第八章 图

张史梁
slzhang.jdl@pku.edu.cn

内容提要

- 图的基本概念
- 存储表示
- 图的基本运算与周游
- 最小生成树
- 拓扑排序
- 关键路径
- 最短路径

图中的一个重要问题

如何乘坐地铁，从昌平到亦庄时间最短？

最短路径的基本概念

- 路径：如果图中从一个顶点可以到达另一个顶点，则称这两个顶点间存在一条路径(从一个顶点到另一个顶点间可能存在多条路径，而每条路径上经过的边数并不一定相同)

- 路径长度：如果图是一个带权图，则路径长度为路径上各边的权值的总和

- 最短路径长度：两个顶点间路径长度最短的那条路径称为两个顶点间的最短路径，其路径长度称为最短路径长度

最短路径问题

- 问题1：从一个顶点到其他各个顶点的最短路径
  - Dijkstra方法：
  - 按路径长度递增的次序产生最短路径；
  - 该算法假设所有边的权都大于等于零

- 问题2：每一对顶点间的最短路径
  - Floyd算法

Dijkstra

- **Edsger Wybe Dijkstra (04/01/1930-08/06/2002),** 荷兰皇家艺术与科学学院的院士，美国科学院院士，英国计算协会Fellow
- 获得多种奖项：
  - 1972 Turing Award
  - 1974 AFIPS Harry Goode Award
  - 1982 IEEE Computer Pioneer Award

Edsger Dijkstra是1950年代ALGOL语言的一个主要贡献者。ALGOL高级编程语言已经成为结构清晰，数学基础严谨的一个典范。E. W. Dijkstra是现代编程语言的主要贡献者之一，为我们理解程序语言的结构，表示方法与实现做出了巨大的贡献。E. W. Dijkstra 15年的学术著作覆盖了图论的理论工作，教育手册，解释文章和编程语言领域的哲学思考。

Dijkstra

- Dijkstra是计算机科学与工程领域许多概念、术语的缔造者：
  - 结构化程序设计
  - 问题分解
  - 同步
  - 死锁
  - “哲学家晚餐”
  - 栈
  - 向量
  - ……

例子：有5个哲学家坐在一个圆桌上。食物摆在桌子中间，桌子上总共有5把叉子，每个哲学家的左右手各有一把。因为吃饭时，哲学家需要同时使用左手和右手的叉子，所以哲学家必须和他左边和右边的哲学家共享叉子。在这个实验中，假定哲学家每次吃饭的时间长度为单位1，吃完一次后则放下两把叉子。如果等待10个单位时间长度之后，哲学家仍没有得到叉子吃饭，则被饿死。你需要设计一种方法使得任何一个哲学家都不被饿死。

- GoTo Statement Considered Harmful （1968）

在现代编程语言方面，E. W. Dijkstra也以他著名的反对(过分)使用GOTO语句的文章而著名。这篇文章被认为是现代编程语言逐渐不鼓励使用GOTO 语句，而使用编程控制结构，如while loop等等的一个分水岭。

Dijkstra

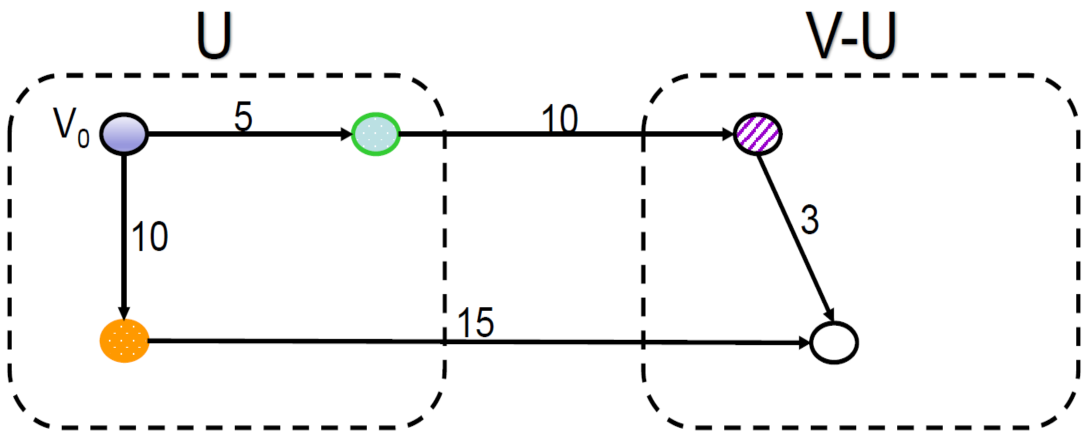
- 年轻时代，Dijkstra在University of Leiden（荷兰最古老的大学）学习理论物理，但很快他就意识到其兴趣不在于理论物理虽然获得了其数学和理论物理的学位。
- 后来，Dijkstra从University of Amsterdam获得了其博士学位。1962年-1984年，作为一个数学教授任职于Eindhoven University of Technology.
- 1984年至1999年，作为计算机系系主任任职与美国UT Austin（德克萨斯州大学奥斯汀分校），并于1999年退休。

最短路径问题

- 问题1：从一个顶点到其他各个顶点的最短路径
  - Dijkstra方法：
  - 按路径长度递增的次序产生最短路径；
  - 该算法假设所有边的权都大于等于零

