CSE 250: Depth- and Breadth-First Search

Lecture 20

Oct 16, 2024

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Reminders

- PA2 released
  - Implement Graph Traversal.
  - Due **Tuesday**, Oct 22
- Midterm Report out Today
  - Extrapolated from PA1, WA1, Midterm1
  - If you don't like your grade, have a chat (Fridays @ 10 AM)
- Midterm Grades
  - Out soon... (waiting on makeups)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `addVertex`: $O(1)$
- `removeEdge`: $O(1)$
- `removeVertex`: $O(M)$
- `incidentEdges`: $O(M)$
- `hasEdgeTo`: $O(M)$

Space Used: $O(N + M)$
(constant space per vertex, edge)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Adjacency List Summary

Starting with an edge list:

- Store a linked list of in/out edges with each vertex
- Store the linked list node for the in/out lists with each edge

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `addVertex`: $O(1)$
- `removeEdge`: $O(1)$
- `removeVertex`: $O(\texttt{deg}(v))$
- `incidentEdges`: $O(1) + O(1)$ per `next()`
- `hasEdgeTo`: $O(\texttt{deg}(v))$

Space Used: $O(N + M)$
(constant space per vertex, edge)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

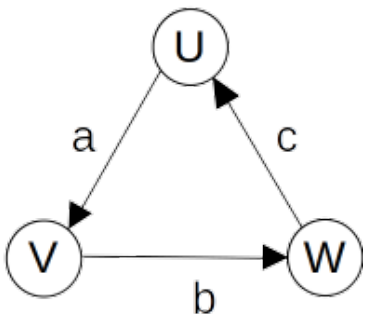
hasEdgeTo

Can we cut `hasEdgeTo` down to $O(1)$?

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

The Adjacency Matrix Data Structure

Origin \ Destination	U	V	W
U	-	a	-
V	-	-	b
W	c	-	-



© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`:
- `addVertex`, `removeVertex`:
- `incidentEdges`:
- `hasEdgeTo`:

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`: $O(1)$
- `addVertex`, `removeVertex`:
- `incidentEdges`:
- `hasEdgeTo`:

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`: $O(1)$
- `addVertex`, `removeVertex`: $O(N^2)$
- `incidentEdges`:
- `hasEdgeTo`:

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`: $O(1)$
- `addVertex`, `removeVertex`: $O(N^2)$
- `incidentEdges`: $O(N)$
- `hasEdgeTo`:

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`: $O(1)$
- `addVertex`, `removeVertex`: $O(N^2)$
- `incidentEdges`: $O(N)$
- `hasEdgeTo`: $O(1)$

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`: $O(1)$
- `addVertex`, `removeVertex`: $O(N^2)$
- `incidentEdges`: $O(N)$
- `hasEdgeTo`: $O(1)$

Space Used:

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`: $O(1)$
- `addVertex`, `removeVertex`: $O(N^2)$
- `incidentEdges`: $O(N)$
- `hasEdgeTo`: $O(1)$

Space Used:

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`: $O(1)$
- `addVertex`, `removeVertex`: $O(N^2)$
- `incidentEdges`: $O(N)$
- `hasEdgeTo`: $O(1)$

Space Used:

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Edge List Summary

- `addEdge`, `removeEdge`: $O(1)$
- `addVertex`, `removeVertex`: $O(N^2)$
- `incidentEdges`: $O(N)$
- `hasEdgeTo`: $O(1)$

Space Used: $O(N^2)$

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

A few more definitions

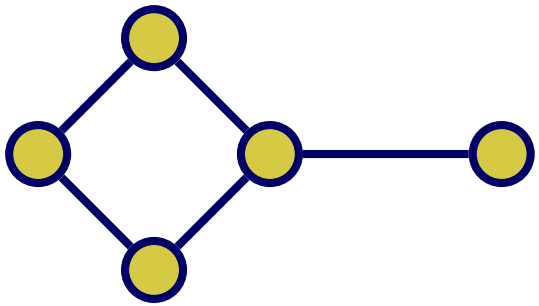
A graph is **connected** if...

- ...there is a path between every pair of vertices.

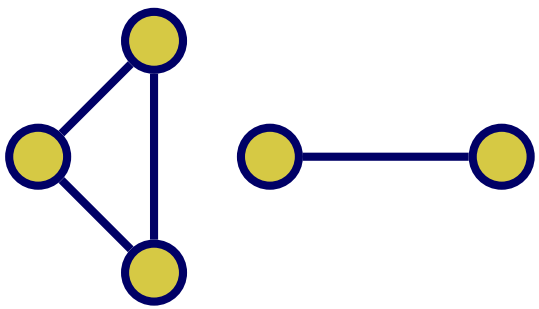
A **connected component** of G is a maximal, connected subgraph of G

- "maximal" means that adding any other vertices from G would break the connected property.
- Any subset of G's edges that makes the subgraph connected is fine.

Connected Graph



Disconnected Graph



A few more definitions

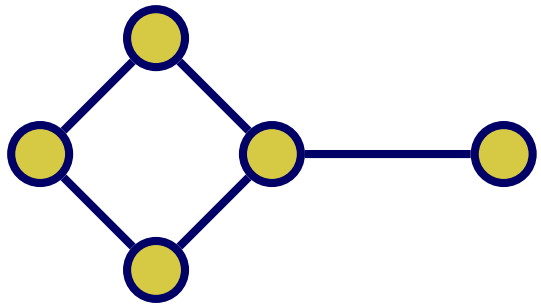
A graph is **connected** if...

- ...there is a path between every pair of vertices.

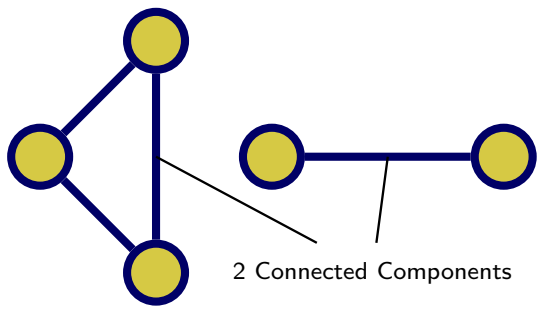
A **connected component** of G is a maximal, connected subgraph of G

- "maximal" means that adding any other vertices from G would break the connected property.
- Any subset of G's edges that makes the subgraph connected is fine.

Connected Graph



Disconnected Graph



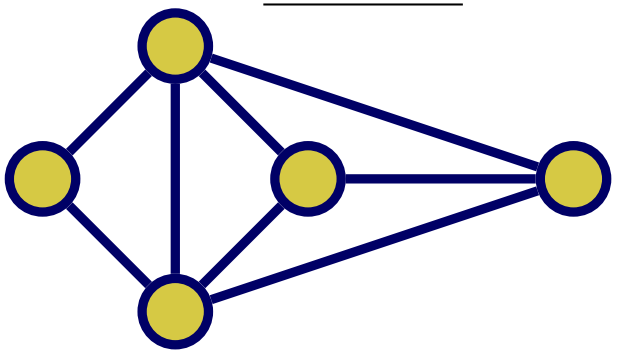
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

A few more definitions...

A **spanning tree** of a connected graph G is:

- A spanning subgraph of G
- A tree

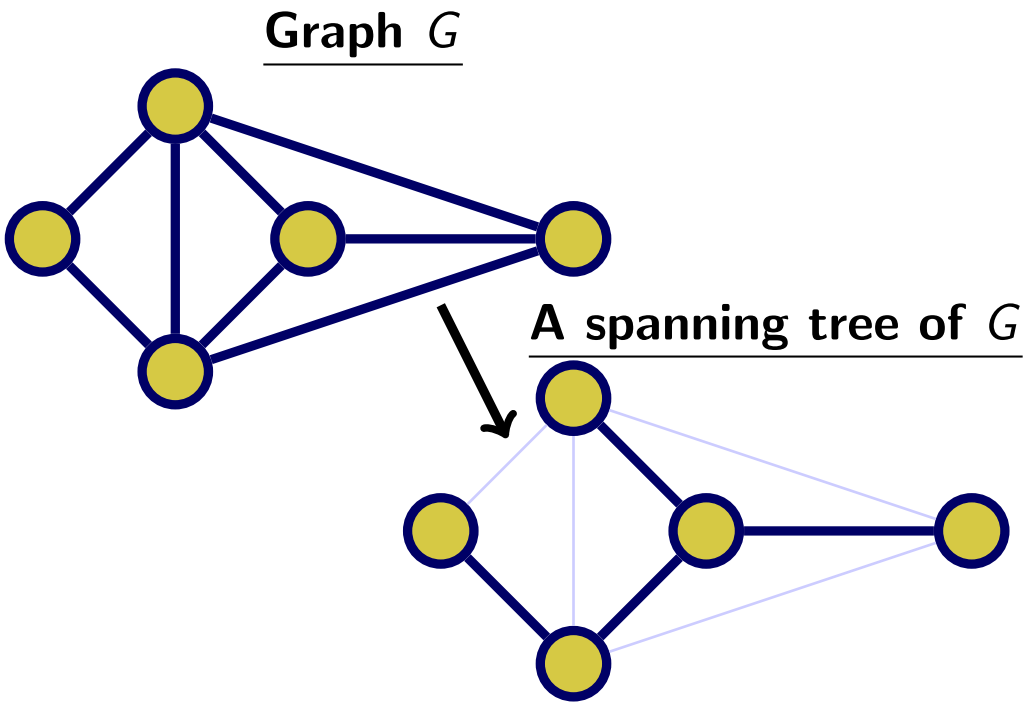
Graph G



A few more definitions...

