CSE 326: Data Structures
Lecture #18
Exploring Graphs

Bart Niswonger

Summer Quarter 2001

Today's Outline

- Stuff Bart didn't finish Friday
- Graph Algorithms
  - Shortest Path
    - Djikstra
  - Minimum Spanning Tree
    - Kruskal
    - Prim

Single Source, Shortest Path

Given a graph **G = (V, E)** and a vertex **s** $\in$ **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
- multiple weight types to optimize

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

Dijkstra's Pseudocode

(actually, our pseudocode for Dijkstra's algorithm)

Mark every node as unknown

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 *n* with the lowest cost

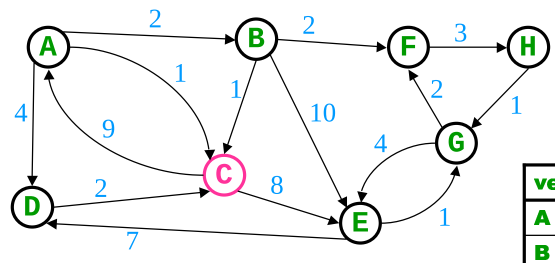
    Mark *n* as known

    For each node *a* which is adjacent to *n*

        a's cost = min(*a*'s old cost,

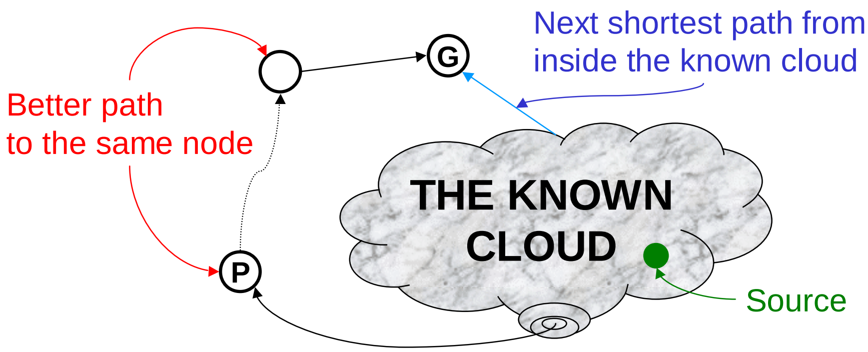
                         n's cost + cost of (*n*, *a*))

Dijkstra's Algorithm in Action



vertex	known	cost
A		
B		
C		
D		
E		
F		
G		
H		

The Cloud Proof



But, if the path to **G** is the next shortest path, the path to **P** must be at least as long.

So, how can the path through **P** to **G** be shorter?

Inside the Cloud (Proof)

Everything inside the cloud has the correct shortest path

Proof is by induction on the # of nodes in the cloud:

- initial cloud is just the source with shortest path 0
- inductive step: once we prove the shortest path to G is correct, we add it to the cloud

Negative weights blow this proof away!

Data Structures (for Dijkstra's Algorithm)

|V| times:

Select the unknown node with the lowest cost

→ findMin/deleteMin

|E| times:

a's cost = min(*a*'s old cost, …)

→ decreaseKey

→ find by name

runtime:

Revenge of Dijkstra Pseudocode

```
Initialize the cost of each node to ∞
s.cost = 0;
heap.insert(s);
while (! heap.empty())
    n = heap.deleteMin()
    for (each node a which is adjacent to n)
        if (n.cost + edge[n,a].cost < a.cost) then
            a.cost = n.cost + edge[n,a].cost
            a.path = n;
            if (heap.contains(a)) then heap.decreaseKey(a)
            else heap.insert(a)
```

Single Source & Goal

Suppose we only care about shortest path from source s to a particular vertex g

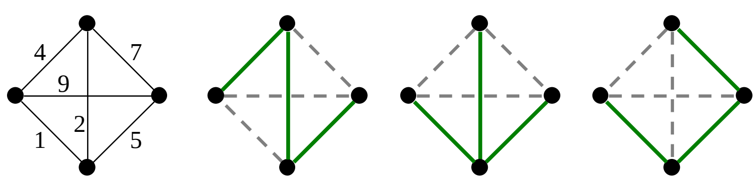
- Run Dijkstra to completion
- Stop early? When?
  - When g is added to the priority queue
  - When g is removed from the priority queue
  - When the priority queue is empty

Spanning Trees

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 (MST): the spanning tree with the least total edge cost.

Applications of MSTs

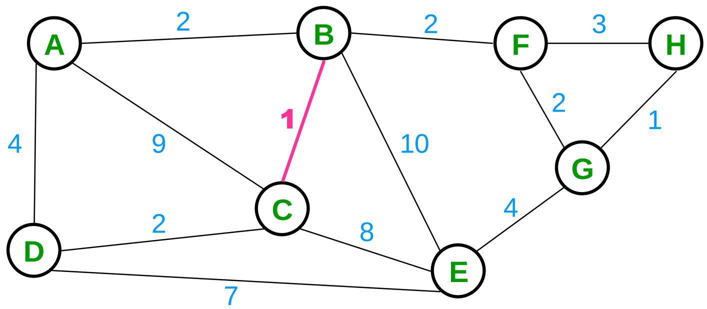
- Communication networks
- VLSI design
- Transportation systems
- Good approximation to some NP-hard problems

