



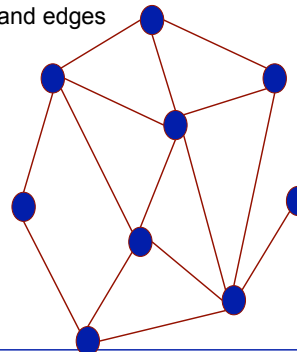
CS200: Graphs

Prichard Ch. 14
Rosen Ch. 11

Graphs



A collection of nodes and edges



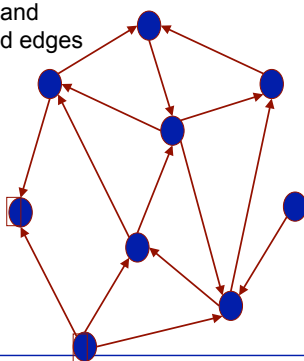
What can this represent?

- A computer network
- Abstraction of a map
- Social network

Directed Graphs



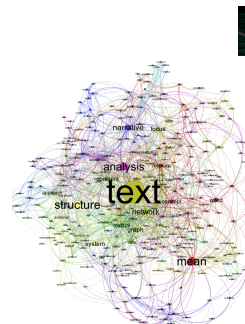
A collection of nodes and directed edges



Sometimes we want to represent directionality:

- Unidirectional network connections
- One way streets
- The web


Graphs/Networks Around Us



<http://lin-ear-th-inking.blogspot.com/2010/12/visualizing-geodetic-information-with.html>

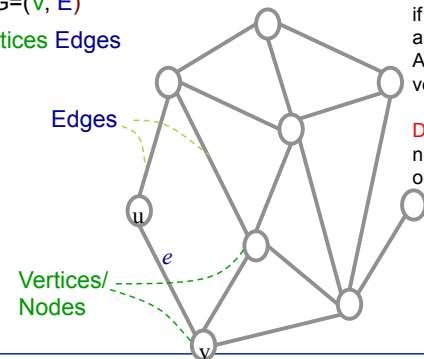
<http://nodustabs.com/wp-content/uploads/2011/12/figure-5-meaning-circulation.png>

Graph Terminology



$G=(V, E)$

Vertices Edges



Edges

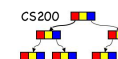
Vertices/
Nodes

Two vertices are **adjacent** if they are connected by an edge.
An edge is **incident** on two vertices

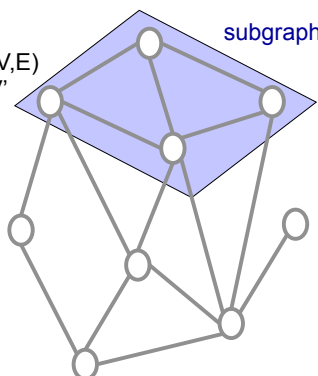
Degree of a vertex:
number of edges incident on it

Graph terminology: 14.1 in Prichard, 10.1 in Rosen

Graph Terminology




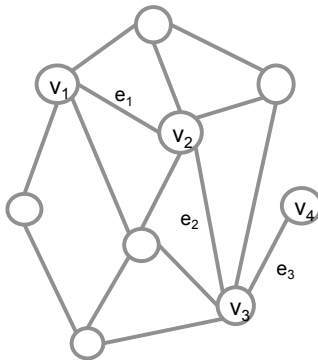
A **subgraph** of a graph $G = (V, E)$ is a graph (V', E') such that V' is a subset of V and E' is a subset of E



subgraph


Paths

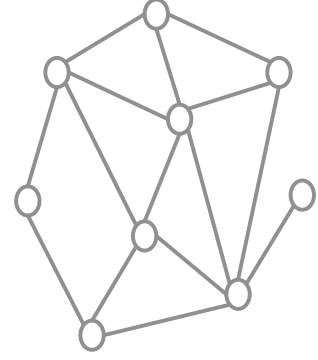




- **Path:** a sequence of edges
- (e_1, e_2, e_3) is a path of length 3 from v_1 to v_4
- In a simple graph a path can be represented as a sequence of vertices

