Graphs: Traversals and Shortest Path Algorithms

(Chapter 9)

CSE 373

Data Structures and Algorithms

Today's Outline

- Announcements
  - Homework #6/7 [redacted].

- **Graphs**
  - ~~**Topological Sort**~~
  - **Shortest Paths 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? BFS
  - Uses a stack? DFS
  - Always finds the shortest path (for unweighted graphs)? BFS

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 finding the shortest path from s to t?

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

Ideas?

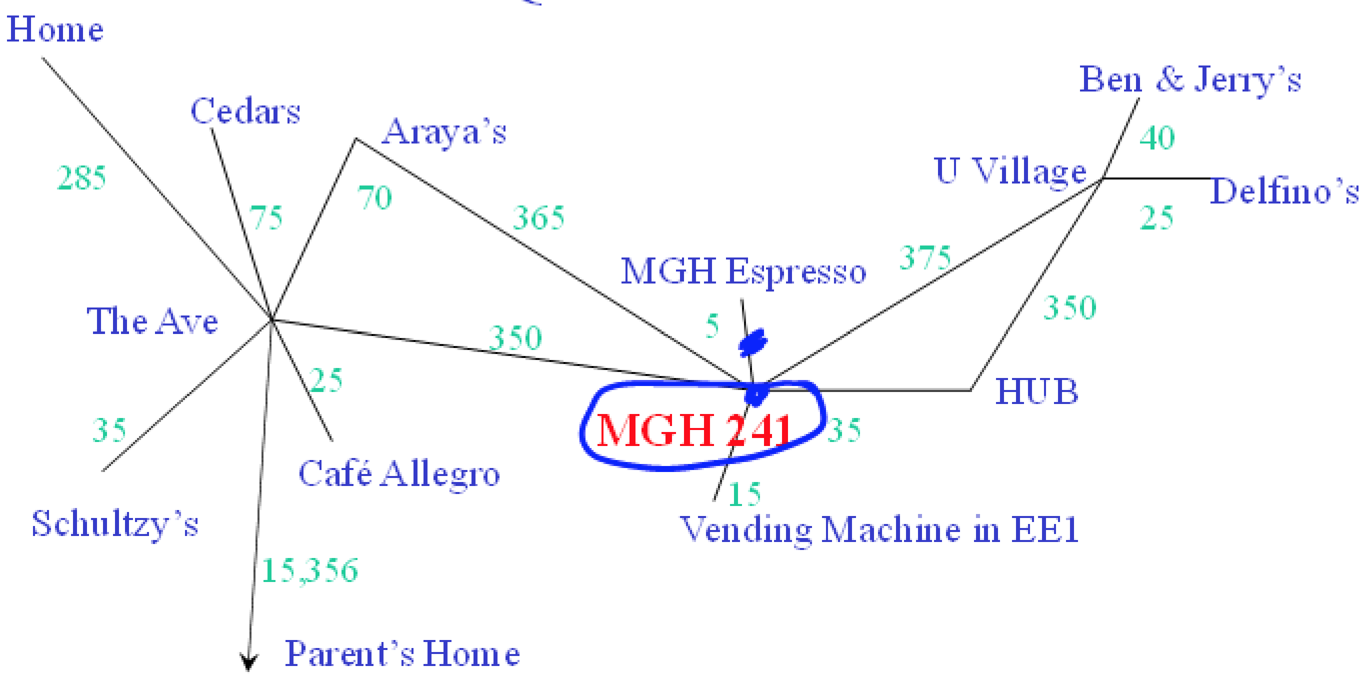
```
void Graph::unweighted (Vertex s){
  Queue q(NUM_VERTICES);
  Vertex v, w;
  for all vertices v do {v.dist = INFINITY;}
  s.dist = 0;
  q.enqueue(s);

  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

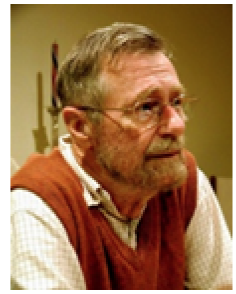
total running time: O( )

Weighted SSSP: The Quest For Food



Can we calculate shortest distance to all nodes from MGH 241?

Edsger Wybe Dijkstra
(1930-2002)

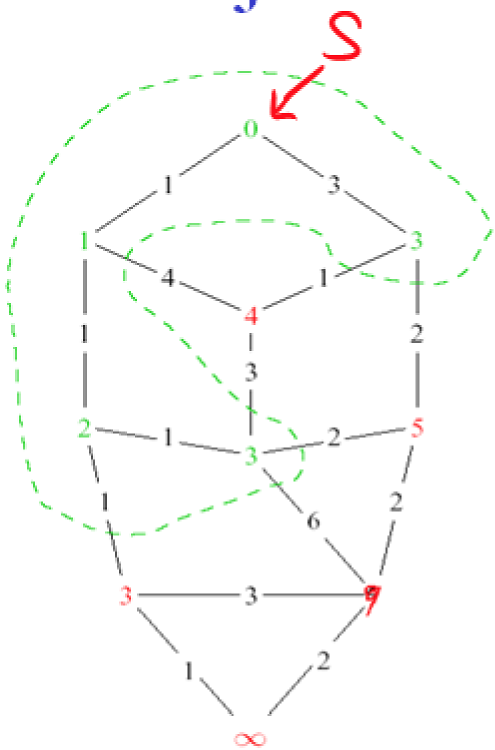


