目录

[类：Node 3](#_Toc498069018)

[函数：Node 3](#_Toc498069019)

[测试1：NodeTest() 3](#_Toc498069020)

[测试2：StartNodeTest() 3](#_Toc498069021)

[类：Edge 3](#_Toc498069022)

[函数：Edge 3](#_Toc498069023)

[测试1：EdgeTest() 3](#_Toc498069024)

[类：Graph 4](#_Toc498069025)

[函数：Graph 4](#_Toc498069026)

[测试1：GraphTest() 4](#_Toc498069027)

[函数：AddNode 4](#_Toc498069028)

[测试1：AddNodeTest() 4](#_Toc498069029)

[测试2：AddStartNodeTest() 4](#_Toc498069030)

[测试3：AddRepeatStartNodeTest() 4](#_Toc498069031)

[测试4：AddRepeatNodeTest() 4](#_Toc498069032)

[测试5：AddRepeatNodeAndStartNodeTest() 4](#_Toc498069033)

[测试6：AddNodeHugeTest() 4](#_Toc498069034)

[函数AddEdge 5](#_Toc498069035)

[测试1：AddEdgeStartNodeToNode() 5](#_Toc498069036)

[测试2：AddEdgeNodeToStartNode() 5](#_Toc498069037)

[测试3：AddEdgeNodeToNode() 5](#_Toc498069038)

[测试4：AddEdgeStartNodeToStartNode() 5](#_Toc498069039)

[测试5：AddEdgeRepeatTest() 5](#_Toc498069040)

[测试6：AddEdgeSourceNullTest() 5](#_Toc498069041)

[测试7：AddEdgeTargetNullTest() 5](#_Toc498069042)

[测试8：AddEdgeSameSourceAndTargetTest() 5](#_Toc498069043)

[测试9：AddEdgeHugeTest() 5](#_Toc498069044)

[函数：GetAllPathBetweenTwoNode 5](#_Toc498069045)

[测试1：GetAllPathBetweenTwoNodeNullStartTest() 6](#_Toc498069046)

[测试2：GetAllPathBetweenTwoNodeNullEndTest() 6](#_Toc498069047)

[测试3\_1：GetAllPathBetweenTwoNodeNoPathTest() 6](#_Toc498069048)

[测试3\_2：GetAllPathBetweenTwoNodeNoPathLoopTest() 6](#_Toc498069049)

[测试4\_1：GetAllPathBetweenTwoNodeOnePathTest() 6](#_Toc498069050)

[测试4\_2：GetAllPathBetweenTwoNodeOnePathLoopTest() 6](#_Toc498069051)

[测试5\_1：GetAllPathBetweenTwoNodeManyPathTest() 7](#_Toc498069052)

[测试5\_2：GetAllPathBetweenTwoNodeManyPathLoopTest() 7](#_Toc498069053)

[测试6\_1：GetAllPathBetweenTwoNodeStartInCycleGraphTest() 7](#_Toc498069054)

[测试6\_2：GetAllPathBetweenTwoNodeStartInCycleGraphLoopTest() 8](#_Toc498069055)

[测试7\_1：GetAllPathBetweenTwoNodeEndInCycleGraphTest() 8](#_Toc498069056)

[测试7\_2：GetAllPathBetweenTwoNodeEndInCycleGraphLoopTest() 8](#_Toc498069057)

[测试8\_1：GetAllPathBetweenTwoNodeSameNodeTest() 8](#_Toc498069058)

[测试8\_2：GetAllPathBetweenTwoNodeSameNodeLoopTest() 9](#_Toc498069059)

[测试9\_1：GetAllPathBetweenTwoNodeIsolatedStartTest() 9](#_Toc498069060)

[测试9\_2：GetAllPathBetweenTwoNodeIsolatedStartLoopTest() 9](#_Toc498069061)

[测试10\_1：GetAllPathBetweenTwoNodeIsolatedEndTest() 9](#_Toc498069062)

