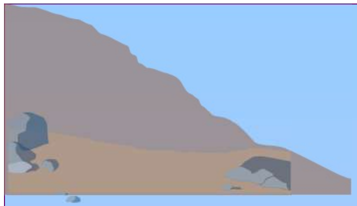
Graph Traversal

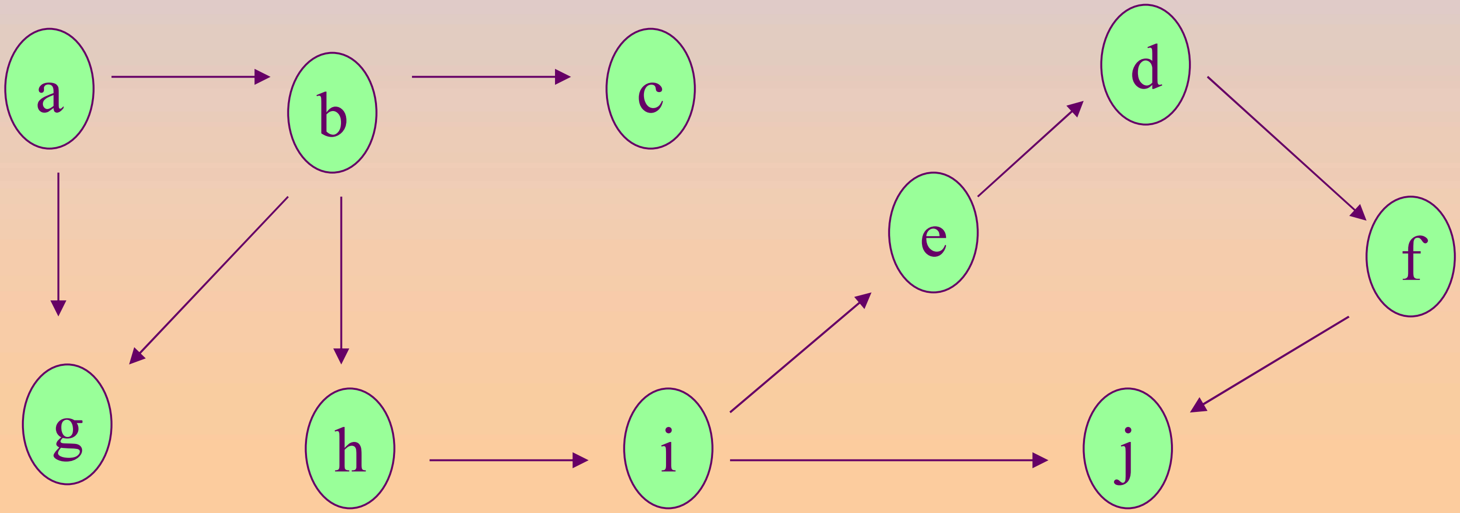


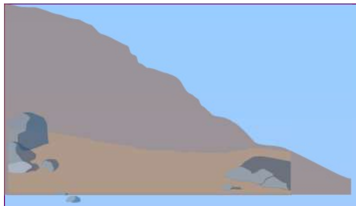
Graph Search (traversal)

- How do we search a graph?
  - At a particular vertices, where shall we go next?
- Two common framework:
    - the depth-first search (DFS)
    - the breadth-first search (BFS) and
  - In DFS, go as far as possible along a single path until reach a dead end (a vertex with no edge out or no neighbor unexplored) then backtrack
  - In BFS, one explore a graph level by level away (explore all neighbors first and then move on)

Depth-First Search (DFS)

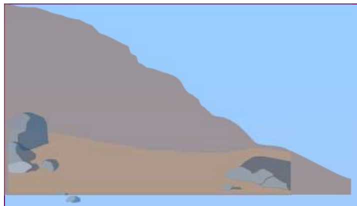
- The basic idea behind this algorithm is that it traverses the graph using recursion
  - Go as far as possible until you reach a deadend
  - Backtrack to the previous path and try the next branch
  - The graph below, started at node a, would be visited in the following order: a, b, c, g, h, i, e, d, f, j





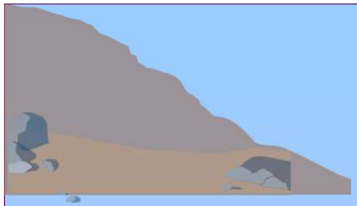
DFS: Color Scheme

- Vertices initially colored white
- Then colored gray when discovered
- Then black when finished

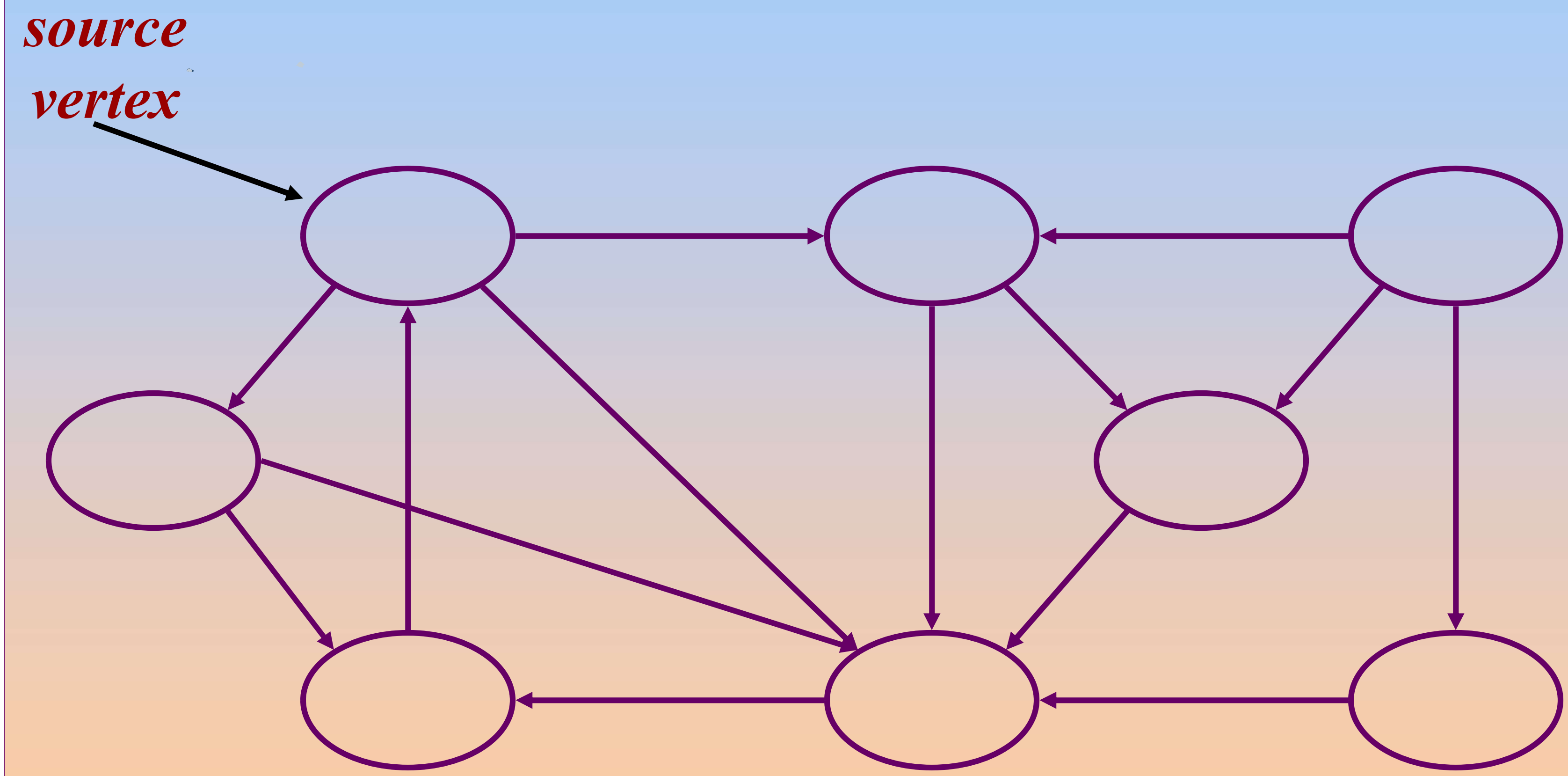


DFS: Time Stamps

- Discover time d[u]: when u is first discovered
- Finish time f[u]: when backtrack from u
- d[u] < f[u]

