

המחלקה להנדסת תוכנה

### פרוייקט גמר בהנדסת תוכנה – תשע"ה

**דוח אב הטיפוס**

**מקבול שלב ה-pairing באלגוריתם BWA בעזרת כרטיס גראפי**

**Parallelize pair end mapping in BWA algorithm using GPU**

|  |
| --- |
| **נ ת ו נ י ם א י ש י י ם** |
| **הראל עוז ידגר**  **ת.ז: 204174643**  **דוא"ל:** [**ozhry9@gmail.com**](mailto:ozhry9@gmail.com) |
| **פרויקט מחקרי** |
| **מנחה אקדמי**  **ד"ר יהודה חסין** |

# תיאור מסגרת הפרויקט

הפרויקט אותו אנו מבצעים הוא פרויקט מחקרי הכולל אלגוריתמיקה וכן מספר מונחים הקשורים לתחום הביולוגיה אשר את חלקם נסביר פה בפשטות לשם הבנת תהליכי הפרויקט.

**DNA**

ה-דנ"א היא מולקולת ענק של חומצות גרעין המורכבת ממספר רב של נוקליאודיטים המאורגנים במבנה של סליל כפול. מולקולת ה-דנ"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. כל מקטע משמעותי ב-דנ"א המכיל מידע לבניית חלבון או הוראות בקרה נקרה גן. רצף הבסיסים המרכיבים את מולקולת ה-דנ"א מקודד גנים שונים.

ה-דנ"א מורכב מארבע תתי יחידות של בסיסים חנקניים (נוקליאודיטים) שהן בעצם ארבע אבני יסוד החוזרות על עצמן לאורכה של מולקולת ה-דנ"א בצירופים שונים. ארבעת הבסיסים הם אדנין (A), גואנין (G), תימין (T) וציטוזין (C).

רצפי ה-דנ"א של שני בני אדם שונים זהה עד לדיוק של כ-99.9%.

לסיכום ניתן לתאר מולקולת דנ"א של אדם כמחרוזת של תווים בשפה {A, C, G, T} המייצגת גדיל אחד מהסליל הכפול (הגדיל השני הוא מעין ראי של הגדיל הראשון ולכן לא צריך לשמור גם אותו).

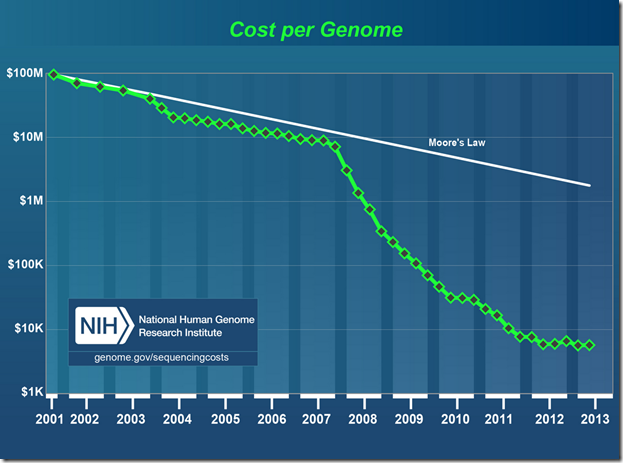
**חשיבותו של ה-DNA**

רצף דנ"א של חולה חשוב מכיוון שרופא המקבל את רצף ה-דנ"א של חולה יכול לאתר מוטציות ברצף ה-דנ"א שמתבטאות בהחלפה של אותיות או מקטעים (החלפות מסוימות גורמות למחלות שונות כגון סרטן) וכן הוא יכול להתאים תרופות אשר יאפשרו הצלחה מרבית של הטיפול.

**Next Generation Sequencing**

עד כה ריצוף דנ"א היה תהליך מורכב ויקר מאד שעלה מאה מליון דולר עבור ריצוף דנ"א בודד ולכן לא ניתן היה לבצע את התהליך בקנה מידה גדול עבור חולים רבים.