[测试10\_2：GetAllPathBetweenTwoNodeIsolatedEndLoopTest() 10](#_Toc498069063)

[测试11\_1：GetAllPathBetweenTwoNodeHugeTest() 10](#_Toc498069064)

[测试11\_2：GetAllPathBetweenTwoNodeHugeLoopTest() 12](#_Toc498069065)

[函数：GetAllPathFromStartNode 12](#_Toc498069066)

[测试1：GetAllPathFromStartNodeNullTest() 12](#_Toc498069067)

[测试2：GetAllPathFromStartNodeNoStartNodeTest() 12](#_Toc498069068)

[测试3：GetAllPathFromStartNodeOneStartNodeNoPathTest() 12](#_Toc498069069)

[测试4：GetAllPathFromStartNodeOneStartNodeOnePathTest() 12](#_Toc498069070)

[测试4：GetAllPathFromStartNodeOneStartNodeManyPathTest() 13](#_Toc498069071)

[测试5：GetAllPathFromStartNodeManyStartNodeOnePathTest() 13](#_Toc498069072)

[测试6：GetAllPathFromStartNodeManyStartNodeManyPathTest() 13](#_Toc498069073)

[测试7：GetAllPathFromStartNode2StartNodeInSamePathTest() 14](#_Toc498069074)

[测试8：GetAllPathFromStartNodeAsTargetTest() 14](#_Toc498069075)

[测试9：GetAllPathFromStartNodeTest() 14](#_Toc498069076)

[测试10：GetAllPathFromStartNodeHugeTest() 14](#_Toc498069077)

# 类：Node

## 函数：Node

Node(string name， Guid id， bool isStartNode = false)

构造函数，通过参数实例化一个节点。

### 测试1：NodeTest()

能够正确实例化一个普通节点

### 测试2：StartNodeTest()

能够正确实例化一个起始节点

# 类：Edge

## 函数：Edge

Edge(Guid source， Guid target)

构造函数，通过源点ID和节点ID，实例化连接两个节点的一条有向边。

### 测试1：EdgeTest()

能够正确实例化一条边 包含源点ID和终点ID

# 类：Graph

## 函数：Graph

Graph()

构造函数，实例化一个图，并对存储图的节点列表(NodeList)和边列表(EdgeList)进行初始化。

### 测试1：GraphTest()

能够正确实例化一个图，且图中的节点列表和边列表为空

## 函数：AddNode

void AddNode(Node node)

给图添加新的节点，将传入的节点保存至图内的节点列表(NodeList)。

### 测试1：AddNodeTest()

添加普通节点

### 测试2：AddStartNodeTest()

添加起始节点

### 测试3：AddRepeatStartNodeTest()

添加重复的起始节点 (ExceptException)

### 测试4：AddRepeatNodeTest()

添加重复的普通节点 (ExceptException)

### 测试5：AddRepeatNodeAndStartNodeTest()

添加重复的起始节点和普通节点(节点ID相同的起始节点和普通节点) (ExceptException)

### 测试6：AddNodeHugeTest()

添加大量节点，并且能够正常添加

## 函数：AddEdge

void AddEdge(Guid nodeSourceID， Guid nodeTargetID)

给图添加新的节点连边，通过节点ID实例化一条有向边(Edge)并存入边列表(EdgeList)

### 测试1：AddEdgeStartNodeToNode()

添加起始节点指向普通节点的边

### 测试2：AddEdgeNodeToStartNode()

添加普通节点指向起始节点的边

### 测试3：AddEdgeNodeToNode()

添加普通节点指向普通节点的边

### 测试4：AddEdgeStartNodeToStartNode()

添加起始节点指向起始节点的边

### 测试5：AddEdgeRepeatTest()

添加重复的边，当添加的边与原图中已存在的边的ID相同则判定为重复边(ExceptException)

