# CS 401: Computer Algorithm I

## DFS / Topological Sort

Xiaorui Sun

# Homework 1

Homework 1 will be out later today (Due Feb 18)

- Writing homework
- The first 5 questions are for all the students
- Question 6 is for graduate student only (Undergraduate students who work on Question 6 receive at most 3 bonus points)
- Submit your homework to gradescope

Guidelines:

- You can collaborate, but you must write solutions on your own
- Your solution should be clear, well-organized, and concise. Spell out main idea.
- Sanity Check: Make sure you use assumptions of the problem
- You can use AI/online tools, but you cannot copy AI/online solutions.
- Late homework will be penalized at a rate of **10%** of the initial grade per late day (e.g. if your homework receives 80 points and you are late for 4 days, you get 48 points)
- Correctness proofs of ALGORITHMS are NOT REQUIRED

# Graph Traversal

Walk (via edges) from a fixed starting vertex $s$ to all vertices reachable from $s$.

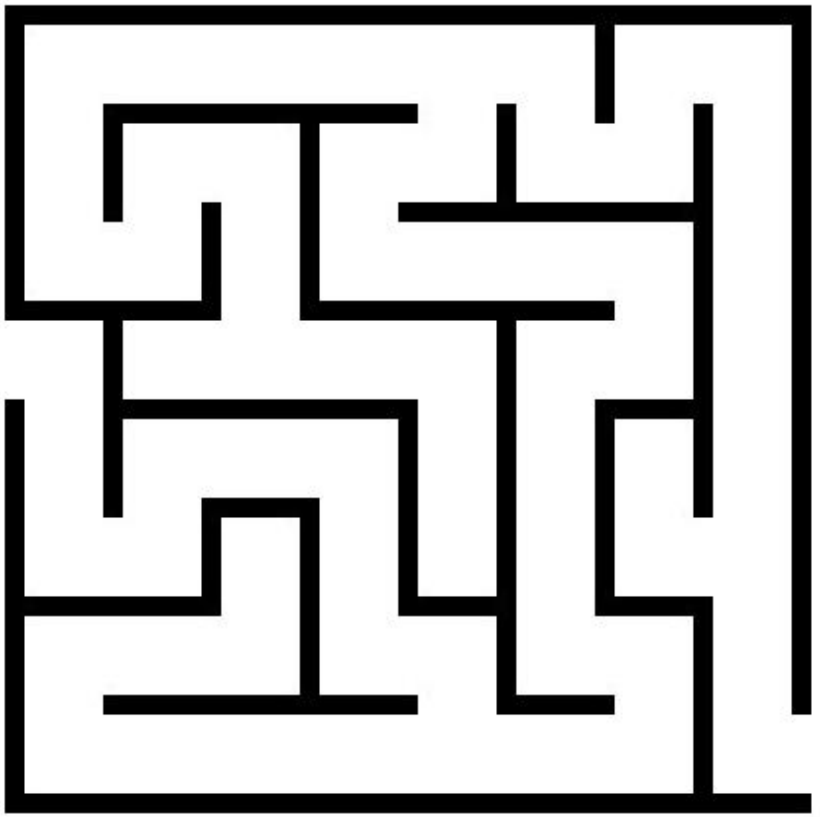
- Breadth First Search (BFS): Order nodes in successive layers based on distance from $s$
- Depth First Search (DFS): More natural approach for exploring a maze;

Applications of BFS:

- Finding shortest path for unit-length graphs
- Finding connected components of a graph
- Testing bipartiteness

# Depth First Search

Follow the first path you find
as far as you can go; back up
to last unexplored edge when
you reach a dead end,
then go as far you can



Naturally implemented using recursive calls or a stack

# DFS(s) – Recursive version

Initialization: mark all vertices undiscovered

DFS($v$)

Mark $v$ discovered

for each edge $\{v, x\}$

if ($x$ is undiscovered)

DFS($x$)

Mark $v$ fully-explored

# DFS(A)

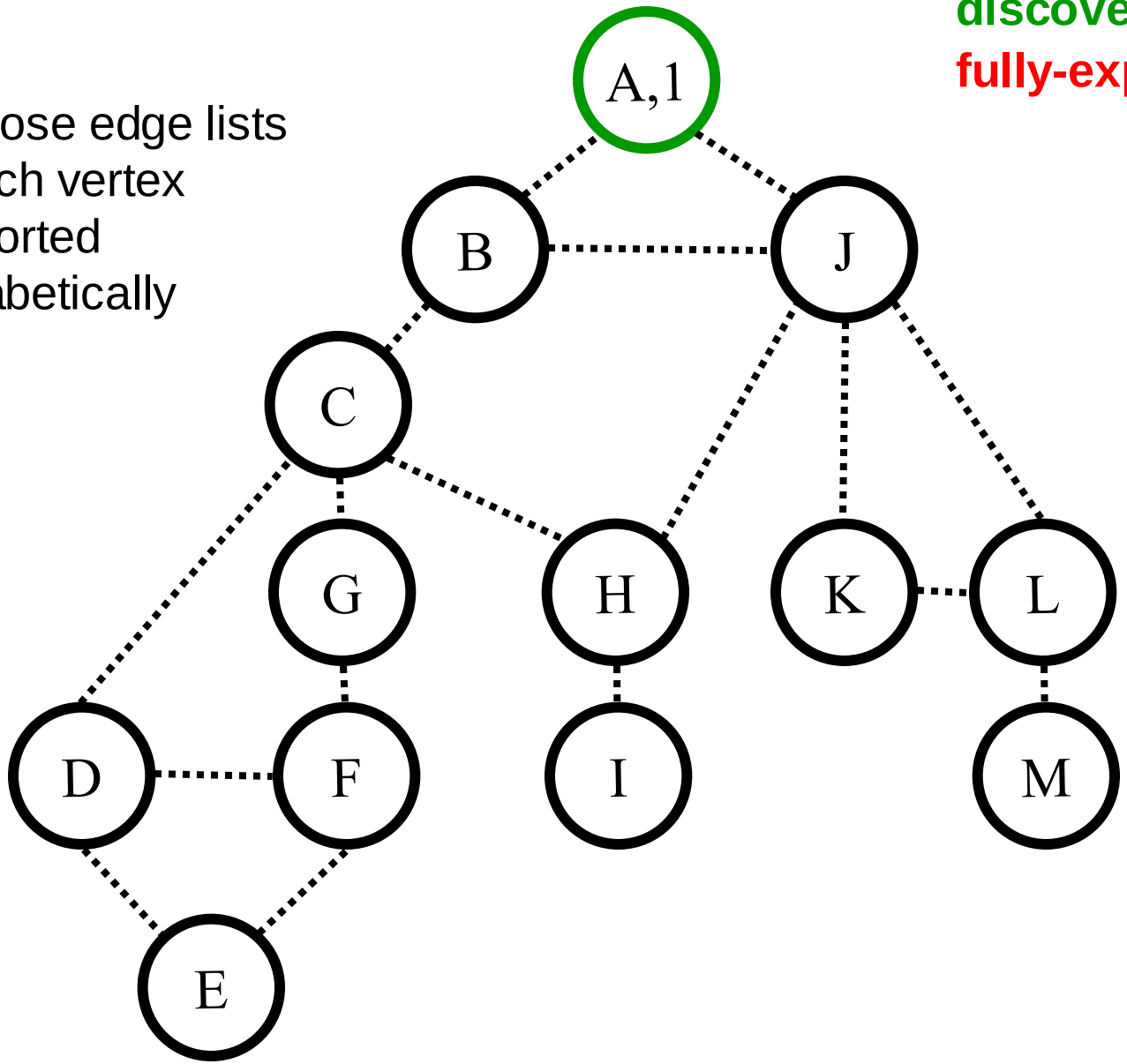
Color code:

**undiscovered**

**discovered**

**fully-explored**

Suppose edge lists at each vertex are sorted alphabetically




| Call Stack (Edge list): |
| --- |
| A (B,J) |

st[] = {1}

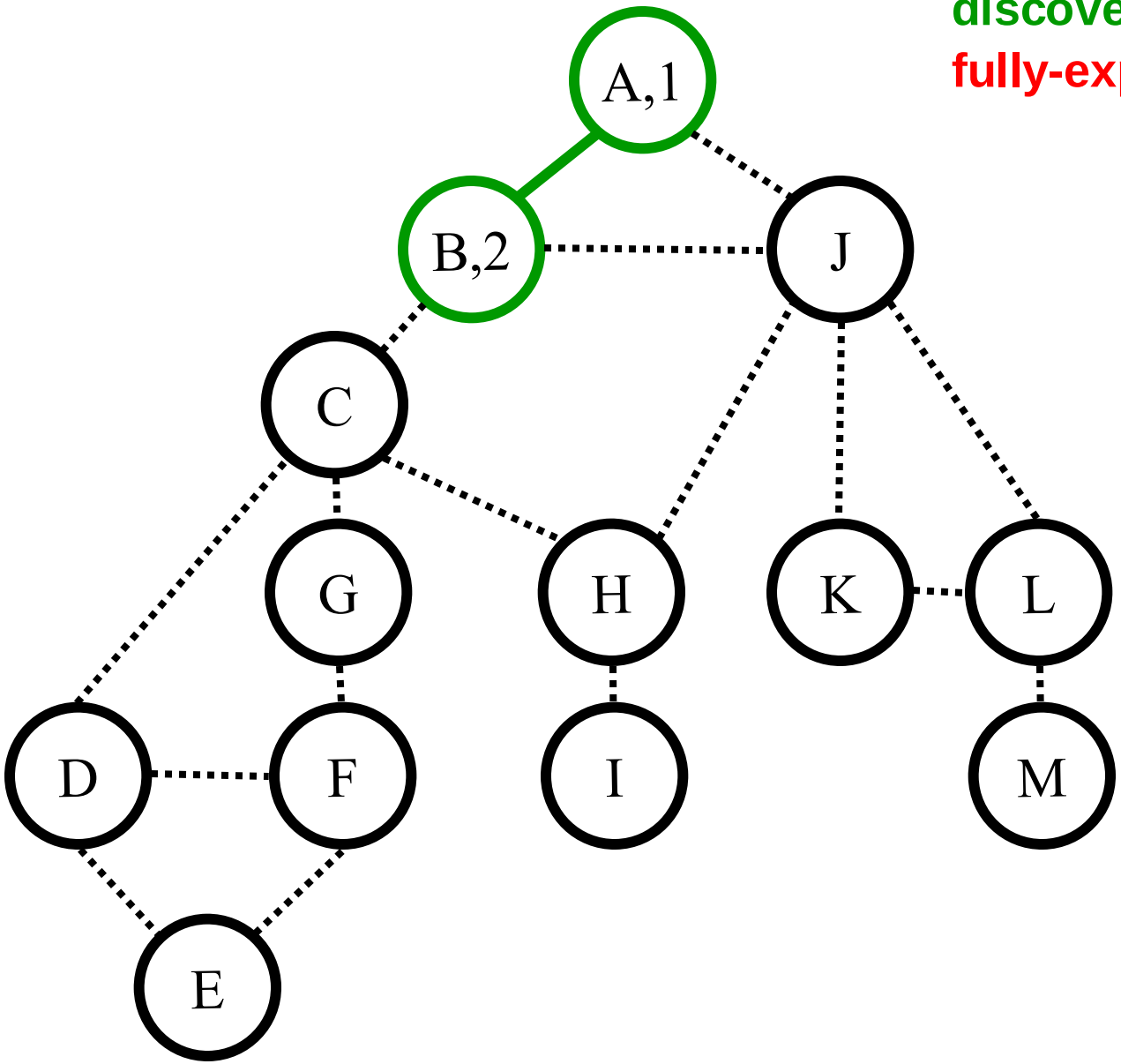
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (A,C,J)

st[] =
{1,2}

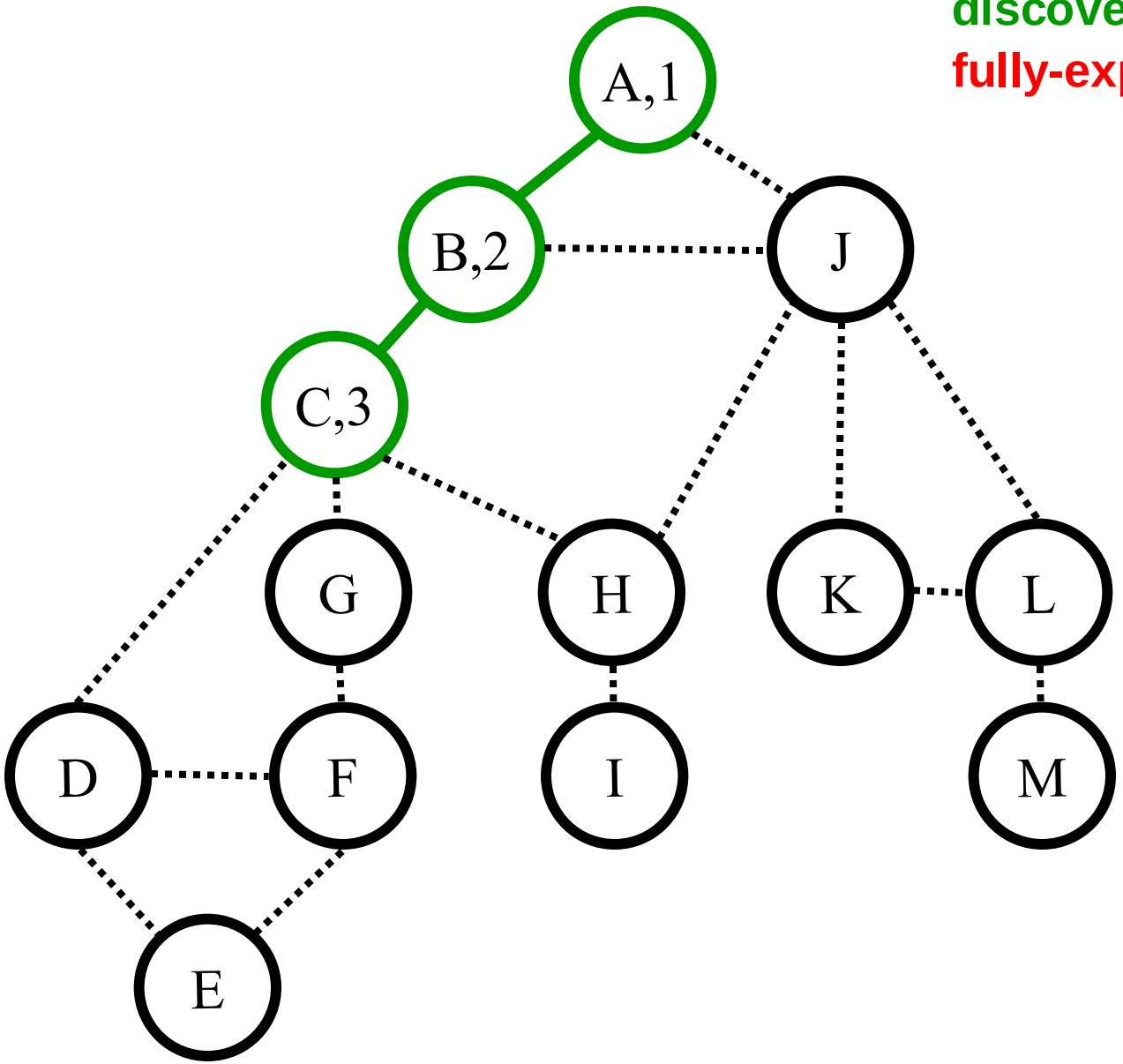
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (B,D,G,H)

st[] =
{1,2,3}

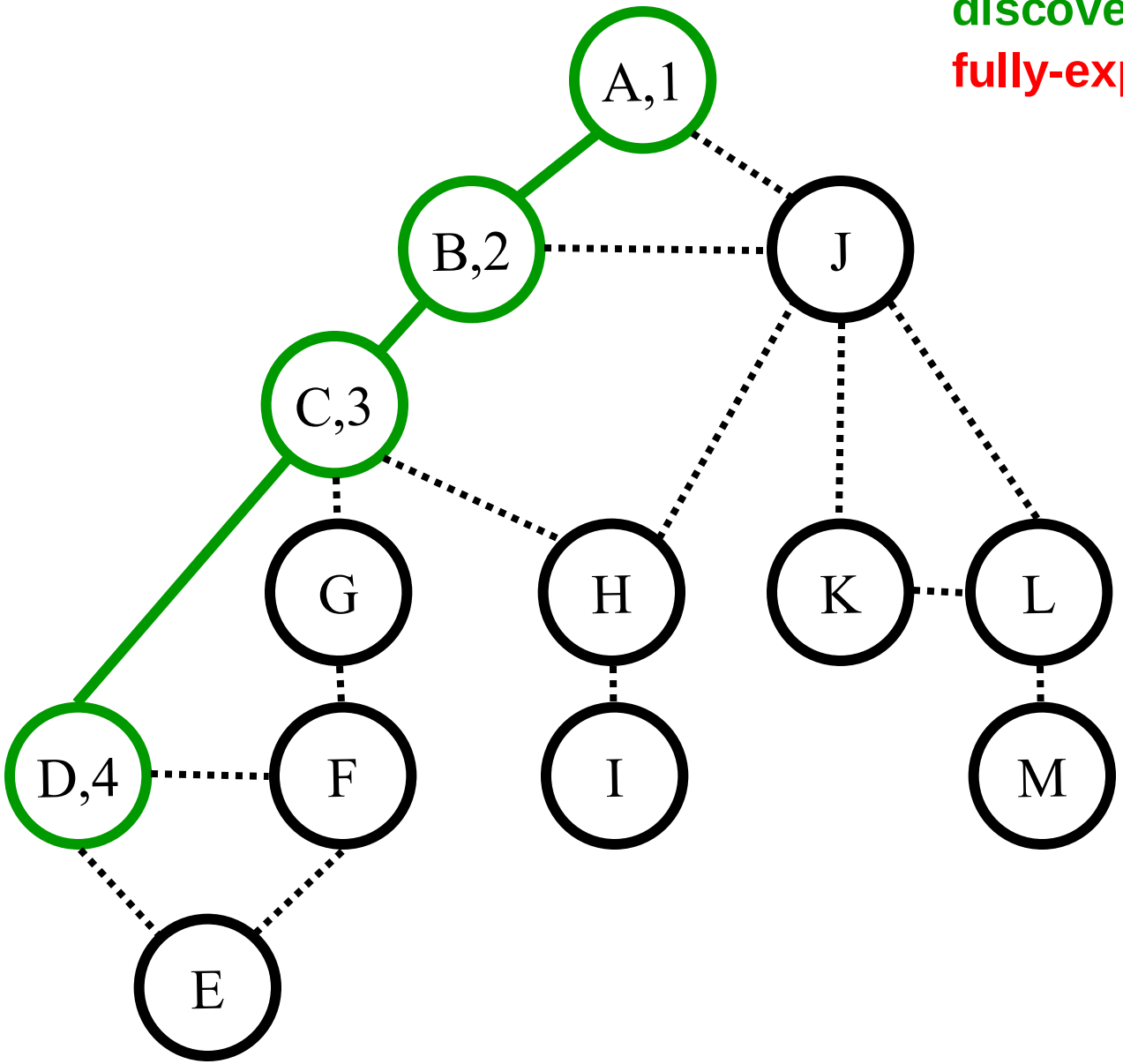
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)
D (C,E,F)

