**安徽大学集成电路学院**

**《算法设计与分析》第八次报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 算法设计与分析 | | | 日期 |  |
| 专业 |  |  |  | 姓名 |  |
| 【实验目的】  本实验的目的是通过使用Matlab语言实现Kruskal算法，进一步理解和掌握最小生成树（Minimum Spanning Tree, MST）这一图论中的重要概念及其在网络设计等领域的应用。具体目标包括：  1. 理解Kruskal算法的基本原理和步骤：包括如何从边的角度出发，通过贪心策略逐步构建最小生成树。  2. 熟悉并掌握Matlab编程：通过编写Kruskal算法的Matlab代码，巩固和提升Matlab编程能力。  3. 分析算法的时间复杂度和空间复杂度：理解Kruskal算法在不同图结构中的效率表现。  4. 实际应用与验证：通过对给定的图数据进行实际操作，验证Kruskal算法的正确性和有效性，理解其在实际问题中的应用场景和限制。 | | | | | |
| 【实验原理（预习）】  Kruskal算法是一种贪心算法，用于找到连通无向图的最小生成树（Minimum Spanning Tree, MST）。该算法的基本思想是按照边的权重从小到大进行排序，然后依次将边加入生成树中，确保不会形成环，直到生成树包含所有顶点为止。  Kruskal算法步骤  1. 初始化：将图中所有的边按照权重从小到大进行排序。  2.生成森林：将每个顶点视为一个独立的集合（即森林中的一棵树）。  3. 边的选择：从排序后的边集合中选取权重最小的边，判断其是否会与现有的生成树形成环。如果不会，则将该边加入生成树。  4. 合并集合：将当前选择的边的两个顶点所在的集合进行合并。  5. 重复步骤3和4，直到生成树包含图中的所有顶点。  **关键操作**   |  | | --- | | 伪代码 | | 算法4.6  Kruskal  输入:连通图 \\顶点数n,边数m  输出:G的最小生成树  1. 按权从小到大排序G中的边，使得E={e1,e2,...,em}  2. T ←∅  3. repeat  4.    e←E中的最短边  5.    if e的两端点不在同一个连通分支  6.    then T ←T∪{e}  7.    E ←E-{e}  8. until T包含了n-1条边 |   - 边的排序：时间复杂度为，其中是图中边的数量。  - 查找集合：使用并查集（Union-Find）结构来实现集合的合并和查找操作，支持路径压缩和按秩合并，时间复杂度接近于常数，具体为，其中是阿克曼函数的反函数，为顶点数。  最小生成树性质   * 连通性：最小生成树保证包含图中所有的顶点，并且是连通的。 * 无环性：最小生成树不包含任何环路。 * 最小权重：在所有可能的生成树中，最小生成树的边权重之和最小。   通过实现和分析Kruskal算法，我们能够深入理解图的最小生成树的构造过程及其应用，如网络设计、聚类分析等领域。该实验也将进一步提升我们对算法设计、实现及优化的综合能力。 | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 【实验内容与记录（题号、操作步骤、数据记录与处理、附图编号、代码等）】  **操作步骤**  定义节点和边：首先定义图的节点数和边集，每条边用其两个节点和权值表示。  调用Kruskal函数：编写并调用Kruskal算法函数来计算最小生成树及其权值。  绘制图形：使用Matlab的绘图功能展示输入图和计算出的最小生成树。  matlab代码   |  | | --- | | Kruskals.m | | function [tree, weight] = Kruskal(nodes, edges)  % Kruskals算法求最小生成树  % nodes: 节点数  % edges: 边集，每行三列，分别是边的两个节点和权值  % tree: 最小生成树的边集  % weight: 最小生成树的权值      edges = sortrows(edges, 3); % 按权值排序      tree = zeros(size(edges)); % 最小生成树的边集      treeCount = 0;% 最小生成树的边数      sets = num2cell(1:nodes);% 节点集合      weight = 0;% 最小生成树的权值      for i = 1:size(edges, 1)% 遍历边集          n1 = edges(i, 1);% 边的一个节点          n2 = edges(i, 2);% 边的另一个节点          s1 = find(cellfun(@(s) ismember(n1, s), sets));% 查找节点所在集合          s2 = find(cellfun(@(s) ismember(n2, s), sets));% 查找节点所在集合      % 算法的关键      %在sets这个元胞数组中使用cellfun和ismember函数查找节点n1和n2所在的集合s1和s2      %cellfun对sets中的每个元素应用了一个匿名函数@(s) ismember(n1/n2, s)，这个匿名函数 @(s) ismember(n1, s) 接受一个参数 s，并检查 n1 是否是 s 的一个元素。