组号: 8

图片包含 游戏机, 画

描述已自动生成

上海大学计算机工程与科学学院

**实 验 报 告**

（数据结构2）

学 期：2025-2025年春季

组 长： 高世衡

学 号： 23122725

指导教师： 朱能军

成绩评定： （教师填写）

二〇二五年3月31日

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **小组信息** | | | | |
| 登记序号 | 姓名 | 学号 | 贡献比 | 签名 |
| 1 | 刘天宇 | 23122724 | 33% |  |
| 2 | 高世衡 | 23122725 | 33% |  |
| 3 | 冯俊佳 | 23122721 | 33% |  |
|  |  |  |  |  |

|  |  |
| --- | --- |
| **实验概述** | |
| 实验0 | （熟悉上机环境、进度安排、评分制度；确定小组成员） |
| 实验一 | 有向网的邻接矩阵验证及拓展 |
| 实验二 | 无向网的邻接表验证及拓展 |
| 实验三 | (*实验题目*) |
| 实验四 | (*实验题目*) |

实验二

一、**实验题目**

无向网的邻接表验证及拓展

1. 掌握图的逻辑结构定义和邻接表储结构的实现。  
 2. 掌握图的基本操作算法的实现。

3. 根据实际问题的需要，设计算法解决问题。

二、**实验内容**

模仿有向图的邻接表类模板，完成（带权：非负）无向网的邻接表类模板的设计与实现，增加如下成员函数：

1. CountDegree(v)，统计顶点 v 的度；

2. ConnectedComponent()，求图的连通分量数目；

3. 设计并实现除Kruskal 和 Prim 以外的另一种最小生成树算 法，如“破圈法”等（仅考虑连通图）；

4. hasUniqueMinTree()，判断无向网是否存在唯一的最小生成 树（仅考虑连通图）。

三、**解决方案**

1、算法设计

（1）数据结构

本实验采用邻接表作为图的存储结构，使用模版AdjListUndirNetwork<ElemType, WeightType> 表示带权无向图，具备良好的空间效率与算法扩展性。

核心结构包括：

1. 顶点数组vexTable[]：每个元素为 AdjListNetworkVex<ElemType, WeightType>，包含顶点数据和一条边链表头指针；

2. 边链表 AdjListNetworkArc<WeightType>：每个节点存储邻接顶点编号、边权值及下一条边指针；

3. 辅助结构：

• Status \*tag：用于访问标记（如 DFS、BFS）；

• int vexNum, arcNum：分别表示顶点和边的数量；

• WeightType infinity：用于表示“无连接”的无穷边权。

此结构兼顾了稀疏图的存储效率与操作灵活性。

1. 算法思想

整个图的功能设计采用面向对象思想，以类模板AdjListUndirNetwork 封装所有图相关操作。对于新增的图分析函数，如：统计顶点 v 的度、求图的连通分量数目、最小生成树算法（Sollin算法及破圈法、存在唯一的最小生成树判断）等，均以邻接表为基础，通过遍历顶点间关系进行逐一判断。例如：

在CountDegree(v) 中，通过顺序遍历顶点 v 的边链表，统计与之直接连接的边数，从而确定其度；

在ConnectedComponent() 中，利用 DFS 深度优先遍历策略，从未访问的顶点出发依次标记相连的顶点，重复该过程直至所有顶点被访问，统计 DFS 启动的次数即为连通分量数目。

在SollinAlgorithm() 最小生成树构造过程中，系统初始化每个顶点为一个独立分量，重复选择每个分量连接到其它分量的最小边，通过并查集结构实现高效分量合并，并构建出整个图的最小生成树。

在GenerateAndPrintMSTByCycleBreaking() 破圈法最小生成树构造中，从原始连通无向图出发，逐步检测并删除图中构成环的最大边，直到剩下的图无环且仍保持连通，最终得到最小生成树。

此外，IsUniqueMST() 函数用于判断是否存在唯一的最小生成树。该函数在最小生成树构造基础上，通过检测图中是否存在多个权值相同的候选边，并尝试替代方案构造，判断不同路径是否能得到相同的最小权总和，从而推断 MST 是否唯一。