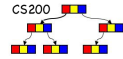
Graph Terminology



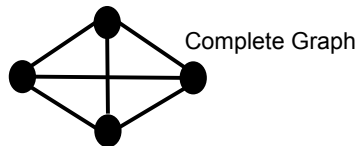


- **Self loop (loop):** an edge that connects a vertex to itself
- **Simple graph:** no self loops and no two edges connect the same vertices
- **Multigraph:** may have multiple edges connecting the same vertices
- **Pseudograph:** multigraph with self-loops

Complete Graphs

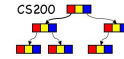


- Simple graph that contains exactly one edge between each pair of distinct vertices.

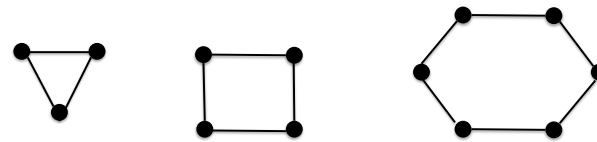


10

Cycles



The cycle C_n , $n \geq 3$, consists of n vertices v_1, v_2, \dots, v_n and edges $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \text{and } \{v_{n-1}, v_n\}$.

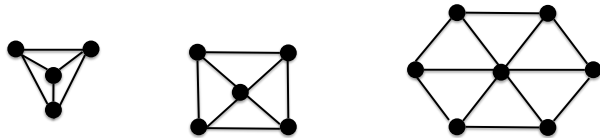


11

Wheels

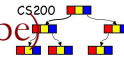


- We obtain the wheel W_n when we add an additional vertex to the cycle C_n , for $n \geq 3$, and connect this new vertex to each of the n vertices in C_n , by new edges

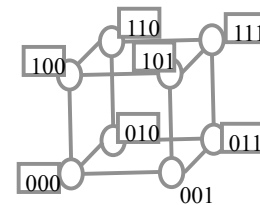


12

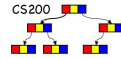
n -Cubes (n -dimensional hypercube)



Hypercube



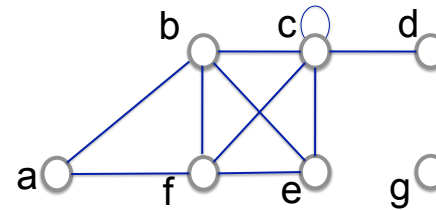
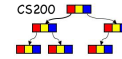
The degree of a vertex



- The degree of a vertex in an undirected graph
 - the number of edges incident with it
 - except that a loop at a vertex contributes twice to the degree of that vertex.

14

Example



$$\begin{array}{ll} \deg(a) = 2 & \deg(d) = 1 \\ \deg(b) = \deg(f) = 4 & \deg(e) = 3 \\ & \deg(g) = 0 \end{array}$$

15

Some Graph Theorems



- **Handshaking:** Let $G=(V,E)$ be an undirected graph with e edges. Then

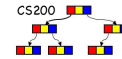
$$2e = \sum_{v \in V} \deg(v)$$

- An undirected graph has an even number of vertices of odd degree.
- Let $G=(V,E)$ be a graph with directed edges. Then

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|$$

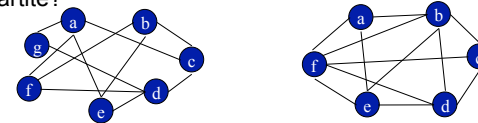
14

Bipartite Graphs



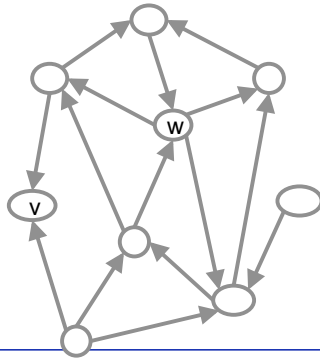
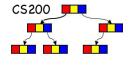
- A simple graph on which the vertex set V can be partitioned into two disjoint sets V_1 and V_2 such that every edge connects a vertex in V_1 to one in V_2 .

