**Реферат з програмування:**

**ПОШУК ЗРАЗКА В РЯДКУ**

**1. Оцінка кількості порівнянь**

**Задача**. У рядку відшукати всі позиції, починаючи з яких інший рядок (*зразок*) *входить в рядок*, тобто є його підрядком. Наприклад, у рядку

ABRACADABRA

зразок ABR входить як підрядок з позицій 1 і 8, зразок A – з позицій 1, 4, 6, 8 і 11, а зразок ARA не входить.

Позначимо через *s* рядок, у якому шукається зразок *x*. Нехай *m* і *n* – довжини рядків *s* і *x*. Можна порівняти з *x* усі підрядки *s* довжини *n*, які починаються з позицій 1, 2, … , *m*-*n*+1. У разі рівності друкується відповідна позиція:

**for** k:=1 **to** m-n+1 **do**

**if** copy(s, k, n)=x **then** writeln(k).

Нагадаємо, що з виклику copy(s, k, n) повертається підрядок рядка *s*, що починається в його позиції *k* та має довжину *n*. Дуже просто, але дуже нерозумно! Адже загальна кількість порівнянь символів є (*m*-*n*+1) *n*. Наприклад, за *m*=255, *n*=128 порівнянь символів буде 1282=16384, хоча *більшість їх насправді зайва*. Ми переконаємося в цьому, розглянувши далі зовсім інші способи пошуку зразка.

Але спочатку оцінимо зверху кількість порівнянь символів. Зафіксуємо довжину рядка *m*. Нехай довжина зразка *n* довільна в межах між 1 та *m*. Тоді (*m*-*n*+1) *n*<*m n*. Як бачимо, різниця *n*2-*n* між *m n* та (*m*-*n*+1) *n* мала за значень *n*, близьких до 1, і велика за *n*, близьких до *m*. За малих значень *n* величиною *n*2-*n* можна нехтувати. Таким чином, наша оцінка

(*m*-*n*+1) *n* = *O*(*m n*)

є досить точною за малих значень *n* і грубою – за великих. Припустивши, що зразки з великою довжиною – явище дуже рідкісне, можна вважати цю оцінку цілком прийнятною.

**2. Метод Бойєра-Мура (спрощений варіант)**

Один із способів суттєво зменшити кількість порівнянь належить Бойєру та Муру [BoMo]. Розглянемо спрощений варіант їх алгоритму. Нехай символи рядка й зразка належать деякому алфавіту. Нехай зразок *x*=*x*[1]*x*[2]…*x*[*n*]. Спочатку для кожного символу *Z* алфавіту визначається номер позиції *p*[*Z*] його *останньої появи* в рядку *x*. Якщо символ *Z* відсутній в *x*, то *p*[*Z*]=0. Наприклад, у зразку 'ababc' *p*['a']=3, *p*['b']=4, *p*['c']=5, а для решти символів *Z* алфавіту *p*[*Z*]=0.

Обчислення масиву *p* очевидне:

*Для всіх символів Z алфавіту p*[*Z*]:=0;

**for** k:=1 **to** n **do** p[*x*[k]]:=k.

Інформація про останню появу символів у зразку використовується так. Порівняємо одразу *s*[*n*] та *x*[*n*]. Якщо *s*[*n*]  *x*[*n*], то найближчим до кінця зразка символом, якому рівний *s*[*n*], є символ *x*[*p*[*s*[*n*]]]. Таким чином, можна не порівнювати *s*[*n*] із жодним із символів зразка між *x*[*p*[*s*[*n*]]] та *x*[*n*]. А це означає, що можна не перевіряти рівність зразка з підрядками, що починаються з позицій 2, 3, … , *n*-*p*[*s*[*n*]]. Наприклад, якщо *x*='ababc', а рядок *s* починається символами aaaba, то *p*[*s*[5]]=3 підказує, що зразок не може починатися в рядку з позиції 5-3=2. Отже, за *s*[*n*] *x*[*n*] можна перейти одразу до порівняння *x*[*n*] із *s*[*n*+(*n*-*p*[*s*[*n*]])].

Якщо *s*[*n*]=*x*[*n*], то можна порівняти попередні символи рядка з відповідними символами зразка, рухаючися від його кінця до початку. Якщо всі відповідні символи рівні, то зразок є підрядком, що починається з першої позиції рядка. Після цього можна переходити до аналізу другої позиції *s*, порівнюючи *x*[*n*] із *s*[*n*+1].