- Legendary figure in computer science; was a professor at University of Texas.
- Invented concepts of structured programming, synchronization, and "semaphores" for controlling computer processes.
- Supported teaching programming without computers (pencil and paper)
- 1972 Turing Award
- "In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind."

Dijkstra's Algorithm for Single Source Shortest Path

- Similar to breadth-first search, but uses a heap instead of a queue:
  - Always select (expand) the vertex that has a lowest-cost path ~~to~~ from the start vertex
- Correctly handles the case where the lowest-cost (shortest) path to a vertex is not the one with fewest edges

Dijkstra's Algorithm: Idea

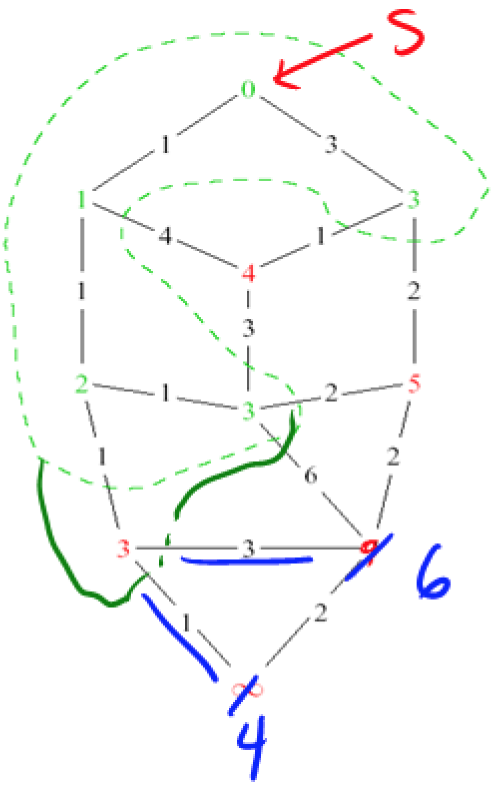


Adapt BFS to handle weighted graphs

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 ∞

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

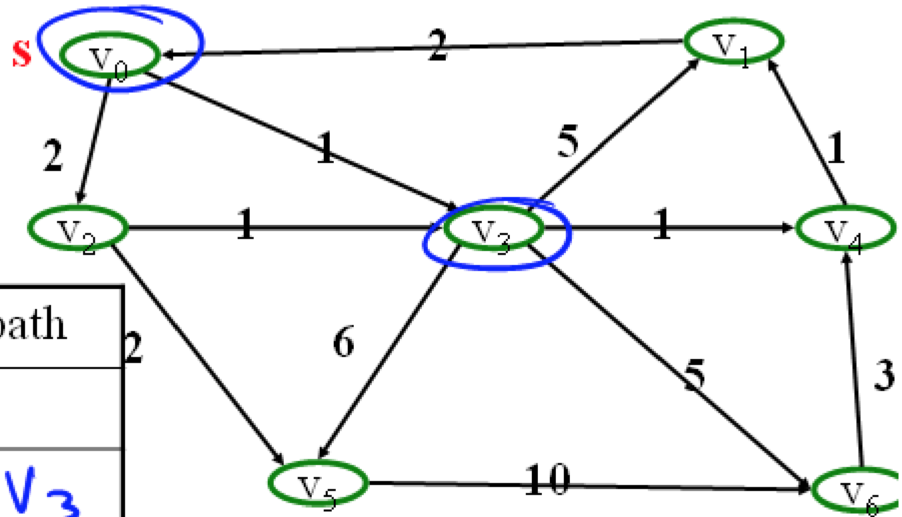
For each node a adjacent to b

a's cost = min(a's old cost, b's cost + cost of (b, a))

