Advanced Programming

Final term paper submission

Mattia Setzu, 544666

Master's Degree Project
Pisa, Italy June 2017

Implementation

The following implementation models the problem as following: *i)* nodes are modelled by *Node*, *ComputationalNode* and *InputNode*, the last one implementing pure values (i.e. pure matrices) too; *ii)* a node is a mapping from a name to a value (input) or an operation (computational), enriched by an input ordered set; *iii)* operations are disjoint from nodes, hence should be modelled in separate classes (*Operation*); *iv)* the code and graph generators (*PyFactory*, *GraphFactory*) receive well-formed graphs. We add the web references at the end of this introductory page [1].

Exercise 1

Design a hierarchy of classes to represent nodes and graph objects. The classes should provide suitable constructors for building nodes and graphs.

file: ./src/main/java/com.github.msetzu.pa.graph.Node.java

```
package com.github.msetzu.pa.graph;

import java.util.HashMap;

public abstract class Node<V> extends HashMap<String, V> {
 String name;
 private int m;
 private int n;

 int getM() { return m; }

 int getN() { return n; }

 protected Node(String name, int m, int n) { this.name = name; this.m = m; this.n = n; }

 public String getName() { return name; }
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.ComputationalNode.java

```
package com.github.msetzu.pa.graph;

import java.util.List;

public class ComputationalNode extends Node<Operation> {
 private final List<Node> inputs;

 public ComputationalNode(String name, Operation operation, List<Node> inputs) {
  super(name, -1, -1);
  put(name, operation);
  this.inputs = inputs;
 }

 public Operation operation() { return this.get(this.name); }

 public List<Node> inputs() { return inputs; }
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.InputNode.java

```
package com.github.msetzu.pa.graph;

import java.util.Optional;

public class InputNode extends Node<Optional<Matrix>> {
 public InputNode(String name, int m, int n, Optional<Matrix> val) {
  super(name, m, n);
```

[1] https://stackoverflow.com/questions/27677256/java-8-streams-to-find-the-duplicate-elements, http://json.org https://stackoverflow.com/questions/12643009/regular-expression-for-floating-point-numbers http://www.adam-bien.com/roller/abien/entry/java_8_streaming_a_string https://stackoverflow.com/questions/34879086/java-8-stream-api-filter-on-instance-and-cast

```
  this.put(name, val);
 }

 public Optional<Matrix> get() { return super.get(name); }
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.Matrix.java

```
package com.github.msetzu.pa.graph;

import java.util.Arrays;
import java.util.stream.IntStream;

public class Matrix {
 private final float[][] matrix;
 private final int m;
 private final int n;

 public Matrix(int m, int n) {
  this.matrix = new float[m][n]; this.m = m; this.n = n;
  IntStream.range(0,m).forEach(row -> Arrays.fill(matrix[row], 0));
 }

 public boolean equals(Object obj) {
  if (!(obj instanceof Matrix)) return false;
  Matrix other = (Matrix) obj;

  return (other.getM() == this.m && other.getN() == this.n) && IntStream.range(0, m)
    .allMatch(row -> Arrays.equals(this.matrix[row], other.matrix[row]));
 }

 private int getM() { return m; }

 private int getN() { return n; }

 public void set(int i, int j, float e) { matrix[i][j] = e; }
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.Operation.java

```
package com.github.msetzu.pa.graph;

public enum Operation {
 SUM, MUL;

 public static boolean isValid(Operation op, Integer[] lOp, Integer[] rOp) {
  switch (op) {case SUM: return lOp[0] == rOp[0] && lOp[1] == rOp[1]; default: return lOp[1] == rOp[0];}
 }

 public static Integer[] dimensions(Operation op, Integer[] lOp, Integer[] rOp) {
  switch (op) { case SUM: return lOp; default: return new Integer[]{lOp[0], rOp[1]}; }
 }
}
```