### 测试6：AddEdgeSourceNullTest()

添加将图中不存在的节点作为源点的边 (ExceptException)

### 测试7：AddEdgeTargetNullTest()

添加将图中不存在的节点作为终点的边 (ExceptException)

### 测试8：AddEdgeSameSourceAndTargetTest()

添加源点和终点为同一节点的边 (ExceptException)

### 测试9：AddEdgeHugeTest()

添加大量的边，并且能够正常添加

## 函数：GetAllPathBetweenTwoNode

List<List<Guid>> GetAllPathBetweenTwoNode(Guid startID， Guid endID)

指定开始节点ID和终点节点ID，返回开始节点到终点节点的所有路径

### 测试1：GetAllPathBetweenTwoNodeNullStartTest()

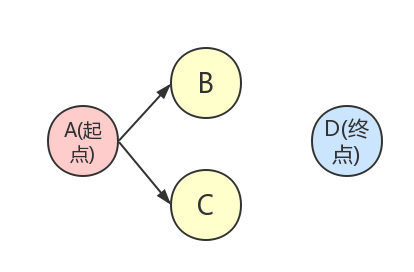
指定图中不包含的节点作为起点(ExceptException)

### 测试2：GetAllPathBetweenTwoNodeNullEndTest()

指定图中不包含的节点作为终点(ExceptException)

