GRAPHS: SEARCH

David Kauchak
CS 62 – Fall 2026

1

Admin

Assignment 9: Text Generator

Final project

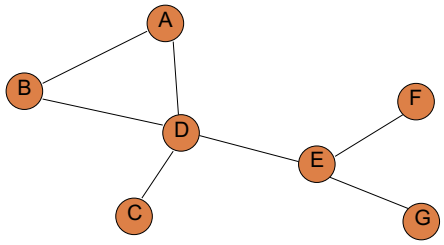
Lab tomorrow:

- Quiz
- Project brainstorming session
- Try and form groups already and ideally attend the same lab

2

Graphs

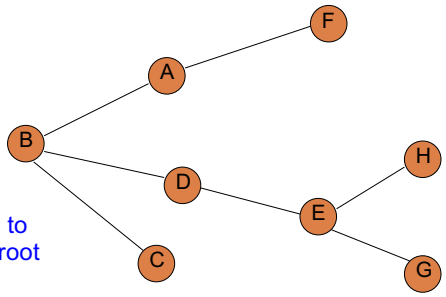
A graph is a set of vertices V and a set of edges (u,v) ∈ E where u,v ∈ V



3

Different types of graphs

Tree – connected, undirected graph without any cycles



also need to specify a root

4

Searching a tree

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

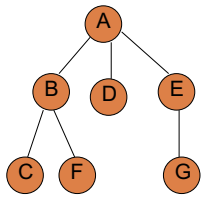
```
treeDFS( start )
    s = new Stack()
    s.add(start)
    treeSearch(s)
```

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

5

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)
```



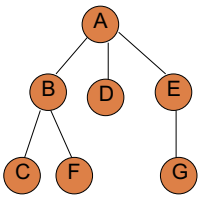
q:

Visited:

6

Tree BFS – 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)
```



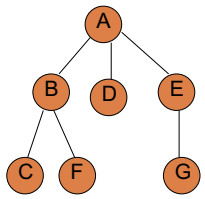
toVisit-queue:

printed: A B D E C F G

7

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)
```



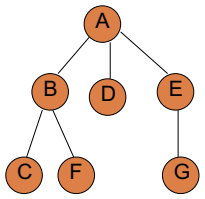
toVisit-stack:

printed:

8

Tree DFS – Depth 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-stack:

printed: A E G D B F C

9

Run-time of graph algorithms

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

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

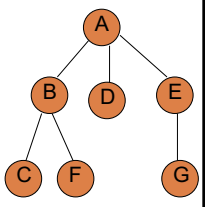
- |V| = number of vertices
- |E| = number of edges

Sometimes, in big-O notation, we'll use just V and E to represent these to simplify notation

10

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 treeSearch?

Assume all of the stack/queue operations are constant.

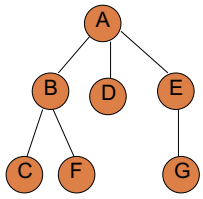
How many times do we visit each vertex?

How many times do we traverse each edge (via the for loop)?

11

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

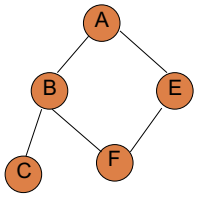
How many times do we traverse each edge (via the for loop)? Exactly once

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

12

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)
```



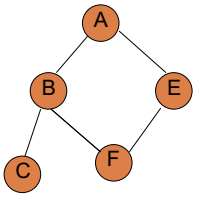
What would happen if we ran treeSearch on this graph?

Won't end!

13

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)
```



How can we fix this?

Keep track of the vertices that we've visited

14

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

15

Searching on graphs

```
graphBFS( start )
    q = new Queue()
    q.add(start)
    graphSearch(q)
```

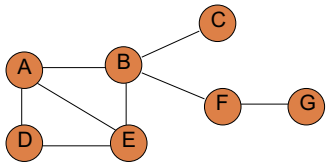
```
graphDFS( start )
    s = new Stack()
    s.add(start)
    graphSearch(s)
```

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

16

BFS

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



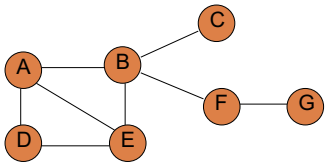
toVisit-queue: A

visited:

17

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

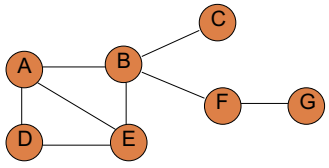
visited:

What order will the nodes get printed out?
Assume edges are traversed alphabetically.

18

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)
```



