



# 廣東工業大學

## 课 程 设 计

课程名称 计算机网络课程设计

题目名称 链路状态路由算法的实现

学生学院 计算机学院

专业班级 16 网络二班

学 号 3116004979

学生姓名 詹泽霖

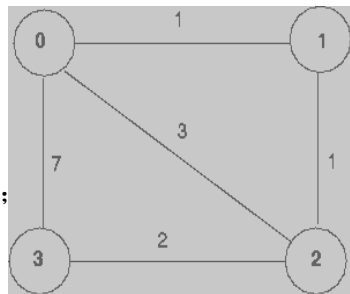
指导教师 黄益民

2018 年 5 月 27 日



# 计算机网络课程设计任务书

|             |   |  |
|-------------|---|--|
| 设计题目        | 链路状态路由算法的实现 ★★  |  |
| 已知技术参数和设计要求 | <p>1.编程实现右图所示简单网络拓扑的链路状态路由算法。</p> <p>1.1 结点之间的连接关系固定；</p> <p>1.2 链路开销可以由用户设定。</p> <p>2.链路状态算法的实现：</p> <p>2.1 链路状态消息的交换(可选，简单起见，可基于静态网络拓扑运行 Dijkstra 算法)；</p> <p>2.2 网络拓扑的描述/构造；</p> <p>2.3 利用 Dijkstra 算法计算路由；</p> <p>2.4 路由表的输出。</p> <p>3.网络拓扑结构的描述(数据结构)，拓扑结构利用文件存储。</p> <p>4.用可视化界面进行程序演示。</p> |  |
| 设计内容与步骤     | <p>1.分析链路状态路由协议与 Dijkstra 算法；</p> <p>2.熟悉线程间通信与同步机制/或进程间通信机制；</p> <p>3.网络拓扑的数据结构定义及文件存储；</p> <p>4.链路状态消息的交换；</p> <p>5.Dijkstra 算法实现；</p> <p>6.结点路由表的显示；</p> <p>7.完成课程设计报告。</p>  |  |
| 设计工作计划与进度安排 | <p>1.熟悉链路状态协议/算法</p> <p>2.链路状态算法的实现方式分析</p> <p>3.链路状态算法实现框架结构设计</p> <p>4.数据结构定义：包括网络拓扑结构、链路状态消息、路由表等</p> <p>5.Dijkstra 算法实现</p> <p>6.课程设计报告</p>   | <p>4 小时</p> <p>4 小时</p> <p>8 小时</p> <p>4 小时</p> <p>10 小时</p> <p>4 小时</p> |



# 目录

|                             |    |
|-----------------------------|----|
| 一. 绪论.....                  | 5  |
| 1.1 设计环境.....               | 5  |
| 1.2 设计思想.....               | 5  |
| 1.2.1 算法思想 .....            | 5  |
| 1.2.2 算法原理 .....            | 5  |
| 1.2.3 链路状态路由协议.....         | 5  |
| 二.链路状态路由算法的实现.....          | 6  |
| 2.1 程序结构图.....              | 6  |
| 2.2 系统详细设计.....             | 6  |
| 2.2.1 头文件 Header.h.....     | 6  |
| 2.2.2 关键算法与函数接口实现.....      | 7  |
| 2.2.2.1 Dijkstra 函数.....    | 7  |
| 2.2.2.2 OutputGraph 函数..... | 8  |
| 2.2.2.3 OutputRoute 函数..... | 9  |
| 三. 功能测试.....                | 10 |
| 3.1 功能面板设计.....             | 10 |
| 3.2 初始化路由表测试.....           | 11 |
| 3.3 创建路由表测试.....            | 11 |
| 3.4 查找路由信息功能测试.....         | 12 |
| 3.5 输出路由拓扑功能测试.....         | 13 |
| 3.6 输出路由表功能测试.....          | 14 |
| 四. 设计总结与体会.....             | 14 |
| 4.1 错误总结.....               | 14 |
| 4.1 部分优化总结.....             | 14 |
| 4.2.1 输出优化.....             | 14 |
| 4.2.2 测试界面优化.....           | 15 |
| 五. 参考文献.....                | 15 |

