Scientific Research Publishing

Open Journal of Applied Sciences, 2025, 15(8), 2337-2343
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

The Nullity Solution Algorithms for Trees

Chengzhi Guo

School of Mathematics and Statistics, Qinghai Minzu University, Xining, China
Email: gcz88066@163.com

How to cite this paper: Guo, C.Z. (2025) The Nullity Solution Algorithms for Trees. *Open Journal of Applied Sciences*, **15**, 2337-2343.
https://doi.org/10.4236/ojapps.2025.158156

Received: July 18, 2025
Accepted: August 16, 2025
Published: August 19, 2025

Abstract

This paper first discusses the storage structure of trees, selects a convenient storage method for solving the nullity of trees, and then applies the relationship between the maximum matching number of trees and the nullity to give an algorithm for calculating the maximum matching number and nullity of any tree.

Keywords

Nullity of Tree, Storage Structure, Maximum Matching Number, Algorithm

1. Introduction

Consider G is a graph, and $V(G)=\{v_1,v_2,\cdots,v_n\}$ is its vertex set. The adjacency matrix of the graph G is denoted by A, which is an n-order matrix. When v_m is adjacent to v_n, $a_{mn}=1$, otherwise, $a_{mn}=0$.

Definition 1.1. Let $P(\lambda)$ be the characteristic polynomial of the adjacency matrix A, and the set of all roots (including duplicates) of $P(\lambda)$ is called the spectrum of the graph G. Among them, the multiplicity of the zero eigenvalue is called nullity of G, denoted by $\eta(G)$.

The nullity of the graph G refers to the multiplicity of the zero characteristic root of its adjacency matrix. Obviously, $\eta(G)=n-r(A)$, in this case, n is the order of the graph G and $r(A)$ is the rank of the adjacency matrix A. When $\eta(G)>0$, *i.e.*, A is a singular matrix, the graph G is called singular; otherwise, it is called nonsingular. Therefore, this problem has a good chemical background [1]-[3]. If a bipartite graph (corresponding to an alternating hydrocarbon) is singular, it means that the molecule represented by the graph is unstable; and this problem is also meaningful for nonsingular bipartite graphs (corresponding to non-alternating hydrocarbons).

Definition 1.2. Given a bipartite graph G, in a subgraph of M, if any two edges in the edge set $\{e_1,e_2,\cdots e_m\}$ of M are not incident to the same vertex,

then M is called a matching. Selecting the largest subset of such edges is called the maximum matching of the graph G.

Theorem 1.1. [4] Let G be a bipartite graph. If the length of every cycle in G is congruent to 2 modulo 4, $\eta(G)=n-2q$, where n is the order of G and q is the maximum matching number.

As a special type of bipartite graph, trees possess unique zero properties. Specifically, a tree with a perfect matching is referred to as a PM tree. As stated in Theorem 1.1:

Theorem 1.2. [5] Let T be a tree, then $\eta(T)=n-2q$, where n is the order of and q is the maximum number of matches.

2. Design of Nullity Algorithm for Tree and Analysis of Algorithm Complexity

2.1. Basic Method

Based on Theorem 1.2, the following algorithm for calculating the nullity of a tree is introduced: First, calculate the maximum matching number q of the tree T by performing hierarchical priority storage and traversing the tree T in layer order to achieve matching and determine the maximum matching number. Then, using Theorem 1.2, the nullity value of the tree T is calculated.

2.2. Storage Structure of Tree Vertices (Nodes)

In order to facilitate the use of hierarchical traversal to achieve tree-to-tree T matching, the tree nodes contain the following five parts of information

among

Mark is the marker field, which is used to mark whether the tree nodes are matched or not with 0 and 1;

Data is the data domain;

Parent is a pointer to the parent node of that node;

Child is a pointer to the child node that points to the node;

Brother is a pointer to the brother node of the node.

2.3. Nullity Algorithm for Tree

The basic steps are:

1) Build a tree T with n vertices, matching set $M=\varphi$ and vertex $S=\varphi$;

2) Starting with any vertex in tree T, fix this vertex as the initial vertex v_0. The hierarchical classification of tree T proceeds as follows: Vertex v_0 is designated as Level 0. After removing v_0, adjacent vertices are classified as Level 1. Subsequently, vertices from Level 1 are removed and their adjacent vertices form Level 2, and so forth. Through this iterative process, all vertices in tree T can be systematically categorized into levels 0, 1, ..., up to Level P. Create a tree T with n vertices, a matching set, and a vertex set;

3) Select a vertex v_{p1} in level P, match v_{p1} with its parent node, mark parent node of v_{p1} and add it to the vertex set S, and add its matching edge to the match-

ing set M. The sibling nodes of vertex v_{p1} do not participate in the matching with its parent node. Select other vertices in level P and repeat the above matching process;

4) Select a vertex in level P-1 and repeat step (3) until all vertices in level 0 have been matched;

5) Count the number of edges that match the set, *i.e.* the maximum number of matches in the tree T;

6) Using Theorem 1.2, the zero value of a tree T is calculated by using the relationship between the maximum matching number of a tree and nullity.

