Routing Algorithms Programming
实验报告

学　　院　计算机与信息技术学院

专　　业　计算机科学与技术

年级班别　2018 级

组长　田　震（学号 18301017 ）

组员　王子龙（学号 18281218 ）

组员　周天宸（学号 18301121 ）

组员　武斯全（学号 18231420 ）

组员　万奕晨（学号 18291020 ）

成　　绩

实验题目 Routing Algorithms Programming

一、 实验目的

理解路由算法协议，并通过编程实现不同路由算法，对不同的路由算法能有进一步的理解和掌握。

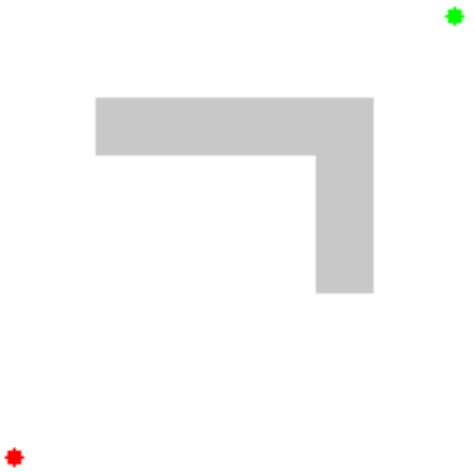
二、 实验环境

操作系统： macOS 11.0.1 Big Sur

编程语言： Java (Java HotSpot(TM) 64-Bit Server VM (build 13.0.1+9, mixed mode, sharing))

集成环境： IntelliJ IDEA 2020.2 (Ultimate Edition)

三、 实验内容和要求



在连线状态算法中，每个节点拥有网络的图谱（一个图）。每个节点将自己可以连接到的其他节点信息发送到网络上所有的节点，而其他节点接着各自将这个信息加入到图谱中。每个路由器即可根据这个图谱来决定从自己到其它节点的最佳路径。

完成这个动作的算法——Dijkstra 算法——创建另一种数据结构——树。节点产生的树将自己视为根节点，且最后这棵树将会包含了网络中所有其他的节点。一开始，此树只有根节点（节点自己）。接着在树中已有的节点的邻居节点且不存在树中的节点集合中，选取一个成本最低的节点加入此树，直到所有节点都存入树中为止。