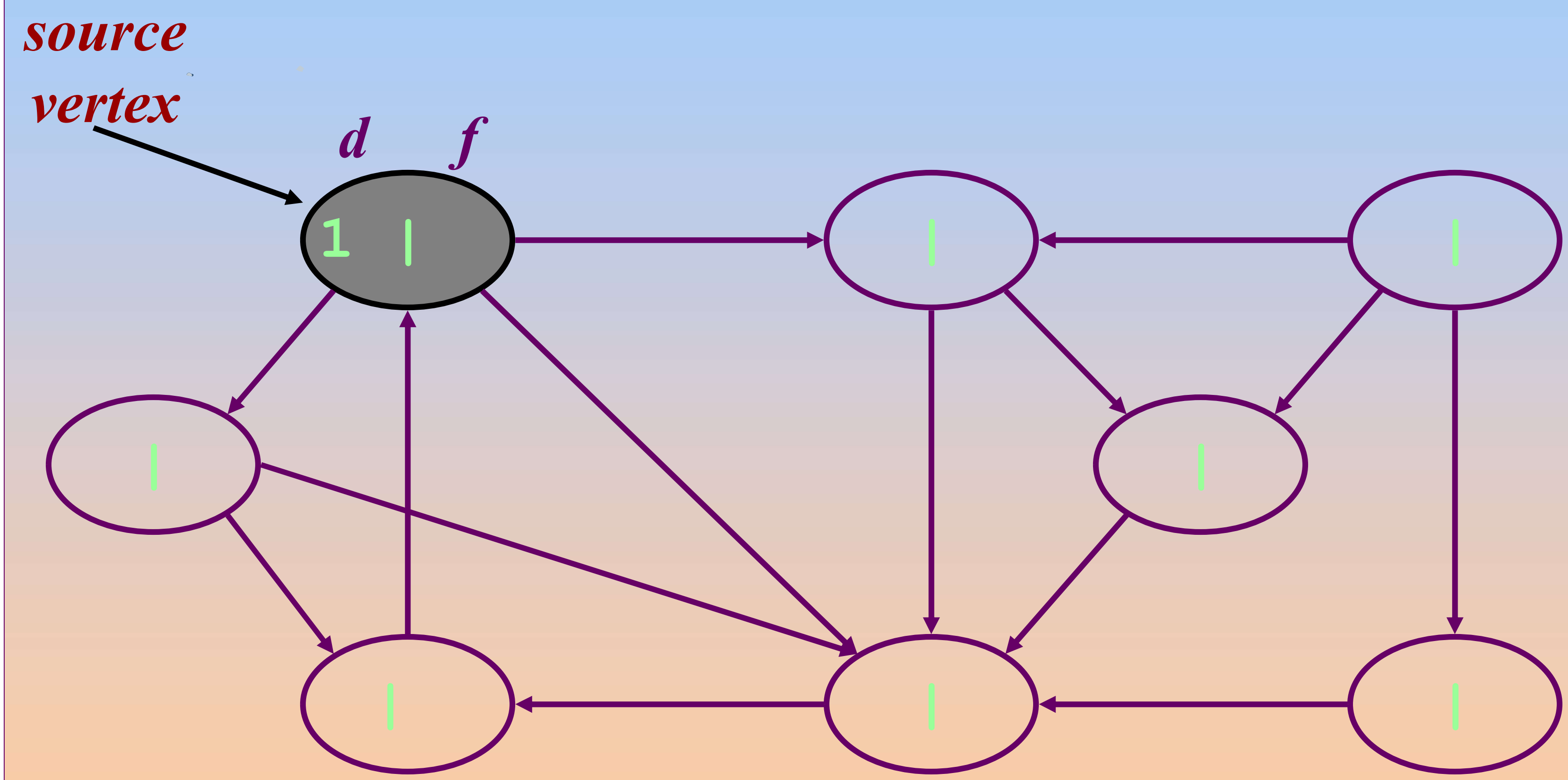


DFS Example

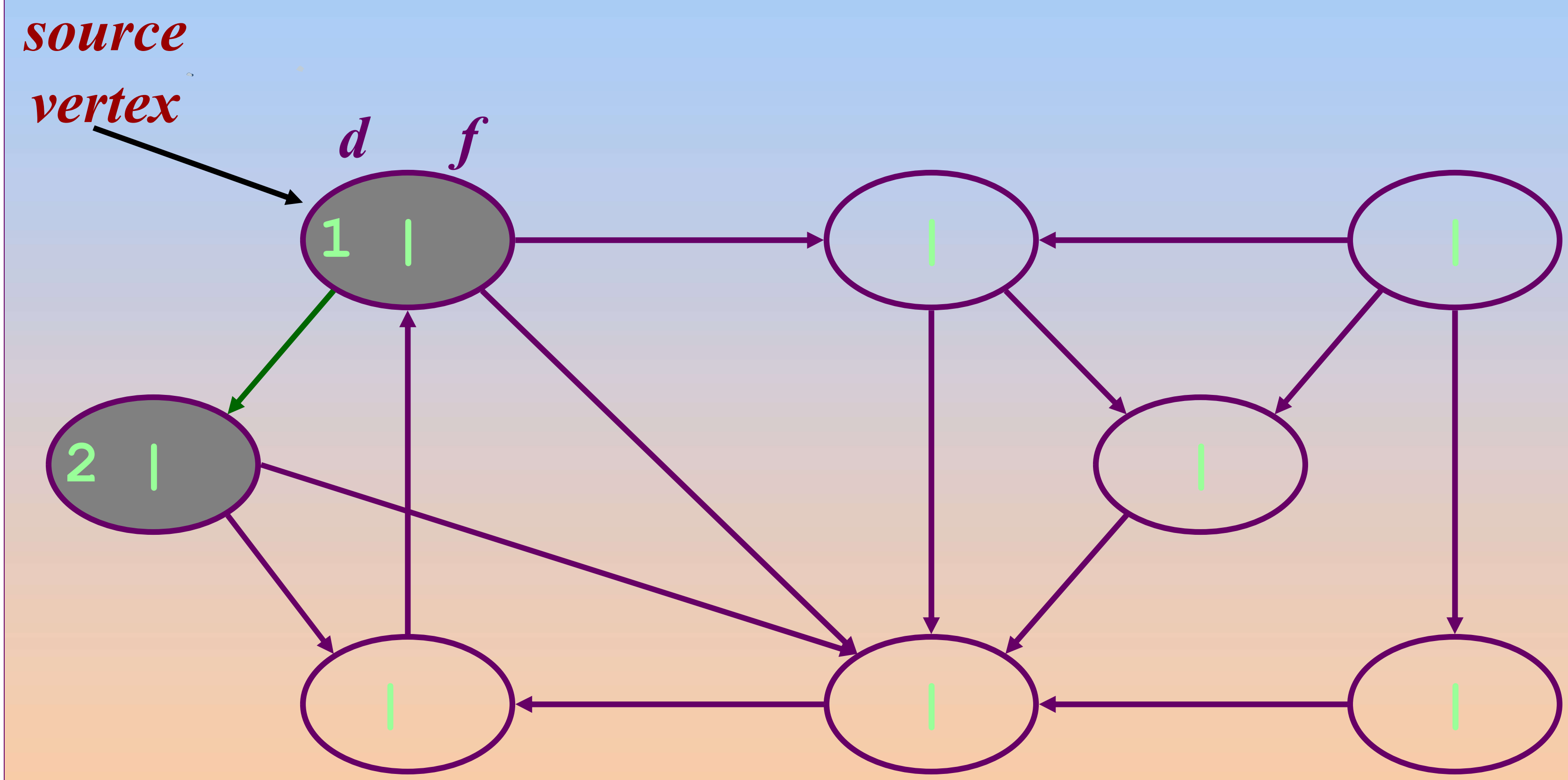


source vertex

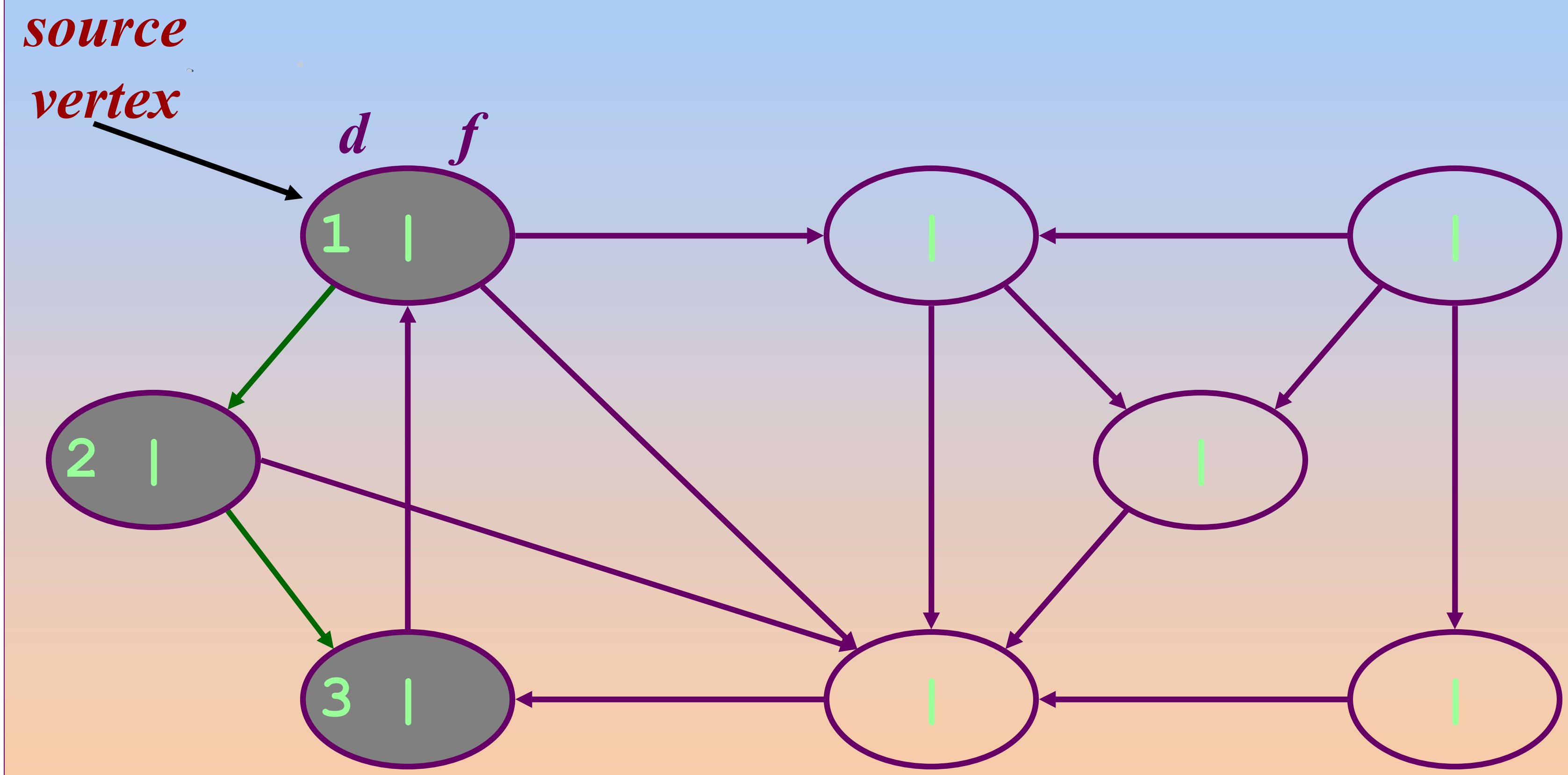
DFS Example



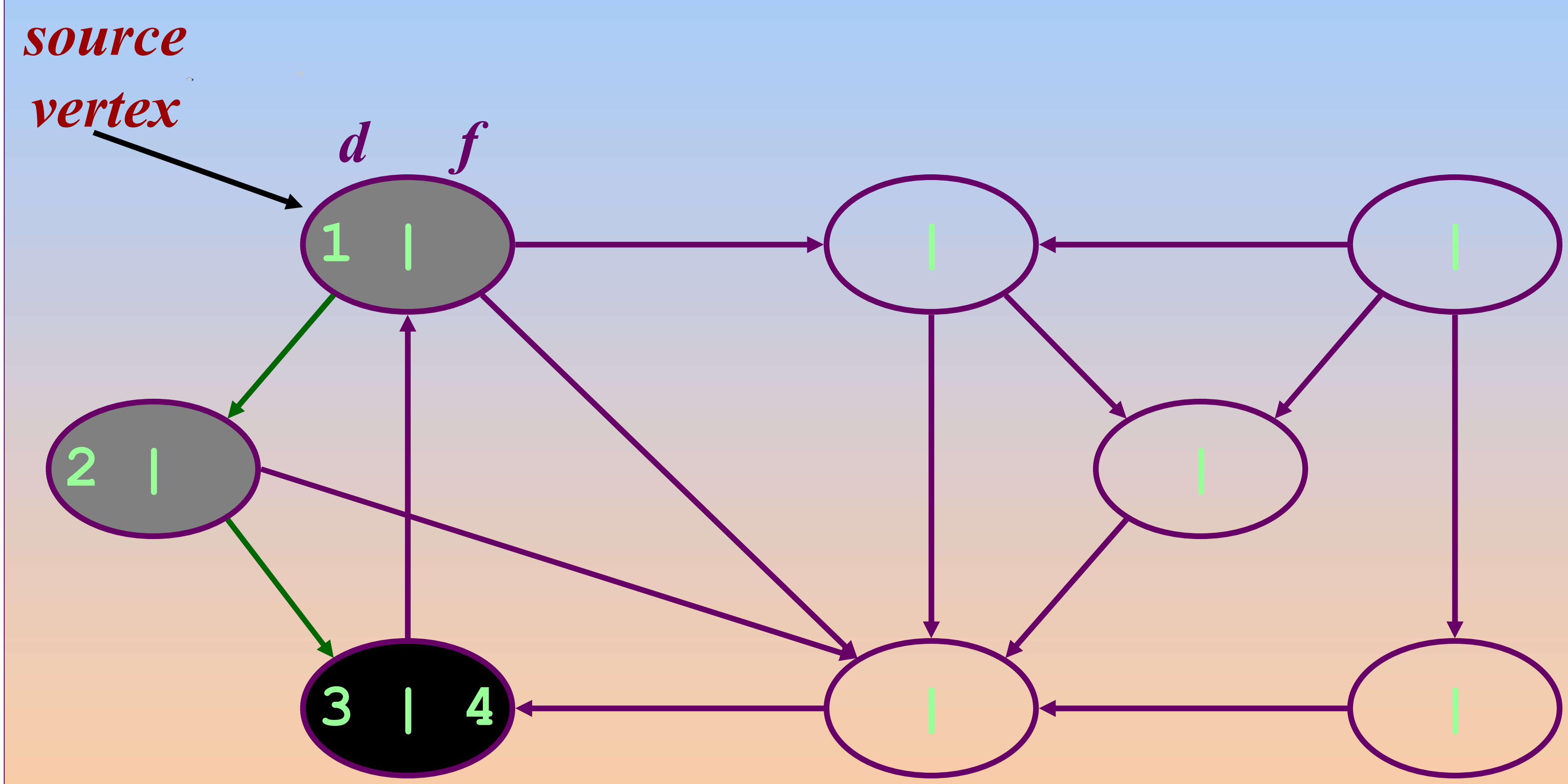
DFS Example



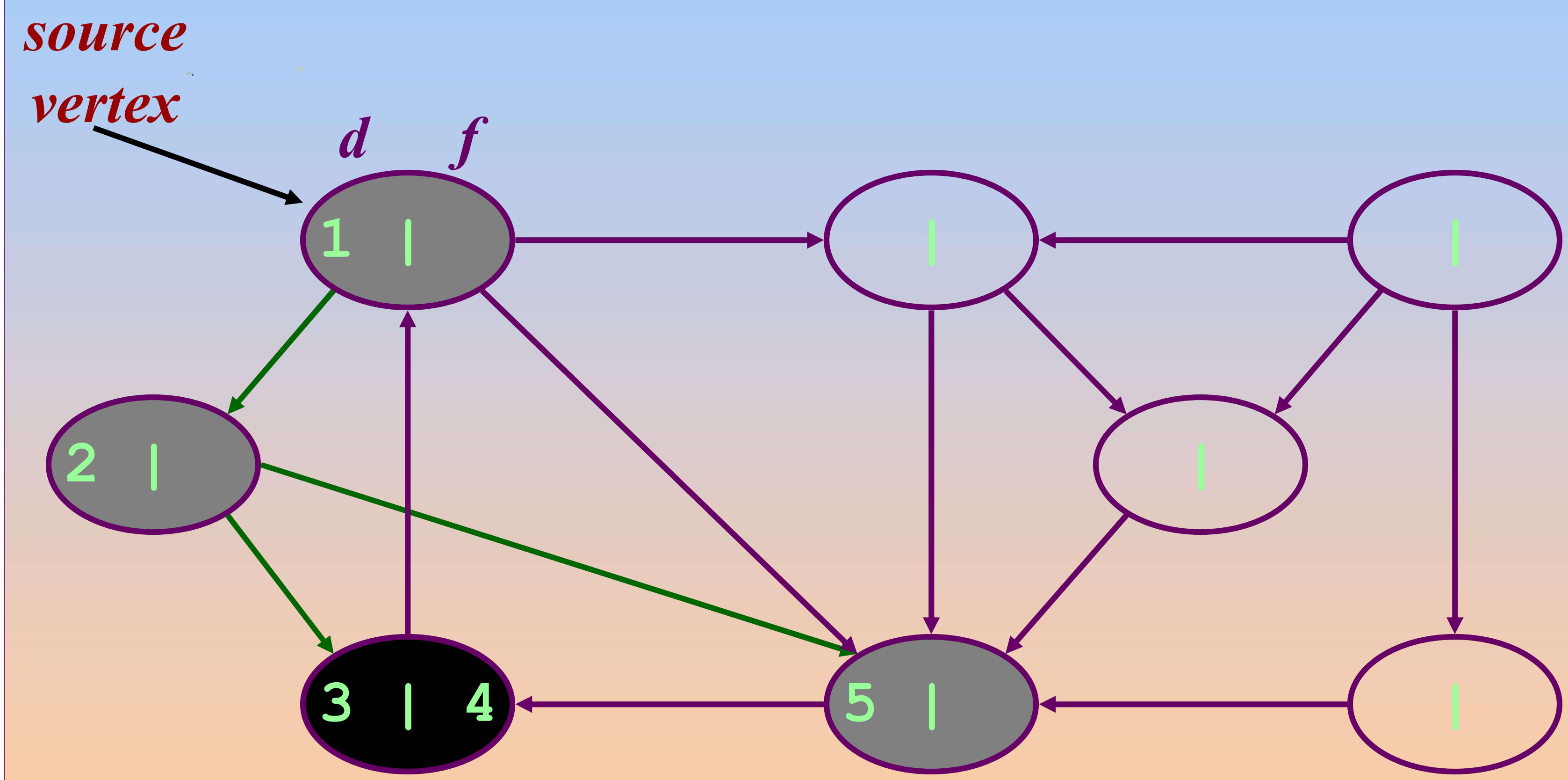
DFS Example

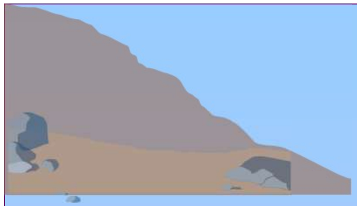


DFS Example



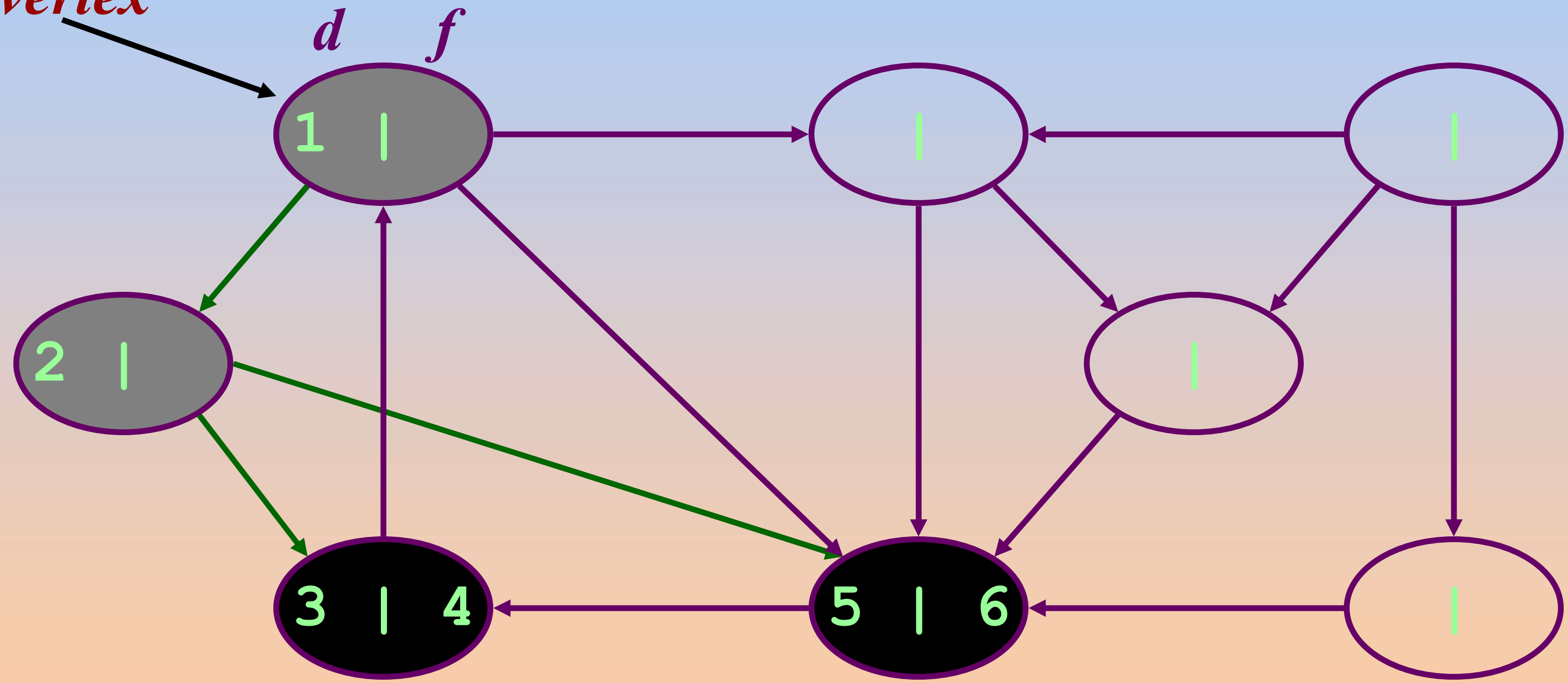
DFS Example



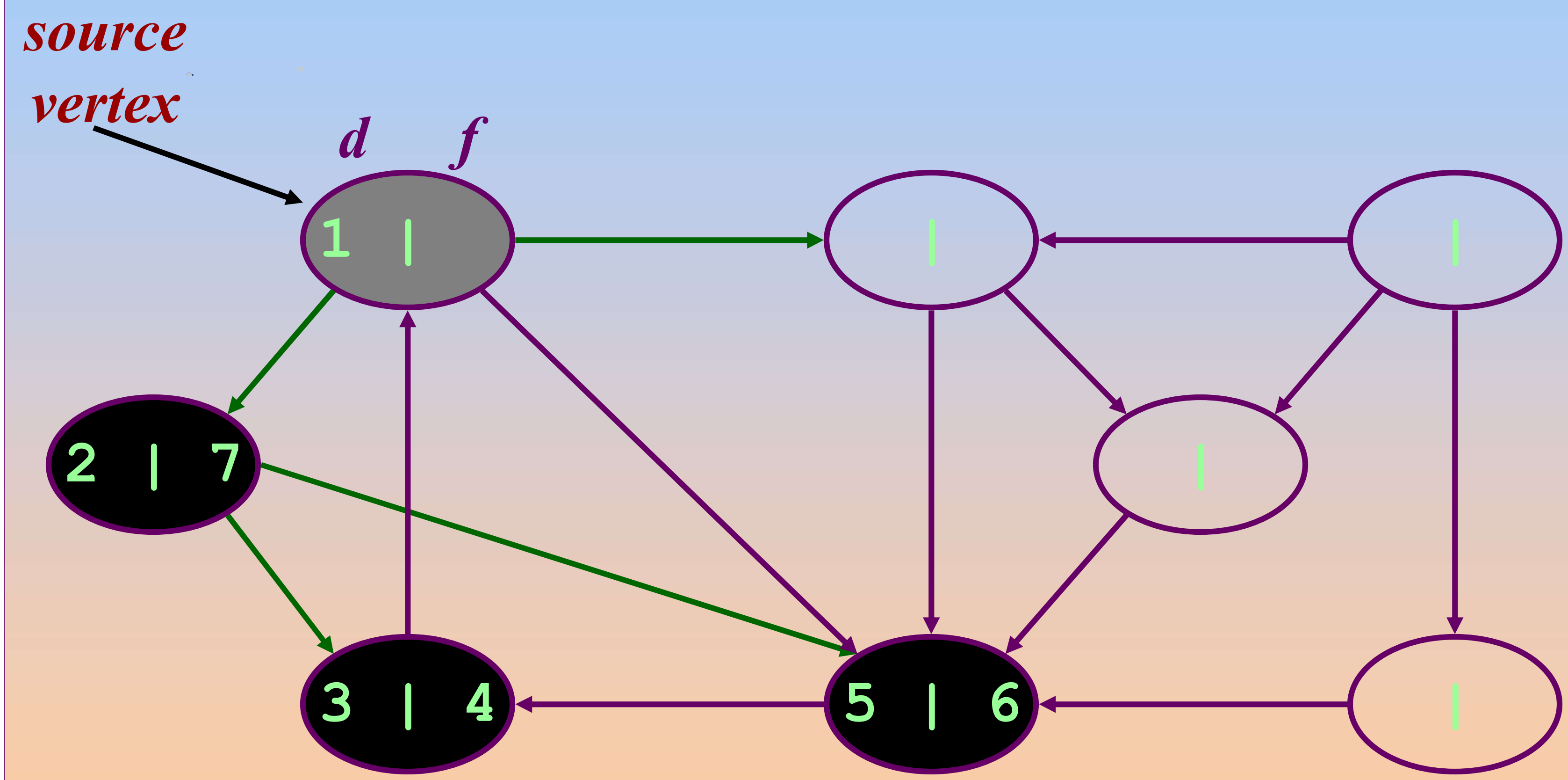


DFS Example

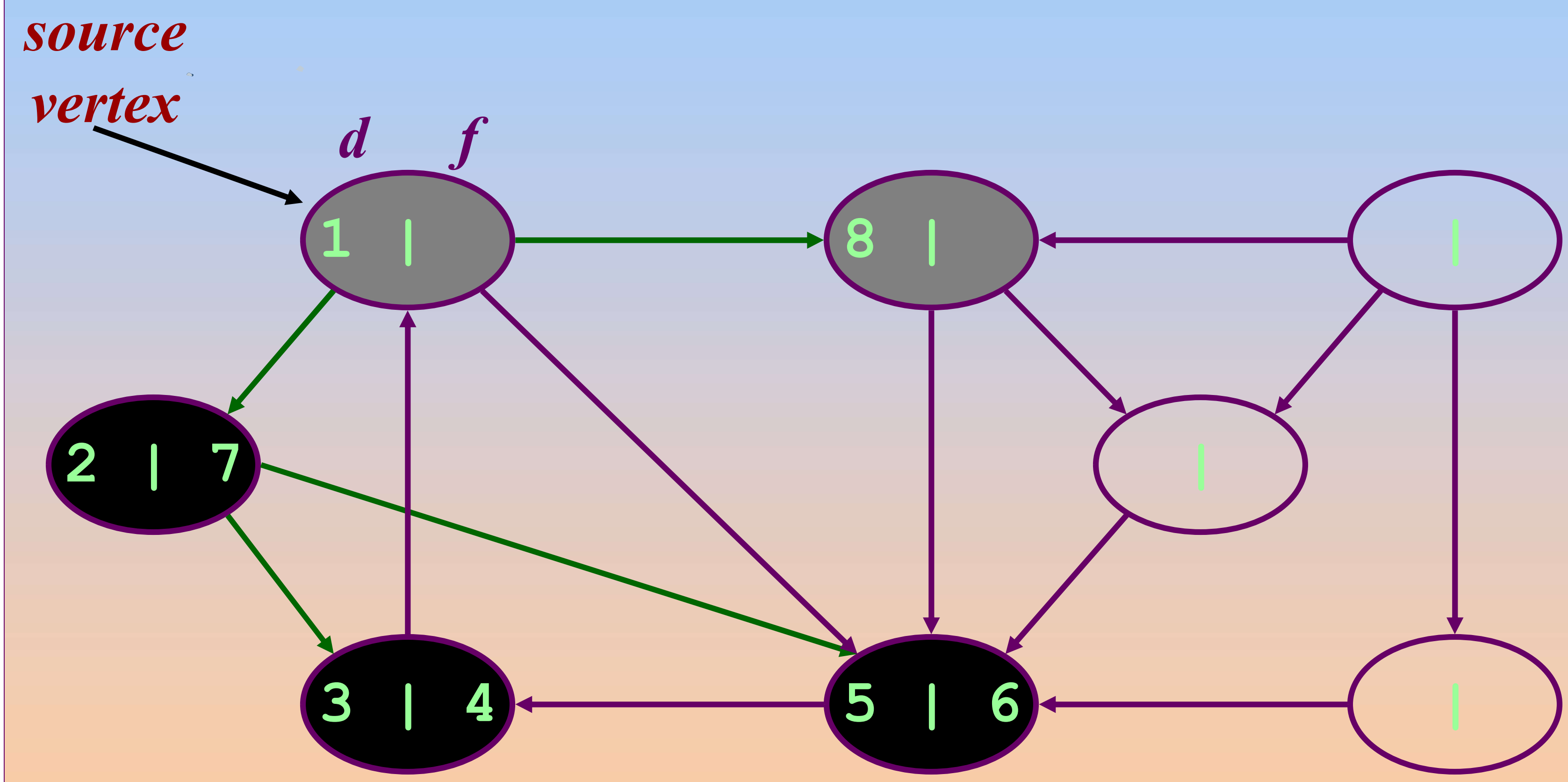
source vertex



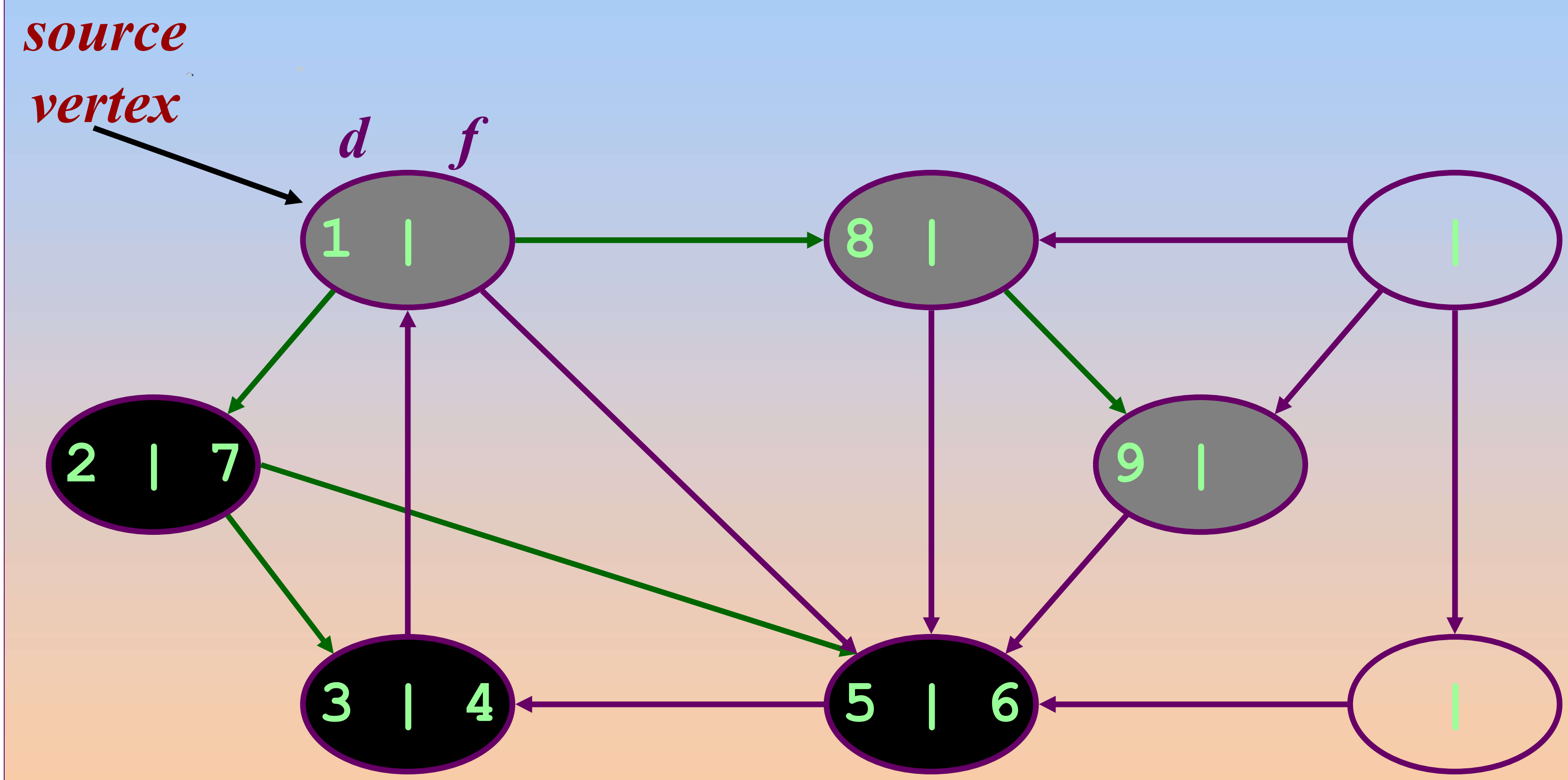
DFS Example



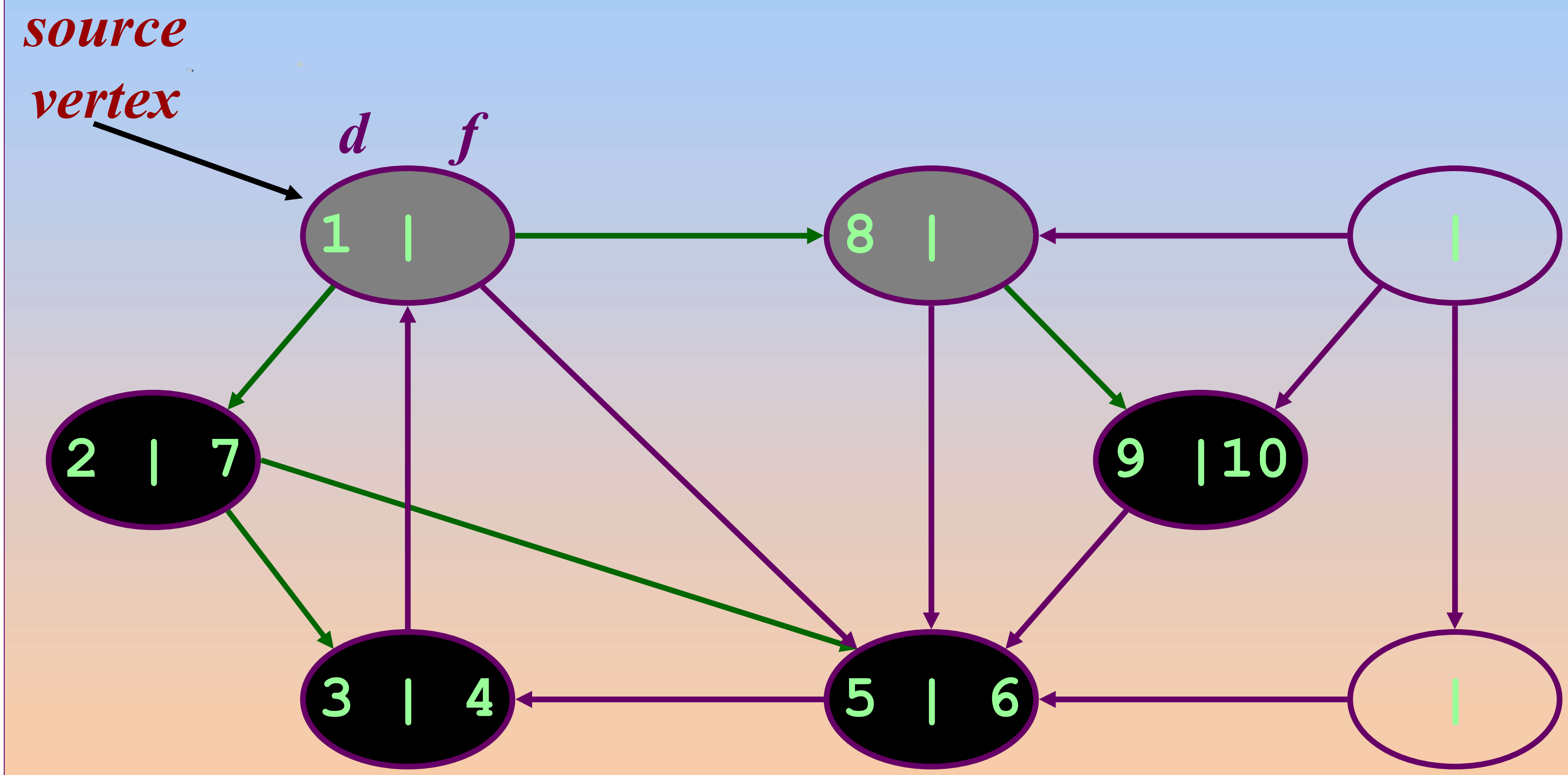
DFS Example



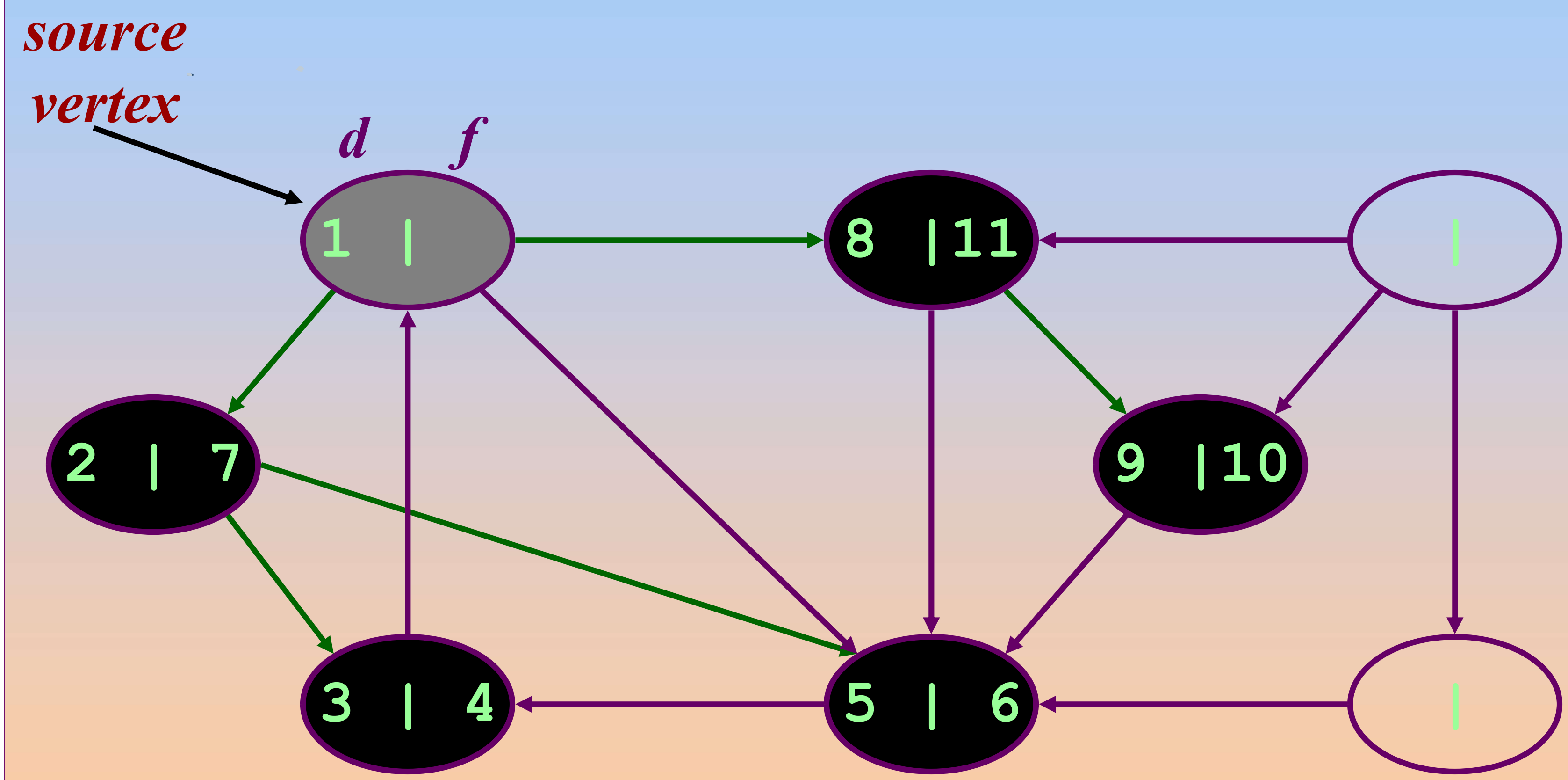
DFS Example



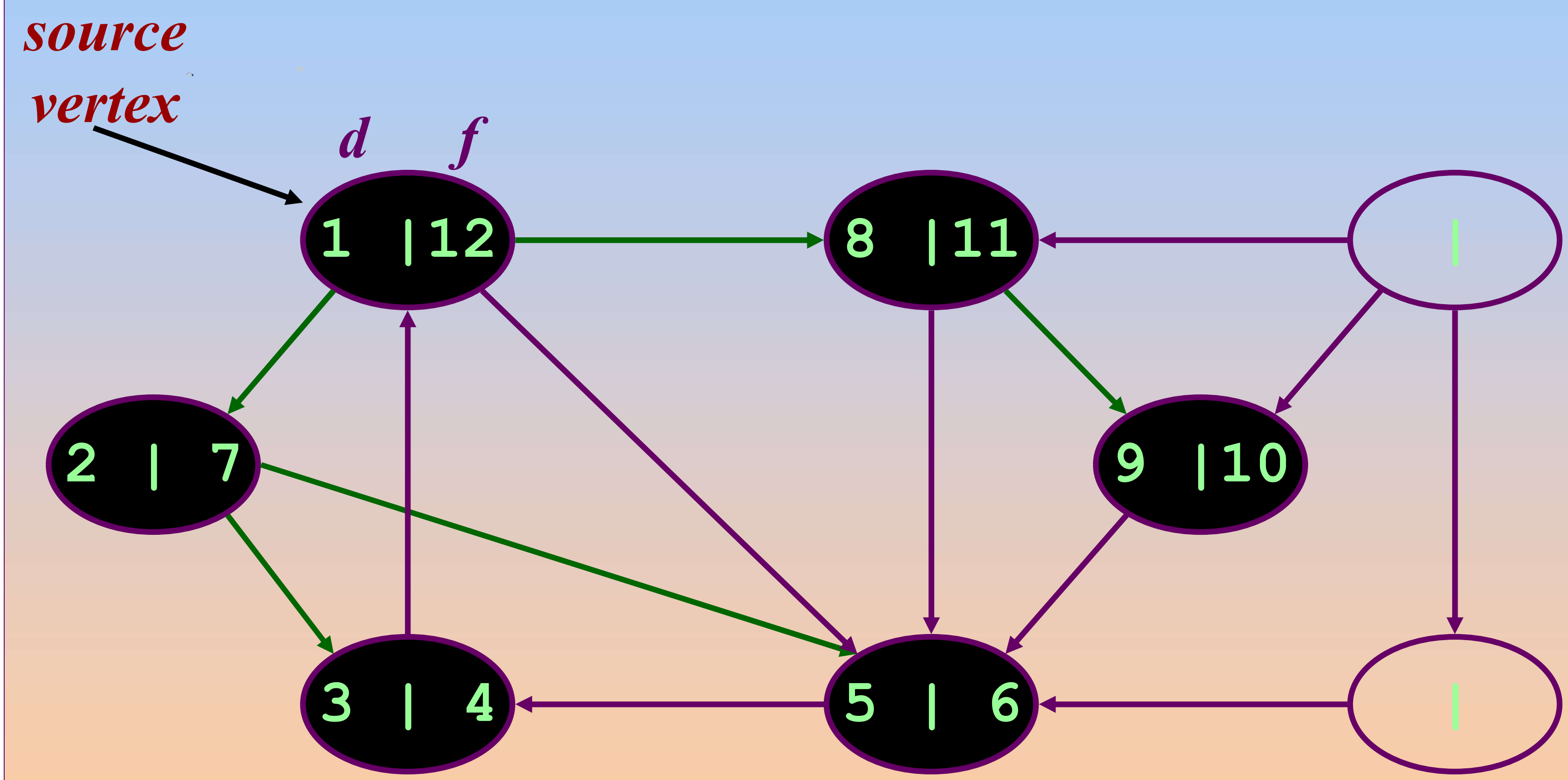
DFS Example

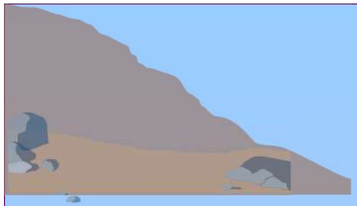


DFS Example

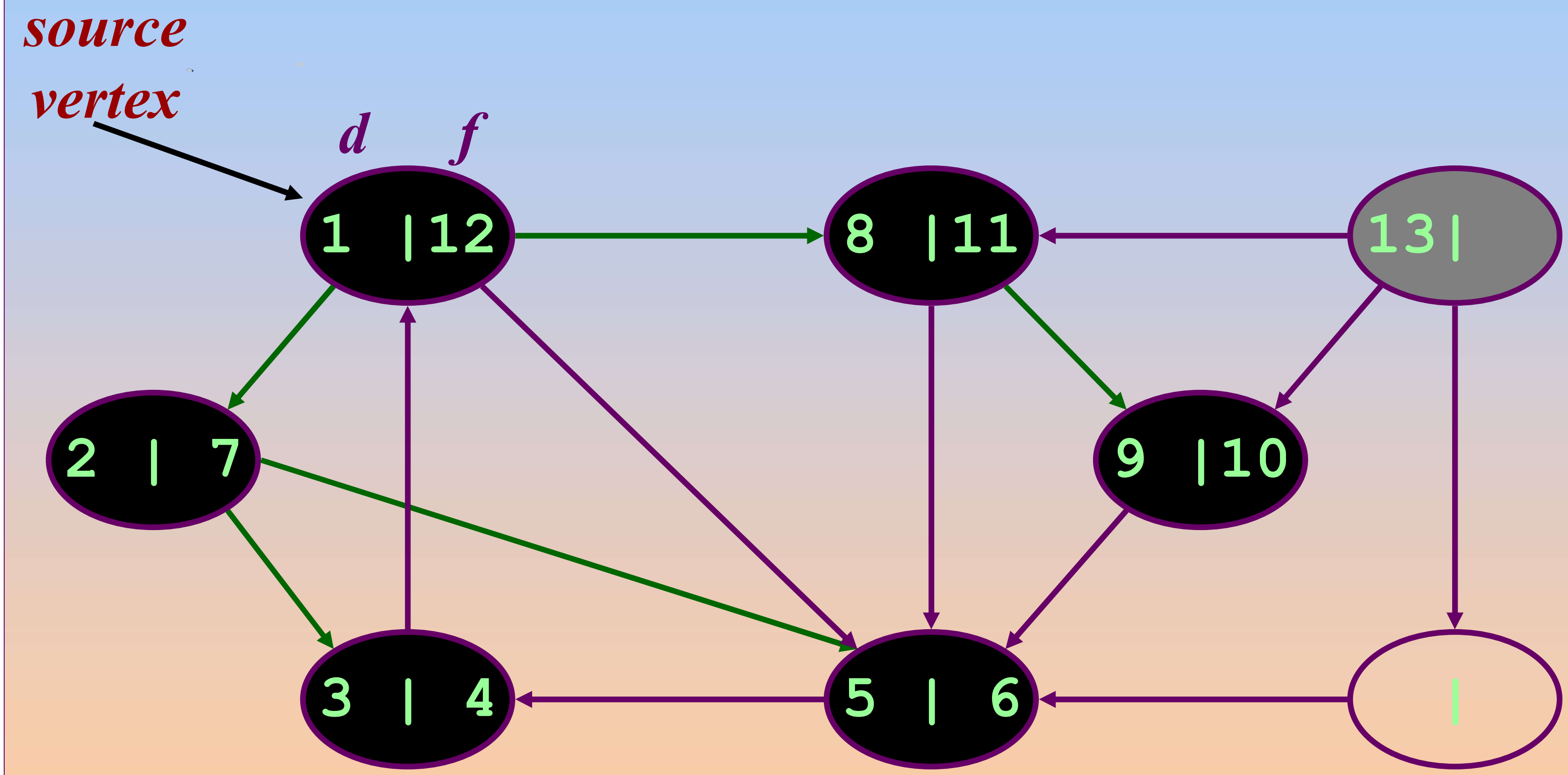


DFS Example



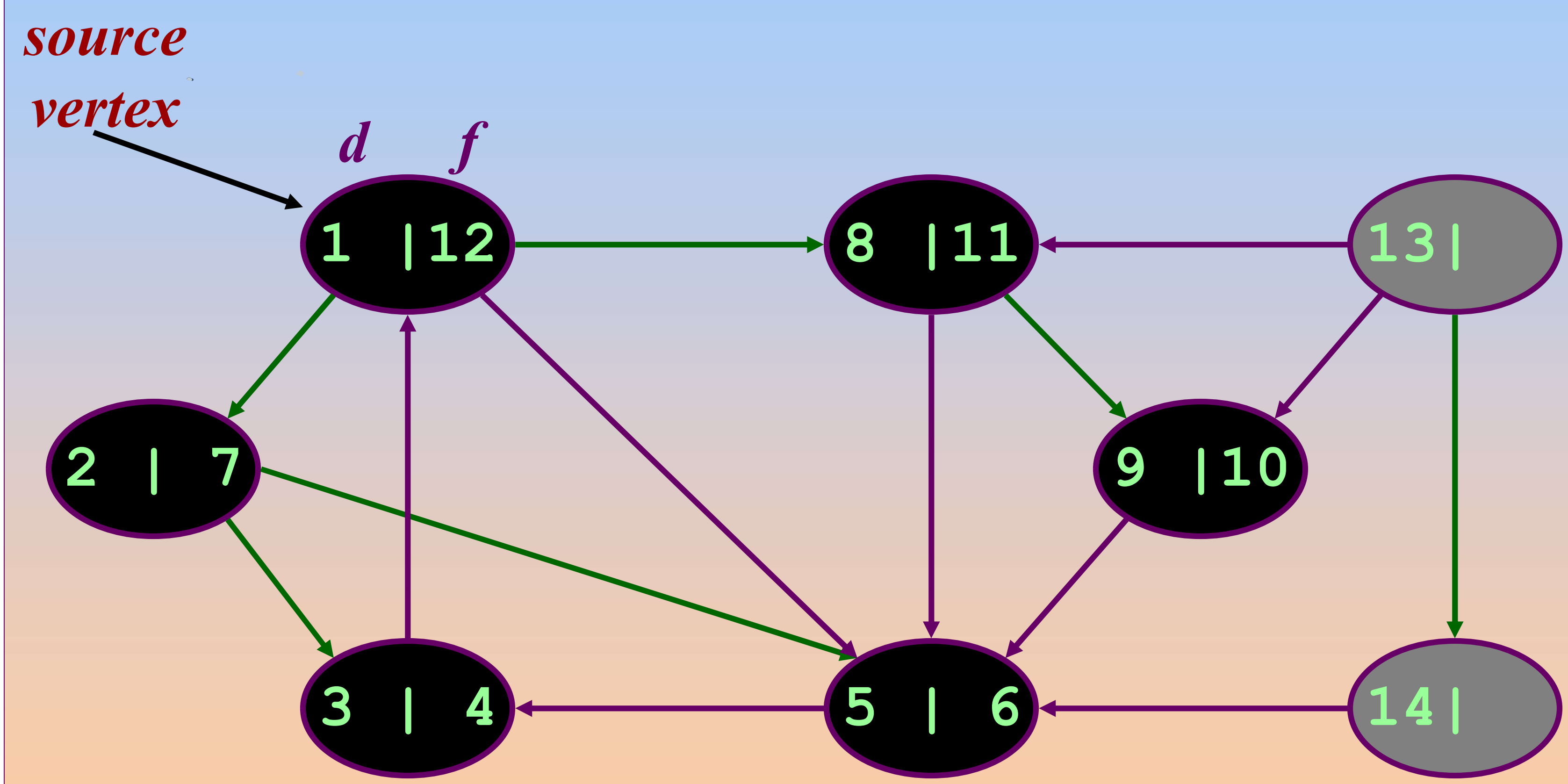


DFS Example

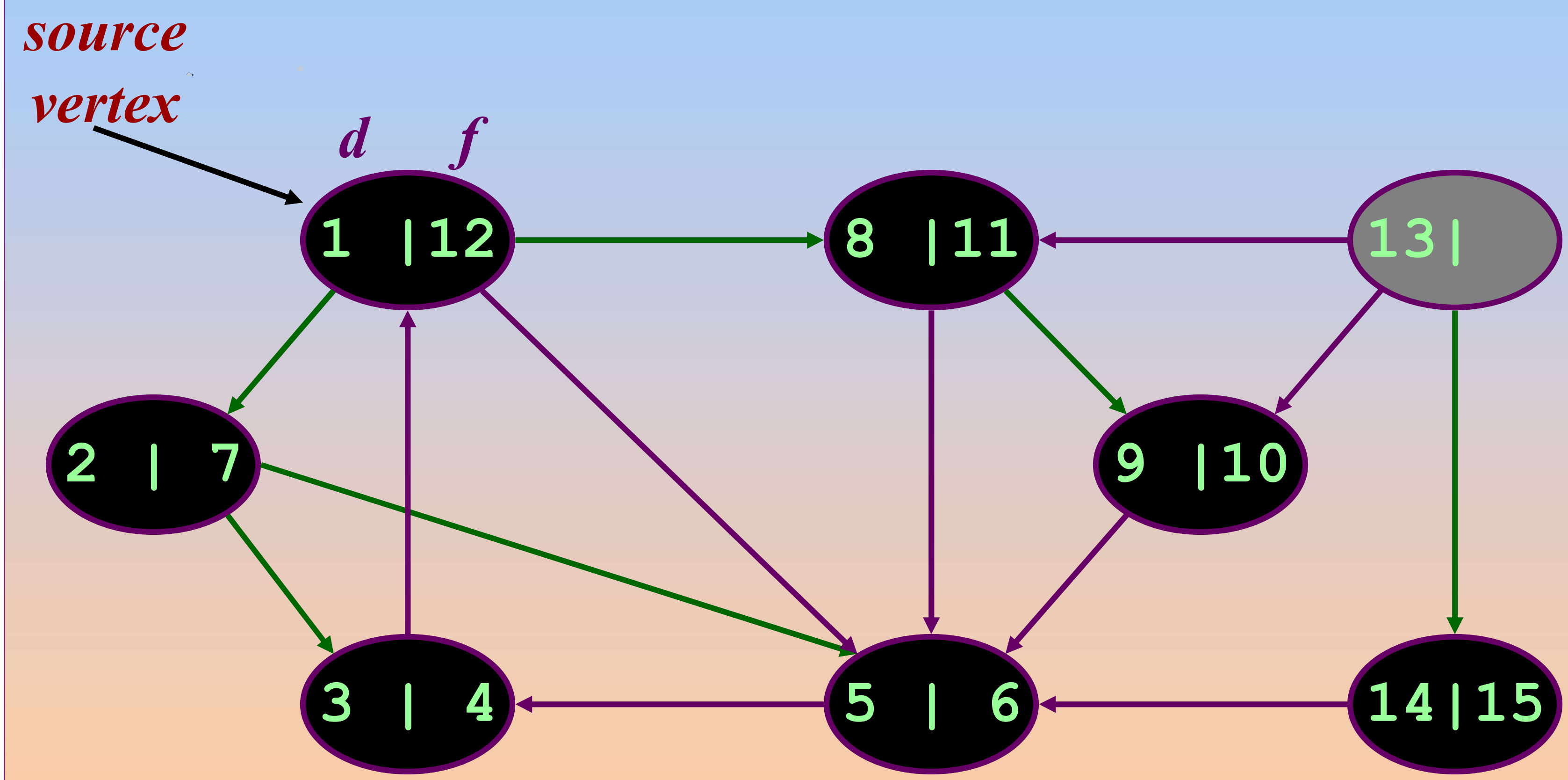


source vertex

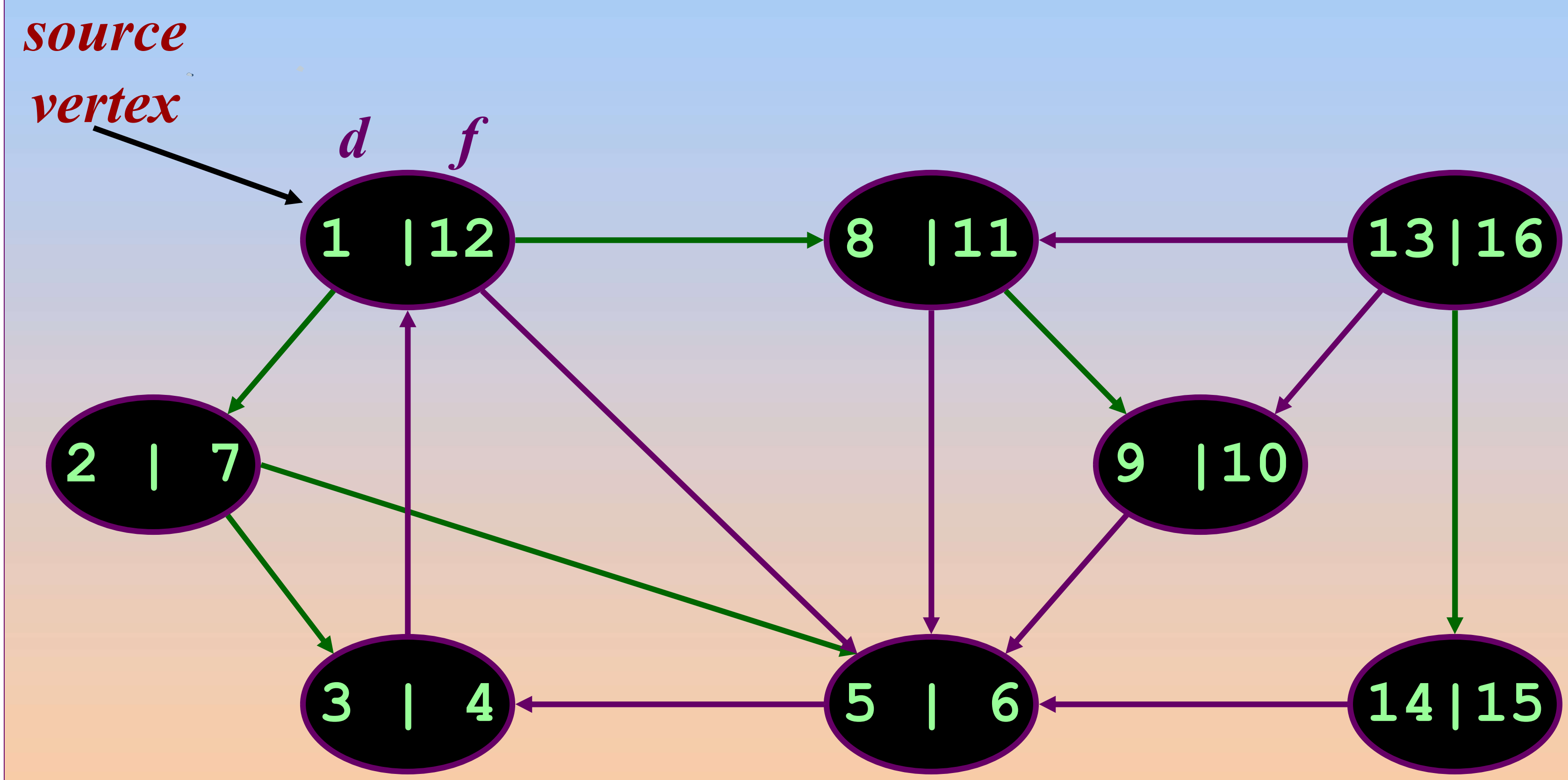
DFS Example

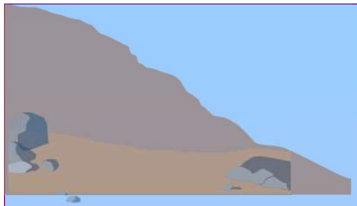


DFS Example

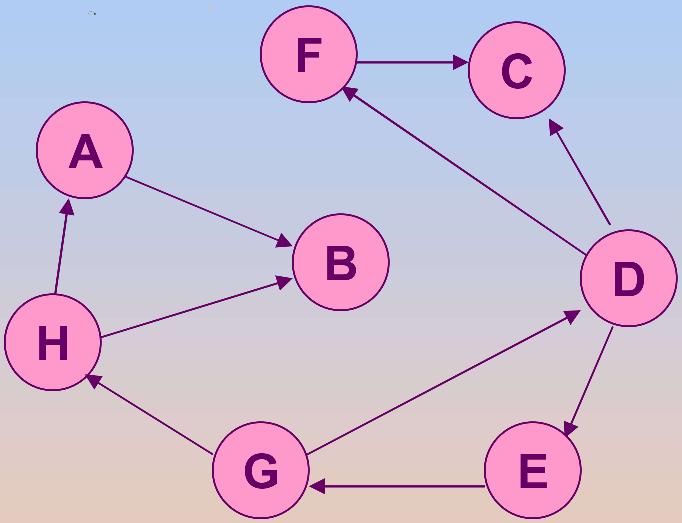


DFS Example





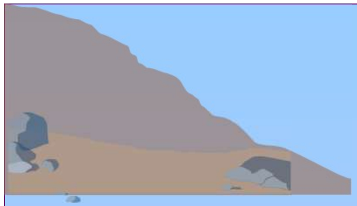
