

图应用

Floyd-Warshall算法

11-F

让我们测量一下自己的活动半径，并待在那个中心吧，
就像蜘蛛待在网的中心一样。

邓俊辉

deng@tsinghua.edu.cn

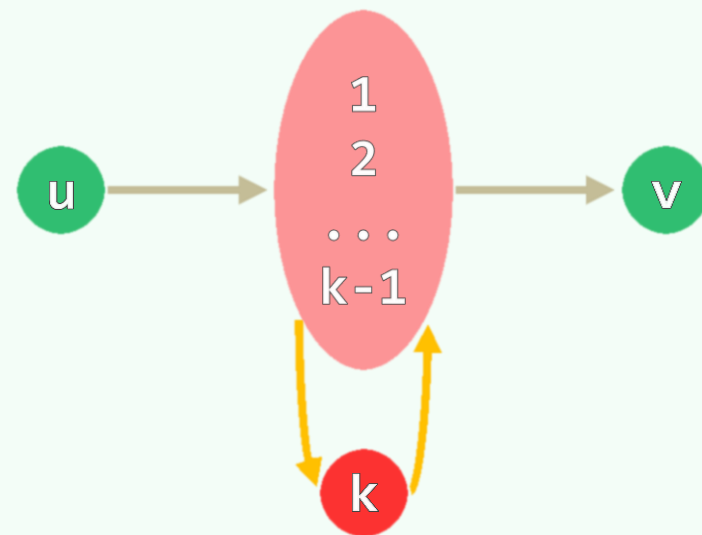
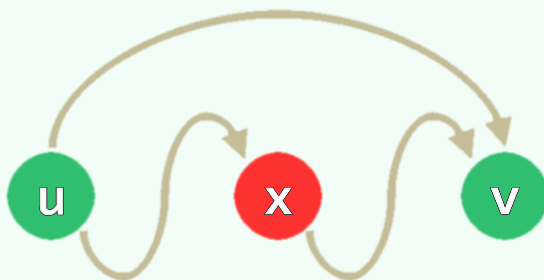
从Dijkstra到Floyd-Warshall

- ❖ 给定带权网络 G ，计算其中**所有点对**之间的最短距离
- ❖ 应用：确定 G 的中心点 (center)
 - $\text{radius}(G, s) = s$ 的半径 = 所有顶点到 s 的最大距离
 - 中心点 = 半径最小的顶点 s
- ❖ 直觉：依次将各顶点作为源点，调用Dijkstra算法
 - 时间 = $n \times O(n^2) = O(n^3)$ —— 可否更快？
- ❖ 思路：图矩阵 ~ 最短路径矩阵
- ❖ 效率： $O(n^3)$ ，与执行 n 次Dijkstra相同 —— 既如此，为何还要用F.W.？
- ❖ 优点：形式简单、算法紧凑、便于实现；允许**负权边**（尽管仍不能有**负权环路**）

问题 + 特点

❖ u 和 v 之间的最短路径可能是

- 不存在通路, 或者
- 直接连接, 或者
- 最短路径(u, x) + 最短路径(x, v)



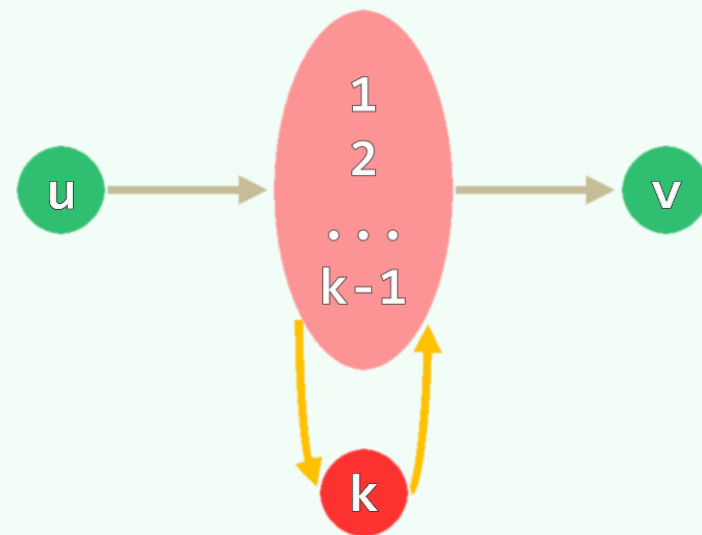
❖ 将所有顶点随意编号: $1, 2, \dots, n$

定义: $d^k(u, v)$ = 中途只经过前 k 个顶点中转, 从 u 通往 v 的最短路径长度

- $d^k(u, v) = w(u, v)$ (if $k = 0$)
- $d^k(u, v) = \min\{ d^{k-1}(u, v), d^{k-1}(u, k) + d^{k-1}(k, v) \}$ (if $k \geq 1$)

蛮力递归

```
weight dist( node * u, node * v, int k )  
  
    if ( k < 1 ) return w( u, v );  
  
    u2v = dist( u, v, k-1 ); //经前k-1个点中转  
  
    for each node x  $\notin$  { u, v } //x作为第k个可中转点  
        u2x2v = dist( u, x, k-1 )  
                + dist( x, v, k-1 ); //递归  
  
    u2v = min( u2v, u2x2v ); //择优  
  
    return u2v;
```



❖ 存在大量**重复**的递归调用，如何**避免**？

❖ 动态规划之**记忆化**：

维护一张表，记录需要**反复计算**的数值

动态规划

```
for k in range(0, n)
  for u in range(0, n)
    for v in range(0, n)
      A[u][v] =
        min( A[u][v], A[u][k] + A[k][v] )
```

