第一次上机，链表实现一元多项方程式相加

简单的算法说明:

一元多项方程式主要有系数和指数，如果用顺序表来表示，会浪费很多空间，所以用链表来实现虽然麻烦但是节约空间。每一项对应一个结点。一个方程式对应一条链表。取最长的那条链表为基准，进行插入，得到相加（减）后的新链表。

输入:

```
2
3
7 0 8 1 2 2
4
-7 0 -7 2 5 3 1 4
```

结果:

```
第一方程式:
7X0 8X1 2X2
第二方程式:
-7X0 -7X2 5X3 1X4
结果:
8X1 -5X2 5X3 1X4
请按任意键继续. . .
```

第二次上机，非递归实现树的前、中、后序遍历

简单的算法说明:

利用栈先进后出的性质，使用栈存储当前没有处理完的节点信息，通过循环把结点压入栈中或从栈中弹出并访问，需要花费额外时间来维护栈并为栈留出空间。因为没有采用递归算法，所以可以空间复杂度较小。(构建了搜索二叉树，所以中序遍历就是升序排序)。

前序遍历，访问顺序为：父结点、左子树、右子树。

中序遍历，访问顺序为：左子树、父结点、右子树。

后序遍历，访问顺序为：左子树、右子树、父结点。

时间复杂度：O(n)，空间复杂度：O(logn)

输入:

```
7
8 7 4 5 9 15 12
```

结果:

```
构建搜索二叉树：
先序遍历：
8 7 4 5 9 15 12
中序遍历：
4 5 7 8 9 12 15
后序遍历：
5 4 7 12 15 9 8
请按任意键继续. . .
```

第三次上机，图的遍历以及最小生成树

基于 dfs 的图的遍历:

首先将图中的每一个顶点都标记为未访问，然后选取一个源点 v 压入栈。获得栈的顶点，将其标为已访问，再利用栈压入该点的所有邻接点中第一个未被访问的点，若无未被访问的邻接点，则弹出栈的顶点。若栈为空，则从 v 的搜索过程结束。此时如果图中还有未被访问的顶点（该图有多个连通分量或强连通分量），则再任选一个未被访问过的顶点，从这个顶点开始新的搜索，直到 V 中所有顶点都已被访问过为止。

基于 bfs 的图的遍历:

首先将图中的每一个顶点都标记为未访问，然后选取一个源点 v 压入队列。获得栈的顶点，将其标为已访问，弹出栈的顶点，再利用队列压入该点的所有邻接点。若栈为空，则从 v 的搜索过程结束。此时如果图中还有未被访问的顶点（该图有多个连通分量或强连通分量），则再任选一个未被访问过的顶点，从这个顶点开始新的搜索，直到 V 中所有顶点都已被访问过为止。

基于 kruskal 算法的最小生成树:

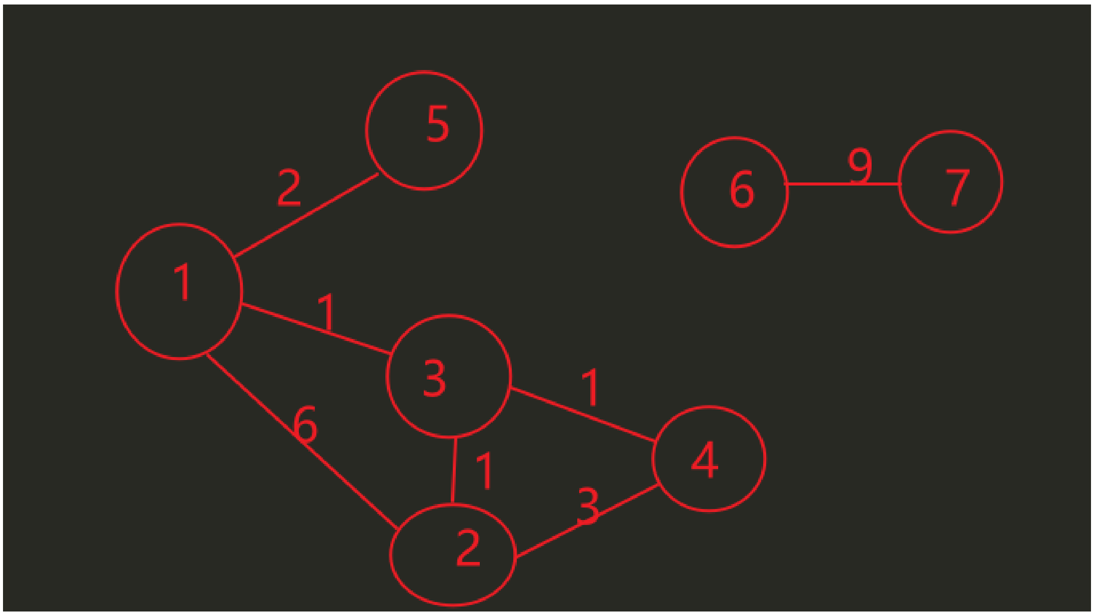
利用贪心算法的思想，每次选取边集合中权重最小的边，然后判断能不能构成回路，若能则舍弃此边，直至边的个数未顶点数-1. 此时如果图中还有未被访问的顶点（该图有多个连通分量或强连通分量），则再任选一个未被访问过的顶点，从这个顶点开始新的搜索，直到 V 中所有顶点都已被访问过为止。

基于 prim 算法的最小生成树:

基于图的顶点的思想，将顶点分为两大集合：已访问集合和未访问集合，随机选取一个源点放入已访问集合，寻找其在未访问集合中的邻接点中且到达源点的路径最小的点 u，放入集合。更新 u 点的未访问的邻接点到达源点的最小路径。直至未找到 u 点。此时如果图中还有未被访问的顶点（该图有多个连通分量或强连通分量），则再任选一个未被访问过的顶点，从这个顶点开始新的搜索，直到 V 中所有顶点都已被访问过为止。

输入:

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



结果:

```
bfs:
1 3 5 2 4
6 7
dfs:
1 3 2 4 5
6 7
基于Kruskal的最小生成树:
根为3的最小生成子树:
(1,3) (2,3) (3,4) (1,5)
根为7的最小生成子树:
(6,7)
14
基于Prim的最小生成树:
根为1的最小生成子树:
1 3 2 4 5
根为6的最小生成子树:
6 7
请按任意键继续. . .
```