Walk-Through



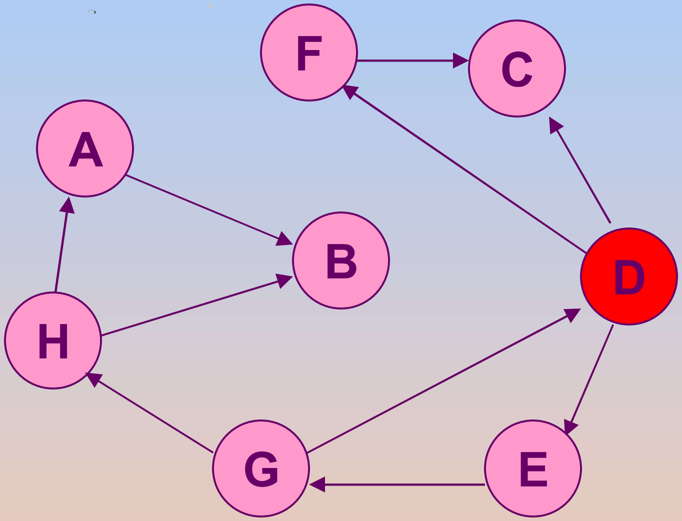
Visited Array

A	
B	
C	
D	
E	
F	
G	
H	

Task: Conduct a depth-first search of the graph starting with node D

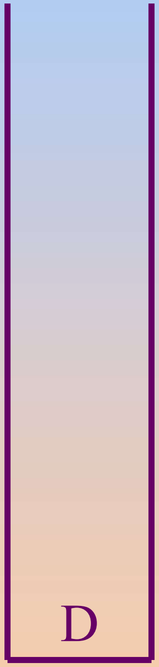


Walk-Through



Visited Array

A	
B	
C	
D	√
E	
F	
G	
H	

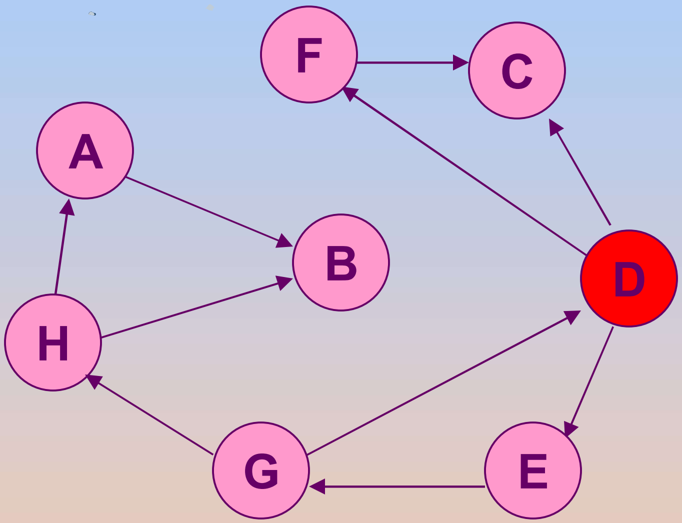


The order nodes are visited:

D

Visit D

Walk-Through



Visited Array

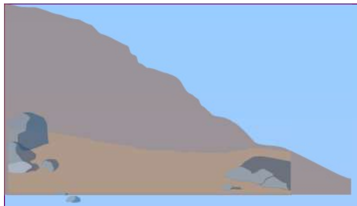
A	
B	
C	
D	√
E	
F	
G	
H	

D

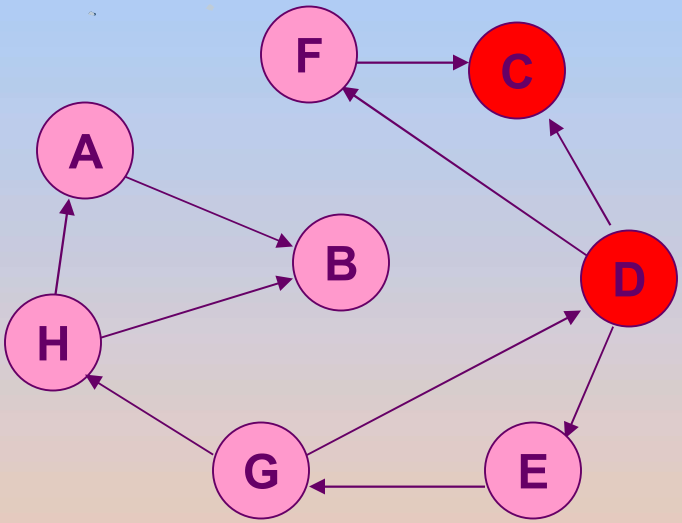
The order nodes are visited:

D

Consider nodes adjacent to D, decide to visit C first (Rule: visit adjacent nodes in alphabetical order)



Walk-Through



Visited Array

A	
B	
C	√
D	√
E	
F	
G	
H	

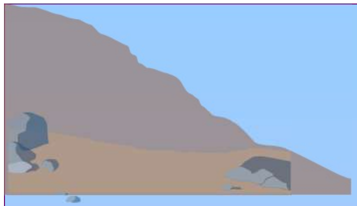
C

D

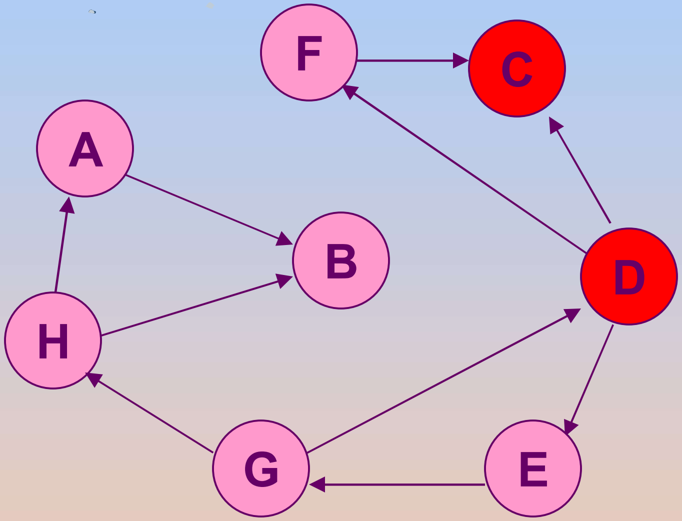
Visit C

The order nodes are visited:

D, C



Walk-Through



Visited Array

A	
B	
C	√
D	√
E	
F	
G	
H	

C

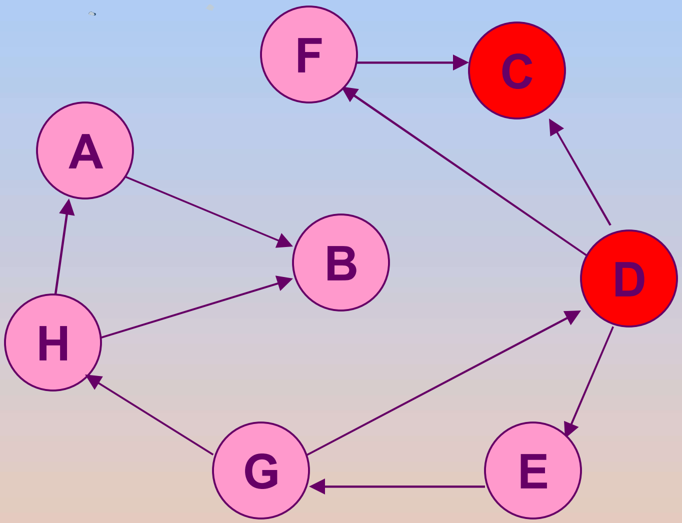
D

The order nodes are visited:

D, C

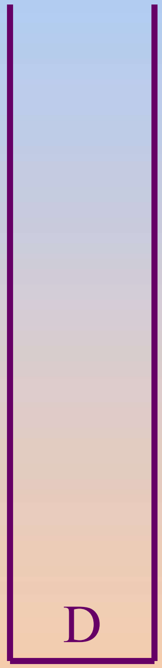
No nodes adjacent to C; cannot continue ➔ *backtrack*, i.e., pop stack and restore previous state

Walk-Through



Visited Array