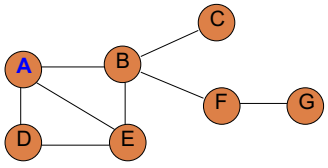
toVisit-queue: A

visited:

19

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)
```



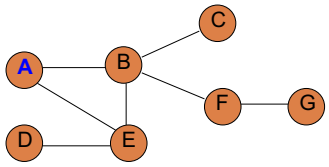
toVisit-queue:

visited: A

20

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)
```



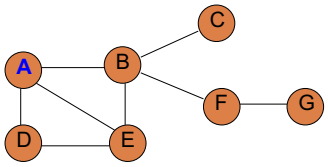
toVisit-queue:

visited: A

21

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)
```



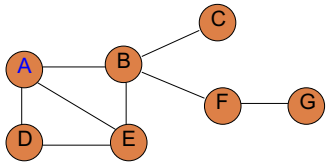
toVisit-queue: B D E

visited: A

22

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)
```



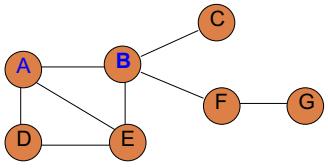
toVisit-queue: B D E

visited: A

23

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)
```



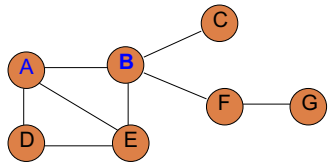
toVisit-queue: D E

visited: A B

24

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)
```



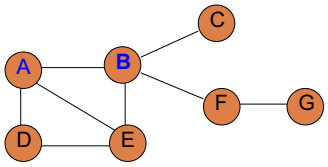
toVisit-queue: D E

visited: A B

25

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 C E F

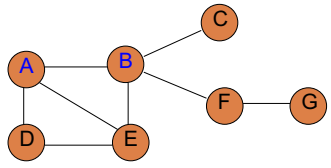
visited: A B

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

26

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)
```



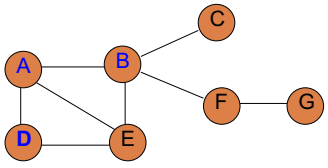
toVisit-queue: D E C E F

visited: A B

27

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)
```



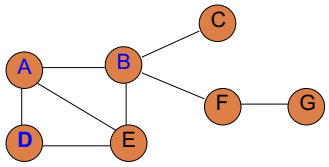
toVisit-queue: E C E F

visited: A B D

28

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)
```



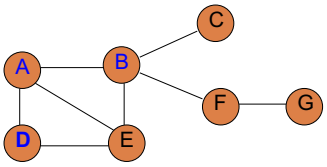
toVisit-queue: E C E F

visited: A B D

29

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

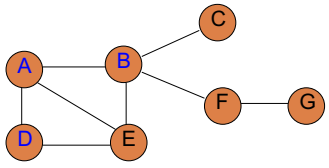
visited: A B D

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

30

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)
```



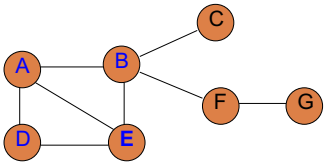
toVisit-queue: E C E F E

visited: A B D

31

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)
```



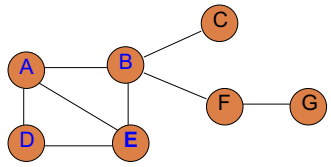
toVisit-queue: E C E F E

visited: A B D E

32

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)
```



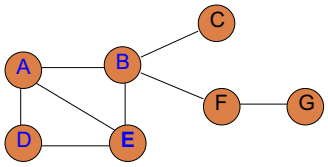
toVisit-queue: E C E F E

visited: A B D E

33

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

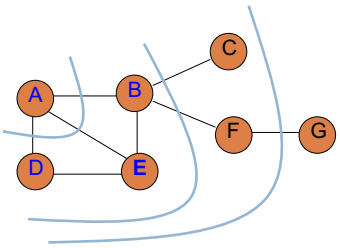
visited: A B D E

No adjacent vertices that haven't been visited

34

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

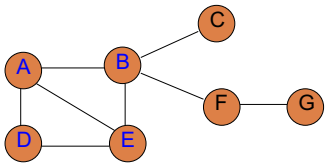
visited: A B D E

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

35

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)
```



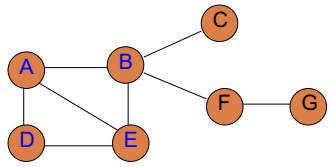
toVisit-queue: E C E F E

visited: A B D E

36

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: C E F E

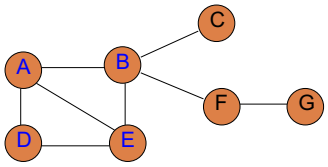
visited: A B D E

E has already been visited

37

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)
```