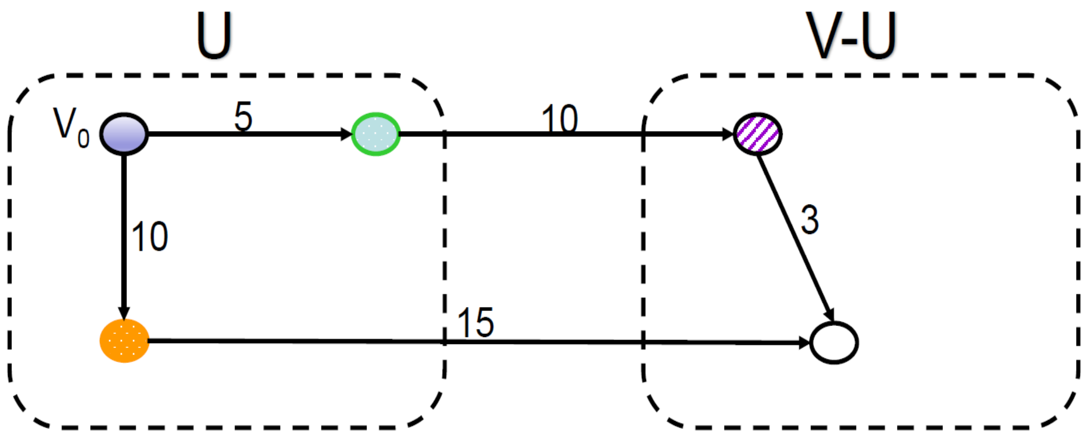
- 问题2：每一对顶点间的最短路径
  - Floyd算法

Dijkstra算法的基本思路



- 设置集合U存放已求出最短路径的顶点，则V-U是尚未确定最短路径的顶点集合；每个顶点对应一个距离值
  - 集合U中顶点的距离值是从顶点v0到该顶点的最短路径长度；
  - 集合V-U中顶点的距离值是从顶点v0到该顶点的只包括集合U中顶点为中间顶点的最短路径长度

Dijkstra算法的基本思路



① 在集合V-U中选择距离值最小的顶点v_{min}加入集合U

② 对集合V-U中各顶点（例如 v_i ）的距离值进行修正，如果加入顶点v_{min}为中间顶点后，使v_0到v_i的距离值比原来的距离值更小，则修改v_i的距离值为更小的值

③ 反复操作，直到从v_0出发可以到达的所有顶点都在集合U中为止

Dijkstra算法实现

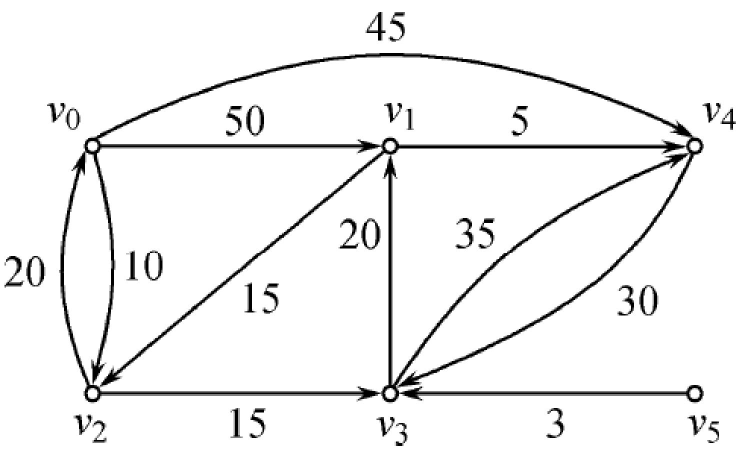
- 数据结构：

  设置一个数组dist[n]，用于存放顶点v_0到v_i的最短路径及其最短路径长度

```
typedef struct
{
        VexType vertex;      /* 顶点信息 */
        AdjType length;      /* 最短路径长度 */
        int prevex;          /* 从v0到vi最短路径上vi的前趋 */
}Path;
Path dist[n];                /* n为图中顶点个数*/
```

Dijkstra算法-例子

- 已知带权图G如图所示及其邻接矩阵A，求从顶点v0到其它各顶点的最短路径



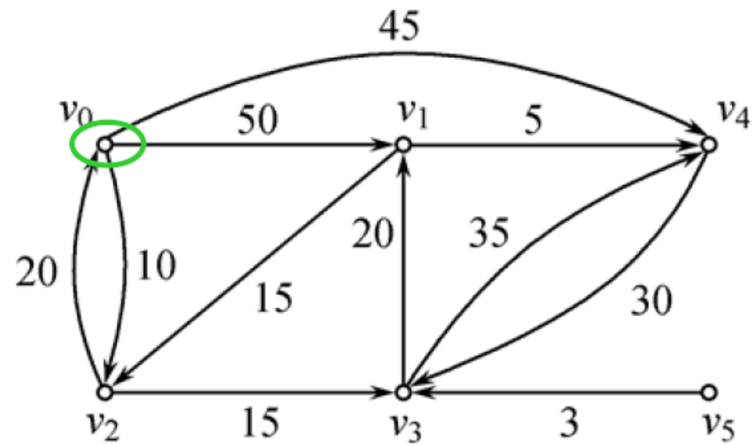
$$\text{arcs} = \begin{bmatrix} 0 & 50 & 10 & \infty & 45 & \infty \\ \infty & 0 & 15 & \infty & 5 & \infty \\ 20 & \infty & 0 & 15 & \infty & \infty \\ \infty & 20 & \infty & 0 & 35 & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty \\ \infty & \infty & \infty & 3 & \infty & 0 \end{bmatrix}$$

Dijkstra算法-例子

① 初始时，集合U中只有顶点v_0，从顶点v_0到其它顶点v_i (i=1,2,…,n-1)的最短路径长度为边(v_0,v_i)的权值。若顶点v_0和v_i不相邻，则假设存在一条从v_0到v_i权为无穷的边