- Bipartite?



- **Theorem:** A simple graph is bipartite iff it is possible to assign one of two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color.

Directed Graphs



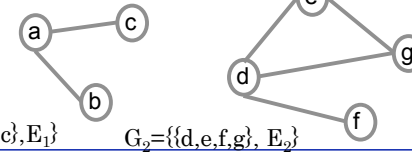
Indegree: number of incoming edges
Outdegree: number of outgoing edges

Connected Components



- An undirected graph is called **connected** if there is a path between every pair of vertices of the graph.
- A **connected component** of a graph G is a connected subgraph of G that is not a proper subgraph of another connected subgraph of G .

$G = \{\{a, b, c, d, e, f, g\}, E\}$



$G_1 = \{\{a, b, c\}, E_1\}$

$G_2 = \{\{d, e, f, g\}, E_2\}$

Graph ADT



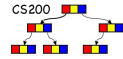
- Create
- Empty?
- Number of vertices?
- Number of edges?
- Edge exists between two vertices?
- Add a vertex
- Add an edge between two vertices
- Delete a vertex (and any connected edges)
- Delete the edge between two vertices
- Retrieve a vertex

Classes for Undirected Graph with Instance Variables



- Edge:
 - vertex1, vertex2
 - weight
- Graph:
 - Number of edges, number of vertices
 - organized collection of vertices and edges

Graph Data Structures - Adjacency Matrix



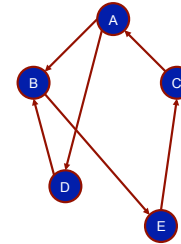
- Vertices
 - labels mapped into indices
 - one vertex mapped to **one** index
 - Values:
 - boolean to indicate presence/absence of edge in (un)directed graph
 - int to indicate value of weighted edge
- Edges
 - square matrix of edges
 - size = number of vertices
 - edge: two (vertex) indices
- useful for dense graphs

Adjacency Matrix Example



Label	Index
A	0
B	1
C	2
D	3
E	4

mapping of vertex labels to array indices



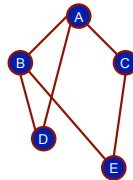
	0	1	2	3	4
0	0	1	0	1	0
1	0	0	0	0	1
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0

Adjacency Matrix:
array of edges indexed by vertex

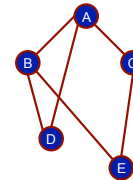
Graph Data Structures - Adjacency List



- Vertices
 - mapped to list of adjacencies
 - adjacency: edge
- Edges: lists of adjacencies
 - linked-list of out-going edges per vertex
- useful for sparse graphs



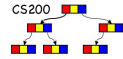
Adjacency List Example



Index	Label
0	A → B → D
1	B → E
2	C → A
3	D → B
4	E → C

mapping of vertex labels to list of edges

Which Implementation Is Best?



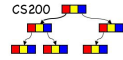
- Which implementation best supports common Graph Operations:
 - Is there an edge between vertex i and vertex j?
 - Find all vertices adjacent to vertex j
- Which best uses space?

Implementation: Edge Class



```
Class Edge {
    private Integer v,w; // vertices
    private int weight;
    public Edge(Integer first, Integer second, int edgeWeight){
        v = first; w = second; weight = edgeWeight; }
    public int getWeight() {
        return weight; }
    public Integer getV() {
        return v; }
    public Integer getW() {
        return w; }
```

Implementation: Graph Class



```
class Graph {
    private int numVertices;
    private int numEdges;
    private Vector<TreeMap<Integer, Integer>> adjList;

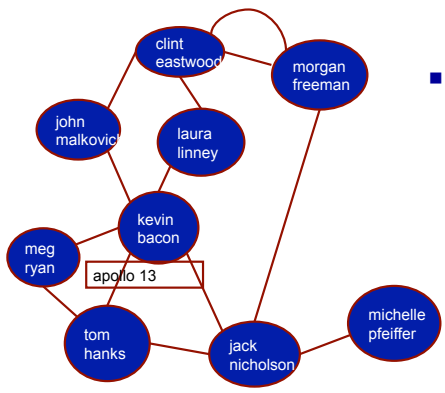
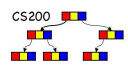
    public Graph(int n) {
        numVertices = n; numEdges = 0;
        adjList = new Vector<TreeMap<Integer, Integer>>();
        for (int i=0; i<numVertices; i++) {
            adjList.add(new TreeMap<Integer, Integer>());
        }
    }
}
```

