Introduction to Graphs

http://people.cs.clemson.edu/~pargas/courses/cs212/common/notes/ppt/

Introduction

- Graphs are a generalization of trees
  - Nodes or verticies
  - Edges or arcs
- Two kinds of graphs
  - Directed
  - Undirected

Introduction: Formal Definition

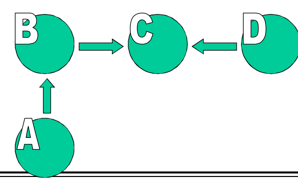
- A graph G = (V,E) consists of a finite set of vertices, V, and a finite set of edges E.
- Each edge is a pair (v,w) where $v, w \in V$

Introduction: Formal Definition

- A **directed** graph, or **digraph**, is a graph in which the edges are ordered pairs
  - $(v, w) \neq (w, v)$
- An **undirected** graph is a graph in which the edges are unordered pairs
  - (v, w) == (w, v)

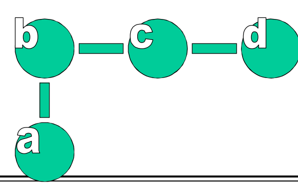
Introduction: Directed Graphs

- In a directed graph, the edges are arrows.
- Directed graphs show the flow from one node to another and not vise versa.

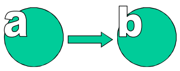


Introduction: Undirected Graphs

- In a directed graph, the edges are lines.
- Directed graphs show a relationship between two nodes.

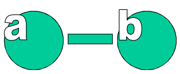


Terminology



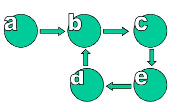
- In the directed graph above, b is **adjacent** to a because (a, b) $\in$ E. Note that a is *not* adjacent to b.
- A is a **predecessor** of node B
- B is a **successor** of node A
- The **source** of the edge is node A, the **target** is node B

Terminology



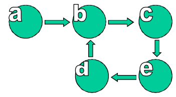
- In the undirected graph above, a and b are **adjacent** because (a,b) $\in$ E. a and b are called **neighbors.**

Terminology



- A **path** is a sequence of vertices $w_1, w_2, \ldots w_n$ such that $(w_i, w_{i+1}) \in E$, $1 <= i < n$, and each vertex is unique except that the path may start and end on the same vertex
- The **length** of the path is the number of edges along the path

Terminology

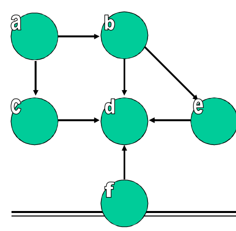


- An **acyclic path** is a path where each vertex is unique
- A **cyclic path** is a path such that
  - There are at least two vertices on the path
  - $w_1 = w_n$ (path starts and ends at same vertex)

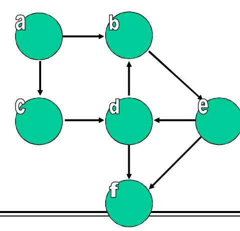
Test Your Knowledge

Cyclic or Acyclic?

1.



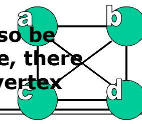
2.



Terminology

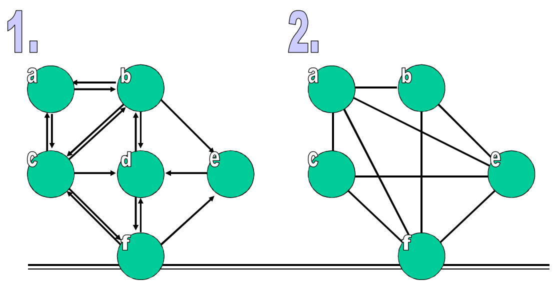
- A directed graph that has *no* cyclic paths is called a **DAG** (a Directed Acyclic Graph).
- An undirected graph that has an edge between every pair of vertices is called a **complete** graph.

Note: A directed graph can also be a complete graph; in that case, there must be an edge from every vertex to every other vertex.



