# 1 项目简介

## 生活中我们会遇到这样的问题：假设一个城市有n个小区，要实现n个小区之间的电网都能够相互接通，构造这个城市n个小区之间的电网，使总工程造价最低，要求在每个小区之间都可以设置一条电网线路，都要付出相应的经济代价。选择其中的n-1条使总的耗费最少。请设计一个能够满足要求的造价方案。通过将n个小区视为n个顶点，电网线路视为顶尖之间的带权值边，形成图结构，基于图结构建立最小生成树即可得出上述问题的最优方案。

# 2 程序说明

## 2.1 数据结构设计

首先，本程序是利用Kruskal算法来建立最小生成树的。一方面，考虑到实际问题中的小区数量，即图结构中的顶点数量较多，而且利用Kruskal算法生成最小生成树的过程中需要对顶点进行频繁地移动与删除，因此本程序中利用链表来存储电网所有的边，而且在往链表中插入节点时就可直接将新节点插入到链表中第一个比待插入边权值大的节点之前，在插入节点建立链表之前就可以顺带完成各边按照权值从小到大的排序。另一方面，本程序中采用了矩阵，即二维数组表示的图结构来存储各小区及其之间的电网路线权值，并以此为基础实现Kruskal算法建立最小生成树。

## 2.2 类结构设计

①图Graphmtx类：

首先，图类中包含着顶点表（用来储存图中的顶点）、邻接矩阵（用来存放边）、maxVertices（顶点集合的最大长度，也反应着邻接矩阵的最大规模）、当前图中实际的顶点数numVertices、当前图中实际的边数numEdgs这些信息。

接下来对本程序中重点使用的图的的类函数，也就是本程序的**核心代码**进行详细的说明：



1.构造函数：首先将传入的参数设定为顶点的最多个数，若用户在实例化类时没有传入参数，则使用默认参数初始化maxVertices，之后将当前图中的顶点个数、边条数都初始化为0，接着利用一个for循环动态内存开辟maxVertices\*maxVertices的二维数组。注意：动态内存申请不能直接开辟二维数组，因此，先开辟maxVertices大小的一维指针数组，然后让每一个指针指向动态内存申请的maxVertices的一维指针数组，即可完成二维数组的开辟。

2.insertVertex函数：功能：插入一个顶点。首先判断当前顶点数量是否已达到设定上限，若触及上限，则不再插入，否则执行语句：VerticesList[numVertices++] = vertex;

需要注意的是，这里的numVertices采用的是后缀自增运算符，先用新节点值赋值给数组新位置后numVertices才会自增。

3.insertEdge函数：功能：插入一条边。首先判断插入的位置是否合理，不合理，如：超过数组范围则返回false，表示插入失败，若插入位置合理，再判断现在要插入边的两个顶点之间的距离是不是INF无限大，是则可以插入，如果不是，那么可能是要在节点自己插入一个非0权值的边，也可能是两个顶点之间已经存在了边，这时不会再次插入新边。

