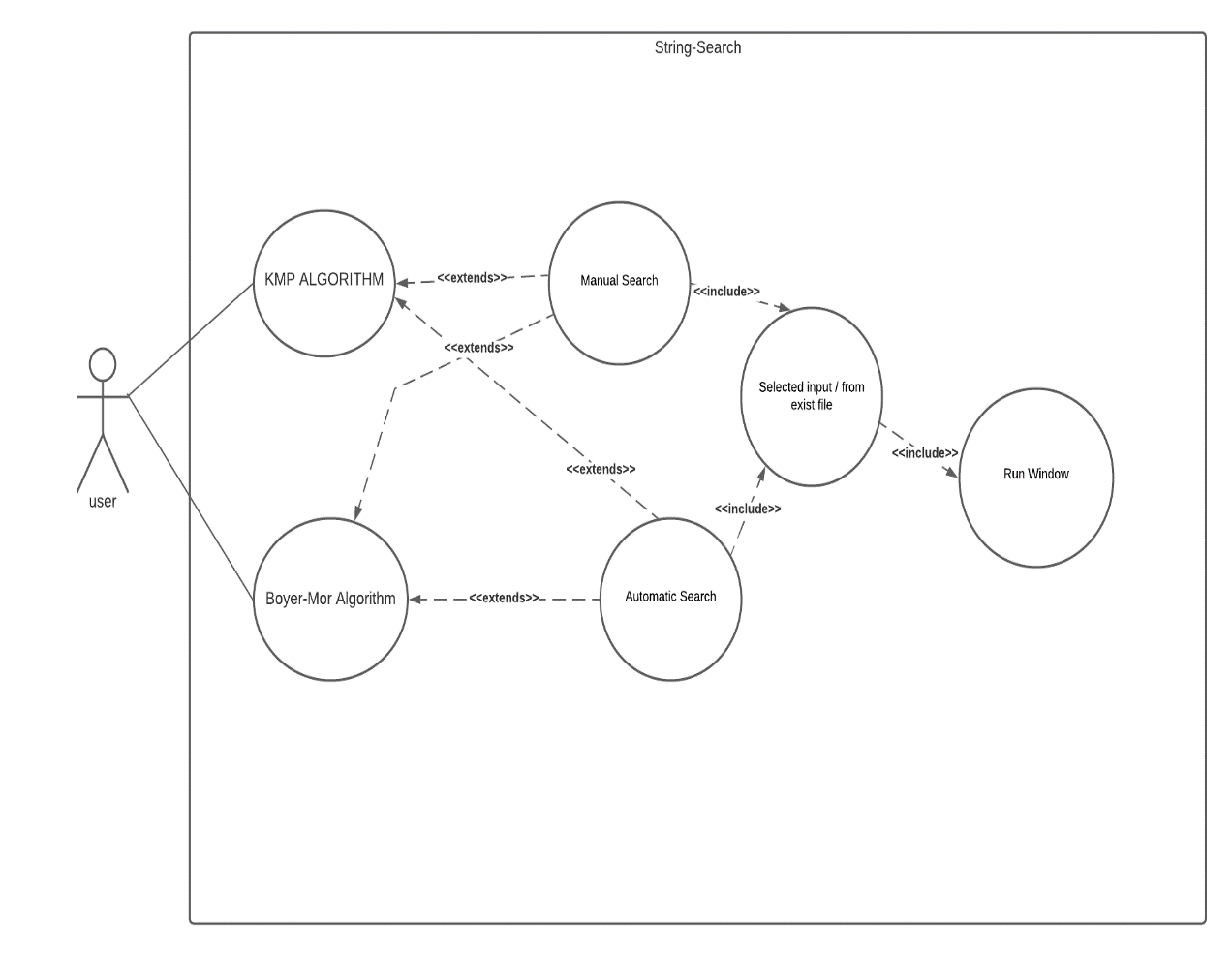
מסמך דרישות מפורט**:**

USE CASE DIAGRAM



**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.

**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.

**איטרציה באלגוריתם זה:**

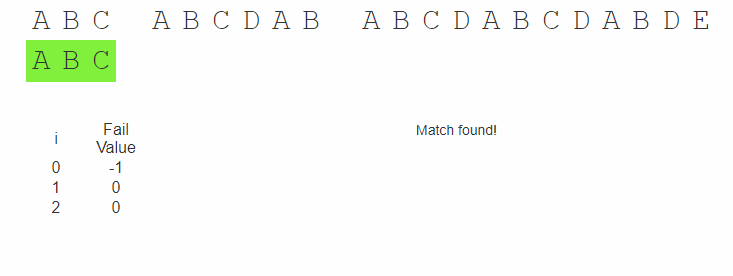
לצורך הדגמה נניח שאנחנו באיטרציה הראשונה, אז בודקים את התו הראשון בטקסט הנבחר (מיקום 0), אם התו הראשון במחרוזת החיפוש שווה לתו הראשון במחרוזת הנבחרה אז ממשיכים לבדוק את התו השני בשתי המחרוזות וכך הלאה עד ש: או שמוצאים כל התווים של המחרוזת (כל התווים שווים בשתי המחרוזות) או שמוצאים אי שוויון.  
במקרה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

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

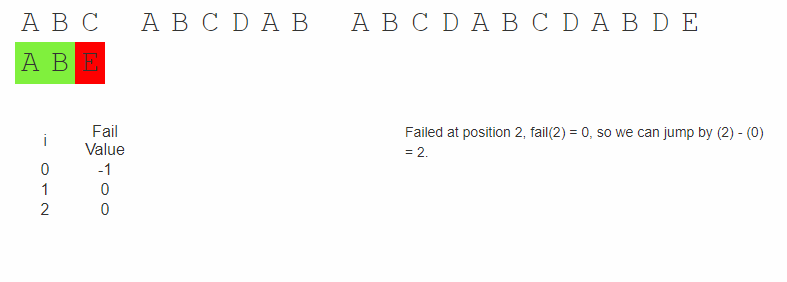
תהליך זה מבוצע עד שנאתר את המחרוזת הרצויה או נגיע לסוף הקלט.

**ויזואליזציה:**

**מקרה ראשון:**

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

**מקרה שני:**

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

הערה: תמונות להמחשה בלבד, ייתכן כמה שינויים על העיצוב.

**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.

**איטרציה באלגוריתם זה:**

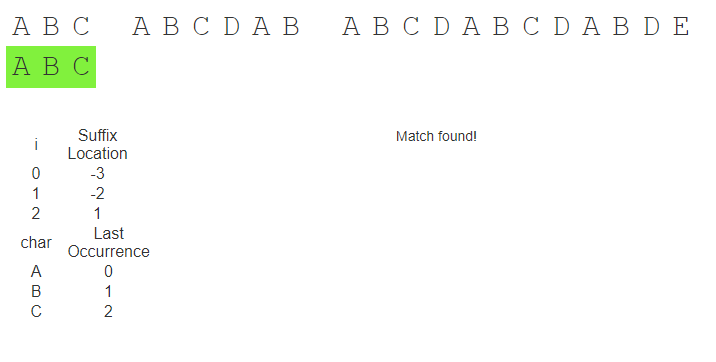
לצורך הדגמה נניח שאנחנו באיטרציה הראשונה, אז בודקים את התו האחרון בטקסט הנבחר (מיקום גודל המחרוזת פחות 1) , אם התו האחרון במחרוזת החיפוש שווה לתו האחרון במחרוזת במקום גודל המחרוזת פחות 1 אז ממשיכים לבדוק את התו לפני האחרון בשתי המחרוזות וכך הלאה עד ש: או שמוצאים כל התווים של המחרוזת (כל התווים שווים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

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.

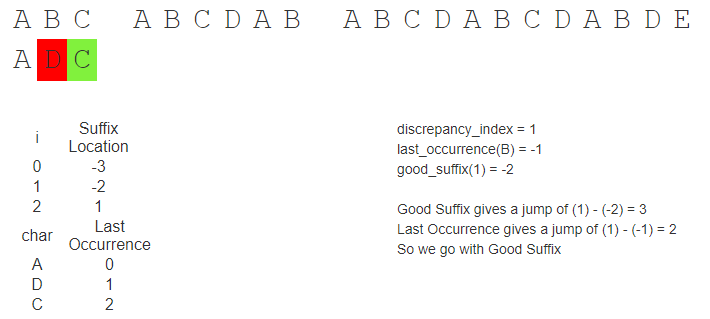
**ויזואליזציה:**

**מקרה ראשון:**

****

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

**מקרה שני:**

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

הערה: תמונות להמחשה בלבד, ייתכן כמה שינויים על העיצוב.

**Input Files:**

**file 1 - kmp.txt:**

In [computer science](https://en.wikipedia.org/wiki/Computer_science), the Knuth–Morris–Pratt [string-searching algorithm](https://en.wikipedia.org/wiki/String-searching_algorithm) (or KMP algorithm) searches for occurrences of a "word" W within a main "text string" S by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters.

The [algorithm](https://en.wikipedia.org/wiki/Algorithm) was conceived by [James H. Morris](https://en.wikipedia.org/wiki/James_H._Morris) and independently discovered by [Donald Knuth](https://en.wikipedia.org/wiki/Donald_Knuth) "a few weeks later" from [automata theory](https://en.wikipedia.org/wiki/Automata_theory). Morris and [Vaughan Pratt](https://en.wikipedia.org/wiki/Vaughan_Pratt) published a technical report in 1970. The three also published the algorithm jointly in 1977. Independently, in 1969, [Matiyasevich](https://en.wikipedia.org/wiki/Yuri_Matiyasevich) discovered a similar algorithm, coded by a two-dimensional Turing machine, while studying a string-pattern-matching recognition problem over a binary alphabet. This was the first linear-time algorithm for string matching.

**file 2 - bm.txt:**

In [computer science](https://en.wikipedia.org/wiki/Computer_science), the Boyer–Moore string-search algorithm is an efficient [string-searching algorithm](https://en.wikipedia.org/wiki/String-searching_algorithm) that is the standard benchmark for practical string-search literature. It was developed by [Robert S. Boyer](https://en.wikipedia.org/wiki/Robert_S._Boyer) and [J Strother Moore](https://en.wikipedia.org/wiki/J_Strother_Moore) in 1977. The original paper contained static tables for computing the pattern shifts without an explanation of how to produce them. The algorithm for producing the tables was published in a follow-on paper; this paper contained errors which were later corrected by [Wojciech Rytter](https://en.wikipedia.org/wiki/Wojciech_Rytter) in 1980. The [algorithm](https://en.wikipedia.org/wiki/Algorithm) [preprocesses](https://en.wikipedia.org/wiki/Preprocessor) the [string](https://en.wikipedia.org/wiki/String_(computer_science)) being searched for (the pattern), but not the string being searched in (the text). It is thus well-suited for applications in which the pattern is much shorter than the text or where it persists across multiple searches. The Boyer–Moore algorithm uses information gathered during the preprocess step to skip sections of the text, resulting in a lower constant factor than many other string search algorithms. In general, the algorithm runs faster as the pattern length increases. The key features of the algorithm are to match on the tail of the pattern rather than the head, and to skip along the text in jumps of multiple characters rather than searching every single character in the text.

יתכן שינויים בתוכן הקבצים במידת הצורך.