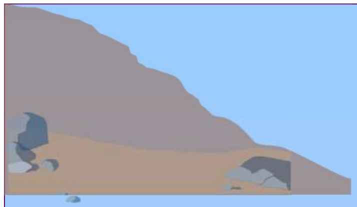
A	
B	
C	√
D	√
E	
F	
G	
H	



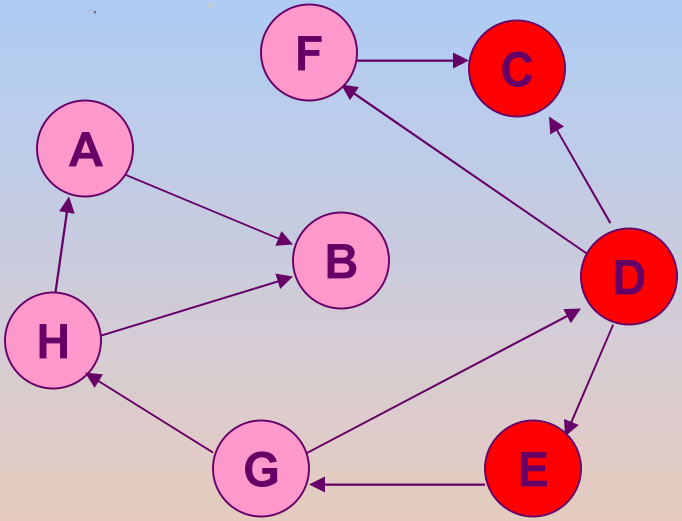
The order nodes are visited:

D, C

Back to D – C has been visited, decide to visit E next



Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	
H	

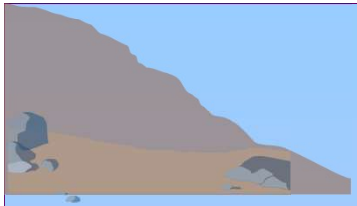
E

D

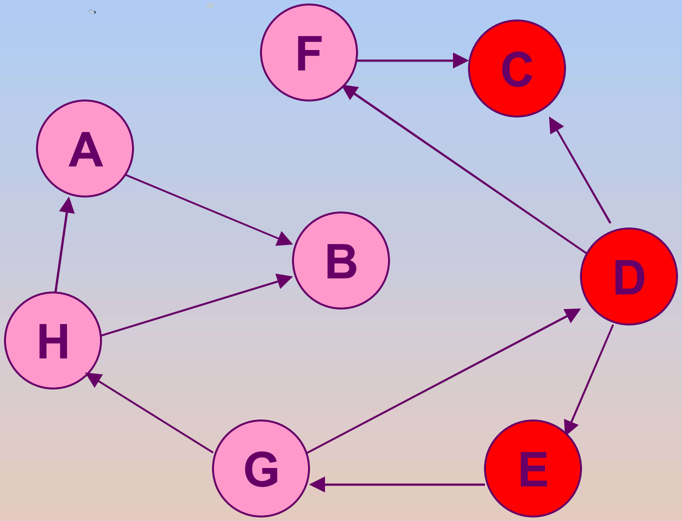
The order nodes are visited:

D, C, E

Back to D – C has been visited, decide to visit E next

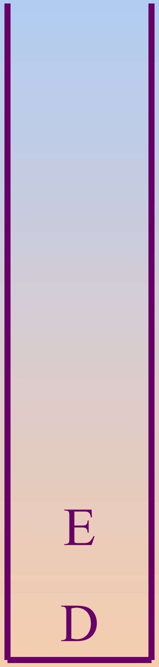


Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	
H	



E

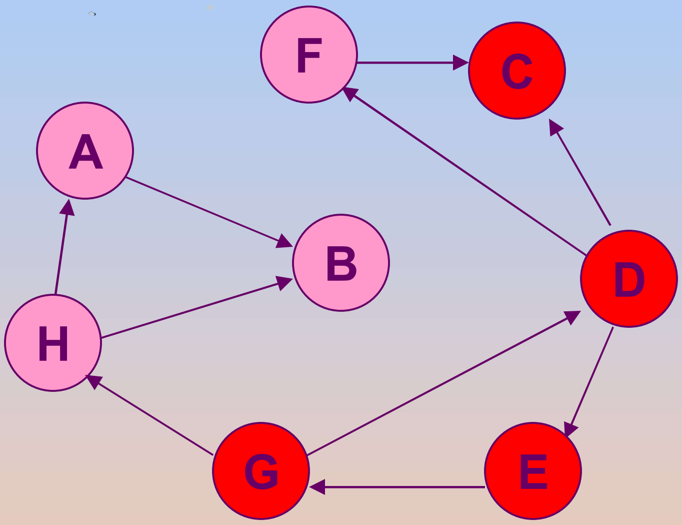
D

The order nodes are visited:

D, C, E

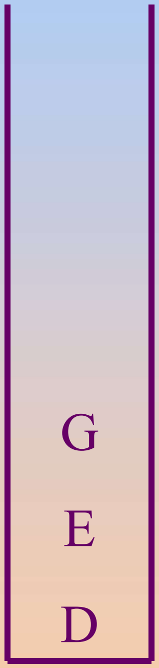
Only G is adjacent to E

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	√
H	



G

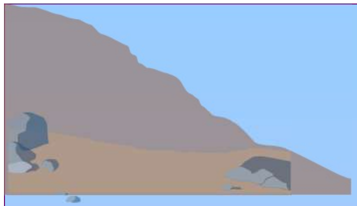
E

D

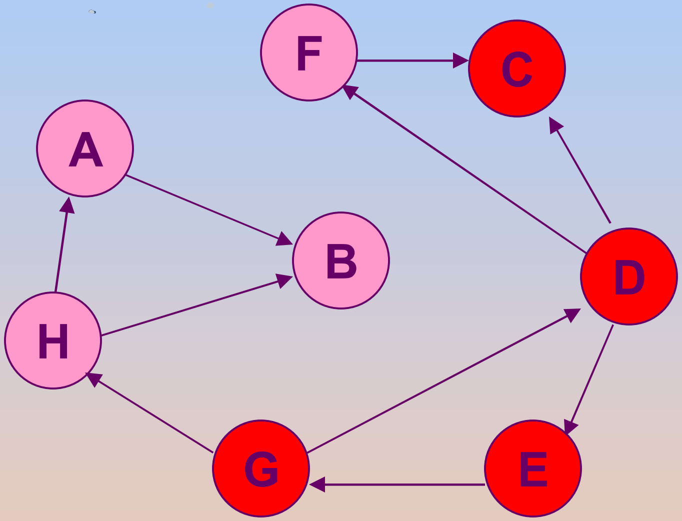
Visit G

The order nodes are visited:

D, C, E, G

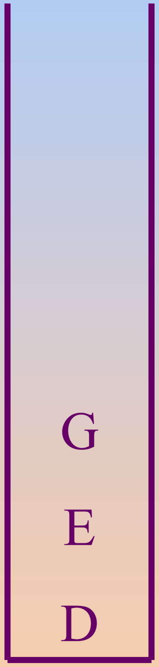


Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	√
H	



G

E

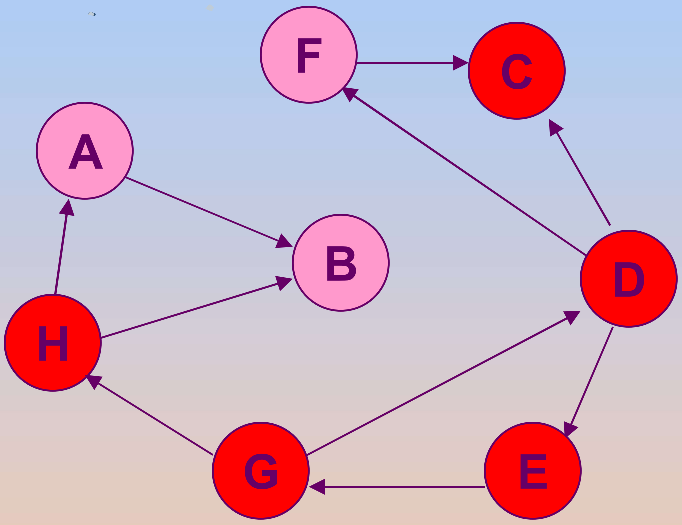
D

The order nodes are visited:

D, C, E, G

Nodes D and H are adjacent to G. D has already been visited. Decide to visit H.

