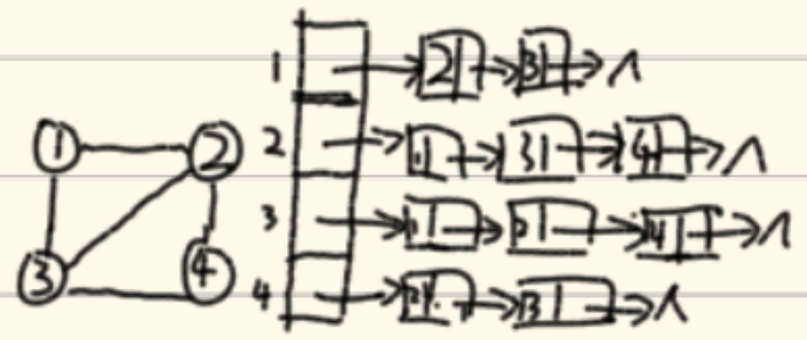
04. 写出从图的邻接表表示转换成邻接矩阵表示的算法。

```
void convert(Graph G, int ans[N][N]){
    for(int i=0; i<N; i++){
        P = G.V[i].first;
        while(P){
            ans[i][P->adjvex] = 1;
            P = P->next;
        }
    }
}
```



	1	2	3	4
1	INF	1	1	INF
2	1	INF	1	1
3	1	1	INF	1
4	INF	1	1	INF

06. 【2021 统考真题】已知无向连通图 G 由顶点集 V 和边集 E 组成，$|E|>0$，当 G 中度为奇数的顶点个数为不大于 2 的偶数时，G 存在包含所有边且长度为$|E|$的路径（称为 EL 路径）。设图 G 采用邻接矩阵存储，类型定义如下：

即：度为奇数的顶点个数只能是0或2

```
typedef struct{                          //图的定义
    int  numVertices,numEdges;           //图中实际的顶点数和边数
    char VerticesList[MAXV];             //顶点表。MAXV 为已定义常量
    int Edge[MAXV][MAXV];                //邻接矩阵
}MGraph;
```

请设计算法 int IsExistEL(MGraph G)，判断 G 是否存在 EL 路径，若存在，则返回 1，否则返回 0。要求：

1）给出算法的基本设计思想。

2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

3）说明你所设计算法的时间复杂度和空间复杂度。

```
int IsExistEL(MGraph G){
    int cnt=0, degree;
    for(int i=0; i<G.numVertices; i++){
        degree=0;
        for(int j=0; j<G.numVertices; j++){
            degree += G.Edge[i][j];
        if(degree % 2 != 0) cnt++;
    }
    return (cnt==0 || cnt==1)? 1:0;
}
```

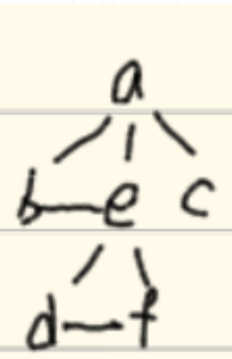
02. 试设计一个算法，判断一个无向图 G 是否为一棵树。若是一棵树，则算法返回 true，否则返回 false。

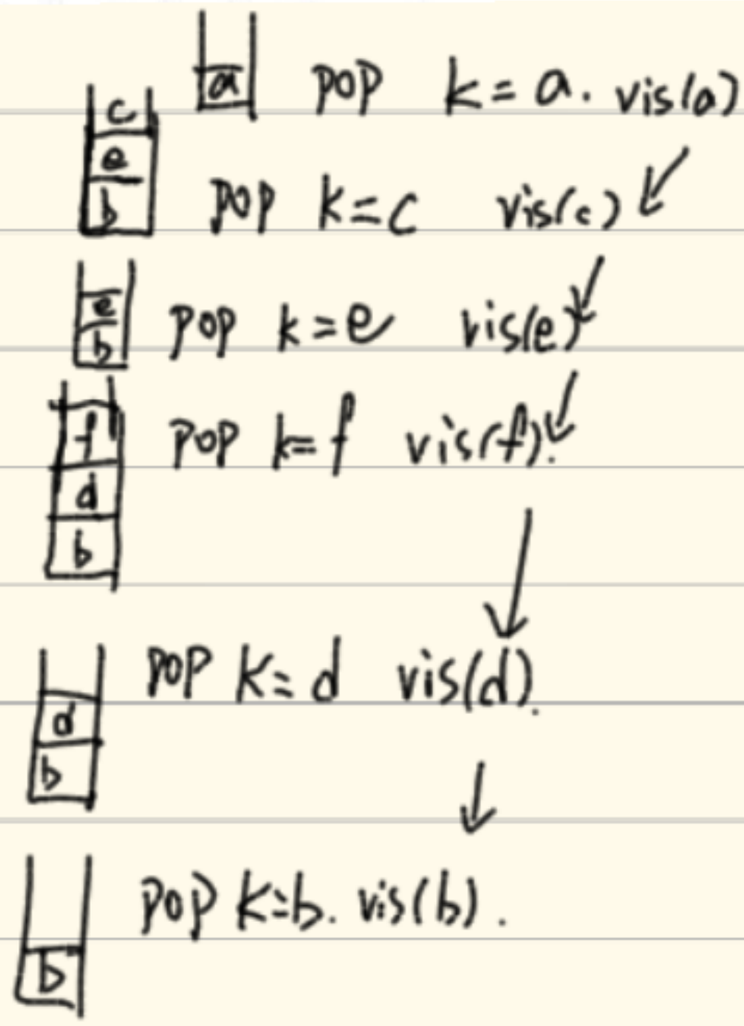
G是一棵树的条件是：G是有n-1条边的连通图.

```
bool isTree(Graph G){
    for(int i=0; i<G.vexnum; i++)
        visited[i] = false;
    int vnum=0, enum=0;
    DFS(G, 1, vnum, enum, visited);
    if(vnum == G.vexnum && enum == (G.vexnum-1))
        return true;
    return false;
}
DFS(Graph G, int v, int &vnum, int &enum, int visited[]){
    visited[v] = true;
    vnum++;
    int w = firstNeighbor(G, v);
    while(w != -1){
        if(!visited[w]){
            enum++;
            DFS(G, w, vnum, enum, visited);
        }
        w = nextNeighbor(G, v, w);
    }
```

