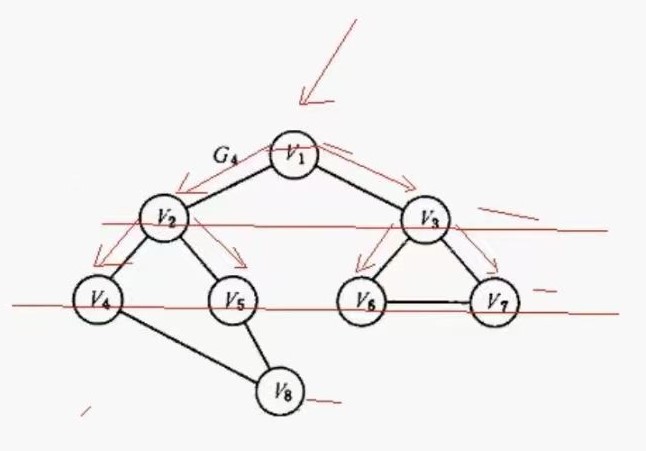
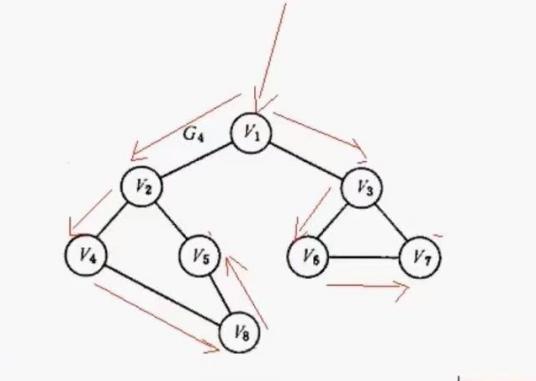
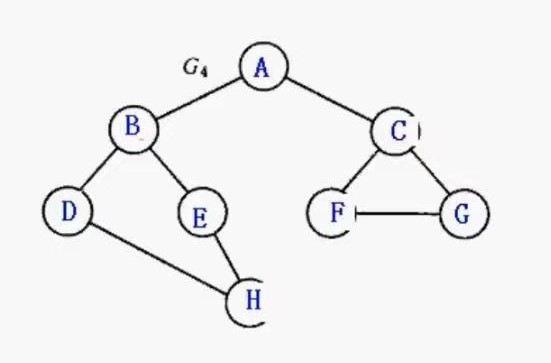
**图的遍历**

两种遍历方式：

深度优先搜索 广度优先搜索（一层层来）





（本节所用图）

**//深度优先**

void DFS(GraphLnk&gl, int v, bool visited[]) {

cout << gl.NodeTable[v].data<<"-->";

visited[v] = true;

int w = GetFirstNeighbor(gl, gl.NodeTable[v].data);

while (w != -1) {

if (!visited[w]) {

DFS(gl, w, visited);

}

//一部分的深度结束后，向广度扩散

w = GetNextNeighbor(gl, gl.NodeTable[v].data, gl.NodeTable[w].data);

}

}

void DFS(GraphLnk&gl, T vertex) {

int n = gl.NumVertices;

bool \*visited = (bool\*)malloc(sizeof(bool)\*n); //这一步是防止顶点的重复遍历，开辟一个布尔类型的数组

assert(visited != nullptr);

for (int i = 0; i < n; i++) {

visited[i] = false;

}

int v = GetVertexPos(gl, vertex);

DFS(gl, v, visited); //visited就是一个元素为布尔值的数组

free(visited);

cout << "Nul";

}

**//广度优先**

//借助队列

void BFS(GraphLnk &gl,T vertex) {

int n = gl.NumVertices;

bool \*visited = (bool\*)malloc(sizeof(bool)\*n); //这一步是防止顶点的重复遍历，开辟一个布尔类型的数组

assert(visited != nullptr);

for (int i = 0; i < n; i++) {

visited[i] = false;

}

int v = GetVertexPos(gl, vertex);

cout << vertex << "-->";

visited[v] = true;

int x;

LinkQueue Q;

InitQueue(Q);

EnQueue(Q, v);

int w;

while (!Empty(Q)) {

GetHead(Q, x);

DeQueue(Q);

w = GetFirstNeighbor(gl, gl.NodeTable[x].data);

while (w != -1) {

if (!visited[w]) {

cout << gl.NodeTable[w].data << "-->";

visited[w] = true;

EnQueue(Q, w);

}

w = GetNextNeighbor(gl, gl.NodeTable[v].data, gl.NodeTable[w].data);

}

}

free(visited);

}