重庆交通大学信息科学与工程学院

综合性实验报告

实验名称	图算法应用实验
课程名称	数据结构 A
专业班级	曙光 2101 班
学　　号	632107060510
姓　　名	王靖
指导教师	鲁云平

2022 年 12 月

《数据结构 A》综合性实验评分标准

评分等级	综合性实验评分标准
优	程序演示完全正确，界面美观，能正确回答 90%及以上的问题；报告规范，分析清楚，严格按照要求条目书写，阐述清楚。能针对综合性题目进行分析，并设计合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
良	按要求完成 80%及以上功能，界面尚可，能正确回答 80%及以上的问题；报告规范，分析清楚，个别条目书写不完全符合要求，阐述基本清楚。能针对综合性题目进行分析，并设计较合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
中	按要求完成 70%及以上功能，能回答 70%及以上的问题；报告基本规范，分析基本清楚，存在 30%以内条目书写不完全符合要求。在教师的指导下，能针对综合性题目进行分析，并设计较合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
及格	按要求完成 60%及以上功能，能回答老师多数问题；报告基本规范，存在 40%以内条目书写不完全符合要求。在教师的指导下，能针对综合性题目进行分析和设计较合适的解决方案，并能使用合适的编程平台进行编程、调试、测试和实现，成功调试功能达 60%以上。
不及格	存在 40%以上功能未完成或抄袭；报告不规范或存在 40%以上条目书写不完全符合要求。无法采用正确的方法完成项目分析和解决方案设计或成功调试测试功能超过 40%未完成。

教师签名	
综合性实验成绩	

说明:

1、综合性实验报告总体要求为格式规范，内容丰富、全面、深入；严禁大量拷贝。

2、请提交 word 文档和 PDF 文档，文件命名格式：学号-姓名。

3、报告主要内容参考下页示例

目录

一、实验目的

1、实现图的邻接矩阵和邻接表结构

2、熟悉图基本术语的含义

3、掌握相关操作的编程实现

4、实现图的一些基本操作

5、掌握最小生成树、最短路算法的设计与分析

6、应用图论相关知识解决相关实际问题

二、实验内容

本实验将针对于 3 个问题进行建模求解

问题 1：

现有村落间道路的统计数据表中，列出了有可能建设成标准公路的若干条道路的成本，求使每个村落都有公路连通所需要的最低成本。

（城市数量不超过 1000，道路数量不超过 3000）

问题 2：

重庆是一个旅游胜地，每年都有成千上万的人前来观光。

为方便游客，巴士公司在各个旅游景点及宾馆，饭店等地都设置了巴士站并开通了一些单程巴士线路。

每条单程巴士线路从某个巴士站出发，依次途经若干个巴士站，最终到达终点巴士站。

一名旅客最近到从前旅游，他很想去某个景点玩，但如果从他所在的饭店没有一路巴士可以直接到达该景点，则他可能要先乘某一路巴士坐几站，再下来换乘同一站台的另一路巴士，这样换乘几次后到达景点。

帮助这名旅客寻找一个最优乘车方案，使他在从饭店乘车到景点的过程中换乘的次数最少。

（巴士线路不超过 100 条，站点数量不超过 500）

问题 3：

重庆城里有若干个地方，都有双向公路连接其中的某些地方。

每两个地方最多用一条公路连接，从任何一个地方出发都可以经过一条或者多条公路到达其他地方，但不同的路径需要花费的时间可能不同。

在一条路径上花费的时间等于路径上所有公路需要的时间之和。

佳佳需要从自己的家出发，拜访若干个亲戚（顺序任意）。怎样走，才需要最少的时间

（地点数量不超过 5 万，公路数不超过 10^5）

三、问题分析

（1）数据方面：采用标准数据进行抽象

（2）功能方面：

问题 1：

在问题 1 中，每个村可以视作点，建设公路需要的花费可以视作是边权，则目的就是选出若干条边，权值和最小的情况下使得图连通——这便是最小生成树问题，即可以使用 Prim 算法或 Kruskal 算法进行求解。

