Actividad Tema 1:

Introducción a la algorítmica

Asignatura: Algoritmos en Bioinformática

Profesor: Vicente Arnau Llombart

Alumno：Qinding Xie

Máster en Bioinformática

Febrero de 2023

Q1. Habrá que definir cómo almacenar los datos de entrada y la estructura de datos que se utilizará para la resolución del algoritmo.

A menudo se dice que un programa = estructura de datos + algoritmo. Cuando se encuentra un problema, o surge una necesidad, uno de los pasos importantes en el diseño de un programa para resolver el problema es diseñar la estructura de datos, que se utiliza en la resolución de problemas principalmente para:

-Almacenar los datos que se van a procesar

-Aplicar estrategias algorítmicas

La estructura de datos puede representarse mediante un cuadruplete de la siguiente manera

Estructura de datos = (D, L, S, O)

Consta de los elementos de datos (D), las relaciones lógicas entre los elementos de datos (L), la estructura en la que se almacenan las relaciones lógicas en el ordenador (S) y las operaciones especificadas (O).

Las estructuras lógicas se utilizan principalmente para el diseño de algoritmos, mientras que las estructuras de almacenamiento se emplean para guiar la programación de la implementación de los algoritmos. Existen dos tipos básicos de estructuras de almacenamiento.

Almacenamiento secuencial: los elementos lógicamente adyacentes se almacenan en celdas de almacenamiento físicamente adyacentes

Almacenamiento encadenado: se añaden algunos campos de dirección o estructuras auxiliares a los elementos de datos para almacenar las relaciones entre los elementos de datos.

Las estructuras de almacenamiento secuenciales tienen direcciones contiguas en memoria, por lo que el acceso es rápido, pero ineficiente en las operaciones de inserción o borrado, que mueven elementos de datos. Las estructuras de almacenamiento encadenadas pueden tener direcciones discontinuas en memoria y son eficientes en las operaciones de inserción y borrado, pero ineficientes en las de búsqueda y recorrido, debido a las operaciones de direccionamiento añadidas.

Las mismas estructuras lógicas (lineal, árbol, grafo, colección) pueden utilizarse como estructuras de almacenamiento secuenciales o encadenadas para almacenar datos y relaciones. La elección de la estructura de almacenamiento se basa principalmente en la eficiencia del algoritmo, que es mejor en términos de tiempo o espacio para el algoritmo, y la elección de los cuales está relacionada con los requisitos; las estructuras básicas de almacenamiento se pueden utilizar ya sea individualmente o en combinación.

Q2. Tendrá que explicarse con detalle el funcionamiento del algoritmo y su coste.

Para escribir el algoritmo, he optado por escribirlo en Python. El enfoque era iterar y comparar para encontrar el mejor resultado de costura, haciendo lo siguiente.

Primero saca SS1 y compara cada uno de SS2-SS10 en un bucle para obtener el nucleótido con más solapamientos (el número de nucleótidos solapados también debe ser mayor que 11 en este punto), y selecciona el mejor para empalmar. En este punto, puede que no haya ninguno que cumpla las condiciones anteriores, por lo que se selecciona SS2 para comparar SS3-SS10 uno a uno, y se selecciona la mejor combinación para empalmar, y los no empalmados también se ponen en datos para esperar a empalmes posteriores. Si SS1 se empalma con éxito con una de ellas, la nueva secuencia se compara y empalma con la biblioteca de datos sucesivamente, hasta completar todas. Al final, se dejan las secuencias que no se han empalmado con las anteriores, y el bucle continúa hasta que no quedan secuencias por empalmar, entonces salta el bucle y se completa toda la operación. Por tanto, no habrá un único resultado, sino varios.

Código:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os, time, functools

# Empeza
startTime = time.time()

# Leer fechiro
with open("Secuencias_2023.fa ", "rt", encoding="utf8") as f:
    datas = [each.strip() for each in f.readlines()]
    datas = list(filter(lambda x:len(x) > 6, datas))

def join(str1, str2):
    result = None
```

```
    # Obtener la longitud de la cadena más corta
    minLength = len(str1) if len(str1) < len(str2) else len(str2)

    # Detección de discordancias para obtener
    # la cadena repetida más larga
    for i in range(1, minLength+1):
        a = str1[-i:]
        b = str2[:i]
        if a == b:
            result = b

    # Si el resultado está presente y es mayor que 11, se desduplica
    #   y se empalma posteriormente, de lo contrario se descarta str2.
    if result and len(result) > 11:
        return str1 + str2[len(result):], len(result) if result else 0, result
    else:
        return str1, len(result) if result else 0, result

def enum(first:str, other:list) -> (str, int):
    result = []
    for each in other:
        result.append(join(first, each))

    a = max(result, key=lambda x:x[1])
    # print(a)
    if first == a[0]:
        return a[0], None
    else:
        return a[0], result.index(a)

result = datas.pop(0)
i = 0
while datas:
    result, index = enum(result, datas)
    if index is None:
        datas.append(result)
        result = datas.pop(0)
        i += 1
        if i == len(datas):
            datas.append(result)
            break
        continue
    i = 0
    #print(result)
```

```
    datas.pop(index)
for each in datas:
    print(each)
    print("\n")
print(f"Se ejecuta durante un total de {time.time() - startTime}segundos")
```

Q3. Se aplicará al fichero Secuencias_2023.fa. Se mostrarán los resultados obtenidos y se proporcionará el tiempo de cálculo del algoritmo

Resultado:

Los 2 resultados empalmados se muestran aquí en 0.002997s.

```
PS C:\Users\Alicia\Desktop\semestre2\algrismo\tarea\字符串最大重叠拼接算法\字符串最大重叠拼接算法> & C:/Users/Alicia/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Alicia/Desktop/semestre2/algrismo/tarea/字符串最大重叠拼接算法/字符串最大重叠拼接算法/Algoritmo1.py
.py
CCCTGAAGCGCGTACACACCGCCCGTCACCCTCCTCAAGTATACTTCAAAGGACATTTAACTAAAACCCCTACGCATTTATATAGAGGAGACAAGTCGTAACATGGTAAGTGTACTGGAAAGTGCACTTGGACGAACCAGAGTGTAGCTTAACACAAAGCACCCAACTTACACTTAGGAGATTTCAACTTAACTTGACCGCTCTGAGCTAAACCTAGCCCCAAACCCACTCCACCTTACTACCAGACAACCTTAGCCAAACCATTTACCCAAATAAAGTATAGGCGATAGAAATTGAAACCTGGCGCAATAGATATAGTACCGCAAGGGAAAGATGAAAAATTATAACCAAGCATAATATAGCAAGGACTAACCCCTATACCTTCTGCAT


TAAAAAGAGGCCTAACCCCTGTCTTTAGATTTACAGTCCAATGCTTCACTCAGCCATTTTACCTCACCCCCACTGATGTTCGCCGACCGTTGACTATTCTCTACAAACCACAAAGACATTGGAACACTATACCTATTATTCGGCGCATGAGCTGGAGTCCTAGGCACAGCTCTAAGCCTCCTTATTCGAGCCGAGCTGGGCCAGCCAGGCAACCTTCTAGGTAACGACCACATCTACAACGTTATCGTCACAGCCCATGCATTTGTAATAATCTTCTTCATAGTAATACCCATCATAATCGGAGGCTTTGGCAACTGACTAGTTCCCCTAATAATCGGTGCCCCCGATATGGCGTTTCCCCGCATAAACAACATAAGCTTCTGACTCTTACC


Se ejecuta durante un total de 0.0029973983764648438segundos
```