图

图的数据结构(邻接矩阵)

```
typedef struct GNode *PtrToGNode;
struct GNode{
    int Nv;   /* 顶点数 */
    int Ne;   /* 边数   */
    VertexType vex[Maxsize]; /* 顶点表*/
    EdgeType G[MaxVertexNum][MaxVertexNum]; /* 邻接矩阵，边表*/
};
typedef PtrToGNode MGraph; /* 以邻接矩阵存储的图类型 */
```

深度优先遍历（递归）

```
void DFS(MGraph graph, int v){
    visited[v] = true;
    visit(v);
    for(int i = 0; i < graph->Nv; ++i){
        if(graph->G[v][i] == 1 && !visited[vex[i]])
            DFS(graph, vex[i]);
    }
}
```

深度优先遍历（迭代）

```
void DFS(MGraph graph, int v){
    stack<int> st;
    visit(v);
    visited[v];
    st.push(v);
    while(!st.empty()){
        int data, i;
        data = st.top();
        for(i = 0; i < graph->Nv; ++i){
            if(graph->G[data][i] == 1 && visited[vex[i]] == 1){
                visit(vex[i]);
                visited[vex[i]] = true;
                st.push(v);
                break;
            }
        }
        if(i == graph->Nv) st.pop();
    }
}
```

图的广度优先遍历(迭代)

```
void BFS(MGraph graph, int v){
    queue<int> que;
    int vertex;
    visit(v);
    visited[v] = true;
    que.push(v);
    while(!que.empty()){
        vertex = que.front();
        que.pop();
        for(int i = 0; i < graph->Nv; ++i){
            if(graph->G[v][i] == 1 && !visited[vex[i]]){
                visit(vex[i]);
                visited[vex[i]] = true;
                que.push(vex[i]);
            }
        }
    }
}
```

图的数据结构描述(邻接表)

```
typedef struct ArcNode{      // 边表结点
    int adjvex;              // 该弧所指向的顶点的位置
    struct ArcNode *next;    // 指向下一条弧的指针
}ArcNode;

typedef struct VNode{        // 顶点表结点
    int data;                // 顶点信息
    ArcNode* first;          // 指向第一条依附该顶点的弧的指针
}VNode, AdjList[MaxSize];

typedef struct{
    AdjList vertices;        // 邻接表
    int vexnum;              // 顶点数目
    int arcnum;              // 边数目
}ALGraph;
```

深度优先遍历(递归)

```
vector<bool> visited;
void DFS(ALGraph graph, int v){
    visited[v] = true;
    ArcNode* p;
    visit(graph.verties[v].data);
    p = G.verties[v].first;
    while(p){
        if(!visited[p->adjvex])
            DFS(graph, p->adjvex);
        p = p->next;
    }
}
```

深度优先遍历(迭代)

```
void DFSTraverse(Graph G,int v){ //图的非递归深度优先遍历
    int i,visited[MaxSize],top;
    ArcNode *stack[MaxSize],*p;
    for(i = 0; i < G.vexnum; i++){     //将所有顶点都添加未访问标志0
        visited[i] = 0;
    }
    printf("%4c",G.verties[v].data); //访问顶点v并将访问标志置为1
    visited[v] = 1;

    top = -1; //初始化栈
    p = G.verties[v].firstArc;     //p指向顶点v的第一个邻接点
    while(top > -1 || p != NULL){
        while(p!=NULL){
            if(visited[p->adjvex] == 1){
                p = p->nextarc;
            }else{
                printf("%4c",G.verties[p->adjvex].data);
                visited[p->adjvex]=1;
                stack[++top] = p;
                p = G.verties[p->adjvex].firstArc;
            }
        }
        if(top > -1){
            p = stack[top--];
            p = p->nextarc;
        }
    }
}
```

广度优先遍历

```
void BFS(Graph G,int v){     //非递归图的广度优先遍历
    ArcNode *p;
    int i,front,rear,visited[MaxSize],queue[MaxSize];
    front = rear = -1;
    for(i = 0; i < G.vexnum; i++){
        visited[i] = 0;
    }
    printf("%4c",G.verties[v].data);
    visited[v] = 1;
    queue[++rear] = v;
    while(front < rear){
        v = queue[++front];
        p = G.verties[v].firstArc;
        while(p ! =NULL){
            if(!visited[p->adjvex]){
                visited[p->adjvex] = 1;
                printf("%4c",G.verties[p->adjvex].data);
                queue[++rear] = p->adjvex;
            }
            p = p->nextarc;
        }
    }
}
```