УДК 683.3.01:004.451.53
ББК 32.976я73
Л 93

Рецензенти: *О.Г. Корченко*, д.т.н., професор (НАУ)
О.Й. Стасюк, д.т.н., професор (КДУЕТТ)

Гриф надано Вченою радою
ІСЗЗІ НТУУ «КПІ»
(Протокол № 3 від 21.07.2008)

Л 93 Методичний посібник з практичних занять з навчальної дисципліни «Методи та засоби комп'ютерних інформаційних технологій: Основи теорії інформаційного пошуку» / Ланде Д.В., Мохор В.В. – К.: ІСЗЗІ НТУУ «КПІ», 2009. – 78 с.

Коротко викладений матеріал, необхідний для проведення практичних занять, в рамках яких мають бути отримані навички реалізації пошукових процесів, моделювання складних графів, засобів обробки потоків текстової інформації, програмування мовою програмування Perl у веб-середовищі.

Навчальне видання призначено студентам, які навчаються за напрямом підготовки «Комп'ютерні науки» і вивчають нормативну навчальну дисципліну «Методи та засоби комп'ютерних інформаційних технологій». Може бути також корисне всім, хто самостійно хоче отримати практичні навички застосування сучасних методів і засобів повнотекстових інформаційно-пошукових систем – курсантам, студентам, аспірантам, спеціалістам інших технічних спеціальностей та програмістам.

УДК 683.3.01:004.451.53
ББК 32.976я73

ЗМІСТ

1. Передмова

Сучасні фахівці, які працюють у сфері комп'ютерних наук, обов'язково повинні мати теоретичні знання та практичні навички роботи з інформаційними технологіями, зокрема із засобами моделювання інформаційного пошуку, інформаційно-пошуковими системами в глобальній інформаційно-телекомунікаційній мережі Інтернет.

Тому теоретичним і технологічним питанням побудови пошукових систем, складних мереж, інформаційних потоків, організації пошуку в комп'ютерних інформаційних мережах і присвячено навчальну дисципліну «Методи та засоби комп'ютерних мереж», яка є нормативною дисципліною з напряму підготовки «Комп'ютерні науки».

В даній навчальній дисципліні практичним заняттям відведено значне місце, так як саме вони дозволяють отримувати необхідні практичні навички. Загальний обсяг практичних занять складає 46 годин навчального часу у комп'ютерному класі. На цих заняттях повинні бути отримані практичні навички з реалізації пошукових процесів, моделювання складних графів, засобів обробки потоків текстової інформації, програмування мовою Perl у веб-середовищі.

З метою ефективного проведення практичних занять в Інституті спеціального зв'язку та захисту інформації НТУУ «КПІ» створено спеціалізоване технологічне середовище, яке розглядається як середовище моделювання, що разраховано на одночасну роботу багатьох користувачів. Разом з тим передбачається, що елементи цього середовища можуть бути перенесені на індівідуальні робочі місця (наприклад, ноутбуки) для автономного проведення самостійної роботи.

Це видання призначено перед усім для підвищення ефективності практичних занять та забезпечення моделювання інформаційних процесів і процедур. Основна мова програмування, елементи якої застосовуються при цьому моделюванні, не є окремим предметом даного курсу. Основні елементи мови Perl викладено у окремому додатку, надано відповідні посилання у списку рекомендованих інформаційних джерел.

Цей посібник розраховано на курсантів, слухачів та студентів бакалаврату, що спеціалізуються в комп'ютерних науках, і відповідно, які мають достатню підготовку в таких галузях математики, як теорія множин, вища алгебра, диференціальні рівняння та теорія ймовірностей, а також здатні оволодіти необхідними навичками для проведення практичних занять: елементами мови програмування Perl, програмуванням у веб-середовищі, типовими засобами статистичної обробки даних.

1. Технологічне середовище проведення практичних занять

Спеціалізоване технологічне середовище (скорочено – **ТС**) проведення практичних занять у рамках курсу «Методи та засоби комп'ютерних інформаційних технологій» реалізовано за технологією «клієнт-сервер» та встановлено на сервері під управлінням операційної системи FreeBSD. Доступ до ТС можливий з боку багатьох користувачів, яким надаються адреси для входження (наприклад, http://192.168.0.16/~user01) та відповідні авторизаційні та аутентифікаційні дані (логін і пароль).

Перше, що бачить користувач після авторизації, це заставка (рис. 1.1), з якої шляхом натискання на кнопку «**Satrt CGI File Manager**» здійснюється перехід до робочої області ТС.

Рис. 1.1. Титульна сторінка технологічного середовища проведення практичних занять

Слід зазначити, що технологічне середовище побудовано як CGI-застосування, тобто доступ до нього з боку користувачів здійснюється через звичайний веб-браузер, наприклад Opera, FireFox або Internet Explorer будь-

яких версій. Передбачається, що за допомогою ТС буде здійснюватися моделювання інформаційних процесів і процедур за допомогою засобів мови маркування текстів HTML та мови програмування Perl. До речі, саме технологічне середовище розроблено виключно за допомогою цих засобів.

Робоча область ТС являє собою своєрідний файловий менеджер (рис. 1.2), за допомогою якого здійснюється створення (кнопка «**Створення нового файлу**»), читання (кнопка «**READ**»), редагування (кнопка «**EDIT**») та вилучення (кнопка «**DEL**») файлів. Крім того, якщо файл являє собою програму мовою Perl або текст у HTML-маркуванні, то можливе (кнопка «**RUN**») його виконання або інтерпретація, відповідно .

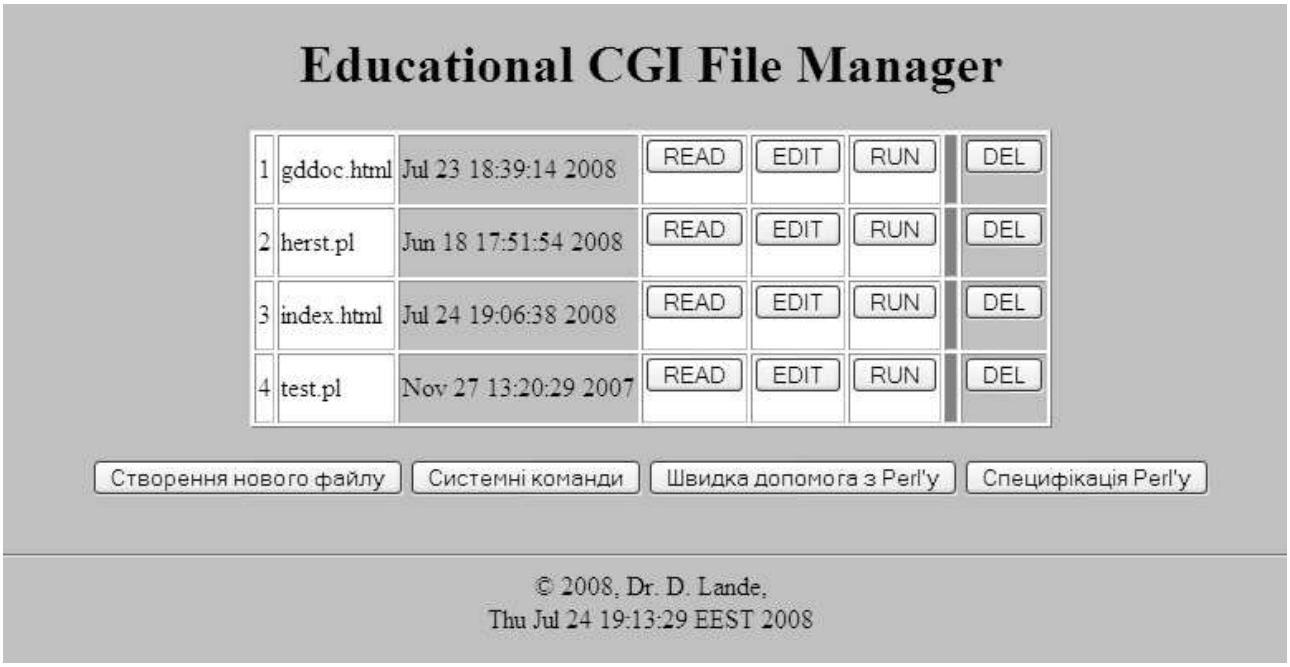


Рис. 1.2. Робоча область ТС

Кожний користувач ТС бачить у робочому вікні перелік файлів з його власної директорії. Оцінка роботи викладачем здійснюється шляхом оцінки якості програм та HTML-файлів, створених та наведених у каталозі користувача.

За замовченням у файловому менеджері кожного користувача відображається файл ***index.html***, який визначає титульну сторінку ТС. Цей файл неможливо випадково видалити кнопкою «**DEL**». Його зміст являє собою текст у HTML-маркуванні, доступний для читання за допомогою кнопки «**READ**»:

```
<html>
<head>
<title>Educational CGI File Manager</title>
</head>
<body bgcolor=silver>
```

```
<center>
<h1>Інститут спеціального зв'язку та захисту інформації<br>
Національного технічного університету України<br>
"Київський політехнічний інститут"
</h1>
<p>
<h2>Методи та засоби<br>
комп'ютерних інформаційних технологій
<br>Теорія інформаційного пошуку<br>
Educational Environment
</h2>
<table border=0>
<tr>
  <td>
    <form method=get action="../filer/filer.pl">
      <input type=hidden name=dir value="down">
      <input type=submit value="Start CGI File Manager">
    </form>
  </td>
</tr>
</table>
<img src="../img/ic331.gif" border=0>
<br>
© 2008, Ланде Д.В.
</center>
</body>
</html>
```

Прокоментуємо зміст цього файлу. Тег <***html***> говорить про те, що це текст у HTML-маркуванні, при відображенні даного файлу у верхній частині вікна поряд з іконкою браузера буде присутній заголовок: «**Educational CGI File Manager**». Колір фону цієї HTML-сторінки буде сірим (***bgcolor=silver***), весь текст буде відображатися по центру (тег (<***center***>)). Для виклику файлового менеджеру ТС застосовується тег ***form***, опції якого означають наступне:

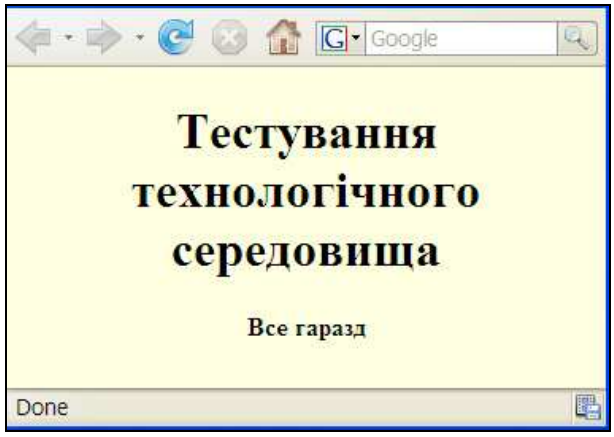
- ***<input type=hidden name=dir value="down">*** - передача програмі, що викликається, параметру з ім’ям dir та значенням down «прихованим» шляхом, тобто без відображення у вікні браузера;
- ***<input type=submit value="Start CGI File Manager">*** - формування кнопки з надписом «Start CGI File Manager», натискання якої викликає CGI-програму.

ТС дозволяє ознайомитися з детальною специфікацією мови Perl (кнопка «**Специфікація Perl’у**») та з коротким викладенням основ програмування у середовищі CGI (кнопка «**Швидка допомога з Perl’у**») . Крім того користувачу доступна робота з системними командами FreeBSD (кнопка «**Системні команди**»), завдяки чому йому доступний широкий арсенал засобів роботи у середовищі операційної системи, наприклад, перегляд параметрів файлів і

директорій (ls –l *), перегляд системної документації (man Perl), сортування файлів, зупинка процедур, що увійшли у нескінченний цикл тощо.

В рамках першого практичного заняття користувачам пропонується самостійно створити власні тексти у HTML-маркуванні, використовуючи кнопку ТС «Створення нового файлу». Необхідно зауважити, що в рамках ТС припустиме ім'я HTML-файлу може складатися лише з латинських букв, цифр і знаку «_» та розширення ***«.html»***. Приклад самостійно створеного файлу (***user01.html***):

```
<html>
<body bgcolor=lightyellow>
<center>
<h1>Тестування технологічного
середовища</h1>
<p>
<b>Все гаразд</b>
</p>
</center>
</body>
</html>
```



Питання для самостійної роботи:

1. Ознайомитися зі специфікацією мови маркування HTML [4].
2. Самостійно засобами ТС створити HTML-файл, перевірити режими перегляду, редагування, інтерпретації та видалення.
3. Проаналізувати вихідний текст динамічного HTML-файлу робочої області ТС.

2. Елементи програмування мовою Perl

Технологічне середовище проведення практичних занять побудовано як сукупність віртуальних робочих місць, за допомогою яких можна писати та налагоджувати програми мовою Perl, створювати та редагувати текстові та HTML-файли. Для створення програм в ТС необхідно активізувати режим створення нового файлу (див. рис. 2.1), після чого ввести назву файлу програми (назва файлу може складатися з латинських букв, знаку підкреслення «_» та цифр), розширення файлу – «***.pl***».

У Додатку до даного методичного посібника наведено короткий опис мови Perl, у тих межах, які необхідні для розуміння та написання програм, які будуть створюватися у рамках цього курсу.

Першим рядком програми має бути вираз, який вказує на місце розташування модуля Perl:

```
#!/usr/local/bin/perl -w
```

Параметр ***–w*** вказує на те, що при налаштуванні Perl буде видавати не тільки інформацію щодо синтаксичних помилок, але й попередження (наприклад щодо одноразового застосування у тексті деякої змінної).

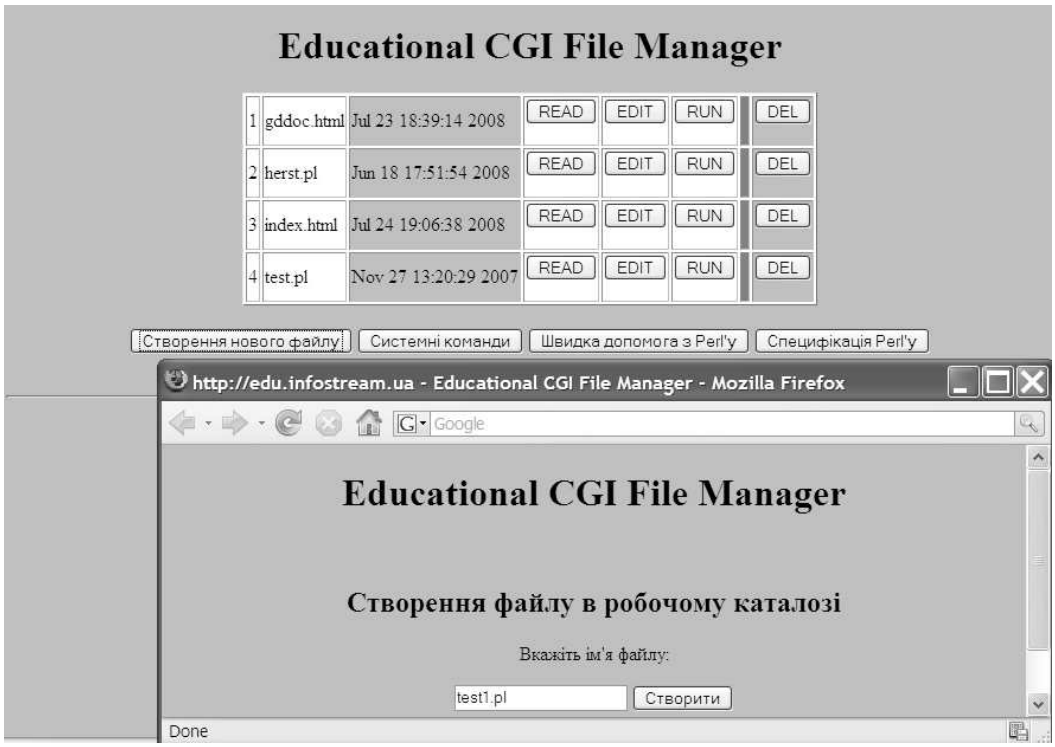


Рис. 2.1. Створення файлу програми test1.pl

У випадку необхідності виявлення помилок треба запустити програму на Perl з режиму «Системні команди», спрямувавши вихідний потік ***STDERR*** у якийсь файл, наприклад:

```
./test1.pl 2> err.txt
```

Після цього засобами технологічного середовища можна переглянути файл ***err.txt*** та ознайомитися з діагностикою щодо можливих помилок та попереджень. Як перше практичне заняття пропонується розробити програму обчислення факторіалів для чисел, що не перевищують наперед заданого числа (рис 2.2).

Для забезпечення виведення даних за допомогою ТС у тексті програми на Perl має бути присутній рядок, що забезпечує коректне застосування протоколу HTTP:

```
print "Content-type: text/html\n\n";
```

У наведеній програмі треба звернути увагу на дію двох операторів - ***for*** та ***print***. Щодо першого, то у ньому застосовується робоча змінна ***$i***, яка змінюється у діапазоні від 1 до 10, а другий оператор має виводити на екран значення факторіалів, які відповідають значенням ***$i***.

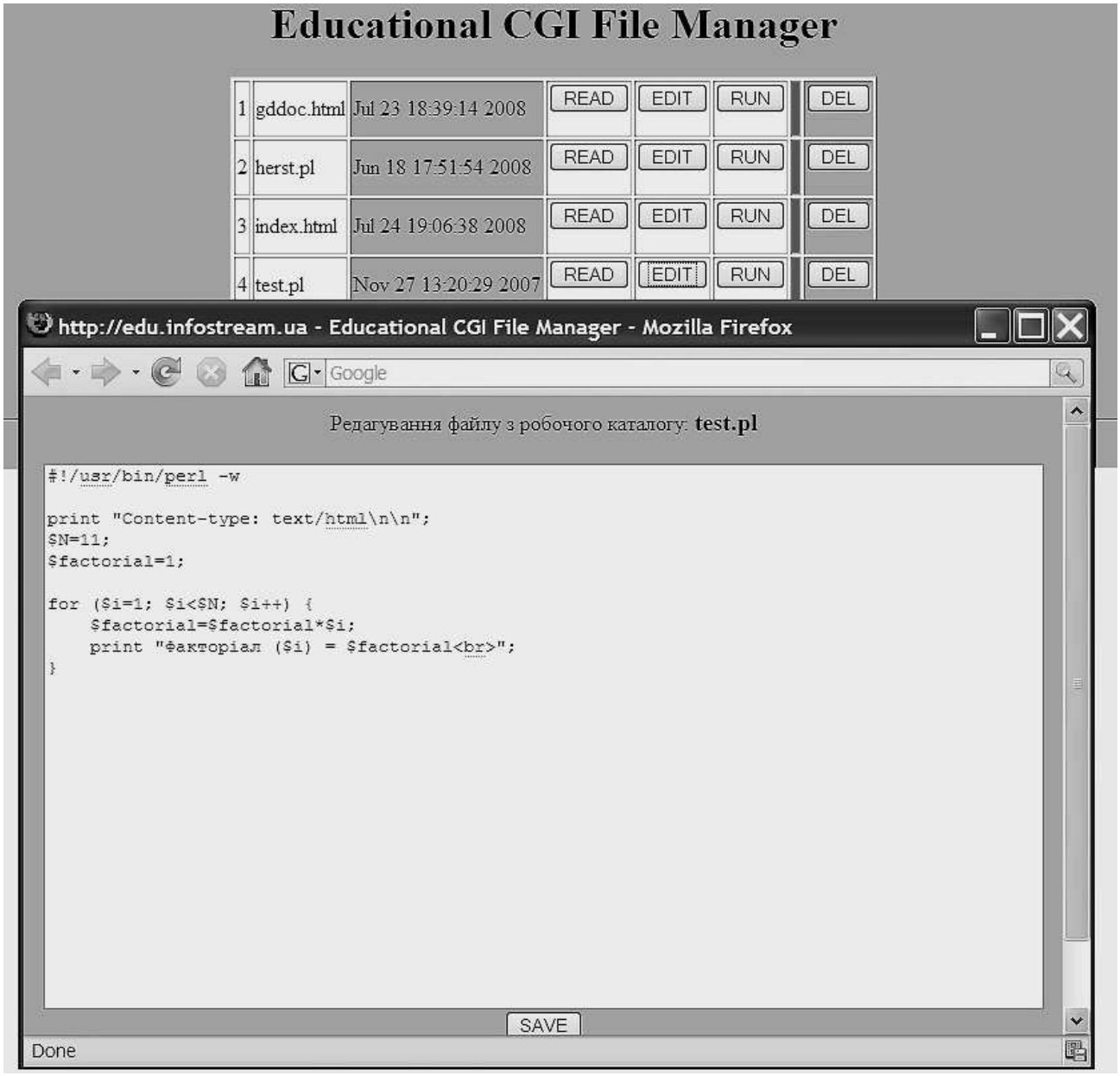


Рис. 2.2. Створення тексту програми обчислення факторіалів

Після збереження тексту програми, її можна запустити на виконання за допомогою кнопки «**RUN**», після чого (у разі успішного налаштування програми) має вивестися:

```
Факторіал (1) = 1
Факторіал (2) = 2
```

```
Факторіал (3) = 6
Факторіал (4) = 24
Факторіал (5) = 120
Факторіал (6) = 720
Факторіал (7) = 5040
Факторіал (8) = 40320
Факторіал (9) = 362880
Факторіал (10) = 3628800
```

Питання для самостійної роботи:

1. Ознайомитися зі специфікацією мови програмування Perl, що наведена у ТС, а також джерелах [5, 6].
2. Самостійно засобами ТС написати програму виведення ряду Фібоначі (числової послідовності, що починається з 1, в якій кожний наступний елемент дорівнює сумі двох попередніх).

