MainProcess: contém o algoritmo principal que conduz os macro processos do sistema. Inclusive ela invoca a execução dos métodos do pacote parsesystem que contém os parses construído por intermédio do Javacc.
Campos (todos estáticos):

- `int iteration;`
  `número` da atual iteração, começando em zero.
- `SetQuerySparql currentSetQuerySparql`
  apenas um alias (para facilitar a escrita) de wholeSystem.getListSetQuerySparql().get(iteration)
- `SystemGraphData currentSystemGraphData`
  apenas um alias (para facilitar a escrita) de wholeSystem.getListSystemGraphData().get(iteration)
- `NodeData nodeDataWithLeastEccentricityAndAverage`
  armazena o nó com menor exentricidade e média das métricas (betweenness,closeness,eigenvector).
  apenas é usado na fase intermediária 3 (2º laço).
- `Node currentNode`
  armazena o nó (objeto da classe StreamGraph) atual que está sendo excluído. Usado na fase intermediária 3 (2º laço).
- `List<Edge> currentEdgeSet`
  armazena o conjunto de arestas (objetos da classe StreamGraph) pertencente ao nó que está sendo excluído. Usado na fase intermediária 3 (2º laço).
- `Concept currentConcept`
  contém o conceito equivalente ao nó atual que está sendo excluído. Usado na fase intermediária 3 (2º laço)

WholeSystem: formada apenas por campos estáticos, agrega todo o sistema de estrutura de dados. Contem para cada iteração (os dados já existentes se repetem): os rdfs dos termos da iteração, o grafo do GephiGraph atualizado para iteração. De forma única, para toda a execução do sistema, contém um objeto StreamGraph. Este último é sempre atualizado primeiramente, com os novos termos (1º laço) ou com exclusões (2º laço) para depois repassar as informações para o objeto GephiGraph específico para a atual iteração.
Essa dinâmica é devido ao fato do StreamGraph aceitar com mais facilidades, inserções com verificação de repetição, exclusões e ainda por ser uma biblioteca fácil de usar. Por outro lado, o GephiGraph foi usado para permitir o cálculo rápido para as métricas de rede (betweenness, closeness, eigencetor, eccentricity) e ainda cálculo do Connected Component (apesar desse algoritmo estar dando erro em alguns casos quando apenas um nó fica sozinho ele não é reconhecido como um componete conectado). Além disso o GephiGraph possui a classe AStar que permite o cálculo rápido do shortest path bem como indica os elementos do caminho. O GephiGraph toolkit também possui um gerador automático de arquivos GEXF (formato popular de grafos, porém, implementado pelo Gephi staff)
Campos (todos estáticos):

- `ConfigTable configTable`
  contém a tabela hashing de variáveis de configuração geral do sistema. A entrada é feita pelo arquivo config.txt no diretório base.
- `UselessConceptsTable uselessConceptsTable`
  contém a tabela hashing com os termos useless. Eles são desconsiderados na etapa da montagem do grafo (Stream Graph). Os seus RDFs são armazenados normalmente nos arquivos e também no objeto da classe SetQuerySparql.
- `RdfsFilesTable rdfsFilesTable`
  contém a tabela hashing com o nome dos termos arquivados no diretório. Cada nome

de termos é transformado para o nome de arquivo equivalente, trocando-se, espaço por underline, dois pontos por ponto e vírgula, maiúscula por ^ seguido da letra.