Then one can obtain nullity algorithm for tree as Algorithm 2.1.

```
CBTree *bt; //Define a tree bt
Zero=0; //Zero represents its nullity, and it is initialized
Vertices=0; //Vertices represents the number of vertices in tree T, and it is initialized
MatchNum=0; //MatchNum represents the maximum matching number of tree T, and it is initialized
bt=CreateCBTree("A(B(D(,E(H,F)),C(G(I(,J(,K))))))"); //Call the constructor of tree T
preorder(bt, &MatchNum, &Vertices); //Call the function to find the maximum matching number of the tree T
Zero=Nullity( &MatchNum, Vertices); //Call the nullity function of the computation tree T
// Result output
printf("the number of vertices in tree T:    %d\n",Vertices);
printf("the maximum matching number of tree T:    %d\n",MatchNum);
printf("nullity of tree T:    %d\n",Zero);
}
```

The sub-functions are as follows:

1:Define the storage structure of a tree

```
typedef struct cbnode
  {       char data;
          int mark;
          struct cbnode *child;
          struct cbnode *brother;
          struct cbnode *parent;
  } CBTree;
```

2:Create tree T using a generalized list

```
CBTree *CreateCBTree(char *str)
{
top= -1, k=0, j=0;
x=NULL;
ch=str[j];
while(ch!='0')    //Create tree
{
```

```
//Convert the tree represented in the form of a generalized list to a layer-first storage format and create a tree T using the generalized list
switch(ch)
{
case '(':top++;st[top]=p;k=1;break;
case ')':top--;break;
case ',':k=2;break;
default:     //Store the nodes of the tree (create tree T)
p=(CBTree*)malloc(sizeof(CBTree));
      p->data=ch;    //Store the nodes of the tree
      p->mark=0;    //Assign initial values to the matching markers
      p->child=p->brother=NULL;
      p->parent=NULL;
      if(x==NULL)
        x=p;
      else
        {
          switch(k)
            {
              case 1:st[top]->child=p;break;
              case 2:st[top]->brother=p;break;
            }
         }
}
        j=j+1;
        ch=str[j];
}     //Complete the creation of tree T
return x;     //Return the stored tree T
}
3:erform maximum matching operation on tree T
void preorder(CBTree *p, int *N, int *vertices)
{
CBTree *str[MaxSize]; //Used to record the order in which vertices are visited during the layer-by-layer traversal of tree T
CBTree *s1[MaxSize],*s2[MaxSize];    //Dual-stack implementation for tree matching
CBTree *m;
int top1=-1; int top2=-1; int top=-1;    //Initialize the top pointer of the stack
top1++;
s1[top1]=p;
while(top1!=-1||top2!=-1)
{ while(top1!=-1)
{top++;
str[top]=s1[top1]; //Implement the storage of vertices of tree T in layers
```

```
if(s1[top1]->child!=NULL) //Perform level-order traversal on the tree
 {
    top2++;
    s2[top2]=s1[top1]->child;
    m=s2[top2];
    m->parent=s1[top1];
 }
while(m->brother!=NULL) //When using sibling nodes in the same layer, push its sibling nodes onto the stack before its child nodes.
{ top2++;
   s2[top2]=m->brother;                                    m->brother->parent=s1[top1]; //Push the brother node onto the stack
m=m->brother;    //Search for the next brother node
}
}
top1--;}
while(top2!=-1) //Use a double stack for data exchange to complete the level-order traversal of the tree.
{ top1++;
s1[top1]=s2[top2];
top2--;}
}    //The traversal of the tree in layers has ended
int i=0,num;
num=top+1;
//Calculate the maximum matching of tree T
while(top!=-1)
{if(str[top]->mark==0&& str[top]->parent->mark==0)
{    //Determine whether a node can match its parent nodes
    str[top]->parent->mark=1;       //Match successful
  i++; }
top--; }//The process of computing the maximum matching of tree T has ended
*N=i; //Maximum matching number of tree T
*vertices=num;    //The number of vertices in tree T
}
4:Calculate the nullity of tree T
int Nullity(int *n, int num)
{ return num-2*(*n);    //Calculate the nullity of tree T}
```

2.4. Algorithm Complexity Analysis

This algorithm first stores the vertices in the tree T (containing n vertices) in a top-down and left-right order in a layer-first manner, and then matches the last vertex with its parent vertices one by one. Therefore, the maximum running time of this algorithm to complete the maximum matching is $n-1$, its time complexity is $O(n)$ and memory complexity is $O(n)$. Theorem 1.2 clarifies the relationship

between a tree's maximum matching number and its nullity vertices. While literature [6] [7] provides an algorithm for calculating the maximum matching number in bipartite graphs with time complexity $O(mn)$, where m represents the number of edges and n denotes the vertex count, the implementation of this algorithm proves challenging. In contrast, literature [8] presents a nullity algorithm for trees that achieves a time complexity of $O\left(n^2/2\right)$ when determining maximum matching.

