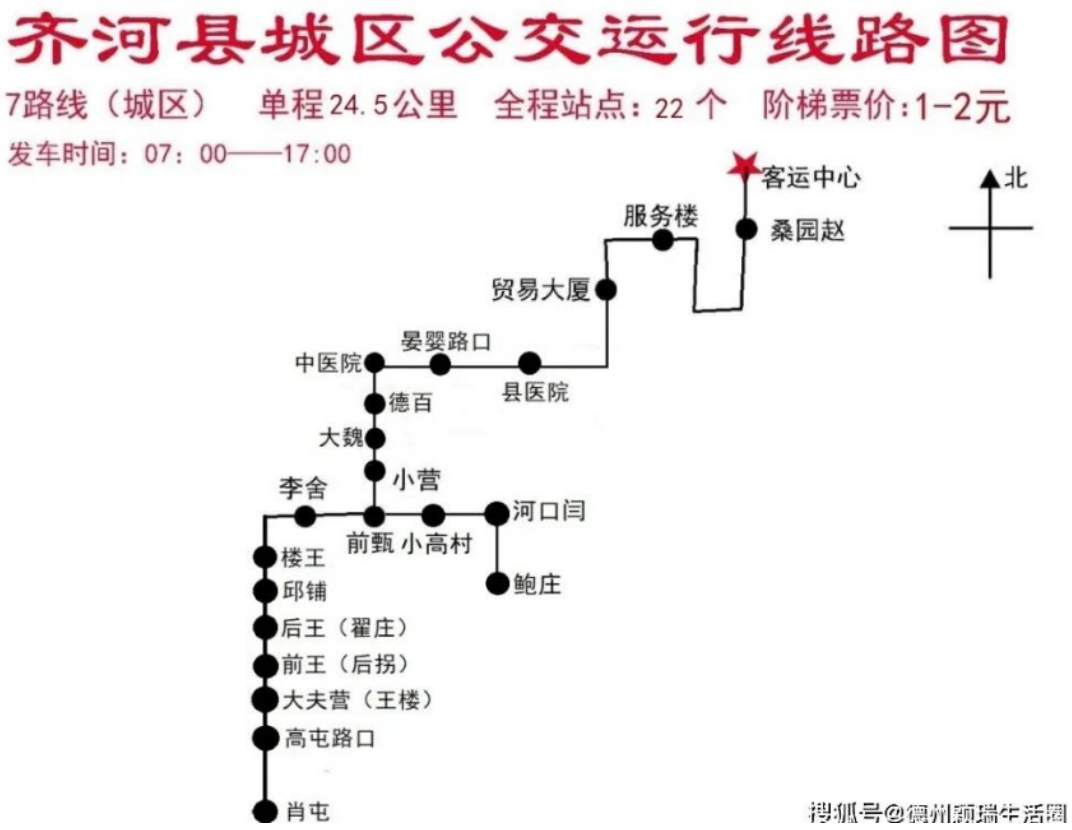
图

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graph(V,R)



公交车线路图是一种典型的图

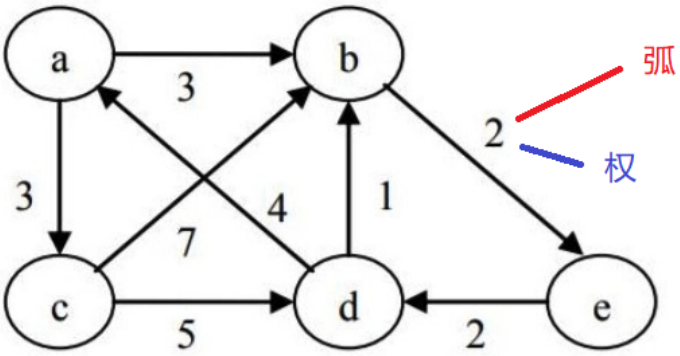
网络(图)

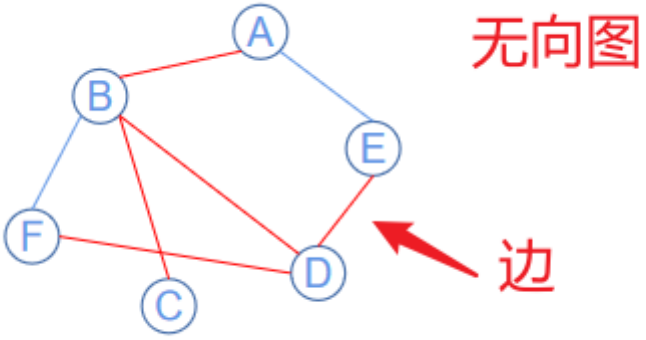
其中，V={vi|vi属于datatype，i=0,1,2...n-1}是图中元素的集合。vi称为顶点(vertex)，当n为0的时候，V为空集。
R={<Vi,Vj>|Vi,Vj属于V且p(Vi,Vj)存在}是图中顶点之间的关系集。p(Vi,Vj)为顶点vi和Vj之间是否存在路径的判定条件，即vi和vj之间有路径存在，则关系(vi,vj)属于R。

