```
template<class VertexType>
void GraphType<VertexType>::AddEdge(VertexType fromVertex, VertexType toVertex, int weight)
{
    int row = IndexIs(vertices, fromVertex);
    int col= IndexIs(vertices, toVertex);
    edges[row][col] = weight;
}
template<class VertexType>
int GraphType<VertexType>::WeightIs(VertexType fromVertex, VertexType toVertex)
{
    int row = IndexIs(vertices, fromVertex);
    int col= IndexIs(vertices, toVertex);
    return edges[row][col];
}
template<class VertexType>
void GraphType<VertexType>::GetToVertices(VertexType vertex, QueType<VertexType>& adjVertices)
{
    int fromIndex, toIndex;
    fromIndex = IndexIs(vertices, vertex);
    for (toIndex = 0; toIndex < numVertices; toIndex++)
        if (edges[fromIndex][toIndex] != NULL_EDGE)
            adjVertices.Enqueue(vertices[toIndex]);
}
```

```
template<class VertexType>
void
GraphType<VertexType>::DepthFirstSearch(Vertex
Type startVertex, VertexType endVertex)
{
    StackType<VertexType> stack;
    QueType<VertexType> vertexQ;
    bool found = false;
    VertexType vertex, item;

    ClearMarks();
    stack.Push(startVertex);
    do
    {
        vertex = stack.Top();
        stack.Pop();
        if (vertex == endVertex)
        {
            cout << vertex << " ";
            found = true;
        }
        else
        {
            if (!IsMarked(vertex))
            {
                MarkVertex(vertex);
                cout << vertex << " ";
                GetToVertices(vertex,vertexQ);
                while (!vertexQ.IsEmpty())
                {
                    vertexQ.Dequeue(item);
                    if (!IsMarked(item))
                        stack.Push(item);
                }
            }
        }
    } while (!stack.IsEmpty() && !found);
    cout << endl;
    if (!found)
        cout << "Path not found." << endl;
}
```

```
template<class VertexType>
void
GraphType<VertexType>::BreadthFirstSearch(Vertex
Type startVertex, VertexType endVertex)
{
    QueType<VertexType> queue;
    QueType<VertexType> vertexQ;

    bool found = false;
    VertexType vertex, item;

    ClearMarks();
    queue.Enqueue(startVertex);
    do
    {
        queue.Dequeue(vertex);
        if (vertex == endVertex)
        {
            cout  << vertex << " ";
            found = true;
        }
        else
        {
            if (!IsMarked(vertex))
            {
                MarkVertex(vertex);
                cout  << vertex << " ";
                GetToVertices(vertex, vertexQ);

                while (!vertexQ.IsEmpty())
                {
                    vertexQ.Dequeue(item);
                    if (!IsMarked(item))
                        queue.Enqueue(item);
                }
            }
        }
    } while (!queue.IsEmpty() && !found);
    cout << endl;
    if (!found)
        cout << "Path not found." << endl;
}
```