Exercise 2

Implement a recursive descent parser, which takes a JSON representation of a graph and builds the corresponding graph. For example, the above computational graph is represented as **[..]**.

```
{ "a": {"type": "input", "shape": [1,1]},
"b": {"type": "input", "shape": [1,1]},
"c": {"type": "comp", "op": "sum", "in": ["a", "b"]},
"d": {"type": "comp", "op": "sum", "in": ["b", [[1]]]},
"e": {"type": "comp", "op": "mult", "in": ["c", "d"]}}
```

Notice that not all the operations are possible, for example summing two vectors of different shapes. The Graph class should provide the methods: isDAG, and isValid to check whether all the variables are defined

and if all the operations are computable. For checking the compatibility, there should be a table specifying the signature for each function, for example:

```
sum: ([n, m], [n, m]) -> [n, m]
mult: ([n, m], [m, k]) -> [n, k]
```

De-coupling In order to de-couple the graph from the operations, we provide the validity constraints on the enumeration holding the operation themselves: we trivially check for dimension correspondence. An intermediate representation of the graph is stored in the map nodes in the form `name -> TYPECASE -> "INPUT|COMP", SHAPE -> "m,n", ...`. Such representation is then accessed by `GraphFactory` to generate the graph.

file: ./src/main/java/com.github.msetzu.pa.graph.json.Token.java

```
package com.github.msetzu.pa.graph.json;

public enum Token {
 O_BRACKET, C_BRACKET, O_SQ_BRACKET, C_SQ_BRACKET, SPACES, COMMA, COLON, END,
 NAME, FLOAT, INT, MATRIX, TYPE, TYPECASE, SHAPE, INPUT, COMP, OP, OPERATION;
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.Operation.java

```
package com.github.msetzu.pa.graph;

public enum Operation {
 SUM, MUL;

 public static boolean isValid(Operation op, Integer[] lOp, Integer[] rOp) {
  switch (op) {case SUM: return lOp[0] == rOp[0] && lOp[1] == rOp[1]; default: return lOp[1] == rOp[0];}
 }

