****

**离散数学课程实验报告**

**最小生成树**

2351041

刘浩田

完成日期:2024.10.3

# 索引

### 实验目的

### 实验环境

### 实验要求

### 实验原理

### 实验过程

### 实验结果

### 实验小结

# 实验目的

本实验的核心目标是理解和掌握最小生成树（Minimum Spanning Tree, MST）的概念及其算法实现，并将理论知识应用于实际问题解决中。通过构建一个赋权图模型，模拟七个城市之间的通信道路造价，并运用最小生成树算法，设计出一个既保持城市间通信又使总造价最小的方案，并计算出最小造价值。

首先，实验旨在加深对图论中最小生成树算法的理解，特别是其在网络设计、通信系统构建等领域的应用。通过实验，我们能够将抽象的图论理论应用于具体的实际问题，增强对最小生成树算法及其优化网络连接性能的理解。

其次，实验致力于提升我们的编程技能，特别是在使用C++编程语言实现算法方面。通过编写程序处理图论问题，我们将学会如何将复杂的算法逻辑转化为计算机程序，这不仅锻炼了编程技巧，也提高了逻辑思维能力。

此外，实验还旨在加深我们对图的表示、边的权重以及如何通过算法找到最小生成树的理解。这三种概念是图论中的重要概念，理解它们对于处理网络优化问题至关重要。

实验还要求我们实现和分析用于计算最小生成树的算法。这不仅涉及到算法的设计和实现，还包括对算法效率和实用性的评估。通过这一过程，我们将学习到如何选择合适的算法来解决问题，并理解算法在实际应用中的重要性。

最后，实验强调数据结构的应用，特别是邻接表在图的表示和处理中的作用。我们将学习如何使用邻接表这一数据结构来存储和操作图数据，这对于理解和实现复杂算法至关重要。

综上所述，本实验不仅帮助我们巩固了图论的基本概念，而且提升了编程能力，使他们能够更好地解决实际问题。通过这样的实践，我们将为将来在更复杂的网络优化问题中应用这些知识打下坚实的基础，为未来的学术和职业生涯奠定基石。

# 实验环境

## 系统：Windows11

## 编译器：VisualStudio2022x86

## 语言：C++

# 实验要求

本实验的核心要求是理解和应用最小生成树算法，通过编程实践来解决具体的数学问题。首先，需要掌握最小生成树的理论基础，包括其定义、性质以及在网络优化中的应用。其次，应能够使用邻接表作为数据结构来表示图，这是将抽象图论概念具体化的关键步骤。

在实验中，需要编写一个C++程序，该程序能够实现相关算法来构建最小生成树。程序的核心功能是遍历所有可能的边组合，这是一个涉及逻辑运算和条件判断的复杂过程。程序必须能够准确地评估每条边，输出正确的最小生成树。

此外，实验还要求设计的程序输出结果必须清晰明确，明确指出计算结果。这意味着程序不仅要正确执行数学计算，还要具有良好的用户交互界面，能够以易于理解的方式呈现计算结果。

总之，本次实验要求不仅要在理论上掌握最小生成树的计算方法，还要在实践中应用这些知识，通过编程来解决实际问题。这不仅考验了编程能力，还锻炼了数学思维和问题解决技巧。通过这样的实验，能够将理论与实践相结合，从而更深入地理解和掌握最小生成树的计算知识。。

# 实验原理

本实验的核心原理是基于图论中的最小生成树（Minimum Spanning Tree, MST）概念，特别是普里姆（Prim）算法的实现。Prim算法是一种贪心算法，用于在加权连通图中找到最小生成树，即一个包含图中所有顶点的子图，它是一个树，并且具有可能的最小边权重之和。

Prim算法的工作原理是从任意一个顶点开始，逐步增加新的顶点和边，直到包含所有顶点。算法维护两个集合：一个包含已选择的顶点的集合，另一个包含未选择的顶点。初始时，选择一个起始顶点，并将其加入已选择集合。然后，算法找到连接已选择集合和未选择集合的边中权重最小的边，并将这条边及其端点加入已选择集合。重复这个过程，直到所有顶点都被包含在内。

