SECTION 4

STATISTICS, DATA PROCESSING
(INFORMATICS) AND MATHEMATICS
BACKTRACKING APPLICATIONS DEVELOPED IN C #

Assoc.Prof. PhD. Valeriu LUPU
„Ştefan cel Mare” University, Suceava, Romania
valeriul@seap.usv.ro
PhD. Student Engineer Cătălin LUPU
Ministry of Administration and Interior, Suceava, Romania
lupucata@yahoo.com
Grigore VASILACHE
Financial-Banking College, Head of IT Department, Chisinau, Moldova
vasilache@moldnet.md

Abstract:
In this article presents applications of Backtracking method using object-oriented programming in C #. Object-oriented programming means programs with new types that integrates both data and methods associated with the creation, processing and destruction of such data. To gain advantages through abstraction programming (the program is no longer a succession of processing, but a set of objects to life (Figure 1), have different properties, are capable of specific actions and interact in the program). Spoke on instantiation new techniques, derivation and polymorphism object types.

Keywords: Backtracking, constructor, destructor, class, member, court

JEL Classification: C61

1. METHOD BACKTRACKING PRESENTATION

Backtracking method applies to issues in which the solution can be represented as a vector - x = (x1, x2, x3, Xk ... Xn) ∈ S, where S is the problem many solutions and S1 S = S1 x S2 x... x Sn And are finished with it mulimi items and xi ∈, and (Y) i = 1 .. n. For each problem give relations between the components of the vector x, which are called internal conditions, possible solutions that satisfy the conditions are called internal solutions result. The method of generating all possible solutions and then determining the solutions result by checking the conditions require internal very long time. Backtracking method avoids this generation and more efficient. Items vector x, get on line in order of increasing value indices, x [k] will receive a value only if the y were items x1 attributed values .. x [k-1]. The award value of x [k] verify fulfillment of conditions relating to continue ... x1 x [k-1]. If these conditions are not met, to stop k, this means that any values I attribute his x [k+1], x [k+1], .. x [n] will not reach a solution result. Backtracking method builds a vector solution progressively from the first component of the vector and going to last with the possible returns on atribuirilor earlier.

The method is applied as follows:
1) is the first choice and value sin S1 is given his x1;
2) is generated involve elements x1 ... x [k-1], with values of S1 .. S [k-1]; for the generation of x [k] to choose the first element of S [k] available for the choice is testing conditions to continue.

May occur following situations:
a) x [k] meets the conditions to continue. If it comes to the final solution (k = n) then displays the solution obtained. If not reached final solution to switch to the next generation item - x [k-1];
b) x [k] does not meet conditions to continue. Try the following value available from S [k]. If you do not find any value in S [k] to meet the continued return to the x [k -1] and resume algorithm for a new value thereof. Algorithm ends when they were taken into account all elements of S1.

The problems solved by this method requires time than running, so it is appropriate to use the method only if we have no other solution algorithm.

If crowds S1,S2,...Sn k have the same number of items the time required for implementation of the algorithm is kn. If the crowds at S1, S2 .. Sn not have the same number of
items, then it notes with "m" minimum cardinalelor crowds S1 ... Sn and "M", the maximum. Time is running in the [m to n .. M to n]. Backtracking method has exponential complexity in most cases ineffective. But it can not be replaced with other variants of solving fastest in the event that is required to determine all the solutions to a problem.

Generating permutation. Reads a natural number n. To generate any permutation set (1, 2, 3, ..., n).

Generating permutation will be made taking into account that any permutation will be composed of distinct elements of the crowd A. For this reason, to generate a permutation, we will ensure that the numbers will be distinct.

Algorithm properly present the case n = 3 (Figure 1).

1
\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c} 
1 & 1 & 2 & 1 & 2 & 1 & 2 & 3 & 1 & 2 & 3 & 2 & 1 \\
\end{array} \]
\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c} 
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
\end{array} \]
\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c} 
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
\end{array} \]

Figure no. 1. Presentation algorithm - generating Commutations

1. load stacked on level 1 value 1;
2. loading value of 1 on the 2nd is not possible, because this value is placed on Level 1 of the stack;
3. loading value of 2 on the 2-century is possible, because this value is no longer seen;
4. 1 of the value of the 3rd fall on level 1;
5. 2 of the value of the 3rd fall on the 2nd;
6. 3 a value of the 3rd not encountered on previous levels, whereas the 3 is completed correctly. Print: 1 2 3

Algorithm continues until the stack becomes empty.

