Лекция №4

АЛГОРИТМЫ ПОИСКА

Поиск – это процесс получения (извлечения) информации (данных) из некоторого информационного пространства с целью ее обработки.

Эффективность алгоритмов поиска информации зависит от упорядоченности и структурированности данных.

Предполагается, что данные имеют ключ, используемый при поиске (для простых значений ключ – сами значения).

Поиск информации основывается на операции сравнения, результатом которой является сообщение об успешном (попадание) и не успешном (промах) поиске.

Классификация алгоритмов поиска

1. Последовательный поиск
2. Бинарный поиск
3. Интерполяционный поиск
4. Поиск индексированием по ключу
5. Дерево бинарного поиска (binary search tree, BST)
6. Поразрядный поиск

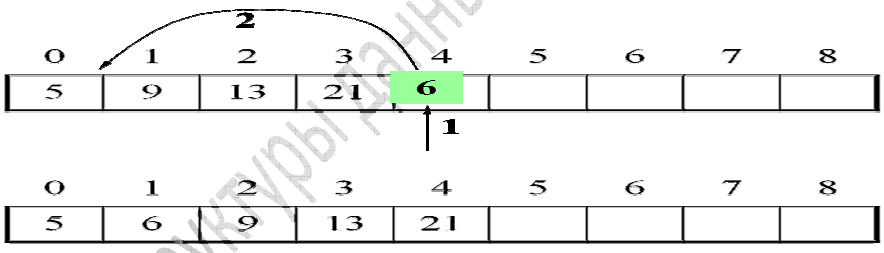
На линейных структурах данных

Последовательный поиск

Применяется для набора данных, организованного в виде массива или списка (упорядоченного или неупорядоченного).

Например, для упорядоченной вставки в массив данных:

- добавляем в конец (рис. а);
- выполняем последовательное сравнение с существующими элементами, двигаясь по ним слева направо;
- находим позицию для вставки (рис. б).



а)

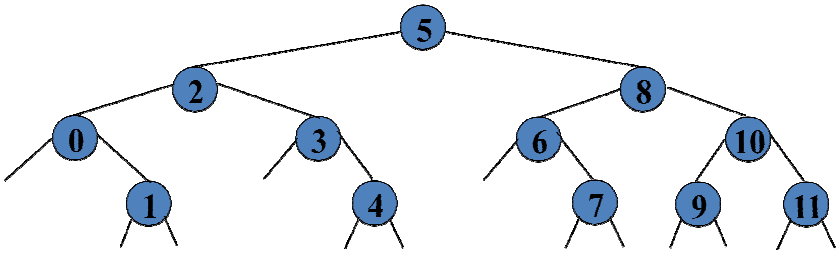
б)

Бинарный поиск

Если к упорядоченному набору данных применить принцип «разделяй и властвуй», то получим бинарный поиск.

Суть – набор данных разделяется на две части, далее определяется часть, к которой принадлежит искомые данные, далее поиск осуществляется в соответствующей части.

На рисунке приведен пример обращения к элементам набора данных в виде бинарного дерева (узлы содержат индексы элементов, размерность набора данных 12 элементов).



```
Find_Binary(Array, key)
    return call  Find_Elem(0, length(Array), key)

Find_Elem(left, right, key)
    if  left  >  right  then
            return  NULL                ► или индекс -1
    med  ←  (right + left)/2            ►делим на две части
    if  key = key(Array[med])  then
            return  Array[med]          ► или индекс med
    if  left  = right  then
            return  NULL                ► или индекс -1
    if  key < key(Array[med])  then
            return call  Find_Elem(left, med-1, key)
    else return  call  Find_Elem(med+1, right, key)
```

Пример выполнения бинарного поиска

```
-89     -67     -45     -23     -11     12      34      56      78      90
Enter key for find object -> -45
FIND -> -45  (count=3)
```

Интерполяционный поиск

Модифицированный бинарный поиск – получение информации о размещении искомого элемента в текущем интервале (выбор диапазона части набора данных определяется в зависимости от значения элемента).