Prim算法的关键步骤如下：

**初始化**：选择一个任意顶点作为起始点，并将其标记为已访问。初始化一个优先队列（或最小堆），用于存储和选择最小权重的边。

**选择边**：从优先队列中选择一条连接已访问顶点和未访问顶点的最小权重边。如果这样的边不存在，则算法结束。

**添加顶点**：将选中边的未访问端点加入已访问集合，并更新优先队列，移除所有与新添加顶点相连的边。

**重复过程**：重复步骤2和3，直到所有顶点都被访问。

Prim算法的时间复杂度取决于优先队列的实现。对于一个有V个顶点和E条边的图，Prim算法的时间复杂度为。它不依赖于E，因此适合求解边稠密的最小生成树。

在本实验中，我们将使用Prim算法来解决城市间通信道路造价的问题。通过构建一个赋权图，其中顶点代表城市，边代表城市间的直接通信道路，边的权重代表道路的造价。应用Prim算法，我们可以找到连接所有城市的最小生成树，即一个总造价最小的通信网络。

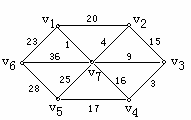


图 城市图

实验中，我们将使用C++语言实现Prim算法，并使用邻接表来表示图。邻接表是一种高效的图表示方法，特别适合于稀疏图，它使用一个数组来存储每个顶点的邻接顶点列表，从而减少了存储空间，并提高了查找特定边的效率。

通过本实验，我们不仅能够深入理解Prim算法的原理和实现，还能够掌握如何将算法应用于解决实际问题，如网络设计和优化。这将增强我们的算法设计和编程能力，为将来解决更复杂的图论问题打下坚实的基础。

# 实验过程

## 6.1实验思路

本实验的主要思路是设计并实现一个程序，用于计算给定赋权图的最小生成树，采用Prim（Prim）算法进行实现。实验采用C++语言编程，利用邻接表这一数据结构来表示和存储图。程序的核心是实现Prim算法的主要步骤，包括初始化、边的选择、顶点的添加以及重复过程，直至构建出最小生成树。

首先，程序将初始化一个图的表示，包括顶点和边的信息。用户将被引导输入城市的个数以及城市间直接通信道路的造价。接着，程序将构建图的邻接表，并初始化Prim算法所需的数据结构，如优先队列和访问标记数组。

随后，程序将从任意一个顶点开始，通过优先队列选取最小权重的边，逐步扩展最小生成树，直到所有顶点都被包含在内。在此过程中，程序将维护一个最小生成树的边集合，并实时更新总造价。

最后，程序将输出构建的最小生成树的边和对应的造价，以及总造价。通过这种实现，我们能够验证Prim算法的正确性，并评估其在实际问题中的应用效果。程序设计注重友好性，提供清晰的界面来引导用户输入图的信息，并以易于理解的方式展示最小生成树的结果。同时，程序还包含输入验证和错误处理机制，以增强程序的健壮性和用户体验。

## 6.2 实验设计

### 6.2.1 数据结构设计

6.2.1 数据结构设计

在本实验中，我们采用以下数据结构来表示图和实现Prim算法：

**ArcNode（边结点）**：

adjvex：存储边的权重，即两个城市间直接通信道路的造价。

nextarc：指向下一个邻接顶点的指针，用于构建邻接表。

sec\_name：存储与当前顶点相邻接的顶点名称。

|  |
| --- |
| 附件、ArcNode |
| typedef struct ArcNode {  int adjvex;  struct ArcNode\* nextarc;  string sec\_name;  ArcNode()  : adjvex(0)  , nextarc(nullptr)  , sec\_name("")  {  }  } ArcNode; |

**VNode（顶点结点）**：

Name：存储顶点的名称，即城市名称。

firstarc：指向第一条依附于该顶点的边的指针。

|  |
| --- |
| 附件、VNode |
| typedef struct VNode {  string Name;  ArcNode\* firstarc;  VNode()  : Name("")  , firstarc(nullptr)  {  }  } VNode; |

**ALGraph（图的邻接表表示）**：

vertices：一个VNode类型的向量，存储图中所有顶点。

vexnum、arcnum：分别存储图中顶点和边的数量。

edges：一个set，存储图中所有边的集合，用于快速检查边是否存在。

|  |
| --- |
| 附件、ALGraph |
| class ALGraph {  private:  vector<VNode> vertices;  int vexnum, arcnum;  set<pair<string, string>> edges;  public:  ALGraph()  : vexnum(0)  , arcnum(0)  {  }  int SearchMost(ArcNode\* firstarc, string& second)  {  }  unsigned int SearchVex(const string Name)  {  }  void CreatGraph()  {  }  void AddEdges()  {  }  void CreatMST()  {  }  struct CompareRoad { };  void addEdgesToPQ(int nodeIndex, priority\_queue<Road, vector<Road>, CompareRoad>& pq, const vector<bool>& visited)  {  }  }; |