4.RemoveEdge函数：功能：删除某条边。将二维数组中对于的值重设为INF无限大值，再使numEdges减一即可。

1. DFS函数：功能：对图进行深度搜索，判断某两个顶点之间是否存在通路。

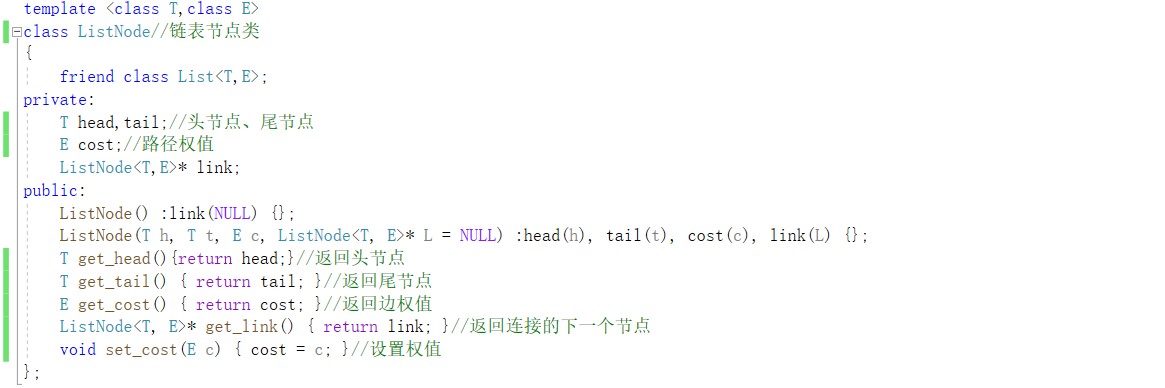
首先定义一个numVertices大小的visited数组并将其元素全部初始化为0，之后将顶点head、visited数组通过参数传入深度优先搜索函数，通过递归进行搜索。具体的递归思路如下：首先将visited[head]从0置为1，表示该节点已访问，之后利用变量w暂存head的下一个邻接顶点，接下来进入循环结构，只要w！=-1，就表明还存在其他的邻接顶点，需要一直循环，在循环过程中，如果visited[w]==0，那就说明该处未被访问过，将w代替head与visited数组作为参数调用深度优先搜索函数，实现递归，此后通过getNextNeighbor函数将w更新为下一个邻接顶点。最终递归结束以后，如果visited[tail]为1，说明head与tail之间已经存在通路，所以就不可以在head于tail之间插入直接相连的边，否则会形成死锁。

1. Kruskal函数：功能：建立最小生成树。

需要注意的是，目前图中所有的边在通过链表建立时就已经形成了按照权值从小到大的排序，因此，把顶点链表是否为空作为循环条件，没在最小生成树中建立一条边，就会从顶点链表中删除对应的顶点，顶点链表的长度自然就会减一。所以，只要顶点链表不空，就一直循环。在循环过程中，首先通过循环结构调用DFS函数，判断是否所有节点之间已经形成通路，即形成最小生成树，若没有生成，则不退出循环：不断地从边链表的第一个位置取边，取到的就是当前边中最小的边，若该边关联的两个顶点之间目前没有直接相连的边，而且通过调用DFS函数判断插入这条边之后不会形成死锁，则通过InsertEdge函数插入新边，并将该边节点从边链表中删除，将关联的顶点从顶点集合中删除。当函数执行完毕后，图中就建立了最小生成树。

