Breadth-First Search

- "Explore" a graph, turning it into a tree
  - One vertex at a time
  - Expand frontier of explored vertices across the *breadth* of the frontier
- Builds a tree over the graph
  - Pick a *source vertex* to be the root
  - Find ("discover") its children, then their children, etc.

Breadth-First Search

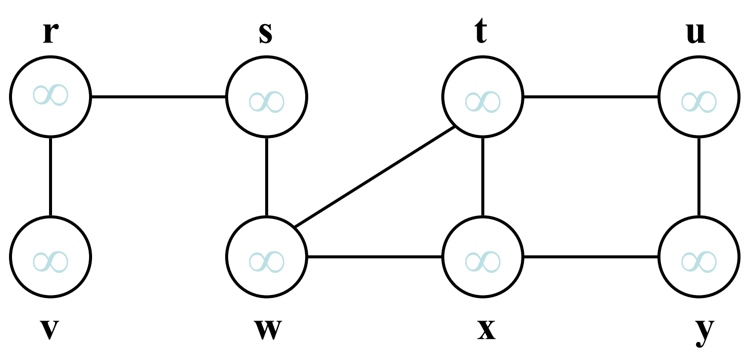
- Again will associate vertex "colors" to guide the algorithm
  - White vertices have not been discovered
    - All vertices start out white
  - Grey vertices are discovered but not fully explored
    - They may be adjacent to white vertices
  - Black vertices are discovered and fully explored
    - They are adjacent only to black and gray vertices
- Explore vertices by scanning adjacency list of grey vertices

Breadth-First Search

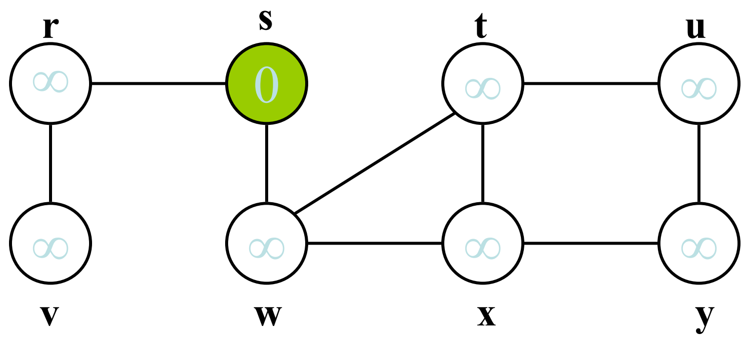
```
BFS(G, s) {
    initialize vertices;
    Q = {s};        // Q is a queue (duh); initialize
  to s
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v ∈ u->adj {
            if (v->color == WHITE)
                v->color = GREY;
                v->d = u->d + 1;
                v->p = u;
                Enqueue(Q, v);
        }
        u->color = BLACK;
    }
}
```

What does `v->d` represent?
What does `v->p` represent?

Breadth-First Search: Example

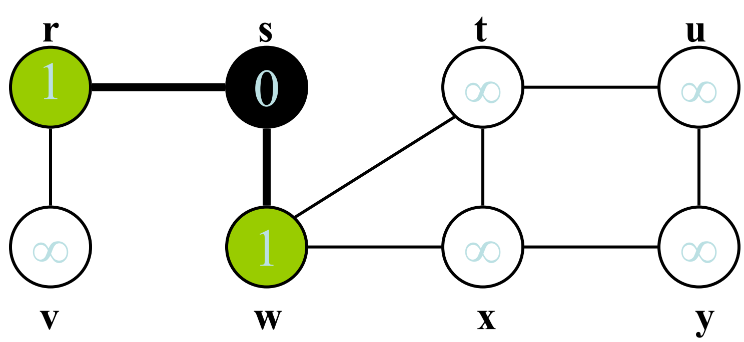


Breadth-First Search: Example



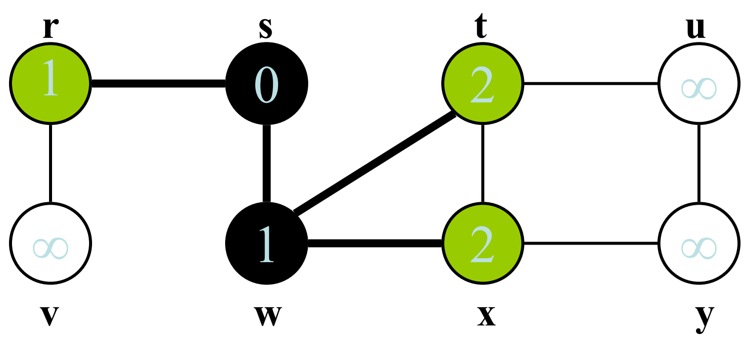
Q: | s |

Breadth-First Search: Example



Q: | w | r |

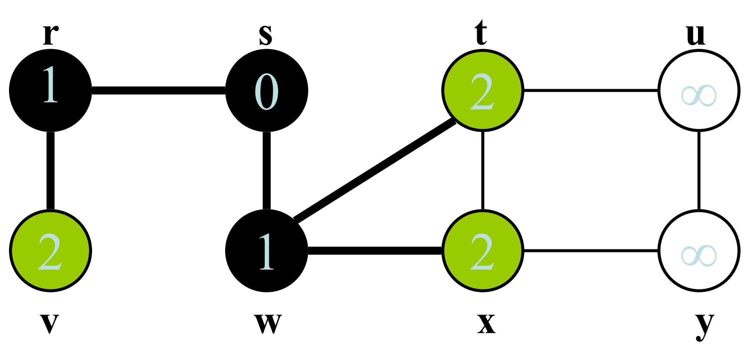
Breadth-First Search: Example



Q:

r	t	x

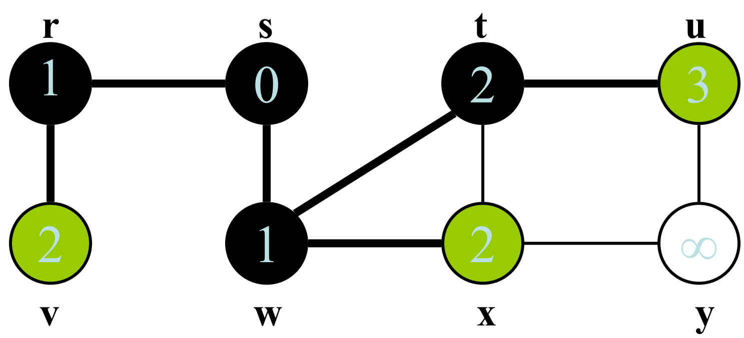
Breadth-First Search: Example



Q:

