EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF
HUMANITIES
Department of General and Computational Linguistics

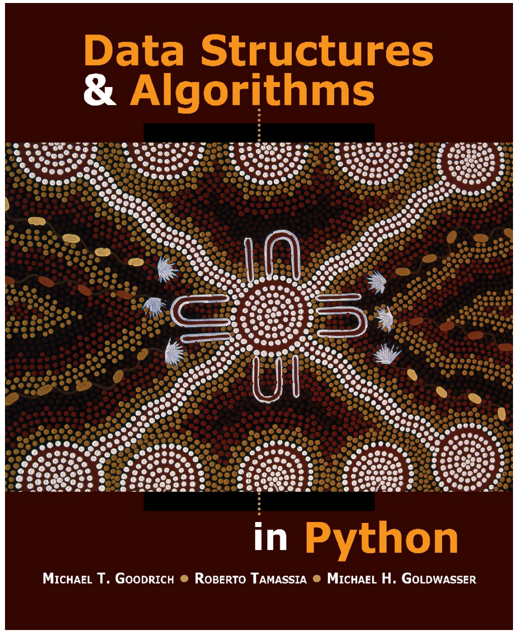
Graph Traversals

Data Structures and Algorithms for CL III, WS 2019-2020

Corina Dima
corina.dima@uni-tuebingen.de

Data Structures & Algorithms in Python

MICHAEL GOODRICH
ROBERTO TAMASSIA
MICHAEL GOLDWASSER



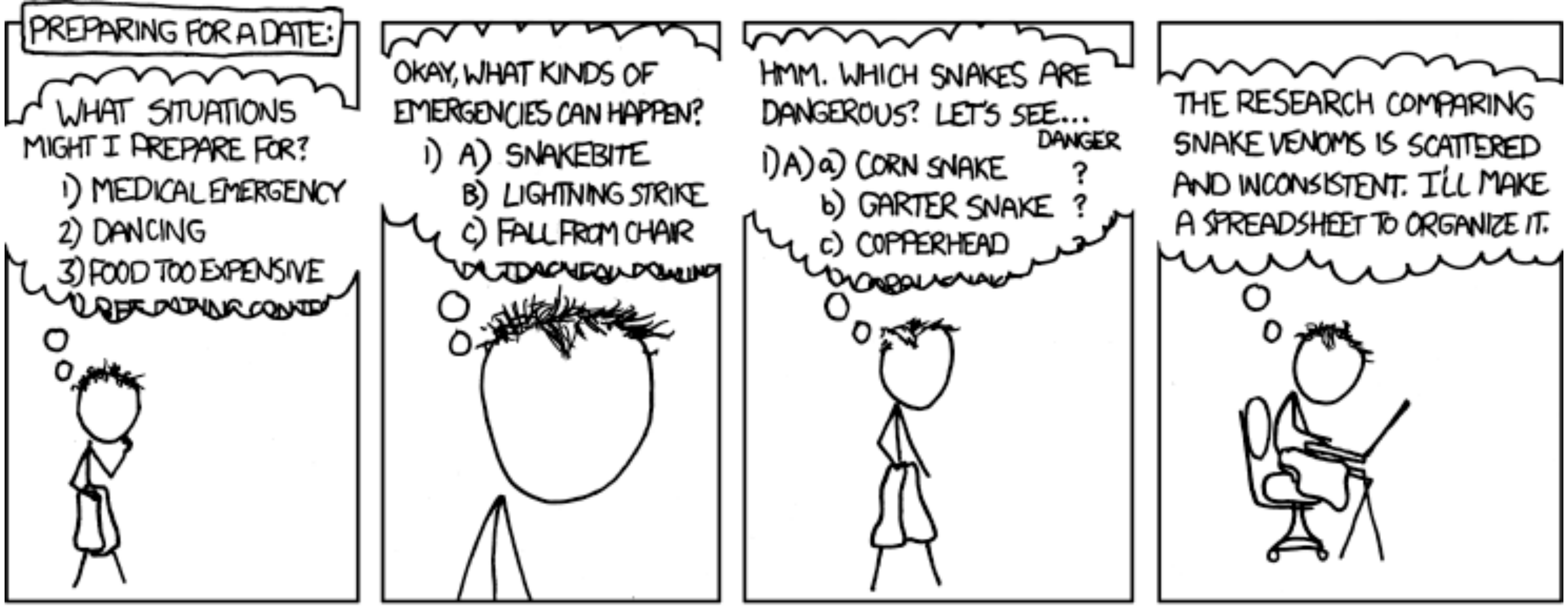
14.3 Graph Traversals

- Depth-First Search
- DFS Implementation and Extensions
- Breadth-First Search

Graph Traversals

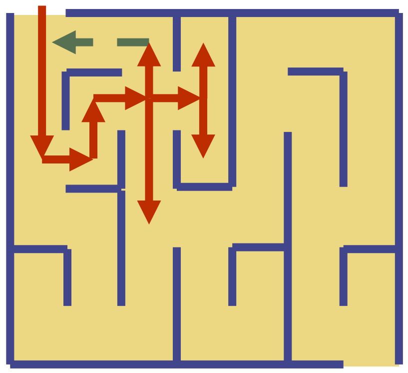
- Formally, a traversal is a systematic procedure for exploring a graph by examining all of its vertices and edges
- A traversal is efficient if it visits all the vertices and edges in time proportional to their number – i.e. in linear time
- Graph traversal algorithms can answer many questions involving reachability in an undirected graph G:
  - Compute a path from a vertex u to a vertex v, or report that such a path does not exist
  - Given a start vertex s from G, compute, for every vertex v of G, a path with minimum number of edges between s and v, or report that no such path exists
  - Test whether G is a connected graph
  - Compute a spanning tree of G, if G is connected
  - Compute the connected components of G
  - Compute a cycle in G, or report that G has no cycles

Depth-First Search



https://xkcd.com/761/
https://www.explainxkcd.com/wiki/index.php/761:_DFS

Depth-First Search: Intuition



- Imagine wandering through a labyrinth with a string and a can of paint without getting lost; each intersection is a vertex
- We begin with a specific starting vertex s of G, and initialize it by tying the string and paining s as $visited$ – s is now our current vertex, call it u
- Traverse G by considering an arbitrary edge (u, v) incident to the current vertex u
  - If (u, v) leads to a $visited$ vertex v (v is painted), then ignore edge
  - If (u, v) leads to an unvisited vertex v, then unroll the string, go to v, paint v as visited, make it the current vertex, and continue the process with the edges incident to v
- Will eventually hit a dead end: a current vertex v where all the incident edges lead to visited vertices; then roll string back up, backtrack to the edge that brought us to v – go back to u, and continue with visiting u
- Finish when backtracking leads back to s and there are no more edges of s to explore

Depth-First Search - Edges

- The DFS traversal identifies the depth-first search tree rooted at the starting vertex s
- Whenever an edge $e = (u, v)$ is used to discover a new vertex during the execution of DFS, the edge is known as a discovery edge or a tree edge
- All other edges from the DFS traversal are called nontree edges, which lead to already visited vertices
- In an undirected graph *explored nontree edges* connect the current vertex to one of its ancestors in the DFS tree – they are called back edges.

Depth-First Search - Algorithm

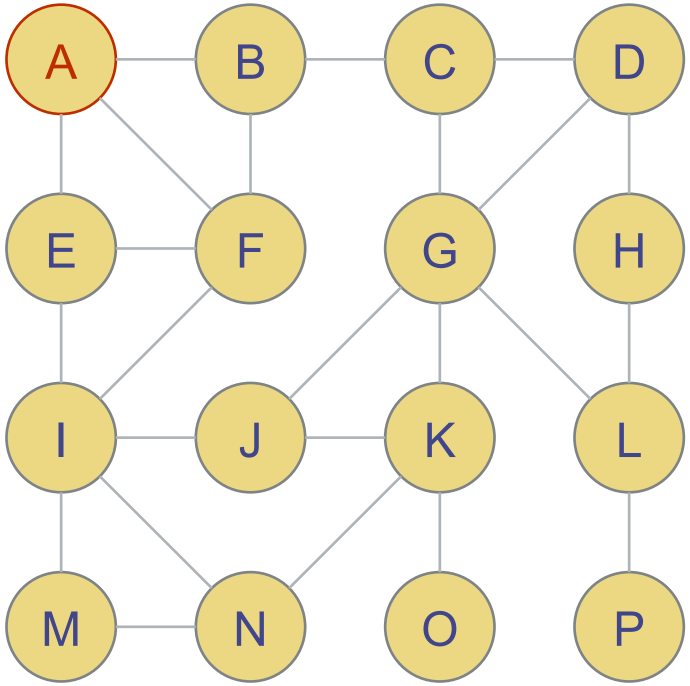
Algorithm DFS(G,u): {We assume u has already been marked as visited}

Input: A graph G and a vertex u of G

Output: A collection of vertices reachable from u, with their discovery edges

```
for each outgoing edge e = (u,v) of u do
    if vertex v has not been visited then
        Mark vertex v as visited (via edge e).
        Recursively call DFS(G,v).
```

DFS in an Undirected Graph - Example

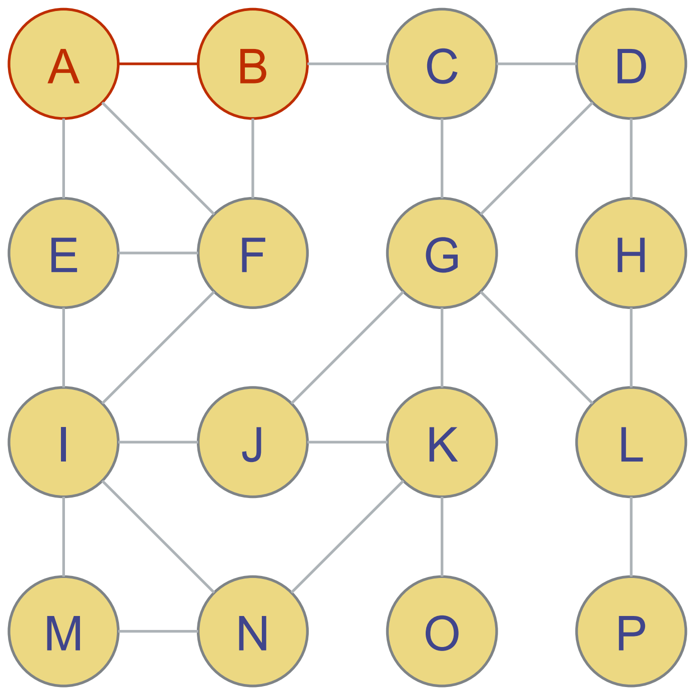


visited	discovery edge
A	None

Current vertex: A
Edges to consider: to B, E, F

- Start from vertex A, which is marked as visited (red)
- Assume that the edges adjacent to a vertex are considered in alphabetical order – e.g. for A: B, E, F

DFS in an Undirected Graph - Example

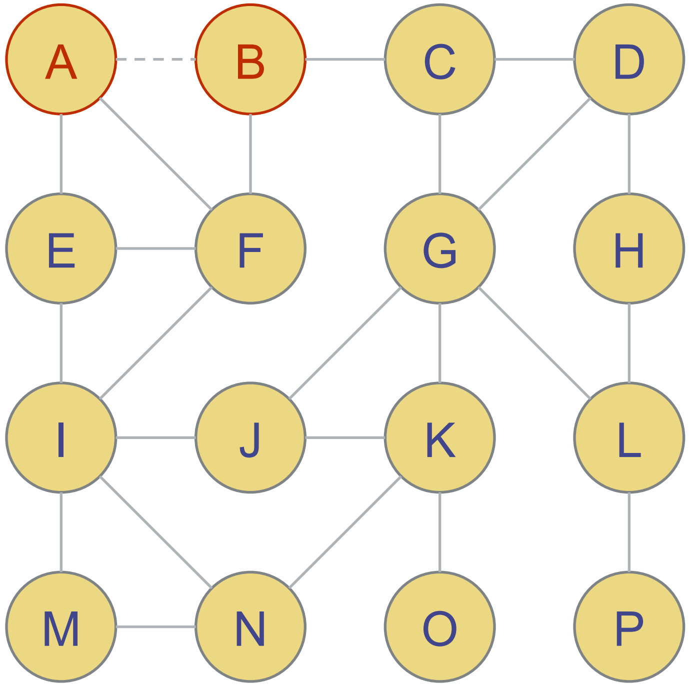


- Consider the edge that leads to B
- B is not visited – mark B as visited
- Mark (A,B) as a discovery edge
- Make B the current vertex

visited	discovery edge
A	None
B	(A,B)

Current vertex: A
Edges to consider: to E, F

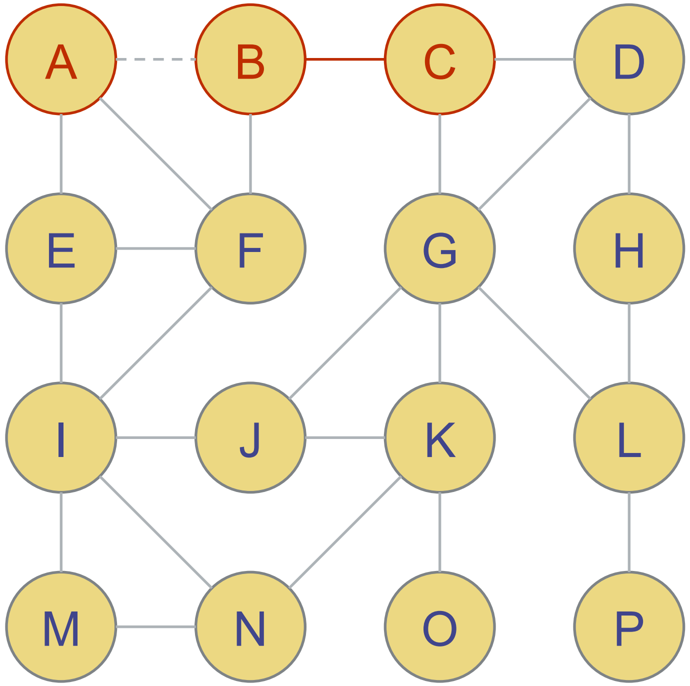
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)

Current vertex: B
Edges to consider: to A, C, F

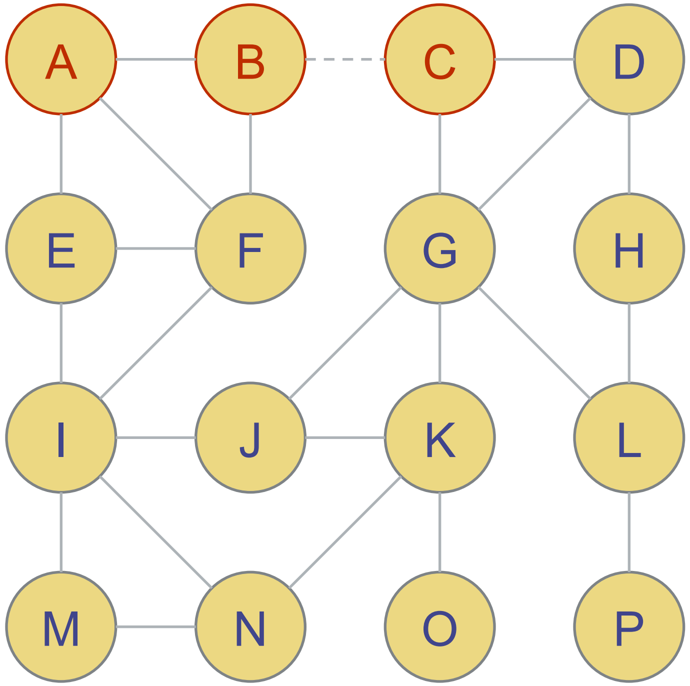
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)

Current vertex: B
Edges to consider: to C, F

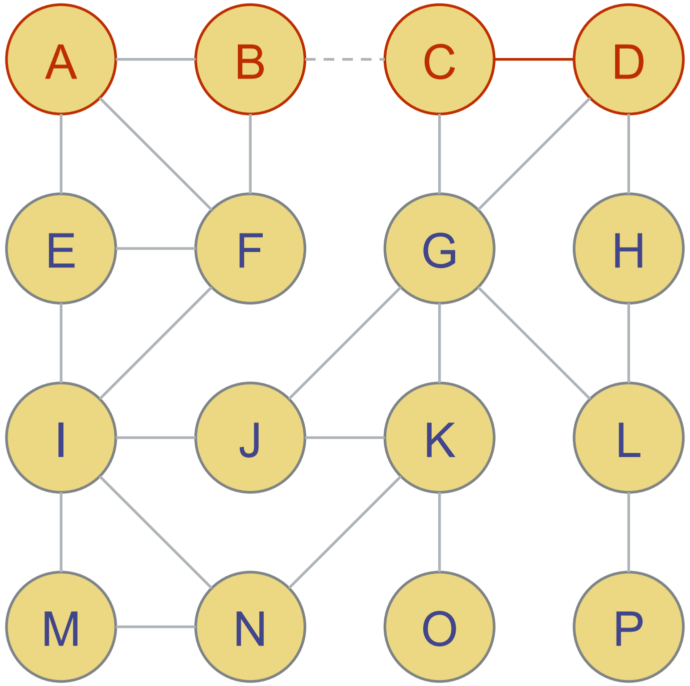
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)

Current vertex: C
Edges to consider: to B, D, G

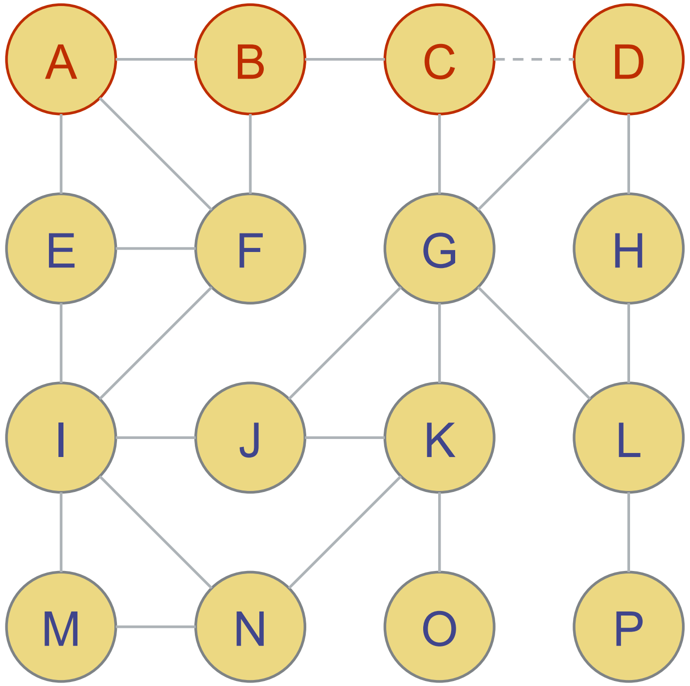
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)

