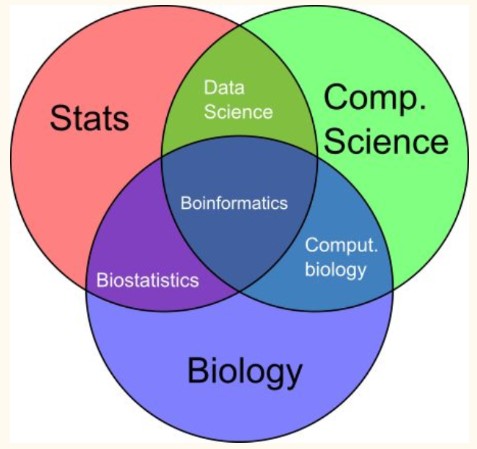
# Увод

Са открићем генома и развојем молекуларне биологије и генетике појавила се потреба за развојем техника које би могле да на ефикасан начин процесирају и анализирају те податке. Истовремено се развијало и рачунарство па су се рачунари наметнули као природан алат за процесирање великих количина података. Тако је дошло до развоја једне нове науке, биоинформатике.



Слика1. Биоинформатика

Биоинфроматика је наука о информацијама и протоку информација у биолошким системима, са посебним акцентом на употребу рачунарских метода у генетици и геномици. (oxford english dictionary). На слици изнад се могу видети везе биологије, статистике и рачунарства и то да се биоинформатика налази у самом пресеку ове три науке. Практично биоинформатика истражује и развија методе, софтверске алгоритме и софтверске алате за складиштење, организовање и анализирање биолошких података.

У раду ће прво бити објашњено шта су ДНА, ген и геном да би могли да разумемо њихов значај. Затим се бити представљен значај процесирања и анализе генома за разумевање његовог функционисања и откривања неправилности у његовој структури које могу бити узроци различитих болести. Да би се овакав тип анализе радио на рачунару, геном се прво мора представити у неком облику погодном за обраду на њему. Биће представњен једноставан начин репрезентовања геноме у рачунарству и примене софтверских алгоритама за његово процесирање. Наредни и највећи део рада биће посвећена примени различитих алгоритама претраживања у анализи генома.

# ДНК, ген и геном

ДНК је нуклеинска киселина која садржи упутство за развој и правилно функционисање свих живих организама. Он представља наследни материјал у свим живим ћелијама. Састоји се од пара молекула међусобно повезаних водоничним везама. Сваки ланац је изграђен од нуклеотида којих има четири врсте: аденин (А), цитозин (Ц), гуанин (Г) и тимин (Т).

Целовит део ДНК који преноси наследну информацију са генерације на генерацију назива се ген. Грађа гена је у ствари грађа ДНК и огледа се у тачно одређеном редоследу четири претходно поменута нуклеотида. Промена тог редоследа, мањак или висак нуклеотида резултира у промени функције гена и назива се генетска мутација.

Геноме је скуп свих гена неког организма и садржи све информације за правилан рад и развој организма. Научници децинијама истражују и анализирају геном у сврху налажења нових лекова, проналаска, разумевања и лечења генетских мутација, као и у сврху разумевања функционисања и развоја самих организама. Због овога је развој биоинформатика, односно развој алата који омогућавају ефикасно и брзо процесирање генома од великог значаја.

# Рачунарска анализа генома

Пошто се ДНК састоји од 4 никлеинске киселине као што је и објашњено у претходном поглављу сам геном се може представити као низ слова из скупа {А, Ц, Г, Т}. Овакав формат репрезентације ДНК је сада погодан за рачунарску обраду. Самим тим многи аспекти анализе генома се могу свести на претраживање и анализу знаковних низова.

Од посебног значаја је операција провере да ли се нека секвена никлеотида налази у геному и на којим местима се она налази. Овај проблем се може пресликати у проблем провере да ли је један знаковни низ подниз другог низа и ако јесте на којим местима се он појављује. Рачунарски алгоритами који се баве овим проблемом називају се *exact string matching* алгортими.