$$m = le + (ri - le) \cdot \frac{1}{2} \qquad m = le + (ri - le) \cdot \frac{(key - mas[le].key)}{(mas[ri].key - mas[le].key)}$$

```
Find_ Interpolation(Array, key)
    return call  Find_Elem(0, length(Array), key)

Find_Elem(left, right, key)
    if  left  >  right  then
            return NULL                 ► или индекс -1
    med  ←  left + (key – key(Array[left])) * (right - left) / (key(Array[right]) - key(Array[left]))
    if  key = key(Array[med])  then
            return Array[med]           ► или индекс med
    if  left = right  then
            return NULL                 ► или индекс -1
    if  key < key(Array[med])  then
            return call  Find_Elem(left, med-1, key)
    else  return call  Find_Elem(med+1, right, key)
```

Пример выполнения интерполяционного поиска

```
-89     -67     -45     -23     -11     12      34      56      78      90
Enter key for find object -> -45
FIND -> -45   (count=1)
```

Поиск индексированием по ключу

Выполняется в два этапа:

1. Вычисление адреса (выполнение преобразование ключа в индекс массива).
2. Разрешение конфликтов (обработка элементов с одинаковым индексом).

Хэширование – это компромисс между временем выполнения и объемом памяти.

АНАЛИЗ АЛГОРИТМОВ

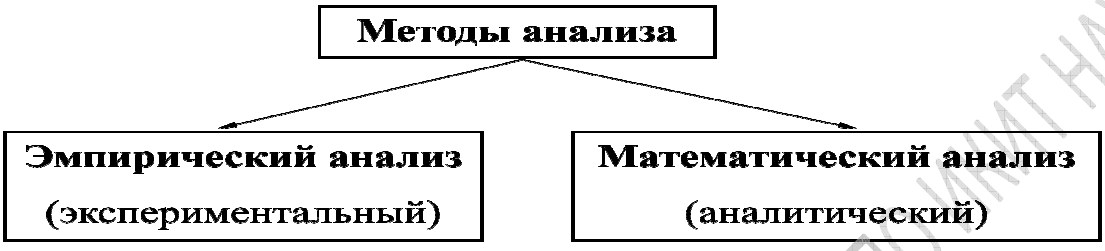
Анализ алгоритма – это методы по определению требуемых ресурсов для выполнения алгоритма (оценка его эффективности).

Под ресурсами понимается время выполнения и объем памяти.

Применяется анализ алгоритмов как на этапе их разработки, так и на этапе реализации.

Методы анализа применяются:

- для выбора наиболее эффективного алгоритма (разные алгоритмы, решающие одну и ту же задачу);
- для определения возможности использования алгоритма в некоторой системе.



Эмпирический анализ – вычисление времени выполнения путем запуска на исполнение реализаций алгоритмов.

Математический анализ – вычисление относительного времени выполнения путем построения математического выражения.

Эмпирический анализ

Допущения при эмпирическом анализе

I категория (реализация)

1. Алгоритмы реализованы на одном и том же языке программирования.
2. Реализация алгоритмов осуществлялась с одинаковой тщательностью.
3. Выполняемый код (программа) алгоритмов был получен с использованием одной и той же среды программирования.
4. Выполнение алгоритмов осуществлялось на вычислительной машине одной и той же архитектурой.

II категория (исходные данные)

1. Использование реальных данных (точное измерение времени выполнения).
2. Использование случайных данных (гарантия анализа именно алгоритма, измерение среднего времени выполнения).
3. Использование необычных данных (измерение наихудшего случая).

Рассмотрим пример эмпирического анализа алгоритмов последовательного и бинарного поиска.

Исходные допущения:

- поиск осуществляется в массиве целых положительных значений;
- размерность массива в диапазоне от 10^3 до 10^6;
- количество операций поиска элемента в диапазоне от 10^4 до 10^7;
- набор данных и поиск элементов один и тот же;
- массив инициализируется случайными значениями.

а) Последовательный поиск на неупорядоченном наборе данных:

```
Find_No_Sort(Array, value)
        for i ← 0 to length(Array) do
                if value = Array[i] then
                        return Array[i]
        return -1
```

1. Неуспешный поиск – проверяется N элементов (в среднем);
2. Неуспешный поиск – N элементов (в худшем);
3. Успешный поиск – N/2 элементов (в среднем).

(время выполнения пропорционально количеству элементов).

б) Последовательный поиск на упорядоченном наборе данных:

```
Find_Sort(Array, value)
        for i ← 0 to length(Array) do
                if value = Array[i] then
                        return Array[i]
                else if value < Array[i] then
                        return -1
        return -1
```

1. Неуспешный поиск – N/2 элементов (в среднем);
2. Неуспешный поиск – N элементов (в худшем);
3. Успешный поиск – N/2 элементов (в среднем).

