最短路

kodylau@hit

最短路问题

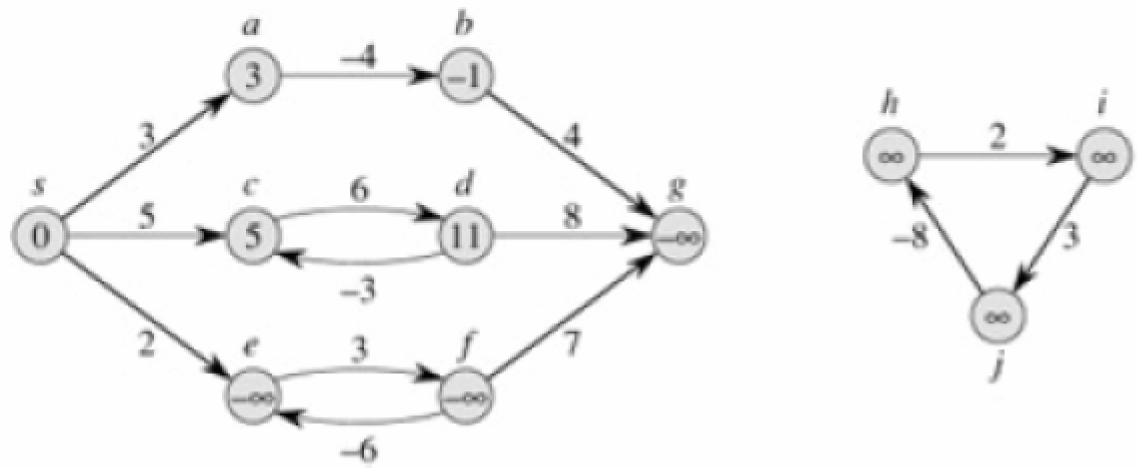
- 最短路径问题是图论研究中的一个经典算法问题， 旨在寻找图（由结点和路径组成的）中两结点之间的最短路径。算法具体的形式包括：
- **确定起点的最短路径问题** - 即已知起始结点，求最短路径的问题。
- **确定终点的最短路径问题** - 与确定起点的问题相反，该问题是已知终结结点，求最短路径的问题。在无向图中该问题与确定起点的问题完全等同，在有向图中该问题等同于把所有路径方向反转的确定起点的问题。
- **确定起点终点的最短路径问题** - 即已知起点和终点，求两结点之间的最短路径。
- **全局最短路径问题** - 求图中所有的最短路径。

最短路问题

- 给加权图和一个起点s, 求s到所有点的最短路(SSSP)
- 最短路有环吗

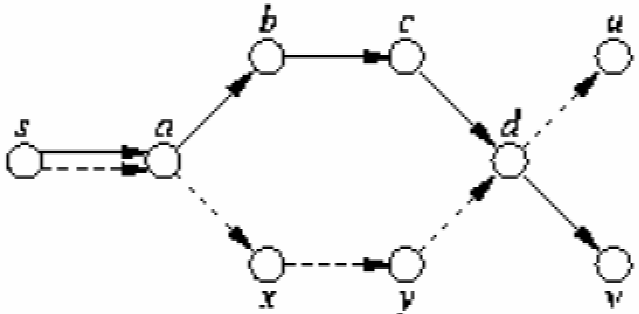
  正环: 何必呢, 删除环则得到更短路

  负环: 无最短路, 因为可以沿负环兜圈子



最优性原理

- 最优性原理: 若最短路u->v经过中间结点w, 则u->w和w->v的路径分别是u到w和w到v的最短路.
- 意义: 贪心、动态规划



If s → a → b → c → d → v and s → a → x → y → d → u are both shortest paths,
then s → a → b → c → d → u is also a shortest path.

最短路算法

- 单源最短路径 SSSP
  - Dijkstra算法
  - Bellman-Ford算法
- 全源最短路
  - Floyd-Warshall算法
  - Johnson算法(自己看，不讲)

松弛操作

- 在松弛一条边(u,v)的过程中，要测试是否可以通过u，对迄今找到的v的最短路径进行改进；如果可以改进的话，则更新d[v]和π[v]。一次松弛操作可以减小最短路径估计的值d[v]，并更新v的前趋域π[v](S到v的当前最短路径中v点之前的一个点的编号)。下面的伪代码对边(u,v)进行了一步松弛操作。
- RELAX(u, v, w)
  - if(d[v]>d[u]+w(u,v))
  - then d[v]←d[u]+w(u,v)
  - π[v]←u

Dijkstra算法

- 基本原理：每次新扩展一个距离最短的点，更新与其相邻的点的距离。当所有边权都为正时，由于不会存在一个距离更短的没扩展过的点，所以这个点的距离永远不会再被改变，因而保证了算法的正确性。不过根据这个原理，用Dijkstra求最短路的图不能有负权边，因为扩展到负权边的时候会产生更短的距离，有可能就破坏了已经更新的点距离不会改变的性质。 与求MST的Prim算法很像。
- 如果用本算法求一个图中全部的最短路，则要以每个点为源调用一次Dijkstra算法。

