Part 10. Graphs

CS 200 Algorithms and Data Structures

1

Outline

- Introduction
- Terminology
- Implementing Graphs
- Graph Traversals
- **Topological Sorting**
- Shortest Paths
- Spanning Trees
- Minimum Spanning Trees
- Circuits

2

Graphs Describing Precedence

- Examples:
  - prerequisites for a set of courses
  - dependencies between programs
- Edge from a to b indicates a should come before b

Graphs Describing Precedence

Batman images are from the book "Introduction to bioinformatics algorithms"

Graphs Describing Precedence

- Want an ordering of the vertices of the graph that respects the precedence relation
  - Example: An ordering of CS courses
- The graph does not contain cycles. Why?

Topological Sorting of DAGs

- DAG: Directed Acyclic Graph
- Topological sort: listing of nodes such that if (a,b) is an edge, a appears before b in the list
- Is a topological sort unique?

A directed graph without cycles

a b c d e f g

a,g,d,b,e,c,f
a,b,g,d,e,f,c

7

Topological Sort - Algorithm 1

```
topSort1(in G:Graph)
  n= number of vertices in G
  for (step =1 through n)
   select a vertex v that has no successors
   aList.add(first_available_loc,v)
   Delete from G vertex v and its edges
  return aList
```

Algorithm relies on the fact that in a DAG there is always a vertex that has no successors

Algorithm 1: Example 1

a b c d e f g

a g d b e c f

9

Algorithm 2: Example 2

A B C D E F G H I

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

Topological Sort - Algorithm 2

- Modification of DFS: Traverse tree using DFS starting from all nodes that have no predecessor.
- Add a node to the list when ready to backtrack.

DFS Example: review

A B C D E F G H I J K L M N O P

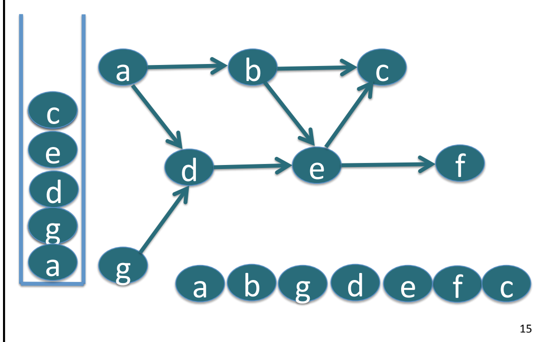
Iterative DFS: review

```
dfs(in v:Vertex)
    s – stack for keeping track of active vertices
    s.push(v)
    mark v as visited
    while(!s.isEmpty()) {
      if (no unvisited vertices adjacent to the vertex
          on top of the stack) {
       s.pop()  \\backtrack
      else {
       select unvisited vertex u adjacent to vertex on
      top of the stack
       s.push(u)
       mark u as visited
      }
    }
```

Topological Sort - Algorithm 2

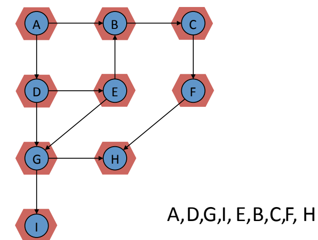
```
topSort2( in theGraph:Graph):List
   s.createStack()
   for (all vertices v in the graph theGraph)
    if (v has no predecessors)
       s.push(v)
       Mark v as visited
   while (!s.isEmpty())
    if (all vertices adjacent to the vertex on top of  the
   stack have been visited)
       v = s.pop()
       aLlist.add(1, v)
    else
       Select an unvisited vertex u adjacent to vertex on
       top of the stack
       s.push(u)
       Mark u as visited
   return aList
```

Algorithm 2: Example 1



15

Algorithm 2: Example 2



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

Outline

- Introduction
- Terminology
- Implementing Graphs
- Graph Traversals
- Topological Sorting
- **Shortest Paths**
- Spanning Trees
- Minimum Spanning Trees
- Circuits

