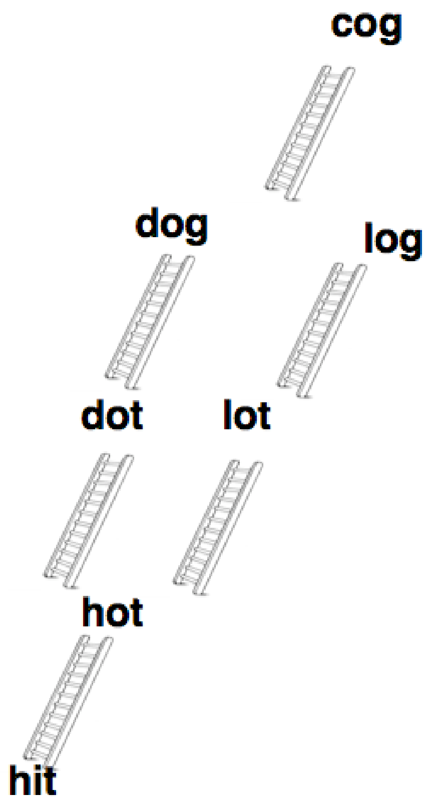
CompSci 100e
Program Design and Analysis II



April 19, 2011

Prof. Rodger

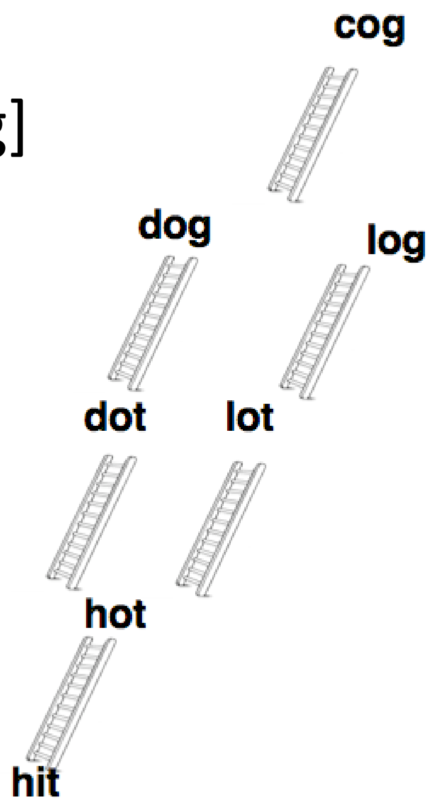
Announcements

- Huffman due Thursday, April 21
- APTS (3) due Tuesday, April 26
  - Talk about AllWordLadders APT today, Internet APT next time
  - Can do extra APTs
- Extra credit assignments due Wed. April 27

- NOTE: NO LATE ASSIGNMENTS accepted after Wed, April 27 11:59pm!!!

Word Ladder APT

- From->[words]->to
  - From hit to cog via [hot,dot,lot,dog,log]
- What words reachable from 'from'?
  - Repeat until we get to 'cog'
- Problem: reachable from 'dot'
  - Why not include 'hot'?
  - Don't re-use words
- Algorithm:
  - Find all words 1-away
  - From each n-away find (n+1)-away

