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：202200400053 | 姓名： 王宇涵 | | 班级： 22级2班 |
| 实验题目：最小生成树 | | | |
| 实验学时：2 | | 实验日期： 2023-12-13 | |
| 实验目的：  掌握最小生成树的 Prim 算法和 Kruskal 算法及其实现。 | | | |
| 软件开发环境：  Vscode | | | |
| 1. 实验内容   **1. 题目描述： 使用 Prim 算法计算最小生成树。**  **输入输出格式：**  **输入**：第一行输入两个整数 n、e，其中 n（1≤n≤200000) 表示点的个数，e（0≤e≤500000）表示边的个数。接下来e 行，每行表示一条边：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**：最小生成树中所有边的权重和。   1. **题目描述： 使用 Kruskal 算法计算最小生成树。**   **输入输出格式**：  **输入：**第一行输入两个整数 n、e，其中 n（1≤n≤200000) 表示点的个数，e（0≤e≤500000）表示边的个数。接下来e 行，每行表示一条边：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：**最小生成树中所有边的权重和。   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）   **Prim算法**  整体思路：  使用邻接表来表示图，其中 h[] 数组存储每个顶点的邻接边链表。  利用优先队列 priority\_queue 来实现最小堆，用于按边权值大小来选取最小的边。  使用 d[] 数组来记录每个顶点到当前生成树的最小边权值。  先将第一个顶点放入最小堆,如果堆不空,就取出堆头元素,如果没有遍历过这个元素,便放入最小生成树集合中,并更新相邻顶点的最小边权值，若可以更新,则将相邻顶点放入最小堆中,重复循环,直到堆空为止  **数据结构和算法**  邻接表：使用 h[] 数组作为邻接表来存储图的边关系。  优先队列：priority\_queue 用于按边权值大小来选取最小的边。  Prim 算法：使用 Prim 算法寻找最小生成树，不断更新生成树中顶点的最小边权值，并选取最小边加入生成树。  **Cruskal算法**  整体思路：  使用并查集来判断图中的节点是否属于同一个连通分量。  结构体 Edge 存储边的信息，包括边的两个端点和边权值。  对所有边按照权值从小到大进行排序。  遍历排序后的边，对于每条边的两个端点，若不在同一个连通分量中，则将它们合并，并将边的权值加入到最小生成树的权值之和中。  **数据结构和算法**  并查集：用于判断节点所属的连通分量。  结构体 Edge：用于存储边的信息。  find(): 并查集中的查找函数，用于找到节点所属的连通分量,并进行路径压缩.  cmp(): 自定义的比较函数，用于sort函数对边按照权值大小进行排序。   1. 测试结果（测试输入，测试输出）   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）   **注意:**本次实验的数据范围比较大,一开始没有注意到res的范围,开了int res,会有WA,最后开了long long res,成功AC.  **问题一:prim算法数据范围太大,无法使用邻接矩阵来做,如何处理?**  答: 可以使用无向邻接表来进行处理,可以避免数据过大的问题  **问题二:cruskal 如何实现对边的合并?**  答: 通过并查集来进行判断,如果边的两个顶点不属于同一个集合,则进行合并,更新res,否则不进行操作   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）   **Prim.cpp**  #include<iostream>  #include<cstring>  #include<queue>  using namespace std;  typedef pair<int,int> PII;  const int N=1e6+10;  int n,m;  int e[N],ne[N],w[N],h[N],idx;  int d[N];  bool st[N];  priority\_queue<PII,vector<PII>,greater<PII>> heap;  void add(int a,int b,int c)  {  e[idx]=b;  ne[idx]=h[a];  w[idx]=c;  h[a]=idx++;  }  long long prim()  {  long long res=0;  memset(d,0x3f,sizeof(d));  d[1]=0;  heap.push({0,1});  while(!heap.empty())  {  auto t=heap.top();  heap.pop();  int val=t.second,dis=t.first;  //取出该结点,包含到集合里面来  if(!st[val])  {  res+=dis;  st[val]=true;  //对于结点的每个边,都进行遍历  for(int i=h[val];i!=-1;i=ne[i])  {  int j=e[i];  //更新的时候才放入  if(d[j]>w[i])  {  d[j]=w[i];  heap.push({d[j],j});  }  }  }  }    return res;  }  int main()  {  ios::sync\_with\_stdio(false);  cin.tie(0);  cin>>n>>m;  memset(h,-1,sizeof(h));  while(m--)  {  int x,y,z;  cin>>x>>y>>z;  {  add(x,y,z);  add(y,x,z);  }  }    cout<<prim();  return 0;  }  **Cruskal.cpp**  #include<iostream>  #include<cstring>  #include<algorithm>  using namespace std;  const int N=3e5+10;  const int M=2\*N;  int n,m;  int idx;  //并查集  int p[N];  struct Edge  {  int v1,v2,w;  }Edges[M];  int find(int x)  {  if(p[x]!=x)  p[x]=find(p[x]);  return p[x];  }  bool cmp(Edge a,Edge b)  {  return a.w<b.w;  }  long long cruskal()  {  long long res=0;  int cnt=0;  for(int i=1;i<=n;i++)  p[i]=i;    sort(Edges,Edges+idx,cmp);  for(int i=0;i<idx;i++)  {  auto k=Edges[i];  int a=k.v1,b=k.v2,c=k.w;  a=find(a);b=find(b);  if(a!=b)  {  p[a]=b;  res+=c;  }  }  return res;  }  int main()  {  ios::sync\_with\_stdio(false);  cin.tie(0);  cin>>n>>m;  int a,b,c;  while(m--)  {  cin>>a>>b>>c;  Edges[idx++]=Edge{a,b,c};  }  cout<<cruskal()<<" ";  } | | | |
|  | | | |