**המחלקה להנדסת תוכנה**

**שיפור ומקבול מציאת התאמה אופטימאלית ב-DNA**

**חיבור זה מהווה חלק מהדרישות לקבלת**

**תואר ראשון בהנדסה**

**מאת**

**הראל עוז ידגר**

**תמוז, תשע"ה יולי, 2015**

**המחלקה להנדסת תוכנה**

**שיפור ומקבול מציאת התאמה אופטימאלית ב-DNA**

**חיבור זה מהווה חלק מהדרישות לקבלת**

**תואר ראשון בהנדסה**

**מאת**

**הראל עוז ידגר**

**(חתימה)**

**מנחה אקדמי: דר' חסין יהודה אישור: תאריך:**

**רכז הפרויקטים: דר' יגל ראובן אישור: תאריך:**

**תקציר**

הפרויקט הינו פרויקט מחקרי, על מנת לאפשר הבנה של הרעיון, הבעיה והפתרון המוצגים בהמשך, יוצגו שלושתם בקצרה פה על מנת לספק ידע כללי של הפרויקט ללא כניסה לפרטיו הקטנים.

DNA

מולקולת הדנ"א מכילה את כל המידע התורשתי הדרוש לבניית החלבונים בתא אצל כל האורגניזמים הידועים. ה-דנ"א מושווה לעיתים למערכת תוכניות מכיוון שהוא מכיל את ההוראות הנחוצות לבניית רכיבי התא.

ה-דנ"א מורכב מארבע תתי יחידות של בסיסים חנקניים שהן אבני היסוד החוזרות על עצמן לאורכה של מולקולת ה-דנ"א בצירופים שונים. ארבעת הבסיסים הם אדנין (A), גואנין (G), תימין (T) וציטוזין (C), ולכן ניתן לתאר מולקולת דנ"א של אדם כמחרוזת של תווים בשפה {A, C, G, T}.

רצף הדנ"א הופך למידע מהותי בתהליך הטיפול הרפואי בשנים האחרונות שכן ניתן לאתר מראש מחלות מסוימות הקשורות לדנ"א (כגון סרטן הנרגם ממוטציות בדנ"א) וכן להתאים טיפולים בעלי יעילות מקסימאלית מכיוון שטיפולים מסוימים אינם יעילים כלל עבור חולים בעלי גנים מסויימים.

NGS

בשנים האחרונות פותחה שיטה יעילה וזולה לריצוף דנ"א של דגימה מסוימת. בשיטה זו לוקחים קריאות (חתיכות), בצורה רנדומאלית ובאורכים קצרים יחסית, מדנ"א הדגימה ומשתמשים בביולוגיה חישובית כדי לאתר את מיקום הקריאה ובסופו של התהליך מקבלים את מיקום כל הקריאות וכך מרצפים את הדנ"א של הדגימה.

כדי לאתר את מיקום כל קריאה יש לבצע חיפוש שלה בתוך רפרנס, דנ"א שמופה בעבר (שכן דנ"א של שני אנשים הוא זהה לדיוק של 99.9% ולכן ניתן לדעת בצורה מקורבת מהיכן הגיעה הקריאה שנלקחה), ולכן מבצעים חיפוש מסוג inexact matching שכן יש שוני מסוים בדנ"א וכן המכונה המבצעת את הקריאות טועה לעיתים בקריאת הגנים.

עבור כל קריאה שעברה את תהליך החיפוש התקבלו מספר התאמות (מיקומים בהן היא נמצאה) כעת יש לבצע החלטה מיהי ההתאמה האופטימאלית בניהן.

הבעיה

תהליך מציאת התאמה אופטימאלית דורש זמן רב שכן יש מיליוני קריאות ועבור כל אחת מהן יש לחשב מהי ההתאמה האופטימאלית מבין ההתאמות שנמצאו עבור הקריאה, כדי לבצע זאת יש לתת ציון לכל התאמה ע"י חישוב הזהות בין הקריאה והמקטע בו נמצאה התאמה. חישוב זה חוזר על עצמו מספר עצום של פעמים מכיוון שהוא מבוצע עבור כל התאמה שנמצאה לכל קריאה שנקלחה ובעצם נוצר צוואר בקבוק בתהליך שדורש משאבים רבים לביצוע העבודה.

הפתרון

הפתרון המוצע הוא מקבול ושיפור תהליך מציאת ההתאמה האופטימאלית עבור כל קריאה. בין היתר ביצענו שיפורים לזמן הריצה של התהליך וכן מקבלנו אותו לשם האצתו תוך ניצול משאבי המחשב בצורה היעילה ביותר.

את זמן הריצה של פונקצית חישוב הציון עבור התאמה שיפרנו מ- ל-וצריכת הזיכרון ירדה מ- ל-.

את זמן ריצת תהליך מציאת ההתאמה האופטימאלית שיפרנו ע"י שימוש בחוטים וקיבלנו שעבור שימוש ב-4 חוטים על מנת לבצע את עיבוד הקריאות במקביל זמן ריצת התהליך קטנה בכ-60%.

**הצהרה**

**תודות**

העבודה נעשתה בהנחיית ד"ר חסין יהודה

עזריאלי - מכללה אקדמית להנדסה ירושלים

המחלקה להנדסת תוכנה.

החיבור מציג את עבודתי האישית ומהווה חלק מהדרישות לקבלת תואר ראשון בהנדסה.

ברצוני להביע תודה לאנשים בהם נעזרתי במהלך הפרויקט.

תודה לסטודנט, אבי טרנר, המבצע פרויקט מקביל לשלי ואיתו התייעצתי במהלך העבודה על הפרויקט.

תודה למרצה, ד״ר שמרית צור דוד, על עזרתה המקצועית בנושא הרצה מקבילית וסנכרון חוטים.

ולבסוף תודה למנחה הפרויקט ד"ר יהודה חסין על העזרה המקצועית, ההכוונה אותה סיפק, הרעיונות השונים אותם העלה וכן הסבלנות הרבה שנתן לי במהלך ביצוע פרויקט הגמר.

**תוכן העניינים**

מסגרת הפרויקט.................................................................................9

תיאור הבעיה

הקדמה..............................................................................10,11

SW........................................................................................12

הגדרת הבעיה.........................................................................13

תיאור הפתרון

שיפור וייעול אלגוריתם SW....................................................14-17

מקבול תהליך מציאת ההתאמה האופטימאלית.........................18-20

תיאור המערכת שמומשה..............................................................21-23

בדיקות.......................................................................................24-28

השוואה לספרות...............................................................................29

מסקנות......................................................................................30-32

ספרות.............................................................................................33

קישורים...........................................................................................34

נספחים......................................................................................35-38

**מילון מונחים**

קריאה - מחרוזת תווים מעל השפה {A,C,G,T} המייצגת קטע קצר מתוך דנ"א אותו רוצים לרצף.

חוט - thread.

רפרנס - מחרוזת תווים מעל השפה {A,C,G,T} המייצגת דנ"א ידוע שרוצף בעבר.

BWA - Burrows-Wheeler Aligner

NGS - Next Generation Sequencing

SW - Smith Waterman

**מסגרת הפרויקט**

בתאריך 14 באפריל 2003 הסתיים "פרויקט גנום האדם" שמטרתו הייתה לקבוע את רצף הבסיסים המרכיבים את הקוד הגנטי של האדם ולזהות בהם אתרים פונקציונליים.

בעקבות פריצת הדרך שהפרויקט הביא לתחום הגנטיקה תהליכים רפואיים הקשורים לגנום פותחו וחשיבותו של רצף הדנ"א של החולה במהלך הטיפול בו עלתה מכיוון שניתן להיעזר בגנום שלו לזיהוי מוטציות שונות הגורמות למחלות (בכללן סרטן) ואף להתאים תרופות שיעבדו בצורה מיטבית עבור החולה.

פרויקט זה מבוצע תחת הנחייתו של דר' חסין יהודה והוא מהווה פרויקט גמר בתואר "הנדסת תוכנה" ב"עזריאלי מכללה אקדמאית להנדסה ירושלים".

סטודנט נוסף העובד על פרויקט הקשור לפרויקט זה הוא טרנר אבי.

**הבעיה**

**הקדמה**

עד ראשית שנות ה-2000, תהליך ריצוף דנ"א מורכב ויקר שעלותו עמדה על מאה מיליון דולר. בשנים האחרונות פותחה שיטה חדשה, שיטת ה-NGS, בה משתמשים בביולוגיה חישובית על מנת לבצע את התהליך במהירות ובעלות נמוכה בצורה משמעותית.