dist[n]={{0,0},{50,0},{10,0},{MAX,-1},{45,0},{MAX,-1}}

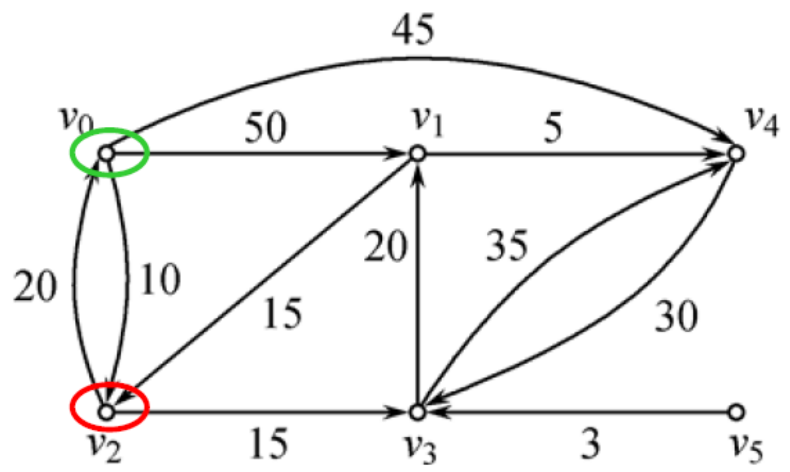
最短路径长度 前驱



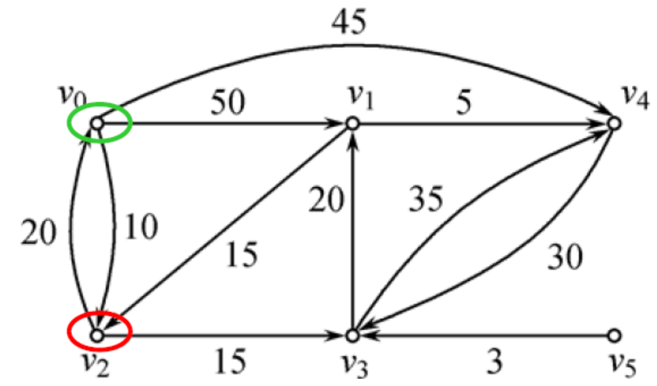
Dijkstra算法-例子

② 在集合V-U中找出距离值最小的顶点v_{min}=2，将其加入集合U，从顶点v_0到顶点v_{min}的最短路径长度就是v_{min}的距离值

dist[n]={{0,0},{50,0},{10,0},{MAX,-1},{45,0},{MAX,-1}}



Dijkstra算法-例子



dist[n]={{0,0},{50,0},{10,0},{MAX,-1},{45,0},{MAX,-1}}

③ 调整集合V-U中顶点的距离值，对v_i ($v_i \in$ V-U)，如果

dist[i].length > dist[min].length+graph.arcs[min][i]

则将顶点v_i的距离值改为dist[min].length+graph.arcs[min][i]，

并将路径上v_i的前趋顶点改为v_{min}，即：dist[i].prevex=min

min=2

- 因为dist[1].length=50，dist[2].length+graph.arcs[2][1]=10+MAX，顶点v1的距离值不需要调整
- 因为dist[3].length=MAX，dist[2].length+graph.arcs[2][3]=10+15=25，顶点v3的距离值调整为25，其前趋顶点为v2
- 同理，顶点v4，v5的距离值不需要调整

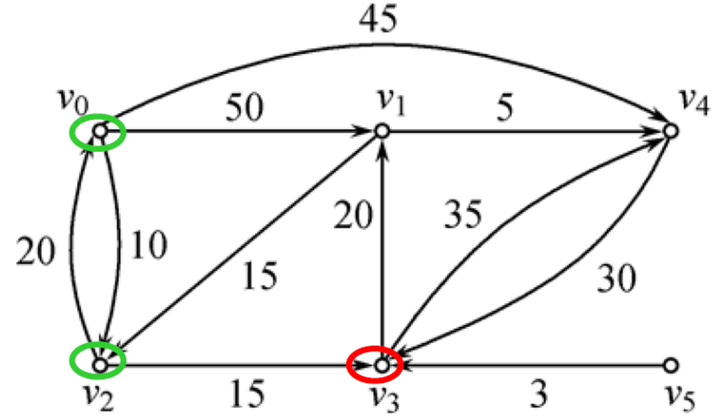
dist[n]为{{0,0},{50,0},{10,0},{25,2},{45,0},{MAX,-1}}

Dijkstra算法-例子

第三步所得结果:
dist[n]为{{0, 0}, {50, 0}, {10, 0}, {25, 2}, {45, 0}, {MAX, -1} }

④ 同理在集合V-U中找出当前距离值最小的顶点v3，并调整集合V-U中顶点的距离值 dist[n]为

{{0,0},{45,3},{10,0},{25,2},{45,0},{MAX,-1}}调整v1



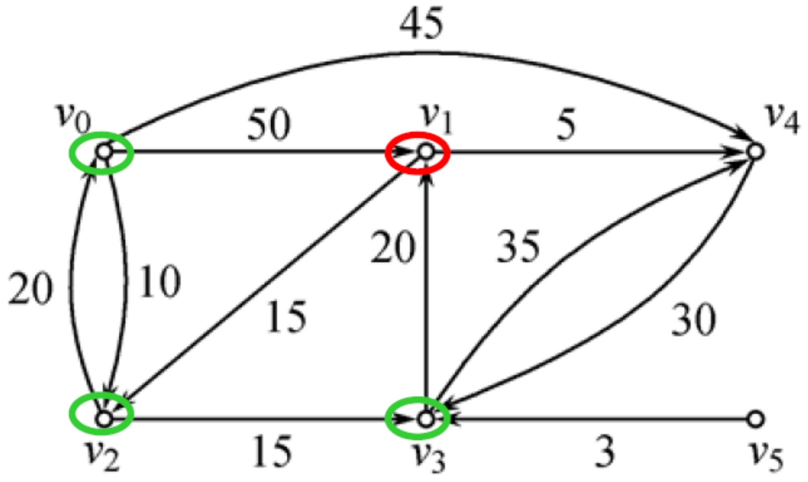
Dijkstra算法-例子

第四步所得结果:

dist[n]为{{0, 0},{45, 3}, {10, 0}, {25, 2}, {45, 0}, {MAX, -1} }

⑤ 在集合V-U中找出当前距离值最小的顶点v1

dist[n]为{{0,0},{45,3},{10,0},{25,2},{45,0},{MAX,-1}}

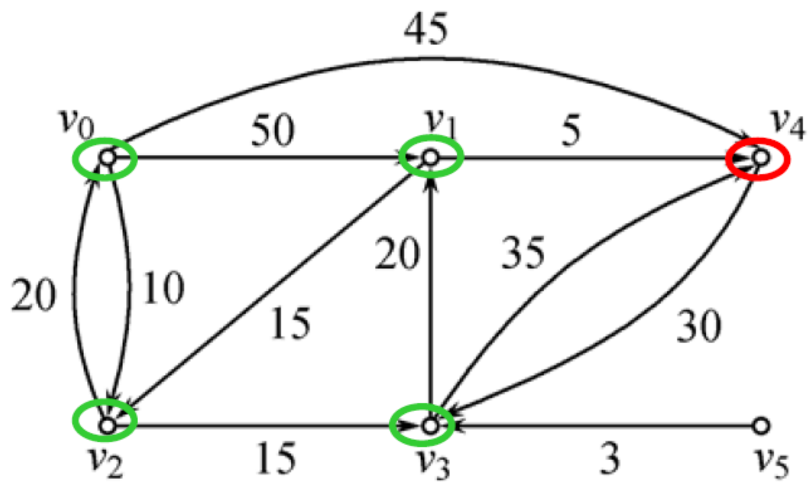


Dijkstra算法-例子

第五步所得结果:

dist[n]为{{0, 0}, {45, 3}, {10, 0}, {25, 2}, {45, 0}, {MAX, -1} }

⑥ 在集合V-U中找出当前距离值最小的顶点v4
dist[n]为{{0,0},{45,3},{10,0},{25,2},{45,0},{MAX,-1}}

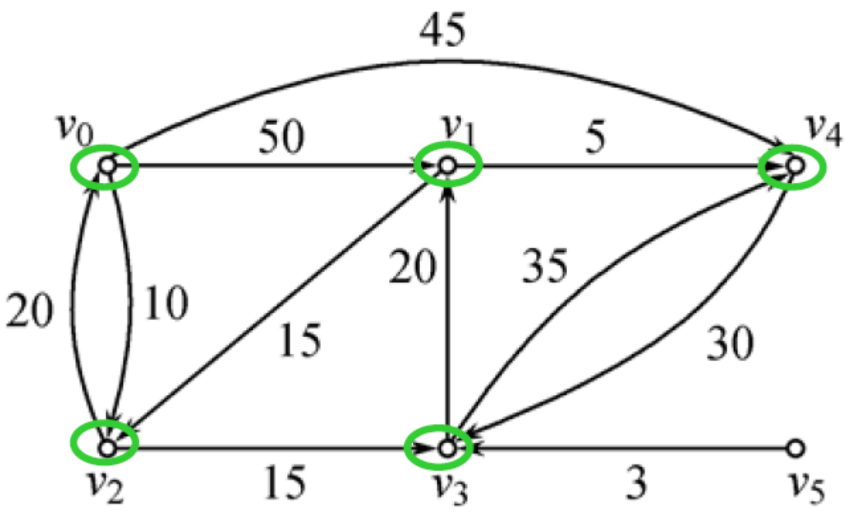


Dijkstra算法-例子

第六步所得结果:

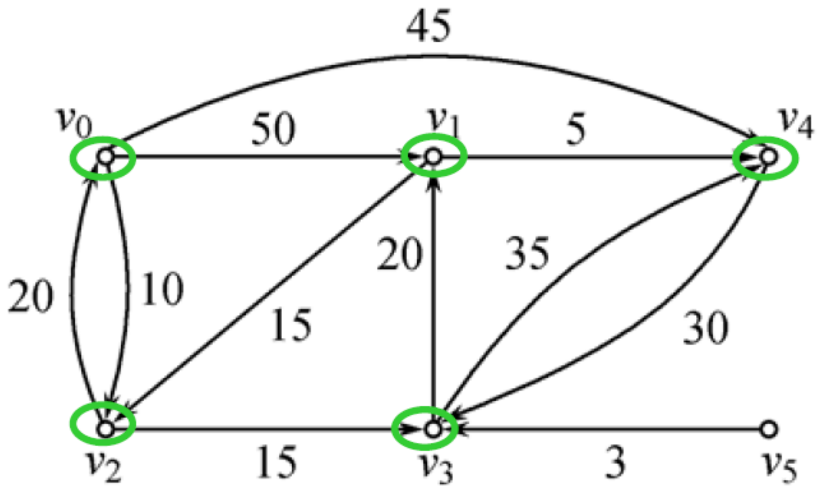
dist[n]为{{0, 0}, {45, 3}, {10, 0}, {25, 2}, {45, 0}, {MAX, -1} }

⑦ 没有可以再加入集合U的顶点了，说明从顶点v0到顶点v5之间无路径相通。



Dijkstra算法

- dist[n]={{0, 0}, {45, 3}, {10, 0}, {25, 2}, {45, 0}, {MAX, -1}}
- 由数组dist的prevex字段得到顶点v0到各顶点的最短路径
  - 如从v0到v1的最短路径，dist[1].prevex=3可知路径上顶点v1的前一个顶点是v3，dist[3].prevex=2可知路径上顶点v3的前一个顶点是v2，dist[2].prevex=0可知路径上前一个顶点是v0，即最短路径为(v0 , v2 , v3 , v1)。
  - 其路径长度为：dist[1].length=45



邻接矩阵表示法 - 存储结构

```
#define MAXVEX 100 //  常数1
typedef char VexType;
typedef float AdjType;

typedef struct {
        VexType vexs[MAXVEX];                      /* 顶点信息 */
        AdjType arcs[MAXVEX] [MAXVEX];             /* 邻接矩阵 */
        int n;                                     /* 图的顶点个数 */
  } Graph;
```

Dijkstra算法

```
void dijkstra(Graph graph, Path dist[])
{
    int  i, j, minvex;
    AdjType min;
    dist[0].length=0;
    dist[0].prevex=0;
    dist[0].vertex=graph.vexs[0];
    graph.arcs[0][0]=1;          /* 表示顶点v0在集合U中 */
    for(i=1; i<graph.n; i++)    /* 初始化集合V-U中顶点的距离值 */
    {
       dist[i].length=graph.arcs[0][i];
       dist[i].vertex=graph.vexs[i];
       if (dist[i].length!=MAX) dist[i].prevex=0;
       else dist[i].preve= -1;
    }
```