Current vertex: C
Edges to consider: to D, G

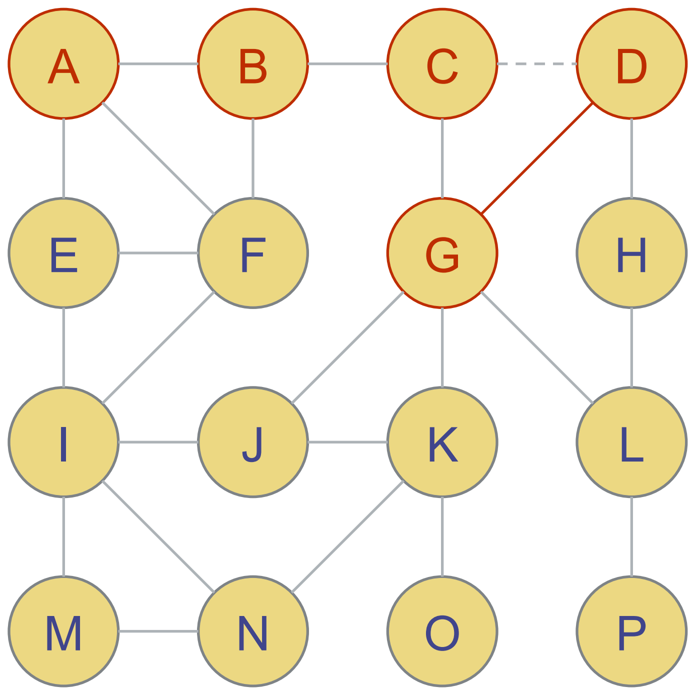
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)

Current vertex: D
Edges to consider: to C, G, H

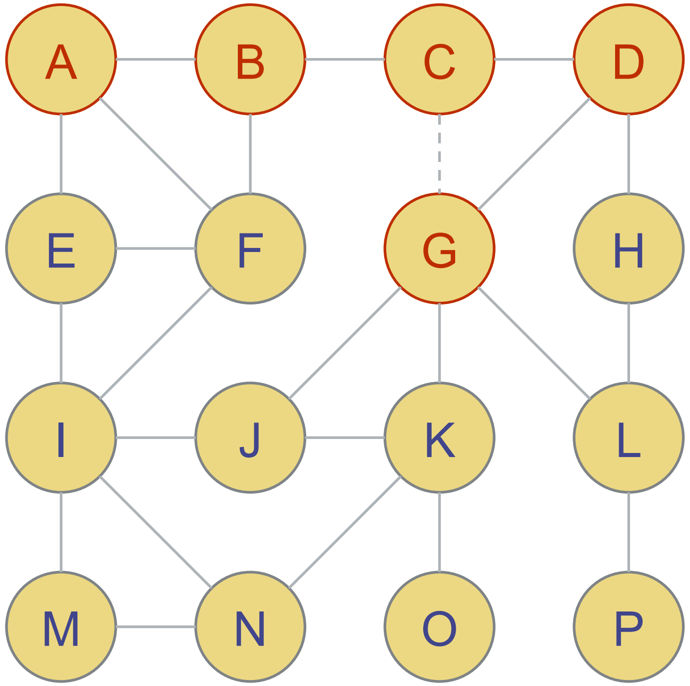
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)

Current vertex: D
Edges to consider: to G, H

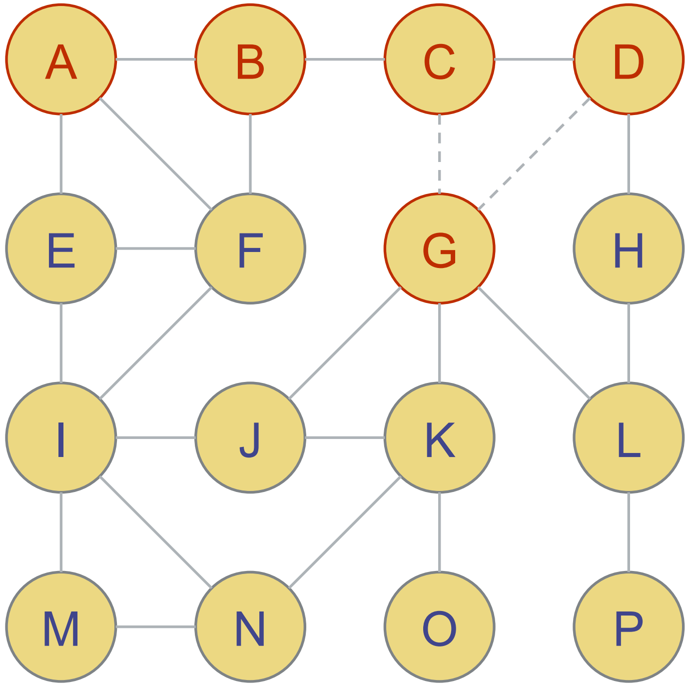
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)

Current vertex: G
Edges to consider: to C, D, J, K, L

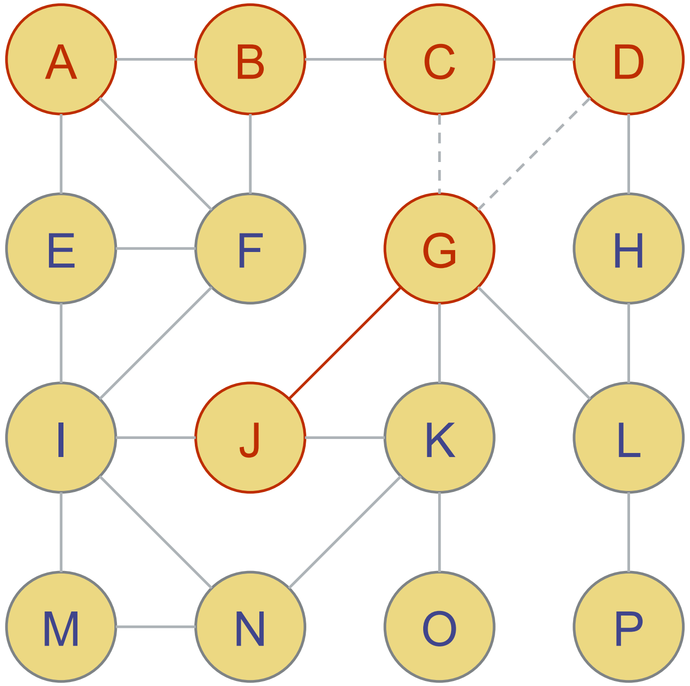
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)

Current vertex: G
Edges to consider: to D, J, K, L

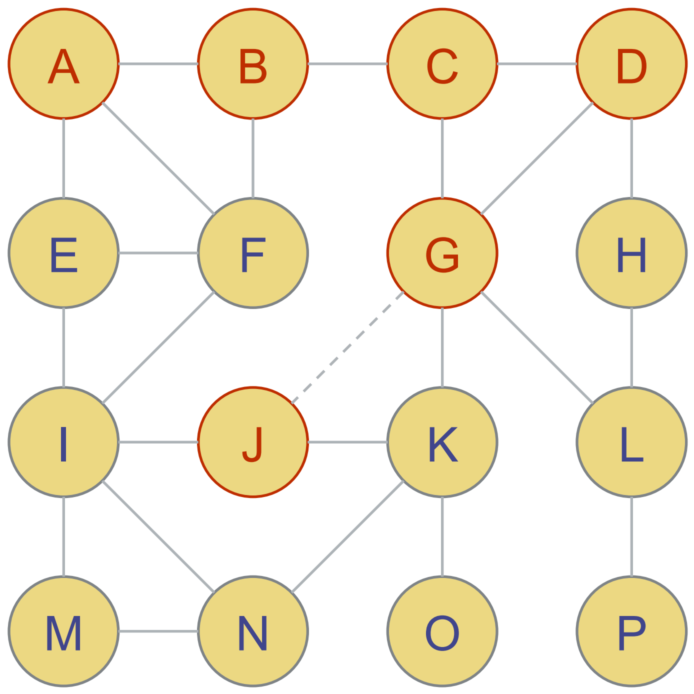
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)

Current vertex: G
Edges to consider: to J, K, L

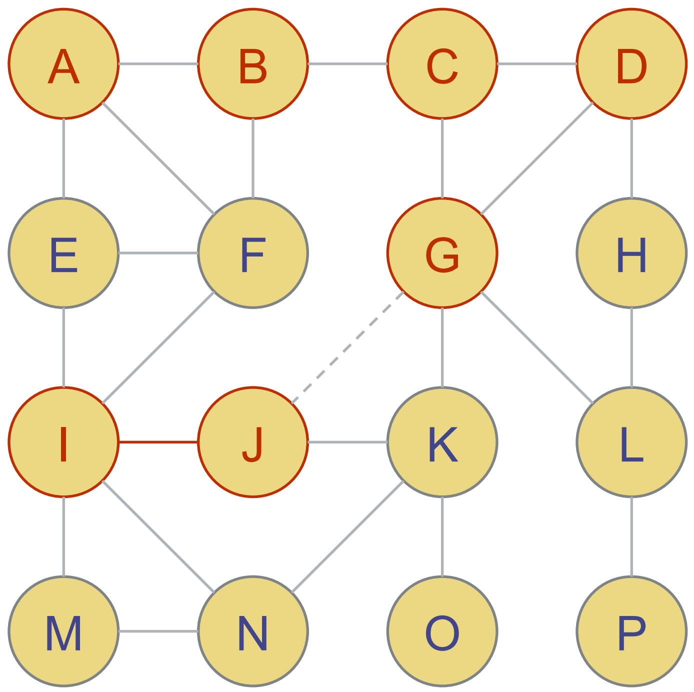
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)

Current vertex: J
Edges to consider: to G, I, K

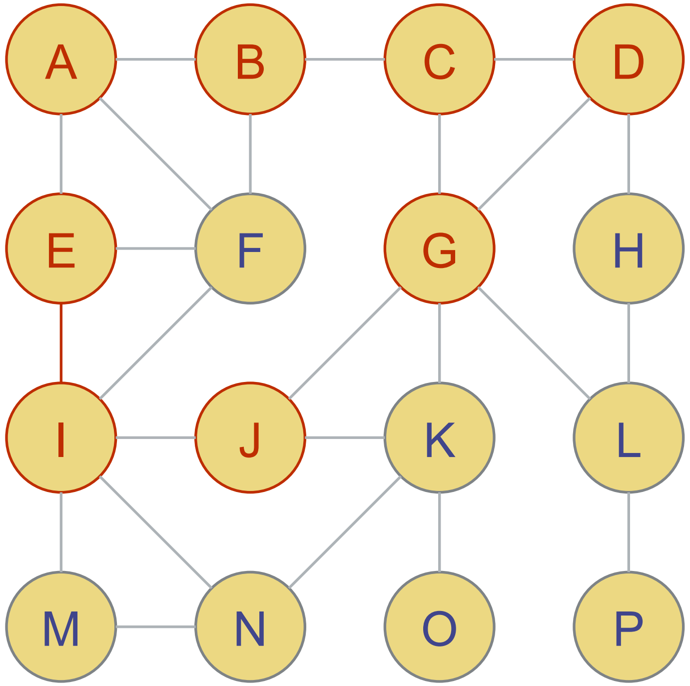
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)

Current vertex: J
Edges to consider: to I, K

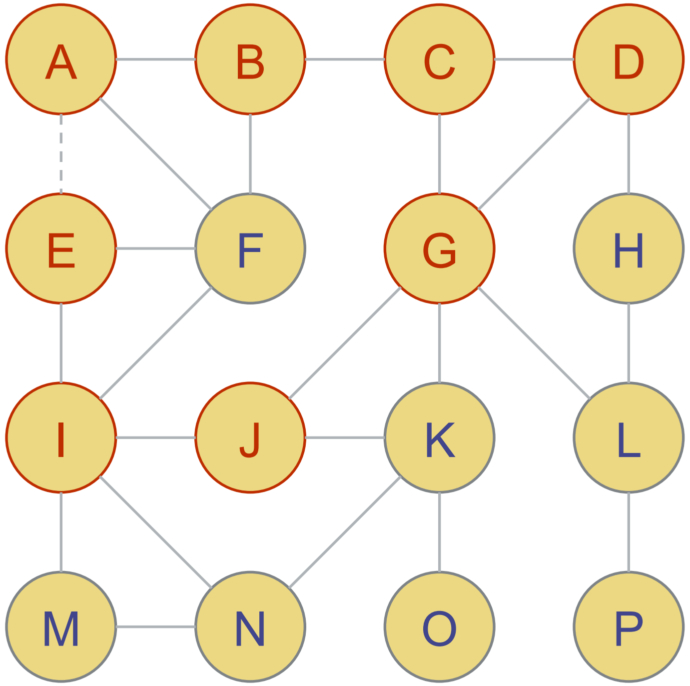
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)

Current vertex: I
Edges to consider: to E,F,J,M,N

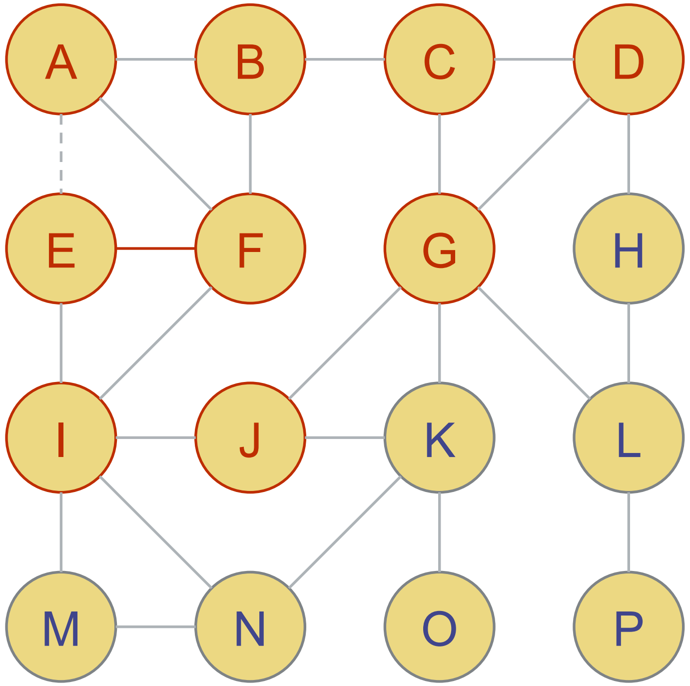
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)

Current vertex: E
Edges to consider: to A, F, I

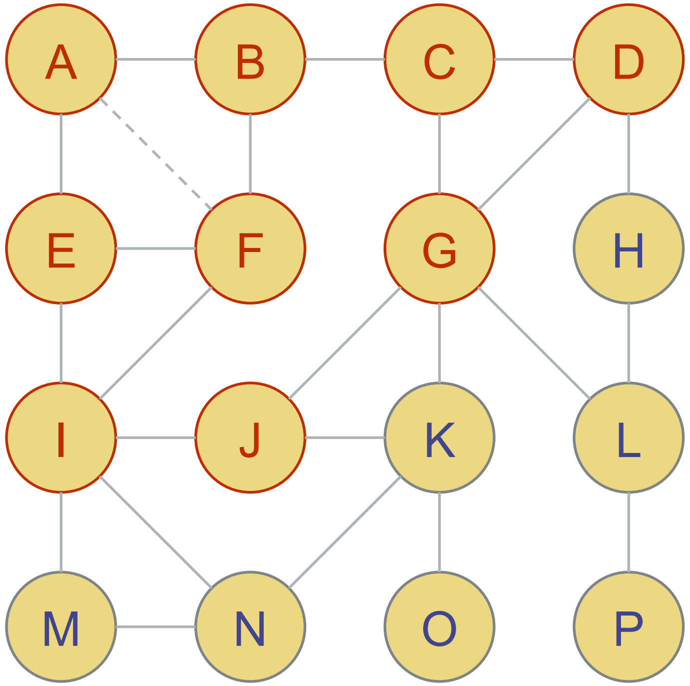
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Current vertex: E
Edges to consider: to F, I

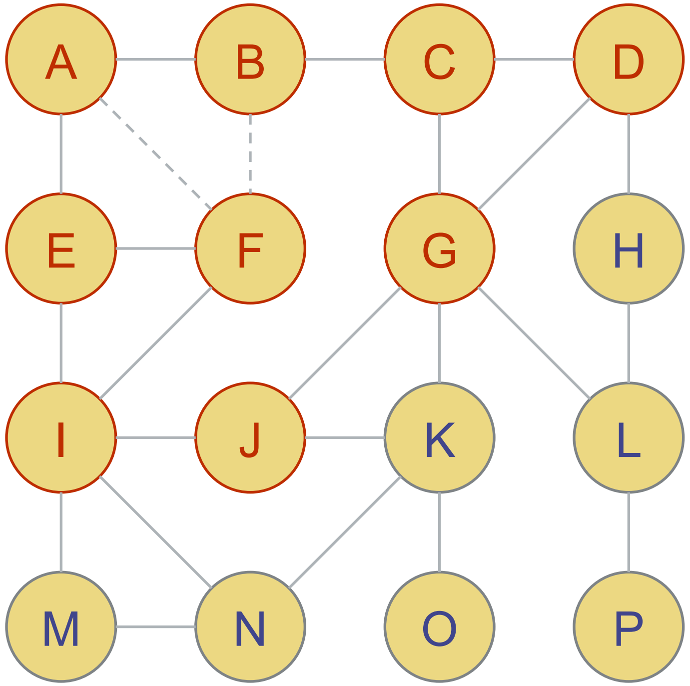
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Current vertex: F
Edges to consider: to A, B, E, I

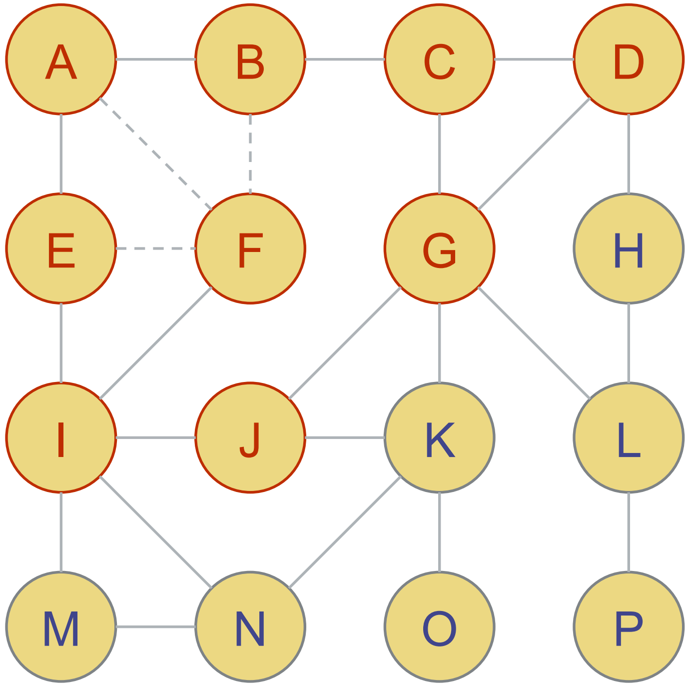
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Current vertex: F
Edges to consider: to B, E, I