בשיטת ה-NGS לוקחים דגימת דנ"א מאדם אשר רוצים למפות את הדנ"א שלו, את הדנ"א חותכים בתהליך כימי במספר מקומות הנבחרים בצורה רנדומאלית באורכים קבועים מראש (35 עד 500 בסיסים) וקוראים את החתיכות שנחתכו בתור מחרוזות של תווים (התווים השונים המייצגים את הבסיסים הקיימים בדנ"א הם A,C,G,T) ולהן קוראים מחרוזות הקריאה. את התהליך מבצעים מספר פעמים בכדי שהסתברותית נקבל קריאות המכסות את כל הדנ"א, מספר זה נקרא הכיסוי (coverage) מכיוון שהוא קובע כמה פעמים כיסינו את הדנ"א, במקרה שלנו משתמשים בכיסוי 30 כלומר סכום אורכי מחרוזות הקריאה הוא פי 30 מאורך הדנ"א הממשי.

בשלב הבא משתמשים באלגוריתמים חישוביים כדי לאתר את מיקום כל אחת מהקריאות על פני הדנ"א האמיתי. בשלב זה באה לביטוי העובדה שדנ"א של בני אדם שונים זהה בדיוק של 99.9%, בכך שהאלגוריתמים יחפשו היכן כל קריאה שנלקחה מהדנ"א אותו רוצים למפות (לו קוראים הרפרנס), מופיעה בדנ"א שרוצף בעבר וכך יספקו את המיקום המשוער בדנ"א ממנו הקריאה נלקחה.

**התהליך**

שלב החיפוש

כפי שצוין לאחר שנלקחו הקריאות מהדנ"א יש לבצע חיפוש שלהן בתוך הרפרנס על מנת למצוא עבור כל קריאה את המיקום שממנו נלקחה. החיפוש יכול להיות משני סוגים:

1. חיפוש מסוג exact matching, כלומר חיפוש הופעה מדויקת בלבד של הקריאה בתוך הרפרנס.
2. חיפוש מסוג inexact matching, כלומר מציאת הופעות מדויקות וכן הופעות לא מדויקות עד שונות במספר תווים שניתן להגדיר מראש.

בהתחשב בעובדה שבדנ"א של שני בני אדם שונים קיים שוני לדרגה כלשהי וכן בעובדה שהמכונות המבצעות את התרגום מבסיסים למחרוזת תווים לא מבצעות את התרגום בצורה מושלמת ולעיתים נופלות טעויות בתרגום, עבור כל קריאה מבצעים חיפוש מסוג inexact matching ומקבלים את קבוצת מיקומי ההופעות של הקריאה בתוך הרפרנס.

קלט שלב החיפוש:

* הקריאות השונות שנלקחו .
* הרפרנס .

פלט שלב החיפוש:

* קבוצות המקיימות: קבוצה מכילה את כל ההתאמות שנמצאו עבור הקריאה ברפרנס .

אחד מהאלגוריתם המבצעים חיפוש של מחרוזות קריאה בתוך הרפרנס הוא אלגוריתם BWA (ראה נספח 1) המבצע חיפוש קריאה בתוך הרפרנס ביעילות התלויה באורך הקריאה וכן מספר הטעויות המאופשרות בלבד ואינו תלוי כלל באורך הרפרנס. האלגוריתם מתחשב במקרים של התאמת גנים, שונות בגנים, הוספה של גנים וכן מחיקה של גנים.

שלב מציאת התאמה אופטימאלית

מכיוון שכל קריאה הגיעה מהדנ"א ממיקום יחיד יש לבצע סינון של ההתאמות שקיבלנו עבור הקריאה ולהישאר עם התאמה אחת בלבד, ההתאמה האופטימאלית.

קלט השלב:

* הקריאות השונות שנלקחו .
* הרפרנס .
* קבוצות ההתאמות עבור כל קריאה בהתאמה .
* פונקצית הציון

פלט השלב:

* עבור כל קריאה מודפסת קואורדינאטת ההתאמה האופטימאלית וכן ציון ההתאמה שמקיימת: לכל מתקיים

בשלב זה משתמשים באלגוריתם SW על מנת לחשב את ציון כל אחת מההתאמות מכיוון שאלגוריתם זה מספק ציון לזהות בין שתי מחרוזות תוך מתן ניקוד למקרים של התאמה, שונות, הוספה וכן מחיקה של גנים.

**SW**

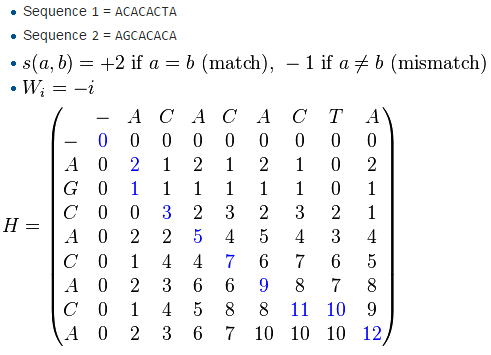
אלגוריתם Smith-Waterman, הינו אלגוריתם המשווה בין שתי מחרוזות ומחזיר ציון עבור הזהות בניהן, האלגוריתם בוחן את כלל אפשרויות ההתאמה בין שתי המחרוזות ומחזיר את הציון המקסימאלי מבין כל ההתאמות שנבדקו.

קלט האלגוריתם:

* שתי מחרוזות a ו-b בעלות אורכים m ו-n בהתאמה.
* פונקצית התאמת התווים, , כלומר הציון עבור זהות או שונות בין התו ה-i במחרוזת a והתו ה-j במחרוזת b.
* פונקצית עלות מחיקה, , כלומר הציון עבור השמטה של תו.
* פונקצית עלות הכנסה, , כלומר הציון עבור הוספה של תו.

האלגוריתם עובד בשיטה הרקורסיבית והוא בודק את כלל אפשרויות ההתאמה לפי הנוסחה:

מכיוון שהאלגוריתם עובד בשיטה הרקורסיבית הוא איננו יעיל כלל ולכן מימוש יעיל שלו ישתמש בעקרון התכנון הדינאמי כדי לשמור את ערכי הביניים, מימוש שכזה קיים בו זמן הריצה הוא וכן צריכת הזיכרון בו הינה .

דוגמה:

עבור המחרוזות

"ACACACTA" ו-"AGCACACA"

והפונקציות המוגדרות משמאל, ציון

ההתאמה בין המחרוזות הוא 12 וכן

ציוני כל ההתאמות שנבדקו מוצגים

בטבלת החישובים משמאל.

**הבעיה**

מכיוון בשלב מציאת ההתאמה האופטימאלית יש לעבור על כל הקריאות , ועבור כל קריאה יש לקחת את קבוצת כל ההתאמות שלה , ולכל התאמה בקבוצה יש להריץ SW על מנת לקבל ציון עבור כל התאמה לשם בחירת התאמה אופטימאלית.

נקבל שיש להריץ את אלגוריתם SW, פעמים, מספר זה של הרצות הינו עצום כאשר יש עשרות מיליוני קריאות וכן מספר התאמות עבור כל אחת מהן (לדוגמה עבור מיליון קריאות וכן חמש התאמות לכל קריאה, כמות הפעמים שיש להריץ SW היא: ).

מספר גבוהה זה של הרצות יוצר צוואר בקבוק ודורש זמן ריצה ארוך וגורם בזבוז רב של משאבים.

**הפתרון**

כדי לפתור את בעיית זמן הריצה וכן בזבוז המשאבים הרב של תהליך מציאת ההתאמה האופטימאלית עבור כל קריאה עבדנו על:

* שיפור אלגוריתם SW ע"י:
  1. שיפור זמן הריצה.
  2. שיפור צריכת הזיכרון.
* מקבול תהליך מציאת ההתאמה האופטימאלית.

**שיפור אלגוריתם SW**

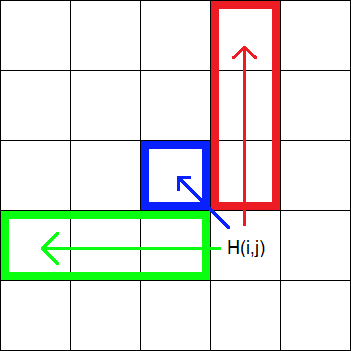
את אלגוריתם ה-SW מימשנו בעזרת תכנון דינאמי וכן הצענו ומימשנו ארבע גרסאות שכל אחת שיפרה זמן ריצה או צריכת זיכרון לעומת הגרסה הקודמת.

מימוש התכנון הדינאמי

מכיוון שחישוב רקורסיבי אינו יעיל ודורש את פתרון אותן תתי בעיות מספר רב של פעמים ניתן להשתמש בתכנון דינאמי על מנת לחשב פתרון עבור כל תת בעיה פעם אחת בלבד.

בתכנון דינאמי נשמור פתרונות לתתי בעיות קטנות ונפתור בעזרתן בעיות גדולות יותר, את הפתרונות לתתי בעיות נשמור במטריצה (מערך דו מימדי) בשם H ואת ערכם של התאים נקבע לפי נוסחת אלגוריתם SW.

ערכו של כל תא תקבע ע"י שימוש בתאים המסומנים:



התא שבאלכסון משמאל (כחול).

התאים שמעליו בעמודה (אדום).