# 正文

## 一. 诸论

### 1.1 设计环境

操作系统: win10

IDE: Dev-C++ 5.11

编程语言: C++

### 1.2 设计思想

#### 1.2.1 算法思想

链路状态算法的思想是要求网络中所有参与链路状态路由协议的路由器都掌握网络的全部拓扑结构信息,并记录在路由数据库中。链路状态算法中路由数据库实质上是一个网络结构的拓扑图,该拓扑图由一个节点的集合和一个边的集合构成。在网络拓扑图中,结点代表网络中路由器,边代表路由器之间的物理链路。在网络拓扑结构图中,每一条链路上可以附加不同的属性,例如链路的状态、距离或费用等。如果每一个路由器所保存的网络拓扑结构图都是一致的,那么每个路由器生成的路由表也是最佳的,不存在错误路由或循环路由。

#### 1.2.2 算法原理

链路状态路由算法的工作原理如下

(1) 在参与链路状态选路的路由器集合中,每个路由器都需要通过某种机制来了解自己所连接的链路及其状态。

(2) 各路由器都能够将其所连接的链路的状态信息通知给网络中的所有其他路由器,这些链路信息包括链路状态、费用以及链路两端的路由器等。

(3) 链路状态信息的通过链路状态分组(LSP)来向整个网络发布。一个LSP通常包含源路由器的标识符、相邻路由器的标识符,以及而知之间链路的费用。每一个LSP都将被网络中的所有的路由器接收,并用于建立网络整体的统一拓扑数据库。由于网络中所有的路由器都发送LSP,经过一段时间以后,每一个路由器都保持了一张完整的网络拓扑图,再在这个拓扑图上,利用最短通路算法(例如Dijkstra算法等),路由器就可以计算出从任何源点到任何目的地的最佳通路。这样,每一个路由器都能够利用通路最短的原则建立一个以本路由器为根、分支到所有其他路由器的生成树,依据这个生成树就可以很容易地计算出本路由器的路由表。

