Operation\_Group类设计

UserInPut 用于完成泡泡覆盖的工作，即判断边的独立性。

属性

Graph = digraph() 不是handle。（判断handle的方法isa(PebbleGame.UserInPut.GraphPlot,'handle')）。相当于作者的pebble index。

GraphPlot 是handle，显示内容的句柄。

GraphFigure 是handle,绘图窗口的句柄。

XData

YData

EdgeReadyForAdd:还没有被增加到图上的边。

EdgeUnableAdd:不能被增加到图上的边。重复边：redundant bound。通过复制EdgeTempletTable来得以初始化。

WorkingEdge:当前正在尝试增加的边，初始化为0. 是二元数组，不是一个表！

LamanNodesList = containers.ArrayList(k): k = (n+1)n/2. 拉曼图的节点。当不能找到第四个泡泡的时候，搜索的路径经过的那些节点。由于边是逐渐增加上去的，所以这个列表内的很多项都需要合并.

EdgeTempletTable: %用于在存储边的时候作为模板。其结构与EdgeReadyForAdd一样，但是一个空表。

LamanEdgesList = containers.ArrayList(k): %用来保存LamanEdges的列表。这个列表后期经过合并，最后得到LamanSubgraphs各图包含的边。根据LamanNodesList中包含的内容，在用到的时候初始化。handle类

RigidClusterEdgeColor; %创建handle着色类。其中含有RigidCluster的编号，对应边的颜色。%在StartIdentifyRigidCluster()中首次初始化。

Graph.Nodes 表设计

注意：我的pebble index在Graph里面。

表结构是逐渐复杂起来的。在UserInPut\_Group， Operation\_Group， Result\_Group三个组里面不同.

Name：默认，节点名称 (UserInPut\_Group，Operation\_Group)

FreePebble：节点的自由泡泡个数 （Operation\_Group）

Pin：节点被钉住的泡泡个数 (Operation\_Group)

颜色卡

b blue 有2个Free泡泡

g green 有1个Free泡泡

k black 没有Free泡泡

y yellow 显示待增加边的两个节点；显示查找过程中经过的路径。

r red 标识某一个节点自由泡泡多余两个

Redundant Bond : 重复边。黑色虚线。 GraphPlot.highlight(s,t, 'LineStyle',’:’)

Revolute Joint / Pivot: 转轴。叉形标记。 GraphPlot.highlight(NodeID, 'Marker',’x’)

c cyan（蓝绿色）

m magenta（紫红色的）

操作方法

访问表：Table{ EdgeReadyForAddNumber,'EndNodes'}

删除表：Table(EdgeReadyForAddNumber,:) = []

为表增加一个列：Table.Size

为表增加一行元素：Table{end+1,:} = [1 1 1]

高亮GraphFigure的某条边：highlight(GraphFigure,[1 3],'NodeColor','g', 'EdgeColor','r','LineWidth',1.5)

高亮GraphFigure的某一条节点：highlight(GraphFigure,nodeIDs)

检查一条边是否存在某一图中: 0 == findedge(Graph,A,B)

向有向图中增加一条边: addedge(G,s,t,w)

生成一个列表：LamanNodes = containers.ArrayList(5)

判断一个对象是不是handle: isa(PebbleGame.Operation.EdgeReadyForAdd,'handle')

方法

[] = IDIntheRange(NodeID)

检查输入的ID是否在ID的范围内.

1. 如果在范围内,则保持安静.
2. 如果不在范围内,则立即出错退出.

[] = EdgeNodeMatch(nodeIDs,[ VertexA, VertexB])

检查nodeIDs是否是VertexA或VertexB其中之一。如果不是，出错退出。

“错误，边和节点号不对应！”

[isCovered] = EdgeisCovered([ VertexA, VertexB])

检查Graph的边[ VertexA, VertexB]是否已经被覆盖了。注意边的顺序。

1. 检查0 == findedge(Graph, VertexA, VertexB)为真。
2. 如果为真，说明没有这么一条边，isCovered = false
3. 如果为假，说明没有这么一条边, isCovered = true

[FreePebbleNumber] = ReadFreePebbleNumber (NodeID)

表查询，给出NodeID节点上的FreePebble个数。

[] = WriteFreePebbleNumber\_ (NodeID, FreePebbleNumber)

1. 如果FreePebbleNumber>2，出错退出。“写入了多余两个泡泡数值”。
2. 表写入，改写NodeID节点上的FreePebble个数为FreePebbleNumber。

[Pin Number] = ReadPin (NodeID)

表查询，给出NodeID节点上的Pin个数。

[] = WritePin \_ (NodeID, Pin Number)

表写入，改写NodeID节点上的Pin个数为Pin Number。

[] = PinPebble(NodeID, NumbertoPin)

暂时钉住NumbertoPin个泡泡不让它们移动.