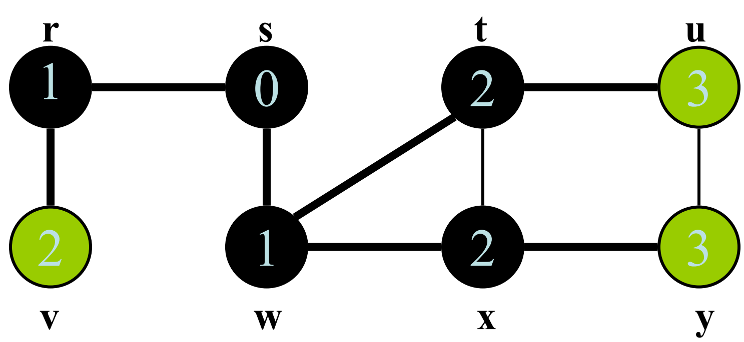
t	x	v

Breadth-First Search: Example



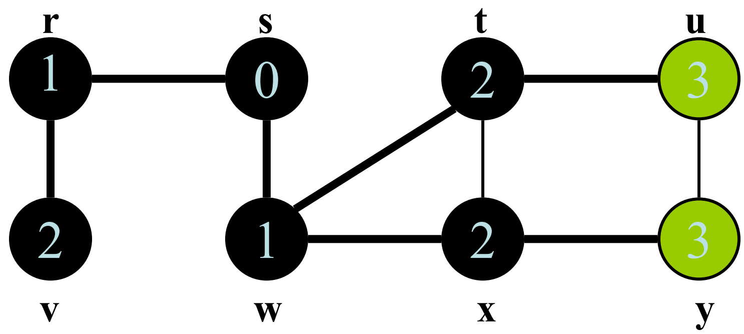
Q: | x | v | u |

Breadth-First Search: Example



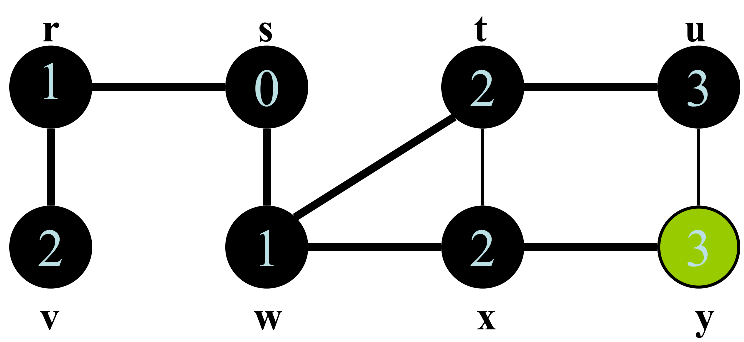
Q: | v | u | y |

Breadth-First Search: Example



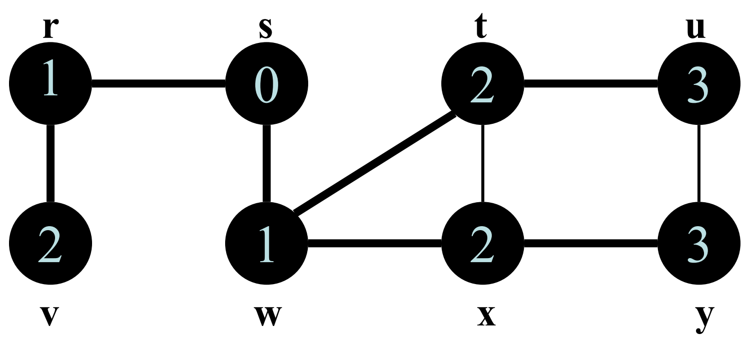
Q: | u | y |

Breadth-First Search: Example



Q: | y |

Breadth-First Search: Example



Q: Ø

BFS: The Code Again

```
BFS(G, s) {
     initialize vertices;
     Q = {s};
     while (Q not empty) {
          u = RemoveTop(Q);
          for each v ∈ u->adj {
               if (v->color == WHITE)
                    v->color = GREY;
                    v->d = u->d + 1;
                    v->p = u;
                    Enqueue(Q, v);
          }
          u->color = BLACK;
     }
}
```

Touch every vertex: O(V)

u = every vertex, but only once (Why?)

So v = every vertex that appears in some other vert's adjacency list

What will be the running time?

Total running time: O(V+E)

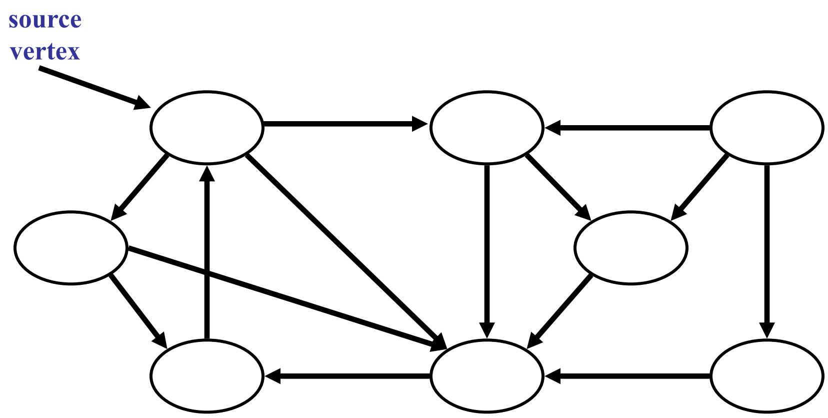
Depth-First Search

- *Depth-first search* is another strategy for exploring a graph
- Explore "deeper" in the graph whenever possible
- Edges are explored out of the most recently discovered vertex v that still has unexplored edges
- When all of v's edges have been explored, backtrack to the vertex from which v was discovered

