GRAPHS

David Kauchak
CS 140 – Fall 2024

1

Admin

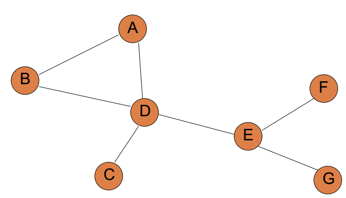
Assignment 7

Group 7

2

Graphs

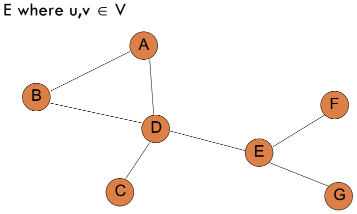
What is a graph?



3

Graphs

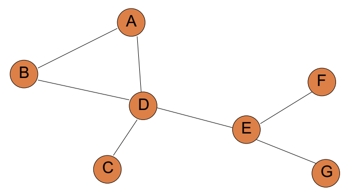
A graph is a set of vertices V and a set of edges $(u,v) \in E$ where $u,v \in V$



4

Graphs

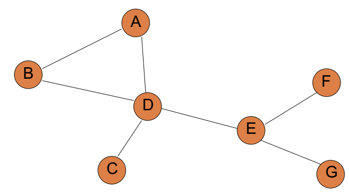
How do graphs differ? What are graph characteristics we might care about?



5

Different types of graphs

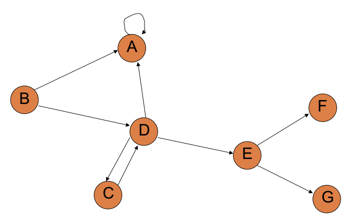
Undirected – edges do not have a direction



6

Different types of graphs

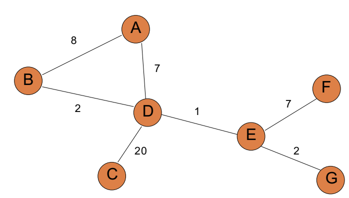
Directed – edges **do** have a direction



7

Different types of graphs

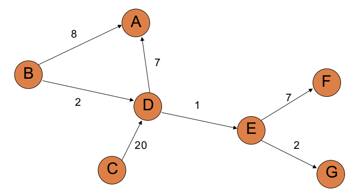
Weighted – edges have an associated weight



8

Different types of graphs

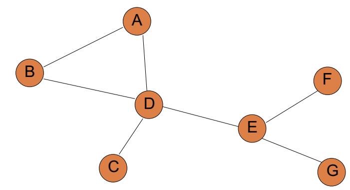
Weighted – edges have an associated weight



9

Terminology

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

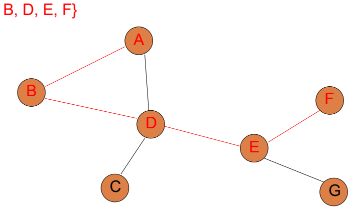


10

Terminology

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

{A, B, D, E, F}

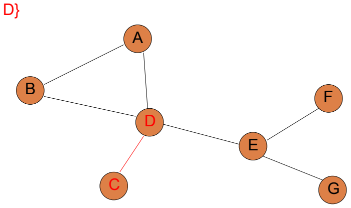


11

Terminology

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

{C, D}

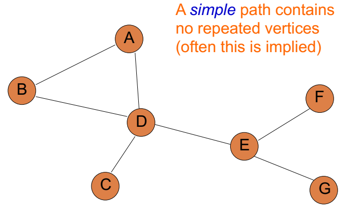


12

Terminology

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

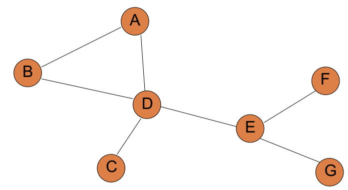
A *simple* path contains no repeated vertices (often this is implied)



13

Terminology

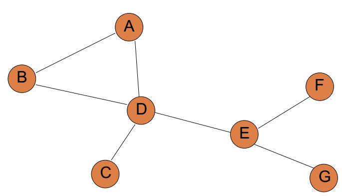
Cycle?