1. IDIntheRange(NodeID)
2. 如果ReadFreePebbleNumber (NodeID)< NumbertoPin。出错退出。“操作不能完成，没有那么多FreePebble可Pin”。
3. 调节FreePebbleNumber：Temp = ReadFreePebbleNumber (NodeID) – NumbertoPin
4. WriteFreePebbleNumber\_ (NodeID, Temp)
5. 调节Pin：Temp = ReadPin (NodeID) + NumbertoPin
6. WritePin \_ (NodeID, Temp)

[] = UnPinPebble(NodeID, NumbertoUnPin)

将NumbertoUnPin个泡泡转换为FreePebble。

1. IDIntheRange(NodeID)
2. 如果ReadPin (NodeID)< NumbertoUnPin，出错退出。“没有那么多Pin泡泡可以变为FreePebble”。
3. = ReadFreePebbleNumber (NodeID) + NumbertoUnPin
4. 如果Temp >2，出错退出。“操作不能完成，会导致当前结点FreePebble太多”。
5. 调节FreePebbleNumber：
6. WriteFreePebbleNumber\_ (NodeID, Temp)
7. 调节Pin：Temp = ReadPin (NodeID) - NumbertoPin
8. WritePin \_ (NodeID, Temp)

[] = AutoUnPinAPebble()

全图操作。如果有泡泡空位的话，自动从Pin的泡泡里面UnPin一个出来。

1. 判断HowManyPin()<=0。如果为真，出错退出。“全图没有任何被Pin住的泡泡。”
2. 遍历每一个NodeID。
3. 如果：ReadPin (NodeID)>0
   1. try
   2. UnPinPebble(NodeID, 1)
   3. catch
   4. 空
   5. End
4. 判断HowManyPin()>0。给出图示。“仍然有被Pin住的泡泡。”

[TotalFreePebbleNumber] = HowManyFreePebble()

计算当前全图共有多少个FreePebble。

[TotalPinNumber] = HowManyPin()

计算当前全图共有多少个Pin。

[] = Show()

生成新的生成新的GraphPlot对象。

1. GraphPlot = Graph.plot('XData',obj.XData,'YData',obj.YData,’ EdgeColor’, ‘k’);
2. 遍历每一个节点
3. 表查询ReadFreePebbleNumber (NodeID)==2，则高亮这个节点highlight(GraphPlot,nodeIDs, 'NodeColor','b')
4. 表查询ReadFreePebbleNumber (NodeID)==1，则高亮这个节点highlight(GraphPlot,nodeIDs, 'NodeColor','g')
5. 表查询ReadFreePebbleNumber (NodeID)==0，则高亮这个节点highlight(GraphPlot,nodeIDs, 'NodeColor','k')
6. 表查询ReadFreePebbleNumber (NodeID)>2，出错退出! “出现了拥有过多泡泡的节点”
7. 表查询ReadFreePebbleNumber (NodeID)<0，出错退出！“部分节点出现了负值泡泡！”

[] = CovertheEdge\_([ VertexA, VertexB])

利用DonateVertex上的FreePebble覆盖[ VertexA, VertexB]。DonateVertex一定是[ VertexA, VertexB]相邻的。

1. DonateVertex = VertexA
2. EdgeisCovered([ VertexA, VertexB]) == true，以检查当前的边是否被覆盖。如果为真,则出错退出.“这条边已经被覆盖了。”
3. ReadFreePebbleNumber (DonateVertex)>0,以检查是否有足够的FreePebble以供覆盖。如果没有，出错退出。“DonateVertex”上没有FreePebble可供覆盖。
4. Graph = Graph. addedge(DonateVertex, VertexB,1)，因为覆盖边就是增加边。增加的边应该离开DonateVertex。
5. Temp = ReadFreePebbleNumber (DonateVertex)
6. WriteFreePebbleNumber\_ (DonateVertex, Temp-1)，改写DonateVertex上的FreePebble数值。
7. Show()，以更新GraphPlot。

[] = UnCovertheEdge\_([ VertexA, VertexB])

解除对边的覆盖，并将泡泡归还给VertexA。

1. EdgeisCovered([ VertexA, VertexB]) == false。说明这条边没有被覆盖，出错退出。“这条边没有被覆盖。”
2. ReadFreePebbleNumber (VertexA)>=2。如果为真，说明没有地方释放FreePebble。出错退出。“Source节点上的FreePebble太多。”
3. Graph = Graph. rmedge(VertexA, VertexB). 注意Graph是一个值类。
4. Temp = ReadFreePebbleNumber (VertexA)
5. WriteFreePebbleNumber\_ (VertexA, Temp+1)，改写VertexA上的FreePebble数值。
6. Show()，以更新GraphPlot。

[] = TryToAddaEdge(EdgeReadyForAddNumber, ‘database’)

开始尝试将某一条边增加到Graph上面去。

1. 检测WorkingEdge是否为空，如果不为空，则出错退出。“上一次操作没有完成。如果这条边不能够被覆盖，建议执行承认失败功能。”
2. 把这条边拷贝到WorkingEdge。
3. 删除EdgeReadyForAdd中的对应EdgeReadyForAddNumber项。（Table(EdgeReadyForAddNumber,:) = []）
4. [] = Show() (千万不能操作Graph)。
5. 高亮待增加的边。（Table{ EdgeReadyForAddNumber,'EndNodes'}）。并且使用highlight(h,[1 3],'NodeColor','y', 'EdgeColor','y','LineWidth',1.5)将待增加的边变成黄色的。

[] = CoverWorkingEdge()