Test Your Knowledge

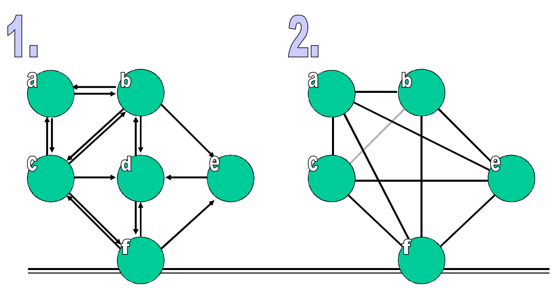
Complete, or "Acomplete" (Not Complete)



Slide 13

Test Your Knowledge

Complete, or "Acomplete" (Not Complete)



Slide 14

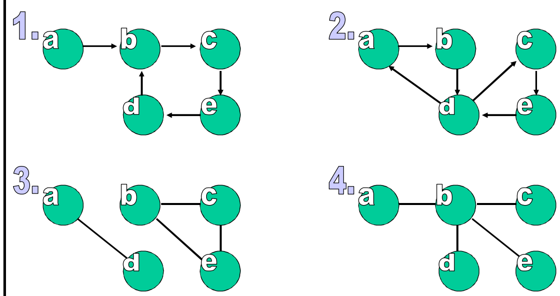
Terminology

- An undirected graph is **connected** if a path exists from every vertex to every other vertex
- A directed graph is **strongly connected** if a path exists from every vertex to every other vertex
- A directed graph is **weakly connected** if a path exists from every vertex to every other vertex, disregarding the direction of the edge

Slide 15

Test Your Knowledge

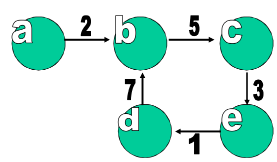
Connected, Strongly connected, or Weakly connected



Slide 16

Terminology

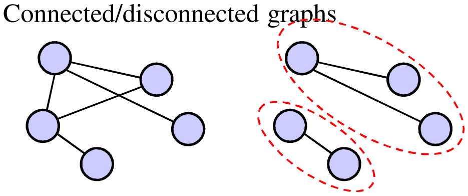
- A graph is known as a **weighted graph** if a weight or metric is associated with each edge.



Slide 17

Various types of graphs

- Connected/disconnected graphs

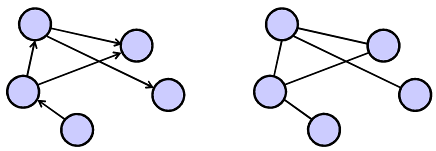


- The circled subgraphs are also known as connected components

Slide 18

Various types of graphs

- Directed/undirected graphs

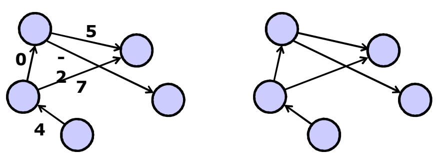


- You may treat each undirected edge as two directed edges in opposite directions

Slide 19

Various types of graphs

- Weighted/unweighted graphs

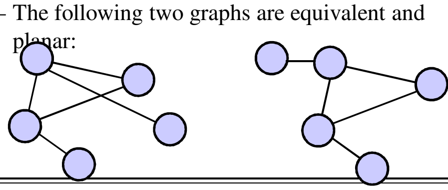


- You may treat unweighted edges to be weighted edges of equal weights

Slide 20

Special graphs

- Planar graphs
  - A graph that can be drawn on a plane without edge intersections
  - The following two graphs are equivalent and planar:

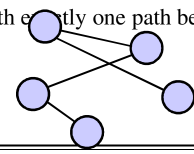


- To be discussed in details in Graph (III) ☺

Slide 21

Special graphs

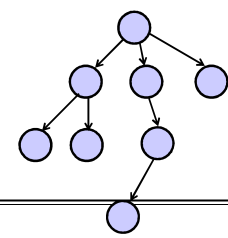
- Tree: either one of the followings is the definition
  - A connected graph with |V|-1 edges
  - A connected graph without cycles
  - A graph with exactly one path between every pair of vertices



Slide 22

Special graphs

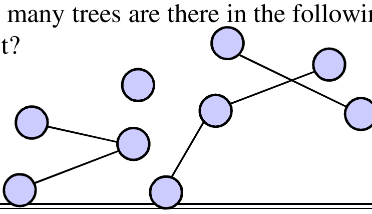
- Tree edges could be directed or undirected
- For trees with directed edges, a root usually exists



Slide 23

Special graphs

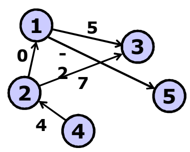
- Forest
  - All connected component(s) is/are tree(s)
- How many trees are there in the following forest?



Slide 24

How to store graphs in the program?

