Graph Algorithms: Shortest Paths

CS211
Fall 2000

Sorting in Linear Time

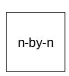
There are several sorting methods that take linear time

- Counting Sort
  - sorts integers from a small range: [0..k] where k = O(n)
- Radix Sort
  - the method used by the old card-sorters
  - sorting time O(dn) where d is the number of "digits"
- How do these methods get around the Ω(n log n) lower bound?
  - They don't use comparisons
- What sorting method works best?
  - QuickSort is best general-purpose sort
  - Counting Sort or Radix Sort can be best for *some* kinds of data

2

Aside: An Open Question on Sorting

How long does it take to a sort an n-by-n table of numbers?



- $O(n^2 \log n)$ because there are n^2 numbers in the table

What if it's an addition table?

+	1	3	5	8
2	3	5	7	10
10	11	13	15	18
12	13	15	18	20
14	15	17	19	22

- Shouldn't it be easier to sort than an arbitrary set of n^2 numbers?

3

Recall Digraphs

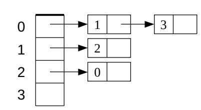
- Adjacency Matrix
  - Space $O(v^2)$

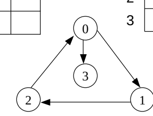
g[u][v] is true iff there is an edge from u to v

	0	1	2	3
0		T		T
1			T	
2	T			
3				

- Adjacency List
  - Space O(e+v)

The list for u contains v iff there is an edge from u to v





4

Recall Weighted Digraphs

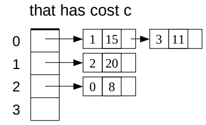
- Adjacency Matrix

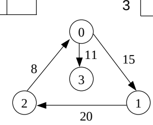
g[u][v] is c iff there is an edge of cost c from u to v

	0	1	2	3
0		15		11
1			20	
2	8			
3				

- Adjacency List

The list for u contains v,c iff there is an edge from u to v that has cost c





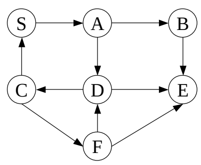
5

Goal: Find Shortest Path in a Graph

- Finding the shortest (min-cost) path in a graph is a problem that occurs often
  - Find the least-cost route between Ithaca and Detroit
  - Result depend on our notion of cost
    - least mileage
    - least time
    - cheapest
    - least boring
  - All of these "costs" can be represented as edge costs on a graph
- How do we find a shortest path?

6

Shortest Paths for Unweighted Graphs




```
bfsDistance(s):
    // s is the start vertex
    // dist[v] is length of s-to-v path
    // Initially dist[v] = ∞ for all v
    dist[s] = 0;
    Q.insert(s);

    while (Q nonempty) {
        v = Q.get();
        for (each w adjacent to v) {
            if (dist[w] == ∞) {
                dist[w] = dist[v]+1;
                Q.insert(w);
            }
        }
    }
```

7

Analysis for bfsDistance

- How many times can a vertex be placed in the queue?
- How much time for the for-loop?
  - Depends on representation
    - Adjacency Matrix: O(v)
    - Adjacency List: $O(e_v)$
- Time:
  - $O(v^2)$ for adj matrix
  - O(e+v) for adj list

```
bfsDistance(s):
    // s is the start vertex
    // dist[v] is length of s-to-v path
    // Initially dist[v] = ∞ for all v
    dist[s] = 0;
    Q.insert(s);

    while (Q nonempty) {
        v = Q.get();
        for (each w adjacent to v) {
            if (dist[w] == ∞) {
                dist[w] = dist[v]+1;
                Q.insert(w);
            }
        }
    }
```

8

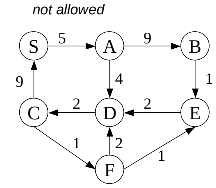
If There are Edge Costs?

- Idea #1
  - Add false nodes so that all edge costs are 1
  - But what if edge costs are large?
  - What if the costs aren't integers?
- Idea #2
  - Nothing "interesting" happens at the false nodes
  - Can't we just jump ahead to the next "real" node
  - Rule: always do the closest (real) node first
  - Use the array dist[ ] to
    - Report answers
    - Keep track of what to do next

9

Dijkstra's Algorithm

- Intuition
  - Edges are threads; vertices are beads
  - Pick up at s; mark each node as it leave the table
- Note: Negative edge-costs are *not allowed*



- s is the start vertex
- c(i,j) is the cost from i to j
- Initially, vertices are unmarked
- dist[v] is length of s-to-v path
- Initially, dist[v] = ∞, for all v

```
dijsktra(s):
    dist[s] = 0;
    while (some vertices are unmarked) {
        v = unmarked vertex with
            smallest dist;
        Mark v;
        for (each w adj to v) {
            dist[w] = min
                [ dist[w], dist[v] + c(v,w) ];
        }
    }
```

10

Dijkstra's Algorithm using Adj Matrix

- While-loop is done v times
- Within the loop
  - Choosing v takes O(v) time
  - For-loop takes O(v) time
- Total time = $O(v^2)$

- s is the start vertex
- c(i,j) is the cost from i to j
- Initially, vertices are unmarked
- dist[v] is length of s-to-v path
- Initially, dist[v] = ∞, for all v

```
dijsktra(s):
    dist[s] = 0;
    while (some vertices are unmarked) {
        v = unmarked vertex with
            smallest dist;
        Mark v;
        for (each w adj to v) {
            dist[w] = min
                [ dist[w], dist[v] + c(v,w) ];
        }
    }
```

11

Dijkstra's Algorithm using Adj List

- Looks like we need a PQ
  - Problem: priorities are updated as algorithm runs
  - Can insert pair (v,dist[v]) in PQ whenever dist[v] is updated
  - At most e things in PQ
- Time O(v + e log e)
- Using a more complicated PQ (e.g., Pairing Heap), time can be brought down to O(e + v log v)

- s is the start vertex
- c(i,j) is the cost from i to j
- Initially, vertices are unmarked
- dist[v] is length of s-to-v path
- Initially, dist[v] = ∞, for all v

```
dijsktra(s):
    dist[s] = 0;
    while (some vertices are unmarked) {
        v = unmarked vertex with
            smallest dist;
        Mark v;
        for (each w adj to v) {
            dist[w] = min
                [ dist[w], dist[v] + c(v,w) ];
        }
    }
```

12