ЛЕКЦІЯ 12

ПОШУК ДАНИХ.

Задача пошуку виникла пізніше, ніж задача сортування. Поява її спричинена головним чином появою автоматизованих інформаційних систем*АІС,* які будуються на основі комп‘ютерів.

Розглянемо звичайну пошукову задачу: як знаходити дані, що зберігаються в пам‘яті комп‘ютера за певною ідентифікацією.

Наприклад, в обчислювальній задачі потрібно знайти *f(x)*, маючи значення *x* і таблицю значень функції *f(x)*; у лінгвістичних задачах може цікавити англійський еквівалент відповідного слова.

Взагалі будемо припускати, що зберігається множина з *n* записів і необхідно визначити місцезнаходження відповідного запису. Розглянемо рядалгоритмів пошуку і наведемо їх короткі характеристики, а також рекомендації застосування.

12.1. Послідовний пошук

Найпростішим методом пошуку певного запису, що знаходиться у невпорядкованій таблиці, є послідовний перегляд кожного запису, доки не буде виявленого злиття (співпадіння) з шуканим. Такий метод називають послідовним або лінійним пошуком.

Алгоритм *L.*

Таблиця містить *n* записів R_1 ,..., R_n з ключами k_1 ,..., k_n . Необхідно знайти запис із заданим ключем *k*.

L1. Ініціалізація індексу проходження таблиці: *i=1*.

L2. Якщо $k=k_i$ - кінець успішний; якщо ні, перехід на L3.

L3. Зміна індексу *i =i+1*.

L4. Перевірка умови *i<n?* Так: перехід на L*2*. Ні - кінець неуспішний.

Ефективність алгоритму можна оцінити, підрахувавши кількість виконаних порівнянь тих значень ключів, які приймають участь у пошуку. Середня кількість таких порівнянь дорівнює величині *n*. Таким чином асимптоматична складність алгоритму - $O(n)$. У зв’язку з малою ефективністю в порівнянні з іншими алгоритмами лінійний пошук зазвичай використовують лише тоді, коли відрізок пошукової системи містить дуже мало елементів, однак лінійний пошук не вимагає додаткової пам’яті або аналізу функції, так що може працювати в потоковому режимі при безпосередньому отриманні даних з будь-якого джерела. Так само, лінійний пошук часто використовується у вигляді лінійнихалгоритмів пошуку максимуму (мінімуму).

Приклад програмної реалізації.

Вихідний код програмної реалізаціїалгоритму лінійного пошуку на мові програмування С має наступний вигляд:

```
int* search(int *arr, int cnt, int req)
{
  for (int i = 0; i < cnt; i++)
        if (arr[i] == req)
              return arr + i;
  return NULL;
}
```

12.2. Двійковий пошук

Другим відносно простим методом доступу до таблиць є метод бінарного пошуку. Двійковий пошук можливий утаблицях, організованих як дерева порівнянь. Елементи в таких таблицях зберігаються в лексикографічному (тобто алфавітному) порядку, або в порядку зростання числових значень ключів.

Загальна стратегія організації дерева полягає у поділі таблиці на підтаблиці, сукупності елементів на підмножини елементів. Тоді процедура пошуку елемента в дереві порівнянь нагадує пошук імені в телефонному довіднику. Для відшукання елемента потрібно спочатку вирішити, в якій підмножині знаходиться елемент, знайти цю підмножину і після цього продовжувати пошук.

12.2.1. Дерева порівнянь на векторній пам‘яті.

Розглянемо спочатку, як можна записати дерево у векторну пам‘ять.

Нехай ключі зберігаються у векторі*КЛЮЧ(i;j)*. Елементи у векторній пам‘яті повинні бути впорядкованими за значенням ключів.

При пошуку певного елемента зробимо коренем дерева вмістиме *КЛЮЧ(m),* де*m=[(i +j)/2]*- найбільше ціле, менше або рівне *(i+j)/2.* Тоді ліве піддерево розміщується у векторі *КЛЮЧ(i;m-1),* а праве - у векторі *КЛЮЧ(m+1;j).* Цей процес повторюється, доки не буде знайдений потрібний елемент, тобто доки процес не зійдеться у одній вершині, для якої обидва індекси у векторі *КЛЮЧ*будуть мати однакові значення.