The source is as follows:

```csharp
I
Varianta C# - OOP
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    public interface back
    {
        int n
        {
            get;
            set;
        }
    }
}
II
Continuare (I)
    { while (_n < 0 || _n > 100);
        }
    public void cit_p()
    {
        do
        {
            Console.WriteLine(_sir);
            _p = int.Parse(Console.ReadLine());
        } while (_p < 0 || _p > 100);
        }
    public virtual void init()
```
public virtual void succesor()
{
    if (_st[_k] < _n)
    {
        _st[_k] = _st[_k] + 1;
        _a_s = true;
    }
    else
        _a_s = false;
}

public virtual void valid()
{
    int i;
    _e_v = true;
    for (i = 1; i <= _k - 1; i++)
        if ((_st[_k] == _st[i]) || (Math.Abs(_st[_k] - _st[i]) == Math.Abs(_k - i)))
            _e_v = false;
}

public virtual bool solutie()
{
    if (_k == _n)
        return true;
    else
        return false;
}

public virtual void tipar()
{
    int i;
    for (i = 1; i <= _n; i++)
        Console.Write(_st[i] + ",");
    Console.WriteLine;
    Console.ReadLine();
}

public virtual void run()
{
    _k = 1;
    init();
    while (_k > 0)
public class dame : back
{
    public static int _n;
    public static int _p;
    public static int _k;
    public static bool _a_s;
    public static bool _e_v;
    public static int[,] _st;
    public static int[,] _matrice;
    public static string _sir;

    public void cit_n()
    {
        do
        {
            Console.WriteLine(_sir);
            _n = int.Parse(Console.ReadLine());
        } while (continuare II)
    }

    (Continuare din II)
    {
        do
        {
            succesor();
            if (_a_s)
                valid();
        } while (continuare II)
    }

    public int n
    {
        get
        {
            return _n;
        }
        set
        {
            _n = value;
        }
    }
    public int p
    {
        get
        {
            return _p;
        }
        set
        {
            _p = value;
        }
    }
    public int k
    {
        get
        {
            return _k;
        }
        set
        {
            _k = value;
        }
    }
    public int[,] matrice
    {
        get
        {
            return _matrice;
        }
        set
        {
            _matrice = value;
        }
    }
    public string sir
    {
        get
        {
            return _sir;
        }
        set
        {
            _sir = value;
        }
    }
}
public class comb : aranj

public class perm : dame

public class aranj : dame

Console.WriteLine("\n");
Console.ReadLine();

public class testare

public static void Main(string[] args)

perm PERM = new perm();
PERM.sir = "N = ";
[ ]

public override void succesor()
{
    if (_st[_k] < _n - _p + _k)
    {
        _st[_k] = _st[_k] + 1;
        _a_s = true;
    }
    else
    _a_s = false;
}

    }

public override void valid()
{
    int i;
    _e_v = true;
    for (i = 1; i <= _k - 1; i++)
    if (_st[_k] == _st[i])
        _e_v = false;
    if (_k > 1)
    if (_st[_k] < _st[_k - 1])
        _e_v = false;
}

}

2. OBJECT-ORIENTED PROGRAMMING IN C #

Object-oriented programming means programs with new types that integrates both data and methods associated with the creation, processing and destruction of such data. To gain advantages through abstraction programming (the program is no longer a succession of processing, but a set of objects to life (Figure 2), have different properties, are capable of specific actions and interact in the program). Spoke on instantiation new techniques, derivation and polymorphism object types.
Figure no. 2. Representation of objects using OOP

An abstract data type is an entity characterized by a data structure and a series of operations to these data. Operations that are accessible from outside the entity formed its interface. A type of data object is a type of data that implements a type of abstract data. We'll call the operations implemented in the type methods. Say that the data and methods are members of a given type of object. The use of such involves type: the definition of its existence, call methods and access to date.