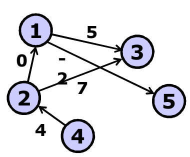
- Usually, the vertices are labeled beforehand
- 3 types of graph representations:
  - Adjacency matrix
  - Adjacency list
  - Edge list



Adjacency matrix

- Use a 2D array

s\t	1	2	3	4	5
1			5		-2
2	0		7		
3					
4		4			
5					

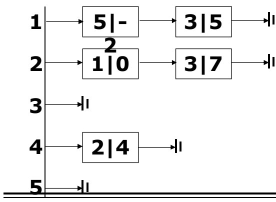


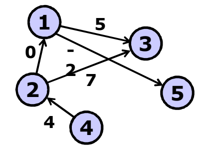
Adjacency matrix

- Memory complexity?
- Time complexity for:
  - Checking the weight of an edge between 2 given nodes?
  - Querying all adjacent nodes of a given node?

Adjacency list

- N vertices, N linked lists
- Each list stores its adjacent vertices





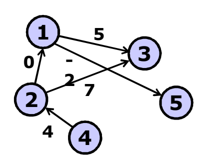
Adjacency list

- Memory complexity?
- Time complexity for:
  - Checking the weight of an edge between 2 given nodes?
  - Querying all adjacent nodes of a given node?

Edge list

- A list of edges

id	x	y	w
0	1	5	-2
1	2	1	0
2	1	3	5
3	2	3	7
4	4	2	4



Edge list

- Memory complexity?
- Time complexity for:
  - Checking the weight of an edge between 2 given nodes?
  - Querying all adjacent nodes of a given node?

Slide 31

Which one should be used?

- It depends on:
  - Constraints
  - Time Limit
  - Memory Limit
  - What algorithm is used

Slide 32

Uses for Graphs

- **Precedence Constraints:** Suppose you have a set of jobs to complete, but some must be completed before others are begun. (For example, Atilla advises you always pillage before you burn.) Here the vertices are jobs to be done. Directed edges indicate constraints; there is a directed edge from job u to job v if job u must be done before job v is begun.

Slide 33

Uses for Graphs

- **Two-Player Game Tree:** All of the possibilities in a board game like chess can be represented in a graph. Each vertex stands for one possible board position. (For chess, this is a very big graph!)

Slide 34

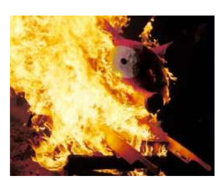
Uses for Graphs

- **Computer network:** The set of vertices V represents the set of computers in the network. There is an edge (u, v) if and only if there is a direct communication link between the computers corresponding to u and v.

Slide 35

Topological Sort

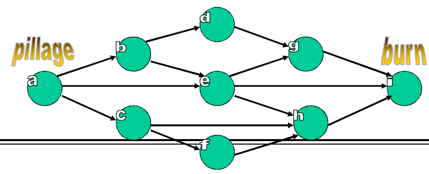
Don't burn before you pillage!



Slide 36

Topological Sort

- Informally, a topological sort is a linear ordering of the vertices of a DAG in which all successors of any given vertex appear in the sequence after that vertex.



Method to the Madness

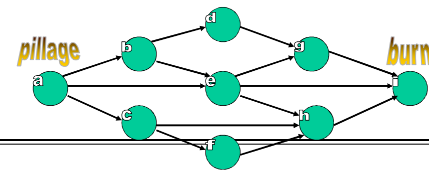
- One way to find a topological sort is to consider the *in-degrees* of the vertices. (The number of incoming edges is the **in-degree**). Clearly the first vertex in a topological sort must have in-degree zero and every DAG must contain at least one vertex with in-degree zero.

Simple Topological Sort Algorithm

- Repeat the following steps until the graph is empty:
  - Select a vertex that has in-degree zero.
  - Add the vertex to the sort.
  - Delete the vertex and all the edges emanating from it from the graph.

Test Your Knowledge

- Give a topological sort for this graph, it should be evident that more than one solution exists for this problem.



Backtracking Algorithm Depth-First Search

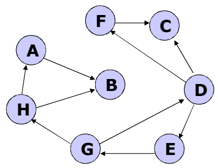
Text

Read Weiss, § 9.6 Depth-First Search and § 10.5 Backtracking Algorithms

Requirements

- Also called Depth-First Search
- Can be used to attempt to visit all nodes of a graph in a systematic manner
- Works with directed and undirected graphs
- Works with weighted and unweighted graphs

Walk-Through

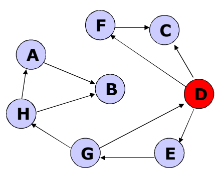


Visited Array

A	
B	
C	
D	
E	
F	
G	
H	

Task: Conduct a depth-first search of the graph starting with node D

Walk-Through



Visited Array

A	
B	
C	
D	√
E	
F	
G	
H	

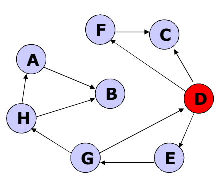
D

The order nodes are visited:

D

Visit D

Walk-Through



Visited Array

A	
B	
C	
D	√
E	
F	
G	
H	

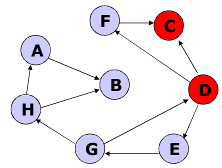
D

The order nodes are visited:

D

Consider nodes adjacent to D, decide to visit C first (Rule: visit adjacent nodes in alphabetical order)

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	
F	
G	
H	

C

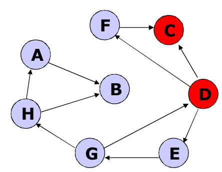
D

The order nodes are visited:

D, C

Visit C

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	
F	
G	
H	

C

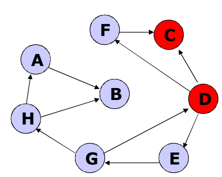
D

The order nodes are visited:

D, C

No nodes adjacent to C; cannot continue ➔ ***backtrack***, i.e., pop stack and restore previous state

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	
F	
G	
H	

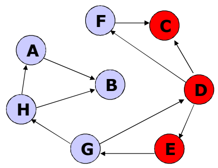
D

The order nodes are visited:

D, C

Back to D – C has been visited, decide to visit E next

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	
H	

E
D

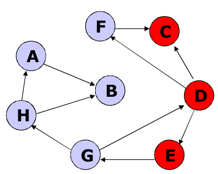
The order nodes are visited:

D, C, E

Back to D – C has been visited, decide to visit E next.

Slide 49

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	
H	

E
D

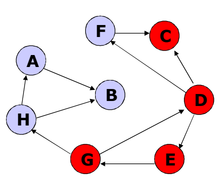
The order nodes are visited:

D, C, E

Only G is adjacent to E

Slide 50

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	√
H	

G
E
D

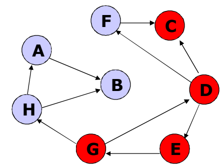
The order nodes are visited:

D, C, E, G

Visit G

Slide 51

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	√
H	

G
E
D

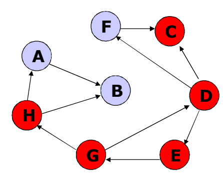
The order nodes are visited:

D, C, E, G

Nodes D and H are adjacent to G. D has already been visited. Decide to visit H.

Slide 52

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	√
H	√

H
G
E
D

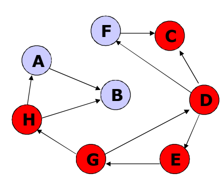
The order nodes are visited:

D, C, E, G, H

Visit H

Slide 53

Walk-Through



Visited Array

A	
B	
C	√
D	√
E	√
F	
G	√
H	√

H
G
E
D

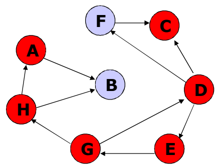
The order nodes are visited:

D, C, E, G, H

Nodes A and B are adjacent to F. Decide to visit A next.

Slide 54

Walk-Through



Visited Array

A	√
B	
C	√
D	√
E	√
F	
G	√
H	√

A
H
G
E
D

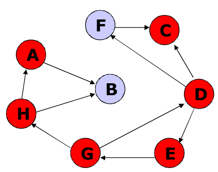
The order nodes are visited:

D, C, E, G, H, A

Visit A

Slide 55

Walk-Through



Visited Array

A	√
B	
C	√
D	√
E	√
F	
G	√
H	√

A
H
G
E
D

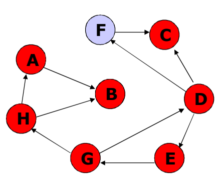
The order nodes are visited:

D, C, E, G, H, A

Only Node B is adjacent to A. Decide to visit B next.

Slide 56

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

B
A
H
G
E
D

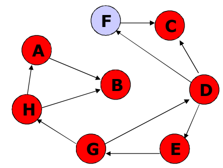
The order nodes are visited:

D, C, E, G, H, A, B

Visit B

Slide 57

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

A
H
G
E
D

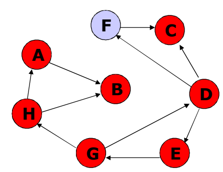
The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to B. Backtrack (pop the stack).

