# CSE 326: Data Structures
# Topic 14: O Vertex, Where Art Thou?

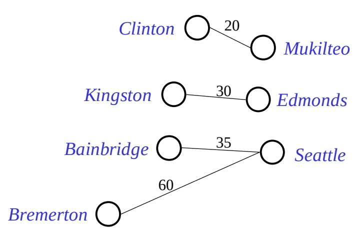
Luke McDowell
Summer Quarter 2003

## Finding the Shortest Path

- Use Breadth-First-Search
- Runtime?

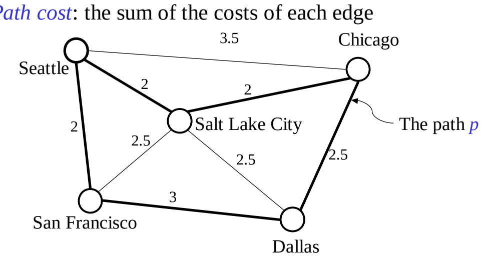
## A Slight Searching Wrinkle: Weighted Graphs

Each edge has an associated weight or cost.



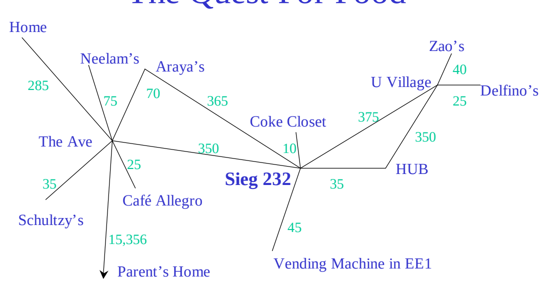

## Diffentiating Between *Path Length* and *Path Cost*

*Path length*: the number of edges in the path
*Path cost*: the sum of the costs of each edge




length(*p*) = cost(*p*) =

## The Quest For Food




*Can we calculate shortest distance to all nodes from Sieg 232?*

## Formally speaking …

Given a graph **`G = (V, E)`** and a vertex **`s`** ? **`V`**, find the shortest path from s to every vertex in **`V`**

Many variations:
- Weighted vs. unweighted
- Cyclic vs. acyclic
- Positive weights only vs. negative weights allowed
- Directed vs undirected graph

Applications?

## Dijkstra, Edsger Wybe

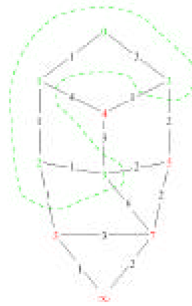
Legendary figure in computer science; was a professor at University of Texas.

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.

E.W. Dijkstra (1930-2002)

## Dijkstra's Idea

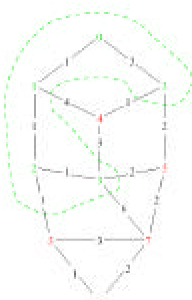


Adapt BFS to handle weighted graphs

Two kinds of vertices:

- Finished vertices
  - Shortest distance is computed
- Unknown vertices
  - Have tentative distance

## Dijkstra's Idea



At each step:

1) Pick closest unknown vertex
2) Add it to finished vertices
3) Update distances

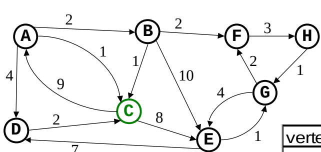
## Dijkstra Pseudocode

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 the unknown node with the lowest cost: *n*
  Mark *n* as known
  For each node *a* which is adjacent to *n*
    *a*'s cost = min(*a*'s old cost, *n*'s cost + cost of (*n*, *a*))

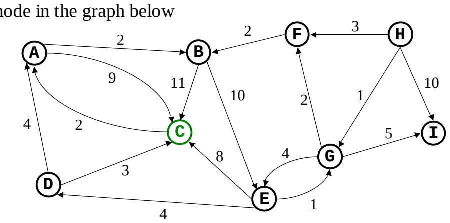
## Dijkstra's Algorithm in Action




| vertex | known | cost |
|---|---|---|
| A | | |
| B | | |
| C | | |
| D | | |
| E | | |
| F | | |
| G | | |
| H | | |

## Time to play at home…

1. Use Dijkstra's algorithm to find the shortest path from **H** to every node in the graph below




2. Under what conditions will Dijkstra's algorithm fail?

3. What data structures should you use to best implement this algorithm? What running time does that yield?

# Dijkstra 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 with the lowest cost: *n*
    Mark *n* as known
    For each node *a* which is adjacent to *n*
        *a*'s cost = min(*a*'s old cost, *n*'s cost + cost of (*n*, *a*))

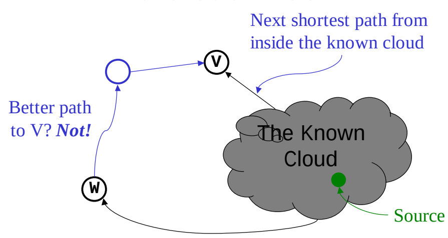
What data structures?

Running time?

# Dijkstra's Algorithm
## for Single Source, Shortest Path

- Classic algorithm for solving shortest path in weighted graphs without negative weights
- A *greedy* algorithm (irrevocably makes decisions without considering future consequences)
- Intuition:
  - 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

# The Cloud Proof




- But, if path to **V** is shortest, path to **W** must be at least as long.
- So, how can the path through **W** to **V** be shorter?