 public static Integer[] dimensions(Operation op, Integer[] lOp, Integer[] rOp) {
  switch (op) { case SUM: return lOp; default: return new Integer[]{lOp[0], rOp[1]}; }
 }
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.json.JSONParser.java

```
package com.github.msetzu.pa.graph.json;

import com.github.msetzu.pa.graph.Operation;
import java.util.*;
import java.util.regex.*;

public class JSONParser {
 private static Pattern pat;
 private static Matcher mtc;
 private String json;
 private Map<String, Map<Token, String>> nodes = new HashMap<>();
 private Stack<String> names   = new Stack<>();
 private Stack<String> types   = new Stack<>();
 private Stack<String> shapes  = new Stack<>();
 private Stack<String> inputs  = new Stack<>();
 private Stack<Operation> operators  = new Stack<>();
 private Token lastActive = Token.C_BRACKET;    // Last active tokens
 private Optional<String> currentNode = Optional.empty(); // Active node

 public JSONParser(String json) { this.json = json; }

 public void parse() throws IllegalArgumentException {
  match(Token.O_BRACKET).matchSequence(Element.NODE).match(Token.C_BRACKET);
 }

 private JSONParser matchNode() throws IllegalArgumentException {
  match(Token.NAME).match(Token.COLON).match(Token.O_BRACKET).match(Token.TYPE).match(Token.TYPECASE);
  lastActive = Token.NAME;

  if (types.peek().contains("input"))
```

```
  return match(Token.COMMA).match(Token.SHAPE).match(Token.O_SQ_BRACKET).match(Token.INT)
    .match(Token.COMMA).match(Token.INT).match(Token.C_SQ_BRACKET).match(Token.C_BRACKET);
 else
  return match(Token.COMMA).match(Token.OP).match(Token.OPERATION).match(Token.O_SQ_BRACKET)
    .matchSequence(Element.COMBINED).match(Token.C_SQ_BRACKET).match(Token.C_BRACKET);
}

private JSONParser match(Token t) throws IllegalArgumentException {
 if (json.isEmpty() && t.equals(Token.C_BRACKET)) return this;

 String sp = " *"; String num = "0*[1-9]+(\\.[0-9]+)?";
 String vector = sp + "\\[" + sp + num + sp + "(," + sp + num + sp + ")*" + sp + "]" + sp;
 String matrix = sp + "\\[" + sp + vector + sp + "(," + sp + vector + sp + ")*" + sp + "]" + sp;

 switch (t) {
  case O_BRACKET: pat = Pattern.compile("^(" + sp + "\\{" + sp + ")"); break;
  case C_BRACKET: pat = Pattern.compile("^(" + sp + "}" + sp + ")"); break;
  case O_SQ_BRACKET: pat = Pattern.compile("^(" + sp + "\\[ *)"); break;
  case C_SQ_BRACKET: pat = Pattern.compile("^(" + sp + "]" + sp + ")"); break;
  case SPACES: pat = Pattern.compile("^(" + sp + ")"); break;
  case COMMA: pat = Pattern.compile("^(" + sp + "," + sp + ")"); break;
  case COLON: pat = Pattern.compile("^(" + sp + ":" + sp + ")"); break;
  case NAME: pat = Pattern.compile("^(" + sp + "\"[a-z]+\"" + sp + ")"); break;
  case FLOAT: pat = Pattern.compile("^(0*[1-9]+(\\.[1-9]+)?)"); break;
  case INT: pat = Pattern.compile("^(" + sp + "[0-9]+" + sp + ")"); break;
  case MATRIX: pat = Pattern.compile("^(" + matrix + ")"); break;
  case TYPE: pat = Pattern.compile("^(" + sp + "\"type\"" + sp + ":" + sp + ")"); break;
  case TYPECASE: pat = Pattern.compile("^(" + sp + "(\"input\")|(\"comp\"))"); break;
  case SHAPE: pat = Pattern.compile("^(" + sp + "\"shape\"" + sp + ":" + sp + ")"); break;
  case INPUT: pat = Pattern.compile("^(" + sp + "\"input\"" + sp + ":" + sp + ")"); break;
  case COMP: pat = Pattern.compile("^(" + sp + "\"comp\"" + sp + "," + sp + ")"); break;
  case OP: pat = Pattern.compile("^(" + sp + "\"op\"" + sp + ":" + sp + ")"); break;
  case OPERATION: pat = Pattern.compile("^(" + sp + "\"((mult)|(sum))\"" + sp + ","
         + sp + "\"in\"" + sp + ":" + sp + ")"); break;
 }
 mtc = pat.matcher(json);
 int index = mtc.find() ? mtc.start() : -1;

 if (index == -1) throw new IllegalArgumentException(t.name());
 String val = json.substring(mtc.start(), mtc.end());

 switch (t) {
  case C_BRACKET:
   if (lastActive.equals(Token.END) || names.size() == 0) return this;
   HashMap<Token, String> map = new HashMap<>();

   map.put(Token.NAME, names.peek());
   nodes.put(names.pop(), map);

   if (types.peek().contains("input")) {
    map.put(Token.TYPECASE, Token.INPUT.name().replace("\"",""));
    map.put(Token.SHAPE, shapes.get(shapes.size() - 2) + " " + shapes.peek());
    shapes.pop(); shapes.pop();
   } else {
    map.put(Token.TYPECASE, Token.COMP.name());
    map.put(Token.OP, operators.peek().name());
    map.put(Token.INPUT, inputs.stream().map(in -> in.replace("\"","") + "~").reduce("", String::concat));
    inputs = new Stack<>();
   }
   types.pop();
   currentNode = Optional.empty();
   lastActive = Token.C_BRACKET;
   break;
  case NAME:
   if (lastActive.equals(Token.NAME)) inputs.push(val);
   else names.push(val.replace("\"", ""));
```

```
    currentNode.ifPresent(v -> currentNode = Optional.of(v));
    break;
   case FLOAT: case MATRIX:
    inputs.push(val); break;
   case INT:   shapes.push(val); break;
   case TYPECASE:  types.push(val); break;
   case OPERATION: operators.push(val.contains("mul") ? Operation.MUL : Operation.SUM); break;
  }

  json = json.substring(mtc.end());
  return this;
 }

 private JSONParser matchSequence(Element e) throws IllegalArgumentException {
  switch (e) {
   case EPSILON: return match(Token.C_BRACKET);
   case NODE:
    try { return match(Token.C_BRACKET); }
    catch (IllegalArgumentException emptySequence) {
     matchNode();     // Sequence with >= 1 elements.
     try { return match(Token.COMMA).matchNode().match(Token.COMMA).matchSequence(Element.NODE); }
     catch (IllegalArgumentException nonEmptySequence) {
      switch (nonEmptySequence.getMessage()) {
       case "COMMA": lastActive = Token.END; return match(Token.C_BRACKET);
       case "NODE": throw nonEmptySequence; // Malformed JSON.
     }}
    }
   case COMBINED:
    try { return match(Token.MATRIX).match(Token.COMMA).matchSequence(Element.COMBINED); }
    catch (IllegalArgumentException exception) {
     switch (exception.getMessage()) {
      case "MATRIX":
       try { return match(Token.NAME).match(Token.COMMA).matchSequence(Element.COMBINED); }
       catch (IllegalArgumentException matrixException) {
        switch (matrixException.getMessage()) {
         case "NAME": throw matrixException; // Not a matrix, nor a name, must be malformed
         case "COMMA": return this;
      }}
      case "COMMA": return this;      // No comma, must be last element.
     }
    }}
  throw new IllegalArgumentException("SWITCH");
 }

 public Map<String, Map<Token, String>> getNodes() { return nodes; }
}

enum Element {EPSILON, NODE, COMBINED }
```

