

# 第七讲 图（中）

浙江大学 陈 越

# 7.1 最短路径问题



# 最短路径问题的抽象

网络:带权的图

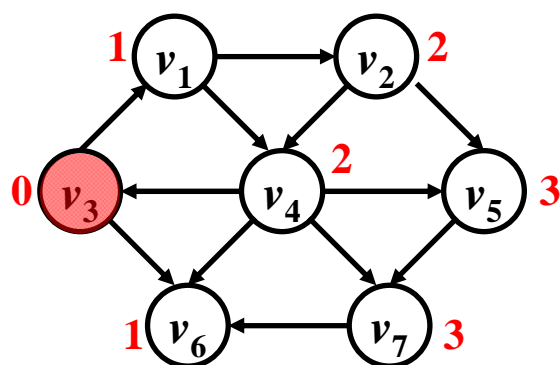
- 在网络中, 求两个不同顶点之间的所有路径中, 边的权值之和最小的那一条路径
  - 这条路径就是两点之间的**最短路径** (**Shortest Path**)
  - 第一个顶点为**源点** (**Source**)
  - 最后一个顶点为**终点** (**Destination**)

# 问题分类

- **单源**最短路径问题：从某固定源点出发(源点是固定的)，求其到所有其他顶点的最短路径
  - (有向) 无权图
  - (有向) 有权图
- **多源**最短路径问题：求任意两顶点间的最短路径

# 无权图的单源最短路算法

- 按照**递增（非递减）**的顺序找出到各个顶点的最短路



**0:**  $v_3$

**1:**  $v_1$  and  $v_6$

**2:**  $v_2$  and  $v_4$

**3:**  $v_5$  and  $v_7$

## BFS !

James Bond 从孤岛跳上岸，最少需要跳多少步？

# 无权图的单源最短路算法

```
void BFS ( Vertex S )
{ visited[S] = true;
  Enqueue(S, Q);
  while(!IsEmpty(Q)){
    V = Dequeue(Q);
    for ( V 的每个邻接点 W )
      if ( !visited[W] ) {
        visited[W] = true;
        Enqueue(W, Q);
      }
  }
}
```

```
void Unweighted ( Vertex S )
{ Enqueue(S, Q);
  while(!IsEmpty(Q)){
    V = Dequeue(Q);
    for ( V 的每个邻接点 W )
      if ( dist[W]==-1 ) {
        dist[W] = dist[V]+1;
        path[W] = V;
        Enqueue(W, Q);
      }
  }
}
```

