Научно-исследовательская работа по информатике

Исследование алгоритмов рекурсивных функций в решении специальных задач.

Выполнила:

Корниенко Дарья Сергеевна

учащаяся 9 класса

МБОУ СОШ № 25

им. Героя Советского Союза Остаева А.Е.

г. Владикавказа

Руководитель:

Маркина Вероника Александровна

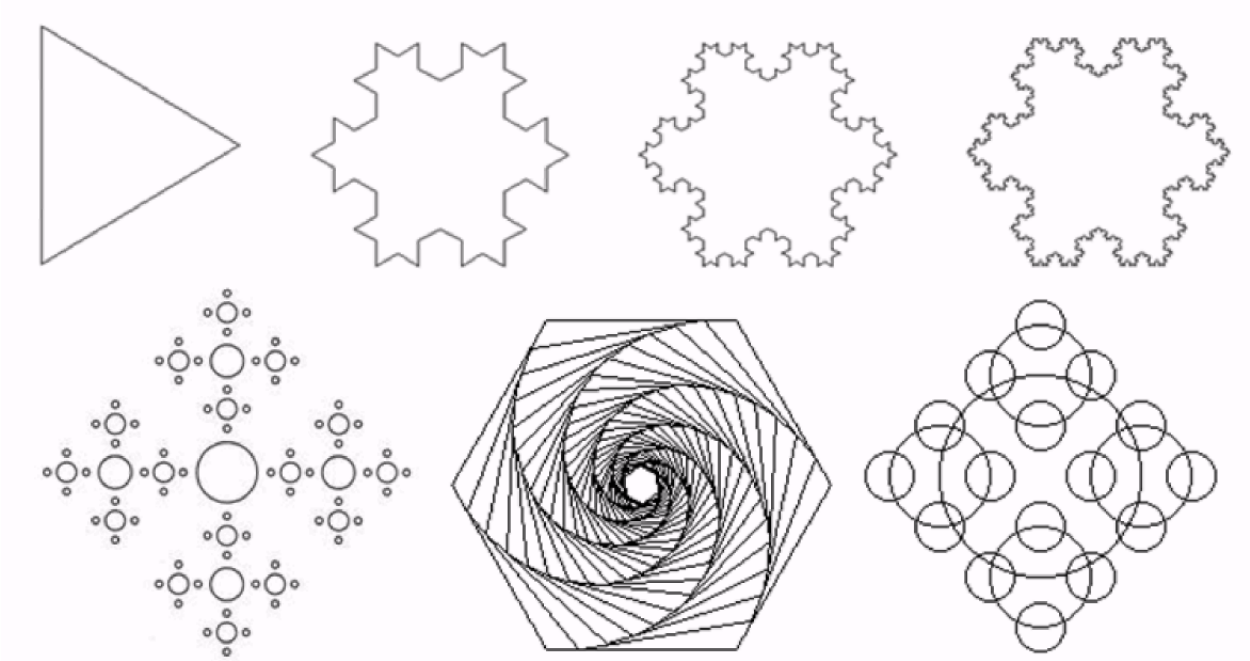
учитель информатики и ИКТ,

г. Владикавказ 2016г.

Содержание

Введение

В окружающем нас мире часто можно встретить объекты, обладающие самоподобием. То есть часть большого объекта в чем-то сходна с самим объектом. Например, ветка дерева повторяет форму и характер ветвления, схожие с самим деревом. Приведенные ниже графические объекты также обладают самоподобием. Такие объекты называются рекурсивными.



В связи с развитием информационных технологий, понятие рекурсивного алгоритма является не основным понятием теории алгоритмов, но одним из главных понятий современной науки. Более того, в XXI-ом веке, так называемый век информации, «алгоритм» является одним из важнейших факторов цивилизации.

Теория алгоритмов как самостоятельная наука появилась в 30-40х годах XX-века и имеет огромное значение во всех направлениях математических наук. Благодаря этой теории находят свои точные определения такие фундаментальные понятия алгоритм, доказуемость, сложность.