A **spanning tree** of a connected graph G is:

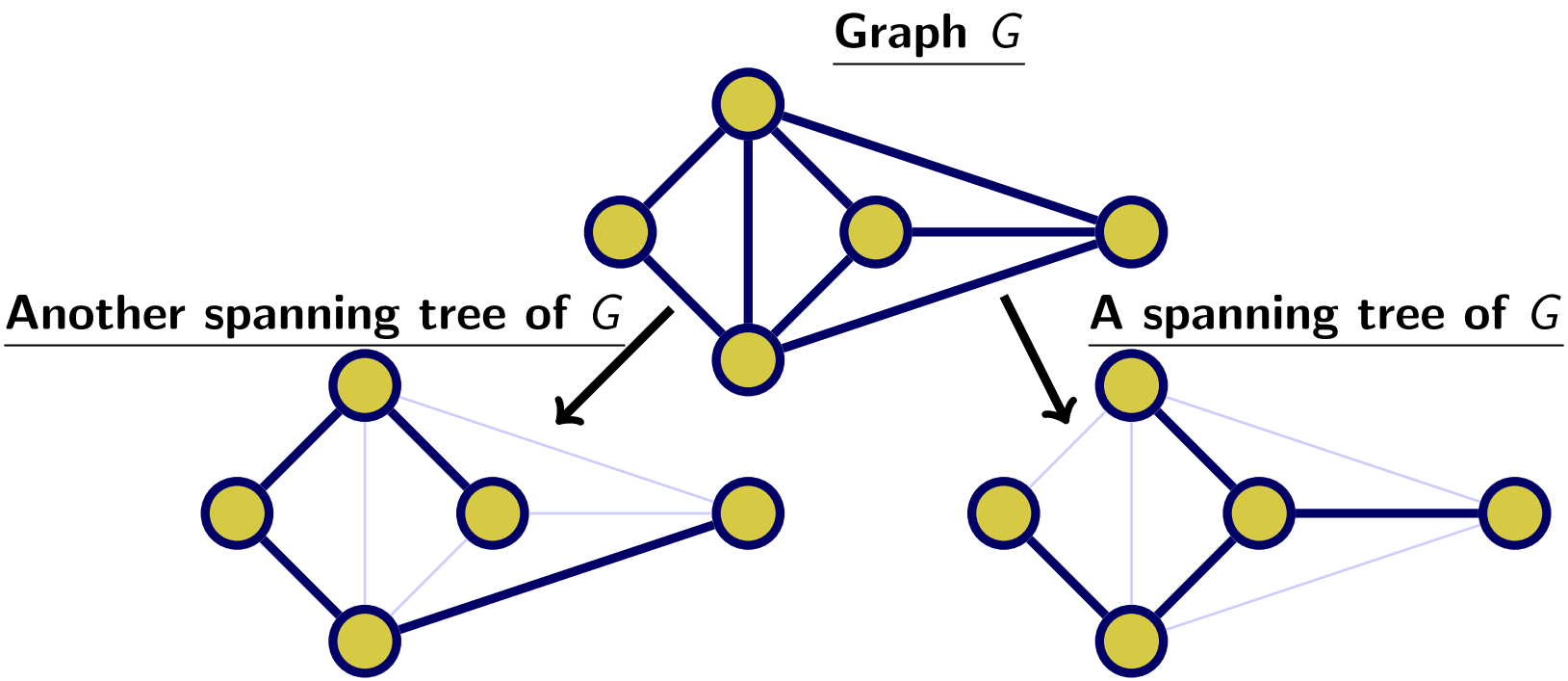
- A spanning subgraph of G
- A tree



A few more definitions...

A **spanning tree** of a connected graph G is:

- A spanning subgraph of G
- A tree



Note: A graph has multiple spanning trees (unless it is already a tree).

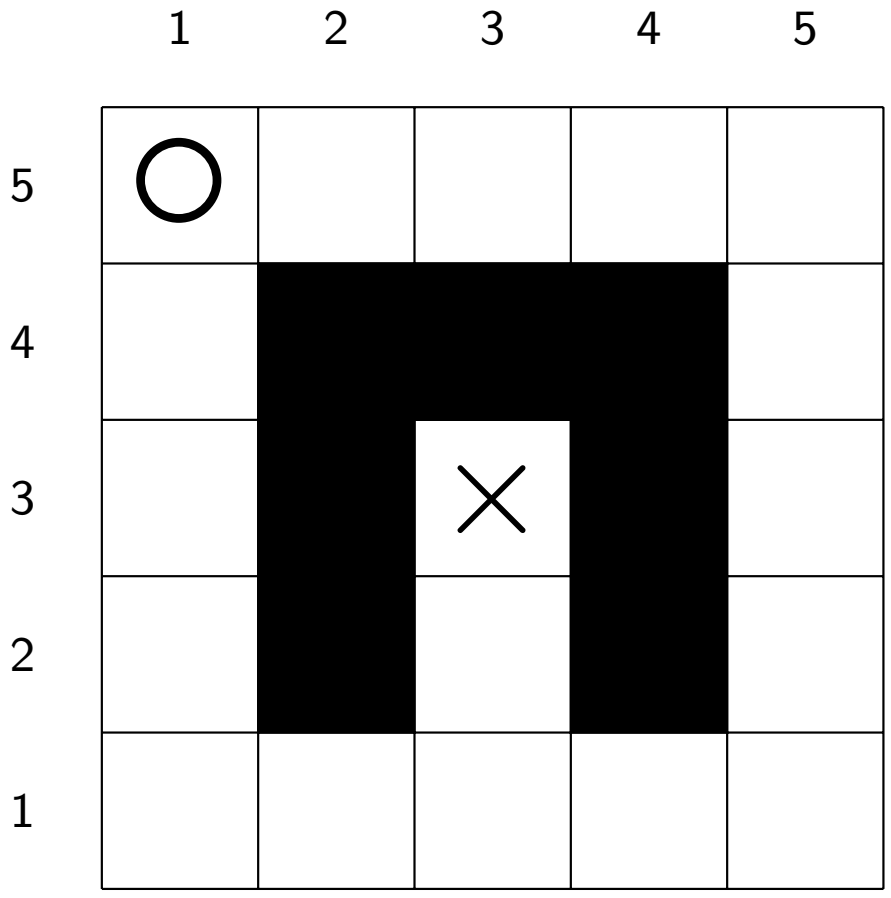
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Back to connectivity...

So how about connectivity?

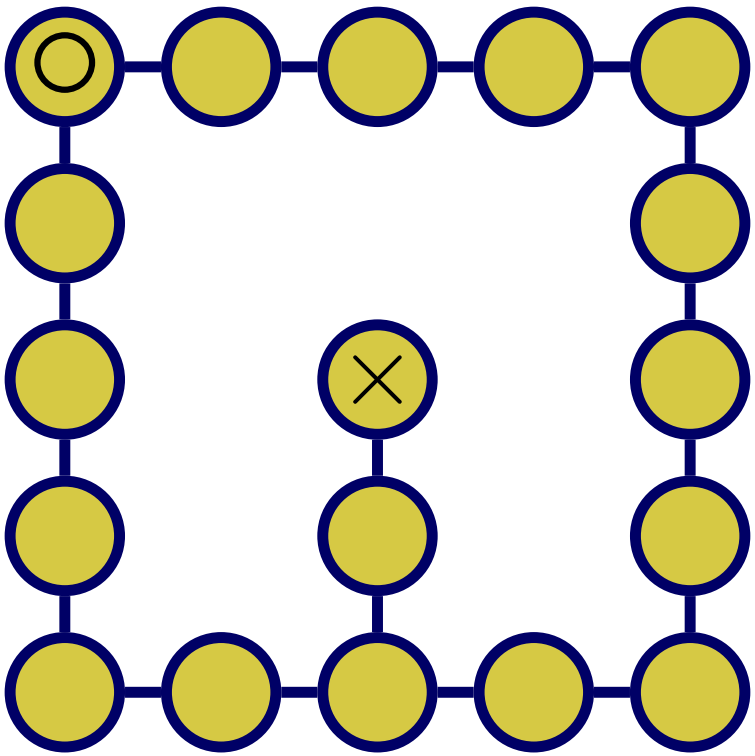
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

... or mazes?



© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

... or mazes?



© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Recall Mazes

- Start at a vertex
- If this is the target vertex, done!
- Push the next adjacent vertex (not filled, or already on the stack) onto the stack
- Explore from that vertex
- When that vertex is explored, pop it and move to the next adjacent vertex
- When everything is explored, return.

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Recall Mazes

- Start at a vertex
- If this is the target vertex, done!
- Push the next adjacent vertex (not filled, or already on the stack) onto the stack
- Explore from that vertex
- When that vertex is explored, pop it and move to the next adjacent vertex
- When everything is explored, return.

This is called Depth-First Search

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Recall Mazes

- Start at a vertex
- If this is the target vertex, done!
- Push the next adjacent vertex (not filled, or already on the stack) onto the stack
- Explore from that vertex
- When that vertex is explored, pop it and move to the next adjacent vertex
- When everything is explored, return.

This is called Depth-First Search Contrast with Breadth-First Search

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Primary Goals

- Visit every vertex in graph $G = (V, E)$.
- Construct a spanning tree for every connected component.

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Primary Goals

- Visit every vertex in graph $G = (V, E)$.
- Construct a spanning tree for every connected component.
  - **Side Effect**: Compute connected components.

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Primary Goals

- Visit every vertex in graph $G = (V, E)$.
- Construct a spanning tree for every connected component.
  - **Side Effect**: Compute connected components.
  - **Side Effect**: Compute a path between all connected vertices.

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Primary Goals

- Visit every vertex in graph $G = (V, E)$.
- Construct a spanning tree for every connected component.
  - **Side Effect**: Compute connected components.
  - **Side Effect**: Compute a path between all connected vertices.
  - **Side Effect**: Determine if the graph is connected.

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Primary Goals

- Visit every vertex in graph $G = (V, E)$.
- Construct a spanning tree for every connected component.
  - **Side Effect**: Compute connected components.
  - **Side Effect**: Compute a path between all connected vertices.
  - **Side Effect**: Determine if the graph is connected.
  - **Side Effect**: Identify any cycles (if they exist).

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Primary Goals

- Visit every vertex in graph $G = (V, E)$.
- Construct a spanning tree for every connected component.
  - **Side Effect**: Compute connected components.
  - **Side Effect**: Compute a path between all connected vertices.
  - **Side Effect**: Determine if the graph is connected.
  - **Side Effect**: Identify any cycles (if they exist).
- Complete in time $O(N + M)$.

