PT

Yvo Hu s2962802

February 2021

1 Introductie

In deze opdracht maken we een bibliotheek met functies voor het doen van berekeningen met DNA-sequenties. Omdat het DNA-alfabet erg klein is (A, C, G en T) en DNA sequenties vaak erg lang (het langste menselijke chromosoom is 250MB) zijn, implementeren we verschillende manieren van het opslaan van de DNA-sequentie.

De vier symbolen in het DNA-alfabet kunnen in twee bits worden gerepresenteerd. Dat maakt het mogelijk om meerdere symbolen in een C++ integer op te slaan (packing). Het precieze aantal symbolen per integer is afhankelijk van de grootte van de integer. De integers in C++ hebben geen vaste grootte.

Om de geschreven code te testen implementeren we 3 opslagtypes(string, array, dubbele pointerlijst) voor de sequentie die elk de volgende methoden bevatten:

length() — geeft de lengte van de sequentie in symbolen;

char at(pos) — geeft het symbool op de positie pos;

concat(seq) — geeft een nieuwe sequentie waarbij de huidige sequentie is geconcateneerd met de sequentie seq;

slice(start, end) — geeft een nieuwe sequentie met de symbolen [start, end);

equal(seq) — checkt of twee sequenties hetzelfde zijn.

2 Opdracht

We maken een klasse aan voor ieder van de 3 opslagtypes met vergelijkbare functies maar aandere implementaties. Ieder van deze klassen manipuleert de sequentie op een andere manier.

String klasse:
Hier zijn alle symbolen opgeslagen als individuele karakters in een string.

Array klasse:
Hier zijn alle symbolen opgeslagen als een binair getal in een integer in een reeks van integers.

Pointerlijst klasse:
Hier zijn alle symbolen opgeslagen als een binair getal in een integer in een dubbel verbonden pointerlijst van integers.

In de array en pointerlijst klasse zijn de symbolen als binair getal als volgt opgeslagen:

A: 00
B: 01
C: 10
D: 11

De lengte van een sequentie wordt tevens ook bijgehouden in een aparte variabele symbolCount zodat aan het eind van de sequentie in de laatste integer de laatste binaire getallen, indien deze niet volledig gevuld is met symbolen, niet in het resultaat verwerkt worden.

We maken bovenop deze 3 klassen ook nog een aparte variant klasse voor ieder van de 3 opslagtypes die allemaal inheriten van een gezamenlijke variant klasse. Deze variant klassen bevatten de informatie die nodig is om de huidige sequentie te manipuleren. Deze klassne bevatten de volgende informatie:

start: Het begin van de positie in symbolen in de sequentie die bewerkt moet worden

end: Het eind van de positie in symbolen in de sequentie die bewerkt moet worden

type: De soort operatie (insertie, deletie, inversie of een combinatie van deze)

sequence: Eventuele sequentie die toegevoegd moet worden bij de operatie insertie

3 Makefile

Om deze library te maken maken we een makefile aan die de verschillende soruce files compileert in object files, deze linkt en vervolgens een shared object file van maakt. Deze wordt uiteindelijk ook gelinkt met de executable.

4 Testprogramma

Het testprogramma biedt de gebruiker de mogelijkheid om verschillende opslagtypes en unsigned integer types te gebruiken voor het manipuleren van de sequenties en deze te testen.

Ten eerste moet men een opslagtype kiezen voor de sequentie (string, array, pointerlijst).

Daarna volgt het integertype (8,16,32,64 bit).

Vervolgens specificeer je het bestand waar de sequentie in plain text is opgeslagen.

En vervolgens specificeer je nog een bestand waar de te gebruiken varianten in opgeslagen zijn in formaat (start, end, type, sequentie).
Er wordt voor de operatie ins niets gedaan met de "end" argument, en er worden voor de operaties del en inv niets gedaan met het "sequentie" argument.

Het resultaat van de toepassing van de varianten is opgeslagen in een nieuw aangemaakt bestand genaamd RESULT.txt.

Ten slotte is er nog de mogelijkheid om varianten gelijk in te lezen vanuit de console, deze op toe te passen op de huidige sequentie, en deze op te slaan in een bestand genaamd VARIANTS.txt.

Men moet bij het invoeren van varianten er goed op letten dat de data in het juiste formaat wordt geleverd.

5 Hoe bouw je de library

Extract de inhoud van het project ergens in 1 folder. Vervolgens open je een console en zorg dat de huidige pad in die folder zit. Typ vervolgens de command "make libpt3.so" in en de library is gebouwd.

6 Hoe bouw en link je het testprogramma

Doe eerst de stappen zoals uitgelegd in het stuk "Hoe bouw je de library"
Typ vervolgens de command "make pt3" om het testprogramma te bouwen, en vervolgens "make link" om het testprogramma te linken met de library.

Alternatief:
Typ de command "make" en alle 3 de vorige stappen worden achter elkaar uitgevoerd.

Om het testprogramma uiteindelijk uit te voeren typ je de command "export LD_LIBRARY_PATH=." zodat de library gevonden kan worden, en vervolgens ./pt3 om het testprogramma te starten.

7 Wie heeft wat gedaan?

De auteur, Yvo, heeft alles zelf gedaan. De git repo link:https://git.liacs.nl/s2962802/pt3