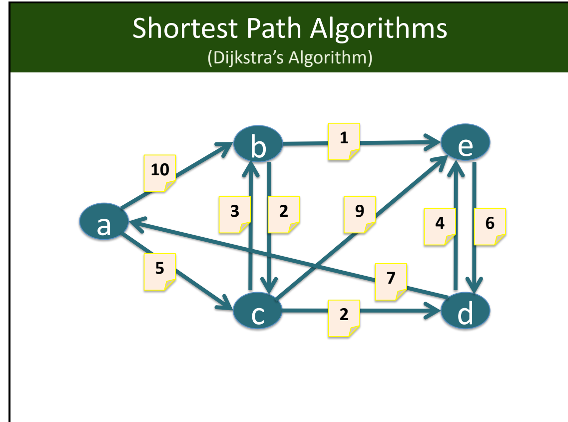
17

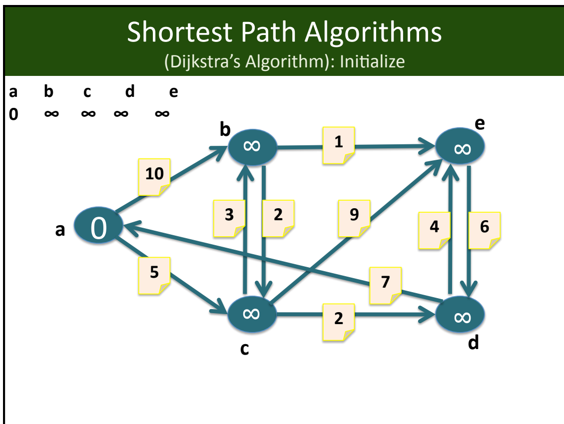
Shortest Path Algorithms
(Dijkstra's Algorithm)

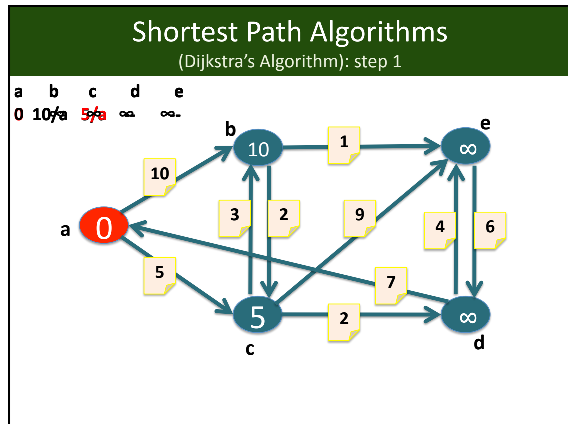
- Graph G(V,E) with non-negative weights ("distances")
- Compute shortest distances from vertex s to ***every other vertex*** in the graph

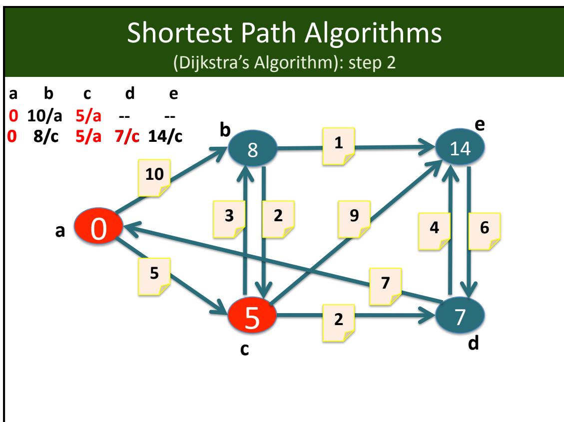
Shortest Path Algorithms
(Dijkstra's Algorithm)

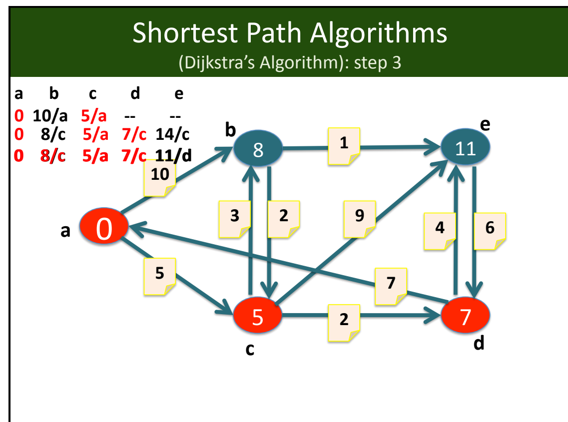
- Algorithm
  - Maintain array d (minimum distance estimates)
    - Init: d[s]=0, d[v]=∞ v∈V-s
  - Priority queue of vertices not yet visited
  - select minimum distance vertex, visit v, update neighbors