03. 写出图的深度优先搜索 DFS 算法的非递归算法（图采用邻接表形式）。

```
void dfs(Graph G, int v){
    Stack s;
    InitStack(s);
    int w;
    for(int i=0; i<G.vexnum; i++)
        visited[i] = false;
    Push(S,v);
    visited[v] = true;
    while(!empty(S)){
        Pop(S,k);
        visit(k);
        for(w=firstNeighbor(G,k); w>=0; w=nextNeighbor(G,k,w)){
            if(!visited[w]){
                Push(S,w);
                visited[w] = true;
            }
        }
    } // while
}
```





a c e f d b

04. 分别采用基于深度优先遍历和广度优先遍历算法判别以邻接表方式存储的有向图中是否存在由顶点 v_i 到顶点 v_j 的路径（$i \neq j$）。注意，算法中涉及的图的基本操作必须在此存储结构上实现。

基于深度优先：

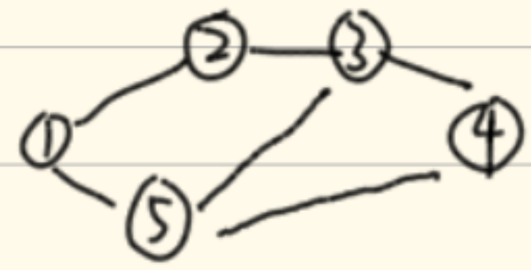
```
int visited[maxsize];
void DFS(Graph G, int i, int j, bool& flag){
    if(i==j){      //若从i走到了j
        flag = true;
        return;
    }
    visited[i] = 1;
    for(int p=firstNeighbor(G,i); p>=0; p=nextNeighbor(G,i,p))
        if(!visited[p] && !flag)
            DFS(G, p, j, flag);
}
```

基于广度优先：

```
bool BFS(Graph G, int i, int j){
    Queue Q; InitQueue(Q); EnQueue(Q,i);
    while(!empty(Q)){
        DeQueue(Q, p);
        visited[p] = 1;
        if(p==j) return true;
        for(int k=firstNeighbor(G,p); k>=0; k=nextNeighbor(G,p,k)){
            if(k==j) return true;
            if(!visited[k]){
                EnQueue(Q,k);
                visited[k] = 1;
            }
        } // for.
    } // while
}
```

05. 假设图用邻接表表示，设计一个算法，输出从顶点 V_i 到顶点 V_j 的所有简单路径。

```
Void findpath(Graph G, int i, int j, int path[], int len, int visited[]){
    // len为路径长度，初始为-1.
    int a;
    len++;
    path[len] = i;
    ArcNode *p;
    visited[i] = 1;
    if(i == j)
        Print(path);
    p = G.adjlist[i].first;
    while(p){
        a = p->adjvex;
        if(!visited[a])
            findPath(G, a, j, path, len, visited);
        p = p->nextarc;
    } // while.
    visited[i] = 0;      // 使结点可用，继续找别的路径.
}.
```



1-2-3-4

1-5-3-4

1-5-4

1-2-3-5-4

06. 试说明利用 DFS 如何实现有向无环图拓扑排序。

分析：设 u是 v 的祖先，则在u的 DFS过程中，必然会对 v调用DFS，那么u的递归深度一定大于 v的递归深度，故对递归深度进行标记，按深度从大到小输出即可，若 u,v 无关系，那么先后随意。

```
bool  visited [maxSize];
void  getDepth(Graph G){
    for(int i=0 ; i<G.vexnum : i++)
      visited[i] = false;
    for(int i=0 ; i<G.vexnum; i++)
      if (!visited[i])  DFS( G, i, 0);
}.
void  DFS (Graph G ,int v, int& time){
      visited[v] = true;  time++;
      for(int i = firstNeighbor(G,v); i>=0; i = nextNeighbor(G,v,i)){
        if(!visited[i])  DFS(G, i, time);
      }.
      times[v] = time;
}.
```