重 庆 大 学

学 生 实 验 报 告

实验课程名称　　　数据结构与算法

开课实验室　　　DS1501

学　　院　软件学院　年级　2021　专业班　软件 X 班

学 生 姓 名　XXX　学　号　2021XXXX

开 课 时 间　2022　至　2023　学年第　1　学期

总 成 绩	
教师签名	XXX

软件学院制

《数据结构与算法》实验报告

开课实验室：DS1501 **2022 年 11 月 24 日**

<table>
<tr><td>学院</td><td>软件学院</td><td colspan="2">年级、专业、班</td><td>2021 级软件工程 X 班</td><td>姓名</td><td>XXX</td><td>成绩</td><td></td></tr>
<tr><td>课程名称</td><td colspan="2">数据结构与算法</td><td colspan="2">实验项目名称</td><td>2022-2023 学年第一学期数据结构与算法上机练习 006</td><td colspan="2">指导教师</td><td>XXX</td></tr>
<tr><td>教师评语</td><td colspan="8">教师签名：
年 月 日</td></tr>
</table>

一、 实验目的

- 请认真学习第 11 章“图”的各项基本知识和算法，掌握图的基本概念，主要算法的基本思想和实现方式；
- 在已经实现关于图的一些基本操作函数（比如 first、next、getMark、setMark、weight 等）的基础上，完成以下任务：
- （1）单源最短路径算法：Dijkstra Algorithm，基于 minVertex 扫描的方案（非队列方案）；
- （2）图的最小生成树算法：Prim 算法（非队列方案）和 Kruskal 算法任选一种；
- 请完成以上实验任务。本次实验报告从（1）和（2）中任选 1 个进行撰写，包括算法关键代码和简单的算法运行时间代价分析。

二、使用仪器、材料

PC 微机；
Windows 操作系统，VS2022 编译环境；

三、实验步骤

1、实现关于图的一些基本操作函数（比如 first、next、getMark、setMark、weight 等）；
2、实现单源最短路径算法：Dijkstra Algorithm，基于 minVertex 扫描的方案（非队列方案）；
3、实现图的最小生成树算法：Prim 算法（非队列方案）和 Kruskal 算法任选一种；
4、完成以上实验任务，撰写实验报告，包括算法关键代码和简单的算法运行时间代价分析。

四、实验过程原始记录(数据、图表、计算等)

1、实现关于图的一些基本操作函数（比如 first、next、getMark、setMark、weight 等）；

关键代码：

```
//书P251 图11.6 图的相邻矩阵实现
class Graphm :public Graph {
private:
    int numVertex, numEdge;
    int** matrix;
    int* mark;
public:
    Graphm(int numVert) { Init(numVert); }
    ~Graphm() {
        delete[]mark;
        for (int i = 0; i < numVertex; i++)
            delete[] matrix[i];
        delete[]matrix;
    }
    void Init(int n) {
        int i;
        numVertex = n;
        numEdge = 0;
        mark = new int[n];
        for (i = 0; i < numVertex; i++) {
            mark[i] = 0;
            matrix = (int**)new int* [numVertex];
        }
        for (int i = 0; i < numVertex; i++) {
            matrix[i] = new int[numVertex];
        }
        for (int i = 0; i < numVertex; i++)
        {
            for (int j = 0; j < numVertex; j++) {
                matrix[i][j] = 0;
            }
        }
    }
```

```
    int n() { return numVertex; }
    int e() { return numEdge; }
    int first(int v) {
        for (int i = 0; i < numVertex; i++) {
            if (matrix[v][i] != 0) {
                return i;
            }
        }
        return numVertex;
    }
    int next(int v, int w)
    {
        for (int i = w + 1; i < numVertex; i++)
        {
            if (matrix[v][i] != 0)
                return i;
        }
        return numVertex;
    }
    void setEdge(int v1, int v2, int wght) {
        if (matrix[v1][v2] == 0) {
            numEdge++;
            matrix[v1][v2] = wght;
        }
    }
    void delEdge(int v1, int v2) {
        if (matrix[v1][v2] != 0) {
            matrix[v1][v2] = 0;
        }
    }
    bool isEdge(int i, int j) {
        return matrix[i][j] != 0;
    }
    int weight(int v1, int v2) {
        return matrix[v1][v2];
    }
    int getMark(int v) {
        return mark[v];
    }
    void setMark(int v, int val) {
        mark[v] = val;
    }
};
```

2、单源最短路径算法：Dijkstra Algorithm，基于 minVertex 扫描的方案（非队列方案）；

（1）关键代码：

①minVertex 扫描

```
int minVertex(Graph* G, int* D)
{
    int i, v = -1;
    for (i = 0; i < G->n(); i++)
    {
        if (G->getMark(i) == 0)
        {
            v = i;
            break;
        }
    }
    for (i++; i < G->n(); i++)
    {
        if ((G->getMark(i) == 0) && (D[i] < D[v]))
            v = i;
    }
    return v;
}
```