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	√
H	√

H

G

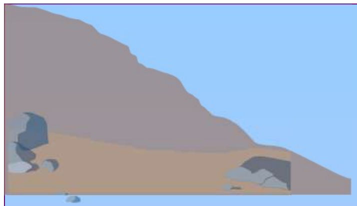
E

D

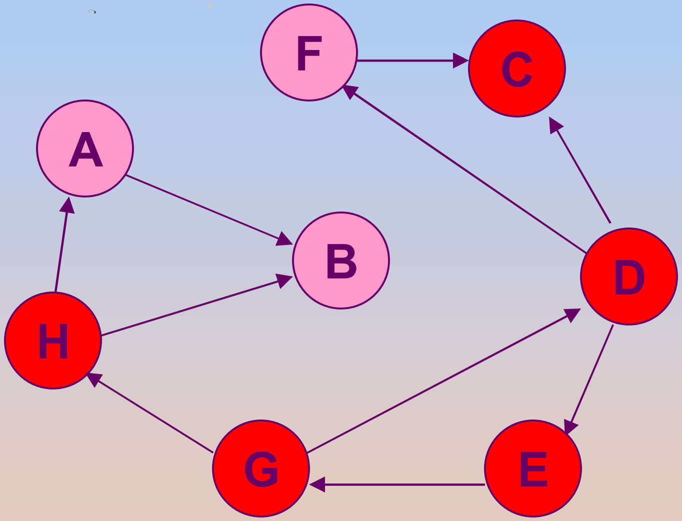
Visit H

The order nodes are visited:

D, C, E, G, H



Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	√
H	√

H

G

E

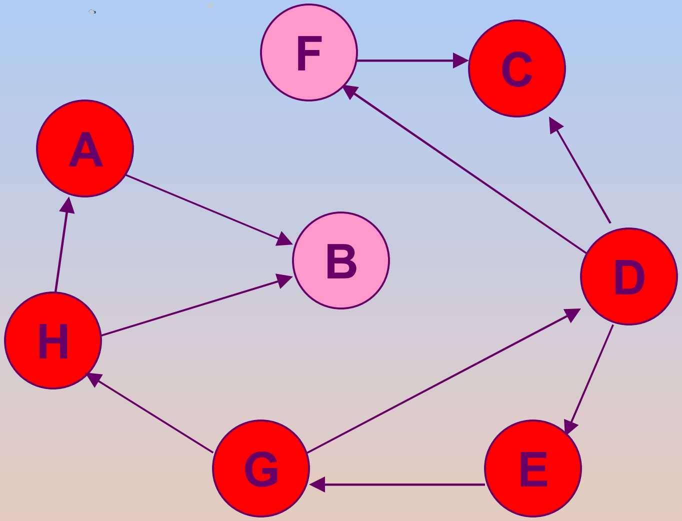
D

The order nodes are visited:

D, C, E, G, H

Nodes A and B are adjacent to F. Decide to visit A next.

Walk-Through



Visited Array

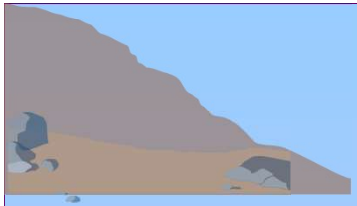
A	√
B	
C	√
D	√
E	√
F	
G	√
H	√

A
H
G
E
D

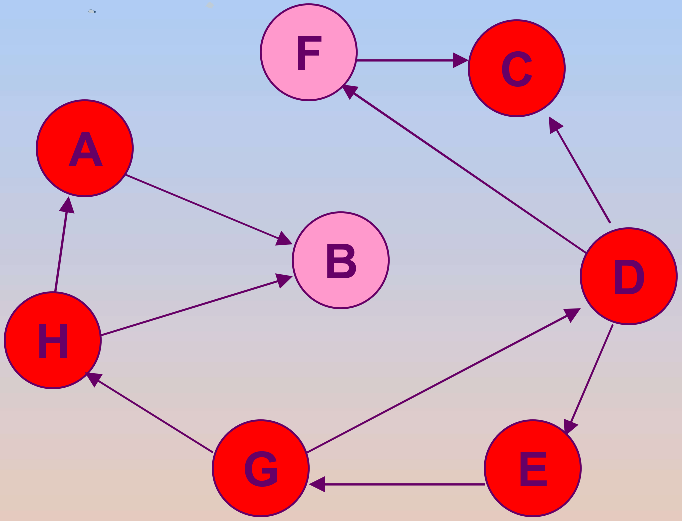
Visit A

The order nodes are visited:

D, C, E, G, H, A



Walk-Through



Visited Array

A	√
B	
C	√
D	√
E	√
F	
G	√
H	√

A

H

G

E

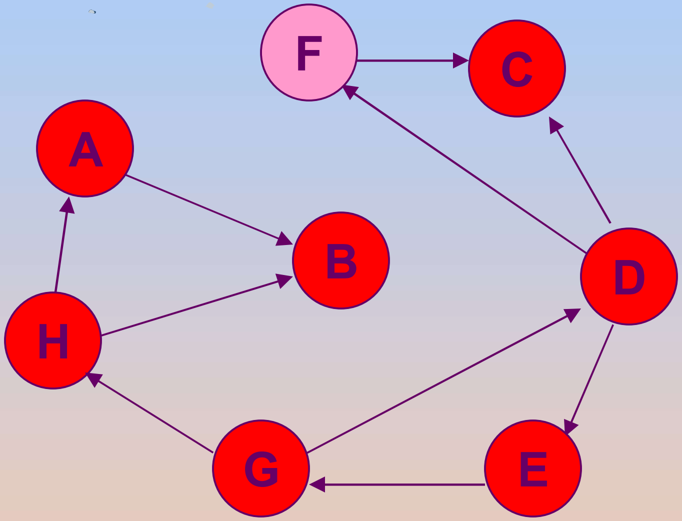
D

The order nodes are visited:

D, C, E, G, H, A

Only Node B is adjacent to A.
Decide to visit B next.

Walk-Through



Visited Array

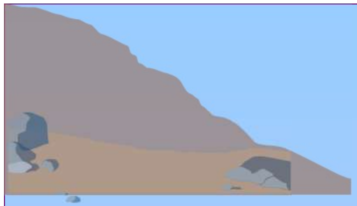
A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

B
A
H
G
E
D

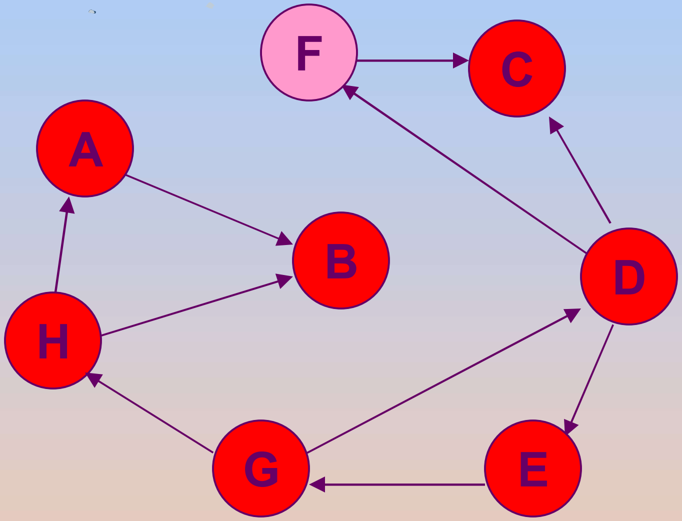
Visit B

The order nodes are visited:

D, C, E, G, H, A, B



Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

A

H

G

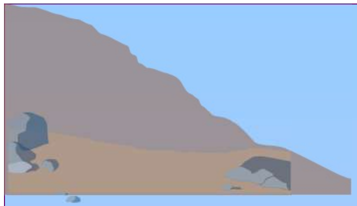
E

D

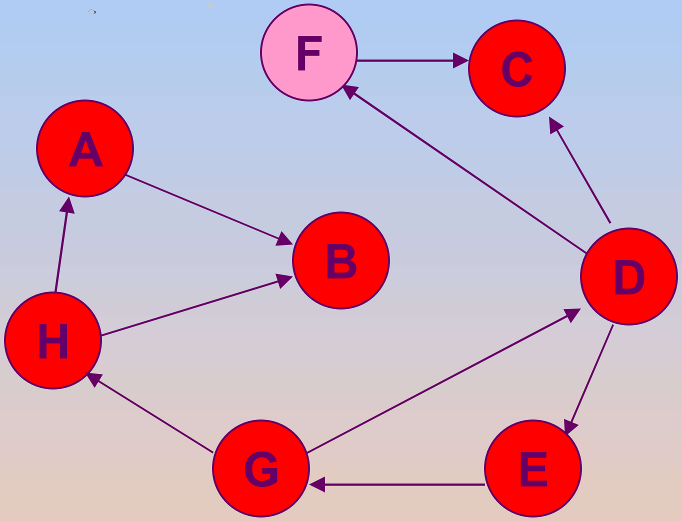
The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to B. Backtrack (pop the stack).



Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

H

G

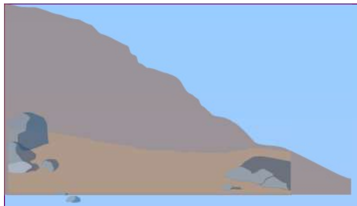
E

D

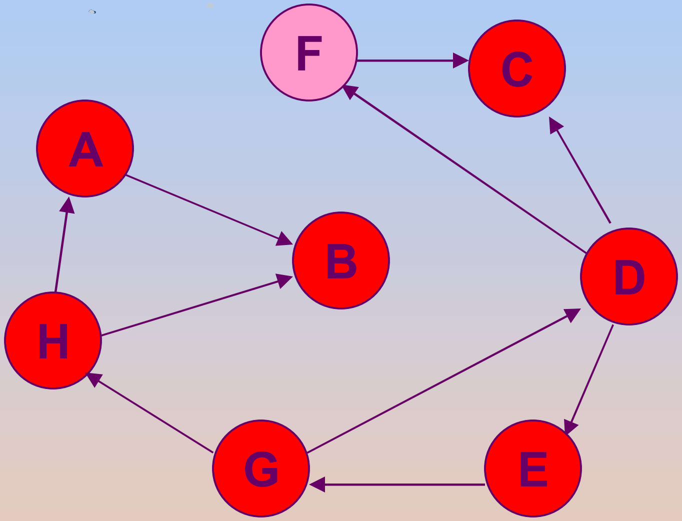
The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to A. Backtrack (pop the stack).



Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

G

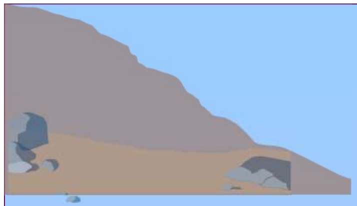
E

D

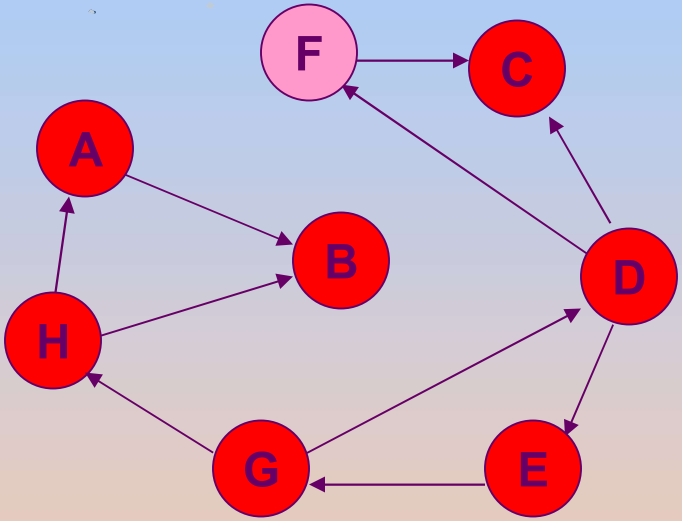
The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to H. Backtrack (pop the stack).



Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

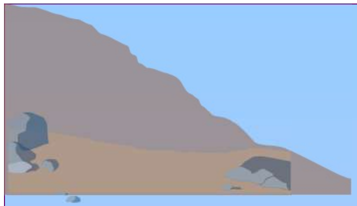
E

D

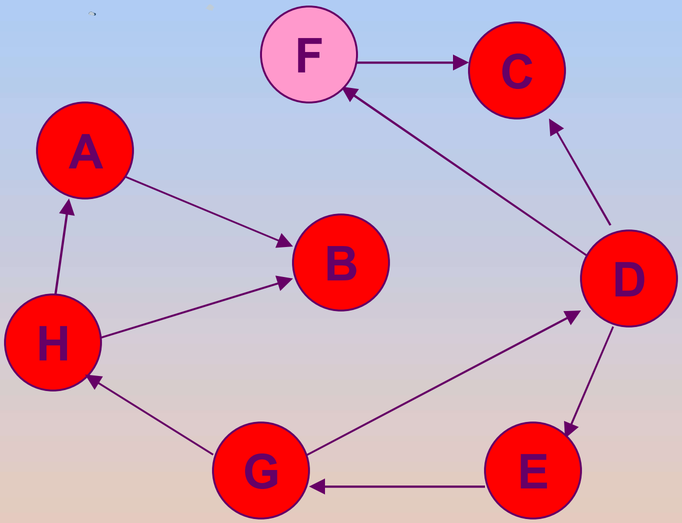
The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to G. Backtrack (pop the stack).

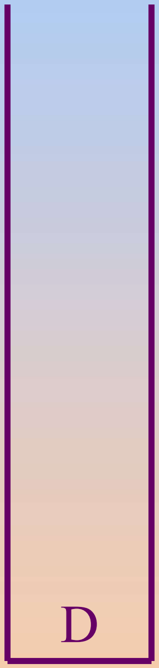


Walk-Through



Visited Array

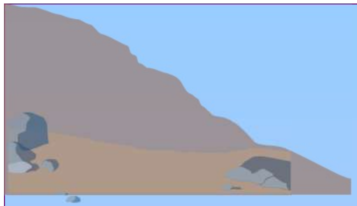
A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√



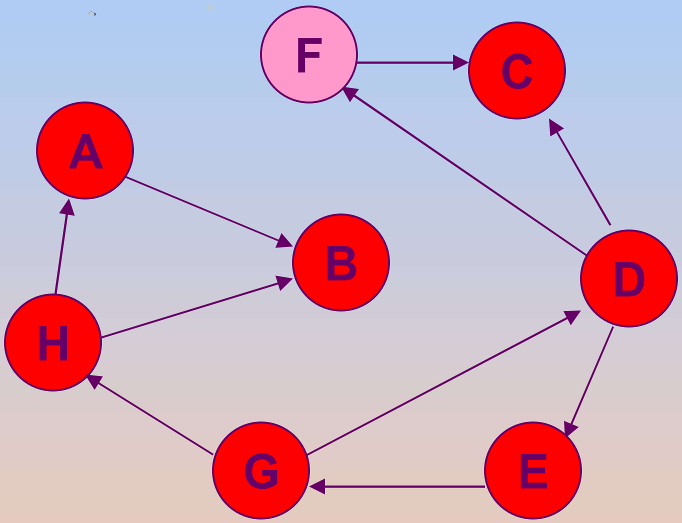
The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to E. Backtrack (pop the stack).

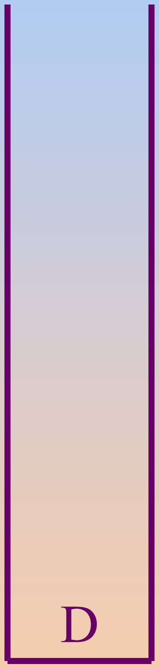


Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

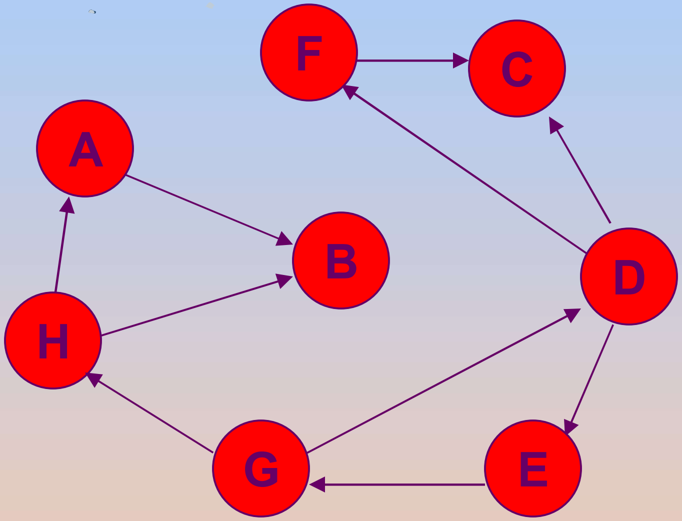


The order nodes are visited:

D, C, E, G, H, A, B

F is unvisited and is adjacent to D. Decide to visit F next.

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

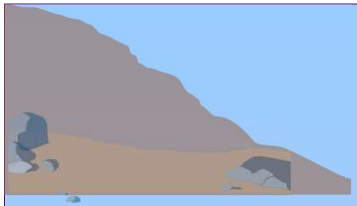
F

D

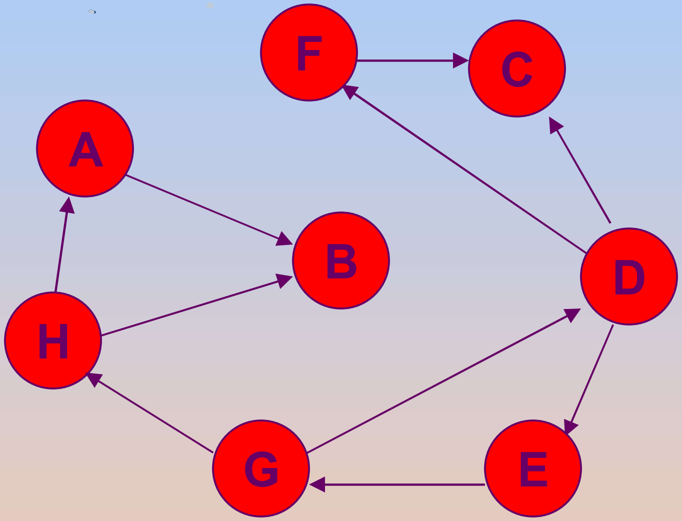
Visit F

The order nodes are visited:

D, C, E, G, H, A, B, F

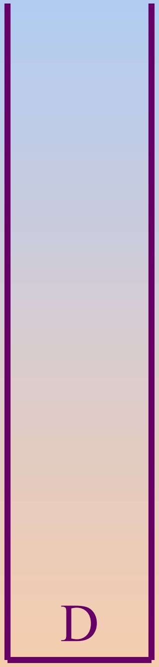


Walk-Through



Visited Array

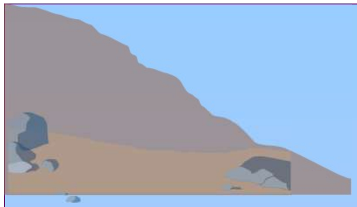
A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√



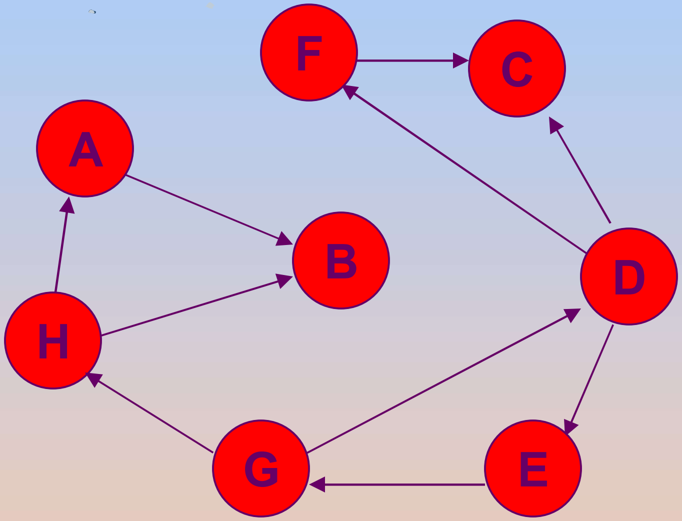
The order nodes are visited:

D, C, E, G, H, A, B, F

No unvisited nodes adjacent to F. Backtrack.



Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

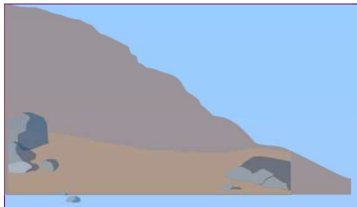
The order nodes are visited:

D, C, E, G, H, A, B, F

No unvisited nodes adjacent to D. Backtrack.

