"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XVI – 2013 – Issue 1
Published by "Mircea cel Batran" Naval Academy Press, Constanta, Romania

ALGORITHM FOR FINDING STRONG CONNECTED COMPONENTS WITH MAXIMUM NUMBER OF EDGES FOR A DIRECTED GRAPH

Paul VASILIU[1]
Ion COLȚESCU[2]
[1] Lecturer Engineer, PhD Naval Academy "Mircea cel Bătrân" Constanța, România
[2] Assistant Professor, PhD, Naval Academy „Mircea cel Bătrân" Constanța, România

Abstract:*This paper introduces a new algorithm for finding the strong connected components of with maximum number of edges for a directed graph. Also, it is presented an implementation in C programming language of the algorithm and an example of usage of the program.*
Key-words: *strong, connected, maximum, edge, directed, graph*

1. INTRODUCTION

A directed graph $G=(X,E)$ is called strongly connected if for each $x,y\in X$ there exist a path from x to y and a path from y to x. The subgraph $G_1=(X_1,E_1)$ of the directed graph $G=(X,E)$ is a strongly connected component if:

1. For any $x,y\in X_1$ there is a path from x to y and a path from y to x.

2. There is no other subgraph $G_2=(X_2,E_2)$ of $G_1=(X_1,E_1)$ with the first property.

Every oriented graph has at least one strongly connected component.

We call successors of a vertix $x_i\in X$ all vertices $x_j\in X$ having the following property : there is a path between x_i and x_j. By definition, vertix x_i is a successor of the x_i vertix. Let it be $S(i)$ the set of successors of the x_i node.

We call predecessors of a vertix $x_i\in X$ all the vertices $x_j\in X$ with the property that exists a path between x_j and x_i. By definition, x_i vertix is a predecessor of the x_i vertix. Let it be $P(i)$ the set of predecessors of x_i node.

If vertix $x_j\in X$ is simultaneously both predecessor and successor of vertix $x_i\in X$ then there is a strongly connected component of the graph $G=(X,E)$ that contains it.

The set of vertices of a strongly connected component is maximal compared with the inclusion relation. Assuming the absurd that another vertix x_j would be simulatenously predecessor and successor of vertix $x_i\in X$ and it does not belong to the strongly connected component. Then, a path from x_j to x_i and a path from x_i to x_j should exist, case in which x_j would already belong to the strongly connected component.

In the following will be described an algorithm that finds all the strongly connected components of a directed graph $G=(X,E)$.

2. ALGORITHM

We will rename the vertices of the set $X=\{x_1,x_2,\ldots,x_n\}$ with $1,2,\ldots,n$. The array of the successors will be called $suc=(suc[1],suc[2],\ldots,suc[n])$. The array will have n components in which each component $suc[i]$ is equal to the connected component from which the successor of the i node, $i=1,2,\ldots,n$, is part of. Let it be defined the array of the predecessors $pred=(pred[1],pred[2],\ldots,pred[n])$. In this array, each $pred[i]$ element is equal to the connected component from which the predecessor of the i node, $i=1,2,\ldots,n$, is part of. We will define nrc as being the number of strongly connected components of the graph.

In the following will be shown a description of the algorithm in pseudo code in C language:

```
start
// Initialization of the suc and pred arrays
for (i=1;i<=n;i++)
        suc[i]=pred[i]=0;
end for
// There is at least one strongly connected component
nrc=1;
// For each vertix
for (i=1;i<=n;i++)
if (suc[i]==0) then
build S(i) şi P(i)
```

for (j=1;j<=n;j++)

if ($j \in S(i)$ && suc[j]==0) then

suc[j]=nrc;
end if

if ($j \in P(i)$ && pred [j]==0) then

pred[j]=nrc;
end if
end for
for (j=1;j<=n;j++)
if (suc[j] != pred[j]) then
suc[j]=pred[j]=0;
end if
end for
nrc++;
end for
for (i=1;i<nrc;i++)
for (j=1;j<=n;j++)
if (suc[j]==i) then
write j in strongly connected component i
end if
end for
end for
write nrc-1 strongly connected components
end

3. FINDING THE STRONGLY CONNECTED COMPONENTS WITH MAXIMUM NUMBER OF EDGES

Lets assume that $G=(X,E)$ and we already have the strongly connected components discovered. In order to find the strongly connected components with maximum number of edges it will be computed the number of edges for each component. This numbers will be stored in an array with nrc-1 elements. In the end, a maximal value of the elements from the array will be found and the maximal positions. The positions of the maximal value are the strongly connected components with maximum number of edges.