התאים שמשמאלו בשורה (ירוק).

ניתוח זמן הריצה – עבור כל תא צריך לעבור על כל התאים שנמצאים מעליו וכן על התאים שנמצאים משמאלו, במקרה הכי גרוע יש תאים כאלו ומכיוון שיש לנו תאים שיש לבצע את החישוב עבורם, זמן הריצה הכולל הינו .

במימוש זה נקבל:

|  |  |
| --- | --- |
| יעילות זמן ריצה |  |
| יעילות זיכרון |  |

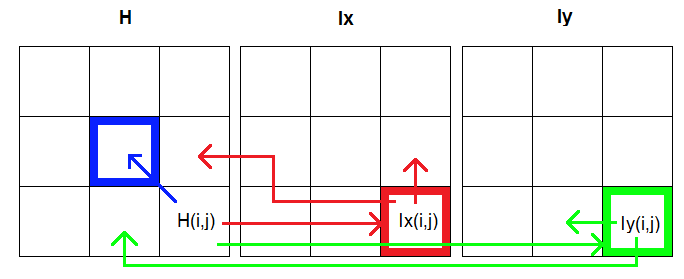
שיפור ראשון

מכיוון שבכל פעם מחושב המקסימום של שורה מסוימת עבור כל התאים בשורה (לכל תא בשורה סורקים את כל התאים שמשמאלו בשורה, ירוק בתמונה הקודמת) וכן בכל פעם מחושב המקסימום של עמודה מסוימת עבור כל התאים בעמודה (לכל תא בעמודה סורקים את כל התאים שמעליו בעמודה, כחול בתמונה הקודמת), במקום לבצע את חישוב המקסימום מחדש עבור כל תא נשתמש בשתי טבלאות נוספות כדי לשמור את ערכי המקסימום שחושבו עד כה.

בטבלה Ix (deletion) נשמור בכל תא את המקסימום מבין כל התאים שנמצאים מעליו בטבלה H (בהתחשב ב-W פונקצית המחיקה).

בטבלה Iy (insertion) נשמור בכל תא את המקסימום מבין כל התאים שנמצאים משמאלו בטבלה H (בהתחשב ב-W פונקצית ההוספה).

החישובים שנבצע:



ניתוח זמן הריצה – בכל פעם אנחנו מחשבים את ערך התא ה-(i,j) בטבלאות Ix ו-Iy על ידי חישוב מקסימום בין שני ערכים, לאחר מכן מחשבים את ערך התא ה-(i,j) בטבלה H על ידי חישוב מקסימום בין ארבע ערכים, חישוב זה נעשה בזמן קבוע. מכיוון שיש לנו תאים שכאלו זמן הריצה הכולל הינו .

לאחר שיפור זה נקבל:

|  |  |
| --- | --- |
| יעילות זמן ריצה |  |
| יעילות זיכרון |  |

שיפור שני

מכיוון שרק הציון הסופי עבור ההתאמה מעניין אותנו ולא הדרך בה הגענו לציון זה, ומכיוון שכעת כל תא בטבלה משתמש רק בתאים שישירות מעליו או משמאלו ניתן לוותר על השמירה של כל הטבלאות ולשמור את הנתונים הנחוצים למילוי שורה חדשה בלבד.

בשיפור זה נמצא את השורה החדשה, נעתיק אותה לישנה ונמלא את השורה הבאה מחדש (מכיוון שהעתקה של שורה שלמה תגדיל לנו את יעילות זמן הריצה חזרה ליעילות של הגרסה הראשונה לא נעתיק את השורה אלא נבצע את החלפת השורות ע"י שימוש במצביעים).

כעת נצטרך לכל אחת מהטבלאות H,Ix,Iy לשמור רק את השורה הקודמת שחושבה וכן זיכרון בו נחשב את ערכי השורה החדשה ולכן לכל טבלה נצטרך כעת רק שתי שורות.

לאחר שיפור זה נקבל:

|  |  |
| --- | --- |
| יעילות זמן ריצה |  |
| יעילות זיכרון |  |

שיפור שלישי

התנאים הבאים מתקיימים:

כדי לחשב את ערך התא ה-(i,j) בטבלה H אנו צריכים רק את התא ה-(i-1,j-1) מהטבלה H, ואת התאים (i,j) מטבלאות Ix ו-Iy.

כדי לחשב את ערך התא ה-(i,j) בטבלה Ix אנו צריכים רק את התא ה-(i-1,j) מהטבלאות Ix ו-H.

כדי לחשב את ערך התא ה-(i,j) בטבלה Iy אנו צריכים רק את התא ה-(i,j-1) מהטבלאות Iy ו-H.

מתקבל שכדי לחשב את ערכי התאים ה-(i,j) בטבלאות H,Ix,Iy צריכים רק את התאים שמקיפים אותם ולא את כל התאים שקדמו להם ולכן נשמור את השורה החדשה בלבד אך נוודה שכל פעם לפני דריסת הערכים עם מילוי הערכים החדשים שחושבו עבור התאים שבטבלאות, נשמור את הערכים שקדמו להם במשתנים נוספים כדי שנוכל לחשב את ערכו של התא הבא.

לאחר שיפור זה נקבל:

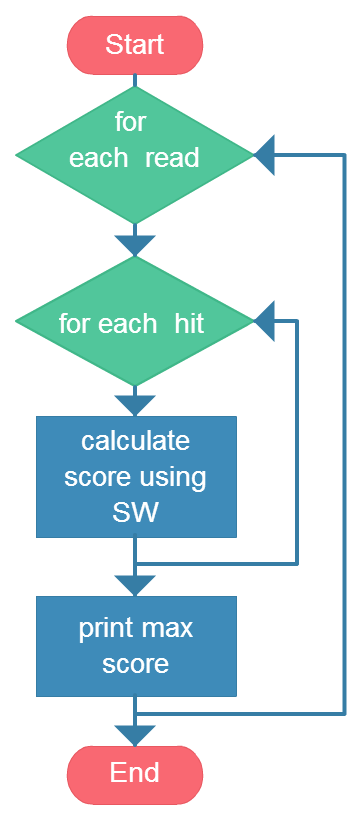
|  |  |
| --- | --- |
| יעילות זמן ריצה |  |
| יעילות זיכרון |  |

שיפור רביעי

מכיוון שערך התא ה-(i,j) בטבלה Iy תלוי בתא ה-(i,j-1) בהטבלאות Iy ו-H ואינו תלוי בכל שאר התאים בטבלה Iy, כלומר הוא תלוי רק בתא אחד שמאלה מבין התאים שבטבלה Iy, ניתן לשמור רק תא אחד מתוך הטבלה Iy ולשמור ישירות אליו את התא הבא שחושב.

לאחר שיפור זה נקבל:

|  |  |
| --- | --- |
| יעילות זמן ריצה |  |
| יעילות זיכרון |  |

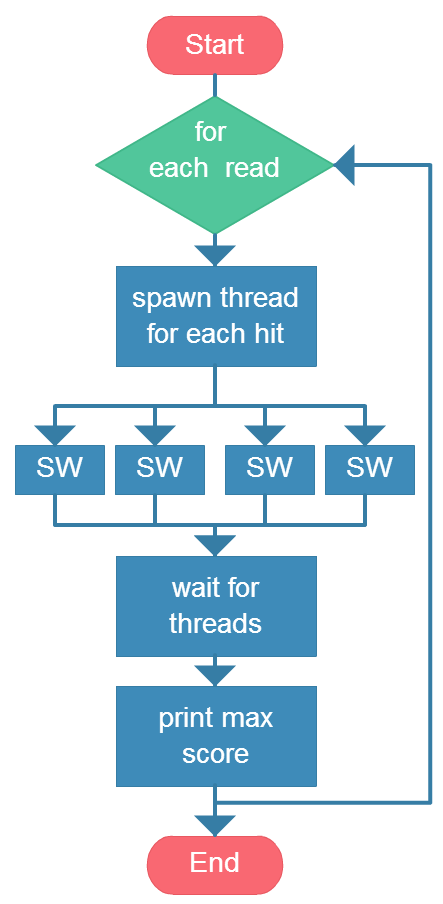
**מקבול תהליך מציאת ההתאמה האופטימאלית**

התהליך מומש בצורה סדרתית בכדי לנתח את אופן פעולתו וכן לזהות אופציות מקבול שונות.

במהלך התהליך עוברים על כל הקריאות בלולאה ועבור כל קריאה עוברים על כל ההתאמות שנמצאו עבורה ומחשבים ציון עבור כל התאמה, לאחר מכן ההתאמה עם הניקוד הגבוהה ביותר נבחרת כהתאמה אופטימאלית והיא מודפסת.

בעקבות ניתוח התהליך הסדרתי, נבחנו מספר אופציות מקבול של התהליך:

1. מקבול thread per hit.
2. מקבול thread per read.
3. מקבול thread per hit inside thread per hit.

