

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS, AND OTHER "NOT COMPUTER PEOPLE":

WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:

```

graph TD
    START([START]) --> D1{FIND A MENU ITEM OR  
BUTTON WHICH LOOKS  
RELATED TO WHAT  
YOU WANT TO DO.}
    D1 -- "I CAN'T  
FIND ONE" --> D2{PICK ONE  
AT RANDOM.}
    D1 -- "OK" --> C1[CLICK IT!]
    D2 -- "OK" --> C1
    D2 -- "I'VE TRIED  
THEM ALL" --> R1[GOOGLE THE NAME  
OF THE PROGRAM  
PLUS A FEW WORDS,  
RELATED TO WHAT YOU  
WANT TO DO. FOLLOW  
ANY INSTRUCTIONS.]
    R1 --> D3{CLICK IT?}
    C1 --> D3
    D3 -- "NO" --> D4{HAVE YOU BEEN  
TRYING THIS FOR  
OVER HALF AN  
HOUR?}
    D3 -- "YES" --> C2[YOU'RE  
DONE!]
    D4 -- "YES" --> C3[ASK SOMEONE  
FOR HELP  
OR GIVE UP!]
    D4 -- "NO" --> D3
  
```


PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN. CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

this is a flow chart, but it is similar to a graph

Announcements

- Huffman – Due April 16
 - You may have a partner
- Burrows-Wheeler – Due April 18
 - You may use the same partner
- Family Tree