2.图的分类

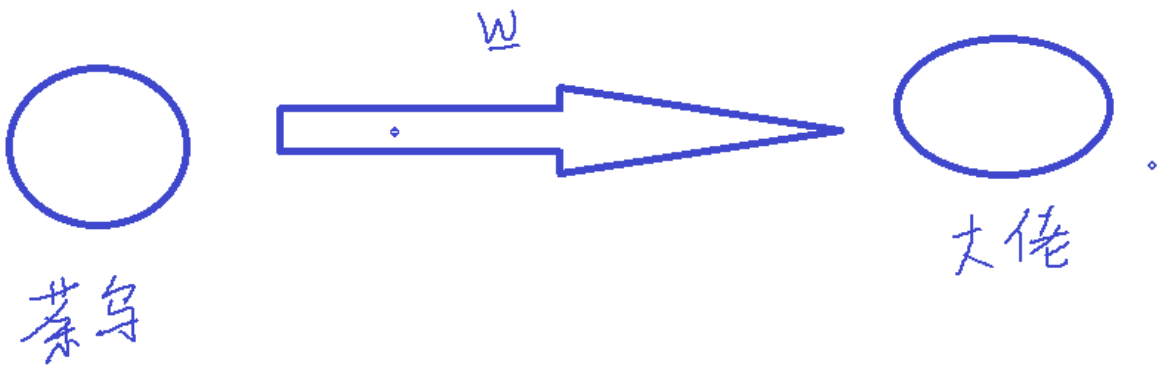
- 有向图 弧
- 无向图 边
- 网

有向图





若在图的关系<Vi,Vj>或<Vj,Vi>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。



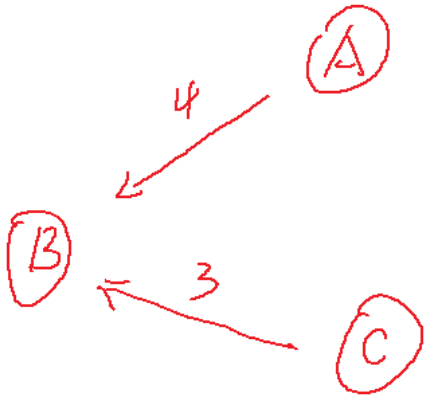
顶点的度

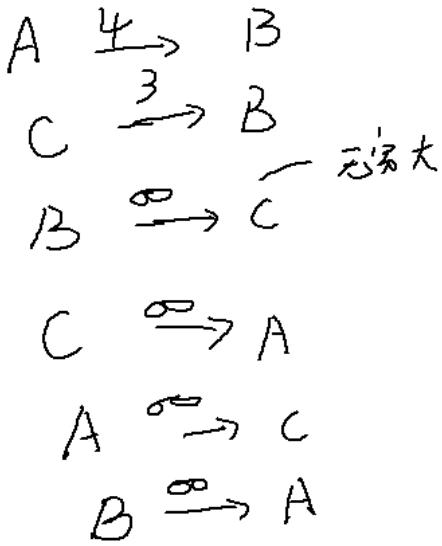
> 顶点的边或弧的条数

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是顶点结合

R是关系集合

可以用两个数组来存储图G

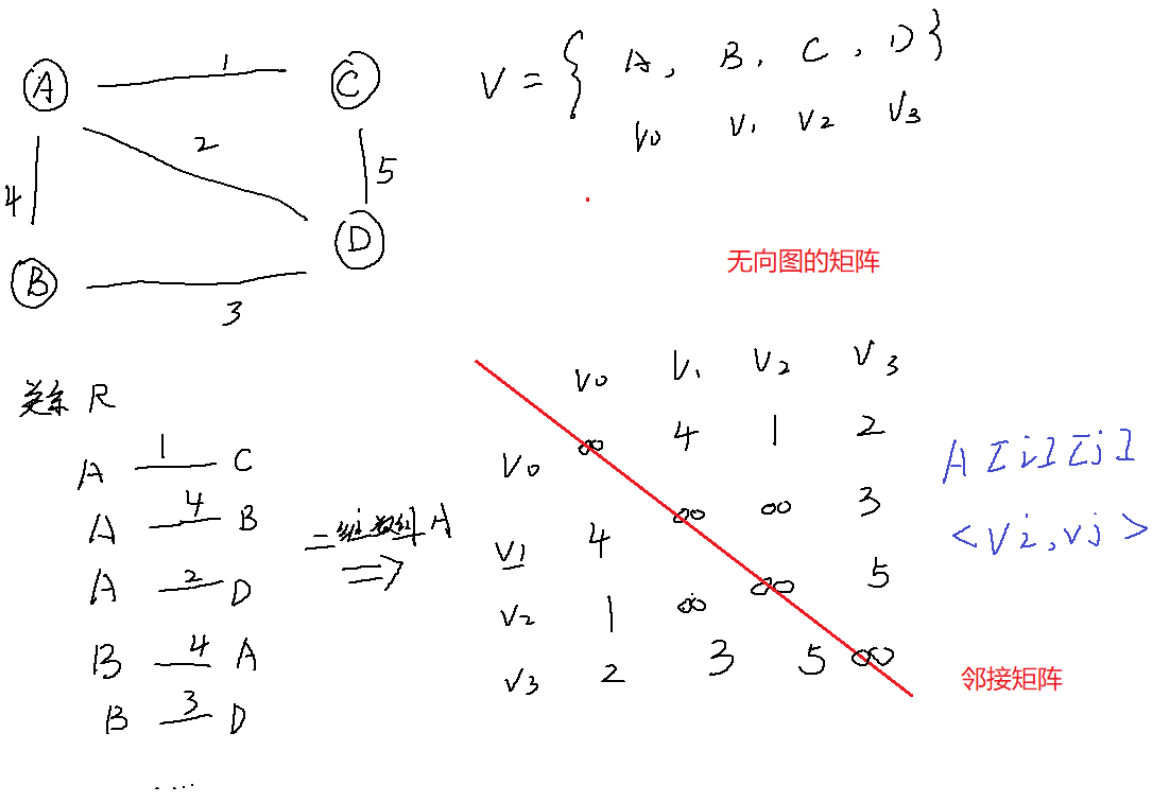
