

哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法

课程类型：必修

实验项目：图形结构及其应用

实验题目：图形结构的应用算法

实验日期：2020 年 11 月 16 日

班级：

学号：

姓名：Youngsc

设计成绩	报告成绩	指导老师
		张岩

一、 实验目的

1. 掌握图的邻接矩阵、邻接表等不同存储形式的表示方法。
2. 掌握图的两种不同遍历方法的基本思想并能编程实现。
3. 掌握构造最小生成树的两种算法思想，并能编程实现。
4. 掌握求单源最短路径和任意两顶点之间的最短路径的算法。
5. 掌握求关键路径的算法，并能编程实现。
6. 能够灵活运用图的相关算法解决相应的实际问题。

二、实验要求及实验环境

（一）实验要求

1. 实现单源最短路径的 Dijkstra 算法，输出源点及其到其他顶点的最短路径长。
2. 利用堆结构（实现的优先级队列），改进和优化 Dijkstra 算法的实现；
3. 实现全局最短路径的 Floyd-Warshall 算法。计算任意两个顶点间的最短距离矩阵和最短路径矩阵，并输出任意两个顶点间的最短路径长度和最短路径。
4. 利用 Dijkstra 或 Floyd-Warshall 算法解决单目标最短路径问题：找出图中每个顶点 v 到某个指定顶点 c 最短路径。
5. 利用 Dijkstra 或 Floyd-Warshall 算法解决单顶点对间最短路径问题：对于某对顶 u 和 v ，找出 u 到 v 和 v 到 u 的一条最短路径。
6. （选做）利用 Floyd-Warshall 算法，计算有向图的可达矩阵，理解可达矩阵的含义；
7. 以文件形式输入图的顶点和边，并显示相应的结果。要求顶点不少于 10 个，边不少于 13 个；
8. 软件功能结构安排合理，界面友好，便于使用。

（二）实验环境

1. 硬件环境
 - a) Legion Y7000P 2019 PG0
 - b) CPU: Intel(R)_Core(TM)_i7-9750H_CPU_@_2.60GHz 2.59GHz
 - c) 内存(RAM): 16GB DDR4
2. 系统环境
 - a) Windows10 家庭中文版
3. 开发工具
 - a) Dev C++5.11
 - b) gcc (GCC) 9.2.0

(三)

三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系）

本实验存图方式采用邻接表存图，在储存原图的同时，将所有边反向相储存新图，以便于解决单目标最短路径问题。

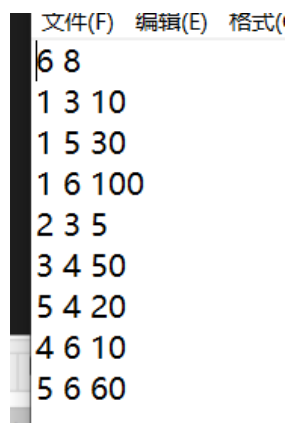
算法部分分为两部分，分别用两个命名空间表示，一个是 Dijkstra，另一个是 Floyd-Warshall。

在 Dijkstra 部分中，可以实现单源最短路径问题和单目标最短路径问题，单目标最短路径问题我们可以通过讲图所有便反向来使其转化成单源最短路径问题。通过堆的优化使 Dijkstra 的时间复杂度得到优化，从而更高效地解决问题，而在 Dijkstra 的 Main 函数中，我们可以通过判断传入的图为原图还是反向后的图来判断解决的问题使单源最短路径问题还是单目标最短路径问题，从而输出相应的提示信息。在输出信息时，最短距离直接调用 Dijkstra 的结果数组即可，最短路径需要在 Dijkstra 的过程中记录每个点的前驱节点，从而采用递归的方式或者循环嵌套的方式来输出路径。

在 Floyd-Warshall 部分中，我们分别用 Main, Main2, Main3 函数来解决任意两点间最短路径问题，给定点对间的最短路径问题和可达矩阵。输出路径时，只需要枚举所有点，查询该点是否在所查询路径上，然后递归输出即可。

四、测试结果

输入数据如下：



```
文件(F) 编辑(E) 格式(O)
6 8
1 3 10
1 5 30
1 6 100
2 3 5
3 4 50
5 4 20
4 6 10
5 6 60
```

