ARRAY PROCESSING SPEED INCREASE

Pavel Gudanets
Supervisor: Dr. habil. sc. ing., Professor Eugene Kopytov

Transport and Telecommunication Institute
I Lomonosov Str, Riga, LV-1019, Latvia
E-mail: pavel-gudanets@navigator.lv

Introduction

There exist many developed methods of sorting and searching for the elements in the array [1]. This work offers a number of algorithms raising the speed of array processing:

1. Algorithm of search in any array with the use of multiplicate criterion
2. Algorithm of search in the ordered array by a method of approximation
3. A method of avoiding the logic comparisons allowing to increase the speed of some sortings and to find extremities in the array for the minimal time
4. Method of the complex data arrangement in the multidimensional array

All these ideas are united by the general principle constituting their essence: we achieve the faster and the more effective solution of the most different tasks with the help of some mathematical receptions. Actually the attention is paid to the formation of the special type of ptogramming, its potential has not been completely described in the given work. It is important to emphasize that the efficiency of the offered methods in many respects depends on the architecture of the used machine. This is quite natural because the technical realization of the computer should be adapted to the logics of the programs and the most effective decisions, rather than to the reverse ones.

The work is displayed on the one-dimensional array M[1.. N] consisting of N not repeating elements (keys). The required key is designated through x.

2. Algorithm of search in any array with the use of multiplicate criterion

The idea of algorithm. As we do not have any criteria in the not sorted array for defining a serial number of an element coincided with the key x, the sequential searching being the only way of searching. However it can be improved by checking the equality to x on each step and not to one but several elements. For this purpose it is enough to use the following rule: expression (M[1]-x)*(M[2]-x)*…*(M[d]-x) is equal to zero when only one of the elements M[1], M[2],…, M[d] is equal to x.

In the considered below example of the algorithm which had been written in the Pascal language we get the amount of elements assigned for one step, d being equal to 4; and to reduce the amount of comparisons in the operator of a cycle – we use the barrier where the additional element M[n+1] with the written required key is used. It is obvious, that length of the array should be not less than n+d, where n – the quantity of the used cells, otherwise the error of an exit for borders of the array is possible.

```
M[N+1]:=x;
i:=4;
While (M[i-3]-x) * (M[i-2]-x) * (M[i-1]-x) * (M[i]-x) <> 0 do i:=i+4;
if M[i-3]=x then i:=i-3
            else if M[i-2]=x then i:=i-2
                                 else if M[i-1]=x then i:=i-1;
if i<N+1 then Writeln('The serial number is: ',i)
         else Writeln('There is no such element in the array');
```

The given algorithm has the same temporary complexity as the sequential search. Thus the efficiency of the offered algorithm in many respects depends on the fact whether the operations of subtraction and multiplication in the used computer will be carried out with the faster comparison than logical one.

2. Algorithm of search in the ordered array by a method of approximation

The idea of algorithm. The algorithm is based on the array ordered by increase and containing the elements located in an interval [M[1]; M[N]], the required element x being probably also in it. Let's attribute to the interval [M[1]; M[N]] some kind of a percentage scale and divide it into identical sites. Comparing x to the numbers M[1] and M[N] we can establish approximately that "percent" which is necessary for x. The more uniformally elements are distributed the exacter results will be reached. The integrated description of a method below is being submitted:

1. If x is less than M[1] or more than M[N], the search is completed.
2. We put in a cell N+1 meaning x+1 as a barrier.
3. We calculate the approximate position x in a range [M[1]; M[N]] using the formula:

$$i:=\left|\frac{N\times(x-M[1])}{M[N]-M[1]+1}\right|+1$$

where the modular brackets mean the rounding off the result up to the whole part and changing the mark to plus, if the result is negative. If the first index of the array is any number C will be necessary to use the formula of a kind:

$$i:=\left|\frac{N\times(x-M[C])}{M[N+C-1]-M[C]+1}\right|+C$$

4. Increase i, while M[i] < x.
5. Reduce i, while M[i] > x.
6. If M[i] = x, the element is found, otherwise the element being not present in the array.

The realization of the considered algorithm in the Pascal language will be:

```
if (x<M[1])or(x>M[N]) then Writeln ('There is no such element in the array')
else begin
M[N+1]:=x+1;
i:=Abs(Round( (N/(M[N]-M[1]+1))*(x-M[1])+1 ));
while M[i]<x do i:=i+1;
while M[i]>x do i:=i-1;
if M[i]=x then Writeln ('Element''s serial number is ',i)
          else Writeln ('There is no such element in the array');
end;
```

If N and x – the large numbers, then the multiplication N*(x-M[1]) can give the result leaving the limits of a digit grid of the announced type. Therefore we recommend to make division firstly, as it is shown in the submitted algorithm. It is also possible to reduce a range (in which presumably there is x) by several iterations of binary search. It will reduce the meanings included in the formula.

The speed of algorithm "by approximation" depends only on the degree of the uniformity of the distribution of the elements in the array. So the search in the array having the enough uniform distribution of keys[1] is carried out 2-3 times faster than by binary search already for N=100. In comparison to the hash-search the considered algorithm does not require the additional memory for the hash-array.

3. A method of reducing the quantity of logic comparisons

At the solution of many tasks there exists the following necessity – to compare the meanings in i and j cells of the array M; if the order of the increase of the elements is broken (for example, at i>j M[i]<M[j]), it is required to make the rearrangement of elements. In this case the usual design of logical comparison is used, all the exchange sortings being based on it. There is a desire to invent the means by which the necessary rearrangements avoiding comparisons would be possible.

[1] For testing we use the arrays created by the random-number generator. It creates sequences just with the suitable uniformity of the elements distribution.

