|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 |  | 学号 |  | 实验成绩 |  |
| 专业班级 |  | 实验时间 | 2025-06-14 | 实验地点 | 计算中心 |

实验五：图实验

1实验目的

1. 掌握图的基本概念。
2. 掌握图的存储结构的设计与实现，基本运算的实现。
3. 熟练掌握图的两种遍历算法、遍历生成树及遍历算法的应用。
4. 根据具体给定的需求，合理设计并实现相关结构和算法。

2 实验要求

1. 对图进行合理封装；
2. 在相应的存储结构上，实现基本运算；
3. 设计算法实现相应的实验任务；
4. 程序运行、测试正确；
5. 实验程序有较好可读性，各运算和变量的命名直观易懂，符合软件工程要求；
6. 程序有适当的注释，程序的书写要采用缩进格式。

3 实验任务

图的封装：

* 邻接表：

typedef char PointType;

typedef struct EdgeNode {

struct EdgeNode \*next;

int weight;

int adjvex;

} EdgeNode;

typedef struct PointNode {

int inDegree;//入度，用于拓扑排序

struct EdgeNode \*firstEdge;

PointType data;

} PointNode,AdjList[MAX];

typedef struct {

AdjList adjList;

int numEdges,numNodes;

} GraphAd;

* 邻接矩阵

typedef char VertexType; /\* 顶点 \*/

typedef int EdgeType; /\* 边上的权值 \*/

typedef struct {

VertexType vexs[MAXVEX];

EdgeType arc[MAXVEX][MAXVEX];/\* 邻接矩阵 \*/

int numNodes, numEdges; /\* 图中顶点数和边数 \*/

} MGraph;

编写算法求解下列问题：

1. 邻接表存储的图G，求给定图中的边（或弧）的数目。 实验测试数据基本要求：

第一组数据： udg8.grp

第二组数据： udg115.grp

第三组数据： dg6.grp

第四组数据： f14.grp

5 算法设计与实现描述

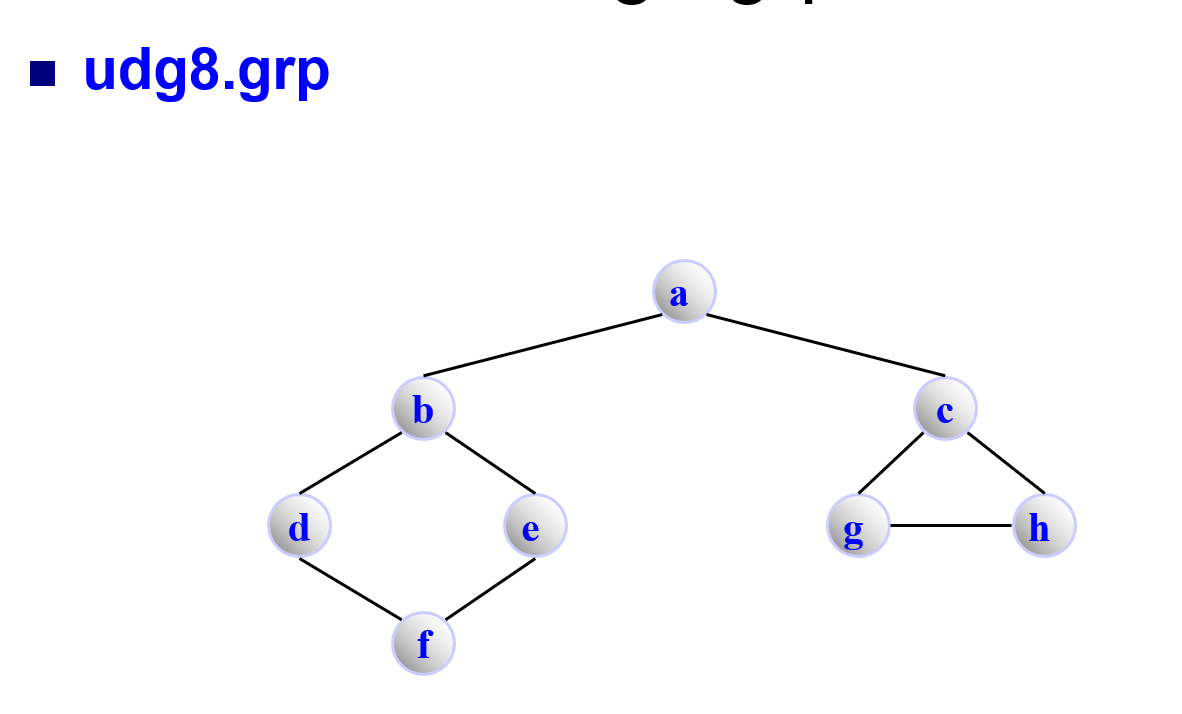
（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

