ПРЕДИСЛОВИЕ

Для разработки программ недостаточно просто знать какой-либо язык программирования. Процесс проектирования программ предполагает несколько этапов, начиная с постановки задачи и заканчивая получением эффективно и правильно работающей программы. Важнейшим начальным этапом является формализация поставленной задачи. На этом этапе строится математическая модель задачи с привлечением различных математических конструкций, таких как системы линейных или дифференциальных уравнений, множества, графы, матрицы и т. п. Выбор математических конструкций зависит от характера решаемой задачи.

После построения математической модели производится поиск методов решений исходной задачи в терминах этой модели. Для вычислительных задач метод решения формально описывается в форме *алгоритма*, который представляет собой конечную последовательность инструкций, имеющих четкий смысл и выполняемых с конечными вычислительными затратами за конечное время. Другими словами, алгоритм представляет собой формально описанную процедуру решения задачи, получающую некоторые исходные данные (*вход алгоритма*) и выдающую результат вычислений. Требования, предъявляемые к исходным данным и решению, определяются формулировкой задачи.

Алгоритм оперирует данными как исходными и результирующими, так и промежуточными, формируемыми самим алгоритмом для обеспечения его работы. Поэтому важным фактором при разработке алгоритмов является также выбор соответствующих структур данных. Часто от выбора структур данных зависит эффективность выполнения алгоритма.

В большинстве случаев поставленная перед разработчиком задача может иметь разные методы решения и, соответственно, различные алгоритмы решения. В связи с этим возникают вопросы оценки качества алгоритмов, сравнения характеристик различных алгоритмов и выбора на основе такого анализа наиболее эффективного алгоритма решения поставленной задачи.

Тема 1. Алгоритмы и их сложности

1.1. Псевдокод для записи алгоритмов

В учебном пособии для записи алгоритмов используется *псевдокод*, в котором логика алгоритма строится при помощи управляющих конструкций (операторов) языка программирования Паскаль. При этом для более удобного восприятия алгоритмов (а также для большего абстрагирования от конкретного языка программирования) синтаксис некоторых операторов псевдокода отличается от синтаксиса соответствующих операторов языка Паскаль.

Кроме операторов в псевдокоде используются также такие конструкции языков программирования, как переменные, выражения, условия, процедуры и функции. Псевдокод не имеет фиксированного набора типов данных, поэтому переменные могут представлять произвольный тип (целый, массив, строку, запись и т. д.). При этом описания типов в алгоритмах не приводятся, а тип данных какой-либо переменной и ее область действия становятся ясными из контекста.

Выражение и условие, представляющее собой любое выражение, принимающее значения **true** или **false**, записываются в виде традиционной математической конструкции, а не в соответствии с синтаксисом какого-либо языка программирования. В качестве выражения или условия может выступать также произвольная фраза на естественном языке, задающая правило вычисления значения выражения, например, фраза «случайный элемент из множества *S*» является выражением, а фраза «*x* простое число» – условием.

Процедуры (**procedure**) и функции (**function**) имеют тот же смысл, что и в языке Паскаль. Отличие заключается в том, что при определении процедуры или функции не указываются типы формальных параметров (а для функций – и тип самой функции), а также нет их разделения на параметры-переменные и параметры-значения. Эта информация становится ясной из контекста. Переменные, используемые в теле процедуры или функции, считаются локальными, если глобальность какой-либо переменной не оговорена особо или не следует из контекста.

Псевдокод использует следующие операторы:

1) оператор присваивания вида

переменная ← *выражение*

вычисляет значение выражения и присваивает его переменной. В ряде случаев для сокращения записи используется одновременное присваивание, например, оператор *var*1 ← *var*2 ← *expr* эквивалентен последовательности двух операторов присваивания: *var*1 ← *expr* и *var*2 ← *expr*. Время выполнения оператора присваивания определяется временем вычисления выражения и временем самого присваивания;

2) условный оператор вида

if *условие* **then** *оператор* **else** *оператор*

или

if *условие* **then** *оператор*

имеет тот же смысл, что и в Паскале. Синтаксическая двусмысленность, возникающая при использовании второго варианта условного оператора, разрешается стандартным образом: **else**-часть всегда сопоставляется ближайшей предшествующей **then**-части, которой еще не сопоставлена **else**-часть. Время выполнения условного оператора определяется как сумма времени, необходимого на вычисление значения условия и его проверки, и времени выполнения оператора в **then**-части или оператора в **else**-части в зависимости от того, какой из них выполнялся;

3) оператор варианта вида

$$\textbf{case}\begin{cases} \textit{условие}: \textit{оператор} \\ \vdots \\ \textit{условие}: \textit{оператор} \end{cases},$$

синтаксис которого существенно изменен по сравнению с синтаксисом соответствующего оператора в языке Паскаль для более удобного восприятия логики алгоритма. В данном операторе выполняется тот оператор, для которого выполняется условие. Время выполнения оператора варианта складывается из времени вычисления значения условия и его проверки и времени выполнения соответствующего оператора;

4) операторы цикла с предусловием вида

while *условие* **do** *оператор*

и постусловием вида

repeat *оператор* **until** *условие*

имеют тот же смысл, что и в языке Паскаль. Время выполнения оператора определяется как сумма времен всех исполняемых итераций цикла, каждая из которых состоит из времени вычисления и проверки условия выхода из цикла и времени выполнения оператора тела цикла. При этом следует учитывать, что в операторе **while** число вычислений и проверок условия выхода из цикла на единицу больше, чем число итераций, а в операторе **repeat** число вычислений и проверок условия выхода из цикла равно числу итераций;

5) оператор цикла с параметром вида

for *переменная* ← *выражение*1 **to** *выражение*2

by *выражение*3 **do** *оператор*.