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

`DFS(G)`

Input

- Graph $G = (V, E)$

Output

- Label every edge as a:
  - Spanning Edge: Part of the spanning tree
  - Back Edge: Part of a cycle

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

`DFSOne(G, v)`

Input

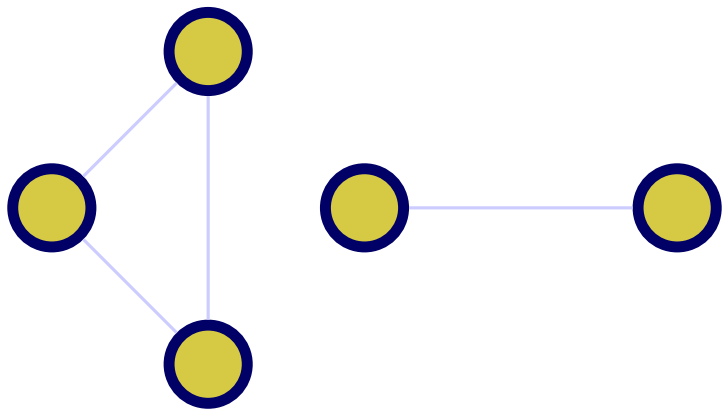
- Graph $G = (V, E)$
- Start vertex $v \in V$

Output

- Label every edge in v's connected component as a:
  - Spanning Edge: Part of the spanning tree
  - Back Edge: Part of a cycle

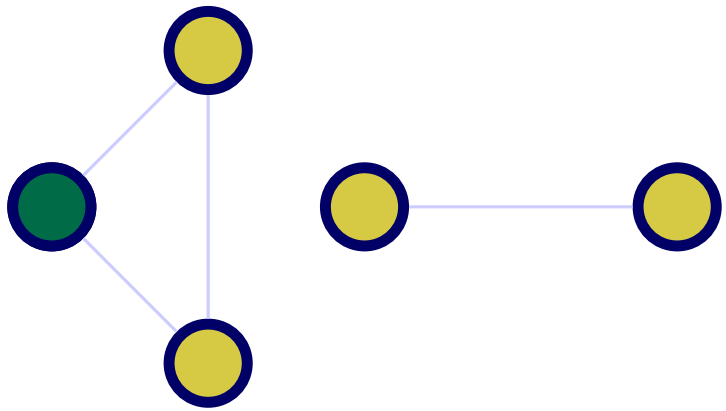
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



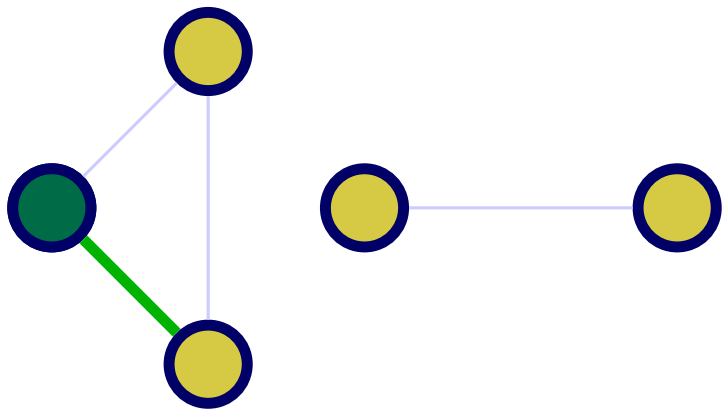
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



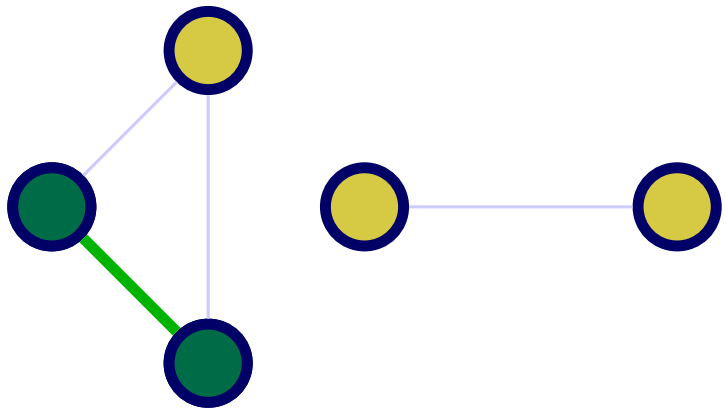
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



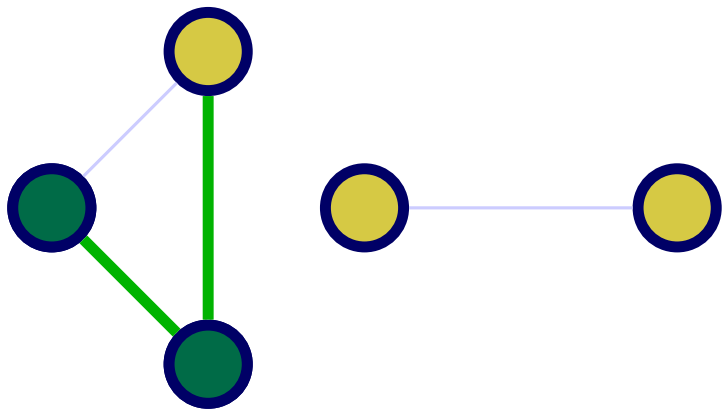
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



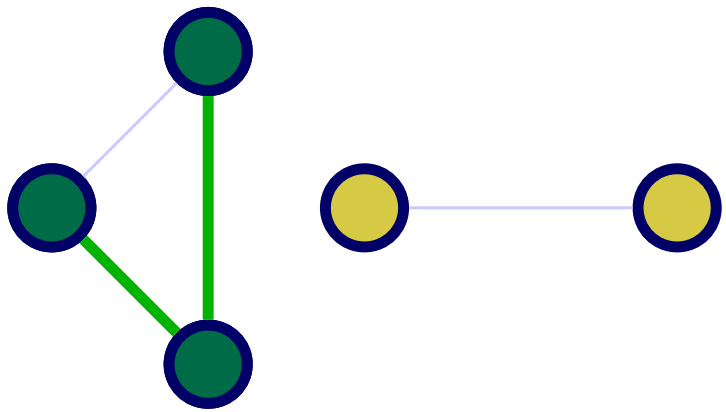
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



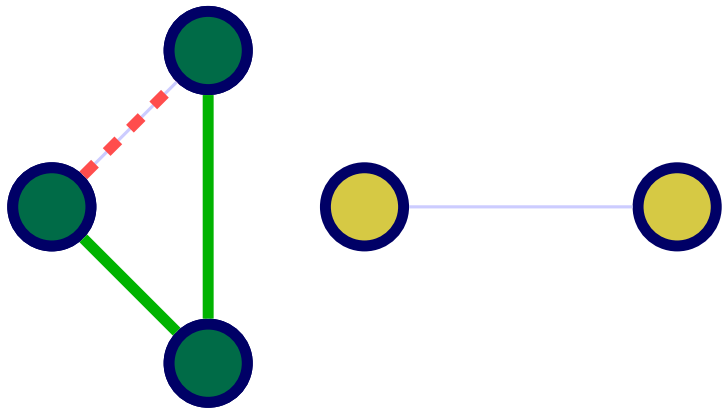
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



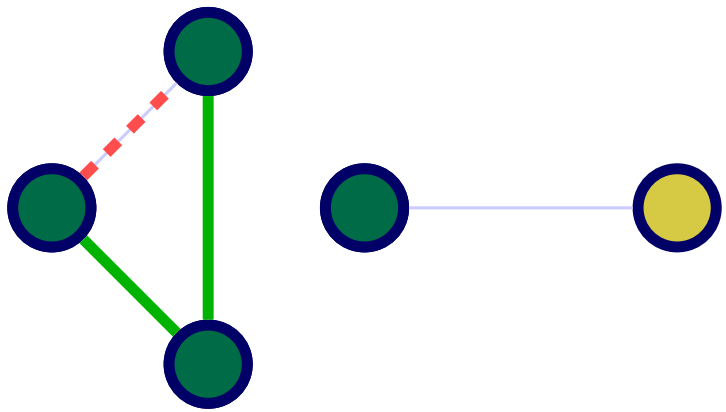
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



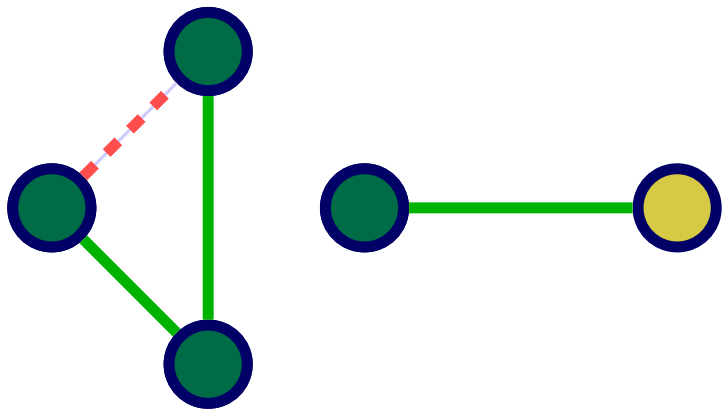
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



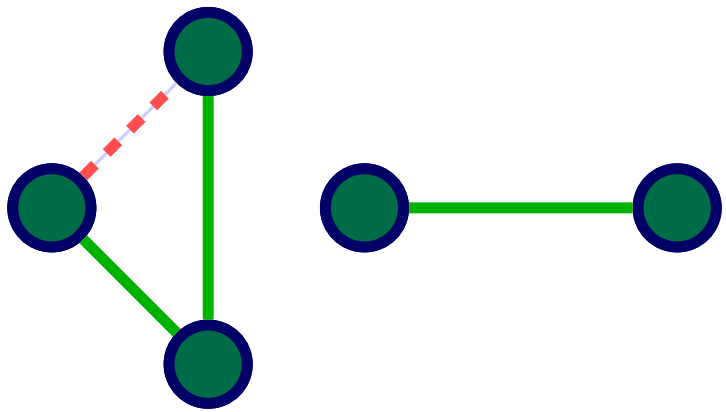
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS Example



© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

```
1  public enum VertexLabel { UNEXPLORED, VISITED }
2  public enum EdgeLabel { UNEXPLORED, SPANNING, BACK }
3
4  public void DFS(Graph<VertexLabel,EdgeLabel> graph)
5  {
6    for(Vertex v : graph.vertices)
7                  { v.setLabel(VertexLabel.UNEXPLORED); }
8    for(Edge e   : graph.edges)
9                  { e.setLabel(EdgeLabel.UNEXPLORED); }
10   for(Vertex v : graph.vertices)
11   {
12     if(v.getLabel() == VertexLabel.UNEXPLORED)
13     {
14       DFSOne(graph, v)
15     }
16   }
17 }
```

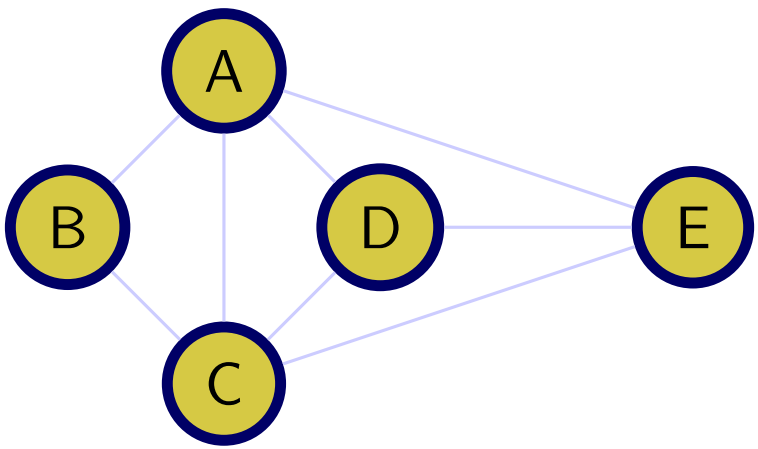
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFSOne

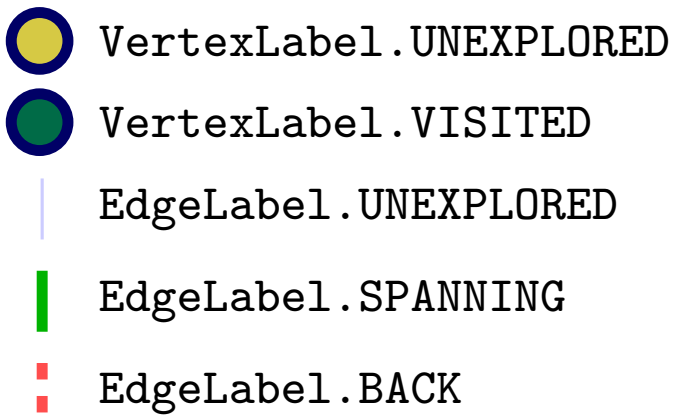
```
1   public void DFSOne(Graph<...> graph, Vertex start)
2   {
3     start.setLabel(VertexLabel.VISITED)
4
5     for( edge : start.getIncident() ) {
6       if(edge.getLabel == EdgeLabel.UNEXPLORED){
7         Vertex otherVertex = edge.getOppositeVertex(start);
8
9         if(otherVertex.getLabel == VertexLabel.UNEXPLORED){
10          edge.setLabel(EdgeLabel.SPANNING);
11          DFSOne(graph, otherVertex);
12        } else {
13          edge.setLabel(EdgeLabel.BACK);
14        }
15      }
16    }
17  }
```

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

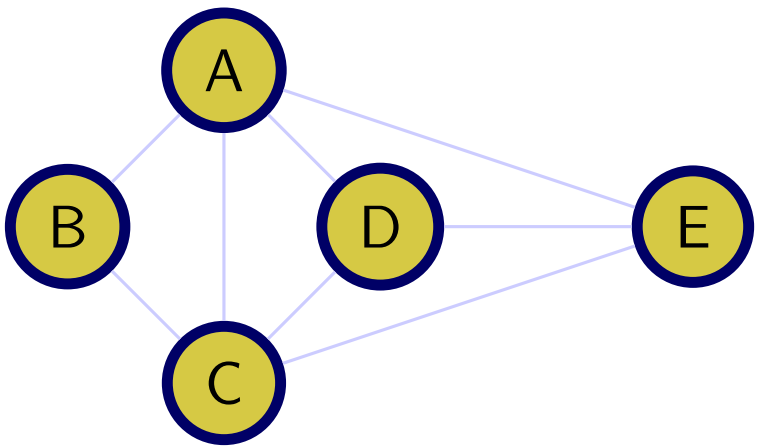


Call Stack



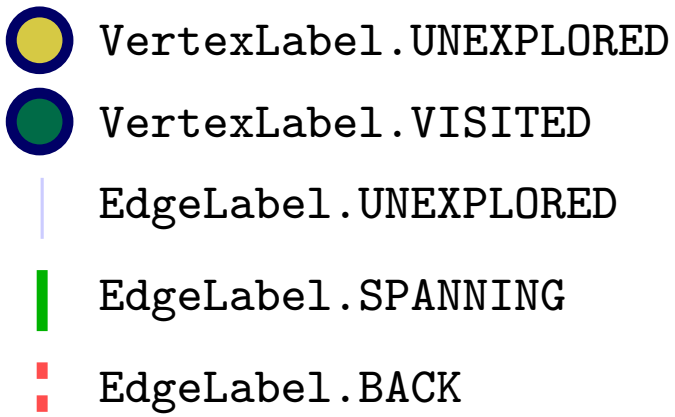
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)