Dijkstra算法

```
for(i=1; i<graph.n; i++)     //对图中的每个结点寻找最短路径
    {
        min=MAX;     minvex=0;
        for(j=1; j<graph.n; j++)     /*在V-U中选出距离值最小顶点*/
          if( (graph.arcs[j][j]==0) && (dist[j].length<min) )
          {
              min=dist[j].length;
              minvex=j;
          }
       if(minvex==0) break;     /*  从v0没有路径可通往集合V-U中顶点 */

       graph.arcs[minvex][minvex]=1;     /* V-U路径最小顶点minvex */
```

Dijkstra算法

```
        for(j=1; j<graph.n; j++)      /*  调整集合V-U中顶点的最短路径 */
        {
            if(graph.arcs[j][j]==1) continue;
            if(dist[j].length>dist[minvex].length+graph.arcs[minvex][j])
            {
                dist[j].length = dist[minvex].length+graph.arcs[minvex][j];
                dist[j].prevex = minvex;
            }
        }  // 结束调整for循环

    }  // 结束每个结点寻找最短路径
}
```

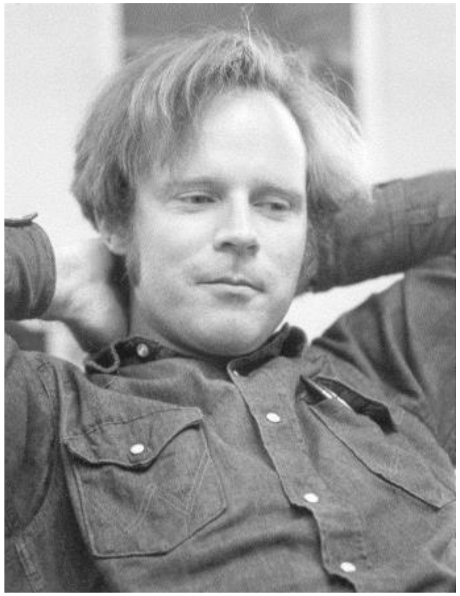
Dijkstra算法分析

- 算法初始化部分的时间复杂度为O(n)，
- 求最短路径部分由一个大循环组成，其中外循环运行n-1次，内循环为两个，均运行n-1次。
- 所以，算法的时间复杂度为$O(n^2)$

Floyd算法

- 对于计算图中每一对顶点间的最短路径
  - Floyd算法
  - 使用Dijkstra方法：依次把图中每个顶点作为起始点，应用Dijkstra方法求出从起始点到图中其它顶点的最短路径。时间复杂度为$O(n^3)$

Floyd



- Robert W. Floyd (06/08/1936—09/25/2001)，第十三位图灵奖(1978年)获得者
- 斯坦福大学计算机科学系教授，“自学成才的计算机科学家”
  - 14岁就完成了其中学教育，然后进入芝加哥大学，并在年仅17岁时获得文学学士学位。1958年，Robert获得了其第二个学士学位，专业为物理。
  - 在Robert年仅27时，他被CMU聘请为副教授。6年后，获得了斯坦福大学的终身教授的职务。
  - 在斯坦福大学，Robert与Donald Knuth成为同事和亲密的朋友。

Floyd算法的基本思路

- 时间复杂度为$O(n^3)$，但算法较为简单，容易理解。
- 对于图G=(V,E)，有n个顶点。求顶点vi到vj的最短路径
  - 首先考虑路径(vi, v0)和(v0,vj)是否存在？如果存在，则比较(vi,vj)和(vi,v0)+(v0,vj)的路径长度，取较短者为当前的最短路径

    (vi, vj)

    (vi, v0, vj)

  - 其次，考虑从vi到vj是否存在包含v1为中间点的路径? 如果存在，则将其与前一次求得的允许v0为中间点的最短路径长度比较，取较短者为当前的最短路径。

    (vi, v0, vj)

    (vi, {v0, v1}, vj)

Floyd算法的基本思路

- 如果(vi,…,vk)和(vk, …,vj)分别是从vi到vk和从vk到vj允许k个顶点v0,v1,…, vk-1为中间点的最短路径，则将(vi,…,vk)+(vk, …,vj) 与已经得到的从vi到vj允许k个顶点v0,v1,…,vk-1为中间点的最短路径进行比较，取较短者为从vi到vj允许k+1个顶点v0,v1,…,vk为中间点的最短路径。

$$(v_i, \{v_0,v_1,\ldots,v_{k-1}\}, v_j)$$

$$(v_i, \{v_0,v_1,\ldots, v_{k-1},v_k\}, v_j)$$

………………

- 依次类推，直到加入顶点vn-1为止，则得到的是vi到vj允许n个顶点v0,v1,…,vn-1为中间点的最短路径。此时，已经考虑了所有顶点为中间点的可能性，因此得到结果。

Floyd算法 – 例子

- 为了在算法中不破坏原始的关系矩阵，需要定义一个与关系矩阵同样大小的矩阵处理存放每对顶点间的距离值（或最短路径长度），并以关系矩阵作为其初始状态，
- 为了保存全部最短路径的轨迹，需要另外设计一个与关系矩阵同样大小的整数矩阵，存放vi到vj最短路径上vi的**后继顶点**的下标。

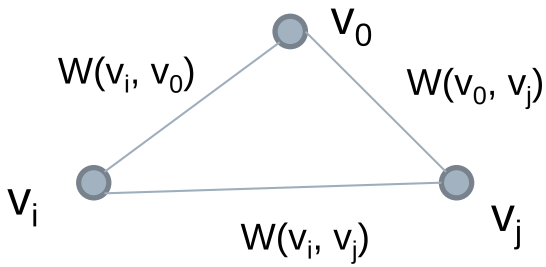
```
最短路径及长度存储结构
typedef struct
{   /* 存放每对顶点间最短路径长度 */
    AdjType a[MAXVEX][MAXVEX];
    /* nextvex[i][j]存放vi到vj最短路径上vi的后继顶点的下标值 */
   int nextvex[MAXVX][MAXVEX];
}ShortPath;
```