Slide 58

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

H
G
E
D

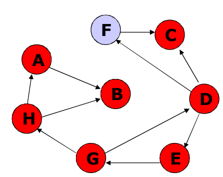
The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to A. Backtrack (pop the stack).

Slide 59

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

G
E
D

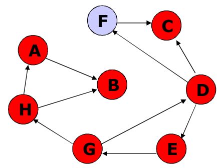
The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to H. Backtrack (pop the stack).

Slide 60

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

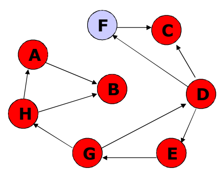
E
D

The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to G. Backtrack (pop the stack).

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

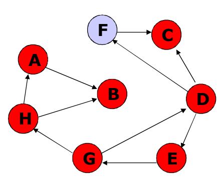
D

The order nodes are visited:

D, C, E, G, H, A, B

No unvisited nodes adjacent to E. Backtrack (pop the stack).

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	
G	√
H	√

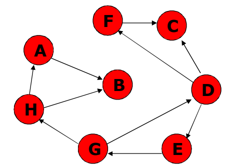
D

The order nodes are visited:

D, C, E, G, H, A, B

F is unvisited and is adjacent to D. Decide to visit F next.

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

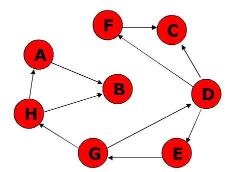
F
D

The order nodes are visited:

D, C, E, G, H, A, B, F

Visit F

Walk-Through



Visited Array

A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

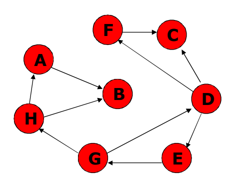
D

The order nodes are visited:

D, C, E, G, H, A, B, F

No unvisited nodes adjacent to F. Backtrack.

Walk-Through



Visited Array

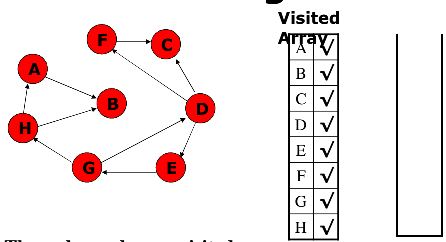
A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

The order nodes are visited:

D, C, E, G, H, A, B, F

No unvisited nodes adjacent to D. Backtrack.

Walk-Through



The order nodes are visited:

D, C, E, G, H, A, B, F

Stack is empty. Depth-first traversal is done.

Consider Trees

1. **What depth-first traversals do you know?**
2. **How do the traversals differ?**
3. **In the walk-through, we visited a node just as we pushed the node onto the stack. Is there another time at which you can visit the node?**
4. **Conduct a depth-first search of the same graph using the strategy you came up with in #3.**

Breadth-First Search

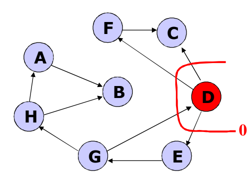
Text

Read Weiss, § 9.3 (pp. 299-304) Breadth-First Search Algorithms

Requirements

- Can be used to attempt to visit all nodes of a graph in a systematic manner
- Works with directed and undirected graphs
- Works with weighted and unweighted graphs

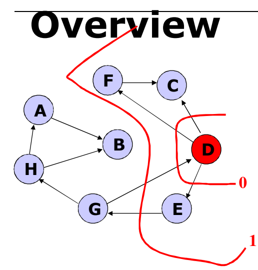
Overview



Breadth-first search starts with given node

Task: Conduct a breadth-first search of the graph starting with node D

Overview



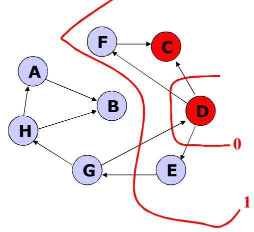
Breadth-first search starts with given node

Then visits nodes adjacent in some specified order (e.g., alphabetical)

Like ripples in a pond

Nodes visited: D

Overview



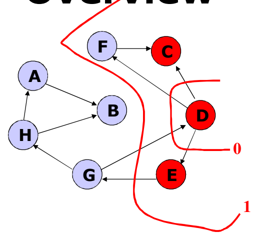
Breadth-first search starts with given node

Then visits nodes adjacent in some specified order (e.g., alphabetical)

Like ripples in a pond

Nodes visited: D, C

Overview



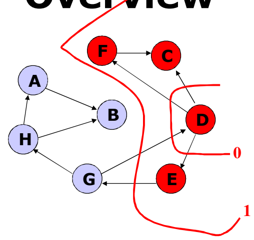
Breadth-first search starts with given node