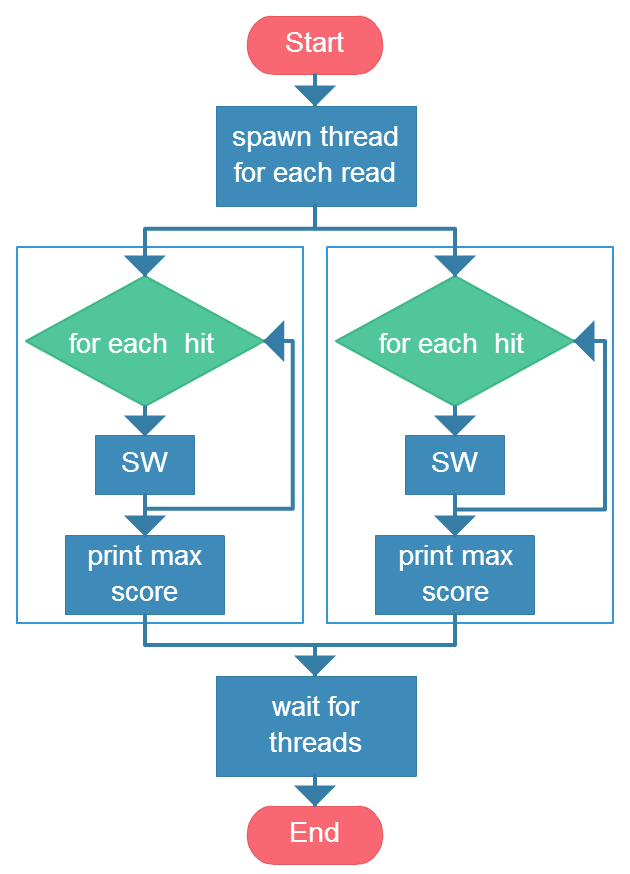
מקבול Thread per hit

במקום לחשב את ציון ההתאמה עבור כל התאמה ולאחר מכן לעבור לחישוב ציון ההתאמה הבאה וכן הלאה, נבצע את חישוב ציון כל ההתאמות עבור קריאה מסוימת במקביל.

באופציה זו לוקחים את מחרוזת הקריאה וכן את קבוצת ההתאמות שלה ומריצים את אלגוריתם SW עבור כל התאמה במקביל, לאחר סיום החישובים נערכת בדיקה מיהי ההתאמה שנתנה ציון מקסימאלי והיא נבחרת כהתאמה האופטימאלית.

מקבול זה אינו מוצלח מכיוון שתקורת התהליך גבוהה מאד, זמן פתיחת ה-threads, זמן ההמתנה שיסיימו את חישוב הציון וכן זמן ה-context switch, עולה על זמן החישוב עצמו ואינו זניח כלל.

כחלק מהמסקנות בעקבות מימוש הקוד הוחלט שאין טעם לממש את האופציה השלישית מאשר ששם יפתחו אף יותר threads מאשר באופציה זו.



מקבול Thread per read

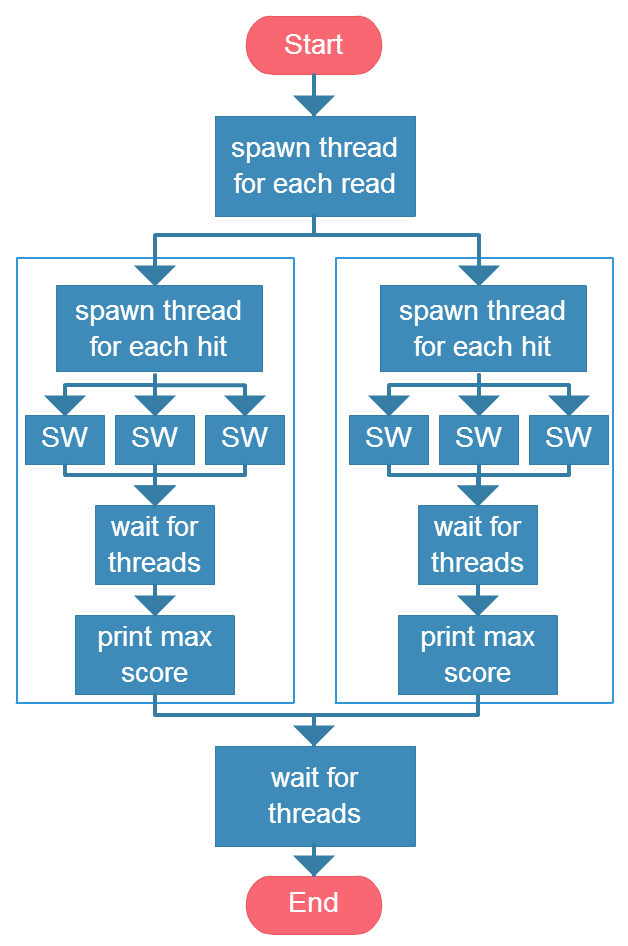
במקום לחשב מיהי ההתאמה האופטימאלי עבור כל קריאה בצורה סדרתית, נבצע את תהליך מציאת ההתאמה האופטימאלית עבור מספר קריאות במקביל.

אופציה זו מומשה ע"י threadpool, שזהו קוד הפותח מספר threads וכן תור עבודות, הקוד המשתמש ב-threadpool שולח עבודות לתור וברגע שאחד ה-threads מתפנה הוא לוקח את העבודה שבראש התור, ומבצע אותה.

העבודות הנשלחות לתור הן תהליך מציאת ההתאמה האופטימאלית מבין כלל ההתאמות שנמצאו עבור קריאה נתונה, התהליך יחשב עבור כל התאמה את ציונה וכן בודק מיהי ההתאמה האופטימאלית ומדפיס אותה.

הקוד שמומש נבדק והתקבל שהזמן העבודה שהתהליך דורש התקצרה בכ-40% עבור שני חוטים ועד כ-60% עבור ארבע חוטים (לא נבדקו יותר מכיוון שאלו הן מגבלות המערכת איתה עבדנו ויותר חוטים יגרמו ל-context switch שיעלה את זמן הריצה הכולל חזרה).

מקבול Thread per hit inside thread per read

הרצת התהליך המטפל בכל קריאה במקביל ובתוכו חישוב ציוני כל ההתאמות בצורה מקבילית, כלומר שילוב שני האופציות שהוצעו עד כה.

באופציה זו תהליך מציאת ההתאמה האופטימאלית עובד במקביל עבור מספר קריאות וכן עבור כל קריאה מחשבים את ציוני כלל ההתאמות שלה במקביל.

בגלל שזמן התקורה של האופציה הראשונה היה גבוה והדבר גרע מזמן הריצה של הקוד, הסקנו שגם באופציה זו זמן התקורה יהיה גבוה גם כן מכיוון שהאופציה זו מממשים את אותו תהליך יצירת threads כמו באופציה הראשונה ואף יוצרו יותר threads כאן כי הם יוצרו עבור כל ההתאמות של כמה קריאות בניגוד לצירת threads עבור כל ההתאמות של קריאה אחת בלבד ולכן הדבר יוסיף context switch רבים ויוריד אף יותר מיעילות התהליך.

**תיאור המערכת**

**מהי המערכת**

המערכת היא תוכנה הלוקחת קריאות, רפרנס וכן קבוצות של היסטים בהם מצאנו התאמה עבור הקריאות בתוך הרפרנס בהתאמה ומספקת עבור כל קריאה את ההתאמה האופטימאלית מתוך כלל ההתאמות שנמצאו עבורה וכן את ציון התאמה זו.

בכדי להפעיל את המערכת, תחילה מומשה מערכת נוספת למציאת ההתאמות עבור הקריאות בתוך הרפרנס ע"י שימוש בקוד BWA. בנוסף מומשו מספר גרסאות של אלגוריתם SW על מנת לשפר עוד יותר את זמן הריצה וכן צריכת המשאבים של המערכת.

**מערכת מציאת התאמות**

במערכת זו מומשו שני שלבים:

1. Index – השלב הראשוני שבו מבצעים עיבוד של הרפרנס בו נחפש בעתיד, מחושבים שתי פונקציות (C ו-O) וכן מערך מיקומים (suffix array).
2. Alignment – מציאת כלל ההתאמות של קריאה אותה מחפשים בתוך הרפרנס שעובד בשלב הקודם.

קלט המערכת:

* הרפרנס, דנ"א מרוצף מראש.
* הקריאות, קטעי דגימות של דנ"א אותו רוצים לרצף.

פלט המערכת:

* קבוצת היסטי ההתאמות עבור כל מחרוזת קריאה.

הכלים המשמשים לפתרון

לעבודה על שני שלביו הראשונים של אלגוריתם BWA, index ו-alignment השתמשנו בשפת JAVA ובסביבת העבודה ECLIPSE בגלל נוחות הכתיבה בשפה עילית, קלות הדיבוג וכן הכלים הקיימים בה ומפשטים את העבודה (כגון ה-garbage collector). מכיוון שמקבול שלבים אלו אינו חלק מפרויקט זה, לא עבדנו על מימושם שלהם בשפת C כי חשוב לנו פלט אותם שני שלבים.