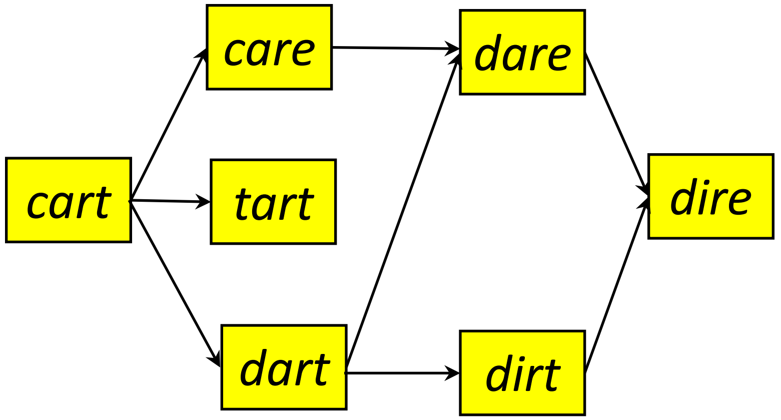


Digression: word ladders

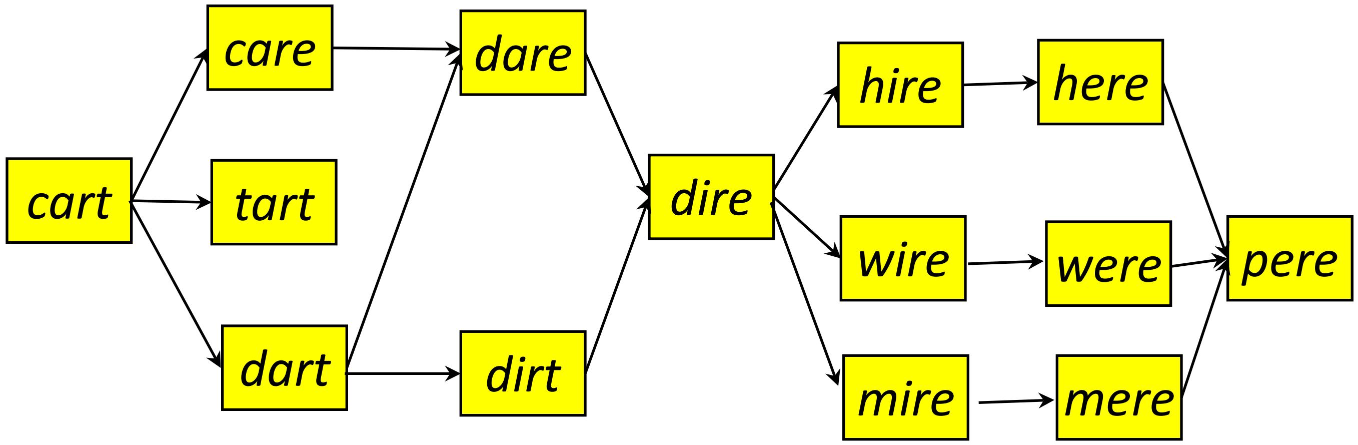
- How many ladders from *cart* to *dire* as shown?
  - Enqueue *dare* more than once?
  - Downside? Alternative?

- We want to know number of ladders that end at *W*.
  - What do we know initially?
  - When we put something on the queue, what do we know?
  - How do we keep track?