Then visits nodes adjacent in some specified order (e.g., alphabetical)

Like ripples in a pond

Nodes visited: D, C, E

Overview



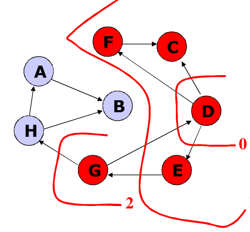
Breadth-first search starts with given node

Then visits nodes adjacent in some specified order (e.g., alphabetical)

Like ripples in a pond

Nodes visited: D, C, E, F

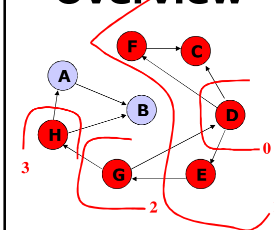
Overview



When all nodes in ripple are visited, visit nodes in next ripples

Nodes visited: D, C, E, F, G

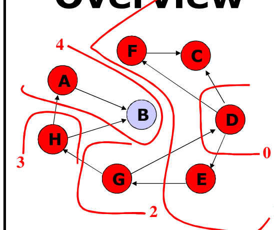
Overview



When all nodes in ripple are visited, visit nodes in next ripples

Nodes visited: D, C, E, F, G, H

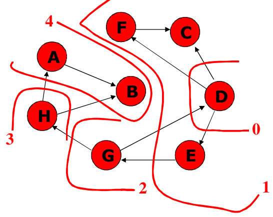
Overview



When all nodes in ripple are visited, visit nodes in next ripples

Nodes visited: D, C, E, F, G, H, A

Overview

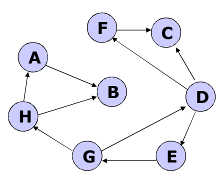


When all nodes in ripple are visited, visit nodes in next ripples

Nodes visited: D, C, E, F, G, H, A, B

Slide 79

Walk-Through



Enqueued Array

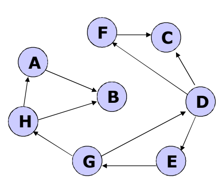
A	
B	
C	
D	
E	
F	
G	
H	

Q →

How is this accomplished? Simply replace the stack with a queue! Rules: (1) Maintain an *enqueued* array. (2) Visit node when *dequeued.*

Slide 80

Walk-Through



Enqueued Array

A	
B	
C	
D	√
E	
F	
G	
H	

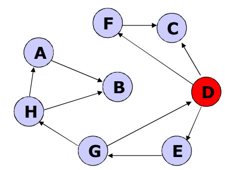
Q → D

Nodes visited:

Enqueue D. Notice, D not yet visited.

Slide 81

Walk-Through



Enqueued Array

A	
B	
C	√
D	√
E	√
F	√
G	
H	

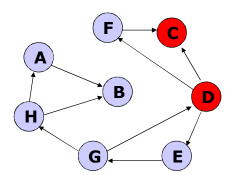
Q → C → E → F

Nodes visited: D

Dequeue D. Visit D. Enqueue unenqueued nodes adjacent to D.

Slide 82

Walk-Through



Enqueued Array

A	
B	
C	√
D	√
E	√
F	√
G	
H	

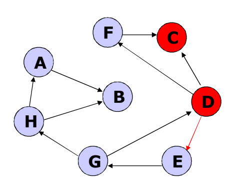
Q → E → F

Nodes visited: D, C

Dequeue C. Visit C. Enqueue unenqueued nodes adjacent to C.

Slide 83

Walk-Through



Enqueued Array

A	
B	
C	√
D	√
E	√
F	√
G	
H	

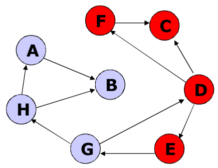
Q → F → G

Nodes visited: D, C, E

Dequeue E. Visit E. Enqueue unenqueued nodes adjacent to E.

Slide 84

Walk-Through



Enqueued Array

A	
B	
C	√
D	√
E	√
F	√
G	√
H	

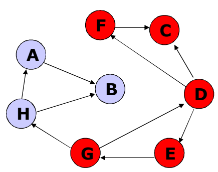
Q → G

Nodes visited: D, C, E, F

Dequeue F. Visit F. Enqueue unenqueued nodes adjacent to F.