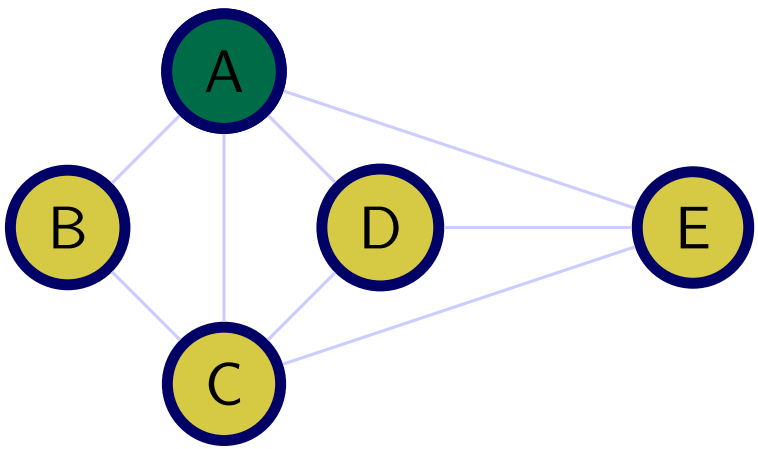
Call Stack

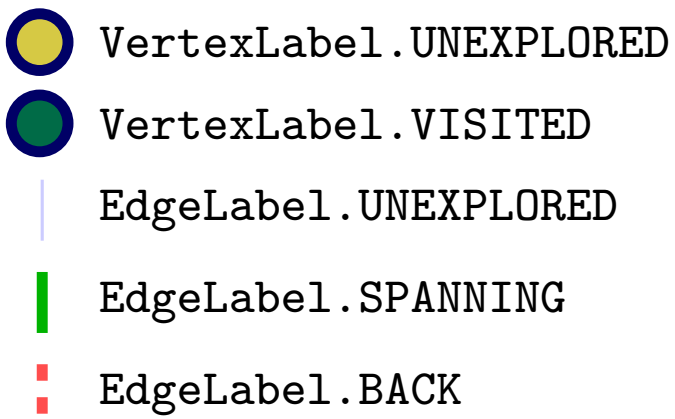
`DFS(G)`



© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





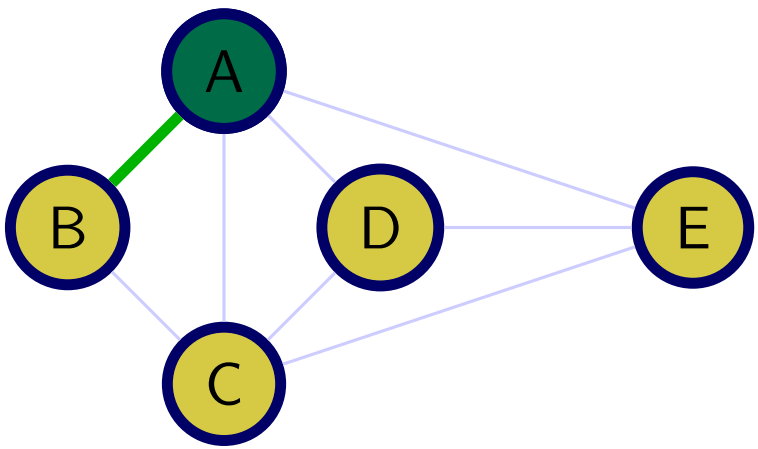
Call Stack

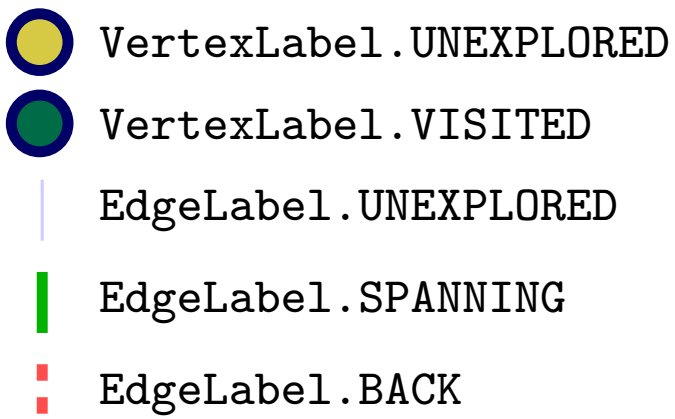
`DFS(G)`

`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





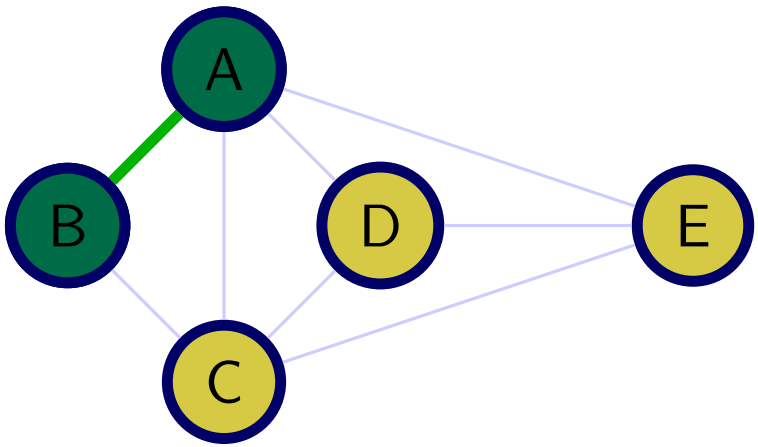
Call Stack

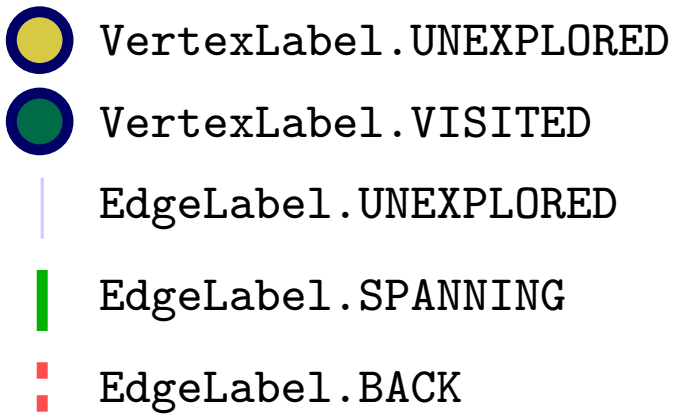
`DFS(G)`

`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

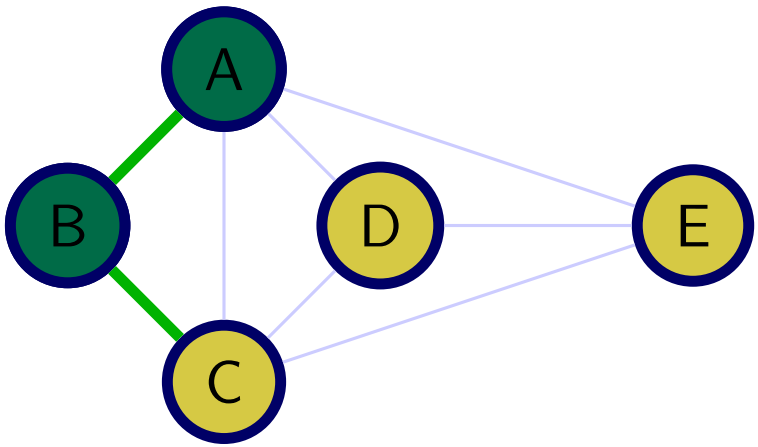
`DFS(G)`

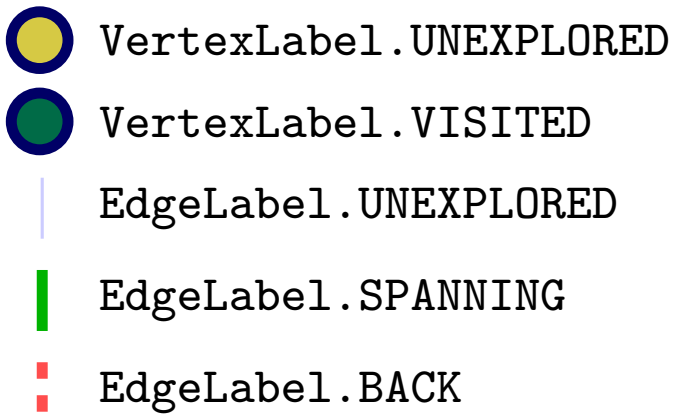
`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

`DFSOne(G, B)` ($\rightarrow$ A, C)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

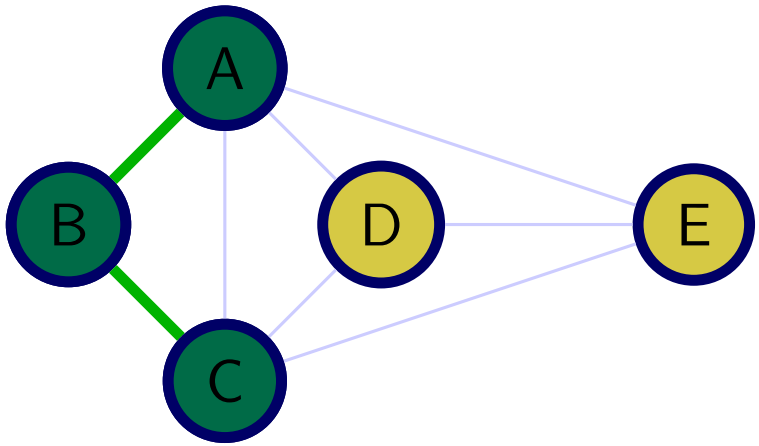
`DFS(G)`

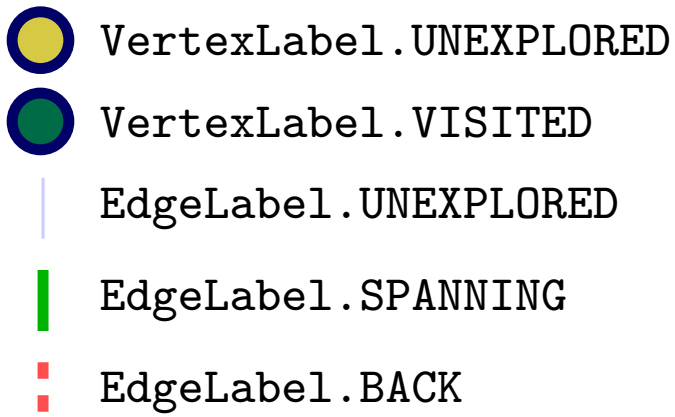
`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

`DFSOne(G, B)` ($\rightarrow$ A, C)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

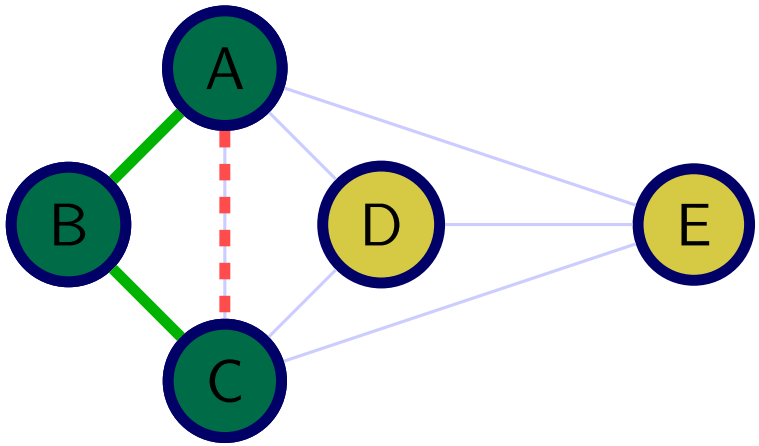
`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

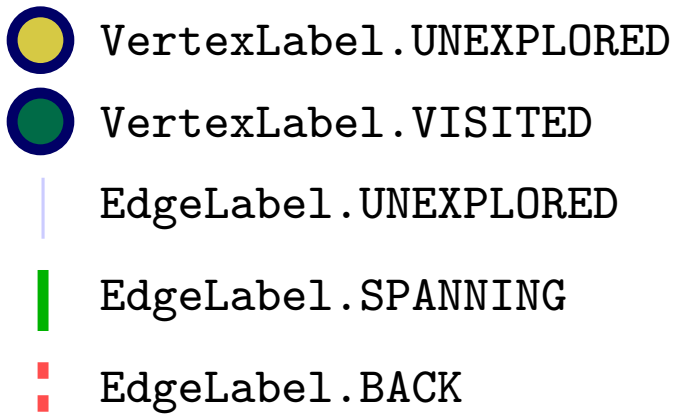
`DFSOne(G, B)` ($\rightarrow$ A, C)

`DFSOne(G, C)` ($\rightarrow$ A, B, D, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

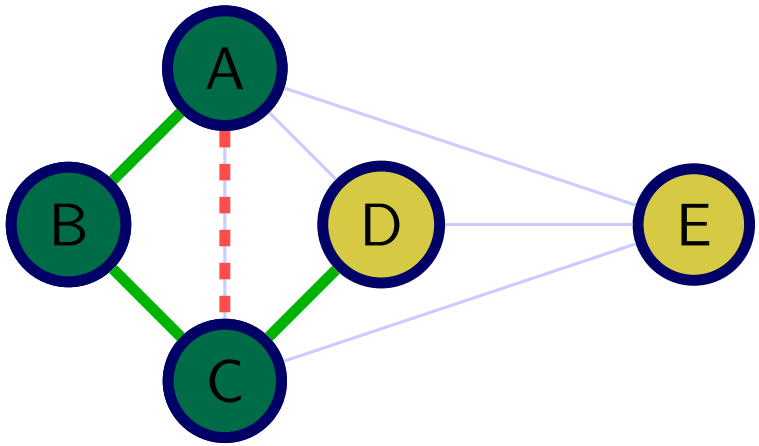
`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

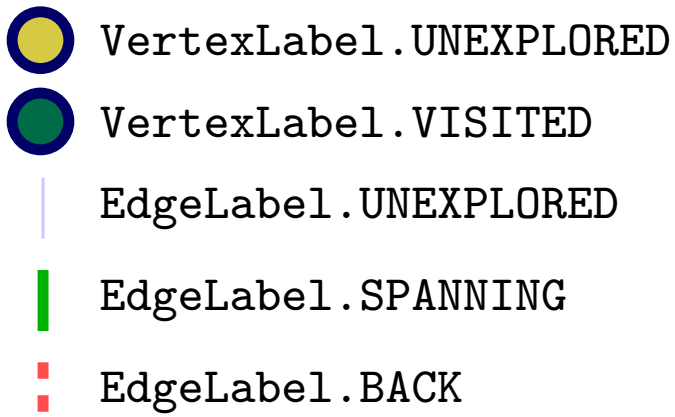
`DFSOne(G, B)` ($\rightarrow$ A, C)

`DFSOne(G, C)` ($\rightarrow$ A, B, D, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

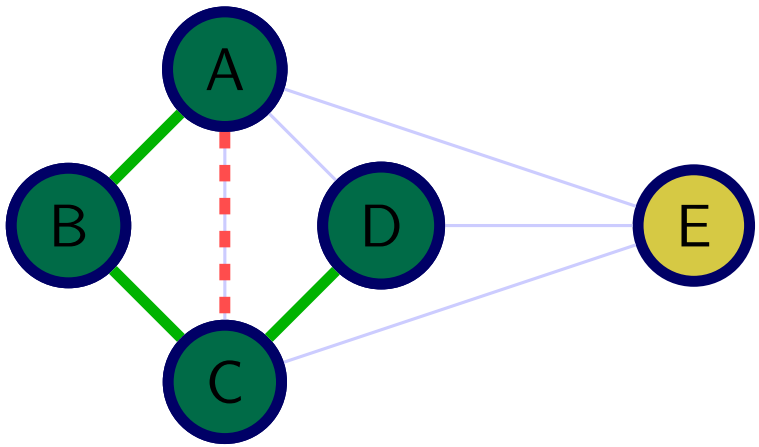
`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

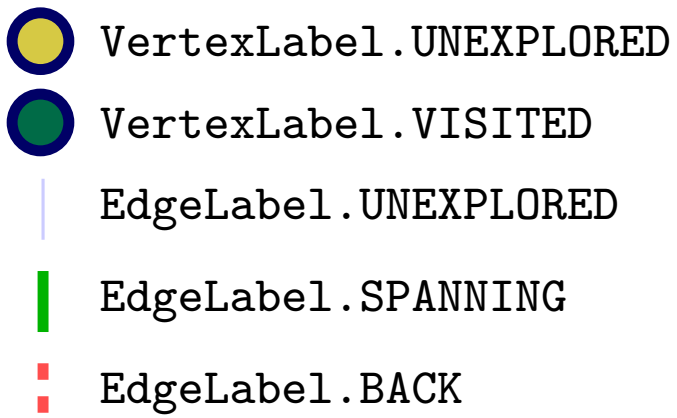
`DFSOne(G, B)` ($\rightarrow$ A, C)