We will need one boolean variable R and array C consisting only of two cells. The description of this algorithm is given further:

1. Consider the difference M[i]-M[j]. If the difference is less than zero (elements break the order), the variable R accepts the meaning 1, if the difference is more than or is equal to zero, R := 0. If numbers are stored in a sign format (it depends on the used compiler or interpreter), it is possible to calculate meaning of R at once by reading the bit that stores the sign of the number. In the other case it is necessary to use the complicated mathematical formula – and the efficiency of algorithm is lost.

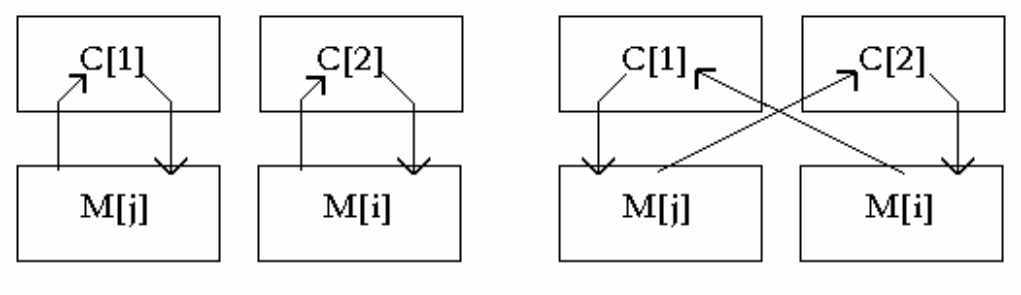
2. We write down the compared meanings of the array C as follows:

```
C[1+R]:=M[j];
C[2-R]:=M[i];
```

3. We read them out back in cells of the initial array M:

```
M[j]:=C[1];
M[i]:=C[2];
```

The model of the process is submitted in the fig. 1:



a) R = 0 b) R = 1

Figure 1

As we see, if R=0, then the elements are put back into the same cells from which they had been written down; otherwise they cross-wise enter the C's cells and in return the reading is interchanging the position[2].

The similar design of the logical comparison is also used for sorting by a choice and for some other sortings, and also by search for extremities-meanings in the array, as it will be shown in the following example:

```
max:=M[1];
for i : = 2 to N do
 if M[i]>max then max:=M[i];
Writeln ('The maximum is: ', max);
```

These algorithms can also be improved by the described method:

```
C[2]:=M[1];
for i := 2 to N do
 begin
   R:= sign(C[2]-M[i]);
```

[2] There is another way, it is written below:

```
R:=sign(M[i]-M[j]);
C[1]:=M[j];
C[2-R]:=M[i];
M[i]:=M[i-R];
M[j]:=C[1];
```

Obviously, this way suits only the case j=i-1, besides it is more difficult for perception. Therefore the first way is recommended for use.

```
  C[1+R]:=M[i];
 end;
Writeln ('The maximum is: ', C [2]);
```

As we see, if C[2]<M[i] then R=1, and consequently the new maximum C is calculated C[2]:=M[i], otherwise the improper meaning enters the other cell: C[1]:=M[i].

For the simultaneous search for the maximum and minimum it is recommended to use the array C from three cells. Minima are filtered in C[1] and maxima are filtered in C[3], C[2] being some kind of a bin in which all improper meanings are thrown off.

4. Method of the complex data structure in the multi-dimensional array

N steps are necessary for finding the element with index n in the not sorted array, in sorted - log n. It is obvious that the large speed is reached due to the occurrence of the internal logics in the distribution of elements. Also it is evident that the more complicated is this internal organization, the faster search should be, and if we design such complicated organized array correctly, sorting also will take place. Winning in speed, we do not lose in the other parameters: in the amount of the memory and the complexity of a code. Such effect is achieved because of the strict position of every element in the universum of array.

The order by increase creates the only dimension of the organization. If we wish to make it more complicated, we can add k dimensions.

To show the idea we will use the simple two-dimensional organization for our example. For this purpose we shall create the array of a kind:

```
M: array[0..t-1] of array of Integer;
```

The first dimension of the array divides it into t subarrays (it is desirable to announce them dynamic), and the belonging of an element x to one of them is determined by the rest from the division x mod t. Each of these groups is ordered by increase – forming the second dimension of internal organization of the array.

0 will be written down in the zero cell of every subarray. The zero cell will store the quantity of the used cells in the given subarray – just to avoid the necessity to count subarray length using the function every time we need it. The new element x will be written down in the cell [x mod t, max+1], where max is the meaning of the zero cell, then the max will be increased. To count the quantity of the elements in the whole array it will be enough to summarize the meanings of the zero cells.

The search of the element x is carried out by the obvious algorithm:

1. Determine x mod t.
2. Find x in the selected subarray by the method of the approximation or the binary search

Thus, the search occurs t times (!) faster than at the usual organization of an array, and after each addition of the new element there is no need to sort the whole array from the beginning up to the end, but only its t part.

Conclusions

The first two algorithms developed by the author are introduced to the educational process during learning the course "Structures of the data and algorithms of their processing" [2].

The further direction of the work consists of the investigation of such characteristic of the arrays as the degree of the uniformity of the distribution of elements. The value of this degree is useful for the solution of various tasks, for example, in the algorithm of search by the method of the approximation. It is necessary to develop the methods of estimating and updating the non-uniformities in the distribution of elements. This necessity is comparable to the necessity of balancing the trees.

References

1. Вирт Н. *Algorithms + data structure = programs*. Москва: Мир, 1985. 406 с.
2. Копытов Е., Иванова С., Птицына И.*Структуры данных и их обработка на компьютере: Учеб.пособие* / Под ред. Е. Копытова. Рига: Институт транспорта и связи, 2003. 128 с.