- Initialize and update per-word statistics

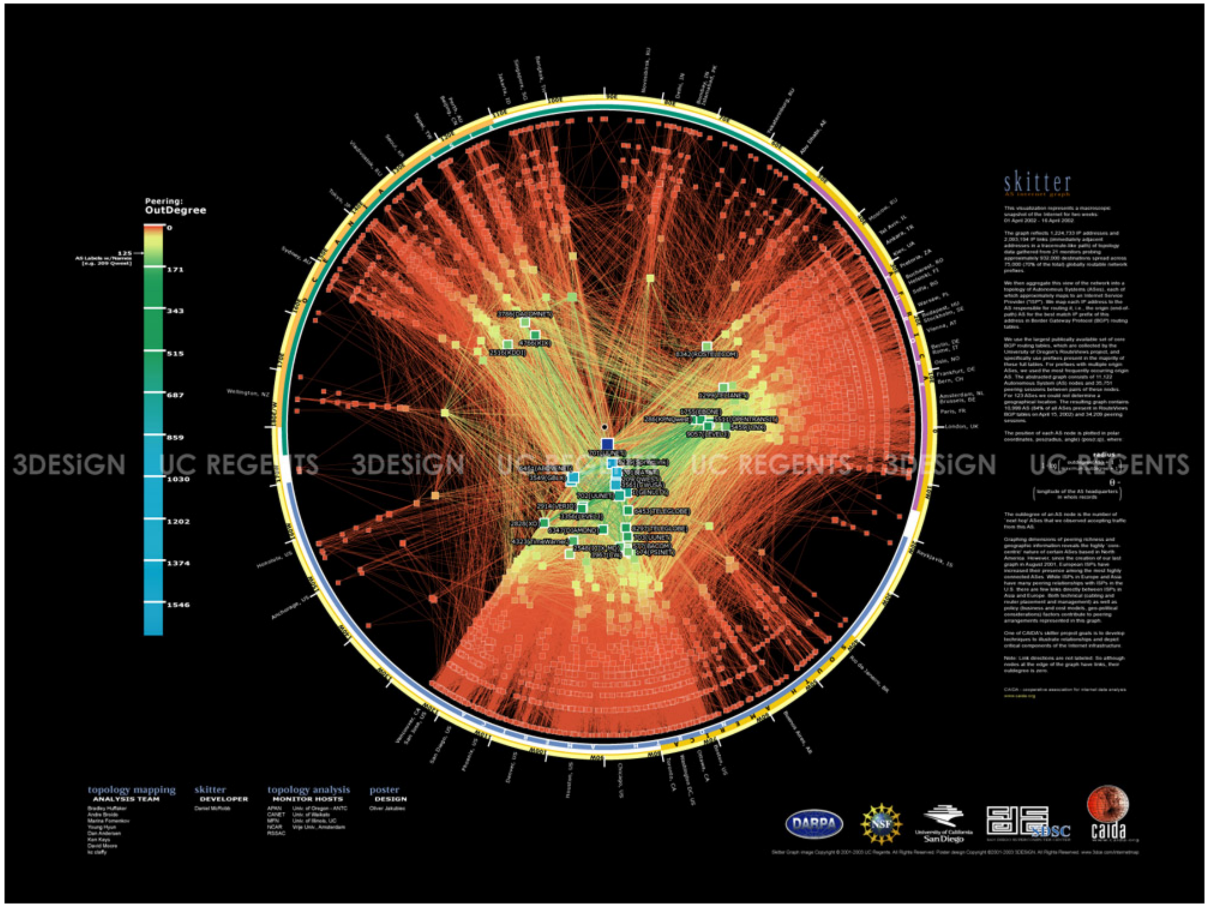


Word Ladder: more details



- # ladders that end at dare
  - At each word *W*
- Ladder length to *W*
  - Calculable from??
- Maps? Queue? Sets?

Graphs, the Internet, and Everything



http://www.caida.org/

Graphs: Structures and Algorithms

- Mapquest, Tomtom, Garmin, Googlemap
  - How do you get from here to there?
  - What's a route? How is this known?
- What about The Oracle of Bacon, Erdos Numbers, and Word Ladders?
  - All can be modeled using graphs
  - What kind of connectivity does each concept model?
- Graphs are everywhere in the world (of algorithms?)
  - What is a graph? Algorithms on graphs?
  - Graph representation?

Sir Tim Berners-Lee

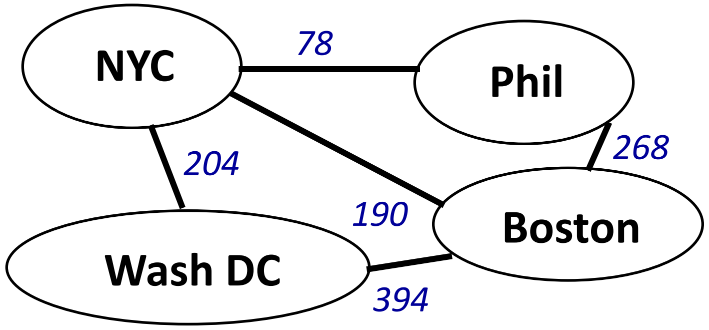
I want you to realize that, if you can imagine a computer doing something, you can program a computer to do that.

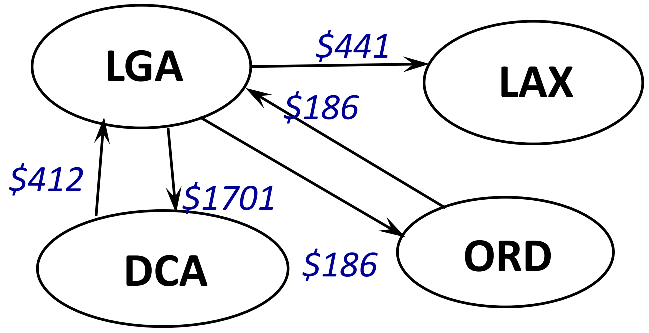
Unbounded opportunity... limited only by your imagination. And a couple of laws of physics.