st[] =
{1,2,3,4}

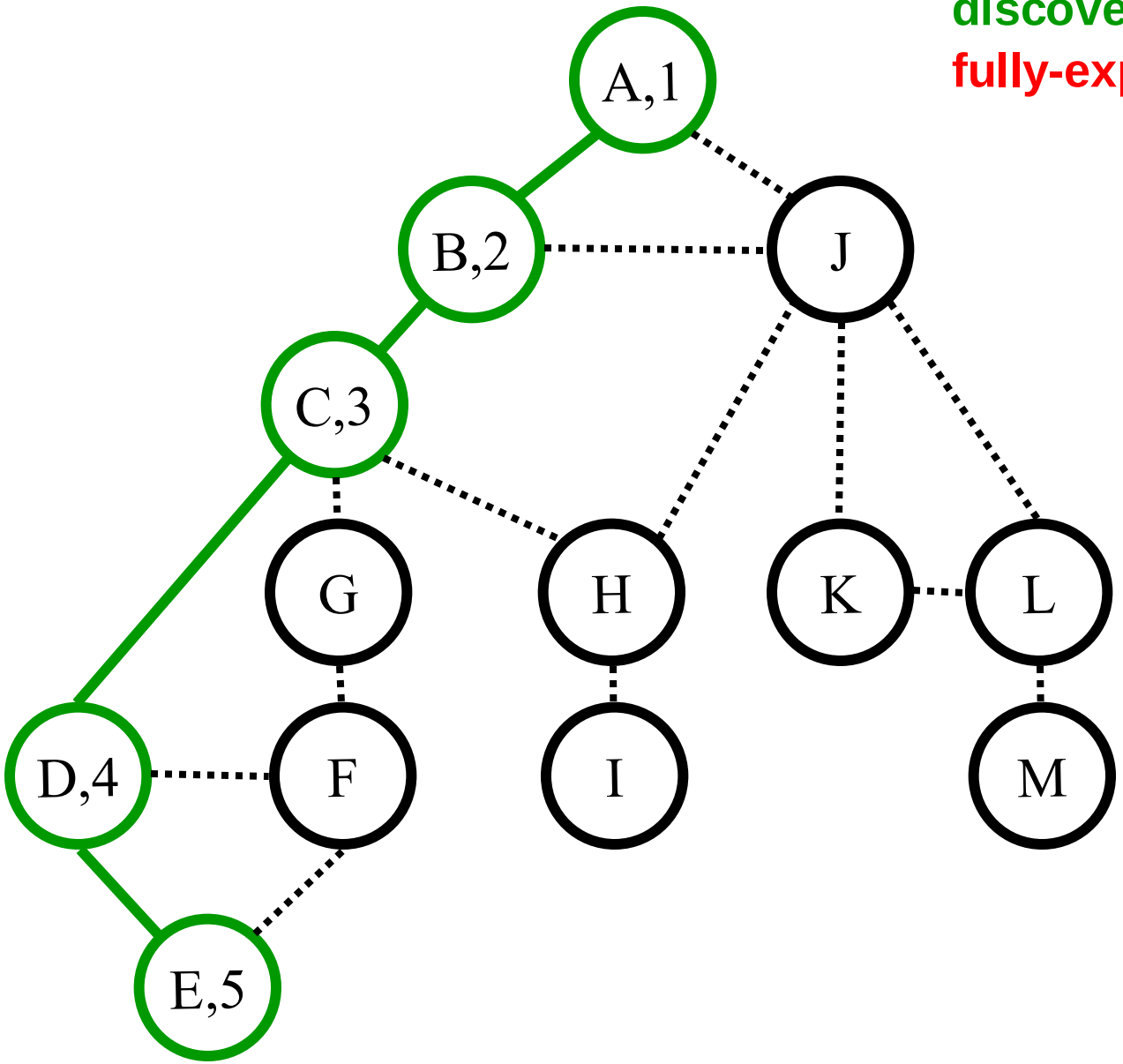
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)
D (~~C~~,~~E~~,F)
E (D,F)

st[] =
{1,2,3,4,5}

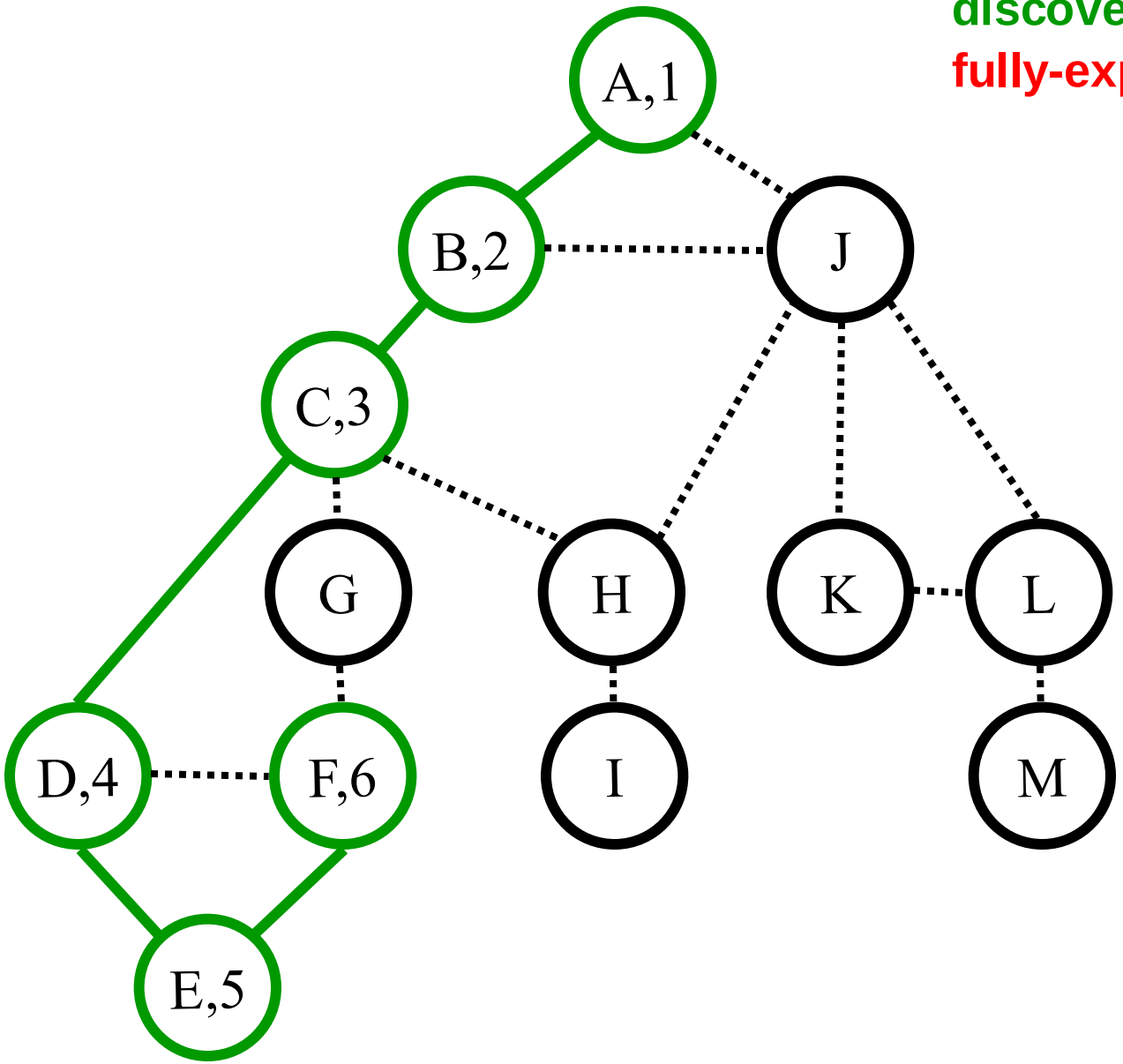
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)
D (~~C~~,~~E~~,F)
E (~~D~~,~~F~~)
F (D,E,G)

st[] =
{1,2,3,4,5,
6}

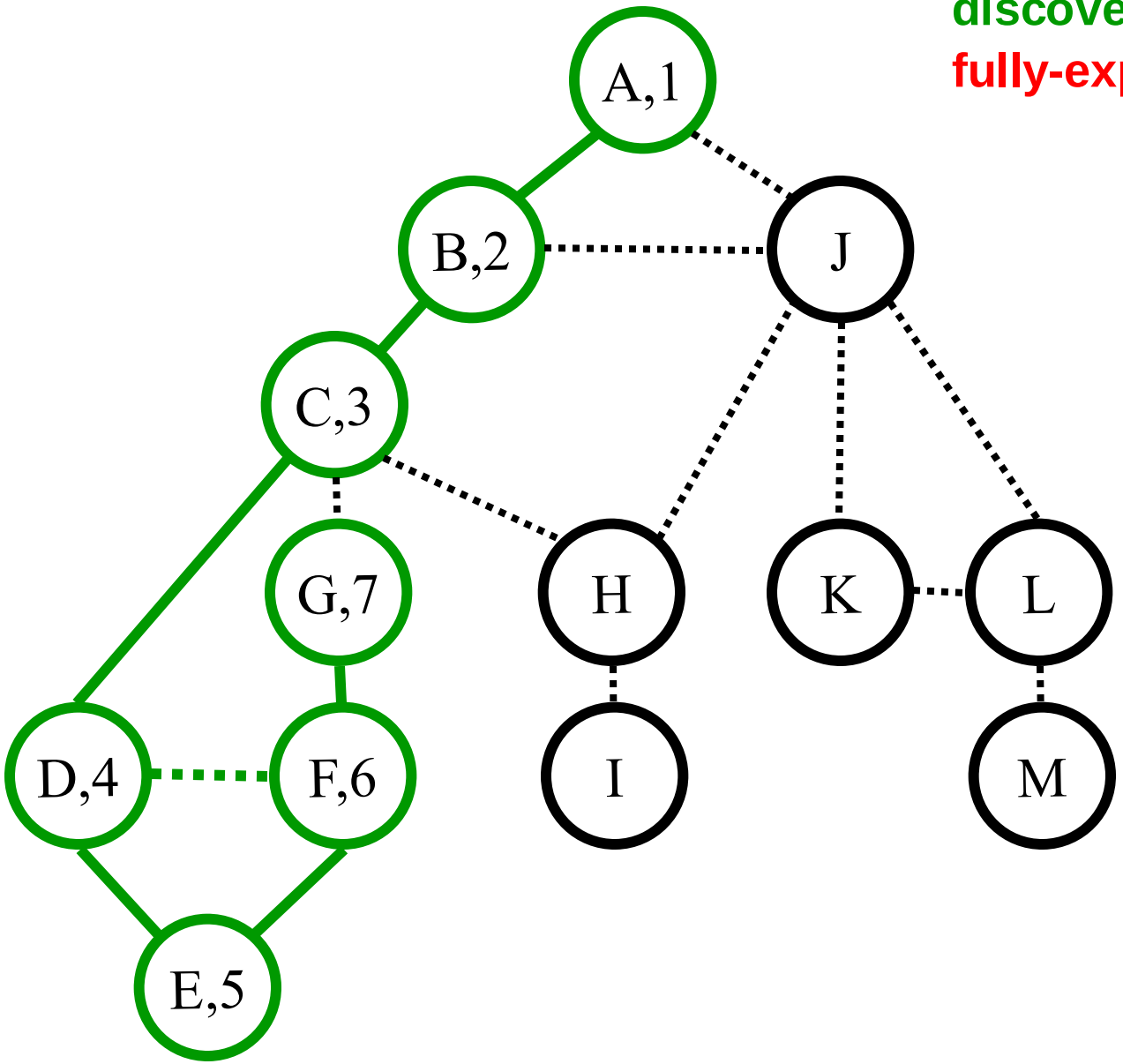
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)
D (~~C~~,~~E~~,F)
E (~~D~~,~~F~~)
F (~~D~~,~~E~~,~~G~~)
G (C,F)

st[] =
{1,2,3,4,5,
6,7}

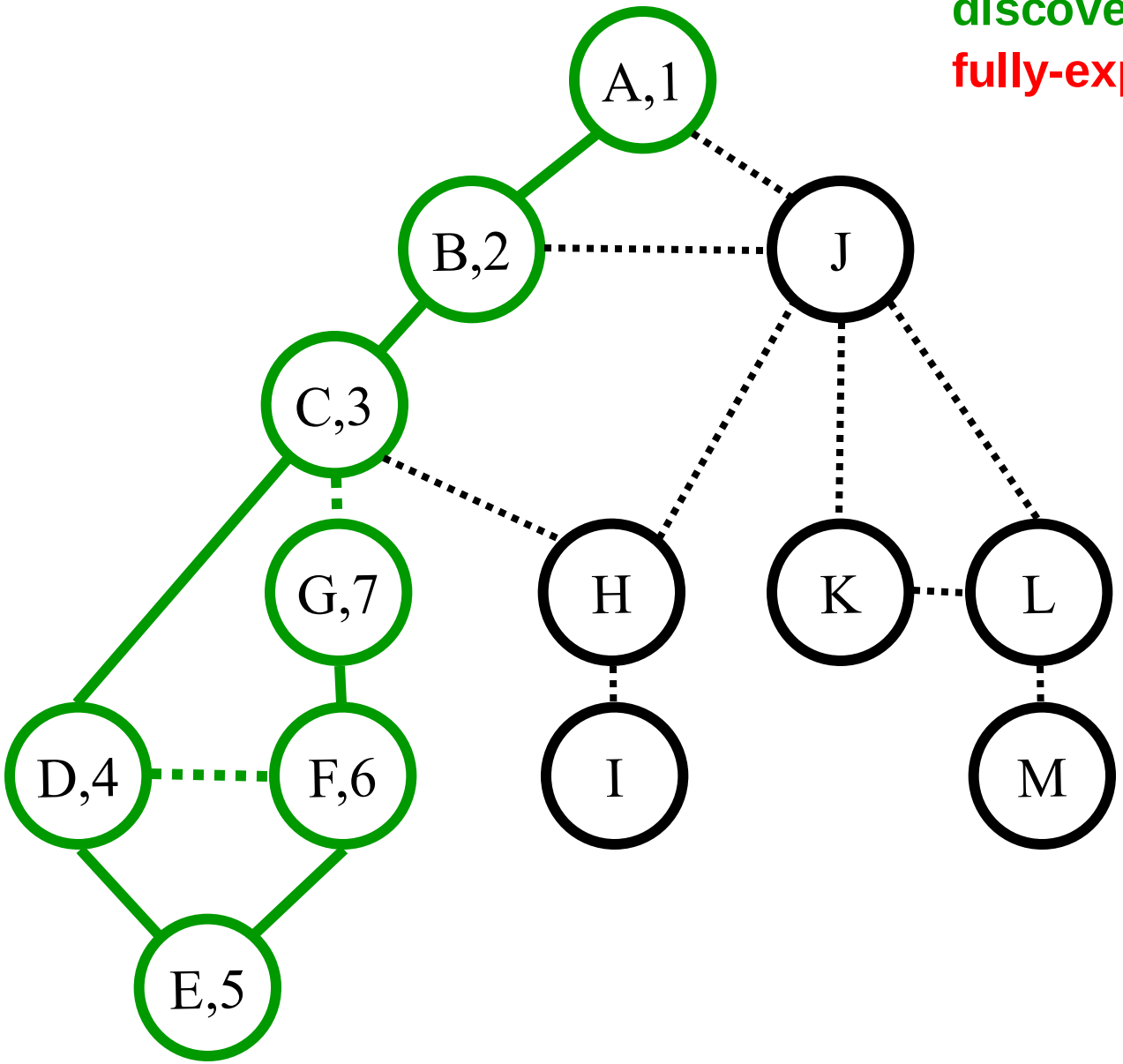
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)
D (~~C~~,~~E~~,F)
E (~~D~~,~~F~~)
F (~~D~~,~~E~~,~~G~~)
G (~~C~~,~~F~~)

st[] =
{1,2,3,4,5,
6,7}

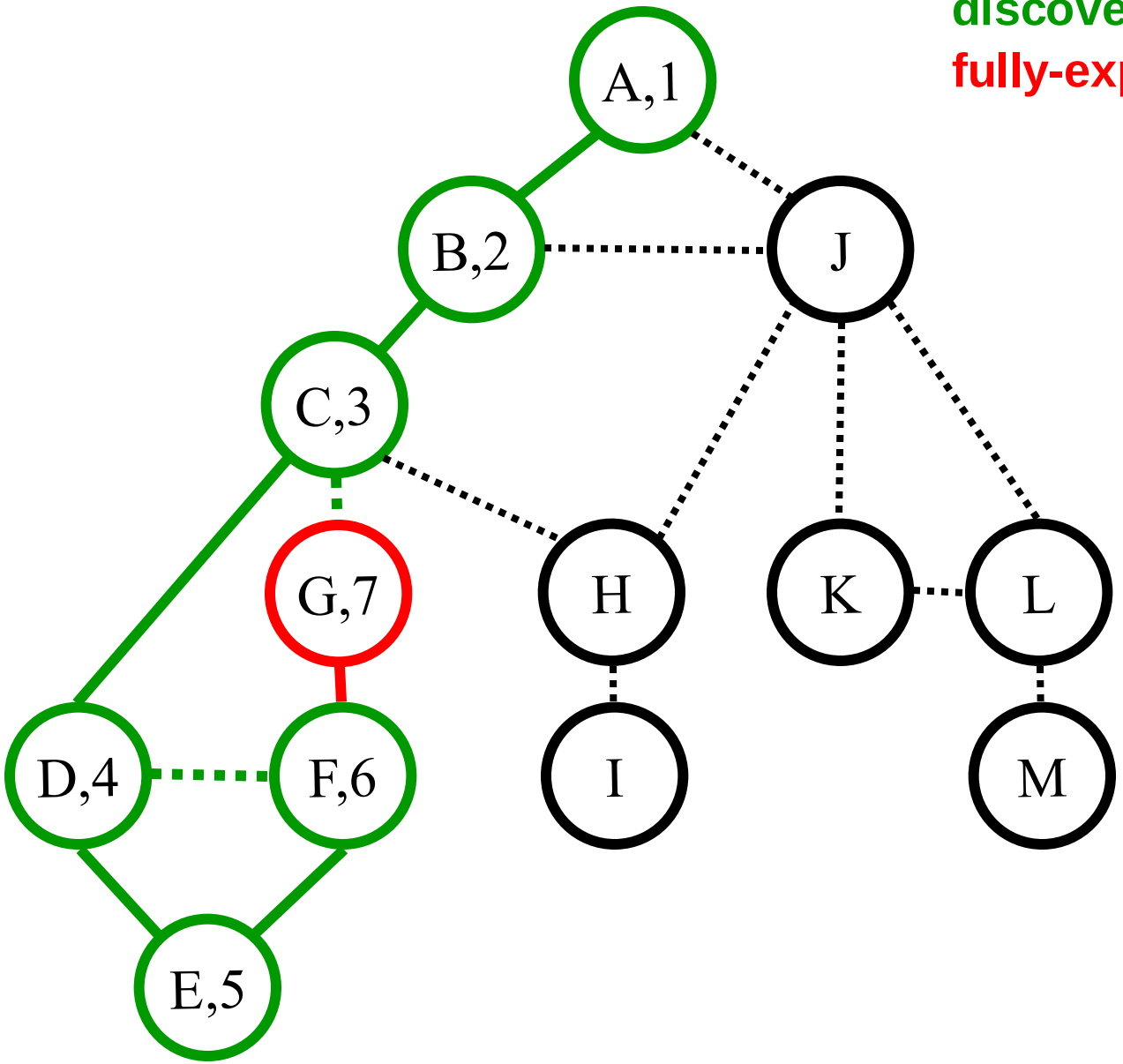
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)
D (~~C~~,~~E~~,F)
E (~~D~~,~~F~~)
F (~~D~~,~~E~~,~~G~~)