一个一维数组V存储图G顶点集合Vi

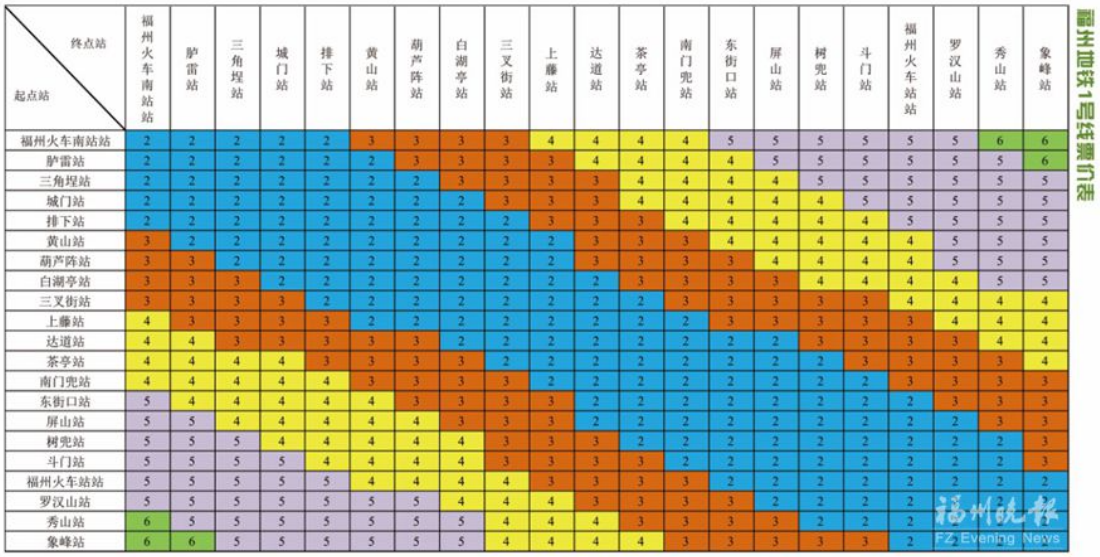
一个二维数组A存储图G顶点之间的关系R

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] ===> <vi,Vj> W





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"
{
```

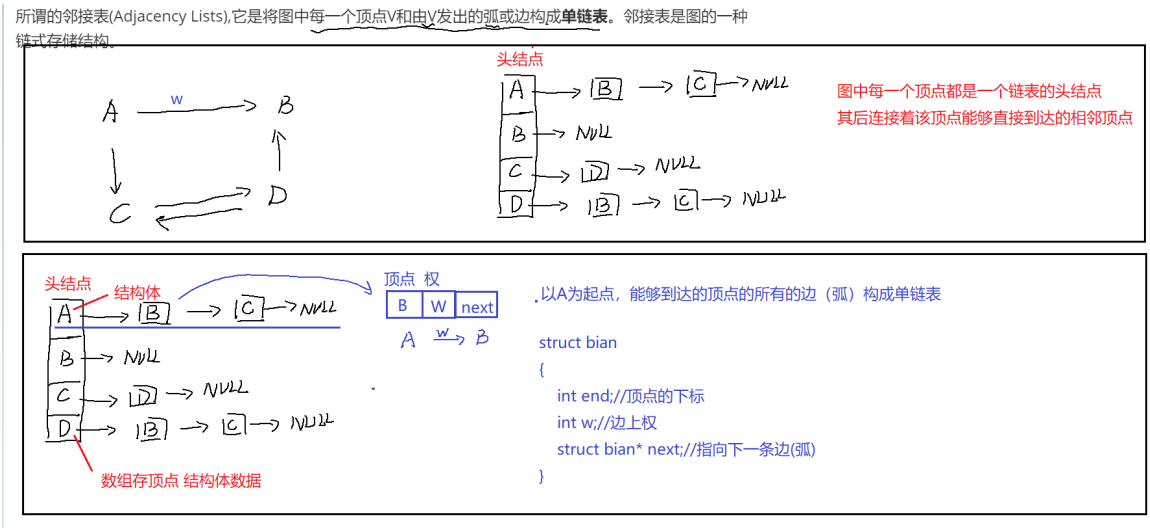
```
    Vtype V[MAXN];//一维数组用来保存图中的顶点
    Adjtype[MAXN][MAXN];//二维数组，“邻接矩阵”，存储“边”