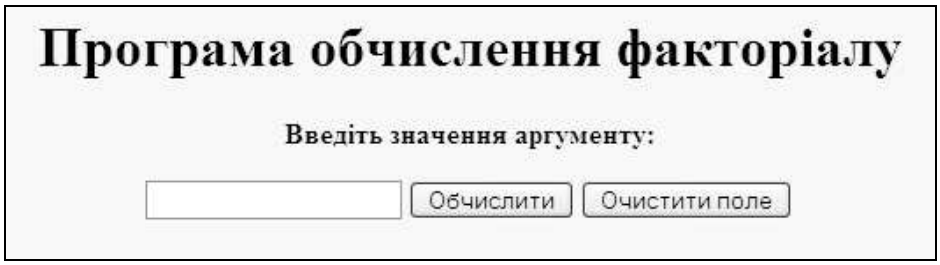
3. Програмування мовою Perl з CGI-інтерфейсом

Для передавання параметрів від HTML-файлу до програми на Perl застосовується так званий ***CGI***-інтерфейс (Common Gateway Interface). У складі мови маркування HTML є спеціальний тег ***form***, за допомогою якого здійснюється виклик програм, розміщених на веб-сервері, та передавання їм параметрів. Застосування цих засобів добре відоме, наприклад, при використанні форм введення запитів у інформаційно-пошукових системах.

Як приклад розглянемо виклик програми обчислення факторіалу числа, яке вводиться у спеціальне поле. Відповідне значення має передатися програмі – скрипту мовою Perl. Нижче наведено фрагмент HTML-файлу в якому у форму можна ввести значення, яке буде передано відповідній програмі.

```
<center>
<h1>Програма обчислення факторіалу</h1>
<p>
 <b>Введіть значення аргументу:</b>
</p>

<p>
 <form method=get action="factor.pl">
   <input type=text name=s>
   <input type=submit value="Обчислити">
   <input type=reset value="Очистити поле">
 </form>
</p>
</center>
```

В інтерфейсі веб-браузера відобразиться наступне:



За допомогою тега <input type=text name=s> передається параметр ***s*** – аргумент для обчислення факторіалу.

В CGI-програму параметри передаються двома методами – ***GET*** та ***POST***.

1) Метод ***GET*** надсилає вхідні параметри, які програма отримує зі змінної середовища оточення ***QUERY_STRING***. У наведеному прикладі параметр передається таким рядком (якщо ввести аргумент 4):

```
factor.pl?s=4
```

2) За допомогою метода ***POST*** надсилаютья вхідні параметри у тілі HTTP-запиту, а програма отримує їх через стандартний вхідний потік ***STDIN***.

```
POST HTTP/1.0 /down/factor.pl
    ... [Інші рядки заголовку]
s=4
```

Нижче наведено програму обчислення факторіалу:

```
#!/usr/local/bin/perl -w
print "Content-type: text/html\n\n";
use CGI;
my $q = new CGI;
$N=$q->param('s');

$f=1;
for ($i=1; $i<=$N; $i++) {
  $f=$f*$i;
}
print "Факторіал ($N) = $f";
```

Результат: `Факторіал (4) = 24`

Для приймання параметру у цій програмі застосовувався стандартний зовнішній модуль CGI (виклик - `use CGI;`). Після цього створюється спискова змінна ***$q*** (`my $q = new CGI;`), яка містить у собі всі параметри, що було передано програмі. Для приймання параметра ***s*** у змінну ***$N*** використовувався наступний оператор:

```
$N=$q->param('s');
```

Питання для самостійної роботи:

1. Ознайомитися зі специфікацією CGI-інтерфейсу [7].
2. Самостійно засобами ТС створити відповідну HTML-форму та написати CGI-програму виведення ряду Фібоначі, що викликається з цієї форми.

4. Вибір тестового документального масиву

Для моделювання процесів інформаційного пошуку застосовується вибірка документів із заздалегідь підготовленого тестового документального масиву у форматі системи InfoStream. Окремі документи з цього масиву – марковані спеціальним чином текстові фрагменти. Для проведення подальших практичних робіт передбачається відібрати з цього масиву лише документи, наведені українською мовою. Файл з вихідними документами, наведеними різними мовами знаходиться на сервері за адресою «***../../test.txt***». В Інтернеті масив текстових документів розміщено, наприклад за адресою: ***http://infostream.ua/ling/test.txt.gz***. Формат маркованого тестового масиву визначається такими правилами:

- документ від документа відокремлюється рядком, який починається послідовністю символів «***»;
- кожний рядок документа складається з двох частин:
- перші три позиції - код поля;
- позиції з одинадцятої до 256 - текст поля.

В документах присутні такі основні поля, які будуть застосовуватися надалі:

№ поля	***Код поля***	***Назва поля***
1	010	Заголовок
2	021	Анонс статті
3	020	Стаття
4	030	Джерело
7	036	Код мови
8	040	Дата, час сканування
9	050	URL
11	038	Ключові слова
15	007	Рубрики, прізвища, коди країн

Приклад документа:

```
*** 3
010       "Яндекс" собрал $72,6 млн за год
021       Инернет-портал "Яндекс" в 2006 году заработал в 2 раза
021       больше,чем в 2005-м. Рентабельность по показателю OIBDA
021       достигла 59,4% по сравнению с 53,3% в 2005 году,
021       сообщила компания
020       Чистая прибыль выросла с $12,5 млн в 2005 году до
020       $29,9 млн, выручка более чем в 2 раза до $72,6 млн,
020       пишет агентство Reuters.
020       Показатель OIBDA (операционная прибыль до вычета
020       амортизации основных средств и нематериальных активов)
020       увеличился на 130% до $43,1 млн.
020       "Финансовые результаты компании за 2006 год уже
020       позволили нам сделать несколько приобретений в 2007
020       году. Среди них социальная сеть "МойКруг.ру", система
020       электронных платежей "Яндекс.Деньги" и команда
020       разработчиков мобильных приложений компании "Смартком",
020       рассказал гендиректор компании Аркадий Волож.
020
020       Число рекламодателей "Яндекса" увеличилось в 2006
020       году на 45% и превысило 30 тыс.
020       Лаура Магомедова
030       Re-port.ru
036       lang.RUS
040       2007.05.01 15:53
050       http://re-port.ru/news/8103/
038       КОМПАН ЯНДЕКС OIBD ПОКАЗАТЕЛ ПРИБЫЛ СИСТЕМ REUTERS
038       АГЕНТСТВ ВЫРУЧК ДОСТИГЛ ЗАРАБОТАЛ ИНЕРНЕТ
007       rubr33
007       name.Волож z.Волож
007       name.Магомедов z.Магомедова
```

Процедура вибору документів з тестового масиву має послідовно зачитувати текстові поля документів у деяку буферну область (позначену змінною ***$doc***). Якщо у полі ***036*** буде знайдена ознака ***lang.UKR***, то зчитану буферна область ***$doc*** передбачається записати у файл ***«ukr.txt»*** з метою подальшої обробки. Нижче наведено відповідний фрагмент програми мовою Perl.

```
$j=0;
$doc="";
open F1,">./ukr.txt";
$mova=0; # ознака мови
open File, "../../test.txt";
while ($_=<File>){
     $buf=$_;
     if ($buf=~/^036.*?lang\.UKR/) {
        $mova=1; # мова - українстька
     }
     if ($buf=~m/^\*\*\*/) {
```

```
        if ($mova==1) { # виведення текстової частини документа
        $j++;
        print F1 "*** ",$j;
        print F1 $doc;
      }
       $doc="";
       $mova=0;
      }
      if ($buf=~m/^0[12][01]\s+(.*)$/) { # вибір текстових полів
        $doc=$doc."\n".$1;
      }
}
if ($mova==1) { # виведення останнього документа
    $j++;
    print F1 "*** ",$j;
    print F1 $doc;
}
close File;
close F1;
```

Питання для самостійної роботи:

1. Ознайомитися з особливостями використання регулярних виразів у мові Perl [7].
2. Самостійно засобами ТС створити програму вибору з тестового масиву документів з вибраного джерела (наприклад, «Електронні Вісті»).

5. Побудова словника текстового масиву

Для побудови словника тестового масиву, сформованого програмою, наведеною у попередньому розділі, передбачається зчитати весь масив у буфер (змінна ***$doc***), після чого застосувати до інформації, що накопичилася у буфері спеціальну процедуру побудови словника. Ця процедура (дамо їй назву ***dict_build***) вилучає з тексту всі розділяючі символи та цифри, змінюючи їх на пробіли, переводить букви до одного регістру та за допомогою функції ***split*** будує масив слів, які сортує за допомогою спеціальної функції ***sort***.

Нижче наведено фрагмент програми мовою ***Perl***, з якої здійснюється виклик процедури ***dict_build()***.

```
    $doc="";
    open FILE,"../ukr.txt";

    while($_=<FILE>){
     $doc=$doc.$_; # накопичення тексту
    }
    close FILE;
    dict_build();
```

Процедура побудови словника текстового масиву буде застосовуватися й надалі. Варіант тексту процедури наведено нижче:

```
sub dict_build() {

 # Побудова словника

 $wrk=$doc; # робоча змінна
 $wrk = ~s/[\r\n]/ /gs; # заміна кінцевих символів рядків
 $wrk = ~tr/[\:-\@]/ /;
 $wrk = ~tr/,;:()?[]!"@\-\.\/\t/ /;
 $wrk = ~tr/0-9/ /;
 # Переведення до верхнього регістру
 $wrk = ~tr/a-z/A-Z/;
 $wrk = ~tr/[ю-ъ]/[Ю-Ъ]/; # діапазони літер у KOI8-u
 $wrk = ~tr/ієї/ІЄЇ/; # деякі українські літери
 $wrk = ~s/\s+/ /g; # заміна множинних пробілів на одиничні

 @w=split(" ",$wrk); # формування масиву слів

 # Кількість слів (без одного):
 $wm=$#w;

 # Сортування словника
 @w=sort(@w);
}
```

Питання для самостійної роботи:

1. Написати вираз для перекодування текстів, наведених не у кодуванні KOI8-u, як у прикладі, а у кодуванні CP1251.
2. Самостійно засобами ТС створити програму побудови та виведення словника для кожного документа з тестового масиву.

6. Частотний розподіл слів у словнику

Дж. Ципф показав, що розподіл слів природної мови підкоряється сталій закономірності: якщо для якого-небудь досить великого тексту скласти список всіх слів, що зустрілися в ньому, а потім ранжирувати ці слова у порядку убування частоти їхньої появи у тексті, то для будь-якого слова добуток його рангу та частоти появи буде величиною сталою: $f * r = c$, де f - частота появи слова у тексті; r - ранг слова у списку; c - емпірична стала величина. Б. Мандельброт уточнив закон Ципфа, довівши: $f * r^e = c$, де e - близька до одиниці величина, що може змінюватися залежно від властивостей тексту та мови.

Для перевірки закону Ципфа для масивів текстових повідомлень реалізуємо відповідну CGI-процедуру.

На першому етапі розробимо HTML-файл, у текстовий масив (***textarea***) форми якого у ручному режимі введемо текст ***ukr.txt***, створений програмами, описаними в розділі 4, який можна отримати у режимі читання за допомогою засобів ТС.

Нижче наведено фрагмент HTML-файлу zipf.html, з якого викликається процедура розрахунку частот слів - ***zipf.pl***. Необхідно звернути увагу на метод передачі параметру у формі (***method=post***), який дозволяє зняти обмеження на обсяг текстового масиву.

```
<form action="zipf.pl" method=post>
     <input type=submit value="Обчислити">
     <input type=reset value="Очистити поле">
     <textarea  name=s rows=30 cols=80>
     </textarea>
</form>
```

Програма мовою Perl має видати список кількості появ у текстовому масиві окремих слів. Передбачається, що отриманий таким чином список буде завантажено у програму ***Microsoft Excel***, встановлену на робочому місті користувача, для подальшої обробки. Нижче наведено фрагмент програми ***zipf.pl***, яка приймає введений текст з вхідного потоку, здійснює її перекодування, формує словник текстового масиву, після чого формує список кількості появ окремих слів.

```
# Приймання тексту як вхідного потоку
$buf=<STDIN>;
$buf=~tr/+/ /;
$buf=substr($buf,2); # Видалили початок рядка «s=»
# Перекодування тексту
$buf =~ s/%(..)/pack("c",hex($1))/ge;

# Побудова словника
dict_build();
# @ns – масив кількості появ слів
$old="=";
$j=0;
$ns[$j]=1;
for ($i=0; $i<$nw+1; $i++) {
   if ($w[$i] eq $old) {
      $ns[$j]++;
   }
   else {
       $j++;
       $ns[$j]=1;
       $old=$w[$i];
       print "$ns[$j-1]\n";
   }
}
```

Після виконання програми zipf.pl в інтерфейсі ТС буде відображено несортований список кількості появ слів. Далі цей список через буфер обміну завантажується до «книги» в інтерфейсі програми Microsoft Excel, нормується (здійснюється перехід від кількості появ до частот), сортується у зворотному порядку, тобто ранжирується (чим вище частота слова, тим нижче значення рангу). Після цього засобами Microsoft Excel будується графік, у якому вказується логарифмічна шкала для обох осей (рис. 6.1). Використовуючи засоби апроксимації (знаходження лінії тренду), виводу рівняння та значення похибки можна переконатися у справедливості закону Ципфа не тільки для окремого тексту, але й для масиву текстових повідомлень, що розглядався як тестовий.

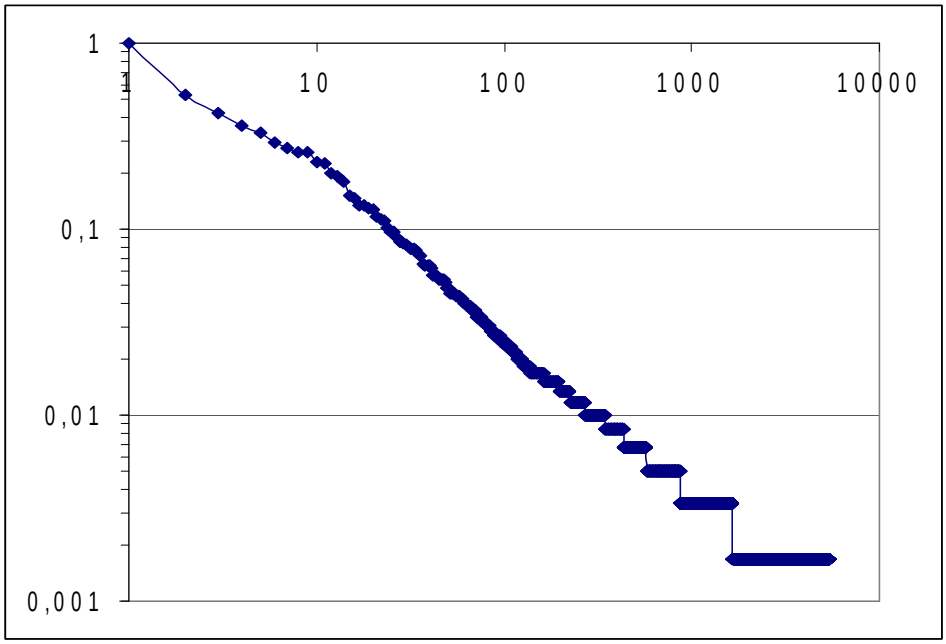


Рис. 6.1. Діаграма «ранг-частота» слів у логарифмічній шкалі

Питання для самостійної роботи:

1. Самостійно побудувати графік залежності «ранг-частота» для тексту будь-якого оповідання, написаного українською мовою.
2. За допомогою режиму «Побудова лінії тренду» пакету Microsoft Excel отримати степеневу апроксимацію, а також визначити параметри розподілу Ципфа.

7. Перевірка закону Хіпса

Емпіричний закон Хіпса зв'язує обсяг документа з обсягом словника унікальних слів, які входять у цей документ. Здавалося б, словник унікальних слів повинен насичуватися, а його обсяг стабілізуватися при збільшенні обсягу тексту. Однак цього не виконується на практиці. У відповідності до закону

Хіпса, значення обсягу документа та словника унікальних слів зв'язані співвідношенням:

$$v(n) = Kn^{\beta},$$

де $v-$ це обсяг словника унікальних слів, складений з тексту, який містить n унікальних слів. K та β – емпірично обумовлені параметри.

Фрагмент HTML-файлу heaps.html, з якого викликається процедура розрахунку обсягу словника унікальних слів - ***heaps.pl***. У цьому випадку метод передачі параметру у формі також, як і у попередньому випадку - ***post***.

```
<form method=post action="heaps.pl">
     <input type=submit value="Обчислити">
     <input type=reset value>
     <br> <textarea name="s" rows=30 cols=80>
     </textarea>
</form>
```

Нижче наведено фрагмент тексту програми виводу даних залежності обсягу словника унікальних слів від розміру тексту.

```
$doc=$buf;

dict_build1(); # виклик процедури, аналогічної dict_build(),
               # однак без сортування словника

$nu=0;         # кількість унікальних слів
$u[0]=$w[0];   # нульовий елемент масиву унікальних слів
$nu++;

$fmas="";      # рядок для накопичення результатів
for ($i=1; $i<=$nw; $i++) {
     for ($j=0; $j<$nu; $j++) {
          if ($w[$i] eq $u[$j]) {
               goto xxx;
          }
     }
     $u[$nu]=$w[$i];
     $nu++;
     xxx:
     # виведення кількості унікальних слів через 500 слів тексту
     if ($i/500 == int($i/500)) {
       $nus=$nu-1;
       print "$i -  $nus<br>\n";
       $fmas=$fmas." ".$nus; # формування масиву результів
     }
}

# форма виклику програми побудови графіку
print "$fmas<br>
<form action='hipsr.pl' method=post target=_new>
```

```
  <input type=hidden name=f1 value=\"$fmas\">
  <input type=submit value='Graph It!'>
</form>";
```

Побудова графіку залежності обсягу словника унікальних слів від розміру тексту здійснюється за допомогою графічної бібліотеки GD, використання якої вказується у тілі програми у явному вигляді (***use GD::Graph::lines;***). При відображенні графічної інформації заголовок HTML-файлу містить не звичайне ***Content-type: text/html***, а ***Content-type: image/png***, що вказується за допомогою окремого рядка програми (***print($q->header(-type=>'image/png'));***). Нижче наведено текст програми побудови графіку.

```
#!/usr/local/bin/perl -w
use CGI;
use GD::Graph::lines;
my $q = new CGI;
my $graph = GD::Graph::lines->new(800, 600);
my $gformat = $graph->export_format;
print($q->header(-type=>'image/png'));

$b=$q->param('f1');  # рядок параметрів
@a=split(" ",$b);    # перетворення рядка в масив
$n=$#a+1;
$m=-32000;           # ініціалізація максимального значення
for ($i=0; $i<$n; $i++) {
 if ($a[$i]>$m) {$m=$a[$i];}
 $data[0]->[$i]=$i*500; # ось абсцис
 $data[1]->[$i]=$a[$i]; # ось ординат - значення
}
$m=$m*1.01;          # найбільше значення ординат
$graph->set(         # формування «обкладинки» графіку
      x_label        => 'Word Number',
      y_label        => 'Unq Word Number',
      title          => 'Heaps law graph',
      y_max_value    => $m,
      y_tick_number  => 8,
      x_label_skip   => 1,
      y_label_skip   => 2
) or die $graph->error;

# виведення графіку
my $gd = $graph->plot(\@data)->png or die $graph->error;
print $gd;
```

У результаті роботи програми отримується графік, подібний до наведеного на рис. 7.1.