להלן גרף של עלות ריצוף דנ"א בדולרים לפי שנים החלק משנת 2001:



בשנים האחרונות פותחה שיטה חדשה, שיטת ה-next generation sequencing או בקצרה NGS, המשתמשת בעובדה שה-דנ"א של בני אדם הוא זהה לרמה של 99.9% וכן באלגוריתמים שונים בתחום מדעי המחשב. בשיטה זו במקום לבצע קריאה של רצף ה-דנ"א כמו בעבר אנו משתמשים ב-דנ"א שכבר רוצף כבסיס לריצוף ה-דנ"א החדש.

בתהליך החדש במקום לבצע קריאות דנ"א ארוכות וחיבור בינהן אנחנו מבצעים קריאות קטנות רבות, בין 30bp לבין 200bp (bp-bases per read כלומר מספר התווים בקריאה), ועל ידי שימוש באלגוריתמים מתקדמים אנו מוצאים את המיקום היחסי שלהם ביחס לרצף ה-דנ"א של אדם בריא (רצף ידוע מראש). בגלל שרצף ה-דנ"א של שני בני אדם זהה ב-99.9% אזי נוכל למצוא את המיקום של כל אחת מהקריאות הקטנות ע"י השוואת כל קריאה לעומת רצף דנ"א ידוע מראש (במידה ונאפשר מרווח טעות שנובע מהשוני של 0.01% בין הרצפים).

**Burrows-Wheeler Aligner**

Burrows-Wheeler Aligner או בקצרה BWA היא חבילת תוכנה המשמשת למיפוי רצפי דנ"א קצרים מול רצף דנ"א ארוך.

ב-BWA יש שלושה גרסאות:

* bwa backtrack
* bwa sw
* bwa mem

הגרסה הראשונה עובדת על קריאות עד אורך של 100bp, הגרסאות האחרון עובדות על קריאות באורכים של 70bp-1Mbp. אנו נתמקד בגרסה הראשונה (סיבה תבוא בהמשך).

האלגוריתם שעומד בבסיסו של BWA 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 שיש הבדלים ברצפי ה-דנ"א של אנשים שונים וכן בעובדה שלעיתים יש גם טעות קריאה של המכונה.

ניתן להשתמש בחבילת התוכנה עבור רצפי דנ"א שנעשו בנפרד או עבור רצפי דנ"א שנקראו כזוג. כאשר נחפש את הרצפים הקטנים בתוך הרצף הארוך נוכל להשתמש בעובדה ששני רצפים שנקראו יחד מופיעים קרוב אחד לשני כדי לחשב את המיקום הנכון שבו הרצפים הופיעו בדיוק גבוהה יותר מאשר אם היה לנו רק רצף אחד. בגלל שסיכויי הוודאות גבוהים יותר עבור רצפים שנקראו כזוג אנו נעבוד עם זוגות ולא עם האופציה הפשוטה ביותר שהיא רצפי קריאה בודדים.

בחבילת התוכנה יש שלושה שלבים עיקריים

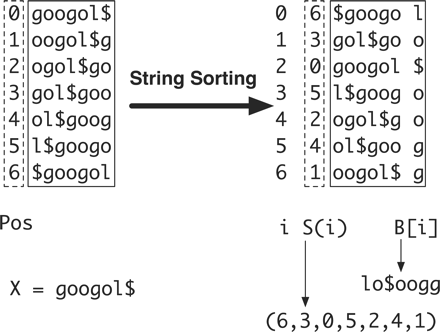
* שלב 1 - Index

בשלב הראשון נטען רצף דנ"א שלם לזיכרון ומתבצעת עבודה מקדימה.

תחילה לוקחים את המחרוזת ומייצרים את כל המחרוזות שניתן לייצר ע"י סיבוב של התווים במחרוזת, לכל מחרוזת נשמור גם את המספר של תווים שעברו מההתחלה לסוף בתוך מערך שייקרא suffix array (מספר זה הוא בעצם המיקום של התו הראשון במחרוזת שהתקבלה בתוך המחרוזת המקורית). לאחר מכן ממיינים את המחרוזות לפי סדר לקסיקוגרפי תוך עדכון הסדר בהתאם בתוך ה-suffix array. בסופו של דבר נשמור את האות האחרונה של כל מחרוזת וכך נקבל את suffix array וכן את ה-circulation string (שנסמן באות B).

