Ann. For. Res. 65(1): 10016-10021, 2022
ISSN: 18448135, 20652445

ANNALS OF FOREST RESEARCH
https://www.e-afr.org/

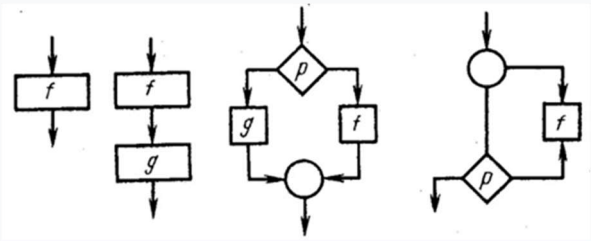
IMPLEMENTING MINIMUM GRAPH COVERING IN PYTHON

M.M.Aripov[1], Sh.Sh.Axmadaliyev[2], X.M.Xasanov[3], M.M.Botirov[4]
[1]Associate Professor, [2]Senior teacher, [3,4]Assistant teacher of the Department of Informatics,
Kokand State Pedagogical Institute

Abstract – A structural programming method and a graph model of the program are considered. The concept of a packed adjacency matrix is introduced. An algorithm and a Python program for constructing the minimum number of paths that cover all branches of the program graph are given.

Key words - testing, minimum program graph coverage, packed adjacency matrix, DD-paths, vertex, branches, g-graph, h-graph, algorithm complexity, Python, IF, WHILE.

Among the many methods proposed to increase the reliability of programs, the method of structured programming has gained great importance. A program is said to be structured with respect to some basic set of program control structures if it is composed of these control structures suitably interconnected. A generally accepted set of basic program management structures is shown in Figure 1.



a) b) c)

Fig.1. A set of basic program structures:
a – linear program; b - branched type IF; c - is a WHILE loop

These basic structures are called D-elements. Accordingly, a program composed of these basic structural elements is called a D-structured program. It is proved that D-elements are a complete (sufficient) base for compiling any programs or describing any algorithms.

Some program can be described as a graph G=(V,E) consisting of a set of vertices V and a set of arcs E. Let each vertex vi included in V (v_i Є V) represent some linear section (a sequence of commands without branches) programs. Let each arc (v_i v_j) of the graph represent a transfer of control from a linear segment i to a linear segment j. We call a graph G a structural graph or a D graph if it describes a structured program. Although structured programming can increase the

reliability of a program, testing is still necessary because it is necessary to prove the correctness of the program. Tests are generally required to ensure that each instruction or branch in the program is executed at least once.

On the model of the program in the form of a graph, this corresponds to finding a set of paths that cover all the vertices or all the arcs of the graph. Often, a set of tests leading to the execution of all statements or all branches in the program is not enough. On the other hand, testing all paths in the program structure is impractical or even impossible due to their large number. A compromise solution is path selection testing, which provides a test of critical interactions between parts of the program. On the graph describing the program, such interactions can be modeled by introducing the required pairs, i.e. pairs of vertices that must interact on at least one test.

The graph is represented as a packed adjacency matrix (PAM). The packed adjacency matrix A = $\{a_{ij}\}$ of a graph with v vertices is a (v x l) matrix (l is the maximum exit degree of the i-vertex). The degree of entry $d_{inp}(vi)$ and exit $d_{out}(vi)$ of some vertex of the graph means, respectively, the number of incoming and outgoing arcs from the vertices. Each row i of the PAM is filled in random order with the numbers of vertices that are adjacent to vertex i.

The representation of graphs in the form of PAM has the following advantages over other existing representations: for large graphs, the number of columns of PAM is much less than the number of columns of the corresponding adjacency matrix; it is relatively easy to model the process of moving along the graph to build paths; reduces graph processing time. The test criterion is the criterion of branches, where a program branch is understood as a certain sequence of statements that are executed strictly one after another. Thus, a branch is a linear section of a program. To construct the minimum coverage, the graph is divided into DD-paths using the PAM of the original graph. The set of vertices with output degree $d_{out}(vi)>1$, input and output vertices are denoted as D-vertices. Then a DD-path is a simple path between two D-vertices, such that there are no D-vertices with in its boundaries. Then the cycles and loops are determined and the arcs closing them are excluded.

