A Decision Algorithm of Strong Connexity Convenient for Parallel Computation

Ruixian Li

School of Transportation and Vehicle Engineering, Shandong University of Technology, Zibo, China

E-mail: lrx@sdut.edu.cn

Abstract

In this article, the author puts forward a new decision algorithm of strong connexity, which can conduct the two-time search in the decision process simultaneously, and this algorithm is convenient for the parallel computation.

Keywords: Oriented graph, Strong connexity, Parallel computation, Search

1. Introduction

For the strong connexity decision of the oriented graph, the general algorithm is to implement the deeply first search (DFS) from certain peak V, and then reversely search from the last peak W to obtain the decision result. The characteristic of this algorithm is that the start of the second search is the terminal of the first search, i.e. the start time of the second search is limited by the first search.

For the decision algorithm of strong connexity in this article, both two searching processes start from the same peak, and they respectively search in the original graph and in the reverse graph. Therefore, two searches can be implemented at the same time, and this algorithm is convenient for the parallel computation.

2. Definition of the reverse graph

Set the graph G(V, E), and the graph G'(V, E') is the reverse graph of G, where, for any $< V_i, V_j > \in E$, $< V_j ,V_i > \in E'$ (seen in Figure 1).

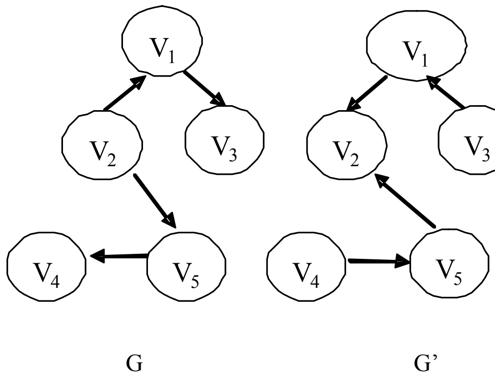


Figure 1. Examples of Reverse Graphs

3. The strong connexity decision theorem of the oriented graph

In an oriented graph G(V, E), for any peak $V_0 \in V$, if it fulfills two following conditions,

(a) Start from V_0, all peaks can be traversed through DFS in G.

(b) Start from V_0, all peaks can be traversed through DFS in G', and G'(V, E') is the reverse graph of the graph G(V, E).

So the oriented graph G(V, E) has strong connexity.

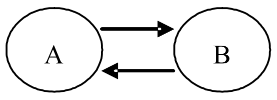
Prove:

Suppose the peak number in the graph is N.

i. When K=2, there are only two peaks in the oriented graph G, and if G fulfills (a) and (b), so it is seen in Figure 2. Obviously, Figure 2 has strong connexity.

ii. Suppose that when the peak number is N, the proposition comes into existence. When the peak number in G is N, and these peaks fulfill the conditions (a) and (b), so the graph G has strong connexity. Next, we will prove when the peak number is N+1, the proposition comes into existence. Suppose that increasing a new peak V (seen in Figure 3), if it fulfills the condition (a), a peak A certainly exists in G, and it appoints to the peak V. In the same way, if it fulfills the condition (b), a peak B certainly exists in G, and the peak V appoints to Q (seen in Figure 3). For any peak in G, because G has strong connexity, there is an approach leading to A and V, on the contrary, the peak V can arrive any peak in G through B. So the graph with the peak V has strong connexity.

From i and ii, to any peak number N, if the oriented graph fulfills (a) and (b), the graph has strong connexity.



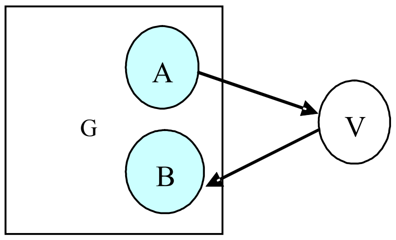


Figure 2. The Strong Connexity Graph When K=2 Figure 3. The Strong Connexity Graph When N+1

4. The strong connexity decision algorithm of the oriented graph

From the decision theorem, the strong connexity decision algorithm of the oriented graph can be obtained (seen in Figure 4).

Supposing that set the graph G, and the reverse graph is G', and the peak number is N, and start from the peak V to search.