הפעולה האחרונה בשלב הנוכחי 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 השלב הבא ולא נתמקד בהן.

דוגמה:

מחרוזת ההתייחסות היא

"googol". נבצע סיבוב של

המחרוזת, נקבל את כל

המחרוזות האפשריות

ונמיין לפי סדר לקסיקוגרפי.

פונקצית O עבור הדוגמה פונקצית C עבור הדוגמה

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

שלב זה הוא קבוע שקורה פעם אחת בלבד בתחילת הריצה ולכן לא נתמקד בחלק זה של האלגוריתם כלל.

* שלב 2 - Alignment

בשלב זה אנו מקבלים מחרוזות קטנות רבות ומבצעים חיפוש שלהן בתוך מחרוזת ההתייחסות (המחרוזת מהשלב הקודם).

החיפוש שמתבצע הוא חישוב מסוג inexact matching, כלומר חיפוש שמתחשב גם במרווח טעות של עד z תווים בכל מחרוזת.

האלגוריתם מבצע את החיפוש תוך התחשבות במקרים הבאים:

יש הכנסה (insertion) של גן, כלומר התו הנוכחי במחרוזת הוא תו שנקרא בטעות ע"י המכונה ואין להתייחס אליו מלבד הקטנת מרווח הטעות.

יש מחיקה (deletion) של גן, כלומר נשמט תו מהמחרוזת ע"י המכונה וצריך להמשיך את הבדיקה לאחר הצבת גן נוסף במיקום הנוכחי (גם כאן נקטין את מרווח הטעות).

יש התאמה בתווים בין מחרוזת ההתייחסות והמחרוזת אותה מחפשים ונמשיך את הבדיקה מהתו הבא.

יש חוסר התאמה בתווים בין מחרוזת ההתייחסות והמחרוזת אותה מחפשים ולכן נקטין את מרווח הטעות ונמשיך את הבדיקה מהתו הבא.

תוך התחשבות בכל המקרים שלהלן השלב הנוכחי מוצא עבור מחרוזת קטנה את כלל המיקומים שלה בתוך מחרוזת ההתייחסות. המיקומים בהם המחרוזת אותה מחפשים מופיעה בתוך מחרוזת ההתייחסות מיוצגים ע"י קבוצה של מקטעים (כל מקטע מורכב מנקודת תחילה ונקודת סיום) כאשר כל מקטע (SA interval) מייצג את נקודת ההתחלה ואת נקודת הסיום של המופעים של המחרוזת שמחפשים בתוך ה-burrows wheeler transform. כדי לשחזר את הקואורדינאטות של ההתאמות ביחס למחרוזת ההתייחסות נעבור על מערך התחיליות (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 הבא – (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 מקטע דנ"א קטן. למרות חשיבותו לא נתמקד בחלק זה מכיוון שפרויקט אחר מתמקד בו וממקבל אותו.

* שלב 3 - Pair end mapping

בשלב הזה אנו מקבלים את תוצאות שלב ה-alignment ומעבדים אותן לתוצאה סופית.

כזכור עבור כל מחרוזת שעוברת את שלב ה-alignment אנו מקבלים קבוצה של מקטעים (כל מקטע מורכב מנקודת התחלה ומנקודת סיום) בשם SA interval, כל מקטע מייצג את המיקומים בהם המחרוזת נמצאה בתוך מחרוזת ההתייחסות (מחרוזת ה-דנ"א השלמה) אך מקודד לפי burrows wheeler transform.

מכיוון שכל מקטע (SA interval) מייצג את המיקומים של המחרוזת בתוך ה-Suffix array, נצטרך תחילה להמיר כל מקטע לקואורדינאטות המתאימות לו בתוך מחרוזת ההתייחסות עצמה ולאחר מכן נחפש את ההתאמה הטובה ביותר מבין כל ההתאמות שהתקבלו.

שלב זה של האלגוריתם מתבצע עבור כל זוג מקטעי דנ"א שנקראו במקביל. חשיבותו של החלק הזה מתבטאת בכל שהוא מעבד את כלל המיקומים שהתקבלו ומספק את התוצאה הטובה ביותר. במהלך הפרויקט אנו נתמקד במקבול התהליך שנמצא בשלב הזה ובניסיון להאיץ את החישובים עד כמה שניתן.