- `StreamGraphData streamGraphData`
  representa a network principal que contém os dados atualizados em cada iteração. Essa network é usada para criar em cada iteração
- `GroupConcept conceptsRegister`
  contém os conceitos originais e selecionados (são aqueles escolhidos por rankeamento de betweenness+closeness e eigenvector) em cada iteração. É um ArrayList e HashMap conjugados.
- `ConceptsGroup originalConcepts`
  preenchido única vez, logo no início do programa, em MainProcess.parseTerms(). Dados consultados com muita frequência. Como é estático agiliza a consulta.
- `int quantityOriginalConcepts`
  preenchido única vez, logo no início do programa, em MainProcess.parseTerms()
- `int quantityPathsBetweenOriginalConcetps`
  a quantidade de caminhos possíveis entre os termos base (qtdeTermos + qtdeTermos-1 + qtdeTermos-2 + ...). É preenchido única vez em MainProcess.parseTerms()
- *`EdgesTableHash edgesTable`*
  contém uma tabela única com todos os links usados, para evitar a repetição. Os classes GephiGraph e StreamGraph não aceitam repetição de id de links.
- `LinkedList<SetQuerySparql> listSetQuerySparql`
  a cada iteração registra o conjunto de queries e rdfs dos conceitos selecionados, além disso, recebe cópia dos conceitos usdos na interação anterior, desde os originais. Sendo assim, acumula a cada iteração os dados.
- `LinkedList<SystemGraphData> listSystemGraphData`
  a cada iteração, contém o grafo formado na classe GephiGraph bem como tabela de nodes com as suas informações e métricas calculadas.
- `int goalConceptsQuantity`
  quantidade meta de conceitos finais no mapa. Calculado em WholeSystem.initGoalmaxConceptsQualtity() e chamado em MainProcess.parseTerms()
- `int maxConceptsQuantity`
  quantidade máxima de conceitos finais no mapa. Calculado em WholeSystem.initGoalmaxConceptsQualtity() e chamado em MainProcess.parseTerms()
- `NodesTableArray sortAverageSelectedConcepts`
  tabela contendo nodes organizados pela média entre as 3 métricas calculadas (betweenness,closeness,eigenvector). Usada apenas na 3º fase intermediária do programa.
- `NodesTableArray sortEccentricityAndAverageSelectedConcepts`
  tabela contendo nodes organizados pela eccentricity + média entre as 3 métricas calculadas (betweenness,closeness,eigenvector).
  contém apenas os nós que foram selecionados ao longo das iterações.
- `NodesTableArray sortEccentricityAndAverageRemainingConcepts`
  tabela contendo nodes organizados pela média entre as 3 métricas calculadas (betweenness,closeness,eigenvector).
  contém todos os nós remanescentes na rede.
- `NodesTableArray finalHeadNodes`
  contém todos os nós que estão no caminho entre os termos seleciondados.
  é preenchido em MainProcess.buildFinalHeadNodesFromOriginalConceptsAndSelectedNodes

- `VocabularyTable vocabularyTable`
  tabela hash (HashMap) com vocabulário lido a partir de um arquivo de entrada (definido em Config.java).
- *`ConceptMap conceptMap`*
  mapa conceitual único. Preenchido na fase final do programa.

SystemGraphData: gera um objeto para cada iteração, colocado-o numa lista em WholeSystem.

- `GephiGraphData gephiGraphData`
  Contém a network da biblioteca org.gephi.graph.api.Graph e vários campos que fazem referência a elementos dessa própria biblioteca (AttributeModel, AttributeTable, AttributeColumn). Grafo da iteração atual.
- `NodesTableHash nodesTableHash`
  tabela hash usada para pesquisar mais rapidamente nodeData, da iteração atual
- `NodesTableArray nodesTableArray`
  armazena todos os nodes com suas informações da iteração atual
- `int connectedComponentsCount`
  quantidade de components concetados na iteração atual
- `Ranks ranks`
  contém as tabelas de ranks para cada grupo de nodes pertencentes a um componente conectado.
- `NodesTableArray betweennessSortTable`
  tabela ordenada com todos os nodes.
- `NodesTableArray closenessSortTable`
  tabela ordenada com todos os nodes.
- `NodesTableArray eccentricitySortTable`
  tabela ordenada com todos os nodes.
- `NodesTableArray eigenvectorSortTable`
  tabela ordenada com todos os nodes.

StreamGraphData: basicamente encapsula a classe Graph da biblioteca externa Graph Stream (pacote org.graphstream.graph).
Um único objeto é gerado e agregado como estático na classe WholeSystem.

GephiGraphData: basicamente encapsula a classe Graph do Graph Gephi Tookit (pacote org.gephi.graph).
Um objeto é gerado e agregado para cada objeto da classe SystemGraphData.

SetQuerySparql: gera um objeto para cada iteração, cada um contendo uma lista de QuerySpaql. A cada nova iteração há o acúmulo nos novos conceitos com os seus dados.
Obs.: em java apenas os ponteiros são repetidos, portanto não há copia de informações na memória.

