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：Prim算法求图的支撑树与联通子图 | 学号：20151910042 | 上机实践日期：2018-07-08 |
| 上机实践编号：4 | 组号： |  |

# 实验目的

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

# 实验内容

1. 写出Prim（反圈法）算法[1]的伪码描述[2]；
2. 用C语言[3]编程实现Prim算法，找出一幅图的最小生成树；
3. 写出Prim算法求解一个图的所有联通子图的算法；
4. 用C语言编程实现求一个图的所有联通子图的Prim算法程序。

# 实验平台

Microsoft Windows 10 Pro Workstation 1803；

Microsoft Visual Studio 2017 Enterprise。

# 算法设计

## 算法背景

普里姆算法（Prim算法），图论中的一种算法，可在加权联通图里搜索最小生成树。意即由此算法搜索到的边子集所构成的树中，不但包括了连通图里的所有顶点，且其所有边的权值之和为最小。该算法于1930年由杰克数学家沃伊捷赫·亚尔尼克发现；并在1957年由美国计算机科学家罗伯特·普里姆独立发现；1959年，艾兹格·迪科斯彻再次发现了该算法。因此，在某些场合，普里姆算法又被称为DJP算法、亚尔尼克算法或普里姆－亚尔尼克算法。Prim算法的工作原理与Dijkstra的最短路径算法相似。本策略属于贪心策略，因为每一步所加入的边都必须是使得树的总权重增加量最小的边。

## 时间复杂度

这个算法的时间复杂度与图的实现方法有关。设图，其中是图的所有节点的集合，是图的所有边的集合。图是由如果采用比较低级的邻接矩阵实现，那么Prim算法的时间复杂度是；如果用二叉堆、邻接表来实现，那么时间复杂度是；如果用斐波那契堆来实现复杂度可以降低至

# 程序代码

## 程序描述

综合考虑了实现难易程度与算法的时间复杂度，我决定采用邻接映射（Adjacent Map）来做为主要的数据结构。邻接映射的主要思想是用哈希表这种快速查找表来代替遍历带来的高时间复杂度，本次实验的结构在仅仅采用哈希表的基础上进行了改进，把所有已经被占据的数组下标记录下来，放入一个动态数组里，为以后查找所有与某个节点相邻的节点或者找寻连接到该点的无向边提供便利。

邻接映射本身就是一张哈希表，key是vertex，value是另外的一个子哈希表，在子哈希表里面，key是与vertex相邻的vertex（并且是有向的，只能从前者到后者），value是两个vertex中间的边。为了能够在一个结构里实现有向图（directed graph）与无向图（undirected graph），这里采用两个大哈希表来实现有向图，其中一个记录的是这种类型，第二个记录这种类型，两者恰好反向。由于无向图的两种上述类型必然是同时存在的，而有向图则不然，所以可以通过这种方式进行处理有向图。由于普里姆算法主要关心无向图，所以代码中并没有实现针对有向图的算法。

普里姆算法实现的具体思路如下。

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **Step 2**  **Step 3**  **Step 4**  **End** | **PRIM-MST**  无向图，初始节点  图的最小生成树  **f**存储一个节点集合，的初值为起点组成的单元素集合  初始化一个图，的初值为空集，把中所有的节点添加到中  通过对中所有的节点分别进行两次哈希表查询：第一次对进行查询，得到一个子哈希表；第二次对第一步得到的子哈希表进行查询，得到这个子哈希表中所有的边。每次查询的时间复杂度均为。这样就找到与集合中所有节点分别相邻的所有的边，将这些边添加到一个临时边集合里。  对于边集合，进行如下处理：对所有终点不在里的边而言，将之权重添加到一个临时空泛型动态数组里，对这个数组进行排序，找到最小权重值；之后从所有的边中，任意挑选一个权重为最小值的边，将其终点添加到节点集合以及图中，然后把这条边也添加到图中。如果图的节点数量等于输入的图的节点数量，GOTO **End**，返回值为；否则，返回**Step 3** |