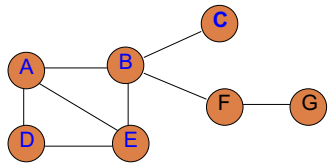
toVisit-queue: C E F E

visited: A B D E

38

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)
```



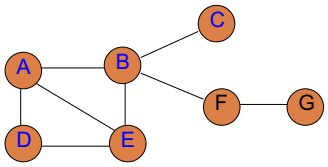
toVisit-queue: E F E

visited: A B D E C

39

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)
```



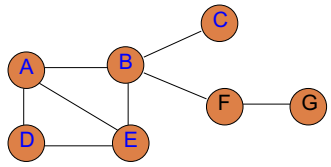
toVisit-queue: E F E

visited: A B D E C

40

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

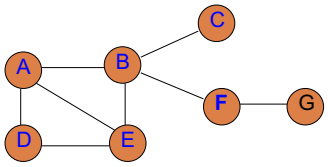
E has already been visited

visited: A B D E C

41

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)
```



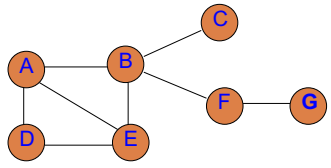
toVisit-queue: E G

visited: A B D E C F

42

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)
```



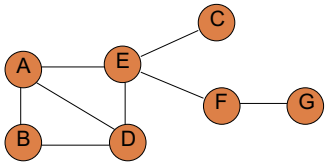
toVisit-queue:

visited: A B D E C F G

43

DFS

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



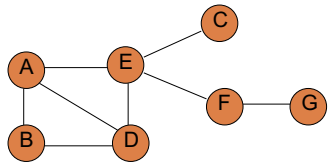
toVisit-stack: A

visited:

44

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: A

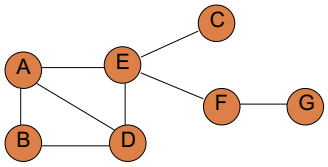
visited:

What order will the nodes get printed out? Assume edges are traversed alphabetically.

45

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)
```



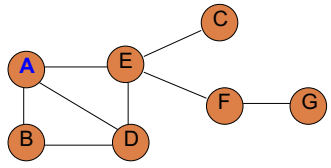
toVisit-stack: A

visited:

46

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)
```



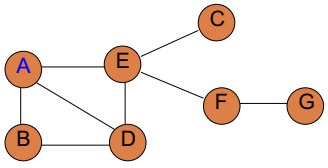
toVisit-stack: B D E

visited: A

47

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)
```



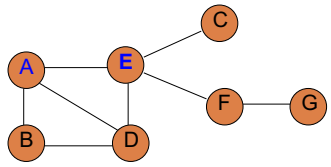
toVisit-stack: B D E

visited: A

48

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)
```



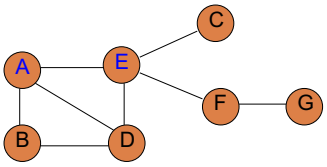
toVisit-stack: B D C D F

visited: A E

49

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)
```



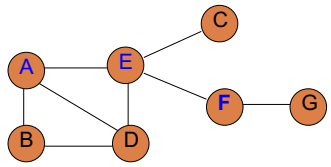
toVisit-stack: B D C D F

visited: A E

50

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)
```



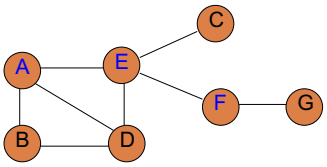
toVisit-stack: B D C D G

visited: A E F

51

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)
```



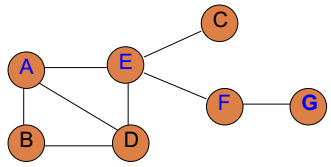
toVisit-stack: B D C D G

visited: A E F

52

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)
```



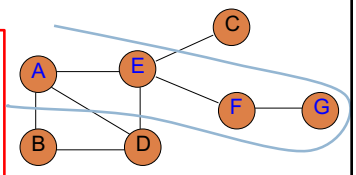
toVisit-stack: B D C D

visited: A E F G

53

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

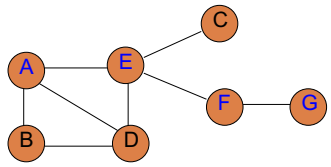
visited: A E F G

Frontier: Go as far down one path as possible

54

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)
```



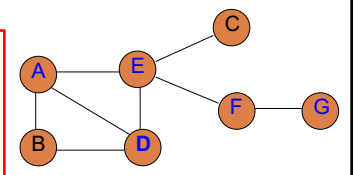
toVisit-stack: B D C D

visited: A E F G

55

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)
```



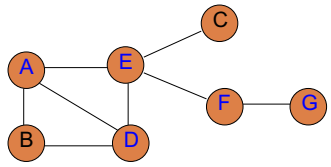
toVisit-stack: B D C B

visited: A E F G D

56

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)
```



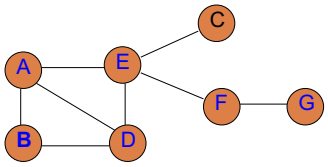
toVisit-stack: B D C B

visited: A E F G D

57

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)
```



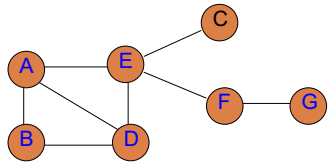
toVisit-stack: B D C

visited: A E F G D B

58

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)
```



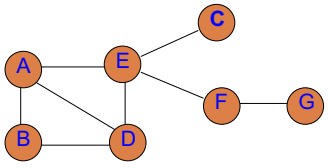
toVisit-stack: B D C

visited: A E F G D B

59

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)
```



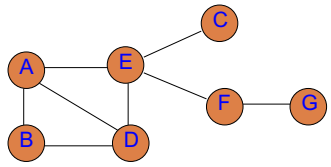
toVisit-stack: B D

visited: A E F G D B C

60

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)
```



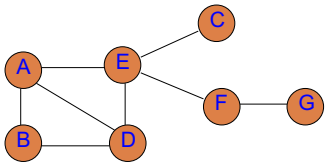
toVisit-stack: B D

visited: A E F G D B C

61

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:

visited: A E F G D B C

62

Recursive DFS

```
dfs( start )
  for each v, visited[v] = false
  dfs-visit( start, visited )

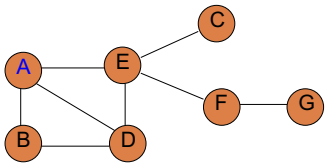
dfs-visit( v, visited )
  if !visited[v]
    // visit v
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        dfs-visit(c)
```

How does the order differ?

63

DFS

```
dfs-visit( v, visited )
  if !visited[v]
    // visit v
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        dfs-visit(c)
```

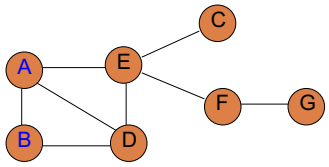


visited: A

64

DFS

```
dfs-visit( v, visited )
  if !visited[v]
    // visit v
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        dfs-visit(c)
```

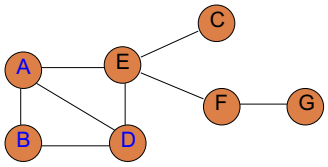


visited: A B

65

DFS

```
dfs-visit( v, visited )
  if !visited[v]
    // visit v
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        dfs-visit(c)
```

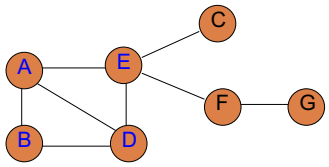


visited: A B D

66

DFS

```
dfs-visit( v, visited )
  if !visited[v]
    // visit v
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        dfs-visit(c)
```

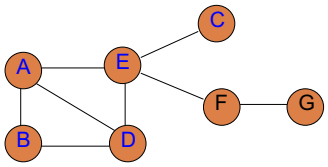


visited: A B D E

67

DFS

```
dfs-visit( v, visited )
  if !visited[v]
    // visit v
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        dfs-visit(c)
```

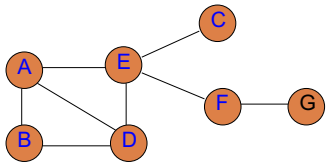


visited: A B D E C

68

DFS

```
dfs-visit( v, visited )
  if !visited[v]
    // visit v
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        dfs-visit(c)
```

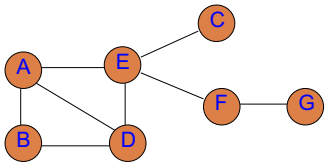


visited: A B D E C F

69

DFS

```
dfs-visit( v, visited )
  if !visited[v]
    // visit v
    visited[v] = true
    for c in v.getAdjacent()
      if !visited[c]
        dfs-visit(c)
```



visited: A B D E C F G

70

graphSearch run-time

```
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?

71

graphSearch run-time

```
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

72

graphSearch run-time

```
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 cost to traverse all of the edges adjacent to a vertext?

73

graphSearch run-time

```
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 cost to traverse all of the edges adjacent to a vertext?

Depends on the graph representation!

74

graphSearch run-time

```
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 cost to traverse all of the edges adjacent to a vertext?

Depends on the graph representation!

Adjacency matrix?

Adjacency list?

75

graphSearch run-time

```
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 cost to traverse all of the edges adjacent to a vertext?

Depends on the graph representation!

Adjacency matrix? V – we have to traverse the whole row

Adjacency list? Like with trees, we traverse each edge exactly once

76

graphSearch run-time

```
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)
```

Overall runtime: Depends on the graph representation!

Adjacency matrix? $O(V^2)$

Adjacency list? O(V+E)

77

graphSearch run-time

```
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 (via the for loop)? Exactly once

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

Nothing changes from treeSearch!
(assuming we're using an adjacency list representation!)

78

Other graph algorithms

Is the graph connected?

Is the graph strongly connected?

Does the graph have a cycle (undirected)?

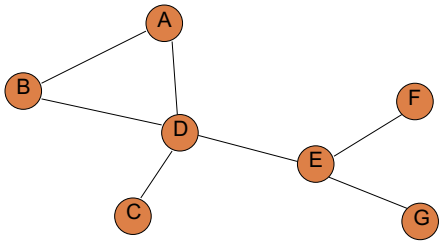
Does the graph have a cycle (directed)?

79

Connectedness

Connected – every pair of vertices is connected by a path

Algorithm?



80

Connectedness

Connected – every pair of vertices is connected by a path

Pick any starting vertex u
Run DFS/BFS from u

For each vertex v:
    if !visited[v]
        return false

return true

Why does this work?

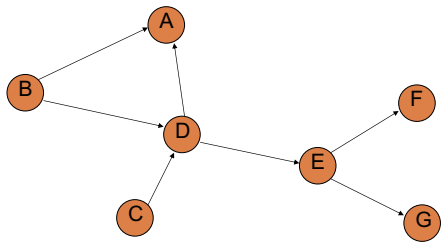
If we can get from u to every vertex then we know a path exists between all vertices.

Path from a to b: a – u – b

81

Strongly connected

Strongly connected (directed graphs) –
Every two vertices are reachable by a path



82

Strongly connected

Strongly connected (directed graphs) –
Every two vertices are reachable by a path

Pick any starting vertex u
Run DFS/BFS from u

For each vertex v:
    if !visited[v]
        return false

return true

Does this work?

83

Strongly connected

Strongly connected (directed graphs) –
Every two vertices are reachable by a path

Pick any starting vertex u
Run DFS/BFS from u

For each vertex v:
    if !visited[v]
        return false

return true

Does this work?

No!

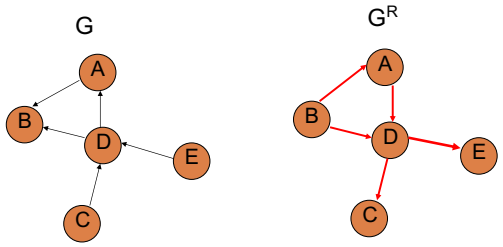
Path from a to b: a – u – b

We know we can get from u to b, but we don't know that we can get from a to s (directed graph!)

84

Reverse of a graph

Given a graph G, we can calculate the reverse of a graph G^R by reversing the direction of all the edges



85

Strongly connected

Strongly-Connected(G)
- Run BFS/DFS from some node u
- If not all nodes are visited:
  return false
- Create graph G^R
- Run BFS/DFS on G^R from node u
- If not all nodes are visited:
  return false
- return true

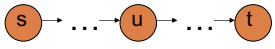
86

Is it correct?

What do we know after the first search?
- Starting at u, we can reach every node

What do we know after the second search (reverse graph)?
- All nodes can reach u. Why?
- We can get from u to every node in G^R, therefore, if we reverse the edges (i.e. G), then we have a path from every node to u

Which means that any node can reach any other node! Given any two nodes s and t we can create a path through u



87

Run-times?

Connectedness

Pick any starting vertex u
Run DFS/BFS from u

What is the run-time?

For each vertex v:
  if !visited[v]
    return false

return true

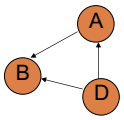
88

Detecting cycles

Undirected graph

- BFS or DFS. If we reach a node we've seen already, then we've found a cycle

Directed graph



have to be careful

89

Detecting cycles

Undirected graph

- BFS or DFS. If we reach a node we've seen already, then we've found a cycle

Directed graph

- Call TopologicalSort (time permitting, more on this later)
- If the length of the list returned $\neq |V|$ then a cycle exists

90