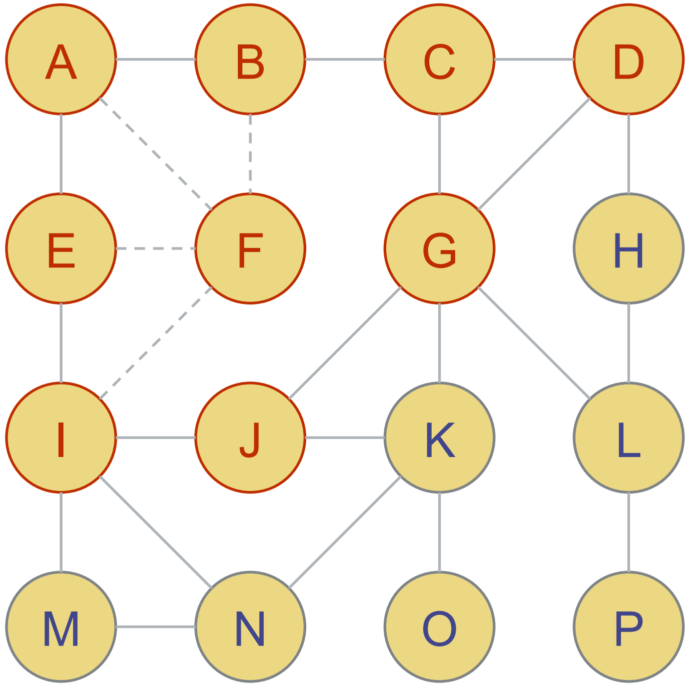
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Current vertex: F
Edges to consider: to E, I

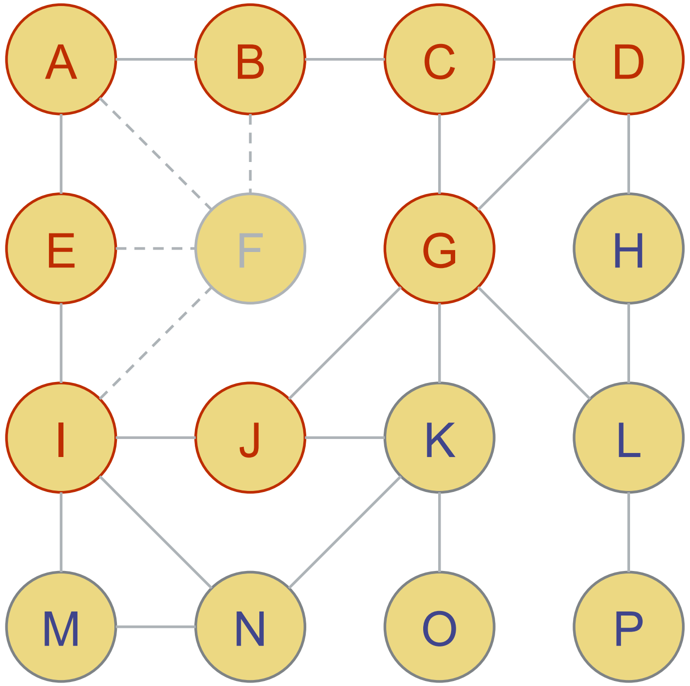
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Current vertex: F
Edges to consider: to I

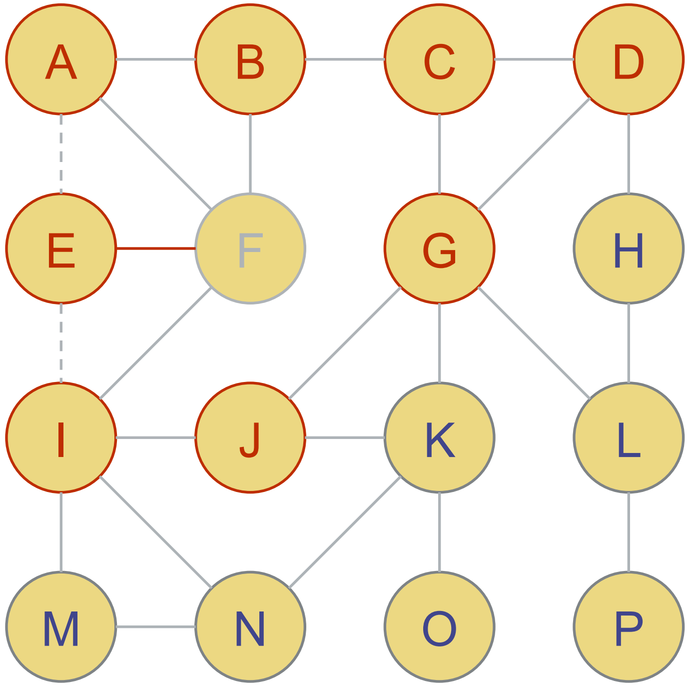
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Finished F (gray), backtracking to E

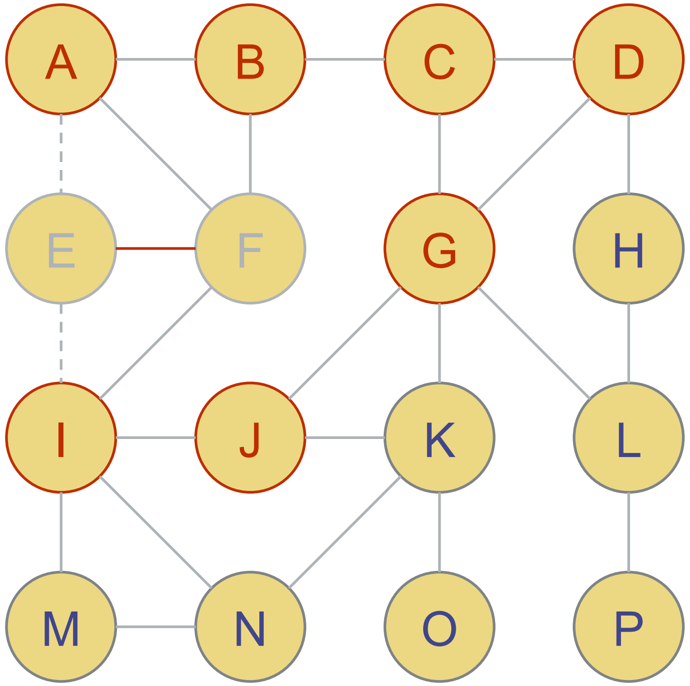
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Current vertex: E
Edges to consider: to I

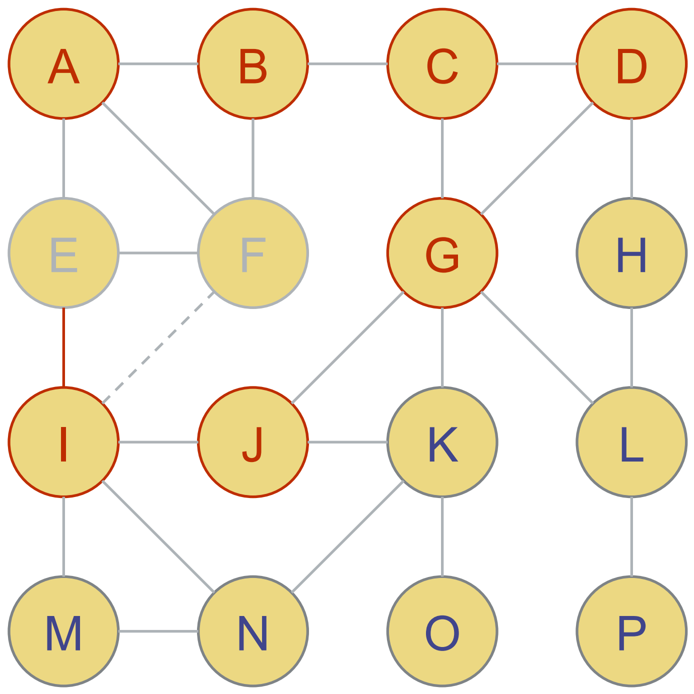
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Finished E, backtracking to I

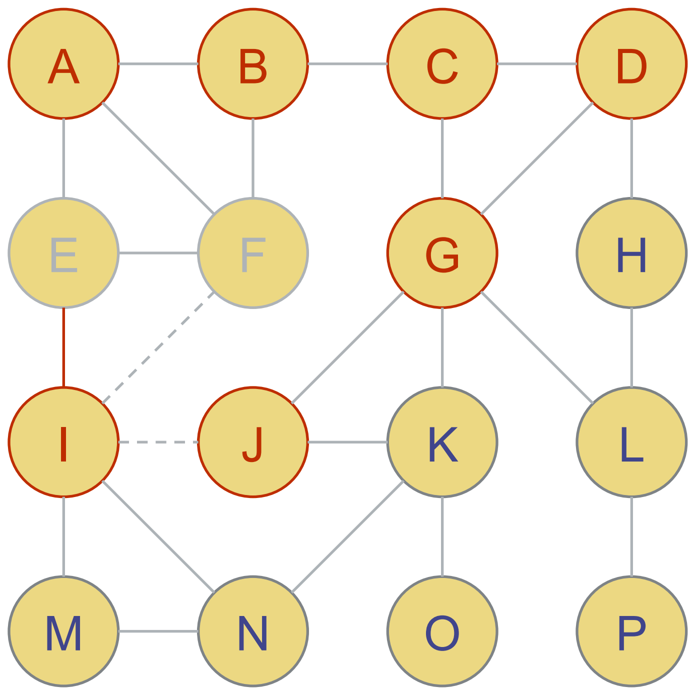
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Current vertex: I
Edges to consider: to F,J,M,N

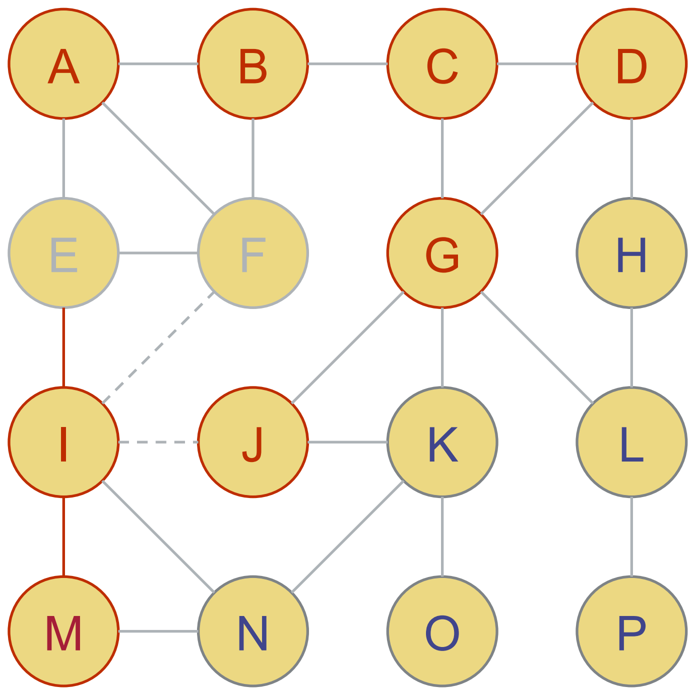
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)

Current vertex: I
Edges to consider: to J,M,N

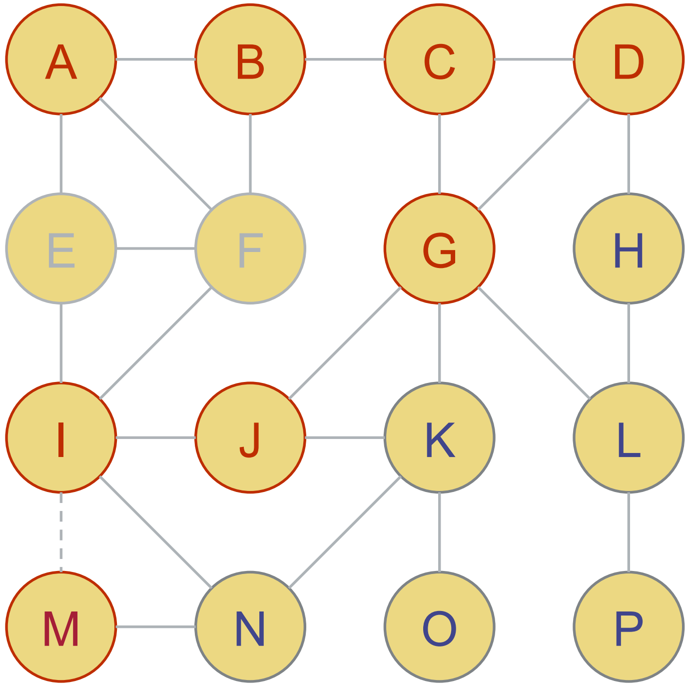
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)

Current vertex: I
Edges to consider: to M,N

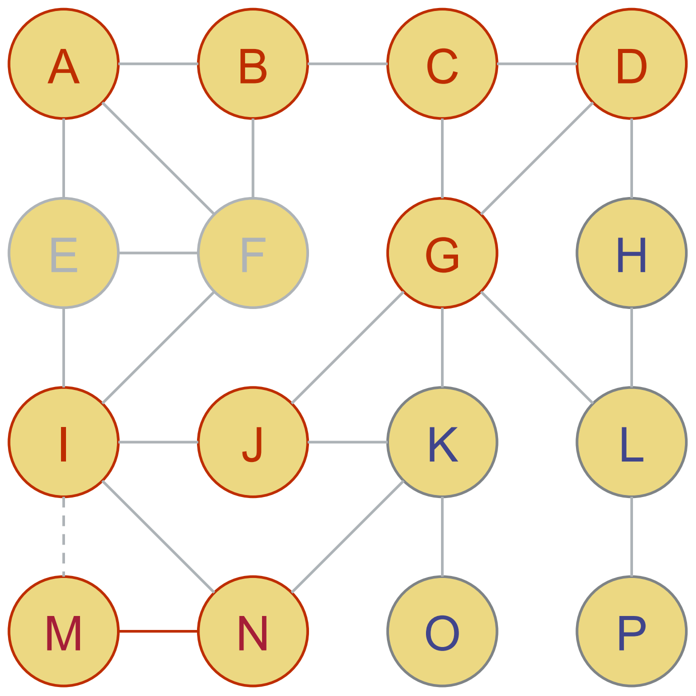
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)

Current vertex: M
Edges to consider: to I, N

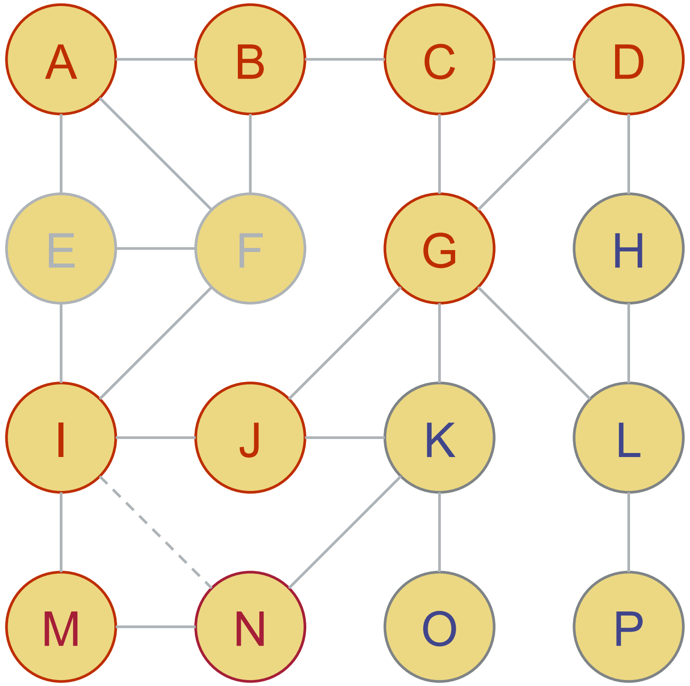
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)

Current vertex: M
Edges to consider: to N

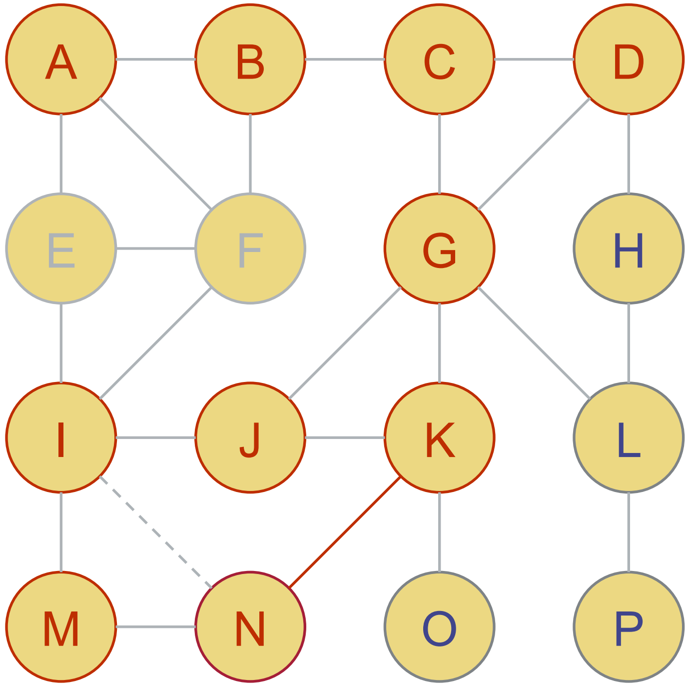
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)

