Title

K-Mer DNA Sequence Assembler

Authors:

August B. Sandoval

Hector Alva

Abstract

Compares the run time of DNA k-mer graph construction algorithms via String Composition and Hashing implementation. Preforming Eulerian Path finding on k-mer graphs to get a DNA sequence from a set of K-mers.

Introduction

K-Mer DNA Sequence Assembler is implementing DNA assembly via String Composition and Hashing. The project then compares both implementation and preforms Eulerian Path finding on both generated graphs of both implementation.

Describe the data structures and algorithms that you have designed using pseudo code.

Data Structure:

Graph:

The graph consists of a list of Nodes

Node:

Node contains fields for a label, a list of outgoing nodes and the number of incoming nodes.

The list of outgoing nodes hold the index values of the outgoing node in the Graph list.

Algorithm:

Hash:

The hash function used is standard C++ unordered\_multimap as it hash template.

The hash key generation takes a DNA sequence in the form of a C++ string and converts each character into values from 0 to 3(e.g. A: 0, C: 1, G: 2, T: 3). Each character value is added to the key value. After each character is added the value is multiplied by 4 except the last character. The key will only take the first 32 characters, giving a maximum bit length of 64 for the hash key.

Pseudocode:

S = DNA sequence.

Key = unsigned integer

For i less then 32 and S length

Convert S[i] to value (e.g. A: 0, C: 1, G: 2, T: 3)

Add value to Key

Multiply key by 4 except last character

Graph assembly via Hashing

The Graph assembly begins by passing a array of k-mer and hashing k-mer. The creation of a empty node array to hold all the node that will be created later.

Then iterate through the k-mer list. Find the the k-mer’s prefix and suffix. Check if the prefix and suffix are in the node array and the id’s from the hash and node array match.

If prefix or suffix is not in node array create new node with that prefix or suffix.

If ID’s don’t match then find correct id and set hash id to the new id.

Link the prefix and suffix nodes

Pseudocode:

Given: k-mers = list of k-mers

nodes = list of k-mer nodes

hash k-mers

for every k in k-mers:

get prefix and suffix

if prefix is not in nodes

create new node for prefix

if suffix id is not equal to hash id

set hash id to new id

if suffix is not in nodes

create new node for suffix

if suffix id is not equal to hash id

set hash id to new id

link prefix and suffix

Eulerian Path:

Compile:

To run the code is use the provided makefile on a Linux box

In the Linux box use the command “make” in the director with the makefile

Run:

After compiling using the makefile

Enter “./P2” in the command line

This will run the executable

1. Results:



1. Discussion: the relative performance of the two methods.

The performance between the construction of deBruijn graph via Hashing vs String Composition was very significant on the last test with a large graph. The Hashing implementation was many times faster than String Composition. As shown in the screenshot above the Hash implementation took 2.35 second vs 7 seconds for String Composition.

1. Distribution of work among team members

The main distribution of work among the group was the August was responsible for implementing a deBruijn graph construction via hashing. While Hector was responsible for implementing Eulerian path on the graph.

Debugging the project was shared among the two members.