file: ./src/main/java/com.github.msetzu.pa.graph.json.factories.GraphFactory.java

```
package com.github.msetzu.pa.graph.json.factories;

import com.github.msetzu.pa.graph.*;
import com.github.msetzu.pa.graph.json.Token;
import static java.util.stream.Collectors.toList;
import static java.util.stream.Collectors.toSet;
import java.util.*;

public class GraphFactory {
 private Map<String, Node> nodes = new HashMap<>();
 private Map<String, Map<Token, String>> map;
 private final Map<String, Graph> graphs = new HashMap<>();

 GraphFactory(Map<String, Map<Token, String>> map) { this.map = map; }
```

```
public Optional<Graph> graph() throws IllegalArgumentException {
 try {
  map.keySet().forEach(this::node);
  return Optional.of(new Graph(roots(), childrenOf(roots())));
 } catch (ClassCastException e) { return Optional.empty(); }
}

private Optional<Graph> node(String nodeName) {
 if ((graphs.containsKey(nodeName))) return Optional.of(graphs.get(nodeName));
 else {
  Graph g;
  Node node;
  Map<Token, String> nodeMap = map.get(nodeName);
  switch (nodeMap == null ? "INPUT" : map.get(nodeName).get(Token.TYPECASE)) {
   case "INPUT":
    String[] shape = nodeMap == null ? new String[]{"-1", "-1"} : nodeMap.get(Token.SHAPE).split(" ");
    final int m = Integer.parseInt(shape[0]);
    final int n = Integer.parseInt(shape[1]);
    node = nodeMap == null  ? new InputNode(nodeName, m, n, Optional.of(matrixOf(nodeName)))
          : new InputNode(nodeName, m, n, Optional.empty());

    g = new Graph(node);
    nodes.put(nodeName, node);
    graphs.put(nodeName, g);
    return Optional.of(g);
   default:
    Operation op = nodeMap.get(Token.OP).equals("SUM") ? Operation.SUM : Operation.MUL;
    Set<String> childrenString = Arrays.stream(nodeMap.get(Token.INPUT).split("~")).collect(toSet());
    List<Graph> childrenGraphs = new ArrayList<>();
    List<String> vacantChildren = childrenString.stream().filter(c -> !(graphs.containsKey(c)))
             .collect(toList());
    List<String> builtChildren = childrenString.stream().filter(graphs::containsKey).collect(toList());

    builtChildren.forEach(child -> childrenGraphs.add(graphs.get(child)));
    vacantChildren.stream().map(this::node).filter(Optional::isPresent).map(Optional::get)
     .forEach(child -> {
      childrenGraphs.add(child);
      graphs.put(child.getRoots().get(0).getName(), child);
    });

    node = new ComputationalNode(nodeName, op, childrenString.stream().map(nodes::get).collect(toList()));
    nodes.put(nodeName, node);
    List<Node> roots = new ArrayList<>(); roots.add(node);
    HashMap<Node, List<Graph>> children = new HashMap<>();

    g = new Graph(roots, children);
    children.put(node, childrenGraphs);
    graphs.put(nodeName, g);
    return Optional.of(g);
 }}
}

private Matrix matrixOf(String nodeName) {
 int m = (int) nodeName.chars().mapToObj(i -> (char) i).filter(c -> c == ']').count() - 1;
 int n;
 List<Long> cols = Arrays.stream(nodeName.split("], *\\[")).map(s -> s.chars().mapToObj(i -> (char) i)
       .filter(c -> c == ',').count()).distinct().collect(toList());

 if (cols.size() > 1) throw new IllegalArgumentException();
 else n = Math.toIntExact(cols.get(0)) + 1;

 Matrix matrix = new Matrix(m, n);
 List<String[]> entries = Arrays.stream(nodeName.split("], *\\[")).map(s -> s.replace("[", "")
     .replace("]", "")).map(s -> s.split(",")).collect(toList());

 for (int i=0; i<m; i++) for (int j=0; j<n; j++) matrix.set(i, j, Float.parseFloat(entries.get(i)[j]));
```

```
  return matrix;
 }

 private List<Node> roots() {
  Map<Node, Boolean> bitMap = new HashMap<>();
  nodes.values().forEach(node -> bitMap.put(node, true));

  nodes.values().stream().filter(ComputationalNode.class::isInstance).map(ComputationalNode.class::cast)
      .forEach(node -> node.inputs().forEach(child -> bitMap.put(child, false)));

  return bitMap.keySet().stream().filter(bitMap::get).map(ComputationalNode.class::cast).collect(toList());
 }

 private Map<Node, List<Graph>> childrenOf(List<Node> roots) throws ClassCastException {
  Map<Node, List<Graph>> m = new HashMap<>();

  roots.stream().map(ComputationalNode.class::cast).forEach(root ->
   m.put(root, root.inputs().stream().map(Node::getName).map(graphs::get).collect(toList())));
  return m;
 }
}
```

