图

大连理工大学

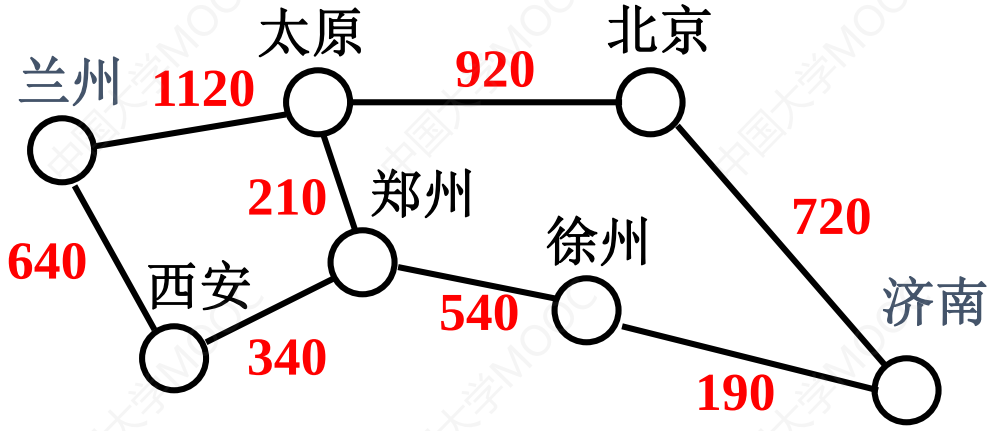
刘 馨 月

主要内容

- 图的基本概念
- 图的存储
- 图的遍历
- 最小生成树
- 最短路径
- 关键路径

最短路径

最短路径



旅客希望停靠站越少越好，则应选择

济南——北京——太原——兰州

旅客考虑的是旅程越短越好，则应选择

济南——徐州——郑州——西安——兰州

最短路径

- 单源最短路径（Dijstra算法）
- 任意点对之间的最短路径（Floyd算法）

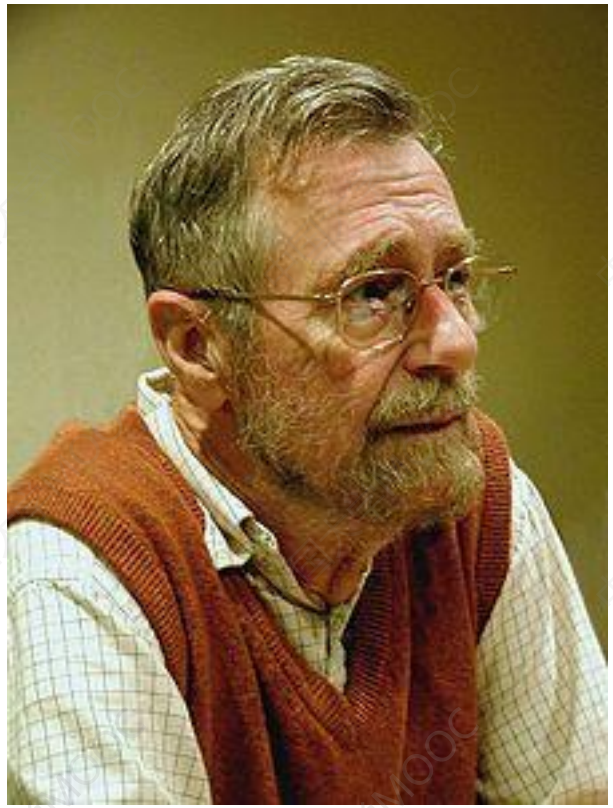
单源最短路径

- 给定一个带权图G = <V，E>，其中每条边（v_i，v_j）上的权W[v_i，v_j]是一个非负实数。另外，给定V中的一个顶点s充当源点。

- 现在要计算从源点s到所有其他各顶点的最短路径，这个问题通常称为单源最短路径(single-source shortest paths)问题。

单源最短路径

- **Dijkstra算法：是由E.W.Dijkstra提出的一种按路径长度递增的次序产生到各顶点最短路径的贪心算法。**



- **E.W.Dijkstra** [Dijkstra中的j不发音，dɛɪkstra]：
  - 1930年5月11日出身于the Netherlands Rotterdam.
  - 1972年获得图灵奖（对开发ALGOL做出了重要贡献）
  - 提出不要使用goto语句
  - 去世于2002年8月6日于Nuenen, the Netherlands.