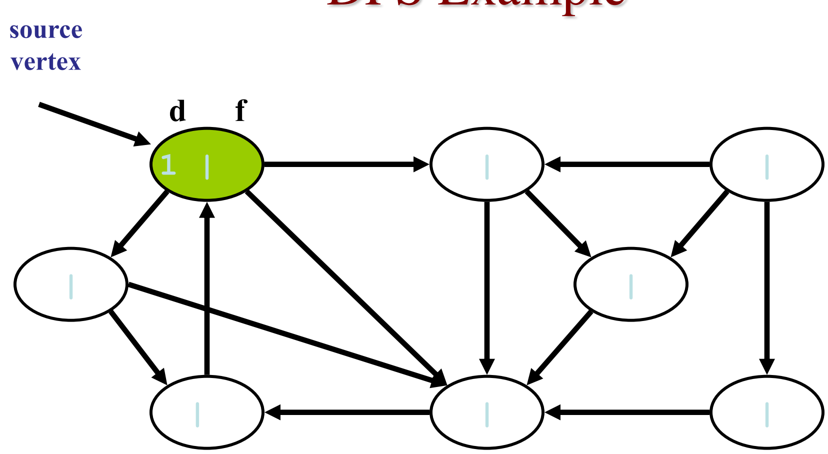
Depth-First Search

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

DFS Example

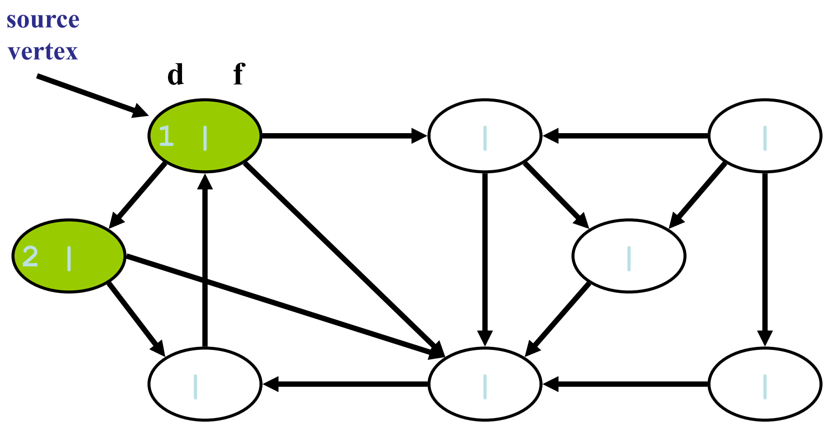


DFS Example

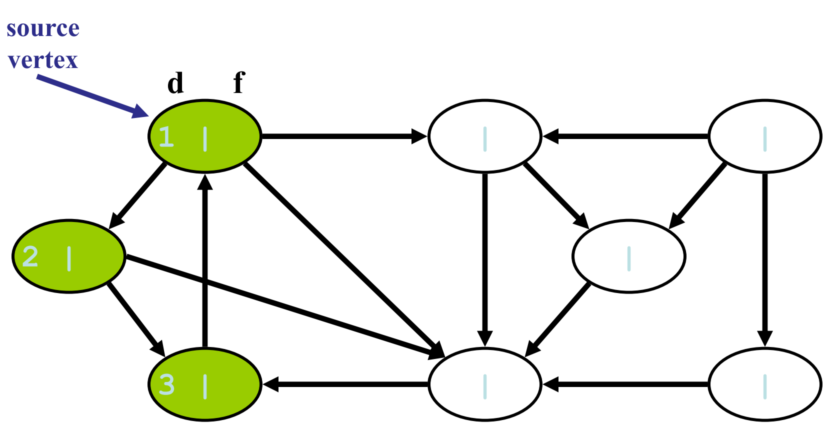


Green in figure -> gray in code

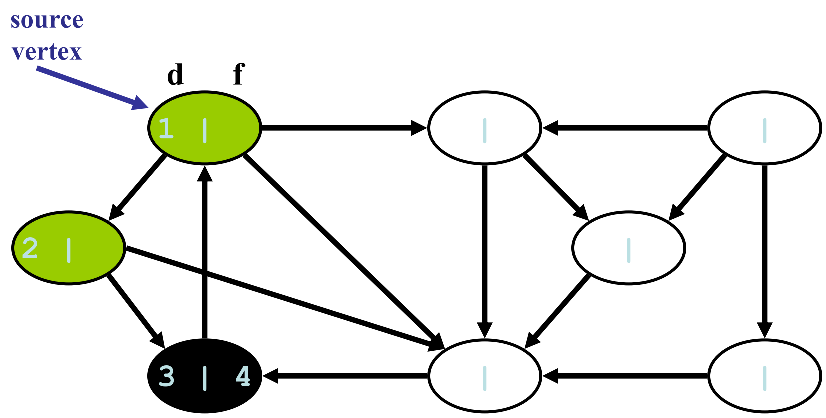
DFS Example



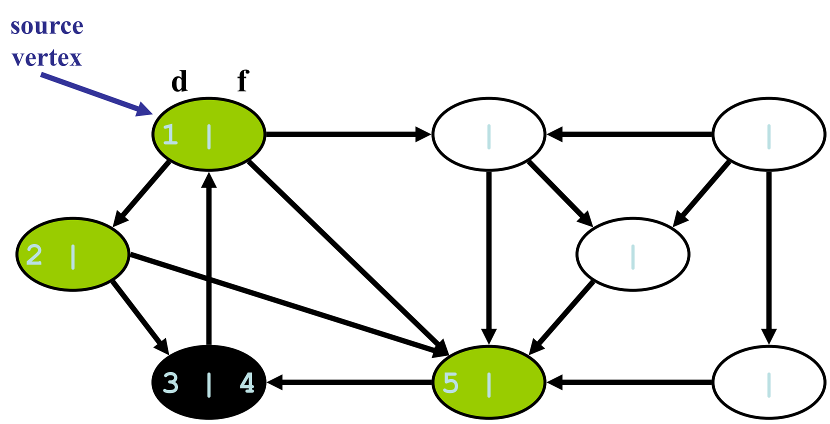
DFS Example



DFS Example



DFS Example



Depth-First Search: The Code

```
DFS(G)
{
   for each vertex u ∈ G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u ∈ G->V
   {
      if (u->color ==
 WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
   u->color = GREY;
   time = time+1;
   u->d = time;
   for each v ∈ u->Adj[]
   {
      if (v->color ==
 WHITE)
         DFS_Visit(v);
   }
   u->color = BLACK;
   time = time+1;
   u->f = time;
}
```

