Today’s Topics:
- Dijkstra’s Shortest Path Algorithm
- Depth First Search
- Spanning Trees
- Minimum Spanning Trees
- Prim’s Algorithm

Covered in Chapter 9 in the textbook

Single Source, Shortest Path Problem
- Given a graph $G = (V, E)$ and a “source” vertex $s$ in $V$, find the minimum cost paths from $s$ to every vertex in $V$

Dijkstra’s Shortest Path Algorithm
1. Initialize the cost of each node to $\infty$
2. Initialize the cost of the source to 0
3. While there are unknown nodes left in the graph
   1. Select the unknown node $N$ with the lowest cost (greedy choice)
   2. Mark $N$ as known
   3. For each node $A$ adjacent to $N$
      If $(N$’s cost + cost of $(N, A)) < A$’s cost
      $A$’s cost = $N$’s cost + cost of $(N, A)$
      $\text{Prev}[A] = N$ //store preceding node

Dijkstra’s Algorithm (greed in action)

Initial

Final
Analysis of Dijkstra’s Algorithm

Yes! Use a priority queue to store vertices with key = cost

- $|V|$ times:
  Select the unknown node $N$ with the lowest cost

- $|E|$ times:
  - deleteMin
  - $A$’s cost = $N$’s cost + cost of $(N,A)$

Total run time = ?

R. Rao, CSE 373

6

Does Dijkstra’s Algorithm Always Work?

- Dijkstra’s algorithm is an example of 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
- In Dijkstra’s case – choose the least cost node, but what if there is another path through other vertices that is cheaper?
- Can prove: Never happens if all edge weights are positive

Analysis of Dijkstra’s Algorithm

- Main loop:
  While there are unknown nodes left in the graph $\leftarrow |V|$ times
  1. Select the unknown node $N$ with the lowest cost $\leftarrow O(|V|)$
  2. Mark $N$ as known
  3. For each node $A$ adjacent to $N$ $\leftarrow O(|E|)$ total
     - If ($N$’s cost + cost of $(N,A)$) < $A$’s cost
     - $A$’s cost = $N$’s cost + cost of $(N,A)$

Total time = $|V| \cdot O(|V|) + O(|E|) = O(|V|^2 + |E|)$

Dense graph: $|E| = \Theta(|V|^2)$ $\Rightarrow$ Total time = $O(|V|^2) = O(|E|)$ $\sqrt{\text{ }}$

Sparse graph: $|E| = O(|V|)$ $\Rightarrow$ Total time = $O(|V|^2) = O(|E|)$ $\sqrt{\text{ }}$

Quadratic! Can we do better?

R. Rao, CSE 373

5
The “Cloudy” Proof of Dijkstra’s Correctness

If the path to \( G \) is the next shortest path, the path to \( P \) must be at least as long. Therefore, any path through \( P \) to \( G \) cannot 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:
- Base case: Initial cloud is just the source with shortest path 0
- Inductive hypothesis: cloud of k-1 nodes all have shortest paths
- Inductive step: choose the least cost node \( G \) has to be the shortest path to \( G \) (previous slide). Add \( k^{th} \) node \( G \) to the cloud

But waitaminute!! What about negative weights??

Negative Weights: Dijkstra’s Achilles Heel

Dijkstra: \( C \rightarrow D \) (cost = -5)
Least cost path: \( C \rightarrow E \rightarrow D \) (cost = -8)

Negative cycles: What’s the shortest path from \( A \) to \( E \)? (or to \( B \), \( C \), or \( D \), for that matter)
Depth First Search (DFS)

- We used Breadth First Search for finding shortest paths in an unweighted graph
  - Use a queue to explore neighbors of source vertex, neighbors of each neighbor, and so on: 1 edge away, two edges away, etc.
- Its counterpart: Depth First Search
  - A second way to explore all nodes in a graph
- DFS searches down one path as deep as possible
  - When no new nodes available, it backtracks
  - When backtracking, we explore side-paths that weren’t taken
- DFS allows an easy recursive implementation
  - So, DFS uses a stack while BFS uses a queue

DFS Pseudocode

- Pseudocode for DFS:
  
  ```
  DFS(v)
  If v is unvisited
    mark v as visited
    print v (or process v)
    for each edge (v,w)
      DFS(w)
  
  Works for directed or undirected graphs
  
  Running time = O(|V| + |E|)
  ```

What about DFS on this graph?

- What happens when you do DFS(“142”)?

Go as deep as possible,
Then backtrack…

We get a “spanning” tree…
DFS and BFS may give different trees...

Spanning Tree Definition

- **Spanning tree**: a subset of 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

Minimum Spanning Tree (MST)

We are given a weighted, undirected graph $G = (V, E)$, with weight function $w: E \rightarrow R$ mapping edges to real valued weights

Problem: Find the minimum cost spanning tree

Why minimum spanning trees?

- Lots of applications
- Minimize length of gas pipelines between cities
- Find cheapest way to wire a house (with minimum cable)
- Find a way to connect various routers on a network that minimizes total delay
- Etc…
Prim’s Algorithm for Finding the MST

1. Starting from an empty tree, \( T \), pick a vertex, \( v_0 \), at random and initialize: \( V' = \{v_0\} \) and \( E' = \{\} \)

2. Choose a vertex \( v \) not in \( V' \) such that edge weight from \( v \) to a vertex in \( V' \) is minimal (greedy again!)

3. Add \( v \) to \( V' \) and the edge to \( E' \) if no cycle is created

4. Repeat until all vertices have been added

Prim’s Algorithm for Finding the MST

1. Starting from an empty tree, \( T \), pick a vertex, \( v_0 \), at random and initialize: \( V' = \{v_0\} \) and \( E' = \{\} \)

2. Choose a vertex \( v \) not in \( V' \) such that edge weight from \( v \) to a vertex in \( V' \) is minimal (greedy again!)

3. Add \( v \) to \( V' \) and the edge to \( E' \) if no cycle is created

4. Repeat until all vertices have been added
Prim’s Algorithm for Finding the MST

1. Starting from an empty tree, $T$, pick a vertex, $v_0$, at random and initialize: $V' = \{v_0\}$ and $E' = \{\}$
2. Choose a vertex $v$ not in $V'$ such that edge weight from $v$ to a vertex in $V'$ is minimal (greedy again!)
3. Add $v$ to $V'$ and the edge to $E'$ if no cycle is created
4. Repeat until all vertices have been added

Prim’s Algorithm for Finding the MST

1. Starting from an empty tree, $T$, pick a vertex, $v_0$, at random and initialize: $V' = \{v_0\}$ and $E' = \{\}$
2. Choose a vertex $v$ not in $V'$ such that edge weight from $v$ to a vertex in $V'$ is minimal (greedy again!)
3. Add $v$ to $V'$ and the edge to $E'$ if no cycle is created
4. Repeat until all vertices have been added

Done!
Total cost = $1 + 3 + 4 + 1 + 1 = 10$
Next Class:
Analysis of Prim’s Algorithm
Kruskal takes a bow – faster MST

To Do:
Programming Assignment #2
(Don’t wait until the last few days!!)
Continue chapter 9