**实验报告**

**《复杂网络动力学基础》**

**第一次大作业**

**姓名：王嘉禾**

**班级：计试001**

**学号：2193211079**

**2022年10月4日**

|  |
| --- |
| 1. **题目、问题描述及求解内容(含网络图)**   在例3.5的无向网络（图3.4）中，共有6 个节点7 条边，N=6，M=7，试利用程序求解6 个节点的介数Bi(i=16)和7 条边的介数Bij(e1e7)，并要求手算B1 和B15 |
| 1. **实验原理**   一张图中点的介数为所有不包含该点的点对之间的经过该点的最短路径条数和所有最短路径条数比值的和；一张图中一条边的介数为所有不为该边两端点的点对之间的经过边的最短路径条数和所有最短路径条数比值的和。 |
| 1. **求解过程(含流程图)** |
| 1. **实验数据** |
| 1. **实验结果与分析**   实验结果：点v1~v6的介数为4,2.5,2.5,0.5,0,0.5，边（按上图运行结果中的顺序）的介数分别为3,3,4,1,3,3,1,与书中所提供参考答案一致。 |
| **六、实验代码、注释及流程图(Matlab、C/C++、Python等)**   1. #include <iostream> 2. #include <cstdio> 3. #include <cstring> 4. #include <cmath> 5. **using** **namespace** std; 6. **const** **int** N=6; 7. **int** a[110][110],minl[110][110],vis[110]; 8. **double** mins[110][110],Bv[110],Be[110][110],Tv[110],Te[110][110]; 10. **void** addedge(**int** x,**int** y) 11. { 12. a[x][y]=a[y][x]=1; 13. **return**; 14. } 16. **void** dij(**int** o) //dijkstra算法，算出单源最短路径 17. { 18. memset(vis,0,**sizeof**(vis)); 19. vis[o]=1; 20. **for**(**int** i=1;i<=N;++i) minl[o][i]=a[o][i]; 21. **bool** flag=1; 22. **int** k; 23. **while**(flag) 24. { 25. flag=0; 26. k=0; 27. **for**(**int** i=1;i<=N;++i) 28. { 29. **if**(vis[i] || (!minl[o][i])) **continue**; 30. flag=1; 31. **if**(!k || (k && minl[o][i]<minl[o][k])) k=i; 32. } 33. vis[k]=1; 34. **for**(**int** i=1;i<=N;++i) 35. { 36. **if**(vis[i] || !a[k][i]) **continue**; 37. **if**(minl[o][k]+1<minl[o][i] || !minl[o][i]) minl[o][i]=minl[o][k]+1; 38. } 39. } 40. **return**; 41. } 43. **void** gets(**int** o,**int** d,**int** tmp,**int** len) //搜索所有点对之间最短路径条数 44. { 45. **if**(len>=minl[o][d]) 46. { 47. **if**(tmp==d) 48. { 49. mins[o][d]++; 50. vis[d]=1; 51. **for**(**int** i=1;i<=N;++i) 52. { 53. **if**(vis[i] && i!=o && i!=d) Tv[i]++; 54. **for**(**int** j=i+1;j<=N;++j) 55. { 56. **if**(vis[i] && vis[j] && a[i][j] && !(i==o && j==d || i==d && j==o)) Te[i][j]++; 57. } 58. } 59. vis[d]=0; 60. } 61. **return**; 62. } 63. **else** 64. { 65. vis[tmp]=1; 66. **for**(**int** i=1;i<=N;++i) 67. { 68. **if**(a[tmp][i] && !vis[i]) gets(o,d,i,len+1); 69. } 70. vis[tmp]=0; 71. } 72. **return**; 73. } 75. **int** main() 76. { 77. memset(a,0,**sizeof**(a)); 78. memset(minl,0,**sizeof**(minl)); 79. memset(mins,0,**sizeof**(mins)); 80. memset(Bv,0,**sizeof**(Bv)); 81. memset(Be,0,**sizeof**(Be)); 82. addedge(1,2); 83. addedge(1,3); 84. addedge(1,5); 85. addedge(2,3); 86. addedge(2,4); 87. addedge(3,6); 88. addedge(4,6); 89. **for**(**int** i=1;i<=N;++i) dij(i); //对每个点做dijkstra算法，可求出所有点对之间的最短路径 90. cout << "点对之间最短路径:\n"; 91. **for**(**int** i=1;i<=N;++i) 92. { 93. **for**(**int** j=1;j<=N;++j) 94. { 95. cout << minl[i][j] << " "; 96. } 97. cout << endl; 98. } 99. **for**(**int** i=1;i<=N;++i) 100. { 101. **for**(**int** j=i+1;j<=N;++j) 102. { 103. **if**(minl[i][j]) //每次搜索一对起终点时，就可以为所有点和边计算介数的一部分 104. { 105. memset(vis,0,**sizeof**(vis)); 106. memset(Te,0,**sizeof**(Te)); 107. memset(Tv,0,**sizeof**(Tv)); 108. gets(i,j,i,0); 109. **for**(**int** k=1;k<=N;++k) 110. { 111. Tv[k]/=mins[i][j]; 112. Bv[k]+=Tv[k]; 113. **for**(**int** p=k+1;p<=N;++p) 114. { 115. Te[k][p]/=mins[i][j]; 116. Be[k][p]+=Te[k][p]; 117. } 118. } 119. } 120. } 121. } 122. cout << "点对之间最短路径条数:\n"; 123. **for**(**int** i=1;i<=N;++i) 124. { 125. **for**(**int** j=1;j<=N;++j) 126. { 127. cout << minl[i][j] << " "; 128. } 129. cout << endl; 130. } 131. cout << "点的介数:\n"; 132. **for**(**int** i=1;i<=N;++i) cout << "vertice " << i << ": " << Bv[i] << endl; 133. cout << "边的介数:\n"; 134. **for**(**int** i=1;i<=N;++i) 135. { 136. **for**(**int** j=i+1;j<=N;++j) 137. { 138. **if**(a[i][j]) cout << "edge(" << i << "," << j << "): " << Be[i][j] << endl; 139. } 140. } 141. **return** 0; 142. } |
| **七、实验感想与建议**  按照书中给出的算法模拟，求点对之间最短路径和最短路径条数的算法比较简单，直接使用dijkstra算法和深度优先搜索即可，计算介数分量的算法需要灵活运用，在搜索过程中先计算经过特定点/边的数量，最后除以总数，其中，对边的判断可以归纳到对点的判断，因为是无权网络，因此最短路径上的边一定是两端点直接唯一连边，直接遍历判断是否相连即可，这其中的算法可以进一步深入思考，例如在搜索时对点边的情况进行临时存储，直接计算而无需遍历等。 |

|  |
| --- |
| **一、题目、问题描述及求解内容(含网络图)**  在互联网的网页链接和搜索引擎中，互联网可以看成一个有向图（无权有向网络），每一个网页是图的一个节点（顶点），网页间的每一次超链接是图的一个边，Google 公司使用了PageRank 算法作为搜索引擎的核心算法。下图描述了一个互联网搜索引擎的网页链接关系图，其中有7 个节点（网页）和18 条边（超链接），N=7，M=18 。  基于复杂网络特征向量中心性、Pagerank 算法和例3.6 的学习，试求：（1）网络的邻接矩阵B、特征向量中心性中最大特征值λ和所对应的特征向量以及归一化后的特征向量，以及该特征向量中心性的分值；（2）网络的Markov 链的状态转移概率矩阵P 及其转置矩阵PT ，最大特征值max ，Markov 链的平稳分布，PageRank 值及其直方图，模型参数d 可选为0.85 |
| 二**、实验原理**  对于结点Vi,它的特征向量中心性分值正比于连接到它的所有节点的中心性分值的总和，在有向图中，正比于有边指向它的所有节点的中心性分值总和。令A为复杂网络的邻接矩阵，则有ATx=λx,由Frobenius定理，只有最大的特征值对应的特征向量才是特征向量中心性测度所需要的，因此求出邻接矩阵转置矩阵的最大特征值对应的特征向量再进行归一化，则得出特征向量中心性。  求PageRank时，只需将邻接矩阵改为Markov链状态转移概率矩阵，并进行相同的操作。 |
| **三、求解过程(含流程图)** |
| 1. **实验数据** |
| **五、实验结果与分析**  结点1~7的特征向量中心性分值分别为 [0.2074 0.1831 0.1788 0.1321 0.1666 0.0588 0.0733]  该网络的Pagerank值分别为[0.2803 0.1588 0.1389 0.1082 0.1842 0.0606 0.0691]  以上两组数据可以直观看出，在大量访问的情况下，网页1和网页5的访问率较高，而网页6和网页7的访问率较低。 |
| **六、实验代码、注释及流程图(Matlab、C/C++、Python等)**   1. **import** numpy as np 2. **import** pandas as pd 3. **from** matplotlib **import** pyplot 4. np.set\_printoptions(precision=4,suppress=True) 5. N=7 6. b=np.zeros((N,N),dtype=int)  #构造邻接矩阵 7. b[0][[1,2,3,4,6]]=1 8. b[1][0]=1 9. b[2][[0,1]]=1 10. b[3][[1,2,4]]=1 11. b[4][[0,2,3,5]]=1 12. b[5][[0,4]]=1 13. b[6][4]=1 15. a=b.transpose() 16. eigenvalue,eigenvector=np.linalg.eig(a) #求出特征值和特征向量 17. lamda=np.real(eigenvalue[0]) #默认第一个为最大特征值 18. X=np.zeros((N),dtype=float) 19. **for** i **in** range(N): 20. X[i]=np.real(eigenvector[i][0]) #取出所对应的特征向量 21. X=X/np.sum(X) #归一化 22. **print**("最大特征值:",end=" ") 23. **print**(round(lamda,4)) 24. **print**("中心性分值:",end=" ") 25. **print**(X) 27. d=0.85 28. A=np.zeros((N,N),dtype=float) 29. **for** i **in** range(N): 30. **for** j **in** range(N): 31. A[i][j]=(1-d)/N+d\*b[i][j]/(np.sum(b[i])) #构造状态转移概率矩阵 32. **print**("状态转移概率矩阵:") 33. **print**(A) 34. A=A.transpose() 35. eigenvalue,eigenvector=np.linalg.eig(A)  #求出特征值和特征向量 36. lamda=np.real(eigenvalue[0]) #默认第一个为最大特征值 37. X=np.zeros((N),dtype=float) 38. **for** i **in** range(N): 39. X[i]=np.real(eigenvector[i][0]) #取出所对应的特征向量 40. X=X/np.sum(X) 41. Y=np.arange(1,N+1,1) 42. **print**("最大特征值:",end=" ") 43. **print**(round(lamda,4)) 44. **print**("Markov链的平稳分布:",end=" ") 45. **print**(X) 46. pyplot.bar(Y,height=X) #画出直方图 47. pyplot.title("PageRank值",fontname="SimHei") 48. pyplot.show() |
| **七、实验感想与建议**  求解特征向量中心性和PageRank值的过程中均要求取最大特征值对应的特征向量，而矩阵在复平面上共有N个特征向量，对于无向图，邻接矩阵为对称阵，其特征值和特征向量均为实数，从定义的角度来说，非最大特征值对应的特征向量也符合一种动态平衡状态，但它们与最大特征值对应的特征向量的区别在哪？这里可以就稳定平衡点和非稳定平衡点的角度进行思考，并且对于有向图来说，包含复数的特征向量是否有实际意义也可以进行探究。 |