第七讲 图（中）

浙江大学 陈 越

7.1 最短路径问题


轨道交通7号线
轨道交通11号线
轨道交通3号线
轨道交通4号线
轨道交通6号线
轨道交通8号线
轨道交通9号线
轨道交通5号线
轨道交通10号线
世博园区轨道交通专用线
嘉定西站
白银路站
嘉定新城站
马陆站
南翔站
桃浦新村站
武威路站
祁连山路站
李子园站
上海西站站
真如站
枫桥路站
上海汽车城站
上海赛车场站
虹桥2号航站楼站
虹桥国际机场
淞虹路站
北新泾站
威宁路站
娄山关路站
中山公园站
金沙江路站
曹杨路站
镇坪路站
中潭路站
上海火车站
长寿路站
降德路站
江苏路站
汉中路站
昌平路站
静安寺站
南京西路站
人民广场站
上大路站
场中路站
大场镇站
行知路站
大华三路站
新村路站
岚皋路站
汶水路站
上海马戏城站
延长路站
中山北路站
中兴路站
上海火车站
新闸路站
曲阜路站
天潼路站
四川北路站
宝山路站
海伦路站
临平路站
大连路站
杨树浦路站
浦东大道站
东宝兴路站
邮电新村站
曲阳路站
四平路站
鞍山新村站
江浦路站
黄兴路站
延吉中路站
黄兴公园站
翔殷路站
嫩江路站
市光路站
外高桥
友谊西路站
安公路站
共富新村站
呼兰路站
通河新村站
共康路站
彭浦新村站
西藏北路站
虹口足球场站
宝杨路站
水产路站
淞滨路站
张华浜站
淞发路站
长江南路站
殷高西路站
江湾镇站
大柏树站
赤峰路站
新江湾城站
殷高东路站
三门路站
江湾体育场站
五角场站
国权路站
同济大学站
延安西路站
虹桥路站
交通大学站
上海图书馆站
常熟路站
陕西南路站
黄陂南路站
大世界站
豫园站
南京东路站
陆家嘴站
东昌路站
源深体育中心站
民生路站
北洋泾路站
德平路站
云山路站
金桥路站
世纪大道站
商城路站
浦电路站
新天地站
老西门站
马当路站
小南门站
陆家浜路站
西藏南路站
南浦大桥站
塘桥站
蓝村路站
浦电路站
世纪公园站
上海科技馆站
杨高中路站
花木路站
紫藤路站
龙柏新村站
龙溪路站
水城路站
伊犁路站
宋园路站
衡山路站
肇嘉浜路站
徐家汇站
上海体育馆站
宜山路站
嘉善路站
打浦桥站
鲁班路站
东安路站
上海体育场站
大木桥路站
卢浦大桥站
世博园区
黄浦江
泗泾站
九亭站
中春路站
七宝站
星中路站
合川路站
漕河泾开发区站
桂林路站
上海南站站
漕宝路站
漕溪路站
龙漕路站
石龙路站
锦江乐园站
莲花路站
外环路站
莘庄站
上海南站
春申路站
银都路站
船厂路站
后滩站
长清路站
世博大道站
耀华路站
成山路站
杨思站
云台路站
凌兆新村站
芦恒路站
灵岩南路站
上南路站
上海儿童医学中心站
临沂新村站
高科西路站
东明路站
高青路站
华夏西路站
杨高南路站
锦绣路站
芳华路站
龙阳路站
张江高科站
金科路站
广兰路站
磁浮
松江大学城站
松江新城站

最短路径问题的抽象

网络:带权的图

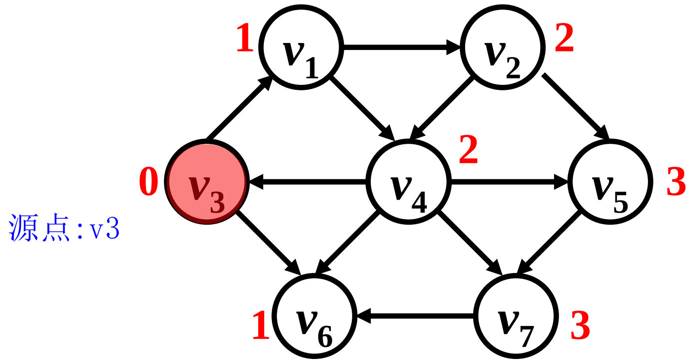
- 在网络中，求两个不同顶点之间的所有路径中，边的权值之和最小的那一条路径
  - 这条路径就是两点之间的最短路径（**Shortest Path**）
  - 第一个顶点为源点（**Source**）
  - 最后一个顶点为终点（**Destination**）