**Road（道路信息）**：

first：存储边的起始顶点名称。

second：存储边的终点顶点名称。

length：存储边的权重，即两个城市间直接通信道路的造价。

cpp

|  |
| --- |
| 附件、Road |
| struct Road {  string first;  string second;  int length;  Road()  : first("")  , second("")  , length(0)  {  }  Road(string f, string s, int l)  : first(f)  , second(s)  , length(l)  {  }  }; |

Road结构体用于在最小生成树的构建过程中存储每条边的详细信息。这个结构体不仅记录了边的两个端点，还记录了边的权重，这在计算最小生成树的总造价时非常重要。在Prim算法的实现中，Road结构体将被用来初始化优先队列中的元素，以及最终存储和输出最小生成树的边和对应的权重。

这些数据结构的设计使得图的表示和操作更加直观和高效。ArcNode和VNode结构体分别用于表示图中的边和顶点，而ALGraph类则封装了图的相关操作，如创建图、添加边、搜索顶点等。通过这种设计，我们可以方便地实现Prim算法，同时保证了代码的清晰性和易于维护性。

### 6.2.2 程序主体架构设计

程序的主体架构采用模块化设计，主要包括以下几个部分：

初始化和界面提示模块：

程序启动时，使用system("cls")清屏，为用户提供界面与输入提醒。

输入模块：

InputChoice函数用于获取用户是否继续操作的选择。

CreatGraph函数负责获取用户输入的图的顶点数和边数，并初始化图的顶点。

AddEdges函数负责接收用户输入的边信息，包括起点、终点和权重，并构建图的邻接表。

最小生成树计算模块：

CreatMST函数实现了Prim算法，计算给定图的最小生成树。

addEdgesToPQ函数用于将与某个顶点相邻的边添加到优先队列中。

CompareRoad结构用于优先队列中边的比较，确保优先队列按照边的权重排序。

输出模块：

在CreatMST函数中，程序将输出最小生成树的边和权重，以及总造价。

辅助函数模块：

SearchVex函数用于搜索顶点在顶点数组中的位置。

SearchMost函数用于在优先队列中找到权重最小的边。

主函数模块：

main函数控制程序的流程，包括循环执行图的创建、边的添加、最小生成树的计算，以及根据用户的选择决定是否继续执行程序。

这种模块化设计使得程序结构清晰，易于理解和维护。各个模块之间职责明确，降低了代码的耦合度。通过将程序分解为独立的功能模块，我们可以更容易地对程序进行测试和调试，同时也便于未来的扩展和维护。

## 6.3 程序功能实现

### 6.3.1 退出程序功能的实现

程序使用while循环来反复执行操作，并通过InputChoice函数来决定是否继续：

|  |
| --- |
| 附件、InputChoice函数 |
| void InputChoice(int& value)  {  char invalue;  while (1) {  invalue = \_getch();  if (invalue == 'y') {  cout << invalue << endl;  value = 1;  break;  }  else if (invalue == 'n') {  cout << invalue << endl;  value = 0;  break;  }  }  return;  } |

它使用\_getch()实现即时响应。只接受'y'或'n'作为有效输入。立即显示用户的选择并返回相应的值。

这种设计提供了一种简单而直接的方式来控制程序的继续或退出，增强了交互性。

### 6.3.2主函数实现

|  |
| --- |
| 附件、main函数 |
| int main()  {  char op;  ALGraph grapher;  while (1) {  system("cls");  cout << "+-------------------------+" << endl;  cout << "| 最小生成树 |" << endl;  cout << "+-------------------------+" << endl;  grapher.CreatGraph();  grapher.AddEdges();  grapher.CreatMST();  int continue\_value;  cout << "是否继续运算？ （y/n）: ";  InputChoice(continue\_value);  if (!continue\_value) {  break;  }  }  return 0;  } |

