МАТЕМАТИЧКИ ФАКУЛТЕТ
УНИВЕРЗИТЕТ У БЕОГРАДУ

Нова метода за асемблирање генома на основу ПФГ електрофорезе

Мастер рад

Ментор:
Проф. др Ненад Митић

Студент:
Александар Вељковић 1090/2014

Београд, 2016.

Садржај

Захвалница

Захваљујем се лабораторији за молекуларну микробиологију Института за молекуларну генетику и генетичко инжењерство Универзитета у Београду на уступљеним материјалима и саветима без којих ова рад не би био написан.

1. Увод

Педесетих година двадесетог века откривена је структура молекула ДНК, есенцијаланог за постојање живота на земљи. Већ двадесетак кодина касније, секвенцирањем ДНК омогућено је читање података сачуваних у полинуклеотидном ланцу. Асемблирањем прочитаних секвенци, омогућено је сагледавање ДНК секвенце организма у целини. Како је процес гел електрофорезе један од елемената првих метода секвенцирања ДНК, није неочекиван покушај примене истог процеса за потребе асемблирања. Овај рад има за циљ да представи нову методу за асемблирање и поправљање лоше асемблиране кциркулатне ДНК, која је заснована на броју и величинама фрагмената добијених гел електрофорезом. Слични покушаји асемблирања су вршени ручно, што је трајало месецима, док се коришћењем ове методе време асемблирања може свести, у неким случајевима, на чак неколико секунди, уз испуњеност предуслова потребних за примену методе.

2. Електрофореза

Електрофореза је процес у коме наелектрисане честице, путују кроз подлогу услед деловања електричног поља [1]. Наелектрисане честице путују различитим брзинама, у зависности од облика, масе или количине наелектрисања честице и вискозност средине. На основу дужине пређеног пута честице, могуће је утврдити особине честице које су утицале на брзину кретања кроз раствор. Електрофореза се, између осталог, користи за раздвајање протеина и фрагмената ДНК молекула.

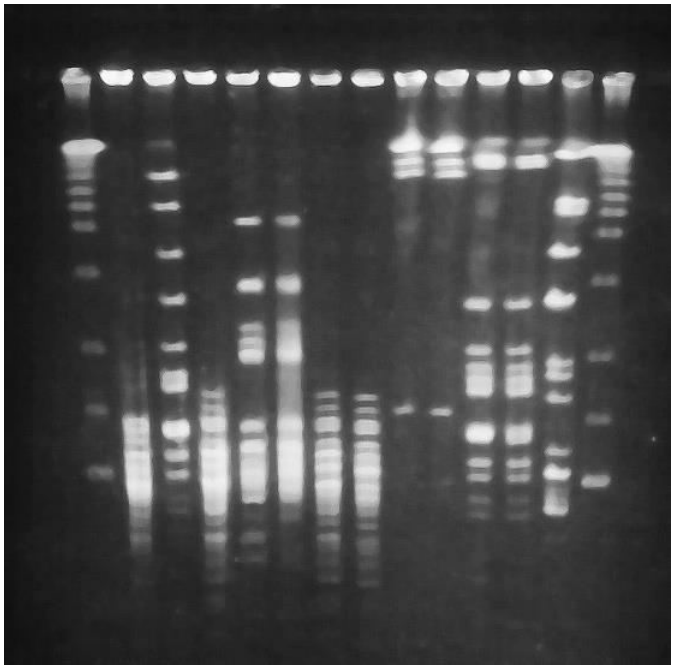
2.1 Гел електрофореза

Најчешће коришћени облик подлоге за примену у електрофорези је гел. Основна особина гела је његова порозност, односно величина пора кроз које пролазе честице. У зависности од захтеване порозности гела, користе се различита једињења за припрему раствора. За припрему порознијег гела користи се 0.1% до 1% раствор агарозе. Порастом концентрације агарозе у гелу величина пора опада. Нпр. концентрација 0,16% даје поре величине око 500 nm, док концентрација 1% даје поре величине око 150 nm [2]. Овакав раствор се користи за раздвајање крупнијих честица, из разлога што веће поре у гелу омогућавају лакше кретање крупнијих честица. За раздвајање ситнијих честица, користи се полиакриламидни гел. Полиакриламид настаје полимеризацијом акриламида и са NN'-метилен-бис акриламидом, који има улогу умреживача и гради порозну структуру гела. Степен умрежености одређује величине пора. Већа количина умреживача резултује мањим порама.

На једној страни гела урезују се канали у које се уливају честице које се раздвајају. На истој страни, где се налазе канали, постављена је електрода поларитета истог као поларитет честица које се раздвајају. Са друге стране гела постављена је електрода супротног поларитета што омогућује кретање честица кроз гел, честице се крећу од електроде истог поларитета ка електроди супротног поларитета. Како би се на основу пређеног пута честица кроз подлогу утврдиле њихове особине, потребно је упоредити их са честицама чије су особине познате. У један од канала на гелу улива се молекулска лествица, која садржи честице чије су особине познате и у односу на које је могуће утврдити особине непознатих честица које се анализирају. Након 30-45 минута од укључивања електричног поља, процес електрофорезе је завршен и непознате честице и

молекулска лествица су прешле одређени пут кроз гел. Да би њихове позиције на гелу биле видљиве потребно је обојити их одговарајућим бојама, у зависности од честица које се испитују. Прећени пут непознатих честица се упоређује са прећеним путем честица молекулске лествице и тиме је целокупан процес завршен.

Прећени пут фрагмената ДНК приликом електрофорезе је логаритамска функција величине фрамената. У случају већих молекула ДНК, овај однос величине и прећеног пута се губи и стандардна гел електрофореза губи на ефикасности. Из тог разлога, 1984. године на универзитету Колумбија, Давид Шварц (*David C. Schwartz*) и Чарлс Кантор (*Charles Cantor*) развили су модификацију стандардне гел електрофорезе коришћењем електричног поља променљивог правца [3]. Електрично поље се мења у три правца у правилним временским интервалима. Овако модификована метода се назива гел електрофореза у пулсном пољу (енг. *Pulsed Field Gel Electrophoresis, PFGE*). Применом ове методе постигнуто је ефикасније раздвајање крупнијих ДНК молекула. Са друге стране, време потребно за обављање електрофорезе је дуже, у односу на стандардну гел електрофорезу, зато што се честице не крећу праволинијском путањом. Електрофореза у пулсном пољу данас представља "златни стандард" за профилирање (енг. *Fingerprinting*) ДНК секвенци. На слици 1 је приказан пример филма, са осветљеним сегментима који одговарају величинама фрагмената ДНК, добијен као резултат процеса гел електрофорезе у пулсном пољу.



Слика 1. Филм са осветљеним сегментима, добијен гел електрофорезом у пулсном пољу

2.2 Рестрикциони ензими

Рестрикциони ензими или рестрикционе ендонуклеазе су молекули који секу молекул ДНК на рестрикционим местима специфичним за ензим. Рестрикциона места су секвенце нуклеотида, дужине 4-8 нуклеотида, најчешће палиндромског типа. Молекули рестрикционог ензима проналазе своја одговарајућа рестрикциона места на ДНК молекулу, везују се за њих и секу фосфодиестарске везе између нуклеотида на рестрикционом месту, чиме се ланац ДНК прекида. Процес сечења ДНК рестрикционим ензимом назива се дигестија ензима и врши се на оба ланца ДНК.

Рестрикциони ензими се у природи налазе у оквиру одбрамбених механизама бактерија и археа. Приликом напада бактериофага рестрикциони ензими бактерије секу уметнуту ДНК вируса и онеспособљавају његово дејство. Ензим метилаза, процесом метилације, додаје метил групу нуклеотидима бактеријске ДНК, чиме се онемогућава везивање рестрикционог ензима на сопствену ДНК и спречава њено сечење.

Рестрикциони ензими су откривени 1970. године, од стране Вернера Арбера, Хамилтона Смита и Даниела Нејтанса, за чије су откриће 1978. добили Нобелову награду. До сада је откривено више од 3000 различитих рестрикционих ензима. Један од првих откривених рестрикционих ензима био је EcoRI ензим, пронађен у бактерији *E.Coli*. EcoRI се везује за рестрикционо место облика GAATTC (одговарајуће место на другој нити ДНК је облика CTTAAG, што представља палиндром). Дигестијом се секвенца ДНК на рестрикционом месту дели на делове:

G | A A T T C
C T T A A | G

Једна од одновних примена рестрикционих ензима у билогији је сечење ДНК молекула као припрема за процес електрофорезе.

2.3 Електрофореза ДНК

Електрофореза ДНК је процедура којом се идентификују, прочишћавају и мере фрагменти ДНК. ДНК се коришћењем рестрикционих ензима сече на фрагменте. Тако добијени фрагменти се уливају у канале са једне стране гела и електрофорезом раздвајају. Фрагменти су раздвојени на основу њихове величине и конформације. Већи фрагменти се

крећу спорије кроз гел док се мањи фрагменти крећу брже и прелазе већи пут кроз подлогу. ДНК плазмида може постојати у три конформације, супернамотај (енг. *supercoil*), отворена-циркуларна (енг. *open-circular*) и линеарна. Супернамотај је природна затворена циркуларна конформација ДНК плазмида и креће се најбрже кроз гел. Линеарна ДНК се креће кроз гел једним крајем напред, али је трење које настаје веће у поређењу са супернавојем, па се зато ДНК са линеарном конформацијом креће спорије кроз гел у односу на супернавој. Отворена-циркуларна ДНК је пресечена циркуларна ДНК која је задржала свој циркуларни облик, али, због пресека ланца ДНК, може мењати свој облик. Ова конформација се због највећег трења најспорије креће спорије кроз гел. Ланци ДНК са конформацијом супернавој и отворена-циркуларна, сечењем рестрикционим ензимом се редукују до линеарне конформације.