Підтвердження закону Хіпса та розрахунок відповідних коефіцієнтів здійснюється за методикою, подібною до описаної у розділі 6.

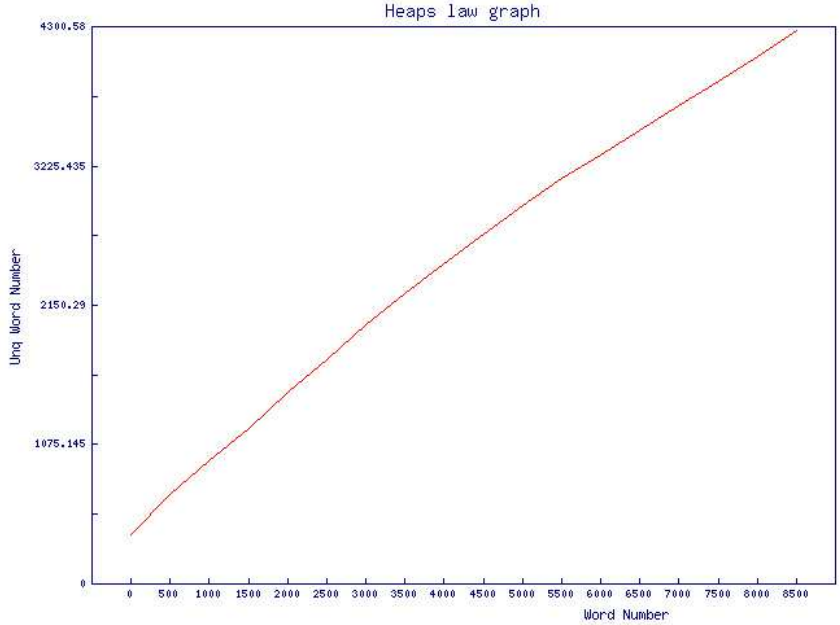


Рис. 7.1. Графік залежності обсягу словника унікальних слів від розміру тексту

Питання для самостійної роботи:

1. Детально ознайомитися з документацією по модулю GD [9].
2. Самостійно побудувати графік залежності обсягу словника унікальних слів від розміру тексту для будь-якого оповідання, написаного українською мовою.
3. За допомогою режиму «Побудова лінії тренду» пакету Microsoft Excel отримати степеневу апроксимацію, а також визначити параметри рівняння Хіпса.

8. Послідовний пошук у сортованому словнику документа

Алгоритм послідовного пошуку за запитом, що складається з одного слова, у сортованому словнику документа передбачає послідовне проходження словника з одночасним порівнянням поточного слова із словника з запитом. Якщо знайдено збіг, то процес пошуку завершається успішно. Якщо ж значення кодів слова зі словника перевищує значення кодів слова із запиту, то процес пошуку переривається з неуспішним результатом.

Нижче наведено фрагмент програми послідовного пошуку у сортованому словнику документа. Передбачається що запит знаходиться у змінній ***$query*** у тесті програми. Для побудови та сортування словника у наведеному фрагменті застосовується підпрограма ***dict_build()***, яку було наведено у попередніх розділах.

```
$query="ПРОГРАМА";
```

```
$doc="";
$doc2="";
$j=0;
open File, "./ukr.txt";
while (<File>)
{
 chomp();
 $buf=$_;

 if ($buf=~m/^\*\*\*/) {
   $j++;
   if ($j>1) {
     dict_build();
     $n=$#w;

     $pp=0;   # ознака результату пошуку -  за замовченням 0

     for ($i=1; $i<=$n; $i++) {
       if ($w[$i] eq $query) { $pp=1; last; }
       if ($w[$i] gt $query) { last; }
     }
     if($pp!=0) {
        # виведення релевантного документа
        print "\n**** $j\n"; print $doc2;
     }
    }
    $doc="";
    $doc2="";
    @u={};
    @nw={};
 }
 else {
     $doc=$doc." ".$buf;    # для побудови словника
     $doc2=$doc2."\n".$buf; # для виведення
 }
}
close File;
```

Наведений алгоритм можна удосконалити, реалізувавши пошук не тільки за цілим словом, але й за його частиною, застосування чого маркується знаком «*****» справа (праве скорочення слова). Суть удосконалення зводиться до використання регулярних виразів замість оператора порівняння ***eq***. Нижче наведено відповідний фрагмент програми.

```
$query="ПРОГРАМ*";
if ($query=/^(.*)\*$/){$query=$1;} # виділення частини перед «*»

# застосування регулярних виразів замість оператора порівняння
```

```
for ($i=1; $i<=$n; $i++) {
   if ($w[$i]=~m/^$query/ ){$pp=1; last;}
   if ($w[$i] gt $query) { last; }
}
```

Питання для самостійної роботи:

1. Самостійно написати програму пошуку у сортованому словнику, в якій передбачається пошук як за повним словом, так і за його правим скороченням.

9. Бінарний пошук у словниковому масиві

Відомо, що кількість операцій при послідовному пошуку в сортованому словнику має порядок $O(N/2)$, де N – обсяг словника, з другого боку, при бінарному пошуку він становить $O(\log_2 N)$, що суттєво економніше. Тому на практиці найчастіше застосовується саме бінарний пошук.

Метод бінарного пошуку ще називають методом «половинного ділення». Суть цього методу складається у тому, що слово із запиту порівнюється з серединним елементом масиву словника. Якщо код запиту співпадає з кодом цього елементу, процес пошуку успішно завершується. В іншому випадку змінюються межі масиву словника. Якщо код слова більше за код серединного елементу словника, то цей елемент визначається як нижня границя словника, якщо менше – як верхня. Таким чином здійснюється ітеративне звуження границь пошуку. Коли верхня та нижня границі співпадають, процес пошуку завершується.

Нижче наведено підпрограму бінарного пошуку в сортованому словнику, яка може застосовуватися у складних пошукових програмах, наприклад у програмі пошуку з використанням логічних операторів (див. розділ 11).

```
sub s_word {

     my($query);
     $query=$_[0]; # Запит - параметр внутрішньої процедури
     $a=-1;        # Результат пошуку
     $l=length($query);
     $kni=0;       # Нижня границя
     $kve=$wm;     # Верхня границя = Кількість слів + 1

     beg_s:
     if (($kve - $kni) < 2) {
          if ((substr($w[$kve],0,$l) eq $query ) ||
              (substr($w[$kni],0,$l) eq $query) {     $a=1;     }
          return $a;
     }
```

```
     $kprov=($kni+$kve)/2; # Середина

     if (substr($w[$kprov],0,$l) eq $query) {
          $a=1;
          return $a;
     }

     if (substr($w[$kprov],0,$l) gt $query) {
          $kve=$kprov;
          goto beg_s;
     }

     if (substr($w[$kprov],0,$l) lt $query) {
          $kni=$kprov;
          goto beg_s;
     }
}
```

Питання для самостійної роботи:

2. Самостійно написати програму бінарного пошуку у сортованому словнику, в якій передбачається пошук як за повним словом, так і за його правим скороченням.

10. Пошук та виведення документів за запитом

Для приймання запиту для подальшого здійснення пошуку використовується CGI-технологія. Запит передається програмі, написаній мовою Perl. Нижче наведено фрагмент HTML-файлу запуску пошукової процедури.

```
<form action="poshuk.pl">
  <input type=text name=s>
  <input type=submit value="Пошук">
  <input type=reset value="Очистити поле">
</form>
```

Цей фрагмент забезпечить можливість введення запиту та подальшої передачі його програмі ***poshuk.pl***. За допомогою веб-браузера користувач отримає наступну форму (рис. 10. 1):

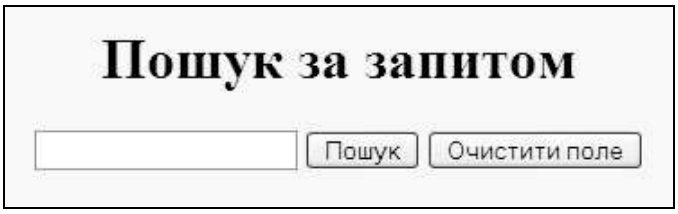


Рис. 10.1. Форма введення запиту

Для пошуку має використовуватися програма, основна частина якої (пошуковий механізм) детально описана у попередньому розділі. Ця програма доповнена фрагментом приймання запиту як параметру:

```
use CGI;
my $q = new CGI;
$query=$q->param('f1');
```

Крім того, текст запиту має бути переведений до верхнього регістру (для забезпечення коректного порівняння зі словником документа), а також з нього мають бути видалені спеціальні символи:

```
$query=~tr/[\:-\@]/ /;
$query=~tr/,;:\(\)\?[]!\"\-\.\/\t/ /;
$query=~tr/0-9/          /;
$query=~tr/a-z/A-Z/;
$query=~tr/ієї/ІЄЇ/;
$query=~tr/[ю-ъ]/[Ю-Ъ]/;
$query=~s/\s+/ /g;
```

Документи, що відповідають запиту (тобто релевантні документи) мають виводитися для наочного відображення за допомогою веб-браузера. Для цього має застосовуватися фрагмент програми, в якому враховується структура документа, детально описана у розділі 4. В результаті наведених нижче засобів форматування, користувач має побачити як результат пошуку перелік документів, що мають вигляд, подібний до наведеного на рис. 10.2.

```
$b_o=""; # мітка поля документа
while ($_=<STDIN>) {
  chomp();
  $buf=$_;
  $b=$buf;
  if  ($buf=~/^\*\*\*/) {
      $i++;
    print "<p><font size=4><b>*** $i</b></font>\n";
  }
  if  ($buf=~/^010/) {  # поле назви
    $buf=substr($buf,10);
    if ($b_o eq "010") { print " "; }
```

```
    else { print "<br>"; }
    print "<font size=4><b>$buf</b></font>\n";
  }
  if  ($buf=~/^021/) {   # поле анотації
    $buf=substr($buf,10);
    if (($b_o ne "021") || ($buf=~/^\s/)) {
      print "<p><font size=2>$buf</font> \n";
    }
    else {
      print "<font size=2>$buf</font> \n";
    }
  }
  if  ($buf=~/^020/) {  # поле основного тексту
    $buf=substr($buf,10);
      $buf=~s/\s+$//g;
    if (($b_o ne "020") || ($buf=~/^\s/) || (length($buf)<2)) {
      print "<p><font size=1>$buf</font> \n";
    }
    else {
      print "<font size=1>$buf</font> \n"
    }
  }
 if  ($buf=~/^040/ || $buf=~/^030/ || $buf=~/^050/) {
    # поля джеререла, дати, URL джерела
    $buf=substr($buf,10);
    print "<font size=2>$buf</font>\n"
  }
  $b_o=substr($b,0,3);
}
```

Питання для самостійної роботи:

1. Самостійно написати програму форматного виводу списку результатів пошуку, які складаються лише з заголовків, анотацій та назв інформаційних джерел.

11. Пошук з використанням логічних операторів

Алгоритм пошуку з використанням логічних операторів розглянемо для випадку застосування у запиті слів, які згруповані у кон’юнктивну нормальну форму. Передбачається використання тільки двох операторів – диз’юнкції (**АБО** - «|») та кон’юнкції (**ТА** – «&»). Передбачається також, що рядок запиту завершується спеціальним символом – «#».

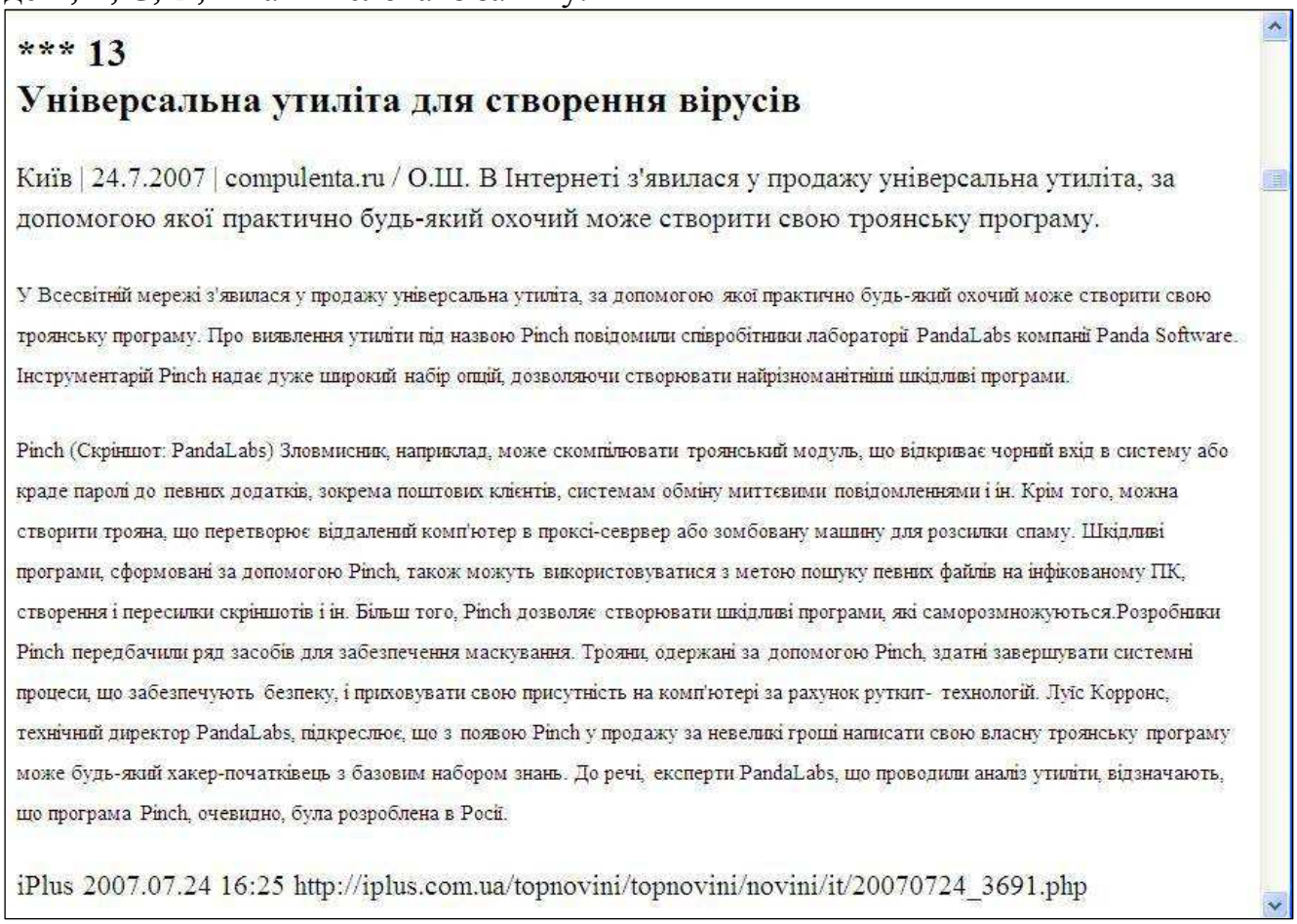
Кон’юнктивна нормальна форма запиту передбачає, що запиту, що має, наприклад, таку форму

A | B | C & D | E & F #

логічно еквівалентний наступному:

(A | B | C) & (D | E) & (F),

де *A, B, C, D, E* та *F* – слова із запиту.



***** 13**

Універсальна утиліта для створення вірусів

Київ | 24.7.2007 | compulenta.ru / О.Ш. В Інтернеті з'явилася у продажу універсальна утиліта, за допомогою якої практично будь-який охочий може створити свою троянську програму.

У Всесвітній мережі з'явилася у продажу універсальна утиліта, за допомогою якої практично будь-який охочий може створити свою троянську програму. Про виявлення утиліти під назвою Pinch повідомили співробітники лабораторії PandaLabs компанії Panda Software. Інструментарій Pinch надає дуже широкий набір опцій, дозволяючи створювати найрізноманітніші шкідливі програми.

Pinch (Скріншот: PandaLabs) Зловмисник, наприклад, може скомпілювати троянський модуль, що відкриває чорний вхід в систему або краде паролі до певних додатків, зокрема поштових клієнтів, системам обміну миттєвими повідомленнями і ін. Крім того, можна створити троянa, що перетворює віддалений комп'ютер в проксі-севрвер або зомбовану машину для розсилки спаму. Шкідливі програми, сформовані за допомогою Pinch, також можуть використовуватися з метою пошуку певних файлів на інфікованому ПК, створення і пересилки скріншотів і ін. Більш того, Pinch дозволяє створювати шкідливі програми, які саморозмножуються.Розробники Pinch передбачили ряд засобів для забезпечення маскування. Трояни, одержані за допомогою Pinch, здатні завершувати системні процеси, що забезпечують безпеку, і приховувати свою присутність на комп'ютері за рахунок руткит- технологій. Луїс Корронс, технічний директор PandaLabs, підкреслює, що з появою Pinch у продажу за невеликі гроші написати свою власну троянську програму може будь-який хакер-початківець з базовим набором знань. До речі, експерти PandaLabs, що проводили аналіз утиліти, відзначають, що програма Pinch, очевидно, була розроблена в Росії.

iPlus 2007.07.24 16:25 http://iplus.com.ua/topnovini/topnovini/novini/it/20070724_3691.php

Рис. 10. 2. Приклад відображення релевантного документа

При створенні програми пошуку з використанням логічних операторів будемо використовувати підпрограму бінарного пошуку у словнику ***s_word()***, описану у розділі 9.

Блок пошуку з використанням логічних операторів, наведений нижче, реалізовано також у вигляді підпрограми.

```
sub search_engine () {
     my($i);
     $ko=0;       # Номер поточного терма у запиті
     $res_qw=0;   # Результат пошуку

     beg:
     if (s_word($q[$ko]) == -1) {
          if (($o[$ko] eq '#') || ($o[$ko] eq '&')) {
               $res_qw=0;
```

```
                goto wyhod;
           }
           else {
                $ko++;
                goto beg;
           }
      }
      else {
           if ($o[$ko] eq '|') {
                koplus:
                $ko++;
                if ($o[$ko] eq '|') {
                     goto koplus;
                }
                $res_qw=1;
                if ($o[$ko] eq '#') {
                     $res_qw=1;
                     goto wyhod;
                }
                if ($o[$hhh] eq '&') {
                     $res_qw=0;
                     $ko++;
                }
                goto beg;
           }
           if ($o[$ko] eq '#') {
                $res_qw=1;
                goto wyhod;
           }
           if ($o[$ko] eq '&') {
                $res_qw=0;
                $ko++;
                goto beg;
           }
      }

      wyhod:
      if ($res_qw == 1) {
           open STDOUT,">>res.txt";
           print "\n***",$query,"\n",$doc;
           close STDOUT;
      }
}
```

Питання для самостійної роботи:

1. Розробити алгоритм та написати програму пошуку з використанням ще одного логічного оператора ТАК-НІ («!»).

12. Середовище програмування клітинних автоматів

Клітинні автомати є корисними дискретними моделями для дослідження динамічних систем. Моделювання багатьох еволюційних та інформаційних процесів може здійснюватися за допомогою так званих клітинних автоматів. Головною перевагою клітинних автоматів є їх абсолютна сумісність із алгоритмічними методами рішення завдань. Скінчений набір формальних правил, заданий на скінченій множині елементів (кліток), допускає точну реалізацію у вигляді алгоритмів.

Клітинний автомат являє собою дискретну динамічну систему, сукупність однакових кліток, певним чином з'єднаних між собою. Всі клітки утворять мережу (решітку) клітинних автоматів. Стан кожної клітки визначаються станом кліток, що входять у її локальний окіл та називаються найближчими сусідами. Тобто околом кінцевого автомата з номером j називається множина його найближчих сусідів. Стан j-го клітинного автомата в момент часу $t + 1$, таким чином, визначається в такий спосіб:

$$y_j(t+1) = F(y_j(t),O(j), t),$$

де F – деяке правило, яке можна виразити, наприклад, мовою булевої алгебри. У багатьох завданнях, вважається, що сам елемент відноситься до своїх найближчих сусідів, тобто $y_j \in O(j)$, у цьому випадку формула спрощується:

$$y_j(t+1) = F(O(j), t).$$

Клітинні автомати у традиційному розумінні задовольняють таким правилам:

- зміна значень всіх кліток відбувається одночасно (одиниця виміру - такт);
- мережа клітинних автоматів однорідна, тобто правила зміни станів для всіх кліток однакові;
- на клітку можуть вплинути лише клітки з його локального околу;
- множина станів клітки скінченна.