QuerySparql: cada objeto representa uma query, o conceito associado e o conjunto de RDFs coletados no banco (ou nos arquivos)

Algoritmo principal:

***** Estágio inicial *****

- `start();`
  lê o arquivo config.txt, cria estrutura de diretórios para armazenar os resultados, inicializa os arquivos de Log

- `parseTerms(parser);`
  lê os termos fornecidos pelo usuário e os coloca em *`WholeSystem listSetQuerySparql`*

- parseUselessConcepts(parser);
  lê os termos useless e os coloca em *`WholeSystem uselessConceptsTable`*

- parseVocabulary(parser);
  lê os termos do vocabulário e os coloca em *`WholeSystem vocabularyTable`*

- readRdfsFileNameToRdfsFileTable();
  lê os nomes de arquivos que contém os RDFs já armazenados e os coloca em *`WholeSystem rdfsFileTable`*

***** Estágio das iterações *****

- do {
  - `indicateIterationNumber();`
    apenas coloca no log o número da iteração.
  - `updateCurrentSetQuerySparqlVar();`
    para facilitar a escrita, cria e atualiza a variável currentSetQuerySparql com WholeSystem.getListSetQuerySparql().get(iteration)
  - `assemblingQueries();`
    monta as queries para cada termo a partir de um modelo (arquivo em Config.java). Coloca elas em SetQuerySparql (no objeto criado na iteração).
  - `collectRDFsAllQueries();`
    coleta da base DBpedia os RDFs para cada termo e os coloca em SetQuerySparql (no objeto criado na iteração).
  - `removeConceptsWithZeroRdfs();`
    percorre a lista da classe SetQuerySparql e remove os objetos cujas quantidades de RDFs são zero.
  - `createCurrentSystemGraphData();`
    cria objeto SystemGraphData e o coloca na lista em WholeSystem.listSystemGraphData
    além disso, para facilitar a escrita, cria a variável currentSystemGraphData com WholeSystem.getListSystemGraphData().get(iteration)
  - `connectStreamVisualization();`
    verifica se é para exibir a network na janela StreamGraph e também no software Gephi, enquanto ele é construído.

- `buildStreamGraphData_buildEdgeTable_fromRdfs();`
  controi o grafo em WholeSystem.streamGraphData a partir dos RDFs da iteração atual (currentSetQuerySparql)

- `showQuantitiesStreamGraph();`
  apenas mostra no Log as quantidades de nós e arestas de streamGraph

- `if(iteration >= 1)`

  - `copyAllObjectsLastIteration();`
    se está na segunda iteração em diante, copia os termos+RDFs (querySparql) da iteração anterior

- `if(isApplyNDegreeFilterTrigger())`

  - `applyNDegreeFilterTrigger(WholeSystem.configTable.getInt("nDegreeFilter"), false);`
    dispara o gatilho para filtrar nós com grau menor do que N sobre o stremGraph a partir de um determinado número mínimo de iterações (config: iterationTriggerApplyNDegreeFilterAlgorithm) desde que ao mesmo tempo a quantidade de nós fosse um determinado valor mínimo (config: quantityNodesToApplyNdegreeFilter). N = nDegreeFilter (definido em config.txt)
    elimina os nós apenas de WholeSystem.streamGraph e de WholeSystem.conceptsRegister (se for o caso, ou seja, se ele é um conceito selecionado)
    nunca elimina um conceito original (fornecido pelo usuário).

- `buildGephiGraphData_NodesTableHash_NodesTableArray_fromStreamGraph();`
  copia todo o grafo de WholeSyste.streamGrapData para gephiGraphData da iteração atual
  além disso ele controi as tabelas com todos os nodes (também a partir do streamGraph)

- `clearStreamGraphSink();`
  apenas fecha a visualização do gephi, se for o caso.

- `classifyConnectedComponent_buildSubGraphs();`
  calcula, por intermédio da classe gephiGraph, os componentes concetados existentes e armazena na própria tabela interna do gephiGraph.
  atualiza a quantide de componentes conectados em systemGraphData.

- `calculateRelationshipLevelBetweenOriginalConcepts();`
  calcula o nível de relacionamento entre os conceitos originais, ou seja, a quantidade de camimhos quebrados. 0 = existem todos os caminhos entre os conceitos, 1 = faltou um caminho, e assim por diante. Se connected component = 1, não há necessidade de calcular.

- `buildGexfGraphFile();`
  cria um arquivo GEXF contendo o atual gephiGraph.

