GRAPHS: SHORTEST PATHS

David Kauchak
CS 62 – Spring 2021

1

Admin

TextGenerator assignment

Lab tomorrow

2

Graph code

https://github.com/pomonacs622021sp/LectureCode/tree/master/GraphExamples

3

Searching on graphs

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

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

4

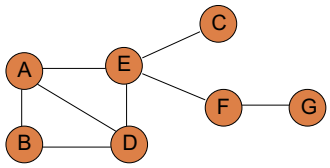
Searching on graphs

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

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

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

5

DFS

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



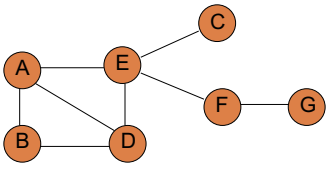
toVisit-stack: A

visited:

6

DFS

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



toVisit-stack: A

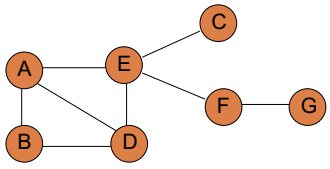
visited:

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

7

DFS

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



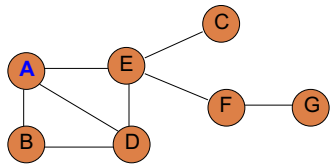
toVisit-stack: A

visited:

