求最小支撑树的普里姆算法

```
void Graph_Matrix :: Prim ( )
{
        int n = graphsize;
         struct LV
         {
             int Lowcost ;
             int Vex ;
          }* closedge = new LV[n];
          struct Edge
         {
              int head;
              int tail;
              int cost;
          }*TE = new Edge[n-1];

          //  初始化邻接矩阵
          for (int i = 0; i<n; i++)
             for (int j = 0; j<n; j++)
                  if (edge[i][j] == 0) edge[i][j] = max;    // max 是预定义的常数
            //  以顶点 0 为初始顶点，初始化数组 closedge
            for (i = 0 ; i < n ; i ++ )
            {
                       closedge[i].Lowcost = edge[0][i] ;
                       closedge[i].Vex = 0 ;
            }
            closedge[0].Vex = -1 ;                        //  顶点 0 进入集合 U
            int count = 0 ;                                     //  支撑树的边记数器 count
            for ( i = 1 ; i < n ; i ++ )                  //  循环 n-1 次
            {
                 int min = max+1 ;                             //  设置最小值 min
                 int v = 0 ;
                 for ( int j = 0 ; j < n ; j ++ )                 //  求当前权值最小的边和该边的终点 v
                     if ( closedge[j].Vex != -1 && closedge[j].Lowcost < min )
                     {
                            v = j ;
                            min = closedge[j].Lowcost ;
                     }
                 if ( v != 0 )                                    //  若 v= =0，说明没有找到符合条件的顶点
                 {
                            //  向支撑树的边集合 TE 中添入一个边
                        TE[count].head = closedge[v].Vex ;
                        TE[count].tail = v;
                        TE[count].cost = closedge[v].Lowcost ;
                            count++ ;                             //  计数器加 1
                            closedge[v].Lowcost = 0 ;   //  修改域值
                            closedge[v].Vex = -1 ;         //  顶点 v 进入集合 U
                            //  因为 v 的进入，而要修改某些值
                            for ( j = 0 ; j < n ; j ++ )
                                  if ( closedge[j].Vex != -1 && edge[v][j] < closedge[j].Lowcost )
                                  {
                                         closedge[j].Lowcost = edge[v][j] ;
                                         closedge[j].Vex = v ;
                                   }
                 }
     }
    for (i=0; i<n-1; i++)
          cout<<"("<<TE[i].head<<", "<<TE[i].tail<<", "<<TE[i].cost<<")"<<"\n";
     delete[] closedge;
    delete[] TE;
}
```