- `if(breakIteration())`

- `break:`
  `se:`
  ou: atingiu a quantidade máxima de iterações (config: maxIteration) – recomendável que o valor seja grande para forçar o estabelecimento das outras condições.
  ou: connected componente = 1 e ao mesmo tempo atingiu quantidade mínima de iterações (config: minIterationToVerifyUniqueConnectedComponent)
  ou: nível de relacionamento entre conceitos originais seja 100% e ao mesmo tempo atingiu o número mpinimo de iterações (config: minIterationToVerifyRelationshipBetweenOriginalConcepts).

- `calculateDistanceMeasuresWholeNetwork();`
  calcula as métricas de distância (betweenness e closeness) para toda a rede e (automaticamente) armazena em gephiGraphData

- `storeDistanceMeasuresWholeNetworkToMainNodeTable();`
  copia os resultados dos cálculos de gephiGraphData para a tabela de nodeData em SystemGraphData
  só pode ser chamado após `calculateDistanceMeasuresWholeNetwork()`
  (foi feito dessa forma para separar o cálculo que é inirente ao gephiGraphData do processo de armazenamento que é feito pelo systemGraphData)

- `calculateEigenvectorMeasureWholeNetwork();`
  calcula o eigenvector para toda a rede e (automaticamente) armazena em gephiGraphData

- `storeEigenvectorMeasuresWholeNetworkToMainNodeTable();`
  copia os resultados dos cálculos de gephiGraphData para a tabela de nodeData em SystemGraphData
  só pode ser chamado após `calculateDistanceMeasuresWholeNetwork()`
  (foi feito dessa forma para separar o cálculo que é inirente ao gephiGraphData do processo de armazenamento que é feito pelo systemGraphData)

- `sortMeasuresWholeNetwork();`
  cria três tabelas agregadas em systemGraphData contendo os nós de toda a rede ordenados por: betweenness, closeness e eigenvector.
  toma como base a tabela geral de nós em systemGraphData (nodesTableArray e nodesTableHash).

- `buildSubGraphsRanks();`
  constroi um gephiGraph para cada componente conectado e clona os nós do gephiGraph principal para cada um deles
  também registra o número do componentes concetado de cada nó na tabela geral de nós de systemGraphData.
  (talvez não tenha que existir esse gephiGraph para cada grupo de componente concetado... pois as medidas não serão calculadas... ver próxima função)

- `buildSubGraphsTablesInConnectedComponents();`
  constroi uma tabela de nós (nodesTableHash e nodesTableArray) para cada componente conectado.
  (foi desistido de calcular as métricas para cada subgrafo, já que isso iria interferir muito pouco no resultado final da seleção)

também cria para cada componente concetado um grupo com os conceitos originais.

- `sortConnectedComponentsRanks();`
  para cada componente conectado cria 4 tabelas ordenadas por: betweenness, closeness, eigenvector e betweenness+closeness.

- `selectLargestNodesByBetweennessCloseness();`
  seleciona os melhores Betweeness+closeness e os coloca em WholeSystem. conceptsRegister, registrando a iteração na qual eles foram selecionados.

- `selectLargestNodesByEigenvector();`
  seleciona os melhores eigenvector e os coloca em WholeSystem. conceptsRegister, registrando a iteração na qual eles foram selecionados.

- `reportSelectedNodesToNewIteration();`
  registra no log relação de conceitos selecionados por cada métrica etc.

- `if(WholeSystem.configTable.getBoolean("additionNewConceptWithoutCategory"))`

  - duplicateConceptsWithoutCategory(iteration);
    cria um conceito selecionado (que talvez nem exista) a partir do conceito "Category:…" retirando-se o sufixo relativo ao termo category.

- `prepareDataToNewIteration();`
  pega os conceitos dos nós selecionados (por betweenness+closeness e eigenvector) e os insere num novo objeto de SetQuerySparql.

  - Mais adiante, na próxima iteração, os conceitos antigos, bem como todos os seus dados, serão também inseridos nessa lista.
    Essa lista vai acumulando, a cada iteração, os conceitos antigos e novos.
    Porém, em Java, não há repetição de alocação de memória, pois o trabalho é sempre com ponteiros.

- `iteration++;`
  contagem das iterções, que começam em zero.

