云南大学数学与统计学院  
《运筹学通论实验》上机实践报告

|  |  |  |
| --- | --- | --- |
| 课程名称：运筹学实验 | 年级：2015级 | 上机实践成绩： |
| 指导教师：李建平 | 姓名：刘鹏 | 专业：信息与计算科学 |
| 上机实践名称：Dijkstra算法求最短路径 | 学号：20151910042 | 上机实践日期：2018-07-08 |
| 上机实践编号：5 | 组号： |  |

# 实验目的

1. 学习Dijkstra算法的使用；
2. 了解Dijkstra算法作为贪心算法能达最优的理论证明。

# 实验内容

1. 写出Dijkstra算法[1]的伪码描述[2]；
2. 用C语言[3]编程实现Dijkstra算法，找出一幅图的给定两个节点间的最短路径；

# 实验平台

Microsoft Windows 10 Pro Workstation 1803；

Microsoft Visual Studio 2017 Enterprise。

# 算法设计

## 算法背景

Dijkstra算法解决的是带权重的有向图上单源最短路径问题，该算法要求所有边的权重都为非负值。假定所有的边，都有权重。Dijkstra算法在运行过程中维护的关键信息是一组节点集合，从源节点到该节点集合中每个节点之间的最短路径都已经被找到。算法重复地从节点集合中选择最短路径估计最小的节点，将加到集合中去，然后对所有从发出的边进行松弛（当然也包括中已经存在的边）。所谓的松弛，指的是对初态的一种调整。若存在一个从到的有向边，那么令中保留的前驱信息。

**松弛**：第一次全体松弛（或称初始化）是指把源点的属性设置为，其他节点的属性设置为无穷大；其他过程中的松弛为：从集合中的通过某种方法制定的顶点开始出发，找到顶点的所有邻边，然后把邻点的属性都改变，如将一个邻点的属性改为，其他邻点的改法完全类似。

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **End** | **INITIALIZE-SINGLE-SOURCE**, 初始化  图，初始节点  已经改变过图  **for** each vertex  let  let  GOTO **End** |

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1:**  **End** | **RELAX**, 对边在时间内进行松弛操作  节点，两个节点之间存在由前者指向后者的边  已知  已经改变过的节点  if  let  let  GOTO **End** |

根据上面给出的两个子函数，加上一个优先队列，可以实现一个复杂度比较低的Dijkstra算法。

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **Step 2**  **Step 3**  **Step 4**  **End** | **DIJKSTRA**, 生成一个图，该图包含任意两点间的最短路径  图，初始节点，权重函数  图，其中任何两点间的仅有一条路，且为这两点在原来图中的最短路  **INITIALIZE-SINGLE-SOURCE**()  let  把节点集合加入到优先队列中  **while**  令为从中弹出的最小的顶点  let  对于图中所有的与相邻的顶点:  **RELAX**()  GOTO **End** |

# 程序代码

## 程序描述

时间关系，4号报告中的一些bug还没有调完，但是提交在即。这段C++程序从开源社区获得。

## 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109 | /\*  \* Copyright (c) 2018, Liu Peng, School of Mathematics and Statistics, YNU  \* Apache License.  \*  \* 文件名称：Shortest\_Path.cpp  \* 文件标识：见配置管理计划书  \* 摘 要：寻找给定顶点间的最短路  \*  \* 当前版本：1.0  \* 作 者：刘鹏  \* 创建日期：2018年7月7日  \* 完成日期：2018年7月7日  \*  \* 取代版本：  \* 原作者 ：刘鹏  \* 完成日期：  \*/  /\*  \* A function based on Dijkstra Algorithm to find the Shortest Path  \* in a undirected graph.  \*/  #include <unordered\_map>  #include <vector>  #include <limits>  #include <algorithm>  #include <iostream>  **using** **namespace** std**;**  class Graph **{**  unordered\_map**<**char**,** const unordered\_map**<**char**,** int**>>** vertices**;**  public**:**  void add\_vertex**(**char name**,** const unordered\_map**<**char**,** int**>&** edges**)** **{**  vertices**.**insert**(**unordered\_map**<**char**,** const unordered\_map**<**char**,** int**>>::**value\_type**(**name**,** edges**));**  **}**  vector**<**char**>** shortest\_path**(**char start**,** char finish**)** **{**  unordered\_map**<**char**,** int**>** distances**;**  unordered\_map**<**char**,** char**>** previous**;**  vector**<**char**>** nodes**;**  vector**<**char**>** path**;**  auto comparator **=** **[&](**char left**,** char right**)** **{** **return** distances**[**left**]** **>** distances**[**right**];** **};**  **for** **(**auto**&** vertex **:** vertices**)** **{**  **if** **(**vertex**.**first **==** start**)** **{**  distances**[**vertex**.**first**]** **=** 0**;**  **}**  **else** **{**  distances**[**vertex**.**first**]** **=** numeric\_limits**<**int**>::**max**();**  **}**  nodes**.**push\_back**(**vertex**.**first**);**  push\_heap**(**begin**(**nodes**),** end**(**nodes**),** comparator**);**  **}**  **while** **(!**nodes**.**empty**())** **{**  pop\_heap**(**begin**(**nodes**),** end**(**nodes**),** comparator**);**  char smallest **=** nodes**.**back**();**  nodes**.**pop\_back**();**  **if** **(**smallest **==** finish**)** **{**  **while** **(**previous**.**find**(**smallest**)** **!=** end**(**previous**))** **{**  path**.**push\_back**(**smallest**);**  smallest **=** previous**[**smallest**];**  **}**  **break;**  **}**  **if** **(**distances**[**smallest**]** **==** numeric\_limits**<**int**>::**max**())** **{**  **break;**  **}**  **for** **(**auto**&** neighbor **:** vertices**[**smallest**])** **{**  int alt **=** distances**[**smallest**]** **+** neighbor**.**second**;**  **if** **(**alt **<** distances**[**neighbor**.**first**])** **{**  distances**[**neighbor**.**first**]** **=** alt**;**  previous**[**neighbor**.**first**]** **=** smallest**;**  make\_heap**(**begin**(**nodes**),** end**(**nodes**),** comparator**);**  **}**  **}**  **}**  **return** path**;**  **}**  **};**  int main**()** **{**  Graph g**;**  g**.**add\_vertex**(**'A'**,** **{** **{** 'B'**,** 7 **},{** 'C'**,** 8 **}** **});**  g**.**add\_vertex**(**'B'**,** **{** **{** 'A'**,** 7 **},{** 'F'**,** 2 **}** **});**  g**.**add\_vertex**(**'C'**,** **{** **{** 'A'**,** 8 **},{** 'F'**,** 6 **},{** 'G'**,** 4 **}** **});**  g**.**add\_vertex**(**'D'**,** **{** **{** 'F'**,** 8 **}** **});**  g**.**add\_vertex**(**'E'**,** **{** **{** 'H'**,** 1 **}** **});**  g**.**add\_vertex**(**'F'**,** **{** **{** 'B'**,** 2 **},{** 'C'**,** 6 **},{** 'D'**,** 8 **},{** 'G'**,** 9 **},{** 'H'**,** 3 **}** **});**  g**.**add\_vertex**(**'G'**,** **{** **{** 'C'**,** 4 **},{** 'F'**,** 9 **}** **});**  g**.**add\_vertex**(**'H'**,** **{** **{** 'E'**,** 1 **},{** 'F'**,** 3 **}** **});**  **for** **(**char vertex **:** g**.**shortest\_path**(**'A'**,** 'H'**))** **{**  cout **<<** vertex **<<** endl**;**  **}**  **return** 0**;**  **}** |

程序代码 1

# 运行结果

图片包含 屏幕截图

已生成极高可信度的说明

运行结果 1

## 代码分析



分析上图之后可以看出，运行结果没有问题。

# 实验体会

实验比较简单，但是时间不够了。感谢开源社区[4]提供的高质量程序

# 参考文献

[1] HILLIER F S, LIEBERMAN G J. 运筹学导论 [M]. 9th ed. 北京: 清华大学出版社, 2010.

[2] CORMEN T H, LEISERSON C E, RIVEST R L, et al. 算法导论 [M]. 3rd ed. 北京: 机械工业出版社, 2013.

[3] 林锐. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.

[4] https://github.com/mburst/dijkstras-algorithm