Current vertex: N
Edges to consider: to I, K, M

DFS in an Undirected Graph - Example

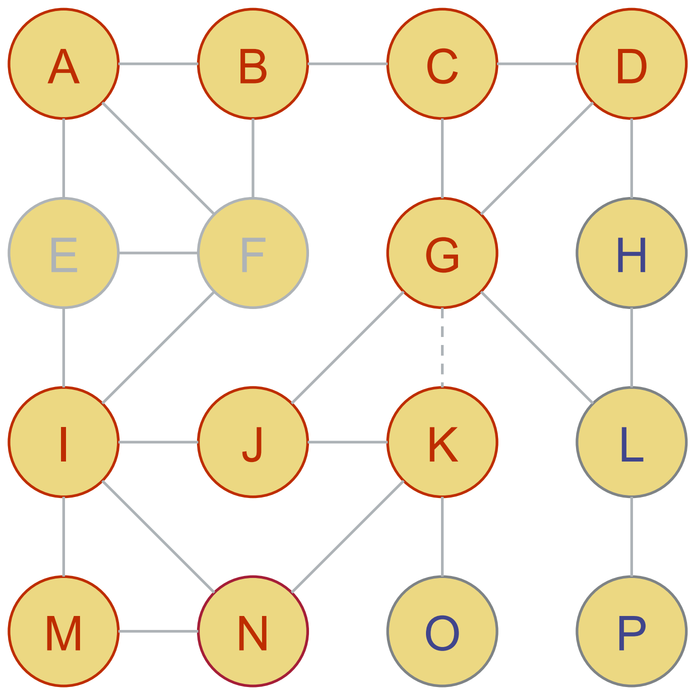


visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)

Current vertex: N
Edges to consider: to K, M

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

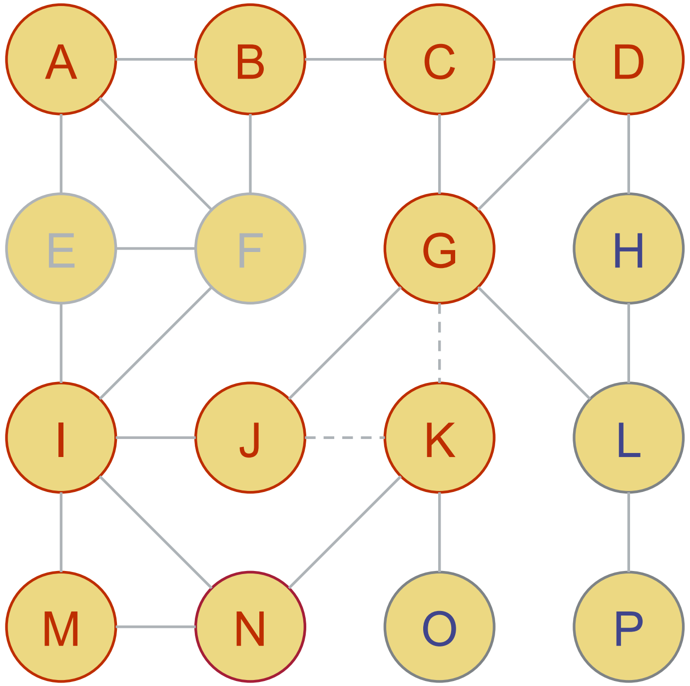
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)

Current vertex: K
Edges to consider: to G, J, N, O

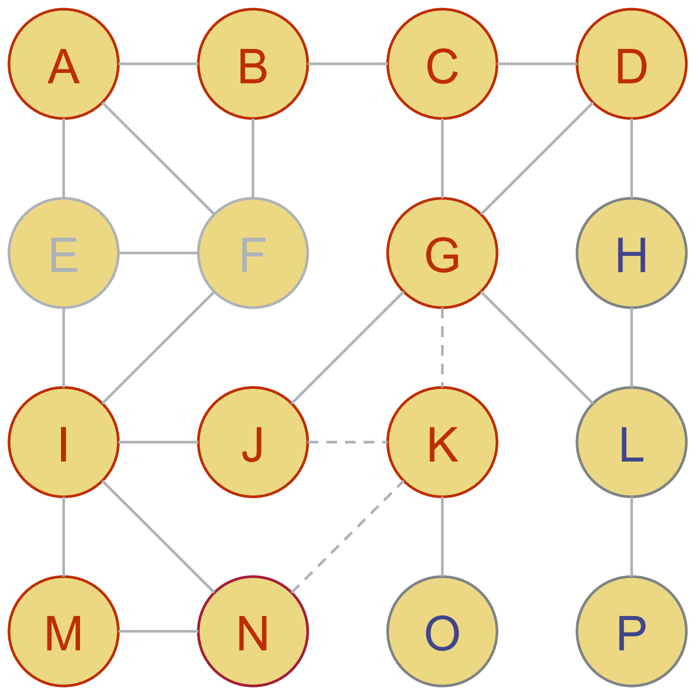
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)

Current vertex: K
Edges to consider: to J, N, O

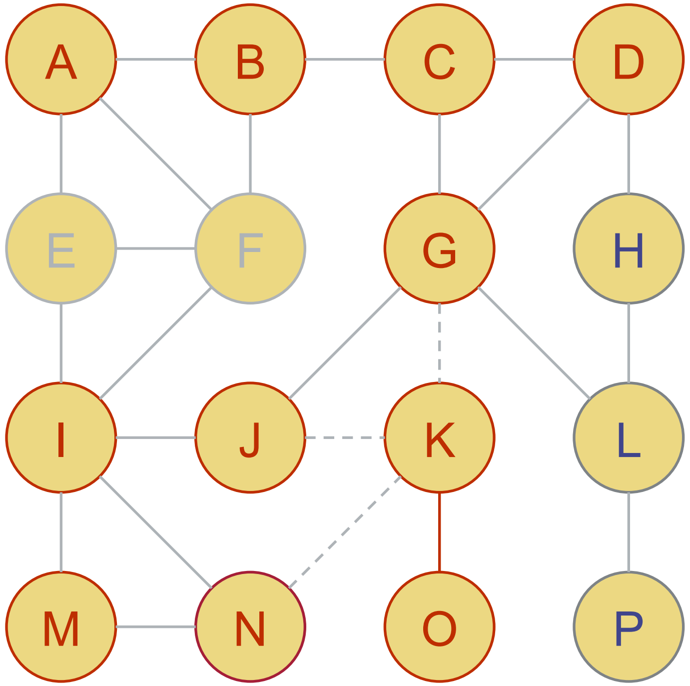
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)

Current vertex: K
Edges to consider: to N, O

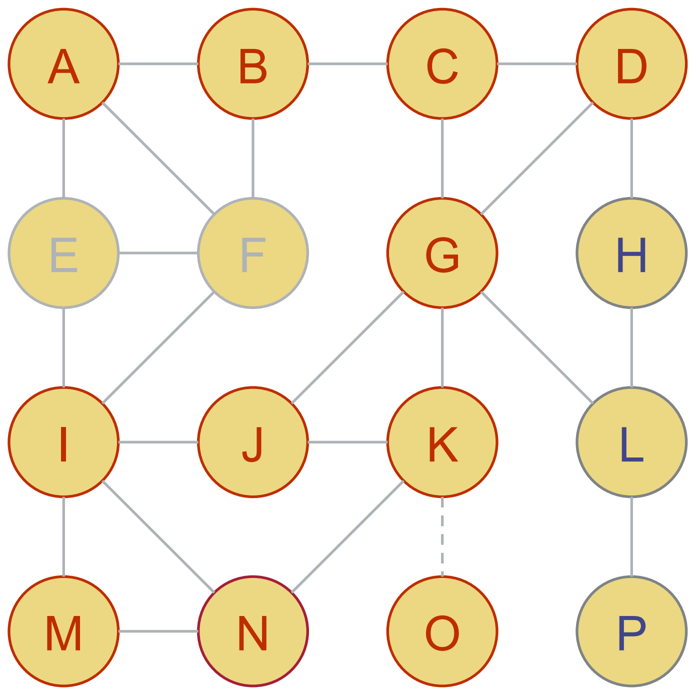
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: K
Edges to consider: to O

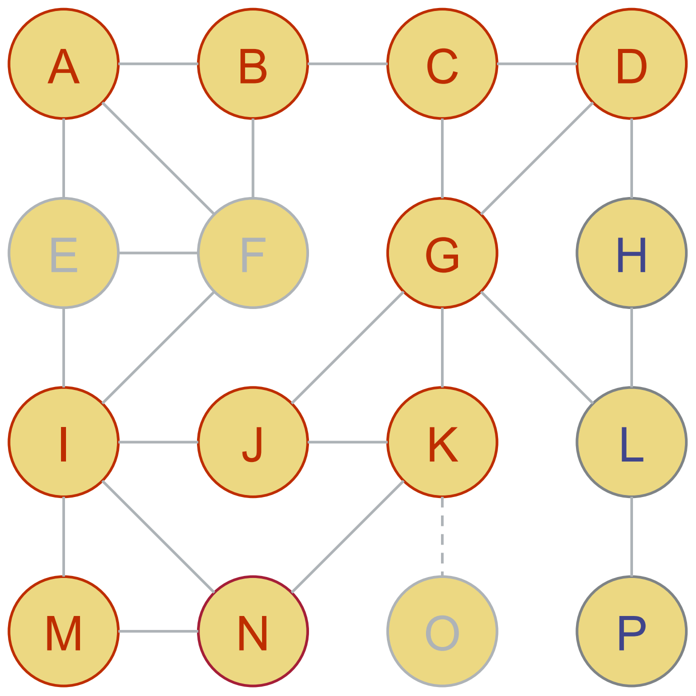
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: O
Edges to consider: to K

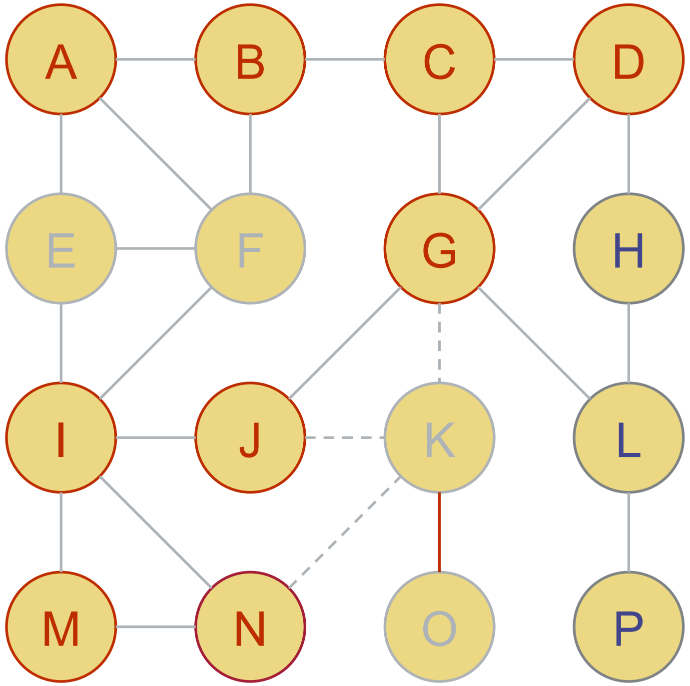
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Finished O, backtracking to K

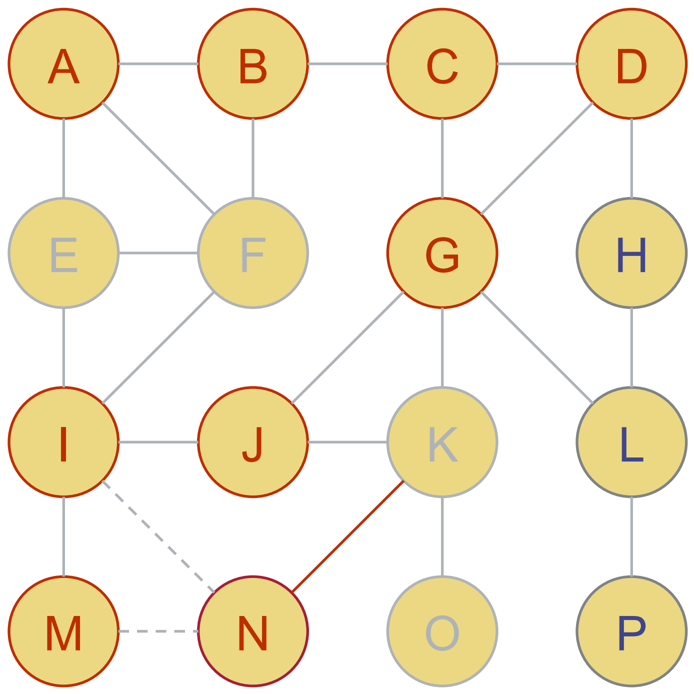
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: K
Edges to consider: -
Finished K, backtracking to N

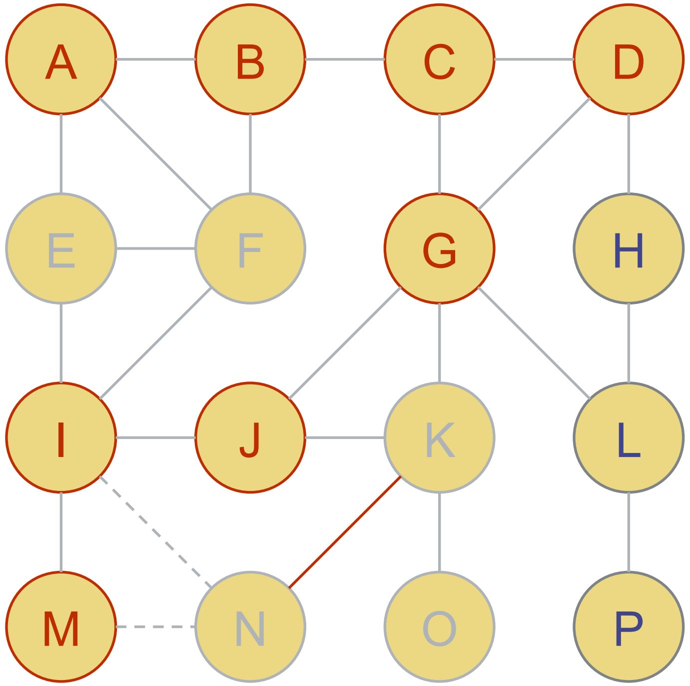
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: N
Edges to consider: M

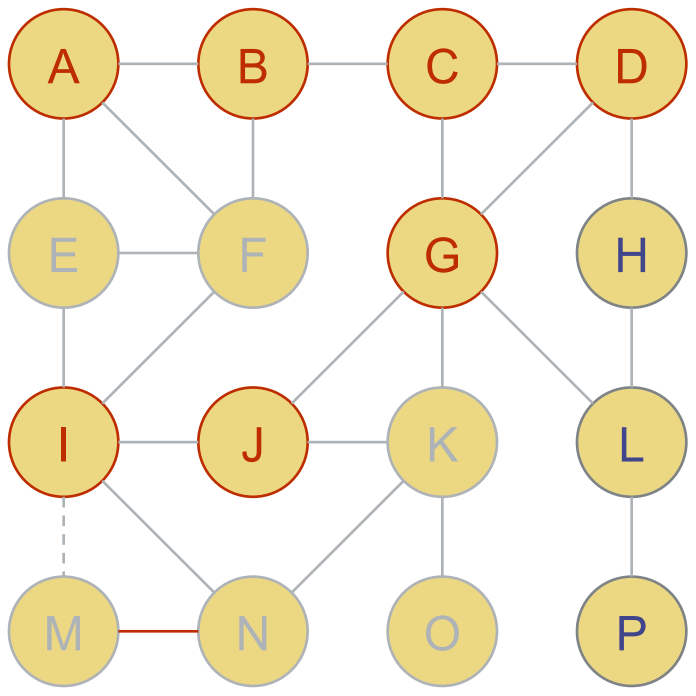
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: N
Edges to consider: -
Finished N, backtracking to M

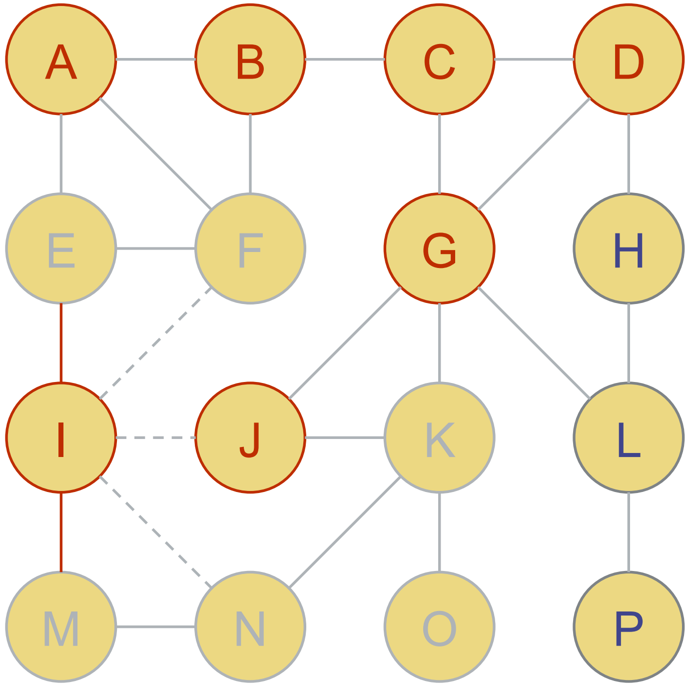
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: M
Edges to consider: -
Finished M, backtracking to I

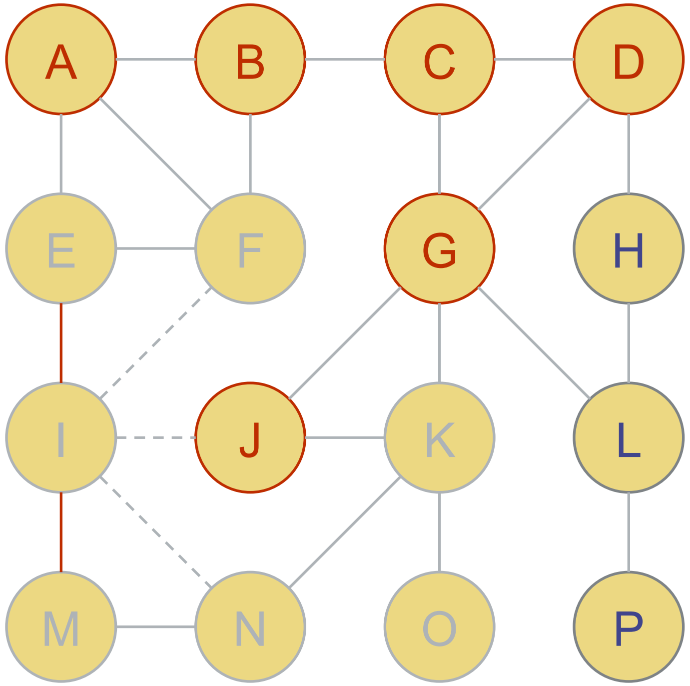
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: I
Edges to consider: N