# תיאור הבעיה

כיום, יש בבית החולים הדסה מכונה של חברת illumina העובדת בשיטת ה-NGS ויוצרת ריצוף חלקי של ה-דנ"א של החולה. המכונה משתמשת באלגוריתם ה-BWA בגרסתו הראשונה, בנוסף המכונה משתמשת ב-coverage 30, כלומר נעשות קריאות דנ"א קצרות שאם נסכום את אורכן הכולל נקבל אורך הגדול פי 30 מאורך דנ"א שלם (מכיוון שהרצפים הקטנים נקראים בצורה אקראית אנו נבצע פי 30 יותר קריאות כדי להגדיל את ההסתברות שנקרא כל חלק ב-דנ"א לפחות פעם אחת).

הבעיה בתהליך היא שחיפוש מחרוזת קטנה (אחת מהקריאות ה-דנ"א הקטנות) מול מחרוזת ההתייחסות (רצף דנ"א ידוע מראש), שלב ה-alignment, וכן תהליך שחזור המיקום, שלב ה-pairing, לוקחים זמן רב ומשאבים וכיום בית החולים שוכר חוות מחשבים (עלויות של עשרות אלפי דולרים ליום) כדי לבצע את הפעולות הללו במהירות.

כיום שלב ה-pairing לוקח פי שלושים יותר זמן מאשר שלב ה-alignment ולכן נתמקד בשלב ה-pairing ונאיץ את פעולתו.

שלב ה-pairing מורכב מכמה תתי שלבים אך אנו נתמקד בשניים העיקריים שבהם:

* שחזור מיקומי הופעת מחרוזת הקריאה הקצרה בתוך מחרוזת ההתייחסות (מחרוזת ה-דנ"א הארוכה), כלומר המרה של ה-SA interval שהתקבלו עבור מחרוזת הקריאה למיקום הממשי בתוך מחרוזת ההתייחסות.

בשלב זה נתמקד בהמשך הפרויקט.

* מציאת המיקום הטוב ביותר מבין כלל המיקומים שנמצאו עבור מחרוזת הקריאה בתוך מחרוזת ההתייחסות.

בשלב זה התמקדנו עד כה ונתאר את ההתקדמות.

הבעיה בשלב ה-pairing היא שצריך לחזור על התהליך עבור כל קריאה שנעשתה, מכיוון שיש קריאות רבות שצריך לעבד וכן מספר התאמות לכל קריאה, נוצר בשלב זה צוואר בקבוק כאשר קריאות רבות עוברות את התהליך אחת אחרי השנייה.

בנוסף לבעייתיות המוסברת לעיל בית החולים משתמש בקריאות קצרות (מחרוזות באורך 35bp, מה שמסביר למה נעשה שימוש בגרסה הראשונה של bwa שכן היא היחידה שעובדת עם קריאות קצרות) שנותנות מספר רב של התאמות בתוך מחרוזת ההתייחסות, עובדה זו גורמת לעומס עבודה בשחזור המיקום של כל מחרוזת מכיוון שהיא גורמת למספר רב של התאמות שצריך לשחזר את מיקום ההתאמה וכן להשוות מול שאר ההתאמות, מה שגורם עוד יותר להגברת אפקט צוואר בשלב זה.

# תיאור הפתרון

כפי שכבר צוין אנו נתמקד בשלב ה-pair end mapping של אלגוריתם ה-BWA.

כזכור בשלב ה-pairing היו שתי פעולות עיקריות:

* שחזור הקואורדינאטות של ההתאמות במחרוזת ההתייחסות ע"י המרה של המקטעים המייצגים את ההתאמות תוך שימוש ב-suffix array.
* בחירת המיקום האופטימאלי מבין כלל המיקומים שאותרו.

כדי להאיץ את החישובים האלה נרצה למקבל את התהליך ונשתמש ב-GPU של חברת NVIDIA העובד בארכיטקטורת CUDA. מכיוון שב-GPU יש מספר רב של מעבדים נוכל להשתמש בחוטים (thread) כדי לבצע חישובים רבים במקביל.

