Floyd-Warshall算法

Floyd-Warshall算法是解决任意两点间的最短路径的一种算法。

目录 [隐藏]

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。

算法描述

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。

- 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 比已知的路径更短。如果是更新它。
- 不可思议的是，只要按排适当，就能得到结果。

```
// 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
    if (i<>k) then
      For j←1 to n do
        if (i<>j) and(k<>j)then
          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)
```

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。

即使问题是求单源最短路径，还是推荐使用这个算法，如果时间和空间允许(只要有放的下邻接矩阵的空间，时间上就没问题)。

计算每一对顶点间的最短路径（floyd算法） 【例题】设计公共汽车线路(1)

```
  现有一张城市地图，图中的顶点为城市，有向边代表两个城市间的连通关系，边上的权即为距离。
现在的问题是，为每一对可达的城市间设计一条公共汽车线路，要求线路的长度在所有可能的方案里是最短的。
```

输入：

```
n  (城市数，1≤n≤20)
  e  (有向边数1≤e≤210)
  以下e行，每行为边（i,j）和该边的距离wij（1≤i,j≤n)
```

输出：

```
  k行，每行为一条公共汽车线路
```

分析： 本题给出了一个带权有向图，要求计算每一对顶点间的最短路径。这个问题虽然不是图的连通性问题，但是也可以借鉴计算传递闭包的思想： 在枚举途径某中间顶点k的任两个顶点对i和j时，将顶点i和顶点j中间加入顶点k后是否连通的判断， 改为顶点i途径顶点k至顶点j的路径是否为顶点i至顶点j的最短路径（1≤i，j，k≤n）。 显然三重循环即可计算出任一对顶点间的最短路径。设 n—有向图的结点个数； path—最短路径集合。其中path[i，j]为vi至vj的最短路上vj的前趋结点序号(1≤i，j≤n)； adj—最短路径矩阵。初始时为有向图的相邻矩阵

我们用类似传递闭包的计算方法反复对adj矩阵进行运算，最后使得adj成为存储每一对顶点间的最短路径的矩阵

```
            (1≤i，j≤n)
```

Var

```
adj：array[1‥n，1‥n] of real;
path：array[1‥n，1‥n] of 0‥n;
```

计算每一对顶点间最短路径的方法如下：

```
  首先枚举路径上的每一个中间顶点k（1≤k≤n）；然后枚举每一个顶点对（顶点i和顶点j，1≤i，j≤n）。
```

如果i顶点和j顶点间有一条途径顶点k的路径，且该路径长度在目前i顶点和j顶点间的所有条途径中最短，则该方案记入adj[i，j]和path[i，j]

```
          adj矩阵的每一个元素初始化为∞;
          for i←1 to n do                {初始时adj为有向图的相邻矩阵，path存储边信息}
            for j←1 to n do
              if wij<>0 then begin adj[i，j]←wij; path[i，j]←i; end{then}
                       else path[i，j]←0;
          for k←1 to n do                                          {枚举每一个中间顶点}
            for i←1 to n do                                         {枚举每一个顶点对}
              for j←1 to n do
                 if adj[i，k]+adj[k，j]<adj[i，j] {若vi经由vk 至vj的路径目前最优，则记下}
                   then begin
                       adj[i，j]←adj[i，k]+adj[k，j];
                       path[i，j]←path[k，j];
                       end，{then}
```

计算每一对顶点间最短路径时间复杂度为W(n3)。算法结束时，由矩阵path可推知任一结点对i、j之间的最短路径方案是什么

```
       Procedure print(i，j);
        begin
         if i=j then 输出i
            else if path[i，j]=0
                   then 输出结点i与结点j之间不存在通路
                   else begin
                        print (i，path[i，j]);     {递归i顶点至j顶点的前趋顶点间的最短路径}
```

```
                              输出j;
                            end; {else}
       end; {print}
```

由此得出主程序

```
   距离矩阵w初始化为0;
   输入城市地图信息（顶点数、边数和距离矩阵w）;
   计算每一对顶点间最短路径的矩阵path;
   for i←1 to n do                                            {枚举每一个顶点对}
    for j←1 to n do
    if path[i, j]<>0                      {若顶点i可达顶点j, 则输出最短路径方案}
    then begin print (i, j) ; writeln; end; {then}
```

时间复杂度

O(N^3)

改进和优化

用来计算传递封包[需要解释]

计算闭包只需将Floyd中的f数组改为布尔数组，将加号改为and就可以了。

```
for (int k=0;k<n;k++)
  for (int i=0;i<n;i++)
    for (int j=0;j<n;j++)
      f[i][j] |= f[i][k] && f[k][j]
```

```
for k:=1 to n do
  for i:=1 to n do
     for j:=1 to n do
      f[i,j]:= f[i,k] and f[k,j] or f[i,j]
```

{{{{{我补充一下，有关证明是正确的}}}}} 本质上是dp，i到j的最短距离，可以经过k,也可以不经过，枚举k就可以了

注：传递闭包 在数学中，在集合 X 上的二元关系 R 的传递闭包是包含 R 的 X 上的最小的传递关系。 例如，如果 X 是(生或死)人的集合而 R 是关系“为父于”，则 R 的传递闭包是关系“x 是 y 的祖先”。再比如，如果 X 是空港的集合而关系 xRy 为“从空港 x 到空港 y 有直航”，则 R 的传递闭包是“可能经一次或多次航行从 x 飞到 y”。 （出自 wikipedia ）

引用&参考

- USACO Training

链接

图论及图论算法 [编辑]

图 - 有向图 - 无向图 - 连通图 - 强连通图 - 完全图 - 稀疏图 - 零图 - 树 - 网络

基本遍历算法：宽度优先搜索 - 深度优先搜索 - A* - 并查集求连通分支 - Flood Fill

最短路：Dijkstra - Bellman-Ford（SPFA） - Floyd-Warshall - Johnson算法

最小生成树：Prim - Kruskal

强连通分支：Kosaraju - Gabow - Tarjan

网络流：增广路法（Ford-Fulkerson，Edmonds-Karp，Dinic） - 预流推进 - Relabel-to-front

图匹配 - 二分图匹配：匈牙利算法 - Kuhn-Munkres - Edmonds' Blossom-Contraction

Floyd-Warshall算法是一个小作品，欢迎帮助扩充这个条目。

3个分类: 需要关注的页面 | 图论 | 小作品