

## 7

**Referências bibliográficas**

- 1 LIU, L.; LIU, L.; LI, Y.; et al. Comparison of Next-Generation Sequencing Systems Comparison of Next-Generation Sequencing Systems. , , n. July, 2012.
- 2 HENSON, J.; TISCHLER, G.; NING, Z. Next-generation sequencing and large genome assemblies Review. , p. 901–915, 2012.
- 3 ALKAN, C.; SAJJADIAN, S.; EICHLER, E. E. Limitations of next-generation genome sequence assembly. , v. 8, n. 1, p. 61–65, 2011.
- 4 ZERBINO, D. R.; BIRNEY, E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. Genome Research, v. 18, n. 5, p. 821–829, 2008.
- 5 LI, Y.; KAMOUSHI, P.; HAN, F.; et al. Memory efficient minimum substring partitioning. Proceedings of the VLDB Endowment, v. 6, n. 3, p. 169–180, 2013.
- 6 GEORGANAS, E.; BULUÇ, A.; CHAPMAN, J.; et al. Parallel de Bruijn Graph Construction and Traversal for de Novo Genome Assembly. International Conference for High Performance Computing, Networking, Storage and Analysis, SC. Anais... . v. 2015-Janua, p.437–448, 2014.
- 7 QUITZAU, J.; STOYE, J. A space efficient representation for sparse de Bruijn subgraphs. bieson.ub.uni-bielefeld.de, 2008
- 8 SURGET-GROBA, Y.; MONTOYA-BURGOS, J. I. Optimization of de novo transcriptome assembly from next-generation sequencing data. Genome research, v. 20, n. 10, p. 1432–40, 2010.
- 9 LI, R.; ZHU, H.; RUAN, J.; et al. De novo assembly of human genomes with massively parallel short read sequencing. Genome research, v. 20, n. 2, p. 265–72, 2010.
- 10 WETTERSTRAND, K. A. DNA Sequencing Costs: Data from the NHGRI Large-Scale Genome Sequencing Program., , n. 2, , 2016.
- 11 STROKES, JON, Understanding Moore's Law. Ars Technica. 2008 Disponível em: <<http://arstechnica.com/gadgets/2008/09/moore/>> Acesso em: 01 de jan. 2016.
- 12 COOK, J. J.; ZILLES, C. Characterizing and optimizing the memory footprint of de novo short read DNA sequence assembly. ISPASS 2009 - International Symposium on Performance Analysis of Systems and Software, p. 143–152, 2009.

- 13 FANG, S.-C.; WANG, Y.; ZHONG, J. A Genetic Algorithm Approach to Solving DNA Fragment Assembly Problem. *Journal of Computational and Theoretical Nanoscience*, v. 2, n. 4, p. 499–505, 2005.
- 14 NAGARAJAN, N.; POP, M. Sequence assembly demystified. *Nature reviews. Genetics*, v. 14, n. 3, p. 157–67, 2013. Nature Publishing Group.
- 15 COOK, JEFFREY J., Scaling short read de novo DNA sequence assembly to gigabase genomes Ph.D. Thesis, Urbana, Illinois, 2011.
- 16 ZERBINO, D. R. Genome assembly and comparison using de Bruijn graphs. *Molecular Biology*, p. 149, 2009
- 17 COMPEAU, P. E. C.; PEVZNER, P. A; TESLER, G. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, v. 29, n. 11, p. 987–91, 2011.
- 18 FLICEK, P.; BIRNEY, E. Sense from sequence reads: methods for alignment and assembly. *Nature methods*, v. 6, n. 11 Suppl, p. S6–S12, 2009
- 19 STERKY, F.; LUNDEBERG, J. Sequence analysis of genes and genomes. *Journal of biotechnology*, v. 76, p. 1–31, 2000.
- 20 SLEATOR, D. D.; TARJAN, R. E. Self-adjusting Binary Search Trees. *Journal of the ACM*, v. 32, n. 3, p. 652–686, 1985.

## 8

**Anexo I – Código das funções PostgreSQL**

Este anexo contém o código fonte das funções utilizadas pela implementação VelvetH-DB.