EXERCISE 3

Use polymorphism to implement a compiler that transforms a graph into executable code in a programming language. Use a template for each function, that represents the code to be generated for the body of the function. The resulting code should correspond to a function, which takes as arguments variables corresponding to the inputs of the graph and returns an array with all the outputs of the graph. Provide the code generated for the example in the Introduction.

Polymorphism We built the program on a top-down approach: starting from the top we add the statements necessary to build the current node based on its operation: in order to do so we delegate to our `OperationCompiler` implementation to whom we provide the node. We abstract over the operations by providing a general `OperationCompiler` that acts as an hub for the various compilers.

file: ./src/main/java/com.github.msetzu.pa.graph.factories.PyFactory.java

```
package com.github.msetzu.pa.graph.factories;

import com.github.msetzu.pa.graph.*;
import com.github.msetzu.pa.graph.compilers.OperationCompiler;
import java.util.*;
import java.util.stream.IntStream;
import static java.util.stream.Collectors.toList;

public class PyFactory {
 private List<String> statements = new ArrayList<>();
 private List<String> graphParameters = new ArrayList<>();
 private StringBuilder py = new StringBuilder("");
 private Set<Node> generatedNodes = new HashSet<>();
 private OperationCompiler compiler = new OperationCompiler();

 public String generate(Graph g) {
  List<Node> roots = g.getRoots();
  roots.forEach(this::generate);

  py = py.append(header());
  IntStream.range(0, graphParameters.size()).forEach(i ->
      py = py.append("\t").append(graphParameters.get(i))
       .append(" = numpy.matrix(sys.argv[").append(i + 1).append("])\n"));
  // Assignments
  for (int i = statements.size() - 1; i >= 0; i--) py = py.append("\t").append(statements.get(i));
  // return
  py = py.append("\treturn ").append("[").append(seq(roots.stream().map(Node::getName)
    .collect(toList()))).append("]\n\n");
```

```
  return py.toString();
 }

 private void generate(Node node) {
  if (generatedNodes.contains(node)) return;
  String name = node.getName();

  if (node instanceof ComputationalNode) {
   ComputationalNode compNode = (ComputationalNode) node;
   List<Node> children = compNode.inputs();

   statements.add(assign(name, compiler.compile(compNode, children.stream().map(Node::getName)
      .collect(toList()))));
   children.forEach(this::generate);
  } else if (!(name.startsWith("["))) graphParameters.add(name);

  generatedNodes.add(node);
 }

 private String header() { return "import numpy\n\nif __name__ == \"__main__\":\n"; }

 private String assign(String name, String val) { return name + " = " + val + "\n"; }

 private String seq(List<String> elements) {
  String s = elements.stream().map(p -> p + ", ").reduce("", String::concat);
  return s.substring(0, s.length() - 2);
 }
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.compilers.OperationCompiler.java

