实验六

问题分析

问题与功能描述：

【问题描述】

村村通宽带

截至2021年11月底，我国现有行政村已全面实现‘村村通宽带’（通宽带是指已通光纤或通4G）。通过调查，获得了若干有可能建设光纤来连通村落的村落间光纤建设成本，求使每个村落都有光纤连通所需要的最低成本。

【输入形式】

输入第1行数据包括村落数目正整数N（<=1000)和候选光纤数目M（<=3N)。接下来M行对应M条道路，每行给出3个以空格分隔的正整数，分别是候选光纤拟连接的村落编号及该光纤建设的预估成本。（注：村落从1~N编号）

【输出形式】

输出使每个村落都有光纤连通所需要的最低成本。如果输入数据不足以保证光纤畅通，则输出-1，表示需要增设更多光纤。

样例分析

【样例输入】

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

【样例输出】

16

【评分标准】

特别说明：要求基于ADT实现，否则计0分。

分析过程

样例中输入的是6个村庄和8条候选光纤，每条光纤有连接的两个村庄编号以及该光纤建设的预估成本。具体来说，第一行表示有6个村庄，编号从1到6；第二行表示有8条候选光纤，每条光纤连接两个村庄，如第一条光纤连接第1个和第3个村庄，建设成本为7。

解题的方法是采用Prim算法（最小生成树算法）来求解。首先，需要将输入的边信息存储在图中，以便Prim算法使用。在实现Prim算法的过程中，需要创建一个option类的对象来处理最小生成树的相关操作。Prim算法使用最小堆（优先队列）来实现，在初始化时，将第0个村庄作为起点（根），将其他点到根的距离设置为无穷大，每次从优先队列中选出最小的边进行更新，直到所有的村庄都被连通。最后，输出Prim算法的结果，即最小生成树的权值和，即为每个村庄都有光纤连通所需要的最低成本。样例输出结果为16。

求解方法

1. 创建一个图 G，这里使用邻接表来表示图
2. 将 G 中所有的顶点的标记都设为 UNVISITED。
3. 输入每条边的起点、终点和权重，调用 setEdge 函数在 G 中加入这条边，注意由于是无向图，因此需要同时设置起点到终点和终点到起点两个方向的边。
4. 初始化 D 数组，D[i] 表示当前已经生成的最小生成树中到顶点 i 的距离。将 D 数组中所有的元素都设为 INFINITY，除了 D[0] 为 0，表示起点到起点的距离为 0。
5. 调用 Prim 函数，该函数接受两个参数：D 数组和起点编号。在 Prim 函数中，首先将起点加入最小生成树的顶点集合 S 中，然后将起点的所有邻居（即与起点有边相连的顶点）加入最小生成树的邻居集合 T 中。接下来重复以下步骤，直到 T 中包含了所有顶点：

   a. 在 T 中找到与 S 最近的顶点 u，将 u 加入 S 中。

   b. 更新 D 数组，如果 D[v] > w(u,v)，则将 D[v] 更新为 w(u,v)，表示将顶点 v 加入最小生成树的最短路径为 S 中的顶点到顶点 v 的路径。
6. 最后返回 D 数组中所有元素的和，即为生成的最小生成树的权重和。

数据结构和算法设计：

【抽象数据类型设计】

1. 顶点（Vertex）：每个顶点用一个整数编号表示。
2. 边（Edge）：边是一条连接两个顶点的有向边，它具有以下属性：
   - 起点：边的起点是一个顶点，可以通过该边从该顶点到达另一个顶点。
   - 终点：边的终点是一个顶点，它被该边连接到。
   - 权值（weight）：每条边都有一个权值，表示连接两个顶点的代价。
3. 图（Graph）：图是由一组顶点和一组边构成的数学结构，它具有以下操作：
   - 初始化：初始化图，设置图的初始大小和类型。
   - 添加边：将一条边添加到图中，包括边的起点、终点和权值。
   - Prim算法：使用Prim算法计算最小生成树，输出最小生成树的权值和。
4. 选项（option）：选项类是一个辅助类，用于Prim算法的执行，其主要操作有：
   - 构造函数：通过图对象的引用构造一个选项对象。
   - Prim算法：实现Prim算法，用于计算最小生成树的权值和。

【物理数据对象设计】

- Graphl类和Graphm类，分别表示使用邻接表和邻接矩阵存储图的数据结构。
- Vertex类，表示图中每个顶点的数据结构，包括顶点编号和邻接表指针等信息。
- Edge类，表示图中每条边的数据结构，包括边的起点、终点和权值等信息。
- option类，用于Prim算法中的最小生成树相关操作。
- int型数组D，用于存储Prim算法中每个顶点到最小生成树的距离。
- int型变量fr_vert，to_vert和wt，分别表示输入的每条边的起点、终点和权值。