`DFSOne(G, C)` ($\rightarrow$ A, B, D, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

`DFSOne(G, A)` (→ B, C, D, E)

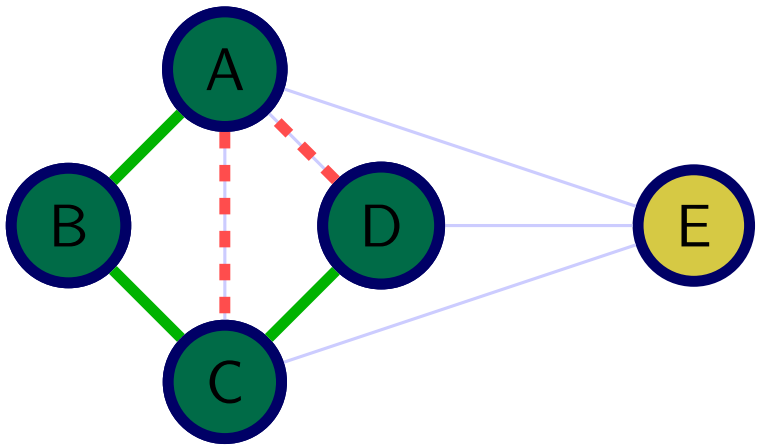
`DFSOne(G, B)` (→ A, C)

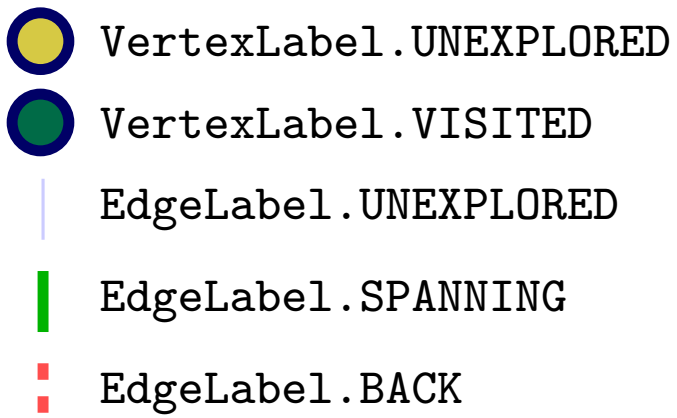
`DFSOne(G, C)` (→ A, B, D, E)

`DFSOne(G, D)` (→ A, C, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

`DFSOne(G, A)` (→ B, C, D, E)

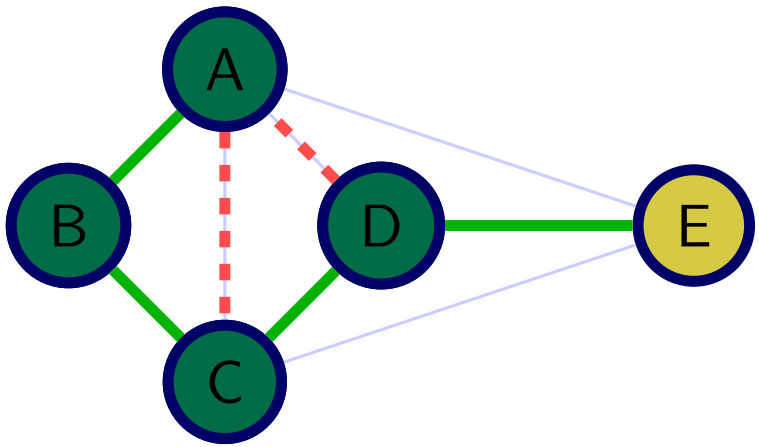
`DFSOne(G, B)` (→ A, C)

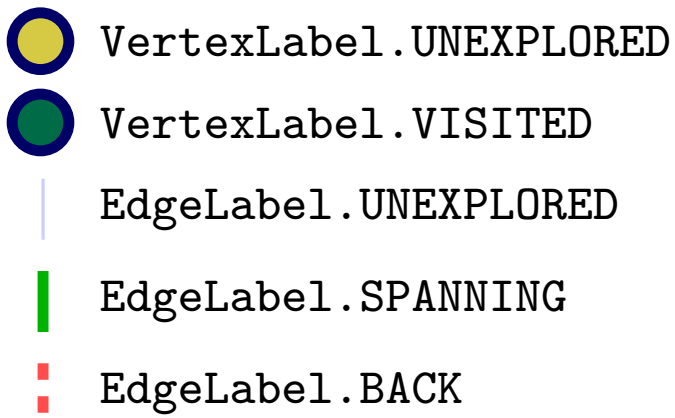
`DFSOne(G, C)` (→ A, B, D, E)

`DFSOne(G, D)` (→ A, C, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

`DFSOne(G, A)` (→ B, C, D, E)

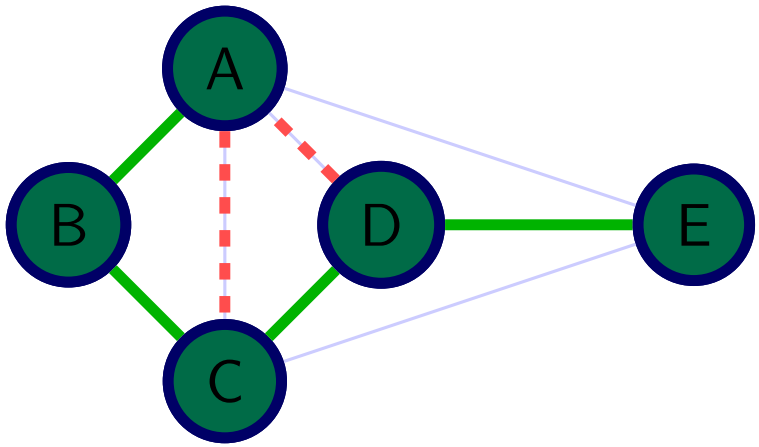
`DFSOne(G, B)` (→ A, C)

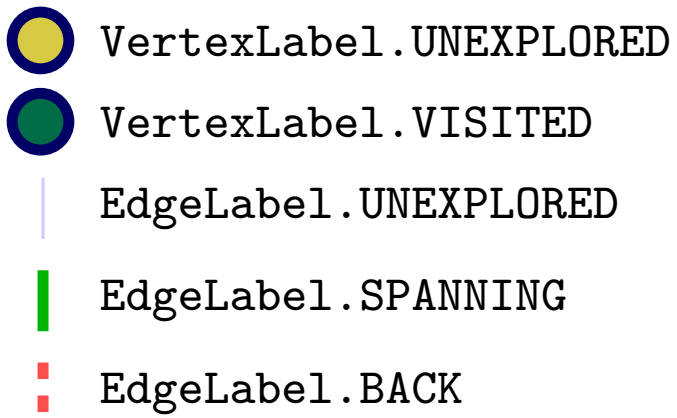
`DFSOne(G, C)` (→ A, B, D, E)

`DFSOne(G, D)` (→ A, C, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

`DFSOne(G, A)` (→ B, C, D, E)

`DFSOne(G, B)` (→ A, C)

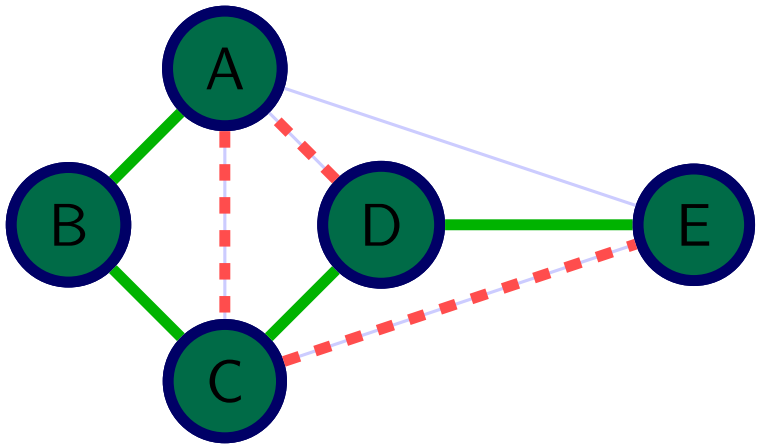
`DFSOne(G, C)` (→ A, B, D, E)

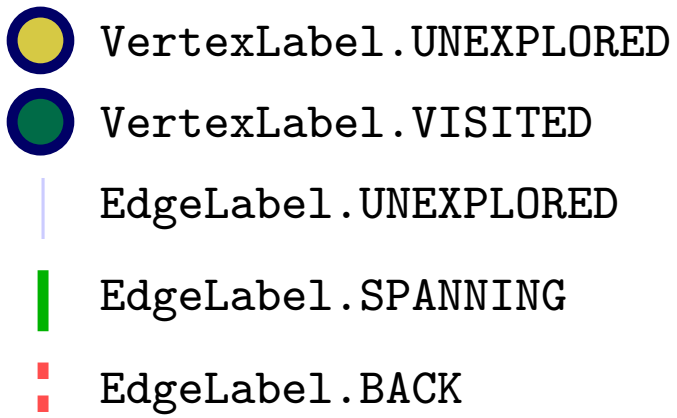
`DFSOne(G, D)` (→ A, C, E)

`DFSOne(G, E)` (→ A, C, D)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

`DFSOne(G, B)` ($\rightarrow$ A, C)

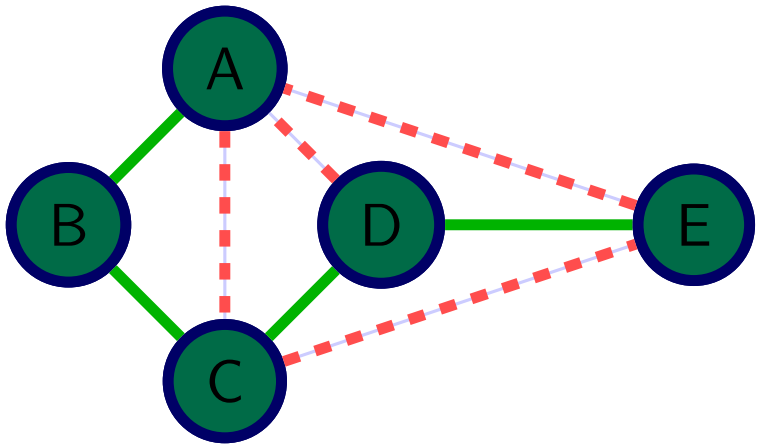
`DFSOne(G, C)` ($\rightarrow$ A, B, D, E)