כדי לקבל את התוצאות הטובות ביותר ממקבול התהליך נרצה להפעיל מספר מקסימאלי של חוטים. כל חוט יריץ את תהליך ה-paring עבור קריאות שונות וכך נקבל האצה של התהליך. בשביל האצה מירבית כל חוט יריץ קוד שיהיה יעיל בזמן ריצה (ברור שיעילות של חוט משפיעה על זמן הריצה הכולל) וכן יעיל בזיכרון. חשיבות יעילות הזיכרון נובעת מהעובדה שהזיכרון ב-GPU הוא מוגבל ולכן נרצה שכל חוט יזדקק לכמות זיכרון מינימאלית כדי לבצע את פעולתו מכיוון שכך מספר החוטים שנוכל להפעיל יהיה גדול יותר.

**Smith-Waterman**

כדי למצוא את המיקום הטוב ביותר של מחרוזת הקריאה בתוך מחרוזת ההתייחסות מתוך כלל המיקומים שנמצאו נרצה לתת ציון לכל מיקום שקיבלנו.

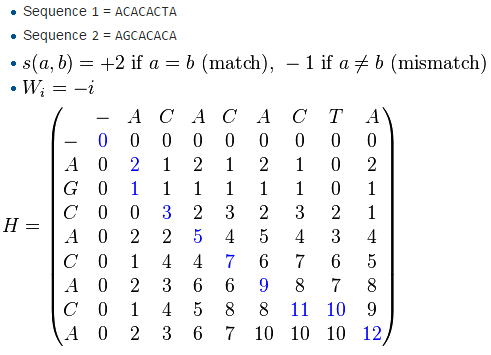
אלגוריתם Smith-Waterman או SW בקצרה, משווה בין שתי מחרוזות ומחזיר ציון עבור ההתאמה בניהן.

כעת האלגוריתם ממומש בקוד הנתון בחבילת התוכנה אך מכיוון שהוא איננו קריא (חוסר תיעוד ואי סדר בפונקציה הקשורה ל-SW) הוחלט לממש את האלגוריתם בעצמנו כדי להבין את עיקרי האלגוריתם. בנוסף יש בפנינו שתי אפשרויות: למקבל את הקוד הקיים במערכת, או לממש את האלגוריתם מחדש, למקבל אותו ולהחליף את הקוד הקיים במערכת.

האלגוריתם עובד עם הנתונים הבאים:

* שתי מחרוזות (מחרוזת a בעלת אורך m ומחרוזת b בעלת אורך n) שנרצה לדעת את ציון ההשוואה בניהן.
* פונקצית התאמת התווים (s(a,b)), כלומר הציון עבור זהות של אותיות בשתי המחרוזות וכן ציון עבור שוני של אותיות בשתי המחרוזות.
* פונקצית עלות של מחיקה (W(k)), כלומר הציון עבור השמטה של תו.
* פונקצית עלות של הכנסה (W(l)), כלומר הציון עבור הוספה של תו.

האלגוריתם עובד בשיטה הרקורסיבית (לא מומש כלל) ומחשב את הערכים לפי:

דוגמה:

עבור המחרוזות

"ACACACTA" ו-"AGCACACA"

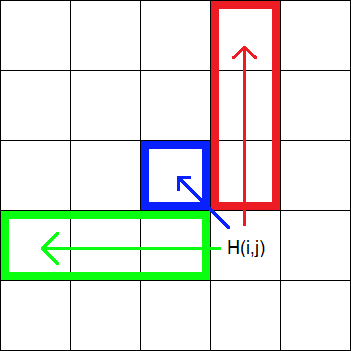
והפונקציות המוגדרות משמאל, ציון

ההתאמה בין המחרוזות הוא 12 וכן כל

חישובים הביניים מוצגים בטבלה.

שיפור ראשון (מומש)

מכיוון שחישוב רקורסיבי אינו יעיל ודורש את פתרון אותן תתי בעיות מספר רב של פעמים שיפור ראשון יהיה להשתמש בתכנון דינאמי.

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 הנוסחה מלמעלה.

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. מכיוון שיש לנו תאים שכאלו זמן הריצה הכולל הינו .

