METHODS FOR SOLVING APPLIED PROBLEMS IN PROBABILITY THEORY BY MATHEMATICAL MODELING

Abstract. *The concept of applied problem and mathematical model is considered in the article. The method of mathematical modeling as the main method of solving applied problems on the example of problems of probabilistic and statistical content is analyzed. Examples of using this method to solve problems are considered.*

Keywords*: applied problem, mathematical modeling, model, probability, statistics, combinatorics.*

Віталіна Бияковська

ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ ПОШУКУ ПІДРЯДКА В РЯДКУ

Анотація*:* *Стаття присвячена розгляду питання пошуку алгоритмів підрядка в рядку реалізовуючи в програмному середовищі . Алгоритм пошуку можна реалізовувати за допомогою алгоритмів послідовного(прямого) пошуку, алгоритм Рабіна – Карпа, Кнута-Моріса-Пратта, а також Бойера–Мура. Перевірка на швидкість і час виконання алгоритмів, а також програмна реалізація алгоритмів з використанням різних функцій.*

Ключові слова*:* *алгоритм послідовного(прямого) пошуку, алгоритм Рабіна-Карпа, алгоритм Кнута-Моріса-Пратта, алгоритм Бойера–Мура, хеш-функція, префікс-функція.*

Актуальність проблеми. Алгоритми пошуку підрядка в рядку є актуальним на сьогоднішній день тому, що ми неодноразово працювати з текстом, наприклад пошук знаходження інформації в інтернеті, а також з текстовими редакторами MS Word, а для того, щоб знайти схожі слова у тексті використовуємо функцію, яка є значно ефективнішою в редагуванні та виправленні документів та пошуку необхідної інформації. Алгоритми пошуку дуже важливі, хоча вони різні, залежно від типу оброблених даних та їх реалізації в програмах, ми враховуємо час пошуку, кількість використаних операцій сортування масиву даних, а потім виконуємо пошук з використанням одного із методів пошуку підрядка. Ми розглядаємо різні алгоритми при виконанні пошуку на основі конкретних завдань та їх вирішенні.

Метою цієї статті є дослідження різних алгоритмів пошуку підрядка в рядку реалізовуючи їх в програмних середовищах, а також розв’язання проблем пов’язаних з пошуком.

Об’єктом дослідження в цій статті є алгоритми опрацювання рядкових даних, а **предметом дослідження** – алгоритми пошуку підрядків у рядках.

Для досягнення сформульованої мети необхідно виконати наступні **завдання**:

- розглянути та проаналізувати алгоритми пошуку підрядка в рядку;
- розглянути програмні реалізації алгоритмів пошуку підрядка в рядку;
- проаналізувати час виконання алгоритмів.

Аналіз останніх досліджень і публікацій. Проблемою програмної реалізації алгоритмів пошуку підрядка в рядку займалися: Коул, Апостоліко, Данкарло, Крошемур, Колуссі та інші, які розробили найбільш ефективні рішення в термінах кількості порівняння символів. Алгоритм Бойера-Мура у роботі Коула показав, що на неперіодичних шаблонах за повний прохід по рядку алгоритм зробить не більше трьох порівнянь.

Алгоритм знаходження пошуку підрядка в рядку реалізовується за допомогою наступних алгоритмів.

Алгоритм послідовного (прямого) пошуку полягає в по символьному порівнянні рядка X=x[1]..x[n] з підрядком Y=y[1]..y[m], де довжина рядка є функція **Length(X):=n** ,а довжина підрядка **Length(Y):=m**, причому $0<m\leq n$. На першому етапі реалізації

алгоритму відбувається порівняння і-го символу масиву X з першим символом масиву Y, якщо збігається, то порівнюється другий символ і так далі, які починаються з позицій 1,2,..m-n+1 у слові X.

Якщо відбувається збіг усіх символів, то під рядок знайдений. У випадку, якщо не відбудеться збігів усіх символів, тобто в рядку не знайдеться підрядок, то відбувається зсування підрядка на одну позицію вправо (i=i+1), і повторюється посимвольне порівняння, як і в попередньому етапі. Зсуви підрядка повторюються до тих пір, поки не виконається умова $i + M > N$ тобто відповідне слово не знайдене, або не відбудеться повний збір символів підрядка з рядком, тобто знайдеться відповідний під рядок.

Алгоритм прямого пошуку має такі недоліки:

1. Висока складність – O(N · M), у найкращому випадку – O((N – M + 1) · M);
2. Якщо не співпав символ з рядком то відбувається перегляд з першого символу зразка й тому може повторно розглядати символ X, які раніше проглядалися. Наприклад: знайдемо рядок good і при пошуку під рядка виявиться, що співпало тільки три символи goo, а четвертий не співпав алгоритм буде продовжувати порівнювати рядок, і не призведе до результату.
3. Дані про текст X, що отримані при перевірці даного зсуву Y, ніяк не використовується при перевірці наступних зсувів і знаходженні підрядка.

