GRAPHS

David Kauchak
CS 62 – Fall 2026

1

Admin

Week recap:

- BST lab
- Hex-A-Pawn

Text Generator assignment

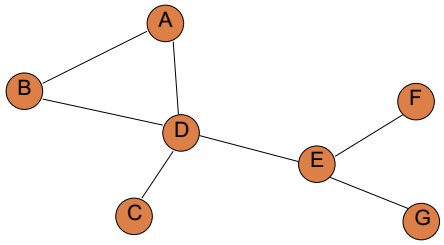
Assignments for the rest of the semester

Spring 2027 pre-enrollment

2

Graphs

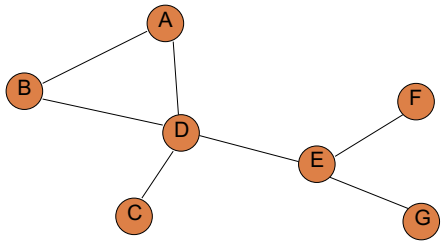
A mathematical model consisting of a set of nodes/vertices and edges



3

Graphs

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

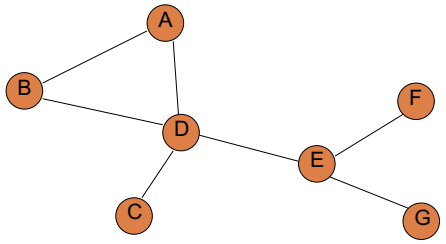


4

Graphs

V = {A, B, C, D, E, F, G}

E = {(A,B), (A,D), (B,D), (C,D), (D,E), (E,F), (E,G)}



5

When do we see graphs in real life problems?

Transportation networks (flights, roads, etc.)

Communication networks

Web

Social networks

Circuit design

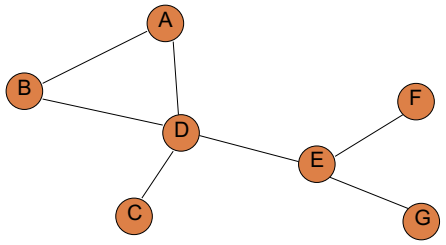
Bayesian networks

6

Graphs

How do graphs differ?

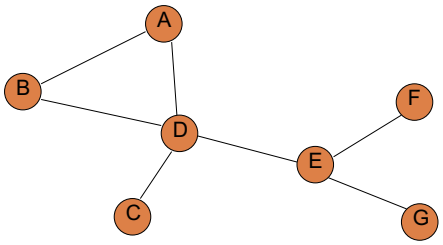
What are graph characteristics we might care about?



7

Different types of graphs

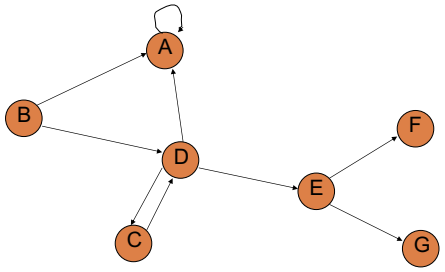
Undirected – edges do not have a direction



8

Different types of graphs

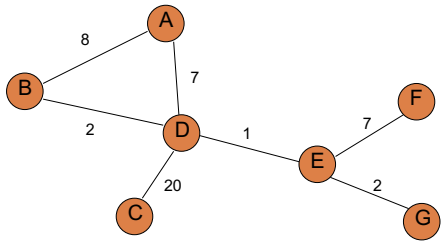
Directed – edges **do** have a direction



9

Different types of graphs

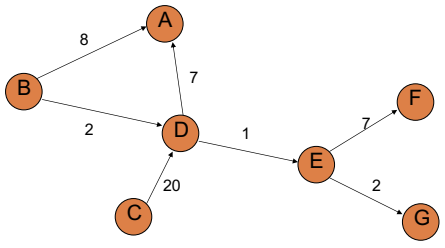
Weighted – edges have an associated weight



10

Different types of graphs

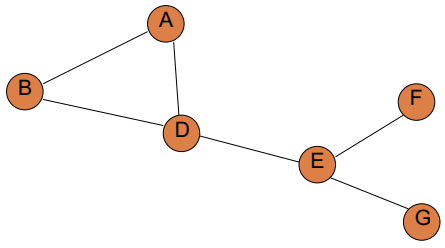
Weighted – edges have an associated weight



11

Terminology

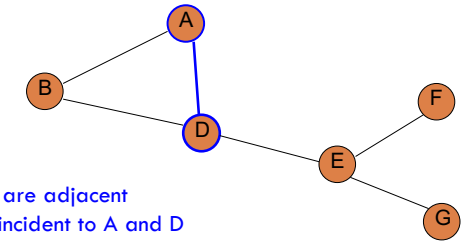
When an edge connects two vertices, we say that the vertices are adjacent and that the edge is incident to both vertices



12

Terminology

When an edge connects two vertices, we say that the vertices are adjacent and that the edge is incident to both vertices

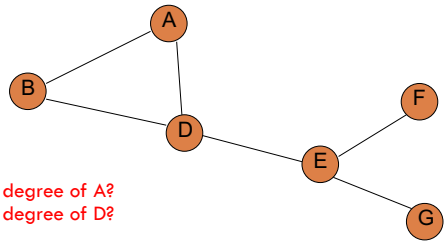


A and D are adjacent
(A, D) is incident to A and D

13

Terminology

The degree of a vertex is the number of edges incident to it

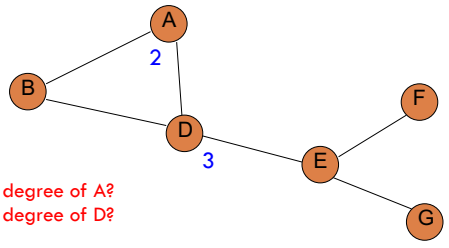


What is the degree of A?
What is the degree of D?

14

Terminology

The degree of a vertex is the number of edges incident to it

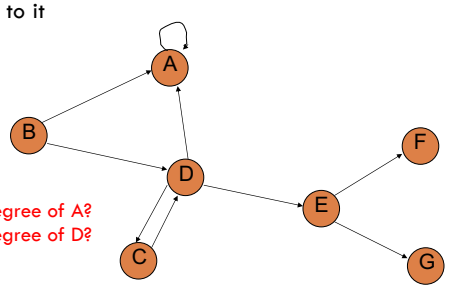


What is the degree of A?
What is the degree of D?

15

Terminology

The degree of a vertex is the number of edges incident to it

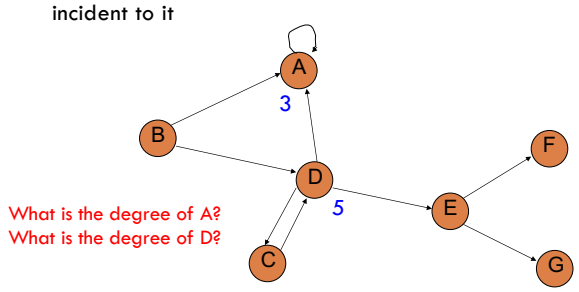


What is the degree of A?
What is the degree of D?

16

Terminology

The degree of a vertex is the number of edges incident to it

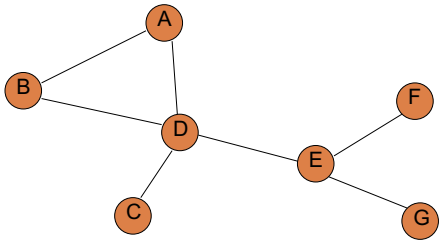


What is the degree of A?
What is the degree of D?

17

Terminology

Path – A path is a sequence of vertices $p_1,p_2,\ldots p_k$ where there exists an edge $(p_i,p_{i+1}) \in E$ and no edge is repeated

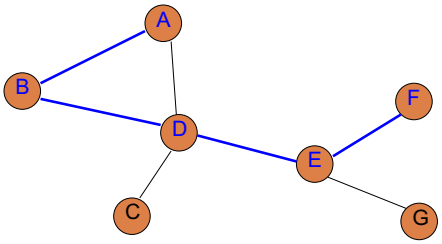


18

Terminology

Path – A path is a sequence of vertices $p_1,p_2,\ldots p_k$ where there exists an edge $(p_i,p_{i+1}) \in E$ and no edge is repeated

{A, B, D, E, F}

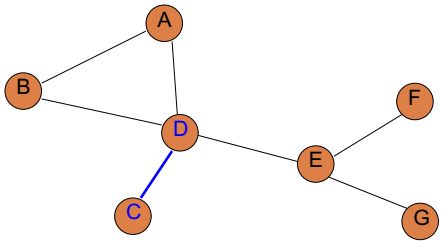


19

Terminology

Path – A path is a sequence of vertices $p_1,p_2,\ldots p_k$ where there exists an edge $(p_i,p_{i+1}) \in E$ and no edge is repeated

{C, D}

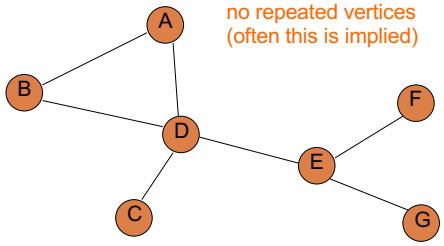


20

Terminology

Path – A path is a sequence of vertices $p_1, p_2, \ldots p_k$ where there exists an edge $(p_i, p_{i+1}) \in E$ and no edge is repeated

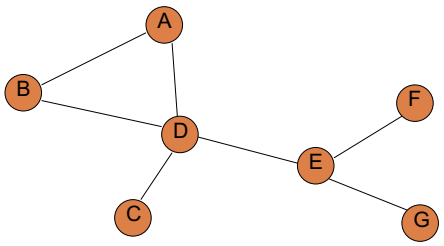
A *simple* path contains no repeated vertices (often this is implied)



21

Terminology

Cycle – A path where the first and last node are the same



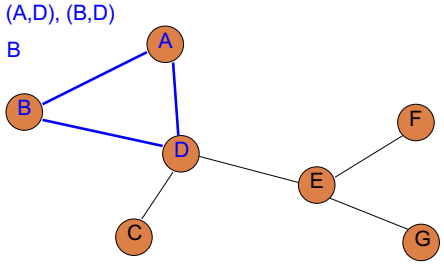
22

Terminology

Cycle – A path where the first and last node are the same

Edges: (A,B), (A,D), (B,D)

Path: B, A, D, B

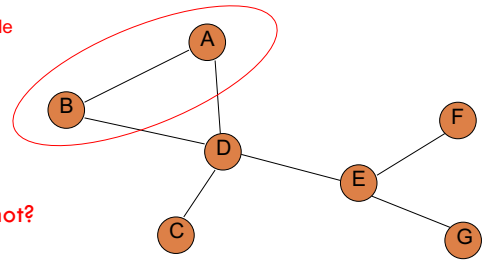


23

Terminology

Cycle – A path where the first and last node are the same

not a cycle



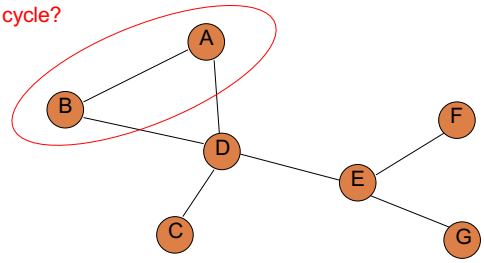
Why not?

24

Terminology

Cycle – A path where the first and last node are the same

Is this a cycle?

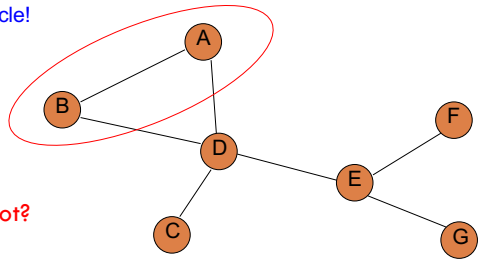


25

Terminology

Cycle – A path where the first and last node are the same

Not a cycle!



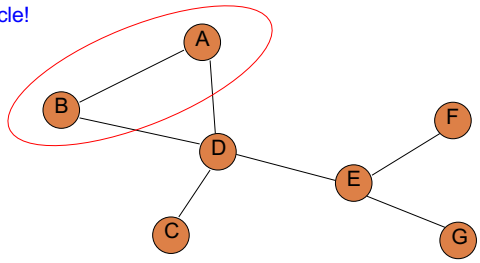
Why not?

26

Path – A path is a sequence of vertices $p_1, p_2, \ldots p_k$ where there exists an edge $(p_i, p_{i+1}) \in E$ and no edge is repeated

Cycle – A path where the first and last node are the same

Not a cycle!

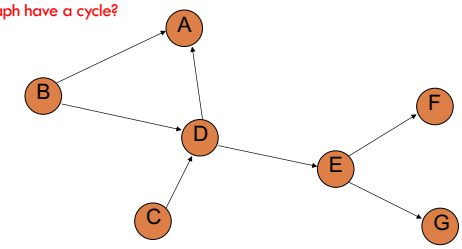


27

Terminology

Cycle – A path where the first and last node are the same

Does this graph have a cycle?

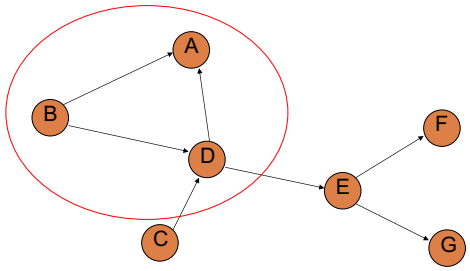


28

Terminology

Cycle – A path where the first and last node are the same

not a cycle

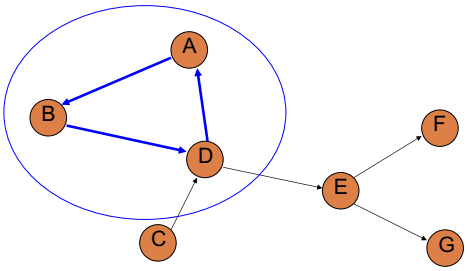


29

Terminology

Cycle – A path $p_1, p_2, \ldots p_k$ where $p_1 = p_k$

This would be a cycle

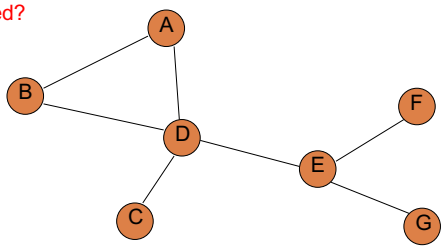


30

Terminology

Connected – every pair of vertices is connected by a path

Is this graph connected?

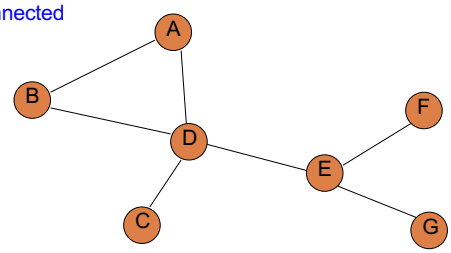


31

Terminology

Connected – every pair of vertices is connected by a path

connected

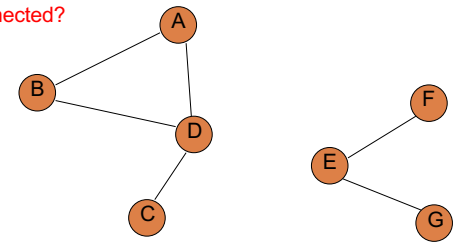


32

Terminology

Connected – every pair of vertices is connected by a path

Is this graph connected?

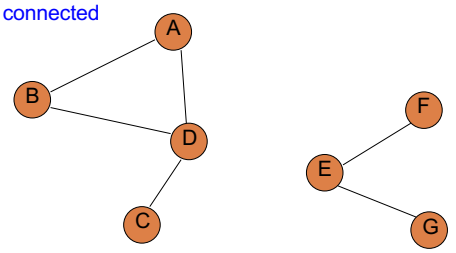


33

Terminology

Connected – every pair of vertices is connected by a path

not connected

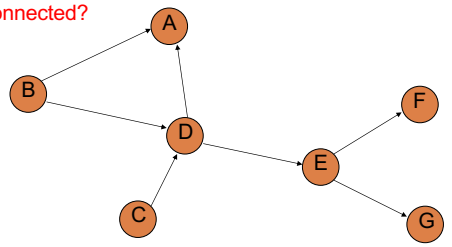


34

Terminology

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

Is this graph strongly connected?

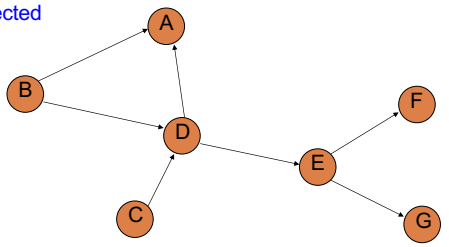


35

Terminology

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

not strongly connected

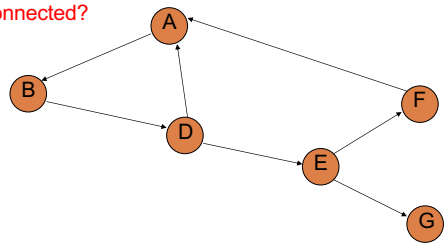


36

Terminology

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

Is this graph
strongly connected?

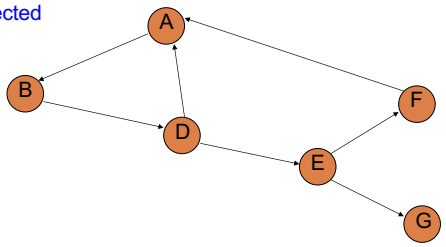


37

Terminology

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

not strongly
connected

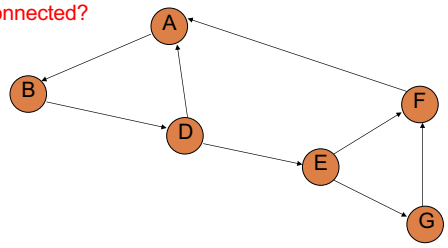


38

Terminology

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

Is this graph
strongly connected?

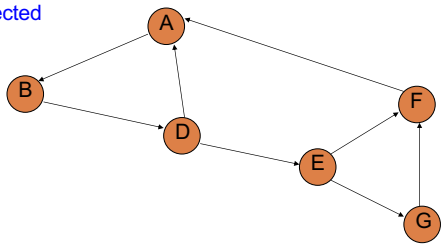


39

Terminology

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

strongly
connected



40

Graphs aren't new…

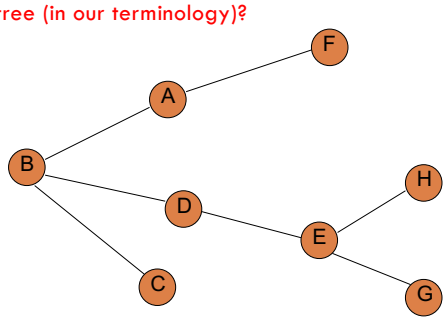
Have we seen graphs in this class already?

Trees!

41

Different types of graphs

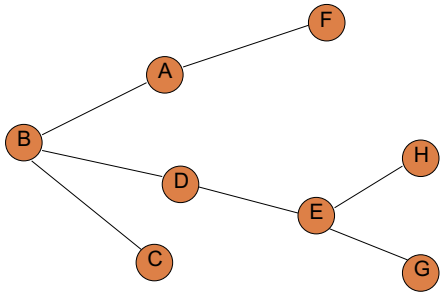
What is a tree (in our terminology)?



42

Different types of graphs

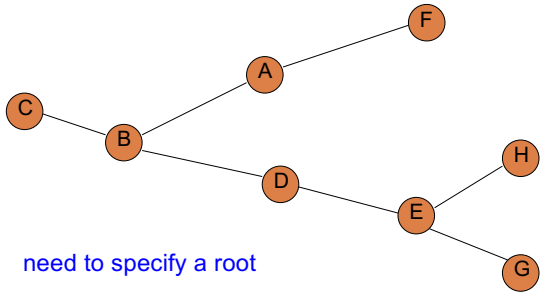
Tree – connected, undirected graph without any cycles



43

Different types of graphs

Tree – connected, undirected graph without any cycles

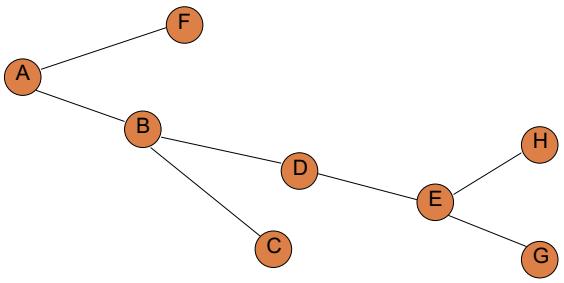


need to specify a root

44

Different types of graphs

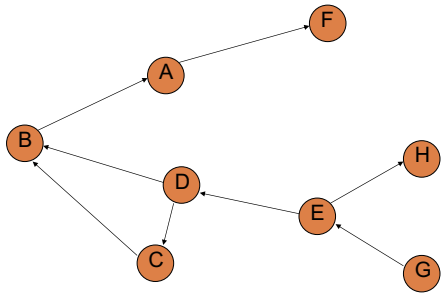
Tree – connected, undirected graph without any cycles



45

Different types of graphs

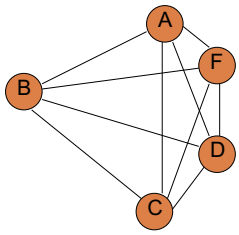
DAG – directed, acyclic graph



46

Different types of graphs

Complete graph – an edge exists between every node



47

Graph questions?

Does it have a cycle?

Is it connected? Strongly connected?

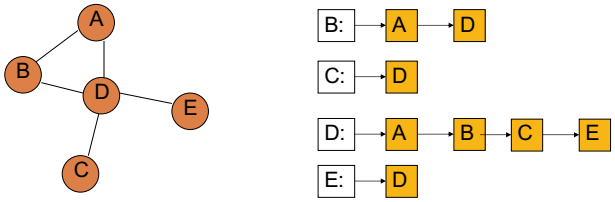
Is there a path from a to b?

What is the shortest path from a to b? In number of edges? In sum of the edge weights?

48

Representing graphs

Adjacency list – Each vertex u ∈ V contains an adjacency list of the set of vertices v such that there exists an edge (u,v) ∈ E

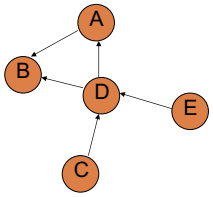


A: → B → D
B: → A → D
C: → D
D: → A → B → C → E
E: → D

49

Representing graphs

Adjacency list – Each vertex u ∈ V contains an adjacency list of the set of vertices v such that there exists an edge (u,v) ∈ E



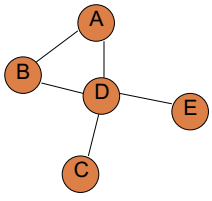
A: → B
B:
C: → D
D: → A → B
E: → D

50

Representing graphs

Adjacency matrix – A |V|x|V| matrix A such that:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



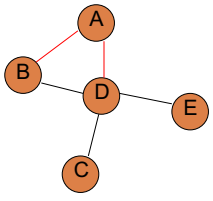
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	0	1	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

51

Representing graphs

Adjacency matrix – A |V|x|V| matrix A such that:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



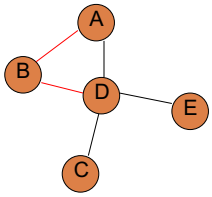
	A	B	C	D	E
A	0	**1**	0	**1**	0
B	1	0	0	1	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

52

Representing graphs

Adjacency matrix – A |V|x|V| matrix A such that:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



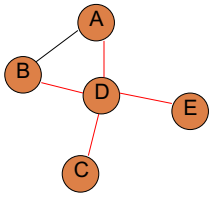
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	0	1	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

53

Representing graphs

Adjacency matrix – A |V|x|V| matrix A such that:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	A	B	C	D	E
A	0	1	0	1	0
B	1	0	0	1	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

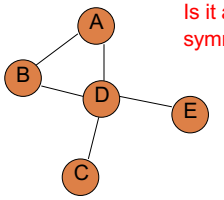
54

Representing graphs

Adjacency matrix – A |V|x|V| matrix A such that:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Is it always symmetric?



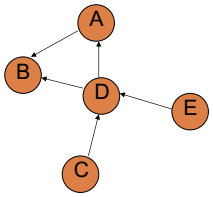
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	0	1	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

55

Representing graphs

Adjacency matrix – A |V|x|V| matrix A such that:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	A	B	C	D	E
A	0	1	0	0	0
B	0	0	0	0	0
C	0	0	0	1	0
D	1	1	0	0	0
E	0	0	0	1	0

56

Adjacency list vs. adjacency matrix

Adjacency list	Adjacency matrix

Pros/Cons?

57

Adjacency list vs. adjacency matrix

Adjacency list	Adjacency matrix
Sparse graphs (e.g. web) Space efficient Must traverse the adjacency list to discover is an edge exists	Dense graphs Constant time lookup to discover if an edge exists Simple to implement For non-weighted graphs, only requires boolean matrix

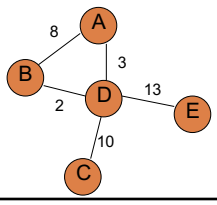
58

Weighted graphs

Adjacency list

- store the weight as an additional field in the list



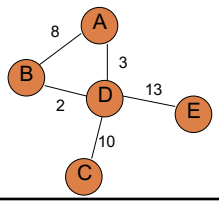


59

Weighted graphs

Adjacency matrix

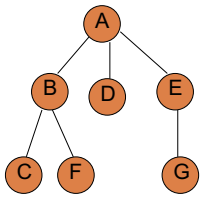
$$a_{ij} = \begin{cases} weight & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	A	B	C	D	E
A	0	8	0	3	0
B	8	0	0	2	0
C	0	0	0	10	0
D	3	2	10	0	13
E	0	0	0	13	0

60

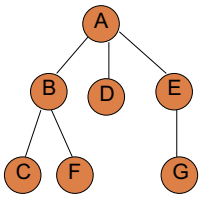
Searching a tree



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

61

Searching a tree



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

62

A flash from the past

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

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

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

63

Searching a tree

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

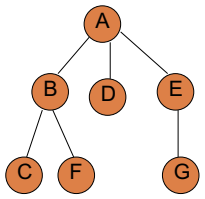
```
treeDFS( start )
    s = new Stack()
    s.add(start)
    treeSearch(s)
```

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

64

Tree BFS

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



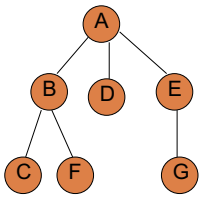
q:

Visited:

65

Tree BFS

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



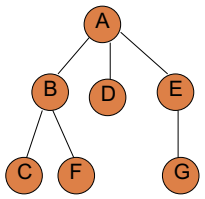
q: A

Visited:

66

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

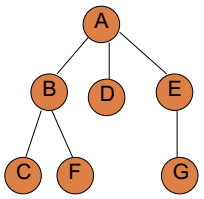
printed:

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

67

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)
```



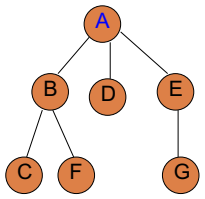
toVisit-queue: A

printed:

68

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)
```



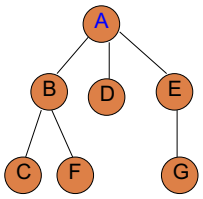
toVisit-queue:

printed: A

69

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)
```



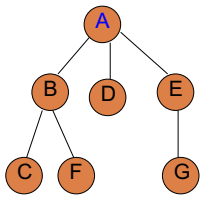
toVisit-queue:

printed: A

70

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)
```



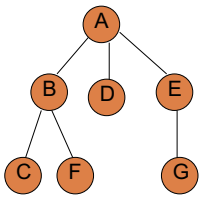
toVisit-queue: B D E

printed: A

71

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)
```



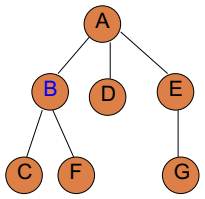
toVisit-queue: B D E

printed: A

72

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)
```



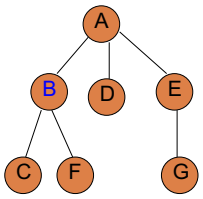
toVisit-queue: D E

printed: A B

73

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)
```



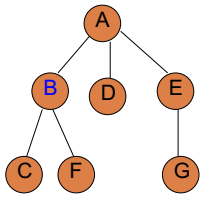
toVisit-queue: D E

printed: A B

74

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)
```



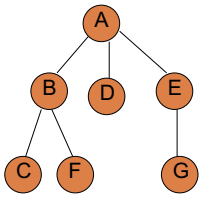
toVisit-queue: D E C F

printed: A B

75

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)
```



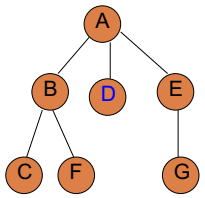
toVisit-queue: D E C F

printed: A B

76

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)
```



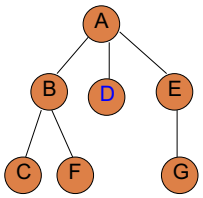
toVisit-queue: E C F

printed: A B D

77

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)
```



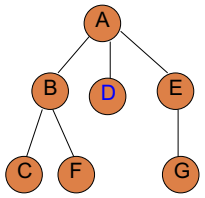
toVisit-queue: E C F

printed: A B D

78

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)
```



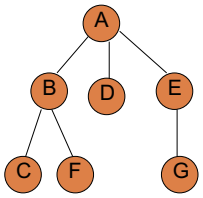
toVisit-queue: E C F

printed: A B D

79

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)
```



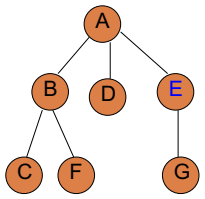
toVisit-queue: E C F

printed: A B D

80

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)
```



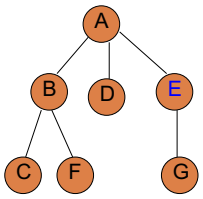
toVisit-queue: C F

printed: A B D E

81

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)
```



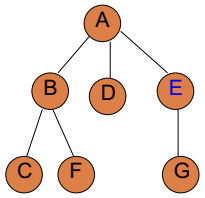
toVisit-queue: C F

printed: A B D E

82

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)
```



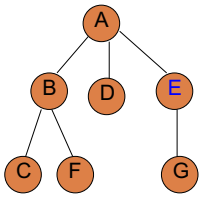
toVisit-queue: C F G

printed: A B D E

83

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

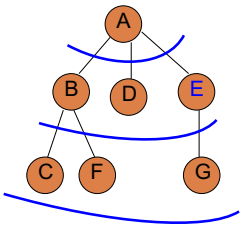
printed: A B D E

How are we exploring the vertices?

84

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

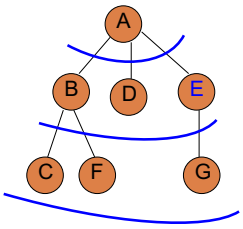
printed: A B D E

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

85

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

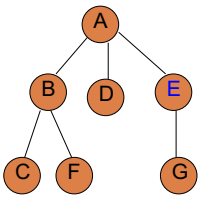
printed: A B D E

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

86

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)
```



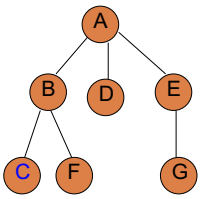
toVisit-queue: C F G

printed: A B D E

87

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)
```



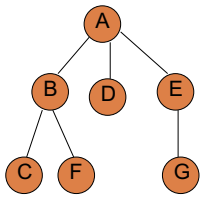
toVisit-queue: F G

printed: A B D E C

88

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)
```



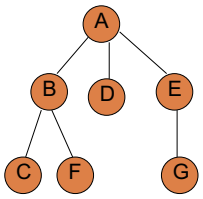
toVisit-queue:

printed: A B D E C F G

89

Tree DFS

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



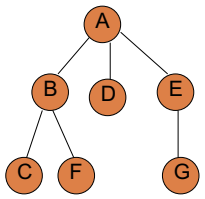
s:

printed:

90

Tree DFS

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



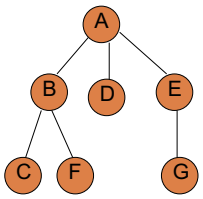
s: A

printed:

91

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

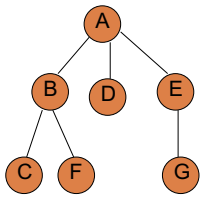
printed:

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

92

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)
```



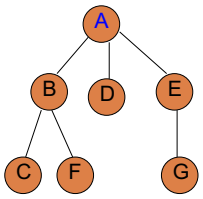
toVisit-stack: A

printed:

93

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)
```



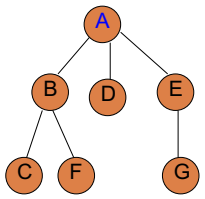
toVisit-stack:

printed: A

94

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)
```



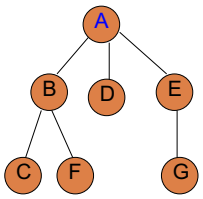
toVisit-stack:

printed: A

95

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)
```



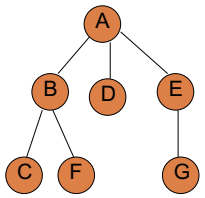
toVisit-stack: B D E

printed: A

96

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)
```



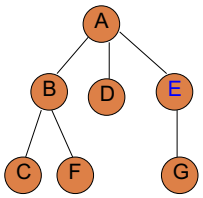
toVisit-stack: B D E

printed: A

97

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)
```



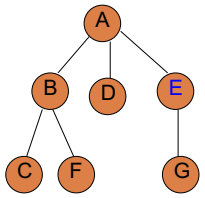
toVisit-stack: B D

printed: A E

98

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)
```



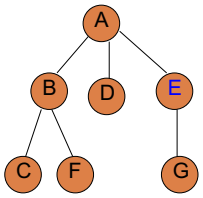
toVisit-stack: B D

printed: A E

99

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)
```



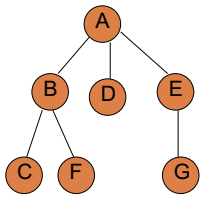
toVisit-stack: B D G

printed: A E

100

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)
```



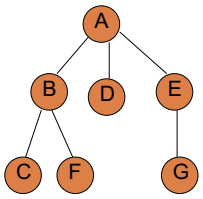
toVisit-stack: B D G

printed: A E

101

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)
```



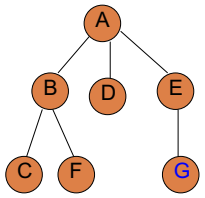
toVisit-stack: B D

printed: A E G

102

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)
```



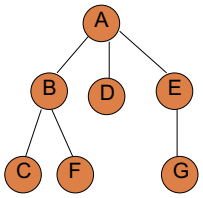
toVisit-stack: B D

printed: A E G

103

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)
```



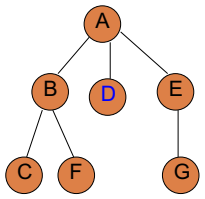
toVisit-stack: B D

printed: A E G

104

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)
```



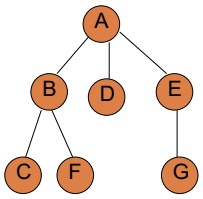
toVisit-stack: B

printed: A E G D

105

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

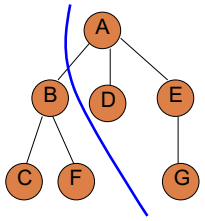
printed: A E G D

How are we exploring the vertices?

106

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

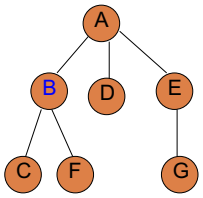
printed: A E G D

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

107

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)
```



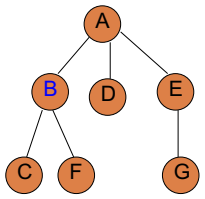
toVisit-stack:

printed: A E G D B

108

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)
```



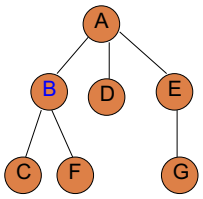
toVisit-stack:

printed: A E G D B

109

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)
```



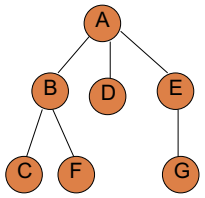
toVisit-stack: C F

printed: A E G D B

110

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)
```



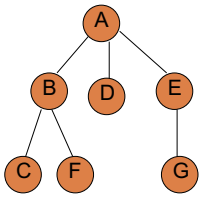
toVisit-stack: C F

printed: A E G D B

111

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

112

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:

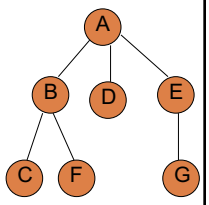
- |V| = number of vertices
- |E| = number of edges

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

113

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.

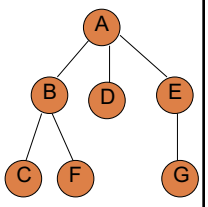
How many times do we visit each vertex?

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

114

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

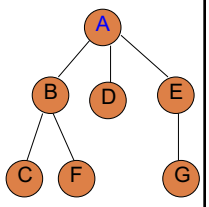
How many times do we traverse each edge (via the for loop)? Exactly once

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

115

What algorithm is this?

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

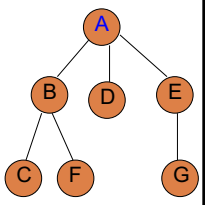


What order will the vertices get printed out?
Assume children are traversed left to right.

116

What algorithm is this?

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

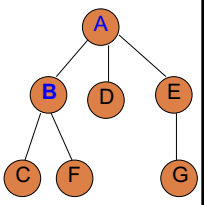


Visited: A

117

What algorithm is this?

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

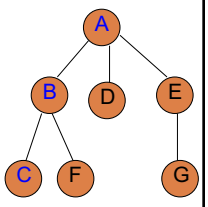


Visited: A B

118

What algorithm is this?

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

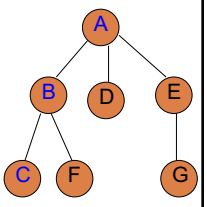


Visited: A B C

119

What algorithm is this?

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

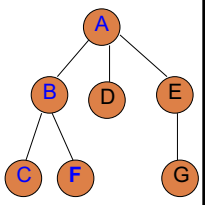


Visited: A B C Now where?

120

What algorithm is this?

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

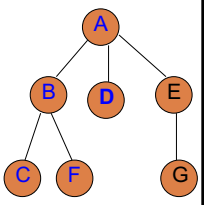


Visited: A B C F

121

What algorithm is this?

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

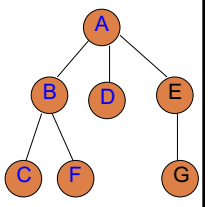


Visited: A B C F D

122

What algorithm is this?

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



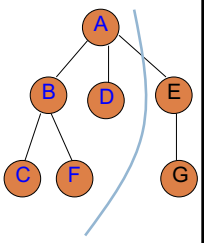
What algorithm is this?

Visited: A B C F D

123

What algorithm is this?

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



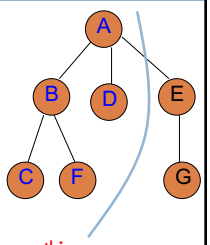
Depth first search!

Visited: A B C F D

124

What algorithm is this?

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



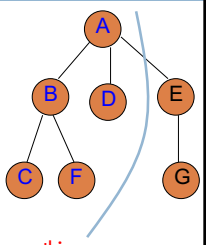
Any difference between this version and the stack version?

Visited: A B C F D

125

What algorithm is this?

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



Any difference between this version and the stack version?

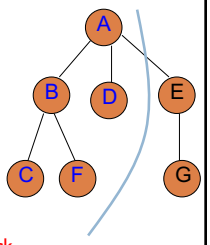
Visited: A B C F D

Traverses in the other direction (left to right in this case).

126

What algorithm is this?

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



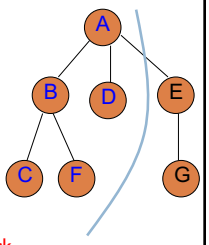
treeDFS used a stack.
Is there a stack here?

Visited: A B C F D

127

What algorithm is this?

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



treeDFS used a stack.
Is there a stack here?

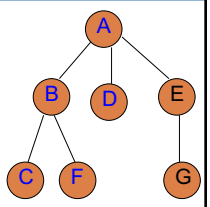
Visited: A B C F D

The run-time stack keeping track of recursive calls!

128

What algorithm is this?

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

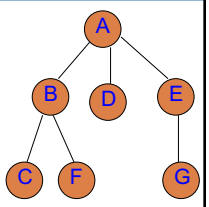


Visited: A B C F D

129

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 E G

130

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)
```

131