Depth-First Search: The Code

```
DFS(G)
{
   for each vertex u ∈ G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u ∈ G->V
   {
      if (u->color ==
 WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
   u->color = GREY;
   time = time+1;
   u->d = time;
   for each v ∈ u->Adj[]
   {
      if (v->color ==
 WHITE)
         DFS_Visit(v);
   }
   u->color = BLACK;
   time = time+1;
   u->f = time;
}
```


What does u->d represent?

Depth-First Search: The Code

```
DFS(G)
{
   for each vertex u ∈ G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u ∈ G->V
   {
      if (u->color ==
  WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
   u->color = GREY;
   time = time+1;
   u->d = time;
   for each v ∈ u->Adj[]
   {
      if (v->color ==
  WHITE)
         DFS_Visit(v);
   }
   u->color = BLACK;
   time = time+1;
   u->f = time;
}
```

What does `u->f` represent?

Depth-First Search: The Code

```
DFS(G)
{
   for each vertex u ∈ G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u ∈ G->V
   {
      if (u->color ==
  WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
   u->color = GREY;
   time = time+1;
   u->d = time;
   for each v ∈ u->Adj[]
   {
      if (v->color ==
  WHITE)
         DFS_Visit(v);
   }
   u->color = BLACK;
   time = time+1;
   u->f = time;
}
```

Will all vertices eventually be colored black?

Depth-First Search: The Code

```
DFS(G)
{
   for each vertex u ∈ G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u ∈ G->V
   {
      if (u->color ==
  WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
   u->color = GREY;
   time = time+1;
   u->d = time;
   for each v ∈ u->Adj[]
   {
      if (v->color ==
  WHITE)
         DFS_Visit(v);
   }
   u->color = BLACK;
   time = time+1;
   u->f = time;
}
```

What will be the running time?

Depth-First Search: The Code

```
DFS(G)
{
   for each vertex u ∈ G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u ∈ G->V
   {
      if (u->color ==
  WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
   u->color = GREY;
   time = time+1;
   u->d = time;
   for each v ∈ u->Adj[]
   {
      if (v->color ==
  WHITE)
         DFS_Visit(v);
   }
   u->color = BLACK;
   time = time+1;
   u->f = time;
}
```


Running time: $O(n^2)$ because call DFS_Visit on each vertex, and the loop over Adj[] can run as many as |V| times

Depth-First Search: The Code

```
DFS(G)
{
   for each vertex u ∈ G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u ∈ G->V
   {
      if (u->color ==
  WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
   u->color = GREY;
   time = time+1;
   u->d = time;
   for each v ∈ u->Adj[]
   {
      if (v->color ==
  WHITE)
         DFS_Visit(v);
   }
   u->color = BLACK;
   time = time+1;
   u->f = time;
}
```

BUT, there is actually a tighter bound.
How many times will DFS_Visit() actually be called?

Depth-First Search: The Code

```
DFS(G)
{
   for each vertex u ∈ G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u ∈ G->V
   {
      if (u->color ==
  WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
   u->color = GREY;
   time = time+1;
   u->d = time;
   for each v ∈ u->Adj[]
   {
      if (v->color ==
  WHITE)
         DFS_Visit(v);
   }
   u->color = BLACK;
   time = time+1;
   u->f = time;
}
```

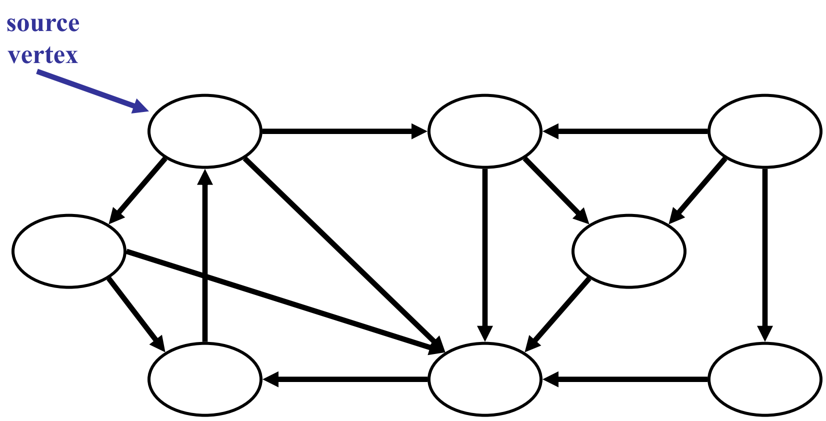

So, running time of DFS = O(V+E)

Depth-First Sort Analysis

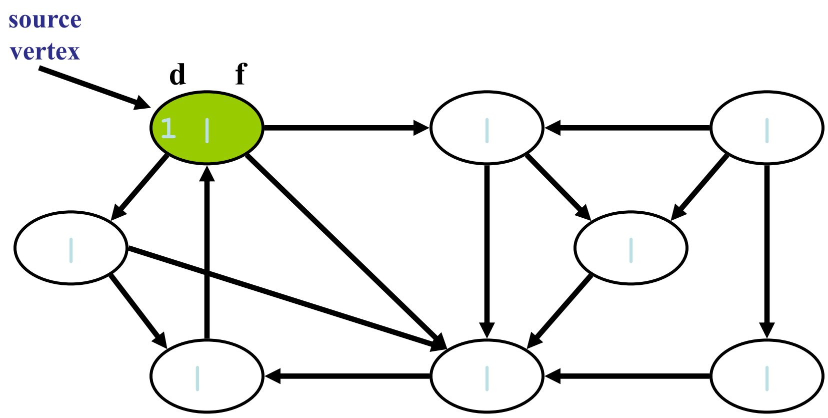
- This running time argument is an informal example of *amortized analysis*