2



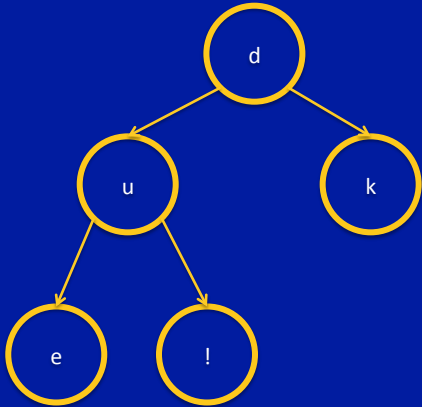
Today

- Intro to graphs
- Coding with graphs

3

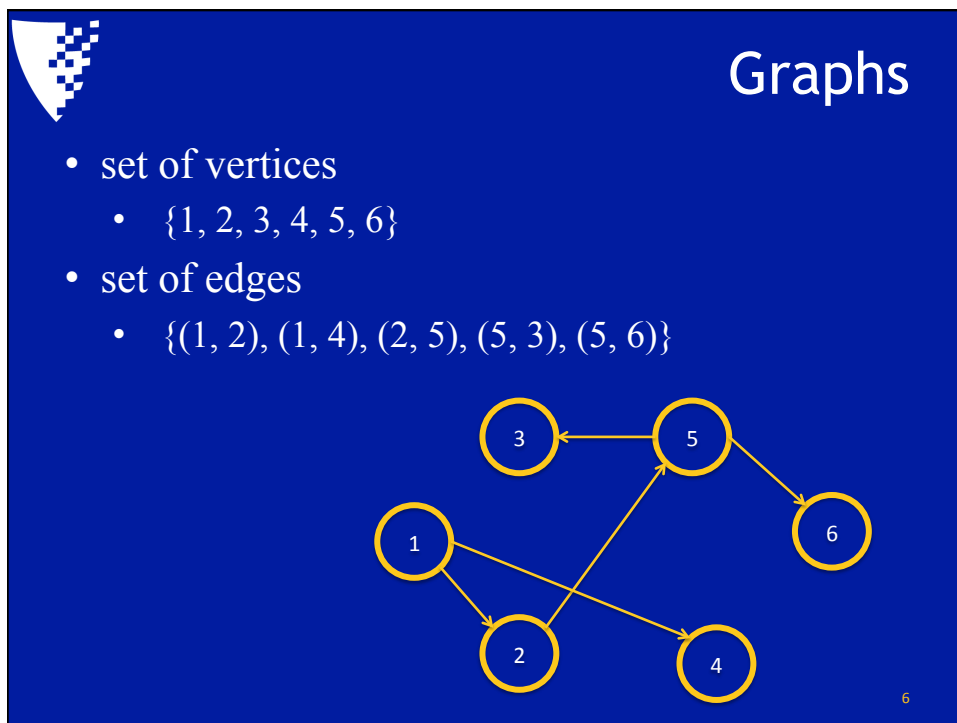
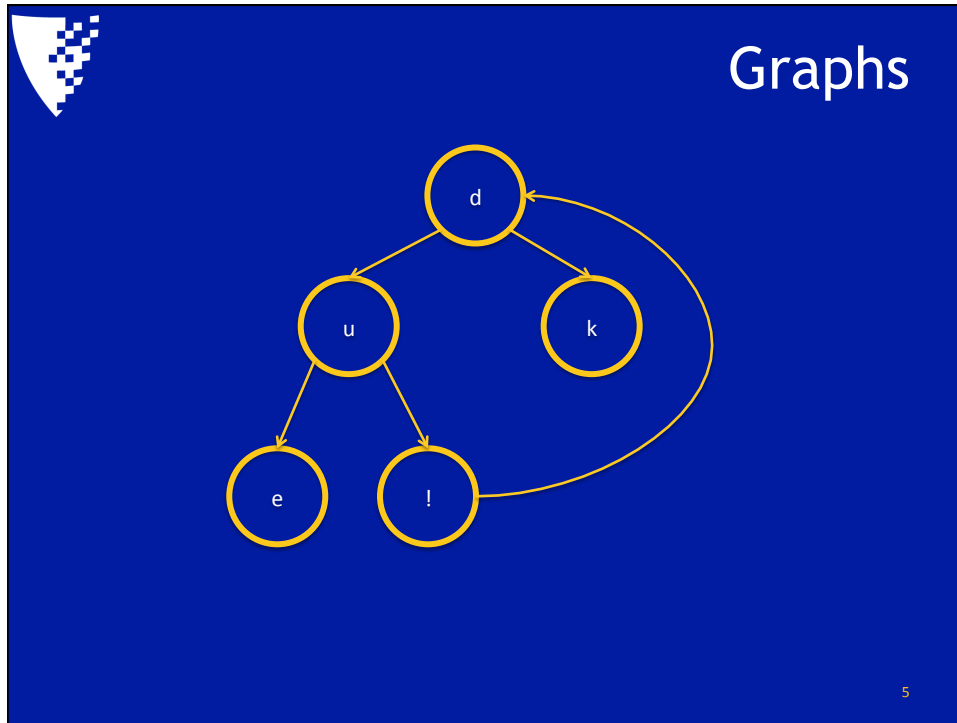


Trees



```
graph TD; d((d)) --> u((u)); d --> k((k)); u --> e((e)); u --> exclamation((!));
```

4



Graphs

- directed graphs* – edge sets are ordered
 - $\{(1, 2), (1, 4), (2, 5), (5, 3), (5, 6)\}$
 - 1 points to 2 – notice the arrow
 - $(2, 1)$ is not an edge

*a.k.a. digraphs

```

graph TD
  1((1)) --> 2((2))
  1((1)) --> 4((4))
  2((2)) --> 5((5))
  4((4)) --> 5((5))
  5((5)) --> 3((3))
  5((5)) --> 6((6))
  
```

7

Graphs

- undirected graphs – edge sets are not ordered
 - $\{(1, 2), (1, 4), (2, 5), (5, 3), (5, 6)\}$
 - $(1, 2)$ is the same as $(2, 1)$

```

graph TD
  1((1)) --- 2((2))
  1((1)) --- 4((4))
  2((2)) --- 5((5))
  4((4)) --- 5((5))
  5((5)) --- 3((3))
  5((5)) --- 6((6))
  
```

8

Graphs

- edges can have weights

A weighted undirected graph with 6 nodes labeled 1 through 6. The nodes are arranged in a roughly circular pattern. The edges and their weights are: (1,2) with weight 3, (1,4) with weight 7, (2,4) with weight 7, (3,5) with weight 4, (4,5) with weight 6, and (5,6) with weight 2.