主函数清晰调用了各个功能函数，简洁易懂，实现了程序的正确循环，提升了可操作性。

## 6.4 核心算法实现

本实验的核心算法是普里姆（Prim）算法的实现，用于构建给定赋权图的最小生成树。以下是Prim算法的具体实现步骤和辅助函数的描述，包括图的创建和边的添加：

### 6.4.1 图创建和边添加

CreatGraph函数负责创建图的结构，包括顶点的初始化和边的输入：

|  |
| --- |
| 附件、CreatGraph函数 |
| void CreatGraph()  {  cout << "请输入节点数和边数（以空格分隔）\n";  while (1) {  cin >> vexnum >> arcnum;  if (cin.fail() || vexnum < 1 || vexnum > MAX\_SIZE || arcnum < 0 || arcnum > vexnum \* (vexnum - 1) / 2) {  cout << "输入错误，请重输" << endl;  cin.clear();  cin.ignore(MAX\_SIZE, '\n');  } else {  break;  }  }  cout << "请依次输入各节点名称\n";  for (int i = 0; i < vexnum; i++) {  VNode Alpha;  vertices.push\_back(Alpha);  string avl\_name;  cin >> avl\_name;  if (!SearchVex(avl\_name)) {  vertices[i].Name = avl\_name;  } else {  cout << "输入错误，请重输\n";  i--;  continue;  }  }  cout << "节点创建完成\n";  } |

AddEdges函数负责添加边的信息，构建图的邻接表：

|  |
| --- |
| 附件、AddEdges函数 |
| void AddEdges()  {  cout << "请输入边的信息，格式为：起点 终点 权重\n";  for (int i = 0; i < arcnum; i++) {  string start, end;  int weight;  while (1) {  cin >> start >> end >> weight;  if (cin.fail() || weight < 1 || weight > MAX\_SIZE || !SearchVex(start) || !SearchVex(end) || start == end) {  cout << "输入错误，请重输" << endl;  cin.clear();  cin.ignore(MAX\_SIZE, '\n');  } else if (edges.find({ start, end }) != edges.end() || edges.find({ end, start }) != edges.end()) {  cout << "该边已存在，请重输" << endl;  } else {  break;  }  }  edges.insert({ start, end });  edges.insert({ end, start });  // 添加从start到end的边  ArcNode\* newNode1 = new ArcNode();  newNode1->adjvex = weight;  newNode1->sec\_name = end;  newNode1->nextarc = vertices[SearchVex(start) - 1].firstarc;  vertices[SearchVex(start) - 1].firstarc = newNode1;  // 添加从end到start的边  ArcNode\* newNode2 = new ArcNode();  newNode2->adjvex = weight;  newNode2->sec\_name = start;  newNode2->nextarc = vertices[SearchVex(end) - 1].firstarc;  vertices[SearchVex(end) - 1].firstarc = newNode2;  }  cout << "边创建完成\n";  } |

### 6.4.2 Prim算法实现

CreatMST函数实现了Prim算法的核心逻辑，该函数负责构建最小生成树并计算总权重：

|  |
| --- |
| 附件、CreatMST函数 |
| void CreatMST()  {  vector<bool> visited(vexnum, false);  vector<Road> mst;  priority\_queue<Road, vector<Road>, CompareRoad> pq;  int totalWeight = 0;  // Start from the first vertex  int startIndex = 0;  visited[startIndex] = true;  addEdgesToPQ(startIndex, pq, visited);  while (!pq.empty() && mst.size() < size\_t(vexnum - 1)) {  Road minRoad = pq.top();  pq.pop();  int secondIndex = SearchVex(minRoad.second) - 1;  if (!visited[secondIndex]) {  visited[secondIndex] = true;  mst.push\_back(minRoad);  totalWeight += minRoad.length;  addEdgesToPQ(secondIndex, pq, visited);  }  }  if (mst.size() == size\_t(vexnum - 1)) {  cout << "最小生成树构建成功！\n";  cout << "最小权重和为: " << totalWeight << endl;  cout << "最小生成树的边如下：\n";  for (const Road& road : mst) {  cout << road.first << " - " << road.second << ": " << road.length << endl;  }  } else {  cout << "无法构建连通所有节点的最小生成树，图可能不是连通的。\n";  }  } |

