

图

1.图的概念

定义：图(graph)是一种非线性数据结构，形式化描述为 $\text{graph}(V,R)$

齐河县城區公交运行线路图

7路线（城区） 单程 24.5 公里 全程站点：22 个 阶梯票价：1-2 元
发车时间：07:00—17:00



搜狐号@德州顺瑞生活圈

公交车线路图是一种典型的图



网络(图)

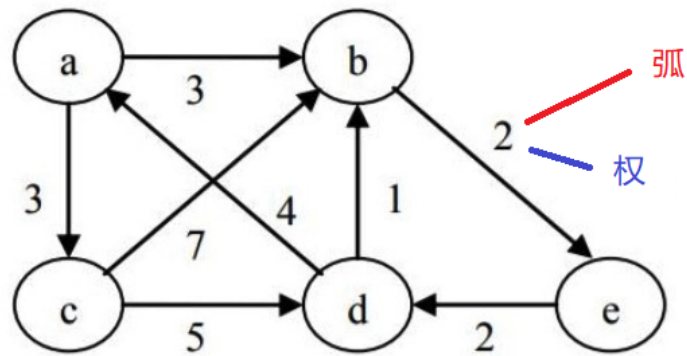
其中, $V=\{v_i | v_i \text{属于datatype}, i=0,1,2 \dots n-1\}$ 是图中元素的集合。 v_i 称为顶点(vertex), 当 n 为 0 的时候, V 为空集。

$R=\{<v_i, v_j> | v_i, v_j \text{属于} V \text{且} p(v_i, v_j) \text{存在}\}$ 是图中顶点之间的关系集。 $p(v_i, v_j)$ 为顶点 v_i 和 v_j 之间是否存在路径的判定条件, 即 v_i 和 v_j 之间有路径存在, 则关系 (v_i, v_j) 属于 R 。

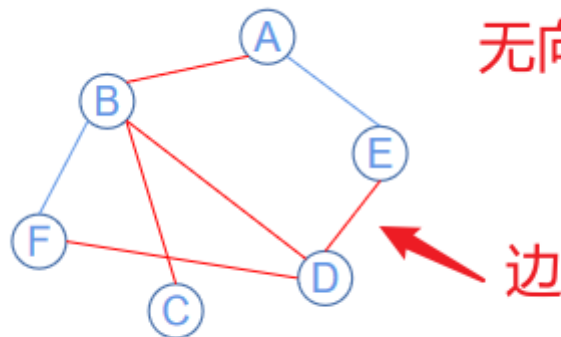
2.图的分类

- 有向图 弧
- 无向图 边
- 网

有向图

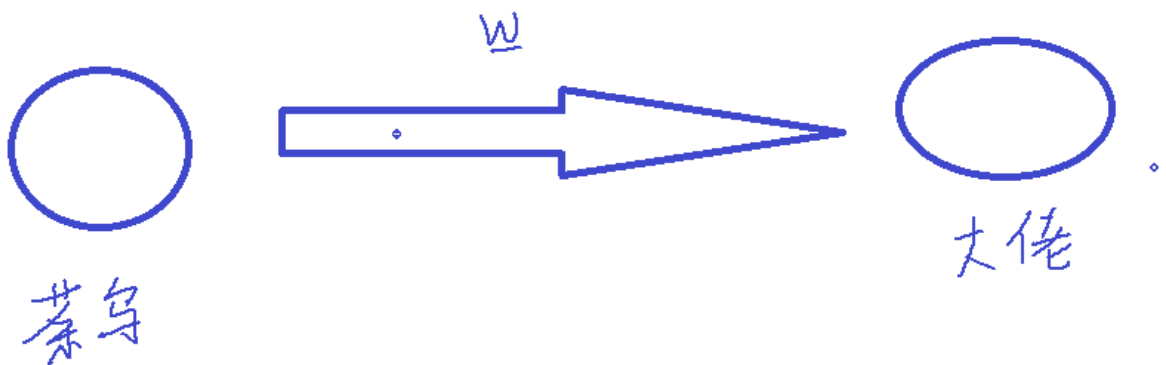


无向图



若在图的关系 $\langle v_i, v_j \rangle$ 或 $\langle v_j, v_i \rangle$ 附加一个值 w 称 w 为弧或边上的权。带权的图称为网。权 w 的具体含义视图在不同的领域的应用而定。