14

Terminology

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



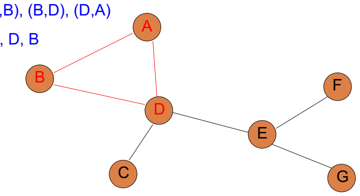
15

Terminology

Cycle – A subset of the edges that form a path where each edge is traversed once such that the first and last node are the same

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

Path: B, A, D, B

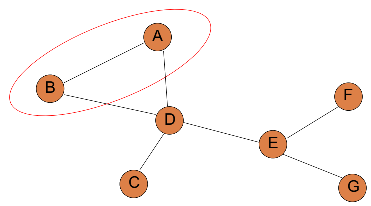


16

Terminology

Cycle – A subset of the edges that form a path where each edge is traversed once such that the first and last node are the same

cycle?

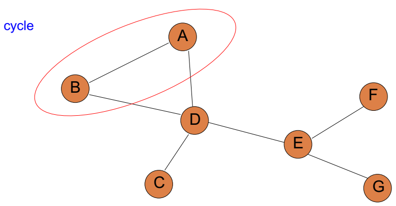


17

Terminology

Cycle – A subset of the edges that form a path where each edge is traversed once such that the first and last node are the same

not a cycle

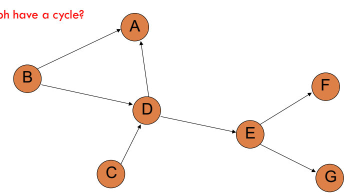


18

Terminology

Cycle – A subset of the edges that form a path where each edge is traversed once such that the first and last node are the same

Does this graph have a cycle?

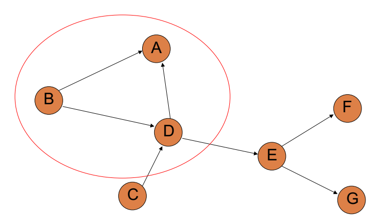


19

Terminology

Cycle – A subset of the edges that form a path where each edge is traversed once such that the first and last node are the same

not a cycle

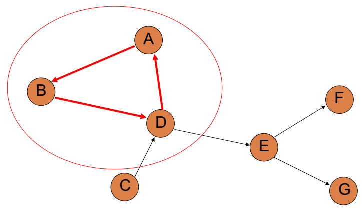


20

Terminology

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

cycle

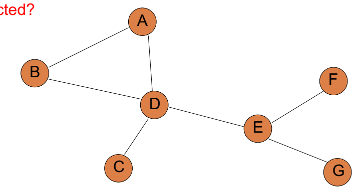


21

Terminology

Connected – every pair of vertices is connected by a path

Is this graph connected?

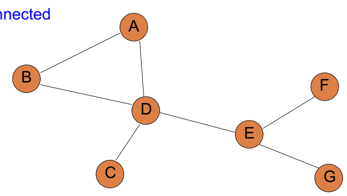


22

Terminology

Connected – every pair of vertices is connected by a path

connected

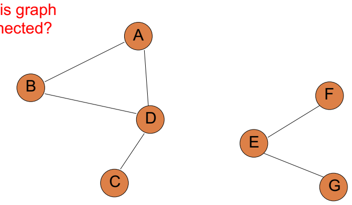


23

Terminology

Connected – every pair of vertices is connected by a path

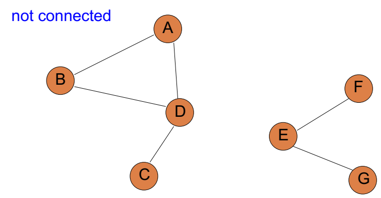
Is this graph connected?



24

Terminology

Connected – every pair of vertices is connected by a path

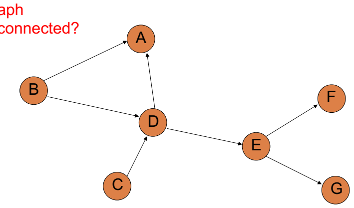


25

Terminology

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

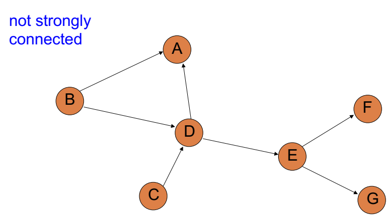
Is this graph
strongly connected?