3. Секвенцирање

Секвенцирање ДНК представља заједнички назив за методе помоћу којих се утврђује тачан редослед нуклеотида у молекулу ДНК. Први метод секвенцирања ДНК, заснован на продужетку прајмера, развио је кинески научник Реј Ву (*Ray Wu*) 1974. године, резултујући парцијалном секвенцом бактериофага Т4 [5]. Три године касније, 1977. године Фредерик Сангер проналази начин за секвенцирање ДНК, заснован на прекидима синтезе молекула ДНК, за чији проналазак 1980. добија Нобелову награду за хемију [6]. Паралелно са окрићем Сангеровог метода, Алан Максам (*Allan Maxam*) и Валтер Гилберт (*Walter Gilbert*) у периоду од 1976. до 1977. године долазе до још једне методе секвенцирања, полазећи у супротном смеру у односу на Сангеров метод, деградацијом полазне ДНК [7]. У најранијем периоду од настанка обе методе, Максам-Гилбертова метода бива популарнија у односу на Сангерови, али убрзо, унапређењем поступака коришћених у Сангеровој методи, она постаје доминантна и достиже се ефикасност секвенцирања од око 1500 секвенцираних базних парова по особи годишње. Аутоматизацијом процеса Сангерове методе и оптимизацијом саме методе, развија се технологија капиларног секвенцирања, коришћењем капиларне електрофорезе [8], која свој врхунац достиже у пројекту "Људски геном" (енг. *The Human Genome Project*) [9]. Капиларним секвенцирањем, ефикасност секвенцирања, деведесетих година двадесетог века, расте на чак 150000 базних парова по машини годишње. Методе које су се развиле 2000-тих година, по завршетку пројекта људског генома, називане су "Секвенцирање следеће генерације" (енг. *Next Generation Sequencing*) и представљају најсавременије методе секвенцирања ДНК, са до више десетина милијарди секвенцираних базних парова по машини годишње.

3.1 Сангерова метода и капиларно секвенцирање ДНК

Сангерова метода се заснива на прекидању синтезе ланца ДНК и добијању префиксних ланаца. Прекидање полимеризације врши се уметањем нуклеотида након којих ДНК полимераза не може да настави синтезу ланца. Поступак се врши посебно за сваки од четири нуклеотида. Тако добијени ланци се обрађују процесом електрофорезе и читањем величина префиксних ланаца са резултата електрофорезе могуће је утврдити тачан редослед нуклеотида у ДНК секвенци.

Нуклеотиди се, у свом природном обику, састоје од пентозе, азотне базе везане за

1' угљеников атом, хидроксилне групе на 3' угљениковом атому и фосфатне групе на 5' угљениковом атому. За 2' угљеников атом, уместо хидроксилне групе, везан је само водоников атом. Због тога се нуклеотиди називају дезоксинуклеотид трифосфати, dNTP, где се под нуклеотидом подразумева нека од четири азотне базе - аденин (dATP), цитозин (dCTP), гуанин (dGTP) и тимин (dTTP). Нуклеотиди се процесом полимеризације везују у ланце и то везивањем фосфатне базе једног нуклеотида и хидроксилне групе другог нуклеотида. Како се фосфатна база налази на 5' угљениковом атому а хидроксилна група на 3', ланац који се добија полимеризацијом има два краја и расте од 5' ка 3'. ДНК полимераза синтетише нови ланац ДНК почевши од прајмера у правцу 5' ка 3'. Модификацијом нуклеотида, која се огледа у дезоксигенисању хидроксилне групе везане за 3' угљеников атом, добија се 2'3'дидезоксинуклеотид трифосфат, или ddNTP. Овако модификован нуклеотид, због недостатка хидроксилне групе на 3' угљениковом атому, нема могућност даљег формирања ланца.

Процес секвенцирања почиње денатурацијом ДНК. Процес денатурације ДНК представља раздвајање двоструког хеликса ДНК на два одвојена ланца, применом високе температуре на којој се водоничне везе између ланаца разбијају. Затим се за један од добијених ланаца везује полазни прајмер и у присуству ДНК полимеразе и нуклеотида врши синтеза другог ланца ДНК. Овај процес подсећа на ПЦР (*PCR, Polimerase Chain Reaction*) али има једну битну разлику. Међу нуклеотидима се налази мали проценат (око 1%) модификованих, ddNTP, нуклеотида који, када се вежу у ланац, онемогућавају ДНК полимеразу да настави са синтезом. Присуством малог процента ddNTP у окружењу, на појединим ланцима се прекида синтеза, јер се на одговарајућем месту уместо dNTP нашао ddNTP, док остали ланци настављају синтезу све док се на следећем месту уместо одговарајућег нуклеотида не веже ddNTP. По завршетку овог процеса добијени су сви ланци који се могу добити прекидањем полимеризације на сваком месту на коме је могао да се веже ddNTP. Поступак се врши појединачно за сваки од ddNTP (ddATP, ddGTP, ddCTP, ddTTP) и добијени ланци сваке итерације се уливају у посебан канал на гелу за електрофорезу. По завршетку гел електрофорезе добијени резултат је сортиран редослед дужина прекинутих ланаца одговарајућим модификованим нуклеотидом. Како је познато који модификовани нуклеотид је прекинуо ланац, читањем резултата електрофорезе, од најкраћег до најдужег фрагмента, добија се тачан редослед нуклеотида на другом ланцу полазне ДНК, чиме је секвенцирање завршено. У почетку је бојење фрагмената раздвојених електрофорезом вршено радиоактивним изотопом ^{32}P , па је и сам процес добијања резултата био дужи, јер је било потребно пресликати добијене позиције фрагмената на рендгенски филм.

Оптимизација Сангерове методе почела је заменом ^{32}P у процесу бојења. Радиоактивни изотоп је замењен флуоресцентним бојама, при чему су различити дидезоксинуклеотиди бојени различитом бојом [10]. Оваквом заменом, поред напорног процеса добијања резултата електрофорезе, елиминисана је и потреба за вршењем четири појединачна процеса прекидања полимеризације већ се посао обављао истовремено за сва четири дидезоксинуклеотида. Елиминисана је и потреба за четири различите траке на гелу за електрофорезу па се електрофореза вршила у капиларним цевима [11]. Процес је аутоматизован увођењем ласера и флуоресцентног детектора. Када ласерски зрак погоди флуоресцентну боју којом су обојени дидезоксинуклеотиди, емитује се светлост одговарајуће таласне дужине за ту боју коју флуоресцентни детектор детектује. На основу таласне дужине светлости одређује се који је нуклеотид погођен ласером. Процес аутоматског секвенцирања је додатно паралелизован на 96 до 384 реакције истовремено.

3.2 Методе секвенцирања следеће генерације

Методе секвенцирања следеће генерације обухватају неколико метода секвенцирања високе пропусне моћи. Почетни корак секвенцирања је припрема узорка. Узорак ДНК који се секвенцира се разбија на ситне делове у процесу фрагментације, након чега се одмах врши амплификација или се прво на оба краја делова разбијене ДНК додају додатне кратке секвенце, адаптери. Адаптери омогућавају везивање кратких, денатуризованих секвенци за подлогу на којој се врши секвенцирање. Следећи корак је кластеровање, умножавање, кратких секвенци на подлози. За овај корак постоје два приступа, ПЦР премошћавање (eng. *Bridge PCR*) и емулзиони ПЦР (енг. *Emulsion PCR*).

Код процеса ПЦР премошћавања, адаптери са оба краја кратких ланаца ДНК се везују за подлогу, образујући лукове секвенци. У присуству ДНК полимеразе и нуклеотида, врши се синтеза другог ланца ДНК. Лукови се, затим, денатуризују, чиме се добијају два одвојена ланца везана за подлогу. Овај процес се понавља док се не достигну кластери од више милиона кратких ланаца везаних за подлогу.

Емулзиони ПЦР користи капљице са адаптерима комплементарним оним који су надовезани на крајеве денатуризованих ланаца ДНК. У присуству емулзионог уља, денатуризованих ДНК ланаца са адаптерима и реагенаса потребних за синтезу новог ланца ДНК, капљице са комплементарним адаптерима се издвајају у мехуриће уља. Унутар сваког од мехурића ланци ДНК се везују за комплементарне адаптере и синтетише се

други ланац. Након синтезе, новодобијени ланац се одваја и везује на другом месту за адаптер на капљици. Понављањем поступка везивања, синтезе и поновног одвајања умножавају се ланци ДНК.

Наредни корак у процесу секвенцирања је корак самог секвенцирања кратких, кластерованих секвенци. Прва метода секвенцирања следеће генерације, пиросеквенцирање, појавила се 2004. године. Поред пиросеквенцирања, најзаступљеније методе секвенцирања су секвенцирање синтезом (енг. *Sequencing by synthesis, SBS*), што је тренутно најпопуларнији метод, секвенцирање лигацијом (енг. *Sequencing by ligation*) и секвенцирање јонским полупроводником (енг. *Ion semiconductor sequencing*). У последњем кораку процеса секвенцирања, сирови подаци добијени секвенцирањем се асемблирају у односу на референтни геном.

4. Асемблирање

Асемблирање генома је процес слагања и надовезивања ДНК секвенци карактера у веће целине, идеално у јединствену целину која одговара полазној секвенцираној ДНК. Најмање целине које се добијају асемблирањем су контиге. Контига представља низ карактера добијен преклапањем кратких, сирових, секвенци добијених секвенцирањем, тако да редослед карактера у секвенци одговара редоследу нуклеотида у ДНК. Како су неке од сирових секвенци у обрнутом редоследу од стварног редоследа нуклеотида, потребно је извршити њихову реоријентацију при спајању у контигу. Контиге се, затим, спајају у веће целине, скафолде или суперконтиге. Скафолди могу садржати празнине између контига, али се и даље захтева да редослед контига и њихових карактера одговара редоследу нуклеотида секвенциране ДНК. Асемблирање се може вршити коришћењем референтног генома. Референтни геном или референтна секвенца је ДНК секвенца која представља репрезентативну модел секвенцу за одређени организам. Контиге се асемблирају и спајају у скафолде на основу редоследа нуклеотида у референтном геному. Други облик асемблирања је асемблирање без коришћења референтног генома, *de novo* асемблирање.