比如公共线路图，顶点表示站，权 w 可以视为两个站台之间的距离或收费。



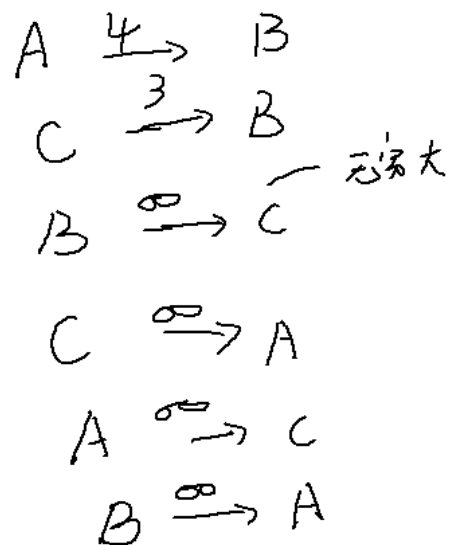
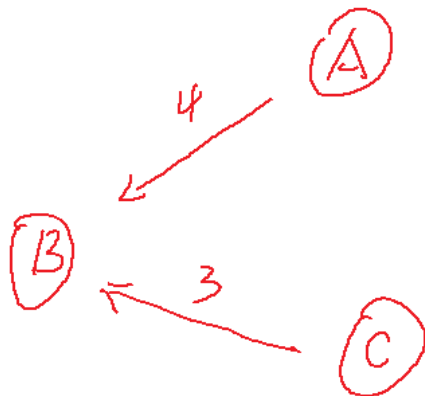
顶点的度

顶点的边或弧的条数

连通图和非连通图

路径

从某个顶点到其他顶点是否存在路径



3.图的存储结构

- 数组表示法(邻接矩阵)
- 邻接表-->逆邻接表
- 十字链表(邻接表+逆邻接表)
- 邻接多重表

a.数组表示法

"邻接矩阵"

$G = \{V, R\}$

V是顶点集合

R是关系集合

可以用两个数组来存储图G

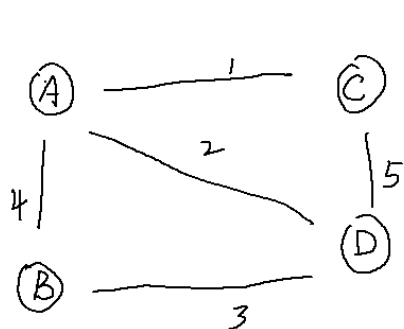
一个一维数组V存储图G顶点集合 V_i

一个二维数组A存储图G顶点之间的关系R

第i行, 表示以顶点 $v[i]$ 为起点的所有的边的权

第j列, 表示以顶点 $v[j]$ 为终点

$A[i][j] \implies \langle v_i, v_j \rangle \quad w$



$V = \{ A, B, C, D \}$
 $v_0 \quad v_1 \quad v_2 \quad v_3$

无向图的矩阵

关系 R

A $\xrightarrow{1}$ C
A $\xrightarrow{4}$ B
A $\xrightarrow{2}$ D
B $\xrightarrow{4}$ A
B $\xrightarrow{3}$ D

