tieLooseEnds(Family list families) – *Goes through list of families. If any family’s last instance not completed (making some lmers have higher freq than others), splits it on the last lmer seen and adds new family to families.*

getAndInsert(seeded lmer representation x, integer i)- *Returns Lmer object corresponding to x in LmerMap, creating one if does not exist. Adds i as last location this Lmer object was seen*

seeded(character array lmer[], seed s)- *Returns representation of lmer containing only characters corresponding to 1 positions of s*

equalOffsets(Lmer u, Lmer v)- *Returns whether distance between 1st and 2nd locations in lmer u is same as corresponding distance in lmer v*

closeEnough(Lmer u, Lmer v) – *Returns whether last location of lmer v is within L of last location of lmer u*

getFamilyIndex(Family array curr\_fams[], integer curr, integer L, Lmer v):

for i = 0 to L-1:

j <- curr – i (wrapping around to L+(curr-i) if negative)

fam <- curr\_fams[j]

if fam != NULL:

u <- fam->lastAdded() # Determine last lmer added to fam

if equalOffsets(u, v) && closeEnough(u,v): # u and v belong to same fam

return j

else:

return L

return L # Note: L signals that no family index found (i.e. need new fam)

splitByLmer(Family fam, Lmer v, boolean keepV) –*Splits fam at v, modifying fam to contain the lmers before the split, and returning a new Family that contains the lmers after the split. Flag keepV indicates whether v is kept in fam or moved to new Family.*

didSplit(Family list families, Family fam, Lmer v):

# case 1: fam has completed its previous instance (all lmers have same freq),

# but v is not the start of a new instance (i.e. not first lmer in fam).

if fam->lastInstanceComplete() && v != fam->getFirstAdded(): newFam <- splitByLmer(fam, v, false)

families.add(newFam)

return true

# case 2: v occurs after it is supposed to (over L past the last lmer seen by fam)

else if not closeEnough(fam->getLastSeen(), v):

newFam <- splitByLmer(fam, fam->getLastSeen(), true)

families.add(newFam)

return true

return false

inExpectedSpot(Family fam, Lmer v) – *Calculates v’s alleged “position” in the family is (i.e. the index of v in the chain of lmers corresponding to fam) based on its most recent location relative to the fam’s expected end. Returns whether or not the lmer at this calculated position is v.*

phRAIDER(sequence G, seed s, integer f):

#Input: Genomic sequence G, spaced seed s, minimum frequency f

L <- |s| # L: minimum length

F\_array <- array length L of families, each initialized to NULL

F\_index <- 0 # F\_index: index of current family under investigation

lastFam <- NULL

for i = 0 to |G| - L: # Scan genome l-mer by l-mer

x <- seeded(G[i:i+l],s) # x is the ith seeded lmer of G

v <- getAndInsert(x, i)

# if v has been seen exactly twice, it is a potential elementary repeat.

# therefore, it must be assigned a family (either existing or new)

if count(v) == 2:

j <- getFamilyIndex(F\_array, F\_index, L, v)

if j == L: # signals that v belongs to no existing family

F\_index <- F\_index+1 % L # go to next spot in F\_array

lastFam <- new Family

F\_array[F\_index] <- Last\_fam

Families.add(Last\_fam)

else:

lastFam <- F\_array[j]

Adopt v into Last\_fam

# else if v has been seen over 2 times, it is already associated with a family.

# therefore, must check that new location of v does not break family

else if count(v) > 2:

Family fam = v->getFamily()

# if do not have to split fam because of v’s new location,

# modify characteristics of fam to account for new location of v

if not didSplit(fam, v):

# if v marks start of new instance, record new expected end

if v == fam->getFirstAdded():

fam->setExpectedEnd(v->back() + fam->getRepeatLength())

# if v is correctly positioned in family, set it as last (correctly) seen

if inExpectedSpot(fam, v):

fam->setLastSeen(v)

tieLooseEnds()