# Inside the Cloud (Proof)

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: Once 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

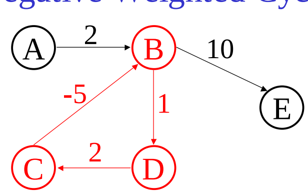
*Dijkstra's Algorithm*

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

*Breadth-first Search*

Some Similarities:

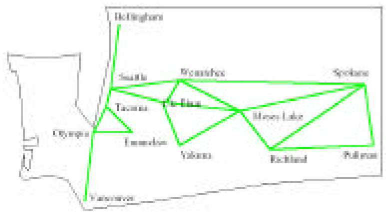
# The Trouble with Negative Weighted Cycles




**What's the shortest path from A to E?**

**Problem?**

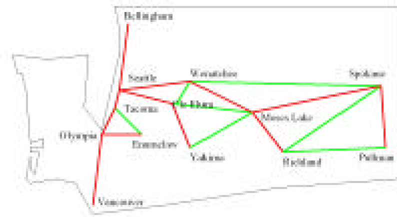
## Some Applications: Moving Around Washington



What's the fastest way from Seattle to Spokane?

Answer:

## Some Applications: Communication in Washington



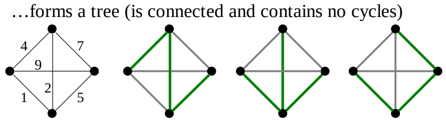
What's the cheapest inter-city network?

## Spanning Tree

*Spanning tree*: a subset of the edges from a connected graph that…

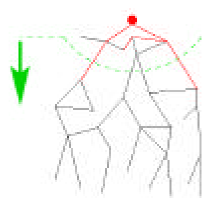
…touches all vertices in the graph (*spans* the graph)

…forms a tree (is connected and contains no cycles)




*Minimum spanning tree*: the spanning tree with the least total edge cost.

## Two Different Algorithms



Prim's Algorithm

Almost identical to Dijkstra's

Kruskals's Algorithm

Completely different!

## Prim's Algorithm for Minimum Spanning Trees

A *node-oriented* greedy algorithm (builds an MST by greedily adding nodes)

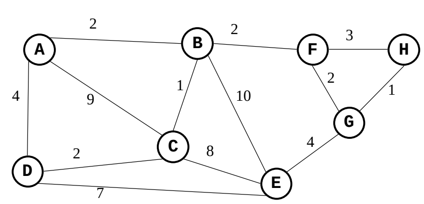
Select a node to be the "root" and mark it as known

While there are unknown nodes left in the graph

- Select the unknown node $n$ with the smallest cost from some known node $m$
- Mark $n$ as known
- Add $(m, n)$ to our MST
- Update cost of all nodes adjacent to $n$

*Runtime:* *Proof:*

## Prim's Algorithm In Action



## Kruskal's Algorithm for Minimum Spanning Trees

An *edge-oriented* greedy algorithm (builds an MST by greedily adding edges)

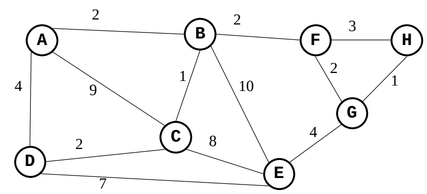
Initialize all vertices to unconnected

While there are still unmarked edges

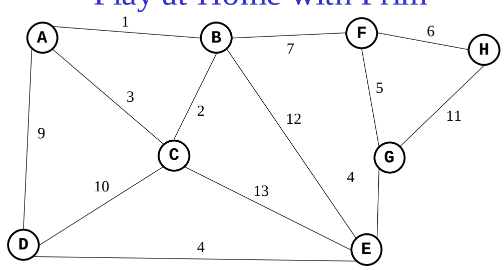
- Pick the lowest cost edge **`e = (u, v)`** and mark it
- If **`u`** and **`v`** are not already connected, add **`e`** to the minimum spanning tree and connect **`u`** and **`v`**

Sound familiar?

## Kruskal's Algorithm In Action



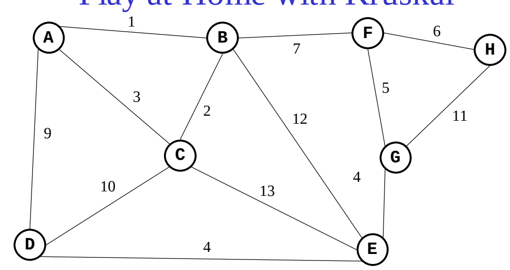

## Play at Home with Prim




1. Starting at node A, find the MST using Prim's method.

(continue on next slide)

## Play at Home with Kruskal




2. Now find the MST using Kruskal's method.
3. Under what conditions will these methods give the same result?
4. What data structures should be used for Kruskal's? Running time?

## Proof of Correctness

We already showed this finds a spanning tree:

- That was part of our definition of a good maze.

Proof by contradiction that Kruskal's finds the minimum:

- Assume another spanning tree has *lower cost* than Kruskal's
- Pick an edge $\mathbf{e_1} = \mathtt{(u, v)}$ in that tree that's *not* in Kruskal's
- Kruskal's tree connects **`u`**'s and **`v`**'s sets with another edge $\mathbf{e_2}$
- But, $\mathbf{e_2}$ must have at most the same cost as $\mathbf{e_1}$!
- So, swap $\mathbf{e_2}$ for $\mathbf{e_1}$ (at worst keeping the cost the same)
- Repeat until the tree is identical to Kruskal's: **contradiction**!

QED: Kruskal's algorithm finds a minimum spanning tree.

## Kruskal's Implementation

Initialize all vertices to unconnected

While there are still unmarked edges

- Pick the lowest cost edge **`e = (u, v)`** and mark it
- If **`u`** and **`v`** are not already connected, add **`e`** to the minimum spanning tree and connect **`u`** and **`v`**

What data structures?

Running time?