דוח סיכום פרויקט

בנושא: אלגוריתמי חיפוש מחרוזות בטקסט

מבצעים: חליל קבלאוי 206758088

סלאם עאסלה 314892761

מנחה: מורדו שלום

דף תודות

אנחנו מודים למנחה הפרויקט ד"ר מורדו שלום על העזרה המקצועית במהלך הפרויקט ויחסו האישי לאורך כל הדרך.

תודה ל ד"ר תמר צמח, שבדקה את הצעת הפרויקט, ואישרה לנו אותה ונתנה הנחיות בכתיבת ספר זה.

על הכל, תודה

תוכן העניינים

עמוד

שער.........................................................................1

דף תודות................................................................2

רשימת איורים ........................................................4

תקציר ....................................................................5

רשימת סמלים וקיצורים ..........................................7

פרק1 : מבוא............................................................8

רקעBM ............................................................8

רקע KMP..........................................................11

פרק2 : תאור כללי ...................................................13

פרק 3 : תאור מפורט ...............................................16

KMP................................................................16

BM..................................................................19

פרק 4 : תוצאות .....................................................23

פרק 5 : סיכום .......................................................30

פרק 6 : נספחים .....................................................30

פרק 7 : רשימת מקורות ..........................................32

רשימת איורים, תמונות, תרשימים

הסבר ויזואלי....................................................................................................11

USE CASE DIAGRAM.................................................................................12

CLASS DIAGRAM.......................................................................................13

SEQUENCE DIAGRAM..............................................................................14

ויזואליזציהKMP.............................................................................................16

ויזואליזציהBM................................................................................................19

תוצאות...........................................................................................................21-26

תקציר

הגדרת הבעיה: לממש ולעשות ויזוליציה של שני אלגוריתמים. הכוונה לתת למשתמש אופציות לבצע צעד או רצף של צעדים של האלגוריתם או של האלגוריתם כולו על קלטים של משתמש ועל קלטים שאנחנו נחליט שהם מעניינים לצורך לימוד האלגוריתם.

האלגוריתמים הם:

* Boyer-Moore
* Knuth-Morris-Prat

**דרך הפתרון ל KMP :**

מטרתו של האלגוריתם היא למצוא מופעים של מחרוזת Pבתוך מחרוזת T (טקסט) כאשר אורך של P הוא m והאורך של Tהוא n.

בכל פעם שהאלגוריתם יתקל בנקודת חוסר התאמה בטקסט, הוא יחזור להתאים את הרישא הארוכה ביותר של התבנית המתאימה לסיפא הארוכה ביותר של הטקסט בנקודה בו נתקלנו בחוסר התאמה.

עיבוד מקדים :

אלגוריתם KMP מעבד מראש מחרוזת ובונה מערך עזר בגודל m (זהה לגודל התבנית) המשמש לדילוג על תווים תוך כדי התאמה. המערך מציין את הקידומת המתאימה הארוכה ביותר שהיא גם סיומת . המחרוזת כלה לא תחשב כקידומת מתאימה.

נקרא למערך LPS שהגודל שלו יהיה שווה לגודל מחרוזת החיפוש LPS .

הערה: LPS במקום אפס שווה לאפס (לא כמו מה שמצוין בתמונה אנחנו מתחילים עם 0 ולא עם -1 )

אנו מחפשים LPS בתבניות משנה. באופן ברור יותר אנו מתמקדים במיתרי משנה של דפוסים שהם קידומת וסיומת. עבור כל תת-תבנית שמתחילה מאפס עד i כאשר i = 0 עד (m-1) LPS במקום m מאחסן אורך של הקידומת המתאימה המקסימלית שהיא גם סיומת של התבנית LPS

איטראציה באלגוריתם זה:

לצורך הדגמה נניח שאנחנו באיטראציה הראשונה, אז בודקים את התו הראשון בטקסט הנבחר (מיקום 0), אם התו הראשון במחרוזת החיפוש שווה לתו הראשון במחרוזת הנבחרת אז ממשיכים לבדוק את התו השני בשתי המחרוזות וכך הלאה עד ש: או שמוצאים כל התווים של המחרוזת (כל התווים שווים בשתי המחרוזות) או שמוצאים אי שוויון.

**דרך הפתרון ל BM :**

האלגוריתם מבצע עיבוד מקדים למחרוזת החיפוש (התבנית), אך לא לטקסט שבו יבוצע החיפוש. תם בויאר-מור משתמש במידע הנאסף בשלב העיבוד המקדים על מנת לדלג על חלקים בטקסט, והודות לכך הוא משיג ביצועים טובים ביחס לאלגוריתמי חיפוש אחרים.

באופן כללי, האלגוריתם רץ מהר יותר ככל שמחרוזת החיפוש ארוכה יותר. התכונות החשובות של האלגוריתם הן שהוא מחפש מסוף מחרוזת החיפוש ולא תחילתה, והוא מדלג בטקסט על מספר תווים במקום לחפש תו-תו.

עיבוד מקדים :

לצורך אלגוריתם זה נצטרך, לבנות טבלת אי התאמה שתשמש אותנו לחשב את מרחק הקפיצה בכל פעם שנקבל אי שוויון בין שני תווים.

הטבלה מכילה מופע אחד בלבד של כל תוו שנמצא במחרוזת החיפוש ונותנת לו ערך לפי הנוסחה הבאה:

Max(1,length\_of\_pattern-index\_of\_current\_char-1)

ובסוף הטבלה מוסיפים תו למשל \* שערכו שווה לאורך מחרוזת החיפוש.

איטראציה באלגוריתם זה:

לצורך הדגמה נניח שאנחנו באיטראציה הראשונה, אז בודקים את התו האחרון בטקסט הנבחר (מיקום גודל המחרוזת פחות 1) , אם התו האחרון במחרוזת החיפוש שווה לתו האחרון במחרוזת במקום גודל המחרוזת פחות 1 אז ממשיכים לבדוק את התו לפני האחרון בשתי המחרוזות וכך הלאה עד ש: או שמוצאים כל התווים של המחרוזת (כל התווים שווים בשתי המחרוזות) או שמוצאים אי שוויון.

רשימת סמלים וקיצורים

BM algorithm = Boyer-Moore algorithm

KMP algorithm = Knuth-Morris-Prat algorithm

**פרק1: מבוא**

הגדרת הבעיה: לממש ולעשות ויזוליציה של שני אלגוריתמים. הכוונה לתת למשתמש אופציות לבצע צעד או רצף של צעדים של האלגוריתם או של האלגוריתם כולו על קלטים של משתמש ועל קלטים שאנחנו נחליט שהם מעניינים לצורך לימוד האלגוריתם.

* Boyer-Moore
* Knuth-Morris-Prat

Boyer-Moore

רקע

**אלגוריתם בויאר-מור** הוא [אלגוריתם חיפוש מחרוזות](https://he.wikipedia.org/wiki/%D7%90%D7%9C%D7%92%D7%95%D7%A8%D7%99%D7%AA%D7%9E%D7%99%D7%9D_%D7%9C%D7%97%D7%99%D7%A4%D7%95%D7%A9_%D7%9E%D7%97%D7%A8%D7%95%D7%96%D7%95%D7%AA) יעיל המשמש מדד השוואה סטנדרטי עבור אלגוריתמי חיפוש. האלגוריתם פותח על ידי החוקרים האמריקאים [רוברט בויאר](https://he.wikipedia.org/w/index.php?title=%D7%A8%D7%95%D7%91%D7%A8%D7%98_%D7%91%D7%95%D7%99%D7%90%D7%A8&action=edit&redlink=1) ו[ג' סטרותר מור](https://he.wikipedia.org/w/index.php?title=%D7%92%27_%D7%A1%D7%98%D7%A8%D7%95%D7%AA%D7%A8_%D7%9E%D7%95%D7%A8&action=edit&redlink=1) ב-1977. האלגוריתם מבצע עיבוד מקדים למחרוזת החיפוש (התבנית), אך לא לטקסט שבו יבוצע החיפוש. האלגוריתם מתאים היטב אפוא ליישומים שבהם תבנית החיפוש קצרה בהרבה מהטקסט או למקרים שבהם נעשה שימוש חוזר בתבנית החיפוש. אלגוריתם בויאר-מור משתמש במידע הנאסף בשלב העיבוד המקדים על מנת לדלג על חלקים בטקסט, והודות לכך הוא משיג ביצועים טובים ביחס לאלגוריתמי חיפוש אחרים. באופן כללי, האלגוריתם רץ מהר יותר ככל שמחרוזת החיפוש ארוכה יותר. התכונות החשובות של האלגוריתם הן שהוא מחפש מסוף מחרוזת החיפוש ולא תחילתה, והוא מדלג בטקסט על מספר תווים במקום לחפש תו-תו.

הגדרות

* ***[ S*[*i*** יסמן את התו באינדקס ה-*i* של המחרוזת *S*, החל מ-1.
* ***S* [*i*..*j*** ]יסמן את תת-המחרוזת ב *S* אשר מתחילה בתו *i* ומסתיימת בתו *j*, כולל.
* **תחילית של *S*** היא תת-מחרוזת *S*[1..*i*] עבור *i* כלשהו בטווח[1… n] {\displaystyle [1,n]}, כאשר *n* היא אורך המחרוזת *S*.
* סיפא של *S* היא תת-מחרוזת *S*[*i*..*n*] עבור *i* כלשהו בטווח[n….1] כאשר *n* היא אורך המחרוזת *S*.
* מחרוזת החיפוש תסומן באמצעות ***P*** ותקרא לעיתים תבנית החיפוש. האורך שלה יסומן באמצעות ***n***.
* הטקסט שבו מחפשים את מחרוזת החיפוש מסומן באמצעות ***T***. האורך שלו יסומן באמצעות ***m***.
* עימוד של *P* ל *T* הוא אינדקס *k* ב *T* שבו התו האחרון של *P* מועמד מול האינדקס k של *T*.
* התאמה של *P* קוראת בעימוד אם *P* שווה ל *T*[(*k*-*n*+1)..*k*].

## תיאור

אלגוריתם בויאר-מור מחפש מופעים של מחרוזת חיפוש P בטקסט שבו מחפשים T באמצעות השוואת תווים בעימודים שונים. במקום שימוש בחיפוש [כוח גס](https://he.wikipedia.org/wiki/%D7%9B%D7%95%D7%97_%D7%92%D7%A1) של כל העימודים בויאר-מור משתמש במידע מעיבוד מקדים של תבנית החיפוש P על מנת לדלג על מספר עימודים גדול ככל האפשר.

לפני שהוצג אלגוריתם זה, השיטה הנהוגה לחיפוש טקסט הייתה לבדוק כל תו בטקסט ביחס לתו הראשון במחרוזת החיפוש, ואם נמצאה התאמה נעשה חיפוש בהתאם לתו הבא במחרוזת החיפוש וכן הלאה. בצורה זו נדרשים להשוות כל תו בטקסט.