这棵树即用来创建路由表、提供最佳的“下一个节点”等，让节点能跟网络中其它节点通信。

Dijkstra 算法伪代码：

```
1 procedure Dijkstra(G: 边全为正权的图)
2   {G带有顶点a = v0, v1, v2...和若干边w(vi, vj)}
3    for i := 1 to n
4        D(vi) := ∞
5    D(a) := 0
6    S := ∅
7    while z ∉ S
8    begin
9          u :=不属于S的D(u)最小的一个顶点
10        S := S ∪ {u}
11        for 所有不属于S的顶点v
12            if D(u) + w(u, v) < D(v) then D(v) := D(u) + w(u, v)
13    end{D(z) =从a到z的最短路长度}
```

```cpp
#include <iostream>
#include <limits>
using namespace std;

struct Node { //定义表结点
  int adjvex; //该边所指向的顶点的位置
  int weight;// 边的权值
  Node *next; //下一条边的指针
};

struct HeadNode{ // 定义头结点
    int nodeName; // 顶点信息
    int inDegree; // 入度
    int d; //表示当前情况下起始顶点至该顶点的最短路径,初始化为无穷大
    bool isKnown; //表示起始顶点至该顶点的最短路径是否已知,true 表示已知，false 表示未知
    int parent; //表示最短路径的上一个顶点
    Node *link; //指向第一条依附该顶点的边的指针
};


void createGraph(HeadNode *G, int nodeNum, int arcNum) {
  cout << "开始创建图(" << nodeNum << ", " << arcNum  << ")" << endl;
  //初始化头结点
  for (int i = 0; i < nodeNum; i++) {
    G[i].nodeName = i+1; //位置 0 上面存储的是结点 v1,依次类推
    G[i].inDegree = 0; //入度为 0
    G[i].link = NULL;
  }
  for (int j = 0; j < arcNum; j++) {
    int begin, end, weight;
    cout << "请依次输入 起始边 结束边 权值: ";
```

```
    cin >> begin >> end >> weight;
    // 创建新的结点插入链接表
    Node *node = new Node;
    node->adjvex = end - 1;
    node->weight = weight;
    ++G[end-1].inDegree; //入度加 1
    //插入链接表的第一个位置
    node->next = G[begin-1].link;
    G[begin-1].link = node;
  }
}

void printGraph(HeadNode *G, int nodeNum) {
  for (int i = 0; i < nodeNum; i++) {
    cout << "结点 v" << G[i].nodeName << "的入度为";
    cout << G[i].inDegree << ", 以它为起始顶点的边为: ";
    Node *node = G[i].link;
    while (node != NULL) {
      cout << "v" << G[node->adjvex].nodeName << "(权:" << node->weight << "
)" << "  ";
      node = node->next;
    }
    cout << endl;
  }
}

//得到 begin->end 权重
int getWeight(HeadNode *G, int begin, int end) {
  Node *node = G[begin-1].link;
  while (node) {
    if (node->adjvex == end - 1) {
      return node->weight;
    }
    node = node->next;
  }
}

//从 start 开始，计算其到每一个顶点的最短路径
void Dijkstra(HeadNode *G, int nodeNum, int start) {
  //初始化所有结点
  for (int i = 0; i < nodeNum; i++) {
    G[i].d = INT_MAX; //到每一个顶点的距离初始化为无穷大
    G[i].isKnown = false; // 到每一个顶点的距离为未知数
  }
```

```
75.   G[start-1].d = 0; //到其本身的距离为0
76.   G[start-1].parent = -1; //表示该结点是起始结点
77.   while(true) {
78.   //==== 如果所有的结点的最短距离都已知, 那么就跳出循环
79.   int k;
80.   bool ok = true; //表示是否全部ok
81.   for (k = 0; k < nodeNum; k++) {
82.     //只要有一个顶点的最短路径未知,ok就设置为false
83.     if (!G[k].isKnown) {
84.       ok = false;
85.       break;
86.     }
87.   }
88.   if (ok) return;
89.   //=========================================
90.
91.   //==== 搜索未知结点中d最小的,将其变为known
92.   //==== 这里其实可以用最小堆来实现
93.   int i;
94.   int minIndex = -1;
95.   for (i = 0; i < nodeNum; i++) {
96.     if (!G[i].isKnown) {
97.       if (minIndex == -1)
98.         minIndex = i;
99.       else if (G[minIndex].d > G[i].d)
100.        minIndex = i;
101.    }
102.  }
103.  //=========================================
104.
105.  cout << "当前选中的结点为: v" << (minIndex+1) << endl;
106.    G[minIndex].isKnown = true; //将其加入最短路径已知的顶点集
107.    // 将以minIndex为起始顶点的所有的d更新
108.    Node *node = G[minIndex].link;
109.    while (node != NULL) {
110.      int begin = minIndex + 1;
111.      int end = node->adjvex + 1;
112.      int weight = getWeight(G, begin, end);
113.      if (G[minIndex].d + weight < G[end-1].d) {
114.        G[end-1].d = G[minIndex].d + weight;
115.        G[end-1].parent = minIndex; //记录最短路径的上一个结点
116.      }
117.      node = node->next;
118.    }
```

```
119.   }
120. }
121.
122. //打印到 end-1 的最短路径
123. void printPath(HeadNode *G, int end) {
124.   if (G[end-1].parent == -1) {
125.     cout << "v" << end;
126.   } else if (end != 0) {
127.     printPath(G, G[end-1].parent + 1); // 因为这里的 parent 表示的是下标，从 0
    开始，所以要加 1
128.     cout << " -> v" << end;
129.   }
130. }
131.
132. int main() {
133.   HeadNode *G;
134.   int nodeNum, arcNum;
135.   cout << "请输入顶点个数，边长个数: ";
136.   cin >> nodeNum >> arcNum;
137.   G = new HeadNode[nodeNum];
138.   createGraph(G, nodeNum, arcNum);
139.
140.   cout << "============================" << endl;
141.   cout << "下面开始打印图信息..." << endl;
142.   printGraph(G, nodeNum);
143.
144.   cout << "============================" << endl;
145.   cout << "下面开始运行 dijkstra 算法..." << endl;
146.   Dijkstra(G, nodeNum, 1);
147.
148.   cout << "============================" << endl;
149.   cout << "打印从 v1 开始所有的最短路径" << endl;
150.   for (int k = 2; k <= nodeNum; k++) {
151.     cout << "v1 到 v" << k << "的最短路径为" << G[k-1].d << ": ";
152.     printPath(G, k);
153.     cout << endl;
154.   }
155. }
```

四、 个人贡献说明

（每组人数最多 5 人，请说明完成实验过程中本人的分工或贡献。）

我在本实验中一同参与了代码的编写，还负责了程序的调试。