Bul. Ştiinţ. Univ. Baia Mare, Ser. B,
Matematică-Informatică, Vol. XVIII(2002), Nr. 2, 211 - 216

PROJECTION OF A VISUAL INTERFACE FOR MANIPULATING ORIENTED GRAPHS USING VISUAL BASIC 6.0

Marieta GÂTA

Abstract: The aim of this paper is to propose a modality to display in a graphic way by modeling and design, using technique of object oriented programing ofered by Visual Basic, searching of the oriented graphs in breadth-first and depth-first.

MSC2000: 68R10, 05C50, 05C85

Keywords: object oriented programming, visual programming, Visual Basic 6.0, user interface, graph theory, paths in graphs, breadth-first, depth-first.

Graphs are structures very spreading in the branch of computing, algorithms of graph being fundamentals in this area.

Let $X=\{x_1,x_2,\ldots x_n\}$ be a finite set. Let $\Gamma\subseteq X\times X$, where $X\times X$ is the cartesian product of the set X with itself. It calls oriented graph the ordered pair $G=(X,\Gamma)$. The elements $x_i\in X$ calls vertices or nodes. The elements of set Γ call arcs or edges. Arc $(x_i,x_j)\in\Gamma$ is denoted by $[x_i,x_j]$.
There are two standard modes of representation the oriented graph: as a set of a adjacency list or as an adjacency matrix. The representation of the oriented graph $G=(X,\Gamma)$ by adjacency matrix is using a boolean matrix with n rows and n columns, calls adjacency matrix. Presume that vertices are numbering 1, 2, 3, …n in a arbitrary way, the matrix $A=(a_{ij})$ with dimension n×n, in such a way as to:

$$a_{ij}=\begin{cases}1, & \text{for } [i,j]\in\Gamma\\ 0, & \text{for } [i,j]\notin\Gamma\end{cases}$$

There is a disadvantage in graph's memorizing through adjacency matrix that many elements of matrix are null, therefore it consumes unnecessary memory. Necessary of memory for the adjacency matrix of a graph is $O(n^2)$ and it is not depending by number of graph's edge, where n is number of graph's vertex. Although representation as list of adjacent is asymptotic at least identically efficiency as well as representation by adjacency matrix, the simplicity of adjacency matrix it possible to make it preferable, whenever graphs has a relative small number of vertices. More, if the graph is without cost, there is an additional advantage of store for the representation by adjacency matrix. In stead of using of a word of memory for each element from matrix, adjacency matrix use only a bit for each element.
Another method of representation of the oriented graphs is also one by list of adjacent. One graph with n vertices will have, a table adj with $|X|$ lists, for each vertex $i\in X$, a concatenate simple list adj[i], which will memorize all the vertices j for which $[i,j]\in\Gamma$ (the list of successors), or, in other way, pointers at all vertices i for which exists a edge $(i,j)\in\Gamma$. The list of the vertex i, adj[i], will be format from totality of the adjacent vertices of i in G. Usually, vertices from each list of adjacent are memorized in a arbitrary order. If G is an oriented graph, the sum of the lengths of all of the lists of adjacent is $|\Gamma|$, because an arc in the shape of (i,j) is represented by the apparition of j in adj[i]. If G is an unoriented graph, the sum of the length of all lists of adjacent is $2\times|\Gamma|$, because if (i,j) is an edge, then i appear in list of adjacent of j and reverse. Indifferent if a graph is or not oriented, the representation by lists of adjacent has the property that the dimension of the necessary memory is $O(\max(X,\Gamma))=O(X+\Gamma)$. The representation by lists of adjacent is very lusty, in the sense that it can be modified for to hear many other variants of graphs.

The storage of the graph with assistance of lists of adjacent present the advantage that, generally (on the whole), occupies less space in the detriment of the access at knowledge more difficult. That is exists another way more speedy for to determine if one given edge (i,j) belongs of graph, than seeking of j in list of adjacent of adj[i], disadvantage who it can be remedied using a graph's representation by the adjacency matrix.

In the practical application it is using the both methods of the representation of the graphs, depending on the algorithm with that it is working on the graph. For analysing the knowledge from vertices of oriented graph, these from behind are searched. There are many methods of searching. Among of these we remember two, which we elaborate it: searching breadth first and searching depth first.