Floyd算法

- 对于图G=(V,E)，有n个顶点，采用邻接矩阵存储。求任意两顶点（v_i到v_j）的最短路径。
  ① 首先考虑 (v_i, v_0)和(v_0,v_j)是否存在，如果存在，则比较(v_i,v_j)和$(v_i,v_0)+(v_0,v_j)$的路径长度，取较短者为当前的最短路径。该路径是从v_i到v_j只允许顶点v_0为中间点的最短路径（这样可求出任意两顶点间只允许顶点v_0为中间点的最短路径）。

$$A_0(v_i, v_j)=W(v_i, v_j)$$

$$A_1(v_i, v_j) =\min(W(v_i, v_j), W(v_i, v_0)+ W(v_0, v_j))$$



Floyd算法实现

② 其次，考虑从v_i到v_j是否存在只包含v_0和v_1为中间点的路径: $(v_i,\ldots,v_1,\ldots,v_j)$，其中$(v_i,\ldots,v_1)$和$(v_1,\ldots,v_j)$分别是前一次找到的只允许顶点$v_0$为中间点的最短路径。如果存在这样的路径，则$(v_i,\ldots,v_1)+(v_1,\ldots,v_j)$为路径$v_i,\ldots,v_1,\ldots,v_j$的长度，将其与前一次求得的只允许$v_0$为中间点的最短路径长度比较，取较短者为当前的最短路径。

$$A_2(v_i, v_j) = \min(A_1(v_i, v_j), A_1(v_i, v_1) + A_1(v_1, v_j))$$

$$A_1(v_i, v_1) = \min(W(v_i, v_1), W(v_i, v_0) + W(v_0, v_1))$$

$$A_1(v_1, v_j) = \min(W(v_1, v_j), W(v_1, v_0) + W(v_0, v_j))$$

Floyd算法实现

③ 如果(vi,…,vk)和(vk , …,vj)分别是从vi到vk和从vk到vj只允许顶点v0,v1,…, vk-1为中间点的最短路径，则将 (vi,…,vk)+(vk , …,vj) 与已经得到的从vi到vj只允许顶点v0,v1,…,vk-1为中间点的最短路径进行比较，取较短者为从vi到vj允许v0,v1,…,vk为中间点的最短路径。

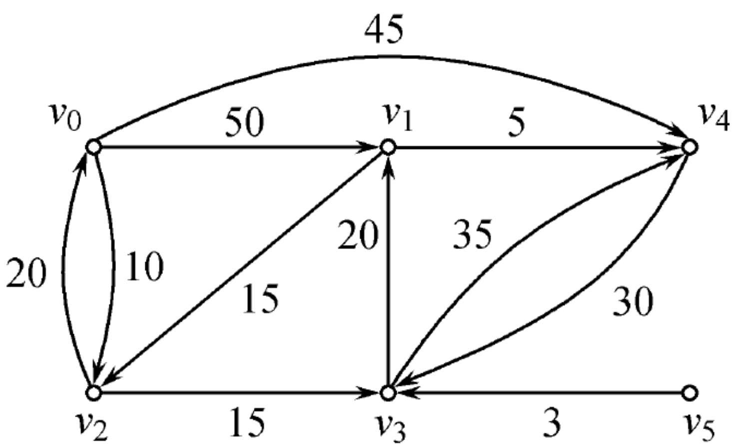
$A_k(v_i, v_j) = \min(A_{k-1}(v_i, v_j), A_{k-1}(v_i, v_k) + A_{k-1}(v_1, v_k))$

………………

依次类推，直到加入顶点vn-1为止，则得到的是vi到vj允许n个顶点v0,v1,…,vn-1为中间点的最短路径。此时，已经考虑了所有顶点为中间点的可能性，因此，得到结果。

Floyd算法 – 例子

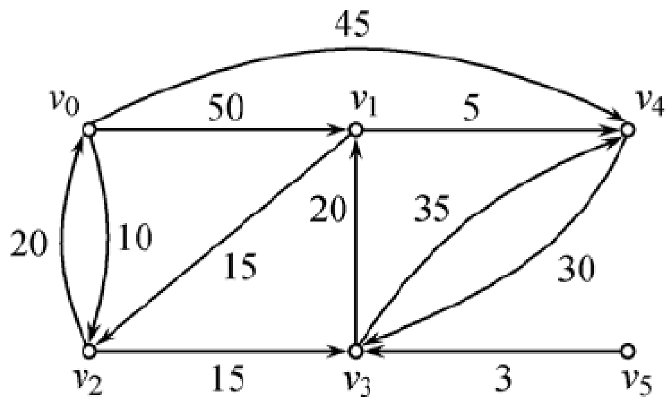
- 已知带权图G如图所示及其邻接矩阵A，求从顶点v0到其它各顶点的最短路径



$$\text{arcs} = \begin{bmatrix} 0 & 50 & 10 & \infty & 45 & \infty \\ \infty & 0 & 15 & \infty & 5 & \infty \\ 20 & \infty & 0 & 15 & \infty & \infty \\ \infty & 20 & \infty & 0 & 35 & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty \\ \infty & \infty & \infty & 3 & \infty & 0 \end{bmatrix}$$