4. IMPLEMENTATION IN C PROGRAMMING LANGUAGE

In the following we present the implementation of the algorithm in C language.
The nodes are numbered from 1,2,...,n and displayed x1,x2,...,xn. The program has as input a text file with m+1 lines on the first line can be found number n, the number of nodes, and number m, the number of edges from graph G separated with a white space on each of the following m lines can be found the edges from the graph

```
include "stdio.h"
#include "conio.h"
#include "malloc.h"
#define dim 1000
// The prototypes of the functions defined in the program
int ** alocmat (int);
int * alocvect(int);
int read_n_m(int &, int &, char *);
int read(char *,int **);
void displaym(int **,int);
void displayv(int *,int,char *);
void display_egdes(int);
void successors(int **,int *,int,int,int);
void predecessors(int **,int *,int,int,int);
int valci(int **,int *,int,int);
int max(int *,int,int &);
void connected(int **,int *,int *,int,int *);

typedef struct arc
{
int x,y;
}edge;

typedef struct set
{
int nelem;
int S[dim];
}set;

edge mu[dim];   // mu array of edges
set start;

// Matrix allocation
int ** alocmat (int n)
```

```
{
int i;
int ** p=(int **) malloc ((n+1)*sizeof (int *));
if ( p != NULL)
for (i=0; i<=n ;i++)
p[i] =(int *) calloc ((n+1),sizeof (int));
return p;
}

// Function for allocating an array with n+1 integer elements
int * alocvect(int n)
{
int *p=(int *)calloc(n+1,sizeof(int));
return p;
}

// Function for reading the input file
// The n number of nodes and the m number of edges are read
int read_n_m(int &n, int &m, char *name)
{
FILE *f;
if((f=fopen(name,"r"))!= NULL){
fscanf(f,"%d %d",&n,&m);
fclose(f);
return 1;
}
else
return 0;
}

// Function for reading the input file and
// building the adjacency matrix
int read(char *name,int **a)
{
int i,j,x,y,n,m;
FILE *f;
if((f=fopen(name,"r")) != NULL){
fscanf(f,"%d %d",&n,&m);
for(i=1;i<=m;i++)
{
fscanf(f,"%d %d ",&mu[i].x,&mu[i].y);
a[mu[i].x][mu[i].y]=1;
}
fclose(f);
return 1;
}
else
return 0;
}

// Function for displaying the matrix n x n
void displaym(int **a,int n)
{
int i,j;
printf("\n\n The adjacency matrix \n\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf(" %2d ",a[i][j]);
printf("\n");
}
}

// Function for displaying the s array
void displayv(int *v,int n,char *s)
{
int i;
printf("\n Array %s = ( ",s);
for(i=1;i<n;i++)
printf(" %2d , ",v[i]);
printf(" %2d ) \n",v[i]);
}

// Function for displaying the edges from the graph
void display_edges(int m)
```

```
{
int i;
printf("\n Graph G has %d edges \n\n",m);
for(i=1;i<=m;i++)
printf(" Edge %d:\t ( x%d , x%d )  \n",i,mu[i].x,mu[i].y);
}

// Function for traversing the graph and building the successors array
void successors(int **a,int *suc,int nod,int n,int nrc)
{
int i,k;
suc[nod]=nrc;
for(k=1;k<=n;k++)
if(a[nod][k]==1 && suc[k]==0)
successors(a,suc,k,n,nrc);
}

// Function for traversing the graph and building the predecessors array
void predecessors(int **a,int *pred,int nod,int n,int nrc)
{
int k;
pred[nod]=nrc;
for(k=1;k<=n;k++)
if(a[k][nod]==1 && pred[k]==0)
predecessors(a,pred,k,n,nrc);
}

// Function for determing the number of  edges for the connected component i
int valci(int **a, int *compi, int nnod, int m)
{
 int i,j,valc=0,gasit,k;
 for(i=1;i<=m;i++)
 {
 gasit=0;
 for(j=1;j<=nnod+1;j++)
 if(mu[i].x==compi[j])
 {
 for(k=1;k<=nnod;k++)
 if(mu[i].y==compi[k])
 gasit=1;
 }
 if(gasit)
 valc++;
 }
 return valc;
}
// Function for determining the maximal value and the position of the maximal value from the x vector
int max(int *x,int nrc,int &poz)
{
int i,max;
i=0;
max=x[i];
poz=i;
for(i=1;i<nrc;i++)
if(max<x[i]){
max=x[i];
poz=i;
}
return max;
}

// The function determines the strongly connected component with maximum number of edges
void connected(int **a,int *suc,int *pred,int n,int m)
{
int i,j,nrc,k,valc,*compi,*values,poz,valmax;
compi=alocvect(n);
nrc=1;
for(i=1;i<=n;i++)
if(suc[i]==0)
{
suc[i]=nrc;
successors(a,suc,i,n,nrc);
predecessors(a,pred,i,n,nrc);
displayv(suc,n,"successors  ");
displayv(pred,n,"predecessors");
```

```
getch();
for(j=1;j<=n;j++)
if(suc[j]!=pred[j])
suc[j]=pred[j]=0;
displayv(suc,n,"successors  ");
displayv(pred,n,"predecessors");
getch();
nrc++;
}
if(nrc==2)
{
printf("\n The graph has only one strongly connected component\n");
printf(" The graph is strongly connected \n");
printf(" The strongly connected component %d \n",nrc-1);
for(j=1;j<=n;j++)
if(suc[j]==nrc-1)
printf(" x%d ",j);
printf("\n");
printf(" Maximum number of edges of the strongly connected component is %d \n",m);
}
else
{
printf("\n The graph has %d strongly connected component \n",nrc-1);
values=alocvect(nrc);
for(i=1;i<nrc;i++)
{
k=0;
printf(" The strongly connected component %d \n",i);
for(j=1;j<=n;j++)
if(suc[j]==i){
printf(" x%d ",j);
compi[k+1]=j;
k++;
}
if(k==1){
printf("\n The strongly connected component %d has no edges \n”,i);
values[i]=0;
}
else
{
valc=valci(a,compi,k,m);
printf("\n The strongly connected component %d has %d edges \n",i,valc);
values[i]=valc;
}
printf("\n");
getch();
}
valmax=max(values,nrc,poz);
if(poz==0)
poz++;
printf(" The strongly connected component %d has maximum number of  edges %d \n",poz,valmax);
}
}

// The main function
int main()
{
int **a,*suc,*pred,n,m;
char name[30];
printf("\n\n File name ");
gets(name);
if(read_n_m(n,m,name)){
printf(" Number of vertices %d \n",n);
printf(" Number of edges %d \n",m);
getch();
a=alocmat(n);
suc=alocvect(n);
pred=alocvect(n);
if(read(name,a)){
displaym(a,n);
getch();
display_edges(m);
getch();
connected(a,suc,pred,n,m);
}
```

```
free(a);
}
else
printf(" Error reading file %s \n",name);
getch();
}
```

