## 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:*

*Structure:*

* *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.*

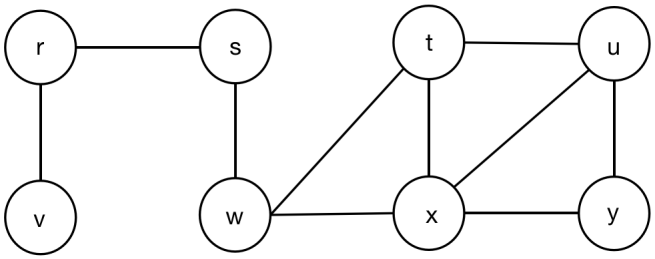
*string color; // The color of node, include White, Gray, and Black.*

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

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

*};*

*Procedure:*

**

**

*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 19 - Find the first non - visited node of node w, with the color of White among all neighbor nodes of source node w. The next non - visited node is t. Update all information of node t.*

*node->value = t;*

*node->priority = w;*

*node->color = Gray;*

*node->formertime = time + 1 = 6 + 1 = 7;*

*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* | *w* | *Gray* | *7* | *0* |
| *x* | *NIL* | *White* | *0* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 20 - Push the node t into the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *t* | *w* | *Gray* | *7* | *0* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 21 - Recursively visit all neighbor nodes of node t.*

**

*Step 22 - Find the first non - visited node of node t, with the color of White among all neighbor nodes of source node t. The next non - visited node is x. Update all information of node x.*

*node->value = x;*

*node->priority = t;*

*node->color = Gray;*

*node->formertime = time + 1 = 8;*

*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* | *w* | *Gray* | *7* | *0* |
| *x* | *t* | *Gray* | *8* | *0* |
| *u* | *NIL* | *White* | *0* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 23 - Push the node x into the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *x* | *t* | *Gray* | *8* | *0* |
| *t* | *w* | *Gray* | *7* | *0* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 24 - Recursively visit all neighbor nodes of node x.*

**

*Step 25 - Find the first non - visited node of node x, with the color of White among all neighbor nodes of source node x. The non - visited node is u. Update all information of node u.*

*node->value = u;*

*node->priority = x;*

*node->color = Gray;*

*node->formertime = time + 1 = 9;*

*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* | *w* | *Gray* | *7* | *0* |
| *x* | *t* | *Gray* | *8* | *0* |
| *u* | *x* | *Gray* | *9* | *0* |
| *y* | *NIL* | *White* | *0* | *0* |

*Step 26 - Push the node u into the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
| *u* | *x* | *Gray* | *9* | *0* |
| *x* | *t* | *Gray* | *8* | *0* |
| *t* | *w* | *Gray* | *7* | *0* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 27 - Recursively visit all neighbor nodes of node u.*

**

*Step 28 - Find the first non - visited node of node u, with the color of White among all neighbor nodes of source node u. The first non - visited node is y. Update all information of node y.*

*node->value = y;*

*node->priority = u;*

*node->color = Gray;*

*node->formertime = time + 1 = 10;*

*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* | *w* | *Gray* | *7* | *0* |
| *x* | *t* | *Gray* | *8* | *0* |
| *u* | *x* | *Gray* | *9* | *0* |
| *y* | *u* | *Gray* | *10* | *0* |

*Step 29 - Push the node y into the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
| *y* | *u* | *Gray* | *10* | *0* |
| *u* | *x* | *Gray* | *9* | *0* |
| *x* | *t* | *Gray* | *8* | *0* |
| *t* | *w* | *Gray* | *7* | *0* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 30 - Find the second non - visited node of node y, with the color of White among all neighbor nodes of source node y. However, there has no other nodes. Update the status of node y.*

*node->value = y;*

*node->priority = u;*

*node->color = Black;*

*node->formertime = 10;*

*node->lattertime = time + 1 = 10 + 1 = 11;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *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* | *w* | *Gray* | *7* | *0* |
| *x* | *t* | *Gray* | *8* | *0* |
| *u* | *x* | *Gray* | *9* | *0* |
| *y* | *u* | *Black* | *10* | *11* |