Тоді алгоритм *D* пошуку ключа *K* у векторі *КЛЮЧ(i;j)* можна записати наступною послідовністю кроків:

Алгоритм D

D0. Ініціалізація індексів *i, j* .

D1. Повторювати кроки D2 - D5 доти, доки *i<j*.

D2. Обчислення індекса кореня дерева:*m=[(i+j)/2]*.

D3. Якщо *[КЛЮЧ (m)]= K, то REZ= m,*кінець.

D4. Інакше: якщо [*КЛЮЧ (m)] <K,*

то *i=m +1* (пошук справа); перехід на D2.

D5. Інакше *j=m-1 (*пошук зліва); перехід на D2;

Основна задача полягає у виключенні на кожному кроці з подальшого пошуку як можна більшої кількості елементів. Оптимальним рішенням буде вибір середнього елемента, оскільки при цьому буде виключена половина кількості елементів. Максимальне число порівнянь дорівнює $O(log_2 n)$, де n- кількість елементів даних.

12.3. Прямий пошук стрічки

Нехай задано масив S з n елементів тамасив P з m елементів, $m<=n$. Необхідно знайти перше входження масиву P умасив S . Алгоритм зводиться до повтору порівнянь окремих елементів.

Алгоритм R

R1. Встановити $i=1..\ n-m,\ j=1..m$.

R2. Якщо $S[i] = P[j]$, то зафіксувати перше співпадіння $k=i$, та перевірити співпадіння всього масиву P у масиві S.При першому неспівпадінні відмінити значення k та продовжити пошук.

R3. Кінець.

Кількість порівнянь дорівнює $n*m$**.**

Приклад.

Перша ітерація, співпав перший символ ‘a’=’a’:

a c a a b c

a a b

Друга ітерація, неспівпадіння ‘a’!=’c’:

a c a a b c

  a a b

Третя ітерація, співпали всі символимасиву P:

a c **a a b** c

  a a b

12.4. Алгоритм Кнута, Моріса і Прата пошуку в стрічці.

Маємо масив символів S з n елементів (текст) та масив P з m - взірець. Необхідно знайти перше входження взірця в масив. Схема алгоритму полягає у поступовому порівнянні взірця з текстом та зсуву по тексту на кількість співпавших символів у разі знайденого неспівпадіння. Алгоритмвикористовує просте спостереження, що коли відбувається неспівпадіння тексту і взірця, то взірець містить у собі достатньо інформації для того, щоб визначити де наступне входження може початися, таким чином пропускаючи кількаразову перевірку попередньо порівняних символів. Попередньо проводиться дослідження взірця та визначається максимально можливий зсув взірця по тексту D_{max}. Для цього вираховується відстань між однаковими символами у взірці, якщо такі є, інакше $D_{max} = m$.

Алгоритм КМП

КМП 1. Встановити *i=0*.

КМП 2. *j=0, d=1*.

КМП 3. Поки *j<m, i<n*

Перевірка: якщо *S[i]=P[j]*, то *d++,i++.j++* поки *d != m*.

КМП 4. Інакше встановити зсув взірця на*d* позицій по тексту якщо *d* <. D_{max} , або наD_{max} позицій якщо *d* > D_{max} .Перейти на крок КМП 2.

КМП 5. Кінець.

Складність алгоритму становить *O(n+m)*.

Приклад.

abcabdabcabc – текст,

abcabc – взірець, m=6

D_{max}=3

Перша ітерація:

abcabdabcabc

abcabc

кількість співпадінь *d=5<m, але d* > D_{max}, тому проводимо зсув взірця на 3 позиції по тексту.

Друга ітерація:

abcabdabcabc

abcabc

кількість співпадінь *d=2<m, d* <. D_{max}, проводимо зсув взірця на 2 позиції по тексту.

Третя ітерація:

abcabdabcabc

abcabc

співпадінь немає, отже *d=1, d* <. D_{max}, проводимо зсув взірця на 1 позицію по тексту.

Четверта ітерація:

abcabdabcabc

abcabc

кількість співпадінь *d=6=m*. Входження взірця знайдено.

