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prim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）   MST（Minimum Spanning Tree，最小生成树）问题有两种通用的解法，**Prim算法**就是其中之一，它是从点的方面考虑构建一颗MST，大致思想是：设图G顶点集合为U，首先任意选择图G中的一点作为起始点a，将该点加入集合V，再从集合U-V中找到另一点b使得点b到V中任意一点的权值最小，此时将b点也加入集合V；以此类推，现在的集合V={a，b}，再从集合U-V中找到另一点c使得点c到V中任意一点的权值最小，此时将c点加入集合V，直至所有顶点全部被加入V，此时就构建出了一颗MST。因为有N个顶点，所以该MST就有N-1条边，每一次向集合V中加入一个点，就意味着找到一条MST的边。  因为题目给出的数据范围的顶点个数的取值范围是n(1<=n<=200000)，所以无法采用邻接矩阵的方法存储边结构，所以我采用**链式前向星**的方法存储边结构，**链式前向星**需要一个head[]数组做为每个顶点能到的边的链的起点   1. **int** head[N];   然后定义边的结构体   1. **struct** Edge 2. { 3. **int** t,w,nxt; 4. }edge[2\*M];   其中，t为这个边的指向的顶点，w为这个边的权重，nxt指向当前顶点能到达的下一条边的索引，因为这是无向图，所以边的容量要考虑总边数的2倍的大小。再定义一个添加边结构的函数   1. **void** add(**int** x,**int** y,**int** \_w){ 2. edge[num].t=y; 3. edge[num].w=\_w; 4. edge[num].nxt=head[x]; 5. head[x]=num++; 6. }   在prim算法中，先初始化min\_cost数组为inf（一个很大的数），然后将顶点1所直接到达的边的min\_cost设置为对应边的权重。   1. **for**(**int** i=1;i<=n;i++) min\_cost[i]=inf; 2. **for**(**int** i=head[1];i!=-1;i=edge[i].nxt){ 3. min\_cost[edge[i].t]=edge[i].w; 4. }    **for**(**int** i=1;i<=n;i++) min\_cost[i]=inf; 5. **for**(**int** i=head[1];i!=-1;i=edge[i].nxt){ 6. min\_cost[edge[i].t]=edge[i].w; 7. }   初始化min\_cost数组结束后进入while循环，每次选出一个没有经过且到点集距离最小的顶点，然后必定是新加入的点影响了其他未加入的点到这个点集的距离，以这个新点为边点，更新所有未加入点到这个点击的距离。不断循环直到所有的店都被遍历到。  **优化**  这样提交到oj上有一半的点过不了，原因是超市，说明时间复杂度太高了，要进行优化，分析我们的代码，发现主要是因为每次“找到一个没有经过且到点集距离最小的顶点”这个过程比较慢，因为是通过遍历n个顶点得到的，   1. //选出一个没有经过且到点集距离最小的顶点 2. **for**(**int** i=1;i<=n;i++){ 3. **if**(!vis[i] && (!v || min\_cost[v]>min\_cost[i])) 4. v=i; 5. }   于是想到可以在这里进行优化，优化的方法是采用优先队列，建立一个优先队列，存储边结构体，   1. priority\_queue<Edge> q;   因为在优先队列中要对Edge边结构体进行排序，且因为优先队列默认降序排列，而我们每次想找到最小值，所以改为升序，所以我们要自己定义边结构体的运算符<   1. **struct** Edge 2. { 3. **int** t,w,nxt; 4. **bool** operator < (**const** Edge& x)**const**{ 5. **return** w>x.w; 6. } 7. }edge[2\*M];   优先队列的初始化方法是将顶点1所直接连通的边的结构体放入优先队列   1. **for**(**int** i=head[1];i!=-1;i=edge[i].nxt){ 2. q.push(edge[i]); 3. }   再进入while循环，每次循环的时候我们想找到距离点集最短的边就是队首元素，取出来删掉即可。  经过优化之后降低了时间复杂度，就不会超时了。   1. 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）     Prim算法的关键是每一次循环要找到，距离点集最短的边，要维护一个min\_cost数组，所以每次在找的时候复杂度是比较高的，大数据量的点是过不了的（超时），所以最终用了堆优化的方法，利用优先队列存储边的结构，将符合条件的边（没有被遍历过，且与点集直接关联）放入优先队列，每次要找到距离点集最短的边就是优先队列队首的边结构。   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）   未优化版   1. #include<bits/stdc++.h> 2. **using** **namespace** std; 4. **const** **int** N = 200005,inf = 0x3f3f3f3f,M = 500005; 5. **bool** vis[N]; 6. **int** n,m,num=0; 7. **long** **long** min\_cost[N]; 8. **int** head[N]; 9. vector<**int**> ve; 11. **struct** Edge 12. { 13. **int** t,w,nxt; 14. }edge[2\*M]; 16. **void** add(**int** x,**int** y,**int** \_w){ 17. edge[num].t=y; 18. edge[num].w=\_w; 19. edge[num].nxt=head[x]; 20. head[x]=num++; 21. } 23. **long** **long** prim() 24. { 25. **long** **long** ans=0,tot=1; 26. vis[1]=**true**; 27. **for**(**int** i=1;i<=n;i++) min\_cost[i]=inf; 28. **for**(**int** i=head[1];i!