\Rightarrow

	v_0	v_1	v_2	v_3
v_0	∞	4	1	2
v_1	4	∞	∞	3
v_2	1	∞	∞	5
v_3	2	3	5	∞

$A[i][j]$
 $\langle v_i, v_j \rangle$

邻接矩阵

西安市公交线路票价表（网络版）

辛家庙公交枢纽站			211路		辛家庙公交枢纽站—沙井村北	
1.0	1.0	1.0	辛家庙西村		全程22.2公里，设31个站点	
1.0	1.0	1.0	井上村			
1.0	1.0	1.0	巨豪建材市场			
1.0	1.0	1.0	大明宫建材家居城			
1.0	1.0	1.0	明珠家居			
1.0	1.0	1.0	城北客运站			
1.0	1.0	1.0	中联家居			
1.0	1.0	1.0	朱宏桥东			
1.0	1.0	1.0	福迪汽贸			
1.0	1.0	1.0	朱宏路机电市场			
1.0	1.0	1.0	大白杨村			
1.0	1.0	1.0	未央宫街办			
1.5	1.0	1.0	李下壕村			
1.5	1.0	1.0	沣惠路			
1.5	1.5	1.0	安美居			
1.5	1.5	1.5	西开公司			
1.5	1.5	1.5	团结中路			
1.5	1.5	1.5	土门岗厦			
1.5	1.5	1.5	丰登路			
2.0	1.5	1.5	丰庆路西段			
2.0	1.5	1.5	锦西小区			
2.0	1.5	1.5	桃园路			
2.0	1.5	1.5	西大新区			
2.0	2.0	2.0	玫瑰大楼			
2.0	2.0	2.0	亚美大厦			
2.0	2.0	2.0	高新路			
2.0	2.0	2.0	高新一中			
2.0	2.0	2.0	博文路			
2.0	2.0	2.0	白沙路			
2.0	2.0	2.0	沙井村北			
网络版制作：Fivos			2011年9月1日更新		©2011 西安公交网(XBUS.CN)	

终点站 起点站	福州火车南站站	庐雷站	三角埕站	城门站	排下站	黄山站	葫芦阵站	白湖亭站	三叉街站	上藤站	达道站	茶亭站	南门兜站	东街口站	屏山站	树兜站	斗门站	福州火车站站	罗汉山站	秀山站	象峰站
福州火车南站站	2	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	5	5	6	6
庐雷站	2	2	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	5	5	6
三角埕站	2	2	2	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	5	5
城门站	2	2	2	2	2	2	2	2	3	3	3	4	4	4	4	4	5	5	5	5	5
排下站	2	2	2	2	2	2	2	2	2	3	3	3	4	4	4	4	4	5	5	5	5
黄山站	3	2	2	2	2	2	2	2	2	2	3	3	3	4	4	4	4	4	5	5	5
葫芦阵站	3	3	2	2	2	2	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5
白湖亭站	3	3	3	2	2	2	2	2	2	2	2	3	3	3	3	4	4	4	4	5	5
三叉街站	3	3	3	3	2	2	2	2	2	2	2	2	3	3	3	3	3	4	4	4	4
上藤站	4	3	3	3	3	2	2	2	2	2	2	2	2	3	3	3	3	3	3	4	4
达道站	4	4	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	3	3	3	4
茶亭站	4	4	4	4	3	3	3	3	2	2	2	2	2	2	2	2	2	3	3	3	4
南门兜站	4	4	4	4	4	3	3	3	3	2	2	2	2	2	2	2	2	3	3	3	3
东街口站	5	4	4	4	4	4	3	3	3	2	2	2	2	2	2	2	2	2	3	3	3
屏山站	5	5	4	4	4	4	3	3	3	2	2	2	2	2	2	2	2	2	2	3	3
树兜站	5	5	5	4	4	4	4	4	3	3	3	2	2	2	2	2	2	2	2	2	3
斗门站	5	5	5	5	4	4	4	4	3	3	3	2	2	2	2	2	2	2	2	2	3
福州火车站站	5	5	5	5	5	4	4	4	4	3	3	3	2	2	2	2	2	2	2	2	2
罗汉山站	5	5	5	5	5	5	5	4	4	4	3	3	3	3	2	2	2	2	2	2	2
秀山站	6	5	5	5	5	5	5	5	4	4	4	3	3	3	3	2	2	2	2	2	2
象峰站	6	6	5	5	5	5	5	5	5	4	4	4	3	3	3	3	3	2	2	2	2