DFS in an Undirected Graph - Example

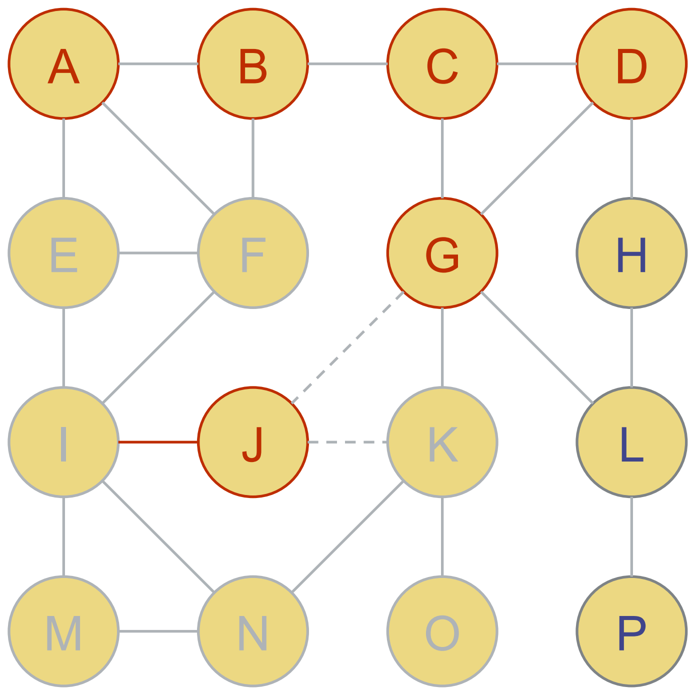


visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: I
Edges to consider: -
Finished I, backtracking to J

UNIVERSITÄT TÜBINGEN

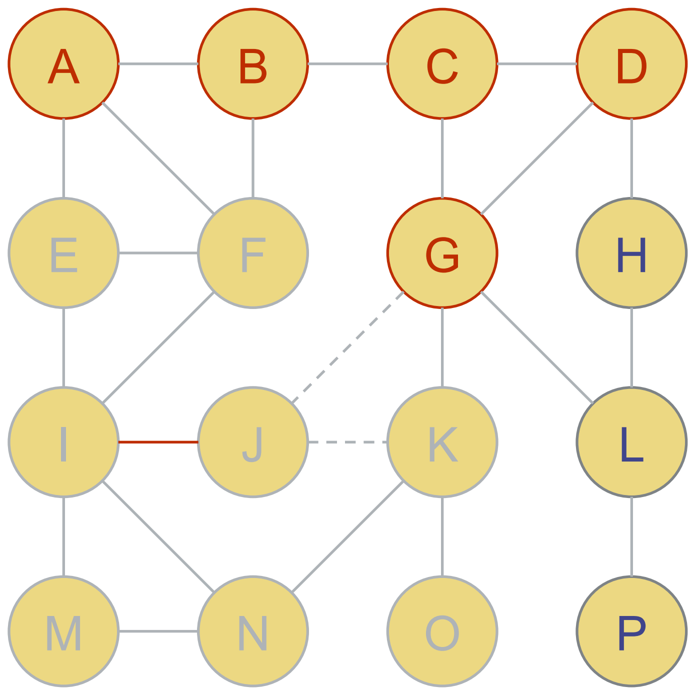
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: J
Edges to consider: to K

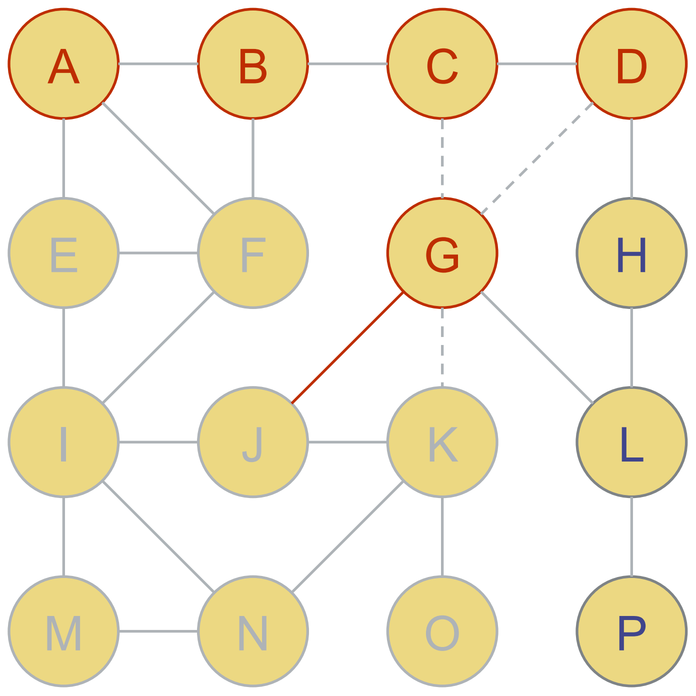
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: J
Edges to consider: -
Finished J, backtracking to G

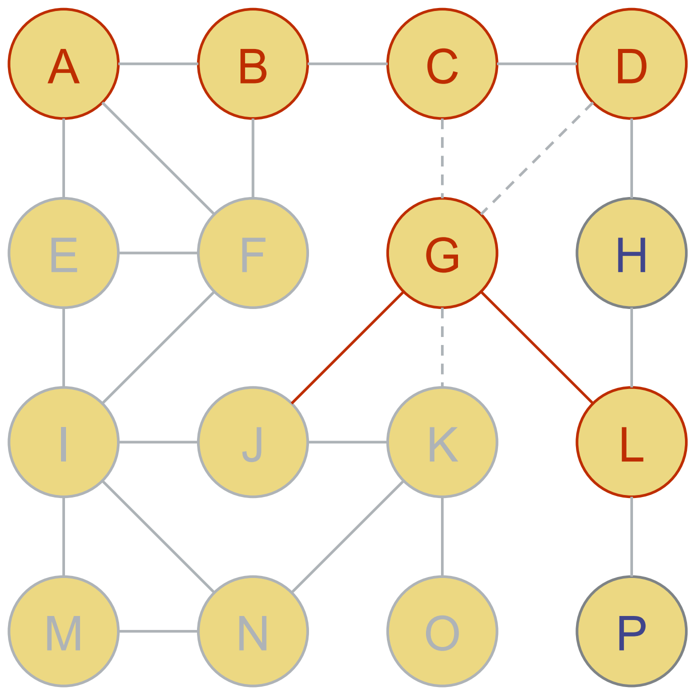
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)

Current vertex: G
Edges to consider: to K, L

DFS in an Undirected Graph - Example

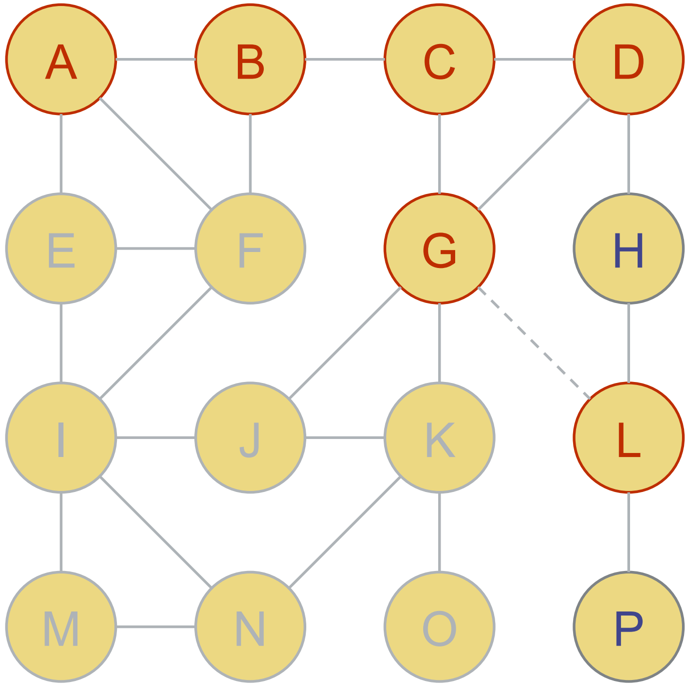


visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)

Current vertex: G
Edges to consider: to L

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

DFS in an Undirected Graph - Example



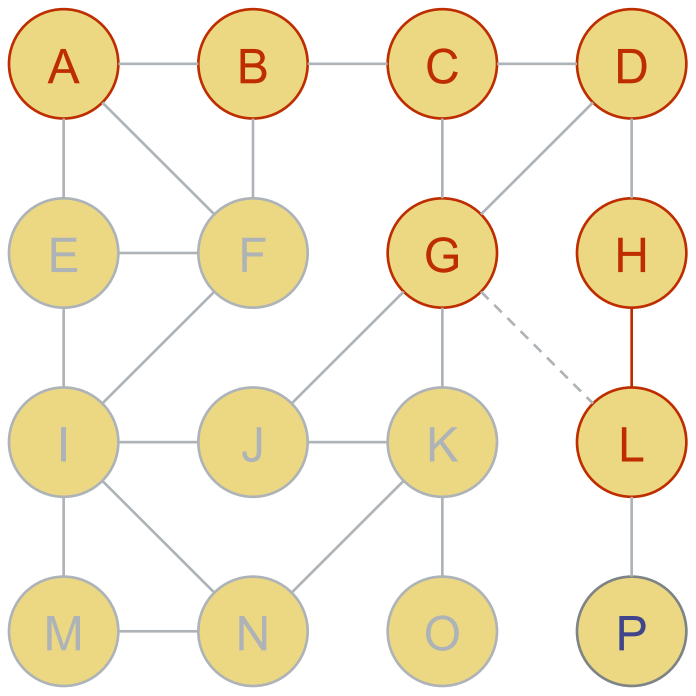
visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)

Current vertex: L

Edges to consider: to G,H,P

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

DFS in an Undirected Graph - Example



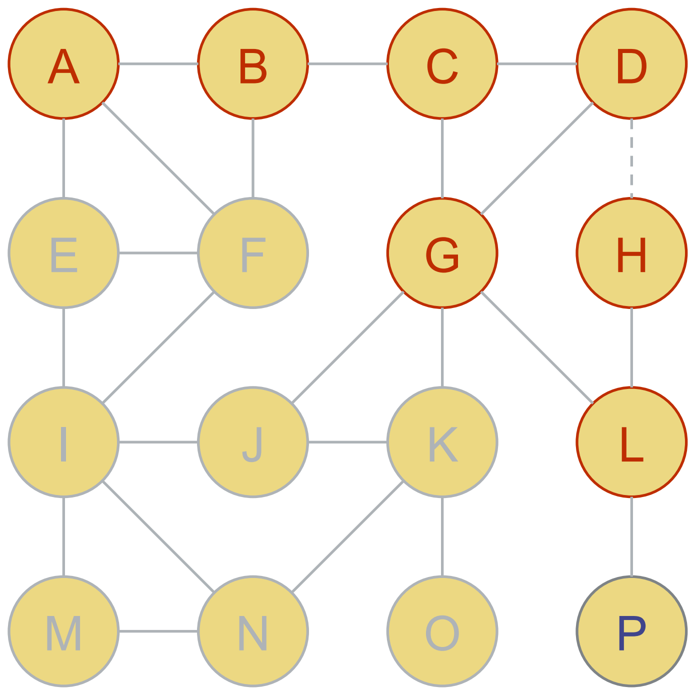
visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)

Current vertex: L

Edges to consider: to H,P

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

DFS in an Undirected Graph - Example

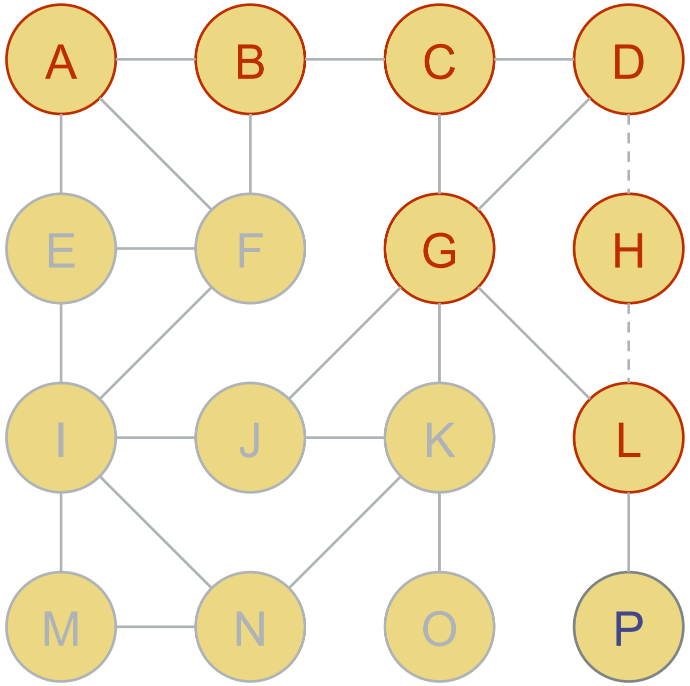


visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)

Current vertex: H
Edges to consider: to D,L

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

DFS in an Undirected Graph - Example

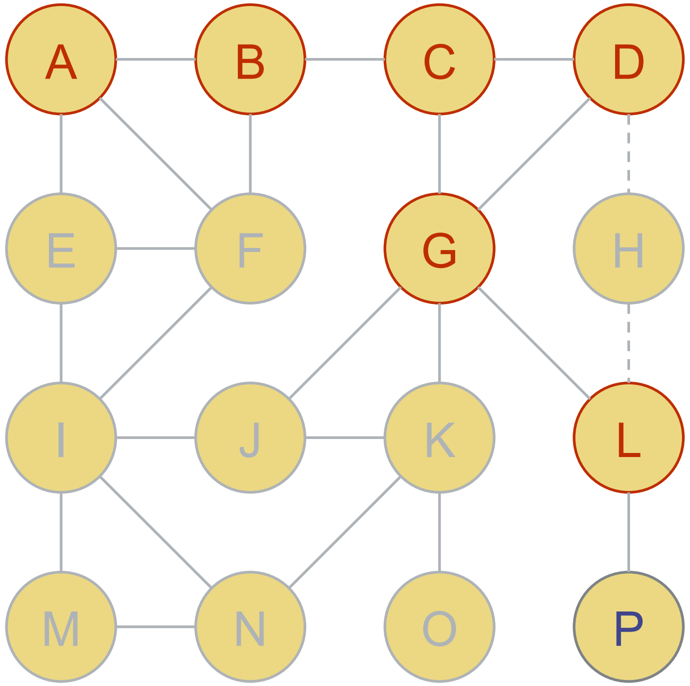


visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)

Current vertex: H
Edges to consider: to L

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

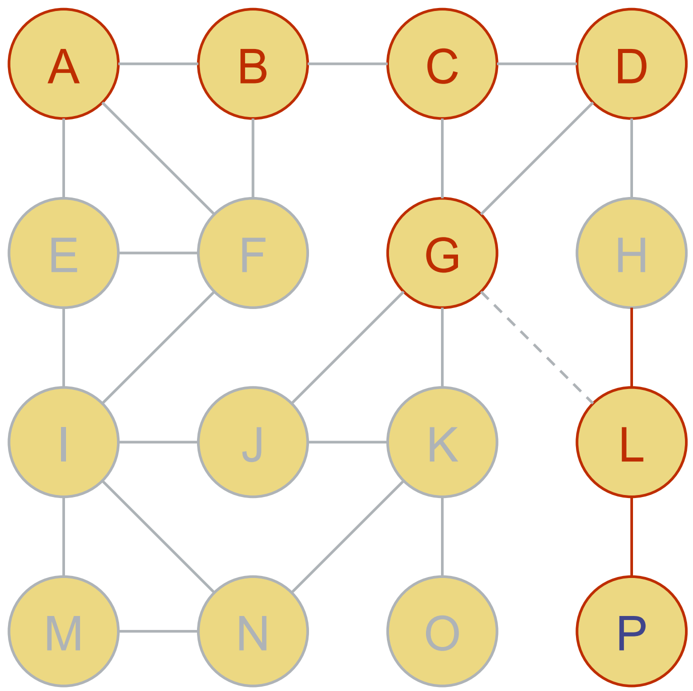
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)

Finished H, backtrack to L

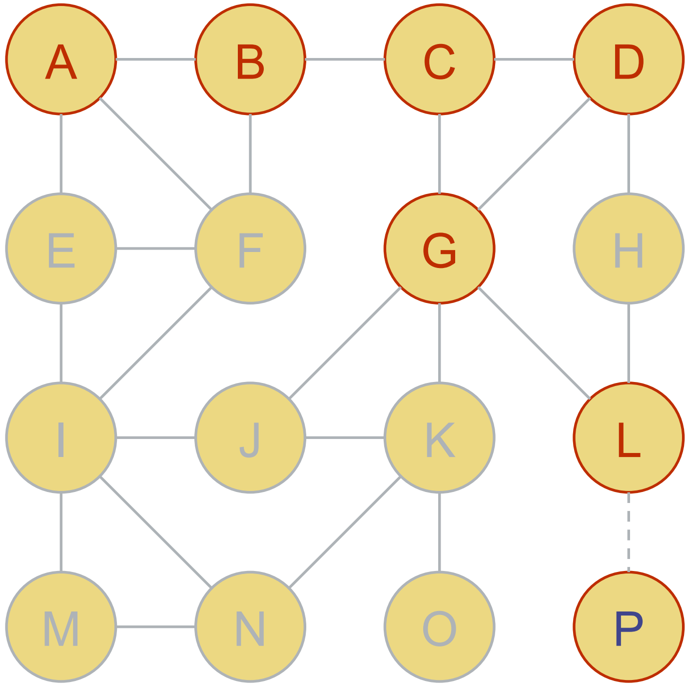
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: L
Edges to consider: to P

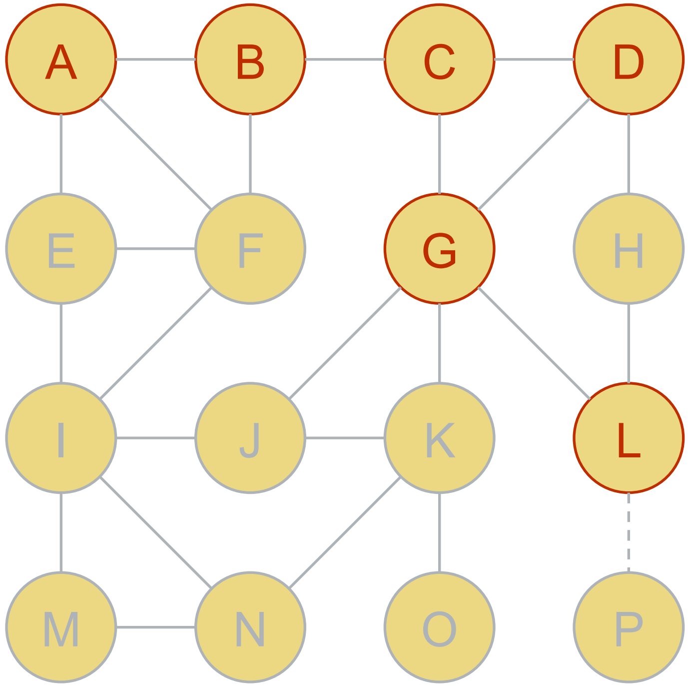
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: P
Edges to consider: to L