26

Terminology

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

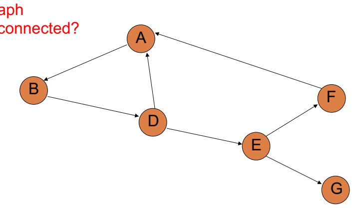


27

Terminology

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

Is this graph
strongly connected?

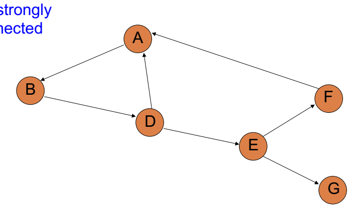


28

Terminology

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

not strongly
connected

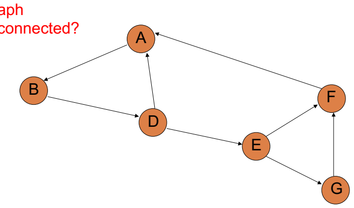


29

Terminology

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

Is this graph
strongly connected?

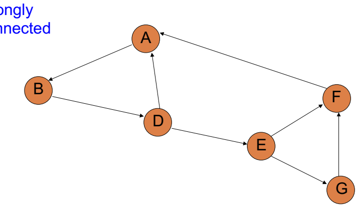


30

Terminology

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

strongly
connected

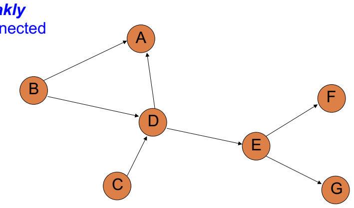


31

Terminology

Weakly connected (directed graphs) –
graph is connected when considered as undirected graph

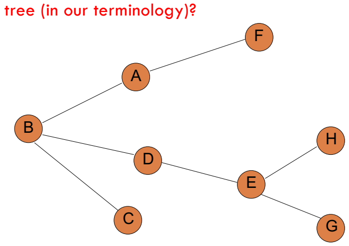
weakly
connected



32

Different types of graphs

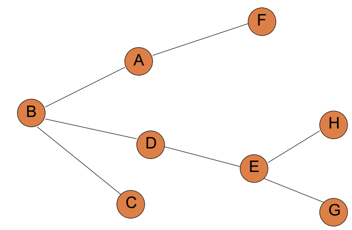
What is a tree (in our terminology)?



33

Different types of graphs

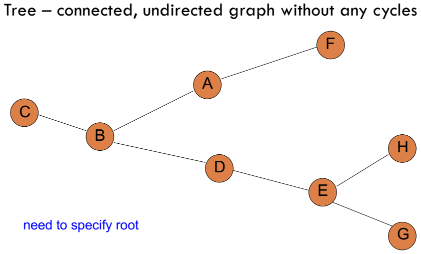
Tree – connected, undirected graph without any cycles



34

Different types of graphs

Tree – connected, undirected graph without any cycles

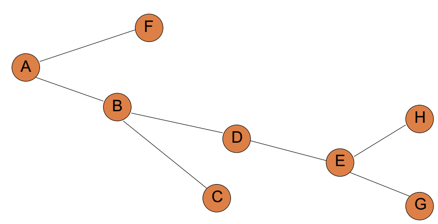


need to specify root

35

Different types of graphs

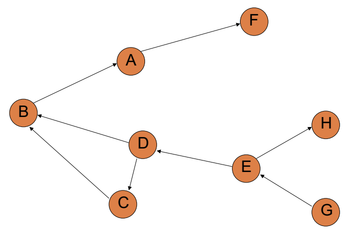
Tree – connected, undirected graph without any cycles



36

Different types of graphs

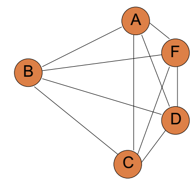
DAG – directed, acyclic graph



37

Different types of graphs

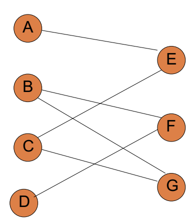
Complete graph – an edge exists between every node



38

Different types of graphs

Bipartite graph – a graph where every vertex can be partitioned into two sets X and Y such that all edges connect a vertex $u \in X$ and a vertex $v \in Y$



39

When do we see graphs in real life problems?

Transportation networks (flights, roads, etc.)