4.1 *De novo* асемблирање

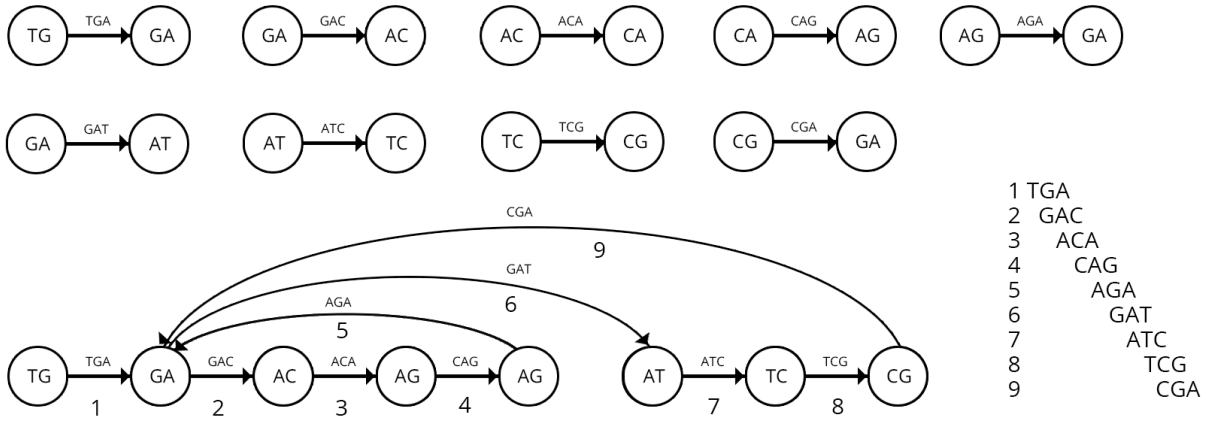
Решавању проблема *De novo* асемблирања се може приступити на више начина. Један од начина је формирању графа и проналажењу циклуса или пута кроз граф, при чему је редослед обиласка чворова аналоган асемблираној секвенци. Природан приступ решавању проблема асемблирања помоћу неусмереног графа био би издвајање свих n-грама из кратких ДНК секвенци добијених од секвенцера, уређаја којим је вршено секвенцирање. У том случају, чворови графа су n-грами добијени од кратких ДНК секвенци, повезани међусобно граном уколико је један n-грам префикс другог. Проналажењем Хамилтоновог пута у тако формираном графу добија се редослед обиласка који даје редослед n-грама у најдужој секвенци која се може саставити од тих n-грама, што представља тражену, асемблирану секвенцу. Проблем код оваквог приступа је проналажење Хамилтоновог пута, што је NP комплетан проблем. Други, мање интуитиван, приступ био би формирање усмереног графа чије су гране n-грами кратких ДНК секвенци а чворови префикси и суфикси n-грама који одговара суседној грани. Чворови са истим префиксима, односно

суфиксима, се спајају и у формираном графу се тражи Ојлеров пут. Како је налажење Ојлеровог пута лакши проблем од проналажења Хамилтоновог циклуса, овај приступ се у пракси показује као бољи избор.

Пример:

Нека је секвенца TGACAGATCGA подељена на 3-граме.
TGA GAC ACA CAG AGA GAT ATC TCG CGA

Одговарајући чворови графа са суседним гранама приказани су на слици 2.



Слика 2. Формирање графа помоћу n-грама и пронађени Ојлеров пут.

Спајањем чворова са истим суфиксом, односно префиксом добија се граф представљен на слици 1 и гране Ојлеровог циклуса пронађеног у графу означене су бројевима. Тако добијени пут одговара редоследу 3-грама у асемблираној секвенци.

5. Асемблирање генома на основу гел електрофорезе у пулсном пољу

Резултати добијени применом гел електрофорезе у пулсном пољу на ДНК представљају величине врагмената ДНК добијених сечењем ДНК рестрикционим ензимом. Раздвајањем асемблиране ДНК секвенце на рестрикционим местима, за која се везује рестрикциони ензим коришћен приликом електрофорезе, добијају се ниске карактера чије су дужине аналогне величинама фрагмената добијених електрофорезом. Уколико је ДНК секвенца лоше асемблирана, сечење секвенце на рестрикционим местима неће дати одговарајуће величине фрагмената. Грешке у асемблирању се најчешће одликују лошим редоследом контига у скафолдима и празнинама између контига, услед лошег поравнања са референтном секвенцом. Тежи облик грешака при асемблирању су лоше асемблиране контиге. Један од разлога лошег поравнања може бити појава инверзије у оквиру генома, која може резултовати инвертованим контигама у целини или њихових делова. Из тог разлога, сортирање контига искључиво на основу поравнања са референтном секвенцом не даје увек исправне резултате. Коришћењем информација о стварним величинама фрагмената, добијених електрофорезом, могуће је променити редослед делова ДНК секвенце тако да се при поновном сечењу преуређене секвенце, на истим рестрикционим местима, добијају ниске карактера дужине аналогне величинама фрагментима добијених електрофорезом. Оваквим преуређивањем, може се добити исправно асемблирана секвенца.

Први проблем на који се наилази при покушају асемблирања секвенце ДНК на основу величина фрагмената добијених електрофорезом је прецизност резултата електрофорезе. Величине фрагмената се читају са филма и пореде у односу на молекулску лествицу. Како су величине фрагмената означене светлијим сегментима неравних ивица, на филму, није могуће потпуно прецизно утврдити величину фрагмента, већ грешка варира до ± 2% од стварне величине фрагмента. Са порастом величине фрагмента, грешка процене величине фрагмента је све већа. Други проблем је могућност постојања више фрагмената исте величине или приближних величина у оквиру исте ДНК. У том случају може бити отежано распознавање ивица светлијих сегмената и утврђивање да ли један сегмент представља један или више фрагмената. Да би се овакви проблеми превазишли, потребно је дозволити одређену грешку величина при асемблирању. Дозвољена грешка се може усклађивати пре процеса асемблирања у зависности од оштећености ДНК секвенце. Коришћењем рачунарских ресурса могуће је аутоматизовати овај процес и у наредном одељку је приказана нова метода асемблирања која врши поправку лоше асемблираних

секвенци, али и асемблирање генома уопште.

5.1 Опис методе

Методи асемблирања претходи гел електрофореза ДНК у пулсном пољу. Након добијања броја и величина фрагмената анализиране ДНК прелази се на методу асемблирања. Почетни корак методе је припрема секвенци за асемблирање. Ако не постоји иницијално асемблирана секвенца, већ су доступне само контиге, врши се сечење контига на местима рестрикционог ензима коришћеног у процесу електрофорезе. За поправљање лоше асемблиране секвенце, иницијално асемблирана ДНК секвенца се дели на скафолде а скафолди на контиге и добијене контиге се секу на рестрикционим местима. У даљем тексту ће се контиге и делови добијени сечењем контига називати јединственим називом полуфрагменти. Полуфрагменти се означавају на основу начина добијања. Контига која не садржи рестрикционо место означава се као средишњи полуфрагмент, зато што се у асемблираној секвенци непосредно после или пре ње не налази рестрикционо место. Ако контига садржи рестрикционо место, полуфрагменти се означавају на три могућа начина. Полуфрагмент који се налази непосредно пре рестрикционог места је последњи полуфрагмент у оквиру фрагмента ограниченог тим рестрикционим местом па се означава као затварајући. Аналогно томе, први полуфрагмент после рестрикционог места се означава као отварајући. Уколико се непосредно пре и непосредно после полуфрагмента налазе рестрикциона места, такав полуфрагмент се означава као комплетан. Комплетни полуфрагменти су исправно асемблирани и могу се изузети из наредног корака асемблирања. Број отварајући и затварајућих полуфрагмената мора бити исти као број фрагмената добијених електрофорезом. Полуфрагментима се одређују почетне позиције при BLAST поравнању са референтном секвенцом, што ће се у наредном кораку искористити за примену хеуристике.

Други корак методе је распоређивање полуфрагмената у групе чије величине одговарају задатим величинама фрагмената добијених електрофорезом, тако да укупна величина контига не прелази задату величину групе, уз дозвољену грешку. Како алгоритам за проналажење свих могућих распореда контига по групама припада класи експоненцијалне временске сложености а сродни проблеми проналажења сума подскупова и ранца су NP комплетни или псеудополиномијални, неопходна је употреба хеуристика и ограничења којима се домен претраге могућих распореда контига сужава. У наведеној имплементацији алгоритма, као хеуристика искоришћено је рачунање грешке BLAST поравнања контига у оквиру групе и постављање ограничења за максималну дозвољену

грешку. Грешка BLAST поравнања рачуна се као укупна дужина поравнања полуфрагмената из групе подељена са укупном дужином свих полуфрагмената у групи. Рачунање грешке поравнања у оквиру група смањује могућност грешке асемблирања услед појаве инверзије, јер вредност грешке представља релативан однос полуфрагмената у групи, а не тачне позиције на којима се они морају наћи. Поред грешке поравнања, проверава се да ли укупна дужина свих полуфрагмената у групи одговара величини неког од фрагмената добијених електрофорезом, уз дозвољену грешку која се експлицитно наводи.

Групе се састављају поштујући одговарајућа правила. Свака група мора садржати тачно један отварајући и тачно један затварајући полуфрагмент. Иницијално се врши формирање група чији су први полуфрагменти отварајући фрагменти. Након тога, покушава се са додавањем средишњих и затварајућих полуфрагмената. Уколико је нови полуфрагмент средишњи и укупна дужина полуфрагмената у групи, са новим средишњим полуфрагментом, не прелази величину највећег фрагмента, група се копира и у новонасталу групу се додаје нови средишњи полуфрагмент. Додавање затварајућих полуфрагмената у групу врши се уколико величина свих полуфрагмената у групи, заједно са затварајућим који би се додао, одговара величини неког задатог фрагмента. Уколико је групи додат затварајући полуфрагмент и грешка BLAST поравнања полуфрагмената не прелази унапред дефинисану границу, група се евидентира као кандидат група, док се од полуфрагмената који нису затварајући формира нова група. Свака кандидат група чува информацију о томе ком фрагменту одговара и који полуфрагмент у оквиру групе је отварајући а који затварајући. Да би се избегла појава група у којима су смештени исти полуфрагменту у пермутованом редоследу, захтева се да полуфрагменти поштују растући поредак по редном броју који су добили при учитаву из улазне датотеке. Када су добијени сви могући кандидати који испуњавају наведене услове, следећи корак методе је одређивање правилног редоследа кандидат група у асемблираној секвенци.

Једна од кандидат група се проглашава за полазну и од ње се врши проналажење даљег редоследа преосталих кандидат група у асемблираној секвенци. Пошто је ДНК кружна, одабир полазне групе није стриктан. Како свака од кандидат група има информацију о томе који полуфрагмент је отварајући а који затварајући у оквиру групе и како је познато да су затварајући и отварајући фрагмент који се налазе непосредно поред рестрикционог места суседни у асемблираној секвенци, може се саставити граф при којем су групе са суседним отварајућим, односно затварајућим фрагментима повезане. Додатним ограничењима се елиминишу путеви кроз граф који сигурно не доводе до исправног редоследа. Ограничења су забрана постојања пута са две кандидат групе које

одговарају истом фрагменту и забрана постојања два иста полуфрагмента у оквиру целокупног пута. Да би пут био прихваћен као потенцијално исправни редослед кандидат група, потребно је да испуњава наведене услове и да је његова дужина једнака броју фрагмената добијених електрофорезом.

Овакав поступак обезбеђује формирање исправаног редоследа кандидат група, али исправан редослед полуфрагмената у оквиру група није загарантован, осим за отварајуће и затварајуће полуфрагменте у групама. За поправку се може искористи сортирање полуфрагмената у групама по почетним позицијама BLAST поравнања или сечење на другим рестрикционим местима и понављање поступка асемблирања, уз знање да су отварајући и затварајући полуфрагменти у групама на исправним позицијама и да су међусобно суседни у суседним групама.

