BACKTRAKING ALGORITHM OPTIMEZED IMPLEMENTED IN C++ AND JAVA

Mihaela Ana Runceanu, Ecaterina Teodoroiu High College, Targu Jiu, ROMANIA
Adrian Runceanu, Constantin Brancusi University, Targu Jiu, ROMANIA

ABSTRACT: In this article we present a solution of one problem implemented in C++ and Java languages. Our solution is integrated in application tool that we use for our students. Our application name is OPT (Online Programming Tool). This tool could be a very useful tool for teaching Computer Programming course.

KEY WORDS: Programming tool, E-learning, Learning Management System, Computer Science, Programming

1. ABOUT BACKTRACKING ALGORITHM

In the real world, a situation that arises frequently is that in which the solving of a problem leads to the determination of vectors of the form x = (x_1, x_2, ..., x_n), where
– each component x_i belongs to a certain finite set V_i;
– there are certain relation requirements that have to be met by the components of the vector, called restraints, so that x represents a solution to the problem if and only if these constraints are met by the components x_1, x_2, ..., x_n of the vector.
The problem to be solved now is to find all the vectors of form x = (x_1, x_2, ..., x_n) that meet these constraints.
The Cartesian product V_1 ×V_2 ×...×Vn is called the space of possible solutions.
The solutions of the problem are those possible solutions that meet the constraints.
One might say that the work speed of modern computers is so high that the exponential time shouldn't bother. In order to clear up this aspect, we acknowledge the fact that computers work at a very high speed, indeed. High values of work time can be reached even for values of n that are relatively small, the explanation being not that computers are not fast enough, but that the exponential function f(n) = an with a > 1 increases very quickly. For example, for a = 3 and n = 50, the work time needed covers centuries.
According to the above mentioned elements, it is highly recommended that for a given problem we should elaborate algorithms that are not exponential. The algorithms that are considered to be efficient are those for which the number of operations is polynomial (i.e. it is expressed in the form of a polynomial in n where n is the number of input data). It is not always possible to avoid the exponential algorithms; one example is the problem of generating all the sub-sets of a set having n elements when the number of results is 2n and therefore the number of operations is bound to be exponential.
The backtracking algorithm aims at avoiding the generation of all possible solutions, thereby cutting short the calculation time. The components of vector

x are assigned values in the ascending order of the indices. This means that xk is assigned a value only after (x_1, x_2, ..., x_{k-1}) have been assigned values that do not infringe the constraints. Moreover, the value of x_k has to be chosen such as (x_1, x_2, ..., x_{k-1}) also meet certain constraints.
Not meeting the constraints expresses the fact that any way we might choose x_{k+1}, ..., x_n, we will not obtain a solution (so the constraints ought to be met for a solution to be obtained). As a result, the next step will be to assign a value to x_k only when the constraints have been met. If they haven't, a new value is chosen for x_k or, in case the finite set of values V_k has been used out, a new choice is attempted for the preceding component x_{k-1} of the vector, decreasing k by one unit, etc. This backtracking gives the name of the method, as it expresses the fact that when we cannot advance, we backtrack the current sequence of the solution.
One has to notice that in certain cases, the fact that x_1, x_2, ..., x_{k-1} meets the constraints is not sufficient to guarantee that we will obtain a solution for which the first k − 1 solutions coincide with these values.
The choice of constraints is of utmost importance, a good choice leading to a substantial reduction in the number of calculations. In the ideal case, the constraints should be not only necessary, but also sufficient in obtaining a solution. But, usually, these are the constraints applied to the first k components of the vector. Obviously, the constraints in the case k = n are the very constraints imposed by the statement of the problem.
Through the backtracking method, any solution vector is built progressively, starting with the first component and up to the last one, eventually backtracking some values previously assigned. We remind that x_1, x_2, ..., x_n are assigned values in sets V_1, V_2,..., Vn. By assigning and failed assigning attempts because of not meeting the constraints, certain values are used out. For a certain component x_k we will mark C_k the set of values used out at the current moment. Obviously, $C_k \subset V_k$.
The process of obtaining the solution vectors by means of the backtracking method is easy to implement, due to the fact that any modification in configuration only affects few elements, namely the k index of the component, component x_k and the set C_k of used out values. [1]