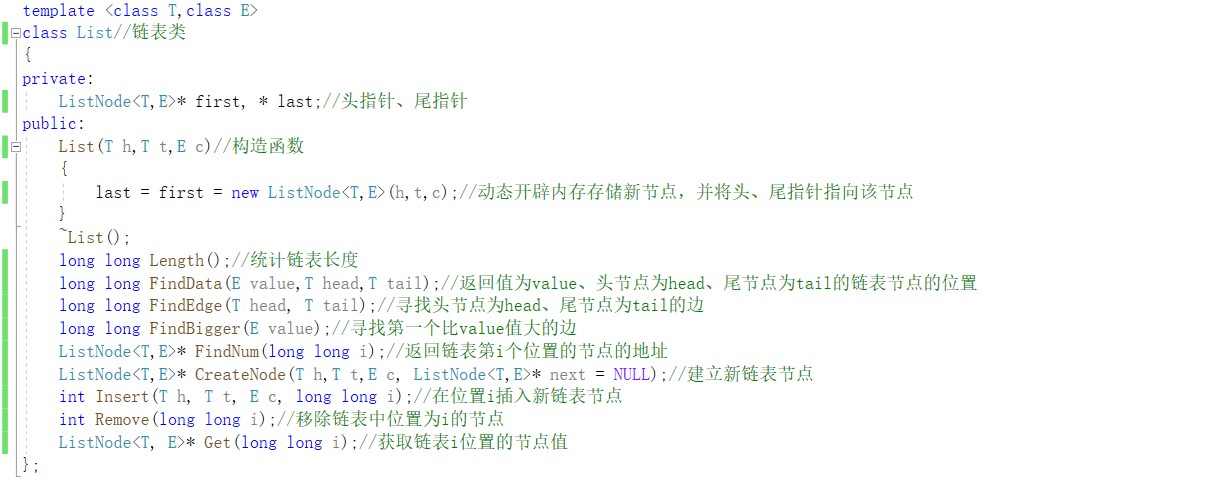
②链表节点ListNode类：

链表中每个节点需要包含的信息是边两端的两个顶点、边的权值，以及该节点连接的下一个节点地址。



## 通过在链表节点类中提供返回两个顶点、返回边权值、设置边权值等功能为链表类中调用相关函数实现完整的链表功能提供了基础支撑。

## ②链表List类：

 List带头节点的链表类包含头指针first指向链表头结点，尾指针last则指向链表中的最后一个节点，主要通过实例化类并不断插入新节点来构建链表，通过删除相应位置的节点来实现删除备选边与备选顶点的功能。此外，上述插入与删除功能的实现依赖于寻找某位置的节点、寻找某值的节点位置等功能，下文将对些类函数的**核心代码**进行详细说明：

1. 构造函数：动态开辟内存存储新节点，在实例化类时将头结点与第一个节点连接。
2. 析构函数：功能是删除单循环链表中的所有节点，主体采用循环结构实现。在本程序的链表中，只要链表不空，就一直循环执行删除操作。具体的删除流程是：用指针q指向头结点的下一个节点，并用q暂存其地址，接着令头结点指向下下个节点，与紧挨头结点的下一个节点脱钩，以便于对其执行删除操作，最后删除q指向的节点。当循环执行完毕后，链表中只余下头结点，将last指针与first指针对齐，最后删除头结点即可完成单循环链表的删除操作。
3. Length函数：功能是返回链表长度，即链表除头结点以外的节点个数，主体采用循环结构实现：首先从头结点指向的下一个节点开始计数，只要节点不空且表示当前位置的p指针没有越过last循环回到first，即未到链表末尾，则一直循环计数，并将指针指向当前节点的下一个节点。最终循环完毕后返回计数结果即可。
4. FindData 函数：功能：返回值为value的链表节点的位置

主体采用循环结构实现，从头结点指向的下一个节点开始，在链表长度范围内查找，如果找到即可以跳出循环，返回计数值。此外，如果p==NULL, 那说明在链表中没找到拥有该值的节点，返回-1，表示未找到。

1. FindEdge函数：功能：寻找两端的顶点为为head、tail的边

主体利用循环结构实现：从头结点指向的下一个节点开始，只要指针指向的节点不空且两个端点不一样，就一直计数、循环。若最终循环结束以后，指针指向的是NULL，那就说明没有找到，将计数值置为-1，最终返回计数值，表示该节点在链表中的位置。

1. FindBigger函数：功能：寻找第一条权值比value大的边的位置

利用循环结构，只要指针指向的节点不为空且节点权值大于待插入节点，就一直计数、循环、将指针后移。那么循环结束以后，计数值便是链表中第一个比待插入节点权值大的节点的位置，将这一值作为参数传递给Insert函数，即可实现按照权值从小到大的顺序插入新节点。

5.FindNum 函数：功能：返回链表第i个位置的节点的地址

主体采用循环结构实现，首先对非法的无效参数进行处理：如果传递的参数小于-1，则为非法参数，直接返回NULL，若参数是-1，则返回头结点，否则通过循环，在链表长度范围内一直循环第i个位置或最后一个节点，通过暂存指针p不断指向下一个节点，这样一来，如果第i个位置在链表长度范围内，返回的是节点地址，如i已经超出了链表长度，此时p的值恰好是NULL，返回NULL表示该位置不存在有效节点。

6.CreateNode函数：功能：建立新节点

通过动态内存开辟，new新节点并初始化，参数中应当包含用于初始化新节点的item值以及所指向的下一个节点的地址，这样就可以建立新节点并将新节点指针指向下一个节点。

7. Insert函数：功能：在位置i插入边权值为value，两端的顶点为h、t的链表节点

首先在第i个位置插入节点当然需要定位i-1位置的节点，以使其指针指向插入的新节点，接着判断第i-1个节点是否为NULL，是，则表明插入的位置为非法位置，返回0表示插入失败；否则表明插入位置有效，执行后续插入操作：先动态开辟内存存储新建立的节点，并让新建立的节点指向原来在i位置的节点（调用CreateNode函数），接着处理尾结点，如果p已经是尾结点的话，那么新插入的节点成为新的尾结点。最后，原来i-1位置的节点指针指向新建节点，此时，新建节点成为第i位置的节点，原来i位置及后面的节点位置随着这一步整体自然后移1位，最终返回1表示插入成功。