代码示例

```
typedef char vtype; //图中顶点元素的类型
```

```
typedef int Adjtype; //边上权的类型
```

```
#define MAXN 100 //图中顶点的最大个数
```

```
typedef struct //描述图 "邻接矩阵"
{
```



```
vtype V[MAXN]; //一维数组用来保存图中的顶点
Adjtype[MAXN][MAXN]; //二维数组，“邻接矩阵”，
存储“边”
```

```
int vexnum; //图中有效顶点的个数
int arcnum; //图中边的条数
//....
}Graph;
```

优势：简单直观，可以快速查到一个顶点到另一个顶点之间的关系。

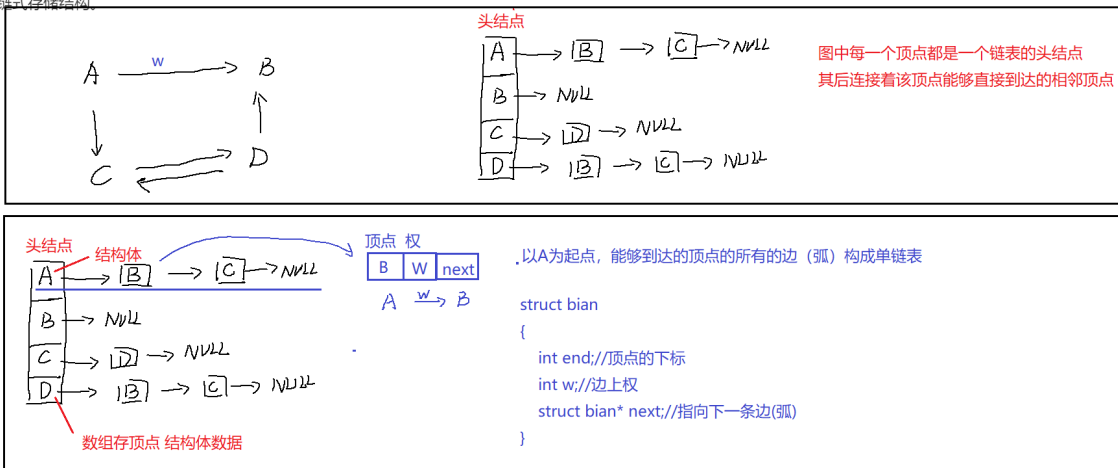
劣势：占用太多的空间。如果有一个图有1000个顶点，其中只有10个顶点之间有关系(稀疏图)，不得不创建一个1000*1000的二维数组，这实在是太浪费了。

