**1. This is the code to evaluate simulated genome assembly results based on kmer (include CR, SCR, DCR, APLC, ADF)**

# -\*- coding: UTF-8 -\*-

import argparse

import time

from pandas import \*

from random import \*

import os

def ArgParse():

group = argparse.ArgumentParser(description='A python script for genome assessment.')

group.add\_argument('-i', '--input', help='assemble result with fasta format.', required=True)

group.add\_argument('-r', '--reference', help='reference sequence with fasta format.', required=True)

group.add\_argument('-k', '--kmer-length', type=int, help='the kmer length used in assessment, default=21.',default=21)

group.add\_argument('-o', '--out-prefix',help='prefix of output files.',required=True)

group.add\_argument('-s', '--sample', help='the number of ref unique kmer sampled, default=all.', default="all")

return group.parse\_args()

def integrateReadLine(fa, prefix):

file\_name = fa.split("/")[-1]

fi = open(fa, "r")

lines = fi.readlines()

fi.close()

fo = open(prefix + "\_" + file\_name, "w")

n = "1"

for line in lines:

if line[0] == ">":

if n == "1":

fo.write(line)

else:

fo.write("\n" + line)

else:

n = "larger than 1"

line = line.strip()

fo.write(line)

fo.close()

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "Integrate read lines in " + fa + " file done.")

def reverseCompleKmer(kmer):

reverse\_complementary\_kmer\_list = []

base\_dic = {"A":"T","T":"A","C":"G","G":"C","N":"N","a":"T","t":"A","c":"G","g":"C","n":"N"}

for base in list(kmer[::-1]):

reverse\_complementary\_kmer\_list.append(base\_dic[base])

reverse\_complementary\_kmer = "".join(reverse\_complementary\_kmer\_list)

return min([kmer.upper(), reverse\_complementary\_kmer.upper()])

def refUniqueKmerSearch(ref, kmer\_length):

fi = open(ref, "r")

lines = fi.readlines()

fi.close()

sumKmer\_dic = {}

for line in lines:

line = line.strip()

if line[0] != ">":

for i in range(len(line) - kmer\_length + 1):

sumKmer\_dic[reverseCompleKmer(line[i:i + kmer\_length])] = sumKmer\_dic.get(

reverseCompleKmer(line[i:i + kmer\_length]), 0) + 1

unique\_kmer = {}

for key, value in sumKmer\_dic.items():

if value == 1:

unique\_kmer[key] = 0

external\_dic = {}

internal\_dic = {}

chr\_name = ""

for line in lines:

line = line.strip().split()[0]

if line != "":

if line[0] == ">":

chr\_name = line[1:]

internal\_dic = {}

else:

for i in range(len(line) - kmer\_length + 1):

kmer = line[i:i + kmer\_length]

kmer = reverseCompleKmer(kmer)

if kmer in unique\_kmer.keys():

internal\_dic[kmer] = (i + 1, i + kmer\_length)

external\_dic[chr\_name] = internal\_dic

refChr\_uniKmer = {}

for K, V in external\_dic.items():

mydic = {}

for key, value in V.items():

if key in unique\_kmer:

mydic[key] = 1

refChr\_uniKmer[K] = mydic # refChr\_uniKmer = {"chr01":{"ATCG":1,...,"ATCG":1}...}

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "Unique kmer count in " + ref + " file done.")

with open("refUniKmer", "w") as fi:

fi.write("{}\t{}\t{}\t{}\n".format("Chr", "Kmer", "Start", "End"))

for key, value in external\_dic.items():

for k, v in value.items():

fi.write("{}\t{}\t{}\t{}\n".format(key, k, v[0], v[1]))

fi.write("\n")

fi.close()

return [refChr\_uniKmer, unique\_kmer]

def getRefUniKmerPos(ref, kmer\_length):

eachchr\_unikmer = refUniqueKmerSearch(ref, kmer\_length)[0]

data = open(ref, "r")

lines = data.readlines()