st[] =
{1,2,3,4,5,
6}

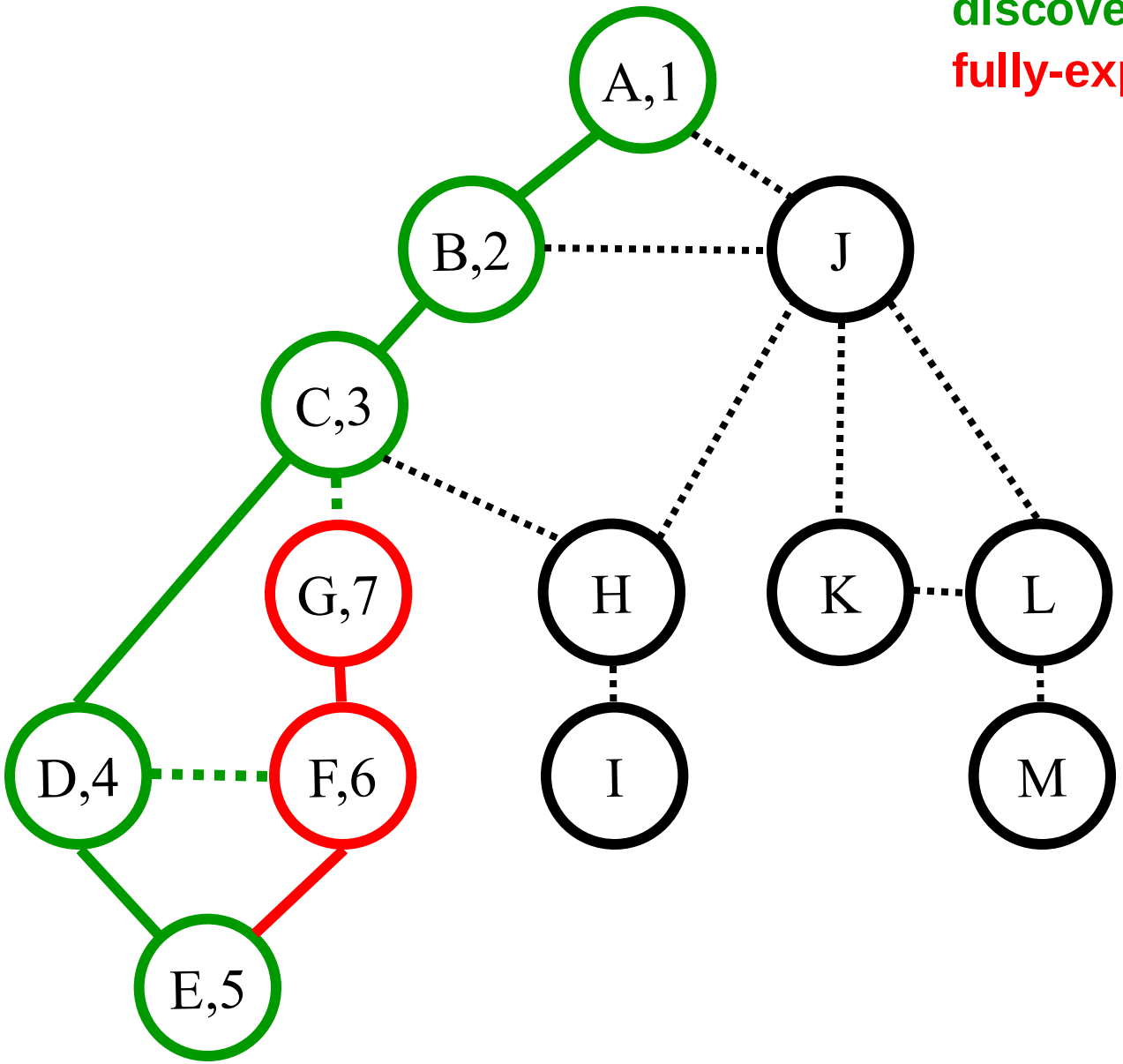
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)
D (~~C~~,~~E~~,F)
E (~~D~~,~~F~~)

st[] =
{1,2,3,4,5}

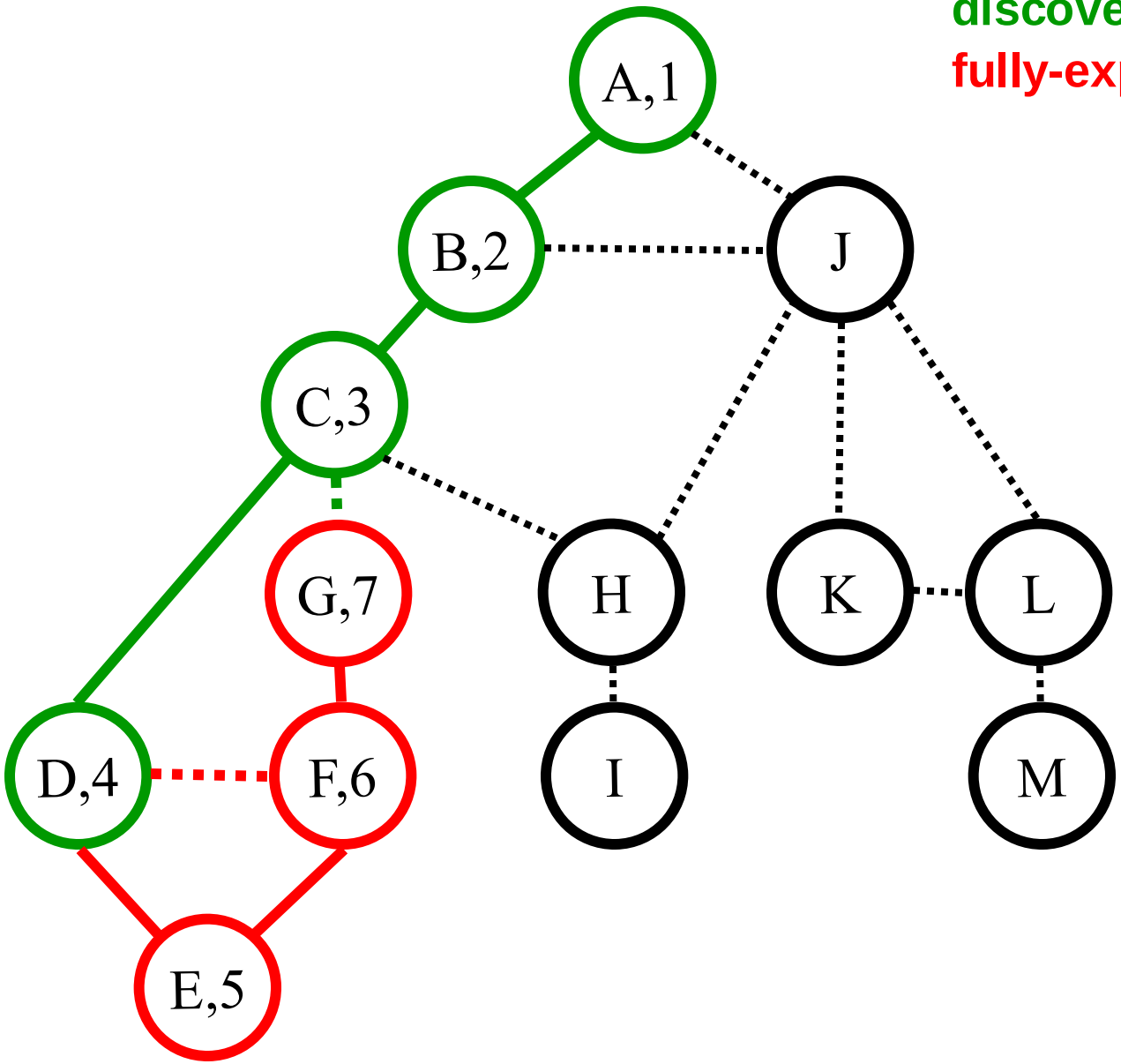
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)
D (~~C~~,~~E~~,~~F~~)

st[] =
{1,2,3,4}

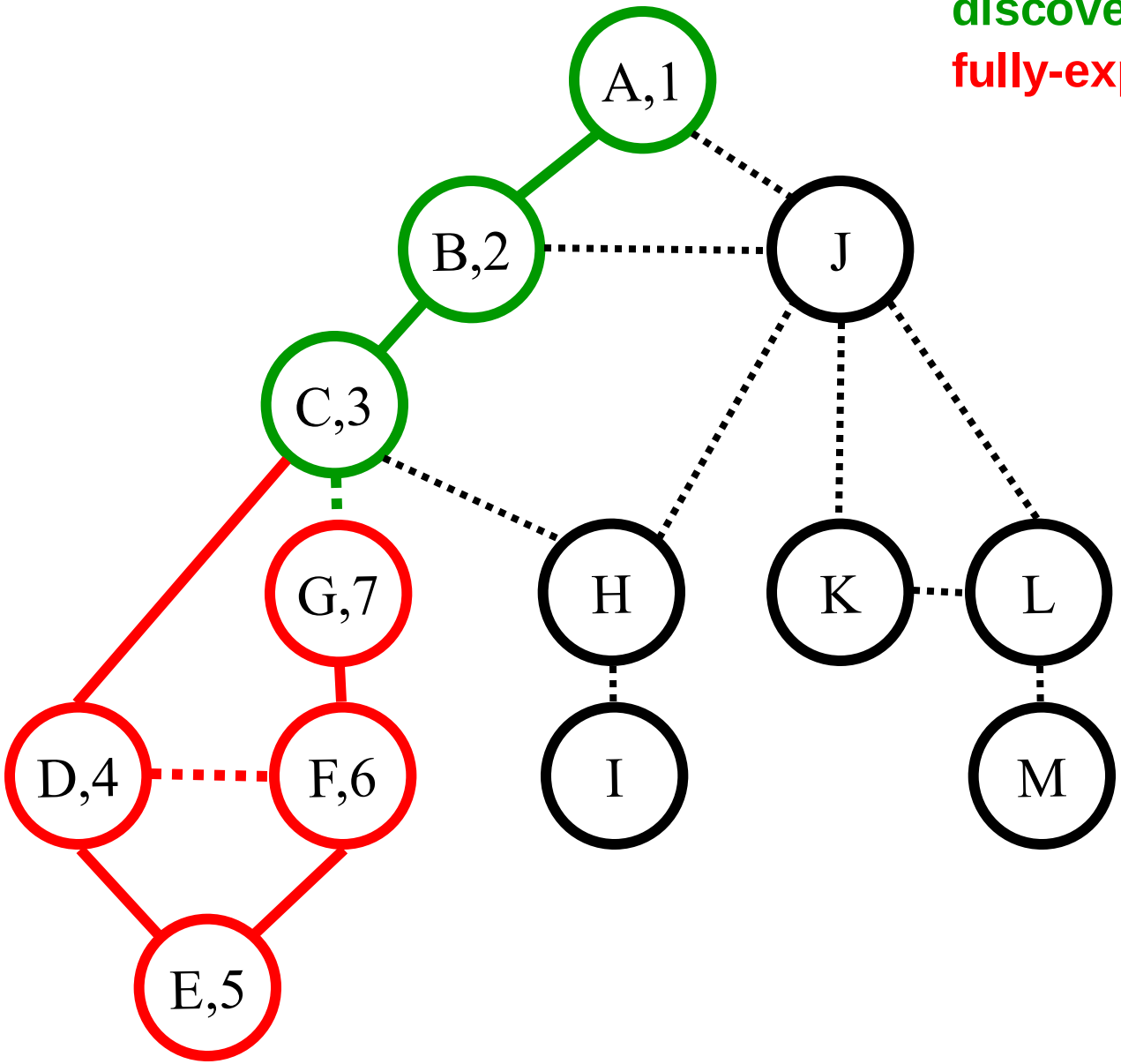
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,G,H)

st[] =
{1,2,3}

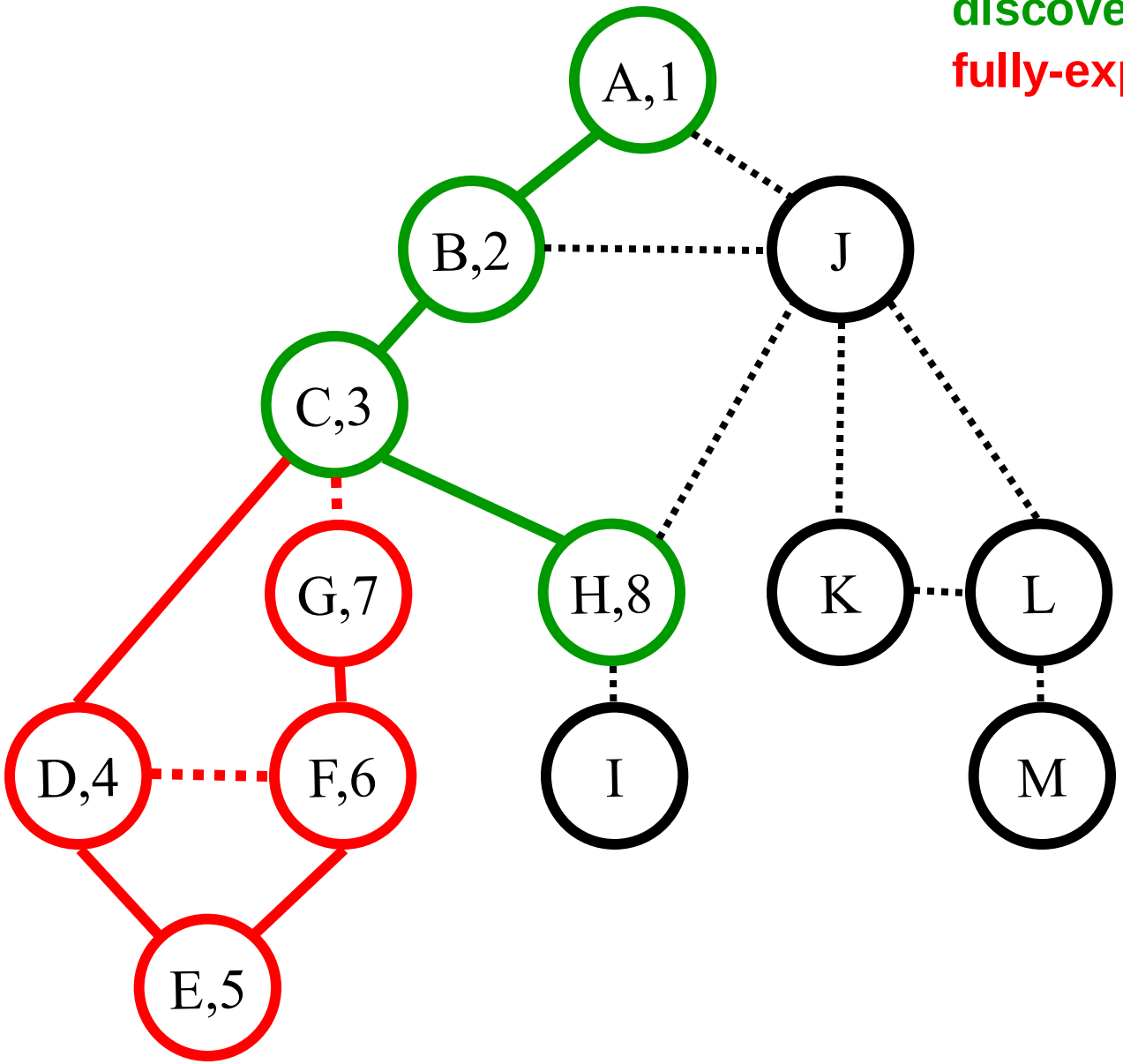
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (C,I,J)

st[] =
{1,2,3,8}

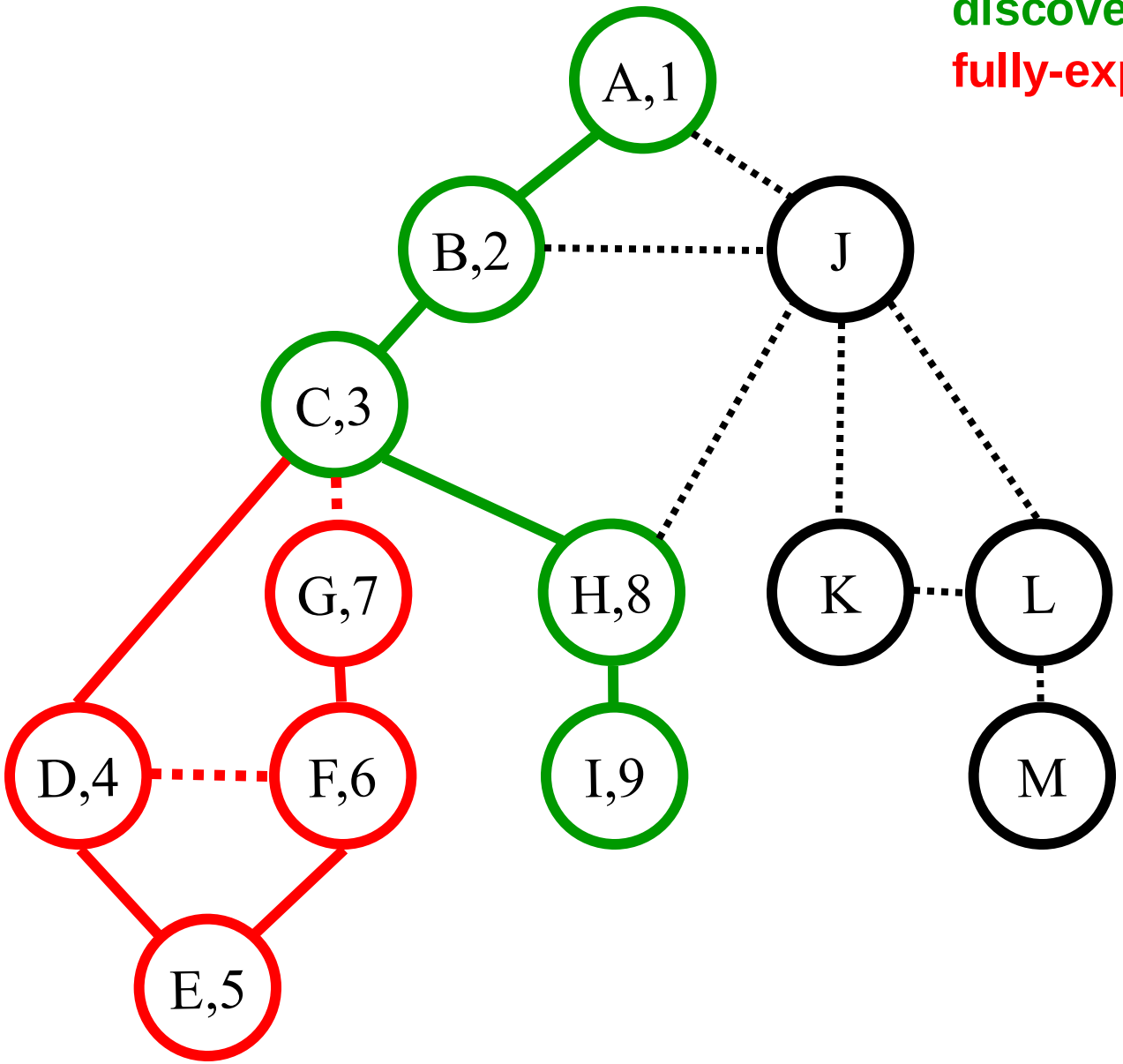
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,J)
I (H)