利用WorkingEdge相关联的两个节点中的任何一个上面的FreePebble来覆盖WorkingEdge。

1. 提取与WorkingEdge相关联的WorkingNodeA和WorkingNodeB
2. IDIntheRange(WorkingNodeA)
3. IDIntheRange(WorkingNodeB)
4. 判断ReadFreePebbleNumber (WorkingNodeA)>0

CovertheEdge\_([ WorkingNodeA，WorkingNodeB])

WorkingNodeA =WorkingNodeB=0. WorkingEdge=0,

Show()，以更新GraphPlot。

**然后返回。**

1. 判断ReadFreePebbleNumber (WorkingNodeB)>0

CovertheEdge\_([ WorkingNodeB，WorkingNodeA)

WorkingNodeA =WorkingNodeB=0. WorkingEdge=0,

Show()，以更新GraphPlot。

**然后返回。**

1. 出错。“WorkingEdge相关联的节点上没有FreePebble。尝试为其寻找FreePebble”

[] = UnabletoCoverWorkingEdge()

如果实在不能够找到FreePebble来覆盖WorkingEdge，那么使用这个函数放弃。

1. 检测WorkingEdge ~= 0。如果是空的，则出错退出。“WorkingEdge是空的。”
2. EdgeUnableAdd{end+1,:}= WorkingEdge，以便把不成功加入的边保存到EdgeUnableAdd里面去。
3. WorkingEdge = 0；
4. Show()，以更新GraphPlot。

[PathtoFreePebble, BreadthFirstSearch] = FindAPebble(nodeID)

