1.9 Anexo I

Código en Java del proceso usado para calcular los pesos de los candidatos para todo el corpus:

```
public void sumarTodosDocumentos(){

        for ( Object documento :
this.documentos.values().toArray()){

            ArrayList<CandidatoData> candidatosDoc
=((DocumentData)documento).getDatosCandidatos();
            for(CandidatoData c:candidatosDoc){
                if
(this.conceptosClaveObjetos.containsKey(c.getPalabra())){
                    CandidatoData
concepto=(CandidatoData)(this.conceptosClaveObjetos.get(c.getP
alabra()));

concepto.setPeso(c.getPeso()+concepto.getPeso());
                }else{

                    CandidatoData nuevoConcepto= new
CandidatoData(c.getPalabra());
                    nuevoConcepto.setPeso(c.getPeso());

this.conceptosClaveObjetos.put(c.getPalabra(), nuevoConcepto);
                    this.conceptosClave.add(nuevoConcepto);
                }
            }
        }

        Collections.sort(this.conceptosClave);
    }
```

1.10 Anexo J

Código en Java del proceso usado para la obtención de los conceptos clave y número de documentos en donde aparecen:

```
public HashMap<String, NodoData> obtenerNodosCorpus()throws
Exception{

        Connection con= ConexionBD.conn;
        String sql= "";
        PreparedStatement pstmt= null;
        ResultSet rset=null ;
        HashMap <String, NodoData> conceptos= new HashMap();
        try{
            sql=" SELECT A.TEXTO,COUNT(B.IDDOCUMENTO) FROM
TESIS.CONCEPTOS_CLAVE A, TESIS.CANDIDATO B "
              + " WHERE A.IDCORPUS=? AND A.TEXTO=B.TEXTO AND
A.IDCORPUS=B.IDCORPUS "
              + " GROUP BY A.TEXTO ORDER BY 2 DESC ";

            pstmt= con.prepareStatement(sql);
            pstmt.setInt(1, this.idCorpus);
            rset=pstmt.executeQuery();
            while (rset.next()){
                NodoData nuevoConcepto= new NodoData();
                nuevoConcepto.setTexto(rset.getString(1));
                nuevoConcepto.setAparicionDocumentos(rset.getInt(2))
;

                conceptos.put(rset.getString(1), nuevoConcepto);
            }
        }catch(Exception e){
            System.out.println("GRAVE ERROR: " +e.getMessage());
            throw e;
        }finally{
          if (pstmt!=null) pstmt.close();
          if (rset!=null)  rset.close();
        }
        this.conceptosNodo=conceptos;
        return conceptos;
    }
```

1.11 Anexo K

Código en Java del proceso usado para la obtención de pares de conceptos clave y número de documentos en donde aparecen:

```
public  HashMap<String,ArrayList<RelacionJerarquiaData>>
obtenerRelacionPosiblesPadres()throws Exception{
        Connection con= ConexionBD.conn;
        String sql= "";
        PreparedStatement pstmt= null;
        ResultSet rset=null ;
        ArrayList<RelacionJerarquiaData> posiblesPadres= new ArrayList();
        HashMap<String,ArrayList<RelacionJerarquiaData>>
relacionesPosiblesPadres= new HashMap();
        try{
            sql=" SELECT B.TEXTO, D.TEXTO, COUNT(DISTINCT B.IDDOCUMENTO) "
              + " FROM TESIS.CONCEPTOS_CLAVE A, TESIS.CANDIDATO B,
TESIS.CONCEPTOS_CLAVE C, TESIS.CANDIDATO D "
              + " WHERE A.IDCORPUS = ? "
              + " AND A.TEXTO=B.TEXTO AND A.IDCORPUS= B.IDCORPUS AND
C.TEXTO=D.TEXTO AND C.IDCORPUS= D.IDCORPUS "
              + " AND A.IDCORPUS=C.IDCORPUS AND B.IDDOCUMENTO=D.IDDOCUMENTO
AND NOT B.TEXTO = D.TEXTO "
              + " GROUP BY B.TEXTO, D.TEXTO ORDER BY 1,2 ";
            pstmt= con.prepareStatement(sql);
            pstmt.setInt(1, this.idCorpus);
            rset=pstmt.executeQuery();
            String actual="";
            String padre="";
            String hijo="";
            while (rset.next()){
                padre=rset.getString(2);
                hijo=rset.getString(1);
                if (!actual.equals(padre) && !actual.trim().equals("")){
                    relacionesPosiblesPadres.put(hijo, posiblesPadres);
                    posiblesPadres= new ArrayList();
                    actual=padre;
                }
                RelacionJerarquiaData relacionData = new
RelacionJerarquiaData();
                relacionData.setNodoPadre(this.conceptosNodo.get(padre));
                relacionData.setNodoHijo(this.conceptosNodo.get(hijo));
                relacionData.setAparicionDocumentos(rset.getInt(3));
                if (relacionData.verificaRelacion())
                    posiblesPadres.add(relacionData);
            }
            if (!posiblesPadres.isEmpty())
relacionesPosiblesPadres.put(hijo, posiblesPadres);
        }catch(Exception e){
            System.out.println("GRAVE ERROR: " +e.getMessage());
            throw e;
        }finally{
          if (pstmt!=null) pstmt.close();
          if (rset!=null)  rset.close();
        }
        return relacionesPosiblesPadres;
    }
```

