Goals:
- Design a dynamic programming algorithm for shortest path

Array dimension?

Group work

1. $P_{v,i} = \begin{cases} P_{u,i-1} + (u,v) & \text{if shortest path with } i \text{ edges goes through } u \text{ immediately prior to } v \\ P_{w,i-1} + (w,v) & " \quad w \quad " \quad " \\ \vdots & \vdots \\ P_{v,i-1}, & \text{If shortest } s \to v \text{ path with at most } i \text{ edges uses fewer than } i \text{ edges.} \end{cases}$

$P_{s,0} = \emptyset$ $\quad P_{v,0} = \text{None}$ $\quad v \in V$, and $v \neq s$

2. Turn into objective function recurrence relation

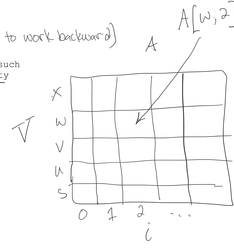
$L(P_{v,i})$ = length of shortest path from $s \to v$, with at most i edges

$$L(P_{v,i}) = \min \left\{ \min_{\substack{u \in V \\ u \neq v \\ (u,v) \in E}} \{ L(P_{u,i-1}) + w(u,v) \}, L(P_{v,i-1}) \right\}$$

$L(P_{s,0}) = 0$ $\quad L(P_{v,0}) = \infty$ $\quad v \neq s$

3. Write Pseudocode to Fill in A (don't need to work backward)

$A[w,2] = L(P_{w,2})$

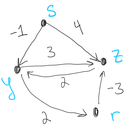


Input: Description of an n-vertex graph via an nxn array w, such that w[u,v] contains weight of edge (u,v). (Weight is infinity if no edge and 0 for w[v,v].) Starting vertex s
Output: Array A containing…?

```
Initialize n×n array A containing ∞'s.
A[s,0] = 0
for i = 1 to n-1:
    for v ∈ V:
        // Find minimum among options!
        for u ∈ V:
            if A[u,i-1] + w[u,v] < A[v,i]:
                A[v,i] ← A[u,i-1] + w[u,v]
```

$$L(P_{v,i}) = \min \left\{ \min_{\substack{u \in V \\ u \neq v \\ (u,v) \in E}} \{ L(P_{u,i-1}) + w(u,v) \}, L(P_{v,i-1}) \right\}$$

Example



Runtime?

A) $O(n)$ B) $O(n^2)$ C) $O(n^3)$ D) Need more info about graph

```
Initialize n×n array A containing ∞'s.
A[s,0] = 0
for i = 1 to n-1:
    for v ∈ V:
        // Find minimum among options!
        for u ∈ V:
            if A[u,i-1] + w[u,v] < A[v,i]:
                A[v,i] ← A[u,i-1] + w[u,v]
```

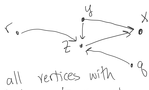
Next step: Work backwards to find optimal paths (Pset)

Bonus: Can use A to find shortest path from s to any other vertex!

Only need consider this case if edge from u to v exists.

$P_{v,i} = \begin{cases} P_{u,i-1} + (u,v) & \text{if shortest path with } i \text{ edges goes through } u \text{ immediately prior to } v \\ P_{v,i-1}, & \text{If shortest } s \to v \text{ path with at most } i \text{ edges uses fewer than } i \text{ edges.} \end{cases}$

Reverse adjacency list:



$W[x] =$ [y, w(y,x) | z, w(z,x)] ← list of all vertices with edges that go to x, and weights

$W[z] =$ [q, w(q,z) | r, w(r,z) | y, w(y,z)]

```
Initialize n×n array A containing ∞'s.
A[s,0] = 0
for i = 1 to n-1:                                   ← O(n)
    for v ∈ V:                                      ← every vertex
        // Find minimum among options!              ← go through list W[v]
        for u ∈ V s.t. (u,v) ∈ E:                   ← each edge going to v
            if A[u,i-1] + w[u,v] < A[v,i]:          } Every edge in graph once
                A[v,i] ← A[u,i-1] + w[u,v]          } O(m)  (m = # of edges)
```

Mini Amortized Analysis

In worst case: $O(m) = O(n^2)$, but if sparse graph can do better

$O(mn)$ runtime

Why is this distributed?

Each calculation is local; if node v stores $A[v,i]$, can determine $A[v,i+1]$ by asking neighbors for information. Doesn't need info from vertices that are further away.