*Еxact string matching* алгортими се могу у основи поделити у две групе. То су алгоритми који спадају групу динамичких алгоритама и они не захтевају предзнање, односно неке додатне информације око свих могућих поднизова који се могу појавит у низу који се претражује. Евентуално се може урадити препроцесирање самог низа који се покушава наћи и који је по правилу много мањи од онога који се претражује.

Другу групу чине алгоритми који прво изврше претпроцесирање самог низа који се претражује и онда се на основу структура података генерисаних током фазе претпроцесирања врши даља претрага. Прва група алгоритама захтева мање хардверских ресурса за саму претрагу, али зато друга група алгоритама врши исте претраге за мање времена.

У наредном поглављу биће представљени алгоритми из прве групе и то прво *Naïve* алгоритам који представља најједноставнију имплементацију *еxact string matching* алгортима и који као такав није применљив на иоле већем сету података. Затим ће бити детаљно објашњена два алгоритма из ове групе који могу да раде на већим сетовима а то су *Boyer-Moore* алгоритам и *Knuth-Morris-Pratt* алгоритам.

У поглављу наког тог биће детаљније представљени неки од алгоритама који претпроцесирају податке пре саме претраге. Прво ће бити објашњена два најједноставнија алгоритма а то су алгоритми који користе сортирани индекс и хеш табелу као помоћне структуре података. Затим алоритми који користе нешто сложеније структуре података као што су сификс *trie* и суфикс стабла. На крају ће бити детаљно објаљен *Burrows-Wheeeler* алгоритам и како се он користу у комбинацији са ФМ индексом за брзо и ефикасно претраживање огромних знаковних низоба.

# Динамички алгоритми претраживања

За почетак треба увести одговарајућу терминологију која ће бити коришћена у остатку рада.

Т – низ који се претражује

Р – низ који се тражи

N = |T| - дужина низа Т

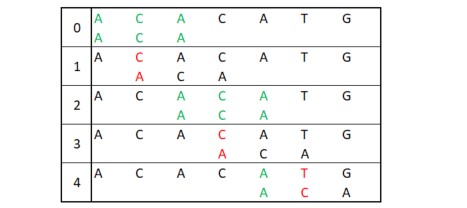
M = |P| - дужина низа Р

Оба низа се састоје из знакова из скупа {А, Ц, Г, Т} јер као што смо већ рекли геном се може репрезентовати низом састављеним од ова четири знака.

## *Naïve* алгоритам

Ово је најједноставнији алгоритам претраживања знаковних низова. Идеја овог алгоритма је да се покуша наћи Р у Т на сваком могућем померају. Тако се креће од почетног помераја у Т и проверава се да ли П постоји у Т закључно са померајем n – m. Претрага за било који померај се прекида уколико одговарајући знакови у оба низа нису иста.

Претпоставимо да имао текст Т =ACACATG и P = ACA. Алгоритам за дате низове ради на следећи начин. На слици 2 дат је приказ изврења овог алгоритма за све помераје. Свака колона представља претрагу за одговарајући померај. Слоба обојена зеленом бојом су ситуације када је досло до поклапања одговарајућих знакова из Т и Р, а црвена боја представља ситуацију када се одговарајући знакови не поклапају и када се претрага на том померају прекида и прелази се на наредни померај. Лако се може израчунати да је у овом конкреом примеру било 10 поређења укупно и да су нађена поклапања на померајима 0 и 2.