=-1;i=edge[i].nxt){ 29. min\_cost[edge[i].t]=edge[i].w; 30. } 31. **while**(**true**){ 32. **int** v=0; 33. //选出一个没有经过且到点集距离最小的顶点 34. **for**(**int** i=1;i<=n;i++){ 35. **if**(!vis[i] && (!v || min\_cost[v]>min\_cost[i])) 36. v=i; 37. } 38. **if**(!v) **break**; 39. tot++; 40. ans+=min\_cost[v]; 41. vis[v]=**true**; 42. //必定是新加入的点影响了其他未加入的点到这个点集的距离，以这个新点为边点，更新所有未加入点到这个点击的距离 43. **for**(**int** i=head[v];i!=-1;i=edge[i].nxt){ 44. **int** to=edge[i].t,weight=edge[i].w; 45. **if**(!vis[to] && min\_cost[to]>weight){ 46. min\_cost[to]=weight; 47. } 48. } 49. } 50. **if**(tot!=n) **return** -1; 51. **return** ans; 52. } 54. **int** main() 55. { 56. cin>>n>>m; 57. **for**(**int** i=1;i<=n;i++) ve.push\_back(i); 58. memset(head,-1,**sizeof**(head)); 59. **int** x,y,w; 60. **for**(**int** i=0;i<m;i++){ 61. cin>>x>>y>>w; 62. add(x,y,w); 63. add(y,x,w); 64. } 65. cout<<prim()<<endl; 66. // system("pause"); 67. **return** 0; 68. }   使用优先队列优化版   1. #include<bits/stdc++.h> 2. **using** **namespace** std; 4. **const** **int** N = 200005,inf = 0x3f3f3f3f,M = 500005; 5. **bool** vis[N]; 6. **int** n,m,num=0; 7. **long** **long** min\_cost[N]; 8. **int** head[N]; 9. vector<**int**> ve; 11. **struct** Edge 12. { 13. **int** t,w,nxt; 14. **bool** operator < (**const** Edge& x)**const**{ 15. **return** w>x.w; 16. } 17. }edge[2\*M]; 19. **void** add(**int** x,**int** y,**int** \_w){ 20. edge[num].t=y; 21. edge[num].w=\_w; 22. edge[num].nxt=head[x]; 23. head[x]=num++; 24. } 26. priority\_queue<Edge> q; 28. **long** **long** prim() 29. { 30. **long** **long** ans=0,cnt=n-1; 31. vis[1]=**true**; 32. **for**(**int** i=head[1];i!=-1;i=edge[i].nxt){ 33. q.push(edge[i]); 34. } 35. **while**(!q.empty()&&cnt){ 36. Edge p=q.top(); 37. q.pop(); 38. **if**(vis[p.t]) **continue**; 39. ans+=p.w; 40. cnt--; 41. vis[p.t]=**true**; 42. **for**(**int** i=head[p.t];i!=-1;i=edge[i].nxt){ 43. Edge pp = edge[i]; 44. **if**(!vis[pp.t]) q.push(pp); 45. } 46. } 47. **return** ans; 48. } 50. **int** main() 51. { 52. cin>>n>>m; 53. **for**(**int** i=1;i<=n;i++) ve.push\_back(i); 54. memset(head,-1,**sizeof**(head)); 55. **int** x,y,w; 56. **for**(**int** i=0;i<m;i++){ 57. cin>>x>>y>>w; 58. add(x,y,w); 59. add(y,x,w); 60. } 61. cout<<prim()<<endl; 62. // system("pause"); 63. **return** 0; 64. } | | | |