How do packets of bits/information get routed on the internet

- Message divided into packets on client (your) machine
- Packets sent out using routing tables toward destination
  - Packets may take different routes to destination
  - What happens if packets lost or arrive out-of-order?
- Routing tables store local information, not global (why?)

- 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 (world?)
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 v₀ v₁ ... vₙ₋₁ is a path where vᵢ and vᵢ₊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
  - Find connected components, groups of connected vertices
- Shortest path between any two vertices (weighted graphs?)
  - Breadth first search is storage expensive
  - Dijkstra’s algorithm is efficient, uses a priority queue too!
- 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/vector/map for this
      - Can keep useful information to help with visited status
  - Order in which vertices visited can be important
  - Storage and runtime efficiency of traversals important
- What other data structures do we have: stack, queue, ...
  - What happens when 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 every node one away from a node one away
  - Visit every node three away from start, ...
- Put vertex on queue to start (initially just one)
  - Repeat: take vertex off queue, put all adjacent vertices on
  - Don’t put a vertex on that’s already been visited (why?)
  - When are 1-away vertices enqueued? 2-away? 3-away?
  - How many vertices on queue?

Code for breadth first

```cpp
void breadthfirst(const string& vertex) // post: breadth-first search done
{
    tset<string> visited;
    tqueue<string> q; q.enqueue(vertex);
    visited.insert(vertex);
    while (q.size() > 0) {
        string current;
        q.dequeue(current);
        // process current
        for (each v adjacent to current){
            if (!visited.contains(v)){ // not visited
                visited.insert(v);
                q.enqueue(v);
            }
        }
    }
}
```

Pseudo-code for depth-first search

```cpp
void depthfirst(const string& vertex) // post: depth-first search done
{
    if (! alreadySeen(vertex)) {
        markAsSeen(vertex);
        cout << vertex << endl;
        for (each v adjacent to vertex) {
            depthfirst(v);
        }
    }
}
```

Clones are stacked up, problem? When are all doors out of vertex opened and visited? Can we make use of stack explicit?

Depth first with stack/no recursion

```cpp
void depthfirst(const string& vertex) // post: depth-first search from vertex complete
{
    tset<string> visited;
    stack<string> st; st.push(vertex);
    visited.insert(vertex); // mark this room
    while (st.size() > 0) {
        string current;
        st.pop(current);
        // process current
        for (each v adjacent to current){
            if (!visited.contains(v)){ // not visited
                visited.insert(v);
                st.push(v);
            }
        }
    }
}
```
Depth and Breadth compared

void breadth(const string& vertex)  
// post: breadth-first search done  
{  
    tset<string> visited; tqueue<string> q;  
    q.enqueue(vertex);  
    visited.insert(vertex);  
    while (q.size() > 0) {  
        string current; q.dequeue(current);  
        // process current  
        for (v adjacent to current){  
            if (!visited.contains(v)){  
                visited.insert(v);  
                q.enqueue(v);  
            }  
        }  
    }  
}

void depth(const string& vertex)  
// post: depth-first search done  
{  
    tset<string> visited; tstack<string> st;  
    st.push(vertex);  
    visited.insert(vertex);  
    while (st.size() > 0) {  
        string current; st.pop(current);  
        // process current  
        for (v adjacent to current){  
            if (!visited.contains(v)){  
                visited->insert(v);  
                st.push(v);  
            }  
        }  
    }  
}

Graph implementations

- Typical operations on graph:
  - Add vertex
  - Add edge (parameters?)
  - AdjacentVerts(vertex)
  - AllVert(...)
  - String->int (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

Graph implementations (continued)

- Adjacency matrix
  - Every possible edge represented, how many?
- Adjacency list uses O(V+E) space
  - What about matrix?
  - Which is better?
- What do we do to get adjacent vertices for given vertex?
  - What is complexity?
  - Compared to adjacency list?
- What about weighted edges?

What about word ladders

- Find a path from white->house changing one letter
  - Real world? Computer vs. human?
  - white write writs waits warts ports forts forte  
  - ... rouse house
- See ladderXXX.cpp programs
- How is this a graph problem? What are vertices/edges?
- What about spell-checking, how is it similar?
  - Edge from accomodate to accommodate  
  - Can also use tries with wild-cards, e.g., acc*date
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 (why not breadth)?
    - Mark nodes visited
    - Repeat, starting from an 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?

Shortest path in weighted graph

- We need to modify approach slightly for weighted graph
  - Edges have weights, breadth first by itself 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, ...

Greedy Algorithms

- A greedy algorithm makes a locally optimal decision that leads to a globally optimal solution
  - Huffman: choose two nodes with minimal weight, 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
- Operating systems and concurrency
- Algol-60 programming language
- Goto considered harmful
- Shortest path algorithm
- Structured programming
  “Program testing can show the presence of bugs, but never their absence”
- A Discipline of programming
  “For the absence of a bibliography I offer neither explanation nor apology”
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.

```c
q.dequeue(vertex w)
foreach (vertex v adjacent to w)
    if (distance[v] == INT_MAX) // not visited
        distance[v] = distance[w] + 1;
        q.enqueue(v);

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

```q.deletemin(vertex w)

```c
foreach (vertex v adjacent to w)
    if (distance[w] + weight(w,v) < distance[v])
        distance[v] = distance[w] + weight(w,v);
        q.insert(vertex(v, distance[v]));
```

Dijkstra’s algorithm works (greedily)

- Choosing minimal unseen vertex to process leads to shortest paths.

```c
q.deletemin(vertex w)
foreach (vertex v adjacent to w)
    if (distance[w] + weight(w,v) < distance[v])
        distance[v] = distance[w] + weight(w,v);
        q.insert(vertex(v, distance[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, then we would have chosen u instead of w.

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.

Greedy Algorithms

- Huffman compression is a greedy algorithm that works
  - Where is “greed” used

- Dijkstra’s algorithm is a greedy algorithm that works
  - Which vertex visited?

- Prim’s Minimal-spanning algorithm (see prim.cpp) works
  - How is this algorithm greedy?

- Making change in US is a greedy algorithm that works
  - Minimal coins for change of $0.75, $0.72, …
  - What if we don’t have nickels: change for $0.32?
Topological sort

- Given a directed acyclic graph (DAG)
  - Order vertices so that any if there is an edge \((v,w)\), then \(v\) appears before \(w\) in the order

- Prerequisites for a major, take CPS 100 before CPS 130
  - Edge(cps100,cps130)
  - Topological sort gives an ordering for taking courses

- Where does ordering start?
  - First vertex has no prereqs
  - “remove” this vertex, continue
  - Depends on in-degree

Prerequisites for a major, take CPS 100 before CPS 130

CPS 100

CPS 130

0 1 2 3 4 5 6

0  belt
1  shirt
2  jacket
3  shoes
4  socks
5  pants
6  underwear

0  1   2   3   4  5   6