b.邻接表

链式结构来存储一个表。

所谓的邻接表(Adjacency Lists),它是将图中每一个顶点V和由V发出的弧或边构成**单链表**。邻接表是图的一种链式存储结构。

所谓的邻接表(Adjacency Lists),它是将图中每一个顶点V和由V发出的弧或边构成**单链表**。邻接表是图的一种链式存储结构。



示例代码

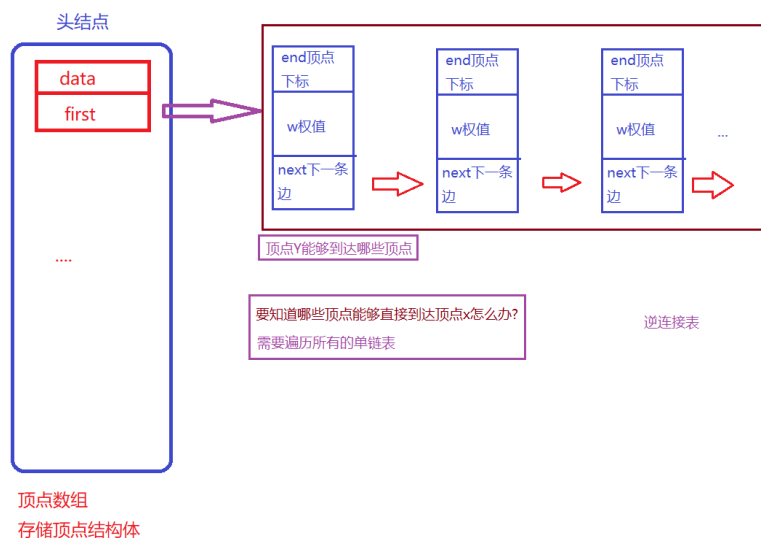
```
typedef char vtype; //图中顶点元素的类型
typedef int Adjtype; //边上权的类型

typedef struct bian
{
    int end_index; //边的终点的下标
    Adjtype w; //边的权值
    struct bian* next; //指向下一条边
}Bian;

//所有的struct bian链起来的单链表，表示由同一个起
//点发出的所有的边
//顶点元素的结构体
typedef struct vertex
{
    vtype data; //顶点的数据域
    Bian* first; //指向该顶点发出的第一条边
}Vertex;
```

```
typedef struct vertex
{
    vtype data; //顶点的数据域
    Bian* first; //指向该顶点发出的第一条边
}Vertex;
```

顶点的数据类型



课后拓展

用邻接表来存储一个图，把创建图的代码实现一下。

4.图的遍历

图的遍历是树的遍历的推广，是按照某种规则(次序)访问图中各定点一次且仅有一次的操作，也是将网状结构按照某种规则线性化的过程。

对图的遍历通常有"深度优先搜索"和"广度优先搜索"的方法。

二者是人工智能(AI)的一个基础。

a.深度优先搜索(DFS)

初始化的时候，图中的各定点均未被访问。以图中某定点(设 v_0)出发，访问 v_0 ，然后再搜索 v_0 的一个邻接点 v_i (若 v_i 未被访问)，则访问 v_i ，再搜索 v_i 的下一个邻接点 v_j (DFS)...若某结点的邻接点全部访问完毕，再回溯(backtracking)到它的上一个顶点，然后再从此顶点又按照深度优先搜索的方法搜索下一个邻接点...直到能访问的点全部访问完毕为止。

```
/*
```

```
    int visited[MAXN] = {0}; //每个顶点有没有被访问
```

```
    0 未被访问
```

```
    1 已被访问
```

```
    DFS(G, v0): 从图G的顶点v0出发，按照DFS的规则去访问图中能访问的所有的顶点(邻接点)
```

```
        1) 要访问v0 v0已被访问
```

```

    printf v0
    visted[0] = 1
2)找到v0的下一个邻接点vi(若vi未被访问),则访问
vi
    if(visted[i] == 0)
        DFS(G,vi)
    ...
3)找到v0的下一个邻接点vj(若vj未被访问),则访问
vi
    if(visted[j] == 0)
        DFS(G,vj)
    ...
直到v0的所有的邻接点都访问完毕为止

再从下一个顶点出发开始访问
DFS(G,v1)
*/

```

b.广度优先搜索

广度优先搜索类似于树的按层次遍历。初始化，图各定点均未被访问。从图中某个定点(设为v0)出发，访问v0，并且访问v0的各个邻接点(BFS)，然后，分别从这些被访问过的顶点出发，仍然按照广度优先搜索的策略来访问其他顶点...直到能访问的都访问完毕为止。