*Step 31 - Pop the node y out of the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
| *y* | *u* | *Black* | *10* | *11* |
| *u* | *x* | *Gray* | *9* | *0* |
| *x* | *t* | *Gray* | *8* | *0* |
| *t* | *w* | *Gray* | *7* | *0* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 32 - Find the second non - visited node of node u, with the color of White among all neighbor nodes of source node u. However, there has no other nodes of node u. Update the status of node u.*

*node->value = u;*

*node->priority = x;*

*node->color = Black;*

*node->formertime = 9;*

*node->lattertime = time + 1 = 11 + 1 = 12;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *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* | *w* | *Gray* | *7* | *0* |
| *x* | *t* | *Gray* | *8* | *0* |
| *u* | *x* | *Black* | *9* | *12* |
| *y* | *u* | *Gray* | *10* | *11* |

*Step 33 - Pop the node u out of the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
| *y* | *u* | *Black* | *10* | *11* |
| *u* | *x* | *Black* | *9* | *12* |
| *x* | *t* | *Gray* | *8* | *0* |
| *t* | *w* | *Gray* | *7* | *0* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 34 - Find the next neighbor node of node x, with the color of White among all neighbor nodes of source node x. However, there has no other nodes of node x. Update the information of node x.*

*node->value = x;*

*node->priority = t;*

*node->color = Black;*

*node->formertime = 8;*

*node->lattertime = time + 1 = 12 + 1 = 13;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *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* | *w* | *Gray* | *7* | *0* |
| *x* | *t* | *Black* | *8* | *13* |
| *u* | *x* | *Black* | *9* | *12* |
| *y* | *u* | *Black* | *10* | *11* |

*Step 35 - Pop the node x out of the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
| *y* | *u* | *Black* | *10* | *11* |
| *u* | *x* | *Black* | *9* | *12* |
| *x* | *t* | *Black* | *8* | *13* |
| *t* | *w* | *Gray* | *7* | *0* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 36 - Find the next non - visited node of node t, with the color of White among all neighbor nodes of source node t. However, there has no other nodes of node t. Update all information of node t.*

*node->value = t;*

*node->priority = w;*

*node->color = Black;*

*node->formertime = 7;*

*node->lattertime = time + 1 = 14;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *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* | *w* | *Black* | *7* | *14* |
| *x* | *t* | *Black* | *8* | *13* |
| *u* | *x* | *Black* | *9* | *12* |
| *y* | *u* | *Black* | *10* | *11* |

*Step 37 - Pop out the node t from the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
| *y* | *u* | *Black* | *10* | *11* |
| *u* | *x* | *Black* | *9* | *12* |
| *x* | *t* | *Black* | *8* | *13* |
| *t* | *w* | *Black* | *7* | *14* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 38 - Find the next non - visited node of node w, with the color of White among all neighbor nodes of source node w. However, there has no other nodes of node w. Update all information of node w.*

*node->value = w;*

*node->priority = s;*

*node->color = Black;*

*node->formertime = 6;*

*node->lattertime = time + 1 = 14 + 1 = 15;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *Gray* | *1* | *0* |
| *r* | *s* | *Black* | *2* | *5* |
| *w* | *s* | *Gray* | *6* | *15* |
| *v* | *r* | *Black* | *3* | *4* |
| *t* | *w* | *Black* | *7* | *14* |
| *x* | *t* | *Black* | *8* | *13* |
| *u* | *x* | *Black* | *9* | *12* |
| *y* | *u* | *Black* | *10* | *11* |

*Step 39 - Pop the node w out of the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
| *y* | *u* | *Black* | *10* | *11* |
| *u* | *x* | *Black* | *9* | *12* |
| *x* | *t* | *Black* | *8* | *13* |
| *t* | *w* | *Black* | *7* | *14* |
| *w* | *s* | *Black* | *6* | *15* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Gray* | *1* | *0* |

*Step 40 - Find the next non - visited node of node s, with the color of White among all neighbor nodes of source node s. However, there has no other nodes of node w. Update all information of node s.*