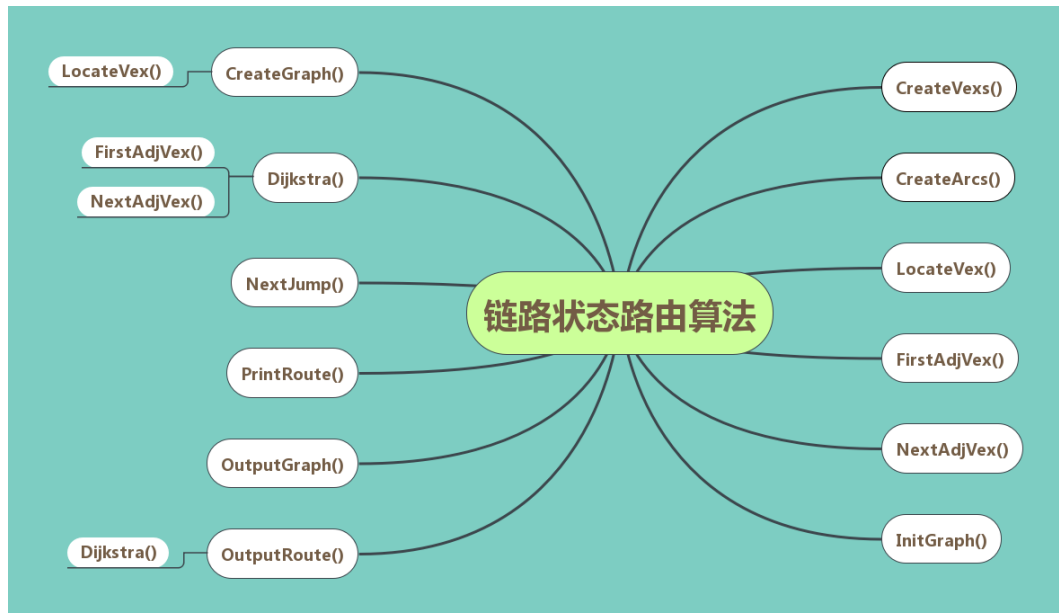
#### 1.2.3 链路状态路由协议

链路状态路由选择协议又称为最短路径优先协议,它基于Edsger Dijkstra的最短路径优先(SPF)算法。它比距离矢量路由协议复杂得多,但基本功能和配置却很简单,甚至算法也容易理解。路由器的链路状态的信息称为链路状态,包括:接口的IP地址和子网掩码、网络类型(如以太网链路或串行点对点链路)、该链路的开销、该链路上的所有的相邻路由器。

链路状态路由协议是层次式的,网络中的路由器并不向邻居传递“路由项”,而是通告给邻居一些链路状态。与距离矢量路由协议相比,链路状态协议对路由的计算方法有本质的差别。距离矢量协议是平面式的,所有的路由学习完全依靠邻居,交换的是路由项。链路状态协议只是通告给邻居一些链路状态。运行该路由协议的路由器不是简单地从相邻的路由器学习路由,而是把路由器分成区域,收集区域的所有的路由器的链路状态信息,根据状态信息生成网络拓扑结构,每一个路由器再根据拓扑结构计算出路由。

## 二. 链路状态路由算法的实现

### 2.1 程序结构图



### 2.2 系统详细设计

#### 2.2.1 头文件 Header.h

功能：存储必要的变量，定义存储结构以及定义函数

代码实现：

```
#ifndef HEADER_H
#define HEADER_H

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<limits.h>
#include<iostream>
using namespace std;
#define UNVISITED 0 //未被访问
#define VISITED 1 //被访问
#define UNSELECTED 0 //未被选择
#define SELECTED 1 //被选择
#define INFINITY INT_MAX //计算机允许的整数最大值
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -1
#define MaxSize 10
#define NOLINK 0
typedef int Status;
```

```

typedef struct{
    int n;           //顶点个数
    int e;           //边的个数
    int *tags;       //访问状态数组
    int **arcs;      //关系数组
    char *vexs;      //顶点数组
}Graph;

typedef struct{
    char v;          //边的起点
    char w;          //边的终点
}ArcInfo;

typedef struct DistInfo{
    int prev;        //当前最短路径上该顶点的前驱顶点位序
    int shortest;     //当前最短路径长度
}DistInfo,*pDistInfo;

Status CreateVexs(char *vexs);           //创建顶点数组
Status CreateArcs(ArcInfo *arcs,int n);  //创建关系数组
int LocateVex(Graph G,char v);          //获取顶点相应的位序
int FirstAdjVex(Graph G,int k);         //寻找第一个邻接节点
int NextAdjVex(Graph G,int k,int m);    //寻找下一个邻接节点
Status InitGraph(Graph &G,char *vexs,int n,ArcInfo *arcs,int e); //初始化路由表
Status CreateGraph(Graph &G,ArcInfo *arcs); //创建路由表
Status Dijkstra(Graph G,int i,pDistInfo &Dist); //迪杰斯特拉算法寻找最短路径
char NextJump(Graph G,DistInfo *Dist,int k); //寻找路由器的下一跳
void PrintRoute(Graph G,DistInfo *Dist,int k); //输出最短路由路径
Status OutputGraph(Graph &G,int VEXSSIZE); //输出路由拓扑图
Status OutputRoute(Graph &G,int VEXSSIZE); //输出各节点的路由信息
#endif

