Recursive Functions

Sai-Keung Wong

National Yang Ming Chiao Tung University

Recursive Functions

- A recursive function calls itself in a calling sequence.
- For example

```
void f( … ) {
   …
   f( … )
   …

 }
```

```
void g( … ) {
   …
   f( … )
   …

}
void f( … ) {
   …
   g( … )
   …

 }
```

Recursive Functions

- A recursive function solves a problem by solving the associated sub-problems and combining the solutions of the sub-problems.

```
void f( … ) {
  …
  f( … )
  …

}
```

```
void g( … ) {
  …
  f( … )
  …

}
void f( … ) {
  …
  g( … )
  …

}
```

Recursive Functions

What is the purpose of the code fragment?

```
int s = 1;
if ( n == 0 ) return s;
s = n*f(n-1);
```

Recursive Functions

What is the purpose of the code fragment?

```
int s = 1;
if ( n == 0 ) return s;
s = n*f(n-1);
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).

Recursive Functions

What is the purpose of the code fragment?

```
int u( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.
???

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

Dry-run: tracing the function call(s)

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

Dry-run: tracing the function call(s)

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

n = 5

```
f(5) {
  …
  s = 5*f(4);
  …
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

Dry-run: tracing the function call(s)

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be terminated?

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5	
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……

Dry-run: tracing the function call(s)

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5	
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……

Dry-run: tracing the function call(s)

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

n = 5

```
f(5) {
  …
  s = 5*f(4);
  …
}
```

……

```
f(n=?) {
  …

  …
}
```

Dry-run: tracing the function call(s)

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5		
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

Dry-run: tracing the function call(s)

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5		
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

Combining the results of sub-problems

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

n = 5

```
f(5) {
  …
  s = 5*f(4);
  …
}
```

……

```
f(0) {
  …
  return 1;
  …
}
```

Combining the results of sub-problems
s=7*f(6)

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be terminated?

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5		
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

s=7*6*f(5)

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

n = 5

```
f(5) {
  …
  s = 5*f(4);
  …
}
```

……

```
f(0) {
  …
  return 1;
  …
}
```

s=7*6*5*f(4)

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5		
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

s=7*6*5*f(4)……

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

n = 5

```
f(5) {
  …
  s = 5*f(4);
  …
}
```

……

```
f(0) {
  …
  return 1;
  …
}
```

s=7*6*5*f(4)……*1

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

n = 5

```
f(5) {
  …
  s = 5*f(4);
  …
}
```

……

```
f(0) {
  …
  return 1;
  …
}
```

s=7*6*5*f(4)……*1

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5		
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

s=7*6*5*4*……*1

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

n = 5

```
f(5) {
  …
  s = 5*f(4);
  …
}
```

……

```
f(0) {
  …
  return 1;
  …
}
```

s=7*6*5*4*3*……*1

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5		
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

s=7*6*5*4*3*2*……*1

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be terminated?

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5		
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

s=7*6*5*4*3*2*1*……*1

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be terminated?

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7	n = 6	n = 5		
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

s=7*6*5*4*3*2*1

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n = 7

```
f(7) {
  …
  s = 7*f(6);
  …
}
```

n = 6

```
f(6) {
  …
  s = 6*f(5);
  …
}
```

n = 5

```
f(5) {
  …
  s = 5*f(4);
  …
}
```

……

```
f(0) {
  …
  return 1;
  …
}
```

s=7*6*5*4*3*2*1 = 7!

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be terminated?

Need boundary case(s)

n	n-1	n -2		0
f(7) { … s = 7*f(6); … }	f(6) { … s = 6*f(5); … }	f(5) { … s = 5*f(4); … }	……	f(0) { … return 1; … }

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n	n-1	n -2		0
f(n) { … s = n *f(n-1); … }	f(6) { … s = (n-1) *f(n-2); … }	f(5) { … s = (n-2) *f(n-3); … }	……	f(0) { … return 1; … }

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

When should be

Need boundary case(s)

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

n

```
f(n) {
  …
  s = n
     *f(n-1);
  …
}
```

n-1

```
f(6) {
  …
  s = (n-1)
     *f(n-2);
  …
}
```

n -2

```
f(5) {
  …
  s = (n-2)
     *f(n-3);
  …
}
```

……

0

```
f(0) {
  …
  return 1;
  …
}
```

s=n*(n-1)*(n-2)*……*1

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Need to give the high level idea about the function f.

f( ) performs the same action every time.
But n is reduced by 1 every time.

When should be terminated?

Need boundary case(s)

n	n-1	n -2		0
f(n) { … s = n *f(n-1); … }	f(6) { … s = (n-1) *f(n-2); … }	f(5) { … s = (n-2) *f(n-3); … }	……	f(0) { … return 1; … }

s=n*(n-1)*(n-2)*……*1*1 = n!

For the case(s), f( ) does not call itself. (directly or indirectly).

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Recursive Functions