Для усунення крайових ефектів скінченні решітки топологічно «згортаються в тор» (рис. 12.1), тобто перший рядок вважається продовженням останнього, а останній - попередником першого. Те ж саме відноситься й до стовпців.

Для проектування інтерфейсу відображення системи клітинних автоматів у рамках даного курсу будуть використовуватися засоби роботи з таблицями в HTML-файлах, а також найпростіші з каскадними стилями (CSS).

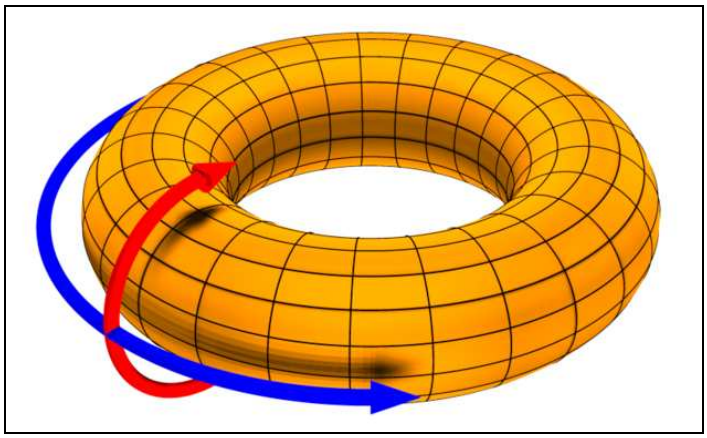


Рис. 12.1. Згортання площини в тор

Нижче наведено фрагмент тестового HTML-файлу, за допомогою якого формується таблиця, що складається з однієї комірки.

```
<body>
     <style type="text/css">
          td.r{font-size:4pt;cellpdding}
     </style>
</body>
<table border=1>
     <tr>
          <td class=r bgcolor=white>
               <font color=white>---
          </td>
     </tr>
</table>
```

Для подальшої реалізації систем клітинних автоматів у веб-середовищі будуть застосовуватися засоби CGI та мови Perl. Нижче наведено фрагмент програми, що формує таблицю розміром 30x30 (рис. 12.2) із застосуванням наведених вище засобів форматування.

```
$N=30;
for ($i=1; $i<=$N; $i++) {
  for ($j=1; $j<=$N; $j++) {
      $z[$i][$j]=0; # ініціалізація значень клітинок
  }
}
$i=int($N/2); $j=$i;
$z[$i][$j]=1;  # вибрана клітинка чорного кольору

print "<html>
<title>sun</title>
<body><style type=\"text/css\">
     td.r{font-size:4pt;cellpdding}
```

```
     </style></body>
<table border=1>";  # початок таблиці

for ($x=1; $x<=$N; $x++) {
  print "<tr>\n";   # виведення нового рядка
  for ($y=1; $y<=$N; $y++) {
     if ($z[$x][$y]==0) { $c="white"; }
     if ($z[$x][$y]==1) { $c="black"; }
     if ($z[$x][$y]==2) { $c="gray"; }
     print "<td class=r bgcolor=$c>
     <font color=$c>---</td>";
  }
  print "</tr>";
}
print "</table>";  # кінець таблиці
```

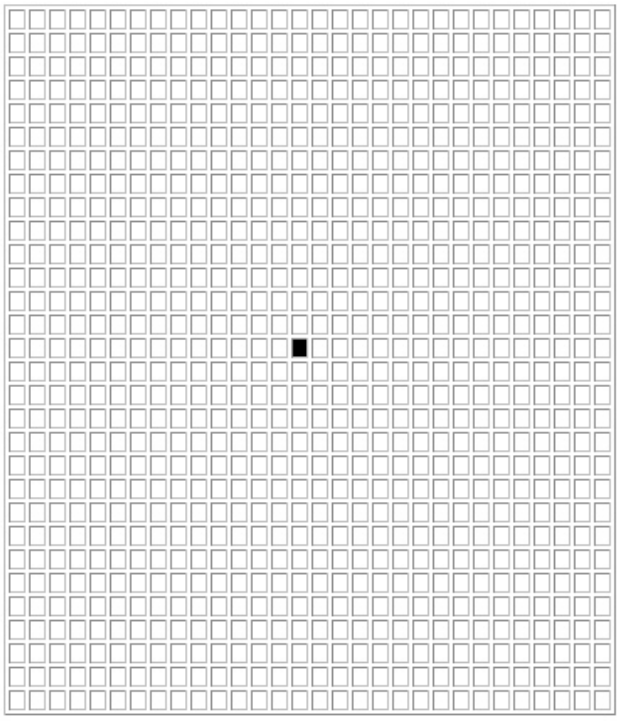


Рис. 12.1. Відображення середовища клітинних автоматів

Питання для самостійної роботи:

1. Детально ознайомитися з методами виведення таблиць у мові HTML [4] та з елементами CSS.
2. Ознайомитись з основами теорії клітинних автоматів, використовуючи матеріали лекцій та рекомендованих інформаційних джерел [11, 14, 16].
3. Розробити програму виведення на поле системи клітинних автоматів заданої заздалегідь конфігурації.

13. Моделювання еволюційних процесів

Серед клітинних автоматів відомі такі, динаміка яких істотно залежить від початкового стану. Підбираючи різні початкові стани, можна одержувати найрізноманітніші конфігурації та типи поводження. Саме до таких систем ставиться класичний приклад - гра «Життя», винайдена Дж. Конвеєм .

З метою отримання навичок моделювання еволюційних процесів розглянемо реалізацію гри «Життя» у середовищі, розгляд якого було розпочато у попередньому розділі.

Правило варіанту гри «Життя», моделюванню якого присвячено цій розділ, такі. Клітина знаходиться у одному з двох станів – живому та неживому (чорному та білому). Якщо сусідами клітини є менше двох або більше трьох чорних клітин, то на наступному кроці вона зафарбовується у білий колір (вмирає). Якщо сусідами клітини є рівно три чорних клітини, то на наступному кроці вона зафарбовується у чорний колір (народжується).

У програмі вводиться масив кількості сусідів для кожної клітини (***@nd***, ***$nd[0]*** – кількість «білих» сусідів, ***$nd[1]*** – кількість «чорних» сусідів). У цих позначеннях правила наведеного варіанту гри «Життя» мають вигляд:

```
if (($nd[1]<2) || ($nd[1]>3)) { # загибель
     $y[$i][$j]=0;
}
if ($nd[1]==3){ # народження
     $y[$i][$j]=1;
}
```

Нижче наведено текст програми реалізації гри «Життя» у веб-середовищі для початкової конфігурації у вигляді хреста на решітці розміром 14x14. У програмі забезпечено її рекурентний виклик. Результати роботи програми наведено на рис. 13.1.

```
#!/usr/bin/perl -w
#  Середовище клітинних автоматів. Гра «Життя»
print "Content-type: text/html\n\n";
$gen=$ENV{"QUERY_STRING"};
if ($gen=~m/gen=(\d+)x(\d+)x(\d+)$/) {
     $gen=$1;    # покоління
     $gen++;
     $N=$2;      # розмір таблиці
     $table=$3;  # значення клітинок таблиці
}
else {
     # Якщо параметри не передалися - здійснюється
     # початкова фаза - ініціалізація таблиці
     $gen=0;     # покоління
     $N=14;      # розмір таблиці
```

```
     $table="";
     for ($i=1; $i<=$N**2; $i++) {
          $a[$i]=0;
     }
     # Первинна конфігурація - хрест
     $i=$N*(1+$N)/2; $a[$i]=1;
     $i=$N*(1+$N)/2+1; $a[$i]=1;
     $i=$N*(1+$N)/2-1; $a[$i]=1;
     $i=$N*(1+$N)/2-$N; $a[$i]=1;
     $i=$N*(1+$N)/2+$N; $a[$i]=1;
     for ($i=1; $i<=$N**2; $i++) {
          $table=$table.$a[$i];
     }
}
# Заповнення масиву комірок
for ($i=1; $i<=$N; $i++) {
     for ($j=1; $j<=$N; $j++) {
          $y[$i][$j]=substr($table,0,1);
          $table=substr($table,1);
     }
}
# Масив @z моделює згортання площини у тор
# шляхом додавання 0 та N+1 рядків та стовпчиків
if ($gen>0) {
for ($i=0; $i<=$N+1; $i++) {
     for ($j=0; $j<=$N+1; $j++) {
          $z[$i][$j]=0;
     }
}
for ($i=1; $i<=$N; $i++) {
     for ($j=1; $j<=$N; $j++) {
          $z[$i][$j]=$y[$i][$j];
     }
}
for ($i=1; $i<=$N; $i++) {
          $z[$i][$N+1]=$y[$i][1];
}
for ($i=1; $i<=$N; $i++) {
          $z[$i][0]=$y[$i][$N];
}
for ($j=1; $j<=$N; $j++) {
          $z[$N+1][$j]=$y[1][$j];
}
for ($j=1; $j<=$N; $j++) {
          $z[0][$j]=$y[$N][$j];
}
for ($i=1; $i<=$N; $i++) {
     for ($j=1; $j<=$N; $j++) {
          # підрахунок кількості сусідів різних кольорів
          $nd[0]=0; $nd[1]=0;
          $nd[$z[$i-1][$j-1]]++;
```

```
          $nd[$z[$i-1][$j]]++;
          $nd[$z[$i-1][$j+1]]++;
          $nd[$z[$i][$j-1]]++;
          $nd[$z[$i][$j+1]]++;
          $nd[$z[$i+1][$j-1]]++;
          $nd[$z[$i+1][$j]]++;
          $nd[$z[$i+1][$j+1]]++;
          # Правила гри «Життя»
          if (($nd[1]<2) || ($nd[1]>3)) { # загибель
               $y[$i][$j]=0;
          }
          if ($nd[1]==3){ # народження
               $y[$i][$j]=1;
          }
     }
  }
}
# Відображення
print "<html><bgcolor bgcolor=lightgray>";
print "<style type=\"text/css\">
         td.r {font-size: 4pt; cellpadding:0; cellspacing:0}
       </style>
       <table border=1><tr><td><table border=1>";
for ($i=1; $i<=$N; $i++) {
     print "\n<tr class=r>\n";
     for ($j=1; $j<=$N; $j++) {
          if ($y[$i][$j]==0) {$b=white;}
          if ($y[$i][$j]==1) {$b=blue;}
          print "<td class=r bgcolor=$b>
                 <font color=$b>---</td>";
     }
     print "\n<\/tr>\n";
}
print "<\/table><\/td><\/tr><tr><td>";
print "<br><center>Generation $gen<br>";
# Підготовка параметрів для виклику наступного кроку
# еволюції та відображення форми
$nd[0]=0; $nd[1]=0;
$tc=$gen."x".$N."x";
for ($i=1; $i<=$N; $i++) {
     for ($j=1; $j<=$N; $j++) {
          $tc=$tc.$y[$i][$j];
          $nd[$y[$i][$j]]++;
     }
}
print "<form action='conway.pl'>
            <input type=submit value='NEXT'>
            <input type=hidden name=gen value='$tc'>
       </form>
Black - $nd[1]; White - $nd[0].
<br><a href='conway.pl'>New Population</a>
```

```
</center></td></tr></tables><\/body><\/html>";
```

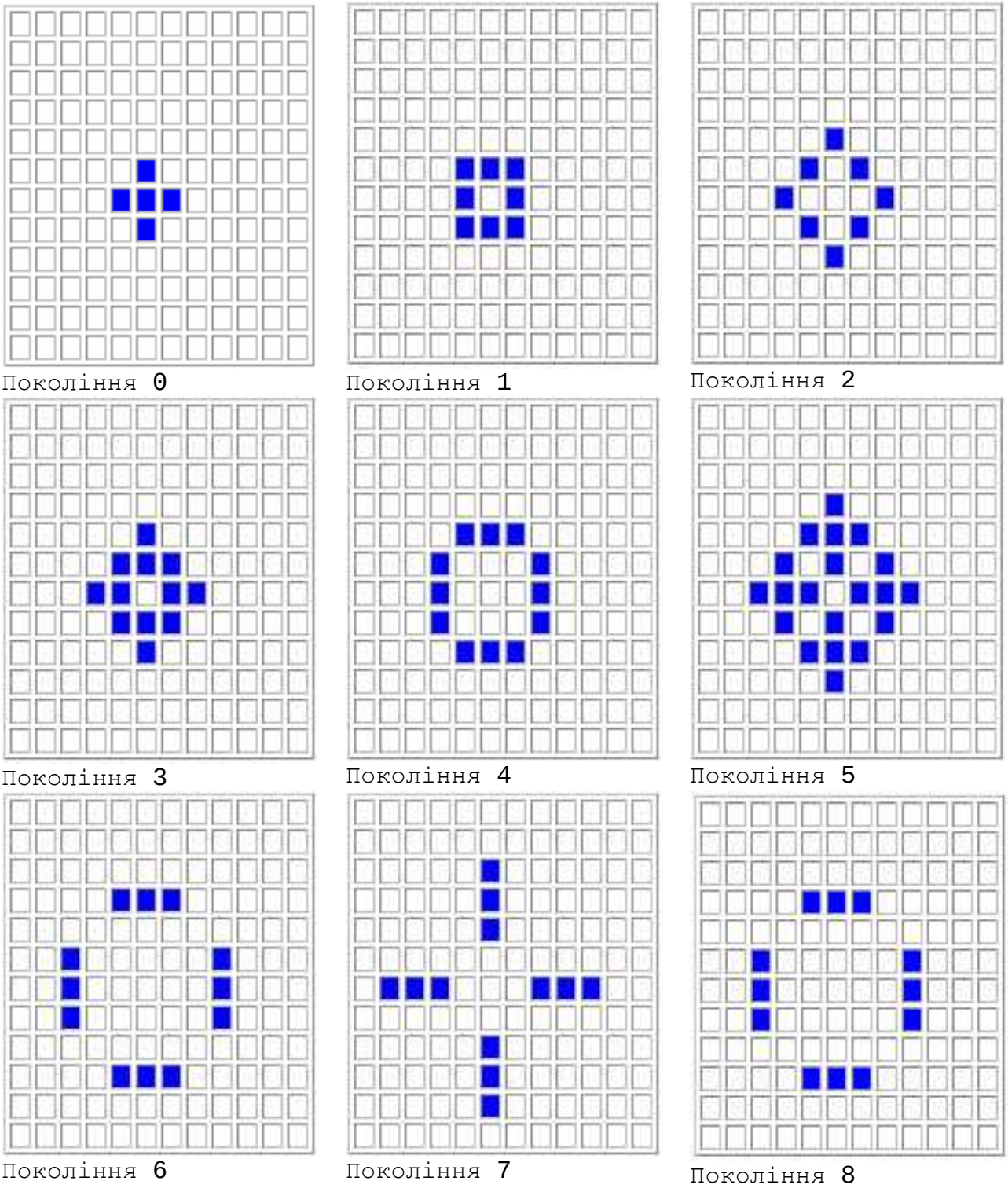


Рис. 13.1. Покоління гри «Життя» з конфігурацією «хрест»

Питання для самостійної роботи:

4. Побудувати графік залежності кількості білих та чорних клітин від кроку еволюції.

5. Створити власні «правила еволюції», додати ще один стан, що буде відображатися окремим кольором (наприклад, жовтим) та програмно реалізувати нову систему клітинних автоматів.

14. Модель дифузії інформації

Для моделювання процесів розповсюдження (дифузії) інформації в рамках даного курсу також використовується метод клітинних автоматів. Як спрощену модель дифузії інформації спочатку розглянемо спрощену модель розповсюдження інновацій. Подібна модель функціонує за наступними правилами: кожен індивід, здатний прийняти інновацію, відповідає одній клітині, на двовимірній площині. Кожна клітина може перебувати у двох станах: 1 - новинка прийнята; 0 - новинка не прийнята. Передбачається, що автомат, сприйнявши інновацію один раз, запам'ятовує її назавжди (стан 1 – не може бути зміненим). Автомат приймає рішення щодо прийнятті новинки, орієнтуючись на думку восьми найближчих сусідів, тобто якщо в околі даної клітки є m прихильників новинки, p - імовірність її прийняття (генерується у ході роботи моделі) і якщо $pm > R$, (R - фіксоване граничне значення), то клітина приймає інновацію.

Динаміці поширення інформації властиві деякі додаткові властивості, розглядалася розширена модель, що відноситься до поширення новин в інформаційному просторі. Додатково передбачається, що клітка може бути в одному з трьох станів: 1 - «свіжа новина» (клітка офарблюється в чорний колір); 2 - новина, що застаріла, але збережена у вигляді відомостей (сіра клітка); 3 - клітка не має інформації, переданої новинним повідомленням (клітка біла, інформація не дійшла або вже забута). Правила поширення новин наступні:

- спочатку все поле складається з білих кліток за винятком однієї, чорної, котра першої «прийняла» новину (рис. 3);
- біла клітка може перефарбовуватися тільки в чорні кольори або залишатися білою (вона може одержувати новину або залишатися «у невіданні»);
- біла клітка перефарбовується, якщо виконується умова: $pm > 1$ (якщо $m > 2$) або $1.5.pm > 1$ (якщо $m < 3$) ;
- якщо клітка чорна, а навколо неї винятково чорні та сірі, то вона перефарбовується в сірі кольори (новина застаріває, але зберігається як відомості);
- якщо клітка сіра, а навколо її винятково сірі та чорні, то вона перефарбовується в білі кольори (відбувається старіння новини при її загальновідомості).

Описана система клітинних автомат цілком реалістично відбиває процес поширення новин серед окремих інформаційних джерел та їх публікацій. На полі розміром 40 х 40 (розміри обрані винятково з метою наочності) стан

системи клітинних автоматів повністю стабілізується за обмежену кількість кроків (рис. 14.1).