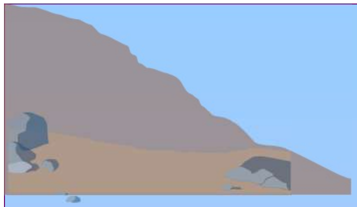
DFS: Application

- Topological Sort
- Strongly Connected Component



Breadth-first Search (BFS)

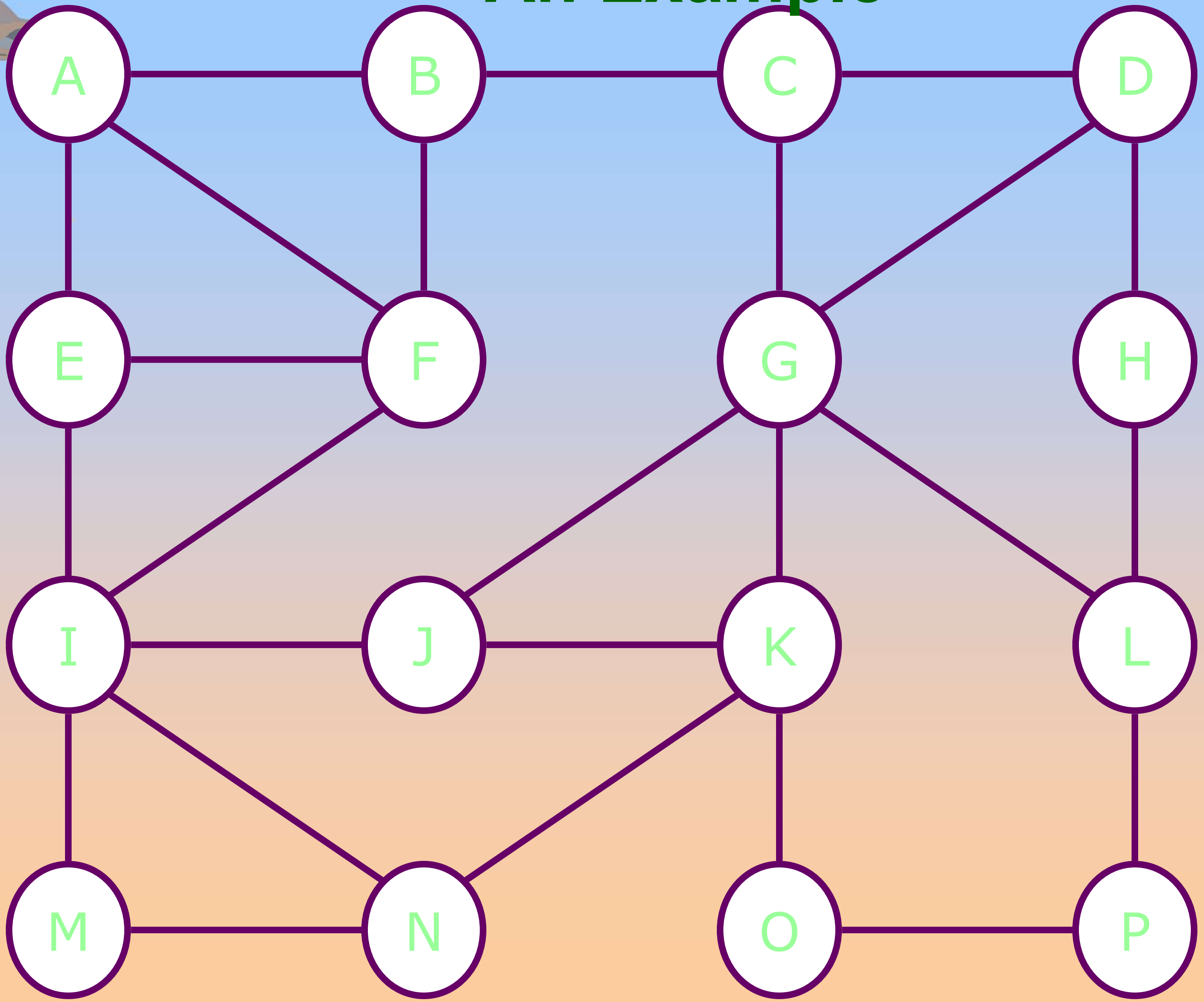
- Search for all vertices that are directly reachable from the root (called level 1 vertices)
- After mark all these vertices, visit all vertices that are directly reachable from any level 1 vertices (called level 2 vertices), and so on.
- In general, level k vertices are directly reachable from a level k – 1 vertices

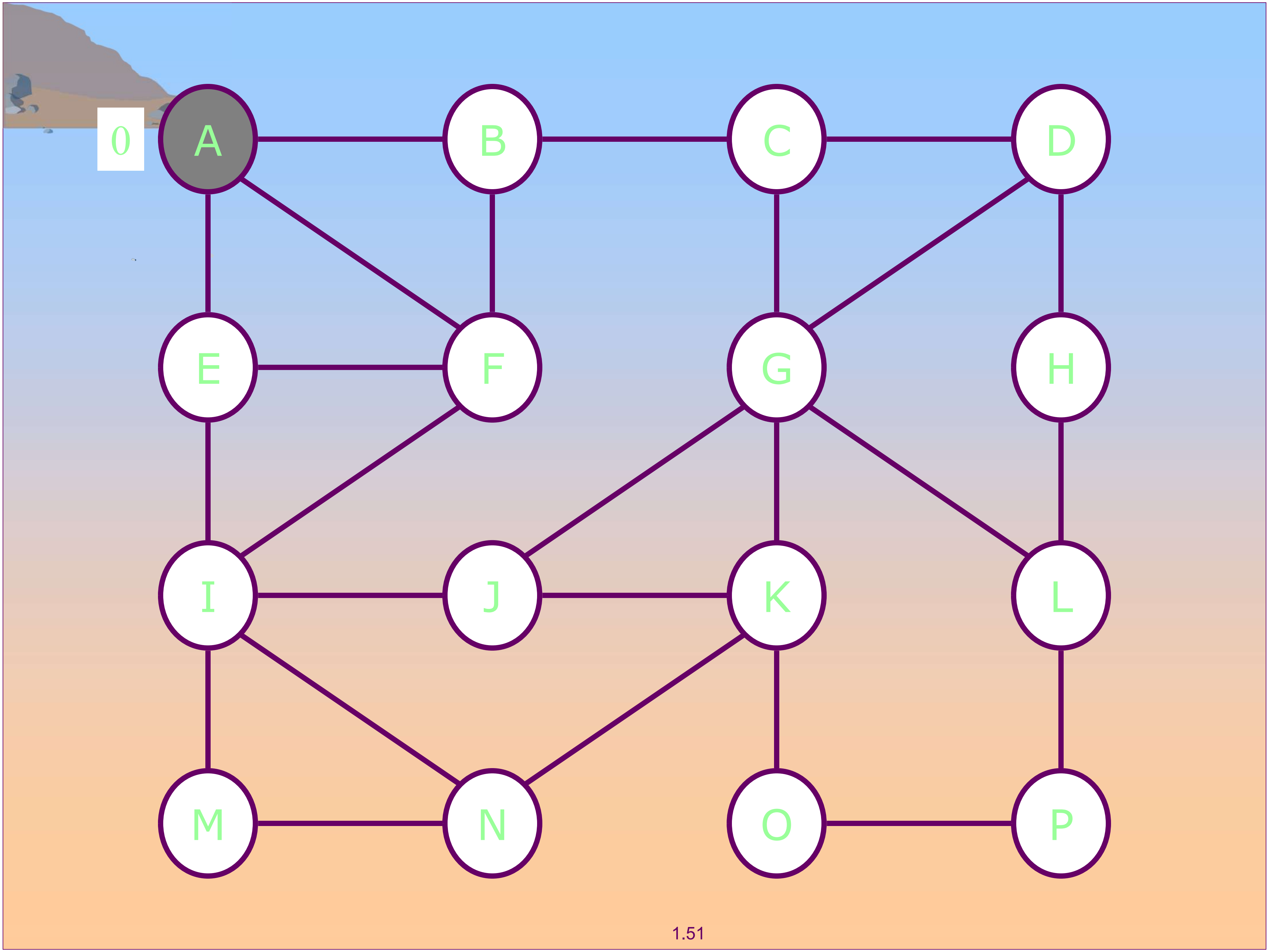


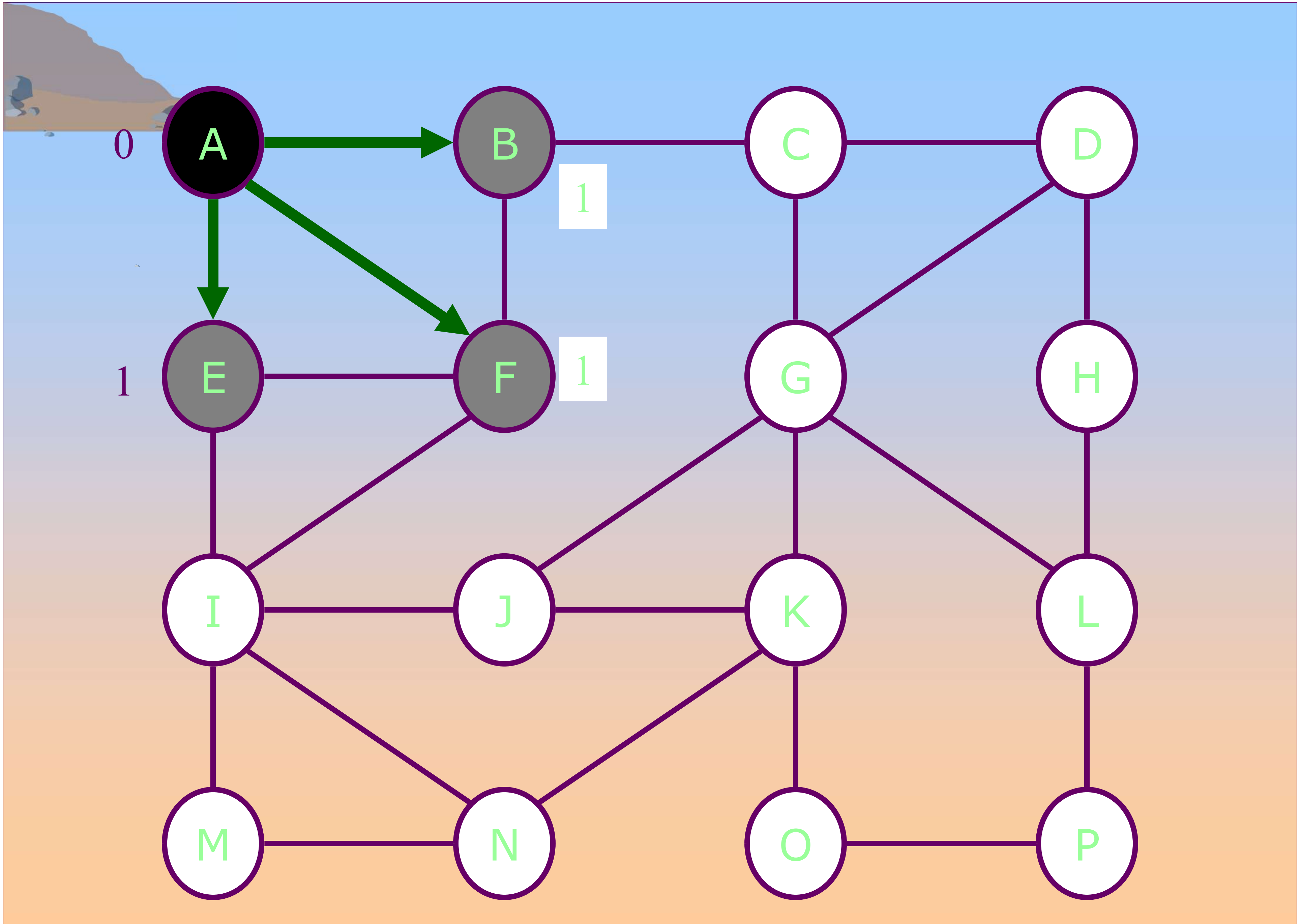
BFS: the Color Scheme

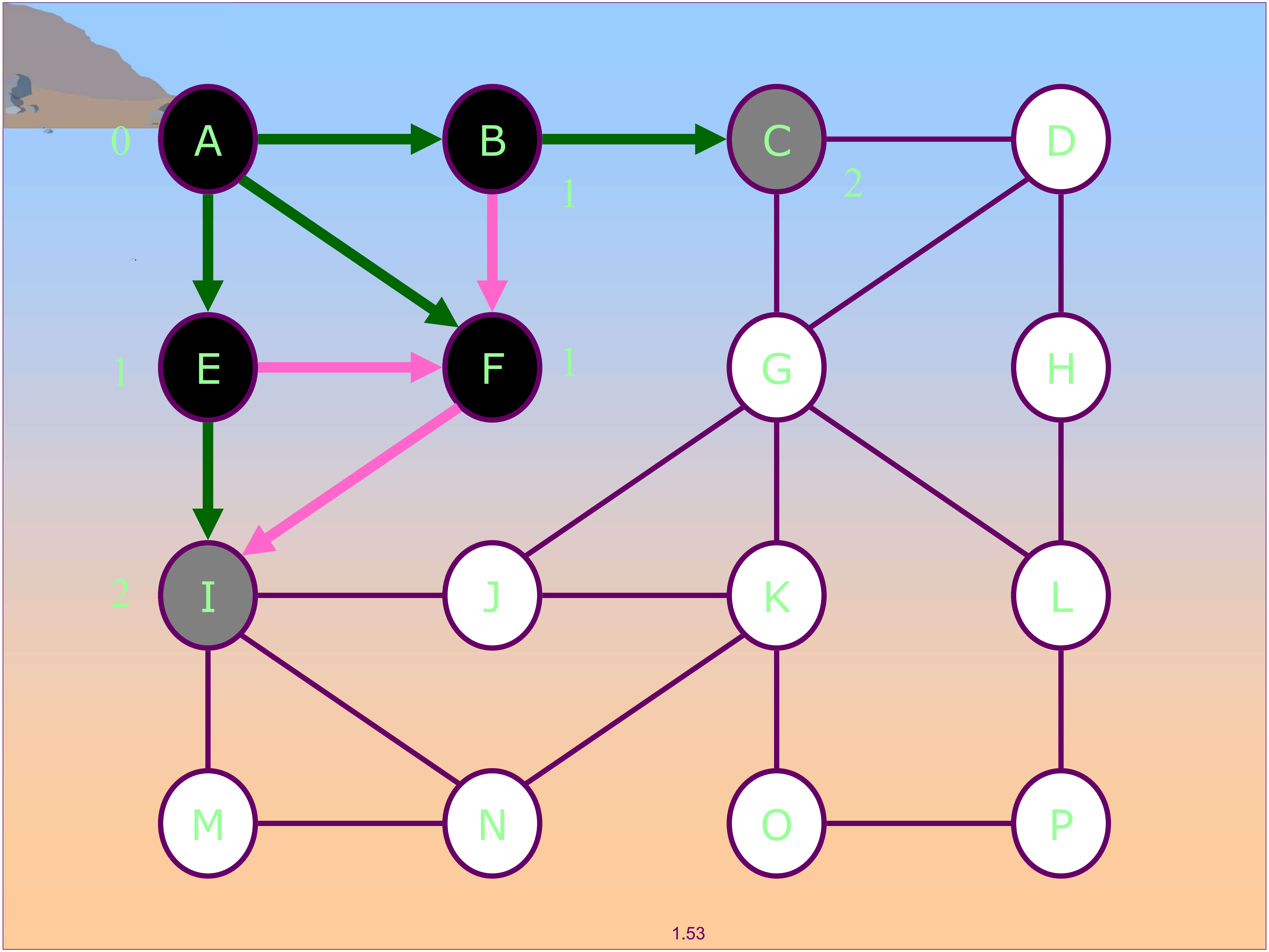
- White vertices have not been discovered
    - All vertices start out white
- Grey vertices are discovered but not fully explored
    - They may be adjacent to white vertices
- Black vertices are discovered and fully explored
    - They are adjacent only to black and gray vertices
- Explore vertices by scanning adjacency list of grey vertices