9

Graphs

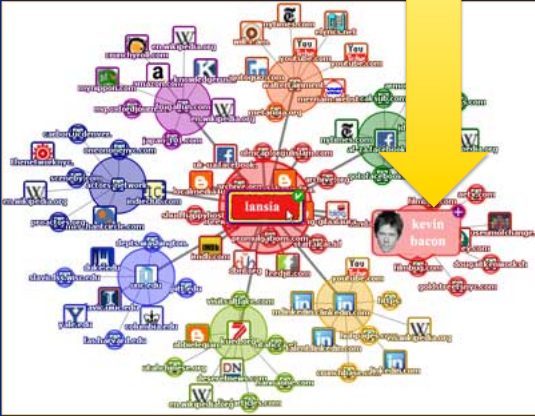
- Why do you care?

A social network graph visualization for Tabitha Peck. The graph is circular, with nodes representing other users and edges representing connections. The nodes are colored in a gradient from blue to red. The name "Tabitha Peck" is displayed in the center of the graph. The Facebook logo is visible in the top right corner of the image.

10

Graphs

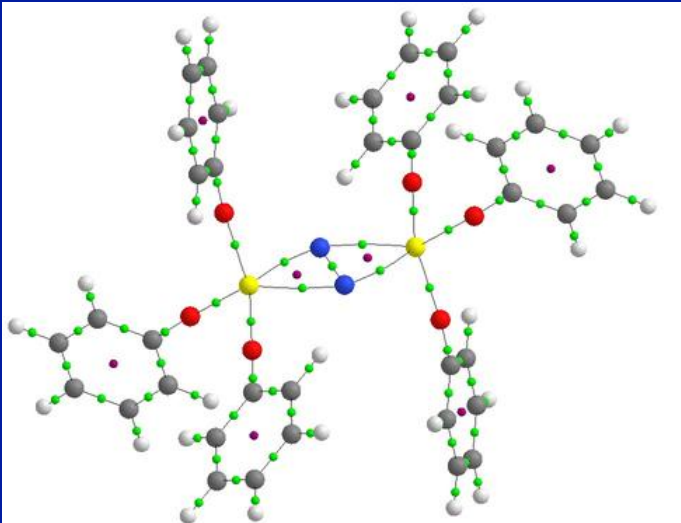
- Kevin Bacon



A complex social network graph where nodes represent individuals and edges represent relationships. The nodes are represented by various icons, including social media logos and profile pictures. Two nodes are highlighted: 'Kevin Bacon' (with a portrait) and 'Innsia' (with a red background). A large yellow arrow points from the top right towards the 'Kevin Bacon' node.

