CSE 373: Graphs
(Shortest Paths, Minimum Spanning Trees)

Chapter 9

Weighted Shortest Paths

- Breadth-first search is no longer sufficient:

- New strategy?
Dijkstra’s Algorithm

- Classic algorithm for solving shortest path problems for weighted graphs
- A greedy algorithm (makes decisions without thinking about the future consequences)
- Intuition:
  - shortest path from source vertex to itself is 0
  - cost of going to its adjacent nodes equals edge weights
  - cheapest edge weight is shortest path to that node
  - continue recursively picking cheapest reachable node

Implementing Dijkstra’s Algorithm

More precisely:
- keep track of the cost of the shortest path found so far from s to each vertex...
  - initialize this cost to $\infty$ for all vertices
  - except s for which it is initialized to 0
- take the vertex with the shortest path found so far, $v$, and mark it as “done”
- for each node $w$ adjacent to $v$, consider whether moving to $w$ after taking the short path from $s$ to $v$ would be better than the best seen so far
- repeat until all vertices are “done”
Dijkstra’s Algorithm: Example

Minimum Spanning Trees

spanning tree: a subset of edges from a connected graph that...
...touch all vertices (span the graph)
...form a tree (are connected and form no cycles)

minimum spanning tree: the spanning tree with the smallest total cost
Prim’s Algorithm

Prim’s Algorithm: (another greedy algorithm)
- a way of finding minimum spanning trees:
  - start with an arbitrary vertex
  - pick the smallest edge adjacent to this vertex
  - continue picking the smallest edge that connects a new vertex to a vertex that’s already been linked in

Prim’s Algorithm Implementation

More precisely:
- for each vertex, keep track of the cheapest edge that could attach it to the growing tree
  - initialize all nodes to $\infty$
- pick an arbitrary vertex as the initial tree
  - mark its cost as 0
- update its adjacent vertices’ cheapest edge cost
- pick the cheapest edge that attaches a new vertex
- see if any of its edges improve the cheapest edges of its adjacent vertices
**Prim’s Algorithm Example**

![Graph](image)

**Kruskal’s Algorithm**

**Kruskal’s Algorithm:**
- Another way to find minimum spanning trees
- Another greedy algorithm:
  - continually add the cheapest edge that would not cause a cycle to form

![Graph](image)
Implementing Kruskal’s Algorithm

- How to pick shortest edges?

- How to ensure an edge won’t cause a cycle?
  - use a union/find algorithm (described in Chapter 8, which we skipped...)

Graph Summary

- More theoretical than much of what we’ve studied
- Yet, plenty of practical applications:
  - cheapest flights from one place to another (shortest path problem)
  - length of wire required to connect several terminals (minimum spanning tree problem)
  - fastest communication path between two computers (shortest path problem)