Implementation: Graph Class Methods



```
public int getNumVertices()
public int getNumEdges()
public int getEdgeWeight(Integer v, Integer w)
public void addEdge(Integer v, Integer w)
public void addEdge(Edge e)
public void removeEdge(Edge e)
public Edge findEdge(Integer v, Integer w)
```

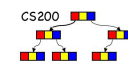
Six Degrees of Kevin Bacon



- Actor x has a Kevin Bacon Number of n if the shortest path between x and Kevin Bacon has length n

Graph made with the help of the oracle of Bacon: <http://oracleofbacon.org/>

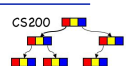
Shortest Path Algorithms



(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

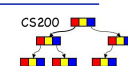
Shortest Path Algorithms



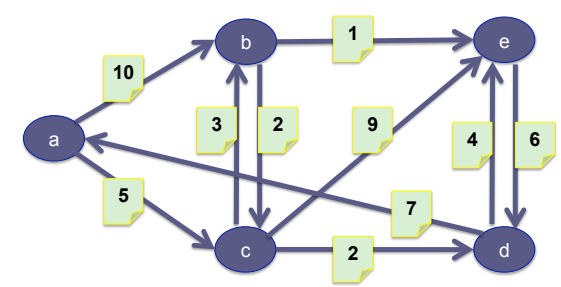
(Dijkstra's Algorithm)