- TCP/IP, HTTP
  - How, Why, What, When?

Vocabulary

- Graphs are collections of *vertices* and *edges* (vertex also called node)
  - Edge connects two *vertices*
    - Direction can be important, *directed edge, directed graph*
    - Edge may have associated weight/cost
- A vertex sequence $v_0, v_1, \ldots, v_{n-1}$ is a *path* where v_k and v_{k+1} are connected by an edge.
  - If some vertex is repeated, the path is a *cycle*
  - A graph is *connected* if there is a path between any pair of vertices





Graph questions/algorithms

- What vertices are reachable from a given vertex?
  - Two standard traversals: depth-first, breadth-first
  - *connected components*, groups of connected vertices

- Shortest path between two vertices (weighted graphs?)
  - BFS works, possibly uses more storage than DFS
  - Dijkstra's algorithm efficient, uses a priority queue!

- Longest path in a graph
  - No known efficient algorithm

- Visit all vertices without repeating? Visit all edges?
  - With minimal cost? Hard!

Depth, Breadth, other traversals

- We want to visit every vertex that can be reached from a specific starting vertex (we might try all starting vertices)
  - Make sure we don't visit a vertex more than once
    - Why isn't this an issue in trees?
    - Mark vertex as visited, use set/array/map for this
  - Order in which vertices visited can be important
  - Storage/runtime efficiency of traversals important

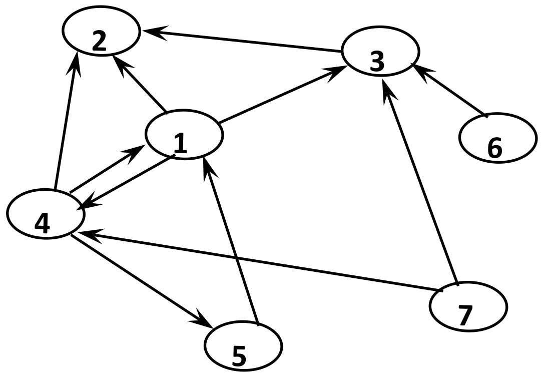
- What other data structures do we have: stack, queue, …
  - What if we traverse using priority queue?

Breadth first search

- In an unweighted graph this finds the shortest path between a start vertex and every vertex
  - Visit every node one away from start
  - Visit every node two away from start
    - This is nodes one away from a node one away
  - Visit every node three away from start, …
- Put vertex on queue to start (initially just one)
  - Repeat: dequeue vertex, enqueue adjacent vertices
  - Avoid enqueueing already visited/queued nodes
  - When are 1-away vertices enqueued? 2-away? N?
  - How many vertices on queue?

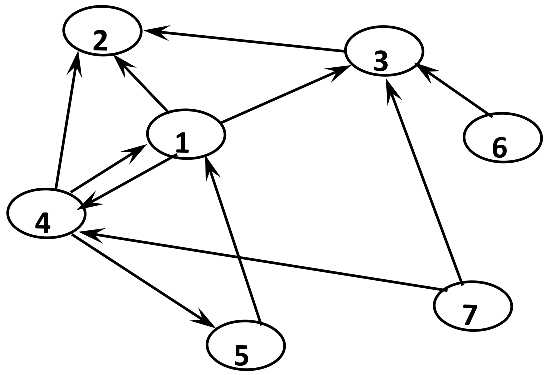
Code for breadth first

```
public void breadth(String vertex){
    Set<String> visited = new TreeSet<String>();
    Queue<String> q = new LinkedList<String>();
    q.add(vertex);
    visited.add(vertex);
    while (q.size() > 0) {
       String current = q.remove();
       // process current
       for(each v adjacent to current){
         if (!visited.contains(v)){// not visited
            visited.add(v);
            q.add(v);
         }
       }
    }
}
```



Pseudo-code for depth-first search

```
void depthfirst(String vertex){
    if (! alreadySeen(vertex)){
       markAsSeen(vertex);
       System.out.println(vertex);
       for(each v adjacent to
  vertex) {
          depthfirst(v);
       }
    }
}
```



- Clones are stacked up, problem? Can we make use of stack explicit?