The proposed algorithm for constructing a minimum cover (MCOV) of a graph consists of the following steps. An example of a program graph is shown in fig. 2.

Stage 1. The i-th vertex is looked through and the adjacent vertex j is determined, the number of which is the maximum among the numbers of adjacent vertices, where i $\in$ { l , n -1;} n is the number of graph vertices.

Stage 2. The arc (v_i, v_j) is viewed. If d_{out} (v_i) > 1 and d_{inp} (v_j) >1 , then the arc g(v_i,v_j) is excluded. If d_{out} (v_i) > 1 and d_{inp} (v_j) = 1, then the arc h(v_i, v_j) noted.

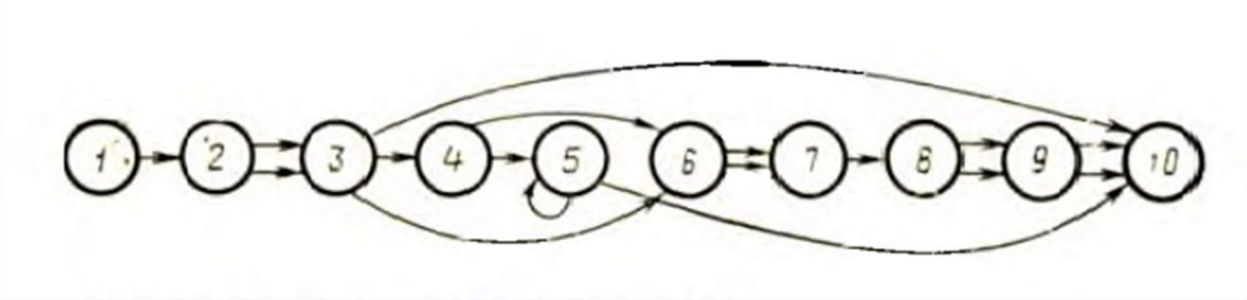


Fig. 2. An example of a program graph

Step 3. Substitute i = j and repeat steps 1-2 until j is equal to the number of the final (output) vertex. The path is fixed as a sequence of values j.

Stage 4. If there are no arcs of type g in the constructed path, then the last arc of type h is excluded.

Stage 5. Stages 1–2 are repeated until the constructed path contains no arcs of type g and h.

An example of constructing a minimal coverage of a program graph. Let the program graph shown in Fig. 2. Graph arcs mean a sequence of computational program operators, graph vertices — branching and union operators. After eliminating the closing cycles of arcs (they are tested separately), the graph in Fig. 2 is described by the following PAM:

1	2	0	0
2	3	3	0
3	4	6	10
4	5	6	0
5	10	0	0
6	7	7	0
7	8	0	0
8	9	9	0
9	10	10	0
10	0	0	0

The first stages of the MCOV algorithm give the following results:

Stage 1. Set i = 1, j = 2. {1, 2}

Stage 2. The arc (vi, vj) is not excluded and is not marked.

Stage 1. Set i = 2, j = 3. {1, 2, 3}

Stage 2. One of the arcs (v_2, v_3) is excluded

Stage 1. Set i = 3, j = 10. p1 = {1, 2, 3, 10}

Stage 2. The arc (v_3, v_{I0}) is eliminated.

Stage 1. Set i = 3, j = 6.

Stage 2. The arcs (v_6, v_7), (v_8, v_9), (v_9, v_{I0}) are excluded, the arc h(v_3, v_6) noted.

The procedures of stages 1–2 are repeated until the path to the final vertex of the graph v10 corresponding to the receipt of the calculation result is determined. In this case, the first path p1 = { 1 , 2, 3, 10} is determined after three steps. The following steps, repeated until there are no arcs of type g and h in the constructed path, allow us to determine the following paths:

p_2 = {1, 2, 3, 6, 7, 8, 9, 10},
p_3 = {1. 2, 3, 4, 6, 7, 8, 9, 10},
p_4 = {1. 2, 3, 4, 6. 7, 8, 9, 10},
p_5 = {1, 2, 3, 4, 5, 10}.

According to the developed algorithm for the minimum coverage of the program graph, a Python program was compiled, which is shown in Table 1.

The program consists of the following parts:

- input of initial adjacency matrix;
- determination of the degree of exit of vertices;
- determination of the degree of entry of vertices;
- creation of a certain path;
- exception of dupes of type g or h in the route of a certain path.