### 测试3\_1：GetAllPathBetweenTwoNodeNoPathTest()

指定的两个节点之间没有可达路径（使用递归方法实现Depth-First-Search），所获得的路径列表.Count值应为0，如下图

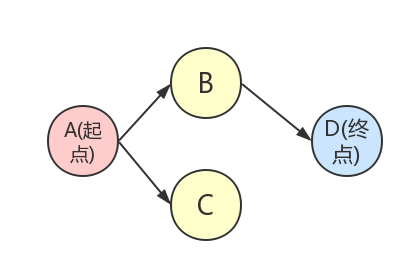


### 测试3\_2：GetAllPathBetweenTwoNodeNoPathLoopTest()

指定的两个节点之间没有可达路径，所获得的路径列表.Count值应为0，（使用循环方法实现DFS）。

### 测试4\_1：GetAllPathBetweenTwoNodeOnePathTest()

指定的两个节点之间存在可达路径（使用递归方法实现Depth-First-Search），且只有一条。



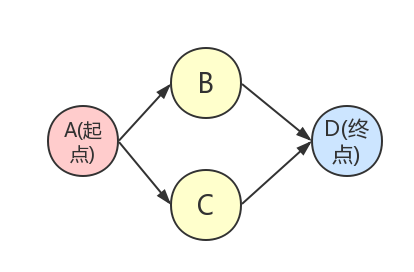
如上图，返回的路径列表应为{A->B->D}

### 测试4\_2：GetAllPathBetweenTwoNodeOnePathLoopTest()

指定的两个节点之间存在可达路径，且只有一条，（ 使用循环方法实现DFS）。

### 测试5\_1：GetAllPathBetweenTwoNodeManyPathTest()

指定的两个节点之间存在可达路径，且有多条。（使用递归方法实现Depth-First-Search）



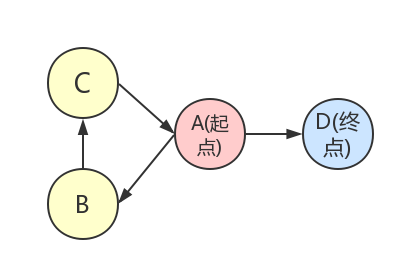
如上图，返回的路径列表应为{A->B->D， A->C->D}

### 测试5\_2：GetAllPathBetweenTwoNodeManyPathLoopTest()

指定的两个节点之间存在可达路径，且有多条，（使用循环方法实现DFS）。

### 测试6\_1：GetAllPathBetweenTwoNodeStartInCycleGraphTest()

包含环状的图 ，且起点在环状结构中，所获得的路径列表中，同一条路径不能有重复节点。（使用递归方法实现Depth-First-Search）



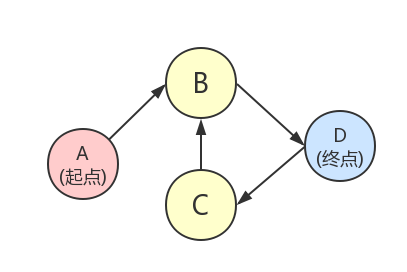
如上图，所获得路径列表应为 {A->D} 不能包含{A->B->C->A->D}

### 测试6\_2：GetAllPathBetweenTwoNodeStartInCycleGraphLoopTest()

包含环状的图 ，且起点在环状结构中，所获得的路径列表中，同一条路径不能有重复节点。（使用循环方法实现DFS）。

### 测试7\_1：GetAllPathBetweenTwoNodeEndInCycleGraphTest()

包含环状的图 ，且终点在环状结构中，同一条路径不能有重复节点。（使用递归方法实现Depth-First-Search）



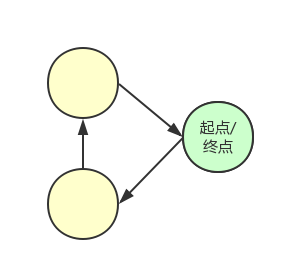
如上图，所获得的路径列表应为{A->B->D}， 不能包含{A->B->D->C->B->D}

### 测试7\_2：GetAllPathBetweenTwoNodeEndInCycleGraphLoopTest()

包含环状的图 ，且终点在环状结构中，同一条路径不能有重复节点。（使用循环方法实现DFS）

### 测试8\_1：GetAllPathBetweenTwoNodeSameNodeTest()

指定同一节点作为起点和终点 ， 所获得的路径列表中，只有一条路径，且该路径只有一个节点（起点或终点）。（使用递归方法实现Depth-First-Search）



如上图，所得路径应为{起点}

### 测试8\_2：GetAllPathBetweenTwoNodeSameNodeLoopTest()

指定同一节点作为起点和终点 ， 所获得的路径列表中，只有一条路径，且该路径只有一个节点（起点或终点）。（使用循环方法实现DFS）

### 测试9\_1：GetAllPathBetweenTwoNodeIsolatedStartTest()

起点为孤立节点，所获得的路径列表.Count值应为0 ，如下图。（使用递归方法实现Depth-First-Search）

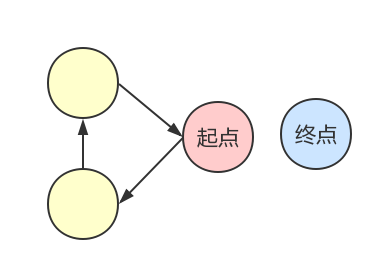
### C:\Users\Har\AppData\Local\Microsoft\Windows\INetCache\Content.Word\未命名文件 (2).png

### 测试9\_2：GetAllPathBetweenTwoNodeIsolatedStartLoopTest()

起点为孤立节点，所获得的路径列表.Count值应为0。（使用循环方法实现DFS）

### 测试10\_1：GetAllPathBetweenTwoNodeIsolatedEndTest()

终点为孤立节点，所获得的路径列表.Count值应为0，如下图。（使用递归方法实现Depth-First-Search）



### 测试10\_2：GetAllPathBetweenTwoNodeIsolatedEndLoopTest()

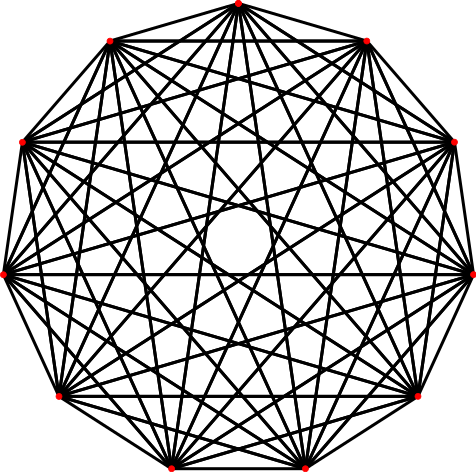
终点为孤立节点，所获得的路径列表.Count值应为0。（使用循环方法实现DFS）