需要注意的是，插入的流程不可随便更改，例如：如果在将原来i-1位置的节点指针指向新建节点之后才处理尾结点，就不能直接通过p->link是否为NULL判断新插入的节点是否应该成为新的尾结点。

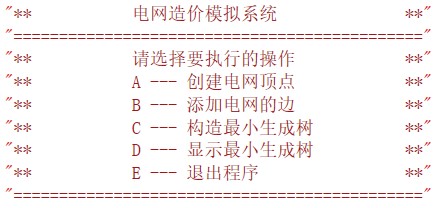
8.Remove函数：功能：删除链表中位置为i的节点

首先调用Find(i-1)定位i-1位置的节点，以便于更改其指针指向，接着如果第i-1位置为NULL或者第i位置为NULL,则表明第i位置本身不存在节点，自然就不需要额外的删除操作，返回0,；否则令i-1位置的指针指向下下个节点，跳过要删除的i位置节点，然后判断尾结点情况，如果要删除的节点是尾节点的话，尾结点指向应前移一位，最终调用delete关键字释放建立节点时为其开辟的内存空间，然后返回1表示删除操作执行成功。同样，删除操作各步流程不可随便更改。

9.Get函数：功能：获取链表i位置的节点值

首先调用Find(i)函数定位到i位置，如果该位置为空，则表明为非法位置，直接返回NULL,否则返回i位置节点的值。

# 3. 程序实现流程

 首先说明的一点是：A-D并不是可以按照任意顺序随意执行的。按照逻辑关系，显示最小生成树之前必须先构造最小生成树，而构造最小生成树之前必须先创建电网顶点再添加顶点之间的边。因此，本程序通过循环与分支结构对程序的执行流程做了限制，例如：用户如果未执行A却输入了C操作，程序会给出相应提示信息。一般来说，用户应当按照从A到D的先后顺序执行功能，当执行完A-D一个循环以后，用户可以再次执行A,重新创建一个图以及顶点，并进行后续操作，或者退出程序。

总体来说，首先由用户输入选项，接着依据用户选项进入不同的分支结构：

A:创建电网顶点：先调用图类函数getVertexPos函数获取顶点在邻接矩阵中对应的位置，再据此通过图类函数insertVertex函数建立电网顶点。

B:添加电网的边：先调用FindEdge函数判断当前图中当前的两个顶点之间是否已经存在边，若存在，则通过set\_cost函数重置边权值即可，若不存在边，则调用Insert函数插入新边。

