**Лекция 3. Рекурсия и рекурсивные алгоритмы**

**Рекурсия** - это одна из фундаментальных концепций в математике и программировании, рекурсия - это одна из форм мышления, этот мощное средство, позволяющее строить элегантные и выразительные алгоритмы.

Объект называется рекурсивным, если он содержит сам себя или определен с помощью самого себя.

Если процедура г. содержит явное обращение к самому себе, то она называется **явно рекурсивной.**

Если процедура г. содержит обращение к некоторой процедуры q, что в свою очередь содержит прямое или косвенное обращение к г, то р - называется **косвенно рекурсивной.**

Но рекурсивная программа не может вызвать к себе бесконечно, иначе она никогда не остановится, таким образом в программе (функции) должна присутствовать еще один важный элемент - так называемая терминальная условие, то есть условие при котором программа прекращает рекурсивный процесс.

Рекуррентности

Рекуррентность - это рекурсивное определение функции. Они широко распространены в математике. Возможно наиболее знакомая Вам с такого рода функций - это факториал.

Факториал - это произведение натуральных чисел от единицы до какого-нибудь данного натурального числа. Он определяется формулой:

N! = N ((N-1) !, для N> = 1 и 0! = 1.

Это прямо отвечает следующей рекурсивной программе:

function factorial (N: integer): integer;

begin

if N = 0 then

factorial = 1

else factorial = N \* factorial (N-1);

end;

Эта программа демонстрирует основные свойства рекурсивных программ программа вызывает сама себя (с меньшим значением аргумента), и у нее есть терминальная условие, при котором она прямо вычисляет результат.

Необходимо также помнить о том, что это - программа, а не формула: например, ни формула, ни программа не работают с отрицательными N, но пагубные последствия попытки сделать вычисления для отрицательного числа больше заметны для программы, чем для формулы. Вызов factorial (-1) приведет к бесконечному рекурсивному циклу. Поэтому перед вызовом данной программы нужно делать проверку условия не отрицательности.

**Второе**, хорошо известно рекуррентное соотношение - соотношение определяющего числа Фибоначчи. Числа Фибоначчи - это элементы числовой последовательности 1,1, 2, 3, 5, 8 ..., в которых каждый последующий элемент равен сумме предыдущих.

FN = FN-1 + FN-2, где N> = 2 и F0 = F1 = 1.

И снова, рекуррентность соответствует простой рекурсивной программе:

function fibonacci (N: integer): integer;

begin if N <= 1 then

fibonacci = 1 else

fibonacci = fibonacci (N-1) + fibonacci (N-2)

end;

Как мы увидим, много интересных алгоритмов можно легко реализовать с помощью рекурсивных программ, и многие разработчики алгоритмов хотят выражать алгоритмы рекурсивно. Но часто случается также и так, что столь же интересный алгоритм скрывается в деталях Не рекурсивные реализации. Давайте рассмотрим такой алгоритм:

const max = 25;

var i: integer;

F: array [0..max] of integer;

procedure fibonacci;

begin F [0] = 1;

F [1]: = 1;

for i: = 2 to max do

F [i] = F [i-1] + F [i-2];

end;

Эта программа вычисляет первые max чисел Фибоначчи, используя массив размера max. Этот метод называет **итерационным**.

Какой же метод лучше? Точных правил для выбора между рекурсивной и не рекурсивной версиями алгоритма решения задачи не существует. Краткость и выразительность большинства рекурсивных процедур упрощает их чтение и сопровождение. С другой стороны, выполнение рекурсивных процедур требует больших затрат и памяти, и времени процессора чем их итерационные аналоги.

Деление пополам.

Большая часть алгоритмов использует два рекурсивных вызова, каждый из которых работает примерно с половиной входных данных. В дизайне алгоритмов такое явление называют "деление пополам"; его часто используют для достижения существенной экономии.

В качестве примера, давайте рассмотрим задачу нанесения делений на линейку: на ней должна быть отметка в точке 1/2 ", метка немного короче через каждые 1/4", еще более короткая через 1/8 "и так далее.



Рисунок 1 - Линейка

Мы также предполагаем, что в наше распоряжение предоставлена ​​процедура mark (x, h) для нанесения метки высотой h единиц в линейку в позицию x. Центральная метка должна иметь высоту n единиц, метки в центрах левой и правой половинок - n-1 единиц, и так далее. Следующая рекурсивная программа - прямой путь достижения нашей цели:

procedure rule (l, r, h: integer);

var m: integer;

begin

if h> 0 then

begin m = (l + r) div 2;

mark (m, h) rule (l, m, h-1);

rule (m, r, h-1);

end;

end;

Идея этого метода заключается в следующем: для того, чтобы промаркировать линейку, мы сначала наносим длинную метку по ее середине. Это делит ее на две равные половины. Теперь мы наносим (более короткие) метки внутри каждой из этих половинок используя ту же процедуру.