data.close()

kmerpos\_dic = {}

chr\_name = ""

for line in lines:

line = line.strip().split()[0]

if line[0] == ">":

chr\_name = line[1:]

kmerpos\_dic[chr\_name] = {}

for line in lines:

line = line.strip().split()[0]

if line[0] == ">":

chr\_name = line[1:]

else:

for i in range(len(line) - kmer\_length + 1):

if reverseCompleKmer(line[i:i + kmer\_length]) in eachchr\_unikmer[chr\_name].keys():

kmerpos\_dic[chr\_name][(i + 1, i + kmer\_length)] = 0

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "Get reference unique kmer position done.")

return kmerpos\_dic # kmerpos\_dic={"chr1":{(1,21):0,...,(41,51):0},...}

def removeOverlapKmer(classed\_kmer\_dic):

set\_option('display.max\_columns', None)

set\_option('display.max\_rows', None)

external\_dic = {}

internal\_dic = {}

for key, value in classed\_kmer\_dic.items():

internal\_dic = {}

value\_list = list(value.items())

value\_list.sort(key=lambda x: x[0][0], reverse=False)

mydic = {}

mydic["start\_pos"] = [value\_list[0][0][0]]

mydic["end\_pos"] = [value\_list[0][0][1]]

mydic["front\_end\_pos"] = [0]

last\_end\_pos = value\_list[0][0][1]

for i in range(1,len(value\_list)):

mydic["start\_pos"].append(value\_list[i][0][0])

mydic["end\_pos"].append(value\_list[i][0][1])

mydic["front\_end\_pos"].append(last\_end\_pos)

last\_end\_pos = value\_list[i][0][1]

mydataframe = DataFrame(mydic)

mydataframe = mydataframe[(mydataframe["front\_end\_pos"] - mydataframe["start\_pos"]) < 0]

for i in mydataframe.index:

internal\_dic[(mydataframe["start\_pos"][i],mydataframe["end\_pos"][i])] = 0

external\_dic[key] = internal\_dic

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()) + ", " + "Remove overlaps between kmers done.")

return external\_dic #external\_dic{"chr/ctg":{(1,21):0,...,(31,51):0},...}

def getHeaderKmer(fasta, headerkmerpos\_dic, kmer\_length):

data = open(fasta, "r")

lines = data.readlines()

data.close()

kmer\_pos\_dic = {}

seqname\_kmer\_dic = {}

sequence\_name = ""

for line in lines:

line = line.strip().split()[0]

if line[0] == ">":

sequence\_name = line[1:]

kmer\_pos\_dic = {}

else:

for key, value in headerkmerpos\_dic.items():

if sequence\_name == key:

for pos in value.keys():

kmer\_pos\_dic[reverseCompleKmer(line[pos[0] - 1:pos[0] + kmer\_length - 1])] = []

seqname\_kmer\_dic[sequence\_name] = kmer\_pos\_dic

for line in lines:

line = line.strip().split()[0]

if line[0] == ">":

sequence\_name = line[1:]

else:

for key, value in headerkmerpos\_dic.items():

if sequence\_name == key:

for pos in value:

seqname\_kmer\_dic[sequence\_name][reverseCompleKmer(line[pos[0] - 1: pos[0] + kmer\_length - 1])].append(pos)

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()) + ", " + "Get header kmer done.")

return seqname\_kmer\_dic # seqname\_kmer\_dic={"chr/ctg":{"ATCG":[(1,21),...,(31,51)]},...}

def randomselectkmer(refkmerdic,sample\_num):

kmerdic = {}

for key,value in refkmerdic.items():

for k,v in value.items():

kmerdic[k] = 0

seed(111)

mylist = sample(list(kmerdic.items()),sample\_num)

newkmerdic = {}

for i in mylist:

newkmerdic[i[0]] = 0

external\_dic = {}

for key,value in refkmerdic.items():

internal\_dic = {}

for k,v in value.items():

if k in newkmerdic.keys():

internal\_dic[k] = v

external\_dic[key] = internal\_dic

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()) + ", random select 200000 reference kmer done.")