לאחר שיפור זה נקבל:

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

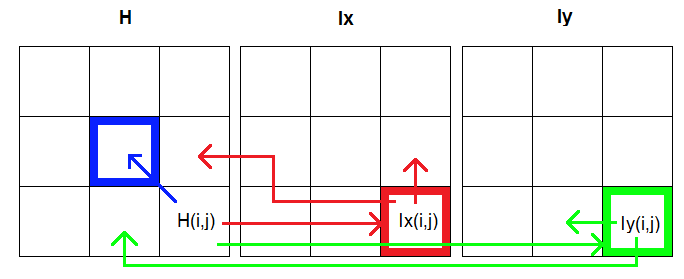
שיפור שני (מומש)

מכיוון שבכל פעם נבדק המקסימום של אותה שורה לכל תא בשורה (לכל תא בשורה סורקים את כל התאים שמשמאלו בשורה, ירוק בתמונה הקודמת) וכן בכל פעם נבדק המקסימום של אותה עמודה לכל תא בעמודה (לכל תא בעמודה סורקים את כל התאים שמעליו בעמודה, כחול בתמונה הקודמת), במקום לבצע את החישוב מחדש עבור כל תא נשתמש בשתי טבלאות נוספות כדי לשמור את המקסימומים המדוברים.

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 שלישי (מומש)

מכיוון שרק התוצאה המקסימאלית מעניינת אותנו ולא הדרך דרכה הגענו לתוצאה המקסימאלית, 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.

בעצם כעת 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 רביעי (מומש)

התנאים הבאים מתקיימים:

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

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

בעצם יוצא שכדי לחשב את התאים ה-(i,j) בטבלאות H,Ix,Iy צריכים רק את התאים שמקיפים אותם ולא את כל התאים שקדמו להם נשמור את השורה החדשה בלבד אך נוודה שכל פעם לפני מילוי הערכים החדשים בתאים 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 |  |

**בדיקות**

בדיקות נכונות:

כדי לבדוק את נכונות הגרסאות השונות ולברר שאין טעות באף אחת מהנוסחאות וכן במימושים השונים שעובדים הרבה עם שמירת נתונים בצד, הוחלט על בדיקת התוצאה הסופית עבור כל אחת מהגרסאות וכן בדיקת שלושת הטבלאות בכל אחת מהגרסאות כדי לוודא שגם חישובי הביניים נעשו נכונה.

כדי לבצע את הבדיקות הוספתי גרסה מיוחדת עבור כל אחת מהגרסאות (תחת התיקייה PrintableSW) שבעזרתה ניתן להדפיס כל אחת מהתוצאות השונות. כל תוצאת הדפסה נשמרה לקובץ והושוותה לקובץ התוצאות המצופה על ידי תוכנה הבודקת זהות בין קבצים.

תוצאות – למרות שהתוצאה הסופית היתה נכונה עבור כל אחת מהבדיקות, הטבלאות שהציגו את חישובי הביניים לא היו נכונות ואותרה שגיאה בנוסחאות לחישוב טבלאות Ix ו-Iy. הנוסחאות תוקנו ובדיקה מחדש נתנה תוצאה נכונה לחלוטין.

בדיקות זמני ריצה:

כדי להשוות בין זמני הריצה נעשו שתי בדיקות שונות:

בדיקה ראשונה – מחרוזות באורך קבוע.

בבדיקה הראשונה הורץ אלגוריתם SW עבור מספר משתנה של מחרוזות כאשר כל מחרוזת הינה באורך 35 תווים.

תוצאות הבדיקה יצאו כמצופה, כלומר זמן הריצה של האלגוריתם היה קבוע מכיוון שאורך המחרוזת היה קבוע והגורם היחיד שהשפיע היה מספר ההרצות (שהשתנה ליניארית).

בדיקה שנייה – מחרוזות באורך משתנה.

בבדיקה השנייה הורץ אלגוריתם SW כ-100,000 פעמים עבור מחרוזות בעלות מספר שונה של אורכים (כדי לבדוק איך אורך המחרוזת ישפיע על יעילות האלגוריתם).