- "Charge" the exploration of edge to the edge:
- Each loop in DFS_Visit can be attributed to an edge in the graph
- Runs once/edge if directed graph, twice if undirected
- Thus loop will run in O(E) time, algorithm O(V+E)
- Considered linear for graph, b/c adj list requires O(V+E) storage
- Important to be comfortable with this kind of reasoning and analysis

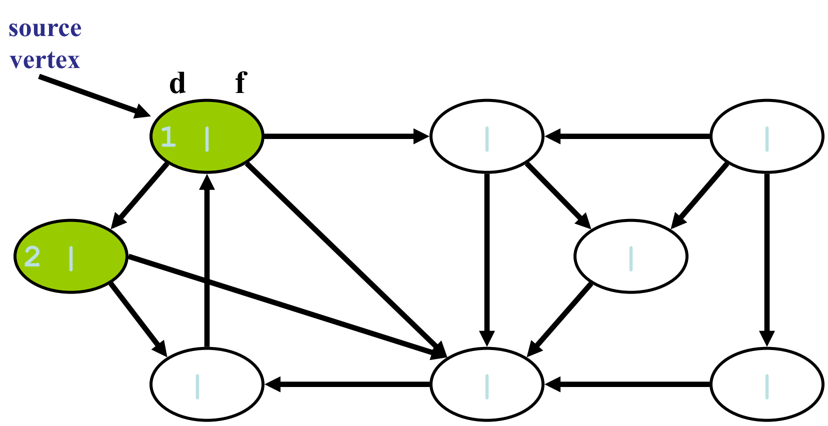
DFS Example



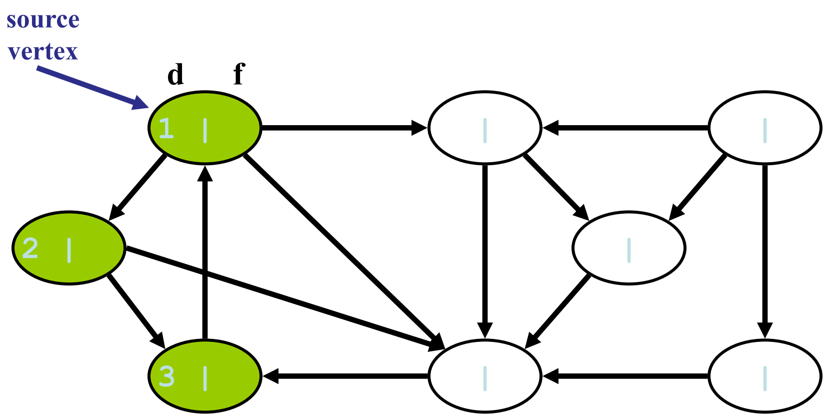
DFS Example



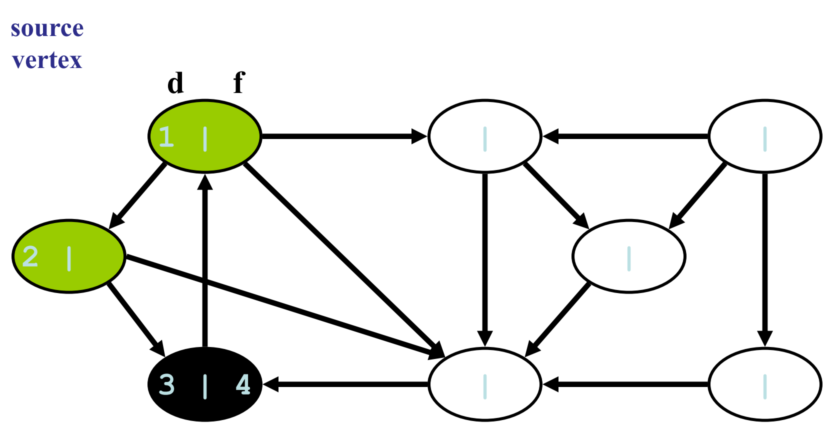