*node->value = s;*

*node->priority = NIL;*

*node->color = Black;*

*node->formertime = 1;*

*node->lattertime = time + 1 = 15 + 1 = 16;*

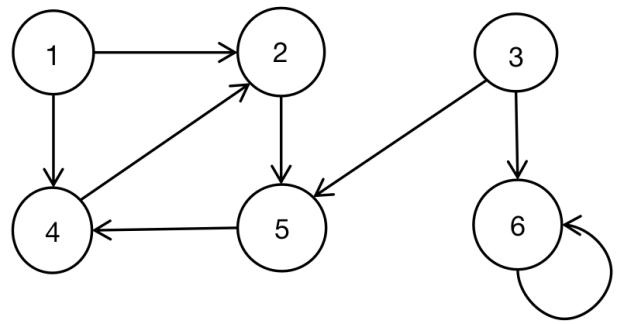
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *s* | *NIL* | *Black* | *1* | *15* |
| *r* | *s* | *Black* | *2* | *5* |
| *w* | *s* | *Gray* | *6* | *0* |
| *v* | *r* | *Black* | *3* | *4* |
| *t* | *w* | *Black* | *7* | *14* |
| *x* | *t* | *Black* | *8* | *13* |
| *u* | *x* | *Black* | *9* | *12* |
| *y* | *u* | *Black* | *10* | *11* |

*Step 41 - Pop the node s out of the queue.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
| *y* | *u* | *Black* | *10* | *11* |
| *u* | *x* | *Black* | *9* | *12* |
| *x* | *t* | *Black* | *8* | *13* |
| *t* | *w* | *Black* | *7* | *14* |
| *w* | *s* | *Black* | *6* | *15* |
| *v* | *r* | *Black* | *3* | *4* |
| *r* | *s* | *Black* | *2* | *5* |
| *s* | *NIL* | *Black* | *1* | *16* |

*End here, the procedure has been finished.*

*Example 2:*





*Procedure:*

Step 1 - Initialize 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* |
| *node1* | *NIL* | *White* | *0* | *0* |
| *node2* | *NIL* | *White* | *0* | *0* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *NIL* | *White* | *0* | *0* |
| *node5* | *NIL* | *White* | *0* | *0* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 2 - Prepare to visit all neighbor nodes of node1.



Step 3 - Update all information of node1.

*node->value = node1;*

*node->priority = NIL;*

*node->color = Gray;*

*node->formertime = 1;*

*node->lattertime = 0;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Gray* | *1* | *0* |
| *node2* | *NIL* | *White* | *0* | *0* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *NIL* | *White* | *0* | *0* |
| *node5* | *NIL* | *White* | *0* | *0* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 2 - Push the node1 into the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *node1* | *NIL* | *Gray* | *1* | *0* |

Step 3 - Recursively visit all neighbor node of node1. Find the first neighbor node of node 1, take the node1 as starting node, and the color of node is White. Checking the Adjacent Linked List, and find that the first neighbor node is node2. Update all related information of node2.

*node->value = node2;*

*node->priority = node1;*

*node->color = Gray;*

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

node->lattertime = 0;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Gray* | *1* | *0* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *NIL* | *White* | *0* | *0* |
| *node5* | *NIL* | *White* | *0* | *0* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 4 - Push the node2 into the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *node2* | *NIL* | *Gray* | *2* | *0* |
| *node1* | *NIL* | *Gray* | *1* | *0* |

Step 5 - Prepare to visit all neighbor nodes of node2.



Step 6 - Find the first non - visited neighbor node of node2, take the node2 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that the first neighbor node is node5. Update all related information of node5.

*node->value = node5;*

*node->priority = node2;*

*node->color = Gray;*

*node->formertime = time + 1 = 2 + 1 = 3;*

node->lattertime = 0;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Gray* | *1* | *0* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *NIL* | *White* | *0* | *0* |
| *node5* | *node2* | *Gray* | *3* | *0* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 7 - Push the node5 into the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| *node5* | *node2* | *Gray* | *3* | *0* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node1* | *NIL* | *Gray* | *1* | *0* |

Step 8 - Prepare to visit all neighbor nodes of node5.



