Shortest paths

Unit weights (min # edges):
  
  **Breadth-first search**
  \[ O(IVI + IEI) \]

Non-negative weights
  
  **Dijkstra**
  \[ O(IEI \log IVI) \] - binary heap
  \[ O(NIVI \log IVI + IEI) \] - Fibonacci heap
Some negative weight edges:

Dijkstra will overestimate distance to v.

We could put vertices back each time their dist value decreases.

Correct algorithm. But very slow if many
void weightedNegative(Vertex s) {
    Queue<Vertex> q = new Queue<Vertex>();
    for each Vertex v {
        v.dist = INFINITY;
    }
    s.dist = 0;
    q.enqueue(s);
    while (!q.isEmpty()) {
        Vertex v = q.dequeue();
        for each Vertex w adjacent to v {
            if (v.dist + cvw < w.dist) { // we found a shorter path
                w.dist = v.dist + cvw;
                w.path = v;
                if (w is not already in q) {
                    q.enqueue(w);
                }
            }
        }
    }
}

Imagine algorithm running in phases.

Phase 0: process s
Phase i+1 begins when we dequeue first vertex added during phase i.
Lemma: By the time phase \( i \) begins, each \( v \).dist is at most length of shortest path using at most \( i \) edges.

\[ \text{dist}^i(v) = \min \text{ path length using } \leq i \text{ edges} \]

By induction, when phase \( i-1 \) began

\[ u \text{.dist}^i \leq \text{dist}^{i-1}(u) \]
We would have pulled \( u \) out of queue with \( u, \text{dist} = \text{dist}(u) + v \) processed edges, so
\[
v, \text{dist} = \text{dist}(u).
\]

\( \Rightarrow \) If shortest path to \( v \) uses \( k \) edges, we have the correct \( v, \text{dist} \) by beginning of phase \( k \).

If no neg. cycle, shortest paths don't repeat vertices, so paths use \( \leq |V|-1 \) edges.
⇒ no vertex enters queue
⇒ $V_1 - 1$ times
⇒ each edge processed
⇒ $V_1 - 1$ times

⇒ $O(V_1 \cdot \text{IEI})$ running time

Worst-case

Could be faster!

If a vertex enters the queue
a $1/V_1$th time ⇒ negative weight cycle
That was an algorithm by Moore.

Bellman-Ford: known.

Check all edges in a for loop.

Repeat until nothing improves.

Takes N-1 V1-1 rounds if no negative cycles.

O(V1 · IE1)
void bellmanFord(Vertex s) {
    for each Vertex v {
        v.dist = INFINITY;
    }

    s.dist = 0;

    repeat |V| - 1 times {
        for each edge v to w {
            if (v.dist + cvw < w.dist) {  // we found a shorter path
                w.dist = v.dist + cvw;
                w.path = v;
            }
        }
    }
}
Suppose $G$ is a DAG, no (negative weight) cycles.

```c
void weightedDAG(Vertex s) {
    for each Vertex v {
        v.dist = INFINITY;
    }
    s.dist = 0;

    for each vertex v in topological order {
        for each Vertex w adjacent to v {
            if (v.dist + cvw < w.dist) {
                // we found a shorter path
                w.dist = v.dist + cvw;
                w.path = v;
            }
        }
    }
}
```

$O(1VI + 1EI)$ time
All-pairs shortest paths

Want distance from s to t for all pairs of vertices s, t.

Output as a 2D array

\[ \text{dist}[i][j] \text{ where} \]

\[ \text{dist}[u][v] : \text{distance from} \quad \overrightarrow{u} \quad \text{to} \quad \overrightarrow{v} \]
Could run SSSP in $O(V) \text{ time}$ for a single source shortest paths.

If unweighted or a DAG,

$$O(V \cdot (V+E)) = O(V E) = O(V^3)$$

Non-negative weights

$$O(V \cdot (E \log V)) = O(V^3 \log V)$$

- or -

$$O(V \cdot (V \log V + E)) = O(V^2 \log V) + V E$$

$$= O(V^3)$$
Neg. weights

\[ O(|V|^2 |E|) = O(|V|^2 |E|) = O(|V|^4) \]

we can do better!

Let \( n = |V| \).

Number the vertices arbitrarily 0 through \( n-1 \).

Consider vertices \( i \) and \( j \).

Internal vertices of a path are the ones not at end points.
$D_{i,j,k}^\text{length of shortest path with internal vertices from } \{0, \ldots, k\}$.

$D_{i,j,k}^k - 1 = w(\bar{i} \to_j)$

$\uparrow$ weight of $\bar{i} \to_j$

or $\infty$ if $\bar{i} \to_j \notin E$

$D_\omega^j, n - 1 = \text{dist}(\bar{i}, \bar{j})$
Path for $D_{\omega,j,k}$ has two cases:

- $\omega 
  \begin{cases} 
    \in k 
    \Rightarrow j 
  \end{cases}$
- $\omega \Rightarrow j \Rightarrow 0

$D_{\omega,j,k} = \min \{D_{\omega,j,k-1}, D_{\omega,k,k-1} + D_{k,j,k-1}\}$

$O(1)^3$ subproblems

Compute in increasing order of $k$,

$O(1)$ per subproblem,

$O(1)^3$ time with DP.
Code on Wed. fill in each distriiJiC J using

D

\[ D \]