תיאור המימוש

המימוש שכתבנו מכיל

* ביצוע טרנספורם BWT (חלק משלב ה-index) וכן חישוב ה-suffix array.
* חיפוש כלל ההתאמות של מחרוזת בתוך הרפרנס (שלב ה-alignment). תלוי באורך המחרוזת ומספר השגיאות המאופשר.

**אלגוריתם SW**

תחת חלק זה מומשו חמש גרסאות של אלגוריתם SW, כל גרסה שיפרה את זמן הריצה או את צריכת המשאבים של הגרסה הקודמת. נבדקה נכונותה של כל גרסה מספר פעמים וכן זמן הריצה שלה.

הגרסאות השונות:

|  |  |  |
| --- | --- | --- |
| גרסה | זמן ריצה | צריכת זיכרון |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

בנוסף מומשה גרסה נוספת המספקת גם את שחזור הפיתרון עבור ההתאמה האופטימאלית שנמצאה. גרסה זו מבוססת על גרסה מספר 2 שמומשה כבר מכיוון שצריך לשמור את כל טבלת הפיתרון והחל מגרסה מספר 3 ההנחה הינה שאין צורך בשמירת כלל טבלת הפתרון אלא רק חלק קטן ממנה הדרוש לחישובים שיבוצעו בעתיד.

תיאור המימוש

המימושים השונים נעשו בשפת C ונבדקו על מספר דוגמאות כדי לאשר את נכונותם.

במימוש שכתבנו ניתן לשנות את הפרמטרים לחישוב הציון ע"י שני משתנים:

* הפונקציה s הקובעת את עלות ההתאמה ועלות השונות בין שני תווים.
* הפונקציה W הקובעת את עלות המחיקה ועלות ההוספה של תו.

**מערכת מציאת התאמה אופטימאלית**

מערכת זו מומשה בצורה סדרתית וכן במספר צורות מקביליות.

מימוש סדרתי

המערכת לוקחת קריאה ואת קבוצת ההתאמות שנמצאו עבורה, עבור כל התאמה המערכת מחשב את ציונה ע"י שימוש באלגוריתם SW. לאחר שחושבו ציוני כל ההתאמות עבור הקריאה הנ"ל, נבחרת ההתאמה האופטימאלית והמערכת משחזרת את פתרון התאמה זו ומדפיסה אותה. לאחר מכן המערכת עוברת לקריאה הבאה וכן הלאה.

מימוש מקבילי ראשון

המערכת מומשה כל שלאחר שהיא לוקחת קריאה ואת קבוצת ההתאמות שנמצאו עבורה, עבור כל ההתאמות היא מחשבת את ציונן בצורה מקבילית תוך שימוש באלגוריתם SW. לאחר שחושבו ציוני כל ההתאמות עבור הקריאה הנ"ל, נבחרת ההתאמה האופטימאלית והמערכת משחזרת את פתרון התאמה זו ומדפיסה אותה. לאחר מכן המערכת עוברת לקריאה הבאה וכן הלאה.

מימוש מקבילי שני

המערכת מומשה כל שהיא מעבדת כל קריאה וקבוצת ההתאמות שנמצאו עבורה במקביל, כלומר מבצעים בצורה מקבילית עבור כל קריאה, חישוב ציון כל התאמה תוך שימוש באלגוריתם SW. לאחר שחושבו ציוני כל ההתאמות עבור הקריאה הנ"ל, נבחרת ההתאמה האופטימאלית והמערכת משחזרת את פתרון התאמה זו ומדפיסה אותה.

תיאור המימוש

המימושים השונים נעשו בשפת C ונבדקו על מספר דוגמאות כדי לאשר את נכונותם.

המימוש שכתבנו מכיל:

* מציאת ההתאמה האופטימאלית עבור כל קריאה בצורה סדרתית.
* מציאת ההתאמה האופטימאלית מבין כלל ההתאמות שנמצאו עבור כל קריאה תוך שימוש ב-threads על מנת לחשב את ציון ההתאמות שהתקבלו לקריאה, במקביל.
* מציאת ההתאמה האופטימאלית מבין כלל ההתאמות שנמצאו עבור כל הקריאות תוך שימוש ב-threads כדי למצוא במקביל את ההתאמה האופטימאלית עבור מספר קריאות, במקביל.
* Threadpool המתחזק מספר threads וכן תור עבודות אליו ניתן לשלוח עבודות (פונקציות) והן יבוצעו בצורה מקבילית ע"י ה-threads שנוצרו מראש.

**בדיקות**

מכיוון שהפרויקט הינו פרויקט מחקרי בו ייעלנו את אלגוריתם SW ומקבלנו את תהליך מציאת ההתאמה האופטימאלית הבדיקות בהן נתרכז הן בדיקות נכונות האלגוריתם וכן בדיקות זמני ריצה בכדי לראות שהשיפורים אותם עשינו אכן משתקפים בזמן הריצה.

**בדיקת נכונות**

**נכונות אלגוריתם SW**

מכיוון שאלגוריתם SW מומש במספר צורות שונות שלא עובדות בצורה straightforward ולא נשמרו כלל חישובי הביניים כפי בהם נשמרים בגרסה הבסיסית, נכונותן של הגרסאות השונות נבדקה ע"י:

1. בדיקת ערך ההחזרה.
2. בדיקת חישובי הביניים המתבצעים תוך כדי ריצת האלגוריתם.

עבור כל גרסה שמומשה נכתבה גרסה נוספת שבה מודפסים כלל החישובים המתבצעים במהלך האלגוריתם בצורת טבלה וכך ניתן לעקוב אחרי אופן הפעולה בצורה קלה ונוחה לקריאה.

לשם בדיקת הנכונות לוקטו מספר דוגמאות הרצה שכללו גם טבלאות המפרטות את כלל חישובי הביניים מספרות שונה באינטרנט וכן דוגמה חדשה הומצאה והורצה באופן ידני. כל גרסה שמומשה נבדקה עם כל דוגמה ופלט ההרצה נשמר ונבדק מול קובץ תוצאות מצופה ע"י תוכנת זהות בין קצבים.

תוצאות

הבדיקות השונות שבוצעו הניבו שלמרות שערכי ההחזרה בכל הגרסאות היו נכונות ותאמו את התוצאה המצופה, החל מגרסה 2 והלאה היו טעויות בחישובי ערכי הביניים בגלל טעות בחישובי הטבלאות Ix ו-Iy.

הטעויות תוקנו והבדיקה בוצעה מחדש למען ווידוי נכונות האלגוריתם פעם נוספת.

**נכונות תהליך מציאת התאמה אופטימאלית**

עבור תהליך מציאת התאמה אופטימאלית ביצענו שתי בדיקת:

1. בדיקת נכונות הקוד.
2. בדיקת נכונות המקבול.

תחילה בדקנו שהקוד העובד בצורה סדרתית הינו נכון ואכן מוצא את ההתאמה האופטימאלית ומספק את שחזור הפתרון עבורה. בכדי לבדוק זאת לקחנו מחרוזת שיצרנו רנדומאלית, ביצענו חיפוש שלה ע"י אלגוריתם BWA בכדי למצוא את כלל ההתאמות ולאחר מכן ביצענו עבור כל התאמה בדיקה ידנית של ציון ההתאמה ע"י שימוש באלגוריתם SW. את תוצאות ההרצה הידנית השוונו לתוצאות ההרצה הממוחשבת ובדקנו שאכן אותרה ההתאמה האופטימאלית ושחזור הפתרון שלה הודפס.

בכדי לבדוק את נכונות מקבול התהליך, הרצנו את תהליך מציאת התאמה אופטימאלית בצורה סדרתית על מספר דוגמאות שונות ושמרנו עבור כל דוגמה את פלטי ההתאמות האופטימאלית, לאחר מכן הורץ הקוד המקבילי על אותן דוגמאות ופלטי ההתאמות האופטימאליות הושווה לפלטי ההרצה הסדרתית.

תוצאות

עבור הבדיקה הראשונה התקבל שלמרות שההתאמה האופטימאלית אותרה, שחזור הפתרון הדפיס תמיד את ההתאמה האחרונה שנמצאה. הטעות בקוד תוקנה ונבדק מחדש עבור מספר קריאות שאכן ההתאמה האופטימאלית נמצאת ומודפסת.

עבור הבדיקה השנייה התקבל שהתהליך אכן עובד נכונה גם בצורתו הממוקבלת והפלטים השונים היו זהים לחלוטין לפלטים שהתקבלו מהרצה סדרתית.

**בדיקת זמני ריצה**

**ריצת אלגוריתם SW**

בדיקה ראשונה שנעשתה היא בדיקת זמן הריצה של האלגוריתם כאשר מספר הקריאות משתנה ואורך כל הקריאות זהה (נבחר 35 כמו הלקוח).

