# 图的应用实验报告

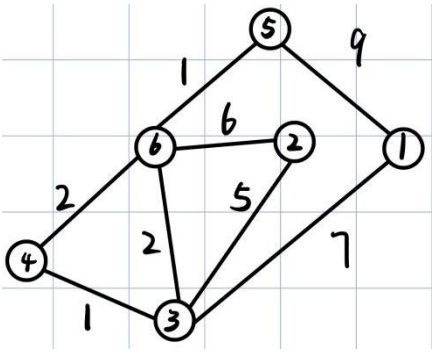
## 一、问题分析

- 处理对象：将每个村落看作图的顶点，将可能建设的光纤看作链接顶点的边，则待处理的对象可看作一幅无向图。
- 实现功能：求出每个村落都有光纤连通所需要的最低成本，即求出无向图的最小生成树，最后对该最小生成树每条边的权重进行求和，输出计算结果。其中还需判断该无向图是否连通，即判断是否能求出最小生成树，如果不能则输出-1。
- 结果显示：如果存在最小生成树，则输出其中边的权重求和结果，否则输出-1。
- 题目输入样例：

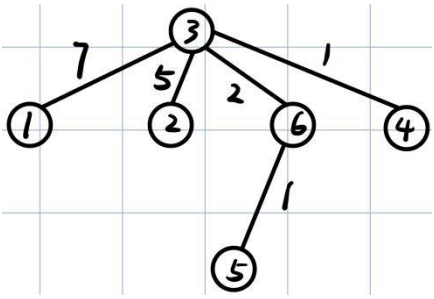
```
6 8
1 3 7
1 5 9
2 3 5
2 6 6
3 4 1
3 6 2
4 6 2
5 6 1
```

- 求解过程：

根据输入结果，可得无向图：




可求得其一最小生成树为




每条边相加求和：7+5+2+1+1=16

最后输出 16。

# 二、数据结构和算法设计

## 1. 抽象数据类型设计：

即 Graph_ADT，其中封装了无向图结构的多种函数接口，如无向图的构造、析构、初始化、获取顶点/边数、获取某个顶点在邻接矩阵中的第一个相邻顶点、获取某一顶点的访问情况以及 Prim 算法的实现等对图进行基本操作的函数，具体设计如下：

```
class Graph {
private:
  void operator =(const Graph&) {}     // Protect assignment
  Graph(const Graph&) {}          // Protect copy constructor

public:
  Graph() {}          // Default constructor
  virtual ~Graph() {} // Base destructor
```

```
 // Initialize a graph of n vertices
   virtual void Init(int n) = 0;

 // Return: the number of vertices and edges
   virtual int n() = 0;
   virtual int e() = 0;

 // Return v's first neighbor
   virtual int first(int v) = 0;

// Return v's next neighbor
   virtual int next(int v, int w) = 0;

 // Set the weight for an edge
 // i, j: The vertices
 // wgt: Edge weight
   virtual void setEdge(int v1, int v2, int wght) = 0;

 // Delete an edge
 // i, j: The vertices
   virtual void delEdge(int v1, int v2) = 0;

 // Determine if an edge is in the graph
 // i, j: The vertices
 // Return: true if edge i,j has non-zero weight
   virtual bool isEdge(int i, int j) = 0;

 // Return an edge's weight
 // i, j: The vertices
 // Return: The weight of edge i,j, or zero
   virtual int weight(int v1, int v2) = 0;

 // Get and Set the mark value for a vertex
 // v: The vertex
 // val: The value to set
   virtual int getMark(int v) = 0;
   virtual void setMark(int v, int val) = 0;

   virtual int getInDegree(int v) = 0;
   virtual int getOutDegree(int v) = 0;
   //Prim算法求最小生成树
   virtual void Prim(int* D, int s) = 0;
    virtual int minertex(int* D) = 0;
   virtual void AddEdgetoMST(int v1, int v2) = 0;
```

```
};
```

## 2. 物理数据对象设计:

用图的邻接矩阵来表示目标无向图，Graphm 类公有继承 Graph 具体实现 Graph 中接口内部的函数，实现实例化，且设置私有成员数据 numVertex（顶点数），numEdge（边数），二维数组 matrix（表示邻接矩阵），一维数组 mark（记录每个顶点访问情况），数据类型均为 int 型。

## 3. 算法思想设计:

1. 根据输入的数据，构造无向图 G，定义一个整型数组 D 记录生成树中相邻节点边的的权重，调用物理数据对象中设计好的 Prim 算法函数。

2. Prim 算法具体思想为：设定一个顶点为根结点，不断更新数组 D 中元素，每次找到权制最小的边，将这条边的加入生成树，将其权制记录在数组 D 中

3. 判断该图是否为连通图，是连通图则输出 D 中所有元素求和的结果，不是连通图这输出-1。

## 4. 关键功能的算法步骤:

1.初始化 D 中元素均为 infinity(100000)，调用 Prim 函数从顶点 1 开始求生成树。

2.Prim 算法具体步骤为：

①选择顶点 1 作为生成树的根结点，将其标记为已访问，然后找到其所有相邻顶点，找到和它相连且权重最小的边，将其加入生成树，把这条边连接的顶点标记为已访问。

②从已访问的顶点中，找到与它们相连且权重最小的边，如果这条边连接的顶点未被访问，将其加入生成树，并将该边权重加入数组 D 中，设置相应未被访问

的顶点为已访问。

③重复步骤②，直到该图的一个连通分量的所有顶点已被访问。

3.判断是否为连通图：遍历数组 D，如果 D 中有一个元素为 infinity，则说明该图不止有一个连通分量，故该图不是连通图，输出-1，反之其只有一个连通分量，即该图本身，所以该图是连通图，将 D 中元素累加求和，输出计算结果。

# 三、算法性能分析

1. 如果是连通无向图，Prim 算法需要遍历所有的顶点和边，并且对所有边的权值进行比较，若顶点数为 n，则最坏情况下边数为 n-1，因为每次都要比较找到权值最小的边，需要对所有对每个顶点都要检查其所有邻接顶点间的边，则每次需比较 n-1 次，共进行 n 次比较，故时间复杂度为 O（$n^2$），最后需要遍历数组 D 判断其是否为连通图，循环 n 次，时间复杂度为 O（n），故总的时间复杂度为 O（$n^2$）。

2. 算法用到一个数组 D 储存生成树边的权值，故空间复杂度为 O（n）。

3.对于本题，Prim 算法优点是简单易懂，易于理解和实现，对处理稀疏图时方便快捷，缺点在于如果题目给出的是稠密图，Prim 算法的时间复杂度较高。