Програма моделювання дифузії інформації, яка передбачає рекурентний запуск, подібна за структурою до наведеної у розділі 13. Відмінності стосуються лише початкової ініціалізації (тут вона аналогічна наведеній у розділі 12), розміру масиву ***@nd*** (у даному випадку вона дорівнює трьом) та правил зміни стану клітинних автоматів. Наведемо відповідні фрагменти.

```
# Ініціалізація масиву комірок:

for ($i=1; $i<=$N**2; $i++) {
     $a[$i]=0;
}
$i=$N*(1+$N)/2; $a[$i]=1;
for ($i=1; $i<=$N**2; $i++) {
     $table=$table.$a[$i];
}

. . .

# Правила:

$p=rand($N)/$N;
if ($y[$i][$j]==0) { # біла
     if ($nd[1]<3) { $p=$p*1.5; }
     $x=$nd[1]*$p;
     if ($x>1) { $y[$i][$j]=1; }
     goto vse;
}
if ($y[$i][$j]==1) { # чорна
     if ($nd[1]+$nd[2]==8) { $y[$i][$j]=2; }
     goto vse;
}
if ($y[$i][$j]==2) { # сіра
     if ($nd[2]+$nd[1]==8) { $y[$i][$j]=0; }
}
vse:;

. . .

# Відображення

for ($i=1; $i<=$N; $i++) {
     print "\n<tr class=r>\n";
     for ($j=1; $j<=$N; $j++) {
          if ($y[$i][$j]==0) {$b=white;}
          if ($y[$i][$j]==1) {$b=black;}
          if ($y[$i][$j]==2) {$b=gray;}
          print "<td class=r bgcolor=$b>
               <font color=$b>---</td>";
```

```
        }
        print "\n<\/tr>\n";
    }
```

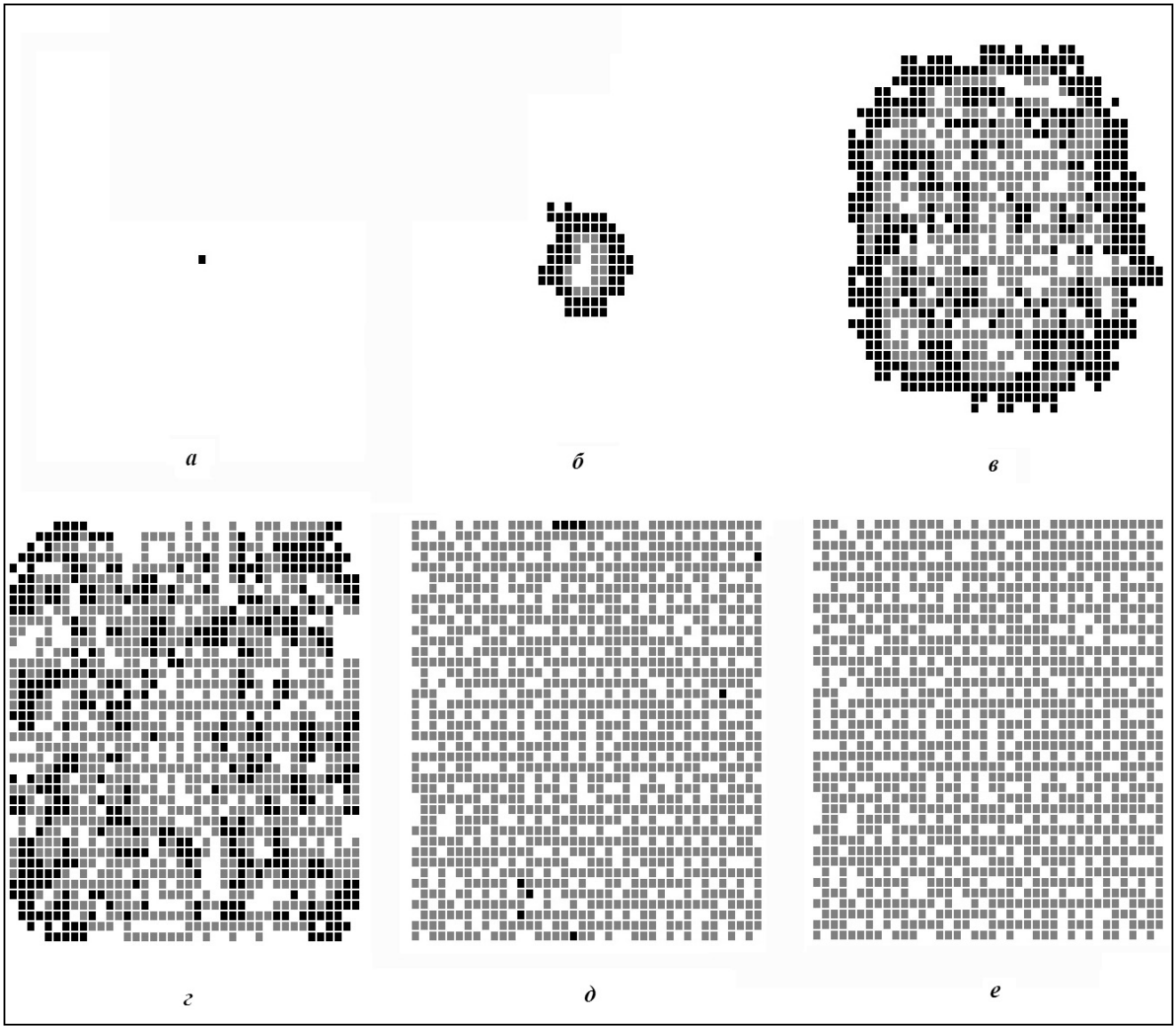


Рис. 54. Процес еволюції системи клітинних автоматів «дифузії новин»: а – вихідний стан; б-д – проміжні стани; е – кінцевий стан

Питання для самостійної роботи:

1. Створити програму автоматичного виведення на екран всіх станів системи клітинних автоматів «дифузії новин» (до настання збігу).

15. Отримання параметрів моделі дифузії інформації

Експерименти з системою клітинних автоматів, що моделюють дифузію інформації на полі розміром 40x40 показують, що період збіжності цієї системи становить від 80 до 150 ітерацій. Типові залежності кількості кліток,

що перебувають у різних станах залежно від кроку ітерації наведені на рис. 14.1. Природно, що сумарна кількість клітин, що перебувають у всіх трьох станах на кожному кроці ітерації, постійна та дорівнює розміру поля.

Для виявлення закономірностей дифузії інформації необхідно дослідити розподіли сірих, білих і чорних клітин на кожному кроці. Графік зміни кількості чорних клітин в залежності від кроку ітерації можна вважати динамікою розподілу новин.

Для відображення відповідних графіків необхідно отримати відповідні числові данні, для чого замінити ітеративний ручний виклик наступного такту системи клітинних автоматів – автоматичним.

Відповідно до цього зникає потреба в передаванні даних через механізм CGI та виникає необхідність формування зовнішнього циклу, з якого необхідно вивести програму у разі збіжності системи клітинних автоматів. Крім того відпадає необхідність візуалізації решітки клітинних автоматів, але треба виводити дані щодо кількості клітин різного кольору на всіх кроках функціонування системи клітинних автоматів.

Крім того, необхідно забезпечити розробку та виклик програми побудови відповідних графіків (рис. 15.1).

Нижче наведено фрагмент програми отримання даних щодо кількості клітин.

```
$fmas="";  # Ініціалізація рядку параметрів для побудови графіка
# Головний цикл виклику системи клітинних автоматів
for ($i2=1; $i2<180; $i2++) {

. . .

# Повторення основного фрагменту програми з розділу 14

. . .

     # Заповнення рядка параметрів для побудови графіка
     $fmas=$fmas." ".$nd[1]." ".$nd[2]." ".$nd[0];\

     # Умова виходу з циклу  - відсутність чорних клітин
     if ($nd[1]==0) {last;}
}

# Виклик програми побудови графіку,
# анлогічній наведеній у розділі 7

print "<form action='id.pl' method=post target=_new>
          <input type=hidden name=f1 value=\"$fmas\">
          <input type=submit value='Graph It!'>
     </form></body></html>";
```

Програма побудови графіку із застосуванням модулю GD:

```
#!/usr/local/bin/perl -w
use CGI;
use GD::Graph::lines;
my $q = new CGI;
my $graph = GD::Graph::lines->new(520, 440);
my $gformat = $graph->export_format;
print($q->header(-type=>'image/png'));
$b=$q->param('f1');
$b="0 ".$b;
@a=split(" ",$b);
$n=$#a+1;
$m=-32000;
$j=1;
$i=1;
while ($i<$n) {
  $b[$j]=$a[$i];  # $nd[1] - чорні
  $i++;
  $c[$j]=$a[$i];  # $nd[2] - сірі
  $i++;
  $d[$j]=$a[$i];  # $nd[0] - білі
  $i++;
  $j++;
}
for ($i=0; $i<$n/3+1; $i++) {
  $data[0]->[$i]=$i;
  $data[1]->[$i]=$b[$i]; # $data[1] - чорні
  $data[2]->[$i]=$c[$i]; # $data[2] - сірі
  $data[3]->[$i]=$d[$i]; # $data[3] - білі
}
$m=1600; # максимальна кількість клітин

$n1=int($n/10)-1; # крок маркування по вісі абсцис
$graph->set(
x_label           => 'Step',
y_label           => 'Number',
title             => 'Information Dynamic',
y_max_value       => $m,
y_tick_number     => 8,
x_label_skip      => $n1,
y_label_skip      => 2
) or die $graph->error;
my $gd = $graph->plot(\@data)->png or die $graph->error;
print $gd;
```

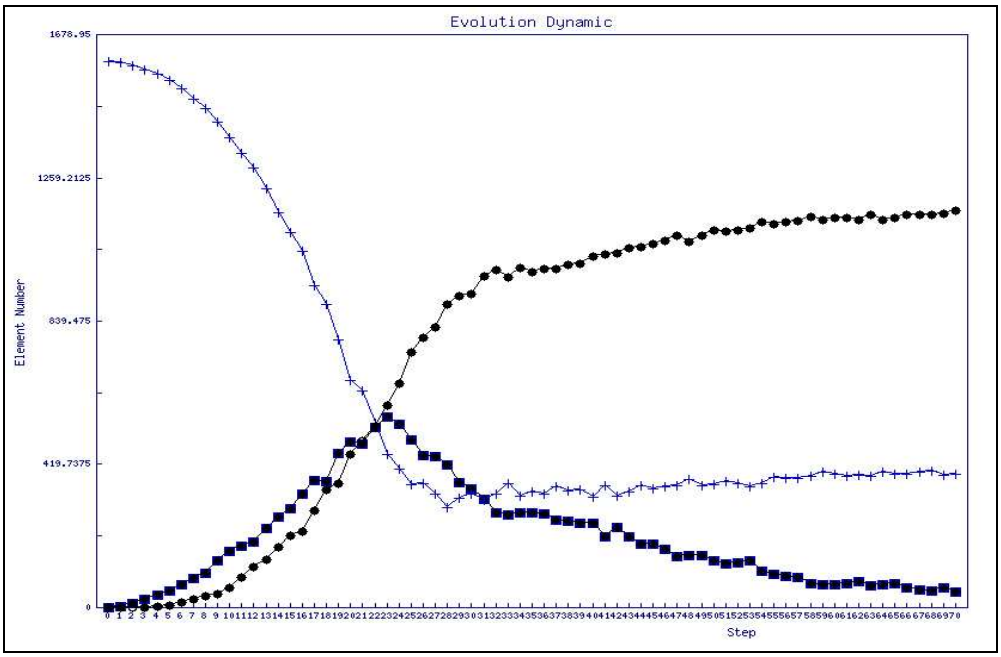


Рис. 15.1. Кількість клітин кожного з кольорів у залежності від кроку еволюції: білі клітки - (+); сірі клітки - (•); *чорні клітки –* (■)

Питання для самостійної роботи:

1. Ознайомитись з аналітичною моделлю еволюції системи клітинних автоматів «дифузії новин» [1, 16].

16. Розрахунок автокореляційної функції

Для дослідження часових рядів інтенсивності публікацій за визначеними темами застосовуються методи статистичного аналізу, у тому числі розрахунок автокореляційної функції. Вигляд цієї функції дозволяє робити висновки щодо властивостей самоподібності вихідного ряду, дозволяє виявляти періодичні складові.

Якщо позначити через X_t член ряду кількості публікацій (кількості електронних повідомлень, що надійшли, наприклад, у день t, $t = 1, \ldots, N$), то функція автокореляції для цього ряду X визначається як:

$$F(k) = \frac{1}{N-k} \sum_{t=1}^{N-k} (X_{k+t} - m)(X_t - m),$$

де m – середнє значення ряду X.

Нижче наведена процедура обчислення автокореляційної функції ряду, що складається з цілих чисел, який міститься у файлі «***dyn.txt***». Після обчислення значень кореляційної функції забезпечується виклик програми побудови відповідного графіку (рис. 16.1), аналогічний до наведеного у розділі 7.

```
#!/usr/bin/perl -w
print "Content-type: text/html\n\n<pre>";

$N=1;
$ksi[0]=0;
$fn="dyn.txt";
open F,"<$fn";

while ($_=<F>) {
     chomp();
     $buf=$_;
     $ksi[$N]=$buf;
     $N++;
}
$sred=0;

for ($i=1; $i<$N; $i++) {
     $sred=$sred+$ksi[$i];
}

$sred=$sred/($N-1);

@x={};
for ($i=1; $i<$N; $i++) {
     $x[$i]=($ksi[$i]-$sred);
}
$fmas="";@y={};

for ($k=0; $k<=$N/4; $k++) {
     # k - крок для розрахунку кореляції

     $y[$k]=0;
     for ($t=1; $t<$N-$k; $t++) {
          $y[$k]=$y[$k]+($x[$t+$k])*($x[$t]);
     }
     $y[$k]=$y[$k]/($N-$k-1);
}
$sig=$y[0];
for ($k=0; $k<=$N/4; $k++) {
     $y[$k]=$y[$k]/$sig;
     printf "%.4f ", $y[$k];
     $fmas=$fmas." ".$y[$k];
}

print "
<form action='correlg.pl' method=post target=_new>
     <input type=hidden name=f1 value=\"$fmas\">
     <input type=submit value='Graph It!'>
</form></body></html>"
```

Програма побудови графіку із застосуванням модулю GD:

```
#!/usr/local/bin/perl -w
use CGI;
use GD::Graph::lines;
my $q = new CGI;

my $graph = GD::Graph::lines->new(520, 440);
my $gformat = $graph->export_format;
print($q->header(-type=>'image/png'));

$b=$q->param('f1');
@a=split(" ",$b);
$n=$#a+1;
$m=-32000;
for ($i=0; $i<$n; $i++) {
  if ($a[$i]>$m) {$m=$a[$i];}
  $data[0]->[$i]=$i;
  $data[1]->[$i]=$a[$i];
}

$m=$m*1.1;
$n1=int($n/10)-1;
$graph->set(
x_label           => 'Number',
y_label           => 'Correllation',
title             => 'Correlation Dynamic',
y_max_value       => $m,
y_tick_number     => 8,
x_label_skip     => $n1,
y_label_skip      => 2
) or die $graph->error;

my $gd = $graph->plot(\@data)->png or die $graph->error;
print $gd;
```

Питання для самостійної роботи:

1. Довідатись, як за допомогою кореляційної функція може бути виявлена періодична складова, що впливає на ряд вимірів [1, 16].
2. Побудувати графік кореляційної функцій засобами Microsoft Excel для k = 0, …, 24. Знайти параметри апроксимації степеневою функцією.

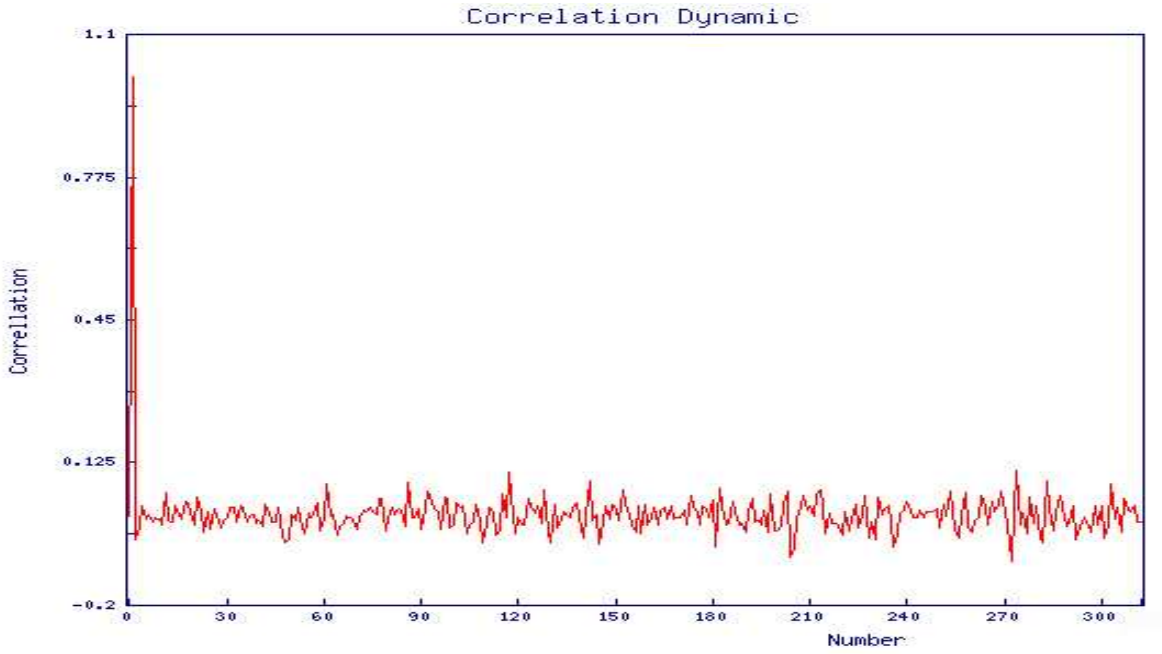


Рис. 16.1. Кореляційна функція

17. R/S-аналіз. Розрахунок показника Херста

Для вивчення фрактальних характеристик часових рядів $F(n)$, $n=1, \ldots, N$, складених зі значень інтенсивності тематичних інформаційних потоків, досліджуються значення показника Херста, який визначається із співвідношення:

$$R(N)/S_N \cong \left(N/2\right)^H, \quad N >> 1.$$

Тут S_N – стандартне відхилення:

$$S_N = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(F(n) - \left\langle F\right\rangle_N\right)^2},$$

$$\left\langle F\right\rangle_N = \frac{1}{N}\sum_{n=1}^{N} F(n),$$

R - так званий розмах:

$$R(N) = \max_{1\le n\le N} X(n,N) - \min_{1\le n\le N} X(n,N),$$

де:

$$X(n,N)=\sum_{i=1}^{n}\left(F(i)-\left\langle F\right\rangle_N\right).$$

Відомо, що показник Херста являє собою міру персистентності - схильності процесу до трендів (на відміну від звичайного броунівського руху). Значення $H>1/2$ означає, що спрямована в певну сторону динаміка процесу в минулому, найімовірніше, спричинить продовження руху у тому ж напрямку. Якщо $H<1/2$, то прогнозується, що процес змінить спрямованість. $H=1/2$ означає невизначеність - броунівський рух.

Якщо розмір ряду вимірів N досить великий, то розрахунок середнього значення і стандартного відхилення при зростанні N за наведеною формулою не є раціональним. Більш економними з обчислювальної точки зору є ітераційні методи. Обгрунтуємо формулу розрахунку середнього $\left\langle F\right\rangle_{N+1}$, якщо відомо $N,\ \ F(N+1),\ \ \left\langle F\right\rangle_N$:

$$\left\langle F\right\rangle_{N+1}=\frac{1}{N+1}\sum_{n=1}^{N+1}F(n)=\frac{1}{N+1}\left(\sum_{n=1}^{N}F(n)+F(N+1)\right)=\frac{1}{N+1}\left(N\left\langle F\right\rangle_N+F(N+1)\right).$$

Формула для обчислення стандартного відхилення S_{N+1} на основі значень $N,\ \ F(N),\ \ F(N+1),\ \ \left\langle F\right\rangle_{N+1},\ S_N$ обгрунтовується дещо складніше:

$$S_{N+1}^2=\frac{1}{N+1}\sum_{n=1}^{N+1}\left(F(n)-\left\langle F\right\rangle_{N+1}\right)^2,$$

$$(N+1)S_{N+1}^2=\sum_{n=1}^{N+1}\left(F(n)^2-2F(n)\left\langle F\right\rangle_{N+1}+\left\langle F\right\rangle_{N+1}^2\right)=$$

$$=\sum_{n=1}^{N}F(n)^2+F(N+1)^2-2\left\langle F\right\rangle_{N+1}\sum_{n=1}^{N}F(n)-2\left\langle F\right\rangle_{N+1}F(N+1)+(N+1)\left\langle F\right\rangle_{N+1}^2.$$

Те ж саме для попереднього значення:

$$NS_N^2=\sum_{n=1}^{N}\left(F(n)^2-2F(n)\left\langle F\right\rangle_N+\left\langle F\right\rangle_N^2\right)=\sum_{n=1}^{N}F(n)^2-N\left\langle F\right\rangle_N^2,$$

Звідки:

$$\sum_{n=1}^{N}F(n)^2=NS_N^2+N\left\langle F\right\rangle_N^2.$$

Підставивши це значення у рівняння для S_{N+1}^2, отримуємо:

$$S_{N+1}^2 = \frac{1}{N+1}\left(NS_N^2 + N\left\langle F\right\rangle_N^2 + F(N+1)^2 - (N+1)\left\langle F\right\rangle_{N+1}^2 \right).$$

Наведені формули ітеративного розрахунку будуть використовуватися в програмі, текст якою наведено нижче.

```
#!/usr/bin/perl -w
# Програма розрахунку показника Херста
print "Content-type: text/html\n\n";
$fmas=""; # Рядок для передачі параметрів графічній програмі
$i=1;
$ksi[0]=0;
$abs_sred=0;
$m=0;
$fn="dyn.txt";
open FILE,"<$fn";
while ($_=<FILE>) { # Зчитування файлу вихідних значень
     chomp();
     $buf=$_;
     $ksi[$i]=$buf;
     $i++;
}
$i--;
close FILE;

$ii=$i;
$abs_sred=0;
$disp=0;
for ($m=1;$m<$ii;$m++) {
   $o_sred=$abs_sred;
   $abs_sred=(($m-1)*$abs_sred+$ksi[$m])/$m;
   $disp=sqrt((($m-1)*($disp**2 + $o_sred**2) +
     $ksi[$m]**2 - $m*$abs_sred**2)/$m);

   $max=-10000;
   $min=10000;
   $xtn=0;
   for ($n=1;$n<=$m; $n++) {
     $xtn=$xtn+($ksi[$n]- $abs_sred);
     if ($max<$xtn) {$max=$xtn;}
     if ($min>$xtn) {$min=$xtn;}
   }

   $rnas=0;
   if($disp>0) {$rnas=($max-$min)/$disp;}
   if ($rnas==0) {$rnas=1;}
   $h=0.5;  # початкове значення – стан невизначеності
   if ($m>2) {
     $h=log($rnas)/log($m/2);
```

```
     print "<br>\n";
     print $rnas;
     $fmas=$fmas." ".$h;
   }
}

print "
<form action='hurst-graph.pl' method=post target=_new>
<input type=hidden name=f1 value=\"$fmas\">
<input type=submit value='Graph It!'>
</form></body></html>"
```

Програма побудови графіку із застосуванням модулю GD:

```
#!/usr/local/bin/perl -w
use CGI;
use GD::Graph::lines;
my $q = new CGI;
my $graph = GD::Graph::lines->new(520, 440);
my $gformat = $graph->export_format;
print($q->header(-type=>'image/png'));
$b=$q->param('f1');
$b="0.5 ".$b;
@a=split(" ",$b);
$n=$#a+1;
$m=-32000;
for ($i=0; $i<$n; $i++) {
  if ($a[$i]>$m) {$m=$a[$i];}
  $data[0]->[$i]=$i;
  $data[1]->[$i]=$a[$i];
}

$m=$m*1.1;
$n1=int($n/10)-1;

$graph->set(
x_label           => 'Number',
y_label           => 'Hurst',
title             => 'Hurst Dynamic',
y_max_value       => $m,
y_tick_number     => 8,
x_label_skip      => $n1,
y_label_skip      => 2
) or die $graph->error;

my $gd = $graph->plot(\@data)->png or die $graph->error;
print $gd;
```

Після виконання програми ***hurst.pl*** в інтерфейсі ТС буде відображено список значень R/S. Далі цей список через буфер обміну завантажується до інтерфейсу

програми Microsoft Excel. Після цього засобами Microsoft Excel будується графік, у якому вказується логарифмічна шкала для обох осей (рис. 6.1). Використовуючи засоби апроксимації можна переконатися у справедливості степеневого співвідношення для значень R/S.