C:构造最小生成树：直接调用Kruskal函数即可。

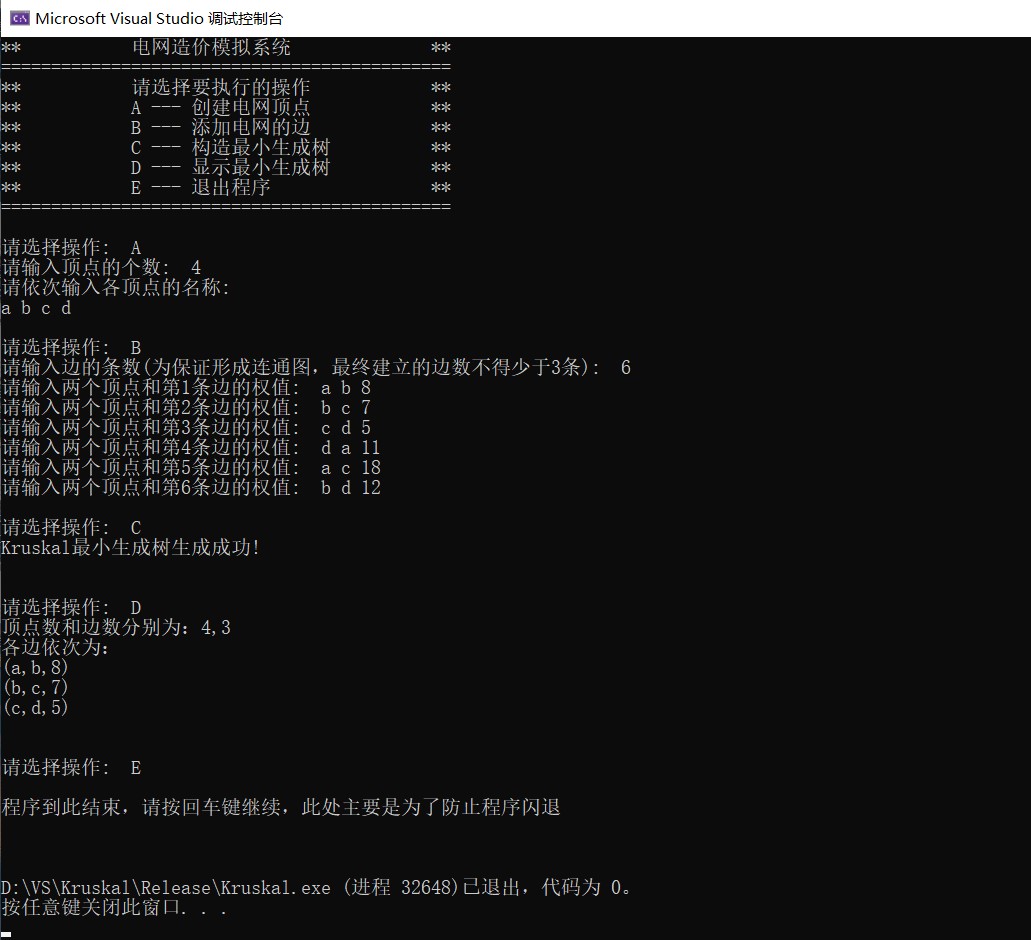
D:显示最小生成树：调用outputGraph函数即可。

# 4.测试

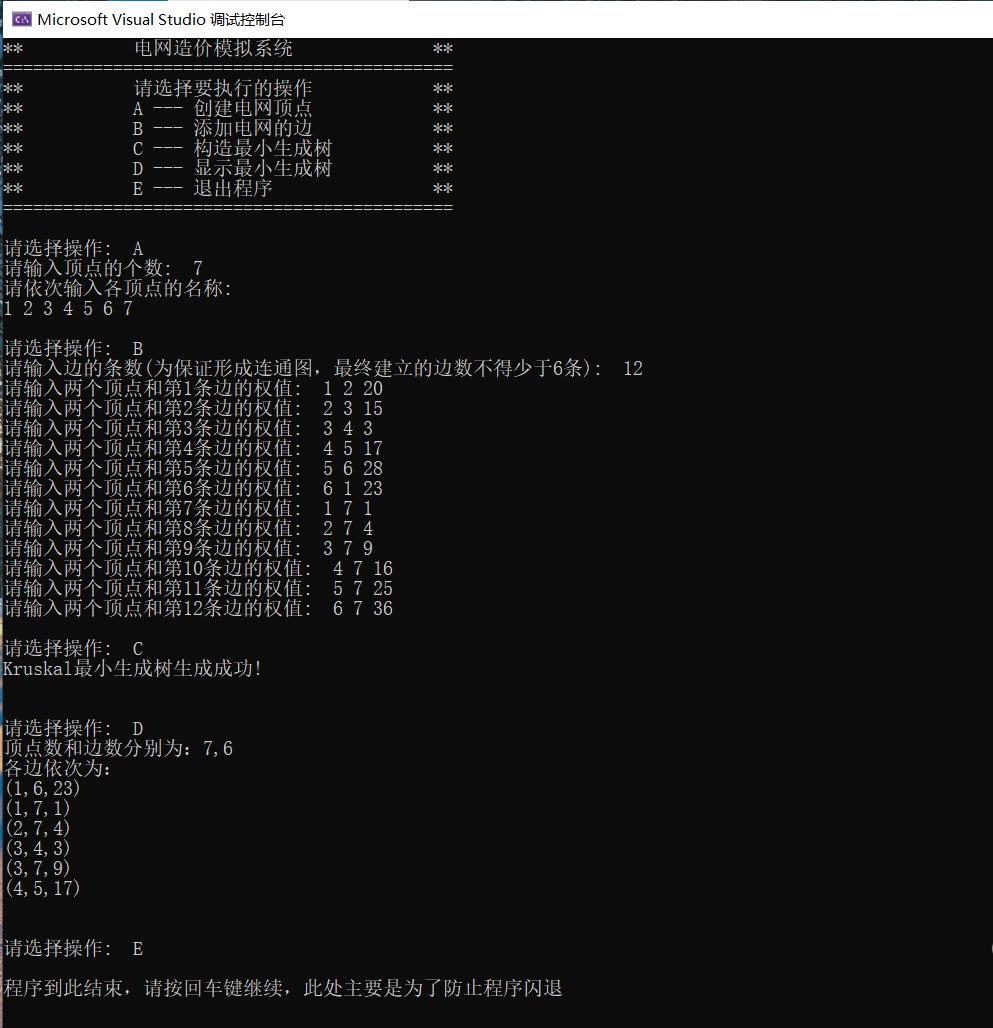