- `} While(true);`

***** Estágio intermediário 1 *****

- `int lastIterationWithinOfLoopWithDistanceMeasuresCalculation = iteration-1;`
  armazena o número da interação que teve o último calculo das métricas

- `indicateAlgorithmIntermediateStage1();`
  apenas apresenta no log a identificação do estagio atual do algoritmo

- `if(isApplyKCoreFilterTrigger()) {`
  se a quantidade de nós for maior do que quantityNodesToApplyKcoreFilter (definido em config.txt)

- `applyKCoreFilterTrigger(2, false);`
  dispara o gatilho de aplicação de k-core sobre o stremGraph, fazendo k = kCoreFilter (definido em config.txt)
  elimina os nós apenas do grafo WholeSystem.streamGraph e também de de WholeSystem.conceptsRegister (se for o caso, ou seja, se ele é um conceito selecionado)
  nunca elimina um conceito original (fornecido pelo usuário).
  O argumento false é para indicar que a verificação de 100% conceitos originais conectados não será feita (custosa e desnecessária poi a aplicação do k-core não quebra caminhos entre os conceitos originais)

- `calculateRelationshipLevelBetweenOriginalConcepts_allCases();`
  executa sem levar em consideração a quantidade de connected componentes (na função similar, ele só calcula se connected > 1)

- `iteration++;`
  `createCurrentSystemGraphData();`
  `buildGephiGraphData_NodesTableHash_NodesTableArray_fromStreamGraph();`
  `classifyConnectedComponent();`
  `buildGexfGraphFile(Time.t2_afterIterationAndKcore);`
  avança uma iteração, calcula novo connected component e registra novo grafo em arquivo GEXF

- `}`

***** Estágio intermediário 2 *****

- `indicateAlgorithmIntermediateStage2();`
  apenas apresenta novo estágio no log.

- `buildFinalHeadNodesFromOriginalConceptsAndSelectedConcepts(lastIterationWithinOfLoopWithDistanceMeasuresCalculation);`
  constroi a tabela de nós head que serão usados depois para a seleção dos nós que estão no shortest path.
  Esses nós podem ser provenientes de três fontes, de acordo com a configuração feita em config.txt: isSelected, isBetweennessCloseness ou isEigenvector

- `filterStreamGraphWithNodesAndEdgesBelongToShortestPathsOfFinalHeadNodes();`
  coleta todos os nós que estão no caminhos de todos os shortest path possíveis entre os nós cbeça selecionados no passo anterior.
  Filtra a rede streamGraph só deixando esses nós.

- `iteration++;`
  `createCurrentSystemGraphData();`
  avança na iteração

- `buildGephiGraphData_NodesTableHash_NodesTableArray_fromStreamGraph();`
  copia todo o grafo de WholeSyste.streamGrapData para gephiGraphData da iteração atual
  além disso ele controi as tabelas com todos os nodes (também a partir do streamGraph)

- `classifyConnectedComponent();`
  calcula, por intermédio da classe gephiGraph, os componentes concetados existentes e armazena na própria tabela interna do gephiGraph.
  atualiza a quantide de componentes conectados em systemGraphData.

- `buildGexfGraphFile(Time.t3_afterHeadNodesPaths);`
  criar o arquivo GEXF com o grafo contido em gephiGraph

`***** Estágio intermediário 3 *****`

- `indicateAlgorithmIntermediateStage3();`
  apenas indica o início do estágio intermediário 3 (destinado a remover a quantidade excessiva de nós no grafo)

- `iteration++;`
  `createCurrentSystemGraphData();`
  avança na iteração

- `buildGephiGraphData_NodesTableHash_NodesTableArray_fromStreamGraph();`
  copia todo o grafo de WholeSyste.streamGrapData para gephiGraphData da iteração atual
  além disso ele controi as tabelas com todos os nodes (também a partir do streamGraph)

- `calculateDistanceMeasuresWholeNetwork();`
  calcula as métricas de distância (betweenness e closeness) para toda a rede e (automaticamente) armazena em gephiGraphData

- `storeDistanceMeasuresWholeNetworkToMainNodeTable();`
  copia os resultados dos cálculos de gephiGraphData para a tabela de nodeData em SystemGraphData
  só pode ser chamado após calculateDistanceMeasuresWholeNetwork()
  (foi feito dessa forma para separar o cálculo que é inirente ao gephiGraphData do processo de armazenamento que é feito pelo systemGraphData)