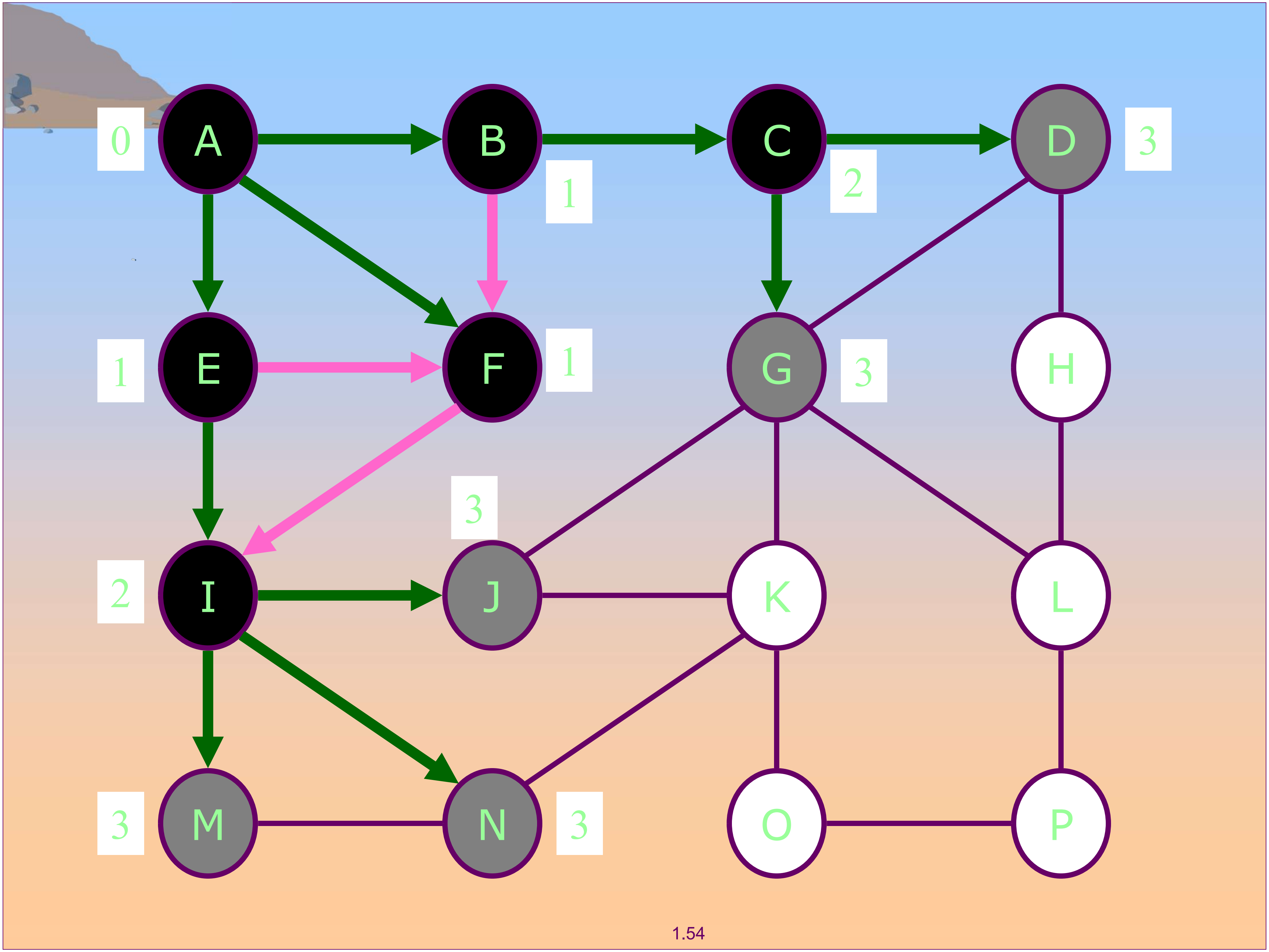
An Example

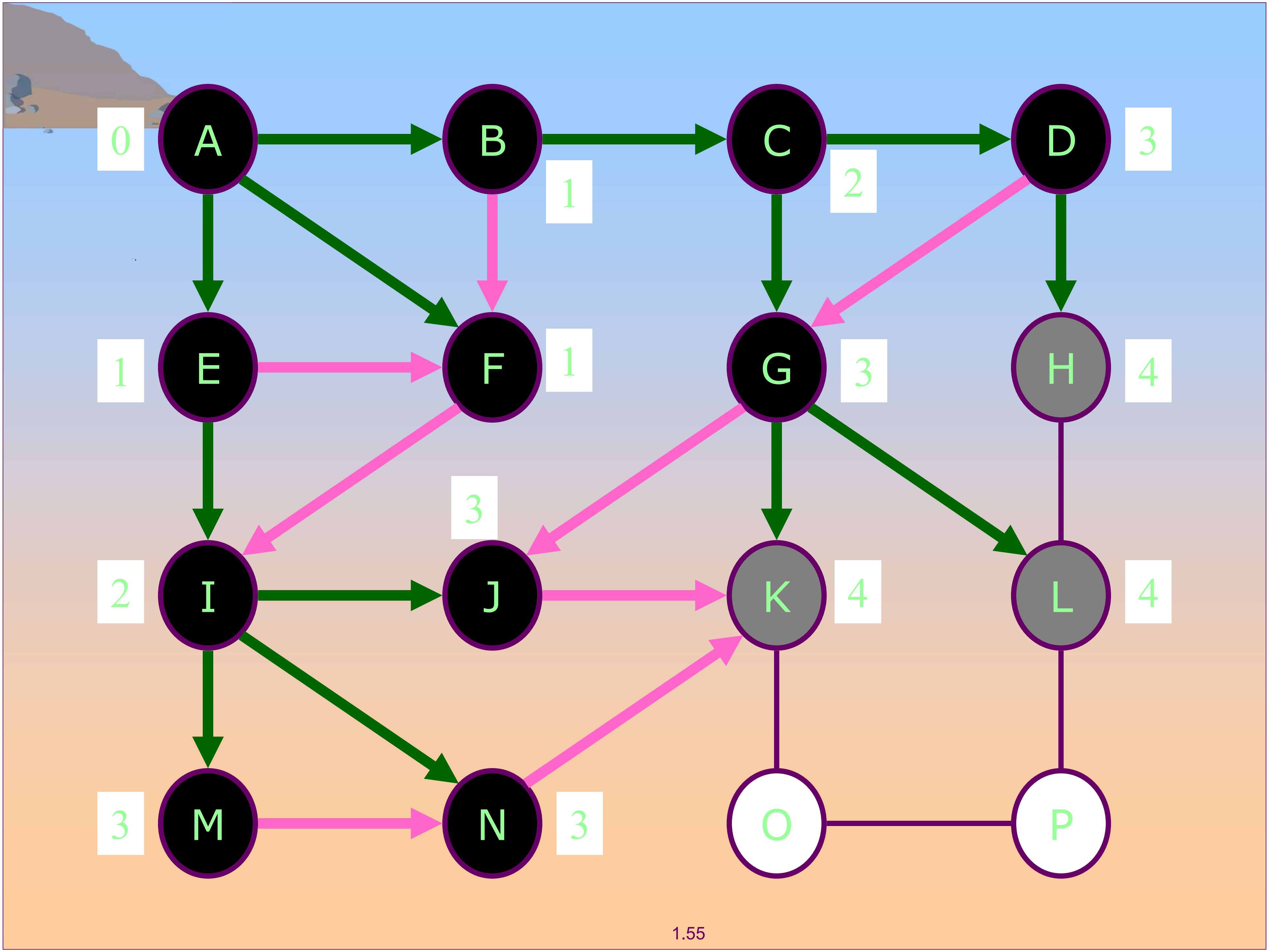


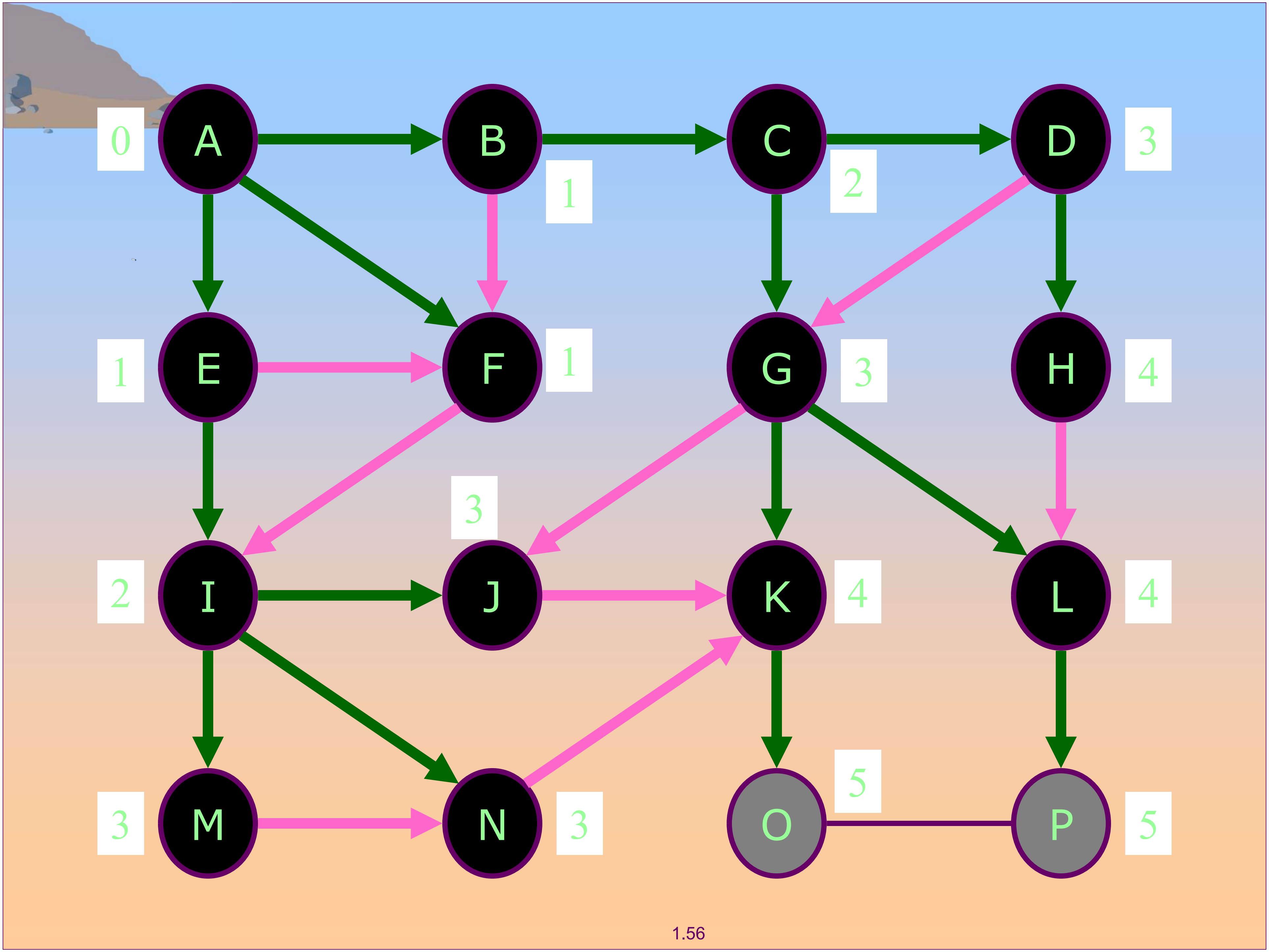

0
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P

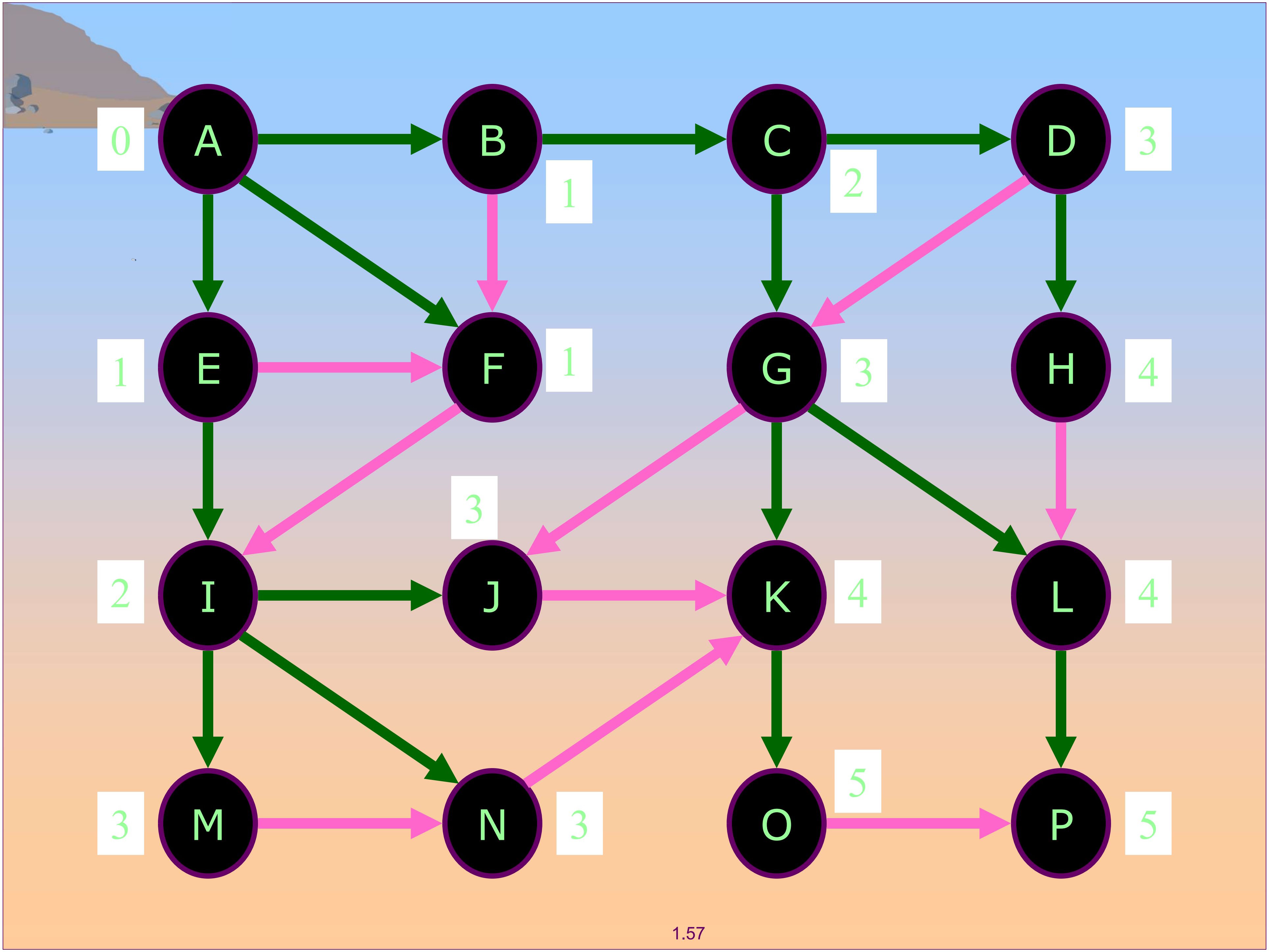

0
A
B
C
D
1
1
E
F
G
H
1
I
J
K
L
M
N
O
P

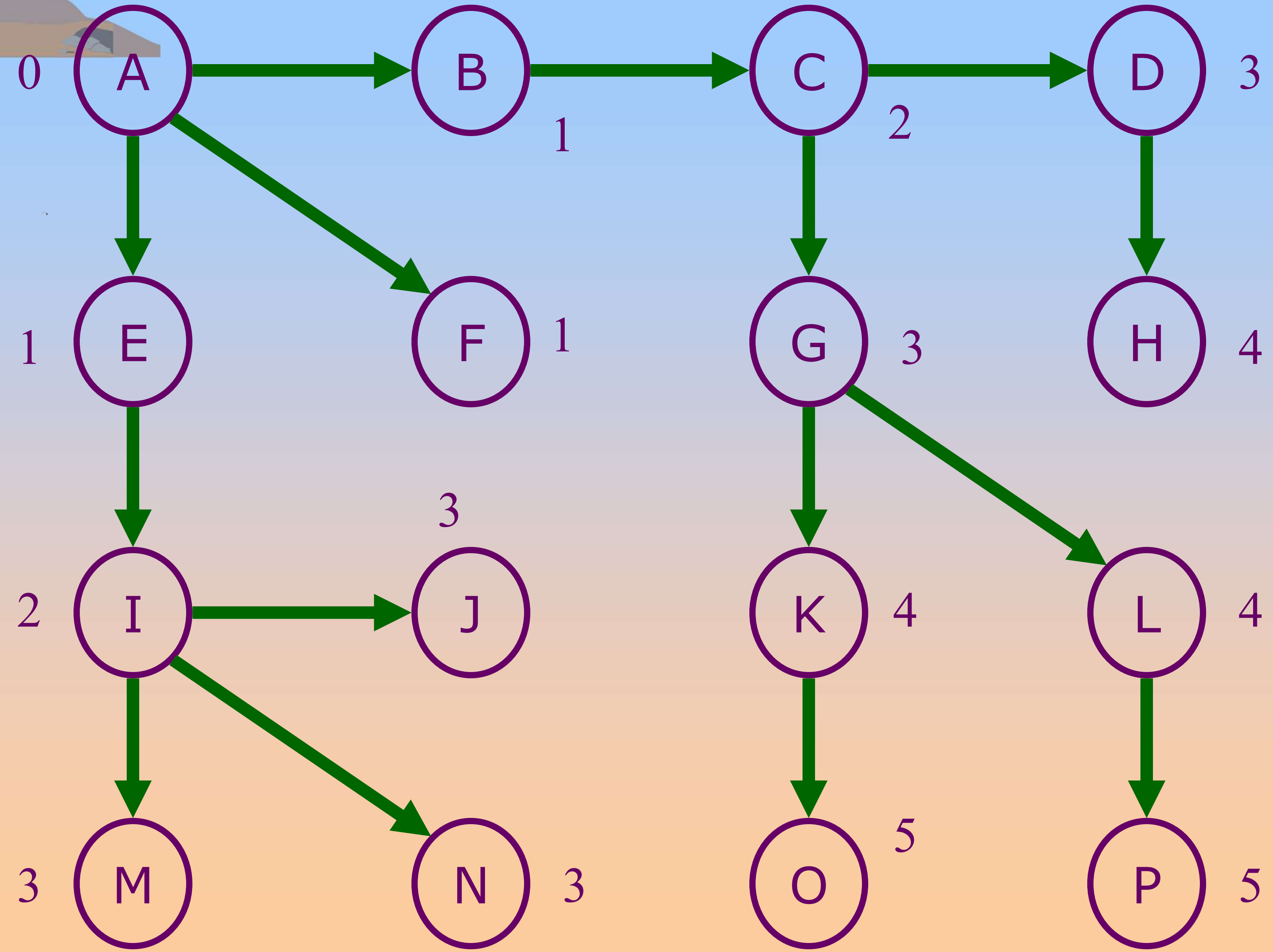

0
A
B
1
C
2
D
E
1
F
1
G
H
2
I
J
K
L
M
N
O
P


0
A
B
1
C
2
D
3
E
1
F
1
G
3
H
3
I
2
J
K
L
M
3
N
3
O
P

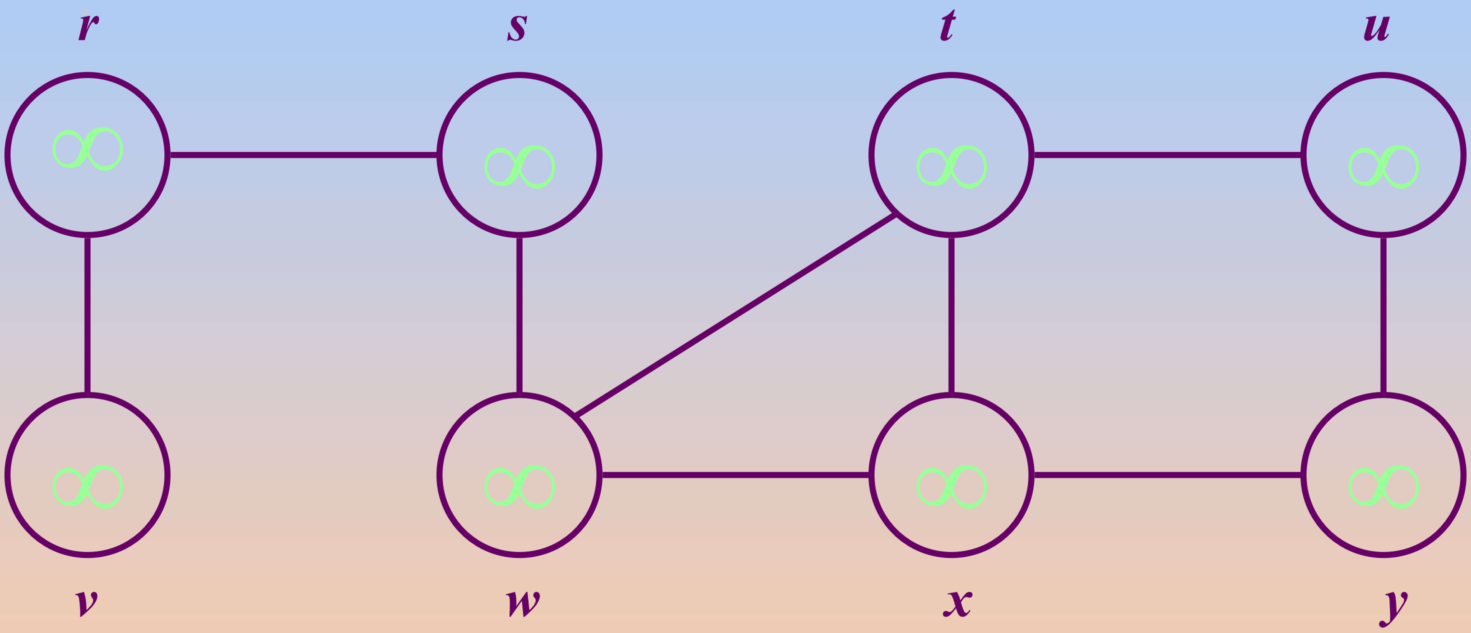

A
0
B
1
C
2
D
3
E
1
F
1
G
3
H
4
I
2
J
3
K
4
L
4
M
3
N
3
O
P

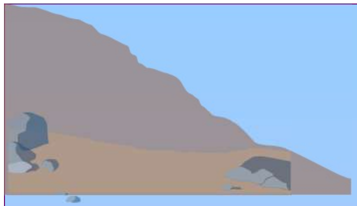

0
A
B
1
C
2
D
3
1
E
F
1
G
3
H
4
3
2
I
J
K
4
L
4
5
3
M
N
3
O
P
5


A
B
C
D
0
1
2
3
E
F
G
H
1
1
3
4
3
I
J
K
L
2
4
4
M
N
O
P
3
3
5
5

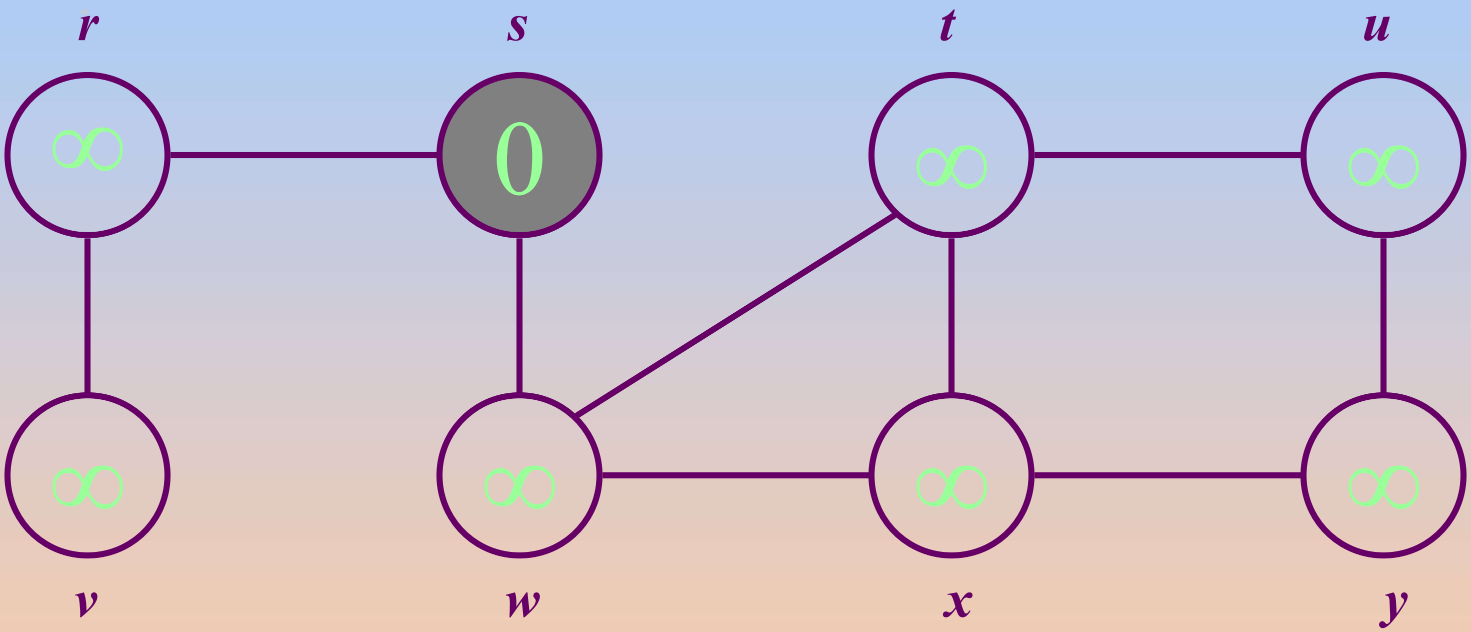

0
A
B
C