כפי שניתן לראות בגלל שזמן הריצה של הרצת SW הוא קבוע (הוא תלוי באורך הקריאות והינו קבוע) השוני הוא במספר הפעמים שחוזרים עליו, נתון זה השתנה בכפולות של 2 וכן גם זמני ההרצה בין הדגימות היו בקירובים של פי 2 אחד מהשני.

בדיקה שנייה שנעשתה היא בדיקת זמן הריצה של האלגוריתם כאשר מספר הקריאות הוא קבוע (נבחר שרירותית 100,000) ואורך הקריאות בכל ריצה ישתנה.

כפי שניתן לראות ככל שאורך הקריאה גדל כך גם זמן הריצה של האלגוריתם גדל שכן זמן ריצת האלגוריתם תלוי באורך הקריאה שמשתנה וניתן לראות שעבור גרסאות 2 עד 5 זמן הריצה גדל בחזקת 2 ואילו גרסה 1 גדלה בחזקה שלישית כפי שנותח תיאורטית עבור זמן הריצה.

**ריצת תהליך מציאת התאמה אופטימאלית**

בכדי להראות כיצד משפיע מקבול תהליך מציאת ההתאמה האופטימאלית על זמן הריצה של התהליך בדקנו עבור מספר משתנה של קריאות את זמן ריצת התהליך עבור:

1. התהליך כאשר הוא רץ בצורה סדרתית.
2. התהליך כאשר הוא רץ בצורה ממוקבלת, כאשר נבדוק עבור גודל threadpool שונה.

כפי שניתן לראות בגרף הרצת התהליך בצורה מקבילית חסכה עד כ-120% מזמן הריצה הכולל וזאת עבור חומרה המסוגלת להריץ 4 חוטים בבת אחת. מערכת המסוגלת להריץ מספר רב יותר של חוטים תחסוך במידה רבה יותר.

**ניתוח צריכת זיכרון**

בדיקות צריכת זיכרון נעשו ע"י ניתוח תיאורטי של הגרסאות השונות של אלגוריתם SW הן על ידי והן על ידי סטודנט נוסף כדי לוודא תוצאות.

|  |  |
| --- | --- |
| גרסה | צריכת זיכרון |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |

על חשיבות ההבדל בין גרסאות 3, 4 וכן 5 ראה מסקנות (מקבול לכרטיס מסך).

**בדיקות יחידה עבור ה-threadpool**

|  |  |  |  |
| --- | --- | --- | --- |
|  | שם הבדיקה | אופן הבדיקה | תוצאות מצופות |
| 1 | זיהוי קלט לא חוקי שלילי | נשלח למערכת מספר שלילי כמספר החוטים | הודעת שגיאה תודפס למשתמש |
| 2 | זיהוי קלט לא חוקי אפס | נשלח למערכת אפס כמספר החוטים | הודעת שגיאה תודפס למשתמש |
| 3 | זיהוי קלט לא חוקי גבוה | נשלח למערכת מספר גדול מידי של חוטים | הודעת שגיאה תודפס למשתמש |
| 4 | סגירת המערכת | נפתח threadpool עם 50 חוטים ונסגר מיד לאחר מכן | המערכת תיפתח ותיסגר מיד לאחר מכן |
| 5 | זיהוי עבודה לא חוקית | נשלחה עבודה שהיא NULL למערכת | העבודה לא תתבצע |
| 6 | ביצוע עבודה | שליחת עבודה יחידה למערכת | העבודה תתבצע |
| 7 | ביצוע מספר עבודות | שליחת מספר עבודות למערכת | העבודות יתבצעו לפי סדר הגעתן למערכת |
| 8 | ביצוע עבודות המגיעות בפרצים | שליחת עבודות למערכת בשני פרצים | העבודות מהפרץ הראשון יתבצעו לפי סדר הגעתן ולאחר מכן העבודות מהפרץ השני לפי סדר הגעתן |
| 9 | שליחת עבודות לאחר התרוקנות התור | שליחת עבודות למערכת, המתנה שיבוצעו ושליחת עבודות נוספות | ביצוע העבודות מהפרץ הראשון לפי סדר הגעתן, המתנה לעוד עבודות ולאחר קבלת הפרץ השני של העבודות, ביצוע שלהן לפי סדר הגעתן |

בבדיקת קוד ה-threadpool ניסיתי לבדוק את כלל המקרים העלולים לקרות במערכת ולוודא שהמימוש שכתבתי אכן פועל בצורה תקנית ונכונה לפי הגדרת ה-threadpool.

**השוואה לספרות**

# את אלגוריתם BWA מימשנו בעזרת מעקב אחרי מאמר המתאר את פעולתו, את נכונות הקוד בדקנו ע"י הרצתו עם דוגמאות שהבאנו מספרות שונה שמצאנו באינטרנט.

# את אלגוריתם SW מימשנו ע"י הנוסחה המתוארת בספרות וכן בדקנו את נכונותו ע"י הרצת דוגמאות שנמצאו במהלך חיפושים אחרי ספרות הנוגעת לאלגוריתם.

# קיים מימוש לאלגוריתם ה-SW בו הוא עובד עם דנ"א מכווץ. השימוש בדנ"א בצורתו המכווצת חוסכת זיכרון רב שכן אין צורך להשתמש ב-8 ביטים עבור קידוד כל אותו אלא רק ב-2 ביטים (שכן השפה היא {A,C,G,T} ולכן לקדד כל אות בשפה צריך ביטים) ולכן הזיכרון הדרוש לשמירת דנ"א מכווץ הוא מגודל הזיכרון הדרוש לשמירת דנ"א לא מכווץ. גם לאחר חיפושים רבים לא מצאנו את ספרות בנושא ואופן מימוש הקוד אינו קריא וניתן לניתוח בצורה פשוטה, ולכן בחרנו לעבוד עם SW וגרסה לא מכווצת של הדנ"א ובמקרה שהיינו מספיקים להקדים את לוח הזמנים תכננו להקדיש זמן עבור מימוש הקוד לגרסה מכווצת.

# בחבילת ה-BWA הקיימת כיום יש אפשרות למקבול תהליך מציאת ההתאמה האופטימאלית אך לאותו המחשב בלבד. מכיוון שאנו פעלנו מתוך רצון למקבל את המערכת בעצמנו ושאפנו להעביר את פעולת התהליך לחישוב בצורה מקבילית על גבי כרטיס מסך לא עבדנו עם הקוד הממוקבל הקיים ויצרנו גרסה משלנו.

# מאז אלגוריתם ה-BWA הומצא, יצאו שתי גרסאות נוספות לאלגוריתם BWA-SA ו- BWA-MEM שמייעלת את הזיכרון שבשימוש ואף ניתנת להרצה מקבילית, אך מכיוון ש- BWA-MEM משפר רק עבור מחרוזות קריאה באורך של 70 תווים ומעלה, והלקוח עובד עם מחרוזות קריאה של 35 תווים בלבד, לא נשתמש בקוד ה- BWA-MEM מאחר והקריאות שבשימוש אינן עונות על התנאי שהוזכר.

# פרויקט barraCUDA הינו פרויקט השואף למקבל את תהליך מציאת ההתאמות וכן את תהליך מציאת התאמה אופטימאלית עבור חישובים על גבי כרטיס מסך. מכיוון שהפרויקט מתרכז גם בתהליך מציאת ההתאמות הוא עדיין בשלבי מקבול תהליך מציאת ההתאמות ואינו הגיע אל שלב מציאת התאמה אופטימאלית עבור כל קריאה.

# מסקנות

# עבור הפרויקט שבוצע הגענו למסקנות עבור שיפורי זמן ריצה וכן שיפורי צריכת הזיכרון.

# שיפור זמן ריצה

# כפי שניתן לראות מהגרפים השונים הקשורים לזמני ריצה שהוצגו בשלב הבדיקות, ייעול זמן הריצה של אלגוריתם SW וכן מקבול תהליך מציאת התאמה טובה ביותר חסכו עיבוד רב והאיצו את ביצוע התהליך המדובר.

# התחלנו בעריכת סט בדיקות כדי להראות את ההבדל בין זמן הריצה בגרסה הסדרתית תוך שימוש במימוש אלגוריתם SW ללא ייעול כלל לעומת ריצה מקבילית עם הגרסה היעילה ביותר שמימשנו, אך מכיוון זמן הריצה הארוך של התהליך ללא שום שיפור הבדיקה הופסקה באמצע מכיוון ששלביה הבאים היו נמשכים מספר שעות.

# בתוצאות החלקיות המסופקות כאן ניתן לראות את ההבדל המהותי בין זמן ריצת התהליך לפני שיפור כלשהו (צורה סדרתית ומימוש לא יעיל של אלגוריתם SW) לעומת זמן ריצת התהליך לאחר השיורים שביצענו בו (מקבול התהליך ל-4 threads וכן שימוש במימוש היעיל ביותר ביצרנו עבור אלגוריתם SW), השיפור שביצענו ע"י המקבול וכן ע"י ייעול אלגוריתם SW הוריד את זמן הריצה בפקטור גבוה במיוחד.

