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：	

一、 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。

二、 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。

三、 实验平台

Microsoft Windows 10 Pro Workstation 1803；

Microsoft Visual Studio 2017 Enterprise。

四、 算法设计

4.1 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。

4.2 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$G = (V, E)$，其中V是图的所有节点的集合，E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$O(|V|^2)$；如果用二叉堆、邻接表来实现，那么时间复杂度是$O((|V| + |E|) \log |V|) = O(|E| \log |V|)$；如果用斐波那契堆来实现复杂度可以降低至$O(|E| + |V| \log |V|)$

五、 程序代码

5.1 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M，key 是 vertex，value 是另外的一个子哈希表m，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$v_1 \to v_2$这种类型，第二个记录$v_1 \leftarrow v_2$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	**PRIM-MST**
Input	无向图G，初始节点s
Output	图G的最小生成树
Begin	
Step 1	**f** 存储一个节点集合V，V的初值为起点组成的单元素集合
Step 2	初始化一个图G，G的初值为空集，把V中所有的节点添加到G中
Step 3	通过对V中所有的节点分别进行两次哈希表查询：第一次对G_{in}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$O(1)$。这样就找到与集合V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E里。
Step 4	对于边集合E，进行如下处理：对所有终点不在V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V以及图G中，然后把这条边也添加到图G中。如果图G的节点数量等于输入的图的节点数量，GOTO **End**，返回值为G；否则，返回 **Step 3**
End	

5.2　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](#)。

```cpp
/*
 * 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;
```

```
45      map_t temp_map;
46      Dynamic_Array *temp_Edge = Dynamic_Array_init();        // set of edges connected with V
47
48      // Generally, loop will stop while len(V) = len(g).
49      // If g is not connected, error will be raised.
50      while (V->n < Length_of_Graph_in) {
51          for (i = 1; i <= V->n; i++) {
52              temp = (Vertex)Dynamic_Array_get_Element(V, i);
53              temp_map = Graph_get_adjacent_Vertices(g, temp);        // submap
54              Dynamic_Array *index = hashmap_used_index(temp_map);    // used slots of the submap
55
56              Edge *tmp;
57              int change_memo = temp_Edge->n;
58              for (j = 1; j <= index->n; j++) {
59                  int addr = (int)Dynamic_Array_get_Element(index, j);
60                  tmp = (Edge *)hashmap_select(temp_map, addr);
61
62                  // only if the destination of Edge(tmp) is not
63                  // included in the map(G), the appending
64                  // operation can be done.
65                  if (hashmap_get(ans->outgoing, (char *)tmp->destination, (void **)&arg)
66                      == MAP_MISSING) {
67                      Dynamic_Array_append(temp_Edge, (any)tmp);
68                  }
69              }
70          }
71
72          // find the minimal value
73          int min_location = Dynamic_Array_min(temp_Edge, f_get_double);
74          Edge *min = (Edge *)Dynamic_Array_get_Element(temp_Edge, min_location);
75          temp = min->destination;
76          Dynamic_Array_append(V, temp);
77          Graph_insert_vertex(ans, temp);
78          Graph_insert_edge(ans, min->origin, temp, min->element);
79      }
80      return ans;
81  }
```