问题 2：

本问题中，每个公交站点可视作一个点，而公交线路中，每两点之间都可视作是一条边，而目的就是经过最小的边数到达目的地，对图进行广度优先搜索（BFS）即可实现。

问题 3：

在这个问题中，需要多次最短路径的运算，而只需要枚举访问的顺序，进行 DFS，然后求出每两个点之间的最短路即可求得最少的访问时间，此时需要使用堆对 Dijkstra 算法进行优化，可以让程序效率更高，或是使用 Spfa 算法。

四、系统设计

（1）数据结构的设计

邻接矩阵存储图

```
class Graph
{
private:
    int n; // 点数量
    int m; // 边数量
    int g[N][N]; // 邻接矩阵
    int d[N]; // 距离连通块的距离
    bool st[N];
public:
    Graph(); // 构造函数
    Graph(int n); // 构造函数 | 初始化点数量
    void addEdge(int p, int x, int c); // 加边
    void bfs(); // BFS
    int prim(); // Prim 求最小生成树
};
```

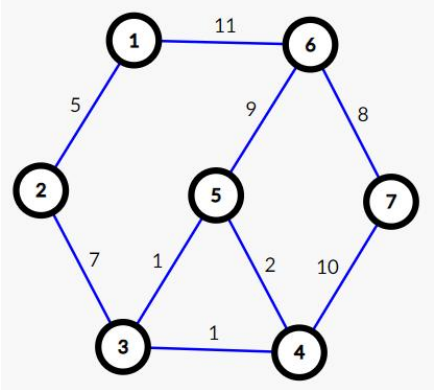
邻接表存储图

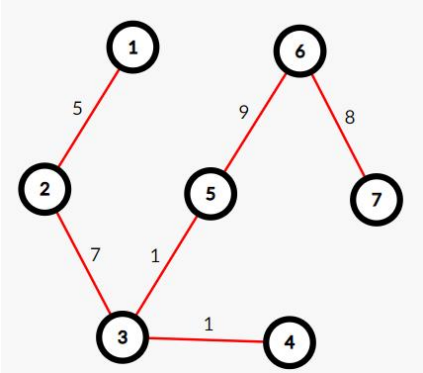
```
class Graph
{
private:
    int n; // 点数量
    int m; // 边数量
    int h[N], e[M], w[M], ne[M], idx; // 数组模拟邻接表
public:
    Graph(); // 构造函数
    Graph(int n); // 构造函数 | 初始化点数量
    void addEdge(int p, int x, int c); // 加边
    int dfs(int u, int start, int dist); // DFS
    void dijkstra(int start, int ds[]); // Dijkstra 求最短路
    void spfa(int start, int ds[]); // Spfa 求最短路
};
```

（2）基本操作的设计

问题 1：

对于问题 1，以下面这个图为例





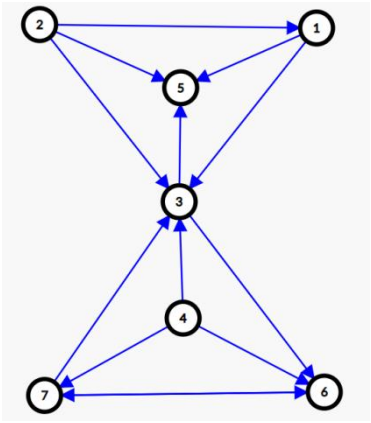
经过 prim 算法运算后的最小生成树如上图所示

问题 2：.

