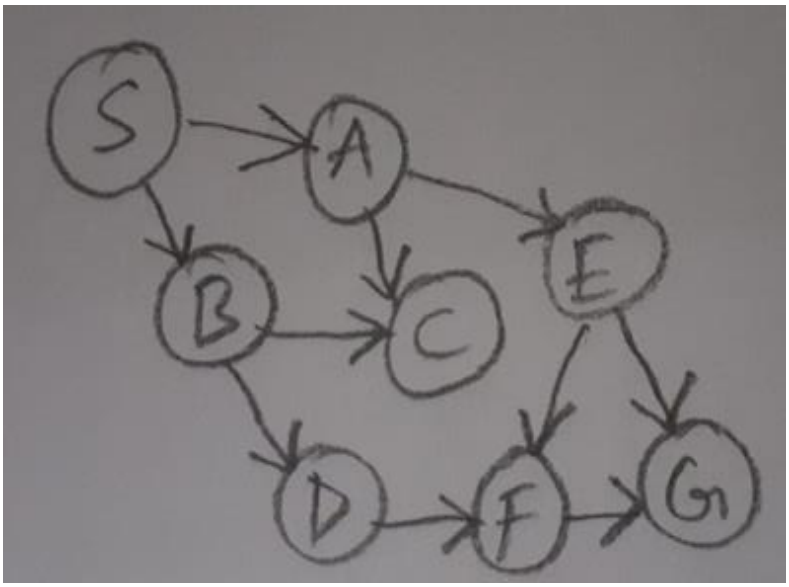

S
A
B
C
E
D
F
G

DFS Algorithm

Initialization process:

0)construct node object for each node in the picture

node.name=S; (only enter the name for each node in the picture)

node.depth=0;

node.parent=null;

node.searched=false;

[node.ID=0000;] (optional)

Store the “nodes” in an “array”

(construct a node and add it to an array, do this for each node in the picture)

nodeList=[nodeList,node] for example

1)define 2 sets called “open” and “closed”

$$open \leftarrow \{\}$$

$$closed \leftarrow \{\}$$

2)update the depth,parent and search fields of S-node:

S.depth=0;

S.parent=null;

S.searched=true;

3)add S to the open set:

$$open \leftarrow Append(S, open)$$

DFS:

While(true)

N=Head(open) pick the 1st node in open [node with the zeroth index in open, N=open[0] for example]

If GoalTest(N)==True return;

Else

1)New=GenChildren(N); Find the nodes to whom there is an arrow pointed from N

2)New←FilterOutSearched(New); Filter out the “searched” nodes from New

3)New(i).parent=[pointer to N], i=1…length(New); update the parent field of each node in New as N [or pointer to N, or ID of N]

4)Delete(N,open); delete N from open

5)closed←Append(N,closed); add N to closed

6)New(i).depth=(N.depth)+1, i=1…length(New); update the depth field of each node in New as depth of N plus 1

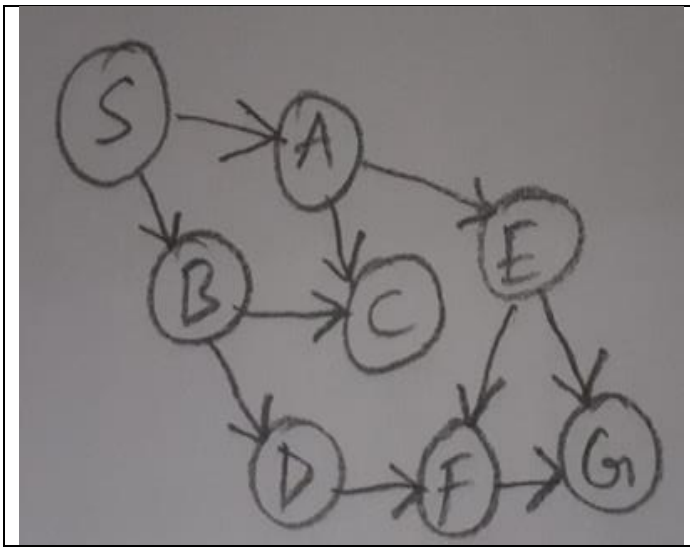
7)New(i).searched=true, i=1…length(New); update the searched field of each node in New as true

8)open← Append(New, open); add N to closed

if open=={} break;

else continue;

EXAMPLE – 1: DFS directed graph

	Node OBJECT:
	node.name=S; (only enter the name for each node in the picture)
	node.depth=0;
	node.parent=null;
	node.searched=false;
	
	Children Generation Order: from Left to right
	For the children of S:
	First B then A is generated and added.