## 程序代码示例

源代码数量太多，这里仅仅给出核心文件Prim.cpp的代码，其他程序在本节报告的附录中给出，工程文件参看我的[GitHub链接](https://github.com/LittleNewton/Operations_Research_Report/tree/master/%23Code/04.%20Prim_Algorithm)。

|  |  |
| --- | --- |
| 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 | /\*  \* Copyright (c) 2018, Liu Peng, School of Mathematics and Statistics, YNU  \* Apache License.  \*  \* 文件名称：Prim.cpp  \* 文件标识：见配置管理计划书  \* 摘 要：Prim算法  \*  \* 当前版本：1.0  \* 作 者：刘鹏  \* 创建日期：2018年6月25日  \* 完成日期：2018年6月26日  \*  \* 取代版本：  \* 原作者 ：刘鹏  \* 完成日期：  \*/  /\*  \* A function based on Prim Algorithm to find the minimal spanning tree  \* of a undirected graph.  \*/  #include "Graph.h"  // Get the Minimal Spanning Tree of a connected graph.  // If the graph is not connected, exception would be raised.  Graph **\***MST\_Prim\_Jarnik**(**Graph **\***g**,** Vertex start**,** Function f**)** **{**  map\_t arg**;** // make up the parameters hashmap\_get fuc needing  **if** **(**MAP\_MISSING **==** hashmap\_get**(**g**,** **(**char **\*)**start**,** **&**arg**))** **{**  printf**(**"fatal Error: bad input!\n"**);**  **return** **NULL;**  **}**  Graph **\***ans **=** Graph\_init**(false);** // the answer of this algorithm  Dynamic\_Array **\***V **=** Dynamic\_Array\_init**();** // set of vertices have been found  Dynamic\_Array\_append**(**V**,** **(**any**)**start**);**  Graph\_insert\_vertex**(**ans**,** start**);**  int i**;**  int j**;**  int Length\_of\_Graph\_in **=** hashmap\_length**(**g**->**outgoing**);**  Vertex temp**;**  map\_t temp\_map**;**  Dynamic\_Array **\***temp\_Edge **=** Dynamic\_Array\_init**();** // set of edges connected with V  // Generally, loop will stop while len(V) = len(g).  // If g is not connected, error will be raised.  **while** **(**V**->**n **<** Length\_of\_Graph\_in**)** **{**  **for** **(**i **=** 1**;** i **<=** V**->**n**;** i**++)** **{**  temp **=** **(**Vertex**)**Dynamic\_Array\_get\_Element**(**V**,** i**);**  temp\_map **=** Graph\_get\_adjacent\_Vertices**(**g**,** temp**);** // submap  Dynamic\_Array **\***index **=** hashmap\_used\_index**(**temp\_map**);** // used slots of the submap  Edge **\***tmp**;**  int change\_memo **=** temp\_Edge**->**n**;**  **for** **(**j **=** 1**;** j **<=** index**->**n**;** j**++)** **{**  int addr **=** **(**int**)**Dynamic\_Array\_get\_Element**(**index**,** j**);**  tmp **=** **(**Edge **\*)**hashmap\_select**(**temp\_map**,** addr**);**  // only if the destination of Edge(tmp) is not  // included in the map(G), the appending  // operation can be done.  **if** **(**hashmap\_get**(**ans**->**outgoing**,** **(**char **\*)**tmp**->**destination**,** **(**void **\*\*)&**arg**)**  **==** MAP\_MISSING**)** **{**  Dynamic\_Array\_append**(**temp\_Edge**,** **(**any**)**tmp**);**  **}**  **}**  **}**  // find the minimal value  int min\_location **=** Dynamic\_Array\_min**(**temp\_Edge**,** f\_get\_double**);**  Edge **\***min **=** **(**Edge **\*)**Dynamic\_Array\_get\_Element**(**temp\_Edge**,** min\_location**);**  temp **=** min**->**destination**;**  Dynamic\_Array\_append**(**V**,** temp**);**  Graph\_insert\_vertex**(**ans**,** temp**);**  Graph\_insert\_edge**(**ans**,** min**->**origin**,** temp**,** min**->**element**);**  **}**  **return** ans**;**  **}** |