Да би се метода могла применити захтева се испуњеност неколико предуслова. У почетно асемблираној ДНК секвенци, контиге морају бити исправно асемблиране јер се сматра да су њихове дужине иницијално исправне. Такође, неопходна је очуваност свих рестрикционих места ензима којим се врши сечење, у противном број група, у које се контиге распоређују, неће одговарати броју стварних фрагмената ДНК добијених сечењем истим ензимом. Рестрикциона места не смеју бити на самом почетку или самом крају контиге. Вредности ограничења задају се сразмерно броју и дужини контига које се асемблирају. Повећање броја и смањење величине контига захтева повећање дозвољене грешке при састављању кандидат група а самим тим и повећање броја кандидат група које задовољавају услов. Из тог разлога, лошији квалитет иницијално асемблиране секвенце коју треба поправити, у погледу броја и величина празнина, негативно утиче на брзину извршавања алгоритма. Још један од захтева је да је ДНК која се асемблира циркуларна, јер у супротном последњи фрагмент линеарне секвенце не би имао затварајући полуфрагмент, а првом фрагменту би недостајао отварајући. Ипак, уколико је познато који полуфрагменти су први и последњи у асемблираној секвенци, могу се додатно означити као отварајући, односно затварајући и метода би се у том случају могла применити, уз одговарајући одабир полазне кандидат групе при асемблирању.

Целокупан низ процедура, које се извршавају у оквиру описане методе асемблирања, приказан је на слици 3.

Слика 3. Редослед процедура у оквиру методе асемблирања генома на основу гел електрофорезе у пулсном пољу

5.2 Имплементација методе

Улазни подаци алгоритма су број и величине фрагената добијених гел електрофорезом у пулсном пољу, дозвољене грешке величине и BLAST поравнања, као и ензим коришћен приликом електрофорезе, тачније, опис рестрикционог места које ензим препознаје. Метода асемблирања је имплементирана у програмском језику C док је програм који врши дигестију ензима и дели иницијално асемблирану секвенцу на полуфрагменте имплементиран у језику Перл. Изворни код програма за асемблирање се налази у додатку А, док се изворни код Перл скрипте налази у додатку Б. Перл програм дели секвенцу на скафолде, скафолде на контиге а контиге, евентуално, на ситније делове, уколико се унутар контиге налазе рестрикциона места задатог ензима. Излаз програма за сечење секвенце су датотеке које садрже полуфрагменте и датотека са описима полуфрагмената. По завршетку сечења секвенце на полуфрагменте, из добијене датотеке, са описима полуфрагмената, учитавају се називи полуфрагмената, њихова ознака, величина и почетна позиција у BLAST поравнању са референтном секвенцом. Вредности почетних позиција BLAST поравнања је потребно посебно навести у оквиру датотеке. Полуфрагменти се чувају у листама и разврстани су на основу ознаке. Чвор такве листе је структура која чува податке:

- Назив полуфрагмента
- Ознака полуфрагмента

- Дужина полуфрагмента
- Почетна позиција полуфрагмента у BLAST поравнању
- Показивач на следећи чвор листе
- Редни број полуфрагмента при учитавању из датотеке

Након учитавања полуфрагмената врши се састављање кандидат група. Групе су, такође, представљене чворовима листе. Сви отварајући полуфрагменти отварају по једну групу. Чвор листе група је структура са подацима:

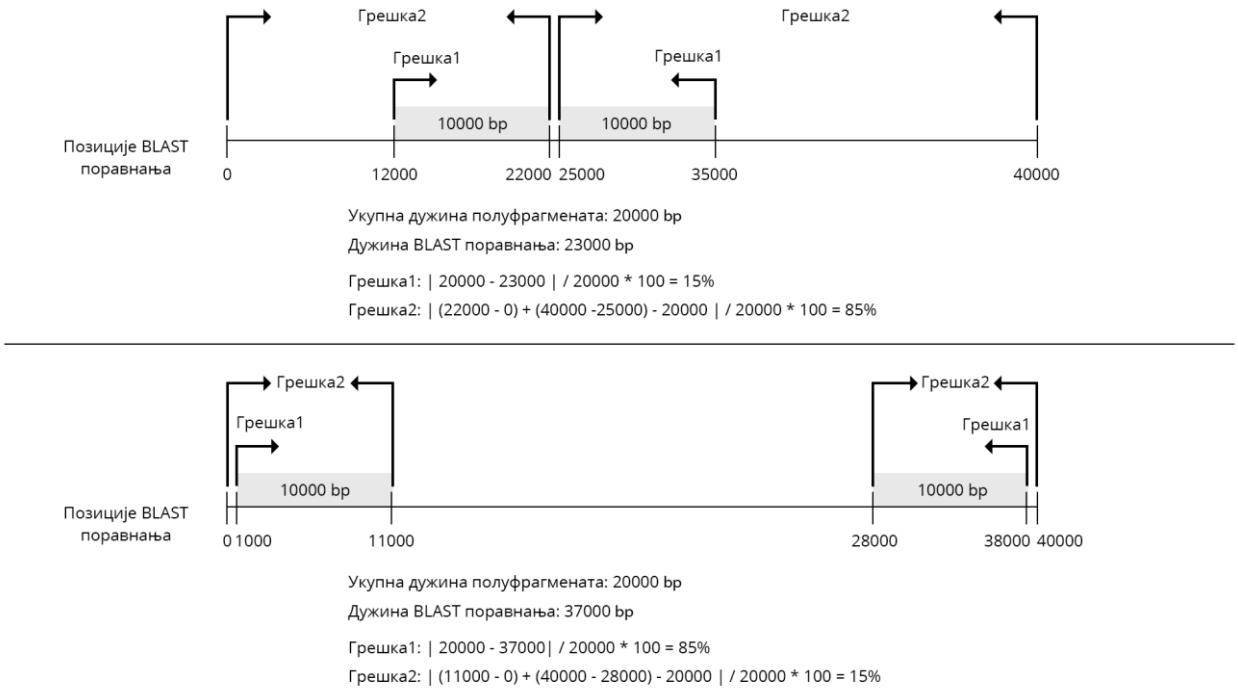
- Показивач на листу полуфрагмената који су смештени у групу
- Сума дужина свих полуфрагмената у групи
- Показивач на отварајући полуфрагмент у гурпи
- Показивач на затварајући полуфрагмент у групи
- Индекс скупа група у коме се налази суседна група у формираном графу
- Апсолутна разлика дужина свих полуфрагмената у групи и процењене дужине поравнања полуфрагмената у групи са референтном секвенцом.
- Величина фрагмента добијеног електрофорезом којој одговара група.

Затим се врши избор средишњих и затварајућих полуфрагмената и формирају остале групе, поштујући дозвољене грешке које се наводе као аргументи командне линије приликом позивања програма. Глобалне променљиве које чувају наведене грешке су:

- GRESKA - Дозвољено одступање величине групе од величине одговарајућег фрагмента, изражено у процентима
- BLAST_GRESKA - Дозвољени однос дужине BLAST поравнања и укупне дужине свих полуфрагмената у групи, изражена у процентима.

Приликом додавања затварајућег полуфрагмента у групу, проверава се да ли укупна дужина полуфрагмената у групи, заједно са затварајућим, одговара некој од величина фрагмената уз додатни услов да грешка BLAST поравнања не прелази задати проценат.
Како су полуфрагменти сортирани по величинама, тражење фрагмента коме би група одговарала врши се коришћењем бинарне претраге. Пошто је ДНК циркуларна, последња контига је суседна почетној контиги, па је при рачунању грешке BLAST поравнања потребно проверити грешку поравнања од првог полуфрагмента ка последњем, у односу на вредност почетка BLAST поравнања и грешку поравнања од краја ка почетку. Пример

могућих распореда контига и вредности грешака BLAST поравнања приказан је на слици 4.

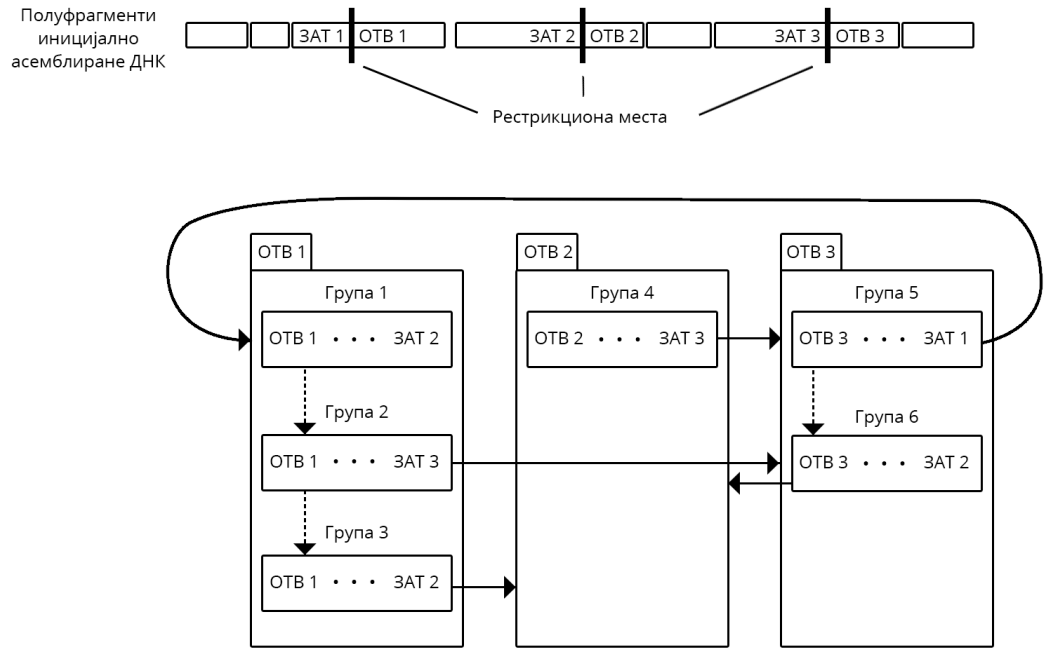


Слика 4. Примери рачунања грешака BLAST поравнања узимајући у обзир цикличну структуру ДНК

Уколико су сви услови испуњени, група се додаје у листу кандидат група. Након формирања свих група, листа кандидат група се претвара у низ који се сортира лексикографски по отварајућим полуфрагментима. Све кандидат групе са истим отварајућим полуфрагментима се смештају у листу предвиђену за тај отварајући полуфрагмент, док се показивачи на листе чувају у низу. Када су кандидат групе разврстане по листама, на основу отварајућих полуфрагмената, врши се повезивање група у оквиру листа са листама група у којима се налазе њихове суседне групе. Распоред група унутар листа и везивање са осталим листама приказано је на слици 5.