- `calculateEigenvectorMeasureWholeNetwork();`
  calcula o eigenvector para toda a rede e (automaticamente) armazena em gephiGraphData

- `storeEigenvectorMeasuresWholeNetworkToMainNodeTable();`
  copia os resultados dos cálculos de gephiGraphData para a tabela de nodeData em SystemGraphData
  só pode ser chamado após calculateDistanceMeasuresWholeNetwork()
  (foi feito dessa forma para separar o cálculo que é inirente ao gephiGraphData do processo de armazenamento que é feito pelo systemGraphData)

- `classifyConnectedComponent_buildSubGraphs();`
  calcula, por intermédio da classe gephiGraph, os componentes concetados existentes e armazena na própria tabela interna do gephiGraph.
  atualiza a quantide de componentes conectados em systemGraphData.

- `createSortEccentricityAndAverageOnlyRemainingConcepts();`
  cria tabela dos nós remanescentes ordenados de forma descrescente em eccentricity e crescente pela média dos valores (betweeness, closeness e eigenvector)
  essa tabela é colocada em
  WholeSystem.sortEccentricityAndAverageRemainingConcepts

- `baseConnectedComponentCount = currentSystemGraphData.getConnectedComponentsCount();`
  armazana a quantidade de connected components para ter como base comparative na próxima fase, ou seja, não permitir que com as exclusões dos nós, a quantidade de connected componente cresça.

- `nodeDataPos = 0`
  variável usada para percorrer todos os nós da lista de candidatos a exclusão, ou seja, enquanto o connected componente estiver crescendo, aumenta o valora dessa vairável para tentar com o próximo.

- `while(WholeSystem.getSortEccentricityAndAverageRemainingConcepts().getCount() + WholeSystem.getQuantityOriginalConcepts() > WholeSystem.getGoalConceptsQuantity() && nodeDataPos < WholeSystem.getSortEccentricityAndAverageRemainingConcepts().getCount()) {`
  Enquanto a quantidade de nós do grafo ainda é maior do que a meta
  E enquanto ainda existirem nós candidatos para eclusão

  - `getNodeDataWithLeastEccentricityAndAverageFromRemainingConcepts();`
    pega o primeiro nó candidato a exclusão, ou seja, com o maior valor de eccentricity e menor valor da média das métricas de distância (betweenness, closeness e eigenvector)

  - `if(WholeSystem.configTable.getBoolean("isFixBugInGephiToolKit")) {`
    se a correção de bug do Gephi foi ligada… (erro em alguns casos onde existe um nó sozinho, e o connected componente é igual a 1, ao invés de 2)

    - `if(isCurrentNodeHasLinkWithAnEspecificOriginalConcept(WholeSystem.configTable.getString("originalConceptWithGephiToolKitBug"))) {`
      se o nó candidato a exclusão faz ligação com o nó a ser corrigido e se este nó só tem uma única aresta, então passa adiante para depois pegar o próximo candidato

      - `nodeDataPos++;continue;`

    - `}`

  - `}`

  - `if(WholeSystem.configTable.getBoolean("isKeepNeighborsOfOriginalConcepts")) {`
    se houver configuração para manter todas as arestas de um nó que representa um termo original, então despreza e passa adiante.

- if(isCurrentNodeHasLinkWithOriginalConcepts()) {
  - nodeDataPos++;
  - continue;
- }

- }

- storeCurrentInformationsAboutEnvironmentAndNodeWillBeDeleted();
  armazena as informações do ambiente para uma possível recuperação depois, caso a remoção do nó tenha aumentado a qtde de connected component

- currentConcept = WholeSystem.getConceptsRegister().getConcept(currentNode.getId());
  DeletedStatus deletedStatus = WholeSystem.getStreamGraphData().deleteNode(currentNode, false);
  remove o nó do streamgraph e guarda o seu conceito equivalente em currentConcept bem como o estado da operação em deletedStatus

- iteration++;
  createCurrentSystemGraphData();
  buildGephiGraphData_NodesTableHash_NodesTableArray_fromStreamGraph();
  classifyConnectedComponent();
  cria outro ambiente e gephiGraph a partir do streamgraph e calcula o novo valor de connected component

- if(currentSystemGraphData.getConnectedComponentsCount() > baseConnectedComponentCount) {
  se a qtde de connected component aumentou…
  - recoverEnvironmentAndNodeAndEdges(deletedStatus);
    nodeDataPos++;
    iteration--;
    recupera o nó e o ambiente.