```
package com.github.msetzu.pa.graph.compilers;

import com.github.msetzu.pa.graph.ComputationalNode;
import java.util.List;

// Compile the suited statement according to the operation
public class OperationCompiler {
 public String compile(ComputationalNode node, List<String> operands) {
  SumCompiler sumCompiler = new SumCompiler();
  MulCompiler mulCompiler = new MulCompiler();

  switch (node.operation()) {
   case SUM: return sumCompiler.compile(operands);
   default: return mulCompiler.compile(operands);
  }
 }

 String val(String raw) { return raw.startsWith("[") ? "numpy.matrix('" + raw + "')" : raw; }
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.compilers.SumCompiler.java

```
package com.github.msetzu.pa.graph.compilers;

import java.util.List;

class SumCompiler extends OperationCompiler {
 String compile(List<String> operands) {
  return operands.stream().reduce("", (a, b) -> val(a) + " + " + val(b)).substring(3);
 }
}
```

file: ./src/main/java/com.github.msetzu.pa.graph.compilers.MulCompiler.java

```
package com.github.msetzu.pa.graph.compilers;

import java.util.Collections;
```

```
import java.util.List;

class MulCompiler extends OperationCompiler {
 String compile(List<String> operands) {
  String mul = "numpy.matmul(";
  StringBuilder res = new StringBuilder("");
  res = res.append(operands.size() > 2 ? Collections.nCopies(operands.size(), mul)
    .stream().reduce(String::concat).get() : mul);
  res = res.append(val(operands.get(0))).append(", ").append(val(operands.get(1))).append(")");

  if (operands.size() > 2)
   res =  res.append(operands.subList(2, operands.size()).stream().map(o -> ", " + val(o) + ")")
     .reduce("", String::concat));

  return operands.size() > 2 ? res.substring(mul.length()) : res.toString();
 }
}
```

Generated code

```
import numpy

if __name__ == "__main__":
 a = numpy.matrix(sys.argv[1])
 b = numpy.matrix(sys.argv[2])
 d = b + numpy.matrix('[[1]]')
 c = a + b
 e = numpy.matmul(c, d)
 return [e]
```

Exercise 4

Extend the code generator in order to perform code optimizations. In particular, the generator should identify cases where multiple operations can be fused into one, for example: should be compiled into code that performs the addition a + b + c with a single nested loop. *Hint: use a template to represent the transformation.*

Algorithm We implement a top-down optimization algorithm similar to the processing phase of the Dijkstra min-distance: starting from the top, we split the lower nodes in two sections: boundary and not boundary, that is the immediate children and not immediate children. Then we merge the boundary nodes through `merge(node, graph)` whenever possible and advance the boundary recursively until we reach the bottom (the input nodes).

