6.图

(xc) Floyd-Warshall算法

邓俊辉

deng@tsinghua.edu.cn

多起点：从Dijkstra到Floyd-Warshall

- 给定图G，计算其中 $\boxed{\text{所有点对}}$ 之间的最短距离
- 应用：搜索图G的中心点（center vertex）

  s中心的半径radius(G, s) = 所有顶点到s的最大距离

  中心点 = 半径最小的顶点s
- Dijkstra：依次将各顶点作为源点，调用Dijkstra算法

  时间 = $n \times \mathcal{O}(n^2) = \mathcal{O}(n^3)$ —— 可否更快？
- 思路： 图矩阵 ~ 最短路径矩阵
- 效率： $\mathcal{O}(n^3)$，与执行n次Dijkstra相同 —— 既如此，为何还要用FW？
- 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）

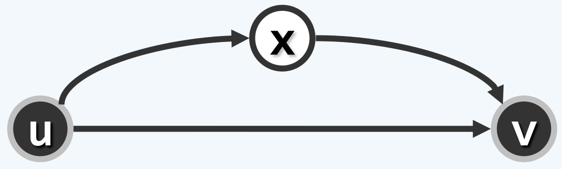
多起点：问题特点

- u和v之间的最短路径可能是

  0) 不存在通路，或者

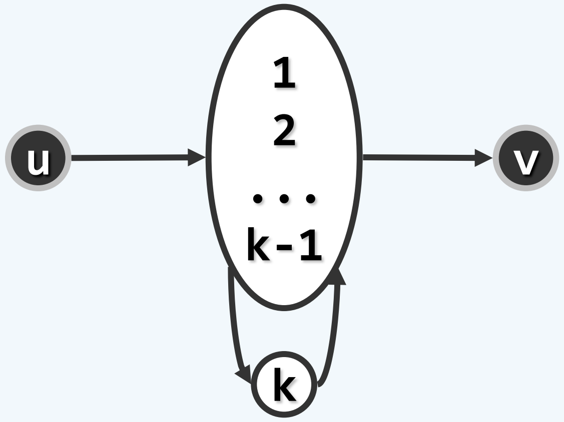
  1) 直接连接，或者

  2) 最短路径(u, x) + 最短路径(x, v)



- 将所有顶点随意编号：

  1, 2, ..., n



- 定义：$d^k(u,v)$

  = 中途只经过 前k个 顶点、联接u和v的最短路径长度

  = $w(u,v)$　　(if $k = 0$)

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

多起点：递归

```
weight dist( node* u, node* v, int k ) { //蛮力递归
   if ( k < 1 ) return w( u, v ); //递归基：中途不经过任何点
   minDist  =  dist( u, v, k - 1 ); //递归：中途可经过前 k-1 个点
   foreach ( node x ∈ {u, v} ) { //枚举其余各点，分别作为第 k 个点
      u2x2vDist  =  dist( u, x, k - 1 )  +  dist( x, v, k - 1 ); //递归
      minDist  =  min( minDist, u2x2vDist ); //优化
   }
   return minDist;
}
```

动态规划

- 复杂度： $T(n,\ 0) = 1$

  $T(n,\ k) = (n-2) \times 2 \times T(k - 1) = 2^k \times (n-2)^k$

  $T(n,\ n-2) = 2^{n-2} \times (n-2)^{n-2}$

  $= \mathcal{O}(n^n)$ //这还仅仅只是一对节点所需的时间

- 注意：蛮力算法中，存在大量的 $\boxed{\text{重复}}$ 递归调用

- 能否避免这些重复计算？如何避免？你应该还记得...

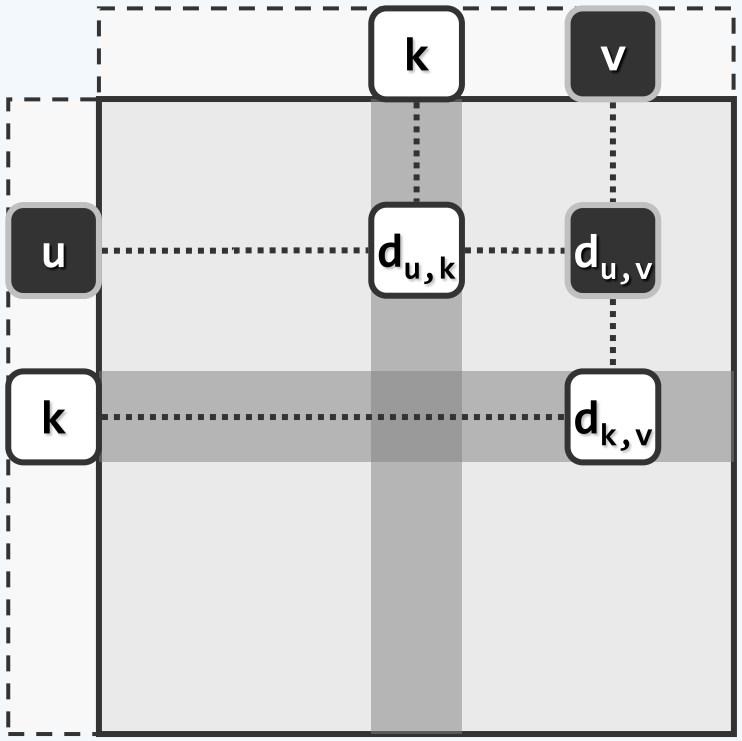
- 动态规划（dynamic programming）

  维护一张表，记录需要 $\boxed{\text{反复计算}}$ 的数值 //如此，只需计算一次

算法

```
// Initialization
for u = 1 to n
for v = 1 to n
   { dist[u][v] = w[u][v]; midV[u][v] = 0; }

// Iteration
for k = 1 to n
for u = 1 to n
for v = 1 to n
   if ( dist[u][v]  >  dist[u][k]  +  dist[k][v] )
      { dist[u][v]  =  dist[u][k]  +  dist[k][v];  midV[u][v]  =  k; }
```



复杂度

- 时间

  $\mathcal{O}(n^2) + \mathcal{O}(n^3) = \mathcal{O}(n^3)$

  与n次调用Dijkstra相同

- 空间

  存储一张n×n的表格，$\mathcal{O}(n^2)$

  每个单元为两个整数

- 对于稀疏图和稠密图，你会分别选择哪种算法？

- 根据midV[][]矩阵

  如何重构出u和v之间的最短路径？需要多长时间？

- 可在$\mathcal{O}(n)$时间内完成 //具体算法...