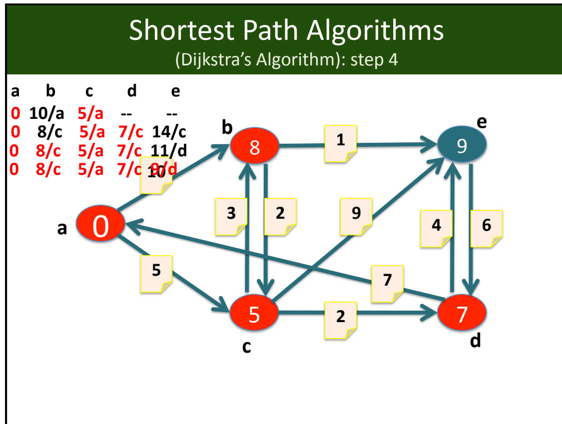


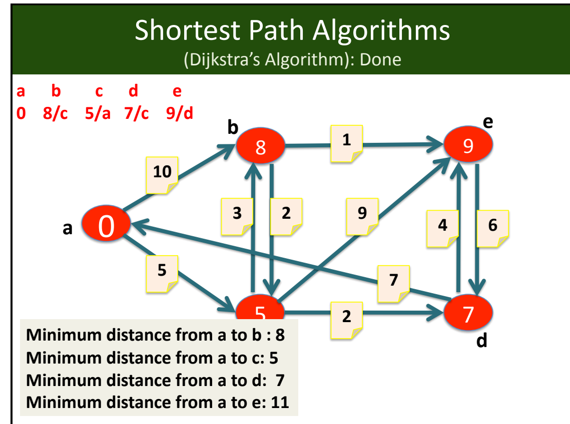


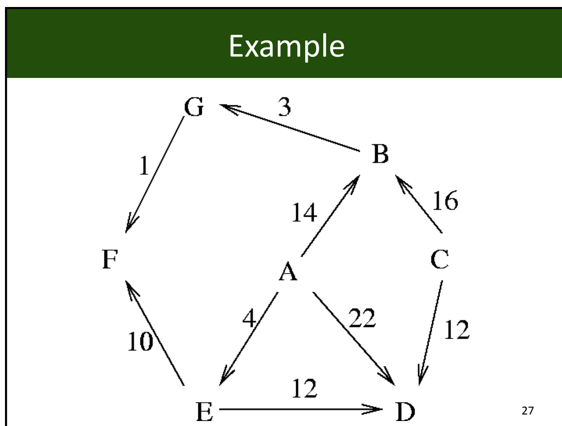






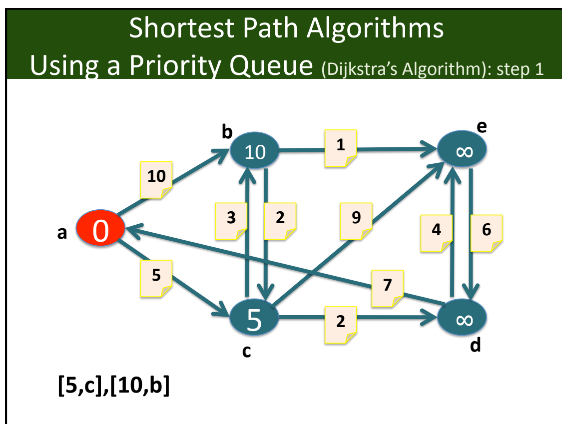


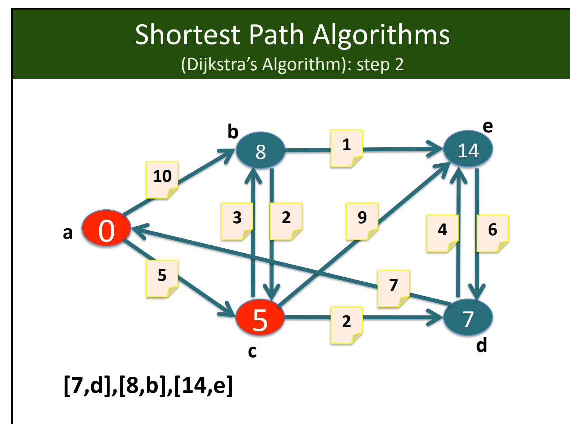


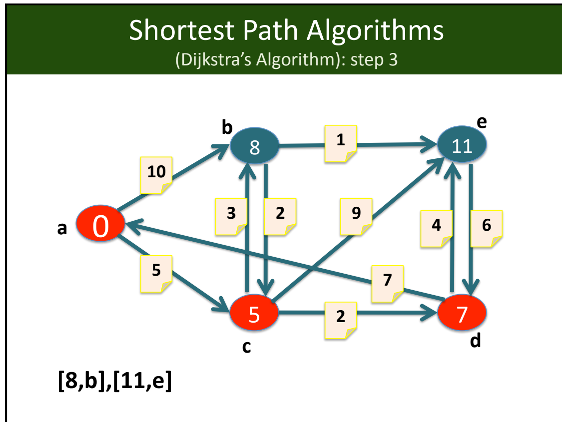


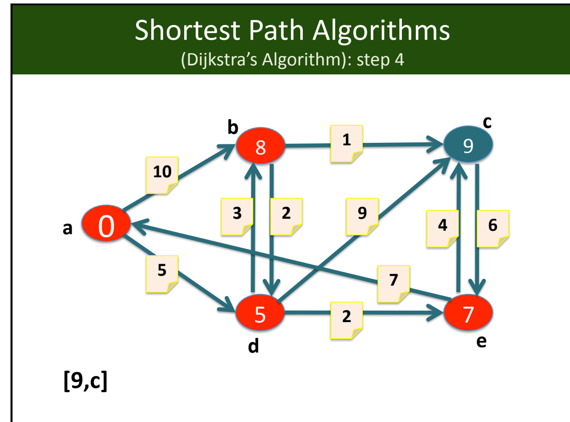
Dijkstra's Algorithm

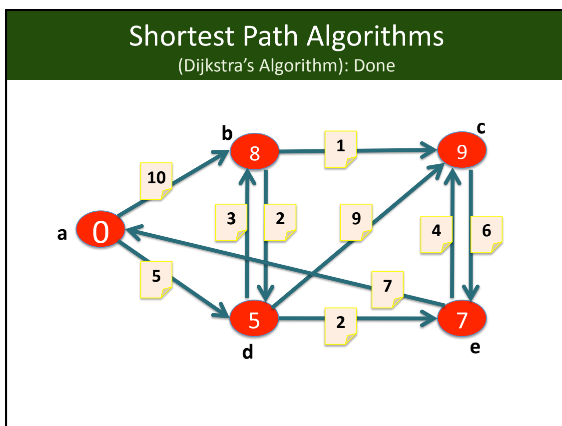
```
Dijkstra(G: graph with vertices v0…vn-1 and weights w
  [u][v])
  // computes shortest distance of vertex 0 to every
  other vertex
  create a set vertexSet that contains only vertex 0
  d[0] = 0
  for (v = 1 through n-1)
   d[v] =  infinity
  for (step = 2 through n)
   find the smallest d[v] such that v is not in
  vertexSet
   add v to vertexSet
   for (all vertices u not in vertexSet)
      if (d[u] > d[v] + w[v][u])
         d[u] = d[v] + w[v][u]
```