To create one path in the worst case, n operations are required, and to build the minimum number of operations, m operations are required, where m is the minimum number of paths that cover all branches of the program graph. Therefore, the complexity of the developed algorithm is O(|v| x |m|) => O(|v|).

Table 1.

Minimum program graph coverage program	Continuation of the program
# Input initial adjacency matriximport	# formation of a set of paths

```
numpy as np
a=np.array([[0,1,0,0,0,0,0,0,0,0],
            [0,0,1,0,0,0,0,0,0,0],
            [0,0,0,1,0,0,0,0,0,0],
            [0,0,0,0,1,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0,1],
            [0,0,0,0,0,0,1,0,0,0],
            [0,0,0,0,0,0,0,1,0,0],
            [0,0,0,0,0,0,0,0,1,0],
            [0,0,0,0,0,0,0,0,0,1],
            [0,0,0,0,0,0,0,0,0,0]])
print(a)
vx=[0,0,0,0,0,0,0,0,0,0]
vix=[0,0,0,0,0,0,0,0,0,0]

# Determining the degree of exit of vertices
for i in range(0,10):
   for j in range(0,10):
      vix[i]=vix[i]+a[i,j]
print(vix)

# Determining the degree of entry
for i in range(0,10):
   for j in range(0,10):
      vx[i]=vx[i]+a[j,i]
print(vx)

# creating paths
p=[0,0,0,0,0,0,0,0,0,0,0,0,0]
ii=[0,0,0,0,0,0,0,0,0,0,0]
jj=[0,0,0,0,0,0,0,0,0,0,0]
ia=[0,0,0,0,0,0,0,0,0,0,0]
ja=[0,0,0,0,0,0,0,0,0,0,0]
w=[0,0,0,0,0,0,0,0,0,0,0]
z=[0,0,0,0,0,0,0,0,0,0,0]
a[i,j]=0
k=0
for i in range(0,10):
    for j in range(9,0,-1):
        if a[i,j]>0:
d=0
q=0
k=0
p[0]=1
for i in range(10,0,-1):
   for j in range(10,0,-1):
      ia[j]=ii[j] # transformation
      ja[j]=jj[j] # transformation
for i in range(0,10):
    d=ja[i]
   #print('d ',d)
   p[k]=9
   if d==9:
     break
   for j in range(0,10):
      if ia[j]==d:
        p[k]=d
        print('put ',p[k])
        print('put ',p)
        k=k+1
        if ia[j]==d or d==9:
          break
      if ia[j]==d or d==9:
        break
p[0]=1

# exclusion of type arcs "g" и "h"
g=0
h=0
for i in range(0,10):
   s=p[i]
   t=p[i+1]
   print('s  t ',s,t)
   if(vix[s]>1 and vx[t]>1):
     a[s,t]=a[s,t]-1
     print('искл. ',a[s,t])
     p[i]=j
     g=1
   elif(vix[s]>1 and vx[t]==1)and g==0:
      a[s,t]=a[s,t]-1
```

```
            ii[k]=i
            jj[k]=j
            k=k+1
```

```
            print('xxx ',s,t)
            if t==9:
              break
          if t==9:
            break
#print('put ',p)
print(a)
```

Literature

1. Iyudu K.A., Aripov M.M. Automating the generation of paths for testing programs written in Fortran. Programming, 1986, No. 7.
2. Iyudu K.A., Aripov M.M. Testing a program based on the minimum coverage of its graph. Control systems and machines, 1985, No. 6.
3. Iyudu K.A., Aripov M.M. Automation of structural testing of programs. Republican conference. Reliability and quality of software. Abstracts of reports. Lvov, January 29-31, 1985.
4. M.M.Aripov, R.N.Normatov, I.M.Siddikov, U.Oripova Fundamentals creating the algebra science and algorithms. Solid state technology. Vol. 63, No. 5, 2020, p.6103-6111.
5. Simon C., Ntafos S., Louis Hakimi. On structured digrafs and program testing. IEEE Trans. On Computers, vol. C-30, № 1, January 1981.
6. Aripov M.M. Structural methods for program testing. Journal of Positive School Psychology. Vol.6, No 10, 2022, p.3428-3431.