假设有三趟巴士，经过的路线分别为

```
6 7

4 7 3 6

2 1 3 5
```

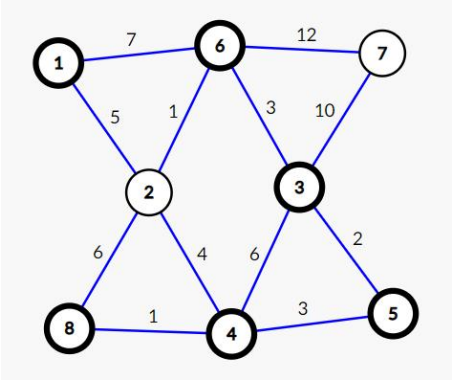
得到有向图如下：



现在要从 3 走到 7，因此需要经过两条边，则答案为 2

问题 3：

如下图：



设起点为 1，要到 3，4，5，6，8 去拜访，则 1->2->6->3->5->4->8 的顺序访问用时最少

五、方案实现

邻接矩阵图类型实现

构造函数

```
Graph::Graph()
{
    memset(e, INF, sizeof(e));
    n = 0, m = 0;
}
Graph::Graph(int n, int m)
{
    this->n = n, this->m = m;
    for (int i = 1; i <= n; i++)
```

```
        for (int j = 1; j <= n; j++)
            if (i == j) e[i][j] = 0;
            else e[i][j] = INF;
}
```

添加一条从 p 到 x 的有向边

```
void Graph::addEdge(int p, int x, int c)
{
    e[p][x] = min(e[p][x], c);
}
```

问题 1:

Prim 算法求最小生成树

```
int Graph::prim()
{
    memset(d, INF, sizeof(d));
    memset(st, 0, sizeof(st));
    int ans = 0;
    for (int i = 1; i <= n; i++)
    {
        int t = -1;
        for (int j = 1; j <= n; j++)
        {
            if (!st[j] && (t == -1 || d[t] > d[j]))
                t = j;
        }
        if (i > 1 && d[t] == INF) return INF;
        if (i > 1) ans += d[t];
        st[t] = true;
        for (int j = 1; j <= n; j++)
        {
            d[j] = min(d[j], e[t][j]);
        }
    }
    return ans;
}
```

解答函数

```
void solve()
{
    cin >> n >> m;
    Graph g(n, m);
    int a, b, c;
    while (m--)
    {
```

```
        g.addEdge(a, b, c);
        g.addEdge(b, a, c);
    }
    int ans = g.prim();
    if (ans > INF / 2) cout << "impossible" << endl;
    else cout << ans << endl;
}
```

问题 2:

BFS 求最小经过边数

```
void Graph::bfs()
{
    memset(d, INF, sizeof(d));
    queue<int> que;
    que.push(1);
    d[1] = 0;
    while (que.size())
    {
        int t = que.front();
        que.pop();
        for (int j = 1; j <= n; j++)
        {
            if (e[t][j] && d[j] > d[t] + 1)
            {
                d[j] = d[t] + 1;
                que.push(j);
            }
        }
    }
}
```

解答程序

```
void solve()
{
    Graph g;
    cin >> n >> m;
    string s;
    getline(cin, s);
    while (m--)
    {
        getline(cin, s);
        stringstream ssin(s);
        int p;
        while (ssin >> p)
        {
```

```
            stop.push_back(p);
        }
        int cnt = stop.size();
        for (int i = 0; i < cnt; i++)
        {
            for (int j = i + 1; j < cnt; j++)
            {
                g.addEdge(stop[i], stop[j], 1);
            }
        }
        stop.clear();
    }
    g.bfs();
    if (d[n] > INF / 2) cout << "impossible" << endl;
    else if (d[n] == 0) cout << 0 << endl;
    else cout << d[n] - 1 << endl;
}
```

邻接表式图实现

构造函数

```
Graph::Graph()
{
    memset(h, -1, sizeof(h));
}
Graph::Graph(int n, int m)
{
    for (int i = 1; i <= n; i++)
    {
        h[i] = -1;
    }
    this->n = n, this->m = m;
}
```

添加一条从 p 到 x 的边

```
void Graph::addEdge(int p, int x, int c)
{
    e[idx] = x;
    w[idx] = c;
    ne[idx] = h[p];
    h[p] = idx++;
}
```

问题 3:

Spfa 求最短路

```
void Graph::spfa(int start, int ds[])
```

```
{
    memset(st, 0, sizeof(st));
    queue<int> que;
    ds[start] = 0;
    que.push(start);
    st[start] = true;
    while (que.size())
    {
        int t = que.front();
        que.pop();
        st[t] = false;

        for (int i = h[t]; i != -1; i = ne[i])
        {
            int j = e[i];
            if (ds[j] > ds[t] + w[i])
            {
                ds[j] = ds[t] + w[i];
                if (!st[j])
                {
                    que.push(j);
                    st[j] = true;
                }
            }
        }
    }
}
```

Dijkstra 算法求最短路

```
void Graph::dijkstra(int start, int ds[])
{
    memset(st, 0, sizeof(st));
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    ds[start] = 0;
    heap.push({0, start});
    while (heap.size())
    {
        PII t = heap.top();
        heap.pop();

        int u = t.second, dist = t.first;
        if (st[u]) continue;
        st[u] = true;

        for (int i = h[u]; i != -1; i = ne[i])
```

```
        {
            int j = e[i];
            if (ds[j] > dist + w[i])
            {
                ds[j] = dist + w[i];
                heap.push({ds[j], j});
            }
        }
    }
}
```

DFS 枚举访问顺序

```
int Graph::dfs(int u, int start, int dist)
{
    if (u > 5) return dist;
    int ans = INF;
    for (int i = 1; i <= 5; i++)
    {
        if (!st[i])
        {
            int next = f[i];
            st[i] = true;
            ans = min(ans, dfs(u + 1, i, dist + d[start][next]));
            st[i] = false;
        }
    }
    return ans;
}
```

解答程序

```
void solve()
{
    cin >> n >> m;
    Graph g(n, m);
    f[0] = 1;
    for (int i = 1; i <= 5; i++)
    {
        cin >> f[i];
    }
    int a, b, c;
    while (m--)
    {
        cin >> a >> b >> c;
        g.addEdge(a, b, c);
        g.addEdge(b, a, c);
```

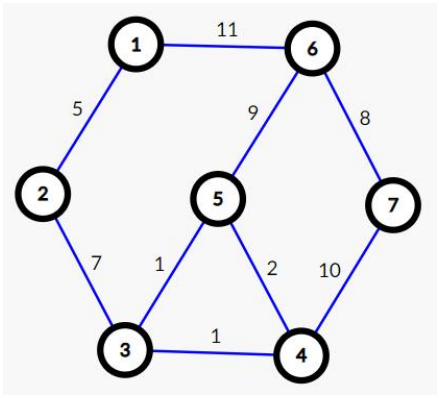
```
    }
    memset(d, INF, sizeof(d));
    for (int i = 0; i <= 5; i++)
    {
        g.dijkstra(f[i], d[i]);
    }

    memset(st, 0, sizeof(st));
    cout << g.dfs(1, 0, 0) << endl;
}
```