Для введених невеликих за обсягом рядків пошук працює швидше, а для багатомегабайтних файлів пошук підрядка займає багато часу. Складність даного алгоритму полягає в тому що, для виявлення збігу символів наприкінці рядка, потрібно зробити n*m порівнянь, тобто O(N*M).

Фрагмент алгоритму послідовного (прямого) пошуку з використанням функції DirectSearch наведено на рис. 1. [5, с. 77].

```
Function DirectSearch(X:string;Y:string; var Place:byte):boolean;
Var Res: boolean ;
i,n,m:integer ;
Begin
n:=length(X);
m:=length(Y);
Res:=FALSE;
i:=1;
while (i <= n-m + 1) And Not(Res) do
if Copy(X,i,m) = Y then
begin
Res:=TRUE;
Place:=i;
end
else i:=i+1;
DirectSearch:=Res;
end;
```

Рис. 1. Реалізація алгоритму послідовного (прямого) пошуку

Наступний алгоритм пошуку підрядка є алгоритм Рабіна-Карпа. Даний алгоритм був розроблений в 1987 році Майклом Рабіном і Річардом Карпом. Він заснований на простій ідеї, і представляє собою модифікацію лінійного алгоритму з використанням хешування. Ідея пошуку полягає в тому, що в слові X=x[1]..x[m], де довжина рядка є функція **Length(X):=n**, ми шукаємо підрядок Y=y[1]..y[n], де довжина підрядка є функція **Length(Y):=m**. Виріжемо віконечко розміром n і будемо рухати його по вхідному рядку при цьому будемо спостерігати чи, не збігається слово X в віконечку з заданим

підрядком. Записуємо деяку функцію, визначену на словах довжини n. У випадку якщо значення функції в рядку і на підрядку різні, то збігу немає. Тільки якщо значення однакові, потрібно перевіряти збіг символів по буквах. Для прискорення перевірки знаходження ідентичності підрядка з рядком використовують хеш-функцію.

Алгоритм використовує той факт, що якщо два рядки ідентичні, то і їх хеш-значення також однакові. Для реалізації хеш-функції потрібно порахувати хеш-значення даного підрядка, а потім знайти підрядок з таким же хеш-значенням.

Існують деякі проблеми пов'язані з хеш-значенням. Перша проблема полягає в тому, що існує багато різних рядків, але для того,щоб мати невеликі хеш-значення ми повинні мати деякі рядки, хеш-значення яких збігається. Це доводить те, що хеш-значення можуть збігатися,а рядки не збігатися. Вирішенням цієї проблеми є хеш-функція яка забезпечує те, що при досить складних вхідних даних це не буде відображатися так часто, і в процесі результатів середній час пошуку підрядка буде не великим.

Ще одна проблема полягає в наступному рядку h: = hash (X[i + 1..i + m]).

Якщо ми перерахуємо хеш-значення для підрядка X [(i + 1) .. (i + m)],нам знадобиться час O (m), і це відбувається у кожному циклі, алгоритм потребуватиме часу O (m · n). Відповідь на дане завдання полягає в тому, що змінна H вже містить хеш-значення для X [i .. (i + m - 1)]. У випадку якщо ми зможемо використовувати його для обчислення наступного хеш-значення за постійний час, тоді наша проблема буде вирішена. Ми зможемо це зробити за допомогою кільцевого хеша. Кільцевий хеш - це хеш-функція, яка використовується спеціально для цієї операції. Ми можемо використовувати цю формулу для підрахунку кожного наступного хешованого значення фіксованого часу: X [(i + 1) .. (i + m)] = X [i .. (i + m - 1)] - X [i] + X [i + m].

Дана функція працює, але в результаті вираз if X[i..(i + m – 1)] =Y буде виконуватися частіше, ніж кільцеві хеш-функції. Зауважимо те, що якщо у випадку поганої хеш-функції, така ж сама як стійка функція, if X[i..(i + m – 1)] = Y, ймовірність виконання функції буде n разів на кожну ітерацію циклу. На рис.1 зображене алгоритм Рабіна-Карпа. [5, с. 79].

```
 function RabinKarp (string X[1..n], string Y [1..m])
Begin
hsub:=hash(Y[1..m])
h:=hash(X[1..m])
for i from 1 to (n–m+1)
 if h==Y
 if X[i..(i+m–1)]=sub
 return i
 hs:=hash(s[(i+1)..(i + m)])
 return not found
 end;
```