Floyd算法 – 例子

- 初始化



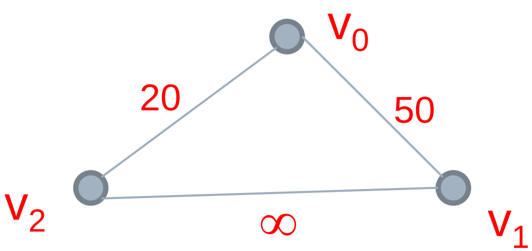
$$A_0 = \begin{bmatrix} 0 & 50 & 10 & \infty & 45 & \infty \\ \infty & 0 & 15 & \infty & 5 & \infty \\ 20 & \infty & 0 & 15 & \infty & \infty \\ \infty & 20 & \infty & 0 & 35 & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty \\ \infty & \infty & \infty & 3 & \infty & 0 \end{bmatrix}$$

$$\text{nextvex}_0 = \begin{bmatrix} 0 & 1 & 2 & -1 & 4 & -1 \\ -1 & 1 & 2 & -1 & 4 & -1 \\ 0 & -1 & 2 & 3 & -1 & -1 \\ -1 & 1 & -1 & 3 & 4 & -1 \\ -1 & -1 & -1 & 3 & 4 & -1 \\ -1 & -1 & -1 & 3 & -1 & 5 \end{bmatrix}$$

Floyd算法 – 例子

$$A_0 = \begin{bmatrix} 0 & 50 & 10 & \infty & 45 & \infty \\ \infty & 0 & 15 & \infty & 5 & \infty \\ 20 & \infty & 0 & 15 & \infty & \infty \\ \infty & 20 & \infty & 0 & 35 & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty \\ \infty & \infty & \infty & 3 & \infty & 0 \end{bmatrix}$$

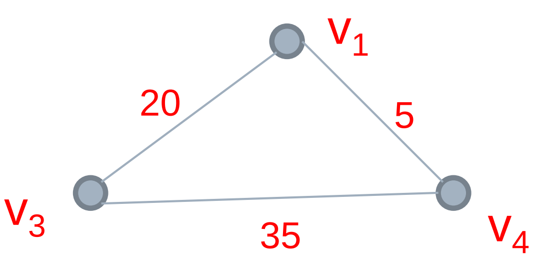
加入第一个顶点 v_0，$A_1[i][j]=\min\{A_0[i][j], A_0[i][0]+A_0[0][j]\}$ $0 \leqslant i \leqslant n-1, 0 \leqslant j \leqslant n-1$



$$A_1 = \begin{bmatrix} 0 & 50 & 10 & \infty & 45 & \infty \\ \infty & 0 & 15 & \infty & 5 & \infty \\ 20 & 70 & 0 & 15 & 65 & \infty \\ \infty & 20 & \infty & 0 & 35 & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty \\ \infty & \infty & \infty & 3 & \infty & 0 \end{bmatrix}$$

$$\text{nextvex}_1 = \begin{bmatrix} 0 & 1 & 2 & -1 & 4 & -1 \\ -1 & 1 & 2 & -1 & 4 & -1 \\ 0 & 0 & 2 & 3 & 0 & -1 \\ -1 & 1 & -1 & 3 & 4 & -1 \\ -1 & -1 & -1 & 3 & 4 & -1 \\ -1 & -1 & -1 & 3 & -1 & 5 \end{bmatrix}$$

加入顶点 v_1，$A_2[i][j]=\min\{A_1[i][j], A_1[i][1]+A_1[1][j]\}$ $0 \leqslant i \leqslant n-1, 0 \leqslant j \leqslant n-1$



$$A_2 = \begin{bmatrix} 0 & 50 & 10 & \infty & 45 & \infty \\ \infty & 0 & 15 & \infty & 5 & \infty \\ 20 & 70 & 0 & 15 & 65 & \infty \\ \infty & 20 & 35 & 0 & 25 & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty \\ \infty & \infty & \infty & 3 & \infty & 0 \end{bmatrix}$$

$$\text{nextvex}_2 = \begin{bmatrix} 0 & 1 & 2 & -1 & 4 & -1 \\ -1 & 1 & 2 & -1 & 4 & -1 \\ 0 & 0 & 2 & 3 & 0 & -1 \\ -1 & 1 & 1 & 3 & 1 & -1 \\ -1 & -1 & -1 & 3 & 4 & -1 \\ -1 & -1 & -1 & 3 & -1 & 5 \end{bmatrix}$$

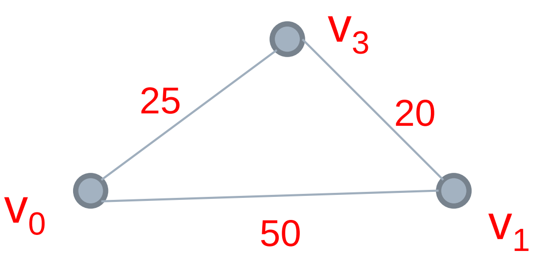
Floyd算法 – 例子

$$A_2 = \begin{bmatrix} 0 & 50 & 10 & \infty & 45 & \infty \\ \infty & 0 & 15 & \infty & 5 & \infty \\ 20 & 70 & 0 & 15 & 65 & \infty \\ \infty & 20 & 35 & 0 & 25 & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty \\ \infty & \infty & \infty & 3 & \infty & 0 \end{bmatrix}$$

加入顶点 v_2，$A_3[i][j]=\min(A_2[i][j], A_2[i][2]+A_2[2][j])$ $\quad 0\leqslant i\leqslant n-1, 0\leqslant j\leqslant n-1$

$$A_3 = \begin{bmatrix} 0 & 50 & 10 & 25 & 45 & \infty \\ 35 & 0 & 15 & 30 & 5 & \infty \\ 20 & 70 & 0 & 15 & 65 & \infty \\ 55 & 20 & 35 & 0 & 25 & \infty \\ \infty & \infty & \infty & 30 & 0 & \infty \\ \infty & \infty & \infty & 3 & \infty & 0 \end{bmatrix} \qquad \text{nextvex}_3 = \begin{bmatrix} 0 & 1 & 2 & 2 & 4 & -1 \\ 2 & 1 & 2 & 2 & 4 & -1 \\ 0 & 0 & 2 & 3 & 0 & -1 \\ 1 & 1 & 1 & 3 & 1 & -1 \\ -1 & -1 & -1 & 3 & 4 & -1 \\ -1 & -1 & -1 & 3 & -1 & 5 \end{bmatrix}$$

加入顶点 v_3，$A_4[i][j]=\min(A_3[i][j], A_3[i][3]+A_3[3][j])$ $\quad 0\leqslant i\leqslant n-1, 0\leqslant j\leqslant n-1$

