Рекурсивные алгоритмы

Рекурсивные определения

Рекурсия - это способ определения объектов (понятий), при котором определение объекта строится, опираясь на само понятие объекта.

Существует несколько категорий задач, допускающих рекурсивные определения. Одна из категорий - математические формулы, определения которых рекурсивны по своей сути. Классический пример функции, определение которой может задаваться в рекурсивной форме F(n)=n!

$$F(n)=n*(n-1)*\ldots*1,\ 0!=1$$

Рекурсивное определение этой функции имеет вид:

$$F(n)=\begin{cases}1, n=0\\ n*F(n-1), n>0\end{cases}$$

где F(n-1)=(n-1)!

Второй пример - это описание синтаксиса конструкций языков программирования с помощью Бэкуса Наура формы (БНФ).

Пример. Определение идентификатора в языке Паскаль:
<буква>::=a|b|…|z, <цифра>::=0|1|…|9;
<идентификатор>::=<буква>|<идентификатор><буква>|<идентификатор><цифра>.

Рекурсивное определение состоит из двух независимых частей: *базовой* и *рекурсивной*.

Базовая часть не является рекурсивным утверждением, она задает определение для некоторой фиксированной группы объектов и определяет условие окончания рекурсии. В первом примере в базовой части утверждается, что объект 0!=1.

Рекурсивная часть определения записывается так, чтобы при цепочке повторных применений она редуцировалась бы к базе.

Рекурсивные процедуры и функции

Категории задач, допускающие рекурсивные определения:
1. Задачи, математические модели которых записываются в виде рекурсивно определенных функций.
2. Структура данных задачи определяется рекурсивным образом. Например: графы, деревья, списки.

3. Методы решения задачи допускают рекурсивное определение (игры, головоломки и др.)

Рекурсивные алгоритмы реализуются в виде подпрограмм, которые определяются в программе, как процедуры или функции.

Подпрограмма называется рекурсивной, если в ее определении присутствует прямо или косвенно вызов самой определяемой подпрограммы.

Явная рекурсия характеризуется существованием в определении подпрограммы оператора обращения к ней самой.

Неявная (косвенная) рекурсия характеризуется тем, что одна подпрограмма обращается к другой, которая через цепочку вызовов других подпрограмм рекурсивно обращается к первой.

Реализация рекурсивных подпрограмм.

Стек – структура данных, которая содержит упорядоченный набор элементов. Если в стек помещается новый элемент, он добавляется в конец упорядоченного набора (на вершину стека). При удалении элемента он тоже выбирается из конца набора (из вершины стека).

В основе реализации рекурсивной подпрограммы лежит структура данных, называемая стеком, в котором хранятся все неглобальные данные, участвующие во всех вызовах подпрограммы, при которых она еще не завершила свою работу.

Стек разбит на фрагменты, представляющие собой блоки последовательных ячеек. Каждый вызов подпрограммы использует фрагмент стека, длина которого зависит от вызывающей подпрограммы.

В общем случае при вызове процедурой А процедуры В происходит следующее:

1. В вершину стека помещается фрагмент нужного размера. В него входят следующие, данные:
   (а) указатели фактических параметров вызова процедуры В;
   (б) пустые ячейки для локальных переменных. определенных в процедуре В;
   (в) адрес возврата (АВ), т.е. адрес команды в процедуре А, которую следует выполнить после того, как процедура В закончит свою работу.

Если В - функция, то во фрагмент стека для В помещается указатель ячейки во фрагменте стека для А, в которую надлежит поместить значение этой функции (адрес значения).

2. Управление передается первому оператору процедуры В.
3. При завершении работы процедуры В управление передается процедуре А с помощью следующей последовательности шагов:
   (а) адрес возврата извлекается из вершины стека

(б) если B функция, то ее значение запоминается в ячейке, предписанной указателем на адрес значения;

(в) фрагмент стека процедуры B извлекается из стека, в вершину ставится фрагмент процедуры A;

(г) выполнение процедуры A возобновляется *с* команды, указанной в адресе возврата.

При вызове подпрограммой самой себя, т.е. в рекурсивном случае, выполняется та же самая последовательность действий.

Рекурсия и итерация

Не всегда рекурсивное решение задачи является лучшим, более простые решения могут быть найдены с помощью итерации. Класс функций, имеющих определение вида

$$F_n(x) = \begin{cases} G(x), \textit{если } n = 0, \\ H(F_{n-1}(x)), \textit{если } n > 0. \end{cases}$$

может всегда быть выражен итеративно так, что применение рекурсии необязательно.

Пример. Необходимо составить определение функции для вычисления значений неких полиномов, определение которых имеет следующий вид:

$$S_n(x) = \begin{cases} 0, \textit{если } n = 0, \\ 2x, \textit{если } n = 1, \\ \frac{2n}{n-1} S_{n-1}(x) + \frac{n-1}{2n} S_{n-2}(x), \textit{если } n>1 \end{cases}$$

По этому определению можно составить два определения функций на языке Паскаль.

Первое определение - рекурсивное - будет иметь следующий вид:

```
function p(n:integer; x:real):real

begin
{ базовое условие }
        if n=0 then p:=0
        else     if n=1 then p:=2*x
{ рекурсивная часть }
                  else p:=2*n/(n-1)*p(n-1,x)+(n-1)/(2*n)*p(n-2,x);
end;
```

Второе определение - итеративное - может быть более эффективным:

```
function p(n:integer; x:real):real;
{ n - степень полинома, x- значение аргумента)

var p0,p1,p2:real; { p0 - значение полинома S_{n-2}(x). p1 - значение полинома S_{n-1}(x),
                     p2 - значение полиномаS_n(x) .}
    i:integer;{ i - счетчик }
```

```
begin
        if n=0 then p:=0
        else    if n=1 then p:=2*x
                else {степень полинома n>1} begin
                        p0:=0;     {значение полинома при i=0}
                        p1:=2*x; {значение полинома при i=1}
                        for i:=2 to n do begin
                                p2:=2*n/(n-1)*p1+(n-1)/(2*n)*p0;
                                p0:=p1;
                                p1:=p2;
                        end;
                        p:=p2;
                 end;
end;
```

Если эффективность оценивать по количеству операций вычитания, которые необходимо сделать, чтобы вычислить некий полином степени n с помощью рекурсивного и итерационного алгоритмов, то итеративный алгоритм будет более эффективен, так как требуется выполнить всего n-1 операцию вычитания. Рекурсивный алгоритм требует большего числа операций вычитания и поэтому будет менее эффективен.