Произвољни скуп отварајућих фрагмената се проглашава за полазни и алгоритмом претраге се тражи пут кроз граф, крећући се по скуповима, преко грана добијених спајањем чворова листе група са скуповима суседних група, и одабирајући по једну групу из скупа. Пут је прихваћен као исправан уколико нема понављања група које одговарају истом фрагменту и ако не постоји ни један полуфрагмент који је на путу искоришћен више од једном. Особина исправног пута која се сама по себи поставља је да је група суседна последњој групи на путу управо прва група на путу, јер је ДНК секвенца

циркуларна.



Слика 5. Распоред група по листама на основу отварајућих полуфрагмената и везивање група са листама у којима се налазе суседни полуфрагменти.

Тренутна имплементација подразумева ручно уношење позиција BLAST поравнања полуфрагмената са референтном секвенцом, али се и овај процес може аутоматизовати.

5.3 Провера методе

За потребе тестирања исправности алгоритма, искоришћена је референтна секвенца бактерије *Lactococcus lactis subsp. lactis* IL1403, при чему су унутар секвенце додате празнине на случајно одабраним местима, али тако да се не оштети ни једно рестрикционо место. Овај процес је обављен како би секвенца одговарала лоше асемблираној секвенци. Секвенца је, затим, сечена на местима празнина и рестрикционим местима рестрикционог ензима SfiI, која су облика GGCCNNNNNGGCC. N представља произвољни нуклеотид. У секвенци је пронађено 7 рестрикционих места ензима SfiI. Дељењем секвенце на полуфрагменте добијено је 17 полуфрагмената, од тога 7 отварајућих, 7 затварајућих и 3 средишња. Датотека добијена као излаз Перл скрипте, допуњена вредностима почетних позиција BLAST поравнања, налази се у додатку Б. Задата дозвољена грешка величине групе била је 5% док је дозвољена грешка поравнања износила 2%. Асемблирање је

вршено на процесору i5-3230M са фреквенцијом од 2,6 GHz уз 8 GB DDR3 RAM меморије, на оперативном систему Linux Ubuntu 14.04. Након формирања група, добијено је 11 кандидат група. Групе су повезане у граф и пронађен је редослед група који одговара редоследу унутар исправно асемблиране секвенце IL1403. Сортирањем полуфрагмената унутар група, на основу BLAST поравнања, добијен је редослед контига који одговара исправно асемблираној секвенци IL1403. Време трајања извршавања алгоритма асемблирања износило је 0.001825s.

6. Закључак

Описана метода је пионирски корак и представља оквир за развој нових метода асемблирања заснованих на истим основама, али проширених новим хеуристикама и ефикаснијим алгоритмима и структурама података. Највећи допринос великој временској сложености алгоритма даје процес састављања могућих кандидат група. Ефикасност методе би се могла подићи увођењем паралелизације у фази формирања кандидат група и коришћењем фрагмената добијених употребом различитих рестрикционих ензима у процесу гел електрофорезе. Већи број фрагмената, добијених различитим рестрикционим ензимима, омогућио би постављање додатних ограничења при састављању кандидат група, што би смањило и број могућих кандидат група елиминишући део сигурно неисправних кандидата. Простор за даља унапређења је велики, али метода и у описаном облику доказује да је асемблирање ДНК на овај начин могуће.

7. Литература

[1] Иван П. , *Elektroforeza*, Уџбеници свеучилишта у Загребу, Загреб, 2006.

[2] Мара А., *Електрофореза 1*, наставни материјал, supa.pharmacy.bg.ac.rs/assets/26738, Београд, 2015

[3] Schwartz DC. , Cantor CR. , *Separation of yeast chromosome-sized DNAs by pulsed field gradient gel electrophoresis*, Cell, 37(1):67-75, 1984

[4] Editors of Encyclopædia Britannica, *Encyclopædia Britannica: Restriction enzyme*, Encyclopædia Britannica, https://www.britannica.com/science/restriction-enzyme, 2009

[5] Padmanabhan R. , Jay E. , Wu R. , *Chemical Synthesis of a Primer and Its Use in the Sequence Analysis of the Lysozyme Gene of Bacteriophage T4*, Proc Natl Acad Sci USA, 70(4):1209–1213, 1974

[6] Sanger F. , Nicklen S. , Coulson AR. , *DNA sequencing with chain-terminating inhibitors*, Proc Natl Acad Sci USA, 74(12), 5463-7, 1977

[7] Maxam AM. , Gilbert W. , *A new method for sequencing DNA*, Proc Natl Acad Sci USA, 74(2): 560-4, 1977

[8] Swerdlow H. and Gesteland R. , *Capillary gel electrophoresis for rapid, high resolution DNA sequencing*, Nucleic Acids Res, 18(6):1415–1419, 1990

[9] Shapiro LJ. , *Human genome project*, West J Med. , 158(2), 181, 1993

[10] Hood LE. et al , *Fluorescence detection in automated DNA sequence analysis*, 321(6071):674-9, 1986

[11] Jorgenson J.W. , Lukacs K.D. , *Zone Electrophoresis in Open-Tubular Glass Capillaries*, Journal of High Resolution Chromatography, 4:230-231, 1981

Додатак А - Изворни код програма за асемблирање

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>

#define MAX_NAZIV 100 // Maksimalna duzina naziva polufragmenta
#define MAX_TIP 15 // Maksimalna duzina oznakaa polufragmenta

#define OTV 1 // Otvarajući
#define SRE 2 // Središnji
#define ZAT 3 // Zatvarajući
#define KOM 4 // Kompletni


int GRESKA; // Dozvoljeno procentualno odstupanje veličine grupe od stvarne veličine fragmenta
int MAX_BLAST_GRESKA; // Dozvoljeno procentualno relativno odstupanje duzine BLAST poravnanja u
odnosu na duzinu polufragmenta

// Struktura koja opisuje polufragment
typedef struct _polufragment
{
    int rbr; // Redni broj polufragmenta
    int duzina; // Dužina polufragmenta
    int blast_pocetna_pozicija; // Pocetna pozicija BLAST poravnanja polufragmenta sa referentnom
sekvencom
    char naziv[MAX_NAZIV]; // Naziv polufragmenta
    int oznaka; // Oynaka polufragmenta
    struct _polufragment *sledeci; // Pokazivač na sledeći polufragment u grupi
} POLUFRAGMENT;

// Struktura koja opisuje grupu
typedef struct _grupa
{
    POLUFRAGMENT *polufragmenti; // Pokazivač na početak liste polufragmenata u grupi
    int ukupna_duzina; // Ukupna duzina polufragmenata u grupi
    struct _grupa *sledeci; // Pokazivač na sledeću grupu u listi
    POLUFRAGMENT *otvarajuci; // Pokazivač na otvarajući polufragment u grupi
    POLUFRAGMENT *zatvarajuci; // Pokazivač na zatvarajući polufragment u grupi
    int indeks_sledeceg; // Indeks sledećeg skupa u formiranju grafa
    float blast_razlika; // Greška BLAST poravnanja
    int odgovarajuci_fragment; // Veličina fragmenta kome odgovara grupa

    // Pokazivači korišćeni tokom formiranja stabla za izbacivanje duplikata
    struct _grupa* l;
    struct _grupa* d;
} GRUPA;

// Strukgura koja opisuje skup grupa
typedef struct {
    GRUPA* grupe; // Lista grupa u skupu
    GRUPA* odabrana_grupa; // Grupa koja je odabrana iz skupa prilikom obilaska grafa
    int rbr; // Redni broj odabrane grupe u obilasku grafa
} SKUP_GRUPA;


typedef struct {
    int *iskorisceni; // Kontrolni niz indikatora da li je i-ti fragment pronadjen tokom obilaska
grafa
} KONTROLA;

int global_min; // najmanja BLAST pocetna pozicija polufragmenata
int global_max; // BLAST pozicija kraja polufragmenta sa najvecom BLAST pocetnom pozicijom

POLUFRAGMENT *novi_polufragment(char* naziv, int oznaka, int duzina, int blast_poz); // Funkcija
kojom se alocira i inicijalizuje novi polufragment
void dodaj_polufragment(POLUFRAGMENT **lista, POLUFRAGMENT *novi); //Funkcija kojom se
polufragment dodaje u listu polufragmenata
POLUFRAGMENT* ucitaj(FILE* ulaz, POLUFRAGMENT **otvarajuci, POLUFRAGMENT **neotvarajuci,
POLUFRAGMENT **zatvarajuci, POLUFRAGMENT **kompletni); // Funkcija koja ucitava polufragmente iz
datoteke i razvrstava ih po tipu
```

```
GRUPA* sastavljanje_kandidata(int *fragmenti, int broj_fragmenata, POLUFRAGMENT *otvarajuci,
POLUFRAGMENT *neotvarajuci, POLUFRAGMENT *zatvarajuci, POLUFRAGMENT *kompletni); // Funkcija koja
sastavlja kandidat grupe

// Pomocne funkcije za poredjenje pri sortiranju
int poredi_fragmente(const void *a, const void *b);
int poredi_fragmente_2(const void *a, const void *b);
int poredi_otvarajuce(const void *a, const void *b);

GRUPA* nova_inicijalna_grupa(POLUFRAGMENT *pf); // Funkcija kojom se alocira grupa sa otvarajucim
polufragmentom
void dodaj_incijalnu_sumu(GRUPA** glava, POLUFRAGMENT *pf); // Funkcija kojom se u listu grupa
dodaje nova inicijalna grupa u listu
void dodaj_polufragment_u_sumu(GRUPA** glava, POLUFRAGMENT *pf); // Funkcija kojom se vrsi
dodavanje polufragmenta u sumu
GRUPA* nova_grupa(POLUFRAGMENT *pf, POLUFRAGMENT *otvarajuci, POLUFRAGMENT *zatvarajuci); //
Funkcija kojom se alocira nova, regularna grupa

int samozatvarajuci(POLUFRAGMENT* otvarajuci, POLUFRAGMENT* zatvarajuci); // Provera da li je
lista polufragmenata zatvorena zatvarajucim polufragmentom koji je susedan otvarajucem

GRUPA* dodaj_u_stablo(GRUPA* grupa, GRUPA* s); // Funkcija kojom se dodaje grupa u stabloo