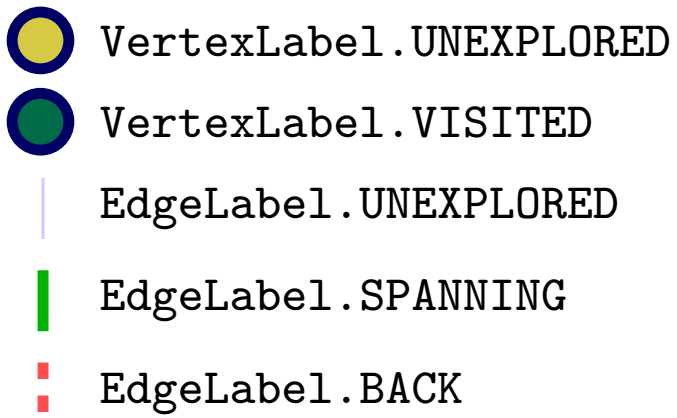
`DFSOne(G, D)` ($\rightarrow$ A, C, E)

`DFSOne(G, E)` ($\rightarrow$ A, C, D)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

`DFSOne(G, B)` ($\rightarrow$ A, C)

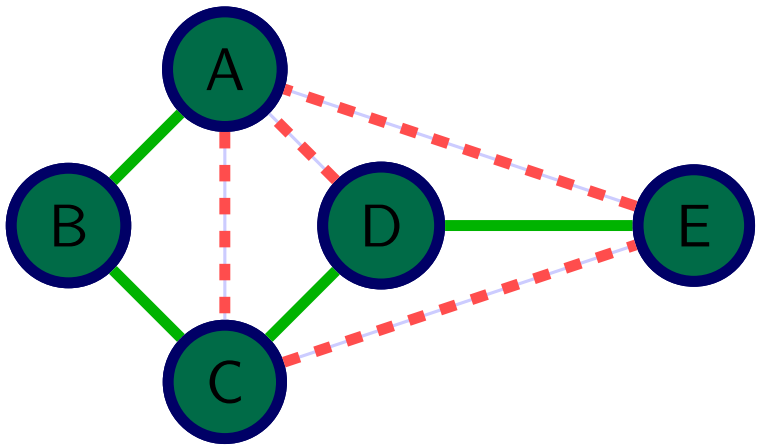
`DFSOne(G, C)` ($\rightarrow$ A, B, D, E)

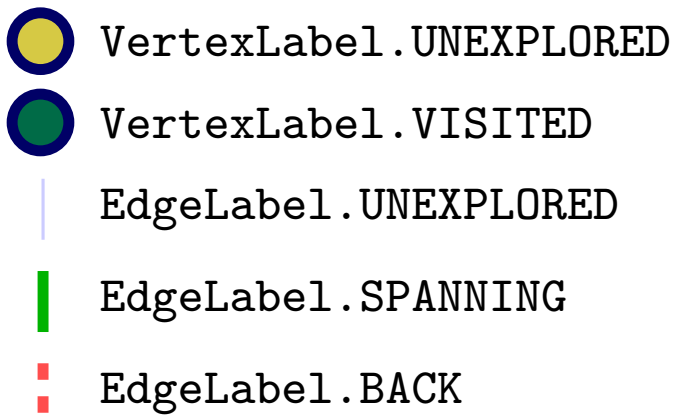
`DFSOne(G, D)` ($\rightarrow$ A, C, E)

`DFSOne(G, E)` ($\rightarrow$ A, C, D)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

`DFSOne(G, A)` (→ B, C, D, E)

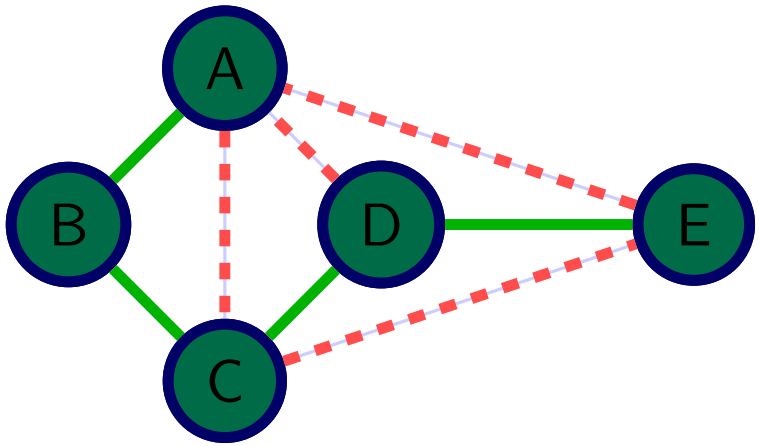
`DFSOne(G, B)` (→ A, C)

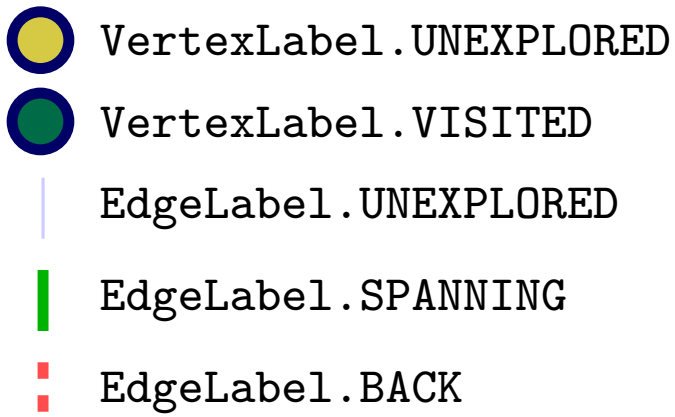
`DFSOne(G, C)` (→ A, B, D, E)

`DFSOne(G, D)` (→ A, C, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

`DFS(G)`

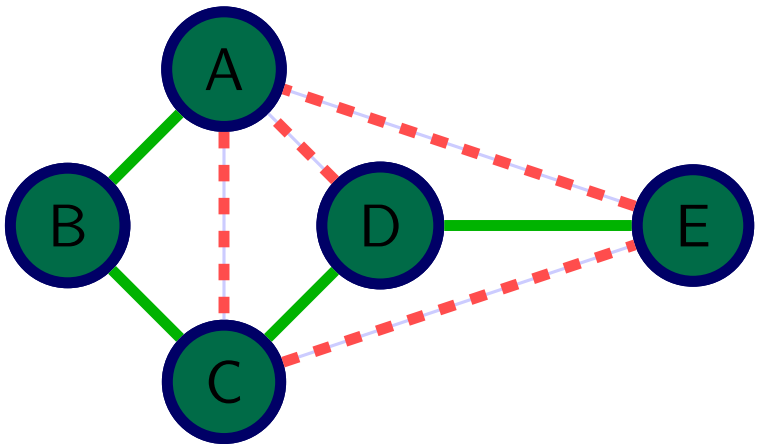
`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

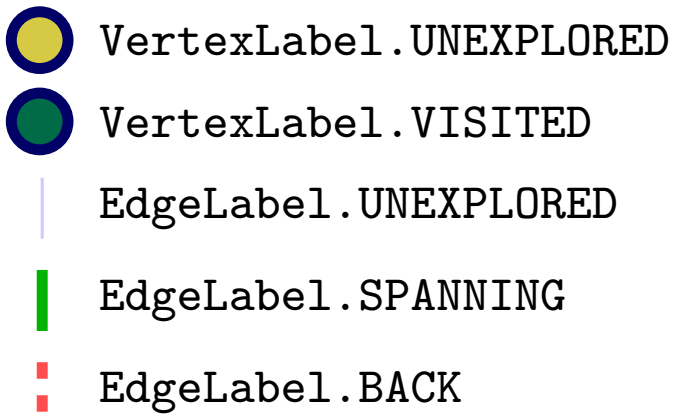
`DFSOne(G, B)` ($\rightarrow$ A, C)

`DFSOne(G, C)` ($\rightarrow$ A, B, D, E)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

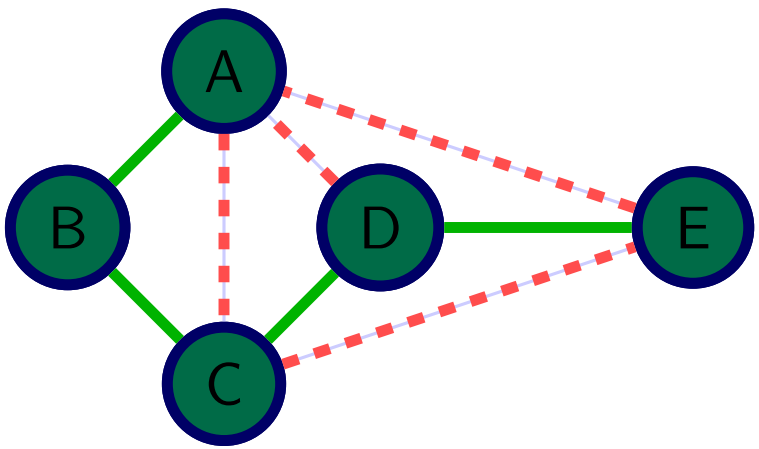
`DFS(G)`

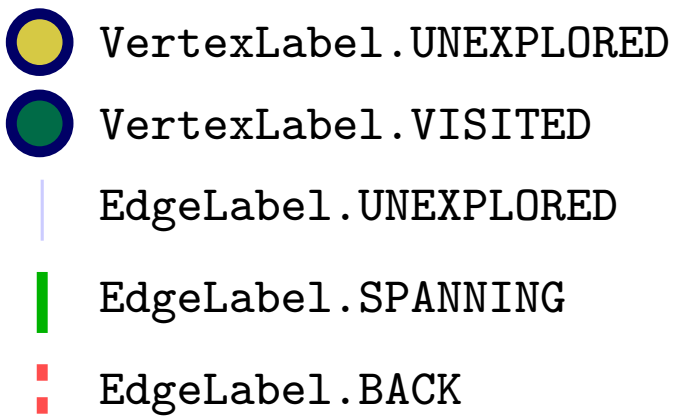
`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

`DFSOne(G, B)` ($\rightarrow$ A, C)