Теория алгоритмов вместе с математической логикой служит основой для построения теории вычислений. Они составляют теоретическую основу для проектирования и применения вычислительных устройств к плохо

формализуемым объектам. Именно, благодаря теории алгоритмов происходит внедрение математических методов в экономику, лингвистику, психологию, педагогику и другие гуманитарные науки.

Наиболее простые примеры алгоритмов известны нам из начальной школы. Это алгоритмы сложения, вычитания, умножения и деления целых чисел в десятичной системе счисления. Аналогичные алгоритмы известны и для произвольных *p*-иных систем счисления.

Объектом исследования является алгоритмы рекурсивной функции.

Цель исследования: возможности области применения в повседневной жизни.

Гипотеза: В технике процедурного программирования рекурсивность в построении подпрограмм проявляется в разработке управляющих структур, которые при выполнении обращаются сами к себе непосредственно или через цепочку других аналогичных структур.

Задачи проекта:

- изучить целесообразность применения рекурсии в программировании обусловленные спецификой задач;
- определить область памяти, предназначенной для хранения промежуточных значений локальных переменных при каждом следующем рекурсивном обращении.

Методы исследования:

- анализ информационных источников
- изучение возможностей программных средств для решения специфических задач.

Из истории

Тезис Чёрча-Тьюринга— фундаментальное эвристическое утверждение, существенное для многих областей науки, в том числе, для математической логики, теории доказательств, информатики, кибернетики, дающее интуитивное понятие о вычислимости. Это утверждение было высказано Алонзо Чёрчем и Аланом Тьюрингом в середине 1930-х годов.

В терминах теории рекурсии, это утверждение формулируется как совпадение классов вычислимых и частично рекурсивных функций. В этой формулировке часто упоминается как просто **тезис Чёрча**.

В терминах вычислимости по Тьюрингу, тезис гласит, что для любой интуитивно вычислимой функции существует вычисляющая её значения машина Тьюринга. Иногда в такой формулировке фигурирует как **тезис Тьюринга**. В виду того, что классы частично вычислимых по Тьюрингу и частично рекурсивных функций совпадают, утверждение объединяют в единый *тезис Чёрча — Тьюринга*.

Одним из свойств алгоритма является конструктивность – объекты из X, над которыми работает алгоритм, должны быть конструктивными (**конструктивный объект** – это такой объект, который может быть набран весь целиком и представлен нам для рассмотрения).

В силу этого свойства алгоритма между объектами счетного множества X и множества натуральных чисел N можно установить взаимно однозначное соответствие и в дальнейшем вместо объекта x є X рассматривать его номер или код объекта, являющийся натуральным числом.

Таким образом, можно рассматривать работу алгоритма не на множестве X, а на более удобном для изучения множестве натуральных чисел N. Поэтому в дальнейшем предметом наших рассмотрений будут служить преимущественно арифметические функции, т.е. функции, имеющие своими областями определения и множествами значений только множества натуральных чисел.

Основная часть.

При первичном осмыслении понятие рекурсии достаточно просто и не требует специальных знаний. Иногда на рекурсию смотрят как на наличие в определении объекта ссылки на сам объект или проявление свойств самоповторения (при этом сколь угодно малая часть объекта подобна всему объекту в целом). Общий случай проявления рекурсивности может быть сформулирован как наличие циклических взаимных обращений в определении объекта, которые в итоге замыкаются на сам объект.

В технике процедурного программирования рекурсивность в построении подпрограмм проявляется в разработке управляющих структур, которые при выполнении обращаются сами к себе непосредственно или через цепочку других аналогичных структур. Бесконечность и незавершенность таких обращений кажущаяся, так как при достижении определенных условий самовызовы завершаются. Во многих конкретных случаях простыми рассуждениями путем отслеживания значений одной или нескольких управляющих величин удается провести доказательство завершимости рекурсивных вычислений за конечное число шагов.

