

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

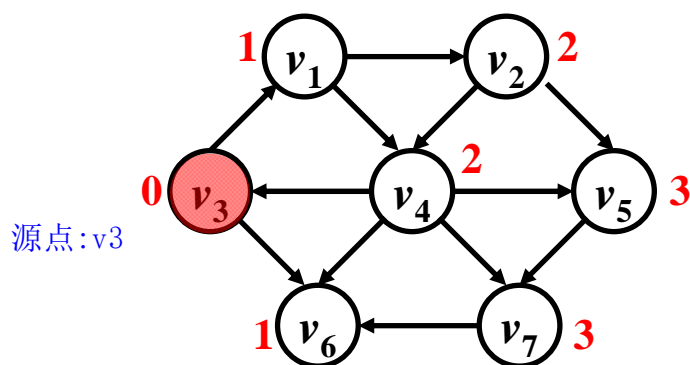
- 在网络中, 求两个不同顶点之间的所有路径中, 边的权值之和最小的那一条路径
 - 这条路径就是两点之间的**最短路径** (**Shortest Path**)
 - 第一个顶点为**源点** (**Source**)
 - 最后一个顶点为**终点** (**Destination**)

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**（如果有并列数字的话：**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 此时 v_3, v_4, v_6 已经被访问过了

到这里所有节点都已经被访问过了, 算法也就结束了

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

/*这边用的是邻接表存储*/

```
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 ) /*相当于每条边都访问了一次*/
      if ( dist[W]==-1 ) {
        dist[W] = dist[V]+1;
        path[W] = V;
        Enqueue(W, Q);
      }
  }
}
```

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

/*每次从里面弹出一个顶点,当一个顶点被弹出来的时候,就意味着这个顶点到s的最短路已经被找到了*/

$dist[W]$ = s到w的最短距离

$dist[S] = 0$

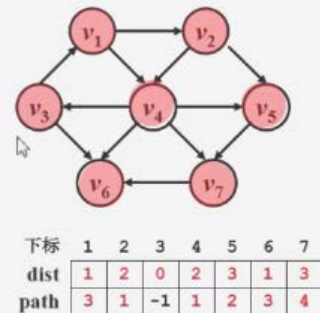
程序一开始时,将源点定义为0

$path[W]$ = s到w的路上一定会经过的某顶点 $path[]$ 记录路径

无权图的单源最短路算法

中国

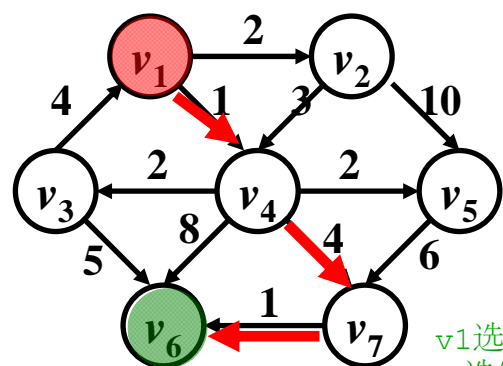
```
void Unweighted ( Vertex S )
{ Enqueue(S, Q);
  while(!IsEmpty(Q)){
    V = Dequeue(Q);
    for ( V 的每个邻接点 W )
      if ( dist[W]==-1 ) {
        dist[W] = dist[V]+1;
        path[W] = V;
        Enqueue(W, Q);
      }
  }
}
```



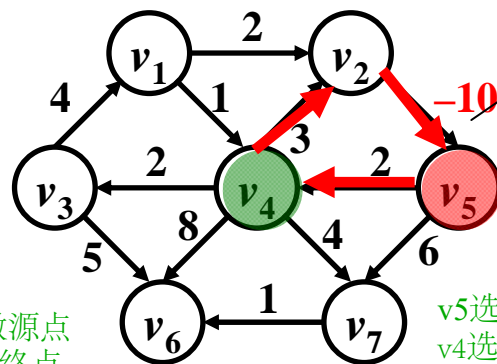
/*dist[]数组里面存的是每一个顶点到源点的最短距离*/

Unweighted(3)

有权图的单源最短路算法



v_1 选做源点
 v_6 选做终点



v_5 选做源点
 v_4 选做终点

一直在这条路上循环,就会不断的变小,最终为负无穷-->后面讨论不考虑该情况

负值圈
(negative-cost
cycle)

□ 按照**递增 (非递减)**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

与BFS相似的地方在于把顶点一个一个往集合里面收的

- 令顶点集合 $S = \{\text{源点 } s + \text{已经确定了最短路径的顶点 } v_i\}$
- 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 中已经收集的顶点。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？反证法）
 - 每次从未收录的顶点中选一个 dist 最小的收录集合当中（贪心算法）
 - 增加一个 v 进入 S ，可能影响另外一个 w 的 dist 值！（如果收录 v 使得 s 到 w 的路径变短，则： s 到 w 的路径一定经过 v ，并且 v 到 w 有一条边， w 一定是 v 的邻接点， v 增加进集合以后，它能影响的是它一圈邻接点）
 - $\text{dist}[w] = \min\{\text{dist}[w], \text{dist}[v] + \langle v, w \rangle \text{ 的权重}\}$

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 */

$$T = O(?)$$

Dijkstra算法中的dist应该如何初始化?
如果s到w有直接的边, 则dist[w]=<s,w>的权重; 否则dist[w]定义为正无穷

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

稀疏图: E 和 V 是同一个数量级的

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;  
                }  
}
```

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