程序代码 1

5.3 具体可运行的代码

```
1   /*
2    * Copyright (c) 2018, Liu Peng, School of Mathematics and Statistics, YNU
3    * Apache License.
4    *
5    * 文件名称: Source.cpp
6    * 文件标识: 见配置管理计划书
7    * 摘 要: 测试 Prim 算法
8    *
9    * 当前版本: 1.0
```

```
10    * 作 者：刘鹏
11    * 创建日期：2018 年 7 月 9 日
12    * 完成日期：2018 年 7 月 9 日
13    *
14    * 取代版本：0.9
15    * 原作者 ：刘鹏
16    * 完成日期：
17    */
18
19
20    // C / C++ program for Prim's MST for adjacency list representation of graph
21
22    #include <stdio.h>
23    #include <stdlib.h>
24    #include <limits.h>
25    #include <time.h>
26
27    // A structure to represent a node in adjacency list
28    struct AdjListNode {
29        int dest;
30        int weight;
31        struct AdjListNode* next;
32    };
33
34    // A structure to represent an adjacency liat
35    struct AdjList {
36        struct AdjListNode *head; // pointer to head node of list
37    };
38
39    // A structure to represent a graph. A graph is an array of adjacency lists.
40    // Size of array will be V (number of vertices in graph)
41    struct Graph {
42        int V;
43        struct AdjList* array;
44    };
45
46    // A utility function to create a new adjacency list node
47    struct AdjListNode* newAdjListNode(int dest, int weight) {
48        struct AdjListNode* newNode =
49            (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
50        newNode->dest = dest;
51        newNode->weight = weight;
52        newNode->next = NULL;
53        return newNode;
54    }
55
56    // A utility function that creates a graph of V vertices
57    struct Graph* createGraph(int V) {
58        struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
```

```c
59      graph->V = V;
60
61      // Create an array of adjacency lists. Size of array will be V
62      graph->array = (struct AdjList*) malloc(V * sizeof(struct AdjList));
63
64      // Initialize each adjacency list as empty by making head as NULL
65      for (int i = 0; i < V; ++i)
66          graph->array[i].head = NULL;
67
68      return graph;
69  }
70
71  // Adds an edge to an undirected graph
72  void addEdge(struct Graph* graph, int src, int dest, int weight) {
73      // Add an edge from src to dest. A new node is added to the adjacency
74      // list of src. The node is added at the begining
75      struct AdjListNode* newNode = newAdjListNode(dest, weight);
76      newNode->next = graph->array[src].head;
77      graph->array[src].head = newNode;
78
79      // Since graph is undirected, add an edge from dest to src also
80      newNode = newAdjListNode(src, weight);
81      newNode->next = graph->array[dest].head;
82      graph->array[dest].head = newNode;
83  }
84
85  // Structure to represent a min heap node
86  struct MinHeapNode {
87      int v;
88      int key;
89  };
90
91  // Structure to represent a min heap
92  struct MinHeap {
93      int size;    // Number of heap nodes present currently
94      int capacity; // Capacity of min heap
95      int *pos;    // This is needed for decreaseKey()
96      struct MinHeapNode **array;
97  };
98
99  // A utility function to create a new Min Heap Node
100 struct MinHeapNode* newMinHeapNode(int v, int key) {
101     struct MinHeapNode* minHeapNode =
102         (struct MinHeapNode*) malloc(sizeof(struct MinHeapNode));
103     minHeapNode->v = v;
104     minHeapNode->key = key;
105     return minHeapNode;
106 }
107
```

```
108  // A utilit function to create a Min Heap
109  struct MinHeap* createMinHeap(int capacity) {
110      struct MinHeap* minHeap =
111          (struct MinHeap*) malloc(sizeof(struct MinHeap));
112      minHeap->pos = (int *)malloc(capacity * sizeof(int));
113      minHeap->size = 0;
114      minHeap->capacity = capacity;
115      minHeap->array =
116          (struct MinHeapNode**) malloc(capacity * sizeof(struct MinHeapNode*));
117      return minHeap;
118  }
119
120  // A utility function to swap two nodes of min heap. Needed for min heapify
121  void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b) {
122      struct MinHeapNode* t = *a;
123      *a = *b;
124      *b = t;
125  }
126
127  // A standard function to heapify at given idx
128  // This function also updates position of nodes when they are swapped.
129  // Position is needed for decreaseKey()
130  void minHeapify(struct MinHeap* minHeap, int idx) {
131      int smallest, left, right;
132      smallest = idx;
133      left = 2 * idx + 1;
134      right = 2 * idx + 2;
135
136      if (left < minHeap->size &&
137          minHeap->array[left]->key < minHeap->array[smallest]->key)
138          smallest = left;
139
140      if (right < minHeap->size &&
141          minHeap->array[right]->key < minHeap->array[smallest]->key)
142          smallest = right;
143
144      if (smallest != idx) {
145          // The nodes to be swapped in min heap
146          struct MinHeapNode *smallestNode = minHeap->array[smallest];
147          struct MinHeapNode *idxNode = minHeap->array[idx];
148
149          // Swap positions
150          minHeap->pos[smallestNode->v] = idx;
151          minHeap->pos[idxNode->v] = smallest;
152
153          // Swap nodes
154          swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
155
156          minHeapify(minHeap, smallest);
```

```
157          }
158     }
159
160     // A utility function to check if the given minHeap is ampty or not
161     int isEmpty(struct MinHeap* minHeap) {
162          return minHeap->size == 0;
163     }
164
165     // Standard function to extract minimum node from heap