- }

- else {
  se a qtde de connected component não aumentou...
  - calculateDistanceMeasuresWholeNetwork();
    storeDistanceMeasuresWholeNetworkToMainNodeTable();
    calculateEigenvectorMeasureWholeNetwork();
    storeEigenvectorMeasuresWholeNetworkToMainNodeTable();
    nodeDataPos = 0;
    createSortEccentricityAndAverageOnlyRemainingConcepts();
    prepara o próximo nó para a remoção, calculando as métricas e uma nova tabela de candidatos a partir da rede modificada (uma vez que um nó excluído, é necessário recalcular tudo pois ele pode influenciar nos valores de todos os outros nós)

o novo candidato é um nó que está novamente na posição zero da tabela de candidatos (nodePos=0)

- }

- }

- if(WholeSystem.getSortEccentricityAndAverageRemainingConcepts().getCount()
  + WholeSystem.getQuantityOriginalConcepts() <= WholeSystem.getGoalConceptsQuantity())
  Verifica se a meta de quantidade de nós para a rede foi atingida, ou se a quantidae de nós candidatos para a exclusão acabaram
  (nós candidatos = todos os nós da rede excluindo-se as exceções e nós que representam os conceitos originais)

  - Log.consoleln("- Goal achieved!!!");

- else {
  se a qtde de nós candidatos acabou sem a qtde meta de nós da rede tenha sido atingida, então vê se pelo menos a qtde está dentro da faixa limite, determinada pela variável de configuração conceptsMinMaxRange (arquivo config.txt)

  - if(WholeSystem.getSortEccentricityAndAverageRemainingConcepts().getCount() + WholeSystem.getQuantityOriginalConcepts() <= WholeSystem.getMaxConceptsQuantity())
    - Log.consoleln(" (However, it is less than the maximum "+WholeSystem.getMaxConceptsQuantity()+" nodes)");
  - else
    - Log.consoleln("- Goal did not achieve.");

- }

- reportAfterSelectionMainConcepts_remainingConcepts();
  simplesmente mostra no log um resumo do estado final da rede

***** Estágio final ***** (mapeamento da rede resultante para o mapa conceitual)

- indicateAlgorithmFinalStage();
  apenas inidica no log o estagio final

- buildGexfGraphFile(Time.t4_afterSteadyUniqueConnected);
  cria oGEXF representando o última rede no streamGraph

- showUselessConceptsStatistic();
  apresenta a estatísitica de uso dos termos useless

- buildRawConceptMapFromStreamGraph();
  controi o primeiro mapa conceitual (ainda em memória principal)

- `upgradeConceptMap_heuristic_01_removeLinkNumber();`
  `upgradeConceptMap_heuristic_02_vocabularyTable();`
  `upgradeConceptMap_heuristic_03_categoryInTargetConcept();`
  `upgradeConceptMap_heuristic_04_categoryInSourceConcept();`
  `upgradeConceptMap_heuristic_05_removeSelfReference();`
  `upgradeConceptMap_heuristic_06_createOriginalConceptsWithZeroDegree();`
  aplica as heurísticas sobre o mapa conceitual (ainda em format txt)

- `buildGexfGraphFileFromConceptMap();`
  cria um GEXF equivalente ao mapa conceitual após a aplicação de várias heurísitcas

- `buildTxtFileFromConceptMap();`
  constroi um arquivo txt a partir do mapa conceitual em memória principal

- `upgradeConceptMap_heuristic_07_putNewLineInCategory();`
  aplica a heristia especialmente destinada para melhora a formatação do arquivo CXL

- `buildCxlFileFromConceptMap();`
  cria o aquivo em formato CXL (aceito pelo CmapTools) do mapa conceitual resultante, contendo abstract;comment de conceitos indicados com borda mais grossa, e destaca a cor de fundo dos conceitos equivalentes aos termos originais (variáveis em config.txt: backGroundcolorOriginalConcept e borderThicknessConceptWithHint)

- `end();`
  fecha os arquivos de log.