思路：邻接表表示的图，直接遍历每个结点的链表，计数即可获得边的数量，注意：无向图看成两个点之间双向箭头，即真实的边数需要在count除以2，代码如下：

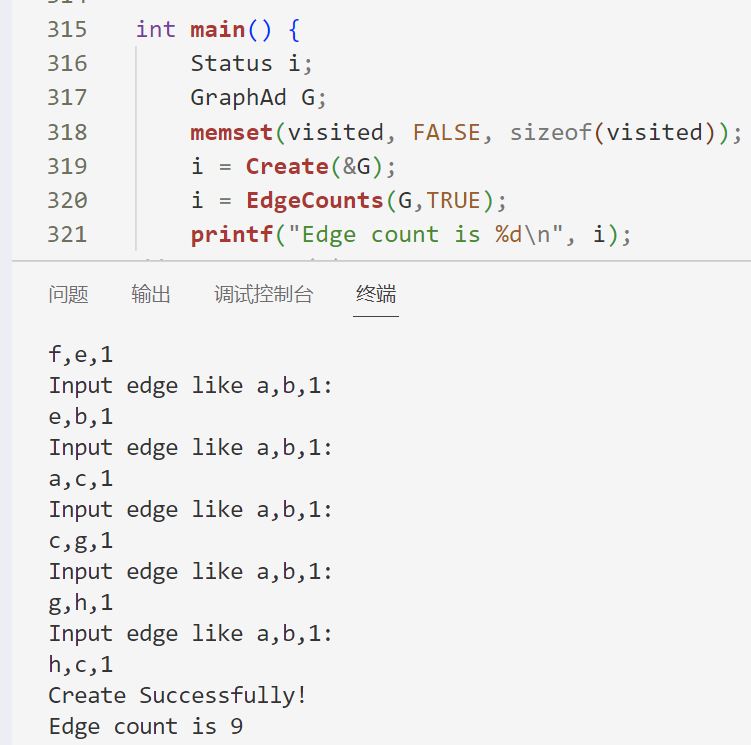
|  |
| --- |
| int EdgeCounts(GraphAd G,Status isWuxiang) {  int count = 0;  for(int i = 0;i < G.numNodes;i++) {  EdgeNode \*ptr = G.adjList[i].firstEdge;  while(ptr) {  count++;  ptr = ptr->next;  }  }  if(isWuxiang == TRUE) {  return count / 2;  } else {  return count;  }  } |

