СРАВНИТЕЛЬНАЯ ОЦЕНКА ЭФФЕКТИВНОСТИ ИТЕРАТИВНОГО И РЕКУРСИВНОГО АЛГОРИТМОВ ПРИ ИЗУЧЕНИИ РАЗДЕЛА «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ» В КУРСЕ «ИНФОРМАТИКА»

Сергей А.

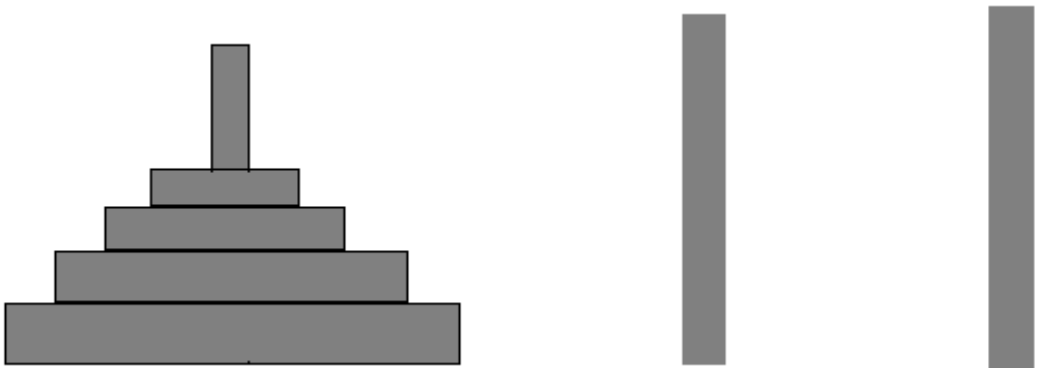
Белорусский национальный технический университет

Comparative analysis of efficiency iteration and recursive algorithms in learning unit "Machinery and data structures" in course "Informatics".

Введение

Как отмечается в учебниках зарубежных авторов Н. Вирт[1] и К. Браунси[2] посвященных различным аспектам применения алгоритмов к решению классических задач информатики, рекурсивные алгоритмы являются эффективным инструментом, позволяющим получать изящные лаконичные решения для достаточно сложных задач. К таким эталонным задачам относится задача «Ханойские башни», краткое содержание которой здесь прилагается.

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



Так же существует легенда, что в одном буддийском монастыре монахи уже тысячу лет занимаются перекладыванием колец. Они располагают тремя пирамидами, на которых надеты кольца разных размеров. В начальном состоянии 64 кольца были надеты на первую пирамиду и упорядочены по размеру. Монахи должны переложить все кольца с первой пирамиды на вторую, выполняя единственное условие – кольцо нельзя положить на кольцо меньшего размера при перекладывании можно использовать все три пирамиды. Монахи перекладывают одно кольцо за одну секунду. Как только они закончат свою работу, наступит конец света.

Итеративный алгоритм.

Существует несколько вариантов решения задачи. В этой статье мы рассмотрим сравнение рекурсивного и итеративного методов в решении задачи.

Начнем с итеративного алгоритма.

В приведенном ниже итеративном алгоритме моделируются перебрасывания дисков.

В основе этого метода лежат следующие закономерности:

1. наименьший диск берется каждый второй раз.
2. нельзя накладывать диски одной четности друг на друга.

Для хранения информации о расположении дисков используется двумерный массив, где в элементе a[0][x] хранится информация о количестве колец на стержне x, а значение a[i][x] соответствует размеру кольца под номером i на стержне x.

Принцип работы: сначала мы перекидываем наименьший диск, затем ищем подходящий по закономерностям, описанным выше, диск и стержень, куда можно его (диск) перебросить.