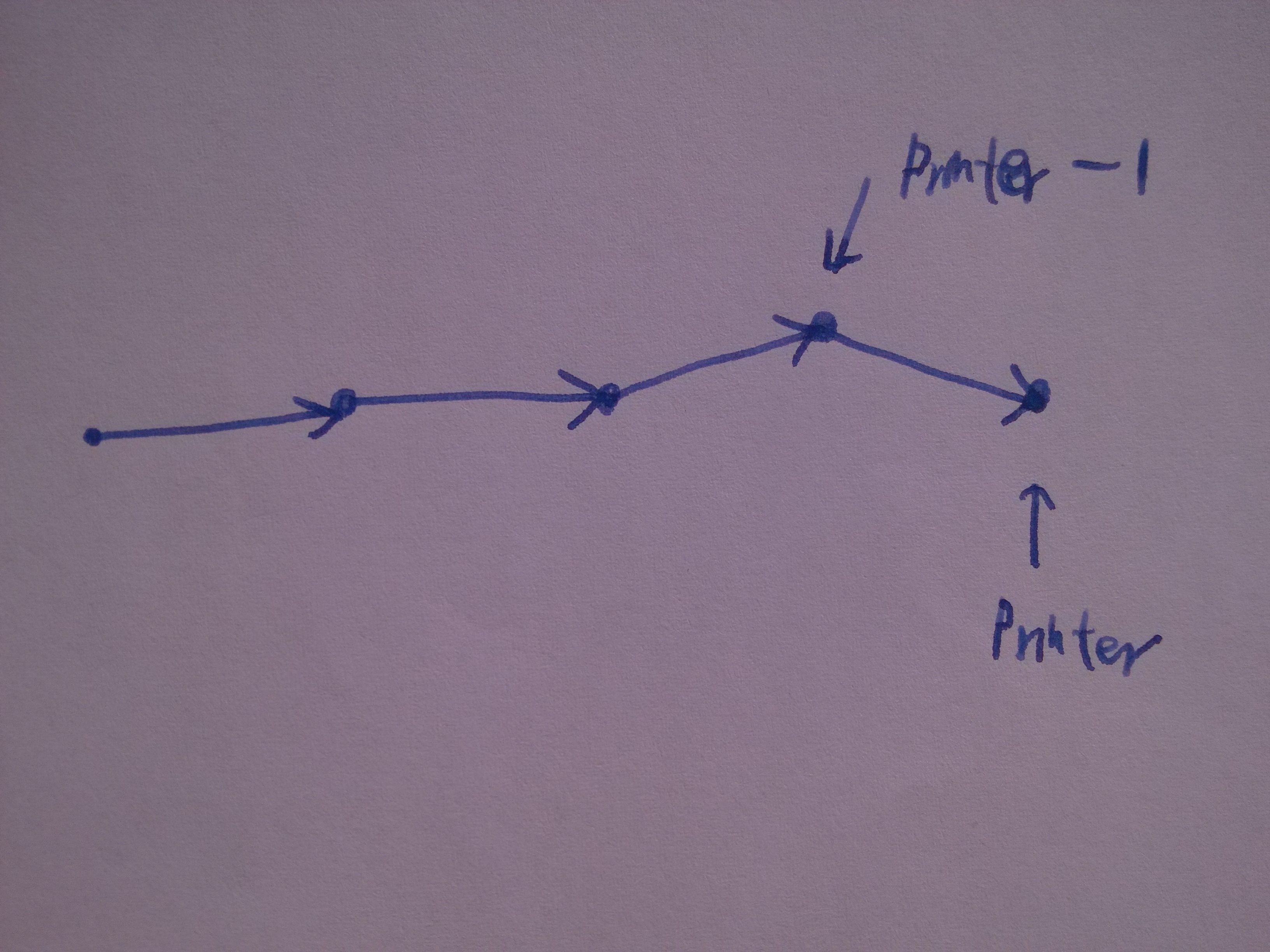
从nodeID开始，沿着donate的方向，发现一个FreePebble

1. IDIntheRange(NodeID)
2. Search = Graph.bfsearch(nodeID)，广度优先算法，找到所有可达节点。
3. BreadthFirstSearch = Search. 当找不到FreePebble时，需要利用BreadthFirstSearch找到Laman Subgraphs。
4. GraphPlot.highlight(Search, 'NodeColor','y', 'EdgeColor','y','LineWidth',1.5),显示广度优先算法信息。
5. 遍历Search，找到第一个ReadFreePebbleNumber (NodeID)>0的节点FirstFreePebbleNode。
6. 如果找到了，则计算最短路径。
7. Show()，以更新GraphPlot。
8. PathtoFreePebble = Graph.shortestpath(nodeID, FirstFreePebbleNode)。然后高亮显示路径。GraphPlot.highlight(PathtoFreePebble, 'NodeColor','y', 'EdgeColor','y','LineWidth',1.5)
9. 如果没找到，则返回PathtoFreePebble=0

[] = RearrangePebble(PathtoFreePebble)

按照PathtoFreePebble指示的方向，从路径尾节点上开始交换覆盖，从而使得头部得到一个FreePebble。

1. 检查PathtoFreePebble == 0,如果是则出错退出。“无效的路径。”
2. 判断PathtoFreePebble长度是否为1，如果是，直接返回。因为不需要移动，当前路径只有一个节点。
3. Show()，以更新GraphPlot。
4. 建立Printer指向最后一个PathtoFreePebble元素。循环执行3到5直到最后。
5. UnCovertheEdge\_([ \*(Printer-1), \*(Printer)])
6. CovertheEdge\_([ \*(Printer), \*(Printer-1)] )
7. Printer = Printer -1
8. Show()，以更新GraphPlot。



[Independent] = IndependentEdge(EdgeReadyForAddNumber)

检测EdgeReadyForAdd中由EdgeReadyForAddNumber指定的边是否是独立边。

1. [] = TryToAddaEdge(EdgeReadyForAddNumber, ‘database’). 将需要检查的边放在WorkingEdge上面。
2. 提取与WorkingEdge相关联的WorkingNodeA和WorkingNodeB备用。
3. CollectedFreePebbleNumber = 0; 记录这个过程一共收集到了多少个FreePebble。
4. IDIntheRange(WorkingNodeA)。数值可靠性检查。
5. IDIntheRange(WorkingNodeB)。数值可靠性检查。

（针对WorkingNodeA收集第一个泡泡。注意，即便WorkingNodeA本身就有FreePebble也是没有关系的。）

1. try 首先尝试在WorkingNodeA上收集第一个FreePebble。除非找不到FreePebble才会出错。
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeA)
   2. [] = RearrangePebble(PathtoFreePebble)
2. catch ME
3. msg = ['需要详细研究。为什么在节点',num2str(WorkingNodeA),'上一个自由泡泡也收集不到！'];
4. causeException = MException('MATLAB:myCode:IndependentEdge',msg)
5. ME = addCause(ME,causeException);
6. Independent = -1. 第一个泡泡都找不到的条件，还没有研究过。所以是无效值。
7. rethrow(ME) . 出错退出了。
8. end
9. [] = PinPebble(WorkingNodeA, 1)。如果把WorkingNodeA上收集的这个泡泡Pin住。
10. CollectedFreePebbleNumber = CollectedFreePebbleNumber + 1. 成功收集到了泡泡。

（针对WorkingNodeB收集第一个泡泡。注意，即便WorkingNodeB本身就有FreePebble也是没有关系的。）

1. try 首先尝试在WorkingNodeB上收集第一个FreePebble。除非找不到FreePebble才会出错。
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeB)
   2. [] = RearrangePebble(PathtoFreePebble)
2. catch ME
3. msg = ['需要详细研究。为什么在节点',num2str(WorkingNodeB),'上一个自由泡泡也找不到！'];
4. causeException = MException('MATLAB:myCode:IndependentEdge',msg)
5. ME = addCause(ME,causeException);
6. Independent = -2. 第二个泡泡都找不到的条件，还没有研究过。所以是无效值。
7. rethrow(ME)
8. end
9. [] = PinPebble(WorkingNodeB, 1), 如果找到了，把WorkingNodeB上收集的第二个泡泡也Pin住。
10. CollectedFreePebbleNumber = CollectedFreePebbleNumber + 1 成功收集到了泡泡。