Step 1-2

Open={"S"},Closed={}	Open={"A",B},Closed={S}
N=Head(open)	N=Head(open)
check if N is GOAL	check if N is GOAL
1)New=GenChildren(N);	1)New=GenChildren(N);
2)New←FilterOutSearched(New);	2)New←FilterOutSearched(New);
3)New(i).parent=[pointer to N], i=1…length(New);	3)New(i).parent=[pointer to N], i=1…length(New);
4)Delete(N,open);	4)Delete(N,open);
5)closed←Append(N,closed);	5)closed←Append(N,closed);
6)New(i).depth=(N.depth)+1, i=1…length(New);	6)New(i).depth=(N.depth)+1, i=1…length(New);
7)New(i).searched=true, i=1…length(New);	7)New(i).searched=true, i=1…length(New);
8)open← Append(New, open);	8)open← Append(New, open);
Check if open is empty	Check if open is empty
Open={A,B},Closed={S}	Open={E,C,B},Closed={A,S}

Step 3-4

Open={"E",C,B},Closed={A,S}	Open={"G",F,C,B},Closed={E,A,S}
N=Head(open)	N=Head(open)
check if N is GOAL	check if N is GOAL→YES!
1)New=GenChildren(N);	
2)New←FilterOutSearched(New);	What is the PATH?
3)New(i).parent=[pointer to N], i=1…length(New);	G
4)Delete(N,open);	E [G.parent]
5)closed←Append(N,closed);	A [E.parent]
6)New(i).depth=(N.depth)+1, i=1…length(New);	S [A.parent]
7)New(i).searched=true, i=1…length(New);	$S \rightarrow A \rightarrow E \rightarrow G$
8)open← Append(New, open);	
Check if open is empty	
Open={G,F,C,B},Closed={E,A,S}	

BFS Algorithm

Initialization process:

0)construct node object for each node in the picture

node.name=S; (only enter the name for each node in the picture)

node.depth=0;

node.parent=null;

node.searched=false;

[node.ID=0000;] (optional)

Store the “nodes” in an “array”

(construct a node and add it to an array, do this for each node in the picture)

nodeList=[nodeList,node] for example

1)define 2 sets called “open” and “closed”

$$open \leftarrow \{\}$$

$$closed \leftarrow \{\}$$

2)update the depth,parent and search fields of S-node:

S.depth=0;

S.parent=null;

S.searched=true;

3)add S to the open set:

$$open \leftarrow Append(S, open)$$

BFS:

While(true)

N=Head(open) pick the 1st node in open [node with the zeroth index in open, N=open[0] for example]

If GoalTest(N)==True return;

Else

1)New=GenChildren(N); Find the nodes to whom there is an arrow pointed from N

2)New←FilterOutSearched(New); Filter out the “searched” nodes from New

3)New(i).parent=[pointer to N], i=1…length(New); update the parent field of each node in New as N [or pointer to N, or ID of N]

4)Delete(N,open); delete N from open

5)closed←Append(N,closed); add N to closed

6)New(i).depth=(N.depth)+1, i=1…length(New); update the depth field of each node in New as depth of N plus 1