תוצאות הבדיקה יצאו שבקירוב עבור כל 15 תווים שנוספים לאורך המחרוזת, זמן הריצה גדל פי 2 (הגרפים אכן אקספוננציאלים אך בגלל שהגרף הראשון גדול כמעט ולא ניתן לראות זאת על הגרפים הקטנים יותר).

בדיקות זיכרון:

בדיקת הזיכרון נעשה על ידי ניתוח תיאורטי של יעילות זיכרון של כל גרסה.

|  |  |
| --- | --- |
| גרסה מספר | יעילות זיכרון |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |

השינוי בין גרסה מספר 3 וגרסה מספר 4, כלומר פי שניים פחות זיכרון, הוא לבדו שינוי עצום מכיוון שתיאורטית עבור כרטיס מסך עם זיכרון מוגבל ואין סוף מעבדים מספר המעבדים שנוכל להפעיל תלוי בכמות הזיכרון שכל מעבד צורך, אם נקטין את כמות הזיכרון של כל מעבד בחצי אזי נוכל להריץ פי שניים יותר מעבדים ולהקטין את זמן הריצה הכולל פי 2.

השינוי בין גרסאות 3ו-4 הוא כזה שניתן להבין בבירור את ההבדל הרב בין צריכת זיכרון ליניארית בגרסאות 3 ו-4 לעומת צריכת הזיכרון הריבועית בגרסאות 1 ו-2.

מכיוון שעדיין לא הוחלט אם נמקבל את הקוד הקיים במערכת או אם נכתוב קוד חדש ונחליף את הקוד הקיים לא ניתן בשלב זה לקבוע האם נצטרך לבצע בדיקות נוספות על הקוד אותו אנו נכתוב (בין הבדיקות יהיו גם בדיקות אינטגרציה וקבלה) או שנצטרך לבצע בדיקות על הקוד הקיים לאחר שנמקבל אותו.

# השוואה לפתרונות בספרות

# כיום המערכת עובדת עם קוד BWA רגיל (יש בו אפשרות מובנית למקבול התהליך אך לאותו המחשב), מאז שהקוד נכתב יצאו שתי גרסאות נוספות לאלגוריתם BWA-SA ו- BWA-MEM.

# מכיוון ש- BWA-MEM משפר רק עבור קריאות של 70bp ומעלה, ובית החולים הדסה עובד עם קריאות של 35bp בלבד, מירב הסיכויים הם שלא נשדרג את הקוד ל- BWA-MEM מכיוון ששדרוג זה לא ישפיע כל עוד בית החולים עובד עם קריאות קצרות אלו.

# אם נרצה לטעון את כל האינדקס הנוצר במהלך ריצת הקוד, נצטרך להשתמש בזיכרון בגודל 1.25 כפול אורך הגנום. אורך הגנום האנושי (בהנחה שנייצג כל אות ע"י שני ביטים) הוא 770 מגה בייט. ולכן טעינת כל האינדקס הנוצר בשלב הראשון של הקוד ידרוש כ-1GB של זיכרון, עם ההתקדמות בטכנולוגיית כרטיסי המסך יש לנו כיום 4GB של זיכרון על כרטיס המסך ולכן נוכל לטעון את כלל האינדקס לזיכרון, זה דבר שלא נעשה עד כה ואנו מצפים שישפר את זמן הריצה בצורה משמעותית.

# רשימת ספרות

# מאמר על אלגוריתם BWA הכולל הסברים על שלבי ה-index, alignment ושחזור פתרון.

# [Fast and accurate short read alignment with Burrows–Wheeler transform](http://bioinformatics.oxfordjournals.org/content/25/14/1754.long)

# הסבר על אלגוריתם SW

# [Smith-Waterman algorithm](http://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm)

# מספר שינויים לפישור יעילות באלגוריתם SW

# [Smith-Waterman variations](http://www.public.asu.edu/~cbaral/cse591-s03/classnotes/seq-align.pdf)

# דוגמאות הרצה נוספות של אלגוריתם SW

# [SW example 1](http://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm)

# [SW example 2](http://amrita.vlab.co.in/?sub=3&brch=274&sim=1433&cnt=1)

# [SW example 3](http://www.slideshare.net/avrilcoghlan/the-smith-waterman-algorithm)