return external\_dic # external\_dic={"chr/ctg":{"ATCG":[(1,21),...,(31,51)]},...}

def getAsbkmer(ctgfile,reffile,refkmer\_dic,kmer\_length):

refkmer = {}

for key,value in refkmer\_dic.items():

for k,v in value.items():

refkmer[k] = 0

file = open(ctgfile,"r")

lines = file.readlines()

file.close()

external\_dic = {}

internal\_dic = {}

ctgname = ""

for line in lines:

line = line.strip().split()[0]

if line[0] == ">":

ctgname = line[1:]

internal\_dic = {}

else:

for i in range(len(line) - kmer\_length + 1):

if reverseCompleKmer(line[i:i + kmer\_length]) in refkmer.keys():

internal\_dic[reverseCompleKmer(line[i:i + kmer\_length])] = internal\_dic.get(reverseCompleKmer(line[i:i + kmer\_length]),[]) + [(i + 1, i + kmer\_length)]

external\_dic[ctgname] = internal\_dic

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "Unique kmer from " + reffile + " file count in " + ctgfile + " file done.")

return external\_dic # external\_dic={"chr/ctg":{"ATCG":[(1,21),...,(31,51)]},...}

def assembleAssessment(ref\_fa, asb\_fa, kmer\_length, sample\_num, prefix):

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()) + ", Begin.")

integrateReadLine(ref\_fa, prefix)

integrateReadLine(asb\_fa, prefix)

refUniKmerPos = getRefUniKmerPos(prefix + "\_" + ref\_fa.split("/")[-1], kmer\_length)

refHeaderKmerPos = removeOverlapKmer(refUniKmerPos)

refUniKmerPos = None

if sample\_num == "all":

refHeaderKmer = getHeaderKmer(prefix + "\_" + ref\_fa.split("/")[-1], refHeaderKmerPos, kmer\_length)

else:

refheaderKmer = getHeaderKmer(prefix + "\_" + ref\_fa.split("/")[-1], refHeaderKmerPos, kmer\_length)

refHeaderKmer = randomselectkmer(refheaderKmer,int(sample\_num))

refheaderkmer = None

refHeaderKmerPos = None

asbHeaderKmer = getAsbkmer(prefix + "\_" + asb\_fa.split("/")[-1],prefix + "\_" + ref\_fa.split("/")[-1],refHeaderKmer,kmer\_length)

with open("random\_refHeaderkmer.dic","w") as file:

file.write(str(refHeaderKmer))

file.close()

with open("refHeaderkmer.out", "w") as file:

file.write("Chr\tKmer\tStart\tEnd\n")

for key, value in refHeaderKmer.items():

for k, v in value.items():

file.write("{}\t{}\t{}\t{}\n".format(key, k, v[0][0], v[0][1]))

file.write("\n")

file.close()

with open("asbHeaderkmer.out", "w") as file:

file.write("Contig\tKmer\tStart\tEnd\n")

for key, value in asbHeaderKmer.items():

for k, v in value.items():

for i in v:

file.write("{}\t{}\t{}\t{}\n".format(key, k, i[0], i[1]))

file.write("\n")

file.close()

asbKmerCount = {}

for Key, Value in asbHeaderKmer.items():

for k, v in Value.items():

asbKmerCount[k] = asbKmerCount.get(k, 0) + len(v)

singleCopyKmerNum = 0

duplicateKmerNum = 0

colKmerNum = 0

for key, value in refHeaderKmer.items():

for k, v in value.items():

colKmerNum += 1

for key, value in asbKmerCount.items():

if value == 1:

singleCopyKmerNum += 1

else:

duplicateKmerNum += 1

singleCopy = singleCopyKmerNum / colKmerNum

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()) + ", " + "Compute single copy rate done.")