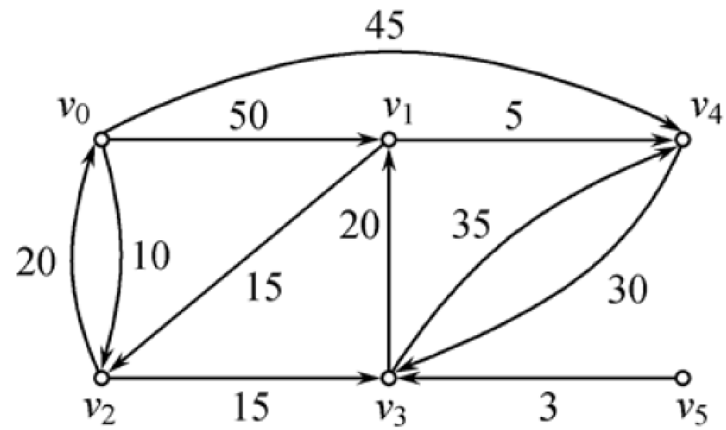


$$A_4 = \begin{bmatrix} 0 & 45 & 10 & 25 & 45 & \infty \\ 35 & 0 & 15 & 30 & 5 & \infty \\ 20 & 35 & 0 & 15 & 40 & \infty \\ 55 & 20 & 35 & 0 & 25 & \infty \\ 85 & 50 & 65 & 30 & 0 & \infty \\ 58 & 23 & 38 & 3 & 28 & 0 \end{bmatrix} \qquad \text{nextvex}_4 = \begin{bmatrix} 0 & 2 & 2 & 2 & 4 & -1 \\ 2 & 1 & 2 & 2 & 4 & -1 \\ 0 & 3 & 2 & 3 & 3 & -1 \\ 1 & 1 & 1 & 3 & 1 & -1 \\ 3 & 3 & 3 & 3 & 4 & -1 \\ 3 & 3 & 3 & 3 & 3 & 5 \end{bmatrix}$$

A_5、A_6与 A_4相同，nextvex_5、nextvex_6与 nextvex_4相同。

Floyd算法

$$A_4 = \begin{bmatrix} 0 & 45 & 10 & 25 & 45 & \infty \\ 35 & 0 & 15 & 30 & 5 & \infty \\ 20 & 35 & 0 & 15 & 40 & \infty \\ 55 & 20 & 35 & 0 & 25 & \infty \\ 85 & 50 & 65 & 30 & 0 & \infty \\ 58 & 23 & 38 & 3 & 28 & 0 \end{bmatrix} \qquad nextvex_4 = \begin{bmatrix} 0 & 2 & 2 & 2 & 4 & -1 \\ 2 & 1 & 2 & 2 & 4 & -1 \\ 0 & 3 & 2 & 3 & 3 & -1 \\ 1 & 1 & 1 & 3 & 1 & -1 \\ 3 & 3 & 3 & 3 & 4 & -1 \\ 3 & 3 & 3 & 3 & 3 & 5 \end{bmatrix}$$



- 求v0到v1的最短路径：
- 由A[0][1]可知v0到v1的最短路径长度为45，
- 最短路径为v0→v2→v3→v1
  - 由nextvex[0][1]=2可知顶点v0的下一顶点为v2，
  - 由nextvex[2][1]=3可知v2的下一顶点为v3，
  - 由nextvex[3][1]=1可知v3的下一顶点为v1，

时间复杂度为O(n^3)，算法简单，容易理解。

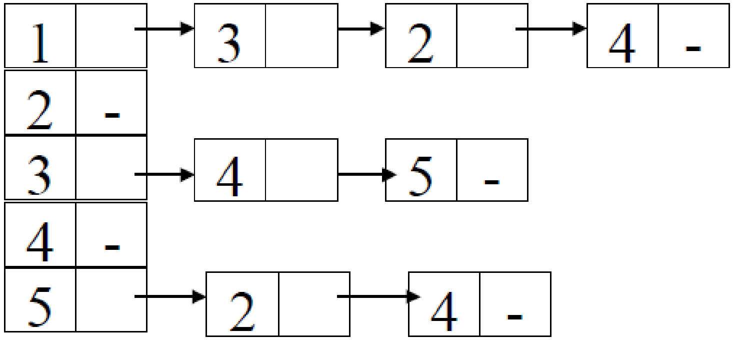
本章小结

- 基本概念：图，无向图，有向图，边带权=>网络
- 图的存储：
  - 邻接矩阵，邻接表（逆邻接表）
  - 通过存储表示，顶点入度、出度计算
- 图的遍历：
  - DFS(深度优先)、BFS(广度优先)
  - 不同存储表示（邻接矩阵、邻接表）下的实现算法。
- 最小生成树：
  - Prim算法和Kruskal算法
- 拓扑排序与关键路径的定义，基本算法实现
- 最短路径：
  - Dijkstra算法

练习

- 已知一个有向图的邻接表存储结构如图所示，根据有向图的深度优先遍历算法,从顶点v1 出发,所得的顶点序列是( )，按广度优先遍历算法得到序列( )。

A. v1,v2,v3,v5,v4
B. v1,v2,v3,v4,v5
C. v1,v3,v4,v5,v2
D. v1,v4,v3,v5,v2
E. v1,v3,v2, v4,v5



练习

- 下列哪一种图的邻接矩阵是对称矩阵？（）

  A．无向图　　B．有向图

  C．AOV网　　D．AOE网

- 一个有n个结点的图，最少有（）个连通分量。

  A．0　　B．1

  C．n-1　　D．n

练习

- 设计一个算法判断无向图G是否是一棵树?
  - 若是树，返回true；否则，返回false

- 解题思路
  - 图G是一棵树的条件是G必须是无回路的连通图或者是有n-1条边的连通图。
  - 对于连通的判断可用图的遍历算法实现；
  - 对于回路的判断，可用图的拓扑排序算法实现。