功能一样例输出：

```

1 ----- 6、求最短路径 -----
请输入源点：
3
请输入要查询的点的编号，返回请输入-1,全部输出请输入0
4
源点3到点4的最短距离为： 50
源点3到点4的最短路径为： 3-->4
-----输出完毕-----

请输入要查询的点的编号，返回请输入-1,全部输出请输入0
2
源点3到点2不可达
请输入要查询的点的编号，返回请输入-1,全部输出请输入0
5
源点3到点5不可达
请输入要查询的点的编号，返回请输入-1,全部输出请输入0
3
源点3到点3的最短距离为： 0
源点3到点3的最短路径为： 3
-----输出完毕-----

请输入要查询的点的编号，返回请输入-1,全部输出请输入0
1
源点3到点1不可达
请输入要查询的点的编号，返回请输入-1,全部输出请输入0
6
源点3到点6的最短距离为： 60
源点3到点6的最短路径为： 3-->4-->6
-----输出完毕-----

请输入要查询的点的编号，返回请输入-1,全部输出请输入0
1
----- 功能选择 -----

```

```

1 ----- 6、求最短路径 -----
请输入源点：
3
请输入要查询的点的编号，返回请输入-1,全部输出请输入0
0
源点3到点1不可达
源点3到点2不可达
源点3到点3的最短距离为： 0
源点3到点3的最短路径为： 3
-----输出完毕-----

源点3到点4的最短距离为： 50
源点3到点4的最短路径为： 3-->4
-----输出完毕-----

源点3到点5不可达，以及二叉树的一些基
源点3到点6的最短距离为： 60
源点3到点6的最短路径为： 3-->4-->6
-----输出完毕-----

请输入要查询的点的编号，返回请输入-1,全部输出请输入0

```

功能二样例输出：

```

2 ----- 5、求可达矩阵 -----
请输入要查询路径的起点，输入-1退出：
1
请输入要查询路径的终点：
3
点1到点3的最短距离是： 10
点1到点3的最短路径是：
1->3
请输入要查询路径的起点，输入-1退出：
-1
----- 功能选择 -----
1、求给定起点的最短路径

```

功能三样例输出：

```

3
请输入终点:
3
请输入要查询的点的编号, 返回请输入-1, 全部输出请输入0
0
点1到终点3的最短距离为: 10
点1到终点3的最短路径为: 1->3
-----输出完毕-----

点2到终点3的最短距离为: 5
点2到终点3的最短路径为: 2->3
-----输出完毕-----

点3到终点3的最短距离为: 0
点3到终点3的最短路径为: 3
-----输出完毕-----

点4到终点3不可达
点5到终点3不可达
点6到终点3不可达
请输入要查询的点的编号, 返回请输入-1, 全部输出请输入0

```

功能四样例输出:

```

4
请输入点对中的第一个点, 输入-1退出:
2
请输入点对中的另一点:
3
点2到点3的最短距离是: 5
点2到点3的最短路径是:
2->3
点3到点2无法到达
-----输出完毕-----

```

功能五样例输出:

```

5
可达矩阵为:
      1  2  3  4  5  6
-----
1 | 1  0  1  1  1  1
2 | 0  1  1  1  0  1
3 | 0  0  1  1  0  1
4 | 0  0  0  1  0  1
5 | 0  0  0  1  1  1
6 | 0  0  0  0  0  1
----- 输出结束 -----

```

五、经验体会与不足

通过本次实验我学到了求最短路径的两种算法 Dijkstra 和 Floyd-Warshall, 并了解了两种算法的原理以及两种算法的差异。其中 Dijkstra 算法可以利用堆优化进行加速。同时加深了我对邻接表存图以及对堆这种数据结

