Admin

Assignment 8

Lab tomorrow:
- Look at Java code examples of graph representations and graph algorithms
- Work session for assignment

Final
A graph is a set of vertices $V$ and a set of edges $(u,v) \in E$ where $u,v \in V$.
Searching a tree

How can we print out all of the vertices in a tree?
Searching a tree

We could do any of the traversals we saw before: pre-order, in-order, post-order
A flash from the past

Stack:
- LIFO
- Add to the back
- Remove from the back

Queue:
- FIFO
- Add to the back
- Remove from the front

```java
public interface Linear<E> {
    public void add(E item);
    public E remove();
    public E peek();
    public boolean empty();
}
```
Searching a tree

```

// treeBFS

def treeBFS(start):
    q = new Queue()
    q.add(start)
    treeSearch(q)

// treeDFS

def treeDFS(start):
    s = new Stack()
    s.add(start)
    treeSearch(s)

// treeSearch

def treeSearch(toVisit):
    while !toVisit.empty():
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren():
            toVisit.add(c)
```
Tree BFS

\[
\text{treeBFS}(\text{start}) \\
q = \text{new Queue()}
q.\text{add(start)}
\text{treeSearch}(q)
\]

q:
Visited:
Tree BFS

treeBFS(start)
q = new Queue()
q.add(start)
treeSearch(q)

q: A
Visited:
Tree BFS

treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)

What order will the nodes get printed out?
Assume children are traversed left to right.

toVisit-queue: A
printed:
Tree BFS

treeSearch(toVisit)
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-queue: A
printed:
Tree BFS

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-queue:
    printed: A
Tree BFS

treeSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

toVisit-queue:
  printed: A
Tree BFS

treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)

toVisit-queue: B D E
printed: A
Tree BFS

treeSearch( toVisit )
while !toVisit.empty()
  v = toVisit.remove()
  // visit v, e.g., print it out
  for c in v.getChildren()
    toVisit.add(c)

toVisit-queue: B D E
printed: A
Tree BFS

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-queue: D E
printed: A B
Tree BFS

treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)

toListVisit-queue: D E
printed: A B
Tree BFS

treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)

toVisit-queue: D E C F
printed: A B
Tree BFS

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-queue: D E C F
printed: A B
Tree BFS

treeSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

toVisit-queue: E C F
printed: A B D
treeSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)
Tree BFS

treeSearch( toVisit )
   while !toVisit.empty()
      v = toVisit.remove()
      // visit v, e.g., print it out
      for c in v.getChildren()
         toVisit.add(c)

toVisit-queue: E C F
printed: A B D
Tree BFS

treeSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-queue: E C F
printed: A B D
Tree BFS

treeSearch(toVisit)
while !toVisit.empty()
  v = toVisit.remove()
  // visit v, e.g., print it out
  for c in v.getChildren()
    toVisit.add(c)

toVisit-queue: C F
printed: A B D E
Tree BFS

treeSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-queue: C F
printed: A B D E
Tree BFS

treeSearch(toVisit)
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)

toVisit-queue: C F G
printed: A B D E
Tree BFS

treeSearch( toVisit )
   while !toVisit.empty()
      v = toVisit.remove()
      // visit v, e.g., print it out
      for c in v.getChildren()
         toVisit.add(c)

toVisit-queue: C F G
printed: A B D E

How are we exploring the vertices?
Tree BFS

treeSearch( toVisit )

while !toVisit.empty()

    v = toVisit.remove()

    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

Frontier: all vertices a given number of edges from the start/root

toVisit-queue: C F G
printed: A B D E
Tree BFS = Tree breadth first search

treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)


toVisit-queue: C F G
printed: A B D E

Frontier: all vertices a given number of edges from the start/root
Tree BFS

treeSearch( toVisit )

while !toVisit.empty()

    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)


ToVisit-queue: C F G

Printed: A B D E
Tree BFS

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-queue: F G
printed: A B D E C
Tree BFS

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-queue:
    printed: A B D E C F G
```
treeDFS( start )
s = new Stack()
s.add(start)
treeSearch(s)
```
Tree DFS

treeDFS(start)

s = new Stack()
s.add(start)
treeSearch(s)

s: A

printed:
```
treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)
```

What order will the nodes get printed out?
Assume children are traversed left to right.
Tree DFS

treeSearch( toVisit )
while !toVisit.empty()
  v = toVisit.remove()
  // visit v, e.g., print it out
  for c in v.getChildren()
    toVisit.add(c)

toVisit-stack: A
printed:
Tree DFS

treeSearch( toVisit )
while !toVisit.empty()
  v = toVisit.remove()
  // visit v, e.g., print it out
  for c in v.getChildren()
    toVisit.add(c)

**toVisit-stack:**

**printed:** A
Tree DFS

treeSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

visit-stack:
printed: A
Tree DFS

treeSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

for c in v.getChildren()
    toVisit.add(c)
Tree DFS

treeSearch( toVisit )
while !toVisit.empty():
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren():
        toVisit.add(c)

toVisit-stack: B D E
printed: A
Tree DFS

treeSearch( toVisit )

while !toVisit.empty()

    v = toVisit.remove()

    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-stack: B D
printed: A E
Tree DFS

treeSearch( toVisit )
while !toVisit.empty()
  v = toVisit.remove()
  // visit v, e.g., print it out
  for c in v.getChildren()
    toVisit.add(c)

toVisit-stack: B D
printed: A E
treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)
Tree DFS

treeSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

toVisit-stack: B D G
printed: A E
Tree DFS

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-stack: B D
printed: A E G
Tree DFS

treeSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

toVisit-stack: B D
printed: A E G
Tree DFS

treeSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-stack: B D
printed: A E G
Tree DFS

treeSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-stack: B
printed: A E G D
treeSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

How are we exploring the vertices?

toVisit-stack: B
printed: A E G D
Tree DFS

```python
treeSearch(toVisit)
while !toVisit.empty():
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren():
        toVisit.add(c)
```

toVisit-stack: B
printed: A E G D

Frontier: go as far down one branch as possible, working right to left
Tree DFS

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

toVisit-stack:
printed: A E G D B
Tree DFS

treeSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

toVisit-stack:
  printed: A E G D B
Tree DFS

treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)

visited-stack: C F
printed: A E G D B
Tree DFS

treeSearch( toVisit )

while !toVisit.empty()

  v = toVisit.remove()

  // visit v, e.g., print it out
  for c in v.getChildren()
    toVisit.add(c)

**toVisit-stack:** C F

**printed:** A E G D B
Tree DFS

treeSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

toVisit-stack:
  printed: A E G D B F C
Run-time of graph algorithms

A graph is a set of vertices $V$ and a set of edges $(u,v) \in E$ where $u,v \in V$

When we analyze graph algorithms, the run-time often includes both the number of vertices AND the number of edges:

- $|V| = \text{number of vertices}$
- $|E| = \text{number of edges}$

Sometimes, in big-O notation, we’ll use just $V$ and $E$ to represent these to simplify notation
treeSearch run-time

treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)

What is the big-O run-time of teeSearch?

Assume all of the stack/queue operations are constant.

How many times do we visit each vertex?

How many times do we traverse each edge (vis the for loop)?
treeSearch run-time

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

How many times do we visit each vertex? Exactly once

How many times do we traverse each edge (vis the for loop)? Exactly once

What is the big-O run-time of treeSearch? $O(|V| + |E|)$. Linear algorithm.
search( v )

// visit v, e.g., print it out
for c in v.getChildren()
    search(c)

What algorithm is this?

What order will the vertices get printed out?
Assume children are traversed left to right.
What algorithm is this?

```
search(v)
    // visit v, e.g., print it out
    for c in v.getChildren()
        search(c)
```

Visited: A
What algorithm is this?

```python
search( v )
    // visit v, e.g., print it out
    for c in v.getChildren()
        search(c)
```

Visited: A B
What algorithm is this?

```
search(v)
  // visit v, e.g., print it out
  for c in v.getChildren()
    search(c)
```

Visited: A B C
search(v)
    // visit v, e.g., print it out
    for c in v.getChildren()
        search(c)

Visited: A B C

Now where?
What algorithm is this?

```python
search(v)
// visit v, e.g., print it out
for c in v.getChildren()
    search(c)
```

Visited: A B C F
What algorithm is this?

```python
search(v)
  // visit v, e.g., print it out
  for c in v.getChildren()
    search(c)
```

Visited: A B C F D
What algorithm is this?

```python
search(v)
// visit v, e.g., print it out
for c in v.getChildren()
    search(c)
```

Visited: A B C F D
What algorithm is this?

search( v )
// visit v, e.g., print it out
for c in v.getChildren()
  search(c)

Visited: A B C F D

Depth first search!
What algorithm is this?

search(v)
// visit v, e.g., print it out
for c in v.getChildren()
    search(c)

Visited: A B C F D

Any difference between this version and the stack version?
What algorithm is this?

```python
search(v)
    // visit v, e.g., print it out
    for c in v.getChildren()
        search(c)
```

Visited: A B C F D

Any difference between this version and the stack version?

Traverses in the other direction (left to right in this case).
What algorithm is this?

```python
def search(v):
    # visit v, e.g., print it out
    for c in v.getChildren():
        search(c)
```

Visited: A B C F D

treeDFS used a stack.
Is there a stack here?
What algorithm is this?

```python
search(v)
    // visit v, e.g., print it out
    for c in v.getChildren()
        search(c)
```

***Visited:*** A B C F D

**treeDFS used a stack.**
**Is there a stack here?**

The run-time stack keeping track of recursive calls!
What algorithm is this?

```python
def search(v):
    # visit v, e.g., print it out
    for c in v.getChildren():
        search(c)
```

Visited: A B C F D
What algorithm is this?

```python
def search(v):
    # visit v, e.g., print it out
    for c in v.getChildren():
        search(c)
```

Visited: A B C F D E G
DFS versions

treeDFS(start)
  s = new Stack()
  s.add(start)
  treeSearch(s)

treeSearch(toVisit)
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

treeRecursiveDFS(v)
  // visit v, e.g., print it out
  for c in v.getChildren()
    treeRecursiveDFS(c)
treeSearch on graphs

treeSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
        toVisit.add(c)

What would happen if we ran treeSearch on this graph?

Won’t ever end!
treeSearch on graphs

```python
treeSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        // visit v, e.g., print it out
        for c in v.getChildren()
            toVisit.add(c)
```

How can we fix this?

Keep track of the vertices that we’ve visited
Searching on graphs

treeSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    // visit v, e.g., print it out
    for c in v.getChildren()
      toVisit.add(c)

graphSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
      visited[v] = true
      for c in v.getAdjacent()
        if !visited[c]
          toVisit.add(c)
Searching on graphs

graphBFS( start )
q = new Queue()
q.add(start)
treeSearch(q)

graphDFS( start )
s = new Stack()
s.add(start)
treeSearch(s)

graphSearch( toVisit )
while !toVisit.empty()
v = toVisit.remove()
if !visited[v]
visited[v] = true
for c in v.getAdjacent()
if !visited[c]
toVisit.add(c)
BFS

toVisit-queue: A
visited:

```java
graphBFS(start)
q = new Queue()
q.add(start)
treeSearch(q)
```
BFS

graphSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
      visited[v] = true
      for c in v.getAdjacent()
        if !visited[c]
          toVisit.add(c)

toVisit-queue: A
visited:
What order will the nodes get printed out?
Assume edges are traversed alphabetically.
graphSearch(toVisit)

while !toVisit.empty()

    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)
graphSearch( toVisit )
  while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
      visited[v] = true
      for c in v.getAdjacent()
        if !visited[c]
          toVisit.add(c)
BFS

def graphSearch(toVisit):
    while !toVisit.empty():
        v = toVisit.remove()
        if !visited[v]:
            visited[v] = true
            for c in v.getAdjacent():
                if !visited[c]:
                    toVisit.add(c)

toVisit-queue:
visited: A
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-queue: B D E
visited: A
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-queue: B D E
visited: A
BFS

graphSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)

toVisit-queue: D E
visited: A B
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-queue: D E
visited: A B
graphSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)

visited: A B

toVisit-queue: D E C E F

Notice that we do add E again to toVisit since we haven’t visited it yet
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-queue: D E C E F
visited: A B
BFS

```python
def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if visited[v] == False:
            visited[v] = True
            for c in v.getAdjacent():
                if visited[c] == False:
                    toVisit.add(c)
```

toVisit-queue: E C E F
visited: A B D
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        visited[v] = True
        for c in v.getAdjacent():
            if not visited[c]:
                toVisit.add(c)

toVisit-queue: E C E F
visited: A B D
BFS

toVisit-queue: E C E F E
visited: A B D

We add E again to toVisit since we still haven’t visited it yet
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-queue: E C E F E
visited: A B D
graphSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)
BFS

```python
def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if !visited[v]:
            visited[v] = true
            for c in v.getAdjacent():
                if !visited[c]:
                    toVisit.add(c)
```

toVisit-queue: E C E F E
visited: A B D E
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-queue: E C E F E
visited: A B D E

No adjacent vertices that haven’t been visited
BFS

graphSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)

toVisit-queue: E C E F E
visited: A B D E

Frontier: all vertices a given number of edges from the start/root
BFS

toVisit-queue: E C E F E
visited: A B D E
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

E has already been visited

toVisit-queue: C E F E
visited: A B D E
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-queue: C E F E
visited: A B D E
BFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

visited = A B D E C

toVisit-queue: E F E
BFS

```python
graphSearch(toVisit)
while !toVisit.empty():
    v = toVisit.remove()
    if !visited[v]:
        visited[v] = true
        for c in v.getAdjacent():
            if !visited[c]:
                toVisit.add(c)
```

**toVisit-queue:** E F E

**visited:** A B D E C
### BFS

**graphSearch**( `toVisit` )

while `!toVisit.empty()`

v = `toVisit.remove()`

if `!visited[v]`

`visited[v] = true`

for c in v.getAdjacent()

if `!visited[c]`

`toVisit.add(c)`

**toVisit-queue:** F E

**visited:** A B D E C

E has already been visited
BFS

```
grapSearch( toVisit )

   while !toVisit.empty()

      v = toVisit.remove()

      if !visited[v]
         visited[v] = true

         for c in v.getAdjacent()
            if !visited[c]
               toVisit.add(c)
```
BFS

graphSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)

toVisit-queue:
    visited: A B D E C F G
DFS

def graphDFS(start):
    s = new Stack()
    s.add(start)
    treeSearch(s)

toVisit-stack: A
visited:
What order will the nodes get printed out? Assume edges are traversed alphabetically.
DFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-stack: A
visited:

A -- E -- C
    |   |
    v   
D -- E -- C
    |
    F

B -- D -- E

DFS

graphSearch( toVisit )

while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)

**toVisit-stack:** B D E

**visited:** A
DFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if !visited[v]:
            visited[v] = true
            for c in v.getAdjacent():
                if !visited[c]:
                    toVisit.add(c)

toVisit-stack: B D E
visited: A
DFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-stack: B D C D F
visited: A E
DFS

```python
def graphSearch(toVisit):
    while not toVisit.empty:
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)
```

toVisit-stack: B D C D F
visited: A E
DFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-stack: B D C D G
visited: A E F
DFS

```java
graphSearch(toVisit)
while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)
```

**toVisit-stack:** B D C D G  
**visited:** A E F
graphSearch(toVisit)

while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)
DFS

graphSearch( toVisit )
while !toVisit.empty()
  v = toVisit.remove()
  if !visited[v]
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        toVisit.add(c)

toVisit-stack: B D C D
visited: A E F G

Frontier: Go as far down one path as possible
DFS

```python
def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)
```

**toVisit-stack:** B D C D

**visited:** A E F G
DFS

def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)

toVisit-stack: B D C B
visited: A E F G D
DFS

graphSearch( toVisit )
while !toVisit.empty()
  v = toVisit.remove()
  if !visited[v]
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        toVisit.add(c)

toVisit-stack: B D C B
visited: A E F G D
DFS

```python
def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)
```

toVisit-stack: B D C
visited: A E F G D B
DFS

```python
def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)
```

toVisit-stack: B D C
visited: A E F G D B
**DFS**

```python
def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)
```

**toVisit-stack:** B D

**visited:** A E F G D B C
DFS

```
grapSearch( toVisit )
while !toVisit.empty()
    v = toVisit.remove()
    if !visited[v]
        visited[v] = true
        for c in v.getAdjacent()
            if !visited[c]
                toVisit.add(c)
```

toVisit-stack: B D
visited: A E F G D B C
DFS

```python
def graphSearch(toVisit):
    while not toVisit.empty():
        v = toVisit.remove()
        if not visited[v]:
            visited[v] = True
            for c in v.getAdjacent():
                if not visited[c]:
                    toVisit.add(c)
```

**toVisit-stack:**

visited: A E F G D B C
**graphSearch run-time**

```python
graphSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        if !visited[v]
            visited[v] = true
            for c in v.getAdjacent()
                if !visited[c]
                    toVisit.add(c)
```

What is the big-O run-time of graphSearch?

Assume all of the stack/queue operations are constant.

How many times do we visit each vertex?

How many times do we traverse each edge (vis the for loop)?
graphSearch run-time

```python
def graphSearch( toVisit )
    while !toVisit.empty()
        v = toVisit.remove()
        if !visited[v]
            visited[v] = true
            for c in v.getAdjacent()
                if !visited[c]
                    toVisit.add(c)
```

How many times do we visit each vertex? Exactly once

How many times do we traverse each edge (vis the for loop)? Exactly once

What is the big-O run-time of teeSearch? $O(|V| + |E|)$. Linear algorithm.

Nothing changes from treeSearch!