12.5. Алгоритм Бойера - Мура пошуку в стрічці

Нажаль співпадіння зустрічаються значно рідше, ніж неспівпадіння. Тому виграш від використання алгоритму КМП в більшості випадків незначний. Інший алгоритму Бойера-Мура базується на наступній схемі: порівняння символів починається з кінця взірця, а не з початку. Нехай для кожного символу x взірця d_x - відстань від самого правого у взірці входження x до кінця взірця. Припустимо, знайдено неспівпадіння між взірцем та текстом. Тоді взірець можна зразу посунути вправо наd_x позицій що є більше або рівне 1. Якщо xу взірці взагалі не зустрічається, то посунути взірець можна зразу на всю його довжину m.

В даному алгоритмі розглядається поняття стоп-символа - це є символ в тексті, який є першим неспівпадінням тексту і взірця при порівнянні справа (з кінця взірця). Розглянемо три можливих ситуації:

1. 1. Стоп-символ у взірці взагалі не зустрічається, тоді зсув дорівнює довжині взірця m.
2. 2. Крайня права позиція kвходження стоп-символа у взірці є меншою від його позиції j у тексті. Тоді взірець можна зсунути вправо на k-j позицій так, щоб стоп-символ у взірці і тексті опинились один під одним.
3. 3. Крайня права позиція kвходження стоп-символа у взірці є більшою від його позиції j у тексті. Такий зсув ігнорується.

У третій ситуації необхідно знайти співпадіння взірця і тексту. Якщо у взірці є ще один такий самий символ, то необхідно зсунути взірець до співпадіння цього символу з символом в тексті. Інакше зсув дорівнює 1.

Приклад [4].

Hoola-Hoola girls like Hooligan - текст,

Hooligan - взірець.

Перша ітерація:

Hoola-Hoola girls like Hooligan

Hooliga**n**

‘n’!=’o’, символ ‘*o’* у взірці стоїть на 6-й позиції з кінця. Зсуваємо взірець на 6-1=5 позицій вправо.

Друга ітерація:

Hoola-Hoola girls like Hooligan

Hooliga**n**

‘n’!=’g’, символ ‘*g’* стоїть на 3-й позиції з кінця. Зсуваємо взірець на 3-1=2 позиції вправо

Третя ітерація:

Hoola-Hoola girls like Hooligan

Hooliga**n**

‘n’!=’r’, символа ‘r’ у взірці взагалі немає. Зсуваємо взірець на всю довжину, тоб то на 8-1=7 позицій.

Четверта ітерація:

Hoola-Hoola girls like Hooligan

Hooliga**n**

‘n’!=‘e‘ , символа ‘e‘ у взірці взагалі немає. Зсуваємо взірець на всю довжину, тоб то 8-1=7 позицій.

Hoola-Hoola girls like Hooligan

Hooliga**n**

‘n’!=’g’, символ ‘*g’* стоїть на 3-й позиції з кінця. Зсуваємо взірець на 3-1=2 позиції вправо

Hoola-Hoola girls like Hooligan

Hooligan

Всі символи співпали. Входження знайдено.

В найгіршому випадку алгоритм потребує*n* порівнянь, *n* - кількість символів у тексті. У найкращих обставинах, коли останній символ взірця завжди не співпадає з символом тексту, число порівнянь дорівнює *n* / *m*, *m* - кількість символів взірця.

12.6. Алгоритми з поверненням

Найбільш цікава область програмування - задачі так званого штучного інтелекту. Тут маємо справу з алгоритмами, які шукають рішення шляхом проб та помилок. Цей процес розділяється на окремі задачі, які переважно розглядаються в термінах рекурсії та вимагають дослідження кінцевого числа підзадач. Методом проб та помилок вибирається деякий ланцюжок висновків. У випадку невдач організовується перебір з поверненням для пошуку іншого ланцюжка. Для організації пошуку правильного ланцюжка використовується дерево пошуку. Зростання такого дерева часто буває експоненціальним. Прикладами таких задач є «Задача про 8 ферзів», «Задача виходу з лабіринту», «Задача про хід коня», «Задача про стабільні браки», «Задача оптимального вибору» та інші.