Рекурсивность в постановке задачи проявляется, если решение для общего случая сводится к аналогичным задачам для меньшего количества входных данных. В таком контексте под рекурсией понимают прием последовательного сведения решения некоторой задачи к решению совокупности "более простых" задач такого же класса и получению на этой основе решения исходной задачи.

Рекурсия в широком смысле - это определение объекта посредством ссылки на себя. **Рекурсия в программировании** - это пошаговое разбиение задачи на подзадачи, подобные исходной.

Рекурсивный алгоритм - это алгоритм, в определении которого содержится прямой или косвенный вызов этого же алгоритма.

В языках программирования процедурной парадигмы предусмотрено использование рекурсивных функций в решении задач.

Функция называется **рекурсивной,** если в своем теле она содержит обращение к самой себе с измененным набором параметров. При этом количество обращений конечно, так как в итоге решение сводится к базовому случаю, когда ответ очевиден.

В программировании выделяют прямую и косвенную рекурсию. *Прямая рекурсия* предусматривает непосредственное обращение рекурсивной функции к себе, но с иным набором входных данных. *Косвенная (взаимная) рекурсия* представляет собой последовательность взаимных вызовов нескольких функций, организованная в виде циклического замыкания на тело первоначальной функции, но с иным набором параметров.

Для решения задач рекурсивными методами разрабатывают следующие этапы, образующие **рекурсивную триаду:**

- параметризация - выделяют параметры, которые используются для описания условия задачи, а затем в решении;
- база рекурсии - определяют тривиальный случай, при котором решение очевидно, то есть не требуется обращение функции к себе;
- декомпозиция - выражают общий случай через более простые подзадачи с измененными параметрами.

Целесообразность применения рекурсии в программировании обусловлена спецификой задач, в постановке которых явно или опосредовано указывается на возможность сведения задачи к подзадачам, аналогичным самой задаче. При этом эффективность рекурсивного или итерационного способов решения одной и той же задачи определяется в ходе анализа работоспособности программы на различных наборах данных. Таким образом, рекурсия не является универсальным способом в

программировании. Ее следует рассматривать как альтернативный вариант при разработке алгоритмов решения задач.

Повысить эффективность рекурсивных алгоритмов часто представляется возможным за счет пересмотра этапов триады. Например, введение дополнительных параметров, не оговоренных в условии задачи, в реализации декомпозиции могут быть применены другие соотношения, а также можно организовать расширение базовых случаев с сохранением промежуточных результатов.

Область памяти, предназначенная для хранения всех промежуточных значений локальных переменных при каждом следующем рекурсивном обращении, образует *рекурсивный стек*. Для каждого текущего обращения формируется локальный слой данных стека (при этом совпадающие идентификаторы разных слоев стека независимы друг от друга и не отождествляются). Завершение вычислений происходит посредством восстановления значений данных каждого слоя в порядке, обратном рекурсивным обращениям. В силу подобной организации количество рекурсивных обращений ограничено размером области памяти, выделяемой под программный код. При заполнении всей предоставленной области памяти попытка вызова следующего рекурсивного обращения приводит к ошибке переполнения стека.

Пример 1. Для целого неотрицательного числа n найдите его факториал.

Разработаем рекурсивную триаду.

Параметризация: n- неотрицательное целое число. База рекурсии: для n=0 факториал равен 1. Декомпозиция: n!=(n-1)!*n.

```
long factor (int n)
{
if (n<0) return 0;        // для отрицательных чисел
    if (n==0) return 1;       // базовый случай: n=0
    return factor(n-1)*n;     // общий случай (декомпозиция)
}
```

Рассмотрим задачи, для которых можно предложить рекурсивные алгоритмы решения, в то время как итерационные алгоритмы были бы сложными и искусственными. (приложение).

Заключение.

1.Свойством рекурсивности характеризуются объекты окружающего мира, обладающие самоподобием.