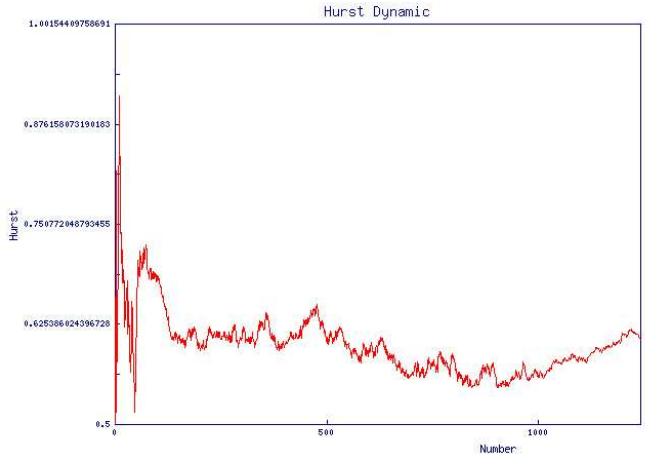


Рис. 17.1. Динаміка зміни показника Херста

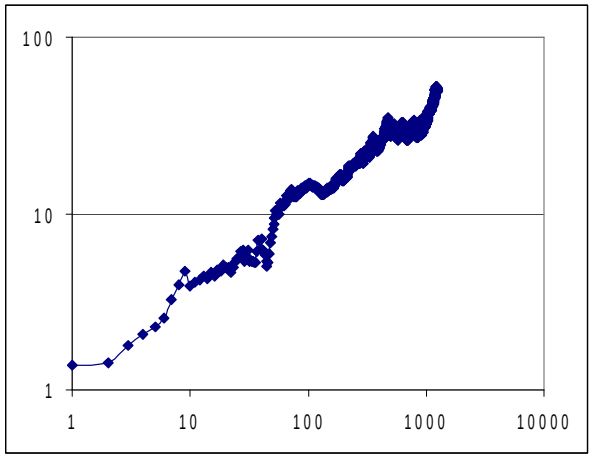


Рис. 17.2. Динаміка зміни показника значень R/S у логарифмічній шкалі

Питання для самостійної роботи:

1. Довідатись, як показник Херста пов'язаний з фрактальними властивостями часового ряду [10, 12].
2. Обчислити показник Херста ряду, утвореному випадковими числами, який можна отримати, використовуючи, наприклад, функцію ***rand ()***.

18. Мережа понять та їх сумісної появи в документах

У рамках даного курсу вивчаються властивості так званих «складних мереж» (complex networks), зокрема створюються та аналізуються мережі зв'язку понять (наприклад, прізвищ персон), що екстрагуються з текстових документальних масивів. Необхідно зауважити, що наявний тестовий масив документів ***«../../test.txt»*** містить маркування, що вказують на прізвища персон, які було згадано у відповідних документах.

Для побудови мережі сумісних згадувань у документах з текстового масиву визначених прізвищ необхідно сформувати два робочих файлу – файл прізвищ персон у порядку, в якому вони зустрічаються в окремих документах, та файл прізвищ персон, що цікавлять користувача.

Перший файл ***«fam.txt»*** повинен мати вигляд, що відповідає наведеному нижче фрагменту:

```
*** 1
007       name.Белых z.Белых
007       name.Долгоруком z.Долгорукому
007       name.Ельцин z.Ельцин
007       name.Жириновск z.Жириновский
007       name.Маркс z.Марксу
*** 2
007       name.Иванов z.Иванов
007       name.Широпаев z.Широпаев
007       name.Штеп z.Штепа
*** 3
007       name.Ансип z.Ансип
007       name.Переседов z.Переседов
*** 4
007       name.Тимошенк z.Тимошенко
*** 5
007       name.Гюл z.Гюля
```

У цьому файлі послідовністю **«***»** відокремлені різні вихідні документи, перед основою прізвища встановлено префікс «***name.***», перед конкретним вживанням (можливо, не у нормальній формі) стоїть префікс «***z.***».

Для побудови цього файлу на основі тестового масиву «***../../test.txt***» має використовуватися програма, фрагмент тексту якої наведено нижче:

```
open FILE,"../../test.txt";
open FOUT,">fam.txt";
while($buf=<FILE>) {
  if ($buf=~m/^\*\*\*/) {
    print FOUT $buf;
  }
  if ($buf=~m/^007.*name/) {
    print FOUT $buf;
  }
}
close FILE;
close FOUT;
```

Ця програма виводить у файл «***fam.txt***» ті рядки, що починаються з ознаки початку документа ***«***»*** та у полі рубрик ***«007»*** містять підрядки з префіксом «***name.***».

Другий файл ***«names.txt»*** має містити прізвища персон та відповідну кількість їх згадувань у документах та мати вигляд, що відповідає наведеному фрагменту:

```
1101     Мороз
808      Путин
719      Кириленк
707      Симоненк
634      Пшеничн
580      Саркоз
```

```
426      Луценк
390      Азаров
382      Кучм
```

Для побудови переліку персон у наведеному форматі виконується процедура, яка включає такі кроки:

а) Виконується програма вибору прізвищ персон та виведення їх у робочий файл «***nam.txt***». Текст цієї програми наведено нижче:

```
#!/usr/bin/perl -w
$fn="fam.txt";
$fo=">nam.txt";
open FILE,$fn;
open FOUT,$fo;
while($buf=<FILE>) {
  if ($buf=~m/^007.*name\.(.*?) /) {
    $buf=$1;
    print FOUT $buf,"\n";;
  }
}
close FILE;
close FOUT;
```

б) У режимі «Системні команди» здійснюється сортування файлу «***nam.txt***» за допомогою відповідної команди мовою Shell:

```
sort –o nam.txt nam.txt
```

в) На основі інформації з файлу «***nam.txt***» здійснюється розрахунок кількості згадувань прізвищ персон та заповнюється вихідний файл «***names.txt***». Нижче наведено текст відповідної програми:

```
#!/usr/bin/perl -w
$doc=""; $i=0;
open FILE,"nam.txt";

open FOUT,">names.txt";
while($_=<FILE>) {
  chomp();
  $buf=$_;
  if ($buf eq $doc) {
    $i++;
  }
  else {
      if ($i>0) { print FOUT "$i  $doc\n";}
      $doc=$buf;
      $i=1;
  }
}
```

```
close FILE;
close FOUT;
```

г) У режимі «Системні команди» здійснюється сортування файлу «***names.txt***» за першим числовим параметром за допомогою команди «sort» з відповідними параметрами:

```
sort –o names.txt names.txt –r -n
```

Файл «***names.txt***» буде використовуватися надалі для вибору прізвищ персон, сумісні згадування яких у документах цікавить користувача, у якого також є можливість коригування цього файлу з інтерфейсу ТС.

Питання для самостійної роботи:

1. Удосконалити процедуру побудови файлу ***«names.txt»*** шляхом включення в головну програму кроків а) – в).
2. Реалізувати розглянуті процедури для побудови мережі компаній (поле 008, префікс – ***«firm.»***).

19. Візуалізація мережі персон

Сформовані засобами, що описані у попередньому розділі, файли «***fam.txt***» і «***names.txt***» фактично визначають мережу, вузлами якої є прізвища персон, а ребрами – зв'язки, що визначаються сумісними згадуваннями цих прізвищ у документах. Крім того, кількість цих згадувань може визначати вагу відповідних ребер.

Нижче розглядається програма побудови візуального образу цієї мережі (рис. 19.1). Передбачається, що прізвищам – вузлам мережі будуть відповідати точки, що рівномірно розподілені по колу, а зв'язкам – хорди.

Для реалізації програми мовою Perl застосовується графічний модуль GD. Для відображення літер кирилиці використовується стандартний шрифт Arial, який розміщено на сервері ТС, а також модуль Iconv, за допомогою якого здійснюється перекодування літер у формат UTF. Побудова моделі та алгоритму візуалізації мережі.

Нижче наведено текст програми, що містить відповідні коментарі.

```
#!/usr/bin/perl -w

# Додаткові модулі
use Text::Iconv;
use GD;

sub InitColors { # Підпрограма визначення кольорів
    my($im) = $_[0];
```

```
    $white = $im->colorAllocate(255,255,255);
    $black = $im->colorAllocate(0,0,0);
    $red = $im->colorAllocate(255,0,0);
    $blue = $im->colorAllocate(0,0,255);
    $green = $im->colorAllocate(0, 255, 0);
    $brown = $im->colorAllocate(255, 0x99, 0);
    $violet = $im->colorAllocate(255, 0, 255);
    $yellow = $im->colorAllocate(255, 255, 0);
}

# Визначення перекодування, яке буде застосовуватися надалі
$converter = Text::Iconv->new("koi8-u", "utf-8");

# Графічна область
$im = new GD::Image(900,900);
&InitColors($im);
$im->transparent($white);
$im->interlaced('true');

# Границя області – червоний квадрат
$im->rectangle(0,0,899,899,$red);

# Значення подвоєнного «пі»
$pi2=6.283;

$M=5000; $N=12;  # Кількість документів та прізвищ персон
$alfa=$pi2/$N;
for($i=0; $i<$N; $i++) {
    $x[$i]=(cos $alfa*$i)*320+450; # координати вузлів
    $y[$i]=(sin $alfa*$i)*320+450;
    $x1[$i]=(cos $alfa*$i)*330+450; # координати надписів
    $y1[$i]=(sin $alfa*$i)*330+450;
}

$i=0;
open FILE,"<names.txt";
while($_=<FILE>) {  # зчитування масиву прізвищ
  chomp();
  $buf=$_;
  if ($buf=~m/\d+\t(.*?)$/) {
    $name[$i]=$1;
    $i++;
    if ($i==$N) { last; }
  }
}
close FILE;

for ($i=0; $i<$N; $i++) {
  for ($j=0; $j<$N; $j++) {
    $mtr[$i][$j]=0; # ініціалізація матриці зв’язків
  }
```

```
}
$i=1; $pr=0; $k=0;

open STDI,"<fam.txt";

while($_=<STDI>) { # зчитування масиву групованих прізвищ
  chomp();
  $buf=$_;
  if ($buf=~m/^\*\*\*/) {
    if ($pr==1) {
      for ($l=0; $l<$k; $l++) {
        for ($p=0; $p<$k; $p++){
          $a=$mas_doc[$l];
          $b=$mas_doc[$p];
          $mtr[$a][$b]++; # зростання ваги зв’язку $a та $b
        }
      }
      $i++;
      if ($i==$M) { last;}
    }
    $pr=0;
    @mas_doc={};
    $k=0;
  }
  if ($buf=~m/^007/) { # рядок, що містить прізвище
    for ($j=0; $j<$N; $j++) {
      $a=$name[$j];
      if ($buf=~m/$a/) { # витяг  прізвища
        $mas_doc[$k]=$j;
        $k++; $pr=1;
      }
    }
  }
}
close STDI;

# Відображення мережі

$delta=2; # граничний рівень ваги зв’язку

for ($i=0; $i<$N; $i++) {
   $im->arc($x[$i],$y[$i],4,4,0,360,$black);
   $lab=$name[$i];
   if ($lab=~m/^name\.(.*?)$/) {$lab=$1;}
   $lab1 = $converter->convert($lab);

   # підпис ліній шрифтом $font
   $font = "/resource/www/test/down/Arial.TTF";

   # Розрахунок нахилу підписів
```

```
   $naklon=-$pi2*$i/$N;
   $im->stringFT($black, $font, 12, $naklon, $x1[$i], $y1[$i],
$lab1);
   for ($j=0; $j<$N; $j++) {
     if ($mtr[$i][$j]>=$delta) { # вага зв'язку перевищує $delta:

            # відображення зв'язку - побудова хорди
            $im->line($x[$i],$y[$i],$x[$j],$y[$j],$blue);
     }
   }
}

# Виведення сформованого графічного образу

binmode STDOUT;
select(STDOUT);
$| = 1;
undef $/;
print "Content-type: image/jpeg\n\n";
print $im->jpeg(900);
```

Питання для самостійної роботи:

1. Детально ознайомитися з режимами створення зображень і виведення тексту за допомогою модуля GD [10].
2. Удосконалити наведену програму шляхом виведення прізвищ персон з нахилом у напрямку радіусу, проведеному від центру кола до відповідного вузла.

20. Розрахунок показника кластерності мережі

Коефіцієнт кластерності для окремого вузла мережі визначається таким чином. Нехай з вузла виходить k ребер, які з'єднують його з k іншими вузлами, найближчими сусідами. Якщо припустити, що всі найближчі сусіди з'єднані безпосередньо один з одним, то кількість ребер між ними становила б $\frac{1}{2}k(k-1)$. Тобто це число, що відповідає максимально можливій кількості ребер, якими могли б з'єднуватися найближчі сусіди вибраного вузла. Відношення реальної кількості ребер, які з'єднують найближчих сусідів даного вузла до максимально можливого (такого, при якому всі найближчі сусіди даного вузла могли би бути з'єднані безпосередньо один з одним) називається коефіцієнтом кластерності вузла i – $C(i)$. Природно, ця величина не перевищує одиниці.

Нижче наведено фрагмент програми мовою Perl, що застосовується для розрахунку коефіцієнта кластерності мережі згадувань персон.

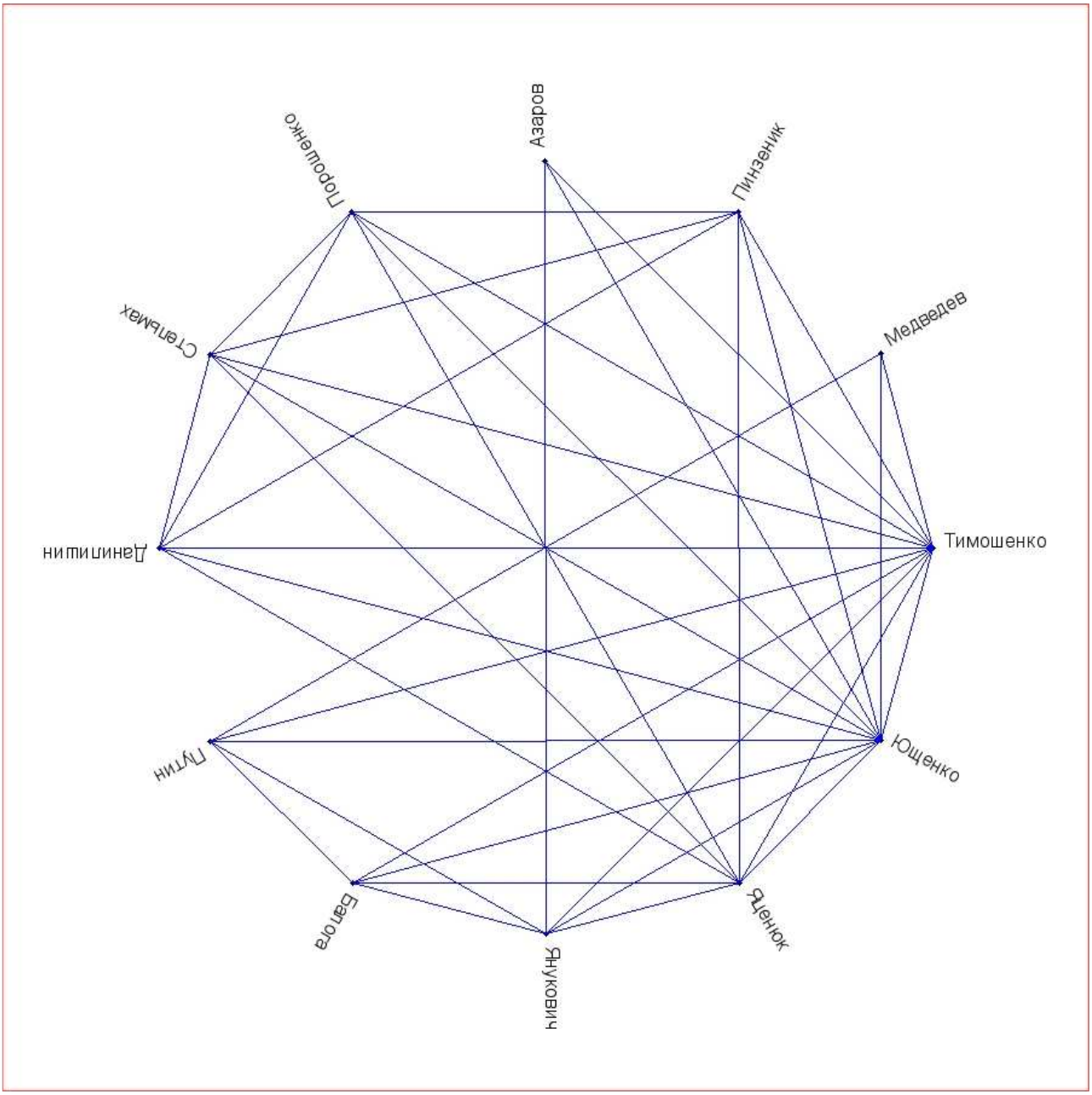


Рис. 19.1. Візуалізація мережі персон

```
# Документів - $M, прізвищ - $N

. . .
# Фрагмент побудови матриці зв’язку @mtr, ідентичний
# наведеному у попередньому розділі

. . .
# Визначається граничний рівень зв’язку $delta
як 0.10 від середньої ваги зв’язку всієї мережі

$delta=0;
for ($i=0; $i<$N; $i++) {
```

```
  print "\n";
  for ($j=0; $j<$N; $j++) {
    $delta=$delta+$mtr[$i][$j];
  }
}
$delta=0.10*$delta/($N**2);

# Перевизначення значень матриці зв'язку

for ($i=0; $i<$N; $i++) {
  for ($j=0; $j<$N; $j++) {
    if ($mtr[$i][$j]>=$delta) {
         $mtr[$i][$j]=1;
    }
    else { $mtr[$i][$j]=0; }
  }
}

# Безпосередній підрахунок коефіцієнту кластерності

$cl=0;
for ($i=0; $i<$N; $i++) {
  $c[$i]=0; $nc[$i]=1;
  for ($j=0; $j<$N; $j++) {
    for ($k=0; $k<$N; $k++) {
      if ($k!=$j) {
            if (($mtr[$i][$j]>0) && ($mtr[$i][$k]>0)  &&
                  ($mtr[$k][$j]>0)) { $c[$i]++; }
            if (($mtr[$i][$j]>0) && ($mtr[$i][$k]>0)) {
                  $nc[$i]++;
            }
      }
    }
  }

  if ($nc[$i]>0) { $c[$i]=$c[$i]/$nc[$i]; }
  $cl=$cl+$c[$i];

  # Виведення коефіцієнту кластерності окремих вузлів
  print "\n c($i) = $c[$i] - $name[$i]";
}
$cl=$cl/$N;

# Виведення коефіцієнту кластерності всієї мережі
print "\n\n$cl\n";
```

Питання для самостійної роботи:

1. Детально ознайомитись з основними показниками складних мереж і окремих вузлів таких [1, 16, 18].