Якщо за деякого *k*>0 *s*[*k*] *x*[*k*], то серед *x*[*k*-1], … , *x*[1] треба відшукати найближчий до *x*[*k*] символ *x*[*j*]=*s*[*k*]. Ця рівність означає, що зразок, можливо, має кінець у рядку в позиції *k*+(*n*-*j*), тобто *n*+(*k*-*j*). Тоді можна знову починати все з кінця зразка, порівнюючи *x*[*n*] із *s*[*n*+(*k*-*j*)].

Нехай змінна last позначає позицію кінця зразка в рядку *s*. Спочатку last=*n*, а його наступним значенням може бути лише, як показує попередній аналіз, або *n*+1, або *n*+(*n*-*p*[*s*[*n*]]), або *n*+(*k*-*j*). За будь-якого з цих значень змінної last наступним її значенням буде так само або last+1, або last+(last-*p*[*s*[*n*]]), або last+*k*-*j*. На основі цих міркувань записується такий спрощений варіант алгоритму Бойєра-Мура:

last:=n;

**while** last<=m **do**

**if** x[n]<>s[last] **then** last:=last+(n-p[s[n]])

**else**

**begin**

k:=n-1; ok:=**true**;

**while** (k>0) **and** ok **do**

**if** x[k]=s[last-n+k] **then** k:=k-1 **else** ok:=**false**;

**if** k=0 **then** {s[last-n+1]…s[last]=*x*}

**begin**

*повідомити про те, що з last-n+1 починається зразок*;

last:=last+1

**end else**

**begin**

*відшукати серед x[1]…x[k-1] найближчий до x[k]*

*символ x[j], рівний s[last-n+k]; якщо такого немає, то j:=0*

last:=last+(k-j)

**end**

**end**.

Зауважимо, що цей спрощений варіант в деяких випадках не рятує від необхідності здійснювати *O*(*m n*) порівнянь символів. Справжній алгоритм Бойєра-Мура забезпечує, що кількість порівнянь символів за будь-яких рядків довжини *m* і *n* оцінюється як O(*m*+*n*), тобто її можна вважати пропорційною сумі довжин рядка й зразка. Ідея цього методу приблизно така сама, як і методу з наступного підрозділу.

**3. Метод Кнута-Морріса-Пратта**

Цей метод уперше описано Моррісом і Праттом у [MorPr]. Він наведений також у книзі [АХУ].

Почнемо порівнювати символи зразка *x*=*x*[1]…*x*[*n*] із символами рядка *s*=*s*[1]…*s*[*m*] із початку. Нехай *s*[1]=*x*[1], … , *s*[*j*-1]=*x*[*j*-1], *s*[*j*] *x*[*j*], де *j n*. Зрозуміло, що зразок не входить у рядок із першої позиції. Можна, звичайно, спробувати почати перевірку з другої позиції, але зовсім не обов'язково. Наприклад, за зразка *x*='ababb' й рядка *s*='ababababbab' після того, як виявилося, що *s*[5]='a' 'b'=*x*[5], є сенс починати наступну перевірку лише з *s*[3], оскільки саме там є входження початку зразка. Символами *s*[3]*s*[4]='ab' водночас закінчується й починається частина зразка *x*[1]*x*[2]*x*[3]*x*[4], і наступне входження зразка можливе, коли *x*[1]*x*[2] займуть місце *x*[3]*x*[4], тобто зразок "зсунеться" відносно рядка одразу на дві позиції. Після цього можна продовжити перевірку від символу *s*[5], тобто *без повернення назад у рядку s*.

Далі виявляється *s*[7] *x*[5], і зразок можна зсунути одразу на дві позиції, щоб *x*[1]*x*[2] знову зайняли місце *x*[3]*x*[4], збігаючися при цьому з *s*[5]*s*[6]. Тепер *s*[7]=*x*[3], *s*[8]=*x*[4], *s*[9]=*x*[5], і входження починаючи з позиції 5 знайдено.

Отже, нехай перевіряється входження зразка від позиції *i*-*j*, *x*[1]…*x*[*j*]=*s*[*i*-*j*]…*s*[*i*-1], а *x*[*j*+1] не збігається з черговим символом рядка *s*[*i*]. У такому разі треба відшукати такий *найдовший початок x[1]…x[k] зразка, що водночас є кінцем підрядка x[1]…x[j]*. Він є також і кінцем підрядка *s*[1]…*s*[*i*-1]!