### 测试11\_1：GetAllPathBetweenTwoNodeHugeTest()

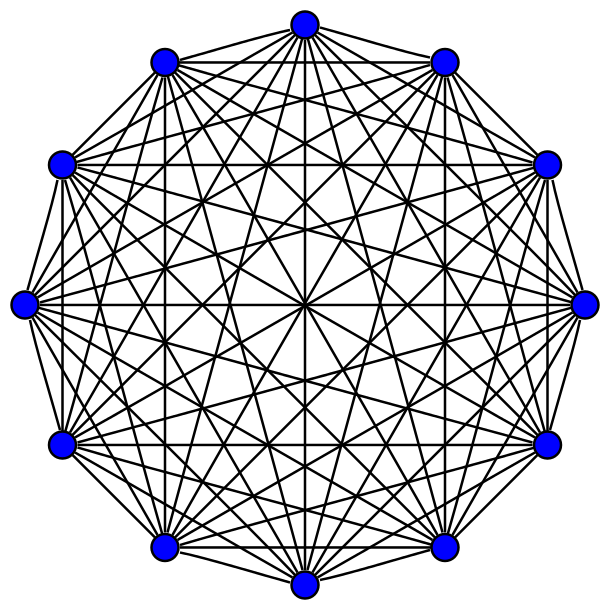
由于缺少测试数据，因此使用完全图（complete graph）来作为复杂图去测试该函数。

在完全图中任意两个节点之间的可达路径是一致的。可通过以下公式来计算完全图中任意两个节点的路径数。*n表示完全图中节点个数，c(n)表示n节点完全图中任意两个节点的路径个数*

本次测试中，仅考虑函数所返回的路径列表中的路径个数是否符合以上公式，而不考虑每条路径的准确性。



上图为n==11的完全图，共有11个节点，55条无向边（110条有向边），任意2点的可达路径为984，610，在测试中，可以通过测试，运行时间约为4秒



上图为n=12的完全图，共有12个节点，66条无项边（132条有向边），任意2点的可达路径为9，846，101条，测试中运行时间过长，无法通过单元测试。

### 测试11\_2：GetAllPathBetweenTwoNodeHugeLoopTest()

使用完全图（complete graph）来作为复杂图去测试，使用循环方法实现DFS。

## 函数：GetAllPathFromStartNode

List<List<Guid>> GetAllPathFromStartNode(Guid targetNodeID)

指定目的节点ID，返回所有起始节点到目的节点的所有路径

### 测试1：GetAllPathFromStartNodeNullTest()

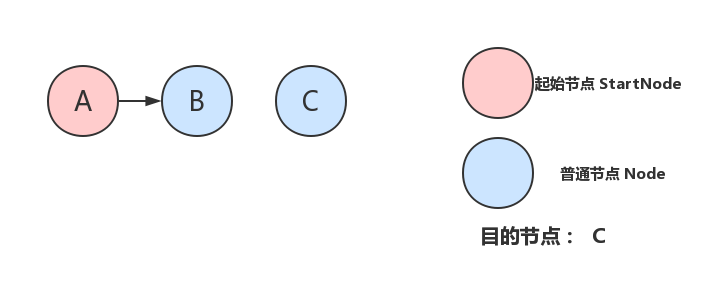
指定图中不包含的节点作为目的节点 (ExceptException)

