山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201900161140 | 姓名： 张文浩 | | 班级： 19级人工智能 |
| 实验题目：贪心算法——最小生成树 | | | |
| 实验学时：2 | | 实验日期： 12.14 | |
| 实验目的：  1、掌握贪心算法，实现kruskal和prim最小生成树算法 | | | |
| 软件开发工具：  Vscode | | | |
| 1. 实验内容   **题目描述**：  使用kruskal算法实现最小生成树  **输入输出格式**：  **输入**：  第一行两个整数n，e。n(1<=n<=200000)代表图中点的个数，e (0≤*m*≤500000) 代表边的个数。 接下来e行，每行代表一条边：   * i j w 表示顶点i和顶点j之间有一条权重为w的边   **输出**：  最小生成树所有边的权重和   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）   Kruskal算法是基于贪心的思想得到的。首先我们把所有的边按照权值先从小到大排列，接着按照顺序选取每条边，如果这条边的两个端点不属于同一集合，那么就将它们合并，直到所有的点都属于同一个集合为止。至于怎么合并到一个集合，那么这里我们就可以用到一个工具——并查集。  换而言之，Kruskal算法就是基于并查集的贪心算法。  1) 克鲁斯卡尔 (Kruskal) 算法，是用来求加权连通图的最小生成树的算法 。  2) 基本思想 ：按照权值从小到大的顺序选择 n-1 条边，并保证这 n-1 条边不构成回路  3) 具体做法 ：首先构造一个只含 n 个顶点的森林，然后依权值从小到大从连通网中选择边加入到森林中，并使森林中不产生回路，直至森林变成一棵树为止  根据前面介绍的克鲁斯卡尔算法的基本思想和做法，我们能够了解到，克鲁斯卡尔算法重点需要解决的以下两个问题：  **问题一** 对图的所有边按照权值大小进行排序。  **问题二** 将边添加到最小生成树中时，怎么样判断是否形成了回路。  问题一很好解决，采用排序算法进行排序即可。  问题二，处理方式是：记录顶点在"最小生成树"中的终点，顶点的终点是"在最小生成树中与它连通的最大顶点"。然后每次需要将一条边添加到最小生存树时，判断该边的两个顶点的终点是否重合，重合的话则会构成回路。  关于并查集的知识：  1、用集合中的某个元素来代表这个集合，则该元素称为此集合的代表元；  2 、一个集合内的所有元素组织成以代表元为根的树形结构；  3 、对于每一个元素 x，pre[x] 存放 x 在树形结构中的父亲节点（如果 x 是根节点，则令pre[x] = x）；  4 、对于查找操作，假设需要确定 x 所在的的集合，也就是确定集合的代表元。可以沿着pre[x]不断在树形结构中向上移动，直到到达根节点。  因此，基于这样的特性，并查集的主要用途有以下两点：  1、维护无向图的连通性（判断两个点是否在同一连通块内，或增加一条边后是否会产生环）；  2、用在求解最小生成树的Kruskal算法里。  具体到代码中，首先要建立边的结构体，因为要对边进行排序，所以要重载 < 运算符。   1. **struct** Node 2. { 3. **int** a, b, c; //a,b,c分别是边的起点终点和边的权重 4. **bool** operator < (**const** Node x) **const** { 5. **return** c < x.c; 6. } 7. }node[N];   并查集找到一个顶点的“归属”，同时要注意使用**路径压缩**。   1. **int** find(**int** x) { 2. **if** (pre[x] == x) { 3. **return** x; 4. } 5. **return** pre[x] = find(pre[x]); 6. }   判断两个顶点x，y是否需要合并，如果进行合并说明加上<x,y>这条边之后不会形成环路，如果利用find函数发现x和y的代表源是同一个，说明加上<x,y>这条边会形成环路，不能加，这条边不能作为最小生成树中的一条边。   1. **bool** join(**int** x, **int** y) { 2. x = find(x); 3. y = find(y); 4. **if** (x == y) **return** **false**; 5. pre[x] = y; 6. **return** **true**; 7. } 8. 测试结果（测试输入，测试输出）   样例输入  7 12  1 2 9  1 5 2  1 6 3  2 3 5  2 6 7  3 4 6  3 7 3  4 5 6  4 7 2  5 6 3  5 7 6  6 7 1  输出  16   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   最小生成树相关概念：  **带权图**：边赋以权值的图称为网或带权图，带权图的生成树也是带权的，生成树T各边的权值总和称为该树的权。  **最小生成树（MST）**：权值最小的生成树。  **最小生成树的性质**：假设G＝(V,E)是一个连通网，U是顶点V的一个非空子集。若(u,v)是一条具有**最小权值的边**，其中u∈U，v∈V－U，则必存在一棵包含边(u,v)的最小生成树。  完成构造网的最小生成树必须解决下面**两个问题**：  （1）尽可能选取权值小的边，但不能构成回路；  （2）选取n－1条恰当的边以连通n个顶点；  **prim算法**适合**稠密图**，**kruskal算法**适合**简单图**。  **kruskal**远离更为简单粗暴，但是需要借助**并查集**这一知识。  克鲁斯卡尔算法的基本思想是**以边为主导地位**，始终选择当前可用的最小边权的边（可以直接快排或者algorithm的sort）。每次选择边权最小的边链接两个端点是kruskal的规则，并实时判断两个点之间有没有间接联通。  **并查集**  并查集是一种树形结构，又叫“不相交集合”，保持了一组不相交的动态集合，每个集合通过一个代表来识别，代表即集合中的某个成员，通常选择根做这个代表。  　　三种**主要操作**：  Make\_Set(x):  建立一个新的集合，其唯一成员就是x，因此这个集合的代表也是x，并查集要求各集合是不相交的，因此要求x没有在其他集合中出现过。  Find\_Set(x):  返回能代表x所在集合的节点，通常返回x所在集合的根节点。有递归和非递归两种方法  Union(x, y):  将包含x,y的动态集合合并为一个新的集合。合并两个集合的关键是找到两个集合的根节点，如果两个根节点相同则不用合并；如果不同，则需要合并。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释） 2. #include<bits/stdc++.h> 3. **using** **namespace** std; 5. **const** **int** N = 500005; 6. **struct** Node 7. { 8. **int** a, b, c; //a,b,c分别是边的起点终点和边的权重 9. **bool** operator < (**const** Node x) **const** { 10. **return** c < x.c; 11. } 12. }node[N]; 14. **int** pre[N]; 16. **int** find(**int** x) { 17. **if** (pre[x] == x) { 18. **return** x; 19. } 20. **return** pre[x] = find(pre[x]); 21. } 23. **bool** join(**int** x, **int** y) { 24. x = find(x); 25. y = find(y); 26. **if** (x == y) **return** **false**; 27. pre[x] = y; 28. **return** **true**; 29. } 31. **int** main() 32. { 33. **int** n, m, p; 34. cin >> n >> m; 35. **int** cnt = 0; 36. **for** (**int** i = 1; i <= n; i++) { 37. pre[i] = i; 38. } 39. **for** (**int** i = 1; i <= m; i++) { 40. cin >> node[cnt].a >> node[cnt].b >> node[cnt].c; 41. cnt++; 42. } 43. sort(node, node + cnt); 44. **long** **long** ans = 0; 45. **for** (**int** i = 0; i < cnt; i++) { 46. **if** (join(node[i].a, node[i].b)) { 47. ans += node[i].c; 48. } 49. } 50. cout << ans << endl; 51. // system("pause"); 52. **return** 0; 53. } | | | |