DFS Example



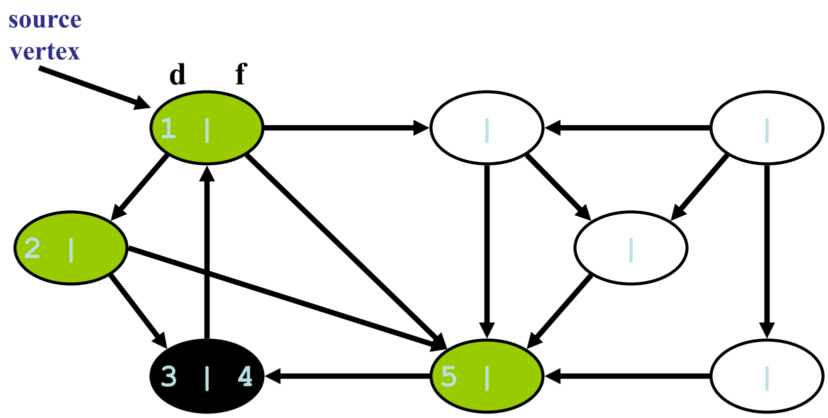
DFS Example



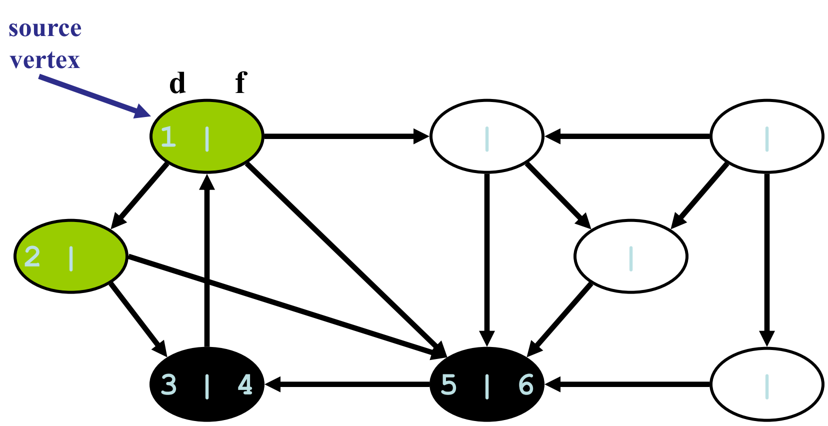
DFS Example



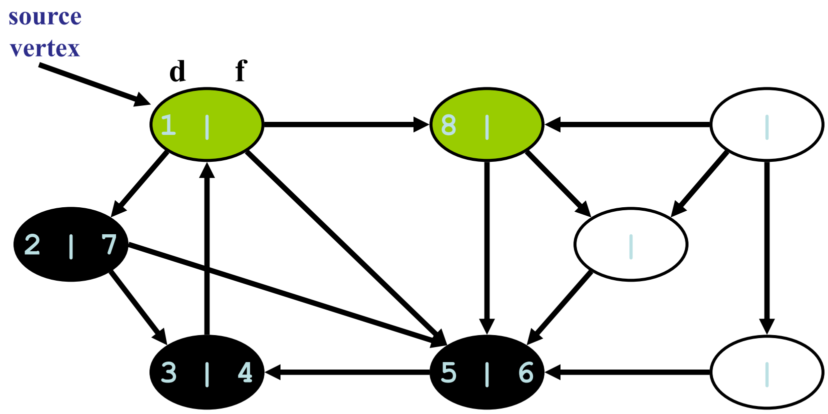
DFS Example



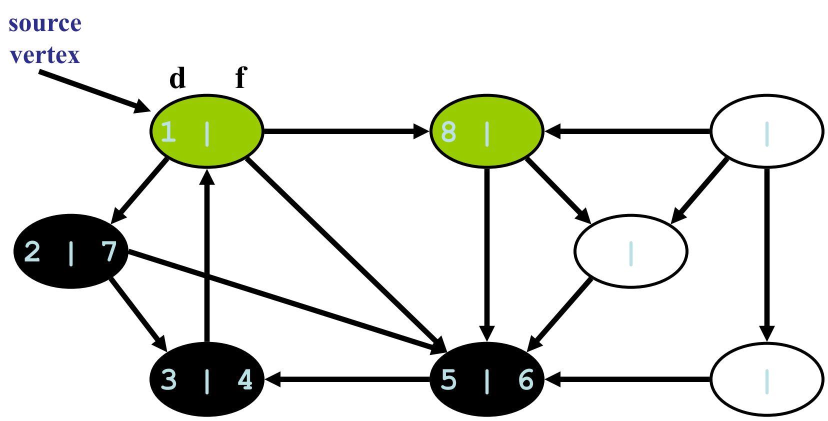
DFS Example



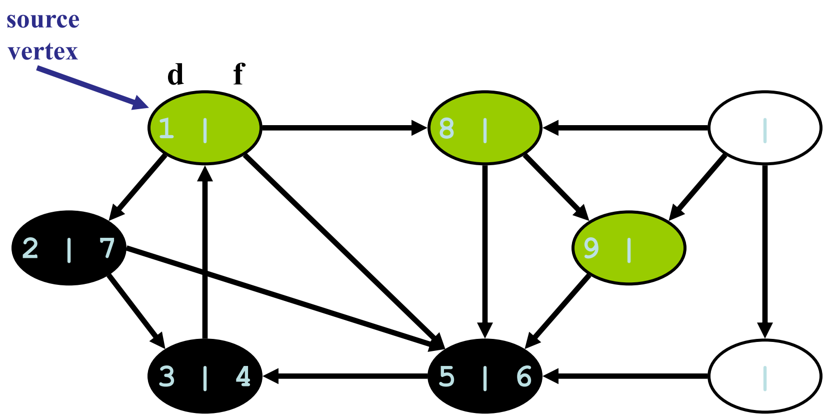
DFS Example



DFS Example

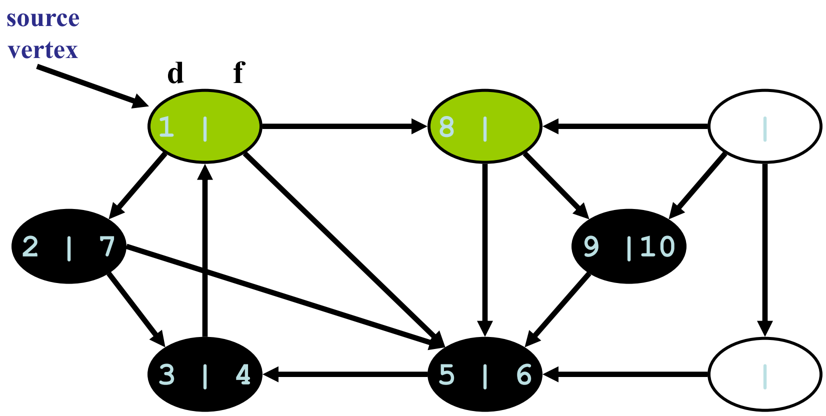


DFS Example

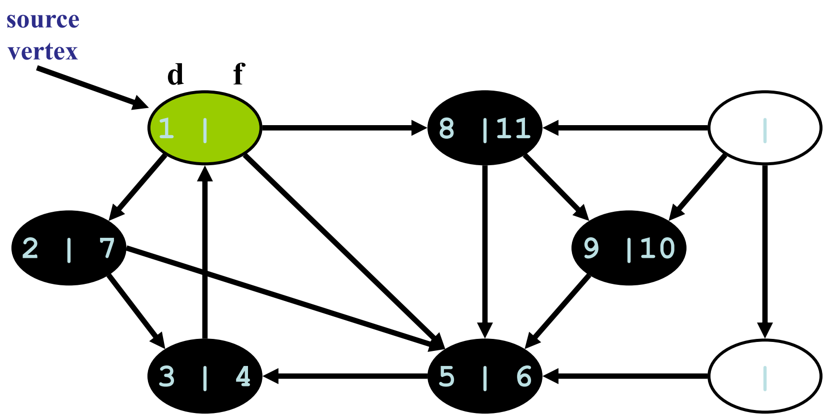


What is the structure of the green vertices?
What do they represent?