```

## 2.2.2 关键算法与函数接口实现

### 2.2.2.1 Dijkstra 函数

**功能：**实现迪杰斯特拉算法寻找最短路由路径

**代码实现：**

```

Status Dijkstra(Graph G,int i,pDistInfo &Dist){
//迪杰斯特拉算法寻找最短路径
    int k,min;
    Dist=(DistInfo *)malloc(sizeof(DistInfo)*G.n); //分配空间
    for(int j=0;j<G.n;j++){
        Dist[j].shortest=INFINITY; //初始化一开始顶点 i 到顶点 j 的最短路径长度都为无限大
        G.tags[j]=UNSELECTED;      //初始化访问状态
    }
    Dist[i].prev=-1;                //初始化顶点 i 的前驱
    Dist[i].shortest=0;             //初始化最短距离
}

```

```

G.tags[i]=SELECTED;                //更新访问状态

for(int j=FirstAdjVex(G,i);j>=0;j=NextAdjVex(G,i,j)){
    Dist[j].prev=i;                //更新顶点 i 邻接节点的前驱
    Dist[j].shortest=G.arcs[i][j]; //顶点 i 到邻接顶点 j 的最短路径
}

for(int m=1;m<G.n;m++){            //按路径长度升序，依次求源点到其他顶点的最短路径
    min=INFINITY;
    k=0;
    for(int j=0;j<G.n;j++){
        if(G.tags[j]==UNSELECTED && Dist[j].shortest<min){ //找到最短的
            k=j;
            min=Dist[k].shortest;
        }
    }
    G.tags[k]=SELECTED;            //更新访问状态
    for(int j=FirstAdjVex(G,k);j>=0;j=NextAdjVex(G,k,j)){ //更新 Dist 数组
        if(G.tags[j]==UNSELECTED&&Dist[k].shortest!=INFINITY&&
            Dist[k].shortest+G.arcs[k][j]<Dist[j].shortest){
            //到 j 最短路径为从源点直接到 j 的弧；或者是经过已某条求得的最短路径再到 j
            Dist[j].shortest=Dist[k].shortest+G.arcs[k][j];
            //更新最短路径
            Dist[j].prev=k;         //更新前驱节点
        }
    }
}
}

```

#### 2. 2. 2. 2 OutputGraph 函数

**功能：**输出路由拓扑图

**代码实现：**

```

Status OutputGraph(Graph &G,int VEXSSIZE){
//输出路由拓扑图
    int i,j;
    FILE *fp=fopen("Graph.txt","w");
    if(fp==NULL){
        cout<<"The file can't open.";
        exit(0);
    }
    else{
        fprintf(fp,"*****路由拓扑表*****\n");
        for(int i=0;i<VEXSSIZE;i++){
            for(j=0;j<VEXSSIZE;j++){

```



```

        if(G.arcs[i][j]!=INFINITY)
            fprintf(fp,"    %-6d",G.arcs[i][j]);
        else
            fprintf(fp,"    %-6d",0);
    }
    fprintf(fp,"\n");
}
fprintf(fp,"    注 : %d 表示无边相连\n",NOLINK);
fprintf(fp,"*****\n");
fclose(fp);
cout<<"路由拓扑表 Graph.txt 输出成功！"<<endl;
}
return 0;
}

```

### 2.2.2.3 OutputRoute 函数

**功能：**输出各节点的路由信息

**代码实现：**

```

Status OutputRoute(Graph &G,int VEXSSIZE){
//输出各节点的路由信息
    int i,j;
    FILE *fp=fopen("Route.txt", "w");
    DistInfo *Dist=(DistInfo *)malloc(sizeof(DistInfo)*VEXSSIZE);
    if(fp==NULL){
        cout<<"The file can't open.";
        exit(0);
    }
    else{
        for(i=0;i<VEXSSIZE;i++){
            fprintf(fp,"*****\n");
            Dijkstra(G,i,Dist);
            fprintf(fp,"          \t  %d 的路由表\n",i);
            fprintf(fp,"          目的路由      下一路由      最短距离\n");
            for(j=0;j<VEXSSIZE;j++){
                fprintf(fp,"          \t%d          \t          %c\n",j,NextJump(G,Dist,j),Dist[j].shortest);
            }
            fprintf(fp,"*****\n");
            fclose(fp);
        }
        cout<<"路由信息表 Route.txt 输出成功！"<<endl;
        return 0;
    }
}

