Алгоритм. Основные понятия

Понятие

Алгоритм – это формально описанная вычислительная процедура, получающая исходные данные и выдающая результат вычисления на выход.

Понятие через отображение

Алгоритм – это некая функция или отображение, которая определяется как

$$F: X \to Y$$

X – множество исходных данных

Y – множество значений

Виды алгоритмов

1. Механический или жесткий
2. Вероятностный
3. Эвристический
4. Линейный
5. Ветвящийся
6. Циклический
7. Вспомогательный

Свойства алгоритма

1. Детерминированность
2. Понятность
3. Результативность
4. Дискретность
5. Массовость
6. Конструктивность объектов

Детерминированность

Предписание, задающее алгоритм должно выполняться однозначно и последовательно для получения конкретного и однозначного результата

Понятность

Все действия должны быть однозначно поняты и выполнены исполнителем, т.е. должны принадлежать системе действий данного исполнителя

Результативность

Указывает на наличие таких исходных данных, для которых реализуемый по заданному алгоритму вычислительный процесс должен через конечное число шагов остановиться и выдать искомый результат

Дискретность

Алгоритм представляет собой упорядоченное конечное множество шагов для получения результата, то есть в любом алгоритме для каждого шага (кроме последнего), можно указать следующий за ним шаг.

Массовость

Каждый алгоритм предназначен для решения любой задачи из некоторого бесконечного множества однотипных задач

Конструктивность объектов

Исходные объекты, промежуточные и конечные результаты – это конструктивные объекты, которые могут быть построены целиком или допускают кодирование в каких-то алфавитах

Эффективность алгоритма

1. Скорость сходимости
2. Время выполнения
3. Удобство использования
4. Простота
5. Читаемость

Способы описания алгоритма

1. Словестное описание
2. Математическая запись
3. Графическая запись
4. Запись на псевдокоде
5. Запись на ЯП

Правильный алгоритм

Алгоритм считается **правильным**, если при любом допустимом входе он заканчивает работу и выдает результат, удовлетворяющий требованиям задачи.

Алгоритм называется **однозначным**, если для одних и тех же данных он дает один и тот же ответ

Неправильный алгоритм

1. Не завершает работу
2. Дает неверный результат

Типы ошибок в алгоритме

1. Синтаксические
   (a+b(a+c)*
2. Семантические
   *S*N, S – строка, N - число*
3. Логические
   Расстояние = скорость + время

Этапы разработки программы

1. Анализ постановки задач
2. Построение математической модели
3. Разработка алгоритма
4. Анализ алгоритма
5. Док-во правильности алгоритма
6. Реализация алгоритма
7. Тестирование программы
8. Оформление документации

Анализ постановки задач

Выяснить при этом входную и выходную информацию, определить идею решения, полноту входной информации, сформулировать, накладываемые на входную информацию ограничения (то есть описать спецификации)

Построение математической модели

Определить какие математические структуры больше подходят для задачи, существуют ли аналоги решения этой задачи. После чего проверить полноту математической модели, удобство работы с ней, реализации

Разработка алгоритма

На этом этапе необходимо тщательно обдумать алгоритм, уяснить его достоинства и недостатки, постараться избавиться от последних и усилить первые

Анализ алгоритма

Оценить какой это алгоритм(линейный, с ветвлениями, с циклами). Т.е. оценить его сложность по памяти и/или быстродействию.

Док-во правильности алгоритма

Предусматривает как доказательство конечности алгоритма, так и анализ его работы на всех ветвях на разных типах входных данных

Реализация алгоритма

Это означает необходимость написания программы на некотором языке программирования и отладка его на реальном компьютере

Тестирование программы

Т.е. провести ее как теоретическое, так и экспериментальное тестирование. Для алгоритма необходимо подготовить полный набор тестов (минимальное подмножество входных данных, покрывающее все случаи)

Оформление документации

Описать технические характеристики, структуру входных и выходных данных, правила использования программы, при необходимости – реализуемые алгоритмы и возможности модификации.