Dijkstra's Algorithm

- How to obtain the shortest paths?
  - At each vertex maintain a pointer that tells you the vertex from which you arrived.

Outline

- Introduction
- Terminology
- Implementing Graphs
- Graph Traversals
- Topological Sorting
- Shortest Paths
- **Spanning Trees**
- Minimum Spanning Trees
- Circuits

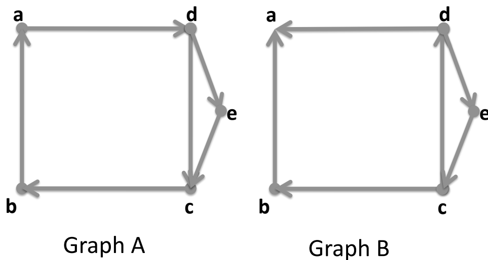
35

Connectedness in Directed Graphs

- A directed graph is ***strongly connected*** if there is a path from *a* to *b* and from *b* to *a* whenever *a* and *b* are vertices in the graph.
- A directed graph is ***weakly connected*** if there is a path between every two vertices in the underlying undirected graph.

36

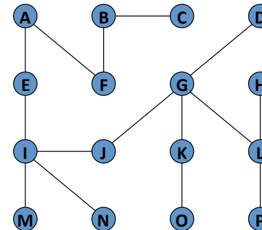
Example



37

Trees as Graphs

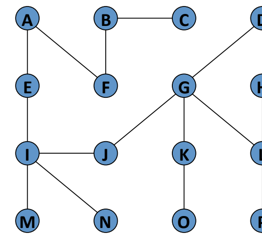
- Tree: an undirected connected graph that has no cycles.



