两个数组来表示图。一个一维数组存储图中顶点信息，一个二维数组（称为邻接矩阵）存储图中的边或弧信息。



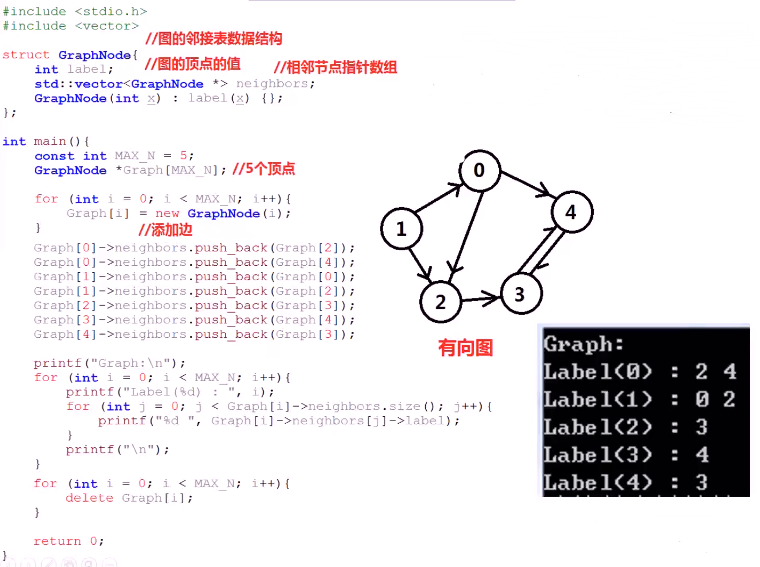


### 有向图

## 邻接表

对于边树相对顶点较少的图，邻接矩阵的结构存在对存储空间极大的浪费。

我们可以考虑把数组与链表结合一起存储，这种方式在图结构也适用，称为邻接表（AdjacencyList）。搜索通常使用邻接表。



### 无向图

### 有向图

## 十字链表

邻接表在对有向图的处理上，有时候需要再建立一个逆连接表。十字链表（Orthogonal List）把邻接表和逆邻接表结合起来。

## 邻接多重表

## 边集数组

# 遍历

深度优先遍历简称DFS（Depth First Search），广度优先遍历简称BFS（Breadth First Search），它们是遍历图当中所有顶点的两种方式。

## 深度优先

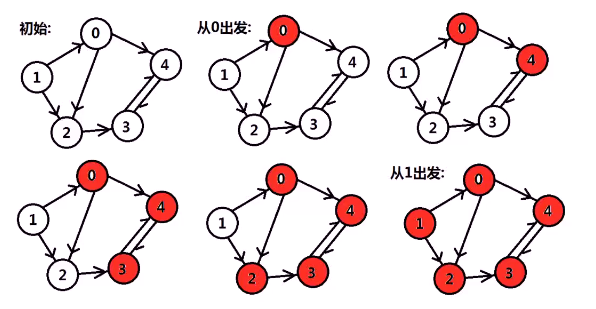
### 概述

先深入探索，走到头再回退寻找其他出路的遍历方式，就叫做深度优先遍历（DFS）。

注：一般所谓“暴力枚举”搜索都是指DFS

### 分析

从图中某个顶点v出发，首先访问该顶点，然后依次从它的各个未被访问的邻接点触发深度优先搜索遍历图，直至图中所有和v有路径相通且未被访问的顶点都被访问到。若此时尚有其他顶点未被访问到，则另选一个未被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。