Slide 85

Walk-Through



Enqueued Array

A	
B	
C	√
D	√
E	√
F	√
G	√
H	√

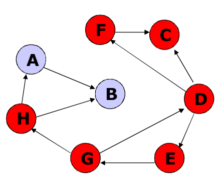
Q → H

Nodes visited: D, C, E, F, G

Dequeue G. Visit G. Enqueue unenqueued nodes adjacent to G.

Slide 86

Walk-Through



Enqueued Array

A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

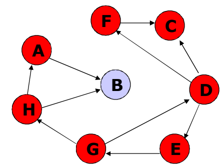
Q → A → B

Nodes visited: D, C, E, F, G, H

Dequeue H. Visit H. Enqueue unenqueued nodes adjacent to H.

Slide 87

Walk-Through



Enqueued Array

A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

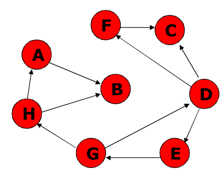
Q → B

Nodes visited: D, C, E, F, G, H, A

Dequeue A. Visit A. Enqueue unenqueued nodes adjacent to A.

Slide 88

Walk-Through



Enqueued Array

A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

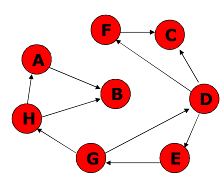
Q empty

Nodes visited: D, C, E, F, G, H, A, B

Dequeue B. Visit B. Enqueue unenqueued nodes adjacent to B.

Slide 89

Walk-Through



Enqueued Array

A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

Q empty

Nodes visited: D, C, E, F, G, H, A, B

Q empty. Algorithm done.

Slide 90

Consider Trees

1. **What do we call a breadth-first traversal on trees?**

Dijkstra’s Algorithm

Text

Read Weiss, § 9.3

Dijkstra’s Algorithm

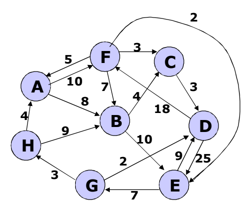
Single Source Multiple Destination
Shortest Path Algorithm

Requirements

- Works with directed and undirected graphs
- Works with weighted and unweighted graphs
- Rare type of algorithm ➔

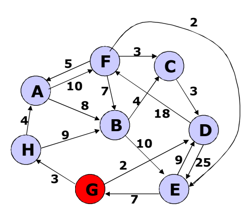
  A greedy algorithm that produces an optimal solution

Walk-Through



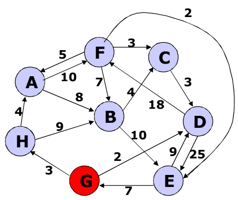
Initialize array

	K	d_v	p_v
A	F	∞	–
B	F	∞	–
C	F	∞	–
D	F	∞	–
E	F	∞	–
F	F	∞	–
G	F	∞	–
H	F	∞	–



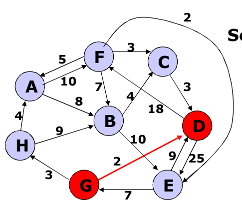
Start with G

	K	d_v	p_v
A			
B			
C			
D			
E			
F			
G	T	0	–
H			



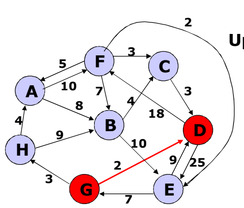
Update unselected nodes

	K	d_v	p_v
A			
B			
C			
D		2	G
E			
F			
G	T	0	–
H		3	G



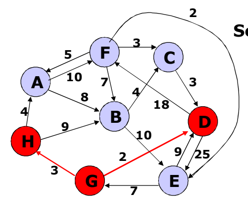
Select minimum distance

	K	d_v	p_v
A			
B			
C			
D	T	2	G
E			
F			
G	T	0	–
H		3	G



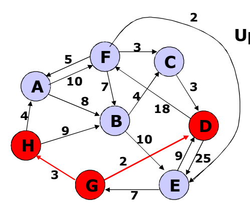
Update unselected nodes

	K	d_v	p_v
A			
B			
C			
D	T	2	G
E		27	D
F		20	D
G	T	0	–
H		3	G



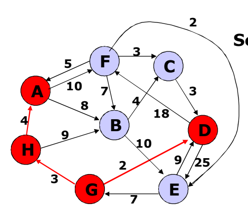
Select minimum distance

	K	d_v	p_v
A			
B			
C			
D	T	2	G
E		27	D
F		20	D
G	T	0	–
H	T	3	G



Update unselected nodes

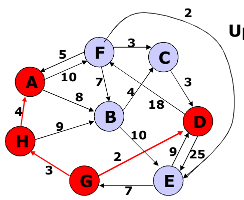
	K	d_v	p_v
A		7	H
B		12	H
C			
D	T	2	G
E		27	D
F		20	D
G	T	0	–
H	T	3	G



