## Depth First Search

*Description:*

Depth First Search Algorithm is just as the name of the algorithm: as long as possible, if there exists node. *( Depth First Search Algorithm can be used not only on the Undirected Graph but also on the Directed Graph. )*

*Phase 1:*

Depth First Search Algorithm would search from the starting edge of the Latest Found Node v, until Each Edge of the node has been found. The process finished.

*Phase 2:*

Once all starting edges of the node v have been visited, the search process would return back to the precursor node of the node v to find the starting edges of the priority node.

*Phase 3:*

As long as all precursor nodes of the current node have been visited, then if there still has some other unfounded nodes, then Depth First Search Algorithm would pick up a random node from all unfounded nodes as the new node, and to repeat the same query process.

*Phase 4:*

The Depth First Search Algorithm starts repeating, until all nodes in the Graph have been visited.

*Key:*

* Multiple Depth Priority Trees of Priority Sub - Graph forms the Depth Priority forest. Still, the edge in the Tree forest is still called the edge of tree.
* Need to attention that, just like Breadth First Search Algorithm, the Depth Priority Tree makes the color of node to display the status of node.
* *The initial color of node equals to White, and once the node has been found, then it turns to Gray. After Adjacent Linked List has been completely scanned, the color of node turns to Black.*

*(Such method ensures that each node exist in only one Depth First Tree. Therefore, the conclusion can be reached, which is to say, all Depth First Tree is disjoint.)*

* Except to create the Adjacent Linked List Tree, the Depth First Tree also create one time stamp for each node to signify when the color of node has been updated to Gray and Black.
* *Two timestamps - one timestamp Node.d is used to record the time it turns from White to Gray, and another timestamp Node.f is used to record the time it turns from Gray to Black.*
* These timestamps provide enough important information for the Graph, normally it can used to help deduce the behavior of Depth First Search Algorithm.

*(Apparently, for each node u, u.d < u.f, which means the node u is White before u.d and turns to Black after u.f.)*

*Example 1:*

*Procedure:*

* *Besides Graph and Adjacent Linked List, data structure stack is also needed to store all visited nodes.*
* *According to the information above, the structure of node is just as below:*

*structure Node {*

*int value; // The value of node.*

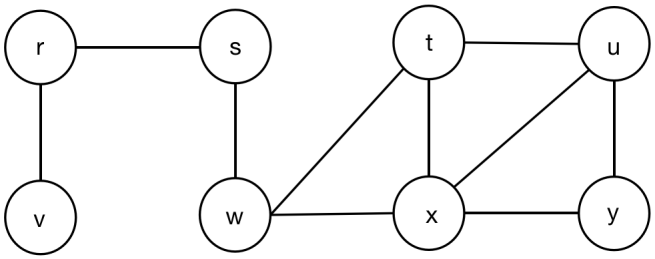
*Node\* priority; // The priority of the current Node.*

*string color; // The color of node, include White, Gray, and Black.*

*int formertime; // Record time when Node turns from White to Gray.*

*int lattertime; // Record time when Node turns from Gray to Black.*

*};*

**

**

*Step 1 - Initialize and update all related information of all nodes.*

*node->value = Value;*

*node->priority = NIL;*

*node->color = White;*

*node->formertime = 0;*

*node->lattertime = 0;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *White* | *0* | *0* |
| *r* | *NIL* | *White* | *0* | *0* |
| *w* | *NIL* | *White* | *0* | *0* |
| *v* | *NIL* | *White* | *0* | *0* |
| *t* | *NIL* | *White* | *0* | *0* |
| *x* | *NIL* | *White* | *0* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 2 - Recursively visit all nodes of source node s and update the node status.*

*node->value = s;*

*node->priority = NIL;*

*node->color = Gray;*

*node->formertime = 1;*

*node->lattertime = 0;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *Gray* | *1* | *0* |
| *r* | *NIL* | *White* | *0* | *0* |
| *w* | *NIL* | *White* | *0* | *0* |
| *v* | *NIL* | *White* | *0* | *0* |
| *t* | *NIL* | *White* | *0* | *0* |
| *x* | *NIL* | *White* | *0* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 3 - Push the source node s into queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 4 - Recursively visit all neighbor nodes of source node s.*