11

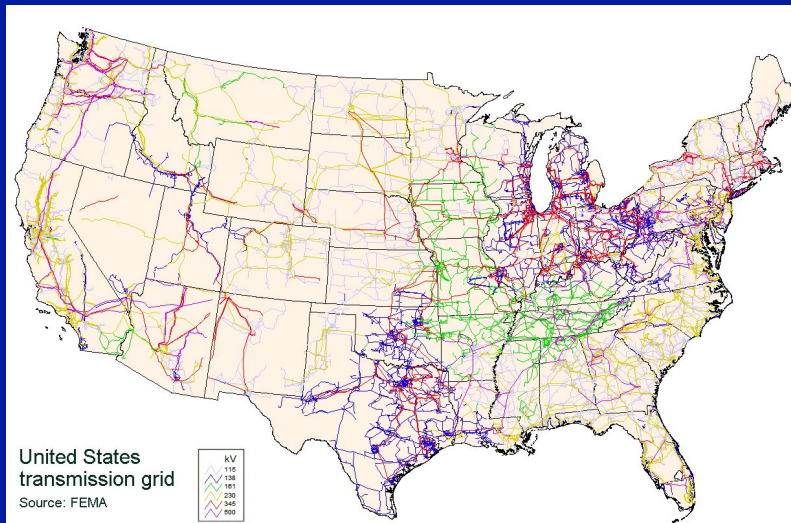
Graphs



A 3D ball-and-stick model of a complex organic molecule. The atoms are represented by spheres of different colors: carbon (grey), oxygen (red), nitrogen (blue), and sulfur (yellow). The bonds are shown as sticks connecting the spheres, illustrating the spatial arrangement of the molecule.

12

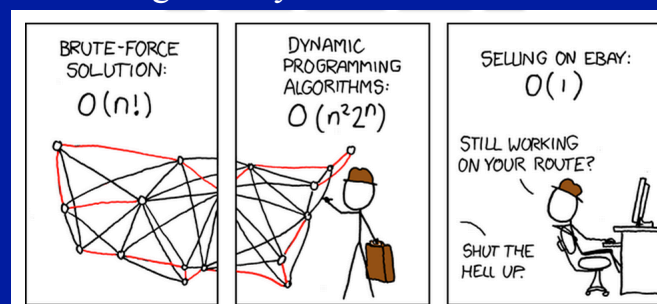
Graphs



13

Graphs

- Traveling salesperson problem
 - Given a list of cities and the distance between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the original city?



14

Graphs

- Depth-first-search
 - explore as far as possible before backtracking

Start at root

```
dfs(vertex)
  if(visited vertex) return;

  visit vertex

  for(adjacent vertices to vertex)
    dfs(adjacent vertex)
```

```

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

15

Graphs

- Depth-first-search
 - explore as far as possible before backtracking

Start at root

```
dfs(vertex)
  if(visited vertex) return;

  visit vertex

  for(adjacent vertices to vertex)
    dfs(adjacent vertex)
```

```

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

A

16

Graphs

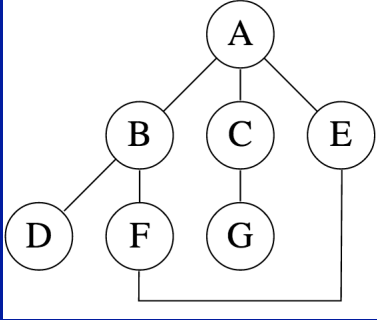
- Depth-first-search
 - explore as far as possible before backtracking

Start at root

```
dfs(vertex)
  if(visited vertex) return;

  visit vertex

  for(adjacent vertices to vertex)
    dfs(adjacent vertex)
```



A B

17

Graphs

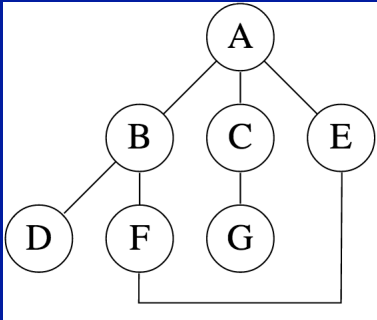
- Depth-first-search
 - explore as far as possible before backtracking

Start at root

```
dfs(vertex)
  if(visited vertex) return;

  visit vertex

  for(adjacent vertices to vertex)
    dfs(adjacent vertex)
```



A B D

18

Graphs

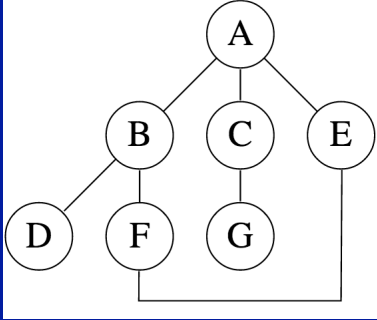
- Depth-first-search
 - explore as far as possible before backtracking

Start at root

```
dfs(vertex)
  if(visited vertex) return;

  visit vertex

  for(adjacent vertices to vertex)
    dfs(adjacent vertex)
```