Размерность задачи

Размерностью задачи (l) будем называть количество информации достаточного для описания задачи

Сложность алгоритма

1. Временная сложность - $T(l)$
   время работы алгоритма
2. Емкостная сложность – $S(l)$
   объем памяти для реализации алгоритма

Пример

```
for( i=0; i < n; i++)
    for( j = n-1; j > i; j-- )
        if ( a[j-1] > a[j] )
            swap(a[j-1], a[j]);
```

$l = n+1$

$T(l) = cn^2$

$S(l) = n+3$

Мера сложности алгоритма

1. Сложность в худшем случае
2. Усредненная сложность

Сложность в худшем случае

1. Зная время работы в худшем случае, мы можем гарантировать, что выполнение алгоритма закончится за некоторое время, даже не зная, какой именно вход (данного размера) попадется.
2. На практике «плохие» входы (для которых время работы близко к максимуму) могут часто попадаться. Например, для базы данных плохим запросом может быть поиск отсутствующего элемента (довольно частая ситуация).
3. Время работы в среднем может быть довольно близко к времени работы в худшем случае. Пусть, например, мы сортируем случайно расположенные n чисел в помощью процедуры Сортировка вставками.

Асимптотические обозначения

Анализируя алгоритм, можно стараться найти точное число выполняемых им действий. Но в большинстве случаев достаточно оценить асимптотику роста времени работы алгоритма при стремлении размера входа к бесконечности (asymptotic efficiency)

f(n) = O(g(n))

f(n) = o(g(n)), если для всякого **ε>0** найдется такое $\mathbf{n_0}$, что при всех **n≥** $\mathbf{n_0}$ выполнено

$$\mathbf{0 \le f(n) \le \varepsilon g(n)}$$

Говорят, что **f(n)** имеет меньший порядок, чем **g(n)**.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

f(n) = O(g(n))

- $2n = o(n^2)$
- $\log_2 n = o(n)$
- $n^2/2 \neq o(n^2)$

f(n) = ω(g(n))

f(n) = ω(g(n)), если для всякого **c>0** найдется такое $\mathbf{n_0}$, что при всех $\mathbf{n \geq n_0}$ выполнено

$$\mathbf{0 \leq cf(n) \leq g(n)}$$

Говорят, что **f(n)** имеет больший порядок, чем **g(n)**.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

$f(n) = \omega(g(n))$

- $2n^2 = \omega(n)$
- $n = \omega(\ln n)$
- $n^2/2 \neq \omega(n^2)$

f(n) = Θ(g(n))

f(n) = Θ(g(n)), если найдутся такие $\mathbf{c_1}$, $\mathbf{c_2}$**>0** и такое $\mathbf{n_0}$, что при всех **n≥** $\mathbf{n_0}$ выполнено

$$\mathbf{c_1 g(n) \le f(n) \le c_2 g(n)}$$

Говорят, что **f(n)** имеет один порядок с **g(n)**.

$$c_1 \le \lim_{n\to\infty} \frac{f(n)}{g(n)} \le c_2$$

$f(n) = \Theta(g(n))$

- $n^2/2 = \Theta(n^2)$
- $n^2-n+2 = \Theta(n^2)$

Если $f(n) = \Theta(g(n))$ и $h(n) = \Theta(g(n))$, это не значит, что $f(n) = h(n)$

$f(n) = \Theta(g(n))$

- Отыскивая асимптотически точную оценку для суммы, мы можем отбрасывать члены меньшего порядка, которые при больших n становятся малыми по сравнению с основным слагаемым.
- Заметим также, что коэффициент при старшем члене роли не играет

f(n) = O(g(n))

f(n) = O(g(n)), если найдется **c>0** и $\mathbf{n_0}$, что при всех $\mathbf{n \geq n_0}$ выполнено

$$\mathbf{0 \leq f(n) \leq cg(n)}$$

Говорят, что **g(n)** есть верхняя оценка **f(n)**.

