Notes\_Python\_operation

# 实际操作函数

## 1.1序列处理

### 1.1.1总结

|  |  |
| --- | --- |
| 功能 | 包含的可用结构 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### 1.1.2将vcf格式转化为fasta格式

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

import sys

import os

import re

import gzip

def info2nucl(info, ref\_na, alt\_nas):

if info == "." or info == "./.":

nucl = ref\_fa

#如果info为"."或"./."，将核酸赋值为参考序列

elif re.match(r"\d/", info):

gti = int(re.sub(r"\d/", "", info))-1

#如果info中包含数字，则将数字替换为空白，而后将其转换为正整数

nucl = alt\_nas[gti]

#从alt\_nas列表中取得核酸序列信息

else:

print(info)

若均不符合，打印info变量

return nucl

fa\_heads = []

dhead\_fas = {}

input = sys.argv[1]

outfile = ""

if input.endswith(".gz"):

opener = gzip.open

outfile = input.replace(".vcf.gz", "")

#如果输入的文件后缀为“.gz”，则使用gzip.open打开，并将后缀去掉的文件名加入下一步

else:

opener = open

outfile = input.replace(".vcf", "")

line\_count = 0

ref\_fa = ""

with opener(input) as IN:

ref\_na = "N"

alt\_na = "N"

print("Processing...\nline[pos:ref\_position]")

for line in IN:

line = line.strip()

#打开文件并去除每一行两侧的空白

if re.match("##", line):

continue

#跳过包含##开头的注释行

elif re.match("#CHROM", line):

line\_spt = re.split("\t", line)

#对于以“#CHROM”开头的标题行，使用制表符将每一行分隔

fa\_heads = line\_spt[9:]

#取分隔后第9个元素以后的所有元素作为变量

len\_fa\_heads = len(fa\_heads)

print ("##There are " + str(len\_fa\_heads) + " samples.")

else:

line\_count += 1

line\_spt = re.split("\t", line)

ref\_pos = str(line\_spt[1])

#储存突变位点的位置

if line\_count%1000 == 0:

print(str(line\_count) + "[" + ref\_pos + "],"),

#当行的数目可以被1000整除时打印位置

ref\_na = line\_spt[3]

#取第3个元素作为参考序列的碱基类型

# if len(ref\_na) != 1:

# print(str(line\_count) + "[" + ref\_pos + "]" + ref\_na + " is not SNP.")

# continue

ref\_fa = ref\_fa + ref\_na

#将参考序列的碱基类型加在一起

# ref\_nas = re.split(",", ref\_na)

alt\_na = line\_spt[4]

#取第4个元素作为替代序列的碱基类型

alt\_nas = re.split(",", alt\_na)

#用”,”分隔作为替代序列的碱基类型

len\_alt\_nas = len(alt\_nas)

vc\_list = line\_spt[9:]

uniq\_vc\_list = list(set(vc\_list))

#将每一行的第九列及其之后的元素存为集合

# print(uniq\_vc\_list)

if len\_fa\_heads == len (vc\_list):

#如果突变信息的数目和序号的数目相同，按顺序处理每个样本的信息

for fa\_head in fa\_heads:

vc\_info = vc\_list.pop(0)

#取得每个vc\_list列表中的第0个元素

nucl = info2nucl(vc\_info, ref\_na, alt\_nas)

#由vc\_info获得vcf信息对应的核酸序列

dhead\_fas.setdefault(fa\_head, "")

#查找fa\_head的键值作为dhead\_fas词典，若无键值，则设定空白作为键值

dhead\_fas[fa\_head] += nucl

#为每个fa\_head索引设置核酸序列的信息

else:

print(len(vc\_list))

exit("Number of headers is not equal to snps.")

with open(outfile + ".fasta", "w") as OUT:

OUT.write(">ref\_H37v\n")

OUT.write(ref\_fa + "\n")

for fa\_head in fa\_heads:

OUT.write(">" + fa\_head + "\n")

OUT.write(dhead\_fas[fa\_head] + "\n")