```

### 三. 功能测试

#### 3.1 功能面板设计

代码实现:

```
while(1){
    cout<<"*****Operation Table*****"<<endl;
    cout<<" 1.初始化路由表  ";
    cout<<" 2.创建路由表"    <<endl;
    cout<<" 3.查找路由信息  ";
    cout<<" 4.输出路由拓扑"  <<endl;
    cout<<" 5.输出路由表    ";
    cout<<" 6.退出模拟"      <<endl;
    cout<<"*****"<<endl;
    cout<<"Enter number to choose operation: ";
    scanf("%d",&i);
    switch(i){
        case 1:{
            InitGraph(G,vexs,VEXSSIZE,arcs,ARCSSIZE);
            break;
        }
        case 2:{
            CreateGraph(G,arcs);
            break;
        }
        case 3:{
            cout<<"请输入源结点: ";
            cin>>start;

            Dijkstra(G,start,Dist);
            for(i=0;i<VEXSSIZE;i++){
                cout<<"*****"<<endl;
                printf("目标结点 : %c\n",G.vexs[i]);
                cout<<"最短路径是 : ";
                PrintRoute(G,Dist,i);
                printf("\n 最短路径长度 : %d\n",Dist[i].shortest);
                NextJump(G,Dist,i);
            }
            break;
        }
        case 4:{
            OutputGraph(G,VEXSSIZE);
            break;
        }
    }
}
```

```

        case 5:{
            OutputRoute(G,VEXSSIZE);
            break;
        }
        case 6:{
            cout<<"退出模拟成功！";
            exit(-1);
            break;
        }
        default:{
            cout<<"操作编号输入错误，请重新选择操作编号"<<endl;
            break;
        }
    }
}

```

面板图示：

```

*****Operation Table*****
1.初始化路由表    2.创建路由表
3.查找路由信息    4.输出路由拓扑
5.输出路由表      6.退出模拟
*****
Enter number to choose operation: _

```

### 3.2 初始化路由表测试

选择操作编号 1

```

*****
Enter number to choose operation: 1
初始化路由表成功

```

### 3.3 创建路由表测试

因为，任何一个操作都要在确定路由表结点个数的情况下进行，所以在程序的一开始设置了路由表结点个数的输入。这一功能在初始化路由表方可进行。

```

*****
请输入路由节点个数：

```

测试过程使用 4 个路由节点的情况进行测试，选择操作编号 2，测试结果如下。

```

*****
请输入路由节点个数：
4

```

```

*****
Enter number to choose operation: 2
请输入第0个节点到第1个节点的权值： 3
请输入第0个节点到第2个节点的权值： 5
请输入第0个节点到第3个节点的权值： 7
请输入第1个节点到第2个节点的权值： 1
请输入第1个节点到第3个节点的权值： 2
请输入第2个节点到第3个节点的权值： 3
创建路由表成功

```

### 3.4 查找路由信息功能测试

这一功能在创建路由表完成后进行，选择操作编号 3，测试结果如下。

源节点为 0:

```
Enter number to choose operation: 3
请输入源结点: 0
目标结点 : 0
最短路径是 : 0
最短路径长度 : 0
目标结点 : 1
最短路径是 : 0 1
最短路径长度 : 3
目标结点 : 2
最短路径是 : 0 1 2
最短路径长度 : 4
目标结点 : 3
最短路径是 : 0 1 3
最短路径长度 : 5
```