(время выполнения пропорционально количеству элементов).

в) Бинарный поиск:

```
Find_Binary(Array, left, right, value)
        while left ≤ right do
                middle ← (left + right)/2
                if value = Array[middle] then
                        return Array[middle]
                if value < Array[middle] then
                        right ← middle – 1
                else    left ← middle + 1
        return -1
```

Проверяется не более $\mathbf{\log_2 N+1}$ элементов:

N=100 -> 50 -> 25 -> 12 -> 6 -> 3 -> 1

Результат сравнения

N	M = 10000	
	Последовательный	Бинарный
250	63	13
500	119	14
2500	570	16
12500	3073	17

N – количество элементов в наборе данных

M – количество выполненных поисков

Математический анализ

Математический анализ применяется если:

1. Отсутствует реализация алгоритма.
2. Необходимо предсказать время выполнения алгоритма в новой среде.

Приступая к математическому (аналитическому) анализу, алгоритм рассматривается с точки зрения сущности, которая имеет параметры, оказывающие влияние на время его выполнения (например, типы абстрактных операций, размерность набора данных).

Необходимо выделить один главный, который оказывает существенное влияние. Обычно он пропорционален величине обрабатываемого набора данных, т.е. параметр N – это размерность набора данных.

Зависимость времени выполнения можно выразить через главный параметр с использованием простых математических выражений, называемых **рост функциями**:

1 – постоянное время выполнения (почти все операции выполняются один раз или несколько).

Например, поиск индексированием по ключу.

N – линейное время выполнения (если каждый элемент подвергается обработке).

Например, последовательный поиск.

logN – логарифмическое время выполнения (задача разбивается на набор подзадач, с уменьшением размера задачи на каждом шаге на некоторый постоянный коэффициент, и решение определяется в одной из подзадач).

Например, бинарный поиск.

NlogN – время выполнения, пропорциональное NlogN (задача разбивается на подзадачи, решения которых затем объединяются).

Например, нисходящая сортировка слиянием.

$\mathbf{N^2}$ – квадратичное время выполнения (когда обрабатываются все пары элементов – двойные циклы).

Например, сортировка алгоритмом вставки в наихудшем случае.

$\mathbf{N^3}$ – кубическое время выполнения (тройные циклы).

Например, сортировка двумерного массива по значениям одной строки алгоритмом пузырька.

$\mathbf{2^N}$ – экспоненциальное время выполнения (прямое решение задачи).

Например, разложение целого числа на простые числа

Для построения аналитического выражения, определяющего время выполнения, необходимо определить все операции, их количество и время исполнения.

Полученное выражение можно преобразовать, выразив через рост функции.

Для приближенной оценки времени выполнения алгоритма при больших значениях N можно упростить математическое выражение отбросив слагаемые «низших порядков».

Оценка порядка роста времени выполнения алгоритма при больших объемах набора данных называется **асимптотической эффективностью**.

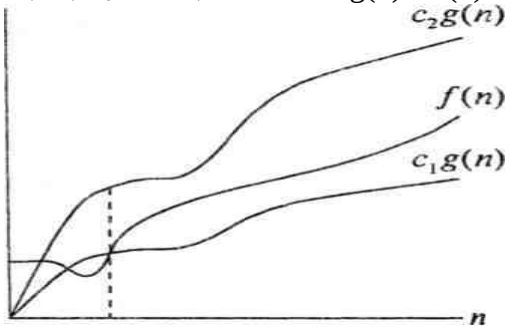
Математическая запись асимптотической оценки называется **«тета-нотацией» (θ-нотацией)**:

$$T(n) = \theta(g(n)),$$

где $\theta(g(n))$ – это множество функций, для которых справедливо высказывание:

$$f(n) \in \theta(g(n)),$$

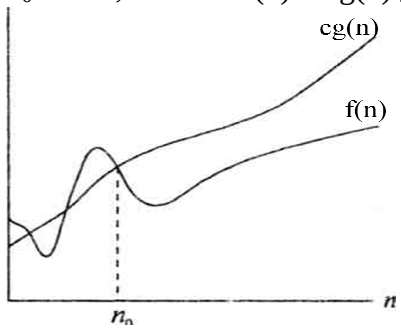
если существуют константы c_1, c_2, n_0 такие, что $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ для всех $n \ge n_0$