Kruskal's Algorithm for MSTs

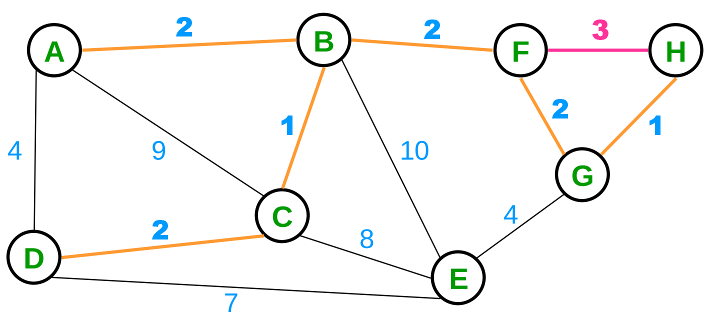
A greedy algorithm:

Initialize all vertices to unconnected
While there are still unmarked edges
    Pick a 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**

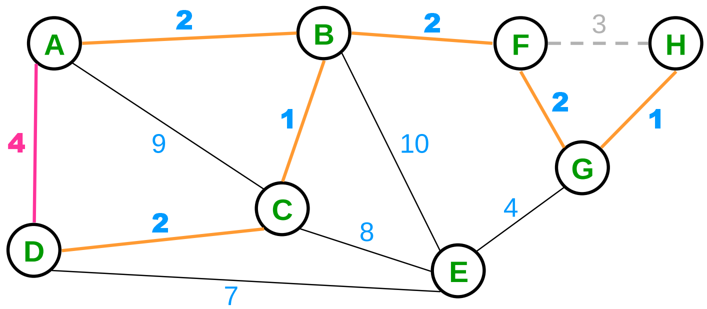
Kruskal's Algorithm in Action (1/5)



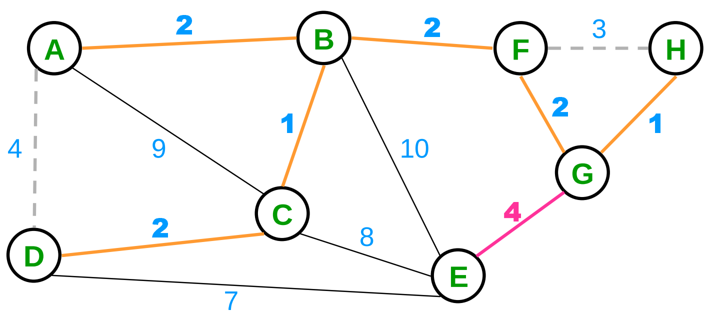
Kruskal's Algorithm in Action (2/5)



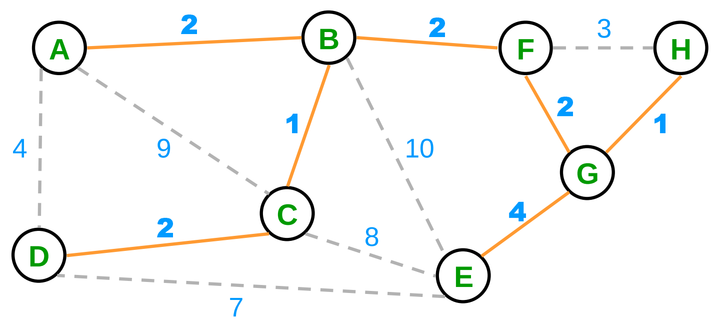
Kruskal's Algorithm in Action (3/5)



Kruskal's Algorithm in Action (4/5)



Kruskal's Algorithm Completed (5/5)



Why Greediness Works

The algorithm produces a spanning tree. *Why?*

Proof *by contradiction*: Kruskal's finds the minimum:

- Assume another spanning tree has *lower cost* than Kruskal's
- Pick an edge **e_1 = (u,v)** in that tree that's *not* in Kruskal's
- Kruskal's connects **u**'s and **v**'s sets with another edge **e_2**
- But **e_2** *must* have at most the same cost as **e_1**!
- So, swap **e_2** for **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 MST

Data Structures (for Kruskal's Algorithm)

Once:
Initialize heap of edges… → buildHeap

|E| times:
Pick the lowest cost edge… → findMin/deleteMin

|E| times:
If **u** and **v** are not already connected…
…connect **u** and **v**. → union

|E| + |E| log |E| + |E| ack(|E|,|V|)

runtime:

Prim's Algorithm

- Can also find Minimum Spanning Trees using a variation of Dijkstra's algorithm:

Pick a initial node

Until graph is connected:

  Choose edge (u,v) which is of minimum cost among edges where u is in tree but v is not

  Add (u,v) to the tree

- Same "greedy" proof, same asymptotic complexity

Does Greedy Always Work?

- Consider the following problem:
  - Given a graph G = (V,E) and a designed subset of vertices S, find a minimum cost tree that includes all of S
- Exactly the same as a minimum spanning tree, except that it doesn't have to include ALL the vertices – only the specified subset of vertices.
  - *Does Kruskal or Prim work?*

Nope!

- Greedy can fail to be optimal
  - because different solutions may contain different "non-designed" vertices, proof that you can covert one to the other doesn't go through
- This Minimum Steiner Tree problem has *no* known solution of O(n^k) for any fixed *k*
  - This is a *NP-complete* problem
  - Finding a spanning tree and then pruning it a pretty good approximation