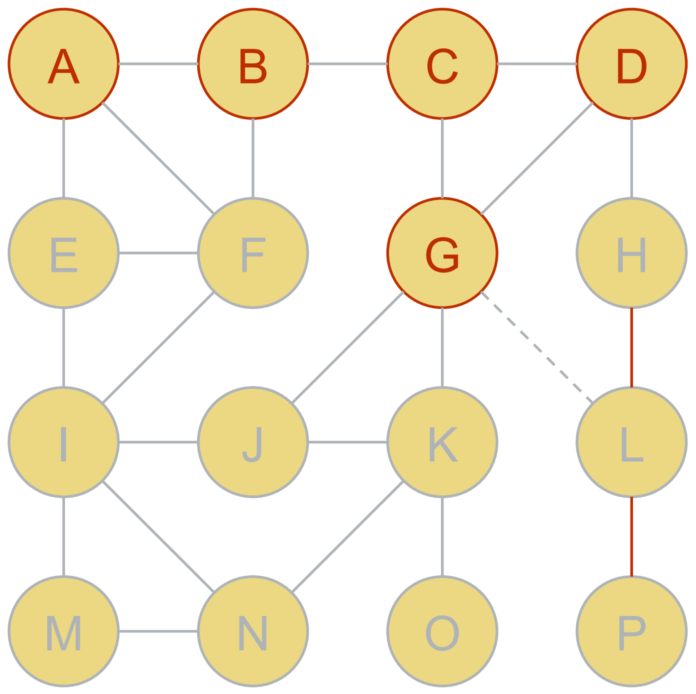
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Finished P, backtracking to L

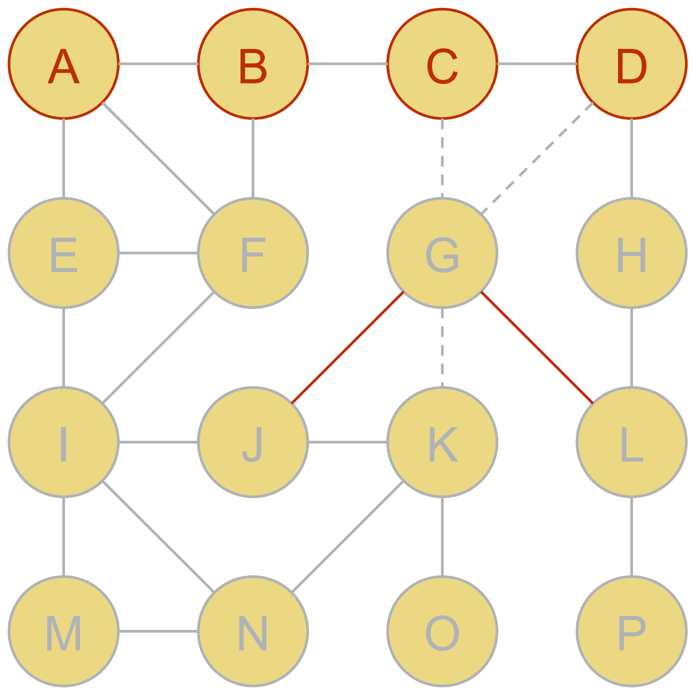
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: L
Edges to consider: -
Finished L, backtr. to G

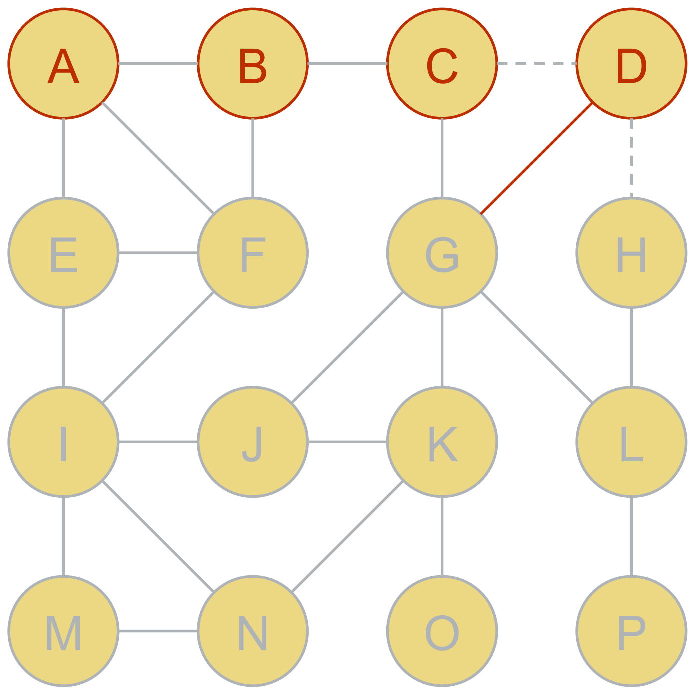
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: G
Edges to consider: -
Finished G, backtr. to D

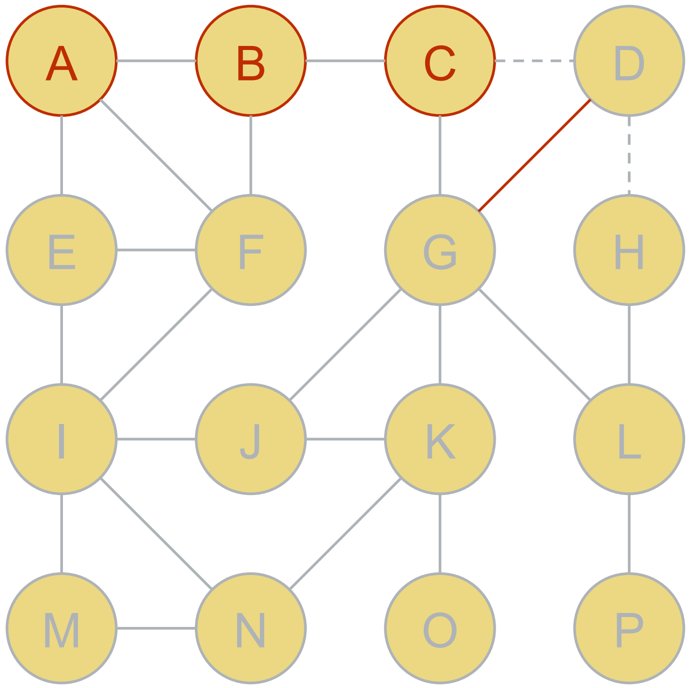
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: D
Edges to consider: H

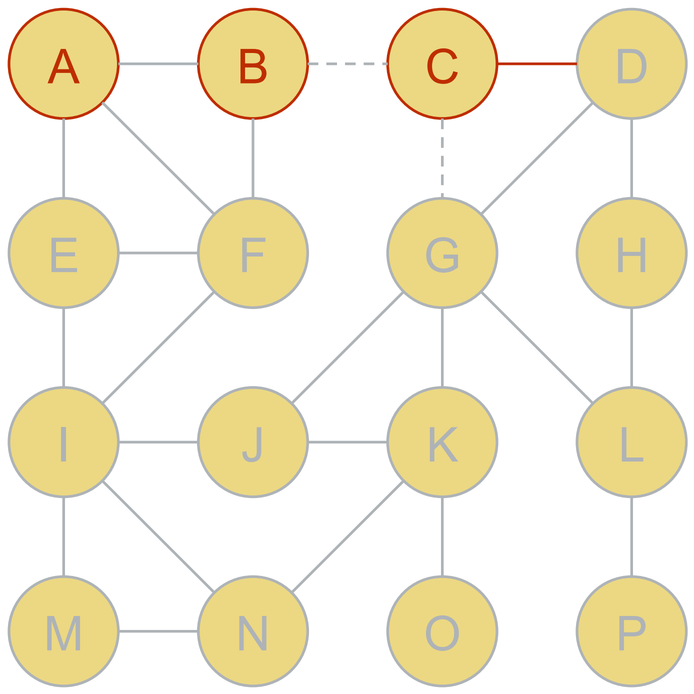
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: D
Edges to consider: -
Finished D, backtr. to C

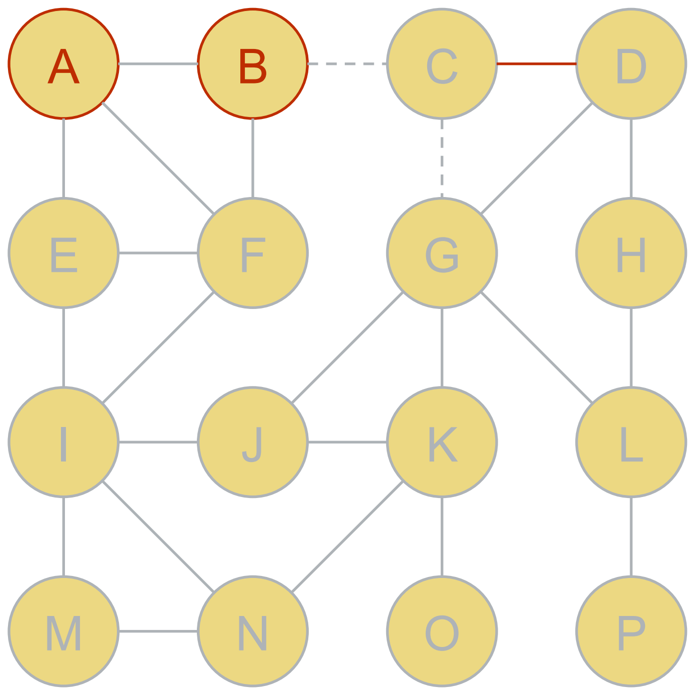
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: C
Edges to consider: G

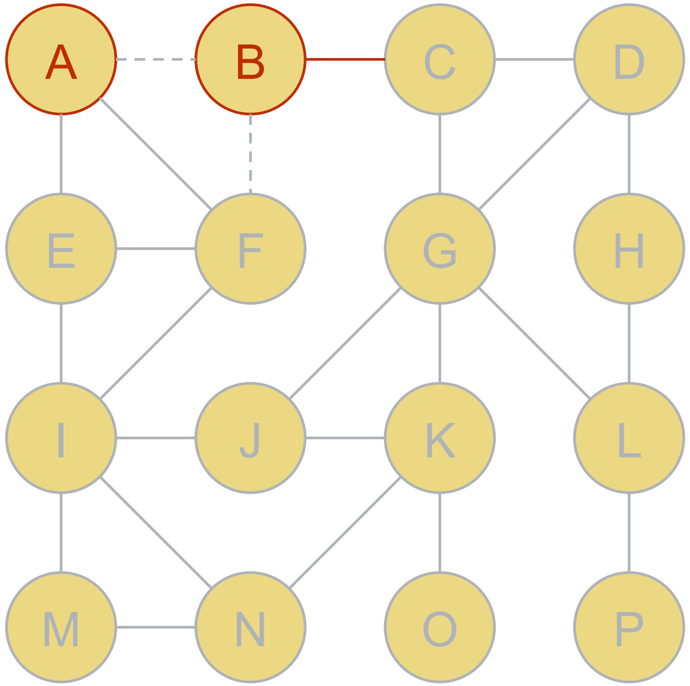
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: C
Edges to consider: -
Finished C, backtr. to B

DFS in an Undirected Graph - Example

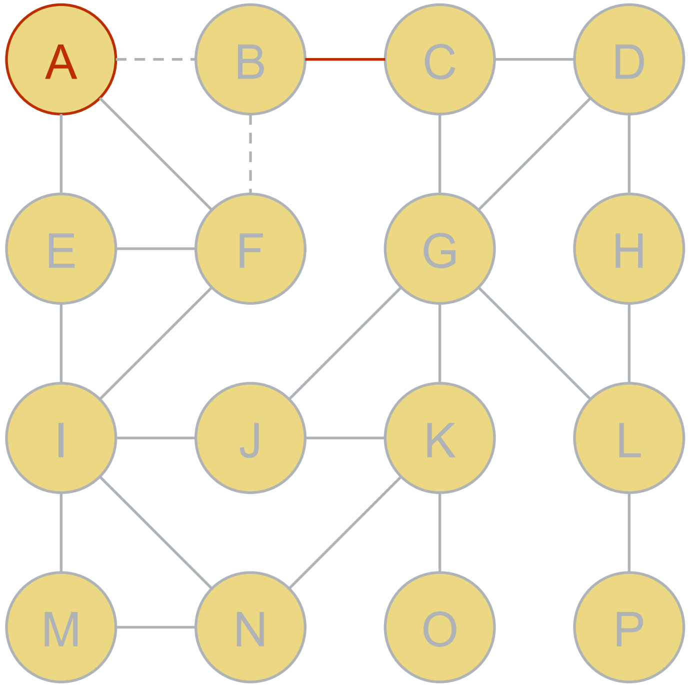


visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: B
Edges to consider: F

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

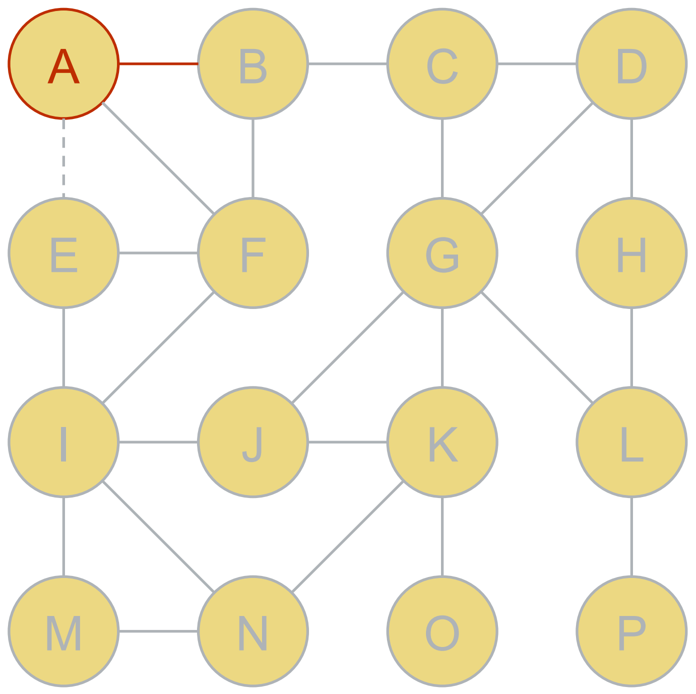
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: B
Edges to consider: -
Finished B, backtr. to A

DFS in an Undirected Graph - Example

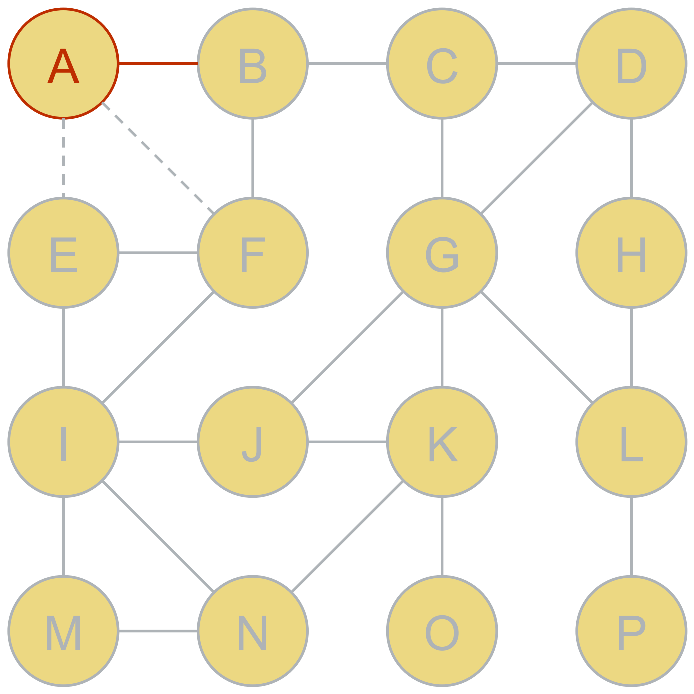


visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: A
Edges to consider: E, F

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

DFS in an Undirected Graph - Example

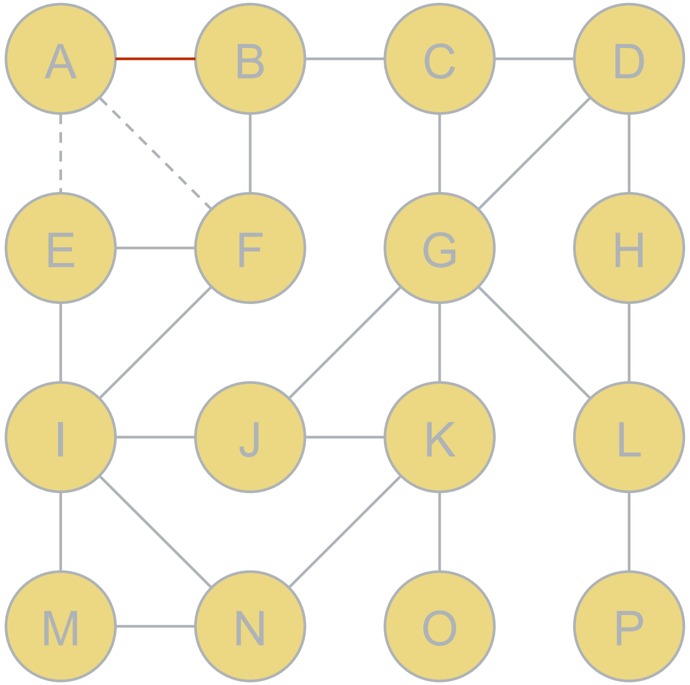


visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: A
Edges to consider: F

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

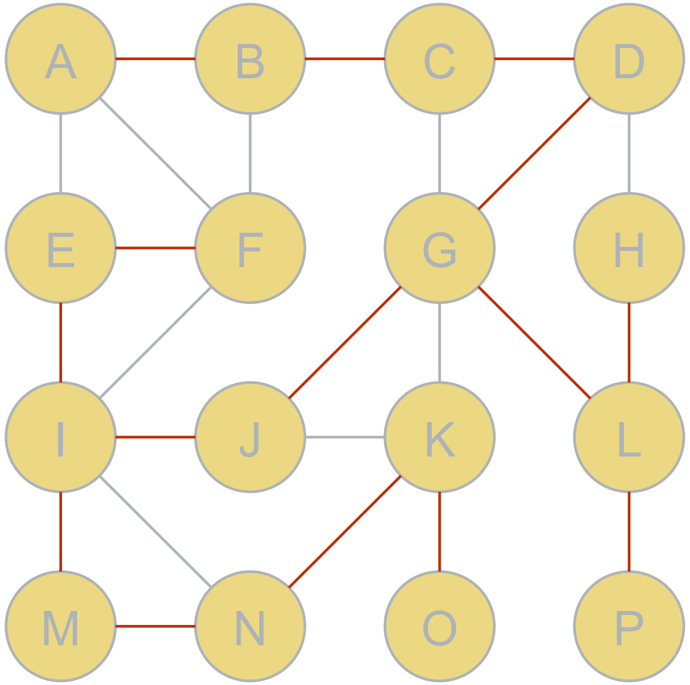
DFS in an Undirected Graph - Example



visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Current vertex: A
Edges to consider: -
Finished A, stop.