Здесь *переменная* является параметром цикла; *выражение*1, *выражение*2 и *выражение*3 – соответственно начальное значение, конечное значение и размер шага изменения параметра цикла. Время выполнения оператора цикла **for** определяется аналогично времени выполнения оператора цикла **while** с учетом операций, связанных с параметром цикла, которые скрыты в операторе **for**. Если параметр цикла является переменной скалярного типа и размер шага определяется функциями следования или предшествования, используются операторы вида

for *переменная* ← *выражение*1 **to** *выражение*2 **do** *оператор*

или

for *переменная* ← *выражение*1 **downto** *выражение*2 **do** *оператор*

соответственно, которые имеют тот же смысл, что и в Паскале. В случаях, когда начальное и конечное значения и размер шага параметра цикла очевидны, используется более простая форма оператора. Например, запись

for $x \in S$ **do** *оператор*

означает, что оператор тела цикла выполняется для каждого элемента множества S, при этом число итераций равно мощности множества;

6) составной оператор имеет тот же смысл, что и в Паскале. В псевдокоде вместо операторных скобок **begin** и **end** используется односторонняя фигурная скобка для обрамления последовательности операторов, образующей составной оператор. Каждый оператор последовательности записывается в отдельной строке, поэтому специальный символ (например, точка с запятой в большинстве языков программирования) для отделения операторов друг от друга не указывается. Время выполнения составного оператора определяется суммой времен выполнения входящих в него операторов;

7) оператор возврата **return** используется в качестве заключительного оператора при определении процедур и функций. Наличие специального оператора возврата делает излишним применение операторных скобок **begin** и **end** для указания начала и конца определения процедуры или функции. В ряде случаев используется расширенная форма оператора возврата вида

return (*выражение*)**,**

где значение вычисленного выражения является значением функции. Время выполнения расширенного оператора возврата определяется временем вычисления выражения. Если же выражение отсутствует, то время выполнения представляет собой некоторую фиксированную величину;

8) оператор вызова процедуры или функции вида

имя_процедуры_или_функции (*фактические_параметры*)

имеет тот же смысл, что и в Паскале. Время выполнения определяется временем вызова с учетом передачи параметров и выполнения процедуры или функции;

9) произвольный оператор представляет собой некоторую фразу на естественном языке, определяющую действие. Такие операторы используются в случаях, когда детали реализации несущественны, очевидны или рассмотрены отдельно. Время выполнения определяется временем, необходимым на реализацию указанного действия.

Для включения в алгоритм различных комментариев, облегчающих его понимание, используется символ //, который начинает комментарий, идущий до конца строки.

Используются следующие соглашения об обозначениях:

а) все логарифмы, если специально не оговорено, берутся по основанию 2 (т.е. запись $\log x$ означает $\log_2 x$), тем более что асимптотические оценки времени выполнения алгоритмов типа $O(\log n)$ не зависят от основания логарифма, поскольку при изменении основания логарифма производится умножение на константу, т. е. $\log_a n = c \log_b n$, где константа $c = \log_a b$;

б) запись вида $\lceil x \rceil$ означает наименьшее целое, большее или равное x, а $\lfloor x \rfloor$ – целая часть x, т. е. наибольшее целое, меньшее или равное x; очевидно, что $x-1<\lfloor x \rfloor \le x \le \lceil x \rceil < x+1$ для любого x, а также $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$ для любого целого n, кроме того, для любого x и для любых целых положительных a и b справедливы равенства $\lceil \lceil x/a \rceil / b \rceil = \lceil x/(ab) \rceil$ и $\lfloor \lfloor x/a \rfloor / b \rfloor = \lfloor x/(ab) \rfloor$.

в) для более компактного представления алгоритмов операция обмена будет записываться в виде $x \leftrightarrow y$ (значения переменных x и y меняются местами), что эквивалентно последовательности из трех операторов присваивания: $t \leftarrow x$; $x \leftarrow y$; $y \leftarrow t$.

Предполагается, что для вычисления логических выражений применяется метод сокращенного вычисления. Если в выражении a **and** b (a и b – логические выражения) установлено, что a = **false**, то без вычисления b можно сказать, что все выражение будет иметь значение **false**. Аналогично для a **or** b, если a = **true**, то и значение всего выражения будет **true.**

Примеры записи алгоритмов

for $x \in V$ **do** $compnum(x) \leftarrow 0$
$c \leftarrow 0$ // счетчик компонент
for $x \in V$ **do if** $compnum(x) = 0$ **then** $\begin{cases} c \leftarrow c+1 \\ COMP(x) \end{cases}$

procedure $COMP(v)$
$compnum(v) \leftarrow c$
for $w \in \mathrm{Adj}(v)$ **do if** $compnum(w) = 0$ **then** $COMP(w)$
return

Построить пирамиду из $x_1, x_2, \ldots, x_n$
for $i \leftarrow n$ **downto** 2 **do** $\begin{cases} x_1 \leftrightarrow x_i \\ \text{Восстановить пирамиду} \\ \quad \text{в } x_1, x_2, \ldots, x_{i-1} \end{cases}$