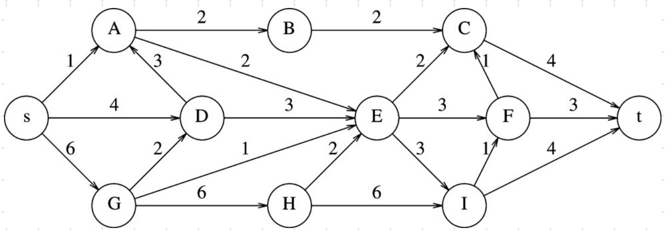
Graphs

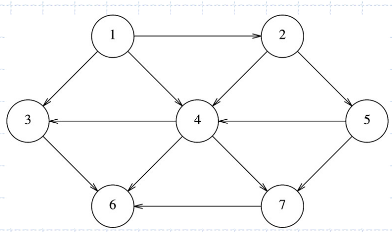


What is a Graph

- A graph consists of a set of vertices(V) and a set of edges(E)
- G = (V,E)
- V: Vertices (nodes)
- E: Edges. Each edge is a pair of vertices (v,w)

where v,w $\in$ V

- a connection between v and w

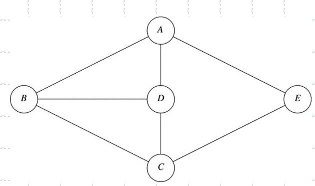


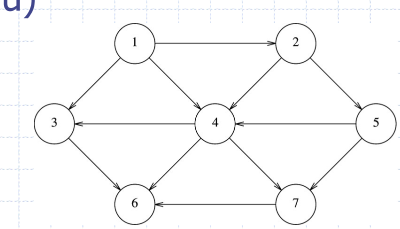
What is a Graph

- Applications:
  - Airport System
    - Airports are vertices
    - Flights are edges
    - Best path may be shortest flying time or fewest edges (connections)
  - Traffic Flow
    - Intersections are vertices
    - Roads are edges
    - Best path may be fastest time or shortest distance

Graph Terminology

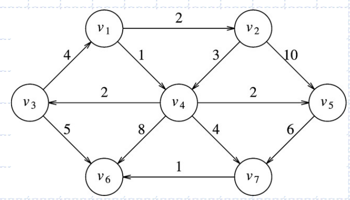
- Directed graph (digraph)
  - Pair is ordered (u,v) does not imply (v,u)
- Undirected graph
  - (u,v) is same as (v,u)





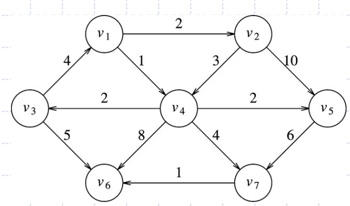
Graph Terminology

- Weight - an edge can have a weight (cost)
  - Notation: w(u,v)
- Path - a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 <= i <= k$
- Path length - the number of edges on the path (k-1)
- There is a path of length zero from every vertex to itself, i.e. shortest path.



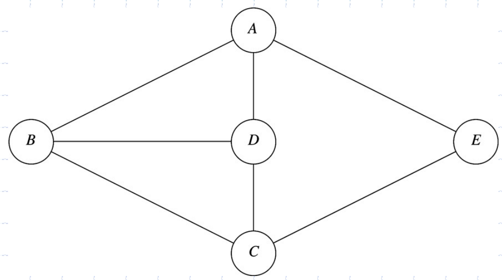
Graph Terminology

- Simple Path
  - all vertices are distinct, except possibly first and last
- Cycle (directed graph)
  - a path of length at least 1 such that $v_1 = v_k$
- Cycle (undirected graph)
  - edges must be distinct
- Connected
  - undirected - a path from every vertex to every other vertex
  - directed
    - Strongly Connected - direction of edges used
    - Weakly Connected - direction of edges not used



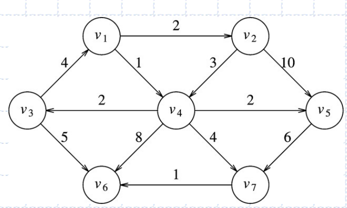
Undirected Graphs

- degree of vertex - # of edges connected to vertex
- complete graph (clique) - all pairs of vertices connected by an edge