DFS in an Undirected Graph - Example



DFS has visited all the vertices of G.
The spanning tree of G, build only from discovery edges, is marked in red. The remaining edges are back edges.

visited	discovery edge
A	None
B	(A,B)
C	(B,C)
D	(C, D)
G	(D,G)
J	(G,J)
I	(J,I)
E	(I, E)
F	(E, F)
M	(I,M)
N	(M,N)
K	(N,K)
O	(K, O)
L	(G,L)
H	(L, H)
P	(L, P)

Properties of a DFS

- Proposition. Let G be an undirected graph for which a DFS traversal starting at vertex s has been performed. Then
  - the traversal visits all vertices in the connected component of s, and
  - the discovery edges form a spanning tree of the connected component of s.
- Justification. Suppose that the vertex w from s's connected component is not visited. Let v be the first unvisited vertex on a path from s to w ($v = w$ is also possible).
  - v is the first unvisited vertex on the path → it has a neighbor u which was visited
  - But when u was visited, edge (u, v) must have been considered
  - Hence v cannot be unvisited – contradiction.
  - A discovery edge is followed only when moving to an unvisited vertex → no cycles are possible → discovery edges form a tree (connected subgraph without cycles)
  - This is a spanning tree because DFS visits all the vertices from the connected component of s

Depth-First Search – Python Implementation

```
def DFS(g, u, discovered):
  """Perform DFS of the undiscovered portion of Graph g starting at Vertex u.

  discovered is a dictionary mapping each vertex to the edge that was used to
  discover it during the DFS. (u should be "discovered" prior to the call.)
  Newly discovered vertices will be added to the dictionary as a result.
  """
  for e in g.incident_edges(u):              # for every outgoing edge from u
    v = e.opposite(u)
    if v not in discovered:                  # v is an unvisited vertex
      discovered[v] = e                      # e is the tree edge that discovered v
      DFS(g, v, discovered)                  # recursively explore from v
```

```
result = {u : None}          # a new dictionary, with u trivially discovered
DFS(g, u, result)
```

Running Time of DFS

- Depth-first search is an efficient method for traversing a tree
- DFS is called at most once for each vertex (because the vertex is marked as visited)
- For an undirected graph, each edge (u, v) is examined at most twice – once from u and once from v
- If $n_s \leq n$ is the number of vertices reachable from the start vertex s and $m_s \leq m$ is the number of edges incident to those vertices then DFS runs in $O(n_s + m_s)$ time if
  - The data structure used to represent the graph can iterate though the edges of a vertex, `incident_edges(v)` in $O(\deg(v))$ time, and can find the opposite vertex, `e.opposite(v)` in $O(1)$ time
  - There is a method to mark the vertex or edge as explored, and to test if a vertex or edge has been explored in $O(1)$ time

Problems Solved using a DFS traversal in an Undirected Graph

a. Computing a path between two given vertices of G, if one exists.

b. Testing whether G is connected.

c. Computing the connected components of G.

d. Computing a cycle in G, or report that G has no cycles.

a. Compute a Path from u to v

- The DFS procedure was already performed for the graph G
- To reconstruct the path from u to v, start at the end of the path
- Look in the `discovered` dictionary for the edge that was used to discover v, and retrieve its other endpoint w
- Add add the endpoint w to a list, look again in the dictionary for the edge used to discover w and obtain its other endpoint.
- Continue until u is reached, then reverse the list and return it

a. Compute a Path from u to v – Python implementation

```
def construct_path(u, v, discovered):
  path = [ ]                                          # empty path by default
  if v in discovered:
    # we build list from v to u and then reverse it at the end
    path.append(v)
    walk = v
    while walk is not u:
      e = discovered[walk]                            # find edge leading to walk
      parent = e.opposite(walk)
      path.append(parent)
      walk = parent
    path.reverse( )                                   # reorient path from u to v
  return path
```

a. Compute a Path from u to v – Running Time

```
def construct_path(u, v, discovered):
  path = [ ]                                      # empty path by default
  if v in discovered:
    # we build list from v to u and then reverse it at the end
    path.append(v)
    walk = v
    while walk is not u:
      e = discovered[walk]                        # find edge leading to walk
      parent = e.opposite(walk)
      path.append(parent)
      walk = parent
    path.reverse( )                               # reorient path from u to v
  return path
```

- Function runs in time proportional to the length of the path, therefore ?

a. Compute a Path from u to v – Running Time

```
def construct_path(u, v, discovered):
  path = [ ]                                          # empty path by default
  if v in discovered:
    # we build list from v to u and then reverse it at the end
    path.append(v)
    walk = v
    while walk is not u:
      e = discovered[walk]                            # find edge leading to walk
      parent = e.opposite(walk)
      path.append(parent)
      walk = parent
    path.reverse( )                                   # reorient path from u to v
  return path
```

- Function runs in time proportional to the length of the path, therefore $O(n)$ + the time needed to perform DFS (which gives us `discovered`)

b. Test whether G is connected

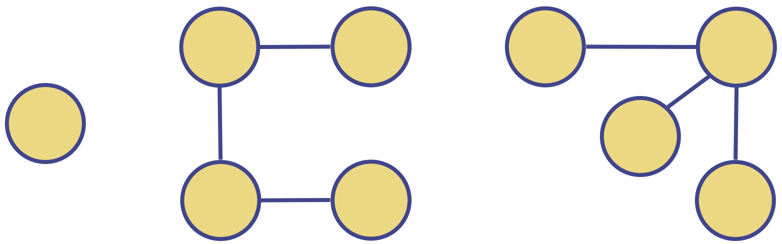
- The DFS procedure was already performed for the graph G, starting from an arbitrary vertex s
- Test if the `discovered` dictionary contains n entries (n is the number of vertices in G)
  - If yes, then G is connected, and all its vertices have been visited
  - If not, then G is not connected, and there is at least a vertex v that cannot be reached from any of the vertices in the connected component of s

b. Test whether G is connected - Runtime

- Runtime: only the time needed to perform the DFS - $O(n + m)$, since querying for the length of `discovered` is $O(1)$

c. Compute the connected components of G

- If an undirected graph is not connected, identify all of its the connected components
- If the initial DFS traversal has not reached all the vertices of a graph G
  - Start another DFS traversal from one of the vertices that are still not visited
  - Visit all vertices that are reachable from the new start vertex
  - Continue performing new DFS searches until all the vertices of G have been visited



Forest

c. Compute the connected components of G – Python implementation

```
def DFS_complete(g):
  """Perform DFS for entire graph and return forest as a dictionary.

  Result maps each vertex v to the edge that was used to discover it.
  (Vertices that are roots of a DFS tree are mapped to None.)
  """
  forest = { }
  for u in g.vertices():
    if u not in forest:
      forest[u] = None                    # u will be the root of a tree
      DFS(g, u, forest)
  return forest
```

- The number of connected components of G can be obtained by counting the number of vertices with a `None` edge in `forest` - these are the root vertices of each of the connected components

c. Compute the connected components of G – Runtime

```
def DFS_complete(g):
  """Perform DFS for entire graph and return forest as a dictionary.

  Result maps each vertex v to the edge that was used to discover it.
  (Vertices that are roots of a DFS tree are mapped to None.)
  """
  forest = { }
  for u in g.vertices():
    if u not in forest:
      forest[u] = None                    # u will be the root of a tree
      DFS(g, u, forest)
  return forest
```

- Although there are multiple calls to DFS, the total running time of DFS complete is $O(n+m)$, because there are n vertices and m edges in total in the graph G, which is not connected
  - Each connected component takes $O(n_{s1}+m_{s1})$ time
  - Each DFS call from DFS_complete explores a different component, $O(n_{si}+m_{si})$
  - The sum is $O(n+m)$

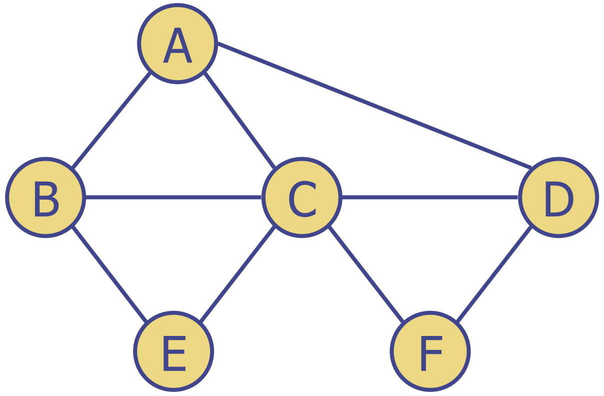
d. Compute a cycle in G

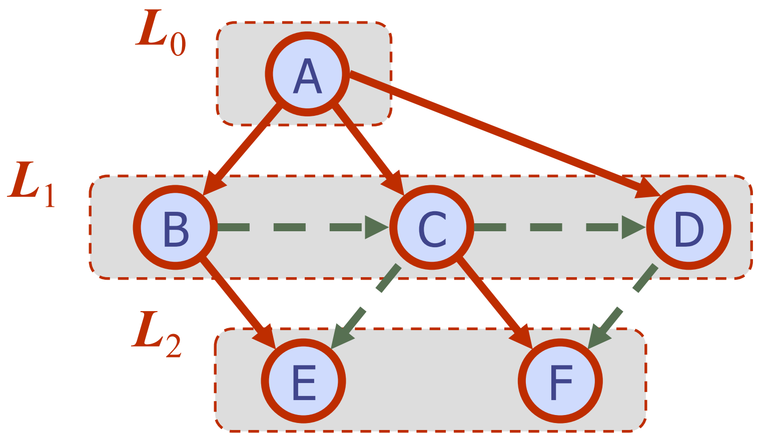
- The DFS procedure was already performed for the graph G
- A cycle exists if and only if a back edge exists with respect to the DFS traversal of that graph
- To obtain the cycle, take the back edge from the descendant to the ancestor and then follow DFS tree edges back to the descendant

Breadth-First Search

Breadth-First Search (BFS) - Intuition

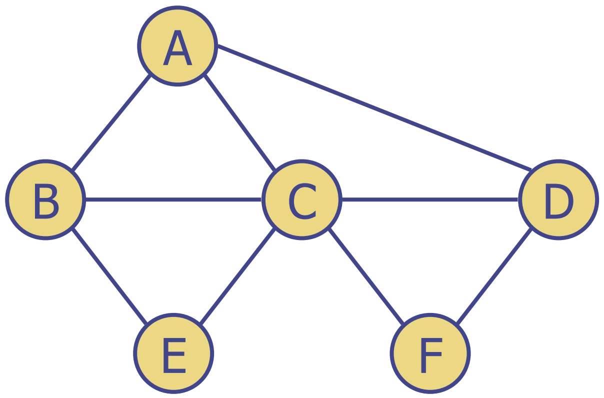
- Depth-first search – imagine a traversal done by a single person exploring a graph
- Breath-first search – imagine sending out, in all directions, many persons that traverse the graph in a collaborative way
- BFS works in rounds and subdivides the vertices into levels
- It starts at vertex s, which is at level 0

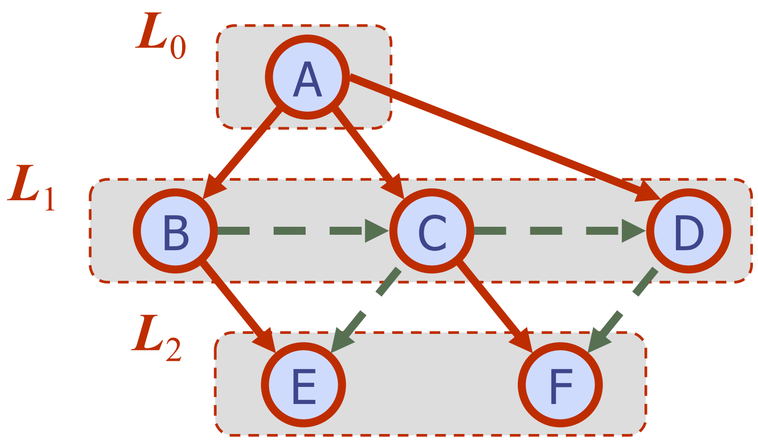




Breadth-First Search (BFS) - Algorithm

- Start at vertex s, which is at level 0; s is marked as visited
- In the first round all the vertices that are adjacent to s are marked as visited – these vertices, which are one step away from s, are placed on level 1
- In the second round all the vertices that are adjacent to any of the vertices on level 1 are marked as visited; these vertices are two steps away from s and are placed on level 2
- The process continues until no new vertices are found in a level



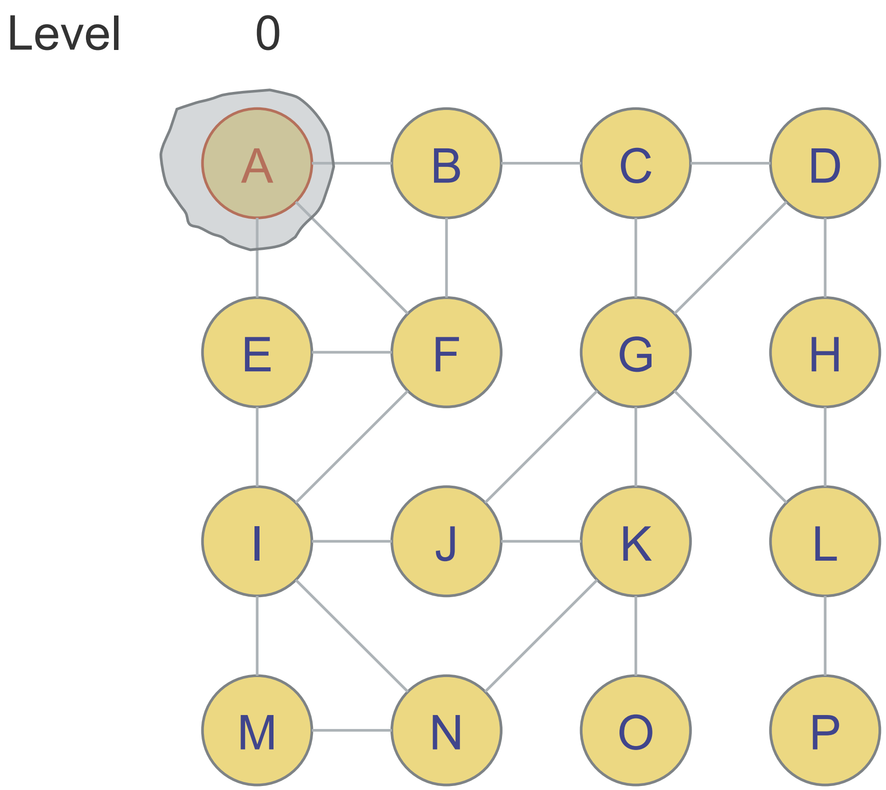


Breadth-First Search (BFS) – Python implementation

```
def BFS(g, s, discovered):
  """Perform BFS of the undiscovered portion of Graph g starting at Vertex s.

  discovered is a dictionary mapping each vertex to the edge that was used to
  discover it during the BFS (s should be mapped to None prior to the call).
  Newly discovered vertices will be added to the dictionary as a result.
  """
  level = [s]                          # first level includes only s
  while len(level) > 0:
    next_level = [ ]                   # prepare to gather newly found vertices
    for u in level:
      for e in g.incident_edges(u):    # for every outgoing edge from u
        v = e.opposite(u)
        if v not in discovered:        # v is an unvisited vertex
          discovered[v] = e            # e is the tree edge that discovered v
          next_level.append(v)         # v will be further considered in next pass
    level = next_level                 # relabel 'next' level to become current
```

Breadth-First Search (BFS) – Example

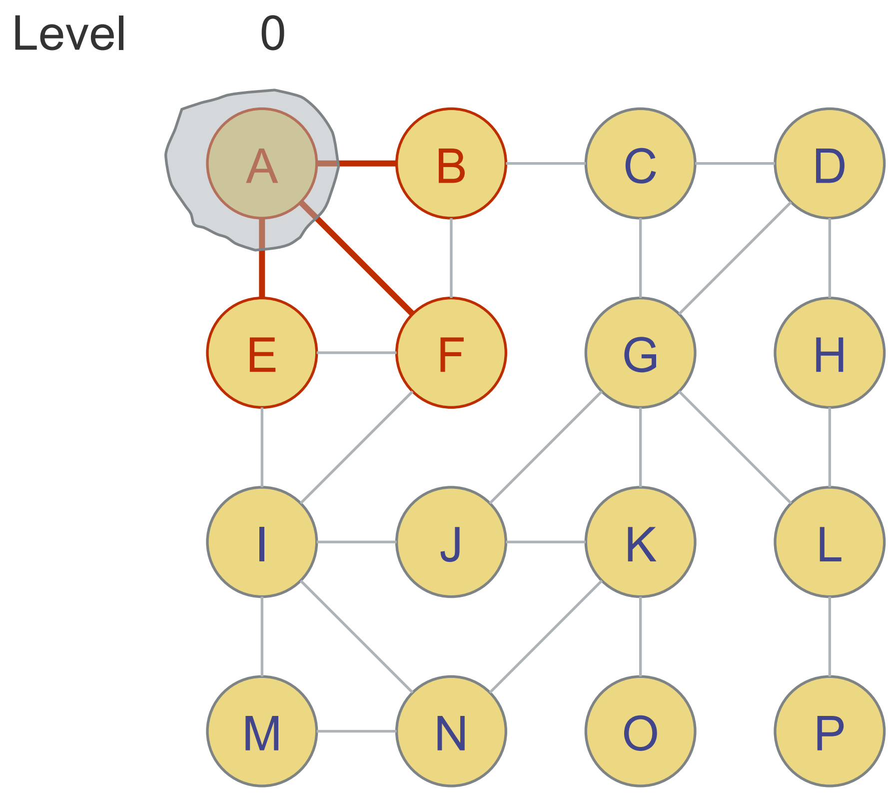


visited	discovery edge
A	None

Current level: A
Edges to consider: to B, E, F

- Start from vertex A, which is marked as visited (red)