**Classes**

In C#, classes are written exactly as in C++. Syntax:

```csharp
[attr] [Change] class [nume_clasa] [: clasa_de_baza] [corp_clasa]
• declarative attributes represents information on the entity define d.
• Modifiers is a sequence of key words: new public protected internal private (changing access) abstract sealed (changing legacy).
• Class base from which to inherit the current class, and there may be one such class.
• Corp_clasă is a block of members of the class declaration:
  • a constant (class values);
  • a fields (variable);
  • Types of data a user-defined;
  • methods (subroutine);
  • constructions;
  • One destructor;
  • A properties (features that can be consulted or set);
  • an Events (signaling instruments);
  • an indexed (indexing that allow courts in the respective class);
  • an operators.
```

Following example defines a hierarchy of classes, namely (Figure 3):

```csharp
class dame {
```

```csharp
```
Public void cit_n(string mes, ref int n){   }
Public void init(int k,ref int[] st) {   }
Public void successor(ref bool a_s, ref int[] st, int n,int k){   }
Public void valid (ref bool ev, ref int[] st, int k ){   }
Public bool solutie(intn, int k) {   }
Public void tipar(int n,int[] st) {   }

Class perm:dame
{
    Public void valid(ref bool ev, int[] st, int k) {   }
}

Class aranj:dame
{
    Public void valid (ref bool ev, ref int[] st,int k) {   }
    Public bool solutie(int n, int k) {   }
    Public void tipar(int n, int[] st) {   }
}

Class comb:aranj
{
    Public void successor(ref bool a_s, ref int[] st, int n,int k) {   }
    Public void valid (ref bool ev, ref int[] st,int k) {   }
}

Class partitii:perm
{
    Public void successor(ref bool a_s, ref int[] st, int n,int k){   }
    Public void valid (ref bool ev, ref int[] st, int k){   }
    Public void tipar(int n, int[] st) {   }
}

Class colorare:dame
{
    Public void successor(ref bool a_s, ref int[] st, int n,int k){   }
    Public void valid (ref bool ev, ref int[] st, int k){   }
    Public void tipar(int n, int[] st) {   }
}

Class comis:dame
{
    Public void init(int k,ref int[] st) {   }
    Public void successor(ref bool a_s, ref int[] st, int n,int k) {   }
    Public void valid (ref bool ev, ref int[] st, int k) {   }
    Public void tipar(int n, int[] st) {   }
}

Class suma:dame
{
    Public void successor(ref bool a_s, ref int[] st, int n,int k) {   }
    Public void valid (ref bool ev, ref int[] st, int k) {   }
    Public void tipar(int n, int[] st) {   }
}

Class part_nr:dame
Constructions

    public Complex(float a, float b)
    {
        re = a;
        im = b;
    }
    public Complex(float a) : this(a, 0) { }
    public Complex() { }

Members

Members of the classes and methods are given.

- fields, properties. Properties are some fields more "smart", meaning they are accessed like ordinary fields, but their implementation is that the methods. get, set and value are keywords in C#.

private float re, im;
public float Re
{
    get
    {
        return re;
    }
    set
    {
        // re = value;
    }
}
public float Im
{
    get
    {
        return im;
    }
    set
    {
        im = value;
    }
}

Metode:
public double Modul()
{
    return Math.Sqrt(re * re + im * im);
}

// supraincarcarea metodei ToString() mostenita de la Object
public override string ToString()
{
    string result = re + (im<0 ? "-" : "+") + Math.Abs(Im) + "i";
    return result;
}

CONCLUSIONS

This article presents the advantages of dbacktracking method and object-oriented programming. One of the major advantages of object oriented programming is the ability to reuse existing code. Can design new classes using a already constructed class is called inheritance. If a class A inherits class B, then B and class methods will be considered as belonging to class A. Heritage and create a class "base" in order to store the common characteristics of different classes, such properties will not be specified in each class separately. Main advantages of using object oriented programming are:

- ease of design and code reuse:
Once tested the correctness of the operation objects of an application, they can be used without any problem in another application. This advantage can be exploited by the formation of libraries of objects. Regarding design, it facilitates the decomposition of complex problems in simple subprobleme, which can be easily modeled using objects (variables will descri be properties of objects shaped their actions and methods).
- abstraction:
Designers can obtain an overall view to the behavior of objects and interactions between them, the details are buried in the composition of objects.

- safety data:

Able objects to behave like "black boxes", it can be used without the knowledge of their composition, providing privacy and decreases the frequency used appearances and the errors related to wrong handling of types of data.

REFERENCES

2. Tudor Sorin - Tehnici de programare - Editura L&S Infomat, București, 1994
3. Tudor Sorin - Tehnici de programare - Editura Teora, București, 1994
5. Ioan Odăgescu - Metode și tehnici de programare - Editura Intact, București, 1994
10. Livovschi L., Georgescu H., Sinteza şi analiza algoritmilor, Editura Știinifică și Enciclopedică, București, 1986
16. Lucian Sasu, Visual C#, curs C#, 2005