Directed Graphs

- in-degree - number of incoming edges
- out-degree - number of outgoing edges



Representation

- Adjacency list:
  - For each node, list its neighbors on outgoing edges
  - Good for sparse graphs
  - For weighted graphs, store weight in linked list
- Adjacency matrix:
  - bit (0,1) matrix to represent presence of edge
  - good for dense graphs
  - weighted - can store weight in matrix
- Tradeoffs
  - Adjacency list: space efficient, easy to grow
  - Adjacency matrix: fast access, but $O(v^2)$ space

Breadth First Search (BFS)

- Distance refers to number of vertices in path
- Visit vertices in increasing order of distance from starting point
- Use a queue to store vertices “to visit”
- Not necessarily unique shortest path

BFS(G,s)

```
for each vertex u in V[G] - [s] do
        color[u] = WHITE (White means undiscovered)
        d[u] = Infinity (distance from s)
        pr[u] = NIL (previous vertex)
color[s] = GRAY, d[s] = 0, pr[s] = NIL, Q = {}
ENQUEUE(Q,s)
while Q ≠ {} do
        u = DEQUEUE(Q)
        for each v in Adjacent[u] do
            if color[v] == WHITE then
                color[v] = GRAY (Gray means discovered, but not expanded)
                d[v] = d[u] + 1
                pr[v] = u
                ENQUEUE(Q, v)
        color[u] = BLACK (Black means expanded)
```

Depth First Search (DFS)

- Explores depth first, then neighbors
  - like pre-order, post-order traversal of trees
- DFS depth not necessarily same as BFS distance
- Discovers vertices before we visit
- Use a stack to store vertices "to visit"
- DFS order not necessarily unique

- http://xkcd.com/761/

DFS(G)

```
for each u in V[G] do
     color[u] = WHITE
     pr[u] = NIL
for each u in V[G] do
     if color[u] == WHITE then
         DFS-VISIT(u)
```

DFS-VISIT(u)

```
color[u] = GRAY (discovered u)
for each v in Adjacent[u] do
     if color[v] == WHITE then
        pr[v] = u
        DFS-VISIT(v)
color[u] = BLACK
```

Single Source Shortest Path

- Given:
  - A graph G = (V,E)
  - A single starting vertex s
- Find shortest path to all other vertices
- For un-weighted graph use BFS
- For weighted graph we have choices
  - Dijkstra's Algorithm (positive weights)
  - Bellman-Ford Algorithm (negative too)

Applications

- Street traffic routing
- Plane trip planning
- Network packet routing

Naïve solution

- Enumerate all routes from A to B
- Add up distances for each route
- Select shortest
- Could be millions of possibilities
  - If cycle, then infinite

Dijkstra's Algorithm

- 50 year old greedy algorithm
- Idea is to keep distance estimates to neighbors of set S
- Add neighbor with shortest distance to set S and update other neighbors
- Doesn't work with negative edges (Bellman-Ford Algorithm does)
- $O(|E|)$ for dense graphs = $O(|V|^2)$
- $O(|E| \log |V|)$ for sparse graphs + priorityQ

Initialization

INITIALIZE-SINGLE-SOURCE(G,s)
for each vertex $\in$ V[G]
  d[v] = INFINITY //shortest path estimate
  pr[v] = NIL
d[s] = 0

Relaxation

- Relaxes the constraint
  - d[v] = d[u] + w(u,v)
- Relax if edge between u and v gives us an improvement

RELAX(u,v,w) //w is edge weight factor
if d[v] > d[u] + w(u,v) then
  d[v] = d[u] + w(u,v)
  pr[v] = u

Dijkstra's Algorithm

```
DIJKSTRA(G,w,s)
  INITIALIZE-SINGLE-SOURCE(G,s)
  S = {} //set S (cloud)
  Q = V[G]
  while Q ≠ {} do
     u = EXTRACT-MIN(Q)
     INSERT(S,u)
     for each vertex v ∈ Adjacent[u]
        RELAX(u,v,w)
```


Dijkstra's on the web

- Dijkstra's algorithm presentation
- Dijkstra's algorithm animation