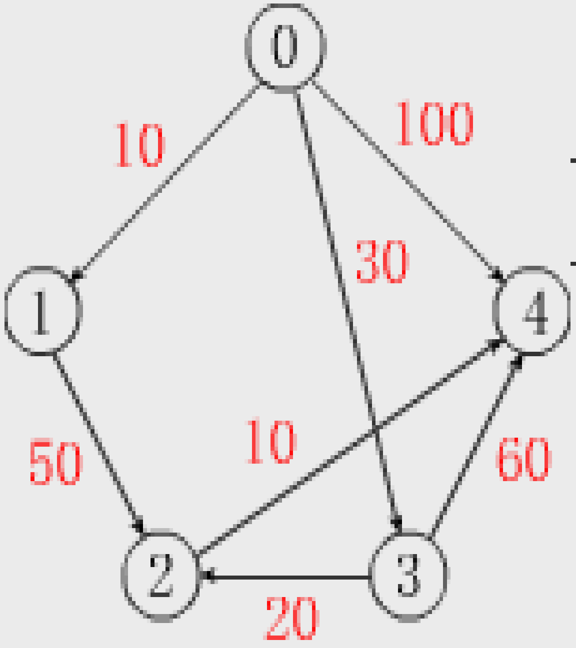
算法流程

- 设s为源，w[u,v]为点u和v之间的边的长度，结果保存在dist[]
- 初始化：源的距离dist[s]设为0，其他的点距离设为无穷大，同时把所有的点的状态设为没有扩展过。
- 循环n-1次：
  - 在没有扩展过的点中取一距离最小的点u，并将其状态设为已扩展。
  - 对于每个与u相邻的点v，执行Relax(u,v)，也就是说，如果dist[u]+w[u,v]<dist[v]，那么把dist[v]更新成更短的距离dist[u]+w[u,v]。此时到点v的最短路径上，前一个节点即为u。
- 结束。此时对于任意的u，dist[u]就是s到u的距离。

算法实现

- **直接实现**
  - 最简单的实现方法就是，在每次循环中，再用一个循环找距离最短的点，然后用任意的方法更新与其相邻的边，**复杂度**显然为O(V^2)
- **二叉堆实现**
  - 使用二叉堆(Binary Heap)来保存没有扩展过的点的距离并维护其最小值，并在访问每条边的时候更新，可以把时间复杂度变成O(V+ElogV)。
  - 当边数远小于点数的平方时，这种算法相对来说有很好的效果。但是当E=O(V^2)时（有时候表现为不限制边的条数），用二叉堆的优化反倒会更慢。因为此时的复杂度是O(V+V^2logV)，大于不用堆的实现的O(V^2)的复杂度。
  - 另外此时要用邻接表保存边，使得扩展边的总复杂度为O(E)，否则复杂度不会减小。

算法演示



有向网G8

G8中从源点0到其余各点的最短路径

原点	中间结点	终点	路径长度
0		1	10
0		2	30
0	3	3	50
0	3. 2	4	60

适用条件与限制

- 有向图和**无向图**都可以使用本算法，无向图中的每条边可以看成相反的两条边。
- 用来求最短路的图中不能存在负权边。

Bellman-Ford算法

- 如有最短路，则每个顶点最多经过一次
  - 这条路不超过*n*-1条边
  - 长度为***k***的路由长度为***k-1***的路增加一条边得到
  - 由最优性原理, 只考虑长度为1…k-1的最短路
  - 算法梗概: 每次迭代依次松弛每条边

复杂度O(VE)

Bellman-Ford算法

- 算法描述
  – 对每条边进行|V|-1次Relax操作;
  – 如果存在(u,v)∈E使得dis[u]+w<dis[v],则存在负权回路;否则dis[v]即为s到v的最短距离。

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| - 1
       do for each edge (u, v) □ E[G]
              do RELAX(u, v, w)
for each edge (u, v) □ E[G]
       do if d[v] > d[u] + w(u, v)
             then return FALSE
return TRUE
```

适用条件&范围

- 单源最短路径(从源点s到其它所有顶点v);
- 有向图&无向图(无向图可以看作(u,v),(v,u)同属于边集E的有向图);
- 边权可正可负(如有负权回路输出错误提示);
- 差分约束系统;

差分约束系统

- 请大家自学！

Floyd-Warshall算法

- Floyd-Warshall 算法用来找出每对点之间的最短距离。它需要用邻接矩阵来储存边，这个算法通过考虑最佳子路径来得到最佳路径。
- 注意单独一条边的路径也不一定是最佳路径。

算法描述

- 从任意一条单边路径开始。所有两点之间的距离是边的权，或者无穷大，如果两点之间没有边相连。
- 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 比己知的路径更短。如果是更新它。

算法描述

- *// dist(i,j) 为从节点i到节点j的最短距离*
- For i←1 to n do
  - For j←1 to n do
    - dist(i,j) = weight(i,j)
- For k←1 to n do *// k为"媒介节点"*
  - For i←1 to n do
    - For j←1 to n do
      - if (dist(i,k) + dist(k,j) < dist(i,j)) then *// 是否是更短的路径?*
        - » dist(i,j) = dist(i,k) + dist(k,j)

Floyd-Warshall算法

- 这个算法的效率是O(*V*^3)。它需要邻接矩阵来储存图。
- 这个算法很容易实现，只要几行。

使用条件&范围

- 通常可以在任何图中使用，包括有向图、带负权边的图。
- 稍作改动即可用来求传递闭包。

第k短路

- 有能力的自学！