构的理解与认知。

六、附录：源代码（带注释）

```
# include <bits/stdc++.h>
# define MAXN 30
# define inf 1000000000
using namespace std;

struct edge{
    int to,val;
    edge(){to = val = 0;}
    edge(int _to,int _val){to = _to,val = _val;}
};

struct Graph{
    vector <edge> List[MAXN];
    int n,m;
    bool isRec;
    Graph(){n=m=isRec=0;}
}gra,rgra;

struct Heap{
    int size;
    pair <int,int> queue[MAXN];
    Heap()          //初始化
    {
        size=0;
    }
    void shift_up(int i)
    {
        while(i>1)
        {
            if(queue[i]<queue[i>>1]) swap(queue[i],queue[i>>1]);
            i>>=1;
        }
    }

    void shift_down(int i)  //下沉
    {
        while ((i<<1)<=size)
        {
            int next=i<<1;
            if (next<size && queue[next+1]<queue[next]) next++;
            if (queue[i]>queue[next])
            {
                swap(queue[i],queue[next]);
                i=next;
            }
            else return;
        }
    }

    void push(pair <int,int> x)  //加入元素
    {
        queue[++size]=x;
        shift_up(size);
    }

    void pop()          //弹出操作
    {
        swap(queue[1],queue[size]);
        size--;
    }
}
```

```

        shift_down(1);
    }

    pair <int,int> top(){return queue[1];}
    bool empty(){return !size;}

}q;

void Input(){
    freopen("dir.txt","r",stdin);
    scanf("%d%d",&gra.n,&gra.m);
    rgra.n = gra.n;
    rgra.m = gra.m;
    rgra.isRec = 1;
    int x,y,z;
    for (int i=1; i<=gra.m; ++i)
    {
        scanf("%d%d%d",&x,&y,&z);
        gra.List[x].push_back(edge(y,z));
        rgra.List[y].push_back(edge(x,z));
    }
    printf("数据读取完毕! \n");
    freopen("CON","r",stdin);
}

namespace DIJ{
    int dis[MAXN],pre[MAXN],s;

    void print(int x){
        if (x == -1) return;
        print(pre[x]);
        printf("%d-->",x);
    }

    void Dijkstra(int s,Graph &G){
        for (int i=1; i<=G.n; ++i) dis[i] = inf,pre[i] = 0;
        pre[s] = -1;
        dis[s] = 0;
        q.push(make_pair(dis[s],s));
        while (!q.empty())
        {
            int now=q.top().second,nowdis=q.top().first;
            q.pop();
            if (dis[now] != nowdis) continue;
            for (auto x:G.List[now])
            {
                if (dis[x.to] <= dis[now]+x.val) continue;
                dis[x.to] = dis[now]+x.val;
                pre[x.to] = now;
                q.push(make_pair(dis[x.to],x.to));
            }
        }
    }

    void doit(int x,Graph &G){
        if (dis[x] == inf)
        {
            if (!G.isRec) printf("源点%d 到点%d 不可达\n",s,x);
            else printf("点%d 到终点%d 不可达\n",x,s);
            return;
        }
        if (!G.isRec) printf("源点%d 到点%d 的最短距离为: ",s,x);
        else printf("点%d 到终点%d 的最短距离为: ",x,s);
        printf("%d\n",dis[x]);
        if (!G.isRec)

```

```

    {
        printf("源点%d 到点%d 的最短路径为: ",s,x);
        print(pre[x]);
        printf("%d\n",x);
    }
    else
    {
        printf("点%d 到终点%d 的最短路径为: ",x,s);
        int now = x;
        while (now != s)
        {
            printf("%d->",now);
            now = pre[now];
        }
        printf("%d\n",s);
    }

    printf("-----输出完毕-----\n\n");
}

bool qury(Graph &G){
    int x;
    printf("请输入要查询的点的编号, 返回请输入-1,全部输出请输入 0\n");
    scanf("%d",&x);
    if (x == -1) return 0;
    if (x == 0)
    {
        for (int i=1; i<=G.n; ++i) doit(i,G);
        return 1;
    }
    if (x > G.n || x < 1)
    {
        printf("输入数据超出范围, 请重新输入: \n\n");
        return 1;
    }
    doit(x,G);
    return 1;
}

Main(Graph &G){
    if (!G.isRec) printf("请输入源点: \n");
    else printf("请输入终点: \n");
    scanf("%d",&s);
    while (s<1 || s>G.n)
    {
        if (!G.isRec) printf("输入源点超出范围, 请重新输入: \n");
        else printf("输入终点超出范围, 请重新输入: \n");
        scanf("%d",&s);
    }
    Dijkstra(s,G);
    while (qury(G));
}

}

namespace Floyed{
    int dis[MAXN][MAXN];
    Make_Matrix(Graph &G){
        for (int i=1; i<=G.n; ++i)
            for (int j=1; j<=G.n; ++j)
            {
                if (i == j) dis[i][j] = 0;
                else dis[i][j] = inf;
            }
    }
}