st[] =
{1,2,3,8,9}

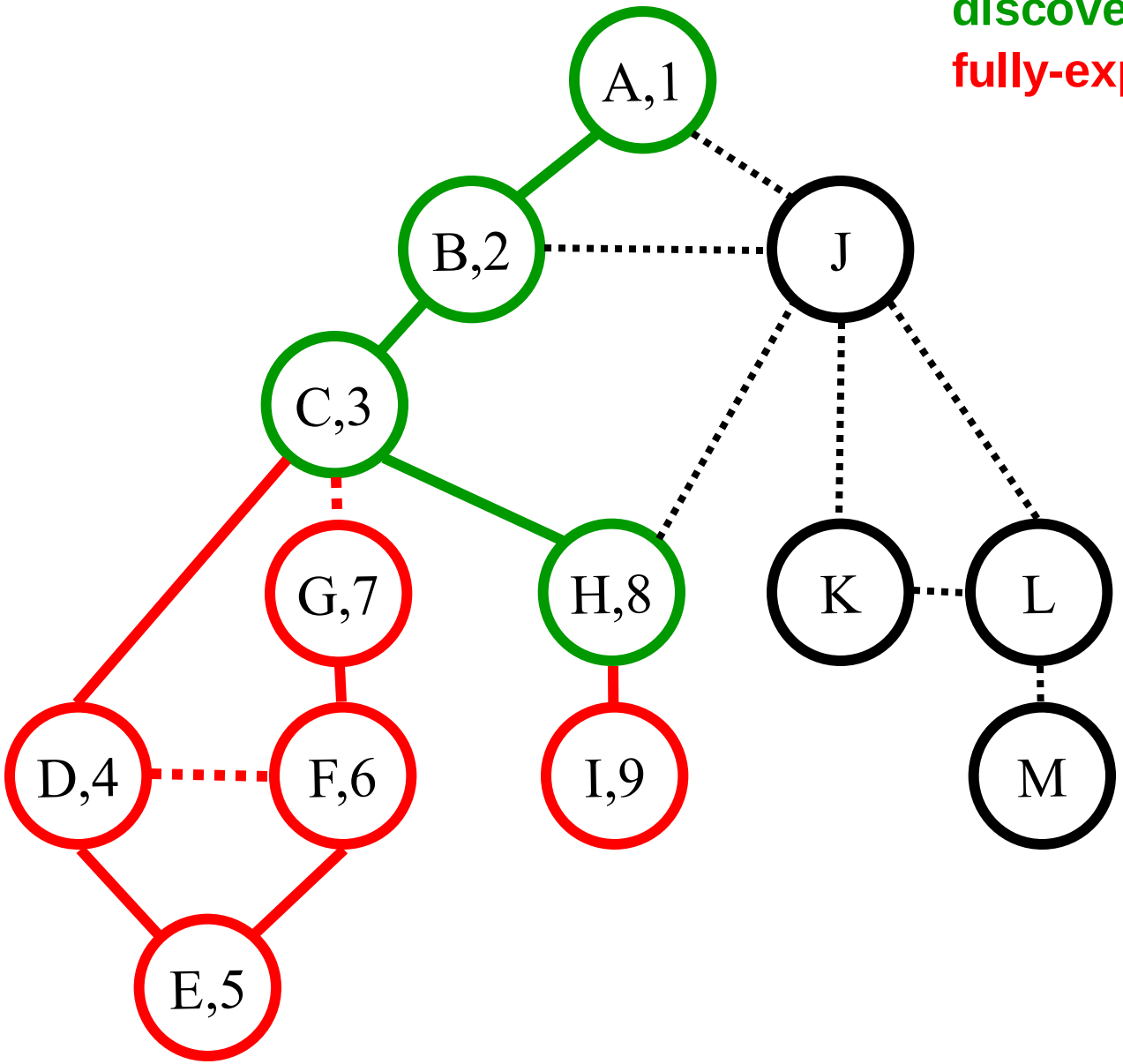
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,J)

st[] =
{1,2,3,8}

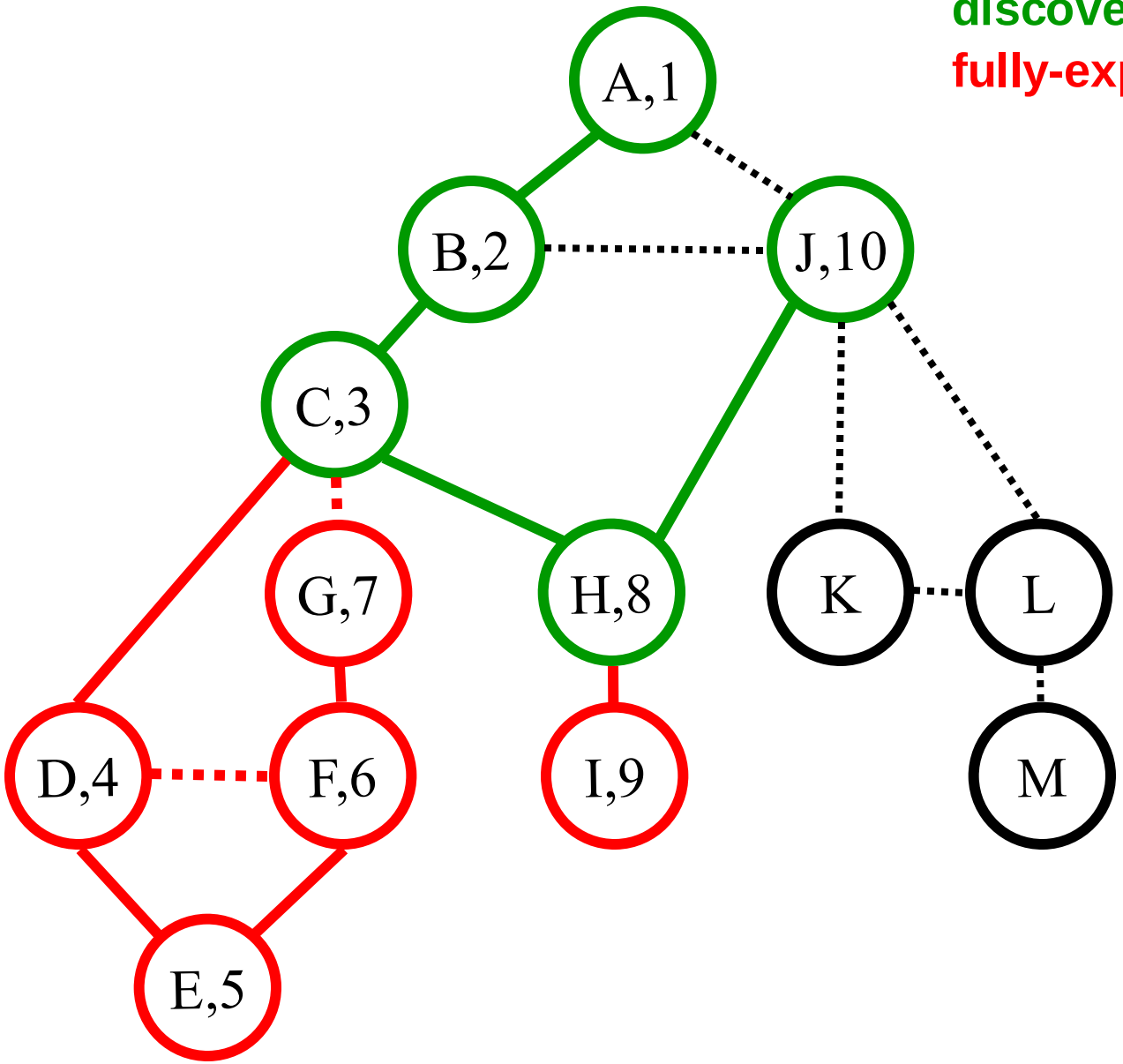
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)
J (A,B,H,K,L)

st[] =
{1,2,3,8,
10}

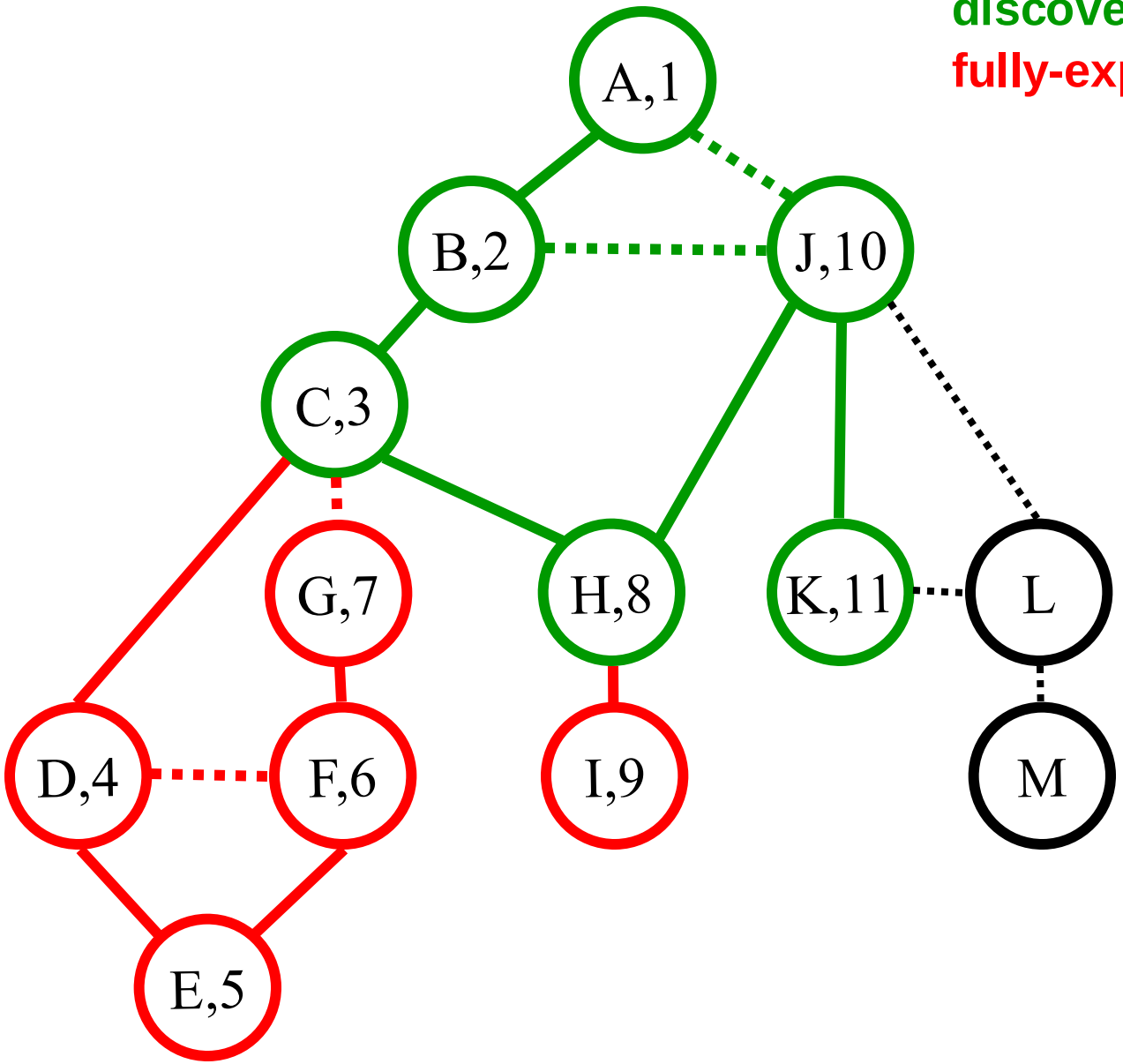
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)
J (~~A~~,~~B~~,~~H~~,K,L)
K (J,L)

st[] =
{1,2,3,8,10
,11}

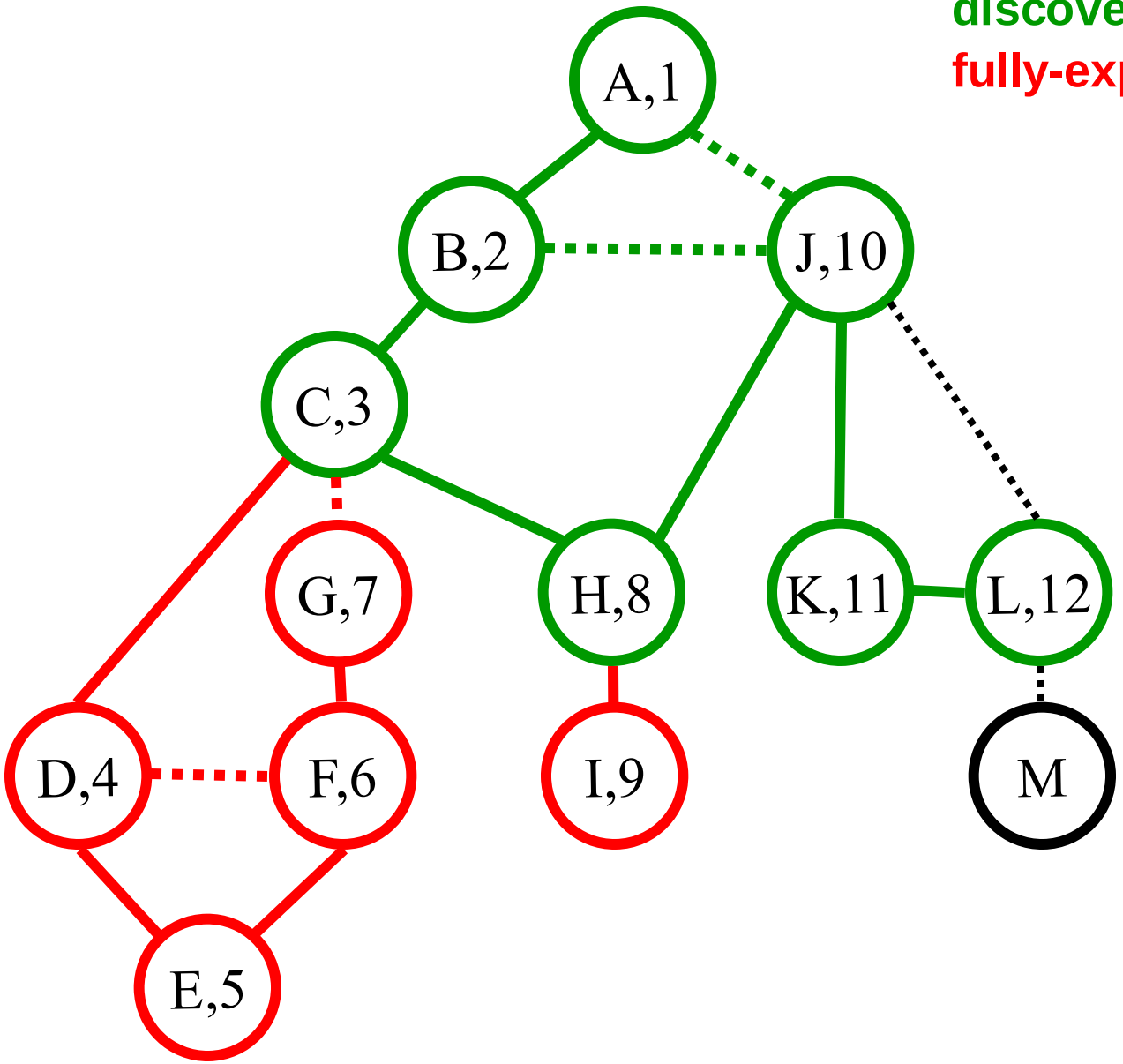
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)
J (~~A~~,~~B~~,~~H~~,K,L)
K (~~J~~,~~L~~)
L (J,K,M)