Перехід від перевіреного початку зразка довжини *j* до перевіреного початку довжини *k* означає зсув зразка відносно рядка *s* одразу на *j*-*k* позицій. Але на меншу кількість позицій зсувати зразок немає сенсу, оскільки *x*[1]…*x*[*k*] – це *найдовший початок зразка, що збігається з кінцем підрядка s*[1]…*s*[*i*-1].

Якщо *x*[*k*+1]=*s*[*i*], то можна продовжувати порівняння від символу *s*[*i*+1]. Якщо *x*[*k*+1] *s*[*i*], то треба відшукати найдовший початок *x*[1]…*x*[*k*1] зразка, що збігається з кінцем *x*[1]…*x*[*k*] (і з кінцем *s*[1]…*s*[*i*-1]), і порівняти *x*[*k*1+1] із *s*[*i*] тощо.

Наприклад, якщо *s*='abababc', а *x*='ababc', то при спробі "прикласти" зразок починаючи з першого символу рядка маємо *x*[1]=*s*[1], *x*[2]=*s*[2], *x*[3]=*s*[3], *x*[4]=*s*[4], *x*[5] *s*[5], тобто *j*=4. Відповідним значенням *k* буде 2, оскільки 'ab' є найдовшим початком рядка 'abab', що є водночас його кінцем. Звідси випливає, що немає сенсу пробувати "прикласти" зразок до рядка, починаючи з його другої позиції, а слід "пересунути" його одразу на *j*-*k*=2 позиції. При цьому гарантується рівність *x*[1]…*x*[*k*] і *s*[*i*-*k*]…*s*[*i*-1], тобто назад від позиції *s*[*i*] в рядку можна не повертатися.

Отже, *якщо для кожної позиції j зразка відома найбільша довжина f(j)<j такого початку зразка x[1]…x[f(j)], що збігається з кінцем x[1]…x[j], то перше входження зразка знаходиться без повернень у рядку s*. Для визначення можливого початку наступного входження треба знати лише *f*(*n*) і продовжувати пошук знову-таки без повернень у рядку! Саме *відсутність повернень у рядку дозволяє оцінити загальну кількість порівнянь як O(m+n), що суттєво менше, ніж O(m n)*. Ми доведемо це далі.

Функція *f*(*j*), що виражає довжину такого найдовшого початку рядка *x*[1]…*x*[*j*], що є водночас його кінцем, називається ***функцією відступів***. Вона показує, до якого символу *x*[*f*(*j*)] треба *відступити в зразку*, коли *x*[*j*+1] не збігається з черговим символом рядка, щоб продовжувати пошук із порівняння чергового символу з символом *x*[*f*(*j*)+1]. Цей відступ рівносильний зсуву рядка на найменшу можливу кількість позицій *j*-*f*(*j*). Займемося тепер обчисленням цієї функції за зразком.

Очевидно, *f*(1)=0. Нехай всі значення *f*(1), … , *f*(*j*-1) уже обчислено, причому *f*(*j*-1)=*k*. Якщо *x*[*j*]=*x*[*k*+1], то кінець рядка *x*[1]…*x*[*j*-1]*x*[*j*] збігається з його ж початком довжини *k*+1, тому *f*(*j*)=*k*+1. Якщо *x*[*j*] *x*[*k*+1], то "наступним кандидатом у кінці" рядка *x*[1]…*x*[*j*-1]*x*[*j*] є рядок *x*[1]…*x*[*f*(*k*)]*x*[*f*(*k*)+1], оскільки саме *x*[1]…*x*[*f*(*k*)] є найдовшим кінцем *x*[1]…*x*[*k*]. Якщо й він не годиться, то наступним є *x*[1]…*x*[*f*(*f*(*k*))+1] тощо. Отже, ми або знайдемо початок довжини *p*, такий, що *x*[1]…*x*[*p*] є кінцем *x*[1]…*x*[*j*], і тоді *f*(*j*)=*p*, або не знайдемо, і *f*(*j*)=0.