（下面就可能出现找不到FreePebble的情况了。）

1. BreadthFirstSearch\_A = BreadthFirstSearch\_B = 0 .新建两个宽度优先算法待存储变量。

（在WorkingNodeA上收集第三个FreePebble。）

1. FailedA = false 在A上第二次收集到泡泡是否失败的标志。
2. try
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeA)
   2. [] = RearrangePebble(PathtoFreePebble)
3. catch ME 发现在WorkingNodeA上面不能发现第三个泡泡。保存宽度优先算法的结果以便生成Laman Subgraphs.
4. BreadthFirstSearch\_A = BreadthFirstSearch
5. FailedA = true
6. end
7. 判断PathtoFreePebble ~= 0. 如果是，说明找到了FreePebble。
8. [] = PinPebble(WorkingNodeA, 1), 把WorkingNodeA上收集的第三个泡泡也Pin住。
9. CollectedFreePebbleNumber = CollectedFreePebbleNumber + 1
10. FailedA = false
11. end

（在WorkingNodeB上收集第四个FreePebble。）

1. FailedB = false
2. try
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeB)
   2. [] = RearrangePebble(PathtoFreePebble)
3. catch ME 发现在WorkingNodeB上面不能发现第四个泡泡。保存宽度优先算法的结果以便生成Laman Subgraphs.
4. BreadthFirstSearch\_B = BreadthFirstSearch
5. FailedB = true
6. end
7. 判断PathtoFreePebble ~= 0. 如果是，说明找到了FreePebble。
8. [] = PinPebble(WorkingNodeB, 1), 把WorkingNodeB上收集的第四个泡泡也Pin住。
9. CollectedFreePebbleNumber = CollectedFreePebbleNumber + 1
10. FailedB = false
11. end
12. CollectedFreePebbleNumber<3。判断是否只找到了两个FreePebble
13. 出错退出。“同时找不到第三个和第四个泡泡，需要自己研究。”
14. 判断CollectedFreePebbleNumber==4。如果是，则说明找到了第四个泡泡。
15. [] = UnPinPebble(WorkingNodeA, 1) 前面把4个泡泡都Pin住了，现在释放WorkingNodeA上面的一个。
16. [] = CoverWorkingEdge()
17. [] = UnPinPebble(WorkingNodeA, 1)
18. [] = UnPinPebble(WorkingNodeB, 2)
19. Independent = true;
20. Show()，以更新GraphPlot。
21. return
22. 判断CollectedFreePebbleNumber~=3。如果是，出错退出。“找到了5个甚至以上的FreePebble，这是不可能的。”
23. 判断FailedB == true. 如果是，则说明在WorkingNodeB上发生了找泡泡失败。
24. [] = UnabletoCoverWorkingEdge() 承认这条边不是独立边，无法覆盖。
25. [] = UnPinPebble(WorkingNodeA, 2)
26. [] = UnPinPebble(WorkingNodeB, 1)
27. LamanSubgraphs.appendElement(BreadthFirstSearch\_B). 保存Laman Subgraphs。
28. Independent = false;
29. Show()，以更新GraphPlot。
30. return
31. 判断FailedA == true. 如果是，则说明在WorkingNodeA上发生了找泡泡失败。
32. [] = UnabletoCoverWorkingEdge(),承认这条边不是独立边，无法覆盖。
33. [] = UnPinPebble(WorkingNodeA, 1)
34. [] = UnPinPebble(WorkingNodeB, 2)
35. LamanSubgraphs.appendElement(BreadthFirstSearch\_A). 保存Laman Subgraphs。
36. Independent = false;
37. Show()，以更新GraphPlot。
38. return

[] = LamanNodesList2LamanEdgesList()

将每一个LamanNodesList中的元素转换为IndependentEdges的边组成的LamanEdges.注意操作中不要污染LamanEdgeTemplet。

1. LamanEdgesList= containers.ArrayList(k)。初始化LamanEdgesList，其大小根据LamanNodesList的有效数值来初始化。LamanEdgesList是handle，复制操作一定注意。
2. IndependentEdges = Graph. Edges; 复制一份IndependentEdges的表
3. 只要LamanNodesList不为空，依次弹出每一个LamanNodesList中的元素，记为LamanNodes。注意不要弹出太多导致错误。
   1. LamanEdge = LamanEdgeTemplet。生成一个空的LamanEdge，准备填充。
   2. 遍历IndependentEdges，设为Edge
      1. 提取Edge的两个点WorkingNodeA，WorkingNodeB。（可能需要str2num转换）
      2. 如果isempty(intersect(LamanNodes, WorkingNodeA)) == false，同时
      3. 如果isempty(intersect(LamanNodes, WorkingNodeB)) == false，
      4. 说明Edge这条边在LamanNodes中。则将Edge加入到LamanEdge
   3. 将LamanEdge压入LamanEdgesList

[] = CombineLamanEdgesList()

将LamanEdgesList中具有重复边的元素加以合并，以精炼LamanEdgesList。注意，LamanEdgesList中的元素是利用IndependentEdges代换来的。所以，不会有两个方向的同一条边。

1. LamanEdgesList\_Temp = containers.ArrayList(k)。k根据LamanEdgesList中有效的元素确定。
2. 只要LamanEdgesList不为空，就一直循环下面的代码3-4。
3. 从LamanEdgesList中弹出一个表LamanEdge
4. 遍历LamanEdgesList中剩余的元素，设每个元素为Temp;
   1. 如果isempty(intersect(LamanEdge, Temp)) == false，说明两者有交集。
   2. 如果有交集。
      1. LamanEdgesList 中删除 Temp。
      2. Temp = union(LamanEdge, Temp)
      3. 将Temp压回到LamanEdgesList中。
   3. 如果没有交集，则将LamanEdge压入到LamanEdgesList\_Temp中。
5. LamanEdgesList执行删除方法。
6. LamanEdgesList=LamanEdgesList\_Temp