st[] =
{1,2,3,8,10
,11,12}

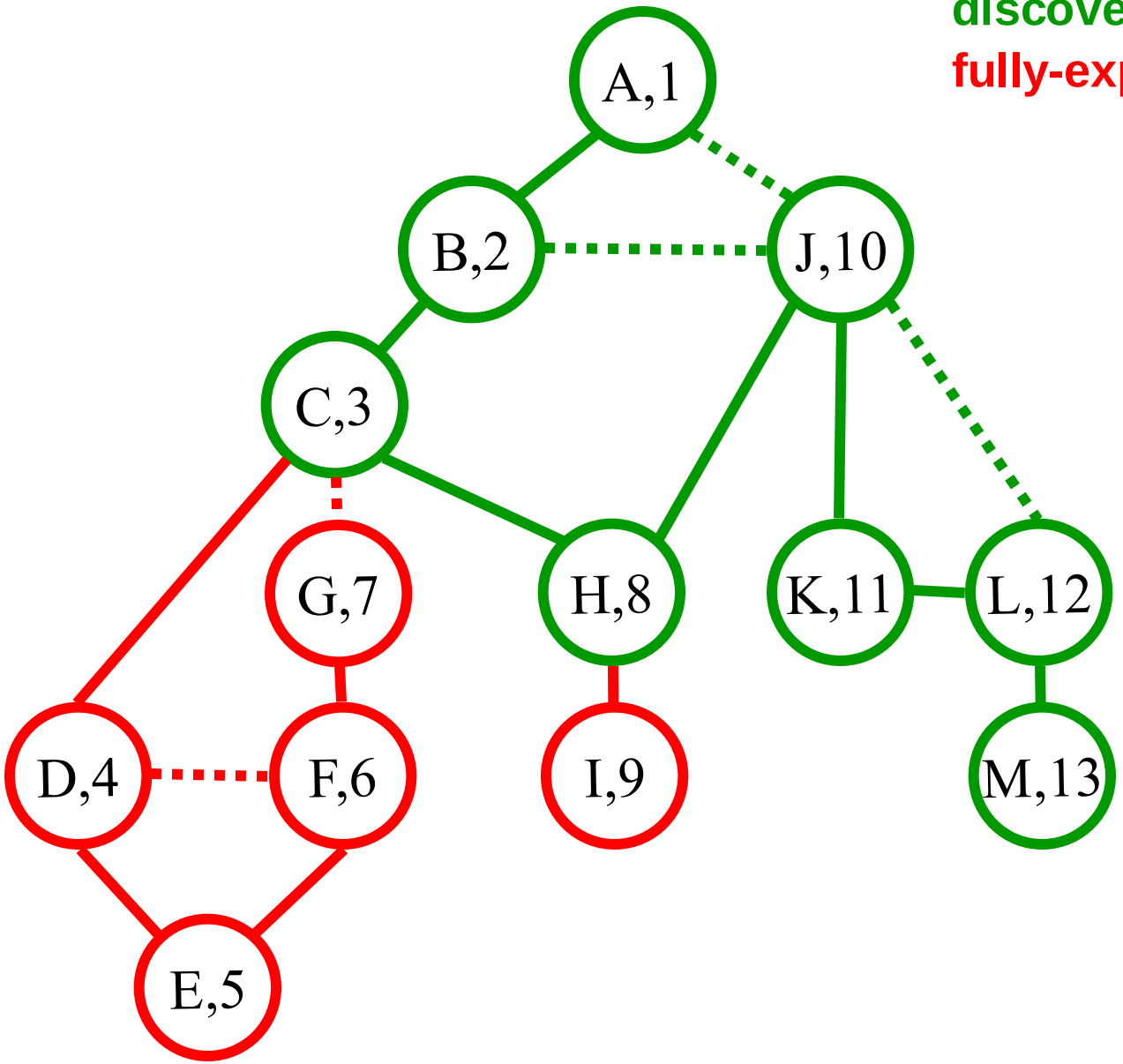
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)
J (~~A~~,~~B~~,~~H~~,K,L)
K (~~J~~,L)
L (~~J~~,~~K~~,M)
M(L)

st[] =
{1,2,3,8,10
,11,12,13}

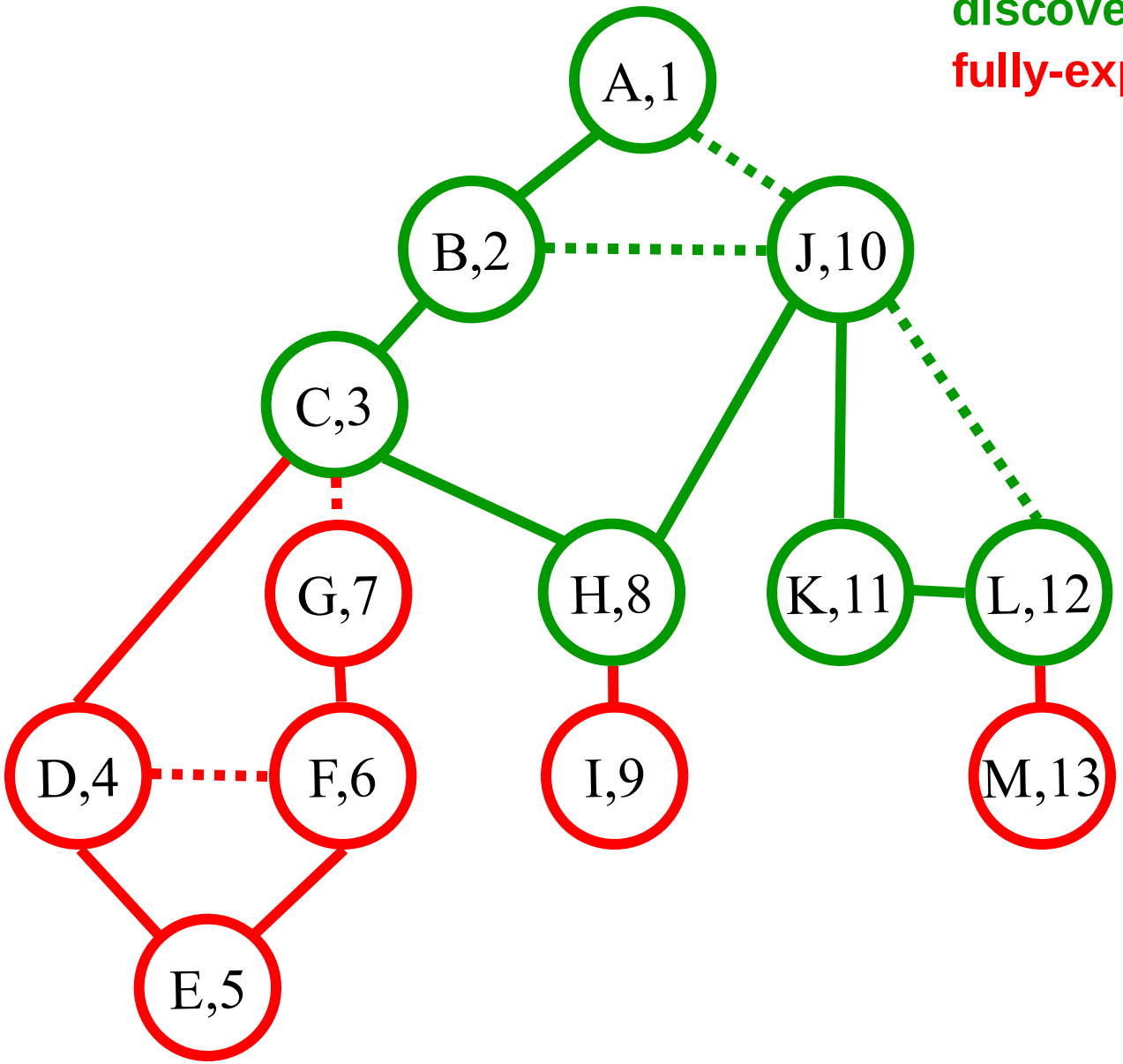
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)
J (~~A~~,~~B~~,~~H~~,K,L)
K (J,L)
L (J,~~K~~,~~M~~)

st[] =
{1,2,3,8,10
,11,12}

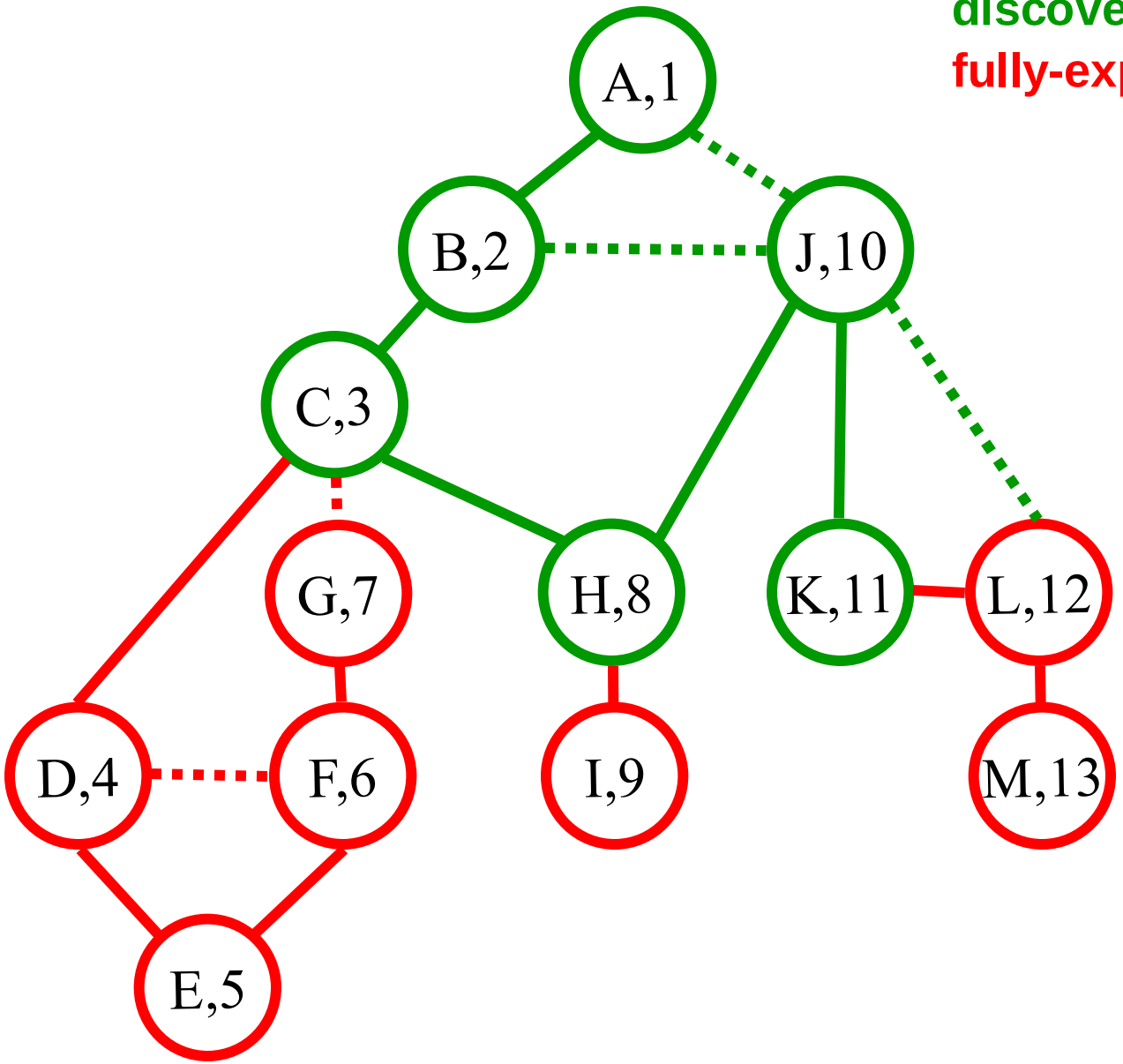
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)
J (~~A~~,~~B~~,~~H~~,K,L)
K (~~J~~,~~L~~)

st[] =
{1,2,3,8,10
,11}

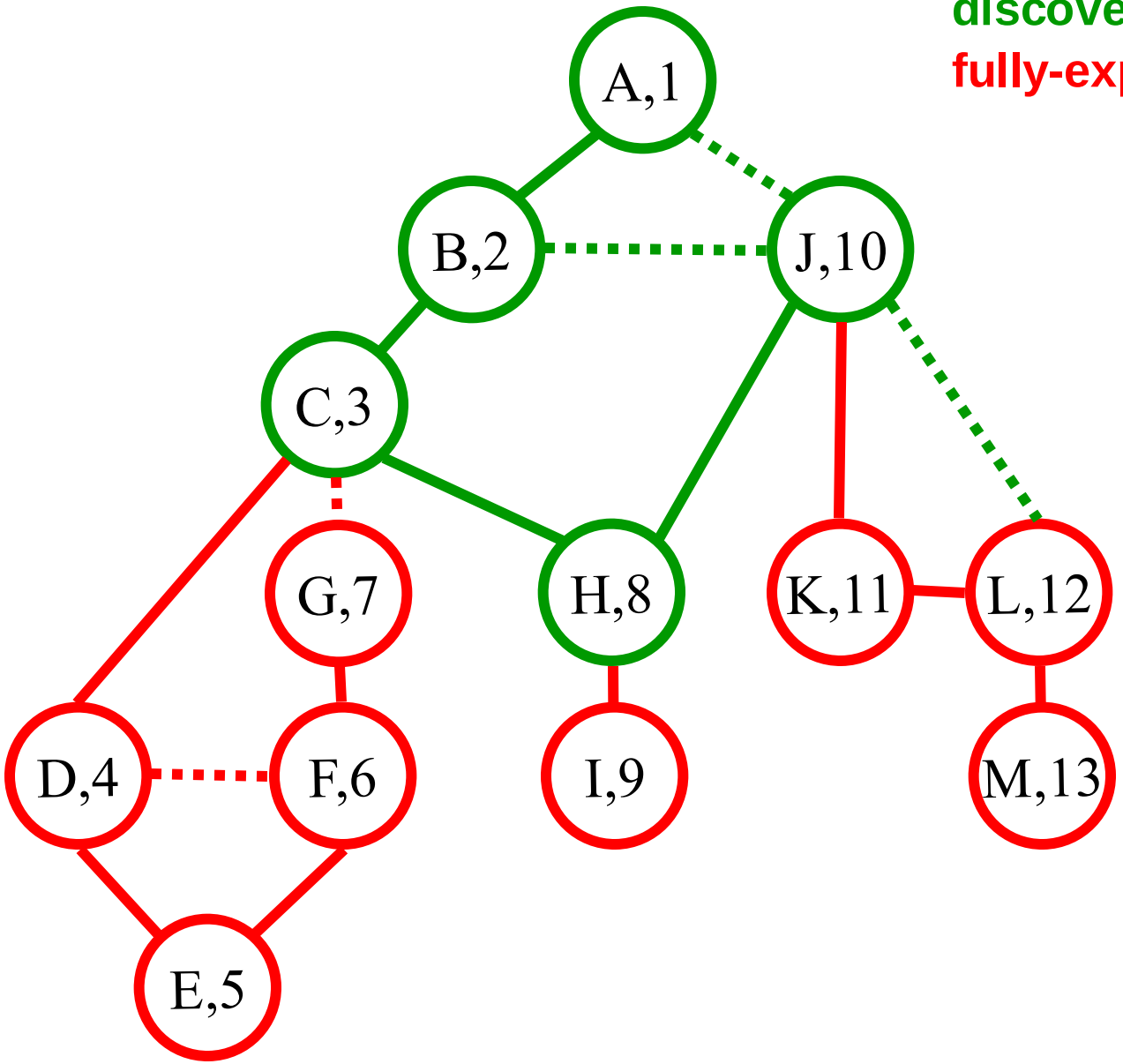
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)
J (~~A~~,~~B~~,~~H~~,K,L)

st[] =
{1,2,3,8,
10}

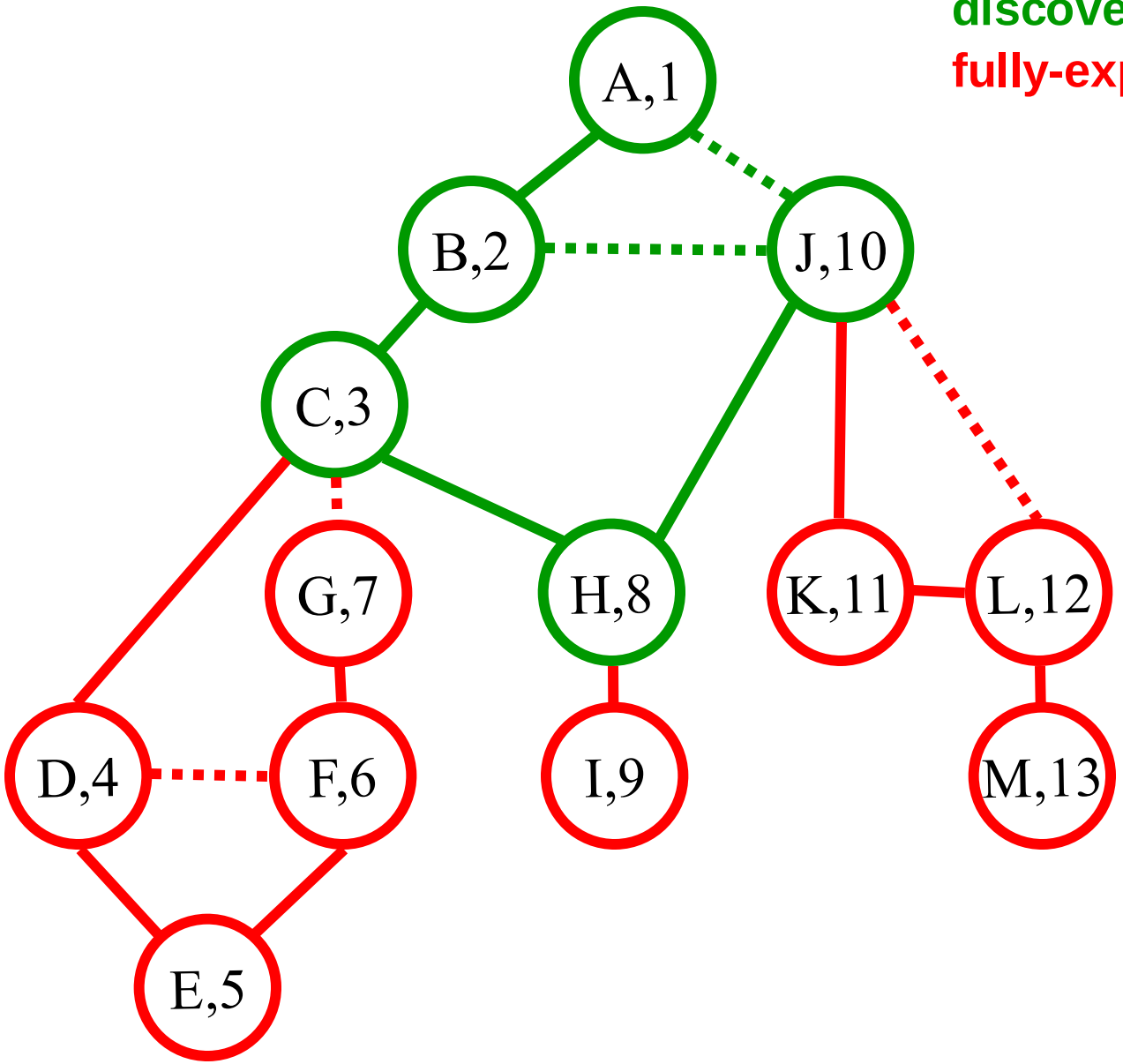
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)
J (~~A~~,~~B~~,~~H~~,~~K~~,~~L~~)