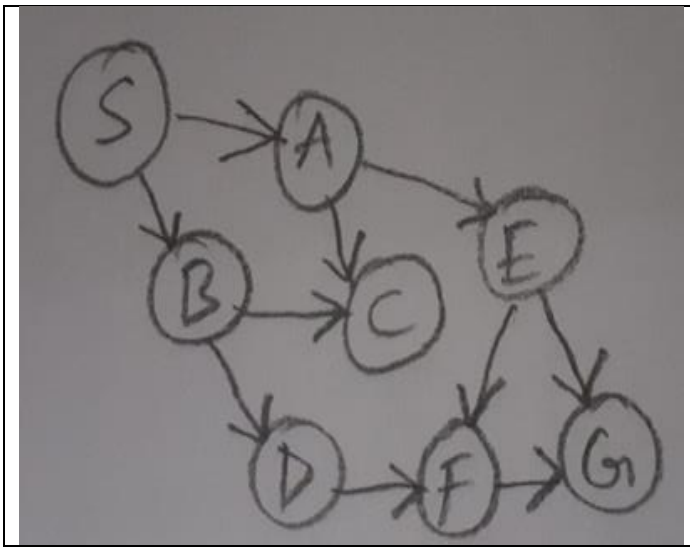
7)New(i).searched=true, i=1…length(New); update the searched field of each node in New as true

8)open← Append(open ,New); add N to closed [THIS IS THE ONLY DIFFERENCE!!!]

if open=={} break;

else continue;

EXAMPLE – 2: BFS directed graph

	Node OBJECT:
	node.name=S; (only enter the name for each node in the picture)
	node.depth=0;
	node.parent=null;
	node.searched=false;
	
	Children Generation Order: from Left to right
	For the children of S:
	First B then A is generated and added.

Step 1-2

Open={"S"},Closed={}	Open={"A",B},Closed={S}
N=Head(open)	N=Head(open)
check if N is GOAL	check if N is GOAL
1)New=GenChildren(N);	1)New=GenChildren(N);
2)New←FilterOutSearched(New);	2)New←FilterOutSearched(New);
3)New(i).parent=[pointer to N], i=1…length(New);	3)New(i).parent=[pointer to N], i=1…length(New);
4)Delete(N,open);	4)Delete(N,open);
5)closed←Append(N,closed);	5)closed←Append(N,closed);
6)New(i).depth=(N.depth)+1, i=1…length(New);	6)New(i).depth=(N.depth)+1, i=1…length(New);
7)New(i).searched=true, i=1…length(New);	7)New(i).searched=true, i=1…length(New);
8)open← Append(open ,New);	8)open← Append(open ,New);
Check if open is empty	Check if open is empty
Open={A,B},Closed={S}	Open={B,E,C },Closed={A,S}

Step 3-4

Open={"B",E,C },Closed={A,S}	Open={"E",C,D},Closed={B,A,S}
N=Head(open)	N=Head(open)
check if N is GOAL	check if N is GOAL
1)New=GenChildren(N);	1)New=GenChildren(N);
2)New←FilterOutSearched(New);	2)New←FilterOutSearched(New);
3)New(i).parent=[pointer to N], i=1…length(New);	3)New(i).parent=[pointer to N], i=1…length(New);
4)Delete(N,open);	4)Delete(N,open);
5)closed←Append(N,closed);	5)closed←Append(N,closed);
6)New(i).depth=(N.depth)+1, i=1…length(New);	6)New(i).depth=(N.depth)+1, i=1…length(New);
7)New(i).searched=true, i=1…length(New);	7)New(i).searched=true, i=1…length(New);
8)open← Append(open ,New);	8)open← Append(open ,New);
Check if open is empty	Check if open is empty
Open={E,C,D},Closed={B,A,S}	Open={G,F,C,D},Closed={E,B,A,S}

Step 5

Open={"G",F,C,D},Closed={E,B,A,S}
N=Head(open)
check if N is GOAL→YES!
What is the PATH?
G
E [G.parent]
A [E.parent]
S [A.parent]
$S \rightarrow A \rightarrow E \rightarrow G$

