Graphs: Traversals and Shortest Path Algorithms

Graph Traversals

- **Breadth-first search**
  - explore all adjacent nodes, then for each of those nodes explore all adjacent nodes

- **Depth-first search**
  - explore first child node, then its first child node, etc. until goal node is found or node has no children. Then backtrack, repeat with sibling.

- Both:
  - Work for arbitrary (directed or undirected) graphs
  - 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)?

The Shortest Path Problem

Given a graph \( G \), edge costs \( c_{ij} \) and vertices \( s \) and \( t \) in \( G \), find the shortest path from \( s \) to \( t \).

For a path \( p = v_0, v_1, v_2, …, v_k \)
- **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 \( c_{ij} \) and vertex \( s \), find the shortest paths from \( s \) to all vertices in \( G \).

All Pairs Shortest Paths (APSP)

Given a graph \( G \) and edge costs \( c_{ij} \), find the shortest paths between all pairs of vertices in \( G \).

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)
- …

SSP: Unweighted Version

Ideas?

```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);
            }
    }
    // total running time: O(                  )
}
```

Weighted SSSP: The Quest For Food

Can we calculate shortest distance to all nodes from CSE 002?

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 Turing 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
   Select an 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 $(b,a)$)

---

```cpp
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: a Greedy Algorithm

Greedy algorithms always make choices that currently seem the best
– Short-sighted – no consideration of long-term or global issues
– Locally optimal - does not always mean globally optimal!!

Dijkstra’s Algorithm: Summary

• Classic algorithm for solving SSSP in weighted graphs without negative weights
• A greedy algorithm (irrevocably 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