```



```

        for (int i=1; i<=G.n; ++i)
            for (auto x:G.List[i])
                dis[i][x.to] = min(x.val,dis[i][x.to]);

        for (int k=1; k<=G.n; ++k)
            for (int i=1; i<=G.n; ++i)
                for (int j=1; j<=G.n; ++j)
                    dis[i][j] = min(dis[i][j],dis[i][k]+dis[k][j]);
    }

void qury(int s,int t,Graph &G){
    if (dis[s][t] == inf)
    {
        printf("点%d 到点%d 无法到达\n",s,t);
        return;
    }
    printf("点%d 到点%d 的最短距离是: %d\n",s,t,dis[s][t]);
    printf("点%d 到点%d 的最短路径是: \n",s,t);
    int now = s;
    while (now != t){
        printf("%d->",now);
        for (int i=1; i<=G.n; ++i)
        {
            if (i == now) continue;
            if (dis[now][i]+dis[i][t] == dis[now][t])
            {
                now = i;
                break;
            }
        }
    }
    printf("%d\n",now);
}

void Main(Graph &G){
    Make_Matrix(G);
    int s,t;
    while (1)
    {
        printf("请输入要查询路径的起点, 输入-1 退出: \n");
        scanf("%d",&s);
        if (s == -1) break;
        while (s<1 || s>G.n)
        {
            printf("输入数据超出范围, 请重新输入: ");
            scanf("%d",&s);
        }
        printf("请输入要查询路径的终点: \n");
        scanf("%d",&t);
        while (t<1 || t>G.n)
        {
            printf("输入数据超出范围, 请重新输入: ");
            scanf("%d",&t);
        }
        qury(s,t,G);
    }
}

void Main2(Graph &G){
    Make_Matrix(G);
    int s,t;
    while (1)
    {
        printf("请输入点对中的第一个点, 输入-1 退出: \n");

```

```

scanf("%d",&s);
if (s == -1) break;
while (s<1 || s>G.n)
{
    printf("输入数据超出范围, 请重新输入: ");
    scanf("%d",&s);
}
printf("请输入点对中的另一点: \n");
scanf("%d",&t);
while (t<1 || t>G.n)
{
    printf("输入数据超出范围, 请重新输入: ");
    scanf("%d",&t);
}
qry(s,t,G);
qry(t,s,G);
printf("-----输出完毕-----\n\n");
}

void Main3(Graph &G){
    memset(dis,0,sizeof(dis));
    for (int i=1; i<=G.n; ++i)
        for (auto x:G.List[i])
            dis[i][x.to] = 1;

    for (int i=1; i<=G.n; ++i) dis[i][i] = 1;

    for (int k=1; k<=G.n; ++k)
        for (int i=1; i<=G.n; ++i)
            for (int j=1; j<=G.n; ++j)
                dis[i][j] |= (dis[i][k]&dis[k][j]);

    printf(" 可达矩阵为: \n  ");
    for (int i=1; i<=G.n; ++i) printf("%3d",i);
    printf("\n  ----- \n");
    for (int i=1; i<=G.n; ++i,printf("\n"))
    {
        printf("%2d |",i);
        for (int j=1; j<=G.n; ++j) printf("%3d",dis[i][j]);
    }
    printf("\n ----- 输出结束 ----- \n\n");
}

int main()
{
    Input();
    while (1)
    {
        int opt;
        printf(" ----- 功能选择 ----- \n");
        printf("          1、求给定起点的最短路径\n");
        printf("          2、求任意两点间的最短路径\n");
        printf("          3、求给定终点的最短路径\n");
        printf("          4、求给定点对的最短路径\n");
        printf("          5、求可达矩阵\n");
        scanf("%d",&opt);
        if (opt == 1) DIJ::Main(gra);
        else if (opt == 2) Floyd::Main(gra);
        else if (opt == 3) DIJ::Main(rgra);
        else if (opt == 4) Floyd::Main2(gra);
        else if (opt == 5) Floyd::Main3(gra);
        else printf("输入序号错误, 请重新输入: ");
    }
}

```

```
    }  
    return 0;  
}
```