166     struct MinHeapNode* extractMin(struct MinHeap* minHeap) {
167          if (isEmpty(minHeap))
168              return NULL;
169
170          // Store the root node
171          struct MinHeapNode* root = minHeap->array[0];
172
173          // Replace root node with last node
174          struct MinHeapNode* lastNode = minHeap->array[minHeap->size - 1];
175          minHeap->array[0] = lastNode;
176
177          // Update position of last node
178          minHeap->pos[root->v] = minHeap->size - 1;
179          minHeap->pos[lastNode->v] = 0;
180
181          // Reduce heap size and heapify root
182          --minHeap->size;
183          minHeapify(minHeap, 0);
184
185          return root;
186     }
187
188     // Function to decreasy key value of a given vertex v. This function
189     // uses pos[] of min heap to get the current index of node in min heap
190     void decreaseKey(struct MinHeap* minHeap, int v, int key) {
191          // Get the index of v in heap array
192          int i = minHeap->pos[v];
193
194          // Get the node and update its key value
195          minHeap->array[i]->key = key;
196
197          // Travel up while the complete tree is not hepified.
198          // This is a O(Logn) loop
199          while (i && minHeap->array[i]->key < minHeap->array[(i - 1) / 2]->key) {
200              // Swap this node with its parent
201              minHeap->pos[minHeap->array[i]->v] = (i - 1) / 2;
202              minHeap->pos[minHeap->array[(i - 1) / 2]->v] = i;
203              swapMinHeapNode(&minHeap->array[i], &minHeap->array[(i - 1) / 2]);
204
205              // move to parent index
```

```
206            i = (i - 1) / 2;
207        }
208    }
209
210    // A utility function to check if a given vertex
211    // 'v' is in min heap or not
212    bool isInMinHeap(struct MinHeap *minHeap, int v) {
213        if (minHeap->pos[v] < minHeap->size)
214            return true;
215        return false;
216    }
217
218    // A utility function used to print the constructed MST
219    void printArr(int arr[], int n) {
220        for (int i = 1; i < n; ++i)
221            printf("%d - %d\n", arr[i], i);
222    }
223
224    // The main function that constructs Minimum Spanning Tree (MST)
225    // using Prim's algorithm
226    void PrimMST(struct Graph* graph) {
227        int V = graph->V;   // Get the number of vertices in graph
228
229        // Array to store constructed MST
230        int *parent = (int *)calloc(V, sizeof(int));
231
232        // Key values used to pick minimum weight edge in cut
233        int *key = (int *)calloc(V, sizeof(int));
234
235        // minHeap represents set E
236        struct MinHeap* minHeap = createMinHeap(V);
237
238        // Initialize min heap with all vertices. Key value of
239        // all vertices (except 0th vertex) is initially infinite
240        int v;
241        for (v = 1; v < V; ++v) {
242            parent[v] = -1;
243            key[v] = INT_MAX;
244            minHeap->array[v] = newMinHeapNode(v, key[v]);
245            minHeap->pos[v] = v;
246        }
247
248        // Make key value of 0th vertex as 0 so that it
249        // is extracted first
250        key[0] = 0;
251        minHeap->array[0] = newMinHeapNode(0, key[0]);
252        minHeap->pos[0] = 0;
253
254        // Initially size of min heap is equal to V
```

```
255    minHeap->size = V;
256
257    // In the followin loop, min heap contains all nodes
258    // not yet added to MST.
259    while (!isEmpty(minHeap)) {
260        // Extract the vertex with minimum key value
261        struct MinHeapNode* minHeapNode = extractMin(minHeap);
262        int u = minHeapNode->v; // Store the extracted vertex number
263
264                            // Traverse through all adjacent vertices of u (the extracted
265                            // vertex) and update their key values
266        struct AdjListNode* pCrawl = graph->array[u].head;
267        while (pCrawl != NULL) {
268            int v = pCrawl->dest;
269
270            // If v is not yet included in MST and weight of u-v is
271            // less than key value of v, then update key value and
272            // parent of v
273            if (isInMinHeap(minHeap, v) && pCrawl->weight < key[v]) {
274                key[v] = pCrawl->weight;
275                parent[v] = u;
276                decreaseKey(minHeap, v, key[v]);
277            }
278            pCrawl = pCrawl->next;
279        }
280    }
281
282    // print edges of MST
283    // printArr(parent, V);
284 }
285
286 // Driver program to test above functions
287 int main() {
288    // Let us create the graph given in above fugure
289    FILE *file_input;
290    file_input = fopen("../dataset/grafo-esparso-10000.txt", "r");
291    struct Graph* graph;
292    char a[999], b[999], c[999], v[999];
293    if (file_input) {
294        fscanf(file_input, "%s\n", v);
295        graph = createGraph(atoi(v));
296        while ((fscanf(file_input, "%s %s %s\n", a, b, c)) != EOF) {
297            addEdge(graph, atoi(a), atoi(b), atoi(c));
298        }
299    }
300    fclose(file_input);
301
302    int i;
303    for (i = 0; i < 173; ++i) {
```

```
304        clock_t tStart = clock();
305        PrimMST(graph);
306        printf("Time taken: %fs\n", (double)(clock() - tStart) / CLOCKS_PER_SEC, i + 1);
307    }
308
309    return 0;
310 }
```

程序代码 2

六、 运行结果

6.1   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。

七、 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。

八、 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.