$f(n) = \Omega(g(n))$

$f(n) = \Omega(g(n))$, если найдется **$c>0$** и **n_0**, что при всех **$n \geq n_0$** выполнено

$$0 \leq cg(n) \leq f(n)$$

Говорят, что **g(n)** есть нижняя оценка **f(n)**.

Теорема

$$f(n) = \Theta(g(n))$$

равносильно условию

$$f(n) = O(g(n)) \text{ и } f(n) = \Omega(g(n))$$

Транзитивность

1. $f(n) = o(g(n))$ и $g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$
2. $f(n) = \omega(g(n))$ и $g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$
3. $f(n) = \Theta(g(n))$ и $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
4. $f(n) = O(g(n))$ и $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
5. $f(n) = \Omega(g(n))$ и $g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$

Рефлексивность

1. $f(n) = \Theta(f(n))$
2. $f(n) = O(f(n))$
3. $f(n) = \Omega(f(n))$

Симметричность

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

Обращение

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

Эффективность алгоритмов

1. Полиномиальные – $f(n) = O(n^m)$
2. Экспоненциальные - $f(n) = O(a^{p(n)})$
3. Факториальные $f(n) = O(n!)$

Полиномиальные алгоритмы

1. Постоянный - $f(n) = O(1)$
2. Линейный - $f(n) = O(n)$
3. Квадратный - $f(n) = O(n^2)$
4. Кубический - $f(n) = O(n^3)$

Примеры полиномиальных алгоритмов

К линейным алгоритмам относится алгоритм нахождения суммы десятичных чисел, состоящих из n1 и n2 цифр. Сложность алгоритма - O(n1 + n2).

Более общий класс полиномиальных алгоритмов –

- алгоритмы деления,
- извлечения квадратного корня,
- решение систем линейных уравнений и др

Примеры экспоненциальных алгоритмов

Задачи, в которых требуется построить:

множество всех подмножеств данного множества,

все полные подграфы некоторого графа
все поддеревья некоторого графа.

Другие типы задач

1. Кратчайшее решение «пятнашек» размера nxn
2. Задача коммивояжёра
3. Проблема раскраски графа
4. Сапер (игра)
5. Тетрис
6. составление расписаний, учитывающих определенные условия;
7. размещение обслуживающих центров (телефон, телевидение, срочные службы) для максимального числа клиентов при минимальном числе центров;
8. оптимальная загрузка емкости (рюкзак, поезд, корабль, самолёт) при наименьшей стоимости;
9. оптимальный раскрой (бумага, картон, стальной прокат, отливки), оптимизация маршрутов в воздушном пространстве, инвестиций, станочного парка;
10. задача распознавания простого числа

Числа Фибоначчи

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55….

$F_0=0, \quad F_1=1, \quad F_n=F_{n-1}+F_{n-2}, \quad n>1$

1. Рекурсивный алгоритм
2. Нерекурсивный алгоритм

Рекурсивный алгоритм

```
int Fibonacci(int n){
if(n==0)
    return 0;
if(n==1)
    return 1;
return Fibonacci(n-1)+Fibonacci(n-2);
}
```

Нерекурсивный алгоритм

```
int Fibonacci(int n){
if(n==0)
	return 0;
int prev = 0;
int current = 1;
for(int i=2;i<=n;++i)
	{
	int temp = current;
	current += prev;
	prev = temp;
	}
return current;
}
```

Числа Фибоначчи

1. Рекурсивный алгоритм
   $T(n) = \Omega(c^n)$ $c = 1{,}618\ldots$ - золотое сечение
   $S(n) = O(n)$
2. Нерекурсивный алгоритм
   $T(n) = O(n)$
   $S(n) = O(1)$