st[] =
{1,2,3,8,
10}

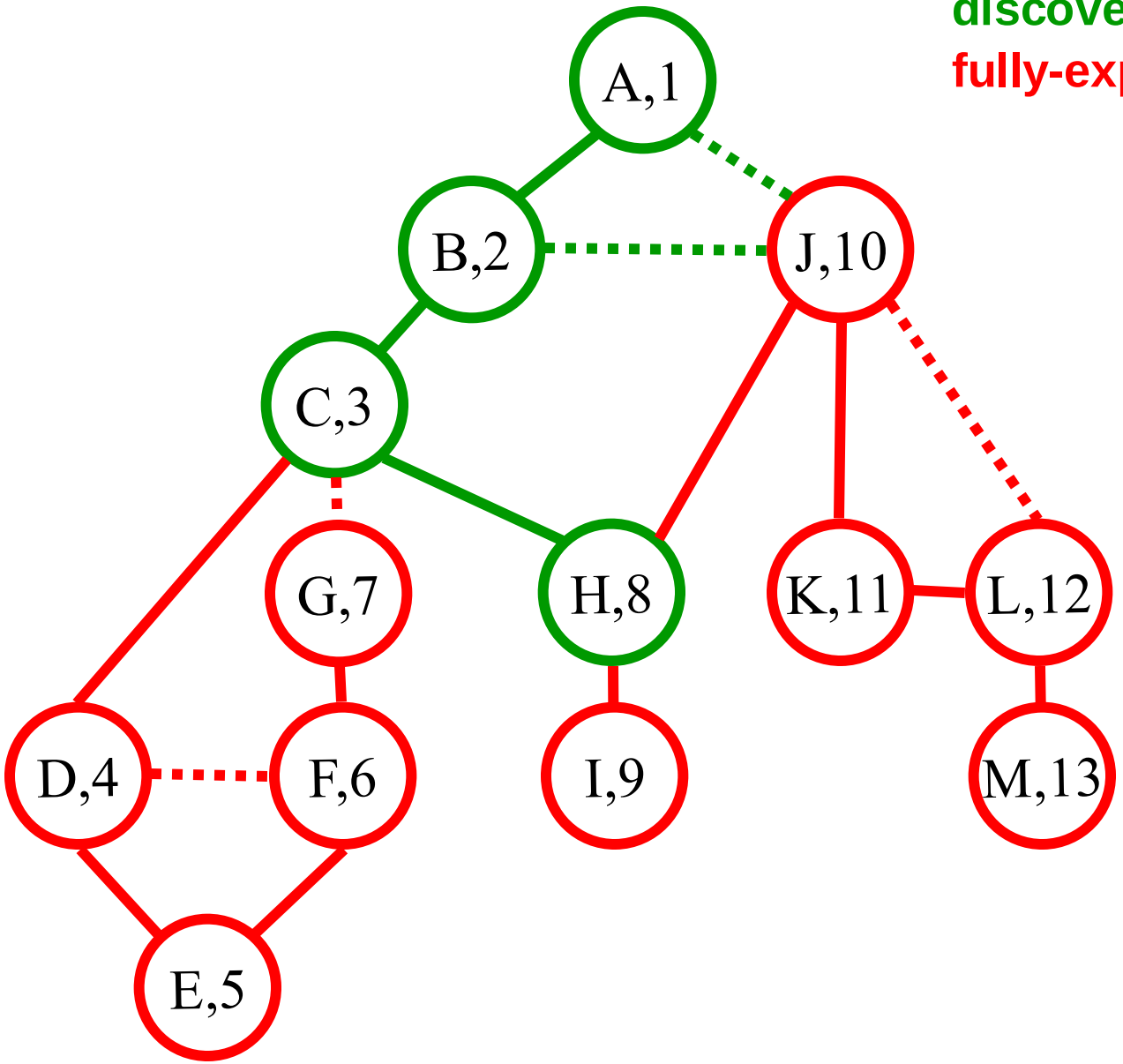
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)
H (~~C~~,~~I~~,~~J~~)

st[] =
{1,2,3,8}

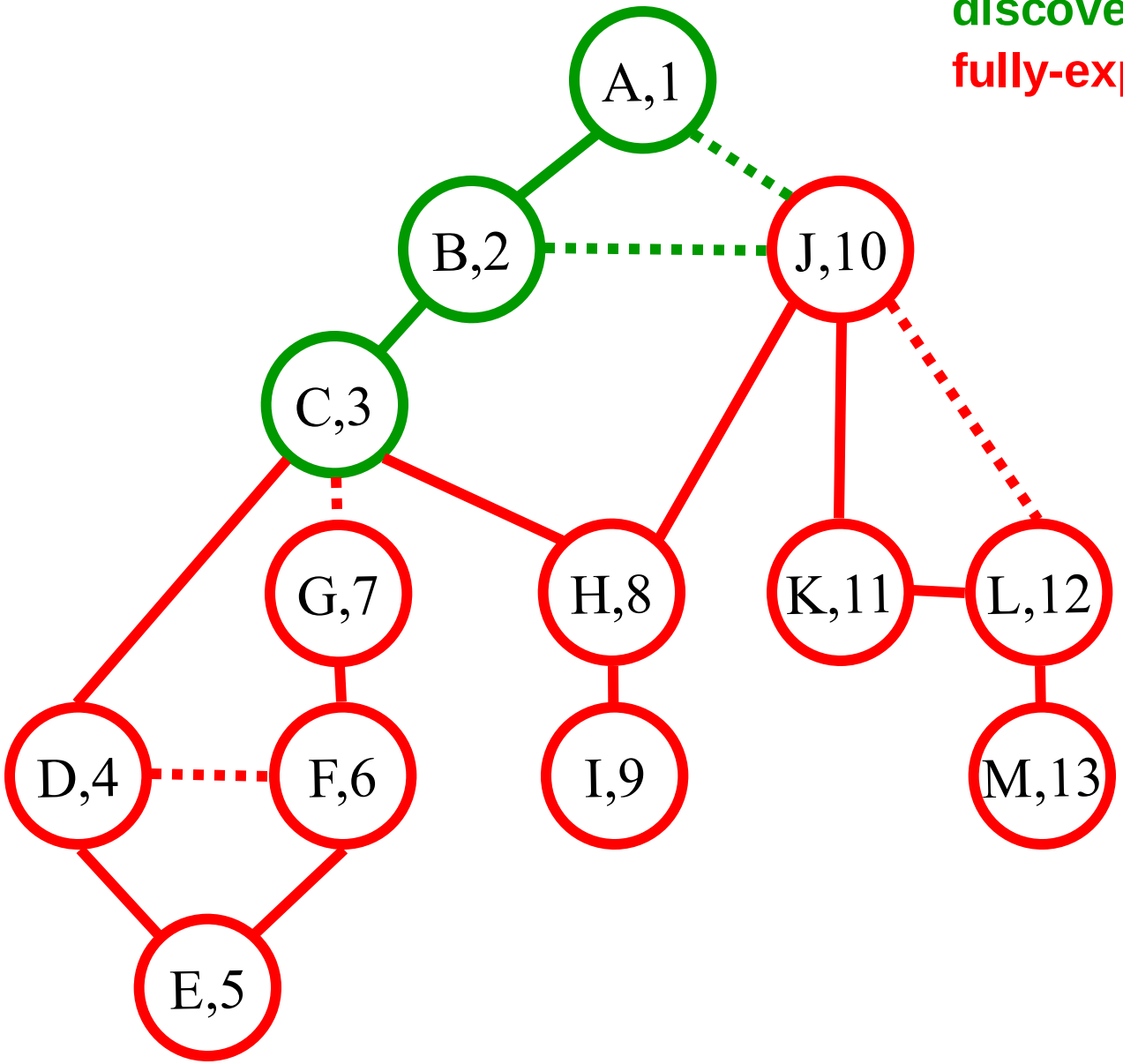
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)
C (~~B~~,~~D~~,~~G~~,~~H~~)

st[] =
{1,2,3}

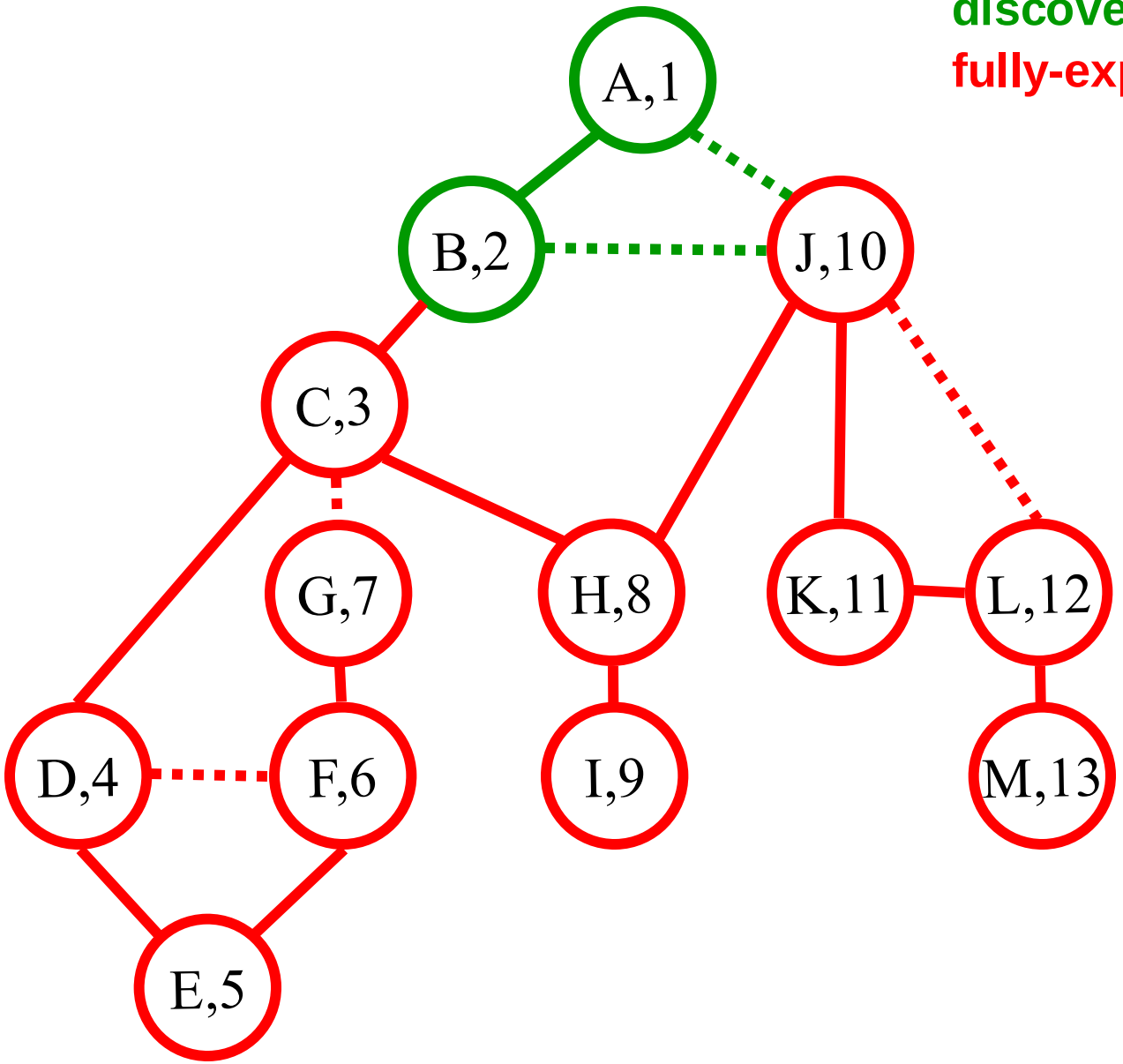
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,J)

st[] =
{1,2}

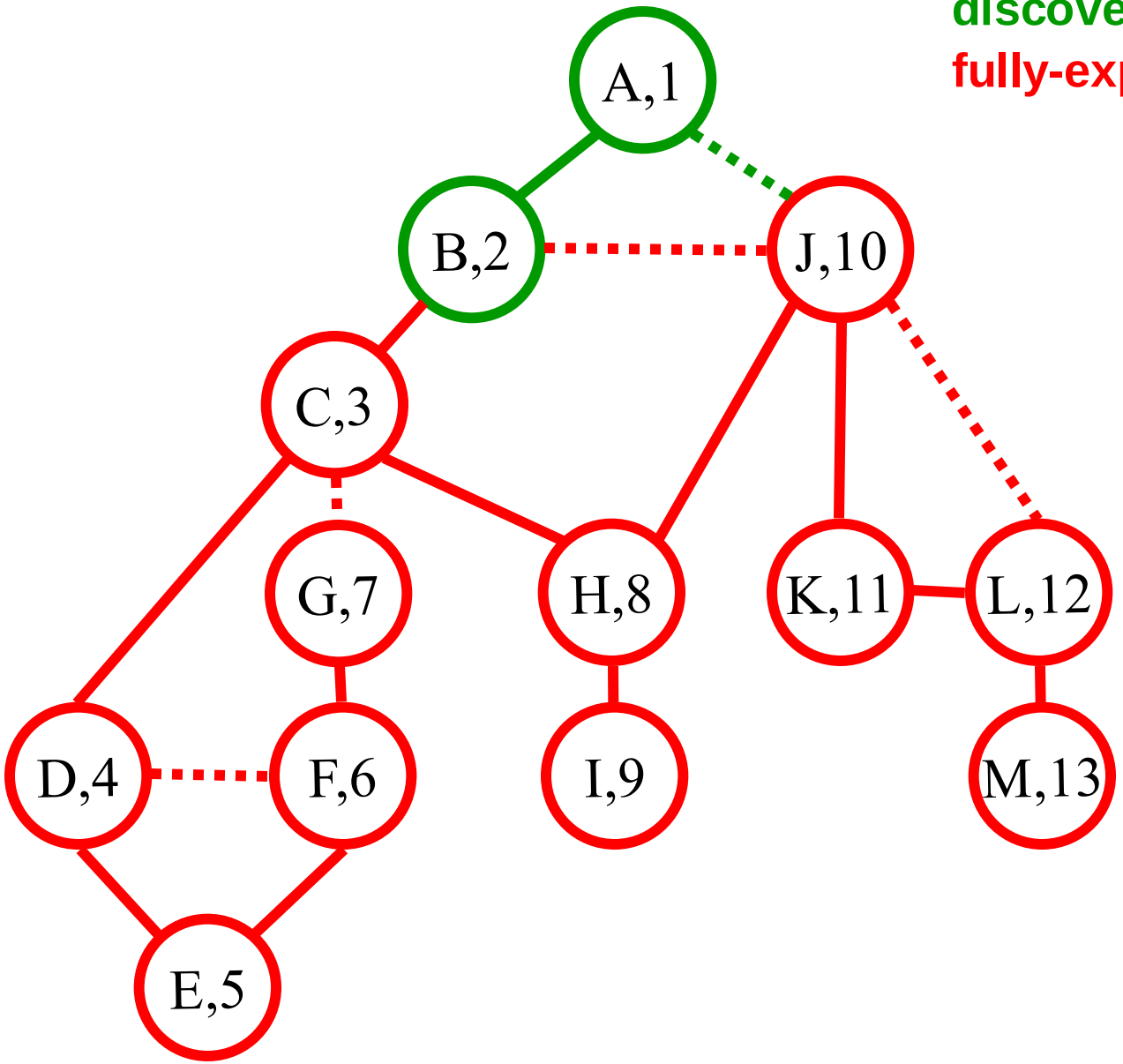
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)
B (~~A~~,~~C~~,~~J~~)

st[] =
{1,2}

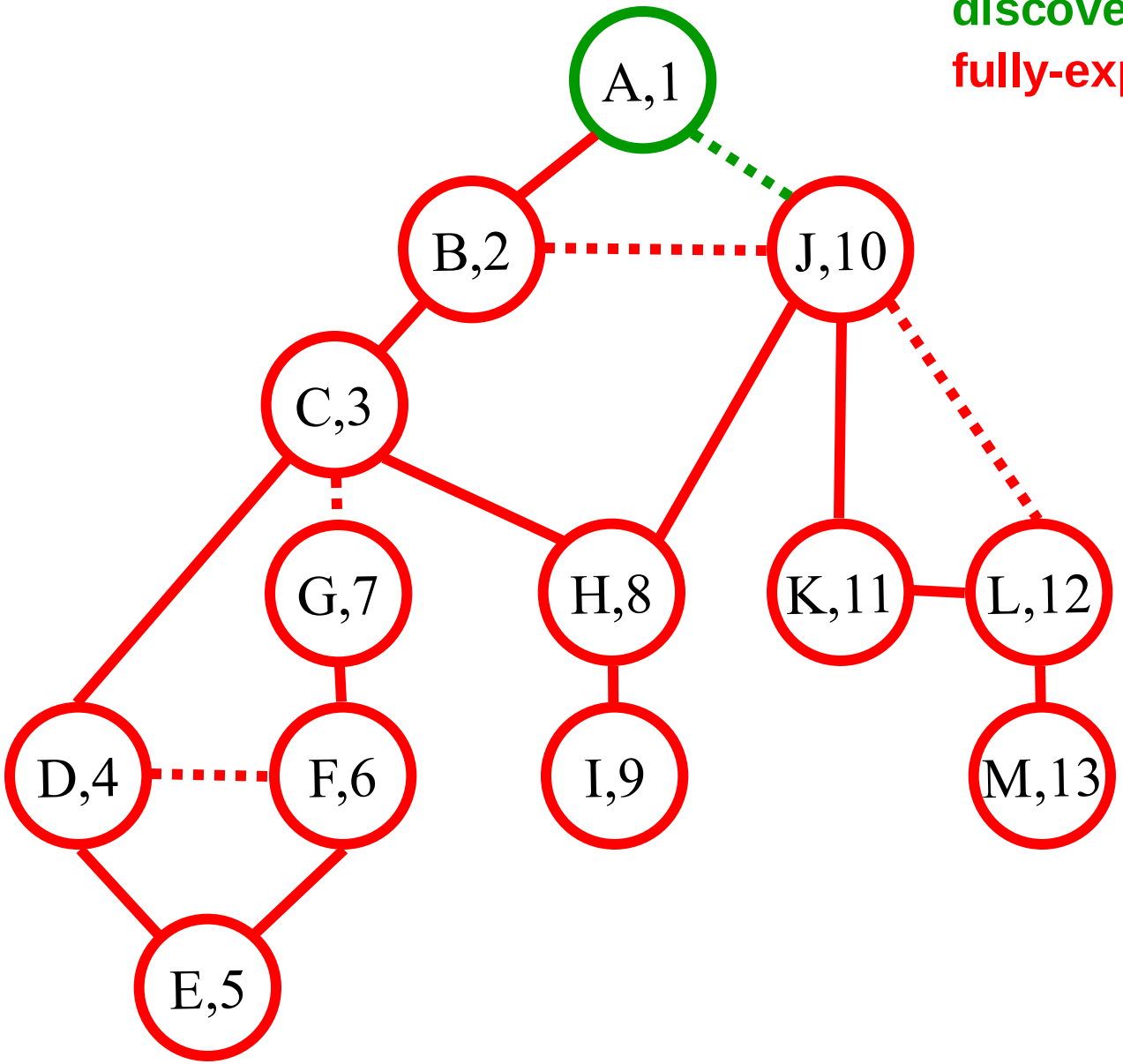
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,J)