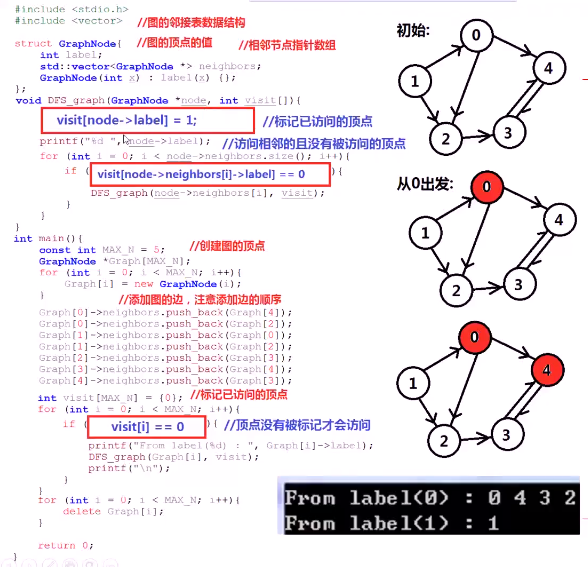
注：二叉树的前序、中序、后序遍历，本质上也可以认为深度优先遍历。

### 实现

实现深度优先遍历的关键在于**“回溯”**，实现广度优先遍历的关键在于**“重放”**。

回溯就是自后向前，追溯曾经走过的路径。

**要想实现回溯，可以利用栈的先入后出特性，也可以采用递归的方式**（因为递归本身就是基于方法调用栈来实现）。



### 应用

#### 最小平方划分

## 广度优先

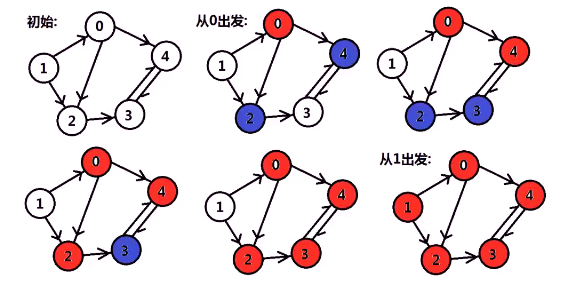
### 概述

一层一层由内而外的遍历方式，就叫做广度优先遍历（BFS）。

注：二叉树的层次遍历本质上也可以认为是广度优先遍历。

### 分析

从图中某个顶点v出发，在访问了v之后依次访问v的各个未曾访问过的邻接点，然后分别从这些邻接点出发依次访问它们的邻接点，并使得“先被访问的顶点的邻接点先于后被访问的顶点的邻接点被访问”，直至图中所有已被访问的顶点的邻接点都被访问到。如果此时图中尚有顶点未被访问，则需要另选一个未曾被访问过的顶点作为新的起始点，重复上述过程，直至图中所有顶点都被访问到为止。



给定某起点a，将a放入缓冲区，开始搜索。

过程：假定某时刻缓冲区内结点为abc，则访问结点a的邻接点a1a2ax，为下一次访问做准备：

辅助数据结构：队列

先进先出

从队尾入伍，从队首出队

只有队首元素可见

BSF分析的两个要素：

1. 结点判重：

如果在拓展中发现某节点在前期已经访问过，则本次不再访问该节点。

显然，第一次访问到该节点时，是访问次数最少的：最小、最短。

1. 路径记录：

一个结点可能拓展出多个结点：多后继a1a2…ax

但是任意一个结点最多只可能有1个前驱（起始结点没有前驱）：单前驱

用结点数目等长的数组pre[0…N-1]

pre[i]=j：第i个结点的前一个结点j

注：再次用到“存索引，不存数据本身”的思路。

### 实现

BFS算法框架：

1. 辅助数据结构

队列q；

结点是第几次被访问到的d[0…N-1]：简称步数；

结点的前驱pre[0…N-1]；

1. 算法描述：

2.1 起点start入队q：

记录步数d[start]=0；

记录start的前驱pre[start]=-1；

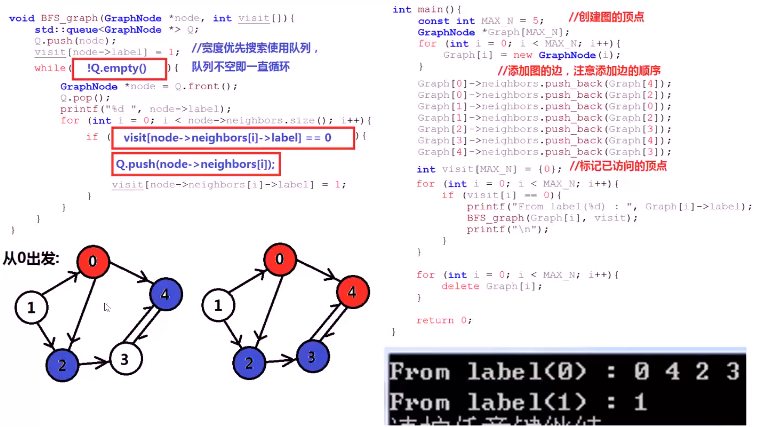
2.2 如果队列q非空，则队首结点x出队，尝试扩展x

找出x的邻接点集合{y|(x,y)∈E}

对每个新扩展结点y判重，如果y是新结点，则入队q；

同时，记录步数d[y]=d[x]+1，前驱pre[y]=x

把遍历过的顶点按照之前的遍历顺序重新回顾，就叫做重放。同样的，要实现重放也需要额外的存储空间，可以利用队列的先入先出特性来实现。



/\*\*

\* 图的顶点

\*/

private static class Vertex

{

int data;

Vertex(int data) {

this.data = data;

}

}

### 应用

#### 单词匹配

#### 周围区域问题

# 应用

## 图的搜索

## 最小生成树

### Prim普利姆算法

### Krusal克鲁斯卡尔算法

## 并查集

## 最短路径

### 单源图/迪杰斯特拉(Dijkstra)算法

### 任意两点/弗洛伊德(Floyd)算法

### 有负边(Bellman-ford)

## 拓扑排序

## 关键路径

# 其他案例

## 课程安排

## 岛屿数量

## 词语阶梯

## 火柴棍摆正方形

## 收集雨水