file: ./src/main/java/com/github/msetzu/pa/graph/optimizers/Optimizer.java

```
package com.github.msetzu.pa.graph.optimizers;

import com.github.msetzu.pa.graph.*;

import java.util.*;

import static java.util.stream.Collectors.toList;
import static java.util.stream.Collectors.toSet;

import java.util.stream.Stream;

public class Optimizer {
 public Graph optimize(Graph g) {
  g.getRoots().forEach(root -> optimize(root, g, g));
  return g;
 }

 private void optimize(Node node, Graph nodeGraph, Graph g) {
  if (node instanceof InputNode) return;

  ArrayList<Graph> boundaryGraphs = new ArrayList<>();
  ComputationalNode compNode = (ComputationalNode) node;
```

```
  Iterator<Graph> i = nodeGraph.childrenOf(node).stream().filter(child -> child.getRoots().get(0)
       instanceof ComputationalNode).iterator();

  while (i.hasNext()) {
   Graph child = i.next();
   Node childRoot = child.getRoots().get(0);
   ComputationalNode compChild = (ComputationalNode) childRoot;

   if (!(compChild.operation().equals((compNode.operation())))) {
    boundaryGraphs.add(child);
   } else if (parents(childRoot, g, new HashSet<>()).size() == 1) {
    int pos = compNode.inputs().indexOf(childRoot);

    compNode.inputs().remove(pos);      // Remove old child
    compNode.inputs().addAll(pos, compChild.inputs()); // Replace with its children
    nodeGraph.getChildren().get(compNode).remove(pos);       // Remove old graph
    nodeGraph.getChildren().put(compNode, merge(node, nodeGraph, child, pos)); // Replace with its children
   }
   boundaryGraphs.forEach(boundary -> optimize(childRoot, child, g));
  }
 }

 private List<Graph> merge(Node parent, Graph graph, Graph child, int position) {
  Node childRoot = child.getRoots().get(0);
  List<Graph> newChildren = graph.getChildren().get(parent);
  List<Graph> previousChildren = newChildren.subList(0, position);
  List<Graph> followingChildren = newChildren.subList(position, newChildren.size());

  return Stream.of(previousChildren, child.childrenOf(childRoot), followingChildren)
       .flatMap(List::stream).collect(toList());
 }

 private Set<ComputationalNode> parents(Node n, Graph g, Set<ComputationalNode> parents) {
  parents.addAll(g.getRoots().stream().filter(ComputationalNode.class::isInstance)
       .map(ComputationalNode.class::cast).filter(r -> r.inputs().contains(n))
       .collect(toList()));
  g.getRoots().stream().filter(ComputationalNode.class::isInstance).map(ComputationalNode.class::cast)
      .forEach(r -> parents.addAll(g.childrenOf(r).stream().map(c -> parents(n, c, parents))
      .flatMap(Set::stream).collect(toSet())));
  return parents;
 }
}
```

EXERCISE 5

C++ templates *C++* allows the definition of dependent types, i.e. types defined w.r.t. parametric values. This technique is aimed at generating suitable code (at least) partially evaluated at compile time. By inserting statically-known values to the template we leverage the compile-time speed and provide a concrete model on which the compiler can optimize through static analysis. It is then trivial to show how this technique comes in handy the closer the input values are to the template ones: the lesser the distance, the lesser the run-time computation. On the other hand, to find the model closest to the run time inputs is often a non-trivial task, and lowers the generality of the overall program.

LINQ expression trees LINQ is a *.NET* component exploiting expression trees to abstract over structured data modelled in a non-Object Oriented fashion, providing easier integration. Common data models are xml files, databases and objects. Possible expressions include `select`, `where`, `sum`, `avg`, `join` and `foreach` to allow iteration. Set operators are provided, such as `intersect`, `union`, etc. Operators are composed and built through lambda expressions implemented with the parametric type `Expression<T>` and stored by the platform in efficient in-memory data structures, the expression trees. Then it is trivial to compute over the above data models through the abstraction layer provided by *.NET*. Moreover, *LINQ* is enriched by a type-inference tool, hence is able to infer and assign types to expression to/from the supported data models. By typing structured untyped data we are able to guarantee some constraints on type safety on the input data. We may also declare new structures and create anonymous ones to use in throw-away computations: such anonymous types are also automatically enriched with common methods (get, set, etc.)