st[] =
{1}

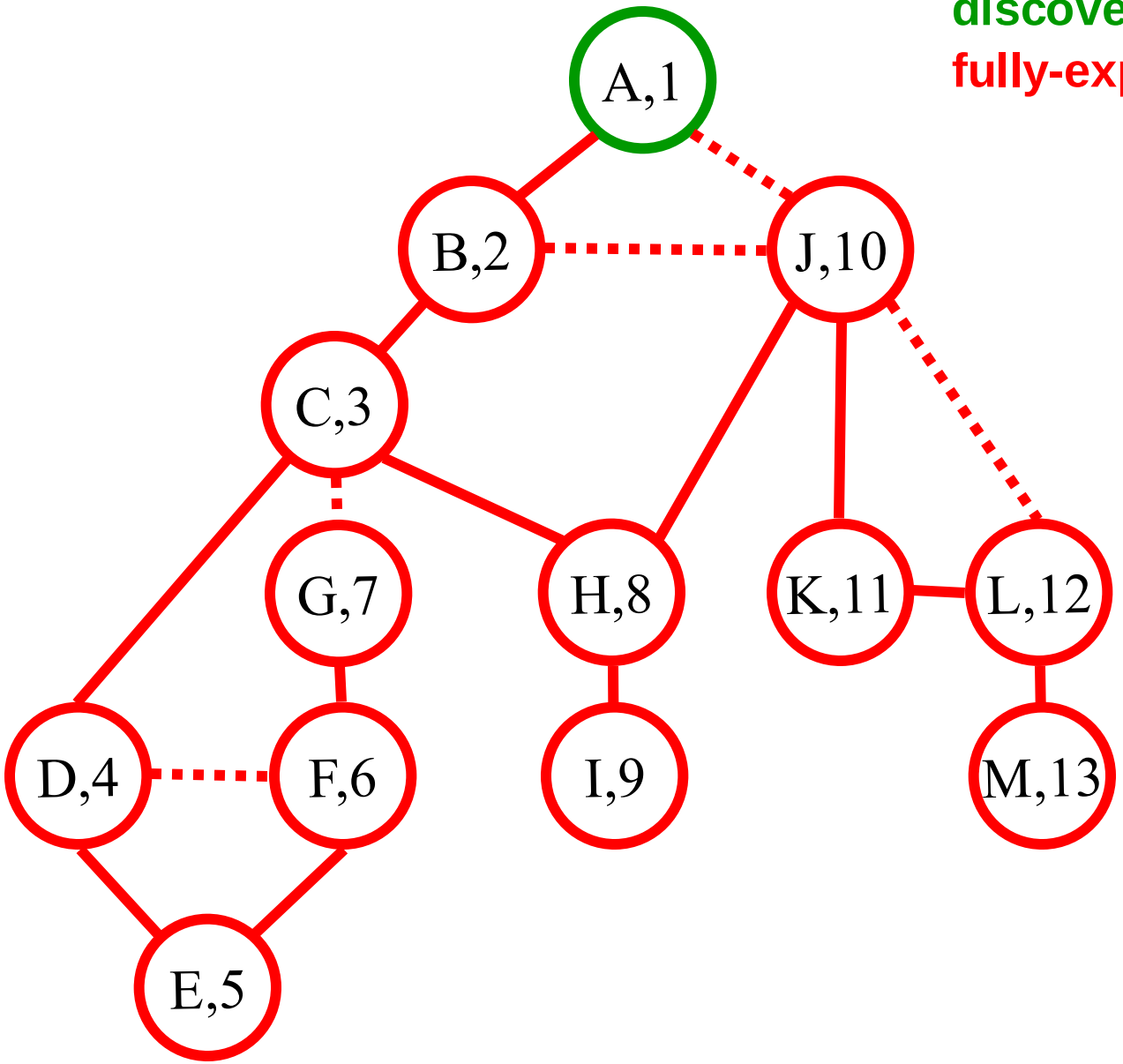
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

A (~~B~~,~~J~~)

st[] =
{1}

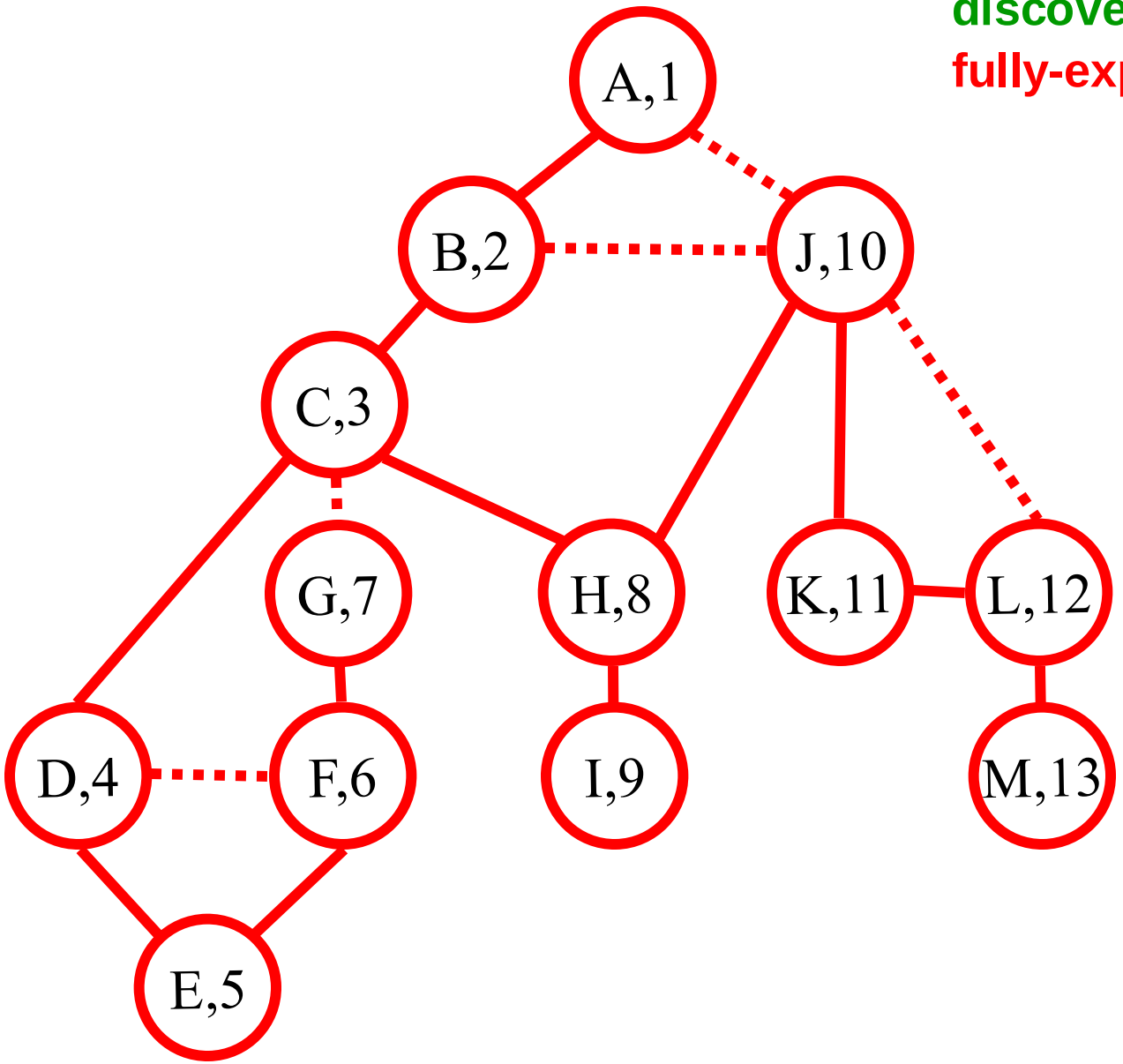
# DFS(A)

Color code:

**undiscovered**

**discovered**

**fully-explored**




Call Stack:
(Edge list)

TA-DA!!

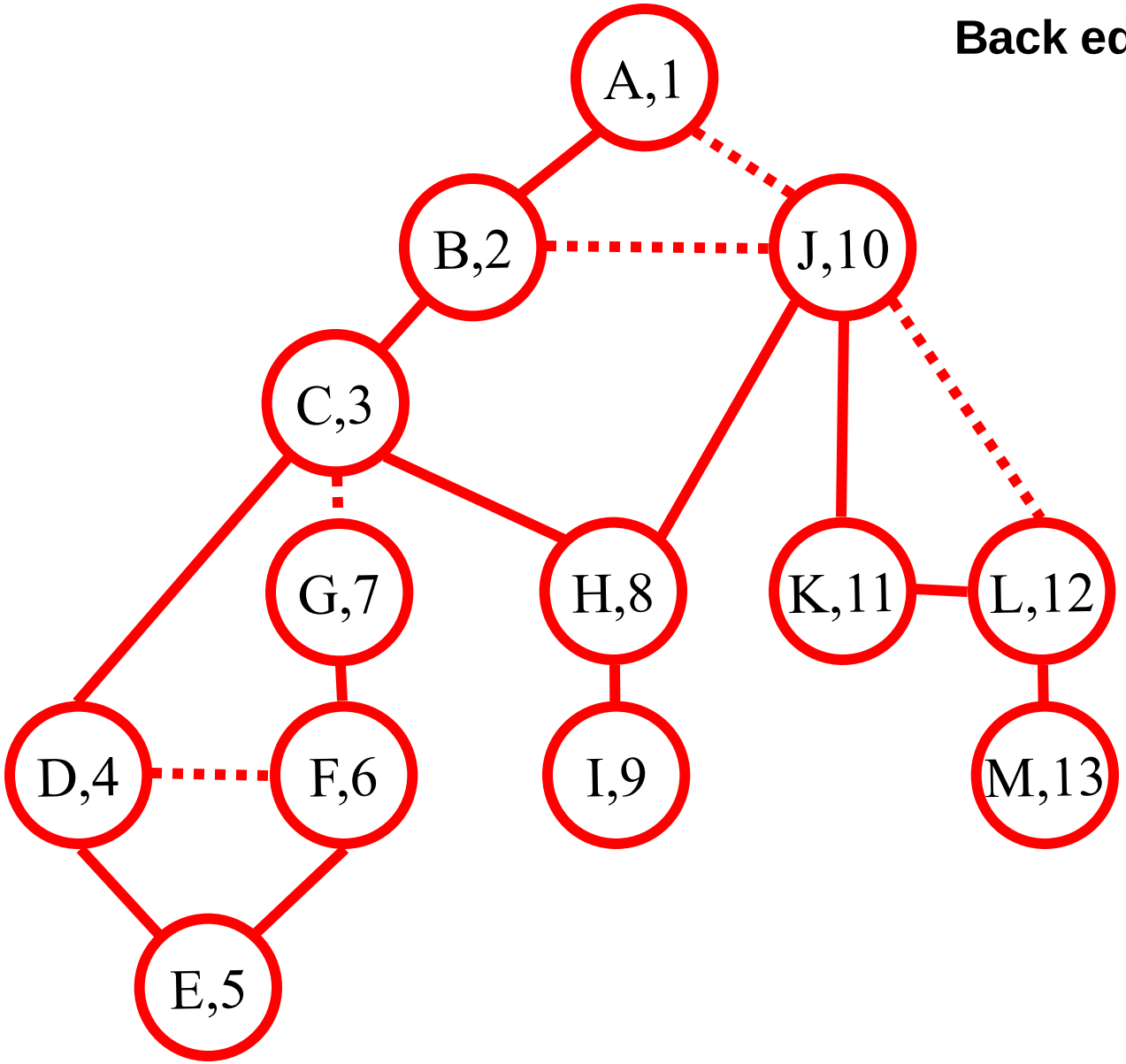
st[] = {}

# DFS(A)

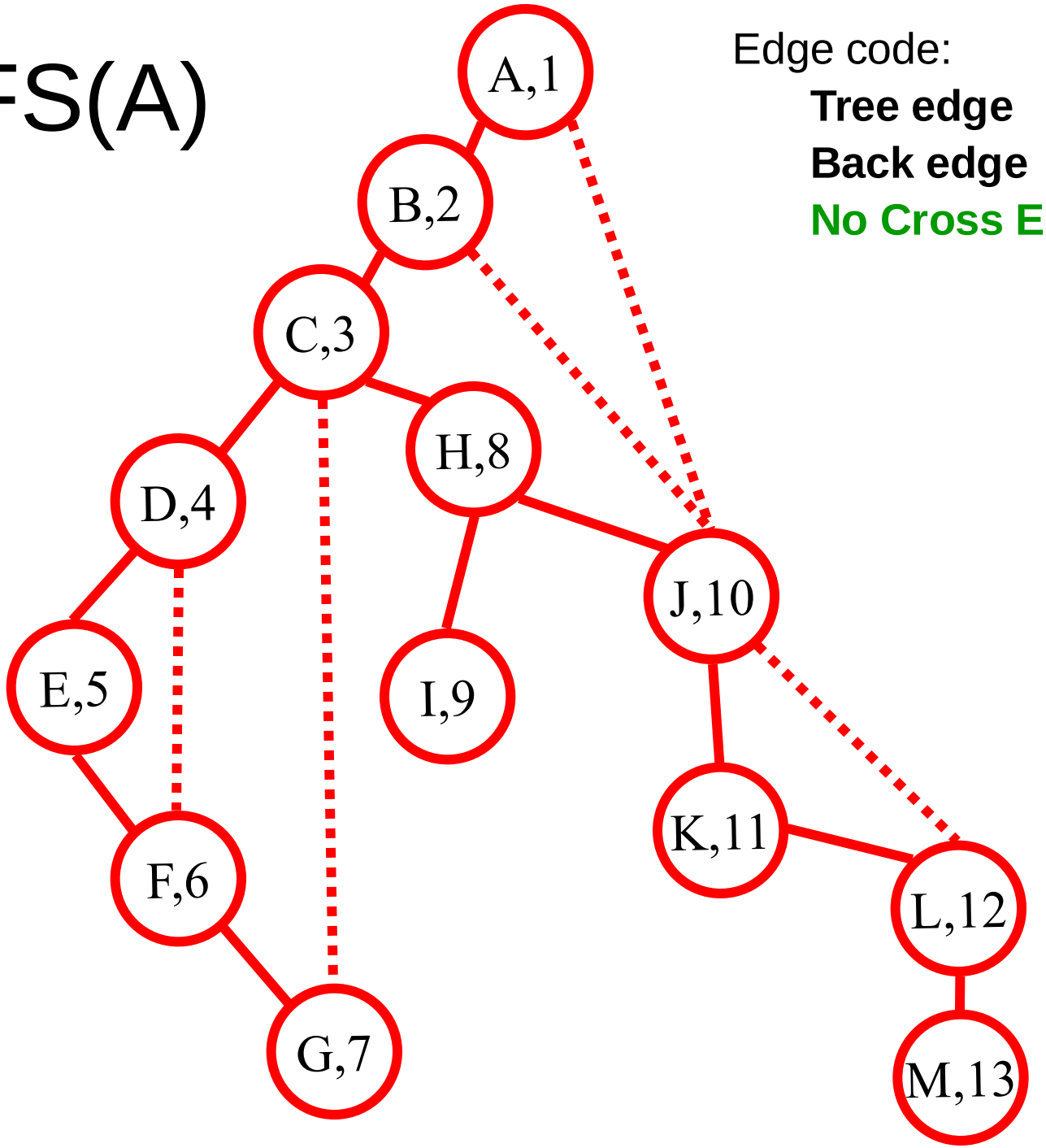
Edge code:

**Tree edge**

**Back edge**



# DFS(A)




Edge code:

**Tree edge**

**Back edge**

**No Cross Edges!**

# Properties of (undirected) DFS

Like BFS($s$):

- DFS($s$) visits $x$ iff there is a path in G from $s$ to $x$

  So, we can use DFS to find connected components
- Edges into then-undiscovered vertices define a ***tree*** – the "DFS tree" of G

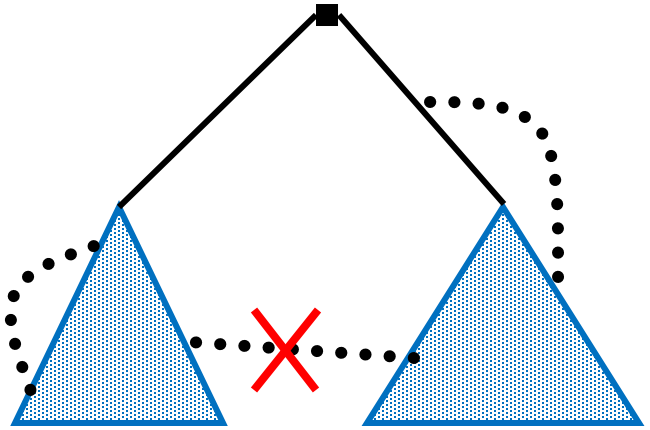
Unlike the BFS tree:

- The DFS tree isn't minimum depth
- Its levels don't reflect min distance from the root
- Non-tree edges never join vertices on the same or adjacent levels

# Non-Tree Edges in DFS

BFS tree ≠ DFS tree, but, as with BFS, DFS has found a tree in the graph s.t. non-tree edges are "simple" in some way.