2. Рекурсия в широком смысле характеризуется определением объекта посредством ссылки на себя.
3. Рекурсивные функции содержат в своем теле обращение к самим себе с измененным набором параметров. При этом обращение к себе может быть организовано через цепочку взаимных обращений функций.
4. Решение задач рекурсивными способами проводится посредством разработки рекурсивной триады.
5. Целесообразность применения рекурсии в программировании обусловлена спецификой задач, в постановке которых явно или опосредовано указывается на возможность сведения задачи к подзадачам, аналогичным самой задаче.
6. Область памяти, предназначенная для хранения всех промежуточных значений локальных переменных при каждом следующем рекурсивном обращении, образует рекурсивный стек.
7. Рекурсивные методы решения задач нашли широкое применение в процедурном программировании.

Список использованных источников

1. http://function-x.ru/cpp_rekursivnye_funkcii.html

2. http://forum.vilkam.ru/index.php?topic=63912.0

3. https://habrahabr.ru/post/139755/

Приложения.

Ключевые термины

База рекурсии - это тривиальный случай, при котором решение задачи очевидно, то есть не требуется обращение функции к себе.

Декомпозиция - это выражение общего случая через более простые подзадачи с измененными параметрами.

Косвенная (взаимная) рекурсия - это последовательность взаимных вызовов нескольких функций, организованная в виде циклического замыкания на тело первоначальной функции, но с иным набором параметров.

Параметризация - это выделение из постановки задачи параметров, которые используются для описания условия задачи и решения.

Прямая рекурсия - это непосредственное обращение рекурсивной функции к себе, но с иным набором входных данных.

Рекурсивная триада - это этапы решения задач рекурсивным методом.

Рекурсивная функция - это функция, которая в своем теле содержит обращение к самой себе с измененным набором параметров.

Рекурсивный алгоритм - это алгоритм, в определении которого содержится прямой или косвенный вызов этого же алгоритма.

Рекурсивный стек - это область памяти, предназначенная для хранения всех промежуточных значений локальных переменных при каждом следующем рекурсивном обращении.

Рекурсия в программировании - это пошаговое разбиение задачи на подзадачи, подобные исходной.

Рекурсия в широком смысле - это определение объекта посредством ссылки на себя.

Пример 3. **Задача о коэффициентах Безу**

Для любых натуральных чисел п и т найдите коэффициенты Безу, то есть такие целые а и Ь, что выполняется равенство: nod(n,m)=a•n+b-m (где nod(n,m) - наибольший общий делитель п и т).

Параметризация.

т, п - данные натуральные числа, неизменяемые параметры;

d - наибольший общий делитель данных чисел, неизменяемый параметр;

bm, bn - коэффициенты Безу при п и т соответственно, эти параметры меняются при очередном рекурсивном вызове функции.

База рекурсии. Если при очередном обращении к функции с передаваемыми параметрами выполняется равенство d=m-bm-n-bn, то коэффициенты Безу найдены. Требуется вывести линейную комбинацию.

Декомпозиция. Если равенство не выполняется, то инкрементно увеличиваем коэффициент при меньшем из чисел (п или т). Следующий вызов рекурсивной функции выполняется с измененным набором отдельных параметров. При этом снова проверяется база рекурсии, и рекурсивный алгоритм повторяется (либо достигается база и функция завершает работу, либо выполняется декомпозиционный переход).

```
//Коэффициенты Безу

#include "stdafx.h"

#include <iostream>

using namespace std;

int nod(int m, int n);

void bezu(int d, int m, int n, int bm, int bn);
```

```
int _tmain(int argc, _TCHAR* argv[]){
    int x,y,del,buf;
    printf("Задача нахождения коэффициентов Безу");
    printf("\nВведите два натуральных числа:");
    printf("\nX= ");
    scanf("%d",&x);
    printf("Y= ");
    scanf("%d",&y);
    if (x < y) {buf = x; x = y; y = buf;}
    del=nod(x,y);
    printf("\nЛинейная комбинация:\n");
    bezu(del,x,y,l,l);
    system("pause");
    return 0; }
//функция нахождения наибольшего общего делителя двух чисел int nod(int m, int n){
    if (m%n==0) return n;
    return nod(n,m%n);
 }
//функция нахождения и вывода на экран коэффициентов Безу
void bezu(int d, int m, int n, int bm, int bn)
{ int pm,pn;
 pm = m * bm;
pn = n * bn;
    //проверка базы рекурсии (выполнение линейной комбинации) if (d == pm - pn)
        printf ("%d = %d*%d - %d*%d", d, bm, m, bn, n); //декомпозиция
         else {
        bn++;
        pn=n*bn;
```

```
        /*если произведение pm больше, чем pn, то порядок
        параметров сохраняется*/
        if (pm > pn) bezu(d, m, n, bm, bn);
        /*если произведение pm меньше, чем pn, то порядок
        параметров изменятеся*/
    else bezu(d, n, m, bn, bm); } }
```