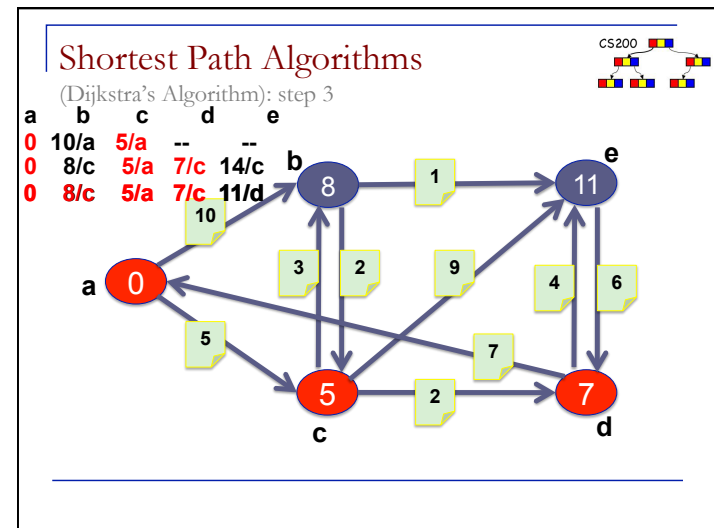
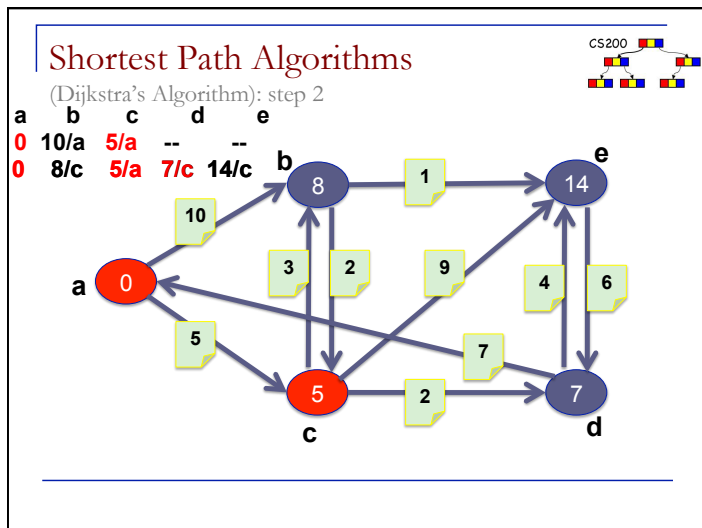
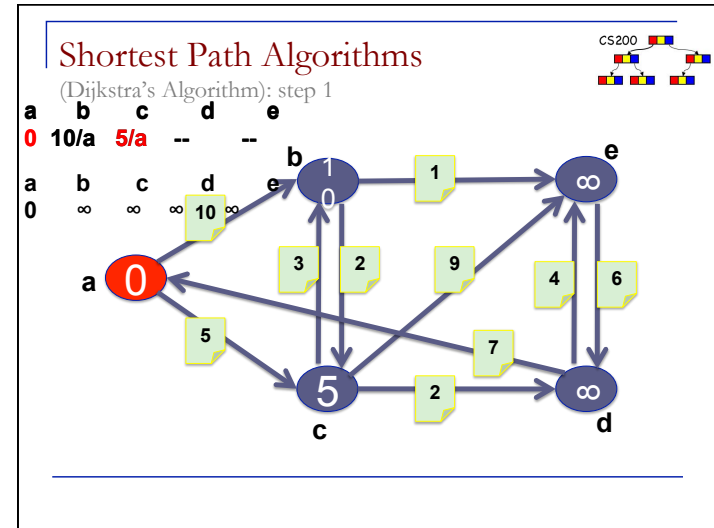
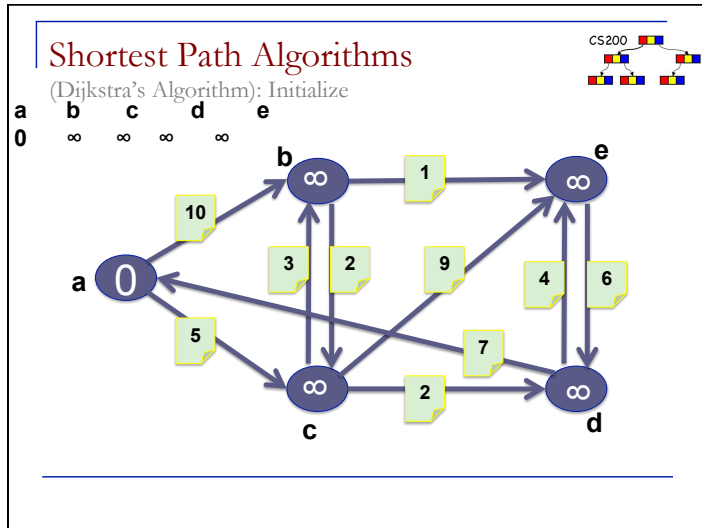
- Algorithm
 - Maintain array d (minimum distance estimates)
 - Init: $d[s]=0, d[v]=\infty \forall v \in V-s$
 - Priority queue of vertices **not yet visited**
 - select minimum distance vertex, visit v , update neighbors
- Interactive Dijkstra's algorithm:
 - http://students.ceid.upatras.gr/~papagel/project/kef5_7_1.htm

