Розділ 8. Алгоритми пошуку. Послідовний пошук елементу. Бінарний пошук елементу. Послідовний пошук рядка

Пошук – знаходження будь-якої конкретної інформації у великому обсязі раніше зібраних даних.

Дані діляться на записи, і кожний запис має хоча б один ключ. Ключ використовується для того, щоб відрізнити один запис від іншого.

Метою пошуку є знаходження всіх записів, що підходять до заданого ключа пошуку.

Пошук елементу в масиві

Для знаходження інформації в неупорядкованому масиві потрібен послідовний пошук, що починається з першого елементу й закінчується при виявленні необхідних даних або при досягненні кінця масиву. Цей метод підходить для пошуку неупорядкованої інформації, але також можна використовувати його й на відсортованих даних. Однак якщо дані вже відсортовані, можна застосувати двійковий пошук, що знаходить дані швидше.

Послідовний пошук

Послідовний пошук дуже легко запрограмувати. Наведений нижче псевдокод ілюструє пошук у масиві символів відомої довжини, поки не буде знайдений елемент із заданим ключем:

```
function int LinearSearch (Array A, int L, int R, int
Key)
 begin
    for X = L to R do
        if A[X] = Key then
            return X
    return -1 // елемент не знайдено
end
```

Функція повертає індекс необхідного елементу, якщо такий існує, або -1 у противному випадку.

Зрозуміло, що послідовний пошук у середньому переглядає (*n / 2)* елементів. У найкращому разі він перевіряє тільки один елемент, а в найгіршому – *n*. Якщо інформація зберігається на диску, пошук може забирати тривалий час. Але якщо дані не впорядковані, послідовний пошук – єдино можливий метод.

Двійковий пошук

Якщо дані, у яких здійснюється пошук, відсортовані, для знаходження елементу можна застосовувати метод, що набагато перевершує попередній – *двійковий пошук*. У двійкового пошуку є й інші назви: *дихотомічний пошук*, *логарифмічний пошук*, *пошук розподілом навпіл*. У ньому застосовується метод половинного розподілу. Спочатку перевіримо середній елемент. Якщо він більший, ніж шуканий ключ, перевіримо середній елемент першої половини, у протилежному випадку – середній елемент другої половини. Будемо повторювати цю процедуру доти, поки шуканий елемент не буде знайдений або поки не залишиться чергового елементу.

Наприклад, знайдемо число 4 у масиві:

1 2 3 4 5 6 7 8 9

При двійковому пошуку спочатку перевіряється середній елемент – число 5. Оскільки воно більше, ніж 4, пошук триває в першій половині:

1 2 3 4 5

Середній елемент тепер дорівнює 3. Це менше, ніж 4, тому перша половина відкидається. Пошук триває в другій

частині:

4 5

Цього разу шуканий елемент знайдений.

У двійковому пошуку кількість порівнянь у найгіршому разі дорівнює $\log^2 n$.

У середньому випадку кількість порівнянь значно нижче, а в кращому – дорівнює 1. Двійковий пошук суттєво швидший за лінійний, відносно простий в реалізації і загальновживаний. Проте, в реальних програмах трапляються випадки помилкового використання лінійного пошуку в упорядкованих даних, що призводять до значного зменшення швидкодії.

Нижче наведений псевдокод функції двійкового пошуку.

```
//Ітеративна версія
BinarySearch(A[0..N-1], value)
 begin
    low = 0
    high = N - 1
    while (low <= high)
     begin
        mid = (low + high) / 2
        if (A[mid] > value)
            high = mid - 1
        else if (A[mid] < value)
            low = mid + 1
        else
        return mid // знайдено
     end
     return -1 // не знайдено
end
```

Одним із варіантів реалізації алгоритму є рекурсивна функція, що отримує масив, шукане значення та початковий і кінцевий індекси елементів в масиві. Далі наведено псевдокод рекурсивної версії двійкового пошуку.

```
//Рекурсивна версія
BinarySearch(A[0..(N – 1)], value, low, high)
 begin
    if (high < low)
        return -1 // не знайдено
        mid = (low + high) / 2
    if (A[mid] > value)
        return BinarySearch(A, value, low, mid - 1)
    else if (A[mid] < value)
        return BinarySearch(A, value, mid+1, high)
    else
        return mid // знайдено
 end
```

Розглянемо псевдокод ітеративної версії з доповненнями для двійкового пошуку, завдяки доповненням даний алгоритм працюватиме швидше.

```
//Ітеративна версія з доповненнями
int first = 0 // Перший елемент у масиві
int last = n  // Елемент, НАСТУПНИЙ ЗА останнім
int mid

if (n == 0)
    begin
         /* масив порожній */
    end
else if (a[0] > x)
    begin
    //елемент не знайдено; якщо вам треба вставити
його зі зсувом – то в позицію 0
    end
else if (a[n - 1] < x)
    begin
    //елемент не знайдено; якщо вам треба вставити
його зі зсувом - то в позицію n
    end
//Якщо переглядається непорожній фрагмент first < last
while (first < last)
    begin
```

```
// УВАГА! На відміну від більш простого
(first + last) / 2, цей код стійкий до переповнень
        mid = first + (last - first) / 2

        if (x <= a[mid])
            begin
                last = mid
            end
        else
            begin
                first = mid + 1
        end
    end

    if (a[last] == x)
        begin
        // Шуканий елемент знайдено. last – шуканий
індекс
    end
else
    begin
        //Шуканий елемент не знайдено. Але якщо треба
його вставити зі зсувом, то його місце – last
    end
```

Питання до розділу:

1. Що таке пошук?
2. Для пошуку якої інформації використовується послідовний метод?
3. В якому випадку використовують двійковий пошук?
4. Поясніть процедуру двійкового пошуку на власному прикладі.
5. Чому дорівнює мінімальна і максимальна кількість порівнянь при двійковому пошуку?