6运行结果截图及说明

第一组数据：udg8.grp

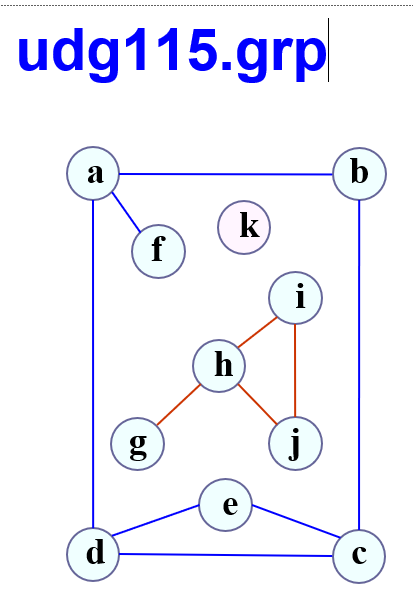


由图可知，共有9条边，运行结果如下：

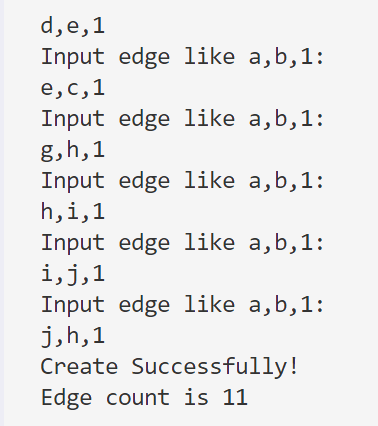


符合预期

同理测试第二组数据udg115

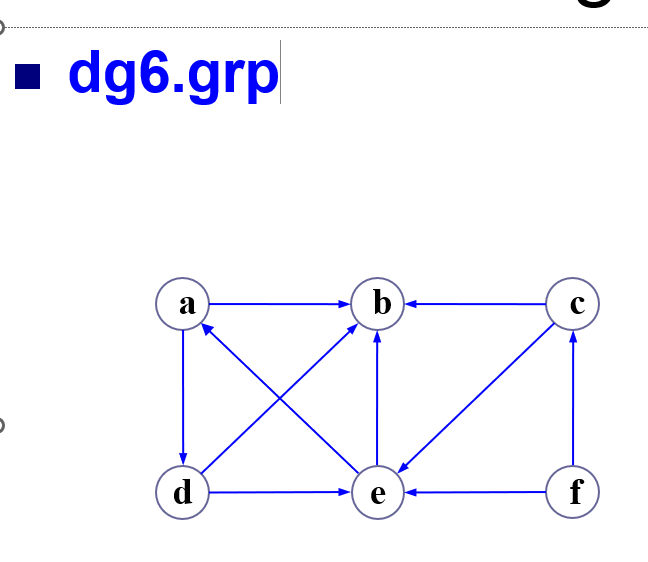


由图可知共有11条边，运行结果如下：

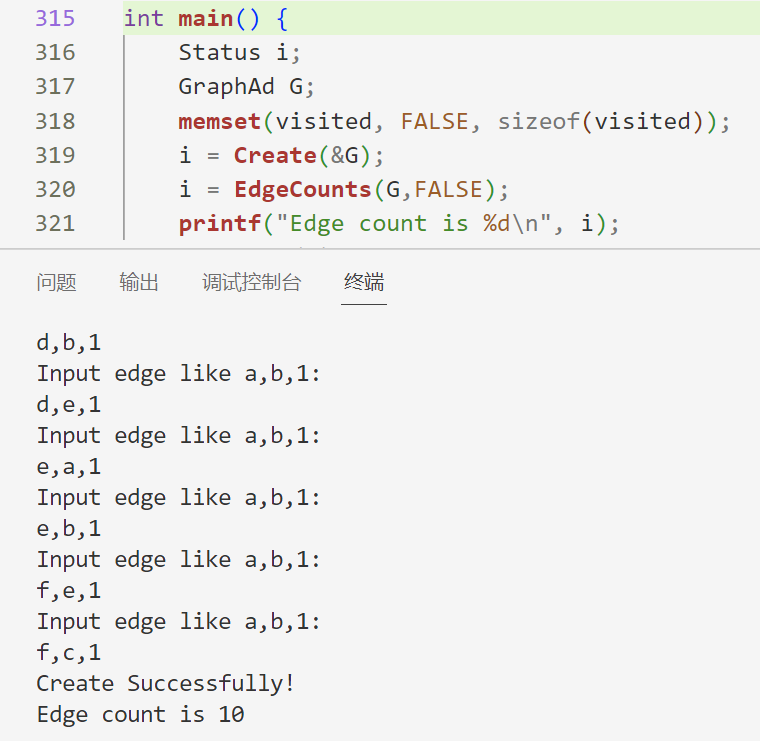


符合预期

第三组数据为dg6，为有向图，需要修改参数

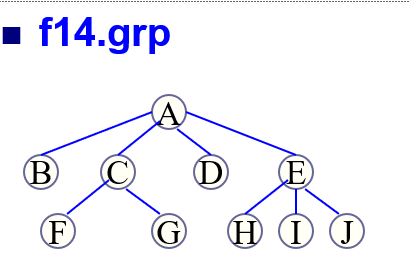


由图可知共有10条边，运行代码结果如下：

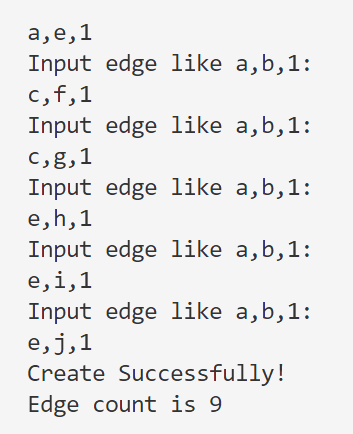


符合预期

第四组数据为f14



由图可知有9条边，运行代码结果如下：



符合预期

1. 邻接矩阵存储的图G，设计算法判断顶点v到 w之间是否存在路径？若存在，则返回TRUE，否则返回FALSE。

第一组数据： udg8.grp

第二组数据： udg115.grp

第三组数据： dg6.grp

第四组数据： f14.grp

5 算法设计与实现描述

（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

思路：借助DFS，从vi开始向下访问，最后返回visited[vj]的值（vj是否被访问到）即可，访问到即代表在一条路径上，可以到达。-

|  |
| --- |
| //设计算法以判断顶点vi到vj之间是否存在路径(vi可沿着到达vj就算有路径，中间不得中断)  void DFS3(MGraph G,VertexType v) {  visited[v] = TRUE;  for (int j = 0; j < G.numNodes; j++) {  if (G.arc[v][j] != 0 && G.arc[v][j] != INFINITY && !visited[j]) {  DFS3(G, j);  }  }  }  Status isHaveEdge(MGraph G,VertexType vi,VertexType vj) {  // 将顶点vi和vj转换为索引 顶点从1开始 索引从0开始  vi -= 1;  vj -= 1;  for(int i = 0;i < G.numNodes;i++) {  visited[i] = FALSE;  }  DFS3(G,vi);  return visited[vj];  } |

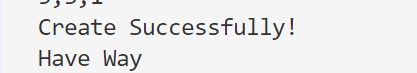
6运行结果截图及说明

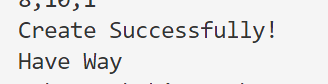
图略（上实验已给），所有均测试点c到f是否能到达，跟据图推理，应该分别为能、能、不能、能

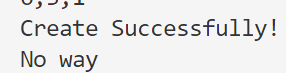
主函数为：

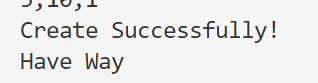
|  |
| --- |
| int main() {  int a = 'a' - 1;  int c = 'c'- a;  int f = 'f'- a;  MGraph G;  Status i;  i = Create(&G);  if(isHaveEdge(G,c,f) == TRUE) {  printf("Have Way");  } else {  printf("No way");  }  } |

运行代码测试如下：









符合预期

1. 邻接表存储的图G，设计算法分别实现函数firstAdj（G，v）和nextAdj(G,v,w)。

第一组数据： udg8.grp

第二组数据： udg115.grp

第三组数据： dg6.grp

第四组数据： f14.grp

5 算法设计与实现描述

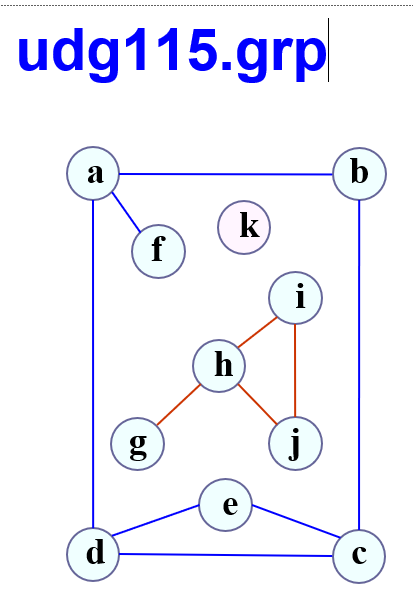
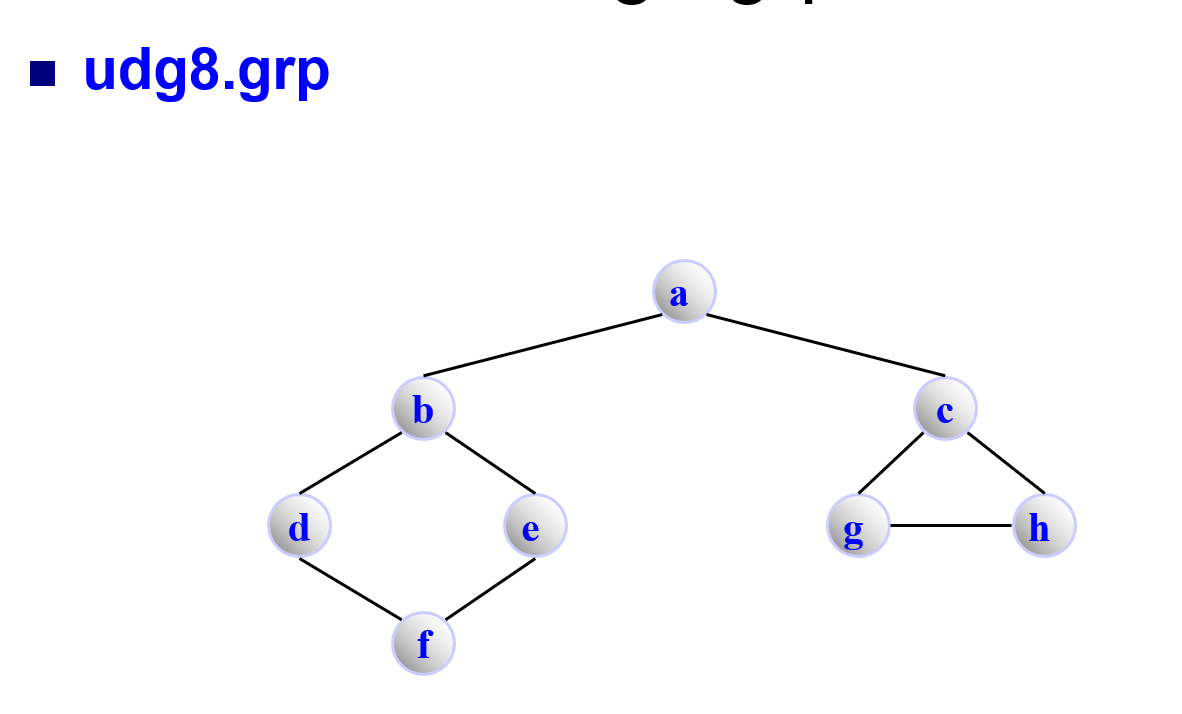
（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