Step 9 - Find the first non - visited neighbor node of node5, take the node5 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that the first non - visited neighbor node of node5 is node4. Update all information of node4.

*node->value = node4;*

*node->priority = node5;*

*node->color = Gray;*

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

node->lattertime = 0;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Gray* | *1* | *0* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *node5* | *Gray* | *4* | *0* |
| *node5* | *node2* | *Gray* | *3* | *0* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 10 - Push the node5 into queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
| *node4* | *node5* | *Gray* | *4* | *0* |
| *node5* | *node2* | *Gray* | *3* | *0* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node1* | *NIL* | *Gray* | *1* | *0* |

Step 11 - Prepare to visit all neighbor nodes of node 4.



Step 12 - Find the first non - visited neighbor node of node4, take the node4 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that all non - visited neighbor node of node4 have been visited. Update all information of node4.

*node->value = node4;*

*node->priority = node5;*

*node->color = Black;*

node->formertime = 4;

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Gray* | *1* | *0* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Gray* | *3* | *0* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 13 - Pop the node4 out of the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Gray* | *3* | *0* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node1* | *NIL* | *Gray* | *1* | *0* |

Step 14 - Find the next non - visited node of node5, take the node5 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that all non - visited neighbor node of node5 have been visited. Update all related information of node5.

*node->value = node5;*

*node->priority = node2;*

*node->color = Black;*

node->formertime = 3;

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Gray* | *1* | *0* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 15 - Pop the node5 out of the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node2* | *node1* | *Gray* | *2* | *0* |
| *node1* | *NIL* | *Gray* | *1* | *0* |

Step 16 - Find the next non - visited node of node2, take the node2 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that all non - visited neighbor node of node2 have been visited. Update all related information of node2.

node->value = node2;

*node->priority = node1;*

*node->color = Black;*

node->formertime = 2;

*node->lattertime = time + 1 = 6 + 1 = 7;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Gray* | *1* | *0* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 17 - Pop the node2 out of the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node1* | *NIL* | *Gray* | *1* | *0* |

Step 18 - Find the next non - visited node of node1, take the node1 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that all non - visited neighbor node of node1 have been visited. Update all related information of node1.

node->value = node1;

node->priority = NIL;

*node->color = Black;*

node->formertime = 1;

*node->lattertime = time + 1 = 7 + 1 = 8;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Black* | *1* | *8* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 19 - Pop the node1 out of queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
|  |  |  |  |  |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node1* | *NIL* | *Black* | *1* | *8* |

Step 20 - Find that there has no nodes in the queue, and but there still have nodes with the color White. Recursively find the first node with the color White, in this step, the node3 has been visited.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Black* | *1* | *8* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node3* | *NIL* | *White* | *0* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 21 - Recursively visit all related nodes of node3.



Step 22 - Update all related information of node3.

node->value = 3;

node->priority = NIL;

*node->color = Gray;*

*node->formertime = time + 1 = 8 + 1 = 9;*

node->lattertime = 0;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Black* | *1* | *8* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node3* | *NIL* | *Gray* | *9* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node6* | *NIL* | *White* | *0* | *0* |

Step 23 - Push the node3 into the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
|  |  |  |  |  |
| *node3* | *NIL* | *Gray* | *9* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node1* | *NIL* | *Black* | *1* | *8* |

Step 24 - Recursively visit all neighbor node of node3.



Step 25 - Find the first non - visited node of node3, take the node3 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that the first non - visited neighbor node of node3 is node6. Update all information of node6.

*node->value = node6;*

*node->priority = node3;*

*node->color = Gray;*

*node->formertime = time + 1 = 9 + 1 = 10;*

node->lattertime = 0;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Black* | *1* | *8* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node3* | *NIL* | *Gray* | *9* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node6* | *node3* | *Gray* | *10* | *0* |

Step 26 - Push node6 into the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node6* | *NIL* | *Gray* | *10* | *0* |
| *node3* | *NIL* | *Gray* | *9* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node1* | *NIL* | *Black* | *1* | *8* |

Step 27 - Find next non - visited node of node6, take the node6 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that there has no other non - visited nodes of around node6. Update all related information of node6.

*node->value = node6;*

*node->priority = node3;*

*node->color = Black;*

node->formertime = 10;

*node->lattertime = time + 1 = 10 + 1 = 11;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Black* | *1* | *8* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node3* | *NIL* | *Gray* | *9* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node6* | *node3* | *Black* | *10* | *11* |

Step 28 - Push node6 into the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node6* | *node3* | *Black* | *10* | *11* |
| *node3* | *NIL* | *Gray* | *9* | *0* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node1* | *NIL* | *Black* | *1* | *8* |

Step 29 - Find next non - visited node of node3, take the node3 as starting node, and the color of node as White. Checking the Adjacent Linked List, and find that there has no other non - visited nodes of around node3. Update all related information of node3.

node->value = node3;

node->priority = NIL;

*node->color = Black;*

node->formertime = 9;

*node->lattertime = time + 1 = 11 + 1 = 12;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node1* | *NIL* | *Black* | *1* | *8* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node3* | *NIL* | *Black* | *9* | *12* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node6* | *node3* | *Black* | *10* | *11* |

Step 30 - Pop the node3 out of the queue.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Value* | *Priority* | *Color* | *Former Time* | *Latter Time* |
| *node6* | *node3* | *Black* | *10* | *11* |
| *node3* | *NIL* | *Black* | *9* | *12* |
| *node4* | *node5* | *Black* | *4* | *5* |
| *node5* | *node2* | *Black* | *3* | *6* |
| *node2* | *node1* | *Black* | *2* | *7* |
| *node1* | *NIL* | *Black* | *1* | *8* |

*End here, the procedure has been finished.*

*Code:*

*structure Graph {*

*Node \* node; // The Node array, which is used to record all nodes of Graph;*

*int size; // The number of nodes of Graph;*

*}*

*structure Node {*

*int value; // The value of node in Graph;*

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

*Node \* next; // The next node of the current node;*

*string color; // The color of node;*

*int formertime; // The formertime of node;*

*int lattertime; // The lattertime of node;*

*}*

*/\**

*\*\* DFS Algorithm which is based on for cycle and stack data structure.*

*\*/*

void DFS(Graph \*graph) {

*/\**

*\*\* If the Graph structure equals to NULL or the size of Graph*

*\*\* equals to 0, return and finish DFS.*

*\*/*

if ( graph == NULL || graph->size == 0 ) {

return;

}

stack<Node \*> temp\_stack; // The stack is used as intermediate // data structure during DFS;

unordered\_map<char, Node \*> table; // The unordered\_map table is used to

// record all nodes in Graph;

Node \* node; // The intermediate structure node.

int size = graph->size; // Assign the size of graph to size.

*/\**

*\*\* All nodes information would be initialized and they should be pushed into table.*

*\*\**

*\*\* At first, all information of nodes are initialized as “White”, the first time*

*\*\* the node has been pushed into stack is 0, and the latter time the node*

*\*\* pop out of stack is also 0.*

*\*\**

*\*\* After that, node has been pushed into table.*

*\*/*

for ( int i = 0; i < size; i ++ ) {

/\*

\*\* Get each node from graph, and assign it to the node.

\*/

node = graph->node[ i ];

/\*

\*\* Initialize all information of node, including priority, color, formertime

\*\* and lattertime.

\*/

node->priority = NIL;

node->color = “White”;

node->formertime = 0;

node->lattertime = 0;

/\* Push all nodes of Graph into unordered\_map. \*/

table[ node->value ] = node;

}

*/\**

*\*\* Initialize the time of all nodes as 0.*

*\*/*

int time = 0;

*/\**

*\*\* Initialize related information for the first node.*

*\*/*

node = graph->node[ 0 ];

node->priority = NIL;

node->color = “White”;

node->formertime = 0;

node->lattertime = 0;

*/\**

*\*\* Recursively visit all nodes of Graph g, and try to find the node with color of*

*\*\* white. What need to do here is to record all nodes which start from the node*

*\*\* and the color of all these nodes is white. During this process, the stack data*

*\*\* structure temp\_stack would be used.*

*\*/*

for ( int i = 0; i < size; i ++ ) {

/\*

\*\* Get all nodes one by one from graph->node array.

\*/

Node \*node = graph->node[ i ];

if ( node->color == “White” ) {

temp\_stack.push\_back(node);

/\*

\*\* Set up the color “Gray” of node, and the formertime of it.

\*/

node = temp\_stack.top();

node->color = “Gray”;

node->formertime = time;

time ++;

while ( node->next != NIL ) {

*/\**

*\*\* If the color of node equals to “White”, then the current node*

*\*\* is available and can be reached.*

*\*\* What we need to do is just to push the node into stack,*

*\*\* and after that get the top element and continue the next round.*

*\*\**

*\*\* Otherwise, if the color of next node equals to “Black” or “Gray”,*

*\*\* then we need to skip to the next node.*

*\*/*

if ( node->next->color == “White” ) {

/\*

\*\* Update related information of next node.

\*/

node->next->priority = node;

node->next->color = “Gray”;

node->next->formertime = time;

node->next->lattertime = 0;

time ++;

temp\_stack.push\_back(node->next);

node = temp\_stack.top();

} else {

/\*

\*\* If the color of node equals to “Black” or “Gray”, skip to

\*\* check the next node.

\*/

node = node->next;

} */\* End if - else. \*/*

} */\* End while. \*/*

} */\* End if. \*/*

if ( node->next == NULL ) {

/\*

\*\* Get the top node from stack, and pop it out of stack.

\*/

node = temp\_stack.top();

temp\_stack.pop();

/\*

\*\* Update the node into unordered\_map table.

\*/

table[ node->value ] = node;

*/\**

*\*\* Update all information of node, including color and lattertime*

*\*\* information. The color “Black” means that the node has been visited*

*\*\* successfully and the lattertime equals to time + 1.*

*\*\* The parameter time equals to time + 1.*

*\*/*

node->color = “Black”;

node->lattertime = time + 1;

time ++;

} */\* End if. \*/*

} */\* End for. \*/*

} */\* End DFS. \*/*

*Pseudo Code*

*Graph ( G, V ):*

* *Vertex Collection V is the vertex collection. All vertexes in the Graph has vertex*

*and edge.*

*Each vertex in G.V has d, f, color, and value:*

*- node.d means the first time that the node has been visited.*

*- node.f means the second time that the node has been visited.*

*- node.color means the color of the node.*

*- node.priority means the priority node of the current node.*

*/\**

*\*\* DFS Algorithm which is based on recursively calling itself.*

*\*/*

void DFS (Graph \*G) {

*/\**

*\*\* The first stage is to initialize all related information of all vertexes.*

*\*/*

for ( each vertex in G.V ) {

vertex.color = “White”;

vertex.priority = NIL;

vertex.d = time;

vertex.f = 0;

}

int time = 0;

for ( each vertex in G.V ) {

DFS\_Visited ( G, G->V, time );

}

}

*/\**

*\*\* DFS\_Visited procedure is used to visit all neighbor vertexes of certain vertex, and find*

*\*\* all neighbor nodes with color of White sequentially. After that visit DFS\_Visited function*

*\*\* from Top to Bottom. Of course, after the bottom has been finished, then return to the*

*\*\* upper level. When the node has been done, then just re-setup the color of node and*

*\*\* make it “Black”, and setup the end time of node.*

*\*/*

void DFS\_Visited ( Graph \* G, V vertex, int time ) {

time = time + 1;

for ( each vex in G->vertex ) {

if ( vex.color == “White” ) {

*/\**

*\*\* Update the color, priority, and the first time the vertex has been visited.*

*\*/*

vex.color = “Gray”;

vex.priority = vertex;

vex.d = time;

*/\**

*\*\* Calling function DFS\_Visited from Top to End.*

*\*/*

DFS\_Visited( G, vex, time );

}

}

vex.color = “Black”;

time = time + 1;

vex.f = time;

}