# שיפור צריכת זיכרון

# בצורתו הנוכחית של הפרויקט לא ניתן לראות את ההבדל הניכר בין צריכות הזיכרון השונות של אלגוריתם SW מכיוון שהיחס בין הזיכרון הקיים במערכת בה התהליך עובד לעומת מספר הליבות (כל חוט ירוץ על ליבה) באותה מערכת הוא גבוה וכל מעבד יכול לעבוד עם זיכרון רב וכמעט ללא הגבלות.

# את חשיבות שיפור צריכת הזיכרון של אלגוריתם SW ניתן לראות במידה ונבחן מערכת מרובת ליבות וכן זיכרון מוגבל, מערכת שכזו היא מעבד גראפי.

# כיום, בכל כרטיס מסך קיים מעבד גראפי המורכב ממאות ואף מאלפי ליבות בניגוד למעבד הרגיל הקיים במחשב שמורכב מליבה אחת עד שמונה ליבות הקיים כיום בשוק, מה שהופך אותם ליותר יעילים יותר מאשר המעבד הרגיל עבור הרצה מקבילית של אלגוריתמים שונים. בנוסף הזיכרון הקיים על כרטיס המסך הוא מוגבל שכן מחובר לו זיכרון משלו ואין לו גישה ישירה אל זיכרון המחשב.

# בתיאוריה נוכל לנצל כל ליבה הקיימת במעבד הגראפי כדי להריץ thread של תהליך מציאת התאמה אופטימאלית. בכדי לנצל את המערכת בצורה מקסימאלית, נרצה להשתמש בכל ליבה לשם הרצת חישובים אך כל ליבה דורשת זיכרון מסוים עבור ביצוע החישובים ולכן נצטרך לבדוק האם קיים זיכרון פנוי שכזה עבור כל ליבה, במידה שכן המערכת תנוצל בצורה מקסימאלית אך במידה ולא מספר ליבות יתבזבזו ללא יכולת לפעול כלל.

# נשקול את המקרה הבא:

# אורך כל קריאה הוא 35 בסיסים ().

# מספר הליבות במעבד הגראפי הוא 2048 (עבור כרטיס מסך gtx980).

# בהנחה שנתעלם מגודל הזיכרון של הקלטים השונים ונתייחס רק לגודל הזיכרון הדרוש על מנת לבצע את חישובי הביניים של אלגוריתם SW נקבל:

# גודל הזיכרון הדרוש לחישוב אלגוריתם SW:

# בגרסתו הראשונה () הוא .

# בגרסתו השנייה () הוא .

# בגרסתו השלישית () הוא .

# בגרסתו הרביעית () הוא .

# בגרסתו החמישית () הוא .

# וכדי לחשב את גודל הזיכרון הכללי שצריך בהנחה שמשתמשים בכל 2048 הליבות נכפל כל תוצאה ב-2048 ונקבל:

# בגרסתו הראשונה () הוא .

# בגרסתו השנייה () הוא .

# בגרסתו השלישית () הוא .

# בגרסתו הרביעית () הוא .

# בגרסתו החמישית () הוא .

# הבדלים אלו משמעותיים ביותר שכן גודל הזיכרון הפנוי בכרטיס המסך הוא מוגבל מראש ואף יותר מכך לאחר שמתחשבים בכך שצריך לטעון אליו את הרפרנס (שבצורתו הלא מכווצת שוקל 3.2GByte) וכן את הקלטים השונים לאלגוריתם ה-SW.

# כפי שניתן להבחין ההבדל בין הגרסה השנייה והחמישית הוא הנרחב ביותר וכן ניתן להשתמש ב- יותר ליבות בגרסה החמישית לעומת הגרסה השנייה.

**רשימת ספרות**

1. מאמר הסבר על שיטת ה-NGS

<http://www.nature.com/jid/journal/v133/n8/full/jid2013248a.html>

1. הסבר על אלגוריתם SW:

[https://en.wikipedia.org/wiki/Smith-Waterman\_algorithm](https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm)

1. הצעות ייעול לאלגוריתם SW:

<http://www.public.asu.edu/~cbaral/cse591-s03/classnotes/seq-align.pdf>

1. דוגמאות עבור ריצת אלגוריתם SW:

<http://vlab.amrita.edu/?sub=3&brch=274&sim=1433&cnt=1>

<http://www.slideshare.net/avrilcoghlan/the-smith-waterman-algorithm>

[https://en.wikipedia.org/wiki/Smith-Waterman\_algorithm](https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm)

1. הסבר על תפקידו של ה-threadpool:
2. <https://en.wikipedia.org/wiki/Thread_pool_pattern>
3. מאמר על אלגוריתם BWA הכולל הסברים על שלבי ה-index, alignment ושחזור פתרון:

<http://bioinformatics.oxfordjournals.org/content/25/14/1754.long>

1. מאגר הקוד של אלגוריתם BWA:

<https://github.com/lh3/bwa>

1. מאגר הקוד של פרויקט barraCUDA:

<http://seqbarracuda.sourceforge.net/>

**קישורים**

1. מאגר הקוד:

<https://github.com/harel-oz-yadgar/BWA-Parallel>

1. יומן פרויקט:

<https://github.com/harel-oz-yadgar/BWA-Parallel/blob/master/final%20presentation/Project%20Diary.docx>

1. סרטון:

<https://www.youtube.com/watch?v=a2k_SZxH6vw>

**נספחים**

1. **BWA**

אלגוריתם BWA הוא אלגוריתם המוצא את קבוצת המיקומים בהם מופיעה מחרוזת קטנה (מחרוזת הקריאה) בתוך מחרוזת ארוכה (מחרוזת ההתייחסות). האלגוריתם מוצא את כלל מיקומי ההופעה של מחרוזת קצרה בתוך מחרוזת גדולה. האלגוריתם מוצא את כל ההופעות המדויקות (exact matching) וכן את כל ההופעות הלא מדויקות (inexact matching) מכיוון שנרצה להתחשב בעובדה שלמרות שההבדלים ברצפי הדנ"א של אנשים שונים הם זעירים, הם עדיין קיימים וכן בעובדה שלעיתים יש גם טעות קריאה של המכונה.

חדשנותו של אלגוריתם BWA היא שזמן החיפוש עבור מחרוזת קטנה בתוך מחרוזת גדולה תלוי אך ורק באורך המחרוזת הקצרה לאחר שבוצע פעם אחת בלבד שלב של חישובים מקדימים על המחרוזת הארוכה. בגלל עובדה זו, זמן החיפוש של מספר רב של מחרוזות קצרות תלוי באורך המחרוזות הקצרות בלבד והחיסכון בעבודה עולה בהרבה על הבזבוז שהיה בשלב החישוב המקדים.

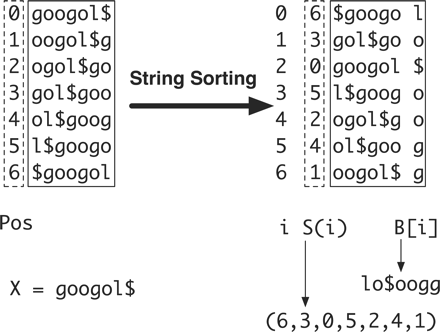
באלגוריתם BWA יש שלושה שלבים:

1. שלב ה-Index

בשלב הראשון נטען רצף דנ"א שלם לזיכרון כמחרוזת הרפרנס ומבוצעים החישובים המקדימים. שלב זה הוא קבוע שמתבצע פעם אחת בלבד ולכן אין טעם להתמקד בשלב זה של האלגוריתם.

תחילה, לוקחים את מחרוזת הרפרנס ומייצרים את כל המחרוזות שניתן להגיע אליהן ממנה ע"י סיבוב של התווים בה, לכל מחרוזת שנוצרת נשמור את מספר התווים שעברו מההתחלה לסוף בתוך מערך שייקרא suffix array. לאחר מכן ממיינים את המחרוזות שנוצרו לפי סדר לקסיקוגרפי תוך עדכון הסדר ב-suffix array בהתאמה. לבסוף יוצרים מחרוזת חדשה (שתקבל את השם circulation string ומסומנת ע"י B) ע"י לקיחת האותיות מסוף כל המחרוזות שמוינו וכן שומרים את ה-suffix array. הפעולה האחרונה בשלב היא חישוב הפונקציה C(a) (מספר האותיות שקטנות לקסיקוגרפית מ-a במחרוזת ההתייחסות) וכן חישוב הפונקציה O(i,a) (מספר ההופעות של האות a בתוך B מתחילתה עד התו ה-i). שתי הפונקציות קשורות לחישוב של שלב ה-alignment.