D
3
1
2
1
E
F
1
G
3
H
4
3
2
I
J
K
4
L
4
5
3
M
N
3
O
P
5

Example

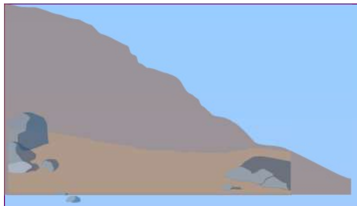




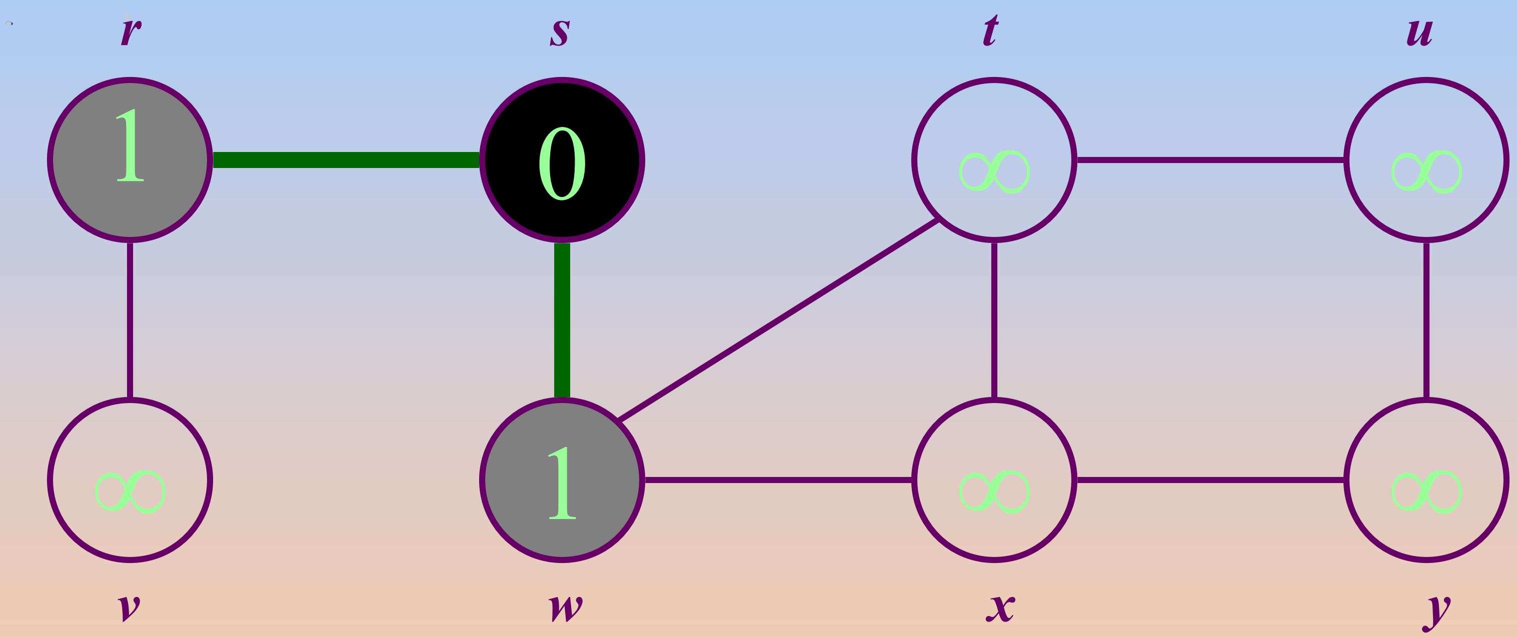
Example



Q: $\boxed{s}$

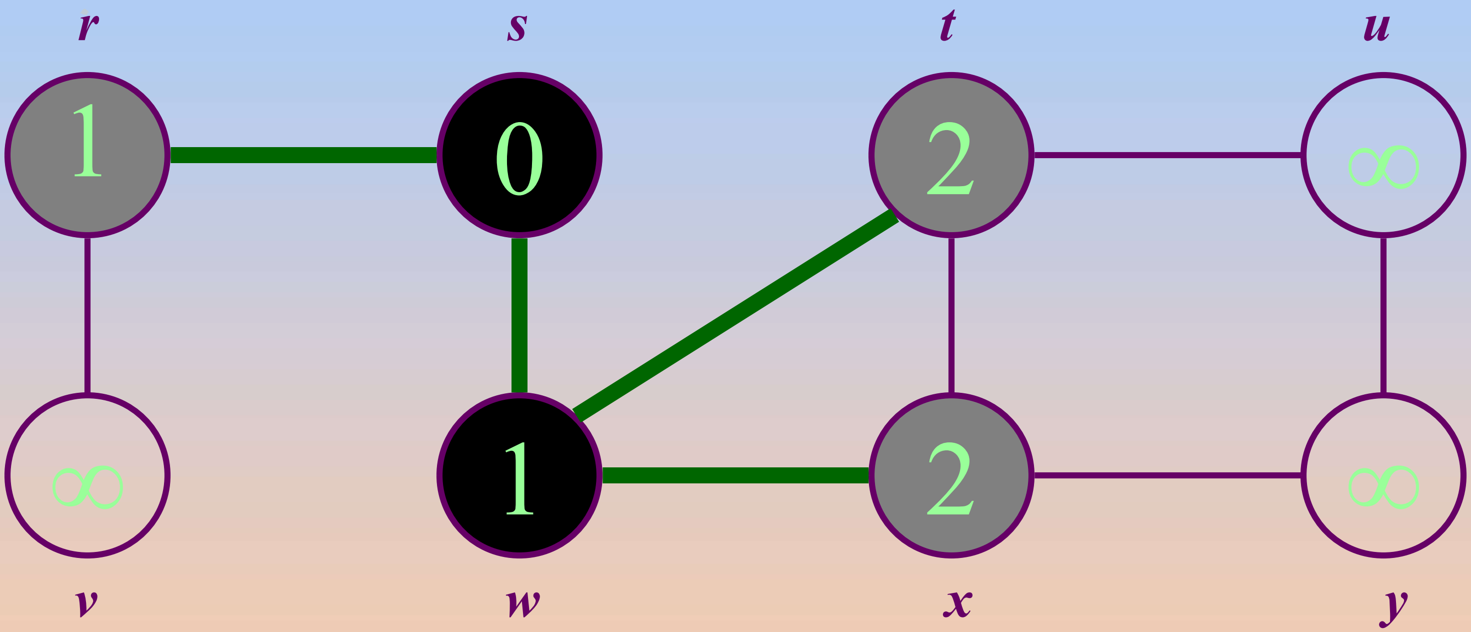


Example



Q: | w | r |

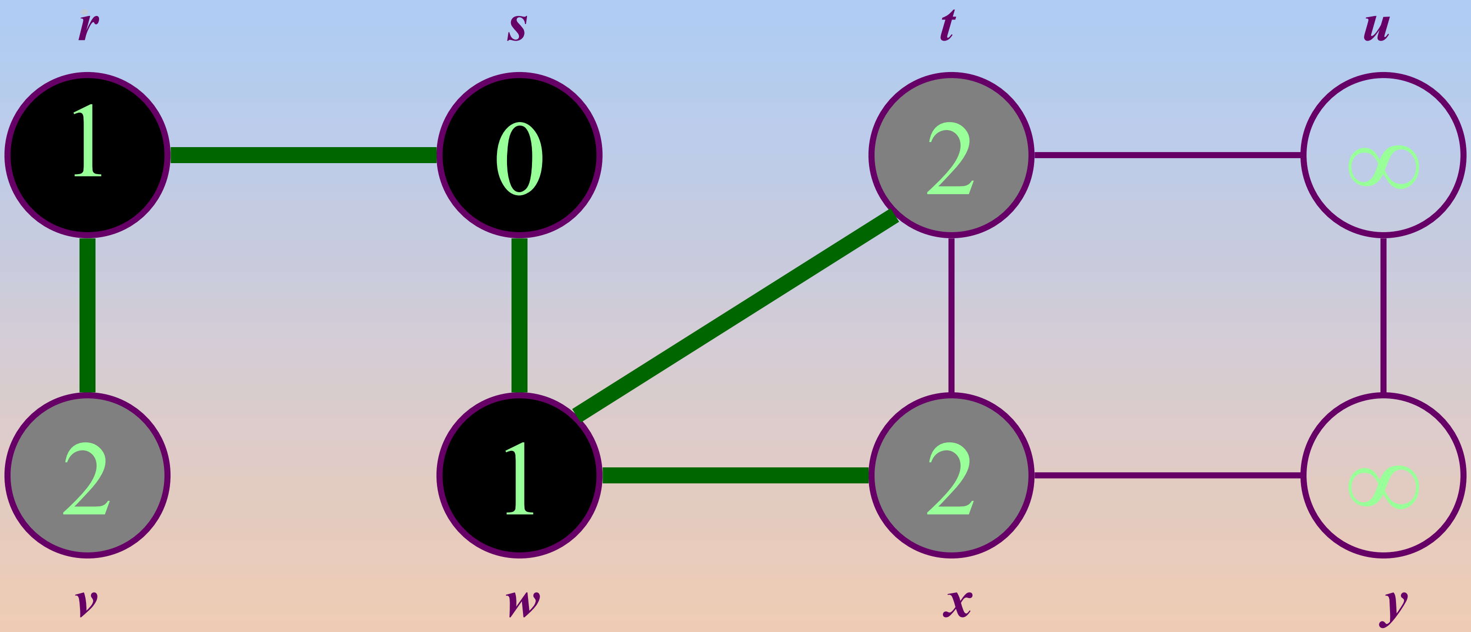
Example



Q:

r	*t*	*x*

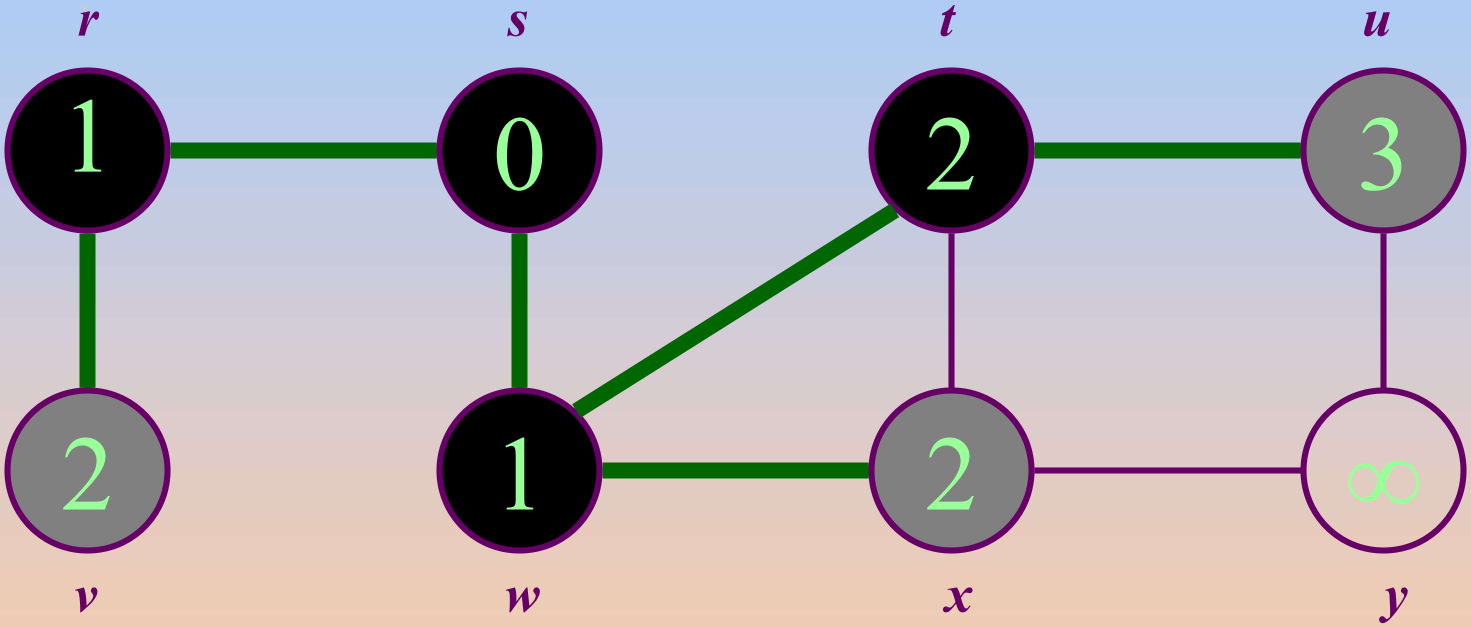
Example



Q:

t	*x*	*v*

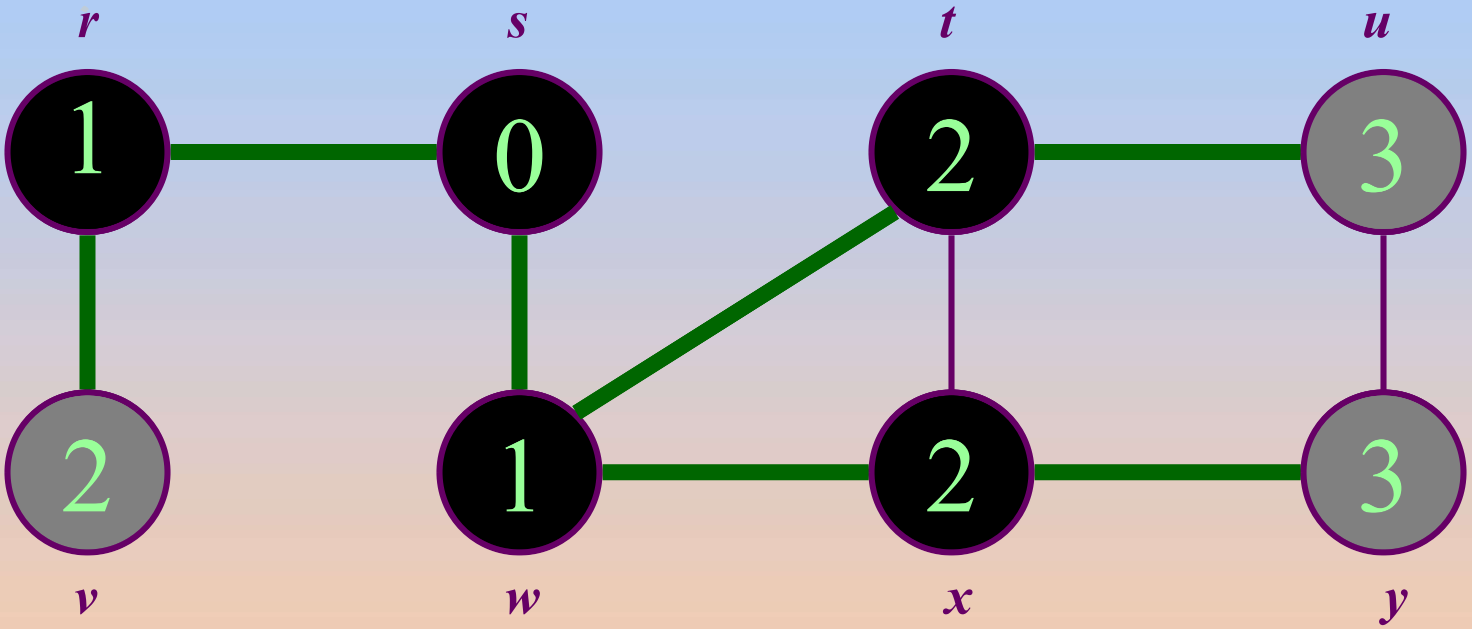
Example



Q:

x	*v*	*u*

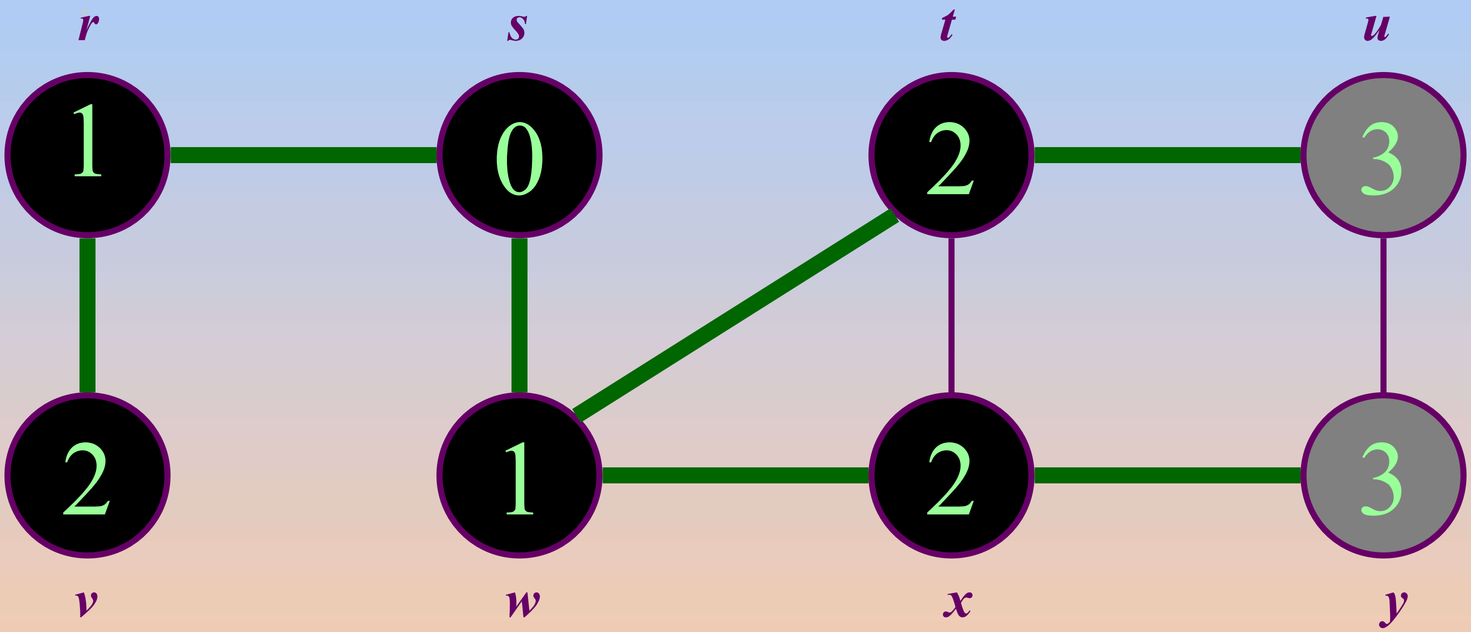
Example



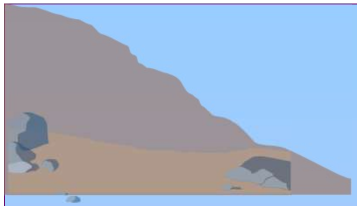
Q:

v	*u*	*y*

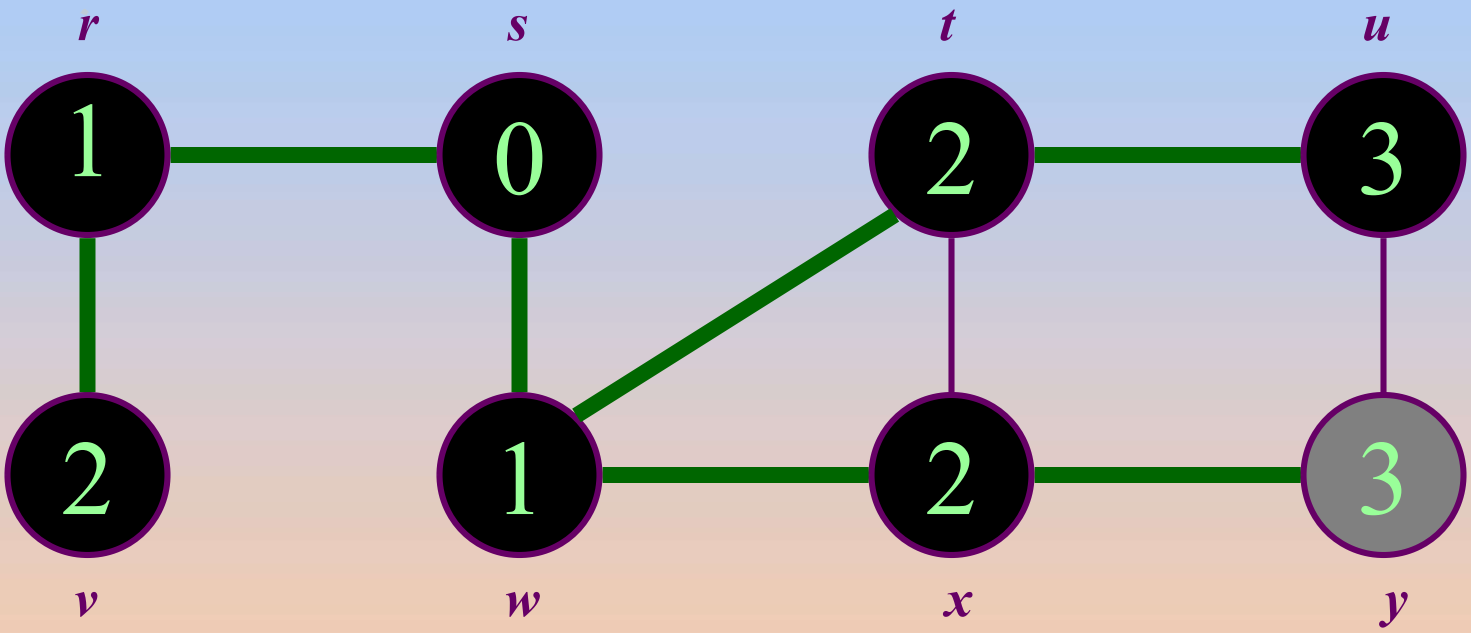
Example



Q:	*u*	*y*

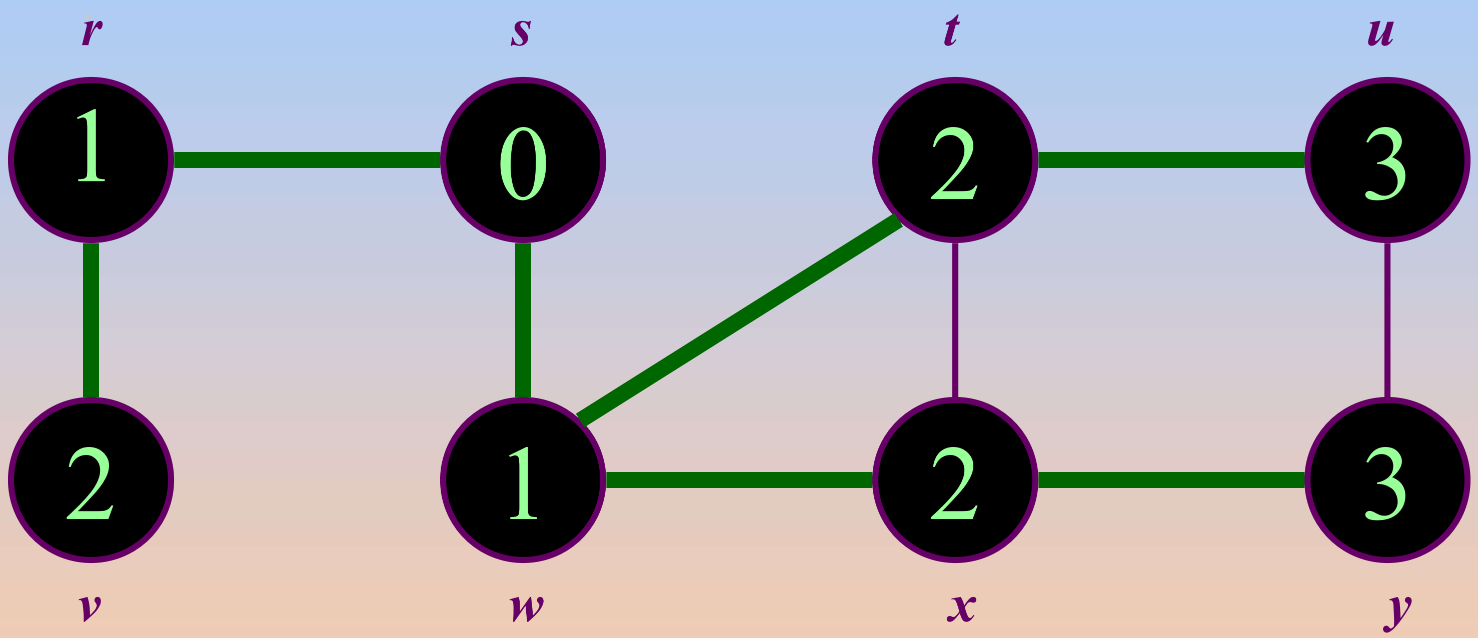


Example



Q: $\boxed{y}$

Example



Q: Ø

BFS: Application

- Shortest path problem