Communication networks

Web

Social networks

Circuit design

Bayesian networks

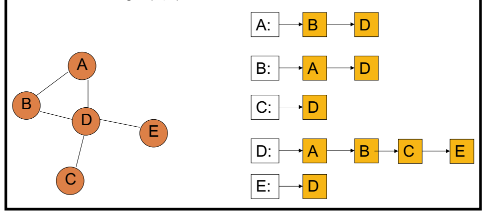
40

Representing graphs

41

Representing graphs

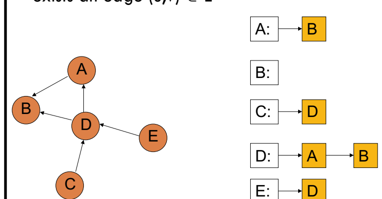
Adjacency list – Each vertex u ∈ V contains an adjacency list of the set of vertices v such that there exists an edge (u,v) ∈ E



42

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

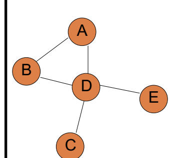


43

Representing graphs

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

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



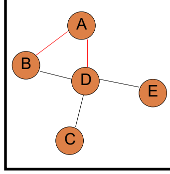
	A	**B**	**C**	**D**	**E**
A	0	1	0	1	0
B	1	0	0	1	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

44

Representing graphs

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

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



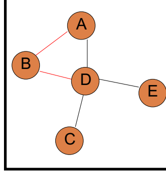
	A	B	C	D	E
A	0	**1**	0	**1**	0
B	1	0	0	1	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

45

Representing graphs

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

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



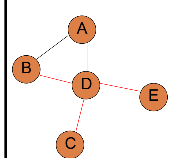
	A	B	C	D	E
A	0	1	0	1	0
B	**1**	0	0	**1**	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

46

Representing graphs

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

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



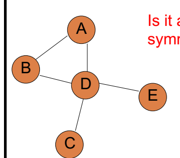
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	0	1	0
C	0	0	0	1	0
D	**1**	**1**	**1**	0	**1**
E	0	0	0	1	0

47

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?

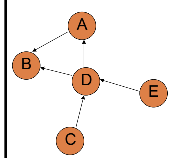
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	0	1	0
C	0	0	0	1	0
D	1	1	1	0	1
E	0	0	0	1	0

48

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

49

Adjacency list vs. adjacency matrix

Adjacency list

Adjacency matrix

Pros and cons?

50

Adjacency list vs. adjacency matrix

Adjacency list

- Sparse graphs (e.g. web)
- Space efficient
- Must traverse the adjacency list to discover is an edge exists

Adjacency matrix

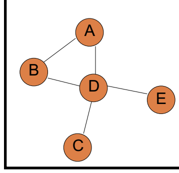
- Dense graphs
- Constant time lookup to discover if an edge exists
- Simple to implement
- For non-weighted graphs, only requires boolean matrix

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]

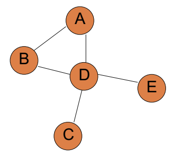
52

Sparse adjacency matrix

Constant time lookup

Fairly space efficient

Not good for dense graphs, why?

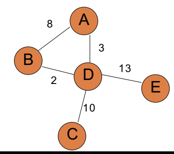


A:	hashtable [B,D]
B:	hashtable [A,D]
C:	hashtable [D]
D:	hashtable [A,B,C,E]
E:	hashtable [D]

53

Weighted graphs

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

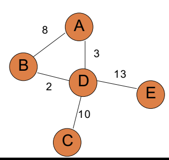


54

Weighted graphs

Adjacency matrix

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



	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?

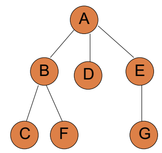
Is the graph bipartite?