Rooted Trees

- A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root

Example: Build rooted trees.



Trees as Graphs

- Tree: an undirected connected graph that has no simple circuits.

Theorem 10-1

An undirected graph is a tree iff there is a unique simple path (no repeated vertices) between any two vertices.

When is a graph a Tree?

- Can explicitly check that the graph is connected and has no cycles. (How?)
- We need an alternative characterization

When is a graph a Tree?: Theorem 10-2

- A connected undirected graph with n vertices must have at least n-1 edges (PROOF: by induction on the number of vertices)

When is a graph a Tree?: Theorem 10-3

- A connected undirected graph that has n vertices and exactly n-1 edges cannot contain a cycle (PROOF: by contradiction with previous statement)

When is a graph a Tree? : Theorem 10-4

- A connected undirected graph that has n vertices and more than n-1 edges must contain a cycle.

When is a graph a Tree?

- Conclusion: A connected graph with *n* vertices and *n-1* edges is a tree.
- In order to check if a graph is a tree we need to check that it is connected and count the number of edges and vertices.

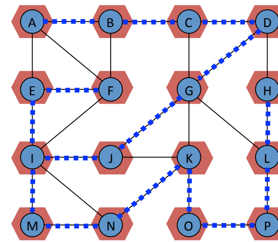
Spanning Trees

- **Spanning tree:** A subgraph of a connected undirected graph G that contains all of G's vertices and enough of its edges to form a tree.
- How to get a spanning tree:
  - Remove edges until you get a tree.
  - Add edges until you have a spanning tree

Spanning Trees - DFS algorithm

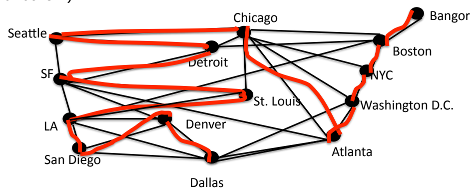
```
dfsTree(in v:vertex)
  Mark v as visited
  for (each unvisited vertex u adjacent to v)
   Mark the edge from u to v
   dfsTree(u)
```

Spanning Tree – Depth First Search Example



Example

- Suppose that an airline must reduce its flight schedule to save money. If its original routes are as illustrated here, which flights can be discontinued to retain service between all pairs of cities (where it may be necessary to combine flights to fly from one city to another?)



51

Outline

- Introduction
- Terminology
- Implementing Graphs
- Graph Traversals
- Topological Sorting
- Shortest Paths
- Spanning Trees
- **Minimum Spanning Trees**
- Circuits

52

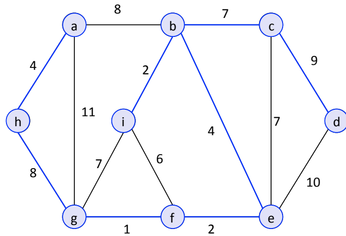
Minimum Spanning Tree

- Minimum spanning tree
  - Spanning tree **minimizing the sum of edge weights**
- Example: Connecting each house in the neighborhood to cable
  - Graph where each house is a vertex.
  - Need the graph to be connected, and minimize the cost of laying the cables.

Prim's Algorithm

- Idea: incrementally build spanning tree by adding the least-cost edge to the tree
  - Weighted graph
  - Find a set of edges
    - Touches all vertices
    - Minimal weight
    - Not all the edges may be used

Prim's Algorithm: Example



{(d,c),(c,b), (b,i), (b,e), (e,f), (f,g), (g,h), (h,a) }

Prim's Algorithm

```
prims(in: G=(V,E):Graph)
  //V_T – current vertices in spanning tree
  //E_T – edges belonging to the spanning tree
  V_T = {w}  // w is an arbitrarily chosen vertex
  E_T = φ  //spanning tree contains no vertices
  initially
  for i = 1 to |V| - 1 do
   find a minimum-weight edge e=(u,v) among
  edges that  connects a vertex in V_T with a
  vertex in V – V_T
   add v to V_T
   add e to E_T
  return E_T
```

Implementing Prim's Algorithm

- Each node not in the tree has an attaching cost – the weight of the smallest edge that connects it to the forming tree (infinity if no such edge exists).
- At each iteration, we retrieve the node with the smallest attaching cost and update the attaching cost of its neighbors.
- Can use a priority queue! (need to add a method for updating priorities).