8

DFS

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



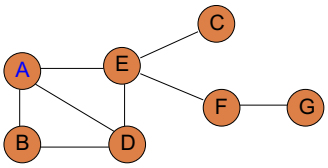
toVisit-stack: B D E

visited: A

9

DFS

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



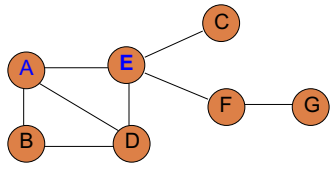
toVisit-stack: B D E

visited: A

10

DFS

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



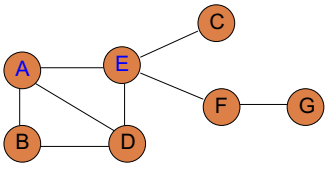
toVisit-stack: B D C D F

visited: A E

11

DFS

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



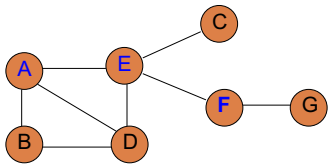
toVisit-stack: B D C D F

visited: A E

12

DFS

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



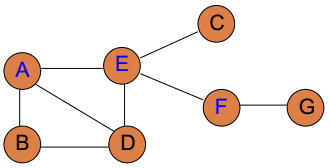
toVisit-stack: B D C D G

visited: A E F

13

DFS

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



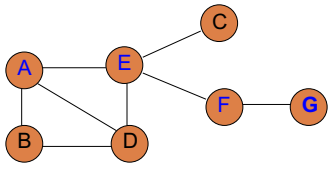
toVisit-stack: B D C D G

visited: A E F

14

DFS

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



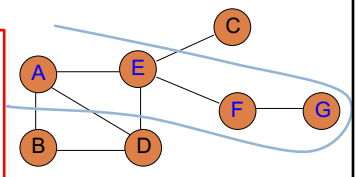
toVisit-stack: B D C D

visited: A E F G

15

DFS

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



toVisit-stack: B D C D

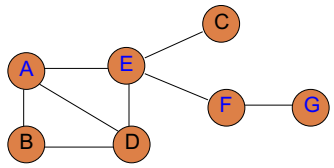
visited: A E F G

Frontier: Go as far down one path as possible

16

DFS

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



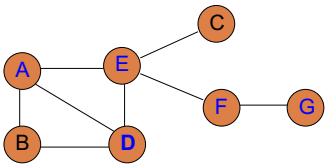
toVisit-stack: B D C D

visited: A E F G

17

DFS

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



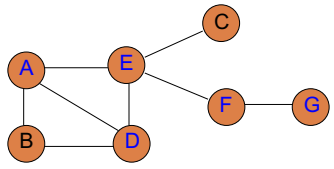
toVisit-stack: B D C B

visited: A E F G D

18

DFS

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



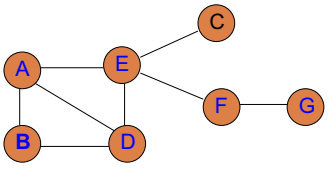
toVisit-stack: B D C B

visited: A E F G D

19

DFS

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



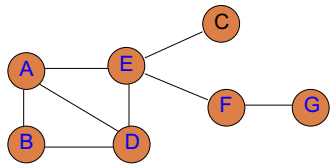
toVisit-stack: B D C

visited: A E F G D B

20

DFS

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



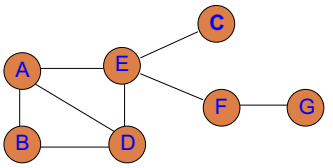
toVisit-stack: B D C

visited: A E F G D B

21

DFS

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



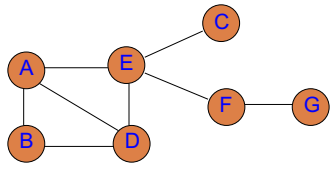
toVisit-stack: B D

visited: A E F G D B C

22

DFS

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



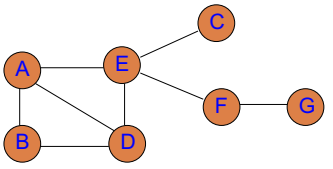
toVisit-stack: B D

visited: A E F G D B C

23

DFS

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



toVisit-stack:

visited: A E F G D B C

24

graphSearch run-time

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

What is the big-O run-time of graphSearch?

Assume all of the stack/queue operations are constant.

How many times do we visit each vertex?

25

graphSearch run-time

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

How many times do we visit each vertex? Exactly once

26

graphSearch run-time

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

What is the cost to traverse all of the edges adjacent to a vertext?

27

graphSearch run-time

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

What is the cost to traverse all of the edges adjacent to a vertext?

Depends on the graph representation!

28

graphSearch run-time

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

What is the cost to traverse all of the edges adjacent to a vertext?

Depends on the graph representation!

Adjacency matrix?

Adjacency list?

29

graphSearch run-time

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

What is the cost to traverse all of the edges adjacent to a vertext?

Depends on the graph representation!

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

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

30

graphSearch run-time

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

Overall runtime: Depends on the graph representation!

Adjacency matrix? $O(V^2)$

Adjacency list? O(V+E)

31

graphSearch run-time

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

How many times do we visit each vertex? Exactly once

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

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

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

32

Search

BFS: breadth first search
- Explores vertices in increasing distance (wrt number of edges) from the starting vertex
- Uses a queue to keep track of vertices to explore

DFS: depth first search:
- Goes as far down a path first and then works its way back
- Two versions: stack and recursive version

Run-time: O(V + E) – assuming adjacency list representation

33

BFS and DFS in java

https://github.com/pomonacs622021sp/LectureCode/tree/master/GraphExamples

Look at Graph interface

Look at AdjacencyGraph and MatrixGraph

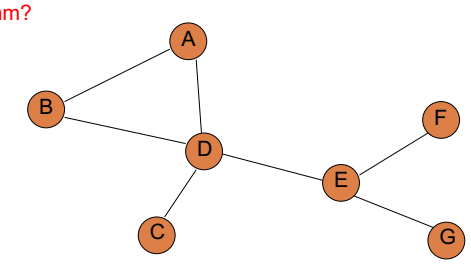
Look at BFS and DFS in Search code

34

Connectedness

Connected – every pair of vertices is connected by a path

Algorithm?



35

Connectedness

Connected – every pair of vertices is connected by a path

Pick any starting vertex u
Run DFS/BFS from u

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

return true

Why does this work?

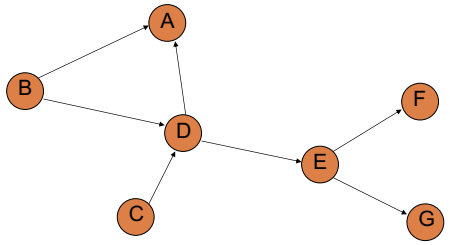
If we can get from u to every vertex then we know a path exists between all vertices.

Path from a to b: a – u – b

36

Strongly connected

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



37

Strongly connected

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

Pick any starting vertex u
Run DFS/BFS from u

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

return true

Does this work?

38

Strongly connected

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

Pick any starting vertex u
Run DFS/BFS from u

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

return true

Does this work?

No!

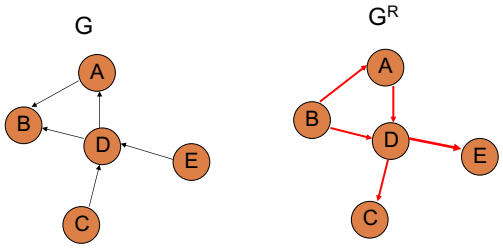
Path from a to b: a – u – b

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

39

Reverse of a graph

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



40

Strongly connected

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

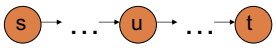
41

Is it correct?

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

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

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



42

Run-times?

Connectedness

Pick any starting vertex u
Run DFS/BFS from u

What is the run-time?

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

return true

43