$T = O(|V| + |E|)$

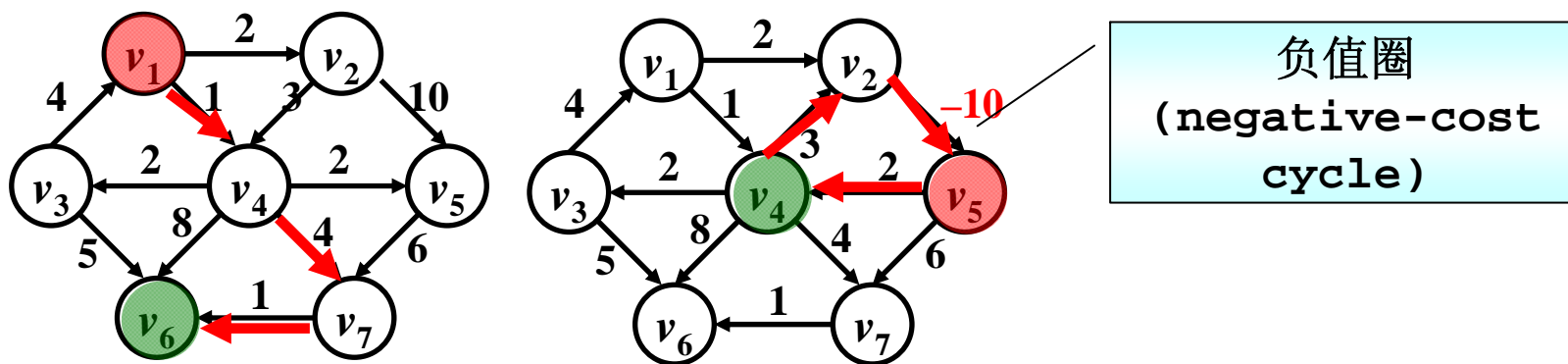
$\text{dist}[W]$  = s到w的最短距离

$\text{dist}[S]$  = 0

$\text{path}[W]$  = s到w的路上经过的某顶点



# 有权图的单源最短路算法



- 按照递增的顺序找出到各个顶点的最短路

## Dijkstra 算法



# 有权图的单源最短路算法

## ■ Dijkstra 算法

- 令  $S = \{\text{源点 } s + \text{已经确定了最短路径的顶点 } v_i\}$
- 对任一未收录的顶点  $v$ ，定义  $\text{dist}[v]$  为  $s$  到  $v$  的最短路径长度，但该路径仅经过  $S$  中的顶点。即路径  $\{s \rightarrow (v_i \in S) \rightarrow v\}$  的最小长度
- 若路径是按照递增（非递减）的顺序生成的，则
  - 真正的最短路必须只经过  $S$  中的顶点（为什么？）
  - 每次从未收录的顶点中选一个  $\text{dist}$  最小的收录（贪心）
  - 增加一个  $v$  进入  $S$ ，可能影响另外一个  $w$  的  $\text{dist}$  值！
    - $\text{dist}[w] = \min\{\text{dist}[w], \text{dist}[v] + \langle v, w \rangle \text{的权重}\}$

# 有权图的单源最短路算法

```
void Dijkstra( Vertex s )
{ while (1) {
    v = 未收录顶点中dist最小者;
    if ( 这样的v不存在 )
        break;
    collected[V] = true;
    for ( v 的每个邻接点 w )
        if ( collected[W] == false )
            if ( dist[V]+E<V,W> < dist[W] ) {
                dist[W] = dist[V] + E<V,W> ;
                path[W] = V;
            }
    }
}
```

/\* 不能解决有负边的情况 \*/

$T = O(?)$

# 有权图的单源最短路算法

- 方法1: 直接扫描所有未收录顶点 –  $O(|V|)$

- $T = O(|V|^2 + |E|)$  ——— 对于稠密图效果好

- 方法2: 将`dist`存在最小堆中 –  $O(\log|V|)$

- 更新`dist[w]`的值 –  $O(\log|V|)$

- $T = O(|V| \log|V| + |E| \log|V|) = O(|E| \log|V|)$

对于稀疏图效果好

# 多源最短路算法

- 方法1: 直接将单源最短路算法调用 $|V|$ 遍

- $T = O(|V|^3 + |E| \times |V|)$  ———— 对于稀疏图效果好

- 方法2: **Floyd 算法**

- $T = O(|V|^3)$  ———— 对于稠密图效果好

# 多源最短路算法

## ■ Floyd 算法

- $D^k[i][j]$  = 路径  $\{ i \rightarrow \{ l \leq k \} \rightarrow j \}$  的最小长度
- $D^0, D^1, \dots, D^{|V|-1}[i][j]$  即给出了  $i$  到  $j$  的真正最短距离
- 最初的  $D^{-1}$  是什么？
- 当  $D^{k-1}$  已经完成，递推到  $D^k$  时：
  - 或者  $k \notin$  最短路径  $\{ i \rightarrow \{ l \leq k \} \rightarrow j \}$ ，则  $D^k = D^{k-1}$
  - 或者  $k \in$  最短路径  $\{ i \rightarrow \{ l \leq k \} \rightarrow j \}$ ，则该路径必定由两段最短路径组成： $D^k[i][j] = D^{k-1}[i][k] + D^{k-1}[k][j]$

# 多源最短路算法

```
void Floyd()  
{  
    for ( i = 0; i < N; i++ )  
        for( j = 0; j < N; j++ ) {  
            D[i][j] = G[i][j];  
            path[i][j] = -1;  
        }  
    for( k = 0; k < N; k++ )  
        for( i = 0; i < N; i++ )  
            for( j = 0; j < N; j++ )  
                if( D[i][k] + D[k][j] < D[i][j] ) {  
                    D[i][j] = D[i][k] + D[k][j];  
                    path[i][j] = k;  
                }  
}
```

$T = O(|V|^3)$