ההבחנה החשובה באלגוריתם זה היא שאם סוף מחרוזת החיפוש מושוות לטקסט, ניתן לדלג על תווים בחיפוש ולא לבדוק כל תו בטקסט. הסיבה לכך היא שאם משווים את התו האחרון של תבנית החיפוש לתו בטקסט ואין התאמה, אין צורך לחפש אחורנית בתבנית החיפוש. אם התו בטקסט לא מתאים לאף אחד מהתווים בתבנית החיפוש, אז התו הבא להשוואה בטקסט נמצא n תווים בהמשך, כאשר n הוא אורך תבנית החיפוש. אם התו נמצא בתבנית החיפוש, נדרשת הזזה חלקית של תבנית החיפוש אל התו המתאים והתהליך חוזר על עצמו. החיפוש בטקסט באמצעות דילוגים לעריכת השוואות במקום לבדוק כל תו ותו מצמצם את מספר ההשוואות הנדרשות ובכך תורם ליעילות האלגוריתם.

באופן פורמלי יותר, האלגוריתם מתחיל בעימודk=n  {\displaystyle k=n}, כך שהתחלת תבנית החיפוש P מועמדת מול ההתחלה של הטקסט T. נערכת השוואה בין התווים ב-P וב-T אשר מתחילה מהאינדקס n בתבנית P ו-k ב-T, והולכת אחורה: המחרוזות מושוות ביחס לסופה של P ועד לתחילתה. ההשוואות ממשיכות עד הגעה לתחילתה של תבנית החיפוש P (במקרה של התאמה) או עד לאי התאמה שבעקבותיה העימוד מוזז קדימה בהתאם לערך המותר המרבי בהתאם למספר כללים. ההשוואות נערכות פעם נוספת בעימוד החדש, והתהליך חוזר על עצמו.

חוקי ההזזה ממומשים באמצעות טבלת חיפוש המוגדרת באמצעות עיבוד מקדים של תבנית החיפוש P.

## כללי ההזזה

ההזזה מחושבת באמצעות שני כללים: כלל התו הלא מתאים, וכלל הסיפא הטובה. הזזה בפועל היא הערך המרבי המוגדר על ידי שני כללים אלו.

### כלל התו הלא מתאים

#### תיאור

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - | - | - | - | X |  | - | - | K | - | - | - |
| A | N | P | A | N |  | M | A | N | A | M | - |
| - | N | N | A | A |  | M | A | N | - | - | - |
| - | - | - | N | N |  | A | A | M | A | N | - |

הדגמה של הכלל עבור תבנית החיפוש **NNAAMAN**.

כלל התו הלא מתאים מתייחס לתו ב-T שבו ההשוואה נכשלת. המופע הקודם של תו זה ב-P נמצא, ומוצעת הזזה המעבירה את המופע אל מול האי התאמה ב-P. אם התו הלא מתאים לא נמצא מלפני P, מוצעת הזזה המזיזה את P מעבר לתו הלא מתאים.

#### עיבוד מקדים

ישנן שיטות שונות בנוגע לטבלאות חיפוש עבור כלל התו הלא מתאים, להלן גרסה פשוטה הפועלת בזמן קבוע: יוצרים מערך דו ממדי שבו האינדקס הראשון הוא התו c באלפבית והאינדקס השני הוא האינדקס i בתבנית החיפוש. טבלת החיפוש תחזיר את המופע של c ב P עם האינדקס השני בגודלו {\displaystyle j<i}j<i או {\displaystyle -1}-1 אם לא נמצא מופע. ההסטה המוצעת תהיהi-j  .{\displaystyle i-j}.

### כלל הסיפא הטובה

#### תיאור

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - | - | - | - | X | - | - | K | - | - | - | - | - |
| M | A | N | P | A | N | A | M | A | N | A | P | - |
| A | N | A | M | P | N | A | M | - | - | - | - | - |
| - | - | - | - | A | N | A | M | P | N | A | M | - |

הדגמה של הכלל עבור תבנית החיפוש **ANAMPNAM**.

כלל הסיפא הטובה הוא כלל מורכב יותר ביחס לכלל הקודם. הוא הסיבה לכך שההשוואות נעשות מסוף תבנית החיפוש ולא מתחילה. הכלל מוגדר כך:

נניח כי עבור עימוד מסוים של תבנית החיפוש ***P*** ו ***T***, תת-מחרוזת ***t*** של ***T*** מתאימה לסיפא של ***P***, אך יש אי התאמה בהשוואה הבאה (בתו הקודם). במקרה זה מחפשים, במידה וקיים, את העותק האחרון ***t'*** של ***t*** ב ***P*** כך ש ***t'*** אינה סיפא של ***P*** והתו הקודם ל ***t'*** ב ***P*** שונה מהתו הקודם ל ***t*** ב ***P***. נעשית הסטה של ***P*** קדימה כך שתת-המחרוזת ***t'*** ב ***P*** מועמדת מול תת-המחרוזת ***t*** ב ***T***. אם ***t'*** לא קיימת, מזיזים את ההתחלה של ***P*** כך שתעבור את סוף ***t''****ב***T***כך שהתחילית של התבנית המוסטת מתאימה לסיפא של***t***ב***T***. אם לא קיימת הסטה כזו, מסיטים את***P***ב***n***מקומות קדימה. אם נמצא מופע של***P***, אז מסיטים את***P***כך שהתחילית של P המוסטת תתאים לסיפא של המופע של***P***ב***T***. אם לא קיימת הסטה כזו, מסיטים את***P***ב***n***מקומות, כלומר, מסיטים את***P***שתעבור את***t***.*

#### **עיבוד מקדים**

כלל הסיפא הטובה דורש שתי טבלאות: אחת לשימוש במקרה הכללי, ואחת לשימוש כאשר המקרה הכללי לא מחזיר תוצאה משמעותית או שמתקיימת התאמה. טבלאות אלה יסומנו L ו - H בהתאמה. ההגדרות שלהן הם כדלקמן:

עבור כל תו i,‏ {\displaystyle L[i]}L[i] הוא העמדה הגדולה ביותר שקטנה מ-n כך שהמחרוזת {\displaystyle P[i..n]}P[i…n] מתאימה לסיומת של {\displaystyle P[1..L[i]]}P[1….L[i]] כך שהתו הקודם לסיומת לא שווה P[i-1] {\displaystyle P[i-1]}

{\displaystyle L[i]}L[i] מוגדר להיות אפס אם אין עמדה המקיימת את התנאי.

נגדיר את H[i]  כסיומת הארוכה ביותר של {\displaystyle P[i..n]} P[i…n] שהיא גם קידומת של P, אם קיימת כזו. אם לא קיימת כזו, הערך של {\displaystyle H[i]}H[i] יהיה אפס.

Knuth-Morris-Prat

**רקע**

**אלגוריתם KMP** הוא אלגוריתם ל[התאמת תבניות](https://he.wikipedia.org/w/index.php?title=%D7%94%D7%AA%D7%90%D7%9E%D7%AA_%D7%AA%D7%91%D7%A0%D7%99%D7%95%D7%AA&action=edit&redlink=1) הפועל בזמן ליניארי במקרה הגרוע. האלגוריתם התגלה בשנת 1970 על ידי [דונלד קנות'](https://he.wikipedia.org/wiki/%D7%93%D7%95%D7%A0%D7%9C%D7%93_%D7%A7%D7%A0%D7%95%D7%AA%27) ו[וואן פראת'](https://he.wikipedia.org/w/index.php?title=%D7%95%D7%95%D7%90%D7%9F_%D7%A4%D7%A8%D7%90%D7%AA%27&action=edit&redlink=1), ובאופן בלתי תלוי על ידי [ג'יימס מוריס](https://he.wikipedia.org/w/index.php?title=%D7%92%27%D7%99%D7%99%D7%9E%D7%A1_%D7%9E%D7%95%D7%A8%D7%99%D7%A1&action=edit&redlink=1). השלושה פרסמו יחדיו את האלגוריתם בכתב העת [**SIAM**](https://he.wikipedia.org/w/index.php?title=SIAM&action=edit&redlink=1) בשנת 1977.

מטרתו של האלגוריתם היא למצוא מופעים של מחרוזת Pבתוך מחרוזת T (טקסט), כאשר  N= {\displaystyle |T|=n,|P|=m}|T|,|P|= M בעוד האלגוריתם הנאיבי לביצוע פעולה זו פועל בזמן {\displaystyle \mathrm {O} (mn)}O(mn) בזמן הגרוע, אלגוריתם KMP עובד בזמן {\displaystyle \mathrm {O} (n)}O(n) בזמן גרוע.

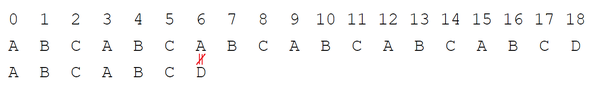
## תיאור האלגוריתם

נניח {\displaystyle T=t\_{0}t\_{1}...t\_{n-1}}T = abcdabc וגם {\displaystyle P=p\_{0}p\_{1}...p\_{m-1}}P= abc

האלגוריתם הנאיבי להתאמת מחרוזות, יחפש עבור כל מיקום בטקסט האם הוא התחלה של המחרוזת. אלגוריתם KMP משתמש ב[טרנזיטיביות](https://he.wikipedia.org/wiki/%D7%98%D7%A8%D7%A0%D7%96%D7%99%D7%98%D7%99%D7%91%D7%99%D7%95%D7%AA) של התאמת מחרוזות על מנת לחסוך השוואות מיותרות. בכל פעם שהאלגוריתם יתקל בנקודת חוסר התאמה בטקסט, הוא יחזור להתאים את הרישא הארוכה ביותר של התבנית המתאימה לסיפא הארוכה ביותר של הטקסט בנקודה בו נתקלנו בחוסר התאמה.

על מנת להתאים בין הרישא הארוכה ביותר לסיפא הארוכה ביותר, נבנה טבלה ייעודית ונוכל להציץ בה בזמן קבוע.

### **הסבר ויזואלי**

כאשר ניתקל בתבנית ABCABCD ובטקסט ABCABCABCABCABCABCD, נקודת אי ההתאמה הראשונה תהיה באינדקס 6[](https://he.wikipedia.org/wiki/%D7%A7%D7%95%D7%91%D7%A5:Kmp_mismatch_at_index_6.png)

ניתן לשים לב, שהאלגוריתם הנאיבי יתחיל לחפש מהאינדקס הבא (1), למרות שניתן לדלג מידית לאינדקס 3. KMP יודע לדלג לאינדקס 3, מכיוון שלאחר שהתאמנו את A באינדקס 0, B באינדקס 1 ו-C באינדקס 2, ברור שלא נוכל להתאים את A באינדקסים 1 או 2.

בנוסף, לאחר שנתחיל את הבדיקה מחדש באינדקס 3, אין טעם לבדוק ששלושת התווים הראשונים זהים, מכיוון שווידאנו זאת כבר קודם, כלומר המחרוזת שהותאמה עד כה היא ABCABC, ומחרוזת זו נגמרת ב-ABC. לכן, יש סיפא באורך 3 (ABC) שמתאימה לרישא באורך 3 גם כן, (ABC), ולכן ניתן לחסוך את ההתאמה של שלושת התווים הראשונים.

על ידי התאמת הרישא לסיפא, אנחנו הלכה למעשה מוודאים שלא יווצר מצב בו אנו מדלגים על התאמה. ניתן לחלק לשני מקרים:

1. אם קיימת סיפא שמתאימה לרישא, אז אי אפשר לדלג על כל אזור אי ההתאמה, אבל בתוך הדילוג אנחנו כבר יודעים שהרישא שווה לסיפא, ולכן אין צורך לבדוק אותה.
2. אם לא קיימת רישא שמתאימה לסיפא, אזי בכל מקום אליו נקפוץ בטווח שכרגע בדקנו לא תמצא התאמה, ולכן ניתן לדלג ישירות על כל החלק המאומת.