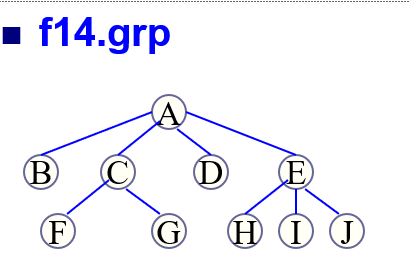
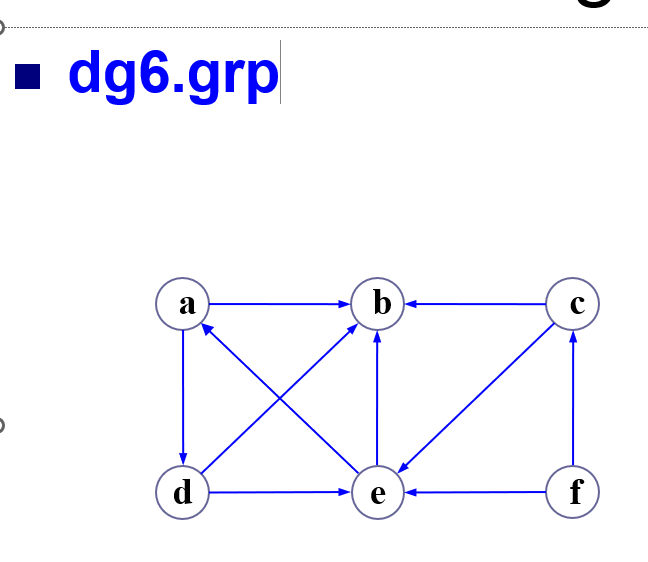
思路：firstadj返回第一个与v连接的点，直接查该点的链表，第一个接的点就是其firstAdj；nextAdj是通过查v的链表，遍历，直到找到w点，返回w的下一个点

|  |
| --- |
| //设计算法实现函数firstadj(G, v)（返回第一个与v连接的点）  PointType firstAdj(GraphAd G,PointType v) {  int vIndex = v - 'a'; // 规整索引  if (vIndex < 0 || vIndex >= G.numNodes) return -1;  EdgeNode \*E = G.adjList[vIndex].firstEdge;  if (E) {  char r = E->adjvex + 'a'; // 转换索引为字符  return r;  } else {  return '\0'; // 没有邻接点  }  }  //设计算法实现函数nextadj(G, v, w)（返回与v相邻并在w之后的下一个顶点）。  PointType nextAdj(GraphAd G,PointType v,PointType w) {  int vIndex = v - 'a'; // 规整索引  int wIndex = w - 'a'; // 规整索引  if (vIndex < 0 || vIndex >= G.numNodes) return -1;  EdgeNode \*E = G.adjList[vIndex].firstEdge;  while (E && E->adjvex != wIndex) {  E = E->next;  }  if (E && E->next) {  char r = E->next->adjvex + 'a'; // 转换索引为字符  return r;  } else {  return '\0'; // 没有后继邻接点  }  } |