Наведені обчислення описуються таким алгоритмом (подамо функцію *f* масивом):

f[1] := 0;

**for** j := 2 **to** n **do**

**begin**

k := f[j-1];

**while** (x[j] <> x[k+1]) **and** (k>0) **do**

k := f[k];

**if** (x[j] <> x[k+1] ) **and** (k=0) **then** f[j] := 0

**else** f[j] := k+1;

**end**;

Оцінимо загальну кількість порівнянь символів, виконуваних за цим алгоритмом. Позначимо через *w*(*j*) кількість виконань тіла циклу за відповідного значення *j*=2, … , *n*. Помітимо, що кожне виконання тіла циклу while зменшує значення *k* не менше, ніж на 1. Звідси *f*[*j*]<=*f*[*j*-1]-*w*(*j*)+1, тобто *w*(*j*)<=*f*[*j*-1]-*f*[*j*]+1. Тоді

*w*(2)+*w*(3)+…+*w*(*n*)  *f*[1]-*f*[2]+1+*f*[2]-*f*[3]+1+…+*f*[*n*-1]-*f*[*n*]+1 =

= *f*[1]-*f*[*n*]+*n*-1  *n*-1.

За кожного *j* порівнянь символів відбувається на 2 більше, ніж виконань тіла циклу – одне додаткове при обчисленні умови в заголовку циклу і одне в умовному операторі. Звідси загальна кількість порівнянь символів не більше 3(*n*-1), тобто прямо пропорційна *n*. Таким чином, загальну кількість порівнянь символів при побудові функції відступів можна оцінити як *O*(*n*).

Тепер, нарешті, наведемо алгоритм пошуку входжень зразка в рядок. Нехай *t* позначає номер поточної позиції в рядку, *j* – номер поточної позиції в зразку, спочатку вони рівні 1. Далі, поки *t m*, виконуємо наступні дії. Порівнюємо символи *x*[*j*] і *s*[*t*]. Якщо вони рівні, маємо входження *x*[1]…*x*[*j*] в кінці рядка *s*[1]…*s*[*t*]. Якщо при цьому *j*=*n*, то можна повідомити про входження зразка починаючи з позиції *t*-*j*+1 і приступати до пошуків наступного входження, поклавши *j*=*f*(*n*). Якщо ж *j*<*n*, то переходимо до наступної пари символів, збільшивши *t* і *j* на 1. За нерівності *s*[*t*] і *x*[*j*] при *j*>1 міняємо значення *j* на *f*[*j*-1]+1, а при *j*=1 збільшуємо *t* на 1. Втім, зміни *j* не мають сенсу, коли *t*=*m*. Ось і все. Наведемо цей алгоритм також і в мові Паскаль:

t:=1; j:=1;

**while** t<=m **do**

**begin**

**if** x[j]=s[t] **then**

**if** j=n **then**

**begin** writeln(t-j+1); j:=f[j] **end**

**else**

**begin** t:=t+1; j:=j+1 **end**

**else** { x[j]<>s[t] }

**if** t<m **then**

**if** j>1 **then** j:=f[j-1]+1 **else** t:=t+1

**else** t:=t+1

**end**.

Оцінимо тепер кількість порівнянь символів при виконанні цього алгоритму. Помітимо, що при кожному виконанні тіла циклу збільшується *t* на 1 або зменшується *j* принаймні на 1 присвоюванням *f*[*j*] чи *f*[*j*-1]+1. Позначимо через *b*(*t*) початкове значення *j* при черговому значенні *t*=1, 2, …, *m*, а через *w*(*t*) – кількість зменшень *j* при відповідному значенні *t*. Оскільки при збільшенні *t* значення *j* або не міняється, або збільшується на 1, то маємо, що *b*(*t*) *b*(*t*-1)-*w*(*t*)+1 за *t*>1, звідки *w*(*t*) *b*(*t*-1)-*b*(*t*)+1. Тоді

*w*(1)+*w*(2)+…+*w*(*m*)  1+*b*[1]-*b*[2]+1+*b*[2]-*b*[3]+1+…+*b*[*m*-1]-*b*[*m*]+1 =

= *m*+*b*[1]-*b*[*m*]  *m*+1.

З алгоритму очевидно, що порівнянь символів відбувається рівно стільки, скільки збільшень t та зменшень *j* разом. Оскільки *t* збільшується *m*-1 разів, загальна кількість порівнянь символів не більше 2*m*, тобто пропорційна *m*, і оцінюється як *O*(*m*).

З урахуванням побудови функції відступів загальна кількість порівнянь символів за будь-яких рядків довжини *m* і *n* оцінюється як *O*(*n*)+*O*(*m*), або *O*(*m*+*n*).