BFS compared to DFS

```
public Set<String> bfs(String start){
    Set<String> visited = new TreeSetString>();
    Queue<String> qu = new LinkedList<String>();
    visited.add(start);
    qu.add(start);

    while (qu.size() > 0){
        String v = qu.remove();
        for(String adj : myGraph.getAdjacent(v)){
            if (! visited.contains(adj)) {
                visited.add(adj);

                qu.add(adj);
            }
        }
    }
    return visited;
}
```

BFS becomes DFS

```
public Set<String> dfs(String start){
    Set<String> visited = new TreeSet<String>();
    Queue<String> qu = new LinkedList<String>();
    visited.add(start);
    qu.add(start);

    while (qu.size() > 0){
        String v = qu.remove();
        for(String adj : myGraph.getAdjacent(v)){
            if (! visited.contains(adj)) {
                visited.add(adj);

                qu.add(adj);
            }
        }
    }
    return visited;
}
```

DFS arrives

```
public Set<String> dfs(String start){
    Set<String> visited = new TreeSet<String>();
    Stack<String> qu = new Stack<String>();
    visited.add(start);
    qu.push(start);

    while (qu.size() > 0){
        String v = qu.pop();
        for(String adj : myGraph.getAdjacent(v)){
            if (! visited.contains(adj)) {
                visited.add(adj);

                qu.push(adj);
            }
        }
    }
    return visited;
}
```

What is the Internet?

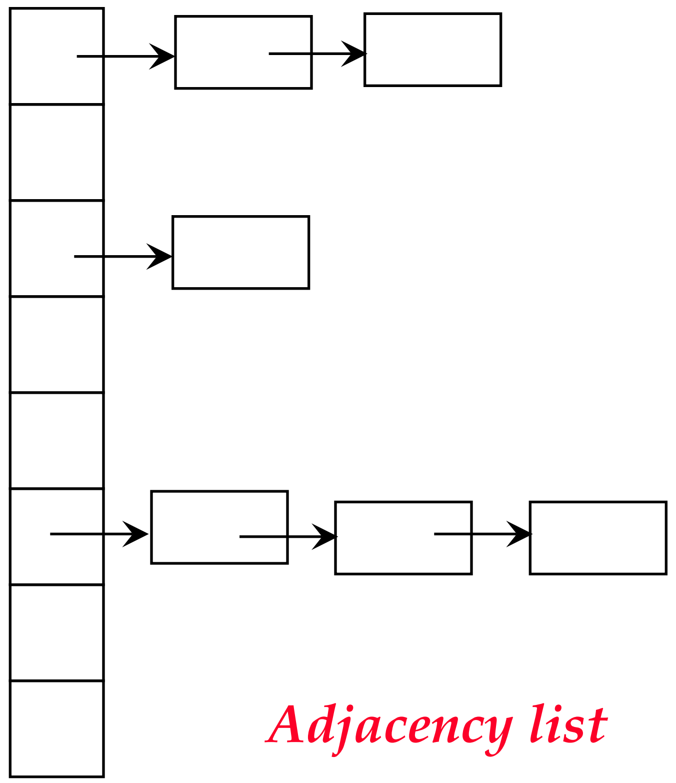
- The Internet was originally designed as an "overlay" network running on top of existing phone and other networks. It is based on a small set of software protocols that direct routers inside the network to forward data from source to destination, while applications run on the Internet to rapidly scale into a critical global service. However, this success now makes it difficult to create and test new ways of protecting it from abuses, or from implementing innovative applications and services.

http://www.intel.com/labs/features/idf09041.htm

Graph implementations

- Typical operations on graph:
  - Add vertex
  - Add edge (parameters?)
  - getAdjacent(vertex)
  - getVertices(..)
  - String->Vertex (vice versa)

- Different kinds of graphs
  - Lots of vertices, few edges, *sparse* graph
    - Use adjacency list
  - Lots of edges (max # ?) *dense* graph
    - Use adjacency matrix



Adjacency list

Graph implementations (continued)

- Adjacency matrix
  - Every possible edge represented, how many?
- Adjacency list uses O(V+E) space
  - What about matrix?
  - Which is better?