6运行结果截图及说明

图略（上实验已给），所有均测试firstAdj(G, 'd')和nextAdj(G, 'd', firstAdj(G, 'd'))，跟据图推理，应该分别为

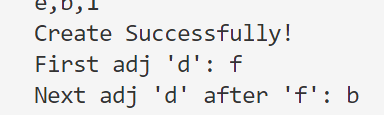


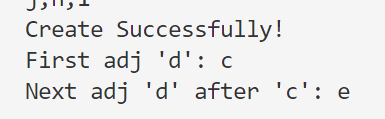


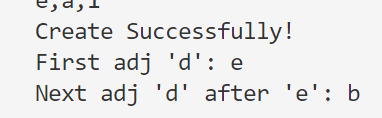
主函数为：

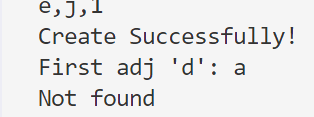
|  |
| --- |
| int main() {  Status i;  GraphAd G;  memset(visited, FALSE, sizeof(visited));  i = Create(&G);  char first = firstAdj(G, 'd');  if (first != '\0') {  printf("First adj 'd': %c\n", first);  } else {  printf("Not found\n");  return 0;  }  char next = nextAdj(G, 'd', first);  if (next != '\0') {  printf("Next adj 'd' after '%c': %c\n",first, next);  } else {  printf("Not found\n");  return 0;  }  } |

测试结果：









符合预期

1. 邻接表存储的图G是有向图，设计算法以判断G是否是一棵以v0为根的有向树，若是返回TRUE，否则返回FALSE。

第一组数据： dg6.grp

第二组数据： dTree11.grp

5 算法设计与实现描述

（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

思路：先用BFS判断环，树不能有环，满足不是环的条件后，再检查是否为连通图，树必须是联通的，都满足则为树

无向图为树即满足没有环路的连通图；先判断是否存在环：DFSisLoop。是则直接return false，无环则继续判断连通性，不连通则return false，连通则return true

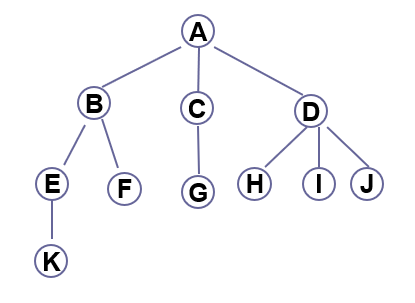
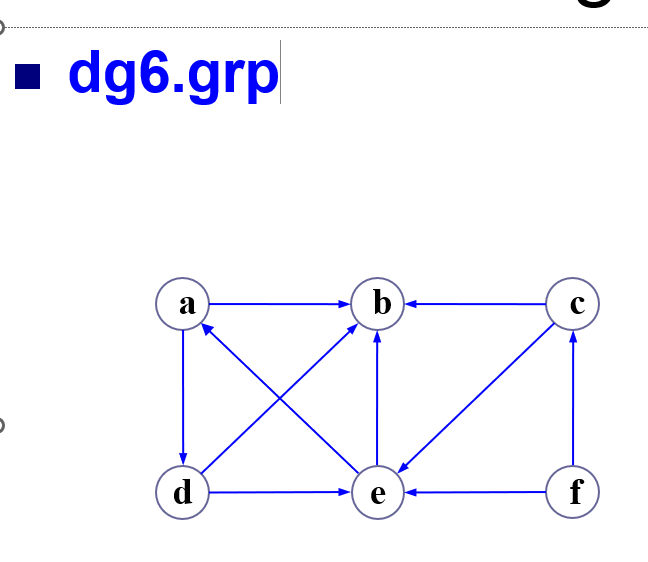
检查环 思路：见DFSisLoop函数，借助DFS和递归，递归调用时传入父结点，若此结点(j)已访问过(visited==true)且不是父结点(j != parent(上一次的i))说明存在环，直接return true

检查连通 思路：检查环完成后，应当无环图才可以执行下面的代码，通过判断是否所有节点都被遍历可判断是否连通

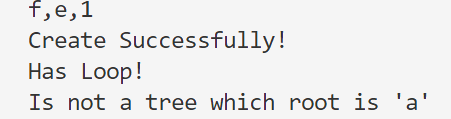
|  |
| --- |
| Status BFSisLoop(GraphAd G, int start) {  Queue Q;  Init(&Q);  int parent[MAX]; // 用于记录每个节点的父节点  for (int i = 0; i < G.numNodes; i++) {  parent[i] = -1;  }  visited[start] = TRUE;  Add(&Q, start);  while (Length(&Q)) {  int u;  DeAdd(&Q, &u);  for (EdgeNode \*p = G.adjList[u].firstEdge; p; p = p->next) {  int v = p->adjvex;  if (!visited[v]) {  visited[v] = TRUE;  parent[v] = u;  Add(&Q, v);  } else if (v != parent[u]) {  return FALSE; // 非父节点已访问，说明有环  }  }  }  return TRUE;  }  // 设计算法以判断G是否是一棵以v0为根的有向树，  Status isRootTreeBFS(GraphAd G,PointType root) {  for (int i = 0; i < G.numNodes; i++) {  visited[i] = FALSE;  }  // 规整索引 转字符为索引  int rootIndex = root - 'a';  if (!BFSisLoop(G,rootIndex)) {  printf("Has Loop!\n");  return FALSE;  }  // 检查连通性  for (int i = 0; i < G.numNodes; i++) {  if (!visited[i]) return FALSE;  }  return TRUE;  } |