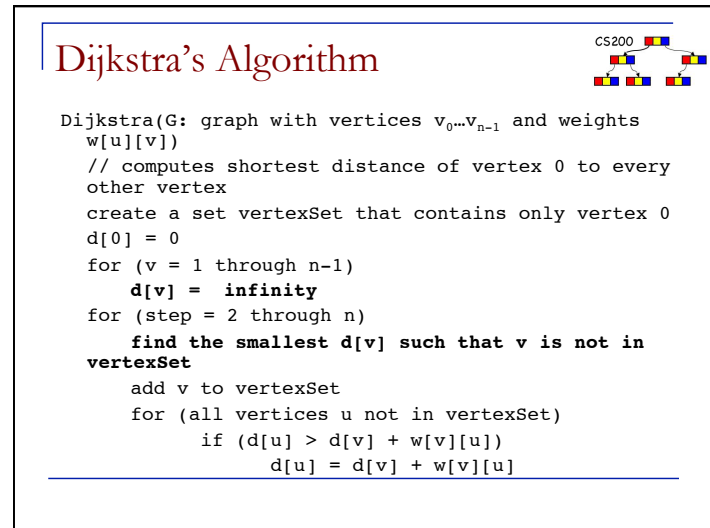
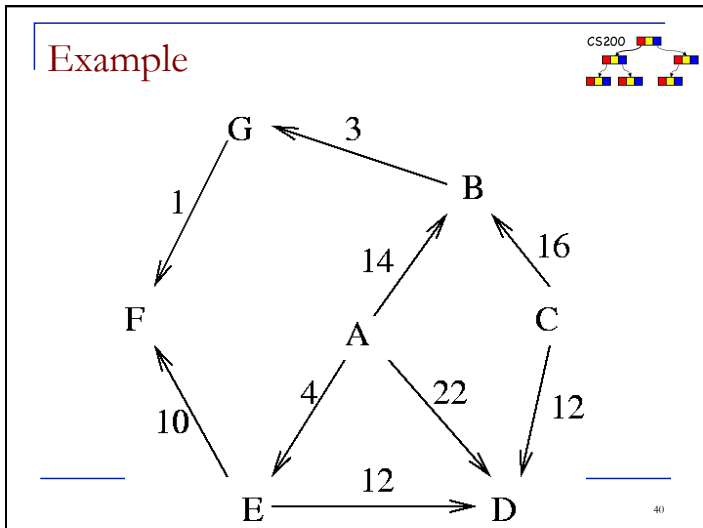
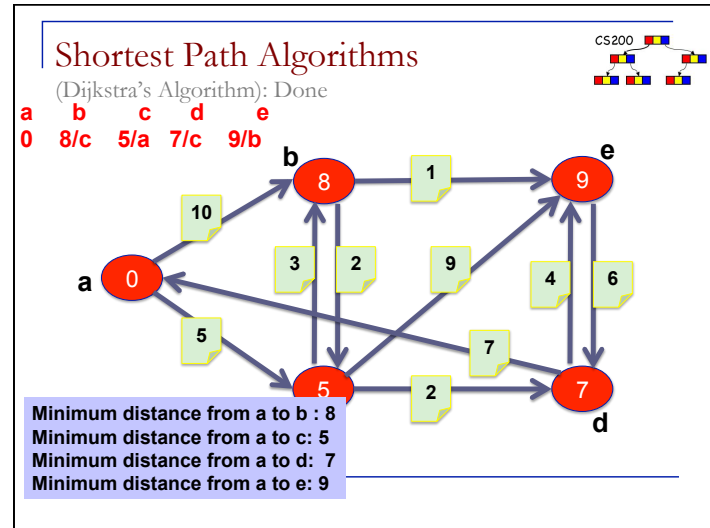
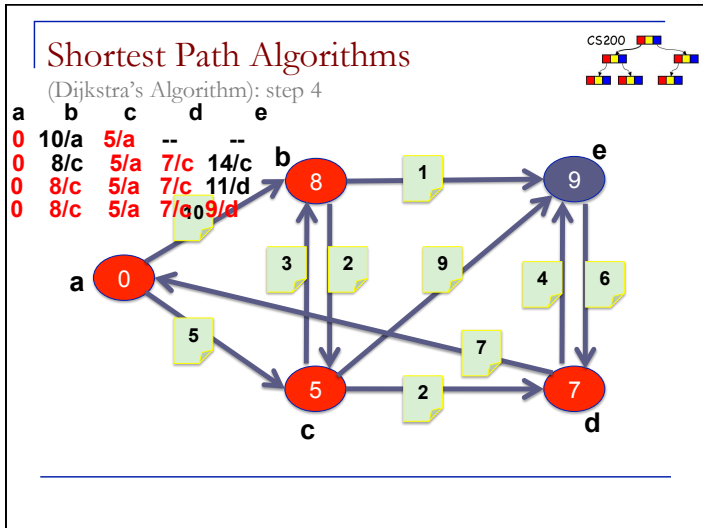
Shortest Path Algorithms