2. Розробити програму та побудувати графік залежності коефіцієнту кластерності в залежності від рангу згадувань персон. Запропонувати інтерпретацію цієї залежності.

21. Розрахунок показника середнього шляху мережі

Відстань між вузлами визначається як кількість кроків, які необхідно зробити, щоб по існуючих ребрах добратися від одного вузла до іншого. Природно, вузли можуть бути з'єднані прямо або опосередковано. Шляхом між вузлами i та j називається найкоротша відстань між ними. Для всієї мережі розглядається поняття середнього шляху, як середню по всіх парах вузлів найкоротшу відстань між ними:

$$l = \frac{2}{n(n+1)} \sum_{i \geq j} d_{ij},$$

де n - кількість вузлів, d_{ij} – найкоротша відстань між вузлами i та j.

Деякі мережі можуть виявитися незв'язними, тобто знайдуться таки вузли i та j, між якими не існує шляху по ребрам, тобто відстань між ними є нескінченною (мережа прізвищ персон також може виявитися незв'язною). Відповідно, середній шлях може виявитися також рівним нескінченності. Для врахування таких випадків вводиться поняття середнього інверсного шляху між вузлами il, що розраховується за формулою:

$$il = \frac{2}{n(n-1)} \sum_{i > j} \frac{1}{d_{ij}},$$

де d_{ij} – найкоротша відстань між вузлами i та j.

Нижче наведено фрагмент програми мовою Perl, що застосовується для розрахунку середнього шляху мережі згадувань персон.

```
# Розрахунок середнього шляху

$M=5000; $N=50;

. . .

# Фрагмент побудови матриці зв’язку @mtr, ідентичний
# наведеному у попередньому розділі

. . .
# Визначається граничний рівень зв’язку $delta
```

```
як 0.10 від середньої ваги зв'язку всієї мережі

$delta=0;
for ($i=0; $i<$N; $i++) {
  print "\n";
  for ($j=0; $j<$N; $j++) {
    $delta=$delta+$mtr[$i][$j];
  }
}
$delta=0.10*$delta/($N**2);

# Перевизначення значень матриці зв'язку

for ($i=0; $i<$N; $i++) {
  for ($j=0; $j<$N; $j++) {
    if ($mtr[$i][$j]<$delta) {
       # Відстань між вузлами значна за величиною:
       $mtr[$i][$j]=$N**3;
    }
    else {$mtr[$i][$j]=1;}
  }
}

# Найпростіший алгоритм обчислення шляху між вузлами

for ($k=0; $k<$N; $k++){
  for ($i=0; $i<$N; $i++){
    for ($j=0; $j<$N; $j++){
      $a=$mtr[$i][$k]+$mtr[$k][$j];
      if ($a < $mtr[$i][$j]) {
        $mtr[$i][$j]=$a;
      }
    }
  }
}

# Розрахунок інверсної мінімальної відстані

$i_rast=0;
for ($i=0; $i<$N; $i++){
  print "\n";
  for ($j=0; $j<$N; $j++){
    if ($mtr[$i][$j]==$N**3) { $mtr[$i][$j]=0;}
    else {$mtr[$i][$j]=1/$mtr[$i][$j]; }

       # Інверсна мінімальна відстань між вузлами

    print $mtr1[$i][$j]," ";
    $i_rast=$i_rast+$mtr1[$i][$j];
  }
}
```

```
# Загальна інверсна мінімальна відстань

$i_rast=$i_rast/($N**2);
print "\nInv_rast=$i_rast\n";
```

Питання для самостійної роботи:

1. Розробити алгоритм обчислення коефіцієнту посередництва (***betweenness***) для вузла мережі [1, 16, 18].
2. Розробити програму та побудувати графік залежності коефіцієнту посередництва в залежності від рангу згадувань персон. Запропонувати інтерпретацію цієї залежності.

22. Побудова моделі “малого світу”

Феномен, притаманний багатьом реальним мережам носить назву ефекту малого світу (Small Worlds). Ефект «малих світів» спостерігається тоді, коли при певному рівні локальних і випадкових «далеких» зв'язків одночасно спостерігаються малий середній шлях та високий рівень кластерності мережі. Цей ефект наочно демонструється абстрактною моделлю, запропонованою Д. Уаттсом і С. Строгатцом, які розглядали мережу, вузли якої розміщені на колі. В початковому стані цієї мережі кожний вузол з’єднується ребрами з чотирма найближчими «сусідами». Тобто у початковому стані мережа – регулярна. Далі випадковим чином з заданим рівнем ймовірності p деякі близькі зв’язки замінюються випадковими «далекими» (саме в цьому випадку виникає феномен «малих світів»).

Програма візуалізації моделі «малих світів» базується на використанні модуля GD та близька за структурою до вже розглянутої програми візуалізації мережі з прізвищ персон. У програмі явним чином вказуються рівні ймовірності p (змінна $P) та кількість вузлів $N. Результати роботи програми, фрагменти з якої наведено нижче, представлені на рис. 22.1.

```
#!/usr/bin/perl -w
# Процедура побудови «малого світу» за Уаттсом-Строгатцем
use GD;

. . .

# Визначення графічної області та визначення координат вузлів

. . .

$N=24; # кількість вузлів
```

```
$P=30; # ймовірність у процентах

# Первинне заповнення масиву зв’язків @mtr

$i=0; $j=0;
for ($i=0; $i<$N; $i++) {
  for ($j=0; $j<$N; $j++) {
    $mtr[$i][$j]=0;
  }
}

for ($i=0; $i<$N; $i++) {
  $mtr[$i][$i+1]=1;
  $mtr[$i][$i+2]=1;
  if ($i>0) { $mtr[$i][$i-1]=1; }
  if ($i>1) { $mtr[$i][$i-2]=1; }
}

$mtr[0][$N-1]=1;
$mtr[0][$N-2]=1;
$mtr[1][$N-1]=1;
$mtr[$N-1][0]=1;
$mtr[$N-2][0]=1;
$mtr[$N-1][1]=1;

# Заміна близьких зв’язків випадковими далекими

for ($i=0; $i<$N; $i++) {
 for ($k=-2; $k<3; $k++) {
  if ($k!=0) {
   $j=$i;
   while ((abs($i-$j)<3) || (abs($i-$j)>($N-2))) {
    $j=int(rand($N));
   }
   $p=int(rand(101));

   if ($p<$P) {
    if ($i>1) {
     $mtr[$i][$j]=1;
     $mtr[$j][$i]=1;
     $mtr[$i][$i+$k]=0;
     $mtr[$i+$k][$i]=0;
    }
    if ($i==1) {
     $mtr[$i][$j]=1;
     $mtr[$j][$i]=1;
     if ($k>-2) {
       $mtr[$i][$i+$k]=0; $mtr[$i+$k][$i]=0;
     }
     else {
       $mtr[$i][$N-1]=0; $mtr[$N-1][$i]=0;
```

```
     }
    }
    if ($i==0) {
     $mtr[$i][$j]=1;
     $mtr[$j][$i]=1;
     if ($k>-1) {
       $mtr[$i][$i+$k]=0; $mtr[$i+$k][$i]=0;
     }
     else {
       if ($k==-2) {$mtr[$i][$N-2]=0; $mtr[$N-2][$i]=0;}
       if ($k==-1) {$mtr[$i][$N-1]=0; $mtr[$N-1][$i]=0;}
     }
    }
   }
  }
 }
}

# Візуалізація мережі

for ($i=0; $i<$N; $i++) {
  $im->arc($x[$i],$y[$i],4,4,0,360,$black);
  for ($j=0; $j<$N; $j++) {
     if ($mtr[$i][$j]>0){

      # зображення ребер - зв’язків

      $im->line($x[$i],$y[$i],$x[$j],$y[$j],$red);
     }
  }

  # Зображення вузлів

  $im->string(gdMediumBoldFont,$x1[$i],$y1[$i],$i,$black);
}
$x="P = ".$P." %";

# Візуалізація значення ймовірності

$im->string(gdMediumBoldFont,10,380,$x,$black);

# Відображення сформованого зображення

binmode STDOUT;
select(STDOUT);
$| = 1;
undef $/;
print "Content-type: image/jpeg\n\n";
print $im->jpeg(400);
```

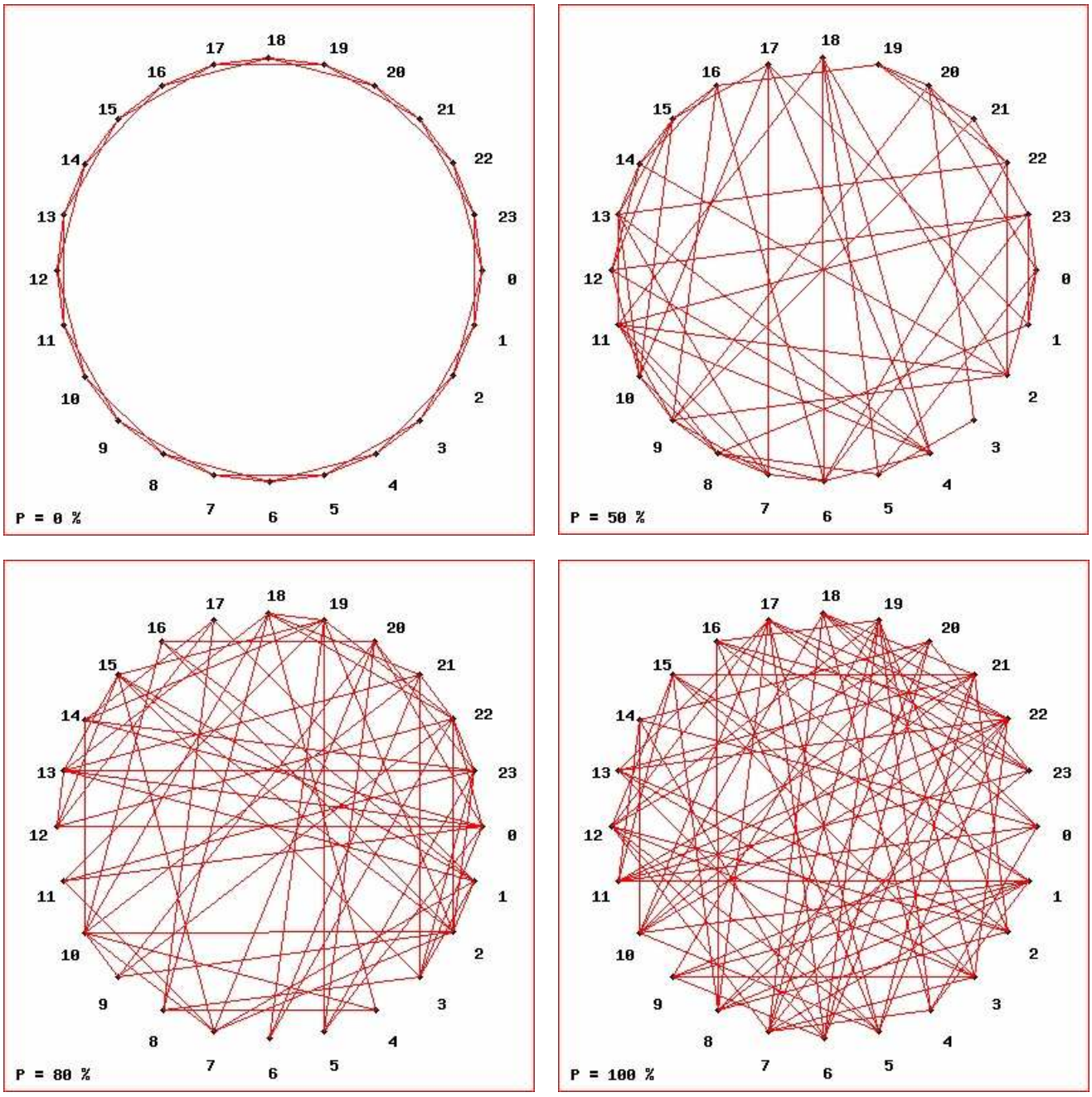


Рис. 22.1. Візуалізація моделі мережі «малого світу» при різних значеннях p

Питання для самостійної роботи:

1. Детально ознайомитися з моделлю Уаттса-Строгатца [1, 19].
2. Розробити програму візуалізації варіанту моделі Уаттса-Строгатца, коли при виникненні «далеких» зв’язків «близькі зв’язки» не перериваються.

23. Дослідження показників мережі «малого світу»

Для побудованої у попередньому розділі моделі «малого світу» за алгоритмом Уаттса-Строгатца досліджуються показники кластерності та середнього шляху, розглянуті у попередніх розділах.

Відображення різних значень коефіцієнтів кластерності та середнього шляху здійснюються двома методами – за допомогою програми, розробленої з використанням бібліотеки GD, аналогічної представленій у розділі 4 (відображення у лінійній шкалі, рис. 23.1 *а* та 23.2 *а*), а також засобами пакету Microsoft Excel (у напівлогарифмічній шкалі, рис. 23.1 *б* та 23.2 *б*)

У текстах програм використовуються фрагменти, наведені у попередніх розділах.

Фрагменти процедури розрахунку кластерності моделі Уаттса-Строгатца:

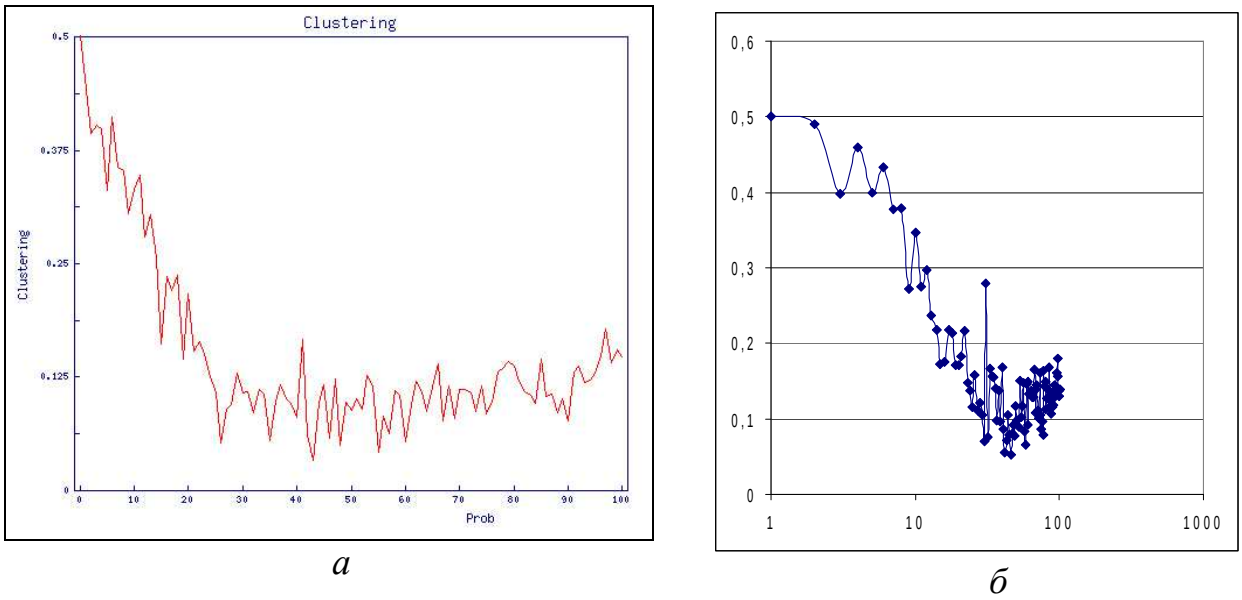
```
#!/usr/bin/perl -w
# Процедура розрахунку кластерності моделі Уаттса-Строгатца

$N=24; # кількість вузлів
$P=30; # ймовірність у процентах

# Основний цикл програми

for ($P=0; $P<=100; $P++) {

     # Побудова мережі для для даного $P

     . . .

     # Визначення коефіцієнту кластерності

     . . .

     # Підготовка рядка передачі параметрів

     $fmas=$fmas." ".$cl;
}

# Виклик програми візуалізації

print "
<form action='smallgc.pl' method=post target=_new>
<input type=hidden name=f1 value=\"$fmas\">
<input type=submit value='Graph It!'>

</form></body></html>"
```

Фрагменти процедури розрахунку середнього шляху моделі Уаттса-Строгатца:

```
#!/usr/bin/perl –w
# Процедура розрахунку середнього шляху за Уаттсом-Строгатцем
print "Content-type: text/html\n\n";

$N=24; # кількість вузлів
$P=30; # ймовірність у процентах

# Основний цикл програми

for ($P=0; $P<=100; $P++) {

     # Побудова мережі для для даного $P

     . . .

     # Визначення мінімальної відстані між вузлами

     for ($k=0; $k<$N; $k++){
       for ($i=0; $i<$N; $i++){
         for ($j=0; $j<$N; $j++){
           $a=$mtr[$i][$k]+$mtr[$k][$j];
           if ($a < $mtr[$i][$j]) {
             $mtr[$i][$j]=$a;
           }
         }
       }
     }

     $n_rast=0;
     for ($i=0; $i<$N; $i++){
       for ($j=0; $j<$N; $j++){
         if ($mtr[$i][$j]==$N**3) { $mtr[$i][$j]=0;}
         $n_rast=$n_rast+$mtr[$i][$j];
       }
     }
     $n_rast=$n_rast/($N**2);

     # Підготовка рядка передачі параметрів

     $fmas=$fmas." ".$n_rast;
}

# Виклик програми візуалізації

print "
<form action='smallgr.pl' method=post target=_new>
<input type=hidden name=f1 value=\"$fmas\">
<input type=submit value='Graph It!'>
</form></body></html>";
```

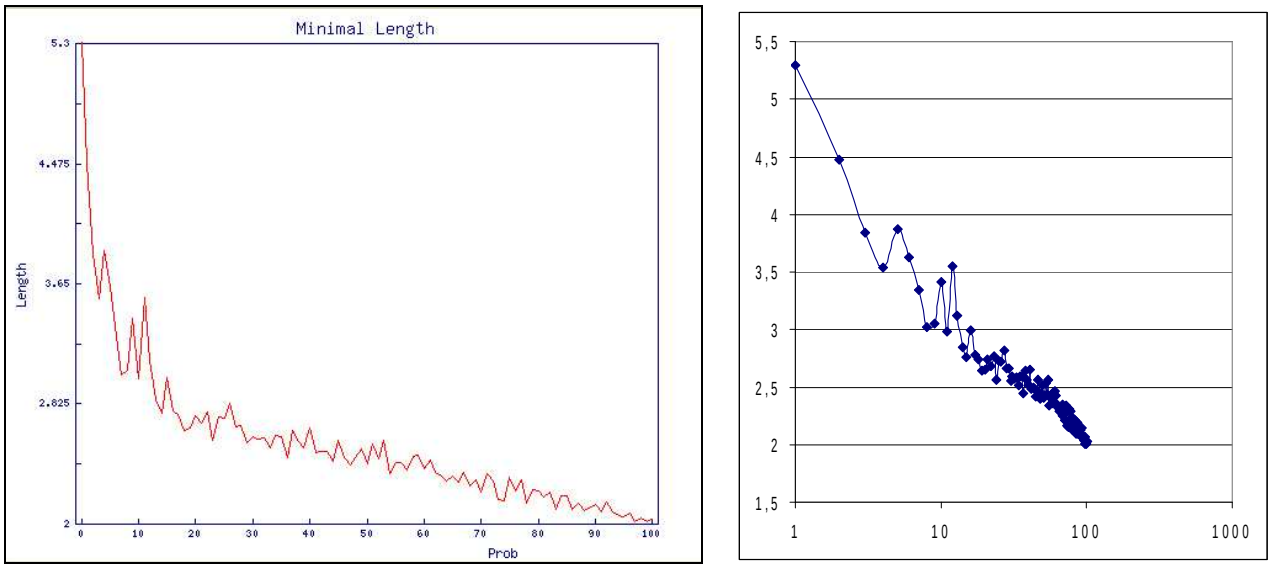
Питання для самостійної роботи:

1. Розробити програми 10-разового обчислення значень коефіцієнтів кластерності та середнього шляху та виведення усереднених значень.
2. Обчислити усереднені коефіцієнти кластерності та середнього шляху для різних кількостей вузлів.