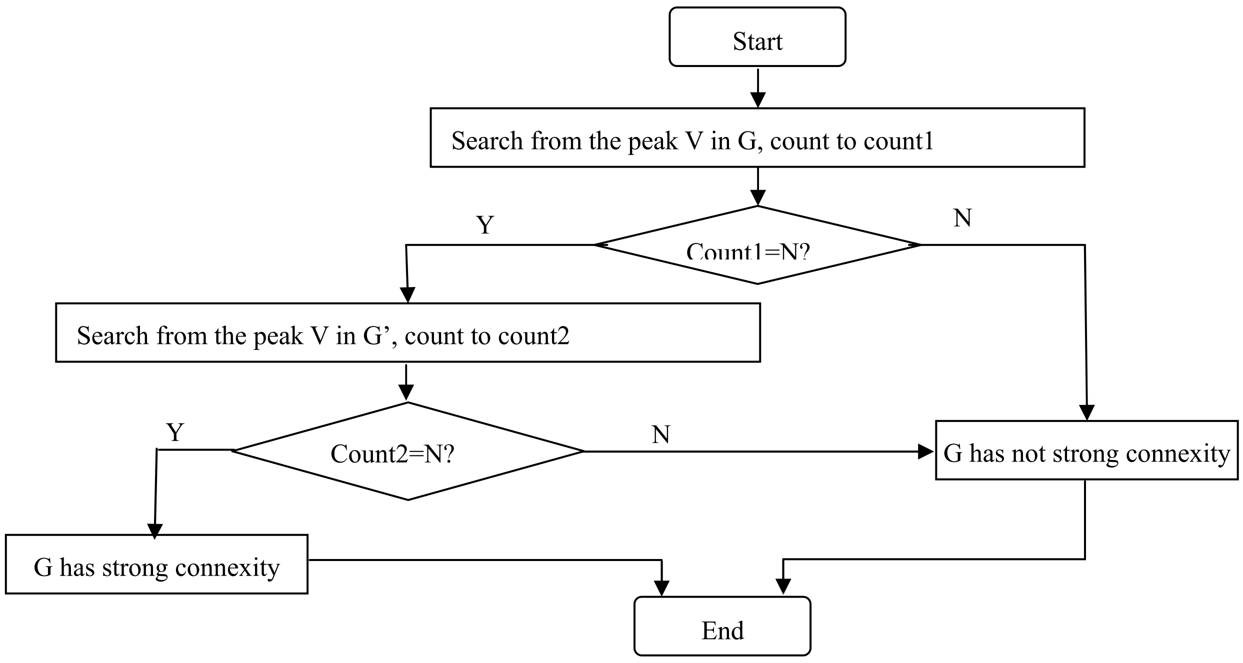

Start
Search from the peak V in G, count to count1
Count1=N?
Y
N
Search from the peak V in G', count to count2
Count2=N?
Y
N
G has not strong connexity
G has strong connexity
End

Figure 4. Flow Chart of the Algorithm

5. Implementation of the strong connexity decision algorithm of the oriented graph

When implementing the algorithm, the adjacency matrix can be adopted to store the graph G. Because the adjacency matrix of its reverse graph is the transpose of the adjacency matrix of G, it needs not to be stored specially, and the information of the reverse graph can be obtained from the adjacency matrix of G.

In the following program, MAXNODE is the constant, and it denotes the maximum peak number which can be computed. "nodecount" is the peak number of the oriented graph G.

void connexity1(bool matrix[][MAXNODE],int startnode)// Implement DFS in the oriented graph which adjacency matrix is "matrix" from "startnode", and the peak numbers searched are stored in the global variable count1.

```
{
    visited1[startnode]=1;count1++;
    for(int k=0;k<nodecount;k++)
    if (!visited1[k] && matrix[startnode][k]) connexity1(matrix,k);
}
void connexity2(bool matrix[][MAXNODE],int startnode)// Implement DFS in the reverse graph which adjacency matrix is "matrix" from "startnode", and the peak numbers searched are stored in the global variable count2.
{
    visited2[startnode]=1;
    count2++;
    for(int k=0;k<nodecount;k++)
    if (!visited2[k] && matrix[k][startnode]) connexity2(matrix,k);
}
bool stronglyconnexity(bool matrixofgraph[][MAXNODE])// Suppose the adjacency matrix of the oriented graph G is "matrixofgraph", decide its strong connexity, and if G is the strongly connected graph, return TRUE, or else, return FALSE.
{
    for(i=0;i<nodecount;i++)    visited1[i]=0;
    for(i=0;i<nodecount;i++)    visited2[i]=0;
    connexity1(matrixofgraph,start);
    if (count1!=nodecount)
        return false;
    connexity2(matrixofgraph,start);
    if (count2!=nodecount)
        return    false;
    else    return true;
}
```

6. Conclusions

The decision algorithm which could decide the connexity of the oriented graph in the article is very simple, and it is convenient for the parallel computation.

References

G. Brassard & P. Bratley. (1996). Fundamentals of Algorithms. Englewood Cliffs, N.J.: Prentice Hall.

Liu, Xiaoli et al. (2005). An Analysis on the Strongly Connexity of Digraph and the Way of Identification. Computer Applications and Software. No. 22(4).

William Ford & William Topp. (1996). Data structures with C++ Using STL. Englewood Cliffs, N.J.: Prentice Hall,1996

Yan, Weimin & Wu, Weimin. (1997). Data Structure. Beijing: Tsinghua University Press.

Zhang, Li'ang & Liutian. (2000). Theoretic Base of Computer. Beijing: Tsinghua University Press.