3. Case Analysis

Figure 1 is a tree with 11 vertices and 4 layers.

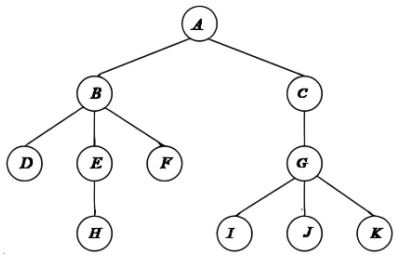

Figure 1. A tree with 11 vertices and 4 layers.

According to the above algorithm:

Input: Represent the tree T in the form of a generalized list

"A(B(D(,E(H,F)),C(G(I(,J(,K))))))"

Output: Storage structure of tree T, Figure 2 is the storage structure diagram of tree T with 5 fields.

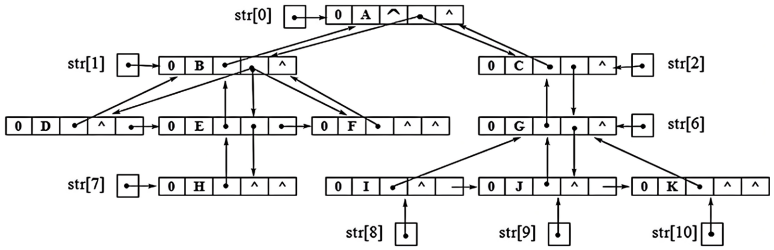

Figure 2. Storage structure of tree T.

Figure 3 is matching results:

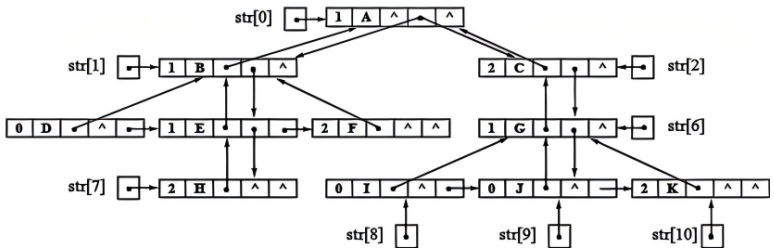

Figure 3. Storage structure diagram of tree T after matching is completed.

Matching edge set: GK, EH, BF, AC, the maximum number of matches is 4, and its nullity is 11 − 2*4 = 3.

4. Conclusion

To facilitate the calculation of the zero degree of a tree, this paper completes the input of the tree in the form of a generalized list, adopts a 5-field node storage method to store the tree, and then applies the relationship between the maximum matching number and the zero degree of the tree to calculate the maximum matching number and zero degree of any tree. The nullity algorithm of trees can be applied to the fields of network stability analysis and signal processing. By analyzing the nullity of a graph (*i.e.*, the number of zero eigenvalues of the matrix), the stability or anti-interference capability of a system can be evaluated. For example, in communication network design, the nullity algorithm can be used to assess the network's resistance to noise or signal interference; in circuit design, it can be used to predict the system's resonance characteristics at specific frequencies. The nullity of trees has a good application background, and there are many problems related to the nullity that need to be solved. For example, the implementation of better hierarchical sorting for large vertex trees, the construction of more concise input representation forms for trees; the general nullity algorithm for graphs and its implementation have not been given yet.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Von Collatz, L. and Sinogowitz, U. (1957) Spektren endlicher grafen. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, **21**, 63-77. https://doi.org/10.1007/bf02941924

[2] Longuet-Higgins, H.C. (1950) Some Studies in Molecular Orbital Theory I. Resonance Structures and Molecular Orbitals in Unsaturated Hydrocarbons. *The Journal of Chemical Physics*, **18**, 265-274. https://doi.org/10.1063/1.1747618

[3] Cvetkovic, D.M., Doob, M. and Sachs, H. (1985) Spectra of Graphs. Johann Barth Verlag.

[4] Cvetkovic, D.M., Gutman, I. and Trinajstic, N. (1972) Graph Theory and Molecular Orbitals, II. *Croatica Chemica Acta*, **44**, 365-374.

[5] Cvetkovic, D.M. and Gutman, I. (1972) The Algebraic Multiplicity of the Number Zero in the Spectrum of a Bipartite Graph. *Matematički Vesnik*, **9**, 141-150.

[6] Schwenk, A.J. (1973) Almost All Trees Are Cospectral. In: Harary, F., Ed., *New Directions in the Theory of Graphs*, Academic Press, 275-307.

[7] Tan, X.Z. and Liu, B.L. (2005) On the Nullity of Unicyclic Graphs. *Linear Algebra and its Applications*, **408**, 212-220. https://doi.org/10.1016/j.laa.2005.06.012

[8] Wu, T.Z. and Hu, S.B. (2010) Zeroth Degree of Graphs T with Radius Less than or Equal to Half the Spectrum. *Journal of Southwest Normal University* (*Natural Science Edition*), No. 4, 95-99.