# ①若用户打乱A-D顺序执行则给出相应提示信息：



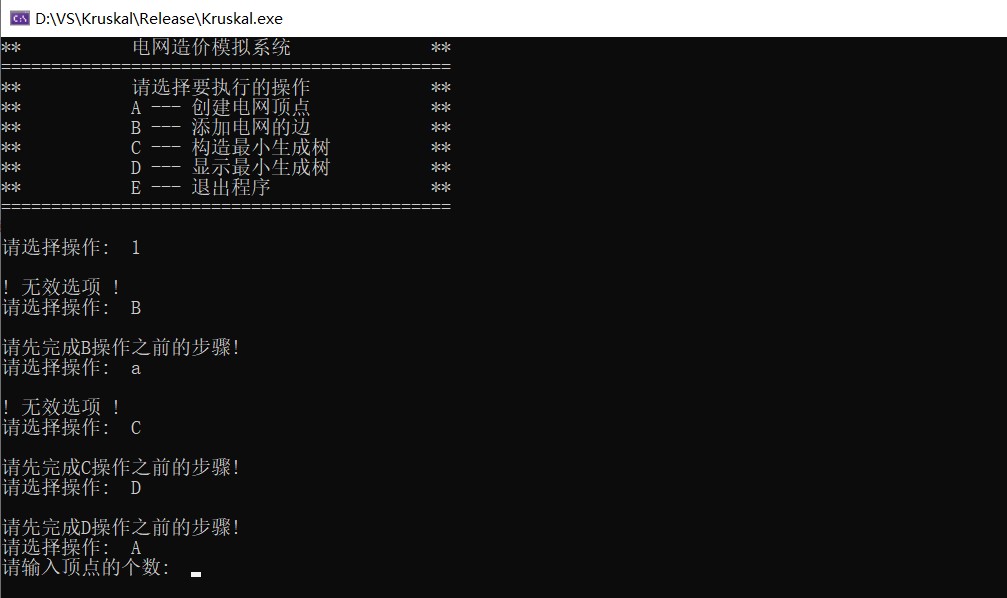
②正常完整功能测试--1



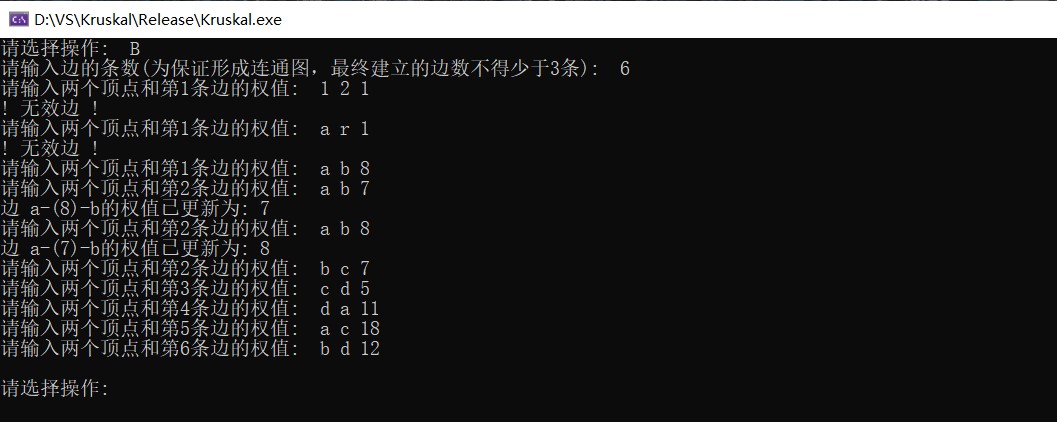
③正常完整功能测试--2：



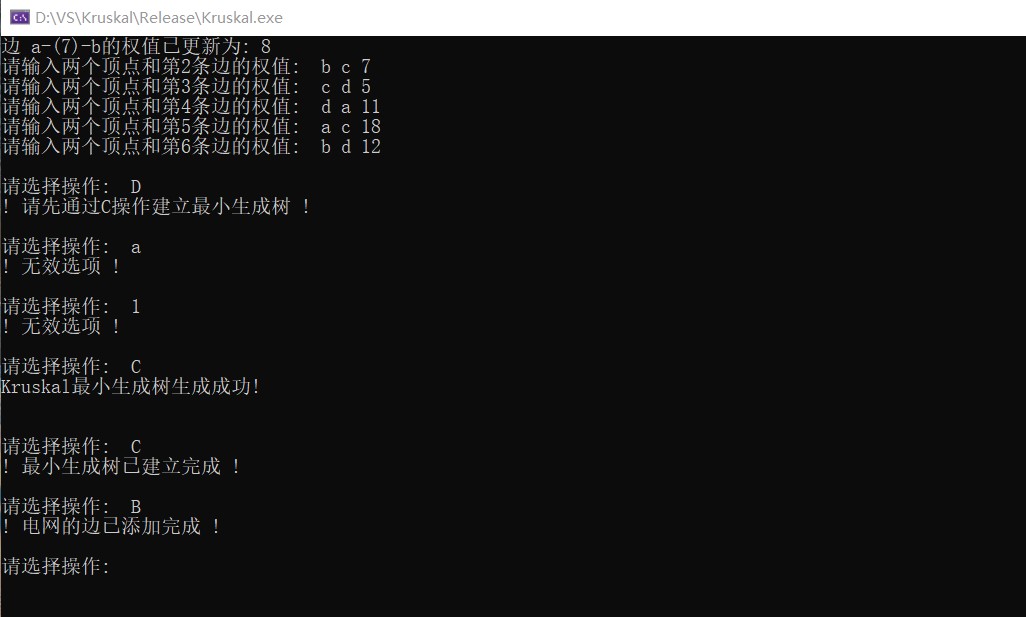