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)
Enqueue(Q, Root(T))
while !Empty(Q)
        v ← Dequeue(Q)
        Visit(v)
        for all c ∈ Children(v)
                Enqueue(Q, c)
```

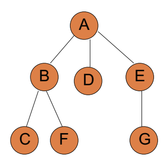
```
TreeDFS(T)
Push(S, Root(T))
while !Empty(S)
        v ← Pop(S)
        Visit(v)
        for all c ∈ Children(v)
                Push(S, c)
```

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

58

BFS

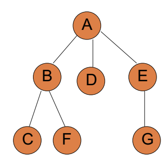
```
TreeBFS(T)
Enqueue(Q, Root(T))
while !Empty(Q)
        v ← Dequeue(Q)
        Visit(v)
        for all c ∈ Children(v)
                Enqueue(Q, c)
```



59

BFS

```
TreeBFS(T)
Enqueue(Q, Root(T))
while !Empty(Q)
        v ← Dequeue(Q)
        Visit(v)
        for all c ∈ Children(v)
                Enqueue(Q, c)
```

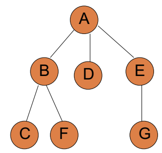


A B D E C F G

60

DFS

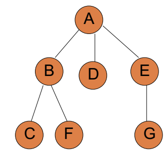
```
TreeDFS(T)
  Push(S, Root(T))
  while !Empty(S)
      v ← Pop(S)
      Visit(v)
      for all c ∈ Children(v)
          Push(S, c)
```



61

DFS

```
TreeDFS(T)
  Push(S, Root(T))
  while !Empty(S)
      v ← Pop(S)
      Visit(v)
      for all c ∈ Children(v)
          Push(S, c)
```

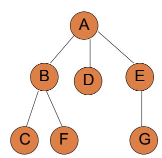


A E G D B F C

62

DFS

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

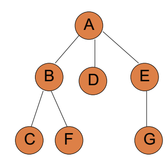


What changes?

63

DFS

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



A B C F D E G

64

Running time of BFS/DFS

Adjacency list
- How many times does it visit each vertex?
- How many times is each edge traversed?
- θ(|V|+|E|) – for trees, assuming a connected graph

Adjacency matrix
- For each vertex visited, how much work is done?
- θ(|V|^2) – for trees, assuming a connected graph

```
TreeBFS(T)
Enqueue(Q, Root(T))
while !Empty(Q)
    v ← Dequeue(Q)
    Visit(v)
    for all c ∈ Children(v)
        Enqueue(Q, c)
```

```
TreeDFS(T)
Push(S, Root(T))
while !Empty(S)
    v ← Pop(S)
    Visit(v)
    for all c ∈ Children(v)
        Push(S, c)
```

65

DFS/BFS

Do they visit all of the nodes?

If the graph is connected or strongly connected


```
TreeBFS(T)
Enqueue(Q, Root(T))
while !Empty(Q)
    v ← Dequeue(Q)
    Visit(v)
    for all c ∈ Children(v)
        Enqueue(Q, c)
```

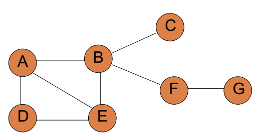
```
TreeDFS(T)
Push(S, Root(T))
while !Empty(S)
    v ← Pop(S)
    Visit(v)
    for all c ∈ Children(v)
        Push(S, c)
```


66

DFS/BFS for graphs

What needs to change for graphs?

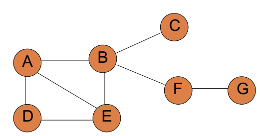
Need to make sure we don't visit a node multiple times



67

BFS for graphs

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

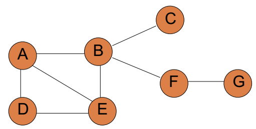


68

BFS for graphs

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

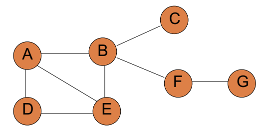
A B D E C F G



69

```
BFS(G, s)
for each v ∈ V
        dist[v] = ∞
dist[s] = 0
Enqueue(Q, s)
while !Empty(Q)
        u ← Dequeue(Q)
        Visit(u)
        for each edge (u, v) ∈ E
                if dist[v] = ∞
                        Enqueue(Q, v)
                        dist[v] ← dist[u] + 1
```

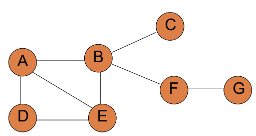
distance variable keeps track of how far from the starting node and whether we've seen the node yet



70

```
BFS(G, s)
for each v ∈ V
        dist[v] = ∞
dist[s] = 0
Enqueue(Q, s)
while !Empty(Q)
        u ← Dequeue(Q)
        Visit(u)
        for each edge (u, v) ∈ E
                if dist[v] = ∞
                        Enqueue(Q, v)
                        dist[v] ← dist[u] + 1
```