#输出参考基因组的核酸序列和每个样本对应的核酸序列

### 1.1.3将fasta文件转换为axt格式文件

import sys

def parseFasta(filename):

fas = {}

idlis = []

id = None

with open(filename, 'r') as fh:

for line in fh:

if line[0] == '>':

header = line[1:].rstrip()

id = header.split()[0]

idlis.append(id)

fas[id] = []

else:

fas[id].append(line.rstrip())

for id, seq in fas.iteritems():

fas[id] = ''.join(seq)

return fas, idlis

ALN, IDlis = parseFasta(sys.argv[1])

outid = "-".join(IDlis)

outseq = "\n".join([ALN[IDlis[0]],ALN[IDlis[1]]])

print(">" + outid)

print(outseq)

### 1.1.4提取gemma的结果

# -\*- coding: utf-8 -\*-

import pandas as pd

import numpy as np

import glob

import os

os.chdir(r"/data2/lizhiyuan/MTB/MTB\_gwas/MTB\_gwas\_filterGEMMA\_plink/output")

for filename in glob.glob("\*.assoc.fisher"):

print(filename)

feature = pd.read\_csv(filename, delimiter="\t", header=0)

feature['resistance'] = filename

feature ['Software'] = "GEMMA"

feature=pd.DataFrame(feature)

print(feature.head(2))

feature.to\_csv('ann\_nohead\_noPEPPE\_assoc\_plink.csv',mode='+a')

1.1.4

构建一棵树（Completing a Tree）

Given: A positive integer n (n≤1000) and an adjacency list corresponding to a graph on n nodes that contains no cycles.

所给：一个不大于1000的正整数n以及一个邻接表，对应一个有n个结点且无环的图。

Return: The minimum number of edges that can be added to the graph to produce a tree.

需得：添加到图中从而构成树的边的最小数目。

数据文件：

10

1 2

2 8

4 10

5 9

6 10

7 9

测试输出：

3

代码：

f = open('input.txt', 'r')

lines = f.readlines()

f.close()

num = lines[0] # 存储结点的数目

edges = lines[1:] # 存储边的关系

i = 1

nodes = []

while i <= int(num):

nodes.append(str(i)) # 用一个列表保存所有节点

i += 1

fedge = edges[0].replace('\n','') # 把邻接表第一行取出来，作为第一“堆”的基础

fedge = fedge.split(' ')

nodes.remove(fedge[0])

nodes.remove(fedge[1])

tree = [fedge] # tree用来存储结点通过邻接表存储的关系形成的“堆”

flag = 1 # flag存储现在tree里有几“堆”

i = 1

while i < len(edges): # i遍历邻接表

edge = edges[i].replace('\n','')

edge = edge.split(' ')

if edge[0] in nodes:

nodes.remove(edge[0]) # 每取出一个边，就把形成边的结点从结点列表中删去

if edge[1] in nodes:

nodes.remove(edge[1])

j = 0

tag = 0

note = []

while j < flag: # j用来遍历各“堆”

if edge[0] in tree[j]: # 如果这条边的一个结点已经在其中一个“堆”里

tree[j].append(edge[1]) # 把这个边加入这一“堆”

tag += 1

note.append(j) # note存储这个结点在哪个“堆”

j += 1

elif edge[1] in tree[j]:

tree[j].append(edge[0])

tag += 1

note.append(j)

j += 1

else: # 如果不在这一“堆”里，j加一，以查找下一“堆”

j += 1

if tag == 0: # 如果在之前存在的“堆”里都没有这条边

tree.append(edge)

flag += 1 # 添加一个新的“堆”

if tag == 2: # 如果两个结点各在两“堆”中，说明这两个“堆”可以合并

flag -= 1

tree[note[0]].extend(tree[note[1]])

tree.remove(tree[note[1]]) # 合并两“堆”并删去其中一个

tree[note[0]].remove(edge[0])

tree[note[0]].remove(edge[1]) # 删去由合并导致的重复结点

i += 1

print(tree)

lennodes = len(nodes) # 记录现在结点列表中还有几个结点，每个都相当于一个单独的“堆”

sum = (flag - 1) + lennodes # “堆”的数量减去一即为需要添加的边的数量

print(sum)

### 1.1.X计算DNA的反向互补序列

# s = "AAAACCCGGT"

f = open("rosalind\_revc.txt",'r')

s = f.read()

re = s[::-1] # 字符串反向

c = "" # 定义字符串c接收互补序列

for i in re:

if i == 'A':

c = c + 'T'

elif i == 'G':

c = c + 'C'

elif i == 'T':

c = c + 'A'

elif i == 'C':

c = c + 'G'

print(c)