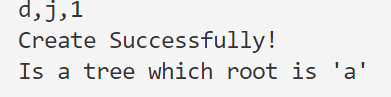
6运行结果截图及说明

所有均测试点a，跟据图推理，应该分别为：不是、是



运行结果如下：





符合预期

1. 设连通图G用邻接表A表示，设计算法以产生dfs（1）的dfs生成树，并存储到邻接矩阵B[][]中。

第一组数据：udg10.grp

第二组数据：udTree11.grp

第三组数据：Udn6.grp

第四组数据：Udn8.grp

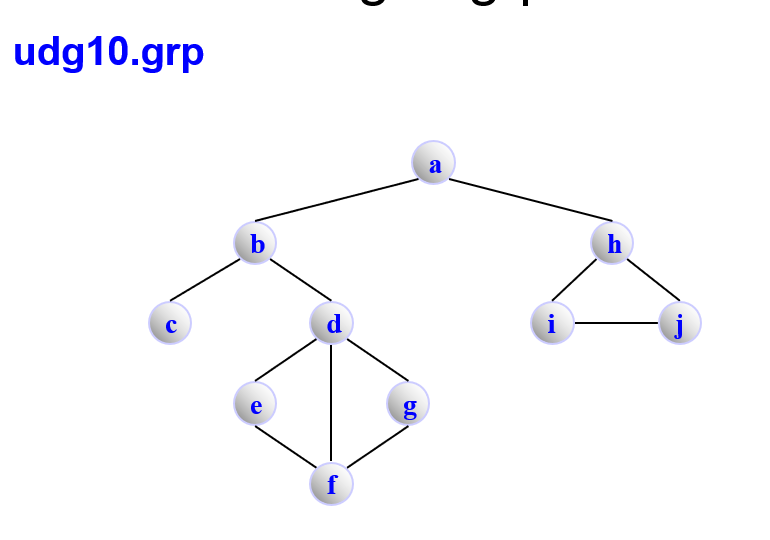
5 算法设计与实现描述

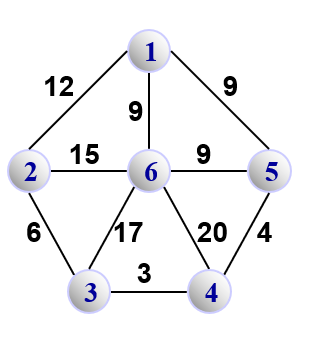
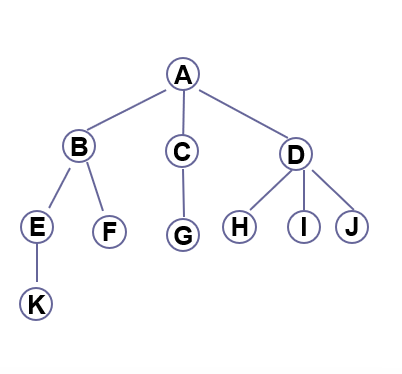
思路：利用DFS，在搜索过程将权值加到邻接矩阵中，搜索结束时邻接矩阵构建完毕