④对无效选项的处理、对未执行A却越级执行后续操作时给出提示信息：



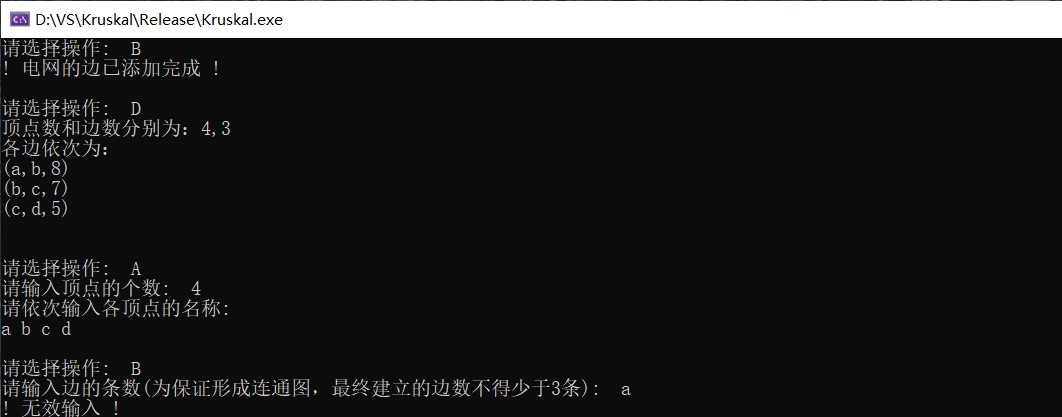
⑤输入边时，输入无效边或者输入重复边（视为对已存在的边权值的更新，不算做新边）



⑥处理无效选项：



⑦B操作输入非法的边的数量：



⑧建立节点时输入无效节点数量：