duplicateRate = duplicateKmerNum / colKmerNum

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()) + ", " + "Compute duplicate rate done.")

ctgkmer\_dic = {}

for key, value in asbHeaderKmer.items():

ctgkmer\_dic[key] = {}

for key, value in asbHeaderKmer.items():

for k, v in value.items():

if k not in ctgkmer\_dic[key].keys():

ctgkmer\_dic[key][k] = 0 # ctgkmer\_dic = {"ctg00001":{"ATCG":0,...,"ATCG":0},...}

chrkmer\_dic = {}

for key, value in refHeaderKmer.items():

chrkmer\_dic[key] = {}

for key, value in refHeaderKmer.items():

for k, v in value.items():

if k not in chrkmer\_dic[key].keys():

chrkmer\_dic[key][k] = 0 # chrkmer\_dic = {"Chr01":{"ATCG":0,...,"ATCG":0},...}

chr\_to\_kmerlist = {}

contig\_to\_chr = {}

for ctg, ctgkmer in asbHeaderKmer.items():

chr\_to\_kmerlist = {}

for chr, chrkmer in refHeaderKmer.items():

chr\_to\_kmerlist[chr] = {}

contig\_to\_chr[ctg] = chr\_to\_kmerlist

for asb\_key, asb\_value in ctgkmer\_dic.items():

for i in asb\_value.keys():

for ref\_key, ref\_value in chrkmer\_dic.items():

if i in ref\_value.keys() and i not in contig\_to\_chr[asb\_key][ref\_key].keys():

contig\_to\_chr[asb\_key][ref\_key][i] = 0 # contig\_to\_chr={"ctg0001":{"chr01":{"ATCG":0,...,"ATCG":0},...},...}chr和ctg共有kmer

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "Get collective header kmer from reference and assemble file done.")

contig\_to\_kmer = {}

for key, value in contig\_to\_chr.items():

kmerlist = {}

for k, v in value.items():

for kmer, it in v.items():

kmerlist[kmer] = 0

contig\_to\_kmer[key] = kmerlist # contig\_to\_kmer = {"ctg0001":{"ATCG":0,...,"ATCG":0},...}

ctgkmer\_dic = None

chrkmer\_dic = None

proportion\_of\_the\_largest\_categories = 0

largest\_categories\_ex = {}

largest\_categories\_in = {}

numerator = 0

denominator = 0

for key, value in contig\_to\_chr.items(): # 找出最大类

largest\_categories\_num = 0

largest\_categories\_in = {}

for k, v in value.items():

if len(v) > largest\_categories\_num:

largest\_categories\_num = len(v)

for k, v in value.items():

if len(v) == largest\_categories\_num:

largest\_categories\_in[k] = v

largest\_categories\_ex[key] = largest\_categories\_in # largest\_categories\_ex={"ctg0001":{"chr01":{"ATCG":0,...,"ATCG":0}},...}

numerator += largest\_categories\_num

denominator += len(contig\_to\_kmer[key])

proportion\_of\_the\_largest\_categories += float(numerator) / float(denominator)

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "Compute proportion of the largest categories done.")

aveEachCtgDistance\_sum = 0

contig\_to\_chr = None

with open("largest\_categories.dic","w") as fi:

fi.write(str(largest\_categories\_ex))

fi.close()

with open("refKmer.dic","w") as fi:

fi.write(str(refHeaderKmer))

fi.close()

with open("asbKmer.dic","w") as fi:

fi.write(str(asbHeaderKmer))

fi.close()

contig\_nums = 0

for key, value in largest\_categories\_ex.items(): # largest\_categories\_ex={"ctg0001":{"chr01":{"ATCG":0,...,"ATCG":0}},...}

ref\_base\_pos\_dic = {}

asb\_base\_pos\_dic = {}

t = 0

eachCtgDistance\_sum = 0

for k, v in value.items():

if len(v) > 1:

contig\_nums += 1

for i in range(len(v)):