Select minimum distance

	K	d_v	p_v
A	T	7	H
B		12	H
C			
D	T	2	G
E		27	D
F		20	D
G	T	0	–
H	T	3	G

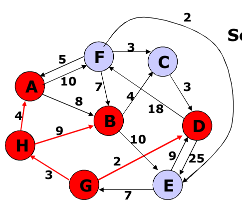
Update unselected nodes



	K	d_v	p_v
A	T	7	H
B		12	H
C			
D	T	2	G
E		27	D
F		17	A
G	T	0	–
H	T	3	G

Slide 103

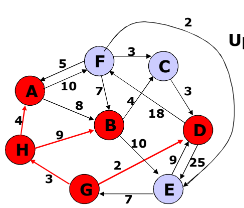
Select minimum distance



	K	d_v	p_v
A	T	7	H
B	T	12	H
C			
D	T	2	G
E		27	D
F		17	A
G	T	0	–
H	T	3	G

Slide 104

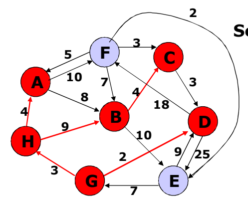
Update unselected nodes



	K	d_v	p_v
A	T	7	H
B	T	12	H
C		16	B
D	T	2	G
E		22	B
F		17	A
G	T	0	–
H	T	3	G

Slide 105

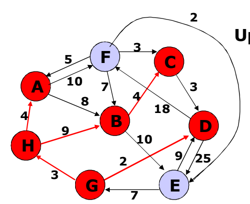
Select minimum distance



	K	d_v	p_v
A	T	7	H
B	T	12	H
C	T	16	B
D	T	2	G
E		22	B
F		17	A
G	T	0	–
H	T	3	G

Slide 106

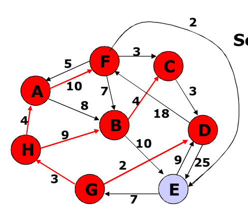
Update unselected nodes



	K	d_v	p_v
A	T	7	H
B	T	12	H
C	T	16	B
D	T	2	G
E		22	B
F		17	A
G	T	0	–
H	T	3	G

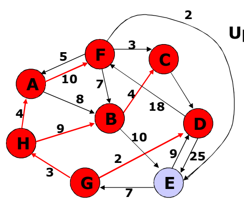
Slide 107

Select minimum distance



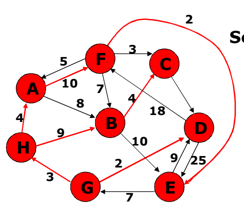
	K	d_v	p_v
A	T	7	H
B	T	12	H
C	T	16	B
D	T	2	G
E		22	B
F	T	17	A
G	T	0	–
H	T	3	G

Slide 108



Update unselected nodes

	K	d_v	p_v
A	T	7	H
B	T	12	H
C	T	16	B
D	T	2	G
E		19	F
F	T	17	A
G	T	0	–
H	T	3	G



Select minimum distance

	K	d_v	p_v
A	T	7	H
B	T	12	H
C	T	16	B
D	T	2	G
E	T	19	F
F	T	17	A
G	T	0	–
H	T	3	G

Done

Dijkstra's Algorithm

http://www.youtube.com/watch?v=8Ls1RqHCOPw

Order of Complexity

- Analysis
  - findMin() takes O(V) time
  - outer loop iterates (V-1) times
  - ➔ O(V^2) time
- Optimal for dense graphs, i.e., |E| = O(V^2)
- Suboptimal for sparse graphs, i.e., |E| = O(V)

Order of Complexity

If the graph is sparse, i.e., |E| = O(V)
- **maintain distances in a priority queue**
- **insert new (shorter) distance produced by line 10 of Figure 9.32**

➔ O(|E| log |V|) complexity

Negative Edge Weights

Read § 9.3.3

Dijkstra's algorithm as shown in Figure 9.32 does not work! Why?

Acyclic Graphs

- Read § 9.3.4
- Combine topological sort with Dijkstra's algorithm

Slide 115

All-Pairs Shortest Paths

- One option: run Dijktra's algorithm |V| times ➔ $O(V^3)$ time
- A more efficient $O(V^3)$ time algorithm is discussed in Chapter 10

Slide 116