[] = StartIdentifyRigidCluster()

启动RigidCluster识别工作，主要为识别做数值准备。

初始化RigidCluster标识和颜色类。

为Graph. Edges表增加Cluster列。

为Graph. Edges表增加Color列。

为Graph. Nodes表增加Visited列。

1. 创建handle着色类。obj.RigidClusterEdgeColor = color.ContrastColor(). 其中含有RigidCluster的编号，对应边的颜色。
2. n= height (Graph.Edges).测出边的个数。
3. Graph.Edges.Cluster = ones(n,1)\*(-1)。为Graph. Edges表准备好Cluster列。

初始化为-1标示无效。

不应该出现0值。

其他整数为对应的Cluster归属。

1. Graph.Edges.Color = zeros(n,3)。为Graph. Edges表准备好Color列用于保存颜色。
2. n= height (Graph.Nodes).测出节点的个数。
3. Graph.Nodes.Visited = zeros(n,1)

(如果这里用false(n,1)，以后这个表就不能写入其他非逻辑数值)

False（0）表示还没有被访问过。

True（1）表示已经被访问过，且为rigid.

2(double)表示已经被访问过，且为floppy.

[IdentifyFinished] = IdentifyARigidCluster()

识别出一个RigidCluster。这个函数运行的前提是认为前面一个标记的RigidCluster完全探索完毕了。如果发现所有的边均被标记了，就会返回true，否则返回false.

1. n = height(Graph.Nodes)。得到节点个数。
2. ToVisitNodes = containers.ZipNumList(n). 生成待分析节点列表。
3. WriteAllVisited\_([1:1:n], false)。标记所有的节点均为没有访问过。
4. e= height(Graph.Edges)。得到边数。
5. IdentifyFinished = false. 生成RigidCluster
6. 遍历Graph.Edges.Cluster(i).
   1. 判断。Graph.Edges.Cluster == -1
   2. 如果找到了，就break. 得到WorkingEdgesID = i。
   3. IdentifyFinished = true
   4. 否则的话，IdentifyFinished = true
7. 判断。IdentifyFinished == true. 如果是，return。所有的边都判断完毕，没有必要继续运行。
8. Graph.Edges.
9. [WorkingNodeA, WorkingNodeB]=PebbleGame.Operation.Graph.findedge(WorkingEdgesID).提取与WorkingEdge相关联的WorkingNodeA和WorkingNodeB。

(为这条边标记RigidCluster数据)

1. RigidClusterEdgeColor.Flash()。生成一个新的编号和颜色。
2. WriteAllCluster\_([WorkingNodeA ; WorkingNodeB], RigidClusterEdgeColor.Counter)
3. WriteAllColour\_([WorkingNodeA ; WorkingNodeB], RigidClusterEdgeColor.ColorRGB)
4. WriteAllVisited\_([WorkingNodeA ; WorkingNodeB],true);

(提取WorkingNodeA邻居)

1. Neighbours WorkingNodeA = AllNeighbour(WorkingNodeA)
2. 遍历Neighbours WorkingNodeA(i)。
   1. 判断。IsVisited(NodeID)==false
   2. 如果是。ToVisitNodes.appendElement(Neighbours WorkingNodeA(i))

(提取WorkingNodeB邻居)

1. Neighbours WorkingNodeB = AllNeighbour(WorkingNodeB)
2. 遍历Neighbours WorkingNodeB(i)。
   1. 判断。IsVisited(NodeID)==false
   2. 如果是。ToVisitNodes.appendElement(Neighbours WorkingNodeB(i))

(精简化ToVisitNodes)

1. ToVisitNodes.ZipList()

(判断是否根本就没有邻居)

1. 判断ToVisitNodes. Count == 0
2. 如果是。IdentifyFinished = true. return.

(下面开始做一个超级迭代)

1. While(ToVisitNodes. Count ~=0)
   1. ExploreNode = ToVisitNodes.removeLast(). 首先弹出来一个。
   2. [Explore, NodeIDs] = ExploreARigidClusterNode([WorkingNodeA WorkingNodeB], ExploreNode)。
   3. 判断Explore==true
      1. 遍历NodeIDs(i), ANode
      2. NeighboursANode = AllNeighbour(NodeIDs(i))
         1. 遍历NeighboursANode(i)
         2. 判断。IsVisited(ANeighboursANode)==false
         3. 如果是。ToVisitNodes.appendElement(ANeighboursANode (i))
   4. ToVisitNodes. ZipList()。化简ToVisitNodes

(最后一步，开始标记)

1. 遍历IsVisited(ANeighboursANode)==true的节点，生成节点数组RigidNodes。
2. [] = WriteAllCluster\_(RigidNodes, RigidClusterEdgeColor.Counter)
3. []= WriteAllColour\_(RigidNodes, RigidClusterEdgeColor.ColorRGB)