ref\_base\_pos\_dic[list(v.items())[i][0]] = refHeaderKmer[k][list(v.items())[i][0]] # ref\_base\_pos\_dic={"ATCG":[(1,21),...,(31,51)]}

asb\_base\_pos\_dic[list(v.items())[i][0]] = asbHeaderKmer[key][list(v.items())[i][0]]

ref\_base\_list = list(ref\_base\_pos\_dic.items())

ref\_base\_list.sort(key=lambda x: x[1][0][0], reverse=False)

for i in range(len(ref\_base\_list) - 1):

refKmerDistance = abs(ref\_base\_list[i + 1][1][0][0] - ref\_base\_list[i][1][0][0])

asbKmerDistance = {}

for j in range(len(asb\_base\_pos\_dic[ref\_base\_list[i][0]])):

for m in range(len(asb\_base\_pos\_dic[ref\_base\_list[i + 1][0]])):

distance = abs(asb\_base\_pos\_dic[ref\_base\_list[i][0]][j][0] - asb\_base\_pos\_dic[ref\_base\_list[i + 1][0]][m][0])

asbKmerDistance[distance] = 0

for j in asbKmerDistance.keys():

eachCtgDistance\_sum += abs(j - refKmerDistance)

t += 1

aveEachCtgDistance\_sum += eachCtgDistance\_sum / t

else:

continue

ave\_distance\_diff = aveEachCtgDistance\_sum / contig\_nums

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "Compute average distance difference done." + "\n")

print("{:<12.7f}Complete\n"

"{:<12.7f}Complete and single-copy\n"

"{:<12.7f}Complete and duplicated\n"

"{:<12.7f}Proportion of the largest categories\n"

"{:<12.7f}ave distance diff\n".format(singleCopy + duplicateRate, singleCopy, duplicateRate,proportion\_of\_the\_largest\_categories, ave\_distance\_diff))

with open("result.report", "w") as fi:

fi.write("{:<12.7f}Complete\n"

"{:<12.7f}Complete and single-copy\n"

"{:<12.7f}Complete and duplicated\n"

"{:<12.7f}Proportion of the largest categories\n"

"{:<12.7f}ave distance diff\n".format(singleCopy + duplicateRate, singleCopy, duplicateRate,proportion\_of\_the\_largest\_categories, ave\_distance\_diff))

fi.close()

if \_\_name\_\_ == "\_\_main\_\_":

opt = ArgParse()

path = os.getcwd()

asb\_seq = opt.input

ref\_seq = opt.reference

kmer\_len = opt.kmer\_length

prefix = opt.out\_prefix

if opt.sample != "all":

sample\_num = int(opt.sample)

else:

sample\_num = opt.sample

asb\_seq\_path = os.path.join(path,asb\_seq)

ref\_seq\_path = os.path.join(path,ref\_seq)

assembleAssessment(ref\_seq\_path, asb\_seq\_path, kmer\_len, sample\_num, prefix)

**2. custom script (QV)**

import argparse

import time

import os

import math

import numpy as np

def ArgParse():

group = argparse.ArgumentParser(description='A python script for genome assessment of QV value.')

group.add\_argument('-s', '--seq', help='raw reads file for genome assessment(fastq)', required=True)

group.add\_argument('-a', '--asb', help='genome assembly file for genome assessment(fasta)', required=True)

group.add\_argument('-k', '--kmer-length', type=int, help='the kmer length used in assessment, default=21.',default=21)

group.add\_argument('-t', '--threats', type=int, help='the threats used to process kmer in Jellyfish, default=10.',default=10)

group.add\_argument('-hs', '--harsh-size', type=str, help='estimated harsh size to store kmer, and this value must up to genome size, default=1G.',default="1G")

return group.parse\_args()

def integrateReadLine(fa):

file\_name = fa.split("/")[-1]

fi = open(fa, "r")

lines = fi.readlines()

fi.close()

fo = open(file\_name.split(".")[0] + "\_integrate.fasta", "w")

n = "1"

for line in lines:

if line[0] == ">":

if n == "1":

fo.write(line)

else:

fo.write("\n" + line)

else:

n = "larger than 1"

line = line.strip()

fo.write(line)

fo.close()