A B D F

19

Graphs

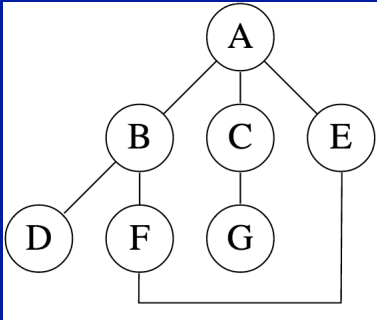
- Depth-first-search
 - explore as far as possible before backtracking

Start at root

```
dfs(vertex)
  if(visited vertex) return;

  visit vertex

  for(adjacent vertices to vertex)
    dfs(adjacent vertex)
```



A B D F E

20

Graphs

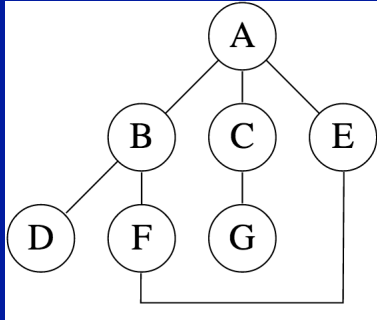
- Depth-first-search
 - explore as far as possible before backtracking

Start at root

```
dfs(vertex)
  if(visited vertex) return;

  visit vertex

  for(adjacent vertices to vertex)
    dfs(adjacent vertex)
```



A B D F E C

21

Graphs

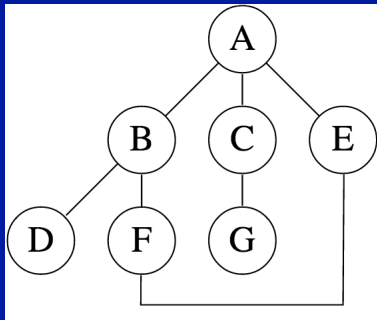
- Depth-first-search
 - explore as far as possible before backtracking

Start at root

```
dfs(vertex)
  if(visited vertex) return;

  visit vertex

  for(adjacent vertices to vertex)
    dfs(adjacent vertex)
```



A B D F E C G

22

Graphs

- Breadth-first-search
 - explore as far as possible before backtracking

Start at root

```

bfs(vertex)
  myQ.enqueue(vertex)

while(!myQ.isEmpty())
  v = myQ.dequeue
  for(adj vertices of v)
    if(adj not visited)
      myQ.enqueue(adj)
  
```

```

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

23

Graphs

- Breadth-first-search
 - explore as far as possible before backtracking

Start at root

```

bfs(root)
  myQ.enqueue(root)

while(!myQ.isEmpty())
  v = myQ.dequeue
  for(adj vertices of v)
    if(adj not visited)
      myQ.enqueue(adj)
  
```

A

```

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

24

Graphs

- Breadth-first-search
 - explore as far as possible before backtracking

Start at root

```

bfs(root)
  myQ.enqueue(root)

while(!myQ.isEmpty())
  v = myQ.dequeue
  for(adj vertices of v)
    if(adj not visited)
      myQ.enqueue(adj)
  
```

A B C E

```

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

25

Graphs

- Breadth-first-search
 - explore as far as possible before backtracking

Start at root

```

bfs(root)
  myQ.enqueue(root)

while(!myQ.isEmpty())
  v = myQ.dequeue
  for(adj vertices of v)
    if(adj not visited)
      myQ.enqueue(adj)
  
```

A B C E D F

```

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

26

Graphs

- Breadth-first-search
 - explore as far as possible before backtracking

Start at root

```

bfs(root)
  myQ.enqueue(root)

while(!myQ.isEmpty())
  v = myQ.dequeue
  for(adj vertices of v)
    if(adj not visited)
      myQ.enqueue(adj)
  
```

A B C E D F G

```

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

27

Code time

- snarf today's code
 - this will be helpful for APT set 7

28



Before you go

- How are things going?
- <http://goo.gl/CAZEB>