All non-tree edges join a vertex and one of its descendents/ancestors in the DFS tree

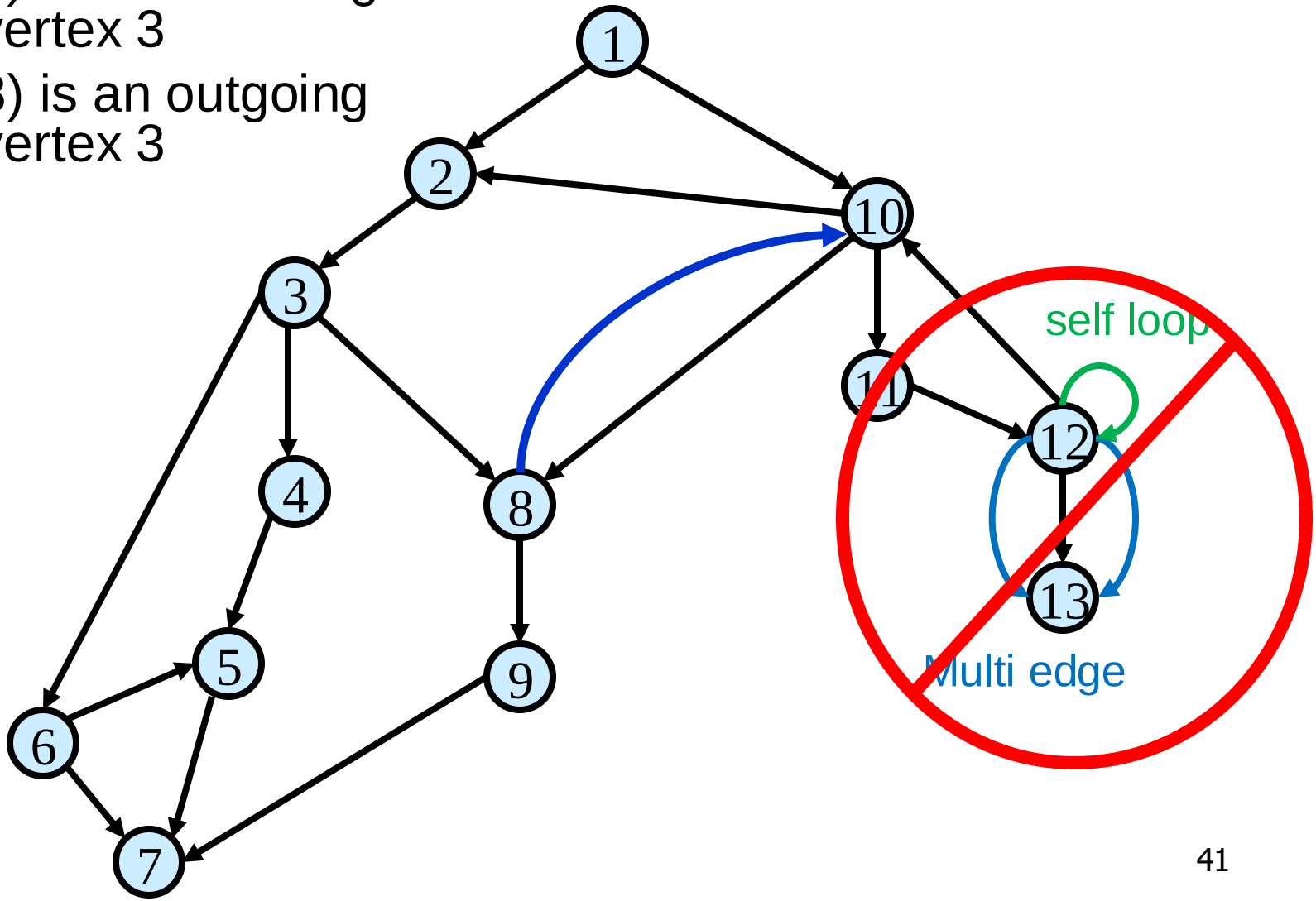


# Directed Graph and Topological Ordering

# Directed Graphs

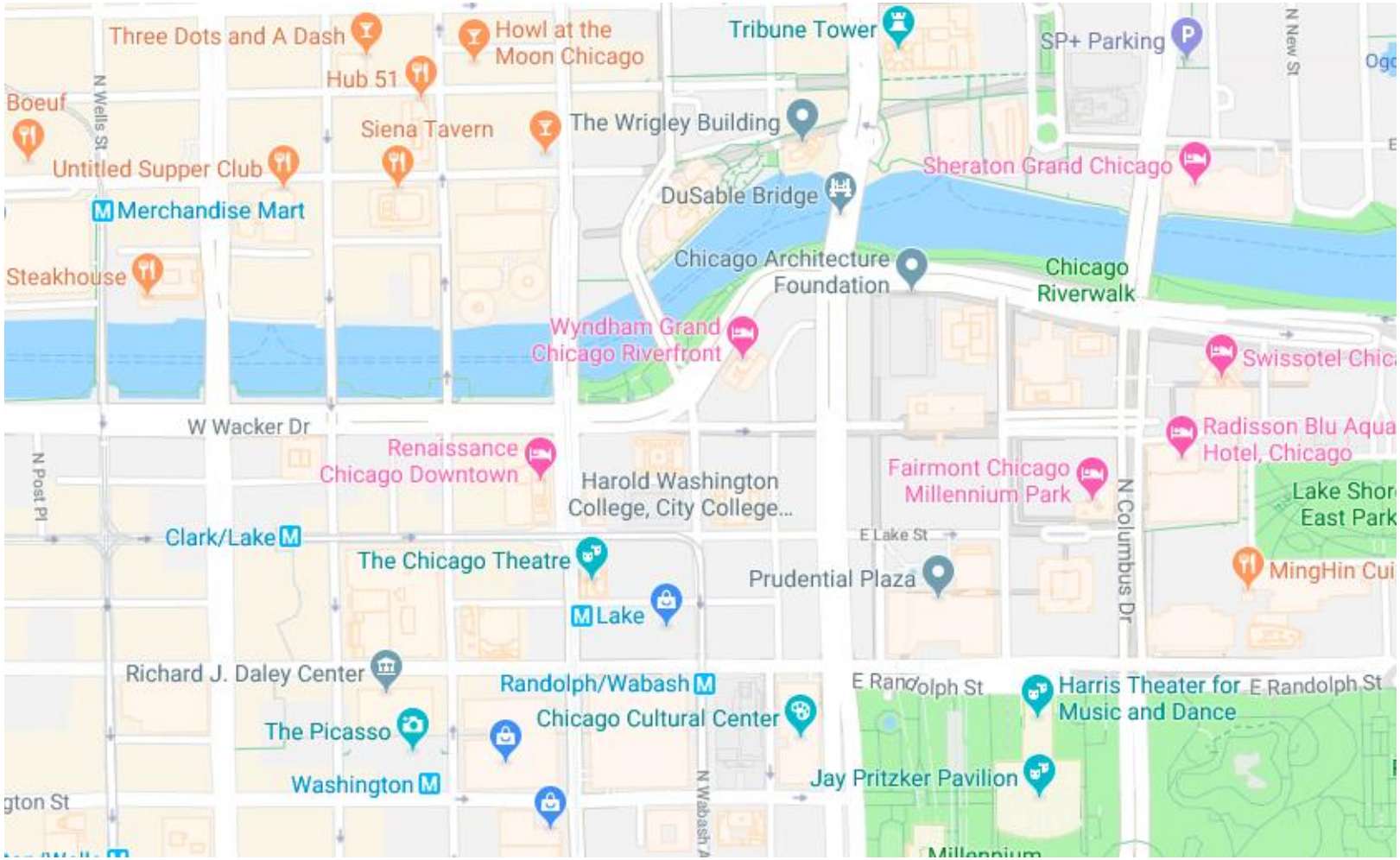
edge (2, 3) is an incoming edge for vertex 3

edge (3, 8) is an outgoing edge for vertex 3



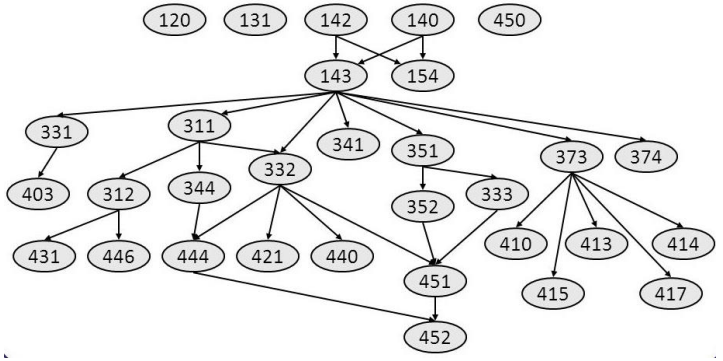
# Directed Graphs

Node = intersection, edge = one-way street



# Precedence Constraints

In a directed graph, an edge $(i, j)$ means task $i$ must occur before task $j$.




Applications

- Course prerequisite:

  course $i$ must be taken before $j$
- Compilation:

  must compile module $i$ before $j$
- Computing overflow:

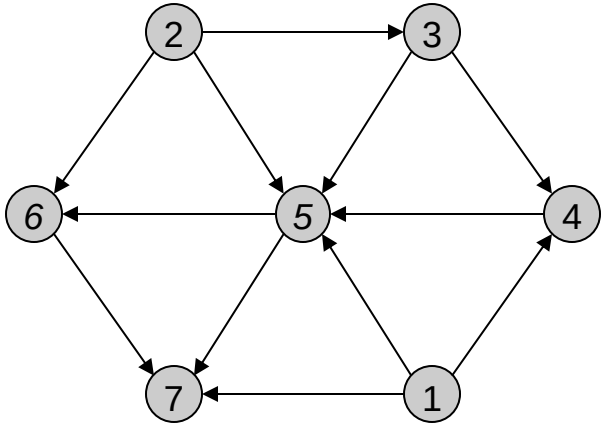
  output of job $i$ is part of input to job $j$
- Manufacturing or assembly:

  sand it before paint it

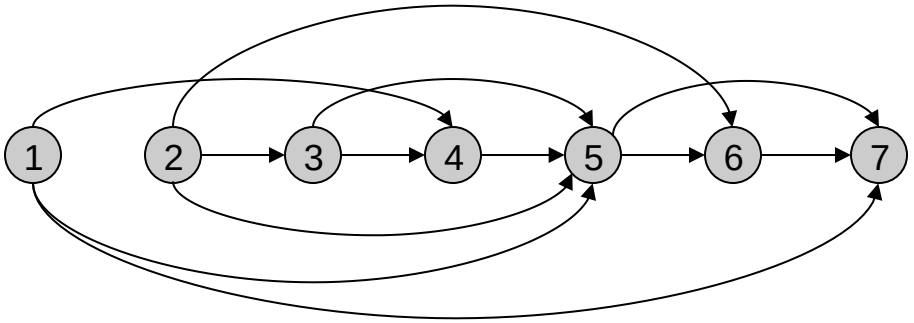
# Directed Acyclic Graphs (DAG)

Def: A DAG is a directed acyclic graph, i.e., one that contains no directed cycles.

Def: A topological order of a directed graph G = (V, E) is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge $(v_i, v_j)$ we have $i < j$.




*a DAG*




*a topological ordering of that DAG– all edges left-to-right*

# DAGs: A Sufficient Condition

Lemma: If $G$ has a topological order, then $G$ is a DAG.

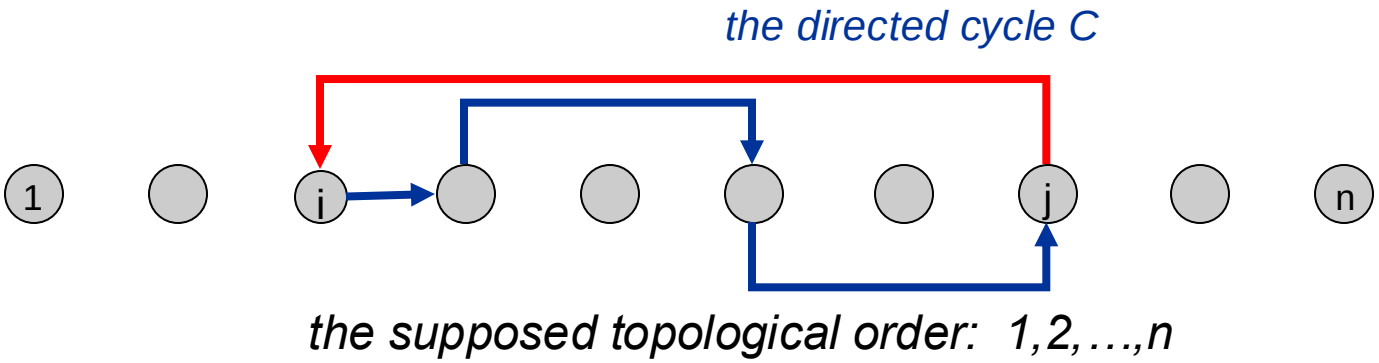
Proof. (by contradiction)

Suppose that $G$ has a topological order $1,2,\ldots,n$ and that $G$ also has a directed cycle $C$.

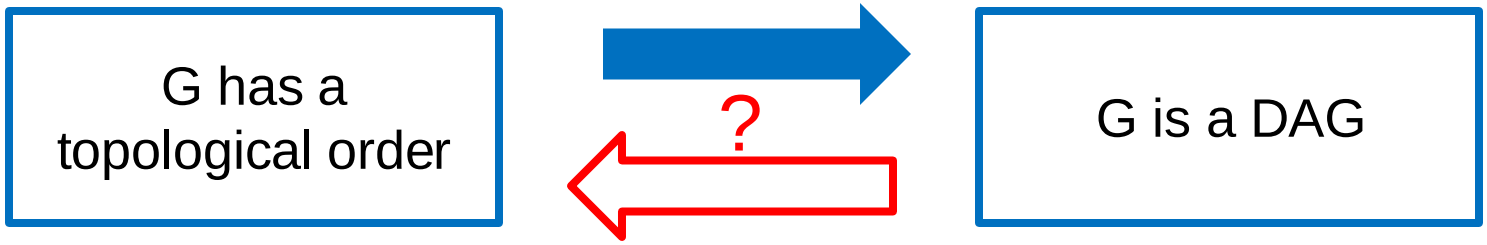
Let $i$ be the lowest-indexed node in $C$, and let $j$ be the node just before $i$; thus $(j,i)$ is an (directed) edge.

By our choice of $i$, we have $i < j$.

On the other hand, since $(j,i)$ is an edge and $1,\ldots,n$ is a topological order, we must have $j < i$, a contradiction



# DAGs: A Sufficient Condition



# Every DAG has a source node

Lemma: If $G$ is a DAG, then $G$ has a node with no incoming edges (i.e., a source).

Proof. (by contradiction)

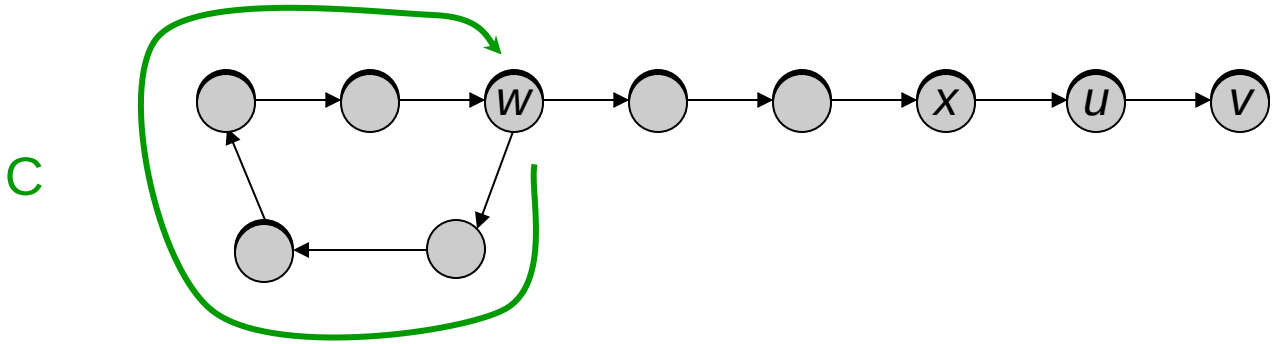
Suppose that $G$ is a DAG and it has no source

Pick any node $v$, and begin following edges backward from $v$. Since $v$ has at least one incoming edge $(u, v)$ we can walk backward to $u$.

Then, since $u$ has at least one incoming edge $(x, u)$, we can walk backward to $x$.

Repeat until we visit a node, say w, twice.

Let C be the sequence of nodes encountered between successive visits to w. C is a cycle.



# DAG => Topological Order

Lemma: If $G$ is a DAG, then $G$ has a topological order

Proof. (by induction on n)

Base case: true if $n = 1$.

Hypothesis: Every DAG with $n - 1$ vertices has a topological ordering.

Inductive Step: Given DAG with $n > 1$ nodes, find a source node $v$.

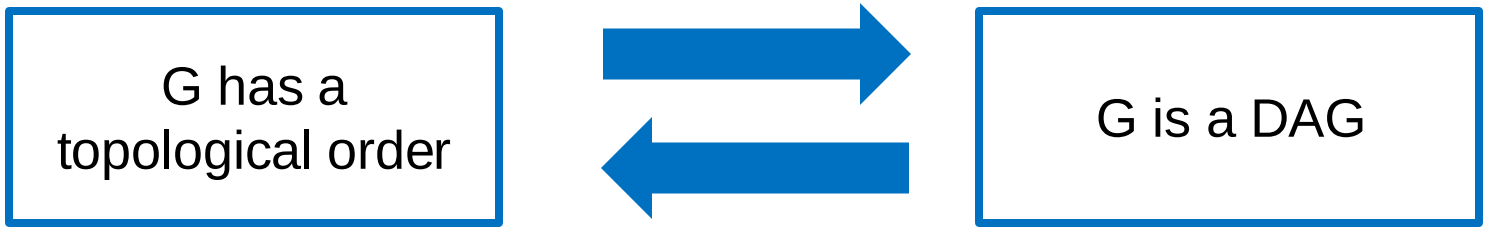
$G - \{ v \}$ is a DAG, since deleting $v$ cannot create cycles.

Reminder: Always remove vertices/edges to use hypothesis

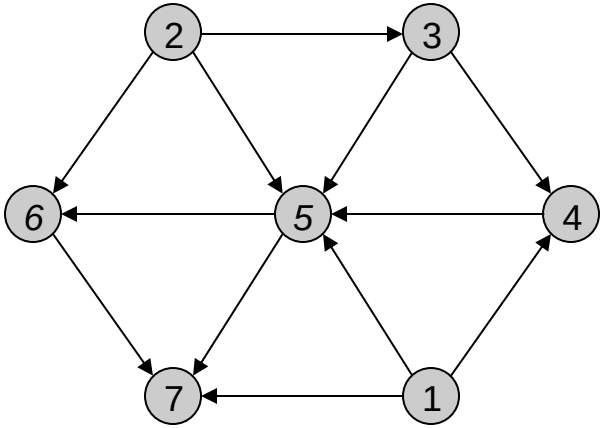
By hypothesis, $G - \{ v \}$ has a topological ordering.

Place $v$ first in topological ordering; then append nodes of $G - \{v\}$ in topological order. This is valid since $v$ has no incoming edges.

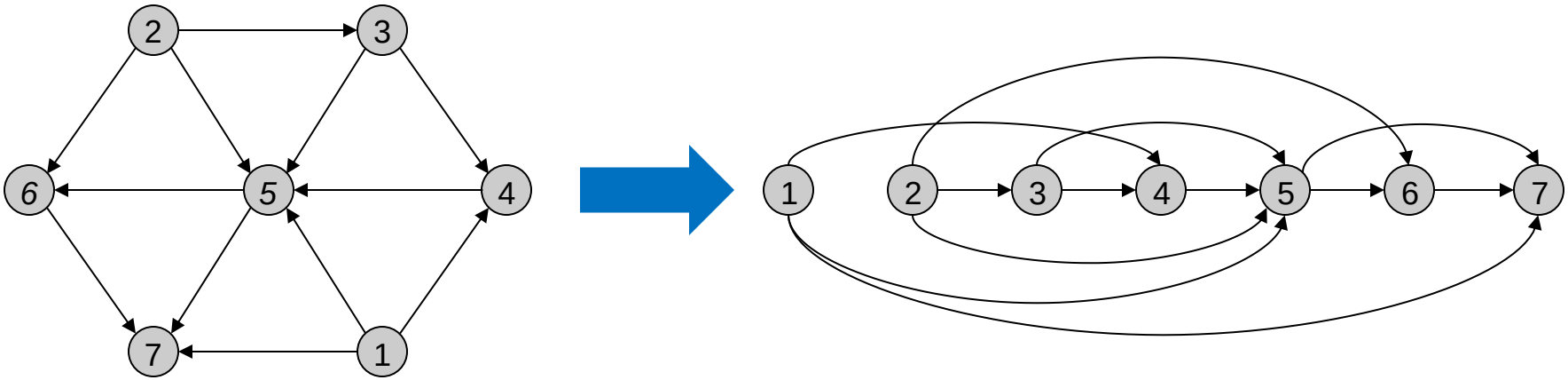
# A Characterization of DAGs



# Topological Order Algorithm 1:  Example



# Topological Order Algorithm 1: Example




*Topological order: 1, 2, 3, 4, 5, 6, 7*

Running time: O(n+m)

- Adjacency list
- Maintain # outgoing edge for each node