2. RECURSIVE BACKTRACKING

Backtracking is a systematic method to traverse the space of possible solutions to a problem.
Programming is a general method, and can be adapted to any problem we want to get all possible solutions, or to select an optimal solution from the set of possible solutions.
Backtracking is the most expensive method but also in terms of execution time.
Overall we shape the solution of the problem as a vector v = (v_1, v_2, v_3, ..., v_n) where each item belongs V_K a finite and ordered S_k, with k = 1, n. In some cases, the sets S_1, S_2, S_3,..., S_n can be identical.
Procedure:
1. At each step k we start from a partial solution v = (v_1, v_2, v_3, ..., v_{k-1}) determined so far and try to extend this solution by adding a new element to the end of the vector.
2. Search in the set S_k, for a new element.
3. If there is an item not selected yet, check whether this element satisfies the conditions of problem, called further conditions.
4. If the circumstances continue to add element partial solution.
5. Check if we obtained a complete solution:
- If we obtained a complete solution and resume a display algorithm in step 1.
- If I did not get a solution, k <- k + 1 and resumes algorithm in step 1.
6. If the circumstances continue resumes algorithm to step 2.
7. If there is no element in the set S_k unverified means that we have no

possibility of this moment, to build the ultimate solution so we must change the choices made in advance so k <- k-1 and resume the problem in step 1.

Return in case of failure or to generate all solutions of the problem led to the designation of "backtracking" method, approximate translation would be 'going back'.

The general form of a function backtracking

Recursive algorithm implementation provided by backtracking method is more natural and therefore easier. Stack segment provided by calling the function is managed automatically by the system. Returning to the previous step is accomplished naturally by closing level of the stack.[2]

Examples of implementation of the method:

```
BK void (int k) // k-vector position is filled
{
   int i;
   for (i = 1; i <= nr_elemente_Sk; i ++)
 // elements through the set Sk
 {
     V [k] = i; // Selects an item from the set
     if (validation (k) == 1)
      // further validates the conditions of the problem
       {
        if (solution (k) == 1)
             // check if a solution was obtained
           display (k); // Display solution
        else
           BK (k + 1);
              // call function for position k + 1
     }
 } // if there is no unselected element in the set Sk,
 } // Close the stack and thus returns to the position vector k-1
```

Function execution ends after they closed all levels of the stack, means that the vector V can not be selected any items from sets S_k.

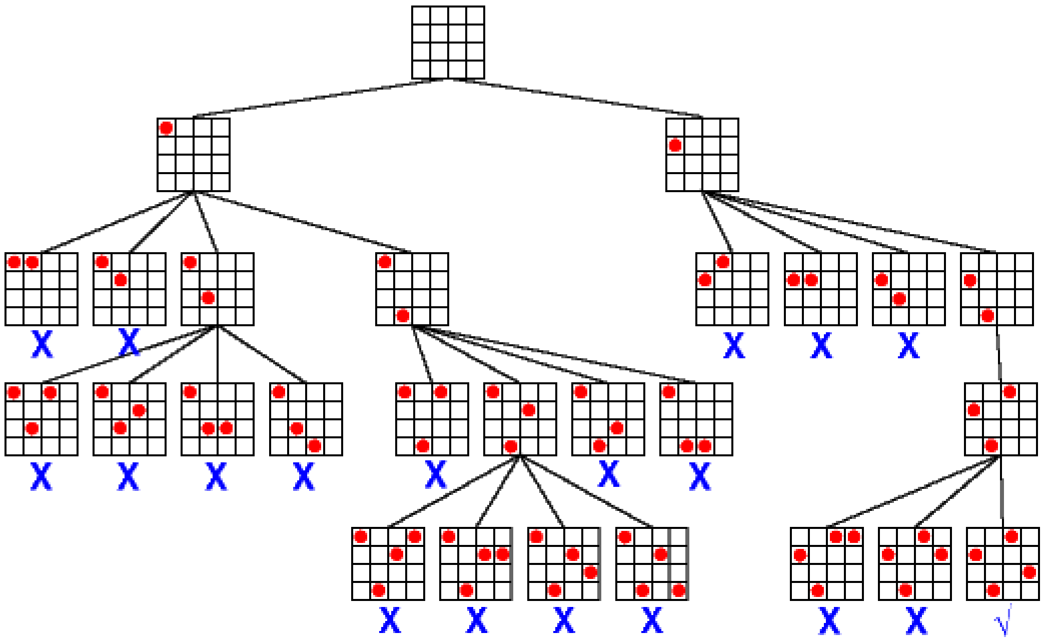

Figure 1. Example: (4-queens problem and Backtracking) (http://kti.mff.cuni.cz/~bartak/constraints/propagation.html)

3. OUR IMPLEMENTATION IN C++ AND JAVA

REQUIREMENT

Determine all subsets with m elements, divisors of a natural given number x.