The first (breadth first), it does beginning from a certain vertex i (source), which we consider it searched, searching then all its descendants, that is the vertices's set j for which $\exists$ $[i,j]\in\Gamma$, then we search all descendants of searched vertices at preceding step. Being given an graph $G=(X,\Gamma)$ and a vertex source i, searching in width explores systematic the edges of G for "to discover" each vertex which is accessible from i. The algorithm calculates the distance (the smallest number of edges) from i at all these accessible vertices. It creates an "width-tree" with the root i, which contains all these accessible vertices. For each vertex j accessible from i, the route from the tree of width from i at j corresponds of "the shortest way" from i at j in G, that is a way which contains a number minimal of edges. The algorithm functions both at oriented graphs and at unoriented graphs. The searching in width it is called such because it is widen uniform, the frontier among the vertices which is discover and the vertices which is undiscovered, at the frontier's width. This means that the algorithm discover all the vertices that are situated at the distance k vis a vis i before to discover some vertex at the distance k+1. For to keep the progress's evidence, the searching in width colors each vertex in green. Observations: each vertex will be searched only a single date. There are more solutions of a searching, because the order of the decendants's searching of a vertex it isn't necessary and it also depends by the mode in which it was memorized the graph. The searching in width it is done by the structure's utilisation named queue, which is implemented in this case by a vector.

The searching in width it is done beginning from a certain vertex i, after the searching of a vertex it is searches the first among of its descendants unsearched yet.

The strategy used at the searching in depth is, in accordance with which it is indicated also by name, to seek "profounder" in the graph whenever this matter is possible. In searching in depth, the edges are explorated beginning from the recent vertex i discovered which it has unexplored edges yet, which it starts from it. When all edges which starts from i, they were explored, the searching is "return" on its own track, for explore the edges which starts from the vertex from which i it was discovered. This operation continues until there will be discovered all the vertices accessible from the initial source vertex. The entire process is repeated until all the vertices are discovered.

Here too it can exist much solutions of searching. The searching in depth it is done by the structure's utilisation named stack, which is implemented implicit in recurrently alternative.

Identically as well as searching's case in width, each time when a vertex is discover, for to keep the evidence of progress, the searching in depth colors each vertex in yellow.

The application has created a class PUNCT.CLS which implements a point from plan. The function Is_inside(ix,iy) from it verifies if a point belongs of a circle (the distance from point at the circle's centre must be less or equal with the circle's radius).

On the form will be appear nine buttons: the button **Vertices** (which by double clicks on the form traces the graph's vertices), when I finish to drawing the graph's vertices it is traces edges, by the pressure first of a button **Arcs** and then by using the technique "drag and drop" it is tracing the lines which join the desired vertices, if it is likes the displaying of the adjacency matrix A, it will press on the button **Display A adjacency matrix**, the button **Searching breadth-first** which displays the graph's vertices searched in width, the button **Display the breadth-first search** which searches in width the graph by coloring it therewith according as searching in green, the button **Searching depth-first** which displays the graph's

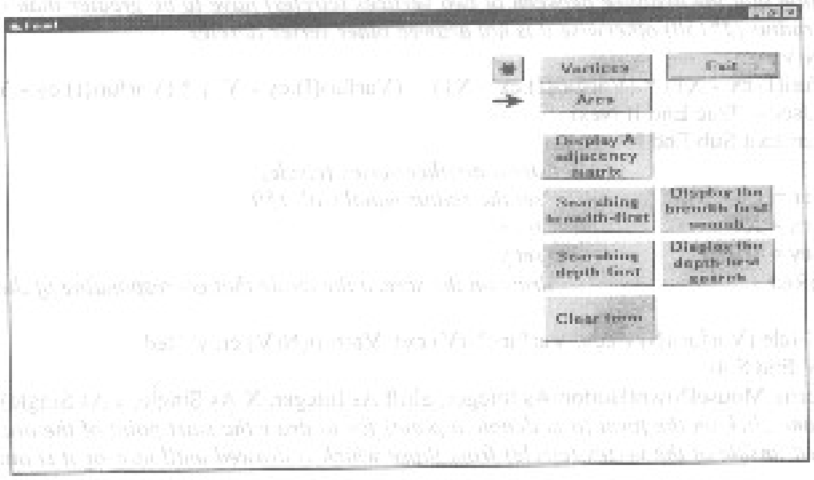
Vertices
Exit
Arcs
Display A adjacency matrix
Searching breadth-first
Display the breadth-first search
Searching depth-first
Display the depth-first search
Clear form