Слика2. Наиве алгоритам

Неефикасност је главни недостатак овог алгоритма и то се могло видети и у овом једноставном примеру је да постоји велики број поређења и за овако мале низове. Највећи број поређења ће бити потребан у случају да се Р садржи у Т на сваком померају и он је *(N – M + 1)xM*, a најмање у слућају да се Р[0] не налази у низу Т и у том случају је потребно *(N – M + 1)* поређења. Како је у случају генома *N* је јако велико и обично много веће од *M*,*(N – M + 1) ≈N* па је *(N – M + 1)xM≈NxM.* На основу овога можемо рећи да је сложеност овог алгоритма *О(NxM).*

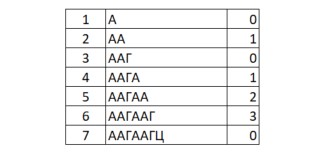
## *Knuth-Morris-Pratt* алгоритам

Највећа мана *Naïve* алгоритма је у томе што он врши доста непотребних поређења која могу бити избегнута. Основни мотив при развоју првих напреднијих алгоритама претразивања знаковних низова је био управо ово, да се прескоче сва непотребна поређења. Информацију када и која поређења треба прескочити може се добити из Р. Први алгоритам који је вршио претпроцесирање Р са циљем да креира неку малу структуру података из које лако може динамички да добија која поређења треба прескочити је *Knuth-Morris-Pratt* алгоритам.

Основна идеја овог алгоритма је да искористи то да су неки знакови већ познати у тренутку када се деси непоклапање одговарајућих знакова из Т и Р. Користећи ову чињеницу могуће је прескочити нека поређења за која сигурно знамо да ће бити успешна.

Стога се *Knuth-Morris-Pratt* алгоритам састоји из два дела. Први део је процес претпроцесирања низа Р како би се креирао један помоћни низ помоћу ког се налазе помераји које треба прескакати у току саме претраге, а други део алгоритма је сама претрага.

Циљ процеса претпроцесирања низа Р је да се креира помоћни низ који садржи позиције од којих треба наставити претраживање пошто се деси непоклапање одговарајућих знакова из Т и Р. Процес претпроцесирања се састоји из проласка кроз све поднизове низа Р који почињу од позиције 0 у Р. То су низови *Р[0:0], P[0:1], P[0:2],..., P[0:M-1]*.У сваког поднизу се тражи најдузи префикс које је уједно и суфикс у том поднизу стим да је цео подниз уједно и суфикс и префикс самог себе па се тај случај прескаче. Затим се дужина одговарајућег најдужег префикса које је уједно и суфикс уписује на позицију у резултујућем низу на ком се одговарајући подниз завршава у Р.



Слика 3. Процес претпроцесирања низа ААГААГЦ

На слици 3. је приказан процес претпроцесирања низа ААГААГЦ. Прва колона представља редни број итерације, наредна одговарајући подниз који се обрађује у тој итерацији а последља колона је дужина најдужег префикса који је уједно и суфикс водећи се правилом које је описано у претходном пасусу. Из овога се лако може добити одговарајући подниз који ће се користити у даљем процесу претраживања а то је *lps = [0, 1, 0, 1, 2, 3, 0].*

Нека су *i* и *j* текуће позиције из Т и Р. Алгоритам креће од помераја 0. Алгоритам се састоји из неколико корака:

1. Ако се знакови из оба низа на одговарајућој позицији подударају оба бројача се повећавају за 1.
2. У случају да се дошло до краја низа Р бројач *ј*се поставља на вредност *lps[j-1]*.
3. У супротном ако се није дошло до краја низа Р и нису се поклапили одговарајући знакови из Т и Р:

3а. Ако је *ј=0*,*i* се увећева за 1

3б. У супротном *ј=lps[j-1].*

У односу на *Naïve* алгоритам, *Knuth-Morris-Pratt* алгоритам постиже побољшање у случају најгорег сценарија а то је да се Р у Т садржи на сваком померају и сложеност у овом случају ће бити*О(N+M).*Што је у случају великог N али и великог M значајно побољшање у односу на *О(NxM)* код *Naïve* алгоритам.