מהגדרה זאת ניתן לקבל את ההרגשה שהאלגוריתם פועל בזמן ליניארי, בהנחה שפעולות הבדיקה על הרישא והסיפא פועלות בזמן קבוע

**פרק 2 :תאור כללי**

USE CASE DIAGRAM

Diagram

Description automatically generated

**פונקציונאליות:**

בתחילת הרצת התוכנית, המשתמש מתבקש לבחור אחד משני האלגוריתמים.

לאחר בחירת האלגוריתם המשתמש יבחר באחת משתי האפשריות הקימיות :

1. הרצה ידנית צעד לאחר צעד על ידי המשתמש.
2. הרצה אוטומאטית.

\*הערה: נפרט על כל איטרציה של כל אלגוריתם בהתאם.

לאחר מכן המשתמש יבחר את הקלט: קלט מתוך קובץ טקסט קיים או קלט כלשהו לבחירתו, ואז יכניס את המחרוזת הרצויה לחיפוש.

לאחר סיום כל השלבים הללו התוכנית תרוץ לפי בחירות המשתמש בהתאם.

**CLASS DIAGRAM**

Diagram

Description automatically generated

**Sequence diagram:**

Chart

Description automatically generated

**פרק 3 :תאור מפורט**

**אלגוריתם Knuth-Morris-Prat :**

מטרתו של האלגוריתם היא למצוא מופעים של מחרוזת Pבתוך מחרוזת T (טקסט) כאשר אורך של P הוא m והאורך של Tהוא n.

בכל פעם שהאלגוריתם יתקל בנקודת חוסר התאמה בטקסט, הוא יחזור להתאים את הרישא הארוכה ביותר של התבנית המתאימה לסיפא הארוכה ביותר של הטקסט בנקודה בו נתקלנו בחוסר התאמה.

**עיבוד מקדים:**

שאלה חשובה עולה מההסבר למעלה היא כמה תווים לקפוץ, כדי לדעת את זה מבצעים עיבוד מקדים למחרוזת בעזרת מערך של שלמים שנותנן לנו את כמות התווים שצריך לקפוץ.

אלגוריתם KMP מעבד מראש מחרוזת ובונה מערך עזר בגודל m (זהה לגודל התבנית) המשמש לדילוג על תווים תוך כדי התאמה. המערך מציין את הקידומת המתאימה הארוכה ביותר שהיא גם סיומת . המחרוזת כלה לא תחשב כקידומת מתאימה.

**איטראציה באלגוריתם זה:**

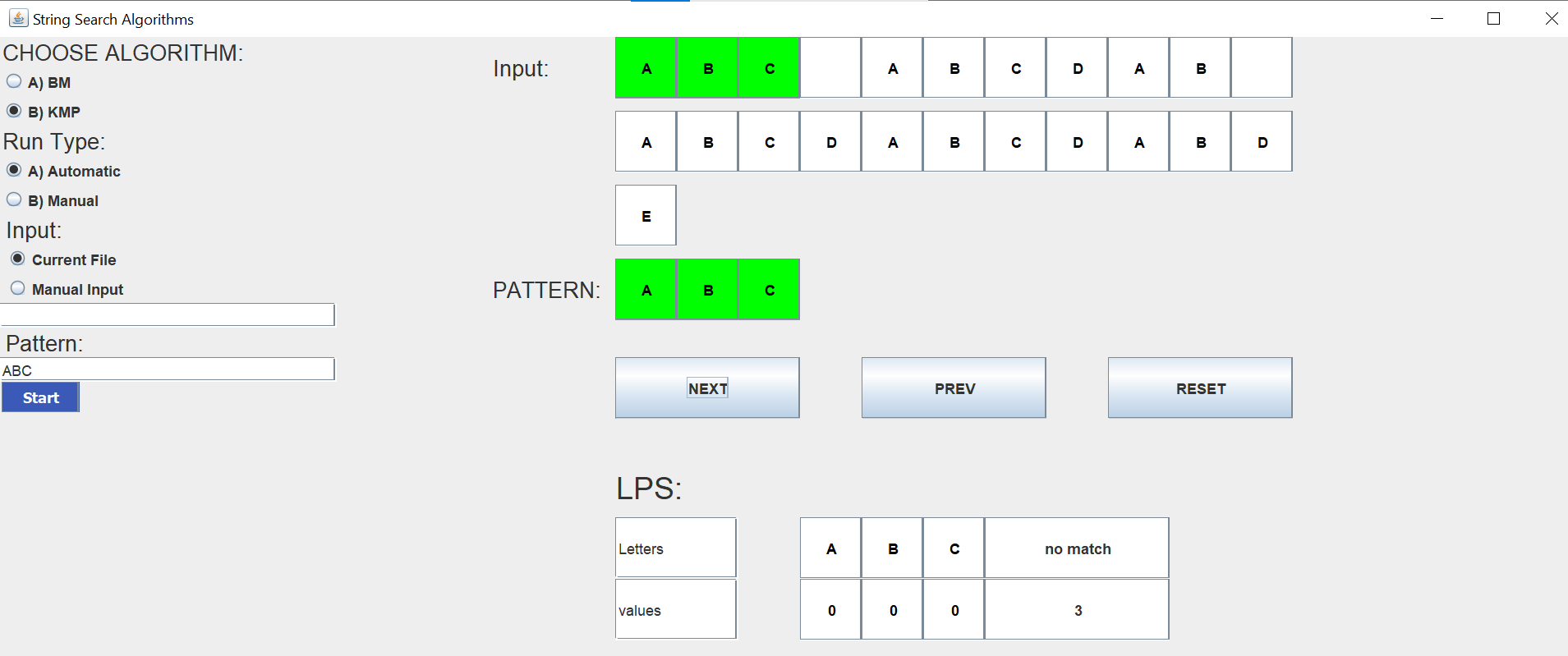
לצורך הדגמה נניח שאנחנו באיטראציה הראשונה, אז בודקים את התו הראשון בטקסט הנבחר (מיקום 0), אם התו הראשון במחרוזת החיפוש שווה לתו הראשון במחרוזת הנבחרת אז ממשיכים לבדוק את התו השני בשתי המחרוזות וכך הלאה עד ש: או שמוצאים כל התווים של המחרוזת (כל התווים שווים בשתי המחרוזות) או שמוצאים אי שוויון.  
במקרה הראשון: המחרוזת אותרה, אז האלגוריתם מודיע על הצלחה וממשיך לחפש את המופע הבא.

במקרה השני: נמצא תווים שונים, אז האלגוריתם מחשב את הקפיצה למיקום התו הבא שצריך ממנו להתחיל את הבדיקה שוב.

תהליך זה מבוצע עד שנאתר את המחרוזת הרצויה או נגיע לסוף הקלט.

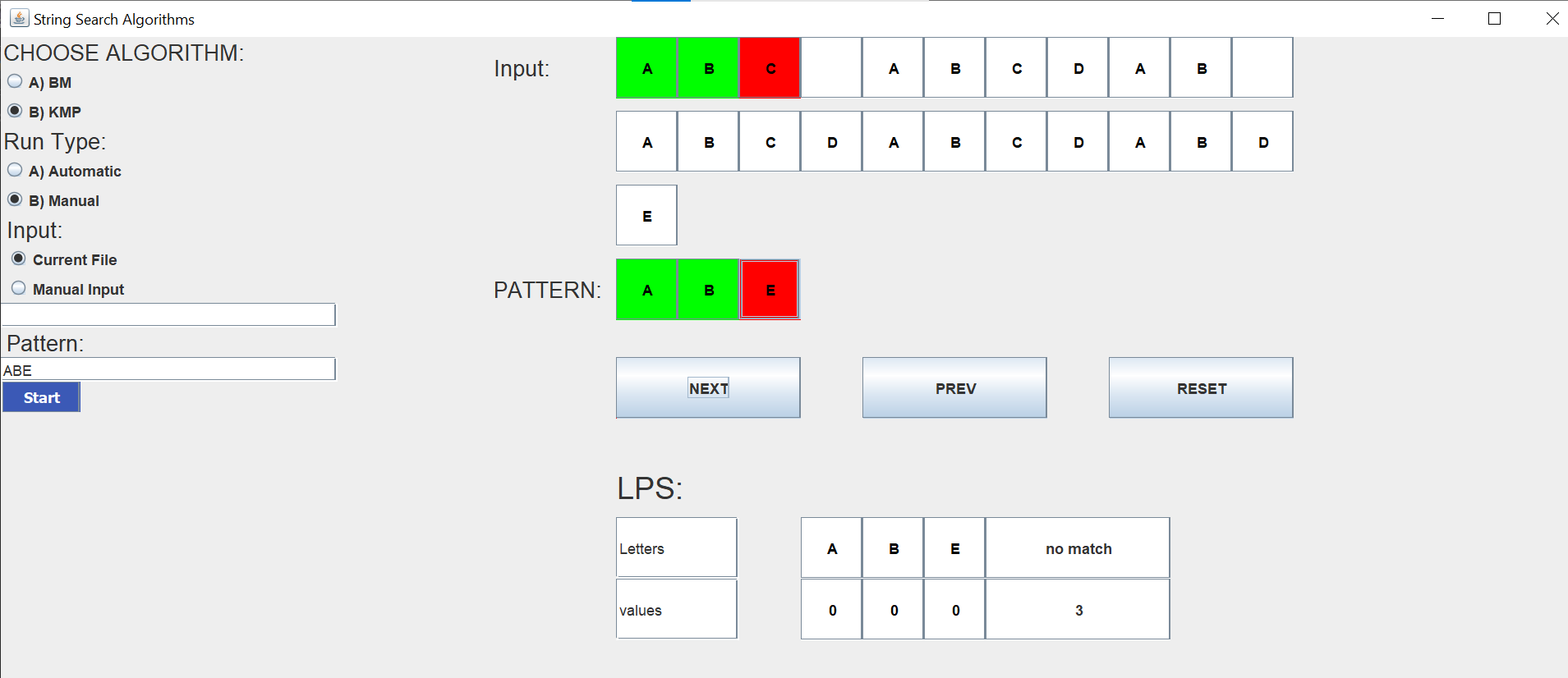
**ויזואליזציה:**

**מקרה ראשון:**



המחרוזת נמצאת, כל תו נמצא נצבע בירוק, וממשיכים לחפש את המופע הבא

**מקרה שני:**



נמצא תו שונה אז נצבע באדום ומחושב גודל הקפיצה (התו הבא שממנו נתחיל לבדוק).

על סימון התו שבודקים בכל שלב נצבע אותו בצע מסויים למשל כחול ואז צבע ירוק אם משווים או אדום במקרה של אי שוויון.

בזמן הויזואליזציה יוצג גם תהליך העיבוד וגם את מערך העזר שחישבנו בתהליך העיבוד שישמש אותנו לצורך חישוב הקפיצה כמו שמתואר על מנת שיהייה ברור למשתמש.

חישוב Fail Value :

לחישוב Fail Value נצטרך מערך שנקרא לו LPS שהגודל שלו יהיה שווה לגודל מחרוזת החיפוש LPS .