```
TreeBFS(T)
Enqueue(Q, Root(T))
while !Empty(Q)
        v ← Dequeue(Q)
        Visit(v)
        for all c ∈ Children(v)
                Enqueue(Q, c)
```



71

DFS on graphs

```
DFS(G)
for all v ∈ V
        visited[v] ← false
for all v ∈ V
        if !visited[v]
                DFS-Visit(v)

DFS-Visit(u)
visited[u] ← true
PreVisit(u)
for all edges (u, v) ∈ E
        if !visited[v]
                DFS-Visit(v)
PostVisit(u)
```

72

DFS on graphs

```
DFS(G)
for all v ∈ V
    visited[u] ← false
for all v ∈ V
    if !visited[v]
        DFS-Visit(v)

DFS-Visit(u)
visited[u] ← true
PreVisit(u)
for all edges (u, v) ∈ E
    if !visited[v]
        DFS-Visit(v)
PostVisit(u)
```

mark all nodes as not visited

73

DFS on graphs

```
DFS(G)
for all v ∈ V
    visited[u] ← false
for all v ∈ V
    if !visited[v]
        DFS-Visit(v)

DFS-Visit(u)
visited[u] ← true
PreVisit(u)
for all edges (u, v) ∈ E
    if !visited[v]
        DFS-Visit(v)
PostVisit(u)
```

until **all** nodes have been visited repeatedly call DFS-Visit

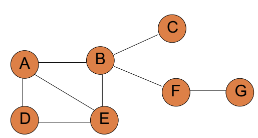
74

DFS for graphs

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

```
DFS(G)
for all v ∈ V
    visited[u] ← false
for all v ∈ V
    if !visited[v]
        DFS-Visit(v)

DFS-Visit(u)
visited[u] ← true
PreVisit(u)
for all edges (u, v) ∈ E
    if !visited[v]
        DFS-Visit(v)
PostVisit(u)
```



75

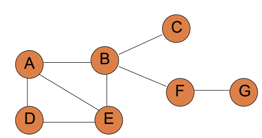
DFS for graphs

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

```
DFS(G)
for all v ∈ V
    visited[u] ← false
for all v ∈ V
    if !visited[v]
        DFS-Visit(v)

DFS-Visit(u)
visited[u] ← true
PreVisit(u)
for all edges (u, v) ∈ E
    if !visited[v]
        DFS-Visit(v)
PostVisit(u)
```

A B C E D F G



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

Connectedness

Given an undirected graph, for every node $u \in V$, can we reach all other nodes in the graph?
Algorithm + running time

Run BFS or DFS-Visit (one pass) and mark nodes as we visit them. If we visit all nodes, return true, otherwise false.

Running time: $O(|V| + |E|)$

79

Strongly connected

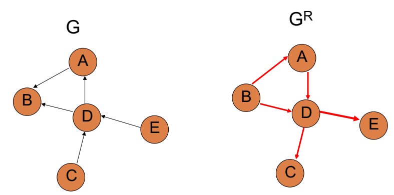
Given a directed graph, can we reach any node v from any other node u?

Can we do the same thing?

80

Transpose of a graph

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



Running time to calculate G^R? $\theta(|V| + |E|)$

81

Strongly connected

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

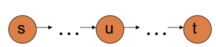
82

Is it correct?

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

What do we know after the second pass?
- 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



83

Runtime?

Strongly-Connected(G)	
- Run DFS-Visit or BFS from some node u	$O(\|V\| + \|E\|)$
- If not all nodes are visited: return false	$O(\|V\|)$
- Create graph G^R	$\theta(\|V\| + \|E\|)$
- Run DFS-Visit or BFS on G^R from node u	$O(\|V\| + \|E\|)$
- If not all nodes are visited: return false	$O(\|V\|)$
- return true	

$O(|V| + |E|)$

84

Handout

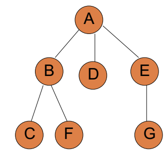
85

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.

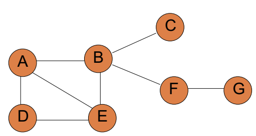
Run-time (in terms of V and E):
- adjacency list
- adjacency matrix



86

BFS/DFS for graphs

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



87