**ДОДАТОК**

**Службові слова стандарту мови Паскаль**

|  |  |  |  |
| --- | --- | --- | --- |
| **and** | **false** | **mod** | **set** |
| **array** | **file** | **nil** | **then** |
| **begin** | **for** | **not** | **to** |
| **case** | **forward** | **of** | **true** |
| **const** | **function** | **or** | **type** |
| **div** | **goto** | **packed** | **until** |
| **do** | **if** | **procedure** | **var** |
| **downto** | **in** | **program** | **while** |
| **else** | **label** | **record** | **with** |
| **end** | **maxint** | **repeat** |  |

**Додаткові слова в Турбо Паскаль**

|  |  |  |  |
| --- | --- | --- | --- |
| **absolute** | **implementation** | **object** | **unit** |
| **constructor** | **inline** | **shl** | **uses** |
| **destructor** | **interface** | **shr** | **virtual** |
| **external** | **interrupt** | **string** | **xor** |

**СПИСОК ЛІТЕРАТУРИ**

[Аб] Абрамов А.С. Элементы анализа программ.- М., 1986.

[АГКС]Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию.- М., 1988.

[Ан] Андерсон Р. Доказательство правильности программ.- М.:, 1982.

[Арс] Арсак Ж. Программирование игр и головоломок.- М., 1990.

[АУ] Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Т.1.- М., 1978.

[АХУ] Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов.- М., 1979.

[БаГо] Бауэр Ф., Гооз Г. Информатика.- М., 1990.

[Белл] Беллман Р. Динамическое программирование. М., 1960.

[БВК] Бородич Ю.С., Вальвачев А.Н., Кузьмич А.И. Паскаль для персональных компьютеров.-Минск., 1991.

[Бу] Бублик В.В. Методические указания и задания к лабораторным занятиям по курсу "Вычислительные машины и программирование".- Киев, 1986.

[Ви77] Вирт Н. Систематическое программирование. Введение.- М., 1977.

[Ви85] Вирт Н. Алгоритмы + Структуры данных = Программы.- М., 1985.

[Григ] Григорьев В.Л. Микропроцессор i486.- М., 1993.

[Грис] Грис Д. Наука программирования.- М., 1984.

[Гро] Грогоно П. Программирование на языке Паскаль.- М., 1982.

[ГД] Гэри М., Джонсон Д. Вычислительные алгоритмы и труднорешаемые задачи. – М., 1982.

[ЙеВи]Йенсен К., Вирт Н. Паскаль. Руководство для пользования.- М., 1993.

[КаСа] Касьянов В.Н., Сабельфельд В.К. Сборник заданий по практикуму на ЭВМ.- М., 1986.

[Кнут] Кнут Д. Искусство программирования для ЭВМ. Основные алгоритмы.- М., 1976. Т. 1. Получисленные алгоритмы.- М., 1977. Т. 2. Сортировка и поиск.- М., 1978. Т. 3.

[М80] Майерс Г. Надежность программного обеспечения.- М., 1980.

[М82] Майерс Г. Искусство тестирования программ.-М., 1982.

[ПоКр]Поляков Д.Б., Круглов И.Ю. Программирование в среде Турбо Паскаль. Версия 5.5. М., 1992.

[Пи] Пильщиков В.Н. Сборник упражнений по языку Паскаль.-М., 1989.

[ПрЧС]Проценко В.С., Чаленко П.Й., Ставровський А.Б. Техніка програмування мовою Сі.- К., 1993.

[РНД] Рейнгольд Э., Нивергельт Ю., Део Н. Комбинаторные алгоритмы. М., 1980

[Сл] Слэйгл Дж. Искусственный интеллект. М.: Мир, 1973.

[СтКо] Ставровський А.Б., Коваль Ю.В. Вступний курс програмування.- К., 1998.

[Фар] Фаронов В.В. Турбо Паскаль 7.0. Начальный курс.- М., 1997.

[Фор] Форсайт Р. Паскаль для всех.- М., 1987.

[BoMo]Boyer R.S., Moore J.S. A fast string searching algorithm.- Comm.ACM, 1977.- № 10

[MorPr]Morris J.H. Jr, Pratt V.R. A linear pattern matching algorithm.- Tech.Rept. 40, Comput. Centre, University of California, Berkeley.- 1970