	T	**F**	**…**		

- What do we do to get adjacent vertices for given vertex?
  - What is complexity?
  - Compared to adjacency list?

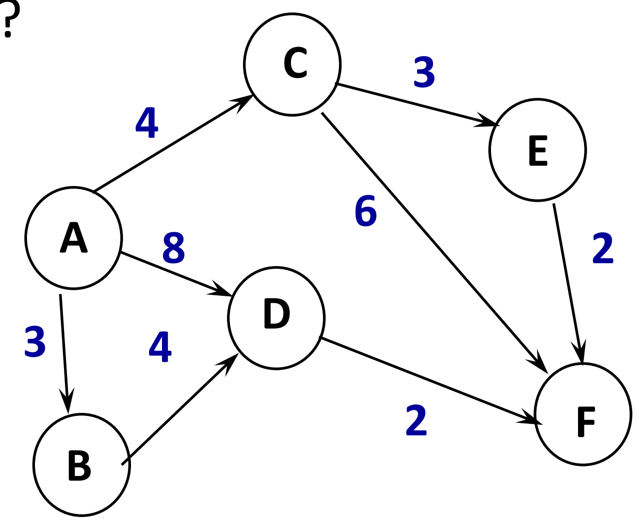
- What about weighted edges?

Shortest path in weighted graph

- We need to modify approach slightly for weighted graph
  - Edges have weights, breadth first doesn't work
  - What's shortest path from A to F in graph below?

- Use same idea as breadth first search
  - Don't add 1 to current distance, add ???
  - Might adjust distances more than once
  - What vertex do we visit next?

- What vertex is next is key
  - Use greedy algorithm: closest
  - Huffman is greedy, …



What about connected components?

- What computers are reachable from this one? What people are reachable from me via acquaintanceship?
  - Start at some vertex, depth-first search (breadth?)
    - Mark nodes visited
  - Repeat from unvisited vertex until all visited

- What is minimal size of a component? Maximal size?
  - What is complexity of algorithm in terms of V and E?

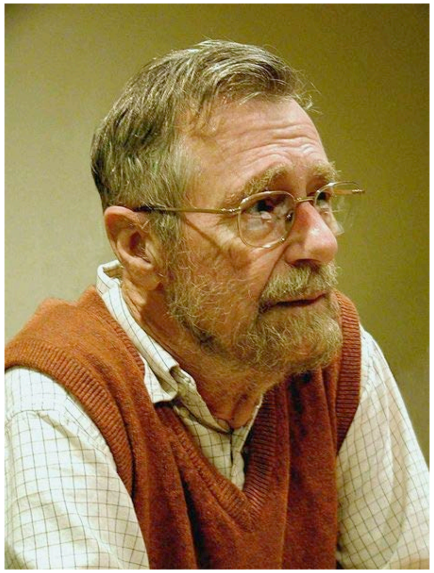
- What algorithms does this lead to in graphs?

Greedy Algorithms Reviewed

- A greedy algorithm makes a locally optimal decision that leads to a globally optimal solution
  - Huffman: choose minimal weight nodes, combine
    - Leads to optimal coding, optimal Huffman tree
  - Making change with American coins: choose largest coin possible as many times as possible
    - Change for $0.63, change for $0.32
    - What if we're out of nickels, change for $0.32?