הערה: LPS במקום אפס שווה לאפס (לא כמו מה שמצוין בתמונה אנחנו מתחילים עם 0 ולא עם -1 )

אנו מחפשים LPS בתבניות משנה. באופן ברור יותר אנו מתמקדים במיתרי משנה של דפוסים שהם קידומת וסיומת. עבור כל תת-תבנית שמתחילה מאפס עד i כאשר i = 0 עד (m-1) LPS במקום m מאחסן אורך של הקידומת המתאימה המקסימלית שהיא גם סיומת של התבנית LPS .

למשל במקרה השני :

***מחרוזת החיפוש היא* :**

|  |  |  |
| --- | --- | --- |
| A | B | E |

אז המערך LPS :

1. המערך יהיה בגודל של מחרוזת החיפוש כלומר הוא שווה לשלוש
2. LPS[0] שווה לאפס
3. ניקח עכשיו את התת מחרוזת AB בדוק אם יש תת מחרוזת שהיא גם סיומת וגם קידומת מכיוון שיש רק שתי תווים שונים LPS[1] יהיה גם 0
4. ניקח עכשיו את התת מחרוזת ABE בדוק אם יש תת מחרוזת שהיא גם סיומת וגם קידומת מכיוון שיש רק שתי תווים שונים LPS[2] יהיה גם 0
5. אם היה במקום E את A אז היינו מקבלים LPS[2]=1 כי היא שווה לתו הראשון במחרוזת החיפוש

***LPS:***

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |

**אלגוריתם Boyer-Moore :**

האלגוריתם מבצע עיבוד מקדים למחרוזת החיפוש (התבנית), אך לא לטקסט שבו יבוצע החיפוש. תם בויאר-מור משתמש במידע הנאסף בשלב העיבוד המקדים על מנת לדלג על חלקים בטקסט, והודות לכך הוא משיג ביצועים טובים ביחס לאלגוריתמי חיפוש אחרים.

באופן כללי, האלגוריתם רץ מהר יותר ככל שמחרוזת החיפוש ארוכה יותר. התכונות החשובות של האלגוריתם הן שהוא מחפש מסוף מחרוזת החיפוש ולא תחילתה, והוא מדלג בטקסט על מספר תווים במקום לחפש תו-תו.

**עיבוד מקדים :**

לצורך אלגוריתם זה נצטרך, לבנות טבלת אי התאמה שתשמש אותנו לחשב את מרחק הקפיצה בכל פעם שנקבל אי שוויון בין שני תווים.

הטבלה מכילה מופע אחד בלבד של כל תוו שנמצא במחרוזת החיפוש ונותנת לו ערך לפי הנוסחה הבאה:

Max(1,length\_of\_pattern-index\_of\_current\_char-1)

ובסוף הטבלה מוסיפים תו למשל \* שערכו שווה לאורך מחרוזת החיפוש.

**איטראציה באלגוריתם זה:**

לצורך הדגמה נניח שאנחנו באיטראציה הראשונה, אז בודקים את התו האחרון בטקסט הנבחר (מיקום גודל המחרוזת פחות 1) , אם התו האחרון במחרוזת החיפוש שווה לתו האחרון במחרוזת במקום גודל המחרוזת פחות 1 אז ממשיכים לבדוק את התו לפני האחרון בשתי המחרוזות וכך הלאה עד ש: או שמוצאים כל התווים של המחרוזת (כל התווים שווים בשתי המחרוזות) או שמוצאים אי שוויון.  
במקרה הראשון: תת צעד: הצלחה

אם איתרנו מופע למחרוזת ועדיין לא הגענו לסוף המחרוזת שהתקבלה בקלט, אז נמשיך לחפש מופעים ועוברים לתו הבא ע"י קפיצה לפי אורך המחרוזת לחיפוש.

במקרה השני: נמצא תווים שונים, אז האלגוריתם מחשב את הקפיצה למיקום התו הבא שצריך ממנו להתחיל את הבדיקה שוב.

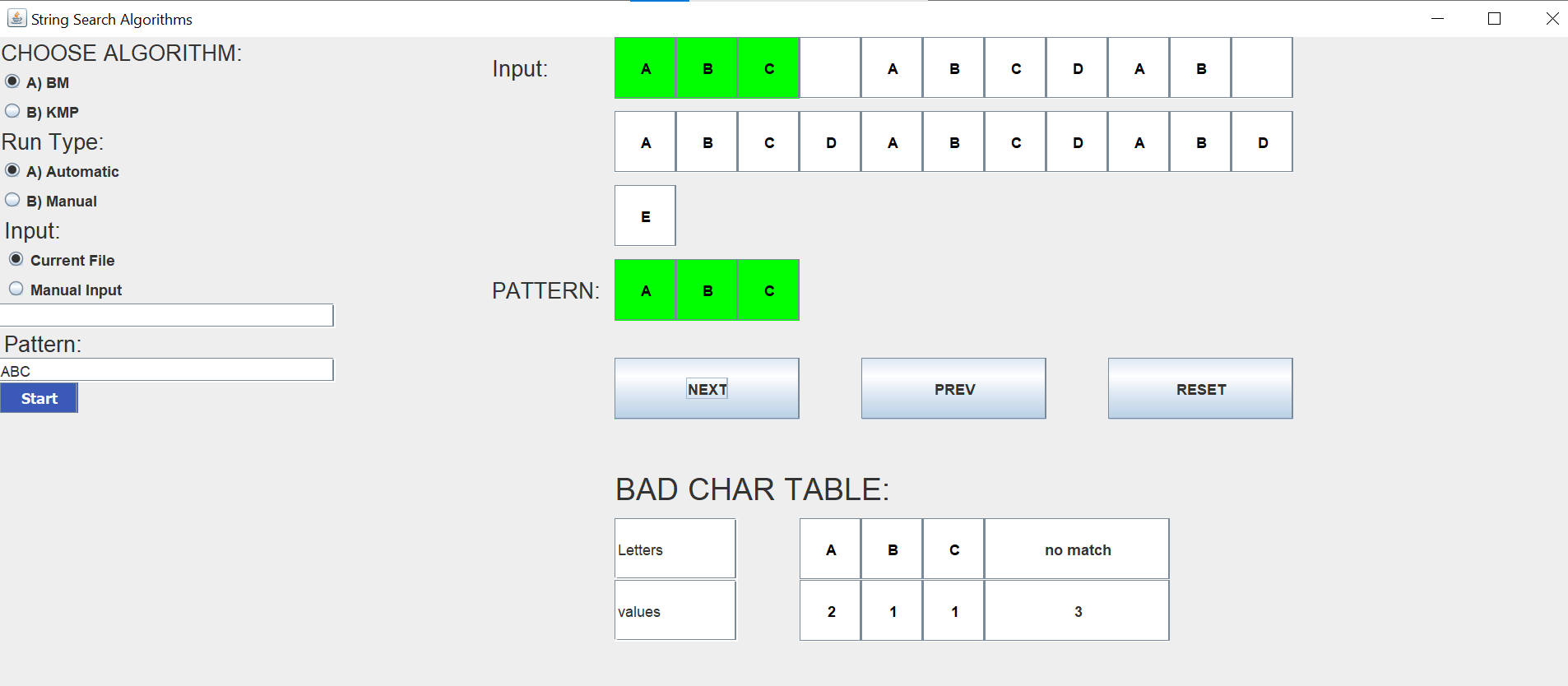
תת צעד: כישלון  
במקרה של אי התאמה כלומר קיבלנו שני תווים שונים, אם התו שייצר את האי שווין לא קיים בכלל במחרוזת החיפוש אז קופצים לפי הערך של \* צעדים שנמצא בטבלת האי התאמה.

אם התו נמצא במחרוזת החיפוש אז מוצאים את הערך שלו מטבלת האי התאמה וקופצים לפי ערך זה.

תהליך זה מבוצע עד שנאתר את המחרוזת הרצויה או נגיע לסוף הקלט.

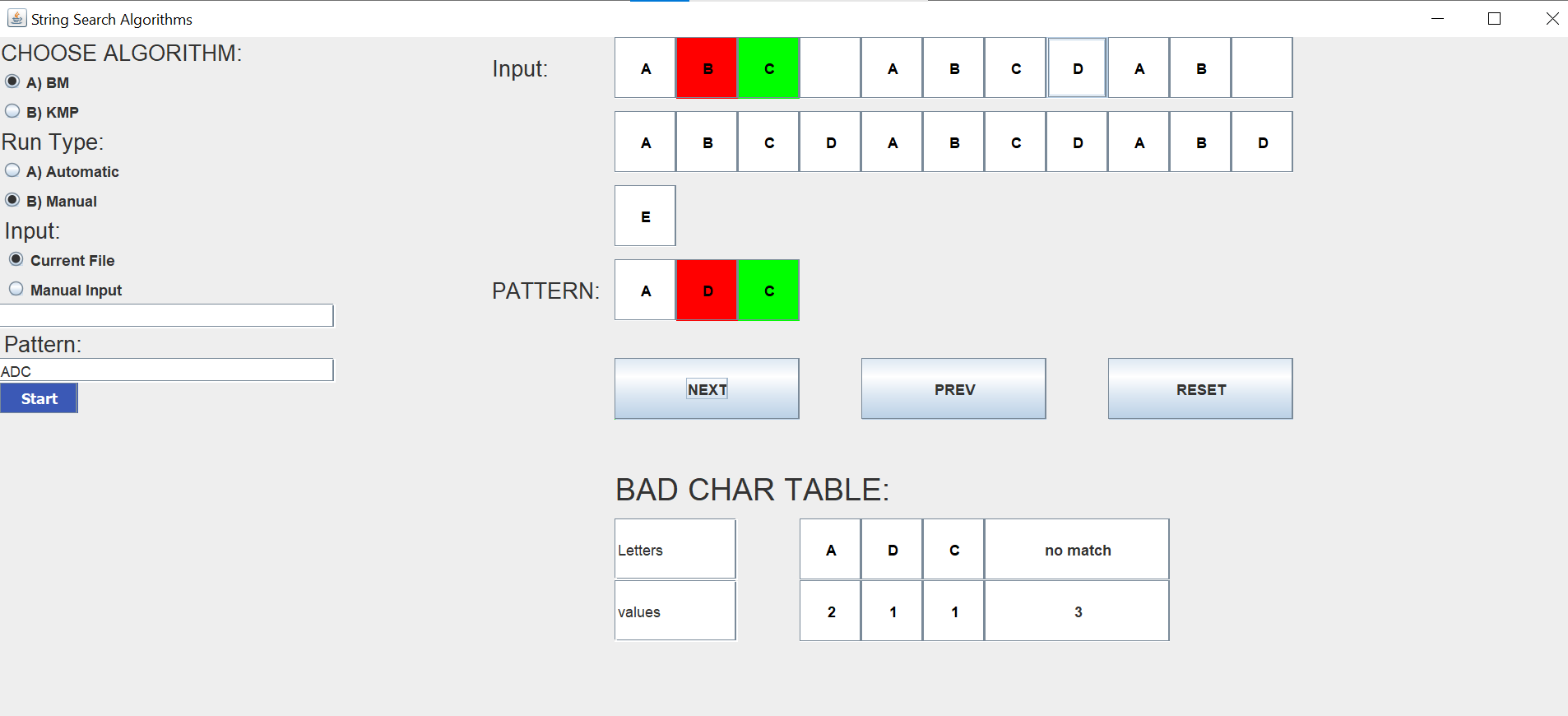
**ויזואליזציה:**

**מקרה ראשון:**



המחרוזת נמצאת, כל תו נמצא נצבע בירוק, בנוסף הודעת הצלחה מודפסת.