ismember 函数返回一个逻辑值，如果 n1 是 s 的一个元素，那么返回 true，否则返回 false。因此，cellfun(@(s) ismember(n1, s), sets) 返回一个逻辑数组，其中的每个元素表示 n1 是否是 sets 中对应元素的一个成员。最后，find 函数被用来找到所有 true 的元素的索引。所以，s1 是一个包含了所有包含 n1 的 sets 元素的索引的数组。          if s1 ~= s2              treeCount = treeCount + 1;              tree(treeCount, :) = edges(i, :);              weight = weight + edges(i, 3);              sets{s1} = union(sets{s1}, sets{s2});              sets(s2) = [];          end      end      tree = tree(1:treeCount, :);  end |  |  | | --- | | Input.m | | % 定义图的节点和边  nodes =10;  edges = [7 8 1; 1 2 2; 1 3 3; 4 6 3; 2 3 5; 5 6 4;4 5 5;2 7 8; 3 8 8; 2 4 18];  % 调用 Kruskal 函数  [tree, weight] = Kruskal(nodes, edges);  % 绘制输入的图  G = graph(edges(:,1), edges(:,2), edges(:,3));  figure;  subplot(1,2,1);  plot(G, 'EdgeLabel',G.Edges.Weight);  title('Input Graph');  % 绘制生成的最小生成树  T = graph(tree(:, 1), tree(:, 2), tree(:, 3));  subplot(1, 2, 2);  p = plot(T, 'EdgeLabel', T.Edges.Weight);  title('Minimum Spanning Tree');  p.NodeColor = 'r';  p.EdgeColor = 'k';  p.MarkerSize = 10;  p.LineWidth = 2;  p.NodeLabel = cellstr(num2str((1:T.numnodes)')); |     得到了正确的结果 |

|  |
| --- |
| 【小结与讨论】  本次实验通过Matlab实现Kruskal算法，并成功计算出给定图的最小生成树，达到了预期的实验目标。以下是我们在实验过程中的一些关键点和反思。  首先，我们对Kruskal算法的实现进行了详细的分步操作。通过对边集按权值排序，以及使用并查集（Union-Find）来管理和合并节点集合，我们确保了算法在处理连通无向图时的效率和正确性。特别是在集合查找和合并操作中，利用cellfun和ismember函数查找节点所在集合的索引，确保了代码的简洁和高效。  实验结果显示，输入图的最小生成树的边集和权值总和与理论预期一致。这不仅验证了Kruskal算法的有效性，还进一步加深了我们对最小生成树问题的理解。从结果可以看出，Kruskal算法通过选择权值最小且不构成环的边，逐步构建了最小生成树，最终生成了一棵连通且无环的树，且其总权值最小。  通过绘制输入图和最小生成树，我们直观地观察到算法的运行过程和结果。输入图和最小生成树的对比清晰展示了Kruskal算法如何通过逐步添加边来构建最小生成树的过程。  在讨论算法的效率时，Kruskal算法的时间复杂度主要由边排序和并查集操作决定。边的排序时间复杂度为，并查集操作接近于常数时间复杂度，其中为边数，为顶点数。  本次实验不仅巩固了我们对Kruskal算法和最小生成树问题的理解，还提升了我们在Matlab环境下进行算法实现和数据可视化的能力。通过实验操作和结果分析，我们对图论算法的实际应用有了更深的认识，为后续更复杂的算法学习和研究打下了坚实的基础。 |