**

*Step 5 - Find the first non - visited node with the color of White among all neighbor nodes of source node s. The first non - visited node is r and update the node status.*

*node->value = r;*

*node->priority = s;*

*node->color = Gray;*

*node->formertime = 1 + 1 = 2;*

*node->lattertime = 0;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *Gray* | *1* | *0* |
| *r* | *s* | *Gray* | *2* | *0* |
| *w* | *NIL* | *White* | *0* | *0* |
| *v* | *NIL* | *White* | *0* | *0* |
| *t* | *NIL* | *White* | *0* | *0* |
| *x* | *NIL* | *White* | *0* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 6 - Push the first non - visited node r into the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *r* | *NIL* | *Gray* | *2* | *0* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 7 - Recursively visit all neighbor nodes of node r.*

**

*Step 8 - Find the first non - visited node with color White among all neighbor nodes of node r. The first non - visited node is v and update the node status.*

*node->value = v;*

*node->priority = r;*

*node->color = Gray;*

*node->formertime = 3;*

*node->lattertime = 0;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *Gray* | *1* | *0* |
| *r* | *s* | *Gray* | *2* | *0* |
| *w* | *NIL* | *White* | *0* | *0* |
| *v* | *r* | *Gray* | *3* | *0* |
| *t* | *NIL* | *White* | *0* | *0* |
| *x* | *NIL* | *White* | *0* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 9 - Push the node v into the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *v* | *r* | *Gray* | *3* | *0* |
| *r* | *s* | *Gray* | *2* | *0* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 10 - Recursively visit all neighbor nodes of node v.*

**

*Step 11 - Find the first non - visited node of node v with the color of White among all neighbor nodes of node v. However, the process failed since the node r has been visited. Update the status of node v.*

*node->value = v;*

*node->priority = r;*

*node->color = Black;*

*node->formertime = 3;*

*node->lattertime = time + 1 = 3 + 1 = 4;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *Gray* | *1* | *0* |
| *r* | *s* | *Gray* | *2* | *0* |
| *w* | *NIL* | *White* | *0* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *t* | *NIL* | *White* | *0* | *0* |
| *x* | *NIL* | *White* | *0* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 12 - Pop the top node v out of the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Gray* | *2* | *0* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 13 - Return back to Step 7, and find the second non - visited node of node r with the color of White among all neighbor nodes of node r. However, the process failed since all neighbor nodes of node r have been visited. Update the status of node r.*

*node->value = r;*

*node->priority = s;*

*node->color = Black;*

*node->formertime = 2;*

*node->lattertime = time + 1 = 4 + 1 = 5;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *Gray* | *1* | *0* |
| *r* | *s* | *Black* | *2* | *5* |
| *w* | *NIL* | *White* | *0* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *t* | *NIL* | *White* | *0* | *0* |
| *x* | *NIL* | *White* | *0* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 14 - Pop top node r out of the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 15 - Return back to Step 4, visit the next node of source node s.*

**

*Step 16 - Find the second node of source node s, with the color of White among all neighbor nodes of source node r. The next non - visited node is w. Update the status of node w.*

*node->value = w;*

*node->priority = s;*

*node->color = Gray;*

*node->formertime = time + 1 = 5 + 1 = 6;*

*node->lattertime = 0;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *Gray* | *1* | *0* |
| *r* | *s* | *Black* | *2* | *5* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *t* | *NIL* | *White* | *0* | *0* |
| *x* | *NIL* | *White* | *0* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 17 - Push the node w into the queue.*

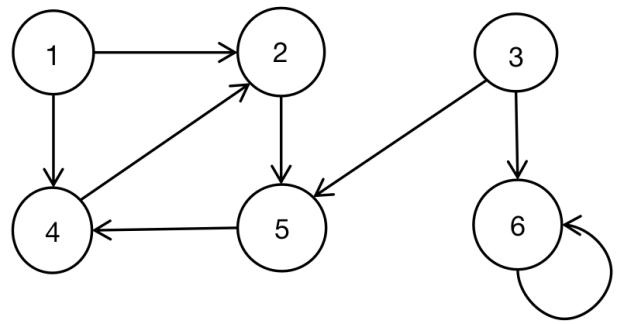
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 18 - Recursively visit all nodes of node w.*

**

*Step 18 -*

*Example 2:*





*Procedure:*

*Code:*