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



路径长度　从源点v3出发

0: ☞ v_3

1: ☞ v_1 **and** v_6

2: ☞ v_2 **and** v_4

3: ☞ v_5 **and** v_7 此时v3, v4, v6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 )
      if ( dist[W]==-1  ) {
        dist[W] = dist[V]+1;
        path[W] = V;
        Enqueue(W, Q);
      }
  }
}
```

/*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*/

/*相当于每条边都访问了一次*/

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

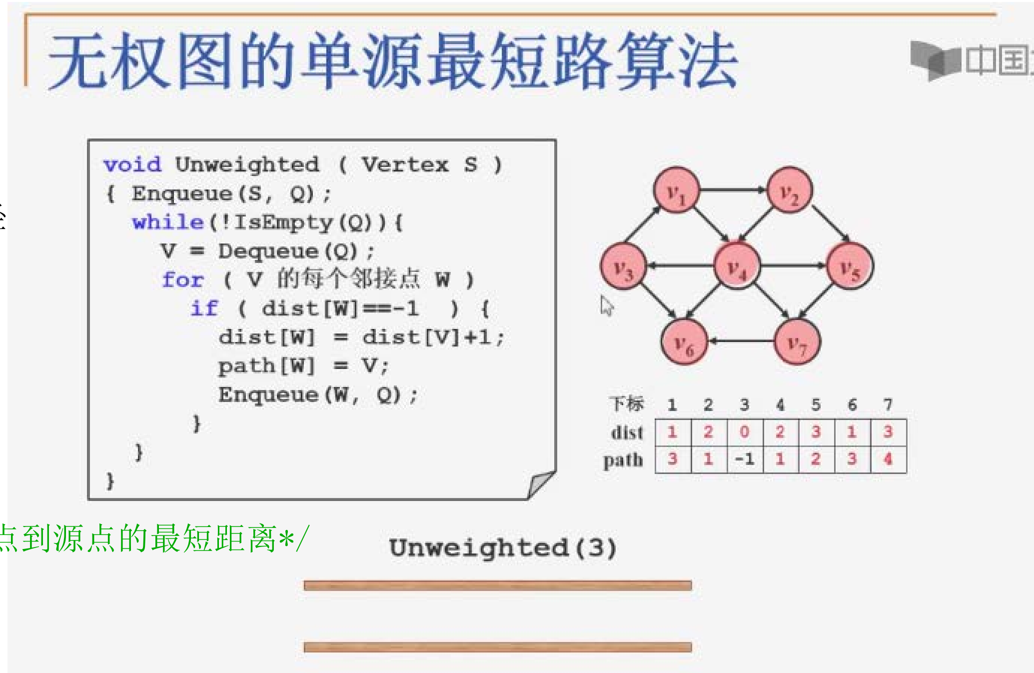
dist[W] = S到W的最短距离

dist[S] = 0

程序一开始时,将源点定义 为0

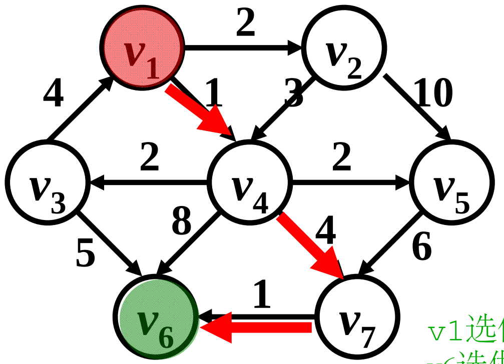
path[W] = S到W的路上一定会经过的某顶点

path[]记录路径

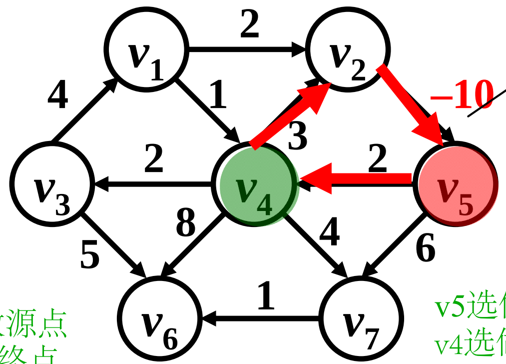


/*dist[]数组里面存的是每一个顶点到源点的最短距离*/

有权图的单源最短路算法



v1选做源点
v6选做终点



负值圈
(negative-cost cycle)

v5选做源点
v4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

- 按照递增(非递减)的顺序找出固定的源点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={源点s + 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→(v_i∈S)→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）
      - dist[w] = min{dist[w], dist[v] + <v,w>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;
        }
  }
  /* Dijkstra算法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算法是一个经典的动态规划算法。用通俗的语言来描述的话，首先我们的目标是寻找从点i到点j的最短路径。从动态规划的角度看问题，我们需要为这个目标重新做一个诠释（这个诠释正是动态规划最富创造力的精华所在）

从任意节点i到任意节点j的最短路径不外乎2种可能，1是直接从i到j，2是从i经过若干个节点k到j。所以，我们假设Dis(i,j)为节点u到节点v的最短路径的距离，对于每一个节点k，我们检查Dis(i,k) + Dis(k,j) < Dis(i,j)是否成立，如果成立，证明从i到k再到j的路径比i直接到j的路径短，我们便设置Dis(i,j) = Dis(i,k) + Dis(k,j)，这样一来，当我们遍历完所有节点k，Dis(i,j)中记录的便是i到j的最短路径的距离。

2).算法描述:

a.从任意一条单边路径开始。所有两点之间的距离是边的权，如果两点之间没有边相连，则权为无穷大。

b.对于每一对顶点 u 和 v，看看是否存在一个顶点 w 使得从 u 到 w 再到 v 比已知的路径更短。如果是更新它。

- **Floyd 算法** 用邻接矩阵来记录这个稠密图
  - **$D^k[i][j]$ = 路径{ i → { $l \le k$ } → j }的最小长度**
  - **D^0, D^1, …, $D^{|V|-1}[i][j]$即给出了i到j的真正最短距离**
  - **最初的D^{-1}是什么？** D矩阵应该初始化为什么？直接定义为带权的邻接矩阵，对角元是0 如果i和j之间没有直接的边，D[i][j]应该定义为正无穷
  - **当D^{k-1}已经完成，递推到D^k时：**
    - 或者$k \notin$最短路径{ i → { $l \le k$ } → j }，则$D^k = D^{k-1}$ 新收进来的k不影响i到j的最短路径，则保持原样就行
    - 或者$k \in$最短路径{ i → { $l \le k$ } → j }，则该路径必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]; /*D[i][j]初始化为邻接矩阵*/
       path[i][j] = -1;   /*path[]数组用来记录路径*/
     }
   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;
         }
}
```

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