**8.1.****Maestra**

```
CREATE OR REPLACE FUNCTION estudol.maestra(file text, k integer,
rangebegin integer, rangefinish integer, step integer)
  RETURNS integer AS
  $BODY$
    DECLARE
      readcount integer;
      tablename text;
      distintKmerNum integer;
      start_time timestamp;
      end_time timestamp;
    BEGIN
      readcount := 0;

      SELECT timeofday() into start_time;

      SELECT estudol.importaaquivo(file) INTO readcount;

      SELECT estudol.processaroadmap(k , rangebegin, rangefinish,
step) INTO distintKmerNum;

      SELECT estudol.geraoutput() INTO readcount;

      SELECT timeofday() into end_time;

      RAISE NOTICE 'Program start time: %', start_time;
      RAISE NOTICE 'Program End time : %', end_time;

      RETURN readcount;
    END;
  $BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION estudol.maestra(text, integer, integer,
integer, integer)
  OWNER TO postgres;
```

## 8.2. Importa Arquivo

```
CREATE OR REPLACE FUNCTION "estudol".importaarquivo(file text)
  RETURNS integer AS
$BODY$
  DECLARE
    readcount text;
    tablename text;
    tablepk text;
    sequencetablename text;

  BEGIN
    file := quote_literal(file);

    EXECUTE 'COPY estudol.sequences (name, dummy, cat, read) FROM
'|| file;
    SELECT COUNT(*) FROM estudol.sequences INTO readcount;
    RAISE NOTICE '% sequences imported', readcount;
    -----
    INSERT INTO estudol.presequences (SELECT replace(read,'N','A')
preread, ID FROM estudol.sequences);
    RAISE NOTICE '%', 'Nucleotideos trocados de N para A';

    RETURN readcount;
  END;
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
```

### 8.3. Processa Roadmap

```

CREATE OR REPLACE FUNCTION "estudol".ProcessaRoadmap(k integer,
rangebegin integer, rangefinish integer, step integer)
  RETURNS integer AS
$BODY$
  DECLARE
    distinctKmersNumb integer;
    sequenceRow RECORD;
    kmerRow RECORD;
    kmerString text;
    explainresult text;
    explain_step int;

    kmerfound boolean;
    annotationClosed          boolean;
    referenceSeqIdAux          integer;
    referenceSeqStartPosAux    integer;
    writeNucleotideIndex      integer;
    readNucleotideIndex       integer;

    referenceSeqId            integer;
    pos                       integer;
    referenceSeqStartPos      integer;
    finish                    integer;
    orderNumb                 integer;
    m integer;

    sequencetable text;
    identificationtable text;
    roadmactable text;
    identificationtablepk text;
    presequencetable text;

    booleanNum              integer;
    mFOUND                   boolean;

    start_time timestamp;
    end_time timestamp;
    sequence_process_start_time timestamp;
    sequence_process_finish_time timestamp;
    kmer_process_start_time timestamp;
    kmer_process_finish_time timestamp;
    kmer_exists_start_time timestamp;
    kmer_exists_finish_time timestamp;
    kmer_insert_start_time timestamp;
    kmer_insert_finish_time timestamp;
    kmer_insert_time interval;
    insert_annotation_start_time timestamp;
    insert_annotation_finish_time timestamp;
    id_stats_exec integer;

  BEGIN

    m = 79;
    explain_step := step;
    id_stats_exec := 0;
    SELECT clock_timestamp() into start_time;

    distinctKmersNumb := 0;

```

```

    FOR sequenceRow IN SELECT * FROM estudol.presequences WHERE
id>=rangebegin AND id<=rangefinish ORDER BY id LOOP

        annotationClosed :=true;
        writeNucleotideIndex := 0;
        pos:=0;
        referenceSeqStartPos:=0;
        finish :=0;

        orderNumb :=0;

        FOR readNucleotideIndex IN 1..m-k+1 LOOP

            kmerString := substr(sequenceRow.read, readNucleotideIndex,
k);
            kmerfound := false;

            SELECT * INTO kmerRow FROM estudol.identification WHERE kmer =
kmerString ;

            GET DIAGNOSTICS booleanNum = ROW_COUNT;

            IF booleanNum > 0 THEN
                mFOUND = true;
            ELSE
                mFOUND = false;
            END IF;

            IF NOT mFOUND THEN
                kmerfound := false;

                INSERT INTO estudol.identification (kmer,"IDInt", read,
pos) VALUES (kmerString
,distinctKmersNumb,sequenceRow.id,writeNucleotideIndex);

                distinctKmersNumb := distinctKmersNumb + 1;
            ELSE
                kmerfound := true;
                referenceSeqIdAux := kmerRow.read;
                referenceSeqStartPosAux := kmerRow.pos;
            END IF;

            IF kmerfound = false THEN
                writeNucleotideIndex:= writeNucleotideIndex + 1;
                IF annotationClosed = false THEN

                    INSERT INTO estudol.roadmap (read, "overlappedRead",
"posInRead", "posInOverlappedRead", "numbOfContinuosKmerOverlapped",
"annotationOrder")
                        VALUES
(sequenceRow.id,referenceSeqId,pos,referenceSeqStartPos,finish,orderNu
mb );

                    orderNumb := orderNumb + 1;

                END IF;

                annotationClosed :=true;
            ELSE

                IF annotationClosed = true THEN
                    referenceSeqId := referenceSeqIdAux;
                    pos :=writeNucleotideIndex;
                    referenceSeqStartPos := referenceSeqStartPosAux;
                    finish :=referenceSeqStartPosAux;
                    finish := finish + 1;
                    annotationClosed := false;
                ELSE