POLUFRAGMENT* dodaj_u_sortiranu_listu(POLUFRAGMENT* lista, POLUFRAGMENT* novi); // Funkcija kojom
se dodaje polufragment u listu polufragmenata sortiranu po pocetnim BLAST pozicijama
void oslobodi(POLUFRAGMENT* sort);  // Oslobadjanje liste polufragmenata

void napravi_niz(GRUPA* stablo, int* i, GRUPA*** niz_grupa); // Funkcija kojom se pravi niz grupa
od stabla grupa
SKUP_GRUPA* napravi_skupove(GRUPA** niz_grupa, int n, int* m); // Funkcija kojom se razvrstavaju
grupe po otvarajucim polufragmentima u skupove

void obilazak(SKUP_GRUPA* grupe_grupa, int trenutna_grupa, int duzina_puta, int broj_fragmenata,
KONTROLA* kont); // Funkcija kojom se vrsi obilazak grafa i pronalaze moguci, regularni, putevi

int kolizija(SKUP_GRUPA* grupe_grupa, GRUPA* grupa, int n); // Funkcija kojom se proverava da li
je doslo do kolizije pri sastavljanju grupa - da li se jedan polufragment pojavljuje na putu vise
od jednom

int main(int argc, char **argv)
{
    if(argc != 4)
    {
        printf("Nisu navedeni svi argumenti, program se poziva sa argumentima: <ulazna datoteka>
<dozvoljena BLAST greska> <dozvoljena greska velicine>!\n");
        exit(EXIT_FAILURE);
    }

    MAX_BLAST_GRESKA = atoi(argv[2]);
    GRESKA = atoi(argv[3]);

    FILE* ulaz = fopen(argv[1],"r");
    if(ulaz == NULL)
    {
        printf("Greska prilikom otvaranja datoteke!\n");
        exit(EXIT_FAILURE);
    }

    // Niz fragmenata
    int *fragmenti;

    // Inicijalizacija lista polufragmenata
    POLUFRAGMENT *otvarajuci = NULL;
    POLUFRAGMENT *neotvarajuci = NULL;
    POLUFRAGMENT *zatvarajuci = NULL;
    POLUFRAGMENT *kompletni = NULL;

    // Inicijalizacija globalnog minimuma i maksimuma
    global_max = 0;
    global_min = INT_MAX;

    // Ucitavanje polufragmenata
    ucitaj(ulaz, &otvarajuci, &neotvarajuci, &zatvarajuci, &kompletni);

    int broj_fragmenata;

    // Ucitavanje broja fragmenata
    printf("Broj fragmenata: ");
    scanf("%d",&broj_fragmenata);
```

```
    if(broj_fragmenata <= 0)
    {
        printf("Neispravan broj fragmenata!\n");
        exit(EXIT_FAILURE);
    }

    // Alokacija niza fragmenata
    fragmenti = (int*)malloc(broj_fragmenata * sizeof(int));

    int i, j;
    // Ucitavanje velicina fragmenata
    for(i = 0; i < broj_fragmenata; i++)
        scanf("%d", &fragmenti[i]);


    // Sortiranje fragmenata po velicinama, za potrebe binarne pretrage
    qsort(fragmenti, broj_fragmenata, sizeof(int), poredi_fragmente);

    GRUPA *grupe = NULL;

    // Sastavljanje liste kandidat grupa
    grupe = sastavljanje_kandidata(fragmenti, broj_fragmenata, otvarajuci, neotvarajuci,
zatvarajuci, kompletni);

    GRUPA* stablo = NULL;


    // Prevodjenje liste u stablo, radi izbacivanja duplikata
    while(grupe)
    {
        stablo = dodaj_u_stablo(stablo, grupe);
        grupe = grupe->sledeci;
    }

    int velicina_niza = 0;
    GRUPA** niz_grupa = NULL;

    // Prevodjenje stabla u niz
    napravi_niz(stablo, &velicina_niza, &niz_grupa);

    // Sortiranje grupa na osnovu otvarajuceg polufragmenta
    qsort(niz_grupa, velicina_niza, sizeof(GRUPA*), poredi_otvarajuce);

    int broj_grupa;
    // Razvrstavanje grupa po skupovima
    SKUP_GRUPA* grupe_grupa = napravi_skupove(niz_grupa, velicina_niza, &broj_grupa);
    if(broj_fragmenata != broj_grupa)
       {
            printf("Nije moguće asemblirati genom sa postavljenim ograničenjima!\n");
            return 0;
       }

    int scaf1, scaf2, cont1, cont2, frag1, frag2;

     // Pravljenje grafa
     // Spajanje grupa sa susednim skupovima
      for(i = 0; i < broj_grupa; i++)
      {
          GRUPA* tmp = grupe_grupa[i].grupe;
          while(grupe_grupa[i].grupe != NULL)
          {

            sscanf(grupe_grupa[i].grupe->zatvarajuci->naziv,"fragment_%d_%d_%d", &scaf1, &cont1,
&frag1);
            for(j = 0; j < broj_grupa; j++)
            {
                sscanf(grupe_grupa[j].grupe->otvarajuci->naziv,"fragment_%d_%d_%d", &scaf2,
&cont2, &frag2);
                if(scaf1 == scaf2 && cont1 == cont2 && frag2 == frag1+1)
                {
                    grupe_grupa[i].grupe->indeks_sledeceg = j;
                    break;
                }
            }

            grupe_grupa[i].grupe = grupe_grupa[i].grupe->sledeci;
          }
          grupe_grupa[i].grupe = tmp;
       }
```

```
       KONTROLA kontrolni_niz;

       kontrolni_niz.iskorisceni = malloc(broj_fragmenata * sizeof(int));
       if(kontrolni_niz.iskorisceni == NULL)
       {
            printf("Greska pri alokaciji!\n");
            return -1;
       }

        // Inicijalizacija kontrolnog niza
       for(i = 0; i < broj_fragmenata; i++)
       {
           kontrolni_niz.iskorisceni[i] = 0;
       }

       // Obilazak grafa i trazenje puteva
       obilazak(grupe_grupa, 0, 0, broj_fragmenata, &kontrolni_niz);

    fclose(ulaz);
    return 0;

}

void obilazak(SKUP_GRUPA* grupe_grupa, int trenutna_grupa, int duzina_puta, int broj_fragmenata,
KONTROLA* kont)
{
    if(duzina_puta == broj_fragmenata) // Pronadjen je put
    {
        int i;
        for(i = 0; i < broj_fragmenata; i++) // Ispis puta
        {
                printf("RBR, %d\n", grupe_grupa[i].rbr);
                printf("Suma: %d, Blast razlika: %f \n", grupe_grupa[i].odabrana_grupa-
>ukupna_duzina, grupe_grupa[i].odabrana_grupa->blast_razlika);

                POLUFRAGMENT* tmp = grupe_grupa[i].odabrana_grupa->polufragmenti;

                while(grupe_grupa[i].odabrana_grupa->polufragmenti)
                {
                    printf("%s - %d\n", grupe_grupa[i].odabrana_grupa->polufragmenti->naziv,
grupe_grupa[i].odabrana_grupa->polufragmenti->duzina);
                    grupe_grupa[i].odabrana_grupa->polufragmenti = grupe_grupa[i].odabrana_grupa-
>polufragmenti->sledeci;
                }

                printf("-----------------------------------\n");

                grupe_grupa[i].odabrana_grupa->polufragmenti = tmp;

                printf("\n");
        }
        printf("\n\n================================================\n\n");
        return;
    }
    else // Put jos uvek nije formiran
    {
        GRUPA* tmp = grupe_grupa[trenutna_grupa].grupe;
        GRUPA* tmp2 = NULL;

        while(grupe_grupa[trenutna_grupa].grupe != NULL)
        {

            int kol = kolizija(grupe_grupa, grupe_grupa[trenutna_grupa].grupe, broj_fragmenata);

            int iskoriscena = 0;

            if(kont->iskorisceni[grupe_grupa[trenutna_grupa].grupe->odgovarajuci_fragment] != 0)
            {
                iskoriscena = 1;
            }

            // Provera da li se sledeca grupa moze dodati na trenutni put
            if((!iskoriscena) && (!kol) && (grupe_grupa[grupe_grupa[trenutna_grupa].grupe-
>indeks_sledeceg].rbr == 0 || (grupe_grupa[grupe_grupa[trenutna_grupa].grupe-
>indeks_sledeceg].rbr == 1 && duzina_puta == broj_fragmenata-1)))
            {
                kont->iskorisceni[grupe_grupa[trenutna_grupa].grupe->odgovarajuci_fragment] = 1;

                // Dodavanje grupe
                grupe_grupa[trenutna_grupa].odabrana_grupa = grupe_grupa[trenutna_grupa].grupe;
```

```
                grupe_grupa[trenutna_grupa].rbr = duzina_puta + 1;

                // Rekurzivni obilazak ostatka grupa
                obilazak(grupe_grupa, grupe_grupa[trenutna_grupa].grupe->indeks_sledeceg,
duzina_puta + 1, broj_fragmenata, kont);
                grupe_grupa[trenutna_grupa].grupe;

                grupe_grupa[trenutna_grupa].odabrana_grupa = NULL;
                grupe_grupa[trenutna_grupa].rbr = 0;

                kont->iskorisceni[grupe_grupa[trenutna_grupa].grupe->odgovarajuci_fragment] = 0;
            }
            grupe_grupa[trenutna_grupa].grupe = grupe_grupa[trenutna_grupa].grupe->sledeci;
        }

        grupe_grupa[trenutna_grupa].grupe = tmp;

    }
}

int kolizija(SKUP_GRUPA* grupe_grupa, GRUPA* grupa, int n)
{
    int i;
    for(i = 0; i < n; i++)
    {
        if(grupe_grupa[i].rbr != 0)
        {
            POLUFRAGMENT* tmp1 = grupe_grupa[i].odabrana_grupa->polufragmenti;
            while(grupe_grupa[i].odabrana_grupa->polufragmenti != NULL)
            {
                POLUFRAGMENT* tmp2 = grupa->polufragmenti;
                while(grupa->polufragmenti != NULL)
                {
                    if(!strcmp(grupa->polufragmenti->naziv, grupe_grupa[i].odabrana_grupa-
>polufragmenti->naziv))
                    {
                        grupa->polufragmenti = tmp2;
                        grupe_grupa[i].odabrana_grupa->polufragmenti = tmp1;
                        return 1; // Postoji kolizija
                    }
                    grupa->polufragmenti = grupa->polufragmenti->sledeci;
                }
                grupa->polufragmenti = tmp2;
                grupe_grupa[i].odabrana_grupa->polufragmenti = grupe_grupa[i].odabrana_grupa-
>polufragmenti->sledeci;
            }
            grupe_grupa[i].odabrana_grupa->polufragmenti = tmp1;
        }
    }
    return 0; // Nema kolizije
}

int poredi_otvarajuce(const void *a, const void *b)
{
    return strcmp((*(GRUPA**)a)->otvarajuci->naziv,(*(GRUPA**)b)->otvarajuci->naziv);
}

SKUP_GRUPA* napravi_skupove(GRUPA** niz_grupa, int n, int *m)
{
    SKUP_GRUPA* niz = NULL;
    int i;
    int j = 0;

    for(i = 0; i < n; i++)
    {
        if(i == 0)
        {
            niz = malloc(sizeof(SKUP_GRUPA));
            if(niz == NULL)
            {
                printf("Greska pri alokaciji!\n");
                exit(EXIT_FAILURE);
            }

            niz[0].rbr = 0;
            niz[0].grupe = niz_grupa[i];
            niz[0].grupe->sledeci = NULL;