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





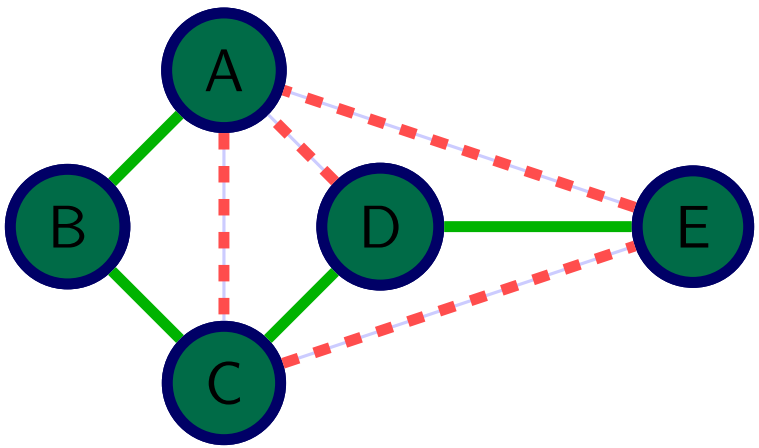
Call Stack

`DFS(G)`

`DFSOne(G, A)` ($\rightarrow$ B, C, D, E)

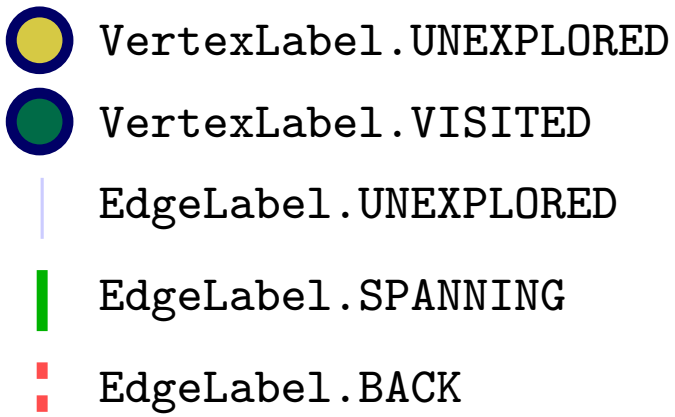
© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)



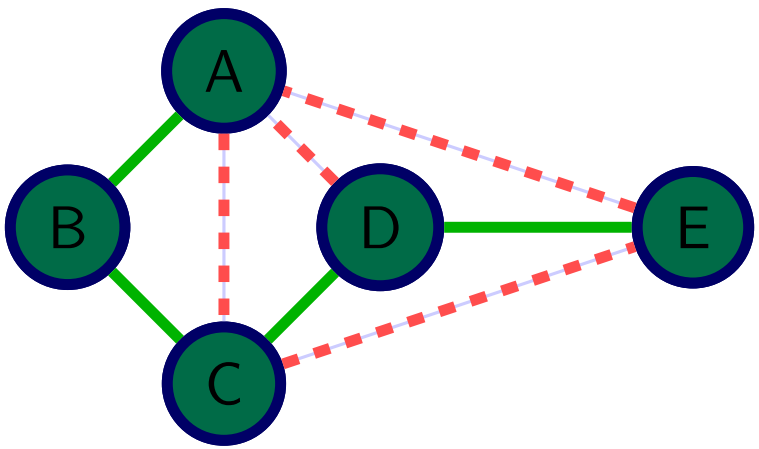
Call Stack

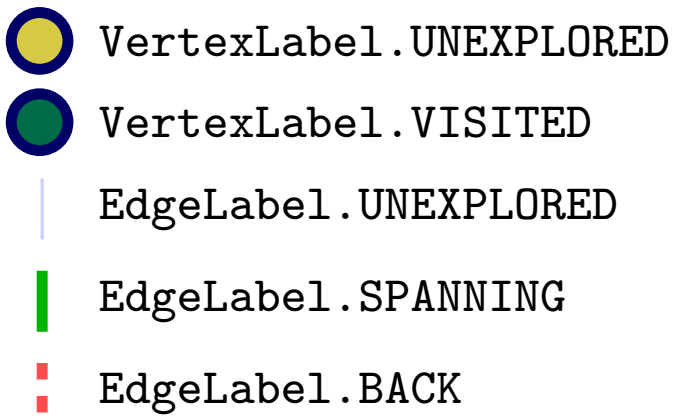
`DFS(G)`



© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)





Call Stack

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS on Mazes

The DFS algorithm is like our stack-based maze solver

- Mark each grid square with `VISITED`
- Mark each path with `SPANNING`
- Only visit each vertex once.

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS on Mazes

The DFS algorithm is like our stack-based maze solver

- Mark each grid square with `VISITED`
- Mark each path with `SPANNING`
- Only visit each vertex once.

DFS won't necessarily find the shortest paths.

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFS

What's the complexity?

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

```
1  public enum VertexLabel { UNEXPLORED, VISITED }
2  public enum EdgeLabel { UNEXPLORED, SPANNING, BACK }
3
4  public void DFS(Graph<VertexLabel,EdgeLabel> graph)
5  {
6    for(Vertex v : graph.vertices)
7                  { v.setLabel(VertexLabel.UNEXPLORED); }
8    for(Edge e   : graph.edges)
9                  { e.setLabel(EdgeLabel.UNEXPLORED); }
10   for(Vertex v : graph.vertices)
11   {
12     if(v.getLabel() == VertexLabel.UNEXPLORED)
13     {
14       DFSOne(graph, v)
15     }
16   }
17 }
```

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

```
1   public enum VertexLabel { UNEXPLORED, VISITED }
2   public enum EdgeLabel { UNEXPLORED, SPANNING, BACK }
3
4   public void DFS(Graph<VertexLabel,EdgeLabel> graph)
5   {
6     O(N)
7
8     for(Edge e    : graph.edges)
9                    { e.setLabel(EdgeLabel.UNEXPLORED); }
10    for(Vertex v : graph.vertices)
11    {
12      if(v.getLabel() == VertexLabel.UNEXPLORED)
13      {
14        DFSOne(graph, v)
15      }
16    }
17  }
```

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

```
1  public enum VertexLabel { UNEXPLORED, VISITED }
2  public enum EdgeLabel { UNEXPLORED, SPANNING, BACK }
3
4  public void DFS(Graph<VertexLabel,EdgeLabel> graph)
5  {
6    O(N)
7
8    O(M)
9
10   for(Vertex v : graph.vertices)
11   {
12     if(v.getLabel() == VertexLabel.UNEXPLORED)
13     {
14       DFSOne(graph, v)
15     }
16   }
17 }
```

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

```
1   public enum VertexLabel { UNEXPLORED, VISITED }
2   public enum EdgeLabel { UNEXPLORED, SPANNING, BACK }
3
4   public void DFS(Graph<VertexLabel,EdgeLabel> graph)
5   {
6     O(N)
7
8     O(M)
9
10    O(N) times
11    {
12      if(v.getLabel() == VertexLabel.UNEXPLORED)
13      {
14        DFSOne(graph, v)
15      }
16    }
17  }
```

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

```
1   public enum VertexLabel { UNEXPLORED, VISITED }
2   public enum EdgeLabel { UNEXPLORED, SPANNING, BACK }
3
4   public void DFS(Graph<VertexLabel,EdgeLabel> graph)
5   {
6     O(N)
7
8     O(M)
9
10    O(N) times
11    {
12      O(1)
13      {
14        DFSOne(graph, v)
15      } else {
16        O(1)
17      }
18    }
19  }
```

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Observation: `DFSOne` is called on each vertex at most once.

- If `v`'s label is `VISITED`, both `DFSOne` and `DFS` skip it

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Observation: `DFSOne` is called on each vertex at most once.

- If `v`'s label is `VISITED`, both `DFSOne` and `DFS` skip it

$O(N)$ calls to `DFSOne`

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

Observation: `DFSOne` is called on each vertex at most once.

- If `v`'s label is `VISITED`, both `DFSOne` and `DFS` skip it

$O(N)$ calls to `DFSOne`

What's the runtime of `DFSOne` **excluding recursive calls**?

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFSOne

```
1   public void DFSOne(Graph<...> graph, Vertex start)
2   {
3     start.setLabel(VertexLabel.VISITED)
4
5     for( edge : start.getIncident() ) {
6       if(edge.getLabel == EdgeLabel.UNEXPLORED){
7         Vertex otherVertex = edge.getOppositeVertex(start);
8
9         if(otherVertex.getLabel == VertexLabel.UNEXPLORED){
10          edge.setLabel(EdgeLabel.SPANNING);
11          DFSOne(graph, otherVertex);
12        } else {
13          edge.setLabel(EdgeLabel.BACK);
14        }
15      }
16    }
17  }
```

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

DFSOne

```
1  public void DFSOne(Graph<...> graph, Vertex start)
2  {
3    O(1)
4
5    O(deg(start)) times {
6      O(1) {
7        O(1)
8
9        O(1) {
10         O(1)
11         DFSOne(graph, otherVertex);
12       } else {
13         O(1)
14       }
15     }
16   }
17 }
```

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

What's the runtime of `DFSOne` **excluding recursive calls**?

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

What's the runtime of `DFSOne` **excluding recursive calls**?

$$O(\texttt{deg}(\texttt{start}))$$

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

What's the sum over all calls to `DFSOne`

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

What's the sum over all calls to `DFSOne`

$$\sum_{v \in V} O(\deg(v))$$

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

What's the sum over all calls to `DFSOne`

$$\sum_{v \in V} O(\deg(v))$$

$$= O\left(\sum_{v \in V} \deg(v)\right)$$

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Depth First Search (DFS)

What's the sum over all calls to `DFSOne`

$$\sum_{v \in V} O(\deg(v))$$

$$= O\left(\sum_{v \in V} \deg(v)\right)$$

$$= O(2M) = O(M)$$

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

To summarize...

Mark Vertices `UNVISITED`
Mark Edges `UNVISITED`
`DFS` Vertex Loop
All calls to `DFSOne`

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

To summarize...

Mark Vertices `UNVISITED`	$O(N)$
Mark Edges `UNVISITED`	
`DFS` Vertex Loop	
All calls to `DFSOne`	

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

To summarize...

Mark Vertices `UNVISITED`	$O(N)$
Mark Edges `UNVISITED`	$O(M)$
`DFS` Vertex Loop	
All calls to `DFSOne`	

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

To summarize...

Mark Vertices `UNVISITED`	$O(N)$
Mark Edges `UNVISITED`	$O(M)$
`DFS` Vertex Loop	$O(N)$
All calls to `DFSOne`	

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

To summarize...

Mark Vertices `UNVISITED`	$O(N)$
Mark Edges `UNVISITED`	$O(M)$
`DFS` Vertex Loop	$O(N)$
All calls to `DFSOne`	$O(M)$

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

To summarize...

Mark Vertices `UNVISITED`	$O(N)$
Mark Edges `UNVISITED`	$O(M)$
`DFS` Vertex Loop	$O(N)$
All calls to `DFSOne`	$O(M)$
	$O(N+M)$

© 2024 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY