**DFS (Depth-first search) אלגוריתם חיפוש לעומק**

**אלגוריתם Depth First Search (DFS)** מקבל כקלט G = (V, E) (מכוון או לא מכוון(

האלגוריתם סורק את הגרף ומוציא מידע גולמי המשמש אלגוריתמים אחרים הפועלים על הגרף, למשל, מציאת רכיבי קשירות בגרף, מציאת מסלול אוילרי ומציאת מעגלים בגרף.

סיבוכיות של DFS היא O(|E| + |V|).

האלגוריתם מתחיל מצומת שרירותי בגרף וקורא לDFS באופן רקורסיבי לכל אחד מבניו.

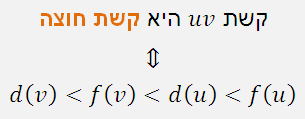
האלגוריתם מסמן לכל צומת בגרף את זמן גילויו (first timestamp) וזמן הנסיגה ממנו (second timestamp):

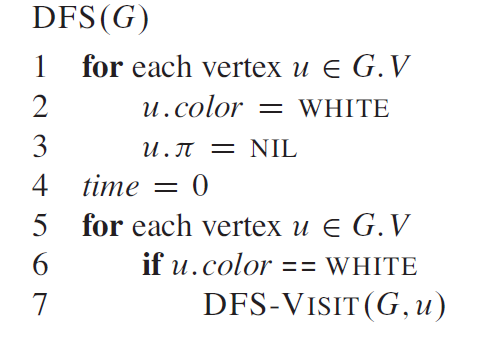
* זמן גילוי - 𝑑(𝑣) הזמן בו 𝑣 מתגלה לראשונה ע"י DFS.
* זמן נסיגה 𝑓(𝑣) - הזמן בו DFS מסיים לעבור על כל שכני 𝑣 .

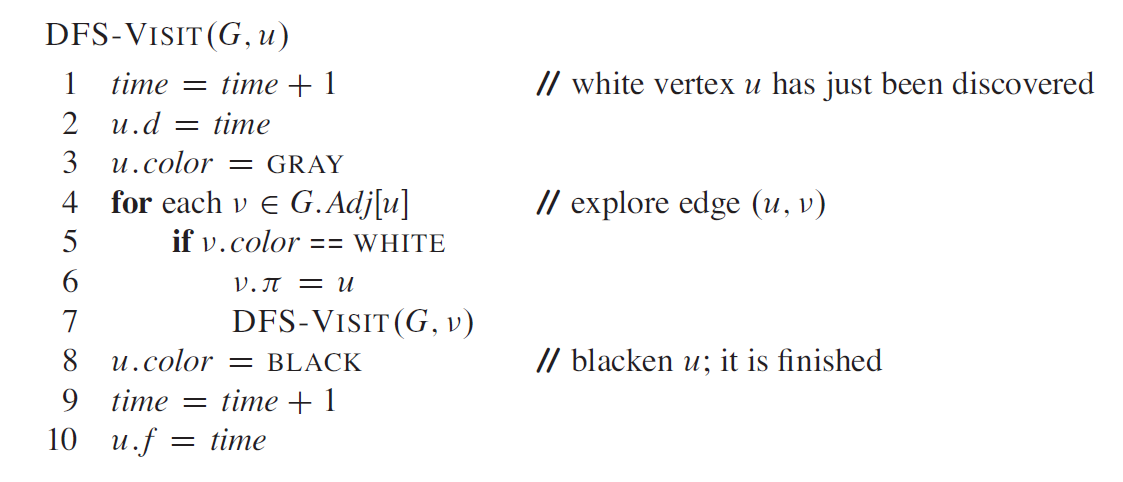
**יער ה-DFS** הוא קבוצת העצים המכוונים המתארת את אופן גילוי הצמתים. כלומר, יחסי הורות המאפיינים איזה צומת גילה איזה צומת במהלך ריצת ה-DFS.

**סיוג הקשתות ביער DFS:**

* **קשת עץ** היא קשת (u,v) כך ש-DFS גילה את vדרך צומת u, היא קשת השייכת ליער ה-DFS.
* **קשת קדמית** היא קשת (u,v) שאינה קשת עץ, המחברת בין u לצומת v בתת העץ שלו ביער DFS, היא קשת המחברת בין צומת לצאצא שלט בעץ ה-DFS.
* **קשת אחורית** היא קשת (u,v) המחברת צומת u לאב קדמון שלו v בעץ ה-DFS.
* **קשת חוצה** היא כל קשת אחרת:קשת שמחברת בין צמתים באותו עץ DFS (כל עוד אין יחסי הורות ביניהם) או צמתים בעצי DFS שונים.