Important points

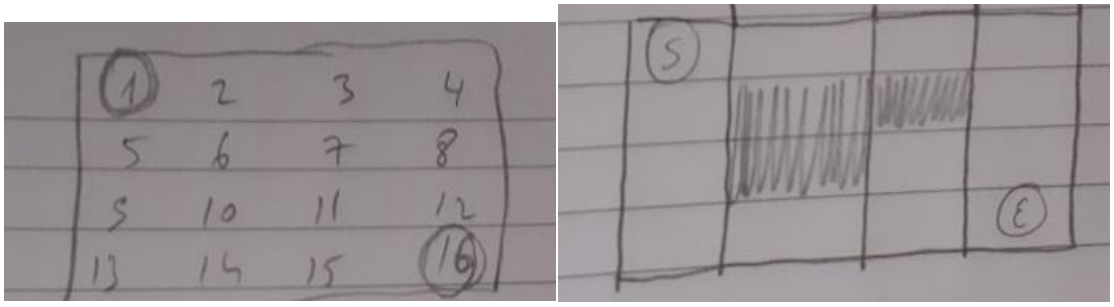
1)if the state-space is finite DFS is complete [it finds the solution]

2)BFS finds the optimal path, its space[storage] complexity is greater than DFS

DFS, STACK, LIFO [open set]

BFS, QUEUE, FIFO [open set]

EXAMPLE 3: DFS, maze path planning



$$nodeID = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \rightarrow \begin{bmatrix} \boxed{S} & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & \boxed{E} \end{bmatrix} \rightarrow Type = \begin{bmatrix} \boxed{5} & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \boxed{3} \end{bmatrix}$$

Node structure
Node.name=1
Node.depth=0
Node.parent=null
Node.searched=false
Node.location=[1,1] i.e. [r=1,c=1]
Node.type=[5] , [3] , [1] , [0] [5=start,3=end,1=wall,0=space]

DFS process

step	open	closed	Chosen one	Gen. Children	FilterOut	Delete N from open	Add New to open	Add N to closed
1	{1}	{}	1	{2,5}	{2,5}	{}	{2,5}	{1}
2	{2,5}	{1}	2	{3,6,1}	{3}	{5}	{3,5}	{2,1}
3	{3,5}	{2,1}	3	{4,7,2}	{4}	{5}	{4,5}	{3,2,1}
4	{4,5}	{3,2,1}	4	{8,3}	{8}	{5}	{8,5}	{4,3,2,1}
5	{8,5}	{4,3,2,1}	8	{12,7}	{12}	{5}	{12,5}	{8,4,3,2,1}
6	{12,5}	{8,4,3,2,1}	12	{16,11}	{16,11}	{5}	{16,11,5}	{12,8,4,3,2,1}
7	{16,11,5}	{12,8,4,3,2,1}	16 → SOLVED!	{}	{}	{}	{}	{}

BFS process

step	open	closed	Chosen one	Gen. Children	Filter Out	Delete N from open	Add New to open	Add N to closed
1	{1}	{}	1	{2,5}	{2,5}	{}	{2,5}	{1}
2	{2,5}	{1}	2	{3,6,1}	{3}	{5}	{5,3}	{2,1}
3	{5,3}	{2,1}	5	{1,6,9}	{9}	{3}	{3,9}	{5,2,1}
4	{3,9}	{5,2,1}	3	{4,7,2}	{4}	{9}	{9,4}	{3,5,2,1}
5	{9,4}	{3,5,2,1}	9	{5,10,13}	{13}	{4}	{4,13}	{9,3,5,2,1}
6	{4,13}	{9,3,5,2,1}	4	{8,3}	{8}	{13}	{13,8}	{4,9,3,5,2,1}
7	{13,8}	{4,9,3,5,2,1}	13	{9,14}	{14}	{8}	{8,14}	{13,4,9,3,5,2,1}
8	{8,14}	{13,4,9,3,5,2,1}	8	{4,12,7}	{12}	{14}	{14,12}	{8,13,4,9,3,5,2,1}
9	{14,12}	{8,13,4,9,3,5,2,1}	14	{10,15}	{15}	{12}	{12,15}	{14,8,13,4,9,3,5,2,1}
10	{12,15}	{14,8,13,4,9,3,5,2,1}	12	{8,16,11}	{16,11}	{15}	{15,16,11}	{12,14,8,13,4,9,3,5,2,1}
11	{15,16,11}	{12,14,8,13,4,9,3,5,2,1}	15	{11,16}	{}	{16,11}	{16,11}	{15,12,14,8,13,4,9,3,5,2,1}
12	{16,11}	{15,12,14,8,13,4,9,3,5,2,1}	16→ SOLVED!