1. 主要操作

除了实现图的基本操作函数（如插入/删除顶点、插入/删除边、顶点数和边数获取、邻接表打印等），还增加了多个分析函数：

·CountDegree(v)，统计顶点 v 的度；

·ConnectedComponent()，求图的连通分量数目；

·SollinAlgorithm()，索林算法求最小生成树；

·GenerateAndPrintMSTByCycleBreaking() ，破圈法求最小生成树；

·hasUniqueMinTree()，判断无向网是否存在唯一的最小生成树。

1. 用例分析
2. 改进方法
3. 源程序代码

本实验基于实验要求实现了基本函数功能的修改和实现，由于基本函数的实现方法类似，因此不做详细描述，仅对本实验中新增的若干个分析函数进行分析和展示。

2.1基本图函数操作函数一览

以下表中所示为本实验基于有向图的邻接表类模板进行修改实现的基本功能函数一览，不再进行过多赘述。

|  |  |
| --- | --- |
| 函数名 | 功能描述 |
| AdjListUndirNetwork(...) | 构造函数，初始化无向图 |
| ~AdjListUndirNetwork() | 析构函数，释放图资源 |
| void Clear() | 清空无向网所有内容 |
| bool IsEmpty() | 判断无向网是否为空 |
| int GetOrder(ElemType& d) const | 获取指定顶点的序号 |
| Status GetElem(int v, ElemType& d) const | 获取顶点 v 对应元素 |
| Status SetElem(int v, const ElemType& d) | 设置顶点 v 的元素为 d |
| WeightType GetInfinity() const; | 取无穷大的值 |
| int GetVexNum() const | 求无向网的顶点个数 |
| int GetArcNum() const | 求无向网的边数个数 |
| int FirstAdjVex(int v) const | 求顶点 v 的第一个邻接点 |
| int NextAdjVex(int v1, int v2) const | 求顶点 v1 相对于 v2 的下一个邻接点 |
| void InsertVex(const ElemType& d) | 插入元素值为d的顶点 |
| void InsertArc(int v1, int v2, WeightType w) | 插入一条v1→v2，权值为w的边 |
| void DeleteVex(const ElemType& d) | 删除元素值为 d 的顶点 |
| void DeleteArc(int v1, int v2) | 删除从顶点 v1 到 v2 的边 |
| WeightType GetWeight(int v1, int v2) const | 求从顶点v1到v2的边的权值 |
| void SetWeight(int v1, int v2, WeightType w) | 设置从顶点v1到v2的边的权值 |
| Status GetTag(int v) const | 获取顶点 v 的标志 |
| void SetTag(int v, Status tag) const | 设置顶点 v 的标志为 tag |
| AdjListUndirNetwork(const AdjListUndirNetwork<ElemType, WeightType>& copy) | 拷贝构造函数 |
| AdjListUndirNetwork<ElemType, WeightType>& operator = (...) | 赋值运算符重载 |
| void Display() | 打印邻接矩阵结构 |