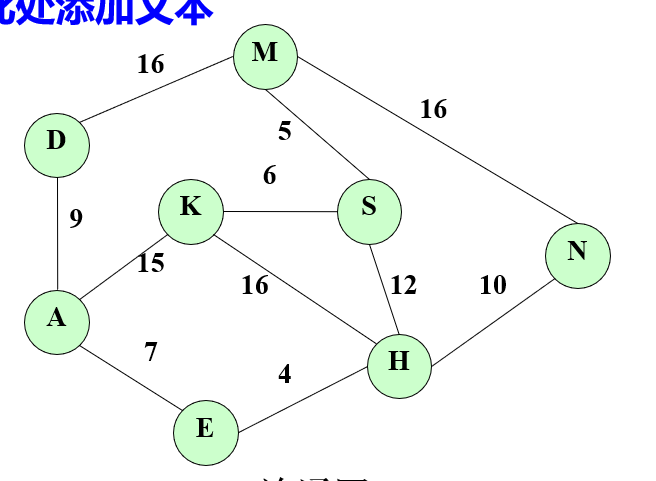
|  |
| --- |
| void DFS5(GraphAd G, int arc[][MAX], int u) {  visited[u] = TRUE;  EdgeNode \*E = G.adjList[u].firstEdge;  while (E) {  int v = E->adjvex;  if (!visited[v]) {  arc[u][v] = E->weight;  // 无向图，矩阵对称,如果不是无向图需要注释掉  arc[v][u] = E->weight;  DFS5(G, arc, v);  }  E = E->next;  }  }  // 设连通图G用邻接表A表示，设计算法以产生dfs（1）的dfs生成树，并存储到邻接矩阵B[][]中。  void Question5(GraphAd G,int arc[][MAX]) {  // 初始化邻接矩阵  for (int i = 0; i < G.numNodes; i++) {  visited[i] = FALSE;  for (int j = 0; j < G.numNodes; j++) {  arc[i][j] = 0;  }  }  int rootIndex = 'a' - 'a';// 转字符为索引  DFS5(G, arc, rootIndex);  } |

6运行结果截图及说明

所有均测试点均从a开始DFS，下面为生成树（结果不唯一）







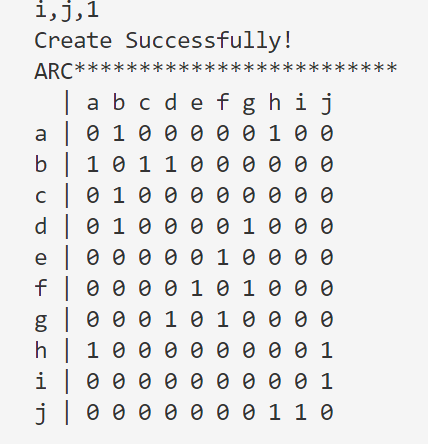
（字母为结点）的主函数如下：

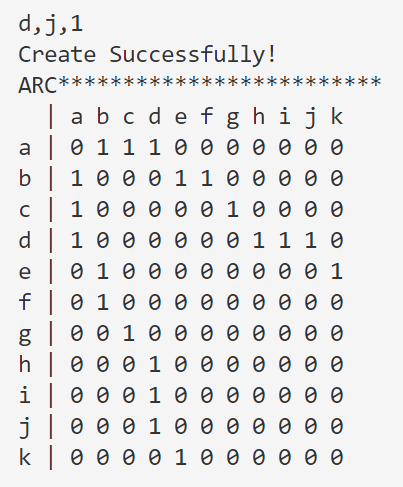
|  |
| --- |
| int main() {  Status i;  GraphAd G;  memset(visited, FALSE, sizeof(visited));  i = Create(&G);  int arc[MAX][MAX];  Question5(G,arc);  printf("ARC\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf(" | ");  for(int i = 0; i < G.numNodes; i++) {  printf("%c ", 'a' + i);  }  printf("\n");  for(int i = 0; i < G.numNodes; i++) {  printf("%c | ",'a'+i);  for (int j = 0; j < G.numNodes; j++) {  printf("%d ", arc[i][j]);  }  printf("\n");  }  } |

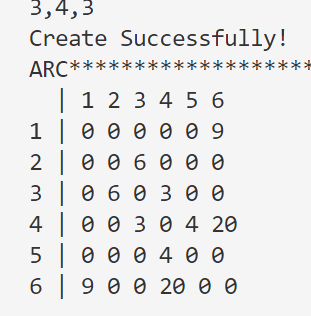
有一组结点为数字，做了打印格式的微调，略

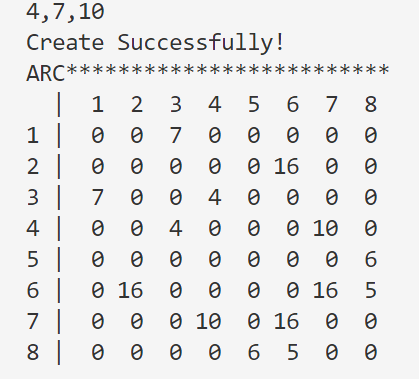
最后一组字母为什么不按顺序？强制改成数字显示了

运行结果如下：









将邻接矩阵与上面的生成树映射对照，符合预期

7总结、心得和建议

图不仅是实际开发中常用到的数据结构，也是实际生活经常遇到的场景，例如现实生活中的地图、最短路径等，软件开发领域也需要地图导航算法，还有一些社交软件的关系图等。图综合了前面所有的数据结构，一些操作需要栈、队列的配合，同时，树也是一种特殊的图。