דוגמה:



מחרוזת ההתייחסות היא

"googol". נבצע סיבוב של

המחרוזת, נקבל את כל

המחרוזות האפשריות

ונמיין לפי סדר לקסיקוגרפי.

פונקצית O שהתקבלה פונקצית C שהתקבלה

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 | a\i |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | g |
| 3 | 3 | 3 | 2 | 1 | 1 | 0 | l |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | o |

|  |  |
| --- | --- |
| C(a) | a |
| 0 | g |
| 2 | l |
| 4 | o |

1. שלב ה-Alignment

בשלב זה מחפשים מחרוזות קטנות רבות (מחרוזות הקריאה) בתוך מחרוזת ההתייחסות, החיפוש שמתבצע הוא חיפוש מסוג inexact matching, כלומר חיפוש שמתחשב גם במרווח טעות של עד z תווים בכל מחרוזת.

האלגוריתם מבצע את החיפוש תוך התחשבות במקרים הבאים:

* הכנסה (insertion) של גן, כלומר התו הנוכחי במחרוזת הקריאה הוא תו שאינו מופיעה במחרוזת ההתייחסות.
* מחיקה (deletion) של גן, כלומר יש תו אחד במחרוזת ההתייחסות שאינו מופיע במחרוזת הקריאה.
* התאמה בתווים בין מחרוזת ההתייחסות ומחרוזת הקריאה.
* חוסר התאמה בתווים בין מחרוזת ההתייחסות והמחרוזת אותה מחפשים ולכן נקטין את מרווח הטעות ונמשיך את הבדיקה מהתו הבא.

תוך התחשבות בכל המקרים שלהלן, בשלב זה נמצאים כלל המיקומים של מחרוזת הקריאה בתוך מחרוזת ההתייחסות. המיקומים בהם מחרוזת הקריאה מופיעה בתוך מחרוזת ההתייחסות מיוצגים ע"י קבוצה של מקטעים כאשר כל מקטע (SA interval) מייצג את נקודת ההתחלה ואת נקודת הסיום של המופעים של המחרוזת שמחפשים בתוך ה-burrows wheeler transform. בכדי לשחזר את הקואורדינאטות של ההתאמות במחרוזת ההתייחסות נעבור על ה-suffix array וניקח את הערך שנמצא בכל אחד מהמיקומים שהופיעו כחלק מאחד המקטעים בקבוצה שהתקבלה בסיום שלב זה של ההרצה.

דוגמה:

עבור מחרוזת ההתייחסות "googol" קיבלנו את ה-suffix array (6,3,0,5,2,4,1). במידה ונחפש את המחרוזת "go" ללא מרווח טעויות נקבל את המקטע [1,2] שזה היכן המחרוזת "go" נמצאת כתחילית ב-suffix array. כדי לשחזר את הפתרון ניקח את הערכים מה-suffix array במיקומים 1 ו-2 (כי אלא המיקומים שהמקטע מכיל, אם היה מכיל עוד בניהם היינו לוקחים גם אותם) ולכן עבור suffix array במקום 1 נקבל 3 ועבור suffix array במקום 2 נקבל 0, כלומר המחרוזת "go" מופיעה (ללא מרווח טעות כמו שצוין) מהתו הרביעי והלאה וכן מהתו הראשון והלאה במחרוזת ההתייחסות.

בשלב זה של האלגוריתם מתבצע חישוב עבור כל מחרוזת קריאה, למרות חשיבותו של שלב זה לא נתמקד בו מכיוון שפרויקט אחר מתמקד בו וממקבל אותו.

1. שלב ה-Pair end mapping

עבור כל מחרוזת קריאה, נמצאו עבורה במהלך שלב ה-alignment קבוצה של התאמות בתוך מחרוזת ההתייחסות אך מקודד לפי burrows wheeler transform.

כזכור עבור כל מחרוזת קריאה נמצאו מספר התאמות, אך ידוע שהקריאה הגיעה ממיקום אחד בלבד בדנ"א (הרי היא נחתכה מרצף הדנ"א) ולכן שלב זה של האלגוריתם ינתח מה היא ההתאמה הנכונה מבין כלל ההתאמות שנמצאו ויספק בסופו של התהליך התאמה אחת בלבד לכל מחרוזת קריאה. ולכן בשלב זה יש להמיר כל התאמה לקואורדינאטה המתאימה לה במחרוזת ההתייחסות ולאחר מכן לבצע השוואה בין כל ההתאמות שהתקבלו.

בעיתיות BWA

בעקבות המספר הרב של מחרוזות קריאה שמגיע לעשרות מיליונים שלבי ה-alignment וכן ה-pair end mapping לוקחים זמן רב ומשאבים, ולכן כל ארגון או חברה הרוצים למפות דנ"א של אדם מסוים צריכים לתחזק/לשכור חוות מחשבים למען ביצוע הפעולה. במקרה שלנו, הלקוח שהוא בית החולים הדסה, שוכר חוות מחשבים לשם ביצוע התהליך בעלות של עשרות אלפי דולרים ליום.

נוסף על הצוואר הבקבוק בשני השלבים שצוינו למעלה, שלב ה-pair end mapping לוקח פי שלושים יותר זמן מאשר שלב ה-alignment וזאת מכיוון שבעוד שבשלב ה-alignment הריצה תלויה באורך מחרוזת הקריאה, בשלב ה-pair end mapping יש לבצע הערכה ולתת ציון לכל התאמה, כלומר עבור כל התאמה שנמצאה יש לרוץ על אורך המחרוזת ולכן עבור מספר התאמות גבוהה זמן ההערכה בשלב ה-pair end mapping עולה בהרבה על זמן החיפוש בשלב ה-alignment.

בשלב ה-pair end mapping יש שני תתי שלבים חשובים:

* המרת ההתאמות למיקומים ממשיים, כלומר שחזור מיקומי הופעת מחרוזת הקריאה בתוך מחרוזת ההתייחסות מהתצוגה אותה שלב ה-alignment מספק (התאמות בתוך ה-suffix array) לתצוגת היסט ממשי במחרוזת ההתייחסות.
* מציאת ההתאמה הטובה ביותר מבין כלל ההתאמות שנמצאו עבור מחרוזת הקריאה בתוך מחרוזת ההתייחסות וזאת ע"י שימוש באלגוריתם SW.

**Abstract**

This project is a research project. To allow a better understanding of the concept, the problem and the solution detailed below, they will be briefly explained here to provide general knowledge of the project without getting into the little details.

DNA

The DNA molecule contains all of the genetic information necessary for building the proteins in the cell in all of the known organisms. The DNA is often compared to a plan scheme because it contains the instructions needed for building the cell components.

The DNA consists of four sub-units of the nitrogenous bases that are function as the cornerstones and are repeated along the DNA molecule in variant combinations. The four bases are adenine (A), guanine (G), thymine (T) and cytosine (C), therefore the DNA molecule of a person can be described as a string of the characters {A, C, G, T}.

In the recent years, knowing the DNA sequence becomes essential in the process of health because it helps detecting beforehand certain diseases associated with the DNA (such as cancer stoned mutations in DNA) and adjusts some of the treatments for maximum efficacy because some treatments are not effective for all patients with certain genes.

NGS

In recent years a new cheap and effective method for DNA sequencing been developed. This method takes readings (pieces) that are randomly selected and relatively short lengths, from the DNA and uses Computational Biology to locate the position of each of the reads and ultimately get the location of all of the readings and sequences the DNA of the specimen.

To find the location of each of the reads, the system has to search them within the reference, a previously sequenced DNA (the DNA sequences of two people is the same to a 99.9% and therefore it can be used to find the location from which a read was taken). The search is inexact matching because although it's small, there is a difference between the DNA sequences and the machine that cuts the reads and interpret them has an error margin.

For each of the readings who passed the search process there is several match's (locations in which it was founded) and now the system have to determine which match is the optimal between them.

The problem

The process of finding the optimal match requires a long run-time since there are millions of reads and for each one of them the system has to calculate which match is the optimal among the matches that was found for that read. In order to complete this task each match found for a specific read gets a score based on the identity between the read and the segment in which the match was found. The score calculation is repeated a huge number of times because it is performed for each match found per read and this creates a bottleneck in the system that requires many resources to complete the job.

The solution

The proposed solution is to parallelize and improve the process of determine the optimal match for each read. Among other things, we improved the run- time of the process and change it so it will be able to run in parallel, utilizing the computer resources most efficiently.

We improved the run-time of the scoring function from to as well as lowered the memory consumption from to .

We managed to improve to overall run-time of the system by using 4 threads in order to process the different reads in parallel by a factor of 60% less run-time.

**Software Engineering Department**

**Improving and parallelizing finding optimal matching in the DNA**

**by**

**Harel Oz Yadgar**

**Academic Supervisor: Dr. Hassin Yehuda**

**Software Engineering Department**

**Improving and parallelizing finding optimal matching in the DNA**

**by**

**Harel Oz Yadgar**

**July 2015 Tamuz 5775**