Figure 1

vertices searched in depth, the button **Display the depth-first search** which search in depth the graph, according as the searching coloring it in yellow, the button **Clear form** which deletes the graph and the calculation that it was done, and the button **Exit** which closes the application.
The searching depth-first is written in next procedure, and the searching breadth-first is similar.

```
Private Sub cmdAdancime_Click()          'search  in depth the graph coloring the vertices in yellow
Dim PauseTime, Start, Finish, TotalTime  'set the yellow than fill color of the graphic objects
FillColor = vbYellow
For I = 1 To contor  For J = 1 To NrV
If J = Coada1(I) Then
  PauseTime = 1                          'set duration
  Start = Timer                          'set start time
  Do While Timer < Start + PauseTime
    DoEvents                             'yield to other processes
  Loop
  Finish = Timer                                'set end time
  TotalTime = Finish - Start                    'calculate total time
  Circle (Varfuri(J).cx, Varfuri(J).cy), Varfuri(J).cr, vbYellow   'draw the circle in yellow
Beep End If Next Next
FillColor = vbRed                        'reverte to red for fill color of the graphics objects
End Sub
Private Sub Form_DblClick()              'set the point at the  vertices matrix
Dim bIsUsed As Boolean
'if DoubleClick on the form (for to draw a new vertex) set that centre of vertex (circle) a point which is situated inside of the vertex (circle) that exist already, it is not draw other vertex (circle)
bIsUsed = False
For I = 1 To NrV
  If Varfuri(I).Is_inside(X1, Y1) Then bIsUsed = True End If Next
```

```
'set the condition that the distance between of two vertices (circles) have to be greater than or equal for
two times the radius (2*150) otherwise it is not drawed other vertex (circle)
For I = 1 To NrV
   If Sqr((Varfuri(I).cx - X1) * (Varfuri(I).cx - X1) + (Varfuri(I).cy - Y1) * (Varfuri(I).cy - Y1)) <= 2 *
150 Then bIsUsed = True End If Next
If bIsUsed Then Exit Sub End If
NrV = NrV + 1                          'draw another vertex (circle)
Varfuri(NrV).cr = 150                  'set the radius equal with 150
Varfuri(NrV).cx = X1                   'set x
Varfuri(NrV).cy = Y1                   'set y
FillColor = vbRed                      'draw on the screen the circle that corresponding of the vertices of
the red color
FillStyle = 0 Circle (Varfuri(NrV).cx, Varfuri(NrV).cy), Varfuri(NrV).cr, vbRed
Print "   "; NrV End Sub
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, y As Single)
'it is verify if one click on the form (it is denote a point) for to draw the start point of the arc, respective
point is situated  inside of the vertex (circle) from those which is drawed until now or it is outside of any
vertex
'if it is so, set the start point of the arc which is drawed, the vertex's center (circle's center) respective
X1 = X Y1 = y
Dim bisIn As Boolean
bisIn = False
For I = 1 To NrV If Varfuri(I).Is_inside(X1, Y1) Then      'Print "it is inside"; I
   If Not bV Then StartV = I Else StartV = 0 End If
   bisIn = True
  End If
Next
If Not bisIn Then                               'Print "          it is outside";
End If
End Sub
Private Sub Form_MouseUp(Button As Integer, Shift As Integer, X As Single, y As Single)
'it is verify if the mouse is arise it on the form (it calls MouseUp) (it is denote the second point) for to
draw the final point of the arc, respective point is situated inside of the vertex (circle) from those which is
draw until now or it is outside of any vertex
'if it is so, set the final point of the arc which is drawed, the vertex's center (circle's center) respective it
is  set the suitable element from the adjacency matrix A equal to 1
If Button = 1 Then For I = 1 To NrV  If Varfuri(I).Is_inside(X, y) Then
   If Not bV Then EndV = I Else EndV = 0  End If End If Next
End If                                          'Print "The circles are "; StartV, EndV
If (StartV <> 0) And (EndV <> 0) Then
 Line (Varfuri(StartV).cx, Varfuri(StartV).cy)-(Varfuri(EndV).cx, Varfuri(EndV).cy)
 A(StartV, EndV) = 1  StartV = 0  EndV = 0 End If End Sub
```

The results for the graph from Figure 2 (with number of vertices 10) are:
-for searching breadth-first 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; for searching depth-first 1, 2, 5, 6, 3, 7, 8, 4, 9 ,10.

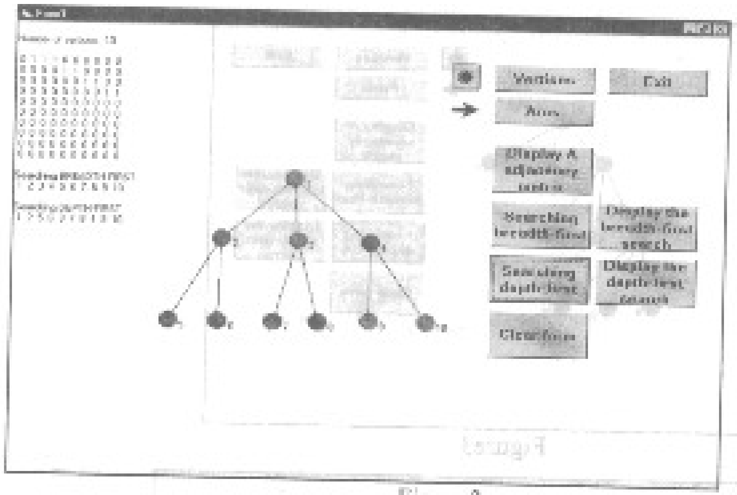
Vertices
Exit
Arcs
Display A adjacency matrix
Searching breadth-first
Display the breadth-first search
Searching depth-first
Display the depth-first search
Clear form