1/

1/ 2/

1/ 2/ 3/

1/ 2/ 3/ 4/

1/ 2/ 3/ 4/

5/

1/ 2/ 3/ 4/

6/ 5/

1/ 2/ 3/ 4/

6/7 5/

8/

1/ 2/ 3/ 4/

6/7 5/

8/9

1/ 2/ 3/ 4/

6/7 5/ 10/

8/9

1/ 2/ 3/ 4/

6/7 5/ 10/11

8/9

1/ 2/ 3/ 4/

6/7 5/12 10/11

8/9

1/ 2/ 3/ 4/13

6/7 5/12 10/11

8/9

1/ 2/ 3/14 4/13

6/7 5/12 10/11

8/9

1/ 2/15 3/14 4/13

6/7 5/12 10/11

8/9

16/

1/ 2/15 3/14 4/13

6/7 5/12 10/11

8/9

16/17

1/18 2/15 3/14 4/13

6/7 5/12 10/11

8/9

16/17

1/18 2/15 3/14 4/13

6/7 5/12 10/11

8/9

0 1 2 3 4 5 6 7 8

timeFirst: [1, 2, 3, 4, 6, 8, 5, 10, 16]

0 1 2 3 4 5 6 7 8

timeLast: [18, 15, 14, 13, 7, 9, 12, 11, 17]

0 1 2 3 4 5 6 7 8

pred: [-1, 0, 1, 2, 6, 6, 3, 6, 0]

**DFS JAVA code**

**import** java.util.ArrayList;

**import** java.util.Arrays;

**public** **class** DFSRecurs {

**private** **final** **int** WHITE=1, GRAY=2, BLACK=3, NIL = -1;// colors

**private** **int** first[], last[], pred[], color[];

**private** **int** time, size;

**private** ArrayList<Integer>[] graph;

**private** **int** startCycle, endCycle;

**private** **boolean** hasCycle;

**public** DFSRecurs (ArrayList<Integer>[] g){

size = g.length;

pred = **new** **int**[size];

color = **new** **int**[size];

first = **new** **int**[size];

last = **new** **int**[size];

**this**.graph = **new** ArrayList[g.length];

**for** (**int** i = 0; i < g.length; i++)

graph[i] = **new** ArrayList<Integer>(g[i]);

}

**public** **void** dfs(**int** s){

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

color[i] = WHITE;

pred[i] = NIL;

first[i] = 0;

last[i] = 0;

}

hasCycle = **false**;

startCycle = NIL;

endCycle = NIL;

time = 0;

visit(s);

}

**private** **void** visit(**int** u){

color[u] = GRAY;

first[u] = ++time;

**for**(**int** v : graph[u]){

**if** (!hasCycle && color[v] == GRAY && pred[u] != v){

hasCycle = **true**;

startCycle = u;

endCycle = v;

}

**if** (color[v] == WHITE){

color[v] = GRAY;

pred[v] = u;

visit(v);

}

}

color[u] = BLACK;

last[u] = ++time;

}

**public** String dfsPath(**int** u, **int** v){

String ans = **null**;

dfs(u);

**if** (color[v] != WHITE){

ans = **new** String() + v;

**while** (v!=u){

v = pred[v];

ans = v + "->" + ans;

}

}

**return** ans;

}

**public** String dfsCycle(){

String ans = **null**;

**boolean** firstCycle = **false**;

**for** (**int** i=0; !firstCycle && i<size; i++){

**int** s = graph[i].get(0);

dfs(s);

**if** (hasCycle){

firstCycle = **true**;

dfs(startCycle);

ans = dfsPath(startCycle, endCycle);

ans = ans + "->" + endCycle;

}

}

**return** ans;

}

**public** **void** print(){

System.out.println("colors: "+Arrays.toString(color));

System.out.println("pred: "+Arrays.toString(pred));

System.out.println("first: "+Arrays.toString(first));

System.out.println("last: "+Arrays.toString(last));

System.out.println("cycle: "+hasCycle);

}

}

**TestDFS class**

**public** **class** TestDFS {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

//DFS\_Loop2.dfs(GraphArrayListInit.initGraph3(), 0);