    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

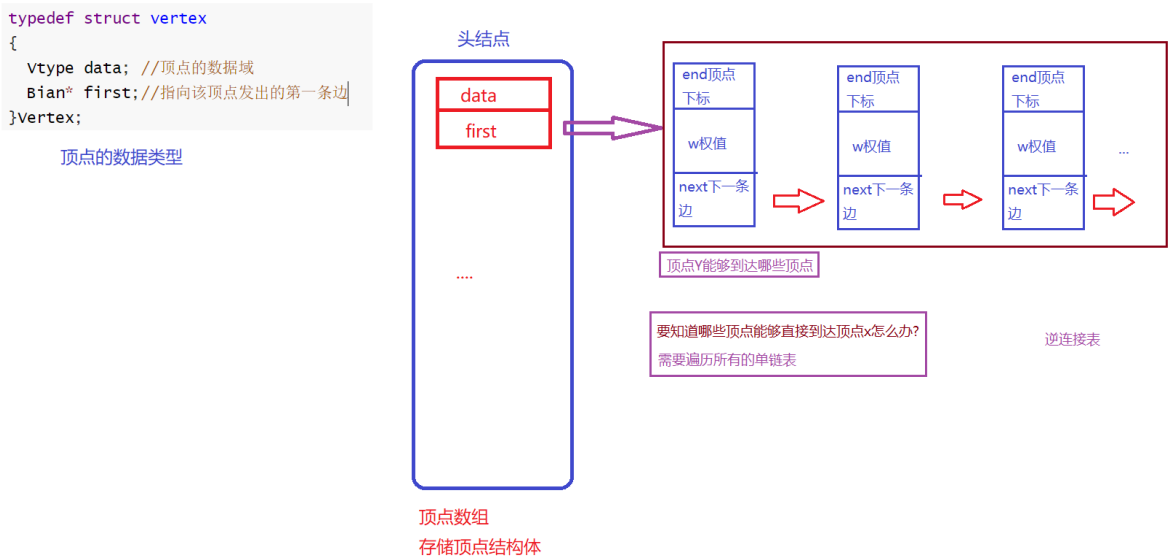
> 链式结构来存储一个表。

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。



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;
```



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V0)出发，访问V0，然后再搜索V0的一个邻接点
Vi(若Vi未被访问)，则访问Vi,再搜索Vi的下一个邻接点Vj(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

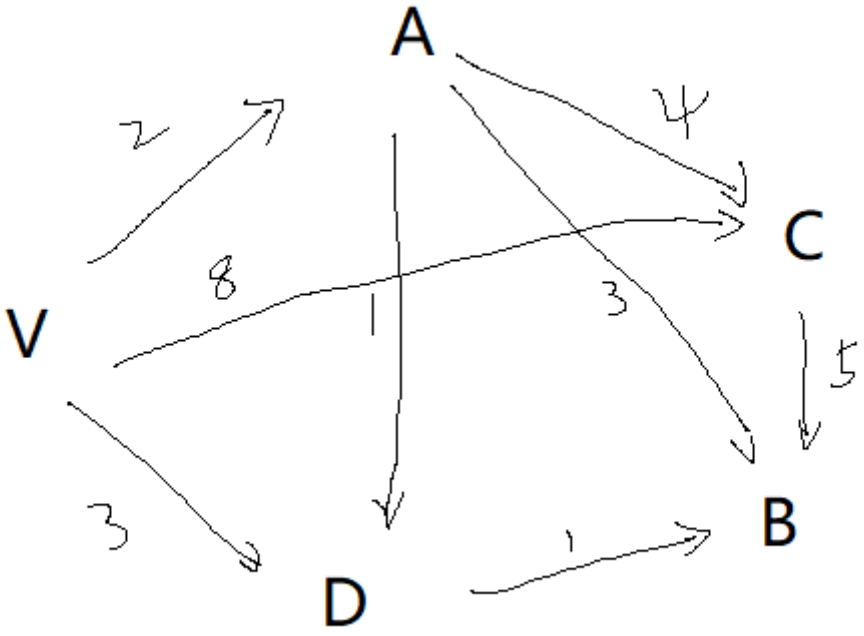
广度优先搜索类似于树的按层次遍历。初始化，图各定点均未被访问。从图中某个定点(设为v0)出发，访问v0，
并且访问v0的各个邻接点(BFS),然后，分别从这些被访问过的顶点出发，仍然按照广度优先搜索的策略来访问
其他顶点...直到能访问的都访问完毕为止。
算法的实现:类似树的按层次遍历
  利用队列

```
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

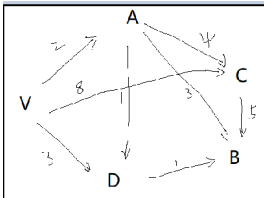
目标:解决从网络上任意一顶点(源点)出发，求它到其他顶点(终点)的最短路径的问题。
长度和路径

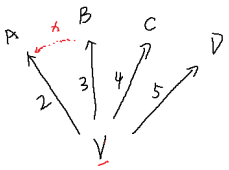
需要三个辅助向量:
1）数组S[n] 标记数组 最短路径有没有求出
S[i]=1 源点V到Vi的最短路径已经求出
S[i]=0 源点V到Vi的最短路径没有求出
2）数组dist[n] 最短路径的距离
dist[i] 保存源点V到Vi的最短路径的长度
初始化:
dist[i] = <V,Vi>上的权w 若<V,Vi>属于R。V可以直接到vi