源节点为 1:

```
Enter number to choose operation: 3
请输入源结点: 1
目标结点 : 0
最短路径是 : 1 0
最短路径长度 : 3
目标结点 : 1
最短路径是 : 1
最短路径长度 : 0
目标结点 : 2
最短路径是 : 1 2
最短路径长度 : 1
目标结点 : 3
最短路径是 : 1 3
最短路径长度 : 2
```

源节点为 2:

```

Enter number to choose operation: 3
请输入源结点: 2

目标结点 : 0
最短路径是 : 2 1 0
最短路径长度 : 4

目标结点 : 1
最短路径是 : 2 1
最短路径长度 : 1

目标结点 : 2
最短路径是 : 2
最短路径长度 : 0

目标结点 : 3
最短路径是 : 2 3
最短路径长度 : 3

```

源节点为 3:

```

Enter number to choose operation: 3
请输入源结点: 3

目标结点 : 0
最短路径是 : 3 1 0
最短路径长度 : 5

目标结点 : 1
最短路径是 : 3 1
最短路径长度 : 2

目标结点 : 2
最短路径是 : 3 2
最短路径长度 : 3

目标结点 : 3
最短路径是 : 3
最短路径长度 : 0

```

### 3.5 输出路由拓扑功能测试


该功能在创建路由表的基础上进行，选择操作编号 4，测试输出如下。

```

Enter number to choose operation: 4
路由拓扑表Graph.txt输出成功!

```

打开 Graph.txt 文件。

 Graph.txt - 记事本

```

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)
*****路由拓扑表*****
0      3      5      7
3      0      1      2
5      1      0      3
7      2      3      0
注 : 0表示无边相连
*****

```

### 3.6 输出路由表功能测试

该功能在创建路由表的基础上进行，选择操作编号 5，测试输出如下。

```
*****
Enter number to choose operation: 5
路由信息表Route.txt输出成功!
```

打开 Route.txt 文件。

```
*****
0的路由表
目的路由  下一路由  最短距离
0          X          0
1          1          3
2          1          4
3          1          5
*****
1的路由表
目的路由  下一路由  最短距离
0          0          3
1          X          0
2          2          1
3          3          2
*****
2的路由表
目的路由  下一路由  最短距离
0          1          4
1          1          1
2          X          0
3          3          3
*****
3的路由表
目的路由  下一路由  最短距离
0          1          5
1          1          2
2          2          3
3          X          0
*****
```

## 四. 设计总结与体会

### 4.1 错误总结

(1) 理解算法时有误，一味的使用贪心算法，与原来算法本意大相径庭，实现时最短路径测试中出现错误。

(2) 在写 PrintRoute 函数的时候，对于递归输出实现有误，多次爆栈。

(3) 在初始化的过程中一开始并没有对访问状态进行定义，导致进行迪杰斯特拉算法过程中出现错误。

### 4.1 部分优化总结

#### 4.2.1 输出优化

在输出路由表时，一开始单单使用文字输出，在直观上不够鲜明的表示路由路径。改进后，使用类似列表的方法进行输出。

```
*****
0的路由表
目的路由  下一路由  最短距离
0          X          0
1          1          3
2          1          4
3          1          5
*****
```

#### 4.2.2 测试界面优化

在保持原本接口不变的情况下，写了 TestBoard 函数，自行创建和进行各种路由的操作，使得界面更直观，代码结构更清晰。

```
*****Operation Table*****
1.初始化路由表    2.创建路由表
3.查找路由信息    4.输出路由拓扑
5.输出路由表      6.退出模拟
```

## 五. 参考文献

- 【1】《数据结构与算法》，张铭，王腾蛟，赵海燕，高等教育出版社
- 【2】《数据结构与算法—学习指导与习题解析》，张铭，王腾蛟，赵海燕，高等教育出版社
- 【3】《C++ primer plus》，Stephen Prata, 人民邮电出版社
- 【4】《图论及其应用》，张先迪，李正良，高等教育出版社