Необходимо обращать особое внимание на терминальную условие рекурсивной программы - в противном случае она никогда не остановится! В вышеприведенной программе мы срока их, когда мы должны нанести метку высоты 0. Рассмотрим пример, в результате вызова rule (0, 8, 3). Мы ставим метку по середине и вызываем rule для левой половины, затем делаем тоже самое для левой половины, и так далее пока высота метки не станет равен 0. В конечном итоге мы возвращаемся с rule и размечаем правую половину подобным

rule (0,8,3) mark (4,3) rule (0,4,2) mark (2,2) rule (0,2,1) mark (1,1) rule (0,1,0) rule (1,2,0) rule (2,4,1) mark (3,1) rule (2,3,0) rule (3,4,0) rule (4,8,2) mark (6, 2) rule (4,6,1) mark (5,1) rule (4,5,0) rule (5,6,0) rule (6,8,1) mark (7,1) rule (6, 7,0)

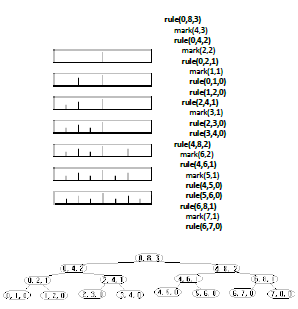


Рисунок 2 - Дерево рекурсивных вызовов для рисования линейки

Другой не рекурсивный алгоритм заключается в том, чтобы рисовать сначала кратчайшие метки, затем чуть менее короткие и так далее, как показано в следующей, довольно компактной, Нерекурсивные программе:

procedure draw (l, r, h: integer);

var i, j: integer;

begin j: = 1;

for i: = 1 to h do

begin

for x = 0 to (l + r) div j do

mark (l + j + x \* (j + j), i) j = j + j;

end;

end;

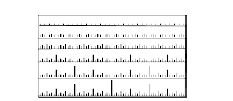


Рисунок 3 – Не рекурсивные рисования линейки

Сейчас создадим двухмерный узор, демонстрирует как простая рекурсия может дать решение о том, что кажется сложным.

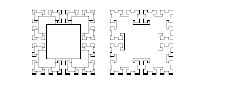


Рисунок 3 - Фрактальная звезда (слева) и ее очертания (справа)

uses graph, crt;

var r, gd, gm: Integer;

procedure star (x, y, r: integer);

begin if r> 2 then

begin star (x + r, y + r, r div 2)

star (x + r, y-r, r div 2)

star (x-r, y-r, r div 2)

star (x-r, y + r, r div 2)

bar (x-r, y-r, x + r, y + r)

end;

end;

begin

clrscr;

Gd = Detect;

write ( 'Enter the lenght:');

readln (r) InitGraph (Gd, Gm, 'с: \ bp \ bgi');

if GraphResult <> grOk then Halt (1);

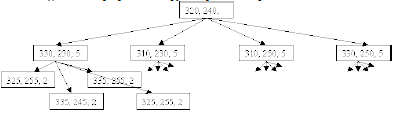
setbkcolor (0);

SetFillStyle (1, 9);

star (getmaxx div 2, getmaxy div 2, r) repeat until keypressed;

end.

Рисующий примитив здесь - просто процедура которая рисует квадрат размера 2r с центром в (x, y). Рекурсивно определенные геометрические узоры подобные этому называют фракталами. Если используется более сложный примитив для рисования и более сложные рекурсивные вызовы (особенно на вещественной и комплексной плоскостях), то можно разработать узоры поразительной красоты и сложности.



Рекурсия

Рекурсия- процесс повторения чего-либо само подобным способом. Например, вложенные отражение, произведенные двумя точно параллельными друг другу зеркалами, является одной из форм бесконечной рекурсии. Данный термин имеет более специальные значения в разных областях знаний-от лингвистики до логики.

Наиболее общее применение рекурсия находит в математике и информатике. Здесь она является методом определения функций, при котором обусловлена ​​функция применена в теле своего же собственного определения. При этом бесконечный набор случаев (значений функции) описывается с помощью конечного выражения, для некоторых случаев может ссылаться на другие случаи, если при этом не возникает циклов или бесконечной цепи ссылок. Фактически это способ определения множестве объектов через самого себя с использованием ранее заданных отдельных правил.

Определение, использует рекурсию, называется индуктивным. Одним из примеров подобного определения является аксиоматическое построение множества натуральных чисел.



