

Topological Sort Algorithm Using Source Removal Method

Algorithm topological sort(a,n,s)

Purpose: To obtain the sequence of jobs to be executed resulting in topological order

Input: a: -Adjacency Matrix of the given graph

n: The number of vertices in graph

Output:

S: Indicates the jobs that are to be executed in order.

for j<-0 to n-1 do

 sum<-0

 for i<-0 to n-1 do

 sum<- sum+a[i][j]

 end for

 Indegree[j]<-sum

end for

top<- -1

for i<-0 to n-1 do

 if(indegree[i]=0)

 top<-top+1

 s[top]<-i

 end if

end for

while top \neq -1

 u<- s[top]

 top<- top -1

 Add u to solution vector T

 For each vector v adjacent to u

 Decrement indegree[v] by one

```

        If (indegree[v]=0)
            Top<- top+1
            S[top]<-v
        end if
    end for
end while

Write T // Output of topological sequence
Return

```

Code:

```

#include <stdio.h>
#include <stdlib.h>

void ts(int **a, int n) {
    int indegree[n], s[n], top = -1, T[n], k = 0;
    for (int j = 0; j < n; j++) {
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += a[i][j];
        }
        indegree[j] = sum;
    }
    for (int i = 0; i < n; i++) {
        if (indegree[i] == 0) {
            s[++top] = i;
        }
    }
    while (top != -1) {
        int u = s[top--];
        T[k++] = u;
        for (int v = 0; v < n; v++) {
            if (a[u][v] == 1) {

```

```

    indegree[v]--;
    if (indegree[v] == 0) {
        s[++top] = v;
    }
}
}
}
}

printf("Topological Order: ");
for (int i = 0; i < k; i++) {
    printf("%d ", T[i]);
}
printf("\n");
}

int main() {
    int n;

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    int **a = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; i++) {
        a[i] = (int *)malloc(n * sizeof(int));
    }

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    ts(a, n);

    for (int i = 0; i < n; i++) {
        free(a[i]);
    }
}

```

```

}
free(a);
return 0;
}

```

Output:

```

Enter the number of vertices: 5
Enter the adjacency matrix:
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0
Topological Order: 1 0 2 3 4

Process returned 0 (0x0)   execution time : 43.704 s
Press any key to continue.
|

```

Topological Sort Algorithm Using DFS

ALGORITHM DFS(u,n,a)

//Input u- From where the DFS traversal start

//n- the number of vertices in the graph

//a –adjacency matrix

//Global variables

// s –to know what are the nodes visited and which are the nodes that are visited.

// j- index variable to store the vertices (only those nodes which are dead nodes

Or those nodes whose nodes are completely explored.

//res- an array which holds the order in which the vertices are popped.

//Output

Res-Indicates the vertices in the reverse order that are to be executed.

Step 1:[Visit the vertex u]

S[u]<-1

Step 2[Traverse deeper in to the graph till we get dead ends or till all vertices
Are visited]

```
For v<- 0 to n-1 do
    If(a[u][v] = 1 and s[v] =0) then
        DFS(v,n,a)
    End if
End For
```

Step 3: [Store the dead vertex or which is completely explored]

```
J<- j+1
res[j]<- u
```

Step 4: [Finished]

```
Return
```

Algorithm topological_order(n,a)

//Input a- adjacency matrix of the given graph
n- the number of vertices in the graph

//Global variables

s- to know what are the nodes visited and what are the nodes not visited
j- Index variable to store the vertices (only those nodes which are dead nodes or those
nodes whose nodes are completely explored
res- an array which holds the order in which the vertices are popped.

//Output

Res

Step 1 for i<-0 to n-1 do

```
S[i]<-0
End for
```

j<-0 //An index to store the vertices which are dead ends and which are completely
explored.

Step 2: [process each vertex in the graph]

```
For u<- 0 to n-1 do
    If(s[u]=0) call DFS(u,n,a)
End for
```

Step 3 [output the Topological sequence by printing in the reverse order of
Popped sequence]

```
For i<- n-1 down to 0
```

```
        Print res[i]
    End for
```

Step 4: [Finished]

```
    return
```

Code:

```
#include <stdio.h>
#include <stdlib.h>

void DFS(int u, int n, int **a, int *s, int *res, int *j) {
    s[u] = 1;
    for (int v = 0; v < n; v++) {
        if (a[u][v] == 1 && s[v] == 0) {
            DFS(v, n, a, s, res, j);
        }
    }
    res[(*j)++] = u;
}

void to(int n, int **a) {
    int s[n];
    int res[n];
    int j = 0;
    for (int i = 0; i < n; i++) {
        s[i] = 0;
    }
    for (int u = 0; u < n; u++) {
        if (s[u] == 0) {
            DFS(u, n, a, s, res, &j);
        }
    }
    printf("Topological Order: ");
    for (int i = n - 1; i >= 0; i--) {
        printf("%d ", res[i]);
    }
}
```

```
}  
printf("\n");  
}  
int main() {  
    int n;  
    printf("Enter the number of vertices: ");  
    scanf("%d", &n);  
    int **a = (int **)malloc(n * sizeof(int *));  
    for (int i = 0; i < n; i++) {  
        a[i] = (int *)malloc(n * sizeof(int));  
    }  
    printf("Enter the adjacency matrix:\n");  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < n; j++) {  
            scanf("%d", &a[i][j]);  
        }  
    }  
    to(n, a);  
    for (int i = 0; i < n; i++) {  
        free(a[i]);  
    }  
    free(a);  
    return 0;  
}
```

Output:

```
Enter the number of vertices: 8
Enter the adjacency matrix:
0 1 1 1 1 0 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 0 0 0 0 1
0 0 0 1 1 0 0 1
0 0 0 0 0 1 1 0
Topological Order: 0 1 5 7 6 4 3 2

Process returned 0 (0x0)   execution time : 16.266 s
Press any key to continue.
|
```