算法的实现：类似树的按层次遍历

利用队列

```

int visited[MAXN] = {0}; //每个顶点有没有被访问
0 未被访问

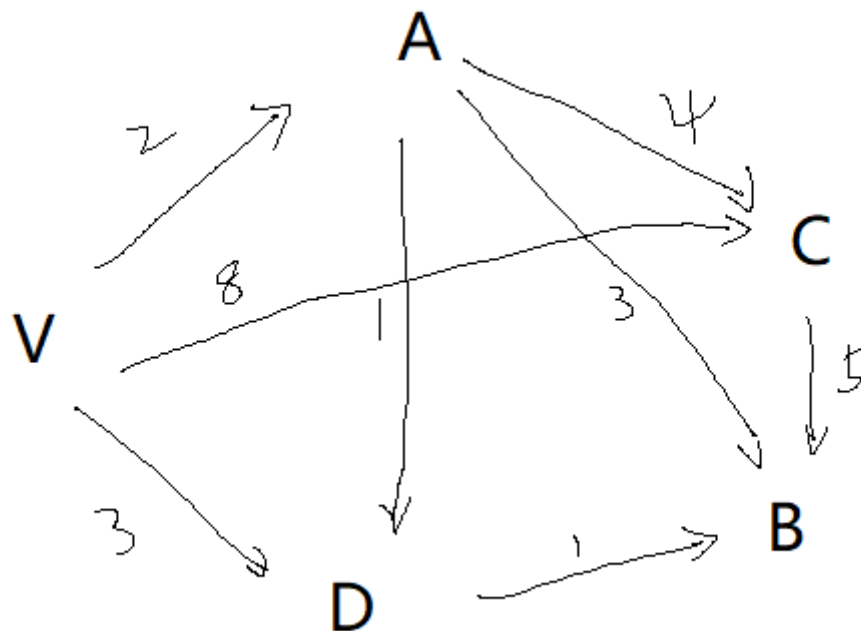
```

1 已被访问

```
void BFS(g)
{
    初始化visited
    for(v=0;v<顶点数;v++)
    {
        if(v0已被访问)
            continue;
        //1.先访问
        //2.标记
        //3.入队
        //4.访问v0的所有邻接点 直到队列为空为止
        while(队列不为空)
        {
            //出队
            //让出队元素的所有的未被访问的邻接点入队
        }
    }
}
```

5.最短路径

通常解决带权有向图(网图)中两个顶点之间最短路径问题。



比如:求V到C的最短路径

6 V->A->C

迪杰斯特拉(DIJKSTRA)算法(迪克斯特拉) <-----
FLOYD(费洛伊德)算法

迪杰斯特拉(DIJKSTRA)算法

目标:解决从网络上任意一顶点(源点)出发,求它到其他顶点(终点)的最短路径的问题。

长度和路径

需要三个辅助向量：

1) 数组 $S[n]$ 标记数组 最短路径有没有求出

$S[i]=1$ 源点 v 到 v_i 的最短路径已经求出

$S[i]=0$ 源点 v 到 v_i 的最短路径没有求出

2) 数组 $dist[n]$ 最短路径的距离

$dist[i]$ 保存源点 v 到 v_i 的最短路径的长度

初始化：

$dist[i] = \langle v, v_i \rangle$ 上的权 w 若 $\langle v, v_i \rangle$ 属于 R 。 v 可以直接到 v_i

$dist[i] = \text{无穷大}$ v 不可以直接到 v_i

3) 向量 $path[n]$ 路径

$path[i]$ 存放的是从源点 v 到 v_i 的最短路径是哪个所经历的顶点

如： $path[i]=BC$ $v \rightarrow B \rightarrow C$

算法步骤

假设图中有 n 个顶点，迪杰斯特拉需要求 $n-1$ 条最短路径。

step1:

显然，从源点 v 到其他顶点的第一条最短路径长度

$dist[u]$

$dist[u] = \min\{dist[w] \mid w=0, 1, 2, \dots, n-1 \text{ 且 } S[w]=0\}$

表示在所有未求出最短路径中找出一条最短的，这条路径作为当前求出的最短路径。

step2:

对所有的 $S[w]=0$ 的顶点，更新 $dist[w]$

if $dist[u] + \langle u, w \rangle < dist[w]$

$dist[w] = dist[u] + \langle u, w \rangle$