(Dijkstra's Algorithm)

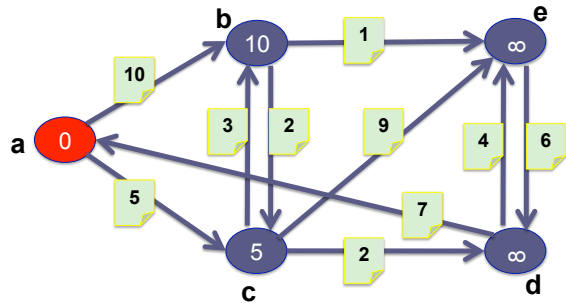
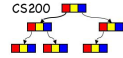






Shortest Path Algorithms

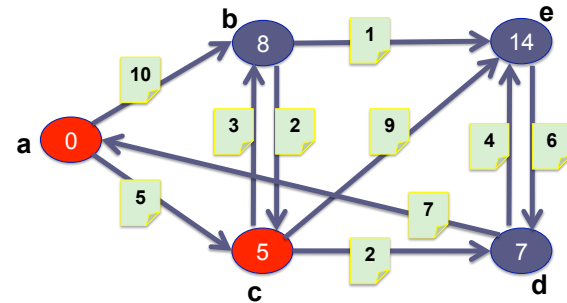
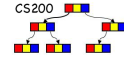
Using a Priority Queue (Dijkstra's Algorithm): step 1



[5,c],[10,b]

Shortest Path Algorithms

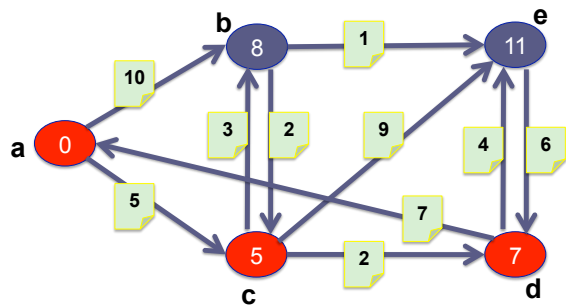
(Dijkstra's Algorithm): step 2



[7,d],[8,b],[14,e]

Shortest Path Algorithms

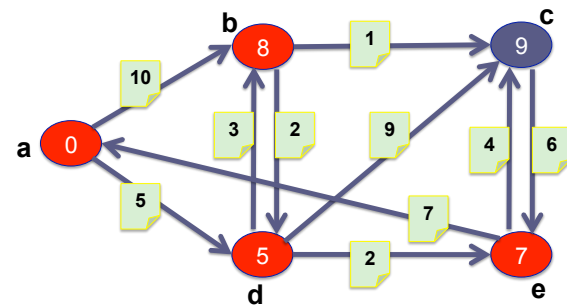
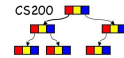
(Dijkstra's Algorithm): step 3



[8,b],[11,e]

Shortest Path Algorithms

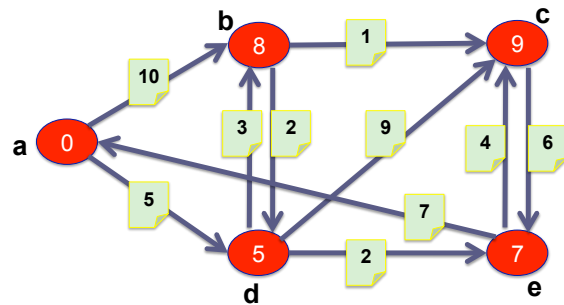
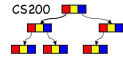
(Dijkstra's Algorithm): step 4



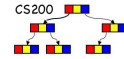
[9,c]

Shortest Path Algorithms

(Dijkstra's Algorithm): Done



Dijkstra's Algorithm



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