### 测试2：GetAllPathFromStartNodeNoStartNodeTest()

在不包含起始节点的图 中获得到目的节点的路径，所获得的路径列表.Count值应为0

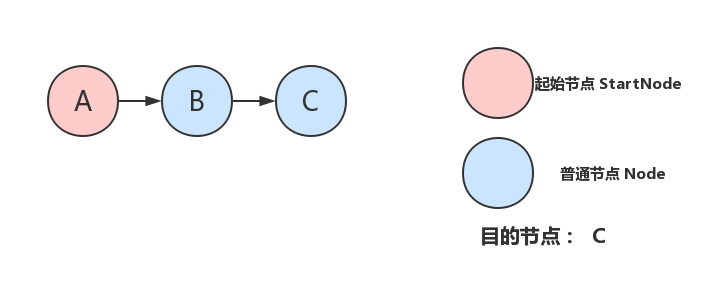
### 测试3：GetAllPathFromStartNodeOneStartNodeNoPathTest()

在包含一个起始节点但无可达路径的图 中获得到目的节点的路径，所获得的路径列表.Count值应为0 ，如下图。



### 测试4：GetAllPathFromStartNodeOneStartNodeOnePathTest()

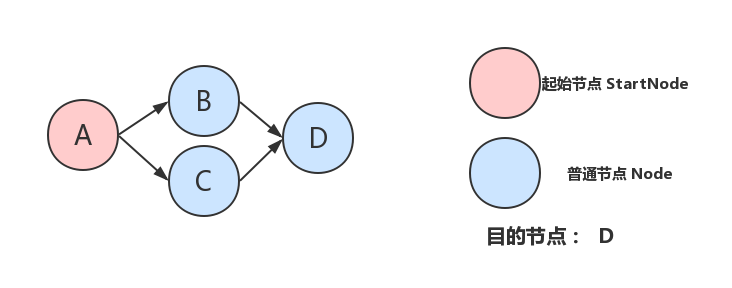
在包含一个起始节点有且仅有一条可达路径的图 中，获得到目的节点的路径。



如上图，指定C 为目的节点，获得的可达路径列表应为{A->B->C}

### 测试4：GetAllPathFromStartNodeOneStartNodeManyPathTest()

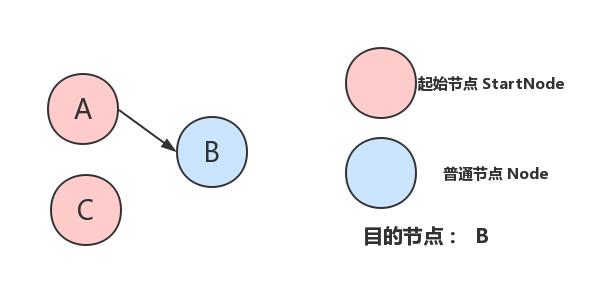
在包含一个起始节点且有多条可达路径的图 中，获得到目的节点的路径。



如上图，指定D为目的节点，获得的可达路径列表应为{A->B->D， A->C->D}

### 测试5：GetAllPathFromStartNodeManyStartNodeOnePathTest()

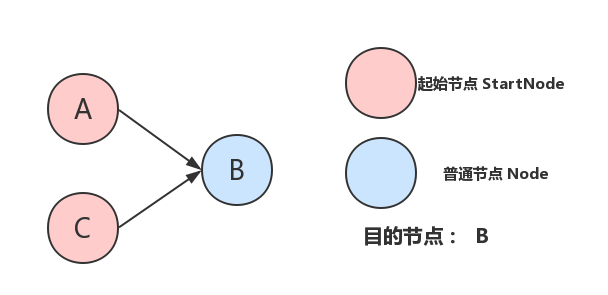
在包含多个起始节点但仅有一条可达路径的图 中，获得到目的节点的路径。



如上图，节点B为目的节点，获得的可达路径应为{A->B}

### 测试6：GetAllPathFromStartNodeManyStartNodeManyPathTest()

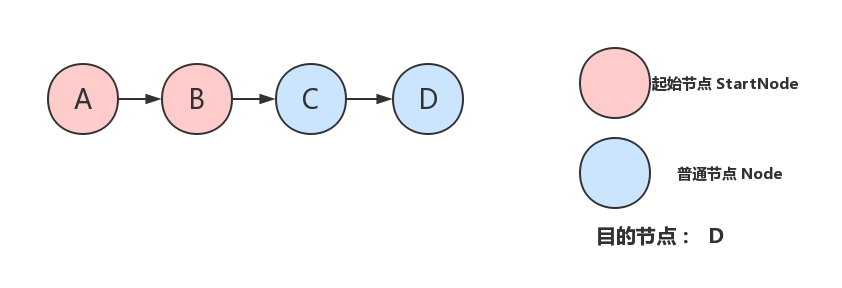