Пример 4. Задача о Ханойских башнях

Ханойская башня является одной из популярных головоломок XIX века. Даны три стержня, на один из которых нанизаны n колец, причем кольца отличаются размером и лежат меньшее на большем. Задача состоит в том, чтобы перенести пирамиду из n колец за наименьшее число ходов с одного стержня на другой. За один раз разрешается переносить только одно кольцо, причём нельзя класть большее кольцо на меньшее.

Существует древнеиндийская легенда, согласно которой в городе Бенаресе под куполом главного храма, в том месте, где находится центр Земли, на бронзовой площадке стоят три алмазных стержня. В день сотворения мира на один из этих стержней было надето 64 кольца. Бог поручил жрецам перенести кольца с одного стержня на другой, используя третий в качестве вспомогательного. Жрецы обязань соблюдать условия:

1. переносить за один раз только одно кольцо;
2. кольцо можно класть только на кольцо большего размера или на пустой стержень.

Согласно легенде, когда, соблюдая все условия, жрецы перенесут все 64 кольца, наступит конец света. Для 64 колец это 18 446 744 073 709 551 615 перекладываний, и, если учесть скорость одно перекладывание в секунду, получится около 584 542 046 091 лет, то есть апокалипсис наступит нескоро.

На рисунке (рис. 3.2) изображена ситуация, иллюстрирующая перекладывание 7 колец со стержня A на B через вспомогательный C.

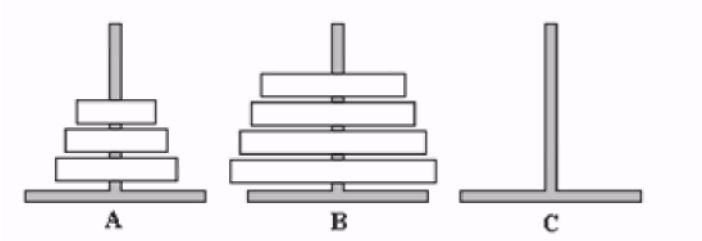


Рис. 3.2. Ситуация при переносе семи колец

Кольцо со стержня A можно перенести на стержень B или C, кольцо со стержня B можно перенести на стержень C, однако, нельзя перенести его на стержень A.

Задача состоит в том, чтобы определить последовательность минимальной длины переноса колец. Решением задачи будем считать последовательность допустимых переносов, каждый из которых имеет вид: A—»B, A—ИЗ, B—*A, B—ИЗ, C—*A, C—*B. Если кольцо всего одно, то задача решается за один перенос A—*B. Для перемещения двух колец требуется выполнить три действия: A—из, A—»B, C—*B. Решение задачи для трех колец содержит семь действий, для четырех - 15.

Напишем рекурсивную функцию, которая находит решение для произвольного числа колец.

Параметризация. Функция имеет четыре параметра, первый параметр -число переносимых колец, второй параметр - стрежень, на который первоначально нанизаны кольца. Третий параметр функции - стержень, на который требуется перенести кольца, и, наконец, четвертый параметр - стержень, который разрешено использовать в качестве вспомогательного.

База рекурсии. Перенос одного стержня.