**מקרה שני:**



נמצא תו שונה אז נצבע באדום, ומחושב גודל הקפיצה (התו הבא שממנו נתחיל לבדוק).

על סימון התו שבודקים בכל שלב נצבע אותו בצע מסויים למשל כחול ואז צבע ירוק אם משווים או אדום במקרה של אי שוויון.

בויזואליזציה נוסיף את ה BAD MACH TABLE ואת תהליך העיבוד המקדים שחישבנו בשלב העיבוד על מנת לתאר איך מחושב אורך הקפיצה שיהייה ברור למשתמש.

חישוב BAD MACH TABLE:

נעשה שימוש במערך עזר נקרא לו Values ניקח את האותיות של מחרוזת החיפוש נבדוק תו תו והערך של Values נחשב אותו לפי הנוסחה :

Max(1,length\_of\_pattern-index\_of\_current\_char-1)

אם תו מופיע יותר מפעם אחת אז צריך לעדכן אותו ונוסיף את תו \* שיהיה שווה לאורך מחרוזת החיפוש

למשל עבור המקרה השני :

מחרוזת החיפוש הייתה :

|  |  |  |
| --- | --- | --- |
| C | D | A |

המערך Values :

שיטת החישוב :

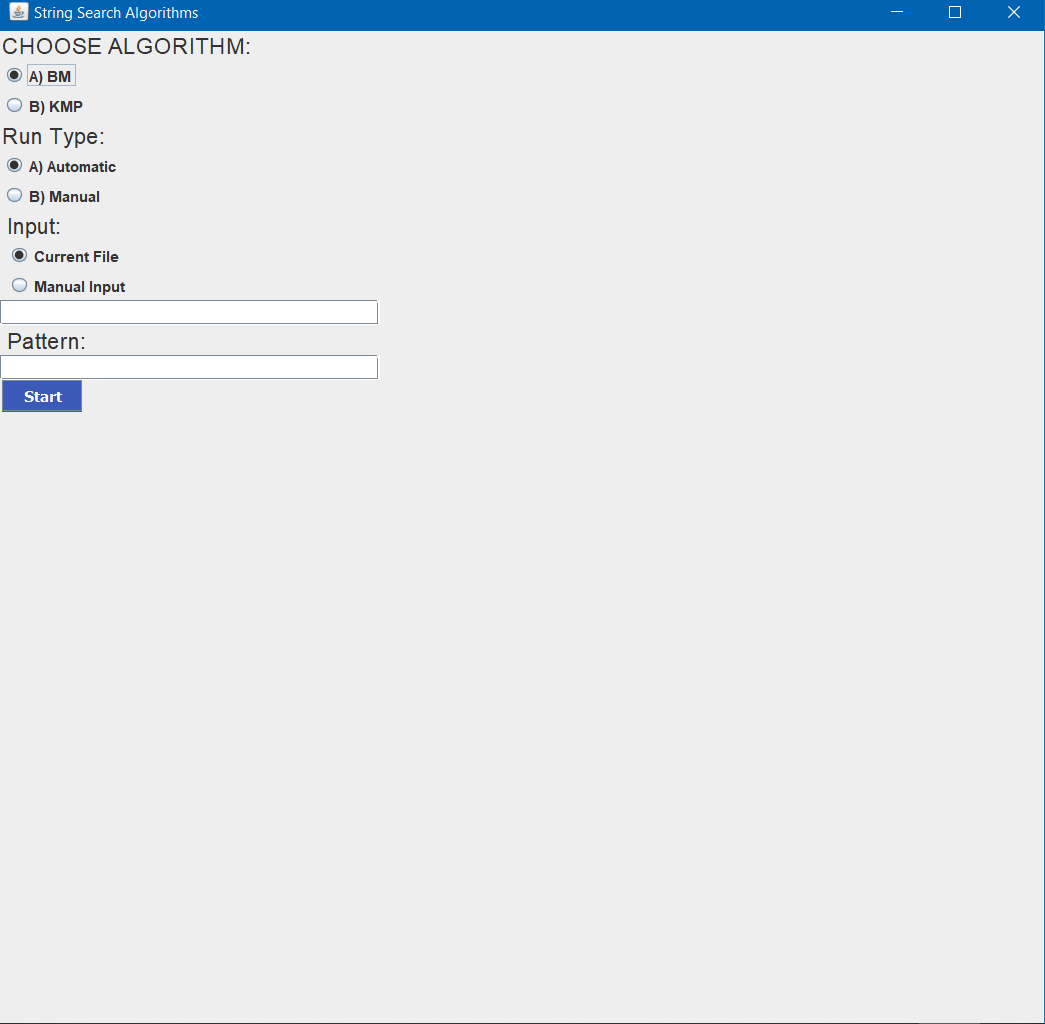
1. התו הראשון שמחשבים אותו הוא A צריך לבחור המקסימום בין 1 ו- (3-0-1=2) שזה שווה לשתיים
2. התו השני שמחשבים אותו הוא D צריך לבחור המקסימום בין 1 ו- (3-1-1=1) שזה שווה לאחד
3. התו השלישי שמחשבים אותו הוא E צריך לבחור המקסימום בין 1 ו- (3-2-1=0) שזה שווה לאחד
4. התו האחרון הוא כוכבית שהוא שווה לשלוש שזה אורך מחרוזת החיפוש נצטרך אותו כאשר אין תווים שלא זהים לתווים הנמצאים

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| \* | C | D | A | Letters |
| 3 | 1 | 1 | 2 | Values |

**פרק 4 תוצאות**

**Step by step results:**

STEP 0: (running the program)

****

When starting the program will display the window, the user has to choose the options he want to,  
the default values is BM Algorithm, Automatic run type with input from input file that stored in the system.

Selected Algorithm: BM (default option)