Рис. 2. Реалізація алгоритму Рабіна-Карпа

Алгоритм Кнута-Морріса-Пратта був розроблений в 1977 році Дж. Моррісом, В. Праттом та Д. Кнутом. Реалізація алгоритму полягає в тому, що ми отримуємо при вході слово X = x [1] x [2] ... x [n] і переглядає алгоритм зліва направо буква за буквою, і заповнюючи при цьому масив натуральних чисел l [1] .. l [n], де l[i] = довжина слова l(x[1]...x [i]). У випадку якщо буде виявлена часткова збіжність підрядка Y=y[1] y[2]…y[m], де довжиною підрядка є функція **Length(Y),** з рядком то можливий зсув

підрядка на декілька позицій вправо і у наступних випадках не знадобиться повторно порівнювати символи які співпали.

Даний алгоритм використовує попередню обробку рядка пошуку, і створює її основі префікс-функції. Ідеясфункції полягає в знаходженні для кожного підрядка X [1 .. i] рядка X найбільшого підрядка X [1 .. j] (j <i), присутнього, як на початку, так і в кінці підрядка (як префікс і як суфікс). Якщо даний префікс рядка X з довжиною *i* довший одного символу, то він є префіксом підрядка з довжиною *i*-1. Слідуючи з функції ми перевіряємо префікс попереднього підрядка Y, у випадку якщо префікс не підходить то ми перевіряємо префікс його префікса.

Наприклад: для рядка abcoabc підрядком є abc. Значення префікс-функції полягає в тому, що ми відкидаємо невірний варіант.

Для обчислення префікса-функції виконуються наступні кроки. Нехай F(X, i)=k,де k-довжина префіксу рядка. Обчислимо префікс-функцію для i+1. Якщо наш рядок X[i+1]=X[k+1], то очевидно, що F(X,i+1)=k+1.Якщо умова не виконалася то підбираємо менші суфікси. Як ми бачим , що X[1…F(X,k), буде суфіксом рядка X[1…i], а для кожного j є (k, i) рядок X[1…j] суфіксом не буде. Отже, ми маємо алгоритм для обчислення префікс-функції:

1. Якщо X[i + 1] = X[k + 1] –> π(X, i + 1) = k + 1.
2. В іншому випадку при k = 0 –> π(X, i + 1) = 0.
3. Інакше – встановимо k: = π(X, k), GOTO 1.

Обчислюючи префікс-функцію ми знаходимо найбільший шуканий префікс. Автори алгоритму Д. Кнут, Д.Морріс і В. Пратт довели, що потрібний час виконання алгоритму складає O(*m* + *n*). Даний алгоритм є значно ефективнішим в реалізації ніж два попередні алгоритми. На рис. 3 зображено алгоритм Кнута-Морріса-Пратта з використанням префікс функцію [5, с. 85].

```
procedure TForm1.Button1Click(Sender: TObject);
Var F: array of Integer;
k, i, Result: integer;
 X, Y: string;
Begin
 SetLength(F, 1+Length(Y));
 F[1] := 0;
 k := 0;
 for i := 2 to Length(Y) do
 begin
  while (k > 0) and (Y[k+1] <> Y[i]) do
 k := F[k];
  if Y[k+1] = Y[i] then
 Inc(k);
 F[i] := k;
 end;
 k := 0;
 for i := 1 to Length(X) do
  begin  while (k > 0) and (Y[k+1] <> X[i]) do
   k := F[k];
  if Y[k+1] = X[i] Then
 Inc(k);
  if k = Length(Y) Then
  begin
   Result := i-length(Y)+1;
```

Рис. 3. Реалізація алгоритму Кнута-Морріса-Пратта з використанням префікс-функції

Алгоритм Бойєра-Мура був розроблений Р. Бойєром і Д. Муром у 1977 році і вважається найбільш швидким та ефективним у процесі пошуку підрядка в рядку, ніж попередні алгоритми. Алгоритм складається з наступних кроків. На першому кроці ми реалізовуємо таблиця зсувів для шуканого підрядка Y = y [1] y [2] ... y [m], де довжина підрядка є **Length(Y)**. На наступному кроці відбувається зсув початку рядка X = x [1] x [2] ... x [n], де довжина рядка це функція **Length(X),** й підрядка і починається перевірка включно з останнім символом під рядка. У випадку якщо останній символ рядка та відповідний йому при пошуку символу рядка не збігаються, то підрядок зміщується відносно рядка на одну величин, отриману з таблиці зміщення, і знову проводиться порівняння, починаючи з останнього символу підрядка. Якщо символи однакові, то ми проводимо порівняння передостаннього символу під рядка і т.д. Якщо всі символи в підрядку відповідають з накладеними символами рядка, це означає, що даний підрядок знайдено і пошук завершено. Якщо якийсь символ під рядком не відповідає відповідному символу рядка, то ми переміщуємо підрядок на один символ праворуч і знову починаємо перевірку з останнім символом. Весь алгоритм працює до тих пір, поки не буде знайдено входження шуканого підрядка, або не знайдеться закінчення рядка.