Декомпозиция. Последовательность переноса колец изображена на рисунке (рис. 3.3).

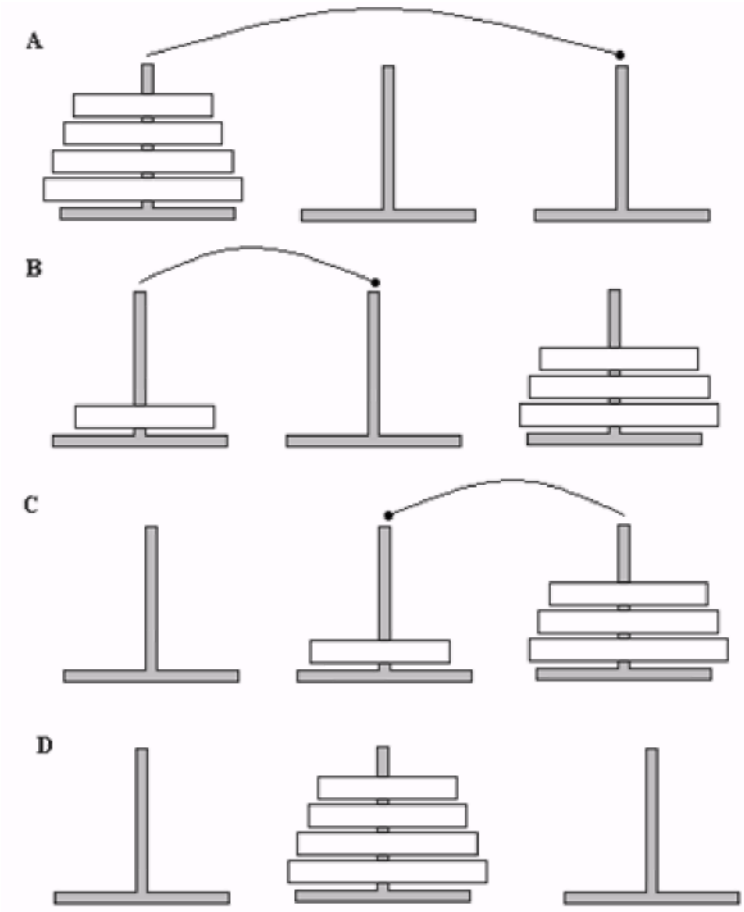


Рис. 3.3. Схема решения задачи о Ханойских башнях для четырех колец

Чтобы перенести n колец со стержня A на стержень B, используя стрежень C в качестве вспомогательного, можно поступить следующим образом:

- перенести n-1 кольцо со стержня A на C, используя стержень B в качестве вспомогательного стержня;
- перенести последнее кольцо со стержня A на стержень B;
- перенести n-1 кольцо со стержня C на B, используя стержень A в качестве вспомогательного стержня.

При переносе n-1 кольца можно воспользоваться тем же алгоритмом, т.к. на нижнее кольцо с самым большим диаметром можно просто не обращать внимания. Перенос одного кольца в программе выражается в том, что выводится соответствующий ход.

```
//Ханойские башни

#include "stdafx.h"

#include <iostream>

using namespace std;

int hanoj(int n, char A, char B, char C);

//Объявление функции перемещения колец с A на C через B

int _tmain(int argc, _TCHAR* argv[ ] ){

    char x='A',y='B',z='C';

    int k,h;

    printf("Задача о Ханойских башнях");

    printf("\nВведите количество колец: ");

    scanf("%d",&k);

    h=hanoj(k,x,z,y);

    printf("\nКоличество перекладываний равно %d",h);

    system("pause");

    return 0; }

//Описание функции перемещения колец с A на C через B int
hanoj(int n, char A, char B, char C){ int num;

    if (n == 1) {printf("\n   %c -> %c", A, C); num = 1;} else {
```

```
num=hanoj(n-1, A, C, B) ; printf("\n
%c -> %c", A, C); num++;
 num+=hanoj(n-1, B, A, C);
}
return num;
}
```