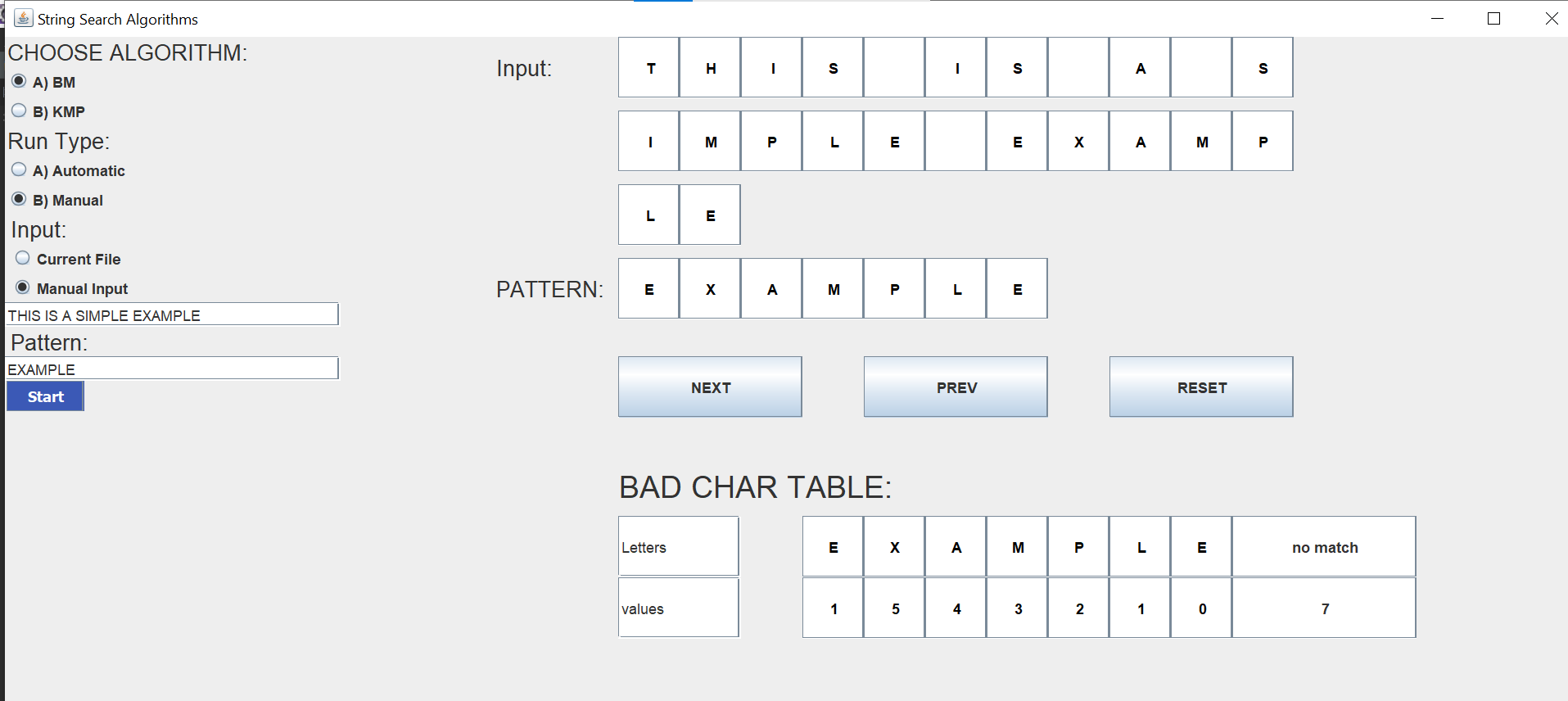
Run Type: Manual

Input: Manual Input

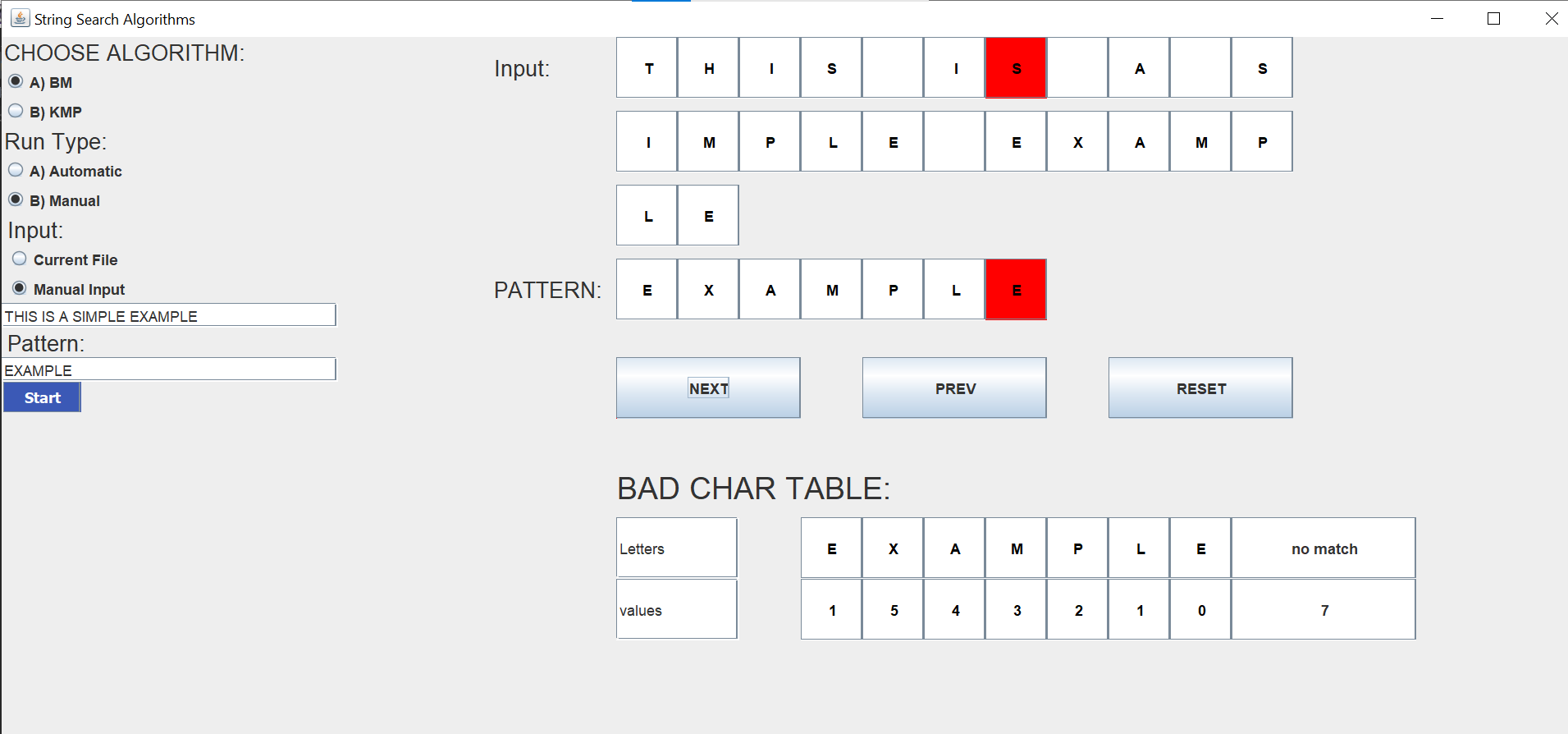
TEXT : THIS IS A SIMPLE EXAMPLE

PATTERN : EXAMPLE

STEP 1:

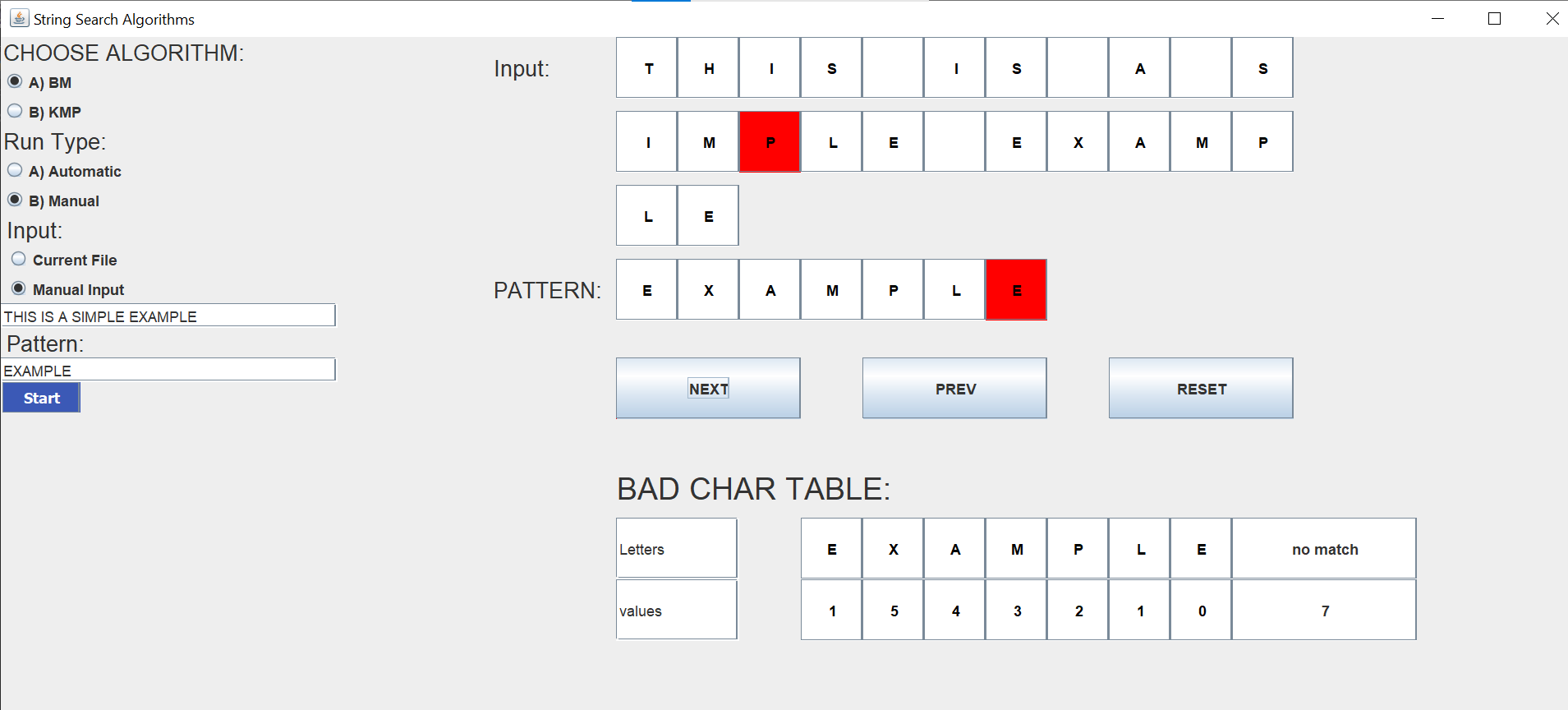


Step 2 :

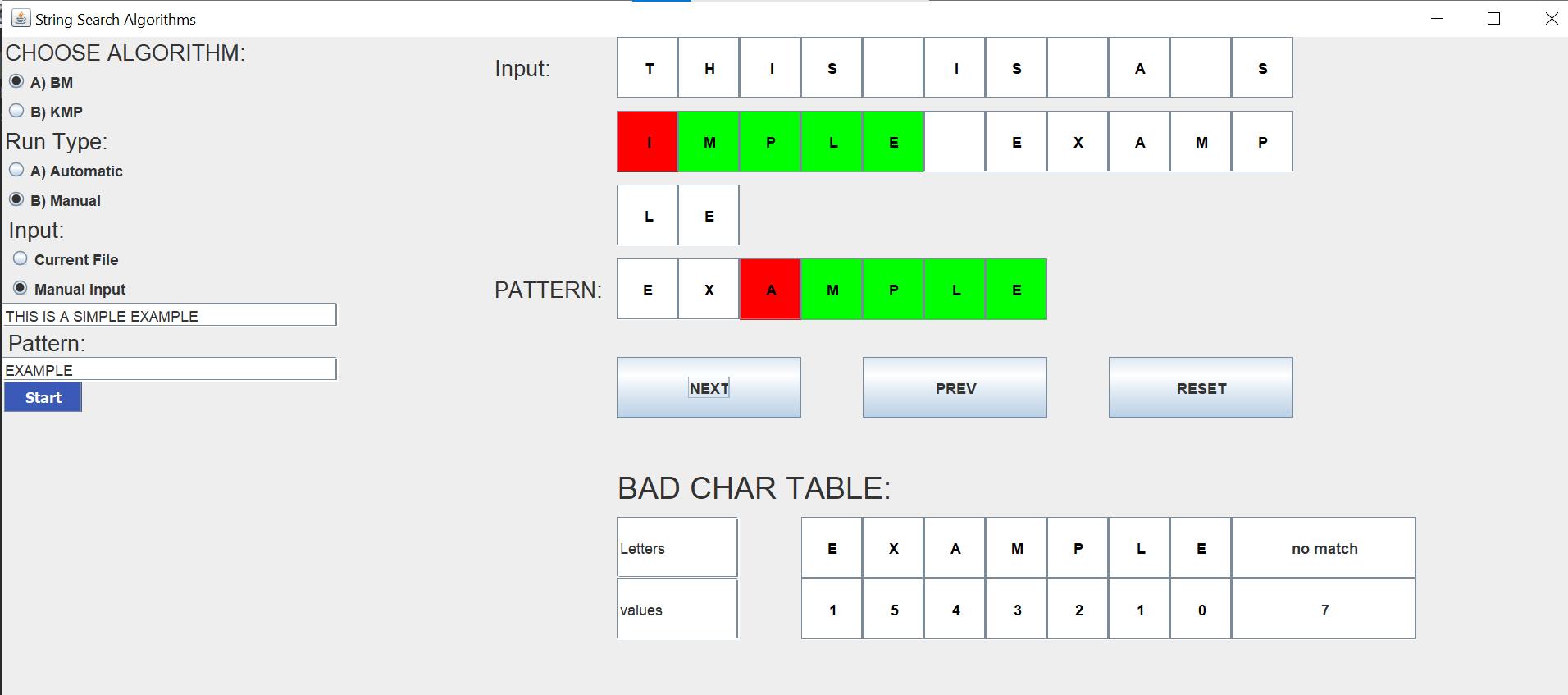


Checking the first letter and we got no matching

Step3:

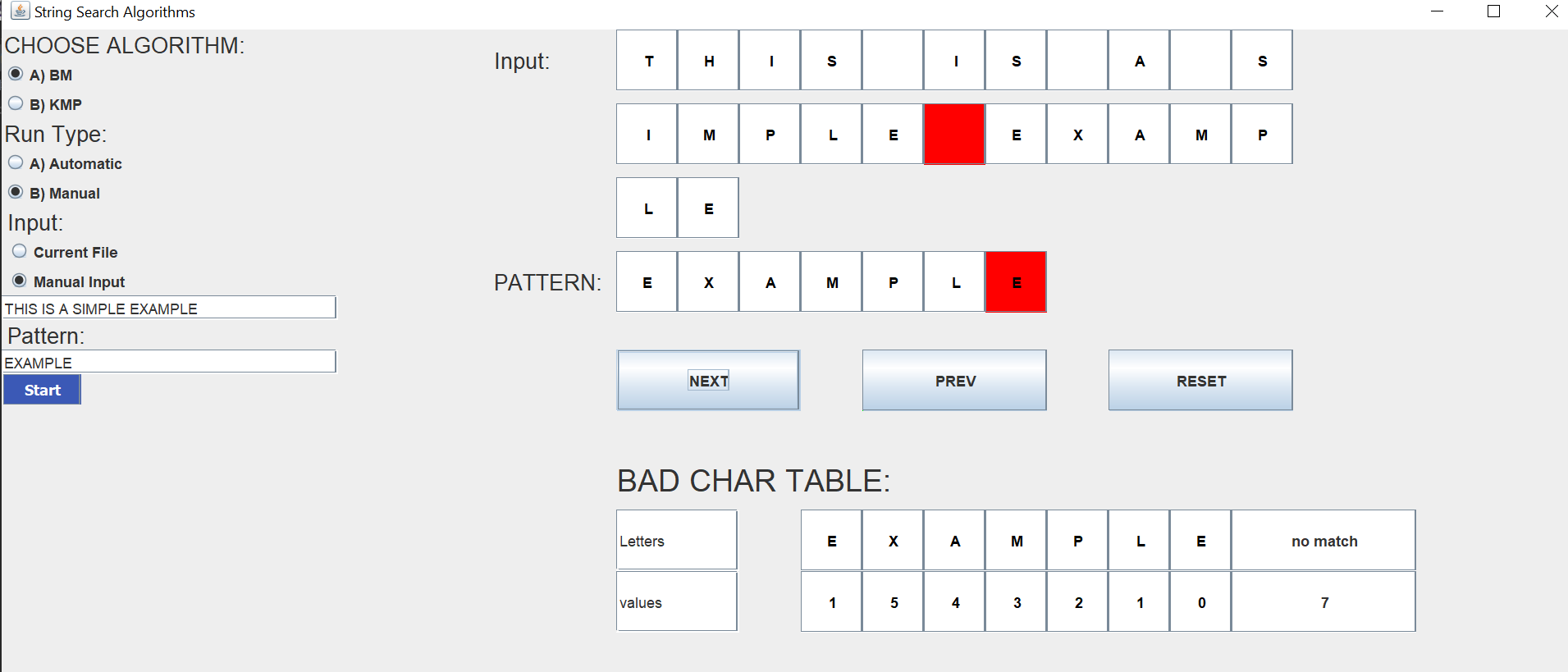


From the last step we checked s with e and according to BAD CHAR TABLE we jump 7 and check the second letter and got no matching

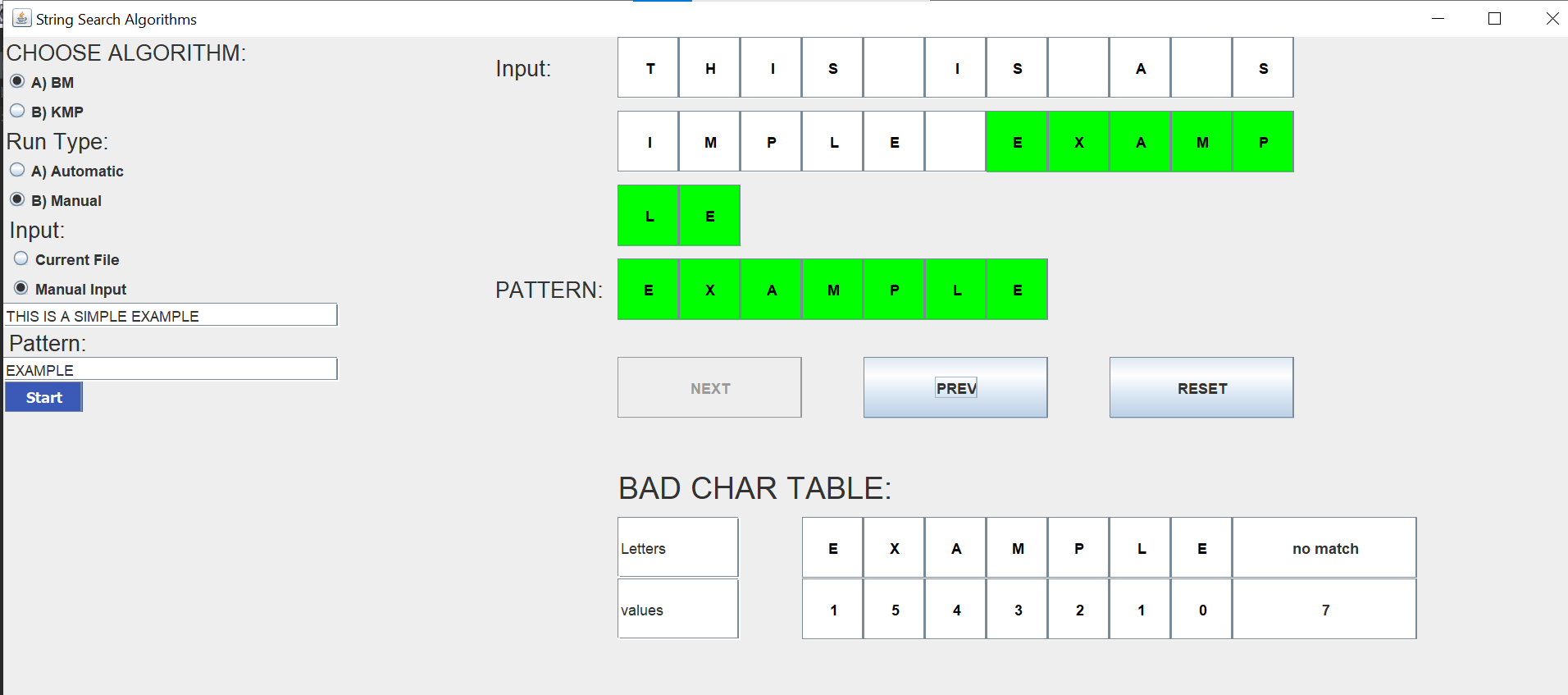
Step 4

From the last step we checked p with e and according to BAD CHAR TABLE we jump 2 and check the second letter and got matching so we check another letter till we got matching or no matching in this step we got no matching

Step 5 :



Step6:



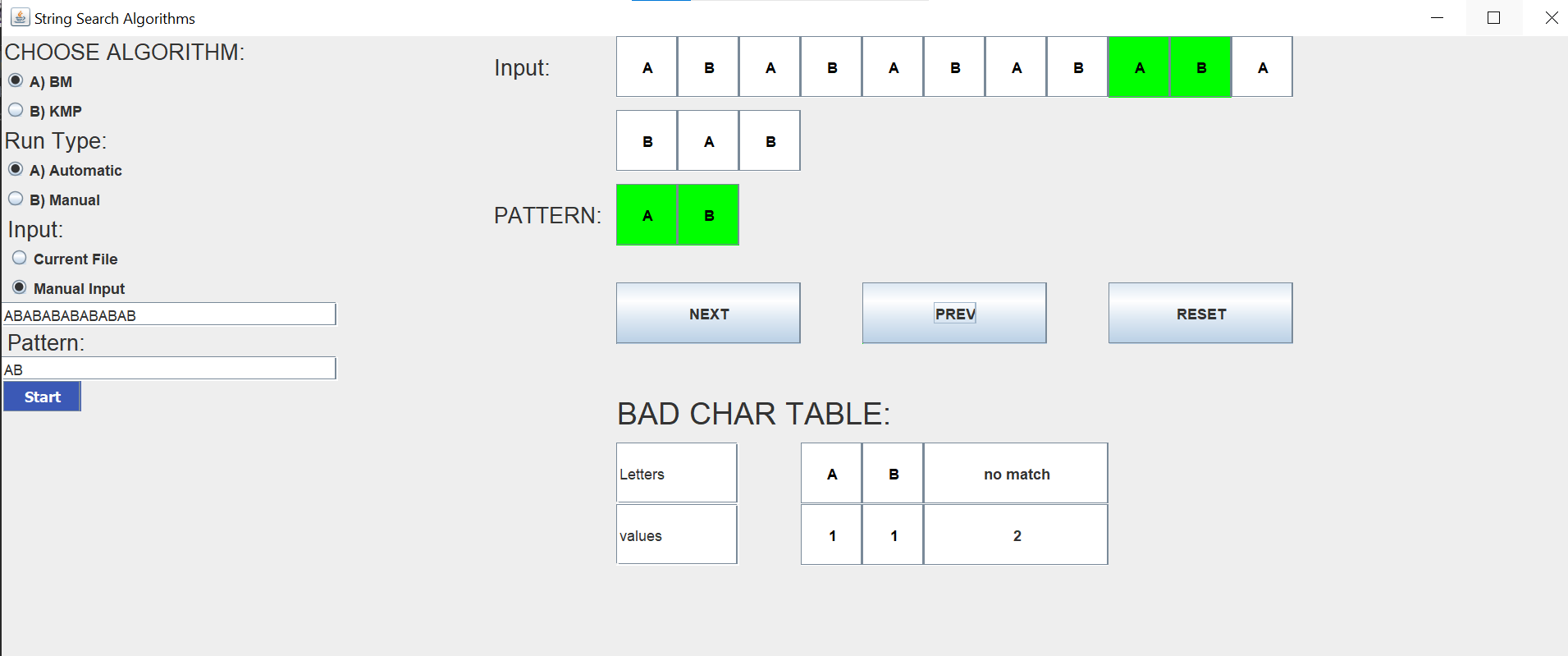
And we found the pattern .

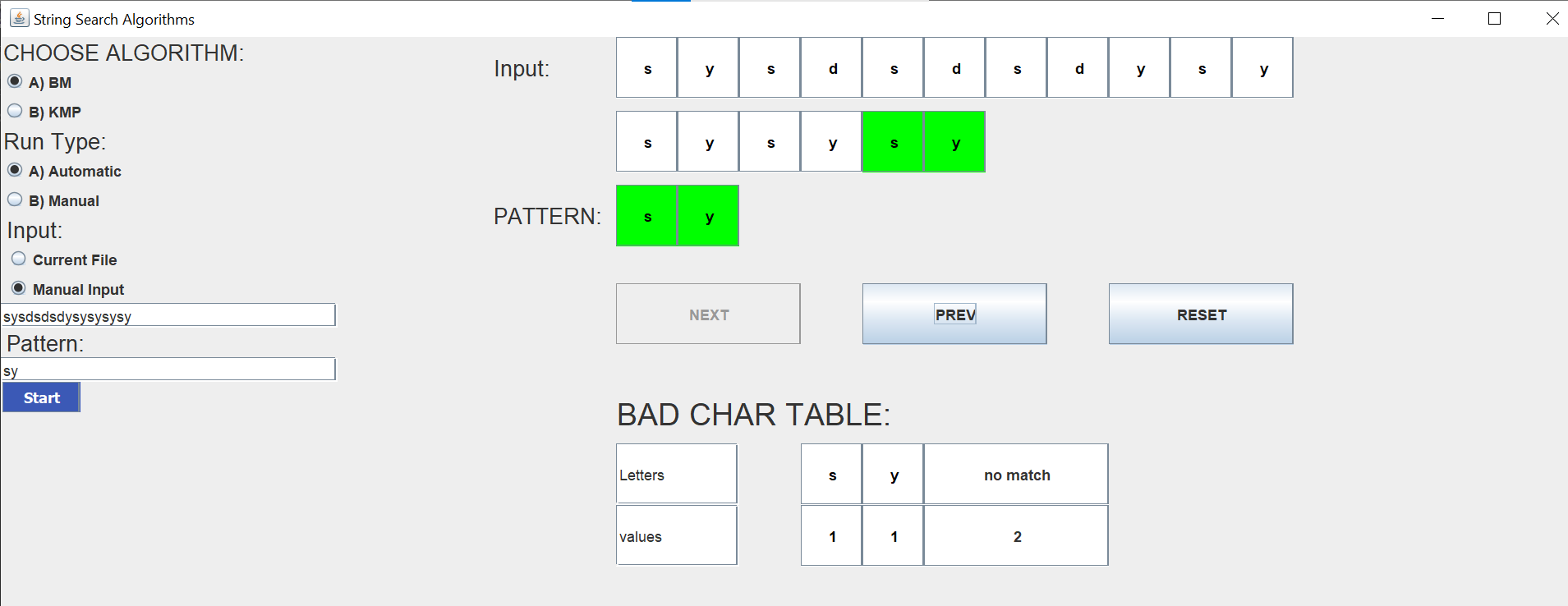
Step of reset button :

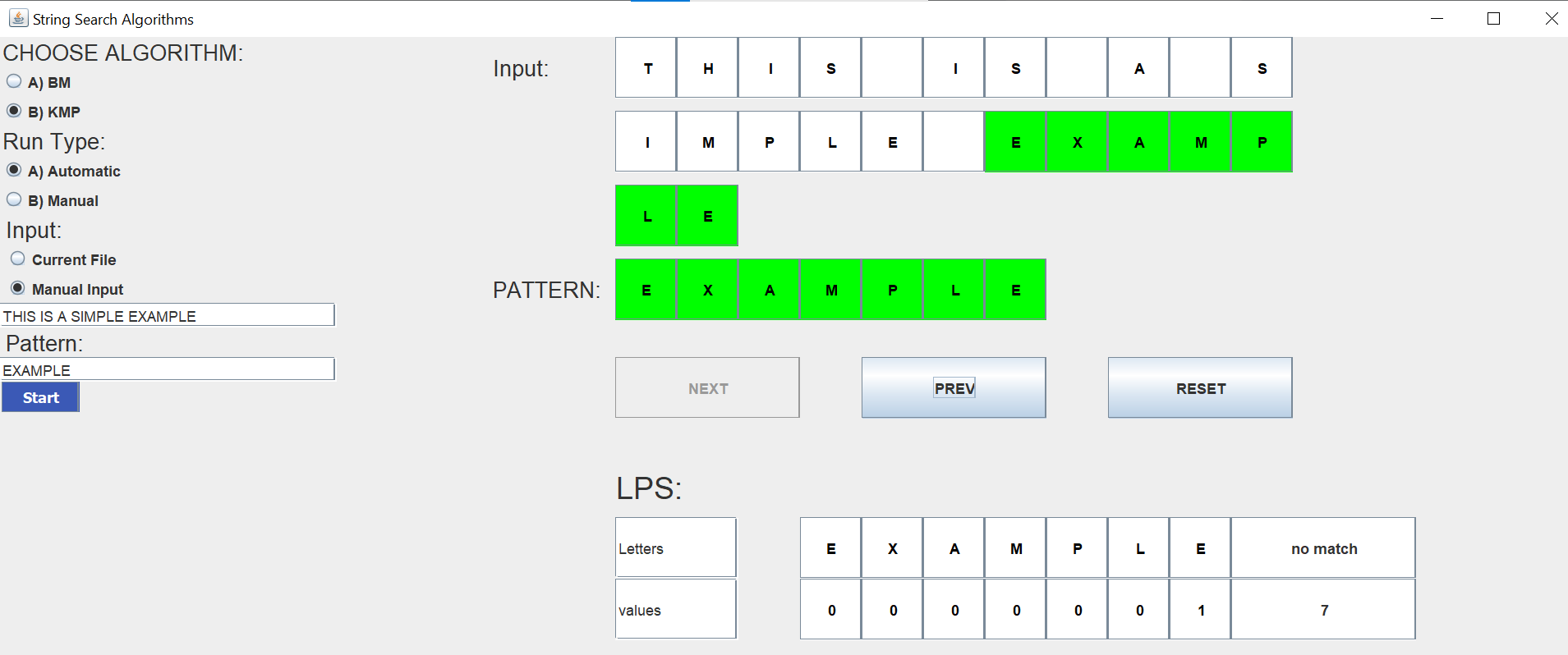
Graphical user interface, application

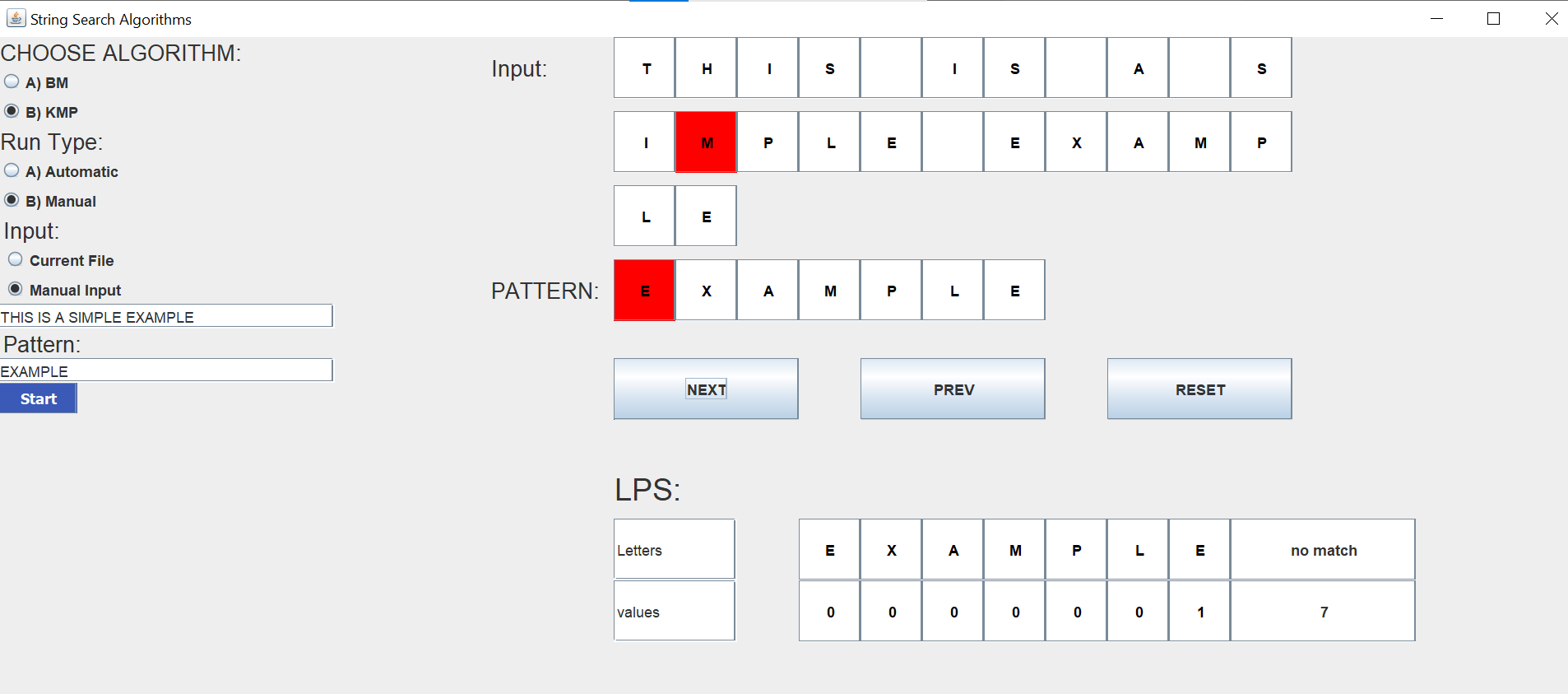
Description automatically generated

More results for another examples









**פרק סיכום ומסקנות**

**ניתוח תוצאות**

בויר מור הוא שילוב של שתי הגישות הבאות.

1) Bad Character Heuristic   
2) Good Suffix Heuristic

ניתן להשתמש בשתי ההוריסטיקות לעיל גם באופן עצמאי לחיפוש מחרוזת בטקסט אם נסתכל על האלגוריתם הנאיבי, הוא מחליק את התבנית על הטקסט בזה אחר זה. אלגוריתם KMP מבצע עיבוד מוקדם על פני התבנית כך שניתן יהיה לשנות את התבנית ביותר מאחד. האלגוריתם של בויר מור עושה עיבוד מוקדם מאותה סיבה. הוא מעבד את המחרוזת ויוצר מערכים שונים לכל אחד משני ההוריסטיקות. בכל שלב, הוא מחליק את התבנית לפי מקסימום השקופיות המוצעות על ידי כל אחת משתי היוריסטיקות. אז הוא משתמש בקיזוז הגדול ביותר שהציעו שני היוריסטיקות בכל שלב.

שלא כמו האלגוריתמים הקודמים לחיפוש תבניות, האלגוריתם של בויר מור מתחיל להתאים מהדמות האחרונה של התבנית.

**נספחים**

– הוראות להפעלת והרצה המערכת:

נבחר במחלקת MAINGUI ונריץ את התוכנה נבחר את האפשרויות שנרצה למשל הרצה אוטומטית או ידנית

יש שתי אפשריות לבחירת קלט :קלט ידני או קלט קיים מתוך קובץ

ונבחר האלגוריתם הרצוי

## הוכחת זמן ריצה

זמן הריצה הכולל של האלגוריתם הואO(m+n) {\displaystyle \mathrm {O} (m+n)}, כאשר הזמן הדרוש לבניית טבלה של התבנית הוא {\displaystyle \mathrm {O} (m)}O(m) וזמן החיפוש הדרוש הואO(n) {\displaystyle \mathrm {O} (n)}. נוכיח:

### אלגוריתם

כאשר אנחנו מבצעים את החיפוש עצמו, בכל איטרציה של כל אחת מהלולאות יש לפחות קידום אחד של i, כלומר לא קיימת איטרציה בה i לא קודם. בנוסף, כל איטרציה של הלולאות מפעילות {\displaystyle \mathrm {O} (1)}O(1) פעולות, ולכן בסך הכל יש לכל היותר {\displaystyle \mathrm {O} (n)}O(n) פעולות.

### טבלה

נסמן מונה C1 הסופר כמה פעמים עודפות האלגוריתם ביקר בתא בודד, ומונה C2 הסופר כמה פעמים האלגוריתם ביקר פעם בודדת. מתקיים {\displaystyle C2\leq n+1}C2<= n+1 מכיוון שיש n+1 תאים בטבלה. כפי שצוין קודם, מתקייםA[i] <= A[i-1] + 1 {\displaystyle A[i]\leq A[i-1]+1}, וגם {\displaystyle A[i]=A[i-1]+1}  A[i] = A[i-1] + 1אם ורק אם קיימת איטרציה אחת בדיוק בנוסף, מתקיים {\displaystyle A[i]<i}A[i] < i לפי הגדרת הרישא. לכן, ניתן לכתוב {\displaystyle A[i]\leq i-1} A[i] <= iכלומר במקרה הגרוע יש לולאה יורדת. נחבר את התנאים - על מנת שיתקיים {\displaystyle A[i]>a} A[i] > aצריך שיתקיים {\displaystyle C2>a}C2>a (לפי התנאי הראשון), ועל מנת שנוכל לקדם את C1, צריך שיתקיים {\displaystyle A[i]>0} A[i] >0. בסך הכל, על מנת לקדם את C1 יש לקדם את C2, ולכן מתקיים {\displaystyle C1\leq C2\leq n+1}C1<=C2<=n+1, ומספר האיטרציות הכולל קטן מ{\displaystyle 2n+2}2n+2, או O(n){\displaystyle \mathrm {O} (n)}() כנדרש.

**רשימת מקורות**

<http://whocouldthat.be/visualizing-string-matching/>

<https://www.educba.com/kmp-algorithm/>

<https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm>

<https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/>

<https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/>

<https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm>