

GRAPHS

David Kauchak  
CS 1.40 – Spring 2024

1

Admin

Assignment 7

Group 7

Hashtables

2

Graphs

What is a graph?

```
graph TD; A --- B; A --- D; B --- D; D --- C; D --- E; E --- F; E --- G;
```

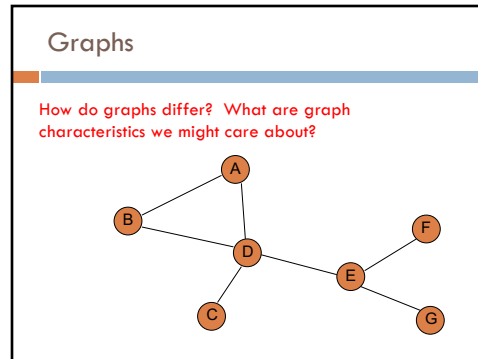
3

Graphs

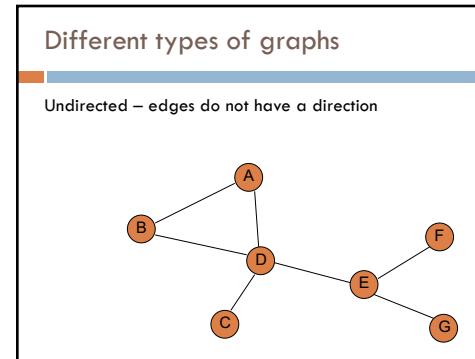
A graph is a set of vertices  $V$  and a set of edges  $(u,v) \in E$  where  $u,v \in V$

```
graph TD; A --- B; A --- D; B --- D; D --- C; D --- E; E --- F; E --- G;
```

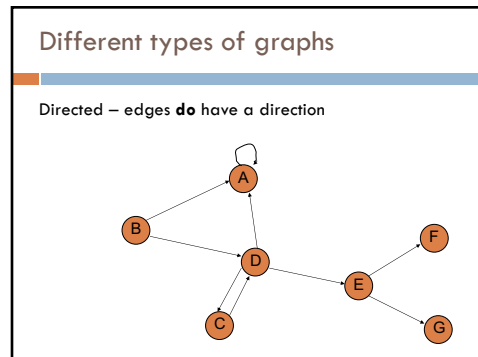
4



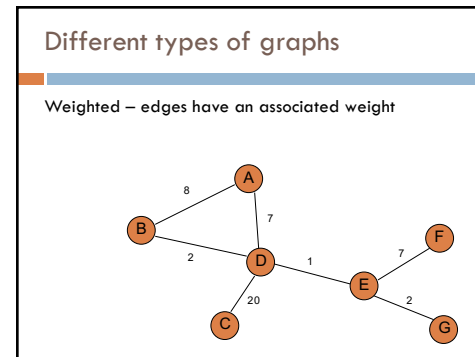
5



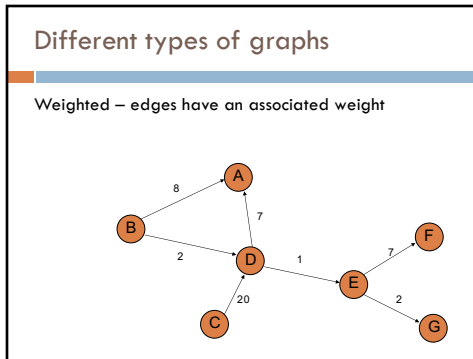
6



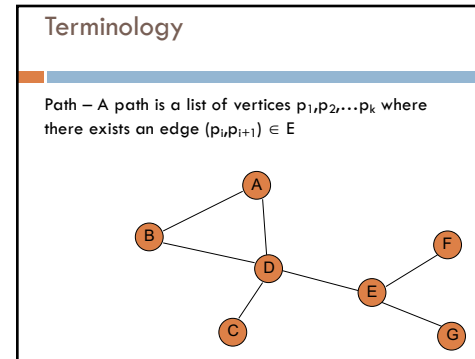
7



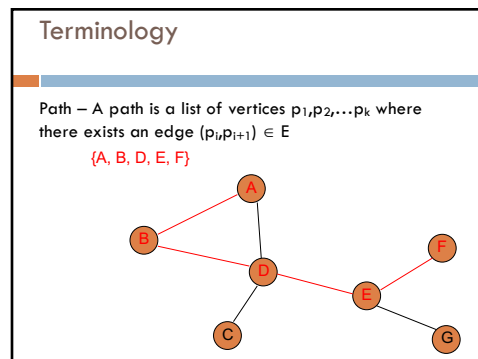
8



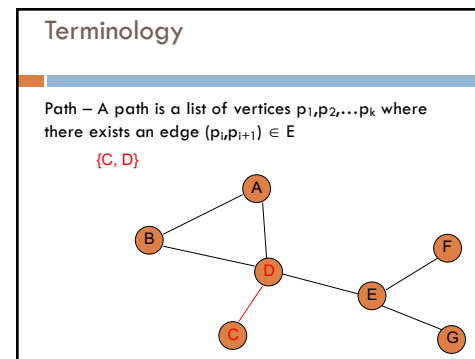
9



10



11



12

### Terminology

Path – A path is a list of vertices  $p_1, p_2, \dots, p_k$  where there exists an edge  $(p_i, p_{i+1}) \in E$

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

13

### Terminology

Cycle?

14

### Terminology

Cycle – A subset of the edges that form a path such that the first and last node are the same

15

### Terminology

Cycle – A subset of the edges that form a path such that the first and last node are the same

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

16

### Terminology

Cycle – A subset of the edges that form a path such that the first and last node are the same

cycle?

17

### Terminology

Cycle – A subset of the edges that form a path such that the first and last node are the same

not a cycle

18

### Terminology

Cycle – A subset of the edges that form a path such that the first and last node are the same

Does this graph have a cycle?

19

### Terminology

Cycle – A subset of the edges that form a path such that the first and last node are the same

not a cycle

20

### Terminology

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

cycle

21

### Terminology

Connected – every pair of vertices is connected by a path

Is this graph connected?

22

### Terminology

Connected – every pair of vertices is connected by a path

connected

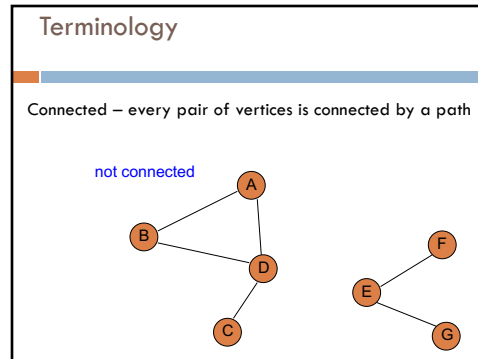
23

### Terminology

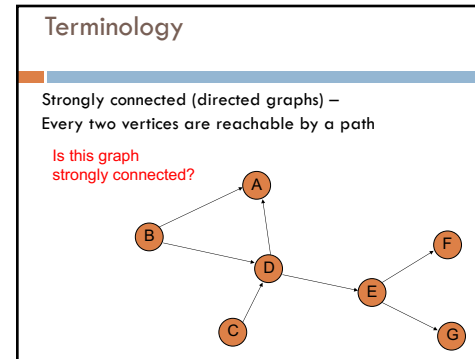
Connected – every pair of vertices is connected by a path

Is this graph connected?

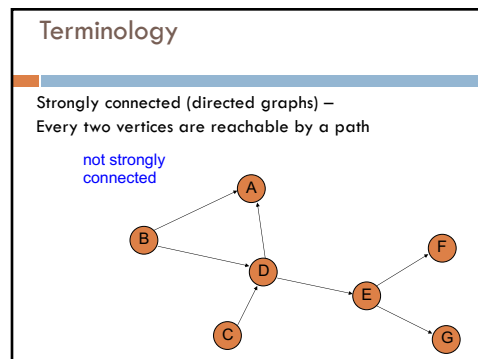
24



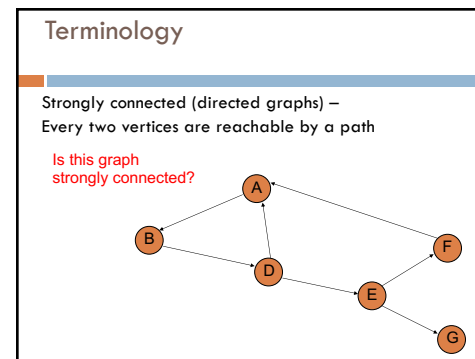
25



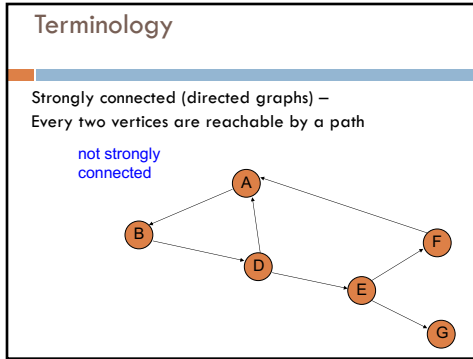
26



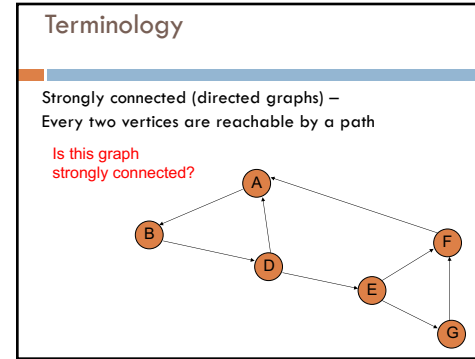
27



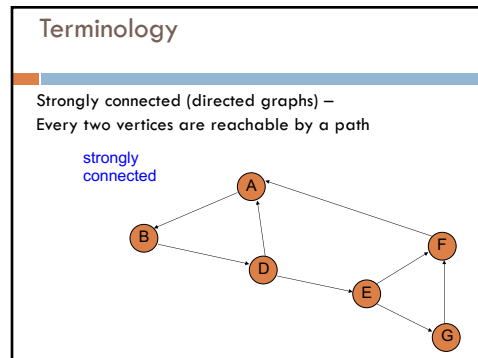
28



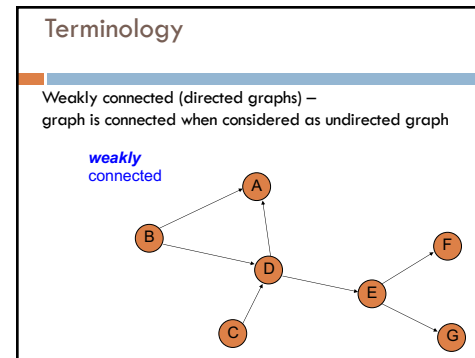
29



30

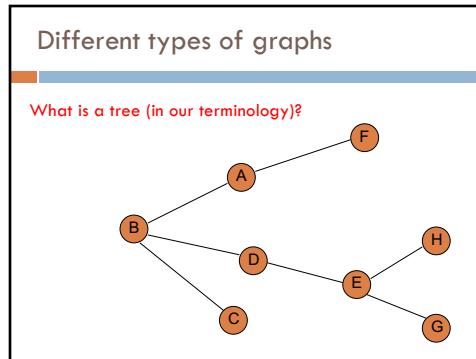


31

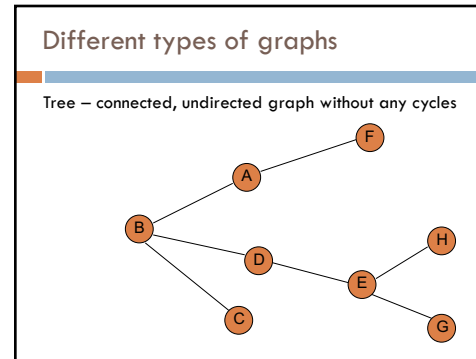


32

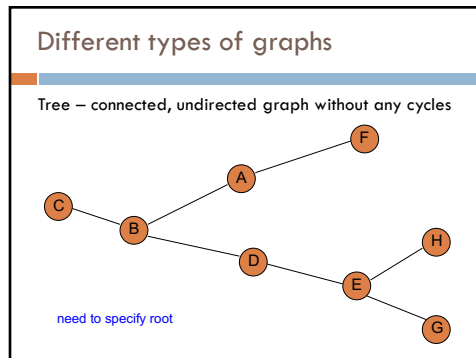




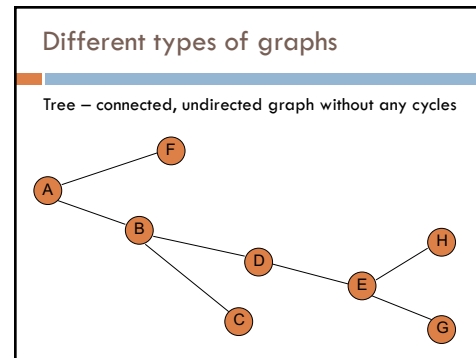
33



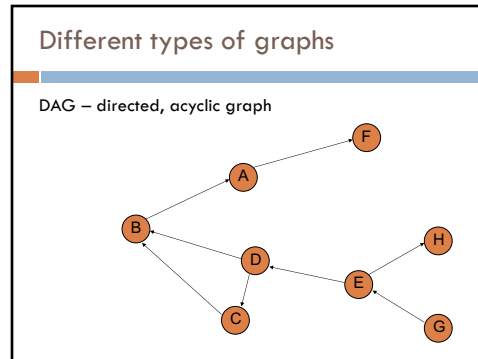
34



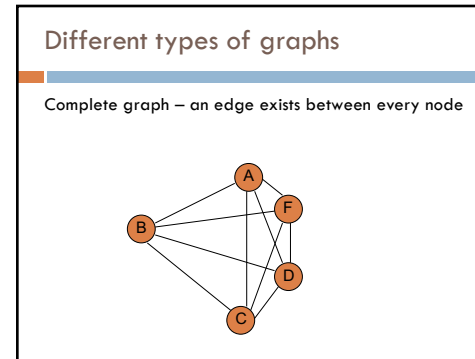
35



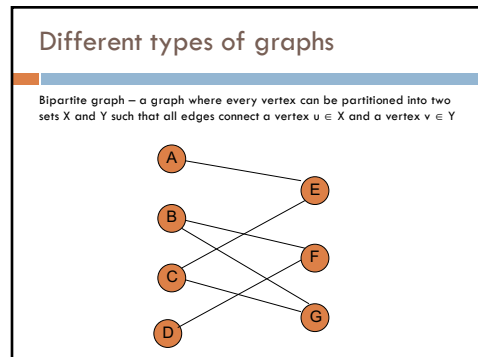
36



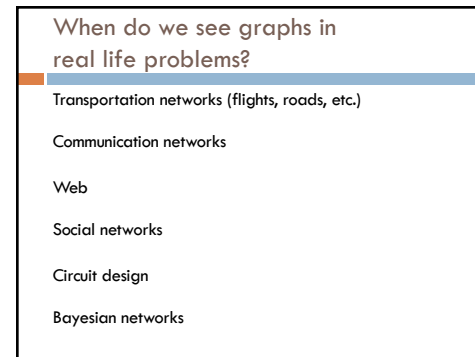
37



38



39



40

### Representing graphs

41

### Representing graphs

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

```

A: B D
B: A D
C: D
D: A B C E
E: D
    
```

42

### Representing graphs

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

```

A: B
B:
C: D
D: A B
E: D
    
```

43

### Representing graphs

Adjacency matrix – A  $|V| \times |V|$  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

44

### Representing graphs

Adjacency matrix – A  $|V| \times |V|$  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

45

### Representing graphs

Adjacency matrix – A  $|V| \times |V|$  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

46

### Representing graphs

Adjacency matrix – A  $|V| \times |V|$  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

47

### Representing graphs

Adjacency matrix – A  $|V| \times |V|$  matrix A such that:

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

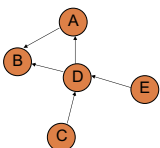
Is it always symmetric?

	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

48

### Representing graphs

Adjacency matrix – A  $|V| \times |V|$  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

49

### Adjacency list vs. adjacency matrix

Adjacency list	Adjacency matrix

Pros and cons?

50

### Adjacency list vs. adjacency matrix

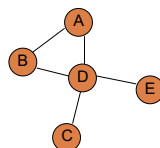
Adjacency list	Adjacency matrix
Sparse graphs (e.g. web) Space efficient Must traverse the adjacency list to discover if an edge exists	Dense graphs Constant time lookup to discover if an edge exists Simple to implement For non-weighted graphs, only requires boolean matrix

Can we get the best of both worlds?

51

### Sparse adjacency matrix

Rather than using an adjacency list, use an adjacency hashtable

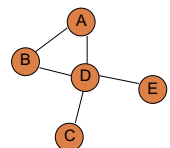


A:	hashtable [B,D]
B:	hashtable [A,D]
C:	hashtable [D]
D:	hashtable [A,B,C,E]
E:	hashtable [D]

52

### Sparse adjacency matrix

Constant time lookup  
 Fairly space efficient  
 Not good for dense graphs, why?

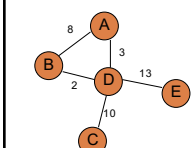
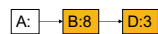


A:	hashtable [B,D]
B:	hashtable [A,D]
C:	hashtable [D]
D:	hashtable [A,B,C,E]
E:	hashtable [D]

53

### Weighted graphs

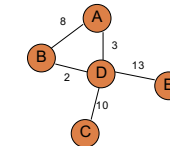
Adjacency list  
 store the weight as an additional field in the list

54

### Weighted graphs

Adjacency matrix

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


	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

55

### Graph algorithms/questions

Graph traversal (BFS, DFS)

Shortest path from a to b

- unweighted
- weighted positive weights
- negative/positive weights

Minimum spanning trees

Are all nodes in the graph connected?

Is the graph bipartite?

56

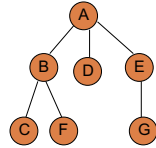
## DFS and BFS

How are they implemented?

What would be the result starting at A?  
If you ask for the children of a node,  
they're given in alphabetical order.

Run-time (in terms of V and E):

- adjacency list
- adjacency matrix



57

## Search implemented

```

TreeBFS(T)
1 ENQUEUE(Q, ROOT(T))
2 while !EMPTY(Q)
3   v ← DEQUEUE(Q)
4   VISIT(v)
5   for all c ∈ CHILDREN(v)
6     ENQUEUE(Q, c)
  
```

```

TreeDFS(T)
1 PUSH(S, ROOT(T))
2 while !EMPTY(S)
3   v ← POP(S)
4   VISIT(v)
5   for all c ∈ CHILDREN(v)
6     PUSH(S, c)
  
```

```

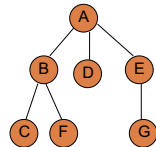
TreeDFS(v)
visit(v)
if not leaf(v)
  for all c in children(x)
    TreeDFS(v)
  
```

58

## BFS

```

TreeBFS(T)
1 ENQUEUE(Q, ROOT(T))
2 while !EMPTY(Q)
3   v ← DEQUEUE(Q)
4   VISIT(v)
5   for all c ∈ CHILDREN(v)
6     ENQUEUE(Q, c)
  
```

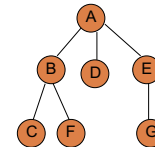


59

## BFS

```

TreeBFS(T)
1 ENQUEUE(Q, ROOT(T))
2 while !EMPTY(Q)
3   v ← DEQUEUE(Q)
4   VISIT(v)
5   for all c ∈ CHILDREN(v)
6     ENQUEUE(Q, c)
  
```



ABDECFG

60

### DFS

```

TreeDFS(T)
1 PUSH(S, ROOT(T))
2 while EMPTY(S)
3   v ← POP(S)
4   VISIT(v)
5   for all c ∈ CHILDREN(v)
6     PUSH(S, c)

```

```

graph TD
  A((A)) --- B((B))
  A --- D((D))
  A --- E((E))
  B --- C((C))
  B --- F((F))
  E --- G((G))

```

61

### DFS

```

TreeDFS(T)
1 PUSH(S, ROOT(T))
2 while EMPTY(S)
3   v ← POP(S)
4   VISIT(v)
5   for all c ∈ CHILDREN(v)
6     PUSH(S, c)

```

```

graph TD
  A((A)) --- B((B))
  A --- D((D))
  A --- E((E))
  B --- C((C))
  B --- F((F))
  E --- G((G))

```

AEGDBFC

62

### DFS

```

TreeDFS(v)
  visit(v)
  if not leaf(v)
    for all c in children(x)
      TreeDFS(v)

```

```

graph TD
  A((A)) --- B((B))
  A --- D((D))
  A --- E((E))
  B --- C((C))
  B --- F((F))
  E --- G((G))

```

What changes?

63

### DFS

```

TreeDFS(v)
  visit(v)
  if not leaf(v)
    for all c in children(x)
      TreeDFS(v)

```

```

graph TD
  A((A)) --- B((B))
  A --- D((D))
  A --- E((E))
  B --- C((C))
  B --- F((F))
  E --- G((G))

```

ABCFDEG

64



## Running time of BFS/DFS

### Adjacency list

- How many times does it visit each vertex?
- How many times is each edge traversed?
- $\Theta(|V| + |E|)$  – for trees, i.e., assuming a connected graph

### Adjacency matrix

- For each vertex visited, how much work is done?
- $\Theta(|V|^2)$  – for trees, i.e., assuming a connected graph

<pre> TREEBFS(T) 1 ENQUEUE(Q, ROOT(T)) 2 while EMPTY(Q) 3   e ← ENQUEUE(Q) 4   VISIT(e) 5   for all c ∈ CHILDREN(e) 6     ENQUEUE(Q, c) </pre>	<pre> TREEDFS(T) 1 PUSH(S, ROOT(T)) 2 while EMPTY(S) 3   v ← POP(S) 4   VISIT(v) 5   for all c ∈ CHILDREN(v) 6     PUSH(S, c) </pre>
--	--

65

## DFS/BFS

Do they visit all of the nodes?

If the graph is connected or strongly connected

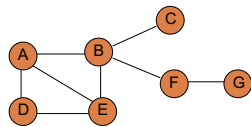
<pre> TREEBFS(T) 1 ENQUEUE(Q, ROOT(T)) 2 while EMPTY(Q) 3   e ← ENQUEUE(Q) 4   VISIT(e) 5   for all c ∈ CHILDREN(e) 6     ENQUEUE(Q, c) </pre>	<pre> TREEDFS(T) 1 PUSH(S, ROOT(T)) 2 while EMPTY(S) 3   v ← POP(S) 4   VISIT(v) 5   for all c ∈ CHILDREN(v) 6     PUSH(S, c) </pre>
--	--

66

## DFS/BFS for graphs

What needs to change for graphs?

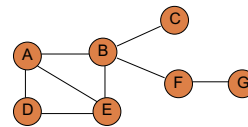
Need to make sure we don't visit a node multiple times



67

## BFS for graphs

What order will BFS visit starting at A (again, assume children are enumerated alphabetically)?



68

### BFS for graphs

What order will BFS visit starting at A (again, assume children are enumerated alphabetically)?

ABDECFG

69

```

BFS(G, s)
1 for each v in V
2   dist[v] = ∞
3 dist[s] = 0
4 ENQUEUE(Q, s)
5 while !EMPTY(Q)
6   u ← DEQUEUE(Q)
7   VISIT(u)
8   for each edge (u, v) in E
9     if dist[v] = ∞
10      ENQUEUE(Q, v)
11      dist[v] ← dist[u] + 1
    
```

distance variable keeps track of how far from the starting node and whether we've seen the node yet

70

<pre> BFS(G, s) 1 for each v in V 2   dist[v] = ∞ 3 dist[s] = 0 4 ENQUEUE(Q, s) 5 while !EMPTY(Q) 6   u ← DEQUEUE(Q) 7   VISIT(u) 8   for each edge (u, v) in E 9     if dist[v] = ∞ 10      ENQUEUE(Q, v) 11      dist[v] ← dist[u] + 1     </pre>	<pre> TREEBFS(T) 1 ENQUEUE(Q, ROOT(T)) 2 while !EMPTY(Q) 3   v ← DEQUEUE(Q) 4   VISIT(v) 5   for all c in CHILDREN(v) 6     ENQUEUE(Q, c)     </pre>
---	--

71

### DFS on graphs

```

DFS(G)
1 for all v in V
2   visited[v] ← false
3 for all v in V
4   if !visited[v]
5     DFS-VISIT(v)

DFS-VISIT(u)
1 visited[u] ← true
2 PREVISIT(v)
3 for all edges (u, v) in E
4   if !visited[v]
5     DFS-VISIT(v)
6 POSTVISIT(v)
    
```

72

### DFS on graphs

```

DFS(G)
1 for all v in V
2   visited[v] ← false
3 for all v in V
4   if !visited[v]
5     DFS-VISIT(v)

DFS-VISIT(u)
1  visited[u] ← true
2  PREVISIT(u)
3  for all edges (u,v) in E
4    if !visited[v]
5      DFS-VISIT(v)
6  POSTVISIT(u)
    
```

mark all nodes as not visited

73

### DFS on graphs

```

DFS(G)
1 for all v in V
2   visited[v] ← false
3 for all v in V
4   if !visited[v]
5     DFS-VISIT(v)

DFS-VISIT(u)
1  visited[u] ← true
2  PREVISIT(u)
3  for all edges (u,v) in E
4    if !visited[v]
5      DFS-VISIT(v)
6  POSTVISIT(u)
    
```

until all nodes have been visited repeatedly call DFS-Visit

74

### DFS for graphs

What order will DFS visit starting at A (again, assume children are enumerated alphabetically)?

```

DFS(G)
1 for all v in V
2   visited[v] ← false
3 for all v in V
4   if !visited[v]
5     DFS-VISIT(v)

DFS-VISIT(u)
1  visited[u] ← true
2  PREVISIT(u)
3  for all edges (u,v) in E
4    if !visited[v]
5      DFS-VISIT(v)
6  POSTVISIT(u)
    
```

75

### DFS for graphs

What order will DFS visit starting at A (again, assume children are enumerated alphabetically)?

```

DFS(G)
1 for all v in V
2   visited[v] ← false
3 for all v in V
4   if !visited[v]
5     DFS-VISIT(v)

DFS-VISIT(u)
1  visited[u] ← true
2  PREVISIT(u)
3  for all edges (u,v) in E
4    if !visited[v]
5      DFS-VISIT(v)
6  POSTVISIT(u)
    
```

ABCEDFG

76

## What does DFS do?

Finds connected components

Each call to DFS-Visit from DFS starts exploring a new set of connected components

Helps us understand the structure/connectedness of a graph

77

## Running time of graph BFS/DFS

Nothing changes!

- Adjacency list
  - $O(|V| + |E|)$
- Adjacency matrix
  - $O(|V|^2)$

78

Handout

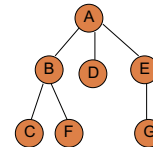
79

## DFS and BFS

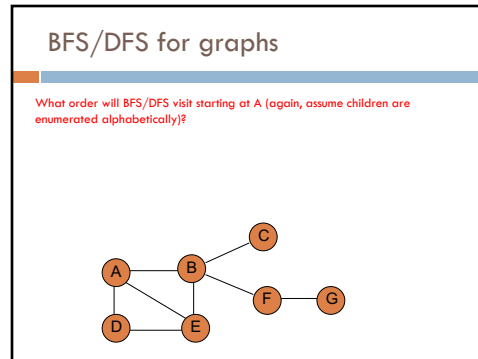
How are they implemented?

What would be the result starting at A? If you ask for the children of a node, they're given in alphabetical order.

- Run-time (in terms of V and E):
- adjacency list
  - adjacency matrix



80



81