        }
```

```
        else {

            if(!strcmp(niz_grupa[i-1]->otvarajuci->naziv,niz_grupa[i]->otvarajuci->naziv))
            {
                niz_grupa[i]->sledeci = niz[j].grupe;
                niz[j].grupe = niz_grupa[i];
            }
            else
            {
                j++;
                niz = realloc(niz,(j+1)*sizeof(SKUP_GRUPA));
                if(niz == NULL)
                {
                    printf("Greska pri alokaciji!\n");
                    exit(EXIT_FAILURE);
                }

                niz[j].rbr = 0;
                niz_grupa[i]->sledeci = niz[j].grupe;
                niz[j].grupe = niz_grupa[i];
                niz[j].grupe->sledeci = NULL;
            }
        }
    }

    *m = j+1;
    return niz;
}

void napravi_niz(GRUPA* stablo, int *i, GRUPA*** niz_grupa)
{
    if(stablo)
    {
        napravi_niz(stablo->l, i, niz_grupa);

        *niz_grupa = realloc(*niz_grupa,(*i + 1) * sizeof(GRUPA*));
        if(*niz_grupa == NULL)
        {
            printf("Greska pri alokaciji!\n");
            exit(EXIT_FAILURE);
        }

        (*niz_grupa)[*i] = stablo;

        *i = *i + 1;

        napravi_niz(stablo->d, i, niz_grupa);
    }
}

POLUFRAGMENT* ucitaj(FILE* ulaz, POLUFRAGMENT **otvarajuci, POLUFRAGMENT **neotvarajuci,
POLUFRAGMENT **zatvarajuci, POLUFRAGMENT **kompletni)
{
    POLUFRAGMENT *polufragmenti = NULL;

    char naziv[MAX_NAZIV];
    char oznaka_niska[MAX_TIP];
    int oznaka;
    int duzina;
    int blast_poz;

    int i = 0;

    while(!feof(ulaz))
    {
        fscanf(ulaz, "%s%s%d%d", naziv, oznaka_niska, &duzina, &blast_poz);
        if(feof(ulaz))
            break;

        if(!strcmp(oznaka_niska, "opening"))
            oznaka = OTV;
        else if(!strcmp(oznaka_niska, "middle"))
            oznaka = SRE;
        else if(!strcmp(oznaka_niska, "closing"))
            oznaka = ZAT;
        else if(!strcmp(oznaka_niska, "COMPLETE_FRAGMENT"))
            oznaka = KOM;
        else
        {
            printf("Neispravna oznaka polufragmenta!\n");
```

```
            exit(EXIT_FAILURE);
        }

        POLUFRAGMENT *novi = novi_polufragment(naziv, oznaka, duzina, blast_poz);

        novi->rbr = i++;

        if(oznaka == OTV)
            dodaj_polufragment(otvarajuci, novi);
        else if(oznaka == SRE)
            dodaj_polufragment(neotvarajuci, novi);
        else if(oznaka == ZAT)
            dodaj_polufragment(zatvarajuci, novi);
        else
            dodaj_polufragment(kompletni, novi);

        if(novi->blast_pocetna_pozicija < global_min)
        {
            global_min = novi->blast_pocetna_pozicija;
        }

        if(novi->blast_pocetna_pozicija + novi->duzina > global_max)
        {
            global_max = novi->blast_pocetna_pozicija + novi->duzina;
        }
    }
}

POLUFRAGMENT *novi_polufragment(char* naziv, int oznaka, int duzina, int blast_poz)
{
    POLUFRAGMENT *novi = (POLUFRAGMENT*)malloc(sizeof(POLUFRAGMENT));

    if(novi == NULL)
    {
        printf("Greska prilikom alokacije!\n");
        exit(EXIT_FAILURE);
    }

    strcpy(novi->naziv, naziv);
    novi->oznaka = oznaka;
    novi->duzina = duzina;
    novi->blast_pocetna_pozicija = blast_poz;
    novi->sledeci = NULL;

    return novi;
}

void dodaj_polufragment(POLUFRAGMENT **lista, POLUFRAGMENT *novi)
{
    if(*lista == NULL)
    {
        *lista = novi;
        return;
    }
    else
    {
        novi->sledeci = *lista;
        *lista = novi;
        return;
    }
}

int poredi_fragmente(const void *a, const void *b)
{
    return *(int*)a - *(int*)b;
}

GRUPA* nova_inicijalna_grupa(POLUFRAGMENT *pf)
{
    GRUPA *nova = (GRUPA*)malloc(sizeof(GRUPA));

    if(nova == NULL)
    {
        printf("Greska prilikom alokacije!\n");
        exit(EXIT_FAILURE);
    }

    nova->l = NULL;
    nova->d = NULL;
    nova->sledeci = NULL;
```

```
    nova->polufragmenti = NULL;

    nova->ukupna_duzina = 0;

    POLUFRAGMENT* novi = novi_polufragment(pf->naziv, pf->oznaka, pf->duzina, pf-
>blast_pocetna_pozicija);

    nova->polufragmenti = novi;

    nova->ukupna_duzina += pf->duzina;

    nova->otvarajuci = pf;
    nova->zatvarajuci = NULL;

    return nova;
}

GRUPA* nova_grupa(POLUFRAGMENT *pf, POLUFRAGMENT *otvarajuci, POLUFRAGMENT *zatvarajuci)
{
    GRUPA *nova = (GRUPA*)malloc(sizeof(GRUPA));

    if(nova == NULL)
    {
        printf("Greska prilikom alokacije!\n");
        exit(EXIT_FAILURE);
    }

    nova->sledeci = NULL;
    nova->polufragmenti = NULL;

    nova->ukupna_duzina = 0;
    nova->l = NULL;
    nova->d = NULL;

    nova->otvarajuci = otvarajuci;
    nova->zatvarajuci = zatvarajuci;

    POLUFRAGMENT *tmp = NULL;

    while(pf != NULL)
    {
        POLUFRAGMENT* novi = novi_polufragment(pf->naziv, pf->oznaka, pf->duzina, pf-
>blast_pocetna_pozicija);

        if(nova->polufragmenti == NULL)
        {
            nova->polufragmenti = novi;
            nova->ukupna_duzina += novi->duzina;
            tmp = nova->polufragmenti;
        }
        else
        {
            nova->polufragmenti->sledeci = novi;
            nova->polufragmenti = nova->polufragmenti->sledeci;
            nova->ukupna_duzina += novi->duzina;
        }

        pf = pf->sledeci;

    }

    nova->polufragmenti = tmp;

    return nova;
}

void dodaj_incijalnu_sumu(GRUPA** glava, POLUFRAGMENT *pf)
{

    GRUPA* nova = nova_inicijalna_grupa(pf);

    if(*glava == NULL)
    {
        *glava = nova;
        return;
    }
    else
    {
        nova->sledeci = *glava;
        *glava = nova;
```

```
        return;
    }
}

GRUPA* sastavljanje_kandidata(int *fragmenti, int broj_fragmenata, POLUFRAGMENT *otvarajuci,
POLUFRAGMENT *neotvarajuci, POLUFRAGMENT *zatvarajuci, POLUFRAGMENT* kompletni)
{

    GRUPA *glava = NULL;
    GRUPA *pom_glava = NULL;
    GRUPA *rep = NULL;

    GRUPA *prethodni;
    GRUPA *sledeci;

    int* rez = NULL;
    int trenutna;

    POLUFRAGMENT * novi;
    GRUPA* n;

    GRUPA* potvrdjene_grupe = NULL;

    POLUFRAGMENT *glava_zatvarajuci;


    // Kompletni polufragmenti su automatski kandidati sa BLAST greskom 0
    while(kompletni != NULL)
    {
        GRUPA* nova = nova_grupa(kompletni, kompletni, kompletni);
        nova->blast_razlika = 0;
        nova->sledeci = potvrdjene_grupe;
        potvrdjene_grupe = nova;
        kompletni = kompletni->sledeci;
    }

    // Formiranje inicijalnih grupa sa otvarajucim polufragmentima
    while(otvarajuci != NULL)
    {
        dodaj_incijalnu_sumu(&glava, otvarajuci);
        otvarajuci = otvarajuci->sledeci;
    }

    // Izdvajanje najveceg fragmenta
    int max_fragment = fragmenti[broj_fragmenata-1];

    while(neotvarajuci != NULL) // Prolazak kroz listu neotvarajucih polufragmenata
    {
        prethodni = NULL;
        pom_glava = glava;
        if(glava != NULL)
            sledeci = glava->sledeci;

        while(glava != NULL) // Dodavanje neotvarajucih polufragmenata u grupe
        {
            sledeci = glava->sledeci;

            trenutna = neotvarajuci->duzina + glava->ukupna_duzina;

            if(neotvarajuci->oznaka == SRE)
            {
                // Provera da li ukupna velicina grupe sa dodatim neotvarajucim polufragmentom ne
prelazi zadatu gresku i da li su ostala ogranicenja ispunjena
                if(trenutna <= max_fragment + max_fragment*GRESKA/100.0 && (glava->polufragmenti-
>oznaka == OTV || (glava->polufragmenti->rbr < neotvarajuci->rbr)))
                {
                    GRUPA* nova = nova_grupa(glava->polufragmenti, glava->otvarajuci, NULL);
                    dodaj_polufragment_u_sumu(&nova, neotvarajuci);
                    nova->sledeci = pom_glava;
                    pom_glava = nova;
                }
            }

            prethodni = glava;
            glava = glava->sledeci;
        }

        glava = pom_glava;

        neotvarajuci = neotvarajuci->sledeci;
```

```
        glava_zatvarajuci = zatvarajuci;

            while(zatvarajuci != NULL)
            {

                        prethodni = NULL;
                        glava = pom_glava;

                        if(glava != NULL)
                            sledeci = glava->sledeci;

                    while(glava != NULL)
                    {

                    sledeci = glava->sledeci;
                    trenutna = zatvarajuci->duzina + glava->ukupna_duzina;

                    if(trenutna <= max_fragment + max_fragment*GRESKA/100.0 && glava->polufragmenti-
>oznaka != ZAT && !samozatvarajuci(glava->otvarajuci, zatvarajuci))
                    {
                        // Ukoliko se grupa moze zatvoriti, proverava se da li postoji fragment cija
velicina odgovara velicini grupe
                       if((rez = bsearch(&trenutna,fragmenti, broj_fragmenata, sizeof(int),
poredi_fragmente_2)) != NULL)
                       {