2.2新增函数实现分析

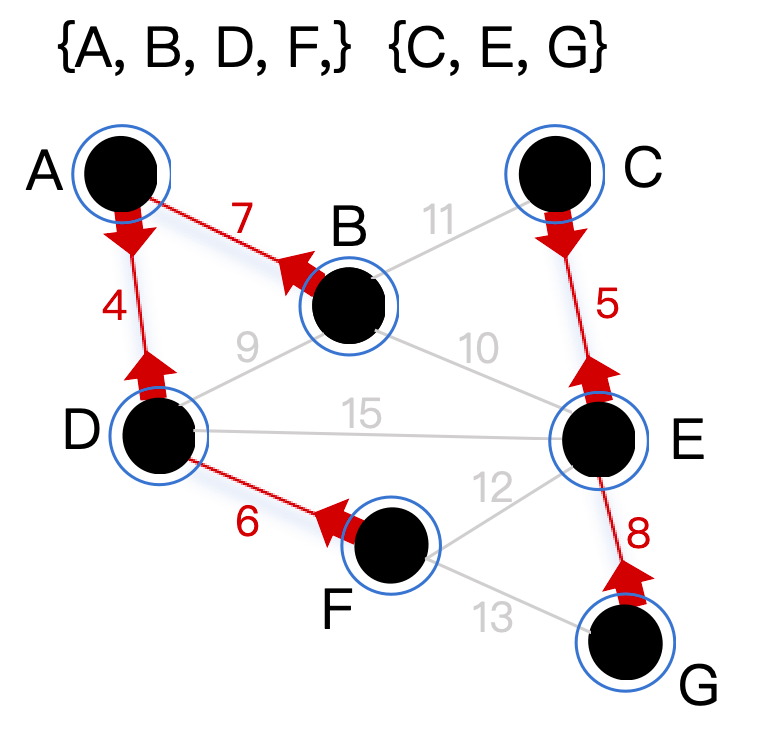
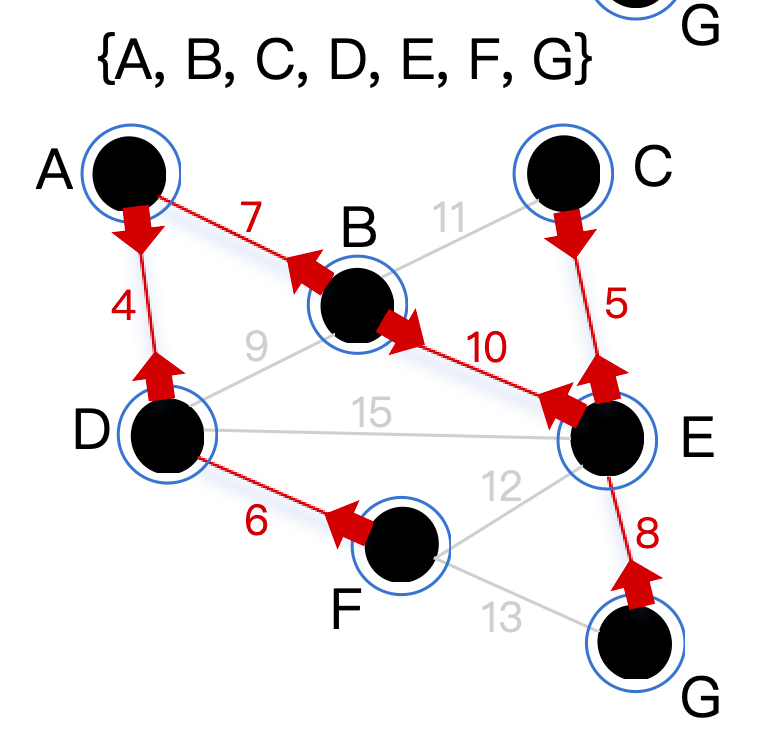
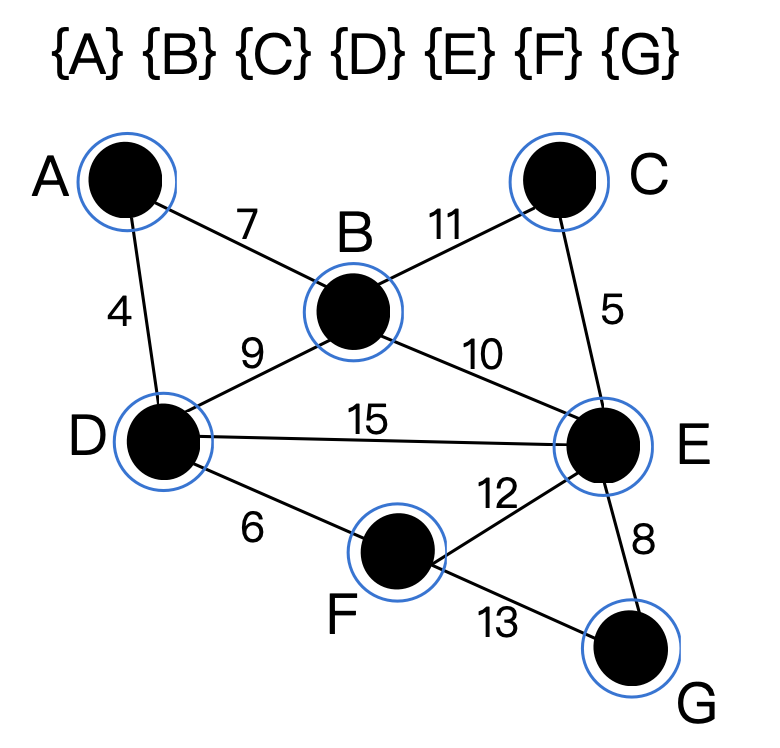
本实验针对有向网（带权有向图）的特点，设计并实现了若干统计与分析功能函数，以便更全面地分析图中节点的结构与关系。

2.2.1 CountDegree函数

2.2.2 ConnectedComponen函数

2.2.3 SollinAlgorithm函数

Sollin算法通过不断选择连接不同组件的最小权重边，并将其添加到最小生成树中，最终构建出一个包含所有顶点且权重最小的树形结构，即最小生成树。初始时，将每个顶点视为一个独立的组件，并初始化parent数组，其中每个顶点的父节点为自身。使用函数findParent来找到顶点所在组件的根节点。在每次迭代中，遍历图中的每个顶点，对于每个顶点，遍历其相邻的顶点，找到连接到其他组件的最小权重边。如果找到的边连接了两个不同的组件，则将其添加到最小生成树中，并将这两个组件合并。使用parent数组来跟踪每个顶点所在的组件，通过路径压缩和按秩合并等技术来优化并查集的操作，提高算法效率。当最小生成树中的边数达到图的顶点数减一时，即所有顶点都在同一个组件中时，算法结束。具体步骤流程图及核心代码如下所示：