- Greedy doesn't always work, but it does sometimes
- Weighted shortest path algorithm is *Dijkstra's* algorithm, greedy and uses priority queue

Edsger Dijkstra



- Turing Award, 1972
- Algol-60 programming language
- Goto considered harmful
- Shortest path algorithm
- Structured programming

"Program testing can show the presence of bugs, but never their absence"

For me, the first challenge for computing science is to discover how to maintain order in a finite, but very large, discrete universe that is intricately intertwined. And a second, but not less important challenge is how to mould what you have achieved in solving the first problem, into a teachable discipline: it does not suffice to hone your own intellect (that will join you in your grave), you must teach others how to hone theirs. The more you concentrate on these two challenges, the clearer you will see that they are only two sides of the same coin: teaching yourself is discovering what is teachable EWD 709

Dijkstra’s Shortest Path Algorithm

- Similar to breadth first search, but uses a priority queue instead of a queue. Code below is for breadth first search (distance[] replaces set)

```
Vertex cur = q.remove();
for(Vertex v : adjacent(cur)){
  if (!visited.contains(v)){ // if distance[v] == INFINITY
      visited.add(v);        // distance[v] = distance[cur]+1
      q.add(v);
  }
}
```

- Dijkstra: Find minimal unvisited node, recalculate costs through node

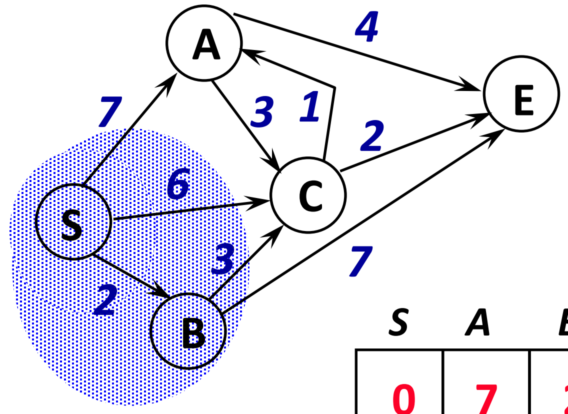
```
Vertex cur = pq.remove();
for(Vertex v : adjacent(cur))
  if (distance[cur] + graph.weight(cur,v) < distance[v]) {
     distance[v] = distance[cur] + graph.weight(cur,v);
     pq.add(v);
  }
```

Shortest paths, more details

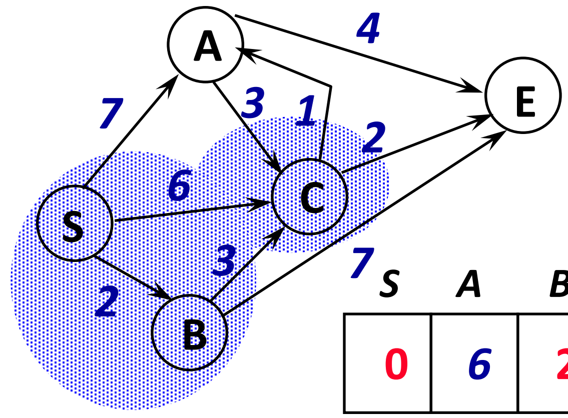
- Single-source shortest path
  - Start at some vertex S
  - Find shortest path to every reachable vertex from S
- A set of vertices is *processed*
  - Initially just S is processed
  - Each pass processes a vertex

After each pass, shortest path from S to any vertex using just vertices from processed set (except for last vertex) is always known

- Next processed vertex is closest to S still needing processing



	S	A	B	C	E	
	0	7	2	6	∞	
process B	*0*	*7*	*2*	*5*	*9*	



S	A	B	C	E	
0	*6*	2	5	*7*	

process C

Dijkstra’s algorithm works (greedily)

- Choosing minimal unseen vertex to process leads to shortest paths

```
Vertex cur = pq.remove();
for(Vertex v : adjacent(cur))
 if (distance[cur]+graph.weight(cur,v) < distance[v]){
     distance[v] = distance[cur] + graph.weight(cur,v);
     pq.add(v);
  }
}
```

- We always know shortest path through processed vertices
  - When we choose *w*, there can’t be a shorter path to *w* than distance[w] – it would go through processed *u,* we would have chosen *u* instead of *w*

Shafi Goldwasser

- RCS professor of computer science at MIT
  - Twice Godel Prize winner
  - Grace Murray Hopper Award
  - National Academy
  - Co-inventor of zero-knowledge proof protocols

  How do you convince someone that you know [a secret] without revealing the knowledge?
- Honesty and Privacy

Work on what you like, what feels right, I know of no other way to end up doing creative work