Dijkstra算法

- 如果v_0到v_j的最短路径为：

  $v_0 \rightarrow v_{x1} \rightarrow v_{x2} \rightarrow v_{xn} \rightarrow v_i \rightarrow v_j$

- 则 $v_0 \rightarrow v_{x1} \rightarrow v_{x2} \rightarrow v_{xn} \rightarrow v_i$是$v_0$到$v_i$的最短路径。

- 按长度顺序产生最短路径时，下一条最短路径总是由一条已产生的最短路径加上一条边形成。

Dijkstra算法基本思想

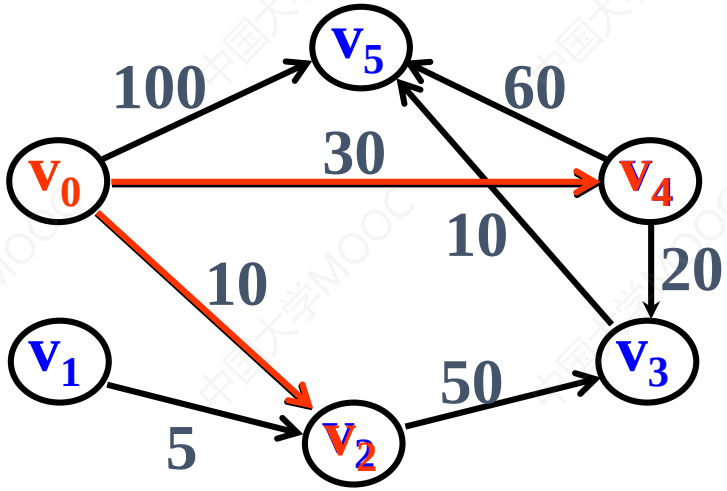
Dijkstra算法

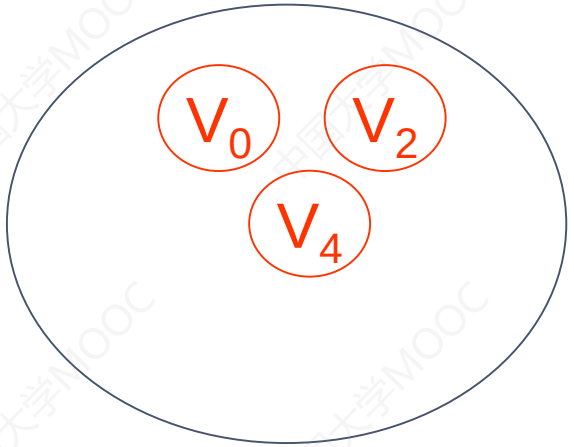
- 集合S表示最短距离已经确定的顶点集。
- 其余的顶点放在另一个集合V-S中。
- 数组D来记录当前所找到的从源点s到每个顶点的最短特殊路径长度。
- 从尚未确定最短路径长度的集合V-S中取出一个最短特殊路径长度最小的顶点u，将u加入集合S，同时修改数组D中由s可达的最短路径长度。