INPUT
The input file submdiv.in contains on the first line the numbers x and m.
OUTPUT

The output file SUBMDIV.OUT contains on each line one subset determined. These subsets are displayed in lexicographical order. For each subset will display the items in ascending order, separated by a space.
Restrictions and specifications

- ✓ $1 \leq m \leq 6$
- ✓ $1 \leq x \leq 1000$
- ✓ if the problem has no solution, the output file will contain the message "no solution"

Example:
submdiv.in
45 4

submdiv.out
```
1 3 5 9
1 3 5 15
1 3 5 45
1 3 9 15
1 3 9 45
1 3 15 45
1 5 9 15
1 5 9 45
1 5 15 45
1 9 15 45
3 5 9 15
3 5 9 45
3 5 15 45
3 9 15 45
5 9 15 45
```

3.1 C++ code

```
#include <fstream>
#include <algorithm>
using namespace std;
ifstream f("submdiv.in");
ofstream g("submdiv.out");
int a[100],st[100],m,x,k,p,i,as,ev;
void init()
{
   st[k]=0;
}
int succesor()
{
   if(st[k]<p)
   {
      st[k]++;
      return 1;
   }
   else
      return 0;
}
int valid()
{
   if(k>=2&&st[k]<=st[k-1])
      return 0;
   return 1;
}
int solutie()
{
   return k==m;
}
void tipar()
{
   for(i=1;i<=m;i++)
      g<<a[st[i]]<<" ";
   g<<endl;
}
int main()
{
   f>>x>>m;
   for(int i=1;i*i<=x;i++)
      if(x%i==0)
         {
            a[++p]=i;
            if(x/i!=i)
               a[++p]=x/i;
         }
   if(p<m)
         g<<"no solution";
   else
   {sort(a+1,a+1+p);
   k=1;
   init();
   while(k)
   {
      do
      {
         as=succesor();
         if(as)
            ev=valid();
      }
      while(!(as&&ev||!as));
      if(as)
         if(solutie())
            tipar();
         else
         {
            k++;
            init();
         }
      else
         k--;
   }
  }
  return 0;
}

   }
```

3.2 Java code

```
import java.util.Scanner;
import java.util.Arrays;
import java.io.FileWriter;

import java.io.FileInputStream;
import java.io.InputStream;

int a[100],st[100],m,x,k,p,i,as,ev;
/** init function **/
public void init()
   {
      st[k]=0;
   }
/** succesor function **/
public int succesor()
   {
    if(st[k]<p)
    {
      st[k]++;
      return 1;
    }
    else
      return 0;
   }
/** valid function **/
public int valid()
   {
        if(k>=2&&st[k]<=st[k-1])
           return 0;
        return 1;
   }
/** solution function **/
public int solution()
   {
       return (k==m);

   }

/** display solution **/
public void display()
   {
File g = new File("SUBMDIV.OUT");
FileWriter write = new FileWriter(g);
write.write("\nSolution : ");
       for(i=1;i<=m;i++)
          write.write(" "+a[st[i]]);
       write.write("\n");
   }
/** Main function **/
public static void main (String[] args)
{
    Scanner scan = new Scanner(System.in);
    File f = new File("SUBMDIV.IN");
    Scanner s = new Scanner(f);
    int x = s.nextInt();
    int m = s.nextInt();
    for(int i=1;i*i<=x;i++)
      if(x%i==0)
         {
            a[++p]=i;
            if(x/i!=i)
               a[++p]=x/i;
         }
    if(p<m)
         write.write("no solution";
    else
     {//sort(a+1,a+1+p);
                  Arrays.sort(a+p);
                  k=1;
                   init();
         while(k)
         {
                do
                {
                        as=succesor();
                        if(as)
                          ev=valid();
                }while(!(as&&ev||!as));
                if(as)
                        if(solution())
                           display();
                        else
                        {
                                k++;
              init();
            }
            else
              k--;
   }
}
   return 0;
}
   write.close();
   }
}
```

4. CONCLUSION

In this article we present an implementation of backtracking algorithm that could be used for explain how we can obtained all solution for one given problem. Our implementation is part of OPT (Online Programmimg Tool) [3] - application adapted for an introductory teaching course called Computer Programming, for students of the technical faculties. In our futures work we implement another teaching course that could be use for teaching both in high college and faculties.

5. REFERENCES

[1] C.-F. GIURGIU. An implementation of the backtracking algorithm for multicore systems, ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY, Volume 13, Number 3, 2010, 241–254

[2] D.-A. Popescu, N. Bold, A.C. Bold An algorithm based on the backtracking method for cropping systems and the rotation of cultures, Procedia Technolgy, 9th International Conference Interdiscplinary in Engineering INTER-ENG 2015, Elsevier, pg. 431-439

[3] Runceanu Adrian, Cercel Constantin, Borcosi Ilie, Grofu Florin, An application integrated into an learning management system for teaching and evaluation engineering disciplines, The 16th edition of the SGEM International GeoConferences

[4] T. Cormen, C. Leiserson, R. Rivest, C. Stein. Introduction to Algorithms. The MIT Press, 3rd ed., 2009.

[5] http://www.runceanu.ro/adrian