```
int s = 1;
if ( n == 0 ) return s;
s = n*f(n-1);
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Recursive Functions

```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

If n is zero, return s.
Otherwise, set s as n*f(n-1).
Then return s.

Recursive Functions

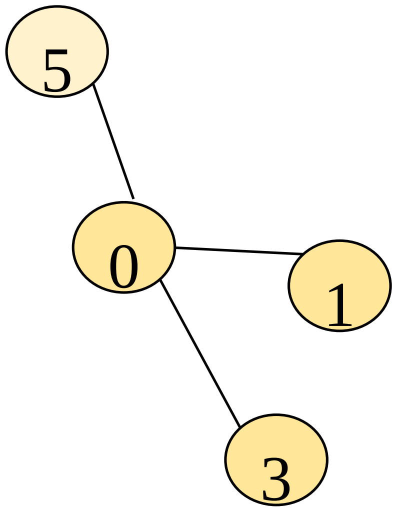
To understand a recursive function
(It does one job.)

① Dry-run: tracing the function call(s)
② Combining the results of sub-problems
③ Generalize the results

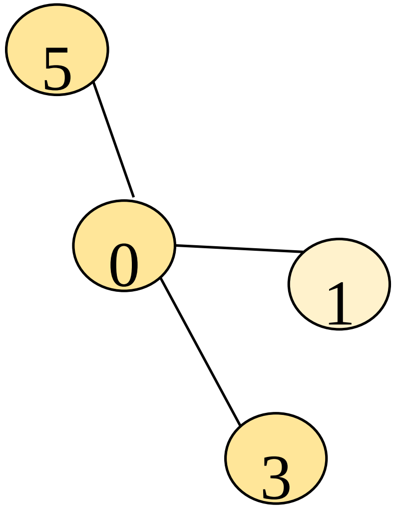
```
int f( int n ) {
    int s = 1;
    if ( n == 0 ) return s;
    s = n*f(n-1);
    return s;
}
```

Depth-First Search

Traverse a graph or a tree in a depth-first manner. That's that, a node is explored if it has not been visited.



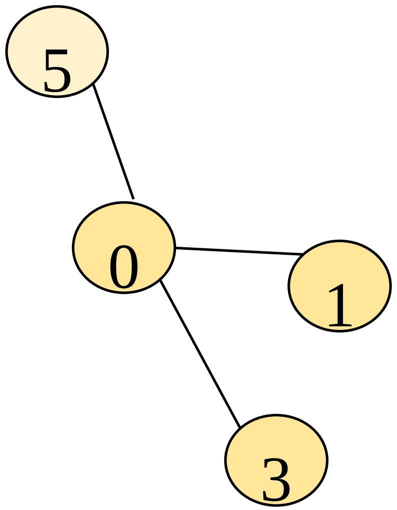

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



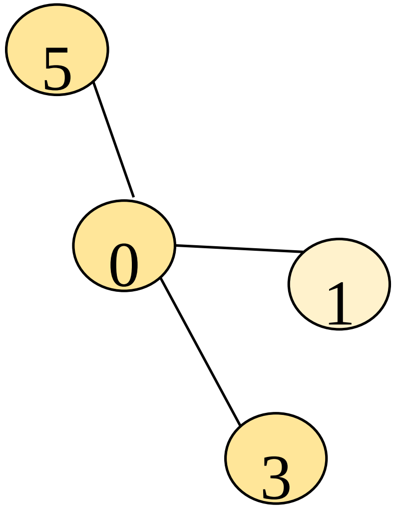
Depth-First Search

Traverse a graph or a tree in a depth-first manner. That's that, a node is explored if it has not been visited.

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```

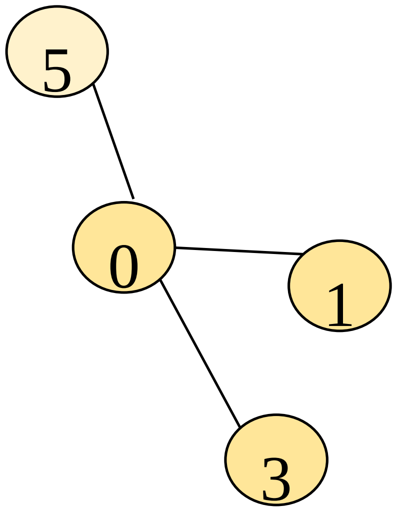



```
Invoke dfs(Node 0). Possible result(s):
```

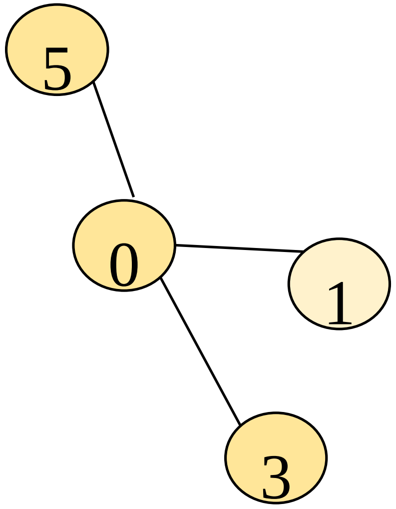


Depth-First Search

Traverse a graph or a tree in a depth-first manner. That’s that, a node is explored if it has not been visited.




```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```




```
Invoke dfs(Node 0). Possible result(s):
3,0,1,5
3,0,5,1
```

Depth-First Search

Traverse a graph or a tree in a depth-first manner. That's that, a node is explored if it has not been visited.

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      foo(w);
```

Depth-First Search

Given a node v = v_0

```
visit v;
for each neighbor w of v
  if (w is not visited)
    foo(w);
```

Depth-First Search

Given a node $v = v_0$

What do you do?

```
visit v;
for each neighbor w of v
  if (w is not visited)
    foo(w);
```

Depth-First Search

Given a node $v = v_0$

```
visit v;
for each neighbor w of v
  if (w is not visited)
    foo(w);
```

1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

Depth-First Search

Given a node $v = v_1$

```
visit v;
for each neighbor w of v
  if (w is not visited)
    foo(w);
```

1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

Depth-First Search

Given a node v = v_2

```
visit v;
for each neighbor w of v
  if (w is not visited)
    foo(w);
```

1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

Depth-First Search

Given a node $v = v_j$

```
visit v;
for each neighbor w of v
  if (w is not visited)
    foo(w);
```

1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      foo(w);
```

1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      foo(w);
```

1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```

1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```

1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```

1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4.   call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```

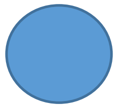
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4.   call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



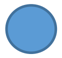
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



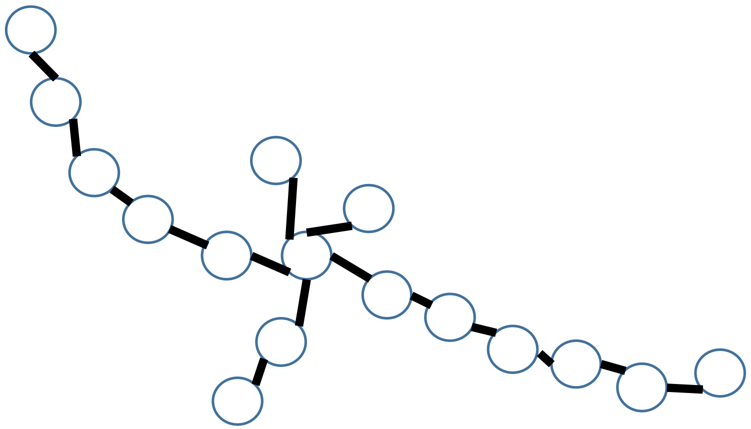
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



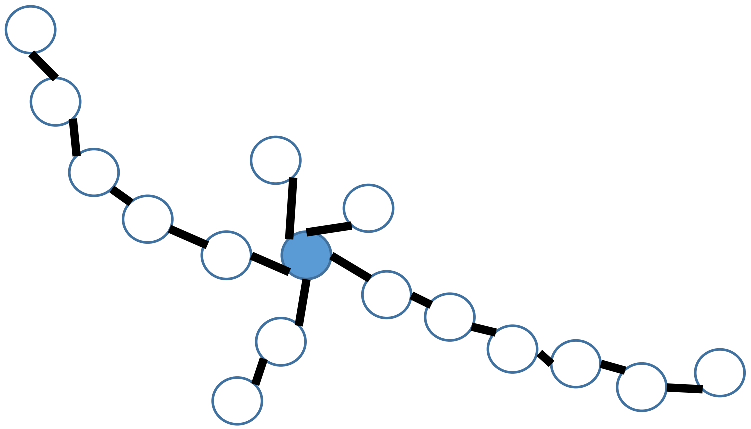
1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



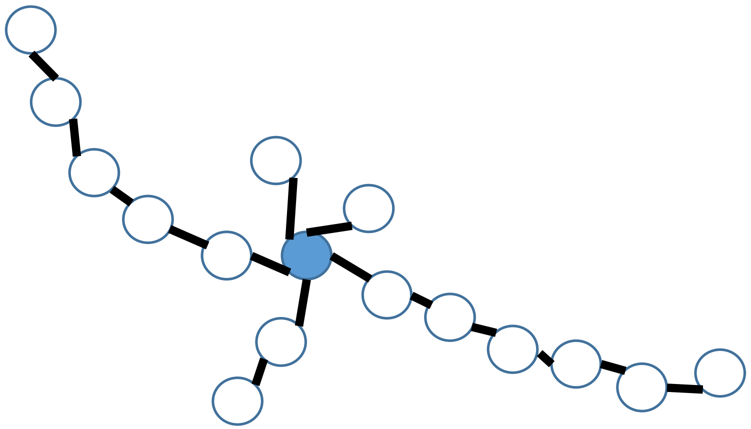
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…

What is the purpose of function dfs?

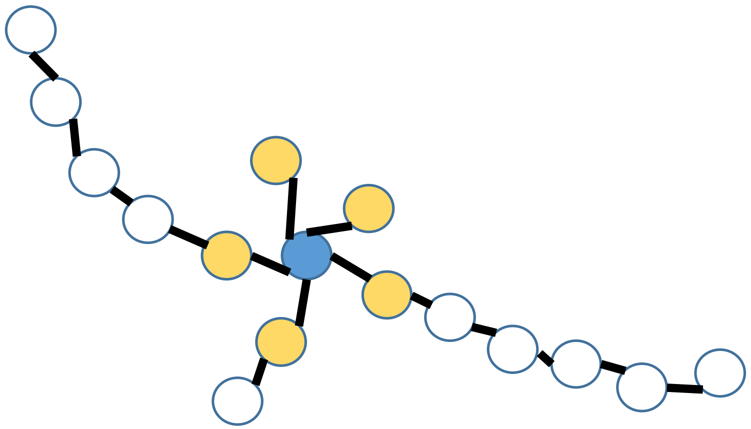
Need to generalize the idea.

Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…

What is the purpose of function dfs?

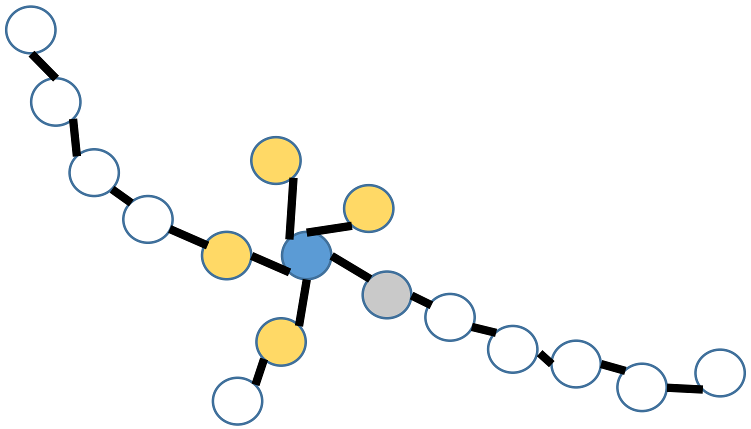
Need to generalize the idea.

Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



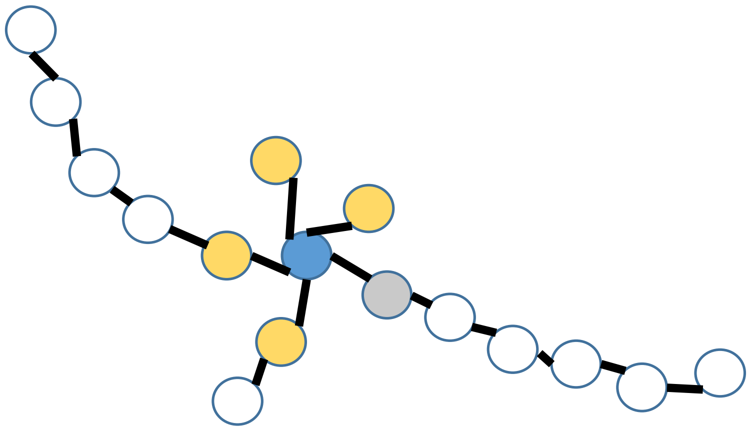
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



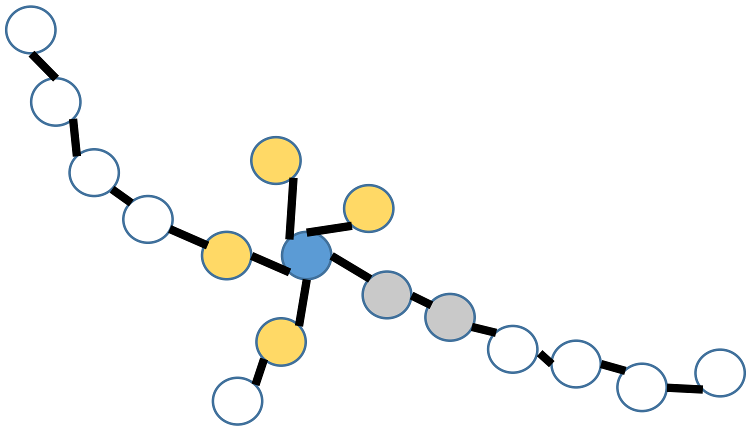
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



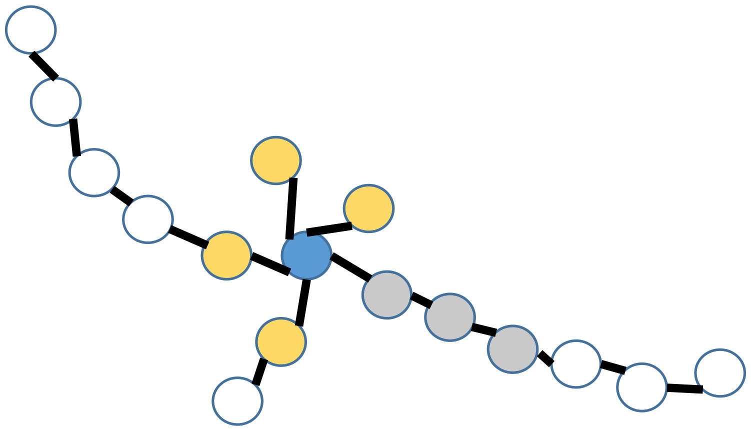
1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



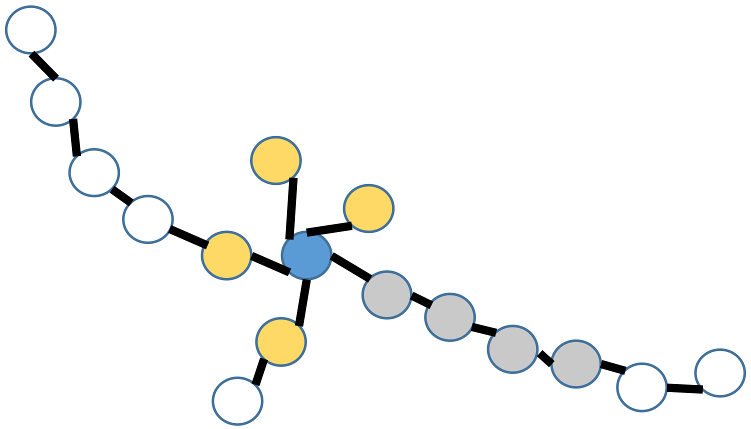
1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…

What is the purpose of function dfs?

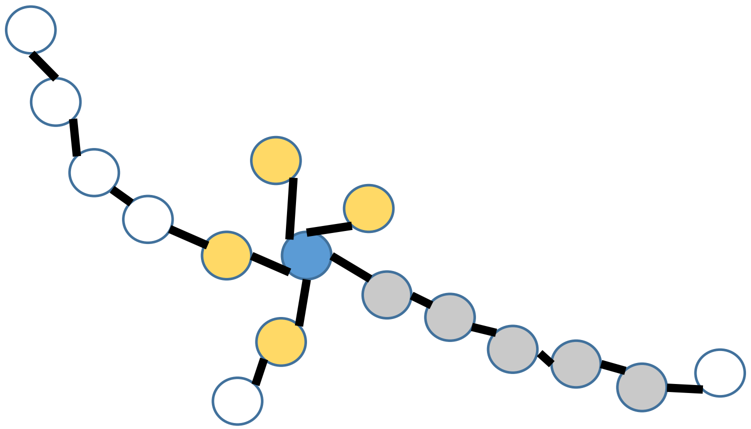
Need to generalize the idea.

Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



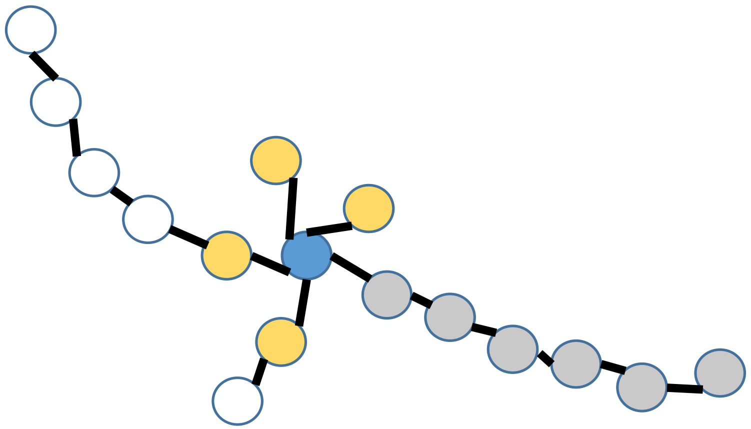
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



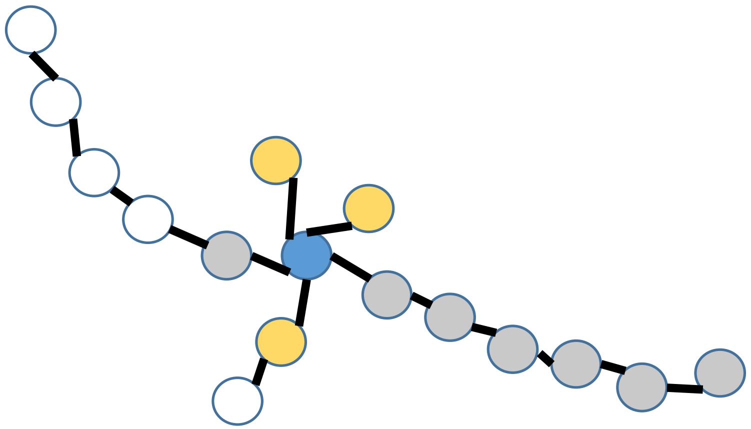
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



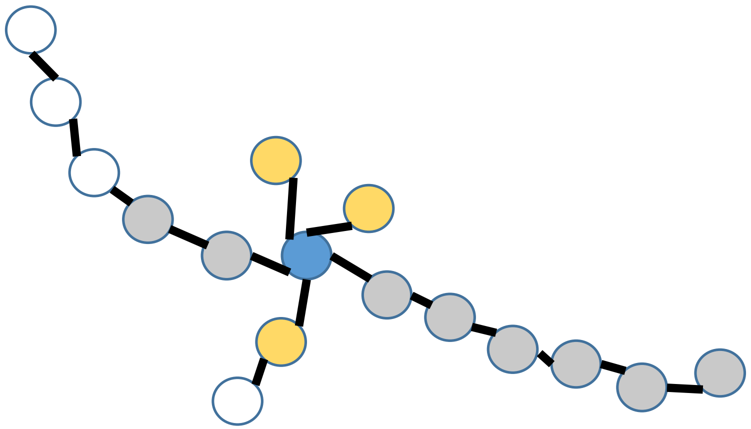
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…

What is the purpose of function dfs?

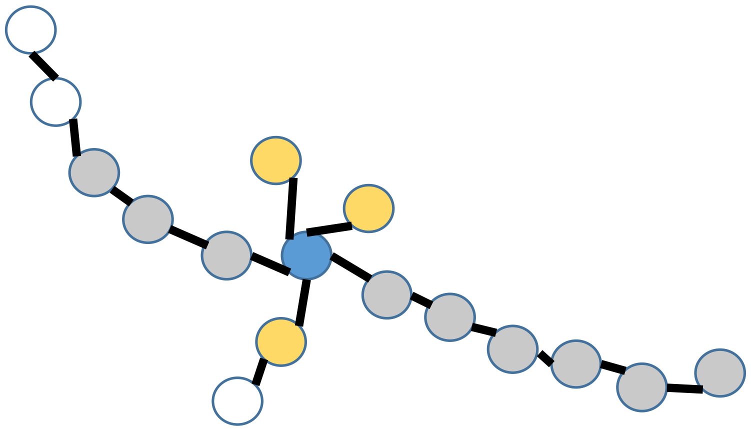
Need to generalize the idea.

Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



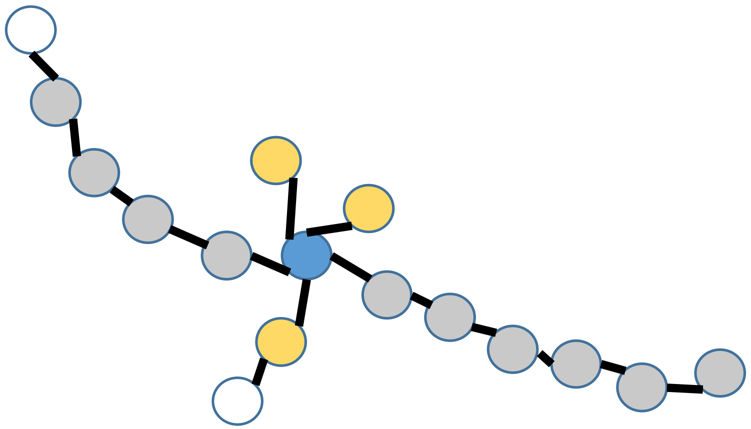
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



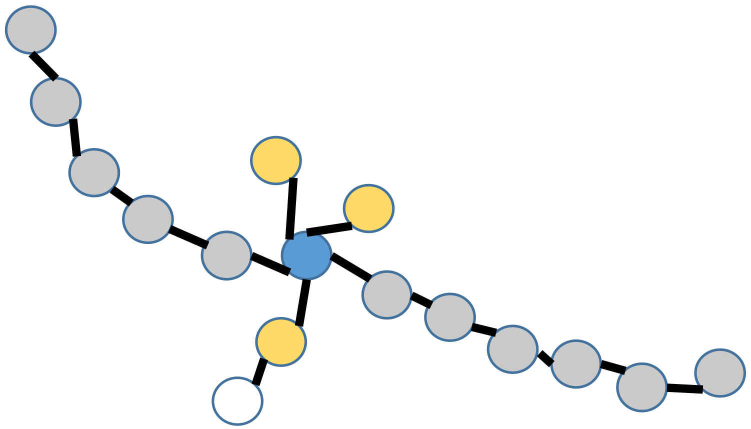
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

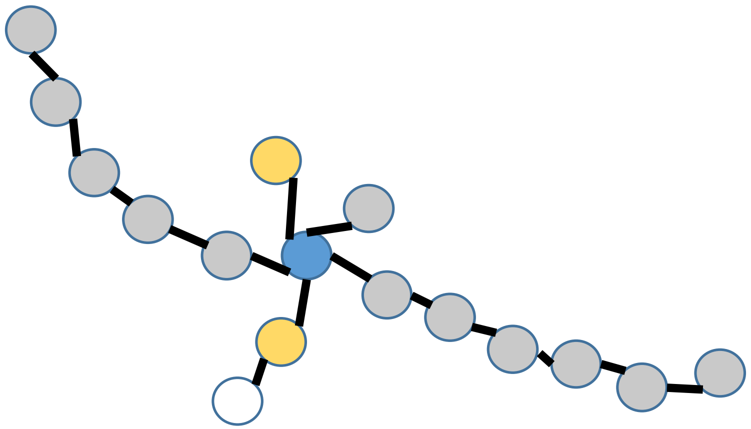
But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Now, you should generalize your findings.

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

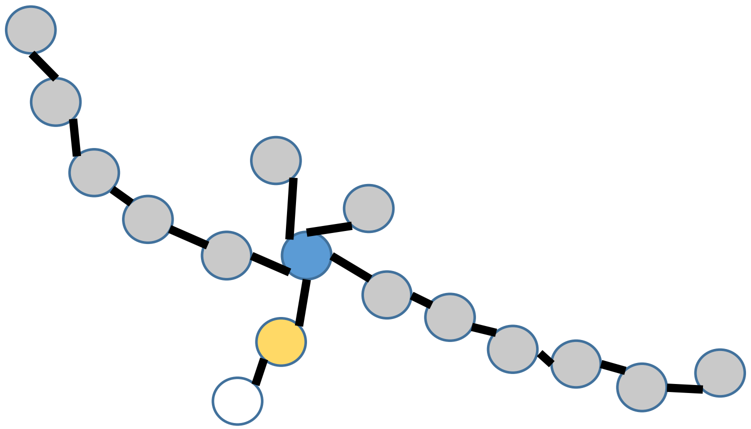
But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Now, you should generalize your findings.

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

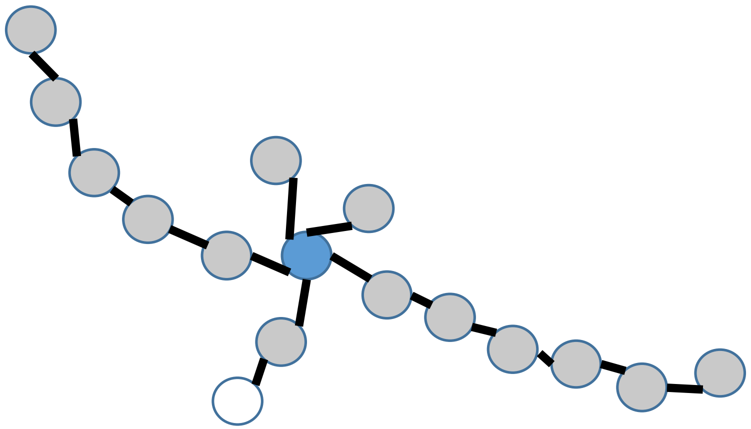
But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Now, you should generalize your findings.

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4.   call foo(w) if w is not visited

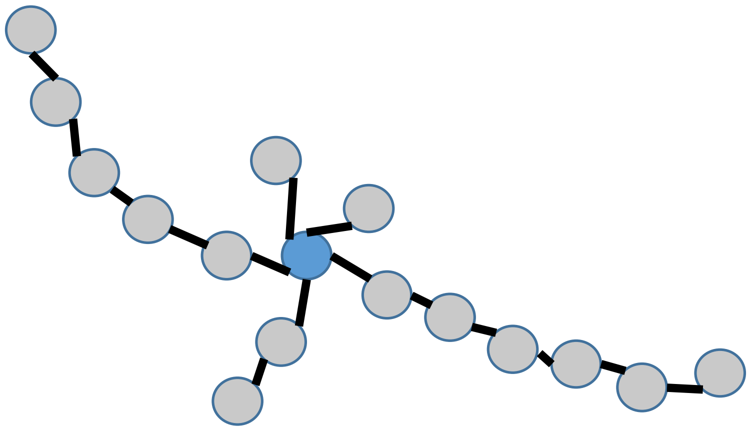
But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Now, you should generalize your findings.

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

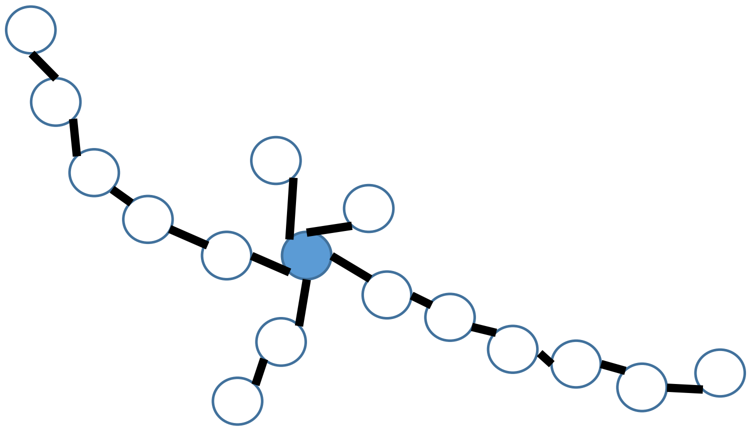
But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Now, you should generalize your findings.

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



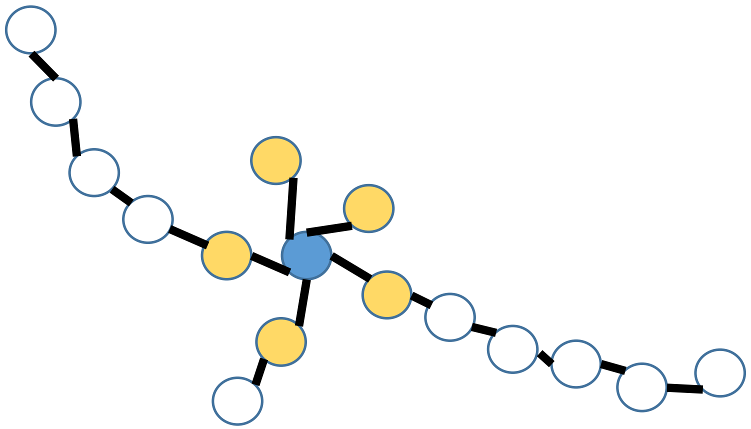
1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4.   call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



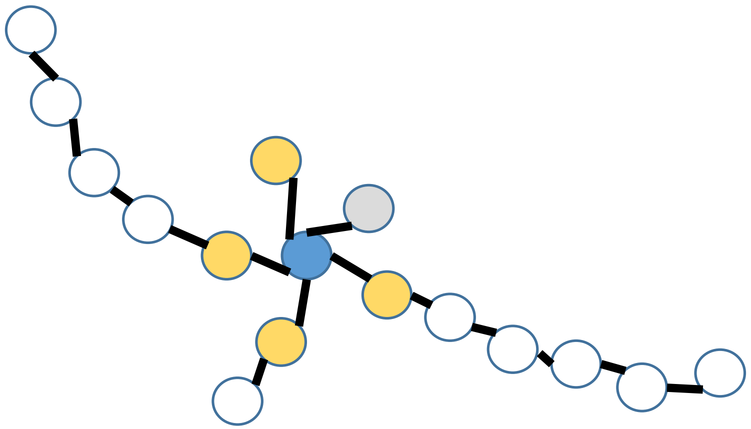
1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait...
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



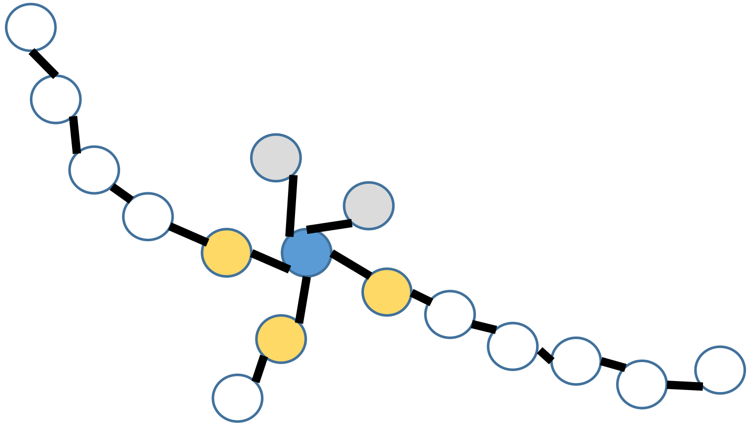
1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…

What is the purpose of function dfs?

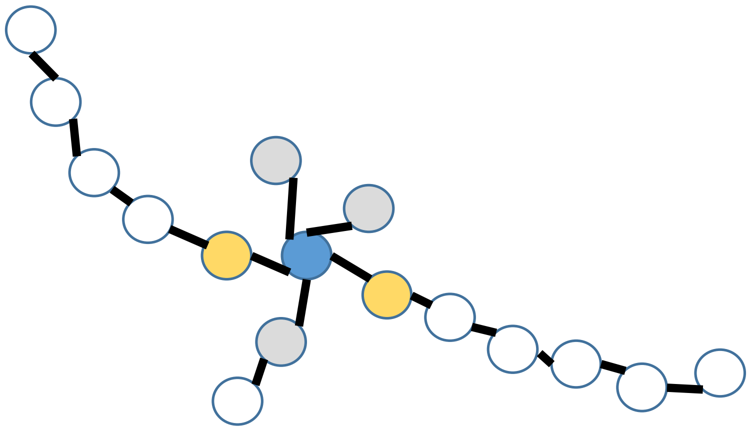
Need to generalize the idea.

Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



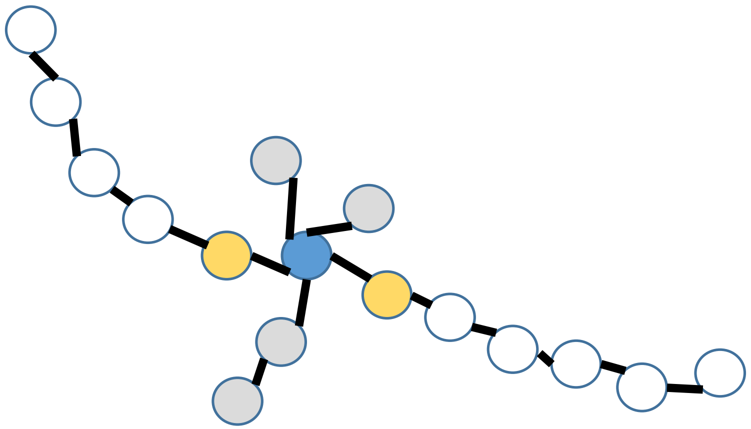
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



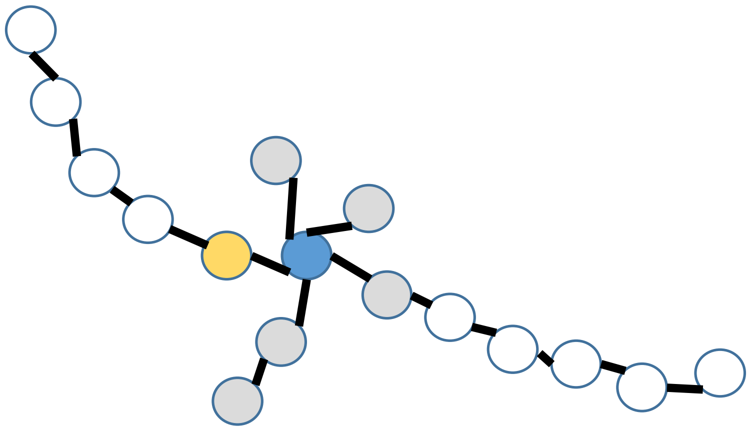
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



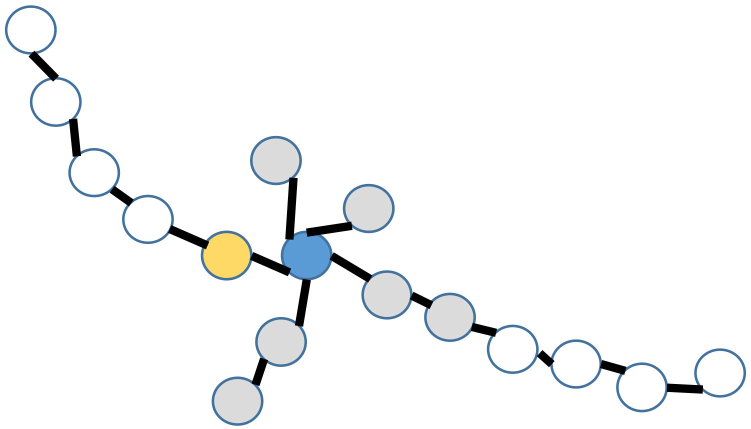
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



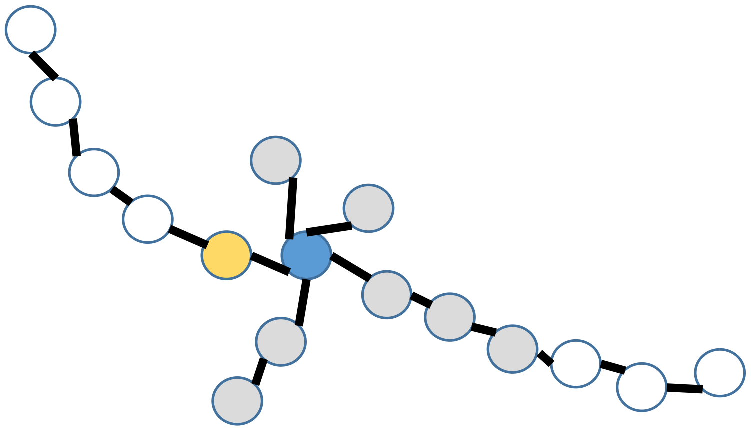
1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



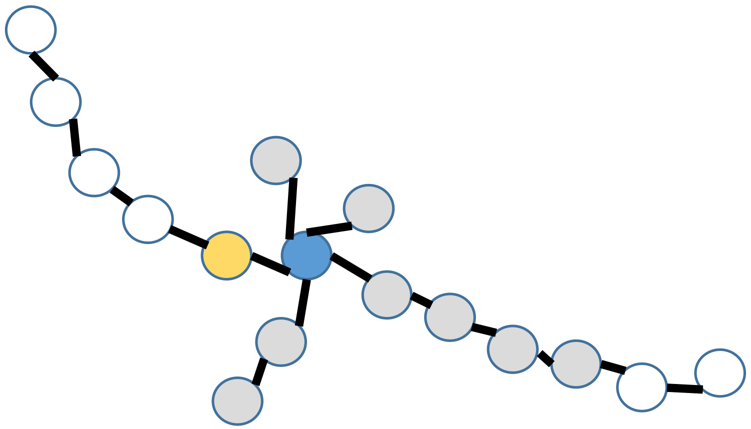
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…

What is the purpose of function dfs?

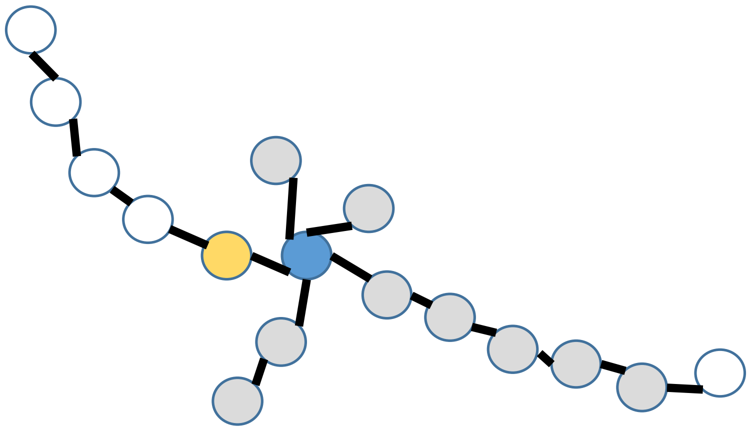
Need to generalize the idea.

Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



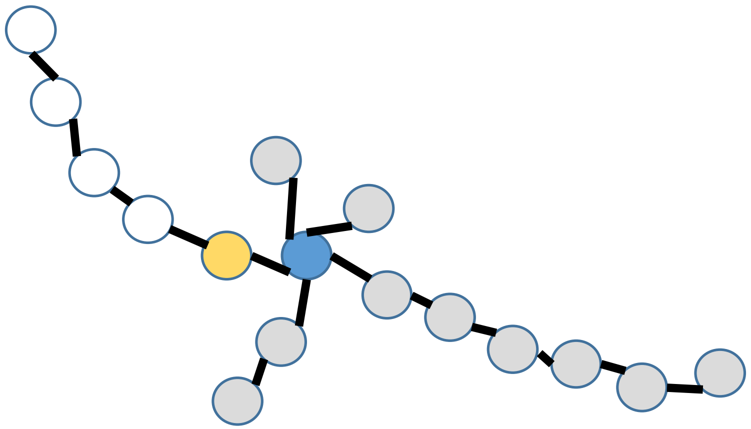
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



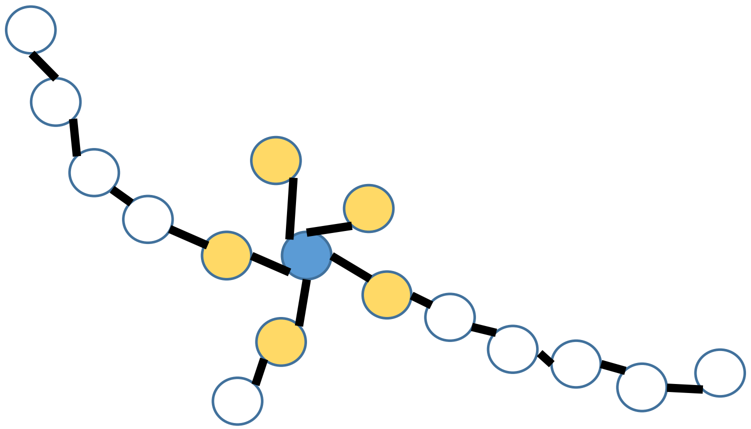
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



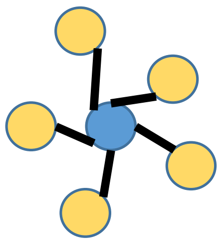
1. What do you do?
2. visit v: display v and mark v as "visited"
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



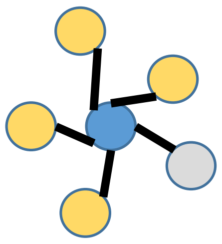
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



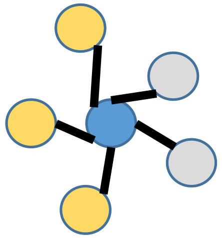
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



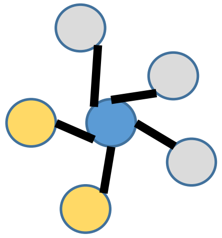
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



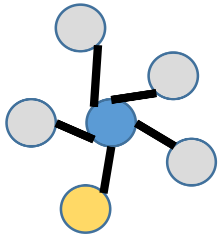
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4.   call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node v = v_j

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



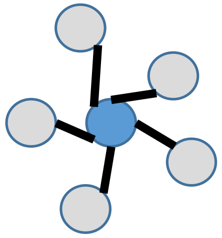
1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…
What is the purpose of function dfs?
Need to generalize the idea.
Work on some GOOD examples to get insights

Depth-First Search

Given a node $v = v_j$

```
dfs(vertex v)
  visit v;
  for each neighbor w of v
    if (w is not visited)
      dfs(w);
```



A bad example does not give you a good insight.

1. What do you do?
2. visit v: display v and mark v as “visited”
3. For each neighbor w of v
4. call foo(w) if w is not visited

But, wait…

What is the purpose of function dfs?

Need to generalize the idea.

Work on some GOOD examples to get insights