②Dijkstra 算法

```
//书P256 图11.10 广度优先图遍历算法的实现
void BFS(Graph* G, int start, AQueue* Q)
{
    int v, w;
    Q->enqueue(start);
    G->setMark(start, 1);
    cout << start << ' ';
    while (Q->length() != 0)
    {
        v = Q->dequeue();
        for (w = G->first(v); w < G->n(); w = G->next(v, w))
            if (G->getMark(w) == 0)
            {
                G->setMark(w, 1);
                cout << w << ' ';
                Q->enqueue(w);
            }
    }
}
```

（2）验证程序：

```
int main()
{
    //书P259 图11.16 最短路径定义示例
    //A:0,B:1,C:2,D:3,E:4;
    Graphm* G;
    G = new Graphm(5);
    G->setEdge(0, 1, 10);
    G->setEdge(0, 2, 3);
    G->setEdge(2, 1, 2);
    G->setEdge(0, 3, 20);
    G->setEdge(1, 3, 5);
    G->setEdge(2, 4, 15);
    G->setEdge(3, 4, 11);
    int D[5] = { 0 };
    int s;
    cin >> s;
    Dijkstra(G, D, s);
    for (int i = 0; i < 5; i++)
    {
        cout << "从顶点"<<s<<"到顶点" << i;
        cout << "的最短路径长度为： " << D[i] << " ";
        cout << endl;
    }
    return 0;
}
```

（3）结果：

Microsoft Visual Studio 调试控

```
0
从顶点0到顶点0的最短路径长度为： 0
从顶点0到顶点1的最短路径长度为： 5
从顶点0到顶点2的最短路径长度为： 3
从顶点0到顶点3的最短路径长度为： 10
从顶点0到顶点4的最短路径长度为： 18
```

（4）运行时间代价分析：

设图共有 V 个顶点，E 条边，因为扫描需要进行V^2次，而且每条边都需要相同次数来更新 D 的值，所以该算法的总时间代价为$\Theta(V^2+E)=\Theta(V^2)$，因为 E 在 $O(V^2)$中，可以忽略。

3、图的最小生成树算法：Prim 算法（非队列方案）和 Kruskal 算法任选一种；

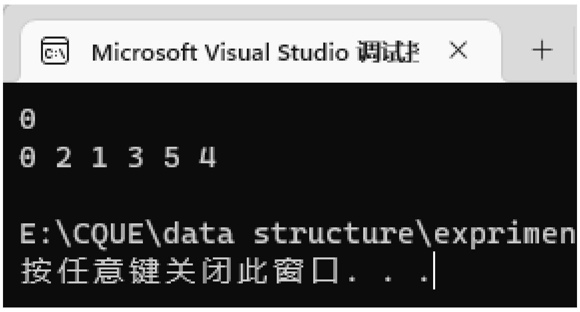
（1）关键代码：

```
//书P262 图11.21 Prim算法的实现
void Prim(Graph* G, int* D, int s, Graph* Gm)
{
    int V[6] = { 0 };
    int i, w;
    for (int i = 0; i < G->n(); i++)
        D[i] = INFINITY;
    D[s] = 0;
    for (i = 0; i < G->n(); i++)
    {
        int v = minVertex(G, D);
        G->setMark(v, 1);
        if (v != s)
            Gm->setEdge(V[v], v, 1);
        if (D[v] == INFINITY) return;
        for (w = G->first(v); w < G->n(); w = G->next(v, w))
            if (D[w] > G->weight(v, w))
            {
                D[w] = G->weight(v, w);
                V[w] = v;
            }
    }
}
```

（2）验证程序：

```
int main()
{
    //书P262 图11.20示例
    //A:0,B:1,C:2,D:3,E:4,F:5;
    Graphm* G;
    Graphm* Gm;
    G = new Graphm(6);
    Gm = new Graphm(6);
    G->Init(6);
    Gm->Init(6);
    G->setEdge(0, 2, 7);
    G->setEdge(0, 4, 9);
    G->setEdge(2, 1, 5);
    G->setEdge(2, 3, 1);
    G->setEdge(2, 5, 2);
    G->setEdge(3, 5, 2);
    G->setEdge(1, 5, 6);
    G->setEdge(5, 4, 1);
    int D[6] = { 0 };
    int s;
    cin >> s;
    Prim(G, D, s, Gm);
    DFS(Gm, 0);
    cout << endl;
    return 0;
}
```

（3）结果：



（4）运行时间代价分析：

Prim 算法与寻找单源最短路径的 Dijkstra 算法非常相似，主要区别在于 Prim 算法不是寻找下一个离起始顶点最近的顶点，而是寻找下一个与 MST 中某个顶点最近的顶点，故该算法运行时间代价仍为 $\Theta(V^2)$。

五、实验结果及分析

结果都已对应显示在原始数据记录中，结果都与预期的分析符合。