Определение асимптотической *верхней границы* называется **<u>О-нотацией</u>**:

$$f(n) \in O(g(n)),$$

если существуют константы с и n_0 такие, что $0 \le f(n) \le cg(n)$ для всех $n \ge n_0$



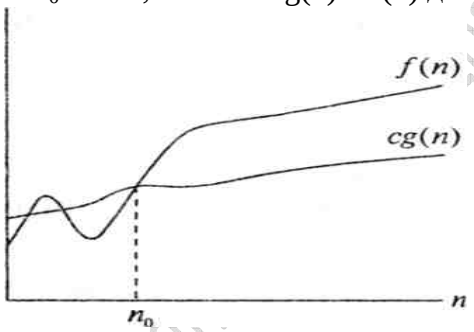
Через О-нотацию описывается наихудший случай.

Например, сортировка по возрастанию алгоритмом вставки массива, если исходно он упорядочен по убыванию.

Определение асимптотической *нижней границы* называется **<u>Ω-нотацией («омега-нотацией»)</u>**:

$$f(n) \in \Omega(g(n)),$$

если существуют константы с и n_0 такие, что $0 \le cg(n) \le f(n)$ для всех $n \ge n_0$



Через Ω-нотацию описывается наилучший случай.

Например, массив данных уже упорядочен.

<u>Пример вычисления асимптотической оценки для алгоритма сортировки вставками (N – размерность набора данных):</u>

```
Sort_Insert (Array)
    for  j ← 1 to length(Array) do          c1 → N
       i ← j                                c2 → N-1
       temp ← Array[j]                      c3 → N-1
       while  i>0  and temp<Array[i-1]  do  c4 → Σ_{j=1}^{N-1} k_j
              Array[i] ← Array[i-1]         c5 → Σ_{j=1}^{N-1} (k_j − 1)
              i ← i − 1                     c6 → Σ_{j=1}^{N-1} (k_j − 1)
    Array[i] ← temp                         c7 → N-1
```

Время выполнения операции

Количество операций, выраженное через N

где $k_j \rightarrow$ количество операций сравнения по итератору i для каждой итерации j.

Относительное время выполнения:

$$T(N) = c_1N + c_2(N-1) + c_3(N-1) + c_4 \sum_{j=1}^{N-1} k_j + c_5 \sum_{j=1}^{N-1}(k_j - 1) + c_6 \sum_{j=1}^{N-1}(k_j - 1) + c_7(N-1)$$

1. Наилучший случай (массив отсортирован в требуемом порядке)

$T(N) = c_1N + c_2(N-1) + c_3(N-1) + c_4(N-1) + c_7(N-1) =$

$= (c_1 + c_2 + c_3 + c_4 + c_7)N + (c_2 + c_3 + c_4 + c_7) = \mathbf{a \cdot N + b}$,

где $a = c_1 + c_2 + c_3 + c_4 + c_7$

$b = c_2 + c_3 + c_4 + c_7$

2. Наихудший случай (массив отсортированный в обратном порядке)

$$\sum_{j-1}^{N-1} k_j - \sum_{j-1}^{N-1} j - \frac{N(N+1)}{2} - 1 \qquad \sum_{j=1}^{N-1}(k_j - 1) = \sum_{j=1}^{N-1}(j-1) = \frac{N(N-1)}{2}$$

$$T(N) = c_1N + c_2(N-1) + c_3(N-1) + c_4\left(\frac{N(N+1)}{2} - 1\right) + c_5\left(\frac{N(N-1)}{2}\right) + c_6\left(\frac{N(N-1)}{2}\right) +$$

$$+ c_7(N-1) = \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)N^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} + c_7\right)N - (c_2 + c_3 + c_4 + c_7)$$

$T(N) = aN^2 + bN + c$, где $a = \frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}$, $b = c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} + c_7$

$c = c_2 + c_3 + c_4 + c_7$

N	aN+b	aN^2+bN+c
1000	9007	3010007
1500	13507	6765007
2000	18007	12020007
2500	22507	18775007
3000	27007	27030007
3500	31507	36785007
4000	36007	48040007
4500	40507	60795007
5000	45007	75050007
5500	49507	90805007
6000	54007	108060007
6500	58507	126815007
7000	63007	147070007
7500	67507	168825007
8000	72007	192080007
8500	76507	216835007
9000	81007	243090007
9500	85507	270845007
10000	90007	300100007