设计一个算法，按照深度优先遍历的思想找出从指定结点出发，长度为 M 的所有简单路径。并将此算法用于邻接表类，在邻接表类中提供一个公有的成员函数 find(start, M)

【解】这个问题相当于从指定结点开始深度优先遍历，而且遍历的深度正好为 M。为此，在遍历时需要记住遍历的深度，当深度达到 M 时，就不需要再递归了。此时需要输出这条路径，于是在遍历的过程中还需要记住整条路径。

由于深度优先遍历是用递归实现的，所以 find 函数也有两个。一个是公有的 find 函数，供用户使用。另一个是私有的 find 函数，实现递归的遍历。公有的 find 函数调用私有的 find 函数找出这些路径。

在调用递归的 find 函数前，公有的 find 函数还需要做一些辅助工作。首先，我们要找的是长度为 M 的简单路径，因此路径上不能有相同的结点，于是定义了一个数组 visited 记录结点是否在路径上。其次，当路径长度等于 M 时要输出这条路径，于是定义了一个数组 stack 保存这条路径。每访问一个结点，都要把结点记录在 stack 中。第三，调用 find 函数时要指出从哪一个结点出发，而递归的 find 函数的参数是起点的序号，因此公有的 find 函数需要将这个结点值转换成结点的序号。公有的 find 函数的定义见代码清单 12-26。

代码清单 12-26 找出从 start 出发长度为 M 的简单路径

```
1.  template <class TypeOfVer, class TypeOfEdge>   // 公有的 find 函数
2.  void adjListGraph<TypeOfVer, TypeOfEdge>::find(TypeOfVer start, int m)
    const
3.  {     bool *visited = new bool[Vers];
4.        int *stack =new int[m], top = 0;       // stack：存储路径信息
5.
6.     for (int i=0; i < Vers; ++i) visited[i] = false;
7.
8.        for (i = 0; i < Vers; ++i)              // 查找起点的编号
9.              if (verList[i].ver == start) break;
10.       if (i == Vers) {cout << "起点不存在" << endl; return;}
11.
12.       find(i, m, top, visited, stack);
13. }
14.
15. template <class TypeOfVer, class TypeOfEdge>   // 私有的 find 函数
16. void adjListGraph<TypeOfVer, TypeOfEdge>::
17.     find(int start, int m, int &top, bool visited[], int st[]) const
18. {
19.   edgeNode *p = verList[start].head;
20.
21.   visited[start] = true;
22.   st[top++] = start;
23.   if (top == m + 1) {                    // 路径长度等于 m，递归终止
24.         cout << endl;
```

```
25.         for (int i = 0; i < top; ++i)           // 输出路径
26.             cout << verList[st[i]].ver << '\t';
27.         visited[start] = false;
28.         --top;
29.         return;
30.   }
31.
32.   while (p != NULL) {                       // 继续深度优先遍历
33.         if (!visited[p->end]) find(p->end, m, top, visited, st);
34.         p = p->next;
35.   }
36.   visited[start] = false;
37.   --top;
38. }
```

私有的 find 函数的定义见代码清单 12-26，它有 5 个参数。第一个参数是遍历的起点的序号。第二个参数是要求的路径长度。第三个参数是当前路径中的结点数，当前路径的长度是结点数减 1。第四个参数是一个 bool 型的数组，记录结点是否在路径上。第 5 个参数是一个存储栈元素的整型数组，记录路径上的结点序号。

私有的 find 函数首先将起点放入这条路径，并标记这个结点已被访问。然后判断路径长度是否已经达到 M。如果长度已达到 M，则输出这条路径，并将最后一个结点从路径上删除，返回上一层调用，检查是否还有其他的途径。否则逐个检查起点的后继结点。如果该后继结点没有被访问过，则对该结点调用递归的 find 函数继续寻找。当所有的后继都检查后，表示这条路径处理完毕。将起始结点从这条路径上删除，返回上一层调用。