Graph Traversals

- Breadth-first search (and depth-first search) work for arbitrary (directed or undirected) graphs - not just mazes!
  - Must **mark** visited vertices so you do not go into an infinite loop!
- Either can be used to determine **connectivity**:
  - Is there a path between two given vertices?
  - Is the graph (weakly) connected?
- Which one:
  - Uses a queue?
  - Uses a stack?
  - Always finds the **shortest path** (for unweighted graphs)?

Graph Connectivity

**Undirected** graphs are **connected** if there is a **path between any two vertices**

**Directed** graphs are **strongly connected** if there is a **path from any one vertex to any other**

**Directed** graphs are **weakly connected** if there is a **path between any two vertices, ignoring direction**

A **complete** graph has an **edge** between every pair of vertices

Today’s Outline

Shortest path algorithms
1. Unweighted graphs: BFS
2. Weighted graphs without negative cost edges: Dijkstra’s Algorithm
3. Negative cost edges but no negative cost cycles

Reading: Weiss, Ch. 9
The Shortest Path Problem

Given a graph $G$, edge costs $c_{i,j}$, and vertices $s$ and $t$ in $G$, find the shortest path from $s$ to $t$.

For a path $p = v_0 \, v_1 \, v_2 \, \ldots \, v_k$

- *unweighted length* of path $p = k$ (a.k.a. length)
- *weighted length* of path $p = \sum_{i=0}^{k-1} c_{i,i+1}$ (a.k.a. cost)

Path length equals path cost when ?

---

Single Source Shortest Paths (SSSP)

Given a graph $G$, edge costs $c_{i,j}$, and vertex $s$, find the shortest paths from $s$ to all vertices in $G$.

- Is this harder or easier than the previous problem?

---

All Pairs Shortest Paths (APSP)

Given a graph $G$ and edge costs $c_{i,j}$, find the shortest paths between all pairs of vertices in $G$.

- Is this harder or easier than SSSP?

- Could we use SSSP as a subroutine to solve this?

---

Variations of SSSP

- Weighted vs. unweighted
- Directed vs undirected
- Cyclic vs. acyclic
- Positive weights only vs. negative weights allowed
- Shortest path vs. longest path
- …
Applications

- Network routing
- Driving directions
- Cheap flight tickets
- Critical paths in project management
  (see textbook)
- ...

SSSP: Unweighted Version

Idea?

```cpp
void Graph::unweighted (Vertex s){
    Queue q(NUM_VERTICES);
    Vertex v, w;
    q.enqueue(s);
    s.dist = 0;

    while (!q.isEmpty()){
        v = q.dequeue();
        for each w adjacent to v
            if (w.dist == INFINITY){
                w.dist = v.dist + 1;
                w.path = v;
                q.enqueue(w);
            }
    }
}
```

Each edge examined at most once – if adjacency lists are used
Each vertex enqueued at most once

Weighted SSSP:
The Quest For Food

Can we calculate shortest distance to all nodes from Allen Center?
Dijkstra, Edsger Wybe

Legendary figure in computer science; was a professor at University of Texas.

Supported teaching introductory computer courses without computers (pencil and paper programming)

Supposedly wouldn’t (until very late in life) read his e-mail; so, his staff had to print out messages and put them in his box.

1972 Turning Award Winner, Programming Languages, semaphores, and …

Dijkstra’s Algorithm: Idea

Adapt BFS to handle weighted graphs

Two kinds of vertices:
- Finished or known vertices
  - Shortest distance has been computed
- Unknown vertices
  - Have tentative distance

Dijkstra’s Algorithm: Idea

At each step:
1) Pick closest unknown vertex
2) Add it to known vertices
3) Update distances

Dijkstra’s Algorithm: Pseudocode

Initialize the cost of each node to $\infty$

Initialize the cost of the source to 0

While there are unknown nodes left in the graph
1. Select an unknown node $b$ with the lowest cost
2. Mark $b$ as known
3. For each node $a$ adjacent to $b$
   - $a$’s cost = min($a$’s old cost, $b$’s cost + cost of $(b, a)$)
void Graph::dijkstra(Vertex s){
    Vertex v,w;

    Initialize s.dist = 0 and set dist of all other vertices to infinity

    while (there exist unknown vertices, find the one b with the smallest distance)
        b.known = true;
        for each a adjacent to b
            if (!a.known)
                if (b.dist + Cost_ba < a.dist){
                    decrease(a.dist to= b.dist + Cost_ba);
                    a.path = b;
                }
    }
}

Dijkstra’s Alg: Implementation

Initialize the cost of each node to $\infty$
Initialize the cost of the source to 0
While there are unknown nodes left in the graph
    Select the unknown node b with the lowest cost
    Mark b as known
    For each node a adjacent to b
        a’s cost = min(a’s old cost, b’s cost + cost of (b, a))

What data structures should we use?

Running time?

Dijkstra’s Algorithm: Summary

- Classic algorithm for solving SSSP in weighted graphs without negative weights
- A greedy algorithm (irreversibly makes decisions without considering future consequences)

- Intuition for correctness:
  - shortest path from source vertex to itself is 0
  - cost of going to adjacent nodes is at most edge weights
  - cheapest of these must be shortest path to that node
  - update paths for new node and continue picking cheapest path
Correctness: The Cloud Proof

How does Dijkstra’s decide which vertex to add to the Known set next?
- If path to $V$ is shortest, path to $W$ must be at least as long (or else we would have picked $W$ as the next vertex)
- So the path through $W$ to $V$ cannot be any shorter!

Correctness: Inside the Cloud

Prove by induction on # of nodes in the cloud:
- Initial cloud is just the source with shortest path $0$
- Assume: Everything inside the cloud has the correct shortest path
- Inductive step: Only when we prove the shortest path to some node $v$ (which is not in the cloud) is correct, we add it to the cloud

When does Dijkstra’s algorithm not work?

Dijkstra’s vs BFS

At each step:
1) Pick closest unknown vertex
2) Add it to finished vertices
3) Update distances

\textit{Dijkstra’s Algorithm}

Some Similarities:

Breadth-first Search

The Trouble with Negative Weight Cycles

What’s the shortest path from A to E?

Problem?