## *Boyer-Moore* алгоритам

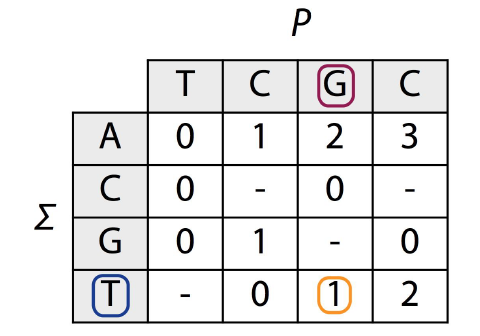
Као и *Knuth-Morris-Pratt* алгоритам и *Boyer-Moore* алгоритам врши претпроцесирање низа Р са циљем да у каснијем претраживању низа Т може да у одговарајућим тренуцима прескочи нека поређење. Међутим основна карактеристика овог алгоритма је да поређење врши у обрнутом поретку, односни пореди одговарајуће знакове почевши од последњем у низу Р, за разлику од *Knuth-Morris-Pratt* алгоритма који поређење врши почевши од првог знака. Управо ово својство у општем случају захтева далеко мањи број поређења него код претходна два алгоритма и омогућава прескакање неких непотребних поређења која би код *Knuth-Morris-Pratt* алгоритам била извршена.

Као и *Knuth-Morris-Pratt* алгоритам и овај алгоритам се састоји из два дела. Први део је процес претпроцесирања низа Р како би у овом случају креирала одговарајућа табла (која зависи од правила које се користи, а о којима ће ускоро бити речи) и на основу које се рачунају помераји које треба прескакати у току саме претраге, а други део алгоритма је сама претрага.

Алгоритам се води са два основна правила при претраживању. Прво правило се назиба правило лошег знака, а друго правило доброг суфикса. Друго правило се може појавити са једном додатном оптимизацијом о којој ће касније бити речи. Оба правила се могу примењивати независно за претраживање и прво ће тако бити и представљена. Међутим обра правила се могу и комбиновати како би се овај алгоритам учинио још ефикаснијим.

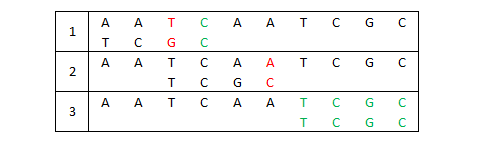
Правило лошег знака је јако једноставно и примењује се у ситуацији када се деси непоклапање знакова из Т и Р. Нека је t одговарајући знак из Т и нека је р одговарајући знак из Р, и нека је t различито од р тада треба Р треба померити у десно за онолико места колико има до најближег знака у Р који је један t ако такав постоји, а ако не онда Р померити до првог знака иза t.

Информација за колико се места треба померити се добија из табеле добијене у фази претпроцесирања. Пример табеле за текст ТЦГЦ дат је на слици 4.



Слика 4. Табела за Р = ТЦГЦ

Број редова одговара броју различитих знакова у Т, а број колона одговара дузини низа Р. Табела се претражује помоћу р и t, тако што се нађе пресек реда и колоне који одговарају тим знаковима. У конкретном примеру ако је p=Г, а t=Т, треба прескочити једано поређење, односно Р се треба померити у десно за 2 места уместо за једно, што би био случај код *Naïve* алгоритма.



Слика 5. Пример

На слици 5. демонстриран је *Boyer-Moore* алгоритам. У првој интерацији долази до непокапања знакова Т и Г, па се из табеле добија да Р треба померити за два места у десно. У другој итерацији долази до непоклапања знакова А и Ц, а како се Р не садржи знак А, Р се помера за целу своју дужину. Коначно у трећој итерацији долази до поклапања свих знакова.

Сложеност алгоритма у случају када је потребно извршити највише поређења је иста као и код *Naïve* алгоритма и износи *О(NxM)*. Међутим у најбољем случају, а то је ситуација када се последњи знак из Р не садржи у Т, сложеност је *О(N/M)*.