1.12 Anexo L

Código en Java del proceso que verifica si los pares de conceptos clave conforman una relación de posible parentesco:

```
public boolean verificaRelacion(){

        boolean esRelacionValida=false;

        calculaProbabilidad();

        //x padre de y
        esRelacionValida = (pXY>=Constantes.benchmark) &&
(pYX<Constantes.benchmark);
        return esRelacionValida;
    }

    public void calculaProbabilidad(){

        this.setpXY((double)((double)this.aparicionDocumentos/(doubl
e)this.nodoHijo.getAparicionDocumentos()));
        this.setpYX((double)((double)this.aparicionDocumentos/(doubl
e)this.nodoPadre.getAparicionDocumentos()));

    }
```

1.13 Anexo M

Código en Java del proceso que identifica el mejor padre de un nodo:

```
public void identificaPadreNodo(NodoData nodo)throws Exception{
        //Si ya tiene padre
        if (nodo.getNodoPadre()!=null) return;
        //Si es el nodo raiz
        if (nodo.getAparicionDocumentos()==-100) return;
        //Si no tiene padre posible
        if (!this.posiblesPadres.containsKey(nodo.getTexto())){
nodo.setNodoPadre(nodoRaiz);nodoRaiz.agregaNodoHijo(nodo);return;}
        ArrayList<RelacionJerarquiaData> relacionesPadre=
this.posiblesPadres.get(nodo.getTexto());
        if (relacionesPadre==null || relacionesPadre.isEmpty()){
nodo.setNodoPadre(nodoRaiz);nodoRaiz.agregaNodoHijo(nodo);return ;}
        //Caso normal
        double maxScore=0;
        RelacionJerarquiaData relacionPadre=null;
        for (RelacionJerarquiaData r:relacionesPadre){
            double score=this.calculaScore(r,1);
            if (score>maxScore){
                relacionPadre=r;
                maxScore=score;
            }
        }
        this.getRelacionesPadres().put(nodo.getTexto(),
relacionPadre);
        nodo.setNodoPadre(relacionPadre.getNodoPadre());
        relacionPadre.getNodoPadre().agregaNodoHijo(nodo);
    }
```

1.14 Anexo N

Código en Java del proceso que obtiene el puntaje de una posible relación de parentesco:

```
public double calculaScore(RelacionJerarquiaData relacion, int
distancia )throws  Exception{
        double score=0;

        if (relacion.getNodoPadre()==null) return 0;
        if (relacion.getNodoPadre().getAparicionDocumentos()==-100)
return 0;
        if (relacion.getNodoPadre().getNodoPadre()==null)
identificaPadreNodo(relacion.getNodoPadre()) ;

        RelacionJerarquiaData relaciontemp= new
RelacionJerarquiaData();
        relaciontemp.setNodoHijo(relacion.getNodoHijo());
        relaciontemp.setNodoPadre(relacion.getNodoPadre().getNodoPad
re());
        relaciontemp.obtenerAparicionDocumento(this.corpus.getIdCorp
us());
        relaciontemp.calculaProbabilidad();
        score= calculaScore(relaciontemp, distancia+1) +
relacion.getpXY();

        return score;
    }
```