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",

time.localtime()) + ", " + "Integrate read lines in " + fa + " file done.")

def getJellyfishKmer(seq\_fq,asb\_fa,kmer\_length,harsh\_size,threats):

os.system("jellyfish count -C -c 7 -m " + str(kmer\_length) + " -s " + harsh\_size + " -t " + str(threats) + " -o " + "seq\_kmer" + str(kmer\_length) + ".out " + seq\_fq)

os.system("jellyfish dump -c -t seq\_kmer" + str(kmer\_length) + ".out > seq\_kmer" + str(kmer\_length) + ".freq")

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()) + ", " + "kmer frequency counting in sequence has done!")

os.system("jellyfish count -C -c 7 -m " + str(kmer\_length) + " -s " + harsh\_size + " -t " + str(threats) + " -o " + "asb\_kmer" + str(kmer\_length) + ".out " + asb\_fa)

os.system("jellyfish dump -c -t asb\_kmer" + str(kmer\_length) + ".out > asb\_kmer" + str(kmer\_length) + ".freq")

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "kmer frequency counting in assembly has done!")

def calcuQV(kmer\_length):

seqkmer\_dict = {}

with open("seq\_kmer" + str(kmer\_length) + ".freq","r") as fi:

lines = fi.readlines()

for line in lines:

kmer = line.strip().split()[0]

seqkmer\_dict[kmer] = 0

fi.close()

asbkmer\_dict = {}

with open("asb\_kmer" + str(kmer\_length) + ".freq","r") as fi:

lines = fi.readlines()

for line in lines:

kmer = line.strip().split()[0]

asbkmer\_dict[kmer] = 0

fi.close()

sharekmer\_dict = {}

for key in seqkmer\_dict.keys():

if key in asbkmer\_dict.keys():

sharekmer\_dict[key] = 0

del seqkmer\_dict

QV = -10 \* np.log10(1 - math.pow(len(sharekmer\_dict)/len(asbkmer\_dict),1/kmer\_length))

with open("QV.value","w") as fi:

fi.write("{:<12.3f}".format(QV))

fi.close()

print("[INFO] " + time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) + ", " + "the QV value is {:<12.3f}!".format(QV))

if \_\_name\_\_ == "\_\_main\_\_":

opt = ArgParse()

seq\_fa = opt.seq

asb\_fa = opt.asb

kmer\_length = opt.kmer\_length

harsh\_size = opt.harsh\_size

threats = opt.threats

path = os.getcwd()

raw\_asb\_fa\_path = os.path.join(path, asb\_fa)

integrateReadLine(raw\_asb\_fa\_path)

seq\_fa\_path = os.path.join(path, seq\_fa)

asb\_fa\_path = os.path.join(path,raw\_asb\_fa\_path.split("/")[-1].split(".")[0] + "\_integrate.fasta")

getJellyfishKmer(seq\_fa\_path,asb\_fa\_path,kmer\_length,harsh\_size,threats)

calcuQV(kmer\_length)

**3. custom script (for assembly, include Hicanu, hifiasm, HiFlye, MECAT2, Miniasm, NextDenovo, Shasta, Peregrine, Verkko, LJA, rust-mdbg)**

HiCanu：

\time -v canu -p rice -d /home/jiangheling/Luohh/Assemble/Truedata/Rice/Hicanu genomeSize=392m useGrid=false merylThreads=30 hapThreads=30 cormhapThreads=30 obtovlThreads=30 utgovlThreads=30 corThreads=30 ovbThreads=30 ovsThreads=30 redThreads=30 oeaThreads=30 batThreads=30 cnsThreads=30 -pacbio-hifi ../../Data/rice.fastq

Hifiasm：

\time -v hifiasm -o rice -t 30 ../../Data/rice.fastq

Hiflye:

\time -v flye --pacbio-hifi ../../Data/rice.fastq -o /home/jiangheling/Luohh/Assemble/Truedata/Rice/Hiflye --genome-size 392m --threads=30

Miniasm:

\time -v minimap2 -t 30 -x ava-pb ../../Data/rice.fastq ../../Data/rice.fastq | gzip -1 > rice.paf.gz

\time -v miniasm -f ../../Data/rice.fastq rice.paf.gz > rice.gfa

\time -v awk '/^S/{print ">"$2"\n"$3}' rice.gfa > rice.fasta

NextDenovo:

nextDenovo run.cfg

run.cfg：

[General]

job\_type = local

job\_prefix = nextDenovo

task = all

rewrite = yes

deltmp = yes

parallel\_jobs = 10

input\_type = raw

read\_type = hifi # clr, ont, hifi

input\_fofn = input.fofn

workdir = 01\_rundir

[correct\_option]

read\_cutoff = 1k

genome\_size = 392m # estimated genome size

sort\_options = -m 40g -t 30

minimap2\_options\_raw = -t 30

pa\_correction = 3

correction\_options = -p 15

[assemble\_option]

minimap2\_options\_cns = -t 30

minimap2\_options\_map = -t 30

nextgraph\_options = -a 1

MECAT2：

\time -v mecat.pl correct config\_file.txt

\time -v mecat.pl trim config\_file.txt

\time -v mecat.pl assemble config\_file.txt

config\_file.txt：

PROJECT=Rice

RAWREADS=/home/jiangheling/Luohh/Assemble/Truedata/Data/rice.fastq

GENOME\_SIZE=392000000

THREADS=30

MIN\_READ\_LENGTH=2000

CNS\_OVLP\_OPTIONS="-kmer\_size 13"

CNS\_PCAN\_OPTIONS="-p 100000 -k 100"

CNS\_OPTIONS=""

CNS\_OUTPUT\_COVERAGE=30

TRIM\_OVLP\_OPTIONS="-skip\_overhang"

TRIM\_PM4\_OPTIONS="-p 100000 -k 100"

TRIM\_LCR\_OPTIONS=""

TRIM\_SR\_OPTIONS=""

ASM\_OVLP\_OPTIONS=""

FSA\_OL\_FILTER\_OPTIONS="--max\_overhang=-1 --min\_identity=-1"

FSA\_ASSEMBLE\_OPTIONS=""

CLEANUP=0

Peregrine：

\time -v /public/home/zhaoxianjia/software/peregrine-2021-0.4.13/target/release/pg\_asm ricereads.lst rice 30

shasta：

\time -v shasta --config HiFi-Oct2021 --input ../../Assemble/Truedata/Data/rice.fastq --assemblyDirectory rice --command assemble --threads 30

Verkko：

\time -v verkko -d /home/jiangheling/Luohh/True/Verkko/Hap --hifi /home/jiangheling/Luohh/True/Data/rice.fastq.gz --threads 30 --sto-run 30 100 24 --mer-run 30 100 24 --ovb-run 30 100 24 --ovs-run 30 100 24 --red-run 30 100 24 --mbg-run 30 100 24 --utg-run 30 100 24 --spl-run 30 100 24 --ali-run 30 100 24 --pop-run 30 100 24 --utp-run 30 100 24 --lay-run 30 100 24 --sub-run 30 100 24 --par-run 30 100 24 --cns-run 30 100 24

rust-mdbg:

rust\_mdbg="/path/to/rust-mdbg"

output\_prefix="example"

input\_fastq="/path/to/test.fastq"

${rust\_mdbg} -k 21 --density 0.003 -l 14 --prefix ${output\_prefix} --threads 30 ${input\_fastq}

LJA:

lja="/path/to/lja"

input\_fastq="/path/to/test.fastq"

output\_dir="/path/to/output"

${lja} -t 30 --diploid --reads ${input\_fastq} -o ${output\_dir}