dist[i] = 无穷大 V不可以直接到Vi
3）向量path[n]路径
path[i]存放的是从源点V到Vi的最短路径是哪个所经历的顶点
如:path[i]=BC V->B->C

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。

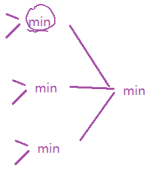
step1:
显然,从源点V到其他顶点的第一条最短路径长度 dist[u]
dist[u]=min{dist[w]|w=0,1,2,..n-1且S[w]=0}
表示在所有未求出最短路径中找出一条最短的，这条路径作为当前求出的最短路径。
step2:
对所有的S[w]=0的顶点，更新dist[w]
if dist[u]+<u,w> < dist[w]
dist[W] = dist[u]+<u,w>





V->A所有的路经里面
<V,A>一定 肯定 必定是最短的那一条路径
WHY???
①:从<V,A><V,B><V,C><V,D>中求出第一条最短路径
假设<V,A>最短路径
S0，V->A的最短路径已求出(已知条件)

②更新其他没有求出的(利用<V,A>
对B
V->B
V->A->B
对C
V->C
V->A->C
对D
V->D
V->A->D



在这三个里面找到那个最小的那个
那么这个最小的那个一定绝对是
V->V的最短路径!
假设V->B最短路径

迪杰斯特拉算法
"贪心算法"

③更新其他的没有求出的
对C
Cmin
V->C
V->A->C
VCmin
V->B->C
V->A->B->C

对D
Dmin
V->D
V->A->D
VDmin
V->B->D
V->A->B->D

在这两个里面找那个最小的那一个，那一个
绝对是V->V的最短路径
假设V->C是最短路径

④更新其他没有求出的
对D
Dmin
V->D
V->A->D
VDmin
V->B->D
V->A->B->D

V->D
V->A->D
V->B->D
V->A->B->D
V->C->D
V->A->C->D
V->B->C->D
V->A->B->C->D

V->D所有的路径