Рисунок 4 - Треугольник Серпинского

-Факториал Целого неотрицательного числа n (обозначается n!) Определяется как при n> 0 и n! = 1 при n = 0

-число Фибоначчи определяются с помощью рекуррентного соотношения:

Первое и второе числа Фибоначчи равны 1

Для n> 2, n - e число Фибоначчи равно сумме (n - 1) -го и (n - 2) -го чисел Фибоначчи

-практически Все геометрические фракталы задаются в форме бесконечной рекурсии (например, треугольник Серпинского).

- Рекурсивный акроним: GNU (GNU Not Unix), PHP (PHP: Hypertext Preprocessor) и т.д.

Рекурсия в программировании

функции

В программировании рекурсия- вызов функции (процедуры) с ее же самой, непосредственно (простая рекурсия) или через другие функции (сложная рекурсия), например, функция A вызывает функцию B, а функция B- функцию A. Количество вложенных вызовов функции или процедуры называется **глубиной рекурсии.**

Преимущество рекурсивного определения объекта заключается в том, что такое конечное определение теоретически способно описывать бесконечно большое количество объектов. С помощью рекурсивной программы же описать бесконечное вычисления, причем без явных повторений частей программы.

Реализация рекурсивных вызовов функций в практически используемых языках и средах программирования, как правило, опирается на механизм стека вызовов - адреса возврата и локальных переменных функций записываются в стек, благодаря чему каждый следующий рекурсивный вызов этой функции пользуется своим набором локальных переменных и за счет этого работает корректно . Оборотной стороной этого достаточно простого по структуре механизма является то, что на каждый рекурсивный вызов требуется некоторое количество оперативной памяти компьютера, и при чрезмерно большой глубине рекурсии может наступить переполнения стека вызовов. Вследствие этого, обычно рекомендуется избегать рекурсивных программ, которые приводят (или в некоторых условиях могут приводить) к слишком большой глубины рекурсии.

Впрочем, есть специальный тип рекурсии, называемый «хвостовой рекурсией». Интерпретаторы и компиляторы функциональных языков программирования, поддерживающих оптимизацию кода (исходного и / или выполняется), автоматически превратят хвостовую рекурсию к итерации, благодаря чему обеспечивается выполнение алгоритмов с хвостовой рекурсией в ограниченном объеме памяти. Такие рекурсивные вычисления, даже если они формально бесконечные (например, когда с помощью рекурсии организуется работа командного интерпретатора, принимающей команды пользователя), никогда не приводят к исчерпанию памяти. Однако, далеко не всегда стандарты языков программирования четко определяют, каким именно условиям должна удовлетворять рекурсивная функция, чтобы транслятор гарантированно превратил ее в итерацию. Один из редких исключений - речь Scheme (диалект языка Lisp), описание которой содержит все необходимые сведения.

Любую рекурсивную функцию можно заменить циклом и стеком.

**данные**

Описание типа данных может содержать ссылки на самого себя. Подобные структуры используются при описании списков и графов. Пример описания списка (C ++):

class element\_of\_list

{

element\_of\_list \* next; / \* Ссылка на следующий элемент того же типа \* /

int data; / \* Какие данные \* /

};

Рекурсивная структура данных зачастую влечет за собой применение рекурсии для обработки этих данных.

Корекурсия

Корекурсия - в теории категорий и информатике тип операции, дуальный к рекурсии. Конечно корекурсия используется (вместе с механизмом ленивых вычислений) для генерации бесконечных структур данных.

общие замечания

Правило использования корекурсии на коданних дуальное правилу применения рекурсии на данных. Вместо сворачивания структуры данных исходя из начального значения аргумента, корекурсия разворачивает результат на основе начального значения аргумента. Необходимо отметить, что корекурсия создает потенциально бесконечные структуры данных, в то время как обычная рекурсия анализирует (разбирает) по необходимости конечные структуры данных. Обычная рекурсия неприменима к коданних, поскольку процесс анализа может никогда не остановиться. Соответственно, корекурсия не может делать данные, поскольку данные всегда конечны.

примеры

Пример использования механизма корекурсии языке Haskell (вычисления бесконечного списка чисел Фибоначчи):

fibs = 0: 1: zipWith (+) fibs (tail fibs)

Другой пример - вычисления бесконечного списка простых чисел:

primes = eratosthenes [2 ..]

where

eratosthenes (x: xs) = x: eratosthenes (filter ((/ = 0). ( 'mod' x)) xs)

Данная функция реализует алгоритм «решето Эратосфена», причем делает это самым непосредственным образом.

Приведенные примеры языке Haskell не совсем корректны, поскольку в языке нет идиомы коданних. В указанных примерах коданни только эмулируются с помощью бесконечного списка.