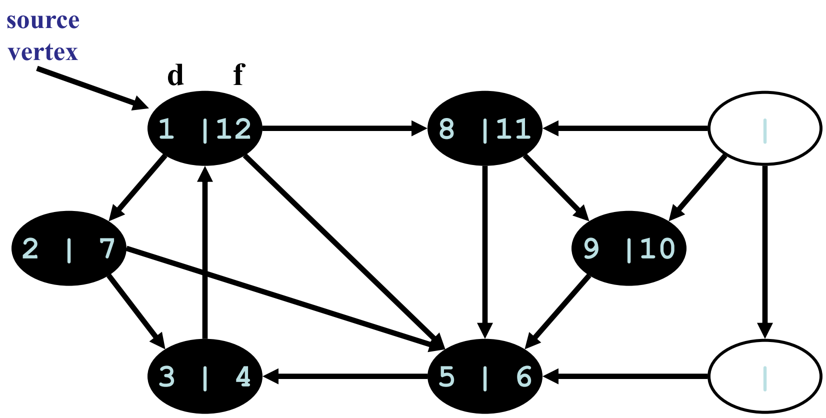
DFS Example



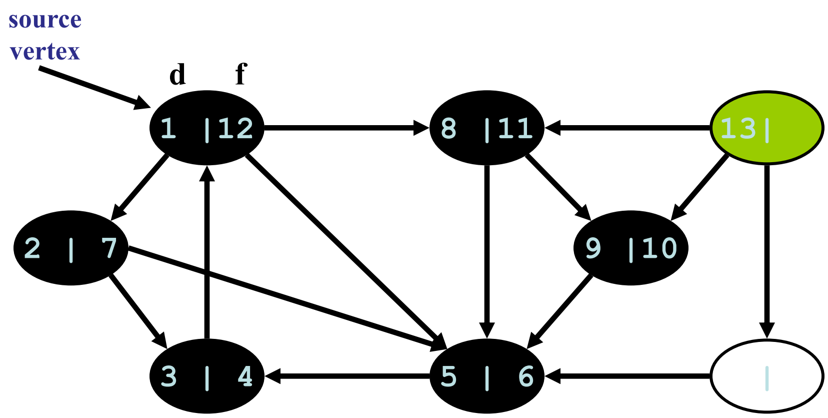
DFS Example



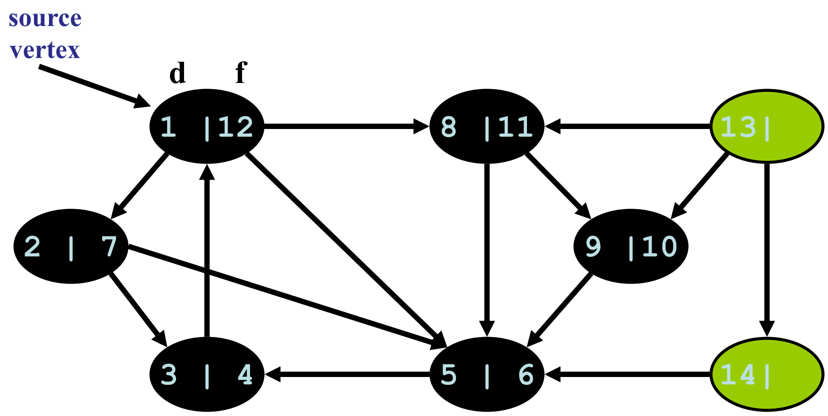
DFS Example



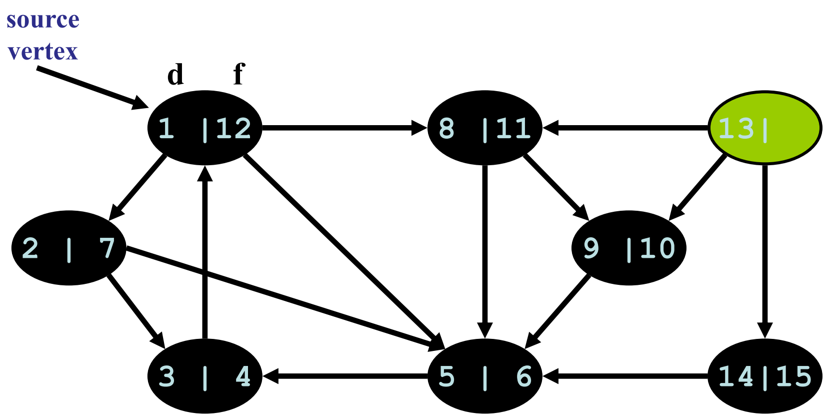
DFS Example



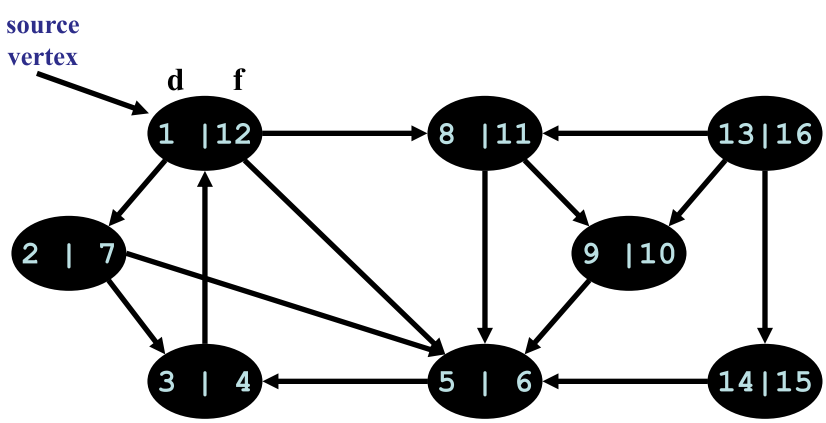
DFS Example



DFS Example



DFS Example



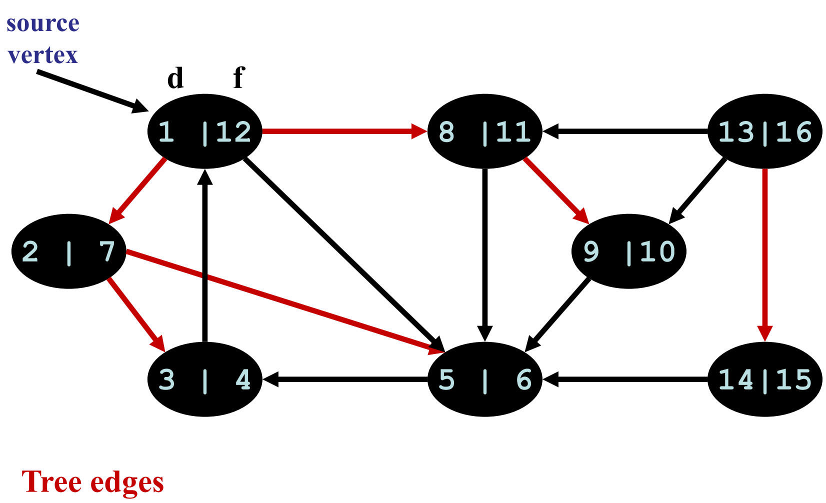
DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:

- *Tree edge*: encounter new (white) vertex
- The tree edges form a spanning forest

- *Can tree edges form cycles? Why or why not?*

DFS Example

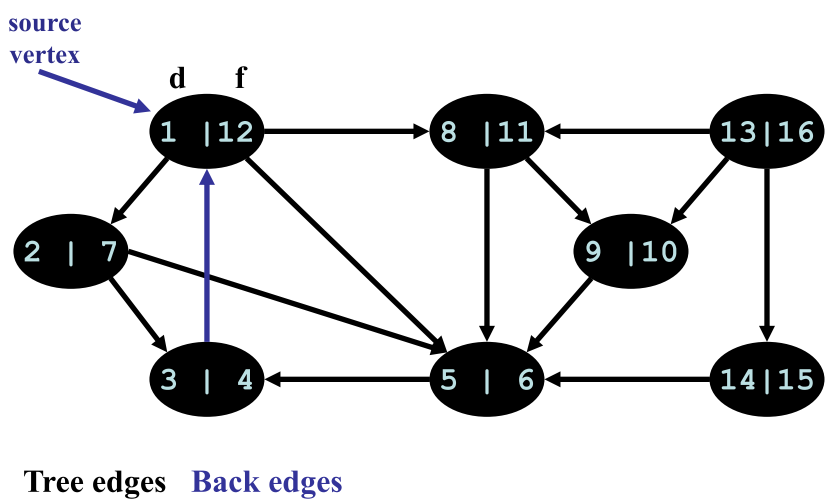


DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:

- *Tree edge*: encounter new (white) vertex
- *Back edge*: from descendent to ancestor
  Encounter a grey vertex (grey to grey)

DFS Example

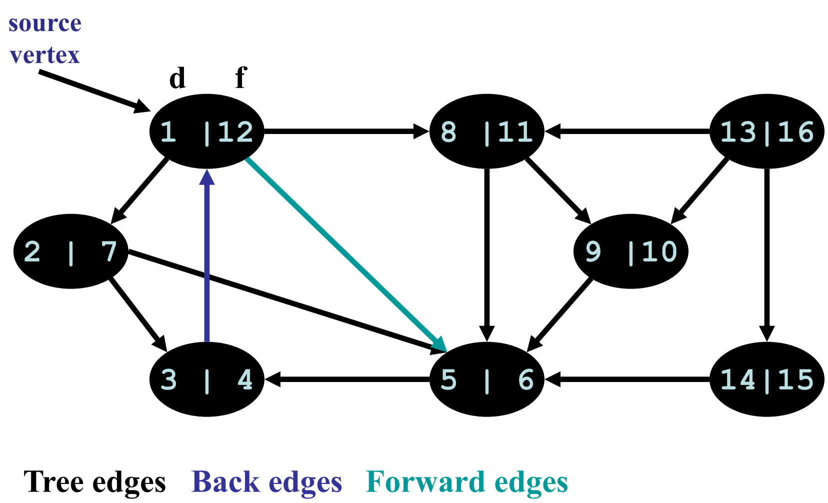


DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  Tree edge: encounter new (white) vertex
  Back edge: from descendent to ancestor
  Forward edge: from ancestor to descendent

  Not a tree edge, though
  From grey node to black node

DFS Example

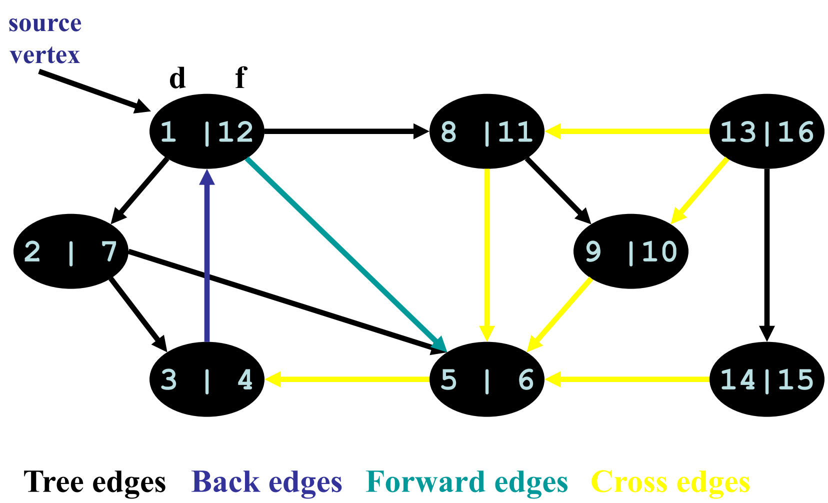


DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  Tree edge: encounter new (white) vertex
  Back edge: from descendent to ancestor
  Forward edge: from ancestor to descendent
  Cross edge: between a tree or subtrees

  From a grey node to a black node

DFS Example



DFS: Kinds of edges

- DFS introduces an important distinction among edges in the original graph:
  Tree edge: encounter new (white) vertex
  Back edge: from descendent to ancestor
  Forward edge: from ancestor to descendent
  Cross edge: between a tree or subtrees
- Note: tree & back edges are important; most algorithms don't distinguish forward & cross