Величина зміщення у випадку невідповідності останнього символу обчислюється виходячи з наступного: зміщення під рядком повинно бути мінімальним, щоб не пропустити входження під рядком у рядку. У випадку якщо підрядка взагалі не містить цього символу, то ми переміщуємо підрядок на величину, рівну його довжині, так що перший символ рядка накладається на наступний за символом,який перевіряється. Якщо заданий символ рядка зустрічається в підрядку, тоді ми зміщуємо підрядок так, щоб символ рядка збігся із самим правим входженням цього символу в підрядок. Величина зміщення кожного символу підрядка залежить від порядку символів у підрядку, тому зручно заздалегідь обчислити зміщення та зберегти його як одновимірний масив, де кожному символу алфавіту відповідає зміщення відносно останнього символу під рядка.

Для обчислення таблиці зсувів підрядка ми повинні вказати вигляд тип таблиці зсувів який ми записали ось таким чином Y_table : array [char] of byte. Перевага даного алгоритму полягає в тому, що необхідно зробити кілька попередніх обчислень над підрядком, щоб порівняння підрядка з початковим рядком проводилося не в усіх позиціях − деякі перевірки пропускаються як ті, що не дають результатів. Найгірший час роботи алгоритму це O($m + n$).

Отже, алгоритм Бойєра і Мура є найбільш ефективним у процесі реалізації і широко використовуваний, а його швидкість виконання підвищується при збільшенні під рядка. На рис. 4 зображено алгоритм Бойєра і Мура [5, с. 89].

```
procedure TForm1.Button1Click(Sender: TObject);
  var
 result:byte; i, j, k: byte;
    Y_len  : byte;
    X_len : byte;
    Y_table : array [char] of byte;
            X, Y : string;
begin
      Y_len := length(Y);
X_len := length(X);
if Y_len < X_len then
begin
   for i := 0 to 255 do
      Y_table[chr(i)] := Y_len;
```

```
        for i := 1 to Y_len-1 do
            Y_table[Y[i]] := Y_len-i;
        i := Y_len; j := i;
        while (j > 0) and (i <= X_len) do
        begin
            j := Y_len; k := i;
            while (j > 0) and (X[k] = Y[j]) do
            begin
                dec(k); dec(j);
            end;
            i := i + Y_table[X[i]];
        end;
            if k > X_len - Y_len then result := 0
        else result := k + 1;
    end
    else
        result :=0;
```

Рис. 4. Реалізація алгоритму Бойєра-Мура

Висновки. Розглянувши чотири алгоритми знаходження підрядка в рядку можна зробити висновок, що алгоритм прямого пошуку не є оптимальним, оскільки характеризується високою складністю *O (N·M)* для пошуку в багатомегабайтних файлах. Провідними є алгоритми Бойєра-Мура та Кнута-Морріса-Пратта, оскільки вони є ефективні та виконують певні класи поставлених завдань.

Список використаних джерел:

1. Альсведе Р., Вегенер І. Завдання пошуку. К.: «Світ», 1982 р.
2. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона. М.: ДМК Пресс, 2010. 272 с.
3. Співаковський О.В. Основи алгоритмізації та програмування. Обчислювальний експеримент. Розв'язання проблем ефективності в алгоритмах пошуку та сортування: Навч. посіб. Херсон: Айлант, 2011. 100 с.
4. Кнут Д. Искусство программирования для ЭВМ. Т.3. Сортировка и поиск. М.: Мир, 1976-1978.
5. Мелешко Є.В. Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. Кропивницький: Видавець – Лисенко В.Ф., 2019. 156 с.

A PROGRAM REALIZATION OF ALGORITHMS OF SEARCHING SUBSTRINGS IN STRINGS

Abstract: *The article is devoted to the consideration of finding algorithms in a string implementing in a software environment. The search algorithm can be implemented using sequential (direct) search algorithms, the Rabin-Carp, Knuth-Morris-Pratt algorithm, and Boyer-Moore. Checking the speed and time of execution of algorithms, as well as software implementation of algorithms using various functions.*

Keywords*: sequential (direct) search algorithm, Rabin-Carp algorithm, Knut-Morris-Pratt algorithm, Boyer-Moore algorithm, hash function, funktsiya prefiksa.*