程序代码 1

## 具体可运行的代码

|  |  |
| --- | --- |
| 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  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310 | /\*  \* Copyright (c) 2018, Liu Peng, School of Mathematics and Statistics, YNU  \* Apache License.  \*  \* 文件名称：Source.cpp  \* 文件标识：见配置管理计划书  \* 摘 要：测试Prim算法  \*  \* 当前版本：1.0  \* 作 者：刘鹏  \* 创建日期：2018年7月9日  \* 完成日期：2018年7月9日  \*  \* 取代版本：0.9  \* 原作者 ：刘鹏  \* 完成日期：  \*/  // C / C++ program for Prim's MST for adjacency list representation of graph  #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #include <time.h>  // A structure to represent a node in adjacency list  struct AdjListNode **{**  int dest**;**  int weight**;**  struct AdjListNode**\*** next**;**  **};**  // A structure to represent an adjacency liat  struct AdjList **{**  struct AdjListNode **\***head**;** // pointer to head node of list  **};**  // A structure to represent a graph. A graph is an array of adjacency lists.  // Size of array will be V (number of vertices in graph)  struct Graph **{**  int V**;**  struct AdjList**\*** array**;**  **};**  // A utility function to create a new adjacency list node  struct AdjListNode**\*** newAdjListNode**(**int dest**,** int weight**)** **{**  struct AdjListNode**\*** newNode **=**  **(**struct AdjListNode**\*)** malloc**(sizeof(**struct AdjListNode**));**  newNode**->**dest **=** dest**;**  newNode**->**weight **=** weight**;**  newNode**->**next **=** **NULL;**  **return** newNode**;**  **}**  // A utility function that creates a graph of V vertices  struct Graph**\*** createGraph**(**int V**)** **{**  struct Graph**\*** graph **=** **(**struct Graph**\*)** malloc**(sizeof(**struct Graph**));**  graph**->**V **=** V**;**  // Create an array of adjacency lists. Size of array will be V  graph**->**array **=** **(**struct AdjList**\*)** malloc**(**V **\*** **sizeof(**struct AdjList**));**  // Initialize each adjacency list as empty by making head as NULL  **for** **(**int i **=** 0**;** i **<** V**;** **++**i**)**  graph**->**array**[**i**].**head **=** **NULL;**  **return** graph**;**  **}**  // Adds an edge to an undirected graph  void addEdge**(**struct Graph**\*** graph**,** int src**,** int dest**,** int weight**)** **{**  // Add an edge from src to dest. A new node is added to the adjacency  // list of src. The node is added at the begining  struct AdjListNode**\*** newNode **=** newAdjListNode**(**dest**,** weight**);**  newNode**->**next **=** graph**->**array**[**src**].**head**;**  graph**->**array**[**src**].**head **=** newNode**;**  // Since graph is undirected, add an edge from dest to src also  newNode **=** newAdjListNode**(**src**,** weight**);**  newNode**->**next **=** graph**->**array**[**dest**].**head**;**  graph**->**array**[**dest**].**head **=** newNode**;**  **}**  // Structure to represent a min heap node  struct MinHeapNode **{**  int v**;**  int key**;**  **};**  // Structure to represent a min heap  struct MinHeap **{**  int size**;** // Number of heap nodes present currently  int capacity**;** // Capacity of min heap  int **\***pos**;** // This is needed for decreaseKey()  struct MinHeapNode **\*\***array**;**  **};**  // A utility function to create a new Min Heap Node  struct MinHeapNode**\*** newMinHeapNode**(**int v**,** int key**)** **{**  struct MinHeapNode**\*** minHeapNode **=**  **(**struct MinHeapNode**\*)** malloc**(sizeof(**struct MinHeapNode**));**  minHeapNode**->**v **=** v**;**  minHeapNode**->**key **=** key**;**  **return** minHeapNode**;**  **}**  // A utilit function to create a Min Heap  struct MinHeap**\*** createMinHeap**(**int capacity**)** **{**  struct MinHeap**\*** minHeap **=**  **(**struct MinHeap**\*)** malloc**(sizeof(**struct MinHeap**));**  minHeap**->**pos **=** **(**int **\*)**malloc**(**capacity **\*** **sizeof(**int**));**  minHeap**->**size **=** 0**;**  minHeap**->**capacity **=** capacity**;**  minHeap**->**array **=**  **(**struct MinHeapNode**\*\*)** malloc**(**capacity **\*** **sizeof(**struct MinHeapNode**\*));**  **return** minHeap**;**  **}**  // A utility function to swap two nodes of min heap. Needed for min heapify  void swapMinHeapNode**(**struct MinHeapNode**\*\*** a**,** struct MinHeapNode**\*\*** b**)** **{**  struct MinHeapNode**\*** t **=** **\***a**;**  **\***a **=** **\***b**;**  **\***b **=** t**;**  **}**  // A standard function to heapify at given idx  // This function also updates position of nodes when they are swapped.  // Position is needed for decreaseKey()  void minHeapify**(**struct MinHeap**\*** minHeap**,** int idx**)** **{**  int smallest**,** left**,** right**;**  smallest **=** idx**;**  left **=** 2 **\*** idx **+** 1**;**  right **=** 2 **\*** idx **+** 2**;**  **if** **(**left **<** minHeap**->**size **&&**  minHeap**->**array**[**left**]->**key **<** minHeap**->**array**[**smallest**]->**key**)**  smallest **=** left**;**  **if** **(**right **<** minHeap**->**size **&&**  minHeap**->**array**[**right**]->**key **<** minHeap**->**array**[**smallest**]->**key**)**  smallest **=** right**;**  **if** **(**smallest **!=** idx**)** **{**  // The nodes to be swapped in min heap  struct MinHeapNode **\***smallestNode **=** minHeap**->**array**[**smallest**];**  struct MinHeapNode **\***idxNode **=** minHeap**->**array**[**idx**];**  // Swap positions  minHeap**->**pos**[**smallestNode**->**v**]** **=** idx**;**  minHeap**->**pos**[**idxNode**->**v**]** **=** smallest**;**  // Swap nodes  swapMinHeapNode**(&**minHeap**->**array**[**smallest**],** **&**minHeap**->**array**[**idx**]);**  minHeapify**(**minHeap**,** smallest**);**  **}**  **}**  // A utility function to check if the given minHeap is ampty or not  int isEmpty**(**struct MinHeap**\*** minHeap**)** **{**  **return** minHeap**->**size **==** 0**;**  **}**  // Standard function to extract minimum node from heap  struct MinHeapNode**\*** extractMin**(**struct MinHeap**\*** minHeap**)** **{**  **if** **(**isEmpty**(**minHeap**))**  **return** **NULL;**  // Store the root node  struct MinHeapNode**\*** root **=** minHeap**->**array**[**0**];**  // Replace root node with last node  struct MinHeapNode**\*** lastNode **=** minHeap**->**array**[**minHeap**->**size **-** 1**];**  minHeap**->**array**[**0**]** **=** lastNode**;**  // Update position of last node  minHeap**->**pos**[**root**->**v**]** **=** minHeap**->**size **-** 1**;**  minHeap**->**pos**[**lastNode**->**v**]** **=** 0**;**  // Reduce heap size and heapify root  **--**minHeap**->**size**;**  minHeapify**(**minHeap**,** 0**);**  **return** root**;**  **}**  // Function to decreasy key value of a given vertex v. This function  // uses pos[] of min heap to get the current index of node in min heap  void decreaseKey**(**struct MinHeap**\*** minHeap**,** int v**,** int key**)** **{**  // Get the index of v in heap array  int i **=** minHeap**->**pos**[**v**];**  // Get the node and update its key value  minHeap**->**array**[**i**]->**key **=** key**;**  // Travel up while the complete tree is not hepified.  // This is a O(Logn) loop  **while** **(**i **&&** minHeap**->**array**[**i**]->**key **<** minHeap**->**array**[(**i **-** 1**)** **/** 2**]->**key**)** **{**  // Swap this node with its parent  minHeap**->**pos**[**minHeap**->**array**[**i**]->**v**]** **=** **(**i **-** 1**)** **/** 2**;**  minHeap**->**pos**[**minHeap**->**array**[(**i **-** 1**)** **/** 2**]->**v**]** **=** i**;**  swapMinHeapNode**(&**minHeap**->**array**[**i**],** **&**minHeap**->**array**[(**i **-** 1**)** **/** 2**]);**  // move to parent index  i **=** **(**i **-** 1**)** **/** 2**;**  **}**  **}**  // A utility function to check if a given vertex  // 'v' is in min heap or not  bool isInMinHeap**(**struct MinHeap **\***minHeap**,** int v**)** **{**  **if** **(**minHeap**->**pos**[**v**]** **<** minHeap**->**size**)**  **return** **true;**  **return** **false;**  **}**  // A utility function used to print the constructed MST  void printArr**(**int arr**[],** int n**)** **{**  **for** **(**int i **=** 1**;** i **<** n**;** **++**i**)**  printf**(**"%d - %d\n"**,** arr**[**i**],** i**);**  **}**  // The main function that constructs Minimum Spanning Tree (MST)  // using Prim's algorithm  void PrimMST**(**struct Graph**\*** graph**)** **{**  int V **=** graph**->**V**;** // Get the number of vertices in graph  // Array to store constructed MST  int **\***parent **=** **(**int **\*)**calloc**(**V**,** **sizeof(**int**));**  // Key values used to pick minimum weight edge in cut  int **\***key **=** **(**int **\*)**calloc**(**V**,** **sizeof(**int**));**  // minHeap represents set E  struct MinHeap**\*** minHeap **=** createMinHeap**(**V**);**  // Initialize min heap with all vertices. Key value of  // all vertices (except 0th vertex) is initially infinite  int v**;**  **for** **(**v **=** 1**;** v **<** V**;** **++**v**)** **{**  parent**[**v**]** **=** **-**1**;**  key**[**v**]** **=** INT\_MAX**;**  minHeap**->**array**[**v**]** **=** newMinHeapNode**(**v**,** key**[**v**]);**  minHeap**->**pos**[**v**]** **=** v**;**  **}**  // Make key value of 0th vertex as 0 so that it  // is extracted first  key**[**0**]** **=** 0**;**  minHeap**->**array**[**0**]** **=** newMinHeapNode**(**0**,** key**[**0**]);**  minHeap**->**pos**[**0**]** **=** 0**;**  // Initially size of min heap is equal to V  minHeap**->**size **=** V**;**  // In the followin loop, min heap contains all nodes  // not yet added to MST.  **while** **(!**isEmpty**(**minHeap**))** **{**  // Extract the vertex with minimum key value  struct MinHeapNode**\*** minHeapNode **=** extractMin**(**minHeap**);**  int u **=** minHeapNode**->**v**;** // Store the extracted vertex number  // Traverse through all adjacent vertices of u (the extracted  // vertex) and update their key values  struct AdjListNode**\*** pCrawl **=** graph**->**array**[**u**].**head**;**  **while** **(**pCrawl **!=** **NULL)** **{**  int v **=** pCrawl**->**dest**;**  // If v is not yet included in MST and weight of u-v is  // less than key value of v, then update key value and  // parent of v  **if** **(**isInMinHeap**(**minHeap**,** v**)** **&&** pCrawl**->**weight **<** key**[**v**])** **{**  key**[**v**]** **=** pCrawl**->**weight**;**  parent**[**v**]** **=** u**;**  decreaseKey**(**minHeap**,** v**,** key**[**v**]);**  **}**  pCrawl **=** pCrawl**->**next**;**  **}**  **}**  // print edges of MST  // printArr(parent, V);  **}**  // Driver program to test above functions  int main**()** **{**  // Let us create the graph given in above fugure  FILE **\***file\_input**;**  file\_input **=** fopen**(**"../dataset/grafo-esparso-10000.txt"**,** "r"**);**  struct Graph**\*** graph**;**  char a**[**999**],** b**[**999**],** c**[**999**],** v**[**999**];**  **if** **(**file\_input**)** **{**  fscanf**(**file\_input**,** "%s\n"**,** v**);**  graph **=** createGraph**(**atoi**(**v**));**  **while** **((**fscanf**(**file\_input**,** "%s %s %s\n"**,** a**,** b**,** c**))** **!=** EOF**)** **{**  addEdge**(**graph**,** atoi**(**a**),** atoi**(**b**),** atoi**(**c**));**  **}**  **}**  fclose**(**file\_input**);**  int i**;**  **for** **(**i **=** 0**;** i **<** 173**;** **++**i**)** **{**  clock\_t tStart **=** clock**();**  PrimMST**(**graph**);**  printf**(**"Time taken: %fs\n"**,** **(**double**)(**clock**()** **-** tStart**)** **/** CLOCKS\_PER\_SEC**,** i **+** 1**);**  **}**  **return** 0**;**  **}** |

程序代码 2

# 运行结果

## 运行结果

考虑到这里并不是用邻接矩阵做的，所以很难输出一个图。代码5.2.1就做了173便Prim算法，查看在一个很复杂的图里运行时间如何。

# 实验体会

为了练习使用Map结构与了解Prim算法，本次实验采用的数据结构为邻接映射，核心的映射实现为CRC32哈希函数，基于这个函数，完成了HashMap的结构与一批相关函数，然后基于这些成果，进一步完成了泛型有向图、无向图结构。

遗憾的是时间有限加上我本人水平有限，程序不能做到尽善尽美，而且目前版本还没有一个良好的图输出程序，就像是采取最简单的邻接矩阵，输出也只是一堆数字，与实际的图没有视觉联系。接下来我可能花点时间完善一下这个输出。

# 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.