5. AN EXAMPLE

Let be the graph $G=(X,E)$ with $X=\{x_1,\cdots,x_8\}$, from table 1:

Table 1 The edges (E) of the graph

E	E	E
(x_1,x_2)	(x_2,x_5)	(x_5,x_8)
(x_1,x_6)	(x_2,x_6)	(x_5,x_6)
(x_1,x_5)	(x_3,x_1)	(x_6,x_4)
(x_1,x_8)	(x_3,x_7)	(x_6,x_7)
(x_2,x_1)	(x_4,x_7)	(x_7,x_8)
(x_2,x_4)	(x_4,x_5)	(x_8,x_6)

For this graph the results are:
File name graph.txt

```
Number of vertices 8
Number of edges 18
The adjacency matrix
0  1  0  0  1  1  0  1
1  0  0  1  1  1  0  0
1  0  0  0  0  0  1  0
0  0  0  0  1  0  1  0
0  0  0  0  0  1  0  1
0  0  0  1  0  0  1  0
0  0  0  0  0  0  0  1
0  0  0  0  0  1  0  0

Graph G has 18 edges
Edge 1:      ( x1 , x2 )
Edge 2:      ( x1 , x6 )
Edge 3:      ( x1 , x5 )
Edge 4:      ( x1 , x8 )
Edge 5:      ( x2 , x1 )
Edge 6:      ( x2 , x4 )
Edge 7:      ( x2 , x5 )
Edge 8:      ( x2 , x6 )
Edge 9:      ( x3 , x1 )
Edge 10:     ( x3 , x7 )
Edge 11:     ( x4 , x7 )
Edge 12:     ( x4 , x5 )
Edge 13:     ( x5 , x8 )
Edge 14:     ( x5 , x6 )
Edge 15:     ( x6 , x4 )
Edge 16:     ( x6 , x7 )
Edge 17:     ( x7 , x8 )
Edge 18:     ( x8 , x6 )
Array successors   = ( 1 , 1 , 0 , 1 , 1 , 1 , 1 , 1 )
Array predecessors = ( 1 , 1 , 1 , 0 , 0 , 0 , 0 , 0 )
Array successors   = ( 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 )
Array predecessors = ( 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 )
Array successors   = ( 1 , 1 , 2 , 2 , 2 , 2 , 2 , 2 )
Array predecessors = ( 1 , 1 , 2 , 0 , 0 , 0 , 0 , 0 )
Array successors   = ( 1 , 1 , 2 , 0 , 0 , 0 , 0 , 0 )
Array predecessors = ( 1 , 1 , 2 , 0 , 0 , 0 , 0 , 0 )
Array successors   = ( 1 , 1 , 2 , 3 , 3 , 3 , 3 , 3 )
Array predecessors = ( 1 , 1 , 2 , 3 , 3 , 3 , 3 , 3 )
Array successors   = ( 1 , 1 , 2 , 3 , 3 , 3 , 3 , 3 )
```

```
Array predecessors =  ( 1 , 1 , 2 , 3 , 3 , 3 , 3 , 3 )
The graph has 3 strongly connected components
The strongly connected component 1
x1  x2
The strongly connected component 1 has 2 edges
The strongly connected component 2
 x3
The strongly connected component 2 has no edges
The strongly connected component 3
x4  x5  x6  x7  x8
The strongly connected component 3 has 8 edges
The strongly connected component 3 has maximum number of  edges 8
```

BIBLIOGRAPHY

1. A. Băutu, P. Vasiliu, *The bases of the computers programming*, Publisher ANMB, Constanța 2009
2. P. Vasiliu, A. Băutu, *Computers programming in C language*, Publisher Europolis, Constanța 2006
3. P. Vasiliu, *Modeling and simulation*, Publisher ANMB, Constanța 2013.