在包含多个起始节点但且有多条可达路径的图 中，获得到目的节点的路径。



如上图，节点B为目的节点，获得的可达路径应为{A->B， C->B}

### 测试7：GetAllPathFromStartNode2StartNodeInSamePathTest()

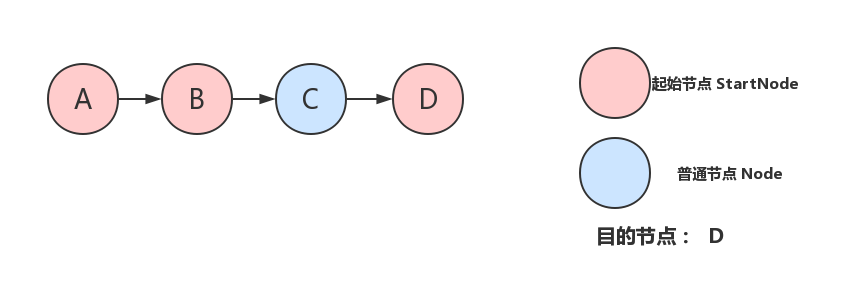
存在2个以上的起始节点在同一条可达路径上 ，所获得的路径列表中，仍旧需要分别计算所有可达路径。



如上图指定D为目的节点，则应返回路径列表为{A->B->C->D， B->C->D}。

### 测试8：GetAllPathFromStartNodeAsTargetTest()

指定起始节点作为目的节点，返回的路径中应存在一条路径，包含只含有一个节点（目的节点、起始节点）的路径。



如上图，指定D为目的节点，则返回的路径列表中，应为{A->B->C->D， B->C->D， D}

### 测试9：GetAllPathFromStartNodeTest()

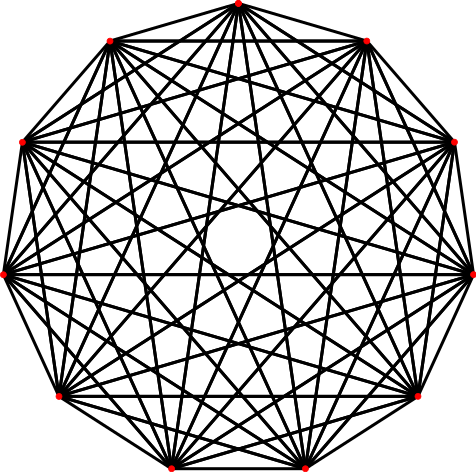
指定普通节点作为目的节点

### 测试10：GetAllPathFromStartNodeHugeTest()

同样使用完全图来作为复杂图测试，起始节点到目的节点的节点个数可用以下公式进行计算，其中n表示完全图中节点个数，c(n)表示n节点完全图中任意两个节点的路径个数，k表示图中起始节点的个数，C(n， k)表示起始节点到目的节点的路径个数

若指定普通节点为目的节点则

若指定起始节点为目的节点则

**

如上图为11个节点的完全图，其中有7个起始节点，指定1个普通节点作为目的节点，可获得6，904，870条可达路径，可通过单元测试，运行时间约为30秒。