```
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;
        }
  }
}
```

Important Features

- Once a vertex is made **`known`**, the cost of the shortest path to that node is known
- While a vertex is still not **`known`**, another shorter path to it might still be found
- The shortest path itself can found by following the backward pointers stored in **`node.path`**



V	Known	Dist	path
v0	T	0	
v1	F	6	V_3
v2	F	2	V_0
v3	T	1	V_0
v4	F	2	V_3
v5	F	7	V_3
v6	F	6	V_3

Order Known

V_0 V_3

Dijkstra's Alg: Implementation

Initialize the cost of each node to ∞

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))

Running time?

Dijkstra's Algorithm: a Greedy Algorithm

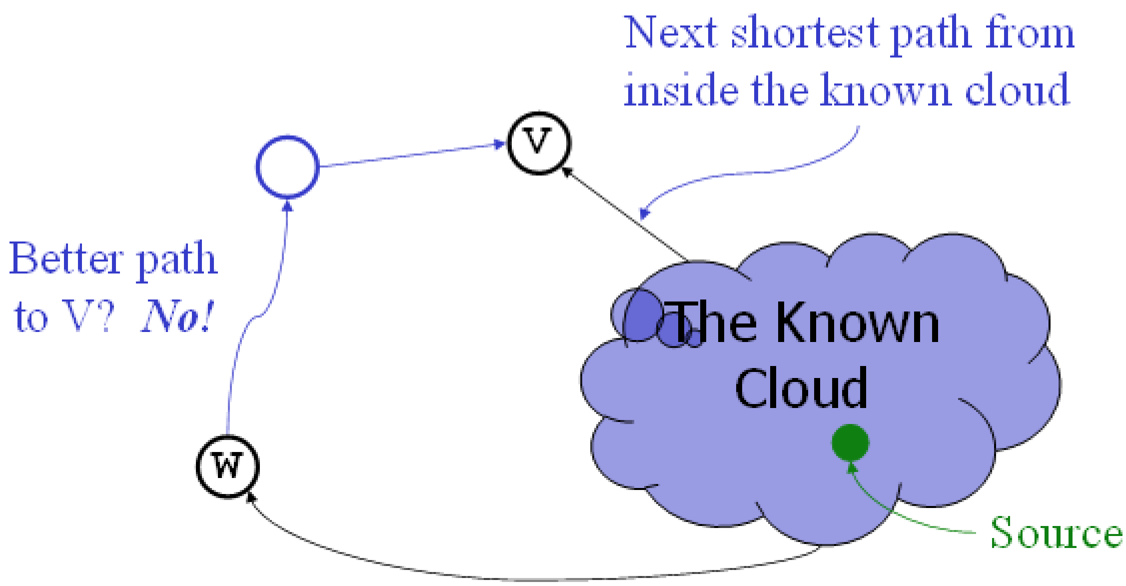
Greedy algorithms always make choices that *currently* seem the best

- 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

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

Dijkstra's Algorithm

At each step:

1) Pick vertex from queue
2) Add it to visited vertices
3) Update queue with neighbors

Breadth-first Search

The Trouble with Negative Weight Cycles

What's the shortest path from A to E?

Problem?