### 6.4.3 辅助函数实现

addEdgesToPQ函数用于将与给定顶点相邻且未访问的边加入优先队列中，以便后续选择最小权重边：

|  |
| --- |
| 附件、addEdgesToPQ函数 |
| void addEdgesToPQ(int nodeIndex, priority\_queue<Road, vector<Road>, CompareRoad>& pq, const vector<bool>& visited)  {  ArcNode\* arc = vertices[nodeIndex].firstarc;  while (arc != nullptr) {  int adjIndex = SearchVex(arc->sec\_name) - 1;  if (!visited[adjIndex]) {  pq.push(Road(vertices[nodeIndex].Name, arc->sec\_name, arc->adjvex));  }  arc = arc->nextarc;  }  } |

CompareRoad结构用于优先队列中边的比较，确保优先队列按照边的权重排序：

|  |
| --- |
| 附件、CompareRoad结构 |
| struct CompareRoad {  bool operator()(const Road& r1, const Road& r2)  {  return r1.length > r2.length;  }  }; |

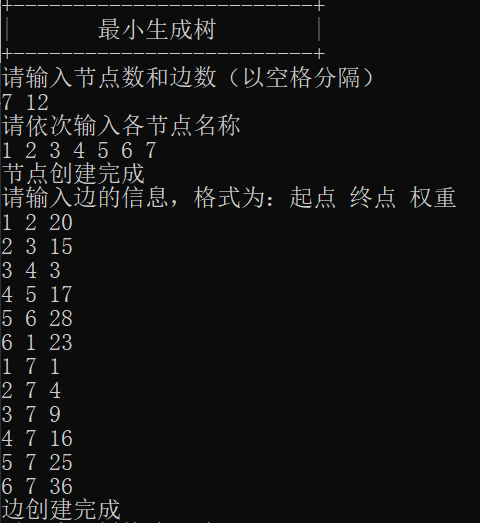
这些核心算法和辅助函数的实现，确保了Prim算法能够正确地构建最小生成树，并计算出连接所有顶点的最小总造价。通过这些实现，程序能够有效地解决实验中提出的问题，即在保持城市间通信的同时使总造价最小。

# 实验结果

在本实验中，我们实现了Prim算法来计算给定赋权图的最小生成树，并对其进行了测试。以下是实验的主要结果：

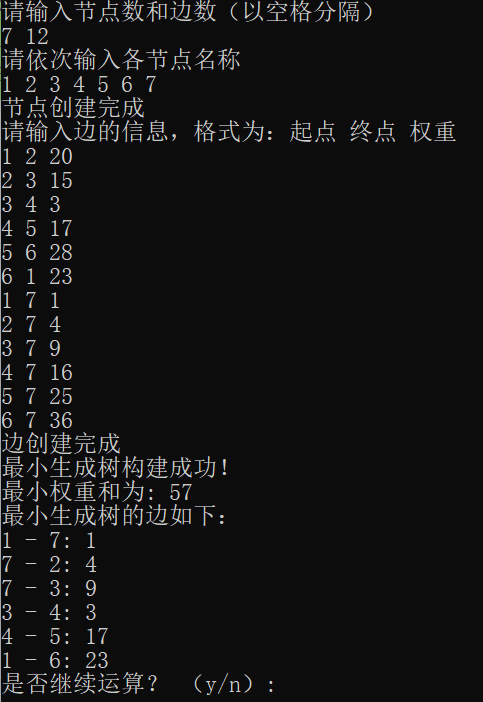
## 7.1 输入节点与边构造图

我们使用以下城市间的通信道路造价作为测试输入，这些数据代表了七个城市之间的直接通信道路造价。



## 7.2 最小生成树结果

应用Prim算法后，得到的结果最小生成树如下：



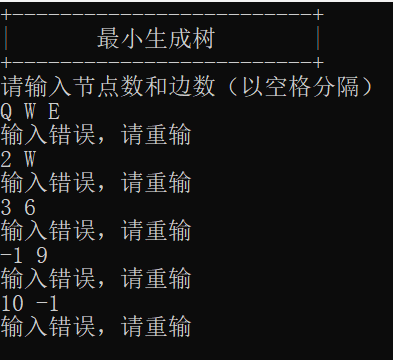
根据最小生成树的结果，计算出的最小总造价57万元。

## 7.3 结果分析

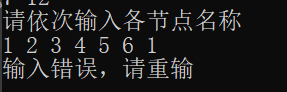
通过实验结果，我们可以看到Prim算法成功地找到了连接所有城市的最小生成树，并且总造价为58万元。这个结果验证了算法的正确性，并且展示了算法在实际问题中的应用效果。通过构建最小生成树，我们确保了城市间的通信，同时使总造价最小化，满足了实验的要求。