System.out.println();

DFSRecurs obj = **new** DFSRecurs(GraphArrayListInit.initGraph());

obj.dfs(0);

obj.print();

System.out.println(obj.dfsPath(0, 2));

System.out.println(obj.dfsCycle());

}

}

**Result :**

colors: [3, 3, 3, 3, 3, 3, 3, 3]

pred: [-1, 0, 1, 2, 3, 4, 5, 6]

first: [1, 2, 3, 4, 5, 6, 7, 8]

last: [16, 15, 14, 13, 12, 11, 10, 9]

cycle: true

0->1->2

4->3->5->4

**GraphArrayListInit** **class**

**import** java.util.ArrayList;

**public** **class** GraphArrayListInit {

**public** **static** ArrayList<Integer>[] initGraph(){//connected graph with circle

**int** size = 8;

ArrayList<Integer>[] graph = **new** ArrayList[size];

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

graph[i] = **new** ArrayList<Integer>();

graph[0].add(1);

graph[1].add(0);

graph[1].add(2);

graph[2].add(1);

graph[2].add(3);

graph[3].add(2);

graph[3].add(4);

graph[3].add(5);

graph[4].add(3);

graph[4].add(5);

graph[4].add(6);

graph[5].add(3);

graph[5].add(4);

graph[5].add(6);

graph[5].add(7);

graph[6].add(4);

graph[6].add(5);

graph[6].add(7);

graph[7].add(5);

graph[7].add(6);

**return** graph;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** ArrayList<Integer>[] initGraph1(){//with cycle

**int** size = 4;

ArrayList<Integer>[] g = **new** ArrayList[size];

**for** (**int** i = 0; i < g.length; i++)

g[i] = **new** ArrayList<Integer>();

g[0].add(1); g[0].add(2); g[0].add(3);

g[1].add(0); g[1].add(2);

g[2].add(0); g[2].add(1);

g[3].add(0);

**return** g;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** ArrayList<Integer>[] initGraph2(){

**int** size = 9;

ArrayList<Integer>[] g = **new** ArrayList[size];

**for** (**int** i = 0; i < g.length; i++)

g[i] = **new** ArrayList<Integer>();

g[0].add(1); g[0].add(5); g[0].add(8);

g[1].add(0); g[1].add(2); g[1].add(4);

g[2].add(1); g[2].add(3); g[2].add(4);

g[3].add(2); g[3].add(6); g[3].add(7);

g[4].add(1); g[4].add(2); g[4].add(6);

g[5].add(0); g[5].add(6);

g[6].add(3); g[6].add(4); g[6].add(5);

g[7].add(3);

g[8].add(0);

**return** g;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** ArrayList<Integer>[] initGraph3(){//with cycle

**int** size = 6;

ArrayList<Integer>[] g = **new** ArrayList[size];

**for** (**int** i = 0; i < g.length; i++)

g[i] = **new** ArrayList<Integer>();

g[0].add(2); g[0].add(1); g[0].add(5);

g[1].add(0); g[1].add(2);

g[2].add(0); g[2].add(1); g[2].add(3); g[2].add(4);

g[3].add(5); g[3].add(4); g[3].add(2);

g[4].add(3); g[4].add(2);

g[5].add(3); g[5].add(0);

**return** g;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** ArrayList<Integer>[] initGraph4(){//with cycle

**int** size = 6;

ArrayList<Integer>[] g = **new** ArrayList[size];

**for** (**int** i = 0; i < g.length; i++)

g[i] = **new** ArrayList<Integer>();

g[0].add(1); g[0].add(3);

g[1].add(0); g[1].add(4);

g[2].add(4); g[2].add(5);

g[3].add(0);

g[4].add(1); g[4].add(2); g[4].add(5);

g[5].add(2); g[5].add(4);

**return** g;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **static** ArrayList<Integer>[] initGraph5(){//without cycle

**int** size = 6;

ArrayList<Integer>[] g = **new** ArrayList[size];

**for** (**int** i = 0; i < g.length; i++)

g[i] = **new** ArrayList<Integer>();

g[0].add(1); g[0].add(3);

g[1].add(0); g[1].add(4);

g[2].add(4); g[2].add(5);

g[3].add(0);

g[4].add(1); g[4].add(2);

g[5].add(2);

**return** g;

}

}