а *б*

Рис. 23.1. Графік зміни коефіцієнту кластерності від p



а *б*

Рис. 23.1. Графік зміни середнього шляху від p

ІНФОРМАЦІЙНІ ДЖЕРЕЛА

1. Основна література:

[1] Ланде Д.В., Корнейко О.В., Мохор В.В. Конспект лекцій з навчальної дисципліни «Методи та засоби комп'ютерних інформаційних технологій»: Основи теорії інформаційного пошуку в Інтернет. – К.: ІСЗЗІ НТУУ «КПІ», 2007. – 165 с.

[2] Ландэ Д.В. Основы интеграции информационных потоков - К.: Инжиниринг, 2006. - 240 с. (http://dwl.kiev.ua/art/monogr-osnov/spusk3.pdf)

[3] Ландэ Д.В. Поиск знаний в Internet. Профессиональная работа – М.: ИД Вильямс, 2005. - 271 с.

[4] Спецификация HTML 4.0 (http://www.citforum.ru/internet/html40/cover.html)

[5] Торкингтон Н., Кристиансен Т. Библиотека программиста: Perl. – М.: Питер, 2001. - 736 с.

[6] Федосеева А. Спецификация языка Perl. URL: http://lib.luksian.com/programming/ perl/spec/

[7] Полянский А. Учебное пособие по CGI-программированию. – М.: Познавательная книга плюс, 2000. – 176 с.

[8] Гошко В. Регулярные выражения и поиск текста в Perl // «Системный администратор». - № 8, 2003. - С. 78-86.

[9] Padala P. Exploring Perl Modules - Part 1: On-The-Fly Graphics with GD // Linux Gazette. – Issue 81, aug. 2002. URL: http://linuxgazette.webhosting76.com/issue81/padala.html

[10] Федер Е. Фракталы -М.: Мир, 1991. - 254 с.

[11] Фон Нейман Дж. Теория самовоспроизводящихся автоматов - М.: Мир, 1971.

2. Додаткова література

[12] Иванов С.А. Стохастические фракталы в Информатике // Научно-техническая информация. - Сер. 2, 2002. – Вып. 8. - С. 7-18.

[13] Ландэ Д.В. Фрактальные свойства тематических информацийнных потоков из Интернет // Реєстрація, зберігання і обробка даних, 2006, Т. 8, № 2.– С. 93 - 99.

[14] Тоффоли Т., Марголус Н. Машины клеточных автоматов. - М.: Мир, 1991. - 280 с.

[15] Сегалович И.В. Как работают поисковые системы. // Мир Internet. – 2002. - № 10. URL: http://www.dialog-21.ru/direction_fulltext.asp?dir_id=15539

[16] Фурашев В.Н., Ландэ Д.В., Брайчевский С.М. Моделирование информационно-электоральных процессов: Монография. - К.: НИЦПИ АпрН Украины, 2007. - 182 стр. URL:

http://dwl.visti.net/art/miep/miep.pdf
[17] Baeza-Yates R., Ribeiro-Neto B. Modern Information Retrieval. ACM Press, 1999. – 513 p.
[18] Newman M.E.J. The structure and function of complex networks. // SIAM Review. - 2003. - Vol. 45. -P. 167–256.
[19] Watts D.J., Strogatz S.H. Collective dynamics of ″small-world″ networks. // Nature. - 1998. - Vol. 393. -P. 440–442.
[20] Wolfram S. A New Kind of Science. Champaign, IL: Wolfram Media Inc., 2002. – 1197 p.
[21] http://www.searcengines.ru/
[22] http://www.cinforum.ru/
[23] http://www.rcdl.ru/
[24] http://www.romip.narod.ru/
[25] http://www.osp.ru/
[26] http://company.yandex.ru/class/
[27] http://company.yandex.ru/grant/
[28] http://www.wikipedia.org/

Додаток. Елементи мови Perl

В рамках цього додатку не ставиться задача повного викладу можливостей мови ***Perl*** (Practical Extraction and Report Language), з якими можна ознайомитися з наведених нижче інформаційних джерел. Наведено лише мінімальний необхідний для проведення практичних занять перелік основних відомостей з мови Perl. З інтерфейсу технічного середовища проведення практичних занять, крім того, доступна повна специфікація мови Perl, а також посібник із застосування CGI-інтерфейсу.

Perl – це мова, що інтерпретується, пристосована для обробки довільних текстових файлів, витягу з них необхідної інформації та виведення повідомлень. Синтаксис виразів Perl-у близький до синтаксису мови С.

Змінні

У мові Perl існує три типи змінних: скалярні, спискові (масиви) та хеші (асоціативні масиви). Перед ім'ям скалярної змінної ставиться знак '***$***', перед масивом '**@**', перед хешем '***%***', наприклад, $scalar_var, @array_var або %hash_var. На відміну від типізованих мов Perl не вимагає об'яви типів своїх об'єктів. Скалярні зміні можуть бути як числові, так і рядкові, але на це не треба вказувати, Perl з контексту в залежності від операцій може визначати типи даних. Наприклад, "123"+"4" буде 127 (або "127") так як операція '+' діє над числами. Якщо ж застосувати операцію конкатенації рядків '.', то рядкове "test" . 1 буде "test1".

Скалярні змінні можуть містити різні прості типи даних, такі як числа, рядки або посилання. Прості скаляри (надалі - змінні) завжди починаються зі знаку долару: $, навіть у тому випадку, коли здійснюється звернення до елементу масиву, наприклад:

```
$day          проста змінна day
$day[28]      29  елемент масиву day
$day{'Feb'}   значення 'Feb' з хешу %day
$#day         останній індекс масиву @day
```

У булевому контексті скаляр приймає значення ***TRUE***, якщо він містить не нульове число або непустий рядок. В Perl існує декілька варіантів запису чисел, наприклад:

```
12345
12345.67
.23E-10
0xffff              шістнадцятерічний запис
0377                вісьмерічний запис
1_234_567_890       підкреслення для зручності читання
```

При запису констант крім звичайних символів використовуються також спеціальні, наприклад:

```
\t      табуляція
\n      переведення рядку
\r      повернення каретки
\b      пробіл
\e      символ Escape
\a      сигнал (alarm)
\f      перехід на наступну сторінку
```

Рядки можуть бути у подвійних та в одинарних лапках, різниця між ними у тому, що у одинарних лапках не здійснюється підстановка змінних, а у подвійних - здійснюється, наприклад:

```
$x='qwerty';
print 'my var is $x'; # виведе my var is $x
print "my var is $x"; # виведе my var is qwerty
```

Списки: спикові змінні починаються з символу '@' та будуються наступним чином:

```
@List1=(1,2,5,70);
@List2=(12,23,@List1); #12,23,1,2,5,70
@Rgb=($r,$g,$b);
```

Perl дозволяє довільно подовжувати масив шляхом присвоювання значення елементу, індекс якого більше, ніж останній індекс масиву. Також можна довільно зменшити довжину масиву. Ім’я простого масиву починається зі знака @, наприклад:

```
@day = ('a','b');
```

Після визначення:

```
$day[3] = 'c';
```

масив day буде містити три елементи: ('a','b','c'). Ініціалізація масиву здійснюється наступним чином: @day = (); або, що те ж саме: $#day = -1. Після цього масив day стане пустим.

Позначення асоціативного масиву (хешу) починається зі знака процент %, наприклад: %day (key1, val1, key2, val2, ...). Хеш складається з пар ключ-значення, весь хеш позначається як %хеш, до окремих елементів доступ здійснюється таким чином: ***$хеш{скалярний вираз}.*** Хеш будується таким чином:

```
$my_hash{1}="doom";
```

```
$my_hash{'visti'}="www.elvisti.com";
$my_hash{1+2}=100;
```

Хеш може бути побудовано з масиву з парною кількістю елементів, де пари перетворюються на ключ-значення, наприклад:

```
%hash=(1,20,2,100);#аналогічно $hash{1}=20;$hash{2}=100;
```

Вилучення з хешу - операція delete, наприклад:

```
delete $hash{1}.
```

Звичайно для зручності читання між «ключем» та «значенням» хешу ставлять оператор ‘=>’, наприклад:

```
%map = (
          'red'   => 0x00f,
          'blue'  => 0x0f0,
          'green' => 0xf00
 );
```

Операції над скалярними змінними

Нижче наведені деякі операції над скалярними змінними:

Операція	Опис	Приклад
+ - * / %	Арифметичні	print 2*7+4/(8%3); print int(127/15); # ціла частина
**	Піднесення до ступеню	print 2**16;
++ --	Інкремент-декремент	$i++;
& | ^ ~ << >>	Побітові	$x=3; $y=4; print $x\|$y; print $x&$y;
== != < > <= >= <=>	Числові операції порівняння	if ($x==9) { print "Ok!"; }
eq ne lt gt le ge cmp	Рядкові операції порівняння	if ($game eq 'doom') { print "You are doomer!\n"; }
|| && !	Логічні	if (($x==9) \|\| ($game eq 'doom')) {print "hello you!\n";}
?:	Умовний оператор	$x=($game eq 'quake'?9:8);
,	Послідовне обчисленння	$x=10,$y=20;
.	Конкатенація	$x='http://'.'www.uic.nnov.ru';
x	Повторенння	$x='1234'x5; # $x='123412341234123412341234'
=~	Співставлення з шаблоном	if ($url=~m/http/) print "HTTP"; }
!~	Те ж саме із запереченням	if ($url!~m/http/) { print "No HTTP"; }
= += -= *= /= %= **= |= &= ^= ~= <<= >>= .= x=	Присвоювання	$x+=$y;

Оператори управління

Для реалізації алгоритмів та управління процесами у мові Perl застосовуються оператори.

Основний умовний оператор в мові Perl – це ***if,*** який для зручності має дві форми:

```
if(умова)оператор;
оператор if умова;
```

Наприклад,

```
$var = 1;
$var2 = 3 if $var > 0; # Результат: $var2 = 3
```

Як доповнення до оператора if можна розглядати оператор ***unless***, який означає if із запереченням, наприклад:

```
unless(($method eq 'GET')||($method eq 'POST')){
   print "Unsupported method";
}
print "Ok" unless $x < $y;
```

Оператор циклу «поки» - ***while*** виконується поки умова, до якої він відноситься, приймає значення TRUE, наприклад:

```
$var = 1;
while ($var < 5) { print $var++ ; } # Печать $var с інкрементом
```

Результат: 1234

Оператор циклу «допоки» - ***until*** виконується до тих пір, допоки умова, до якої він відноситься приймає значення FALSE, наприклад:

```
$var = 1;
print $var++ until $var > 5; # Печать $var с інкрементом
```

В операторах while та until перевірка умови здійснюється перед виконанням коду блоку, до яких вони відносяться, за винятком випадку, коли застосовується оператор do, наприклад:

```
do {
    $_ = ;
    ...
} until $_ eq ".\n";
```

Де перевірка умови здійснюється після виконання блоку. Для управління циклами може також застосовуватися оператор goto для переходу на місце програми, позначене міткою. Мітка складається з ідентифікатора та двокрапки.

У мові Perl існує оператор циклу ***foreach***, який не застосовується у мові С. Цей оператор дозволяє пробігати по всіх елементах масиву, наприклад, здійснюючи присвоєння по черзі його елементам значень деякої змінної. Синтаксис оператора foreach такий:

```
foreach $змінна (@масив){
  блок операторів;
}
або
foreach (@масив){
  оператори;
}
```

Приклад:

```
@місяць = ("січень","лютий","березень");  # Створили масив
foreach $i (@місяць){
  print $i," ";       # Друк $i
}
```

Результат: `січень лютий березень`

Оператор ***for*** повністю аналогічний оператору for у мові С. Наприклад, вираз у заголовку циклу for ($i = 2; $i < 5; ++$i) означає, що спочатку виконується присвоєння ($i = 2), якщо виконується ($i < 5), то цикл продовжується, після виконання чергового циклу здійснюється операція інкременту (++$i). Цикл for може мати такий вигляд:

```
for ($i = 2; $i < 5; ++$i) {
     print $i, " ";
}
print "\nПісля циклу i = $i\n";
```

Результат: `2 3 4`
Після циклу `i = 5`

Функції введення-виведення

Для роботи з зовнішніми файлами або потоками в мові Perl застосовуються функції введення-виведення. Наведемо деякі з них.

Функція ***open***, що відкриває файл, може записуватися таким чином:

```
open(ФАЙЛОВА_ЗМІННА,"им'я файлу");    #відкрити файл для
читання
open(ФАЙЛОВА_ЗМІННА, ">им'я файлу");  #для запису
open(ФАЙЛОВА_ЗМІННА, ">>им'я файлу"); #для запису у кінець
open(ФАЙЛОВА_ЗМІННА,"+<им'я файлу");  #для читання і запису
```

Функція ***close*** закриває відкритий файл, наприклад:

```
close FDEC;
```

Для виведення даних з файлу використовується команда <>. Після того, як файл буде відкритим, з нього можна зчитати рядок у скалярну змінну. Зчитування здійснюється рядок за рядком. У кінці файлу функція зчитування приймає значення ***FALSE***. За замовченням зчитування здійснюється у змінну ***$_***. Наведемо приклад зчитування рядків з файлу паролів:

```
open(STDIN,"/etc/passwd");
while ($string = <STDIN>) {
   chomp($string);
    print $string,”\n”;
}
```

У цьому прикладі позбутися символу повернення каретки у кінці рядка допомагає функція ***chomp()***.

Функція ***print*** виводить рядок або список рядків у файл, який є її першим аргументом. Цей аргумент може бути скалярною змінною, якщо він відсутній, то здійснюється виведення у стандартний вихідний потік ***STDOUT***. Якщо список значень, що підлягають виведенню відсутній, то виводиться значення змінної ***$_***. У даному прикладі виводиться послідовність чисел від 1 до 10 у файл "dec.txt":

```
open (FDEC,">dec.txt");
for ($i=1; $i<=10; $i++) {
   print FDEC $i,”\n”;
}
close FDEC;
```

Функції роботи з рядками

Perl, як мова, орієнтована на обробку текстових даних, містить велику кількість функцій для роботи з рядками, причому можна вибирати різні підмножини Perl для досягнення однакових цілей. Наприклад, для тих, хто прийшов до програмування на Perl від Delphi або Pascal, будуть зручними такі функції, як index або substr. Для тих, хто має практику роботи з UNIX, будуть близькі засоби шаблонів і регулярних виразів (***RegExp***).

Нижче наведено перелік деяких «традиційних» функцій роботи з рядками:

index	- визначення позиції підрядка у рядку (перше входження)
rindex	- визначення позиції підрядка у рядку (останнє входження)
length	- довжина рядка
substr	- підрядок текстового рядка

split - розбивка рядка на набір підрядків за шаблоном-роздільником
chomp - видалення символу кінця рядка (звичайно - \n) з рядка
chr - перетворення числа на символ
ord - перетворення символу на код

Поряд із традиційними для мов високого рівня засобами роботи з рядками, в Perl присутні засоби роботи з регулярними виразами та шаблонами, які були перенесені з UNIX.

При цьому, по суті, регулярні вирази дозволяють зіставляти рядки із зазначеними шаблонами та виконувати заміну підрядків. В Perl є два основних оператора, пов'язаних з регулярними виразами:

```
m/.../ - перевірка збігів, порівняння (matching)
s/.../.../ - підстановка тексту (substitution)
```

Третім, близьким цим двом операторам, є оператор заміни тексту - ***tr/.../.../*** (translation), у якому не використовуються регулярні вирази. Наведемо приклад використання оператору перевірки збігів ***m***:

```
if ($buf=~m/hello/) {
   print "hello user\n";
}
```

У цьому прикладі перевіряється, чи є у рядку $buf підрядок «hello». Якщо це так, то на стандартий вивід повернеться фраза 'hello user'. При цьому сам символ 'm' не є обов’язковим, тому оператор з цього прикладу можна записати просто як ~/hello/.

Можливості оператора перевірки збігу ***m*** покажемо на другому прикладі, що полягає в необхідності виділення числового значення з довільної фрази, наприклад "Perl був створений в 1986 році".

```
$text="Perl був створений в 1986 році";

if ($text=~m/.*\s(\d+)\s.*/) {
$num=$1;
}
```

У цьому прикладі змінна $num прийме значення 1986. Зазначимо, що змінні $1 … $9 використовуються для зберігання фрагментів рядка, що задовольнили відповідним фрагментам шаблона. У шаблоні фрагменти виділяються за допомогою дужок. Кожному фрагменту виділяється номер у тому порядку, у якому він розташований. Відповідна змінна буде містити його значення. Приведемо ще один приклад:

```
$string = "15 яблук, 7 груш, 5 горіхів";
while ($string =~ m/(\d+) (\w+)/g) {
print "$2: $1\n";
```

```
}
# Результат: яблук: 15
#            груш: 2
#            горіхів: 3
```

Наведемо приклад використання оператора підстановки тексту s. Допустимо у фразі "Hello, Basic" необхідно змінити слово "Basic" на слово "Perl". Фрагмент програми буде мати такий вигляд:

```
$text="Hello, Basic";
$text=~s/Basic/Perl/;
```

При використанні традиційних операторів аналогічний фрагмент виглядав би приблизно так:

```
$text="Hello, Basic";
$i=index($text,"Basic");
if ($i>-1) {
  $tmp=substr($text,0,$i);
  $text=$tmp."Perl";
}
```

Оператор ***tr*** також здійснює підстановку, але дещо іншого характеру – він використовується для заміни окремих символів деякими іншими (визначеними) символами.

Можливості кожного з цих операторів можна розширити за допомогою модифікаторів – символів, які дописуються до оператора.

Модифікатори для оператора перевірки збігів (порівняння):

```
g – знаходить всі найдені підрядки
i - ігнорує регістр символів у рядку
m – розглядає рядок як багаторядкове значення
s – розглядає рядок як однорядкове значення
x – дозволяє використовувати розширені регулярні вирази
```

Модифікатори для оператора підстановки:

```
e – обчислює підстановочний вираз перед підстановкою
g – знаходить всі знайдені підрядки
i – ігнорує регістр символів у рядку
m – розглядає рядок як багаторядкове значення
s - розглядає рядок як однорядкове значення
x - дозволяє використовувати розширені регулярні вирази
```

У наведеній нижче таблиці надані деякі вирази (кваліфікатори), які можна застосовувати у шаблонах регулярних виразів.

Кваліфікатори:	***Значення***	***Приклади застосування***
.	Відповідає довільному символу	print if /ab.c/;
[множина символів]	Відповідає довільному символу з даної множини	/[abc]d/; # відповідає ad, bd, cd
[^множина]	Заперечення множини символів	/[^xyz]/;
(....)	Групування елементів (а також збереження у змінних $1 $2 $3 ...)	/(xyz)*/ /([abc].[^xy]qwerty)/
(..|..|..)	Одна з альтернатив	
*****	Повторювання зразку 0 або більше разів	/.*/; # відповідає всьому
?	Повторювання 0 або 1 разів	/(http:\/\/)?.*\.cgi/
+	Повторювання 1 або більше разів	
{n,m}	Повторювання від n до m разів	
{n}	Повторювання точно n разів	
{n,}	Повторювання n та більше разів	
Спеціальні символи:		
\t \r \n ...	Управляючі символи: табуляції, повернення каретки, переведення рядку…	
\d	Відповідає цифрі, аналог [0-9]	
\D	Відповідає нецифровому символу, аналог [^0-9]	
\w	Відповідає літері	
\W	Відповідає нелітеральному символу	
\s	Відповідає пробільним символам (пробіл, табуляція, нові рядки…)	
\S	Відповідає не пробільному символу	

\b	Відповідає границі слова	$test1="this is test";
		$test2="wise";
		if ($test1=~/\bis\b/) { print "1"; } # відповідає
		if ($test2=~/\bis\b/) { print "2"; } # ні

\B	Відповідає не границі слова	/\Bis\B/ відповідає 'wise', але не 'is'

Для того, щоб розмістити в регулярному виразі будь-який спеціальний символ, необхідно поставити перед ним зворотній слеш «\».

Імена зі спеціальними значеннями

У мові Perl є імена, що мають спеціальні значення, наведемо деякі з них:

$_	- у цю змінну за замовченням здійснюється введення, присвоювання, до неї складаються результати пошуку за заданим шаблоном:
	`while(<>){...}`
	`або, що теж саме:`
	`while($_= <>) {...}`

$0	- містить ім’я файлу, у якому знаходиться програма, що виконується
$ARGV	- містить ім’я поточного файлу, з якого здійснюється зчитування
@ARGV	- містить масив аргументів командного рядку, які було передано програмі
%ENV	- містить поточне оточення процесу