Breadth-First Search (BFS) – Example

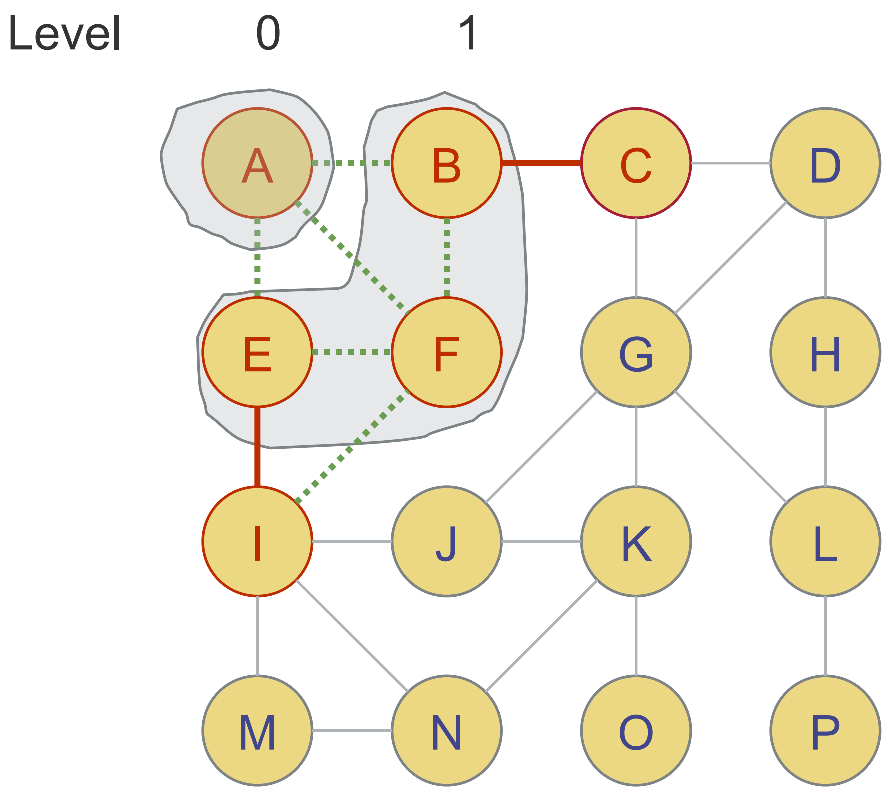


visited	discovery edge
A	None
B	(A,B)
E	(A,E)
F	(A,F)

Current level: A
Edges to consider: to B, E, F

- Mark edges to non-visited vertices with red

Breadth-First Search (BFS) – Example

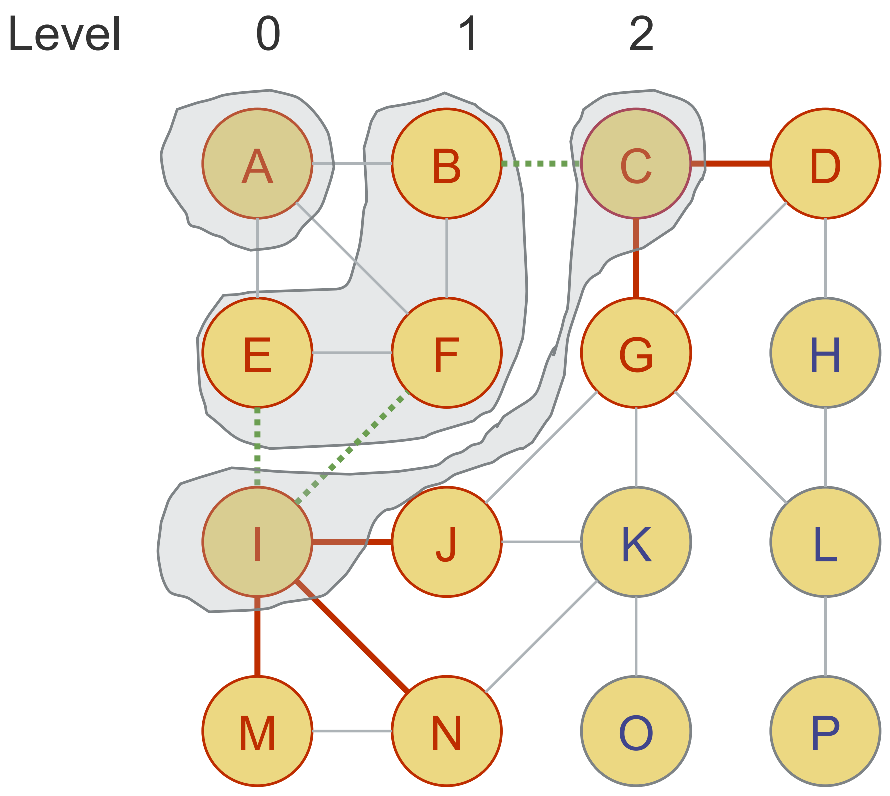


visited	discovery edge
A	None
B	(A,B)
E	(A,E)
F	(A,F)
C	(B,C)
I	(E,I)

Current level: B, E, F
Edges to consider: (B,A), (B,C), (B,F), (E,A), (E,F), (E,I), (F,A), (F,B), (F,E), (F,I)

- Mark edges to non-visited vertices with red
- Mark edges to visited vertices with green (dotted)

Breadth-First Search (BFS) – Example

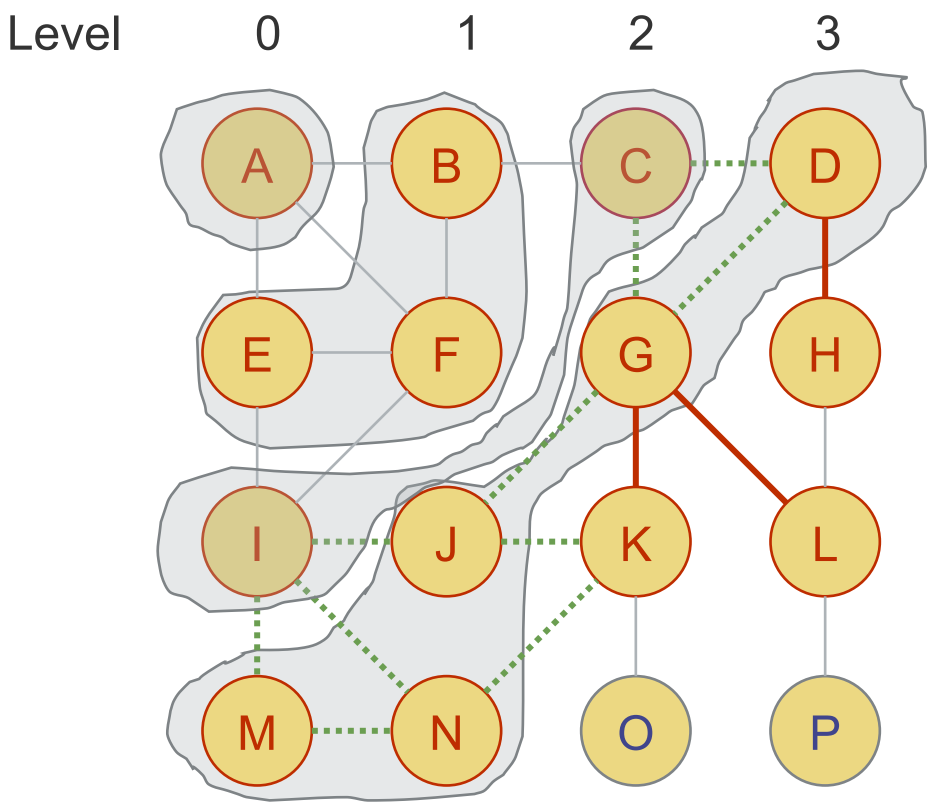


visited	discovery edge
A	None
B	(A,B)
E	(A,E)
F	(A,F)
C	(B,C)
I	(E,I)
D	(C,D)
G	(C,G)
J	(I,J)
M	(I,M)
N	(I,N)

Current level: C, I
Edges to consider: (C,B), (C,D), (C,G), (I,E), (I,F), (I,J), (I,M), (I,N)

- Mark edges to non-visited vertices with red
- Mark edges to visited vertices with green (dotted)

Breadth-First Search (BFS) – Example

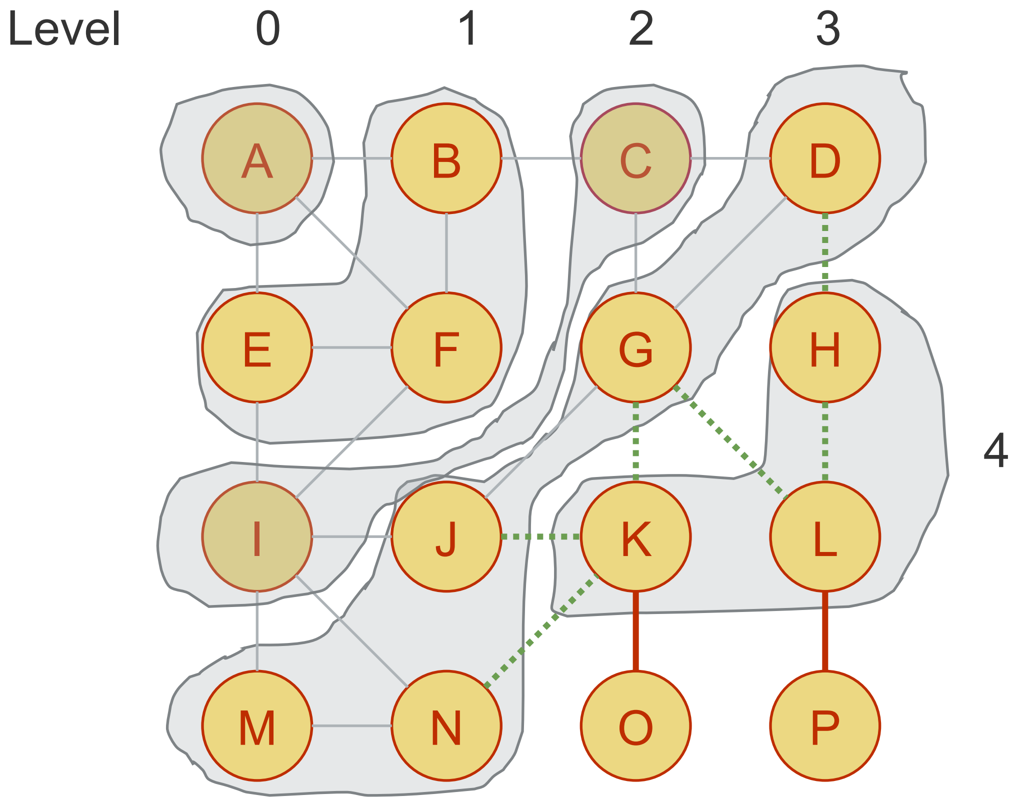


Current level: D, G, J, M, N
Edges to consider: (D,C), (D,G), (D,H), (G, C), (G,D), (G,J), (G,K), (G,L), (J,G), (J,I), (J,K), (M,I), (M,N), (N,I), (N,K), (N,M)

visited	discovery edge
A	None
B	(A,B)
E	(A,E)
F	(A,F)
C	(B,C)
I	(E,I)
D	(C,D)
G	(C,G)
J	(I,J)
M	(I,M)
N	(I,N)
H	(D,H)
K	(G,K)
L	(G,L)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

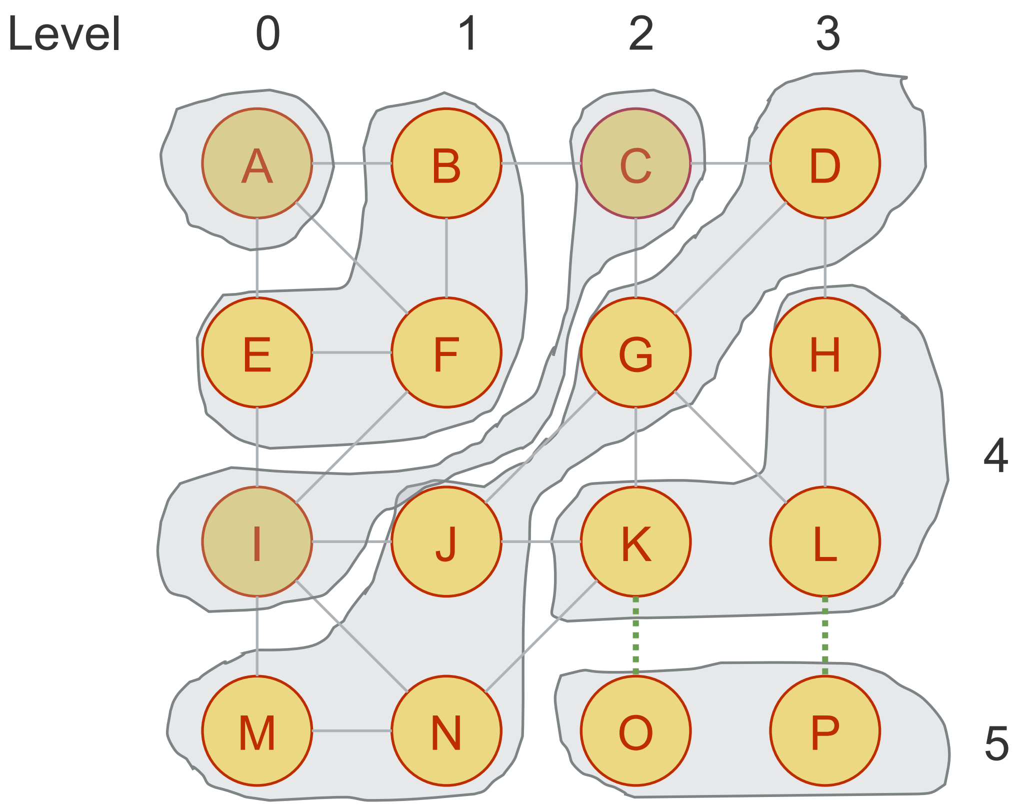
Breadth-First Search (BFS) – Example



Current level: H, K, L
Edges to consider: (H,D), (H,L), (K,G), (K,J), (K,N), (K,O), (L,G), (L,H), (L,P)

visited	discovery edge
A	None
B	(A,B)
E	(A,E)
F	(A,F)
C	(B,C)
I	(E,I)
D	(C,D)
G	(C,G)
J	(I,J)
M	(I,M)
N	(I,N)
H	(D,H)
K	(G,K)
L	(G,L)
O	(K,O)
P	(L,P)

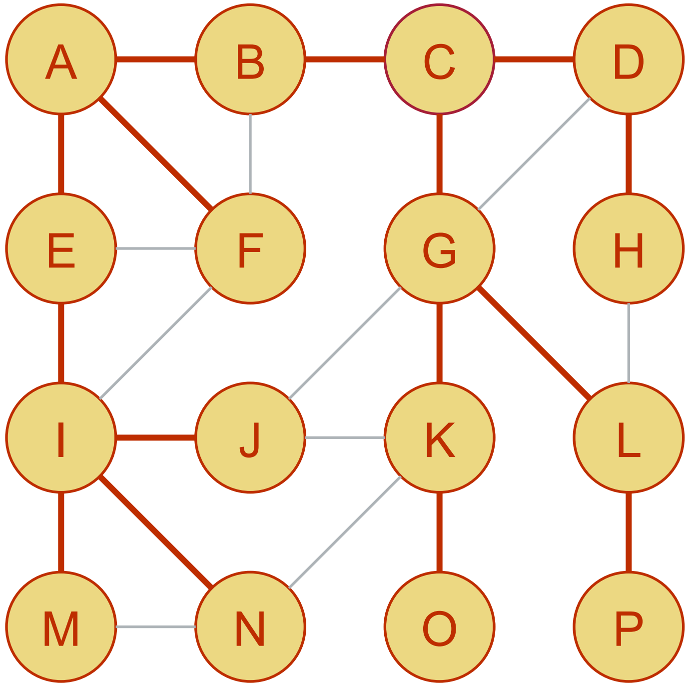
Breadth-First Search (BFS) – Example



Current level: O, P
Edges to consider: (O, K), (P, L)
No new nodes, stop.

visited	discovery edge
A	None
B	(A,B)
E	(A,E)
F	(A,F)
C	(B,C)
I	(E,I)
D	(C,D)
G	(C,G)
J	(I,J)
M	(I,M)
N	(I,N)
H	(D,H)
K	(G,K)
L	(G,L)
O	(K,O)
P	(L,P)

Breadth-First Search (BFS) – Example



- Edges of BFS traversal tree (starting from A) marked in red (discovery edges)

visited	discovery edge
A	None
B	(A,B)
E	(A,E)
F	(A,F)
C	(B,C)
I	(E,I)
D	(C,D)
G	(C,G)
J	(I,J)
M	(I,M)
N	(I,N)
H	(D,H)
K	(G,K)
L	(G,L)
O	(K,O)
P	(L,P)

Breadth-First Search (BFS) - Properties

- Proposition. A path p_1 in a breadth-first search rooted at vertex s to any other vertex v is guaranteed to be the shortest such path from s to v in terms of the number of edges.
- Justification. Suppose that there was another path, p_2 from s to v that was shorter than p_1
  - This means that p_2 is at least one edge shorter than p_1
  - This means that v was already discovered on the previous level by p_2
  - But p_1 is also a path in the BFS tree – so v appears on two levels – contradiction, because the levels are made of disjoint nodes, marked as visited on their first visit

Breadth-First Search (BFS) – Properties (cont'd)

- Consider G, an undirected graph on which a BFS traversal starting at vertex s has been performed. Then:
  - The traversal visits all vertices of G that are reachable from s
  - For each vertex at level i, the path of the BFS tree between s and v has i edges, and any other path of G from s to v has at least i edges
  - If (u, v) is an edge that is not in the BFS tree then the level number of v can be at most 1 greater than the level number of u

- Exercise: try to justify each of these properties using contradiction or induction.

Breadth-First Search (BFS) – Running time

- For a graph G with n vertices and m nodes represented using an adjacency list structure a BFS traversal takes $O(n+m)$ time if the graph is connected if 1 and 2 are satisfied
- As in the DFS case, if $n_s \leq n$ is the number of vertices reachable from s, and $m_s \leq m$ is the number of edges incident to those vertices, then BFS runs in $O(n_s + m_s)$ time if

  1. The data structure used to represent the graph can iterate though the edges of a vertex, `incident_edges(v)` in $O(\deg(v))$ time, and can find the opposite vertex, `e.opposite(v)` in $O(1)$ time
  2. There is a method to mark the vertex or edge as explored, and to test if a vertex or edge has been explored in $O(1)$ time

- A procedure similar to the `DFS_complete()` function can be used to explore the entire graph in cases where the graph is made of multiple connected components

BFS vs. DFS

Undirected Graph Applications	DFS	BFS
Find a set of vertices that are reachable from a given source, and determine paths to those vertices	√	√
Shortest paths		√
Test the connectivity of a graph	√	√
Identify connected components	√	√
Locate a cycle	√	√

Thank you.