Dijkstra算法示例

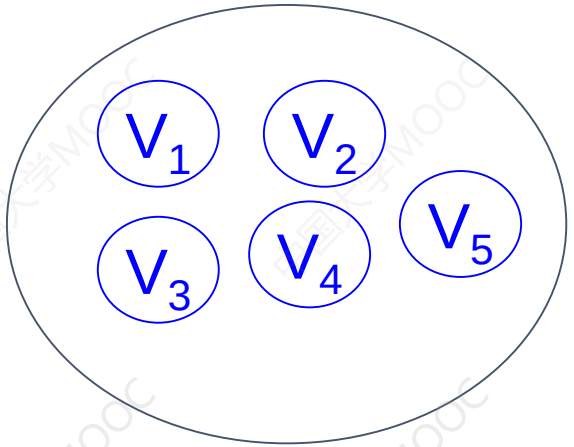
Dijkstra算法

例，





S



V-S

Dijkstra算法基本步骤

Dijkstra算法

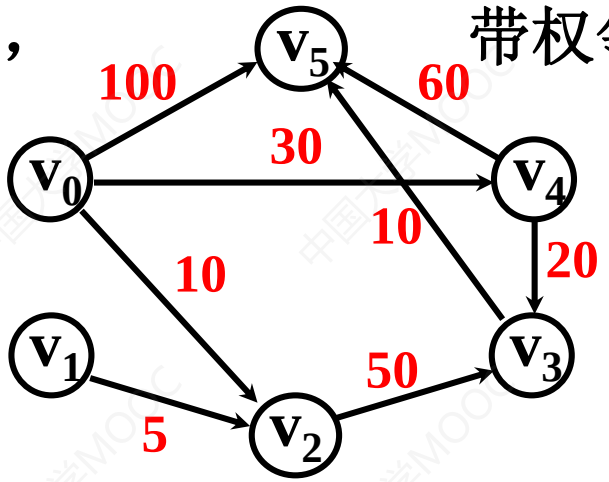
D的初始状态为：如果从源点s到顶点v有弧则D[v]记为弧的权值；否则将D[v]置为无穷大。

每次从尚未确定最短路径长度的集合V-S中取出一个最短特殊路径长度最小的顶点u，将u加入集合S，同时修改数组D中由s可达的最短路径长度：若加进u做中间顶点，使得v_i的最短特殊路径长度变短，则修改v_i的距离值（即当$D[u] + W[u, v_i] < D[v_i]$时，令$D[v_i] = D[u] + W[u, v_i]$）。

然后重复上述操作，一旦S包含了所有V中的顶点，D中各顶点的距离值就记录了从源点s到该顶点的最短路径长度。

Dijkstra算法实例

Dijkstra算法 例，



带权邻接矩阵

	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞			

顶点 \ S		$\{v_2\}$	$\{v_2, v_4\}$	$\{v_2, v_4, v_3\}$	
v_1	∞	∞	∞	∞	
v_2	10				
v_3	∞	60	50		
v_4	30	30			
v_5	100	100	90	60	
最短路径	v_0v_2	v_0v_4	$v_0v_4v_3$	$v_0v_4v_3v_5$	
新顶点	v_2	v_4	v_3	v_5	v_1
路径长度	10	30	50	60	∞

每次修改都用的是最新加入集合 S 的顶点

Dijkstra算法实现

其中的Dist类可以如下定义：

```
Class Dist{
int length;    //与源点s的距离
int pre;       //前面的顶点
};
```

而minVertex()函数可用最小堆（Minheap）等方式实现。

Dijkstra算法实现

```
void Dijkstra(Graph& G,int s, Dist* &D) {
    D=new Dist[G.VerticesNum()];
    //初始化Mark数组、D数组
    //minVertex函数中会用到Mark数组的信息
   for(int i=0;i<G.VerticesNum();i++){
     G.Mark[i]=UNVISITED;
     D[i].length= INFINITY;
     D[i].pre=s;
   }
   D[s].length=0;
```

Dijkstra算法实现

```
for(i=0;i<G.VerticesNum();i++){
  int v=minVertex(G,D);     //找到距离s最小的顶点
  if(D[v].length==INFINITY)
    break;
  G.Mark[v]=VISITED;        //把该点加入已访问组
  Visit(G,v);               //打印输出
```

Dijkstra算法实现

```
// 刷新D数组，因为v的加入D的值需要改变，只要刷新与v相邻的点的值
   for(Edge e=G.FirstEdge(v);G.IsEdge(e);e=G.NextEdge(e)){
       if(D[G.ToVertex(e)].length>(D[v].length+G.Weight(e))){
          D[G.ToVertex(e)].length=D[v].length+G.Weight(e);
          D[G.ToVertex(e)].pre=v;
   }
  }
 }
}
```

Dijkstra算法

- 注意该算法要求图中不存在负权回路。
- 空间复杂度取决于存储方式，邻接矩阵为$O(n^2)$。
- 时间复杂度取决于求权值最小边的方式：
  - 采用最小堆来选择权值最小的边，那么每次改变最短特殊路径长度时需要对堆进行一次重排，此时的时间复杂度为$O((n+e)\log e)$，适合于稀疏图。
  - 通过直接比较D数组元素，确定代价最小的边就需要总时间$O(n^2)$；取出最短特殊路径长度最小的顶点后，修改最短特殊路径长度共需要时间$O(e)$，因此共需要花费$O(n^2)$的时间，这种方法适合于稠密图。

图

大连理工大学

刘 馨 月