                                novi = novi_polufragment(zatvarajuci->naziv, zatvarajuci->oznaka,
zatvarajuci->duzina, zatvarajuci->blast_pocetna_pozicija);
                                n = nova_grupa(glava->polufragmenti, glava->otvarajuci,
zatvarajuci);
                                novi->sledeci = n->polufragmenti;
                                n->polufragmenti = novi;
                                n->ukupna_duzina = glava->ukupna_duzina + zatvarajuci->duzina;
                                n->odgovarajuci_fragment = (int)(rez - fragmenti);

                               int min = n->polufragmenti->blast_pocetna_pozicija;
                               int max = 0;
                               int max_duzina = 0;

                               POLUFRAGMENT* sort = NULL;

                               POLUFRAGMENT* tmp = n->polufragmenti;
                               // Formiranje sortirane liste polufragmenata na osnovu pocetne
pozicije BLAST poravnanja
                               while(tmp != NULL)
                               {
                                   sort = dodaj_u_sortiranu_listu(sort, tmp);

                                   if(tmp->blast_pocetna_pozicija < min)
                                       min = tmp->blast_pocetna_pozicija;

                                   if(tmp->blast_pocetna_pozicija > max)
                                   {
                                       max = tmp->blast_pocetna_pozicija;
                                       max_duzina = tmp->duzina;
                                   }

                                   tmp = tmp->sledeci;
                               }

                               POLUFRAGMENT* pre = sort;
                               POLUFRAGMENT* sort_glava = sort;
                               sort = sort->sledeci;
                               POLUFRAGMENT* tek = sort;

                               if(sort->sledeci != NULL)
                                   tek = sort->sledeci;

                               POLUFRAGMENT* pocetak_praznine = pre;
                               POLUFRAGMENT* kraj_praznine = tek;

                               int velicina_praznine = tek->blast_pocetna_pozicija + pre->duzina -
pre->blast_pocetna_pozicija;

                               sort = sort_glava;
                               // Odredjivanje najvece praznine izmedju fragmenata
                               while(sort->sledeci != NULL)
                               {
                                   tek = sort->sledeci;
```

```
                                pre = sort;

                                if(tek->blast_pocetna_pozicija + pre->duzina - pre-
>blast_pocetna_pozicija > velicina_praznine)
                                {
                                    velicina_praznine = tek->blast_pocetna_pozicija + pre->duzina
- pre->blast_pocetna_pozicija;
                                    pocetak_praznine = pre;
                                    kraj_praznine = tek;
                                }

                                sort = tek;
                            }

                            int prvo_rastojanje = abs((max+max_duzina - min) - n->ukupna_duzina);
                            int drugo_rastojanje = abs((pocetak_praznine->blast_pocetna_pozicija
+ pocetak_praznine->duzina - global_min + (global_max - kraj_praznine->blast_pocetna_pozicija))-
n->ukupna_duzina);


                            oslobodi(sort_glava);

                            n->blast_razlika = prvo_rastojanje < drugo_rastojanje ?
prvo_rastojanje : drugo_rastojanje;

                            // Ukoliko je BLAST greska grupe manja od zadate, grupa postaje
kandidat
                            if(n->blast_razlika*1.0 / n->ukupna_duzina <= MAX_BLAST_GRESKA/100.0)
                            {
                             n->sledeci = potvrdjene_grupe;
                             potvrdjene_grupe = n;
                            }

                              glava = glava->sledeci;

                        }


                    else
                    {
                      prethodni = glava;
                      glava=glava->sledeci;
                    }
                }
                else
                {
                    prethodni = glava;
                    glava=glava->sledeci;
                }
            }
            zatvarajuci = zatvarajuci->sledeci;
            glava = pom_glava;

           }

         zatvarajuci = glava_zatvarajuci;

         glava = pom_glava;

    }

    return potvrdjene_grupe;
}

void dodaj_polufragment_u_sumu(GRUPA** glava, POLUFRAGMENT *pf)
{
    POLUFRAGMENT *novi = novi_polufragment(pf->naziv, pf->oznaka, pf->duzina, pf-
>blast_pocetna_pozicija);
    if((*glava)->polufragmenti == NULL)
    {
        (*glava)->polufragmenti = novi;
        (*glava)->ukupna_duzina = novi->duzina;
        return;
    }

    novi->sledeci = (*glava)->polufragmenti;
    (*glava)->ukupna_duzina += novi->duzina;
    (*glava)->polufragmenti = novi;
}
```

```
int poredi_fragmente_2(const void *a, const void *b)
{
    int prvi = *(int*)a;
    int drugi = *(int*)b;

    if(abs(prvi - drugi) <= drugi*GRESKA/100.0)
        return 0;

    return prvi - drugi;
}

int samozatvarajuci(POLUFRAGMENT* otvarajuci, POLUFRAGMENT* zatvarajuci)
{
    int scaf1, scaf2, cont1, cont2, frag1, frag2;

    sscanf(otvarajuci->naziv,"fragment_%d_%d_%d", &scaf1, &cont1, &frag1);
    sscanf(zatvarajuci->naziv,"fragment_%d_%d_%d", &scaf2, &cont2, &frag2);

    if(scaf1 == scaf2 && cont1 == cont2 && frag1 == frag2+1)
        return 1;
    return 0;
}

GRUPA* dodaj_u_stablo(GRUPA* grupa, GRUPA* s)
{
    if(grupa == NULL)
        return s;

    if(grupa->ukupna_duzina == s->ukupna_duzina && grupa->blast_razlika == s->blast_razlika)
        return grupa;

    if(s->ukupna_duzina < grupa->ukupna_duzina)
    {
        if(grupa->l == NULL)
            grupa->l = s;
        else
            grupa->l = dodaj_u_stablo(grupa->l, s);
        return grupa;
    }
    else
    {
        if(grupa->d == NULL)
            grupa->d = s;
        else
            grupa->d = dodaj_u_stablo(grupa->d, s);
        return grupa;
    }
}


void oslobodi(POLUFRAGMENT* sort)
{
    if(sort)
    {
        oslobodi(sort->sledeci);
        free(sort);
    }
}

POLUFRAGMENT* dodaj_u_sortiranu_listu(POLUFRAGMENT* lista, POLUFRAGMENT* novi)
{
    POLUFRAGMENT* tmp = novi_polufragment(novi->naziv, novi->oznaka, novi->duzina, novi-
>blast_pocetna_pozicija);
    POLUFRAGMENT* glava = lista;

    if(lista == NULL)
    {
        return tmp;
    }

    while(lista->sledeci != NULL && lista->sledeci->blast_pocetna_pozicija < novi-
>blast_pocetna_pozicija)
    {
        lista = lista->sledeci;
    }

    if(lista->sledeci == NULL)
        lista->sledeci = tmp;
    else
    {
```

```
            tmp->sledeci = lista->sledeci;
            lista->sledeci = tmp;
        }

        return glava;
    }
```

Додатак Б - Перл скрипта коришћена за сечење ДНК секвенце

```
#!/usr/bin/perl

open whole_seqence, "<", "il1403.fasta" or die $!;

$input_sequence = "";


# Ucitavanje svih scaffold-a
while(<whole_seqence>)
{
  $line = $_;
  $input_sequence = $input_sequence.$line;
}


# Izdvajanje pojedinacnih scaffold-a
my @scaffolds = split(/>scaffold\d+\s+length=\d+/,$input_sequence);

$num_scaffolds = scalar @scaffolds;
$non_empty_counter = 1;

# Ciscenje sekcence od zaglavlja scaffold-a i belina
#$input_sequence =~ s/>scaffold\d+\s+length=\d+\n//g;
$input_sequence =~ s/>scaffold\d+\s+length=\d+\n//g;
$input_sequence =~ s/\s//g;


# Izdvajanje praznina radi kasnijeg brojanja
my @blanks = split(/[^N]+/,$input_sequence);
$blank = 1;

open report, ">", "report_il1403_sfiI.txt" or die $!;

for(my $i = 0; $i < $num_scaffolds; $i++)
{
  if(length($scaffolds[$i]) != 0)
  {

  # Upis scaffold-a u pojedinacne datoteke
#  open scaffold, ">", "scaffolds/scaffold_".$non_empty_counter or die $!;
#  print scaffold $scaffolds[$i];

# Izdvajanje fragmenata iz scaffold-a
  my @contigs = split(/N+/, $scaffolds[$i]);
  $num_contigs = scalar @contigs;


  $non_empty_contig = 1;

  for(my $j = 0; $j < $num_contigs; $j++)
  {
    $contigs[$j] =~ s/\s//g;
    if(length($contigs[$j]) != 0)
    {

        $non_empty_fragment = 1;
        my @fragments = split(/GGCC\w\w\w\w\wGGCC/, $contigs[$j]); # Secenje na osnovu SfiI
enzima
        $num_fragments = scalar @fragments;

        for(my $k = 0; $k < $num_fragments; $k++)
        {
            if(length($fragments[$k]) != 0)
            {
                $output_contig =
"sfiI_il1403/fragment_".$non_empty_counter."_".$non_empty_contig."_".$non_empty_fragment;
                open contig, ">", $output_contig or die $!;

                print contig $fragments[$k];

                if($non_empty_fragment == 1 and $num_fragments > 1)
                {
                  $type = "closing";
                }
```

```
                else
                {
                  if($non_empty_fragment == $num_fragments and $num_fragments > 1)
                  {
                    $type = "opening";
                  }
                  else
                  {
                    if($non_empty_fragment == 1 and  $num_fragments == 1)
                    {
                      $type = "middle";
                    }

                    else
                    {
                      $type = "SINGLE_FRAGMENT";
                    }
                  }
                }
                    print report
"fragment_".$non_empty_counter."_".$non_empty_contig."_".$non_empty_fragment." ".$type."
".length($fragments[$k])." 0\n";
                    $blank++;

                $non_empty_fragment++;
            }
        }
        $non_empty_contig++;
    }
  }

   $non_empty_counter++;
  }
}
```

Додатак B - Садржај датотеке са подацима о полуфрагментима

```
fragment_1_1_1 middle 2730 1
fragment_1_2_1 closing 93912 2730
fragment_1_2_2 opening 295 96642
fragment_1_3_1 closing 33037 96937
fragment_1_3_2 opening 227910 129974
fragment_2_1_1 closing 58670 357884
fragment_2_1_2 opening 19507 416554
fragment_2_2_1 closing 137193 436061
fragment_2_2_2 opening 212724 573254
fragment_2_3_1 middle 96810 785978
fragment_2_4_1 closing 260392 882788
fragment_2_4_2 opening 103875 1143180
fragment_3_1_1 closing 64800 1247055
fragment_3_1_2 opening 623287 1311855
fragment_3_2_1 middle 11620 1935142
fragment_4_1_1 closing 258166 1946762
fragment_4_1_2 opening 156860 2204928
```