```

```

        IF annotationClosed = true THEN
            referenceSeqId := referenceSeqIdAux;
            pos :=writeNucleotideIndex;
            referenceSeqStartPos := referenceSeqStartPosAux;
            finish :=referenceSeqStartPosAux;
            finish := finish + 1;
            annotationClosed := false;
        ELSE
            IF referenceSeqIdAux = referenceSeqId AND
referenceSeqStartPosAux = finish THEN --continuous repetition of a
previously annotated kmer
                finish := finish + 1;
            ELSE
                INSERT INTO estudo1.roadmap (read, "overlappedRead",
"posInRead", "posInOverlappedRead", "numbOfContinuosKmerOverlapped",
"annotationOrder")
                    VALUES
(sequenceRow.id,referenceSeqId,pos,referenceSeqStartPos,finish,orderNu
mb );

                orderNumb := orderNumb + 1;
                referenceSeqId := referenceSeqIdAux;
                pos :=writeNucleotideIndex;
                referenceSeqStartPos := referenceSeqStartPosAux;
                finish :=referenceSeqStartPosAux;
                finish := finish + 1;
            END IF;
        END IF;
    END IF;

END LOOP;

    IF annotationClosed = false THEN

        INSERT INTO estudo1.roadmap (read, "overlappedRead",
"posInRead", "posInOverlappedRead", "numbOfContinuosKmerOverlapped",
"annotationOrder")
            VALUES
(sequenceRow.id,referenceSeqId,pos,referenceSeqStartPos,finish,orderNu
mb );

        orderNumb := orderNumb + 1;

    END IF;

END LOOP;

SELECT clock_timestamp() into end_time;
RAISE LOG 'Program start time: %', start_time;
RAISE LOG 'Program End time : %', end_time;

RETURN distinctKmersNumb;
END;
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;

```

## 8.4. Gera Output

```

CREATE OR REPLACE FUNCTION "estudo1".GeraOutput()
  RETURNS text AS
$BODY$
  DECLARE
    numbOfReads integer;
    tablename text;
    sequencetablename text;
    roadmaptablename text;
  BEGIN

    RAISE NOTICE '%','Iniciando exportação do arquivo Roadmap';

    DELETE FROM estudo1.tempfile;

    INSERT INTO estudo1.tempfile (
      select line
      from (
        select 0 as id, 0 as pos, count(*) ||'      0
        '||'49' ||'      0' as line from estudo1.sequences
        union all
        select id, 1 as pos, 'ROADMAP '||id as line from
        estudo1.sequences
        union all
        select rdm.read as id, rdm."annotationOrder"+2 as pos,
        rdm."overlappedRead" ||'      ' || rdm."posInRead" ||'      ' ||
        rdm."posInOverlappedRead" ||'      ' ||
        rdm."numbOfContinuosKmerOverlapped" as line FROM estudo1.roadmap
        as rdm
      ) UN
      ORDER BY ID, POS );

    COPY (SELECT line from estudo1.tempfile order by id) TO
    '/tmp/MyRoadmapsNew.txt' WITH CSV QUOTE E'\b';

    RAISE NOTICE '%','Roadmap Gerado com sucesso!';
    RAISE NOTICE '%','Iniciando exportação do arquivo Sequences';

    DELETE FROM estudo1.tempfile;

```



```
INSERT INTO estudol.tempfile (  
    select line  
    from (  
        SELECT id, 1 AS POS, name ||' '|| id|| ' ' ||cat as  
line from estudol.sequences  
        union all  
        select id, 2 AS POS, substr(read,0,61) from  
estudol.sequences  
        union all  
        select id, 3 AS POS, substr(read,61,120) from  
estudol.sequences  
        order by 1,2  
    ) UN);  
  
COPY (SELECT line from estudol.tempfile order by id) TO  
'/tmp/SequencesNew.txt' WITH CSV QUOTE E'\b';  
  
RAISE NOTICE '%','Arquivo Sequences Gerado com sucesso!';  
  
RETURN numbOfReads;  
END;  
$BODY$  
LANGUAGE plpgsql VOLATILE  
COST 100;
```