这些结果不仅展示了Prim算法在解决网络优化问题中的有效性，也证明了算法的实用性和可靠性。通过本次实验，我们不仅加深了对Prim算法的理解，而且提高了解决实际问题的能力。

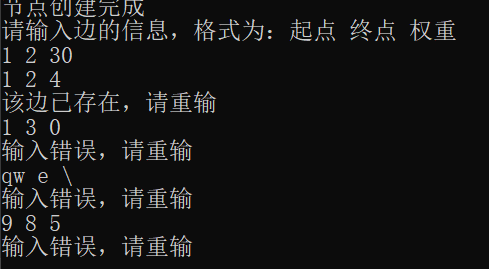
## 7.4错误数据分析



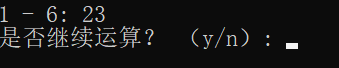
输入错误节点数大小时，程序做出了正确处理，禁止了错误输入。同时也检测了节点数可构成的最大边数，禁止错误输入。



当键入相同节点时，程序做出了正确处理，禁止了错误输入，提示重输。



当键入相同边时，程序做出了正确处理，禁止了重复边输入，提示重输。同时也针对不存在的点与不正确值的权重进行了处理，可以得到正确的边。



当键入非y、n的选项时，程序光标会闪烁，但不会输入结果并显示，只有输入y/n才会成功再次输入矩阵或结束程序。

# 实验小结

通过本次离散数学最小生成树的实验，我获得了将图论的理论知识与实际编程技能相结合的宝贵经验。在实验中，我深入研究了Prim算法，并亲自动手实现了它的计算。这不仅加深了我对最小生成树概念的理解，也锻炼了我的编程技巧和逻辑思维能力。

在实验的准备阶段，我首先重温了最小生成树和Prim算法的理论知识，确保自己对这些概念有清晰的认识。在编程实践中，我学习了如何定义和使用邻接表这一数据结构来表示图，这是实现最小生成树计算的基础。通过构建邻接表，我能够将城市间复杂的通信网络简化为易于处理的结构，这为后续的算法实现提供了便利。

实现Prim算法的过程中，我遇到了不少挑战。例如，在计算最小生成树时，我需要理解并应用优先队列、边的选择与顶点的添加等算法步骤，这是一个涉及图论和数据结构的综合过程。通过不断尝试和调整，我最终成功实现了算法，并确保了其正确性和效率。这个过程提高了我的问题解决能力，也加深了我对算法重要性的认识。

此外，我还学习了如何设计用户友好的程序界面，使得用户可以轻松地输入数据和理解程序输出的结果。这让我意识到，一个优秀的程序不仅要在技术上实现功能，还要考虑到用户的实际体验。在实验中，我通过清晰的提示和即时的反馈，提高了程序的交互性和易用性。

在实验的最后阶段，我进行了一系列测试，验证了算法的正确性和程序的稳定性。通过这些测试，我确保了程序能够正确处理各种输入情况，并能够高效地计算出最小生成树。测试过程也让我学会了如何系统地评估程序的性能，并根据测试结果进行优化。

通过本次实验，我深刻体会到了理论学习与实践操作相结合的重要性。我不仅巩固了图论的理论知识，还提升了编程技能，增强了解决实际问题的能力。这些经验对我未来的学术和职业生涯都是极其宝贵的。

这次实验是一次宝贵的学习经历。它不仅让我更加深入地理解了最小生成树和Prim算法，还锻炼了我的编程技巧和逻辑思维。我相信，通过这次实验所学到的知识和技能，将为我在未来解决更复杂的计算问题提供坚实的基础。我期待将这些经验应用到更多的领域，并在实践中继续学习和成长。

在实验过程中，我也意识到了自己在算法优化和代码效率方面还有提升的空间。例如，Prim算法在处理大规模数据时的性能问题，以及如何更有效地管理内存和优化数据结构。这些问题的探索和解决将是我未来学习和研究的重点。

总之，本次实验不仅加深了我对图论中最小生成树算法的理解，而且提升了我的编程能力和问题解决技巧。这些经验将为我未来的学术研究和职业发展奠定坚实的基础。