在邻接表表示法中，每个顶点的邻接表由链表实现，链表中存储该顶点所有的出边信息。因此，在Graphl类的实现中，使用了一个`vector<Vertex>`数组来存储所有顶点的信息，每个顶点对应一个链表。在Graphm类的实现中，则使用了一个二维数组来存储顶点之间的边的权值。

【算法思想的设计】

1. 初始化一个空的优先队列，用于存储候选边；
2. 初始化一个D数组，用于存储每个村庄到最小生成树的最小距离；
3. 将第一个村庄作为根节点，将D[0]设置为0，并将所有其他点的D值初始化为正无穷大；
4. 将以第一个村庄为起点的所有边加入到优先队列中；
5. 当优先队列不为空时，取出其中的最小边，假设该边连接的两个村庄为v和w；
6. 如果D[w]大于该边的权值，则将D[w]更新为该边的权值，并将该边加入最小生成树中；
7. 将所有以w为起点的边加入到优先队列中，重复步骤5~7，直到所有村庄都被加入到最小生成树中。

最后，输出Prim算法的结果，即最小生成树的权值和，即为每个村庄都有光纤连通所需要的最低成本。

【物理实现】

```
Graph *createGraph(int graph_type, int vert_num);

int main()
{
    Graph* G;
    int vert_num;                       // 图的顶点数，编号从0开始
    cin >> vert_num;
    int graph_type = 1;                 // graph_type=1, 采用临接链表表示图
graph_type=0, 采用临接矩阵表示图
    G = createGraph(graph_type, vert_num);
    option *it = new option(G);

    for (int v = 0; v < G->n(); v++)
        G->setMark(v, UNVISITED);  // Initialize mark bits

    int edge_num;
    cin >> edge_num;

    int fr_vert, to_vert, wt;

    while(edge_num--)
    {
        cin >> fr_vert >> to_vert >> wt;
        fr_vert --;
        to_vert --;
        G->setEdge(fr_vert, to_vert, wt);
```

```
        G->setEdge(to_vert, fr_vert, wt);
    }
    int D[G->n()];
    for (int i = 0; i < G->n(); i++) // Initialize
        D[i] = INFINITY;
    D[0] = 0;
    cout << it->Prim(D, 0) << "\n";
    return 0;
}

Graph *createGraph(int graph_type, int vert_num)
{
    Graph *g;
    if (graph_type)
        g = new Graphl(vert_num);
    else
        g = new Graphm(vert_num);

    return g;
}
```

算法分析

关键步骤

1. 创建图对象

根据输入的顶点数和图的类型（邻接链表或邻接矩阵），创建对应的图对象。

2. 存储图的边信息

将输入的边信息存储在图中，以便Prim算法使用。

3. 初始化

将所有顶点标记为未访问（UNVISITED），并初始化距离数组D，其中D[i]表示从第0个顶点到第i个顶点的最短距离。

4. 将第0个顶点作为起点

将第0个顶点作为起点（根），将其他点到根的距离设置为无穷大，每次从优先队列中选出最小的边进行更新。

5. 更新距离和优先队列

对于每个已经访问的顶点v，枚举与v相邻的顶点w，如果w未访问过且从v到w的距离小于D[w]，则更新D[w]的值，并将(w, D[w])加入优先队列中。

6. 计算最小生成树的权值和

重复执行步骤5直到所有的顶点都被访问过。最后，计算最小生成树的权值和，即为每个顶点都有边连通所需要的最小成本。

7. 输出结果

将Prim算法的结果输出。

算法性能分析

该段代码实现了Prim算法来求解图的最小生成树，算法的时间复杂度为$O(|E|\log|V|)$。其中，$|V|$表示图的顶点数，$|E|$表示图的边数。

具体来说，Prim算法使用一个优先队列来维护每个点到已选中点集的最小边，每次从优先队列中取出权值最小的边，并将其连接的点加入已选中点集中，直到所有点都被加入已选中点集中。由于Prim算法的每个点最多只会加入一次已选中点集，因此时间复杂度为$O(|V|\log|V|)$。而对于每个点加入已选中点集时，需要遍历其邻居节点并更新它们到已选中点集的最小边，由于每条边最多被更新一次，因此需要遍历所有边，时间复杂度为$O(|E|)$。因此，Prim算法的时间复杂度为$O(|E|\log|V|)$。

在该段代码中，使用了邻接表（Graphl）来表示图，邻接表适用于稀疏图，其空间复杂度为$O(|V|+|E|)$。在遍历图时，需要遍历每个点的所有邻居节点，因此时间复杂度为$O(|V|+|E|)$。

综上，该段代码的时间复杂度为$O(|E|\log|V|)$，空间复杂度为$O(|V|+|E|)$。