[Explore, NodeIDs] = ExploreARigidClusterNode(Edge, ExploreNode)

判断ExploreNode相对于edge的性质，**暂时不管边的事情**。

Explore=true，发现了Rigid节点，则NodeIDs里面包含这些节点，也包含ExploreNode。

Explore=false，发现了floppy节点，则NodeIDs里面包含这些节点，也包含ExploreNode。

edge是已经被标记过属于现存的某一个RigidCluster的边。

1. 提取edge的两个点WorkingNodeA，WorkingNodeB。

(输入有效性检查)

1. IDIntheRange(WorkingNodeA)。数值可靠性检查。
2. IDIntheRange(WorkingNodeB)。数值可靠性检查。
3. IDIntheRange(ExploreNode)。数值可靠性检查。
4. EdgeisCovered([WorkingNodeA, WorkingNodeB]) == false。出错退出。“输入的edge在图上并不存在。”

(记录这个过程一共收集到了多少个FreePebble。)

CollectedFreePebbleNumber = 0;

（开始第一和第二泡泡收集过程）

（针对WorkingNodeA收集第一个泡泡。注意，即便WorkingNodeA本身就有FreePebble也是没有关系的。）

1. try 首先尝试在WorkingNodeA上收集第一个FreePebble。除非找不到FreePebble才会出错。
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeA)
   2. [] = RearrangePebble(PathtoFreePebble)
2. catch ME
3. msg = ['需要详细研究。为什么在节点',num2str(WorkingNodeA),'上一个自由泡泡也收集不到！'];
4. causeException = MException('MATLAB:myCode:IndependentEdge',msg)
5. ME = addCause(ME,causeException);
6. Explore = -1. 第一个泡泡都找不到的条件，还没有研究过。所以是无效值。
7. rethrow(ME) . 出错退出了。
8. end
9. [] = PinPebble(WorkingNodeA, 1)。如果能收集到，把WorkingNodeA上收集的这个泡泡Pin住。
10. CollectedFreePebbleNumber = CollectedFreePebbleNumber+1；

（针对WorkingNodeB收集第一个泡泡。注意，即便ExploreNode本身就有FreePebble也是没有关系的。）

1. try 首先尝试在WorkingNodeB上收集第一个FreePebble。除非找不到FreePebble才会出错。
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeB)
   2. [] = RearrangePebble(PathtoFreePebble)
2. catch ME
3. msg = ['需要详细研究。为什么在节点',num2str(WorkingNodeB),'上一个自由泡泡也找不到！'];
4. causeException = MException('MATLAB:myCode:IndependentEdge',msg)
5. ME = addCause(ME,causeException);
6. Explore = -2. 第二个泡泡都找不到的条件，还没有研究过。所以是无效值。
7. rethrow(ME)
8. end
9. [] = PinPebble(WorkingNodeB, 1), 如果找到了，把WorkingNodeB上收集的第二个泡泡也Pin住。
10. CollectedFreePebbleNumber = CollectedFreePebbleNumber+1；

（下面就可能出现找不到FreePebble的情况了。）

（在WorkingNodeA上第二次收集FreePebble。）

1. FailedA = false 在WorkingNodeA上第二次收集到泡泡是否失败的标志。
2. try
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeA)
   2. [] = RearrangePebble(PathtoFreePebble)
3. catch ME 发现在WorkingNodeA上面不能发现泡泡。
4. FailedA = true
5. end
6. 判断FailedA==false. 如果是，说明找到了FreePebble。
7. [] = PinPebble(WorkingNodeA, 1), 把WorkingNodeA上收集的第三个泡泡也Pin住。
8. CollectedFreePebbleNumber = CollectedFreePebbleNumber+1；
9. end

（在WorkingNodeB上第二次收集FreePebble。）

1. FailedB = false 在WorkingNodeB上第二次收集到泡泡是否失败的标志。
2. try
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(WorkingNodeB)
   2. [] = RearrangePebble(PathtoFreePebble)
3. catch ME 发现在WorkingNodeB上面不能发现泡泡。
4. FailedB = true
5. end
6. 判断FailedB==false. 如果是，说明找到了FreePebble。
7. [] = PinPebble(WorkingNodeB, 1), 把WorkingNodeB上收集的泡泡也Pin住。
8. CollectedFreePebbleNumber = CollectedFreePebbleNumber+1；
9. end
10. 判断CollectedFreePebbleNumber ~= 3。如果为真出错退出。
11. Explore = -3
12. “收集到了num2str(CollectedFreePebbleNumber)个泡泡，真是奇怪。”

(下面开始在ExploreNode上找第四个FreePebble)

FailedExploreNode = false

1. try
   1. [PathtoFreePebble, BreadthFirstSearch] = FindAPebble(ExploreNode)
   2. [] = RearrangePebble(PathtoFreePebble)
2. catch ME 发现在ExploreNode上面不能发现泡泡。
3. FailedExploreNode = true
4. end