1. **template** <**class** ElemType, **class** WeightType>
2. //findparent实现 **int** AdjListUndirNetwork<ElemType, WeightType>::findParent
3. (**const** vector<**int**>& parent, **int** v) **const** {
4. **while** (parent[v] != v) {
5. v = parent[v];
6. }
7. **return** v;
8. }
10. **template** <**class** ElemType, **class** WeightType>
11. **void** AdjListUndirNetwork<ElemType, WeightType>::SollinAlgorithm(vector<Edge<WeightType> >& mstEdges){
12. // 初始化组件（每个顶点是一个组件）
13. vector<**int**> parent(vexNum);
14. iota(parent.begin(), parent.end(), 0); // 用0至vexNum-1的连续整数填充parent
15. **int** numComponents = vexNum;
16. **while** (numComponents > 1) {
17. // 每个组件选择连接到其他组件的最小边
18. vector<Edge<WeightType> > cheapest(vexNum, Edge<WeightType>(-1, -1, (WeightType)DEFAULT\_INFINITY));
19. **for** (**int** v = 0; v < vexNum; ++v) {
20. **for** (**int** w = FirstAdjVex(v); w >= 0; w = NextAdjVex(v, w)) {
21. **int** rootV = findParent(parent, v);
22. **int** rootW = findParent(parent, w);
23. **if** (rootV != rootW) {
24. WeightType weight = GetWeight(v, w);
25. **if** (weight < cheapest[rootV].weight) {
26. cheapest[rootV] = Edge<WeightType>(v, w, weight);
27. }
28. }
29. }
30. }
31. // 合并组件并添加边到MST
32. **for** (**int** i = 0; i < vexNum; ++i) {
33. **if** (cheapest[i].from != -1) {
34. **int** rootV1 = findParent(parent, cheapest[i].from);
35. **int** rootV2 = findParent(parent, cheapest[i].to);
36. **if** (rootV1 != rootV2) {
37. mstEdges.push\_back(cheapest[i]);
38. parent[rootV1] = rootV2;
39. --numComponents;
40. }
41. }
42. }
43. }
44. // 输出生成的最小生成树边信息
45. cout << "最小生成树的边：" << endl;
46. **for** (**size\_t** i = 0; i < mstEdges.size(); ++i) {
47. ElemType fromElem, toElem;
48. **if** (GetElem(mstEdges[i].from, fromElem) == ENTRY\_FOUND &&
49. GetElem(mstEdges[i].to, toElem) == ENTRY\_FOUND)
50. {
51. cout << fromElem << " - " << toElem << " 权重: " << mstEdges[i].weight << endl;
52. } **else** {
53. cout << "Error: 顶点索引错误" << endl;
54. }
55. }
56. }

2.2.4 GenerateAndPrintMSTByCycleBreaking函数

2.2.5 bool IsUniqueMST( )函数

1. 实验结果

3.1 展示实验结果

3.2 结果分析

1. 算法分析

4.1 算法时间复杂度分析

* IsConnected()：使用 BFS 遍历图，时间复杂度为 O(V + E)（V 是顶点数，E 是边数）。
* GenerateAndPrintMSTByCycleBreaking()：

统计边数：O(E)（遍历邻接表）。

排序边：O(E log E)（快速排序）。

破圈主循环：最多执行 E 次，每次调用 IsConnected()（O(V + E)），因此总复杂度为 O(E(V + E))。

总时间复杂度：O(E log E + E(V + E))，最坏情况下为 O(E²)（稠密图 E≈V²）。

* 假设顶点数为n，边数为m。在算法中，最耗时的部分是每次迭代中的查找

操作和合并操作。查找操作（findParent函数）：最坏情况下为O(n)，但由于路径压缩的优化，平均情况下可以接近O(1)。合并操作：每个顶点最多进行一次合并操作，因此总的合并操作数为O(n)。每次迭代中遍历边的操作：最坏情况下为O(m)，但由于采用了边的存储优化（使用cheapest 数组），平均情况下可以接近O(m/n)。综上，Sollin算法的时间复杂度在理想情况下接近 O(mlogn)。

4.2 算法空间复杂度分析

* IsConnected()：使用 visited 数组（O(V)）和队列（O(V)），空间复杂度 O(V)。
* GenerateAndPrintMSTByCycleBreaking()：

存储边：O(E)（edges 数组）。

MST 边列表：O(V)（树最多 V-1 条边）。

总空间复杂度：O(V + E)。

* SollinAlgorithm：由parent 数组和cheapest 数组决定，均为 O(n)。

1. 总结与心得