Figure 2

adjacency matrix

$$A=\begin{pmatrix}0&1&1&1&0&0&0&0&0&0\\0&0&0&0&1&1&0&0&0&0\\0&0&0&0&0&0&1&1&0&0\\0&0&0&0&0&0&0&0&1&1\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\end{pmatrix}$$

The results for the graph from Figure 3 (with number of vertices 10) are:
-for searching breadth-first 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; for searching depth-first 1, 2, 5, 8, 9, 10, 6, 7, 3 ,4.
-adjacency matrix

$$A=\begin{pmatrix}0&1&1&1&0&0&0&0&0&0\\0&0&0&0&1&1&1&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&1&1&1\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\end{pmatrix}$$

The results for the graph from Figure 4 (with number of vertices 7) are:
-for searching breadth-first 1, 2, 3, 6, 4, 5, 7; for searching depth-first 1, 2, 4, 5, 3, 6, 7.
-adjacency matrix

$$A=\begin{pmatrix}0&1&1&0&0&1&0\\0&0&0&1&1&0&0\\0&0&0&0&0&1&1\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{pmatrix}$$

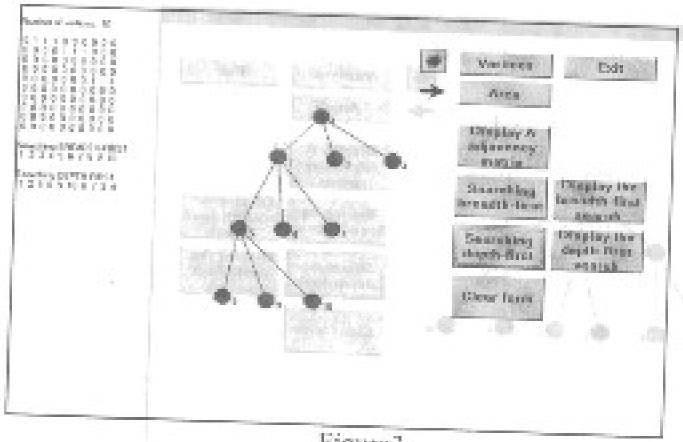
Vertices
Exit
Arcs
Display A adjacency matrix
Searching breadth-first
Display the breadth-first search
Searching depth-first
Display the depth-first search
Clear form

Figure3

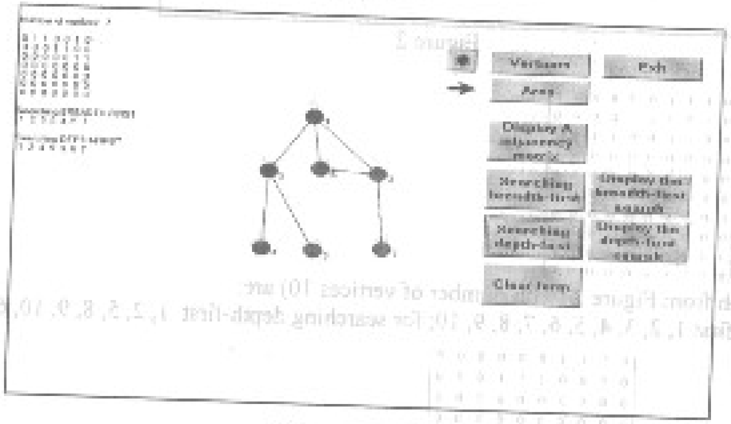
Vertices
Exit
Arcs
Display A adjacency matrix
Searching breadth-first
Display the breadth-first search
Searching depth-first
Display the depth-first search
Clear form

Figure 4

REFERENCES

[1] Cormen T. H., Leiserson C. E., Rivest R. R.: Introduction to Algorithms, Ed. Computer Libris Agora, Cluj Napoca, 2000
[2] Craig J. C., Webb J.: Microsoft Visual Basic 5.0 Developer's Workshop, Ed. Teora, București, 1998
[3] *** Microsoft Corporation: Microsoft Visual Basic 4.0 Programmer's Guide, 1995
[4] Sorin T.: Computer Science, Ed. L&S Infomat, București, 2001

Received: 17.09.2002

Department of Mathematics and Computer Science
Faculty of Sciences North University of Baia Mare
Victoriei 76, 4800 Baia Mare, ROMANIA
E-mail: marietag@ubm.ro