Запишем программу на языке C++.

```
#include <iostream>
#include <conio.h>
int seek1();
void sdvig(int c,int v,int m), seek();
int a[65][4],i,n,c,v,m,j,e;

/*n- общее количество дисков
  e- стержень на котором находится наименьший диск.
  i,j - переменные для обработки массива
  c,v,m- переменные в которых хранится информация о передвигаемых дисках */
main()
{cin>>n;  //Здесь мы указываем сколько дисков будем перекидывать

  /*последующие действия начинают заполнение массива (заполнение 1 стержня и указывание количества дисков на нем) таким образом, чтобы 'нижнему' диску соответствовало тах значение. Также в последующих блоках реализованы два начальных перекидывания диска (для четного и нечетного случая).  */
if (n!=1) //когда у нас имеется всего один диск – это отдельный случай
  {a[0][1]=n-2;
   for (i=1;i<=a[0][1];i++)
    {a[i][1]=n-i+1; a[i][2]=0; a[i][3]=0;}
   a[0][2]=1; a[0][3]=1;
   if (n%2) {a[1][3]=1; e=3; a[1][2]=2;cout<<"1-3 1-2 ";}
    else {a[1][2]=1; e=2; a[1][3]=2; cout<<"1-2 1-3 ";}
 /*Последующий цикл запускает функции в нужном порядке: сначала нужно передвинуть наименьший диск, затем найти не наименьший диск, который можно передвинуть и, затем, перенести его. Повторяется эта последовательность до тех пор пока количество дисков на стержне 3 не станет равным n   */
while (a[0][3]!=n)
{c=0; sdvig(e,seek1(),1);
 seek();
 if (c) sdvig(c,v,m);} }
  else cout<<"1-3";
 getch()
}
```

```
/*функция sdvig передвигает два указанных диска: z-стержень откуда взять диск, x-стержень куда положить, k- размер диска  */
void sdvig(int c,int v,int m)
{cout <<c<<"-"<<v<<" ";
 a[a[0][c]][c]=0;
 a[0][c]--; a[0][v]++;
 a[a[0][v]][v]=m;
 if (m==1) e=v;
}
```

*/*функция seek1 ищет стержень куда можно передвинуть наименьший диск(если нельзя передвинуть на диск большего размера(размер большего диска нечетный), то нужно передвинуть на стержень, на котором нету дисков.)*/*

```
int seek1()
{for (i=1;i<=3;i++)
 if ((!(a[a[0][i]][i]%2))&&(a[0][i])) return i;
 for (i=1;i<=3;i++)
 if (!a[1][i]) return i;
 }
```

*/*функция seek используется для нахождения стержня на который можно передвинуть не наименьший диск. В данном случае возможны два варианта: стержень является искомым если он пустой или размеры его верхнего диска и передвигаемого разной четности*/*

```
void seek()
{for (i=1;i<=3;i++)
 if ((i!=e)&&(a[a[0][i]][i]))
  for (j=1;j<=3;j++)
   if ((a[a[0][i]][i]<a[a[0][j]][j])||(!a[0][j]))
    if (!a[a[0][j]][j]) {c=i; v=j; m=a[a[0][i]][i]; i=4; j=4;}
     else if ((a[a[0][j]][j]-a[a[0][i]][i])%2)                    {c=i; v=j;
m=a[a[0][i]][i];i=4; j=4;}
}
```

Ответ выдается в виде команд ‘z-x’, где z – стержень с которого нужно взять диск, а x – стержень куда нужно положить.
Как видно, из листинга программы получается достаточно громоздкий и требует много машинного времени для ее решения. Расчет для 30 дисков потребовал более 1 минуты на компьютере с тактовой частотой процессора 3000 Mhz.

Рекурсивный алгоритм.

Теперь рассмотрим реализацию рекурсивного алгоритма.

Приведенный для сравнения алгоритм был опубликован в журнале «Программист», 2002. №8. С.82-90.

Как известно, рекурсия представляет собой обращение функции самой к себе.

Задача о перекладывании N дисков с i-го на j-ый стержень может быть декомпозирована следующим образом:

1. Переложить N-1 диск со стержня с номером I на стержень с номером 6-i-j
2. Переложить диск со стержня с номером I на стержень с номером j
3. Переложить N-1 диск со стержня с номером 6-i-j на стержень с номером j

```
#include<stdio.h>
#include<conio.h>
void hanoy(int I, int j, int k)
{
 if (k==1)
  printf("%d-%d  ",I,j);
 else
  {
   hanoy(I,6-i-j,k-1); hanoy(I,j,1); hanoy(6-i-j,j,k-1);
  }
}
void main()
{
 int input;
 scanf("%d",&input);
 printf("\nHanoy s %d diskami:\n",input);
 hanoy(1,3,input);
 getch();
}
```

Как следует из приведенного листинга, рекурсивный метод труднее понять, но при использовании рекурсии размер программы сильно уменьшается. Также выявляется большее быстродействие по сравнению с итеративным способом. Это происходит из-за того, что в рекурсии мы знаем какой диск и куда нужно перебросить, а в итеративном методе мы такой информацией не располагаем и нам нужно постоянно искать подходящие стержни.

Вывод: несмотря на то, что использование рекурсии более сложное, оно помогает достичь более высоких результатов, по сравнению с итеративными методами.

Литература

1. Вирт Н. «Алгоритмы и структуры данных»\Пер.с англ. СПб.: Невский диалект, 2001.
2. Браунси К. «Основные концепции структур данных и реализация в С++» Диалектика: Киев, Вильямс, М, 2002г.