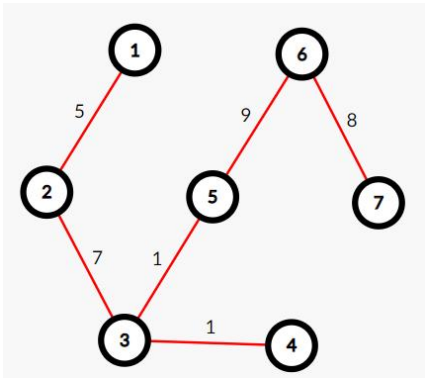
六、系统测试与结果

问题 1：

输入测试：

```
7 9
1 2 5
1 6 11
2 3 7
3 4 1
3 5 1
4 5 2
4 7 10
5 6 9
6 7 8
```





答案应为 31，最小生成树如右图所示

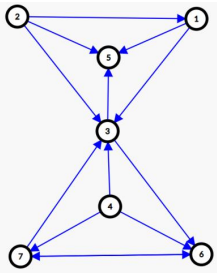
输出答案：

```
31
```

问题 2：

输入测试：

```
3 7
6 7
4 7 3 6
2 1 3 5
```



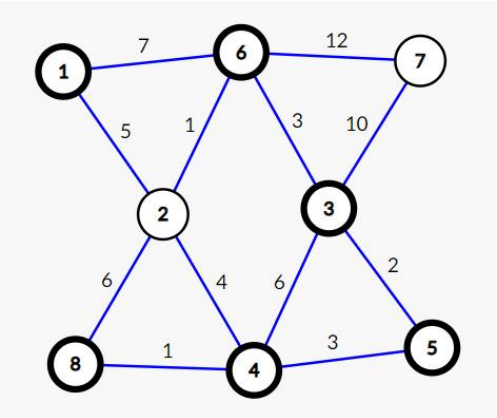
输出答案：

```
2
```

问题 3：

输入测试：

```
8 12
3 4 5 6 8
1 2 5
2 6 1
1 6 7
3 6 3
3 7 10
6 7 12
2 8 6
2 4 4
4 8 1
3 4 6
3 5 2
4 5 3
```



输出答案：

```
15
```

七、实验分析

（1）算法的性能分析

问题 1：

对于 Prim 算法求解最小生成树，平均时间复杂度为$O(n^2)$，在本题中，即最终时间复杂度为$O(n^2)$。

问题 2：

在本问题中，相当于对图做一次广度优先遍历，平均时间复杂度为$O(n+m)$。

问题 3：

本题的算法是经过改进后的，Spfa 算法时间复杂度取决于图的结构，介于$O(m)$~$O(n*m)$,堆优化版 Dijkstra 算法平均时间复杂度为$O(m*log_2 n)$。由于 Spfa 算法时间复杂度不稳定，因此本题选用堆优化的 Dijkstra 算法。DFS 时对 k 个地方访问顺序枚举，时间复杂度为$O(k!)$，若采用 DFS 每次求最短路，时间复杂度为$O(k!*m*log_2 n)$，而本题中可以先预处理最短路，然后进行 DFS，最终时间复杂度优化到$O(k!+m*log_2 n)$

（2）数据结构的分析

由邻接矩阵和邻接表的性质可知，邻接矩阵适合存储稠密图，邻接表适合存储稀疏图；邻接矩阵无法存储重边，因此邻接矩阵适合与重边没有影响时（例如在最短路径问题中，每两个点之间只需要保留权值最小的边）。两者各有利弊，应当根据问题规模和要求进行选择。

八、实验总结

这次实验成功的完成了图数据结构的设计以及相关算法的实现，同时通过构造算法解决实际问题，提升了实际应用能力。对于图的存储和图的相关算法有了更深的理解，算法设计包含了图的广度优先遍历、最小生成树、最短路径相关算法的设计，使用邻接矩阵及邻接表存储图，用链表前向星改善邻接表，使得算法更加安全高效。

图论是数据结构的重要知识，不仅是本实验中涉及到的算法，还有网络流、连通性、拓扑排序、二分图等都是非常接近实际生活的问题且在图论中有很大研究价值的，通过实验的设计，对图论及数据结构有了更深一部理解，在后续学习过程中，更应该对图论有进一步理解，深入学习研究图论。

参考文献：

[1]李春葆.《数据结构教程（第 6 版）》.北京：清华大学出版社.

[2]陈　越,何钦铭.《数据结构（第 2 版）》.北京：高等教育出版社.

[3]殷人昆.《数据结构（用面向对象与 C++语言描述）（第 3 版）》.北京：清华大学出版社

[4]严蔚敏.《数据结构（C 语言版）》.北京：清华大学出版社.

[5]李煜东.《算法竞赛进阶指南》.郑州：河南电子音像出版社.

[6]王桂平,杨建喜,李　韧.《图论算法理论、实现及应用（第 2 版）》.北京：北京大学出版社.

[7]段凡丁.《关于最短路径的 SPFA 快速算法》.成都：西南交通大学学报.1994.4.