（下面开始判断并输出结果）

1. 判断FailedExploreNode == true
2. NodeIDs = BreadthFirstSearch 失败的时候是返回广度优先算法的结果
3. Explore = true
4. WriteAllVisited\_(NodeIDs, true(1,1))
5. WriteAllVisited\_(ExploreNode, true(1,1))
6. 判断FailedExploreNode == false
7. NodeIDs = PathtoFreePebble 成功的时候是返回泡泡移动的过程。
8. Explore = false
9. WriteAllVisited\_(NodeIDs, 2)
10. WriteAllVisited\_(ExploreNode, 2)

(释放Pin住的泡泡)

1. 判断FailedA == true
2. UnPinPebble(WorkingNodeA, 1)
3. UnPinPebble(WorkingNodeB, 2)
4. 判断FailedB == true
5. UnPinPebble(WorkingNodeA, 2)
6. UnPinPebble(WorkingNodeB, 1)

[ClusterNumber] = ReadCluster(Edge)

表读取，读取边的RigidCluster归属信息。

% Edge一定是数值性的横向数组。

1. 提取Edge的两个点WorkingNodeA，WorkingNodeB。
2. IDIntheRange(WorkingNodeA)。数值可靠性检查。
3. IDIntheRange(WorkingNodeB)。数值可靠性检查。
4. edgeID = Graph.findedge(WorkingNodeA, WorkingNodeB)
5. 判断edgeID==0。如果是，则出错退出。“没有这么一条边。”
6. ClusterNumber = Graph.Edges.Cluster(edgeID)

[ColourInformation] = ReadColour(Edge)

表读取，读取边的RigidCluster相关联的颜色信息。

Edge一定是数值性的横向数组

1. 提取edge的两个点WorkingNodeA，WorkingNodeB。
2. IDIntheRange(WorkingNodeA)。数值可靠性检查。
3. IDIntheRange(WorkingNodeB)。数值可靠性检查。
4. edgeID = Graph.findedge(WorkingNodeA, WorkingNodeB)
5. 判断edgeID==0。如果是，则出错退出。“没有这么一条边。”
6. ColourInformation = Graph.Edges.Color (edgeID,:)

[] = WriteAllCluster\_(NodeIDs, ClusterNumber)

将NodeIDs中所包含的所有边的Cluster均设定为ClusterNumber。用于处理同时需要设定很多值的情况。

1. AEdge = Graph.Edges.EndNodes(i,:)。遍历EndNodes，设为AEdge（不要对表使用类似for each的语句）
   1. WorkingNodeA= AEdge(1,1); WorkingNodeB= AEdge(1,2);提取Edge的两个点。当前数据为cell。
   2. edgeID = Graph.findedge(WorkingNodeA, WorkingNodeB)。找到Edge的ID号。
   3. [WorkingNodeA, WorkingNodeB] = Graph.findedge(edgeID)。将cell结构的节点号转换为数字类型的节点号。
   4. 如果isequal(intersect(NodeIDs, WorkingNodeA),WorkingNodeA)，

同时isequal(intersect(NodeIDs, WorkingNodeB),WorkingNodeB)，

说明Edge这条边在NodeIDs中。不要用==判断，因为intersect会返回空矩阵。

* 1. 则Graph.Edges.Cluster(edgeID) = ClusterNumber。

[]= WriteAllColour\_(NodeIDs, ColourInformation)

将NodeIDs中所包含的所有边的Graph.Edges.Color()均设定为ColourInformation。用于处理同时需要设定很多值的情况。

1. AEdge = PebbleGame.Operation.Graph.Edges.EndNodes(i,:)。遍历EndNodes，设为AEdge（不要对表使用类似for each的语句）
   1. WorkingNodeA= AEdge(1,1); WorkingNodeB= AEdge(1,2);提取Edge的两个点。当前数据为cell。
   2. edgeID = Graph.findedge(WorkingNodeA, WorkingNodeB)。找到Edge的ID号。
   3. [WorkingNodeA, WorkingNodeB] = Graph.findedge(edgeID)。将cell结构的节点号转换为数字类型的节点号。
   4. 如果isequal(intersect(NodeIDs, WorkingNodeA),WorkingNodeA)，

同时isequal(intersect(NodeIDs, WorkingNodeB),WorkingNodeB)，

说明Edge这条边在NodeIDs中。不要用==判断，因为intersect会返回空矩阵。

* 1. Graph.Edges.Colour (edgeID,:) = ColourInformation。

[Neighbours] = AllNeighbour(NodeID)

返回NodeID的所有邻居。包括前向邻居和后向邻居。

注意这个是有向图，需要找到preIDs = predecessors(G, NodeID)，sucIDs = successors(G, NodeID)。然后合并在一起。

1. P = Graph.predecessors(NodeID)
2. S = Graph.successors(NodeID)
3. Neighbours = [P ; S]

[] = WriteAllVisited\_(NodeIDs, Logical)

遍历NodeIDs中的所有节点，将每一个节点的Visited均设定为Logical。用于处理同时需要设定很多值的情况。

False（0）表示还没有被访问过。

True（1）表示已经被访问过，且为rigid.

2(double)表示已经被访问过，且为floppy.

1. 遍历NodeIDs，设为NodeID。注意使用NodeIDs(i)语法。
2. Graph.Nodes. Visited = Logical

[Logical] = IsVisited(NodeID)

返回节点NodeID的Visited状态。

1. Logical = Graph.Nodes. Visited(NodeID)