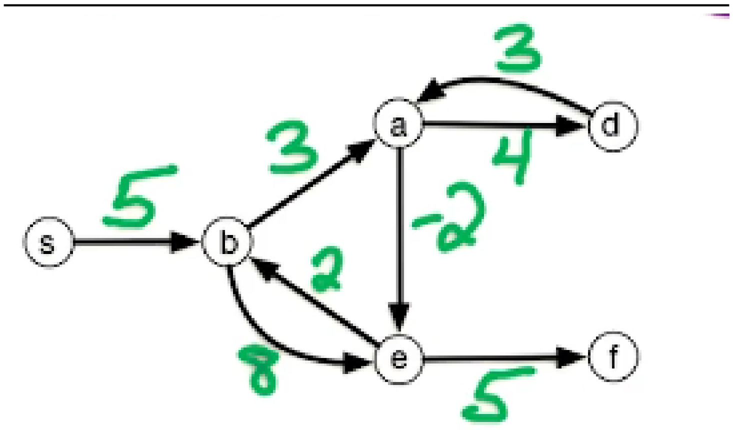
Dynamic Programming 3

Shortest Paths via DP

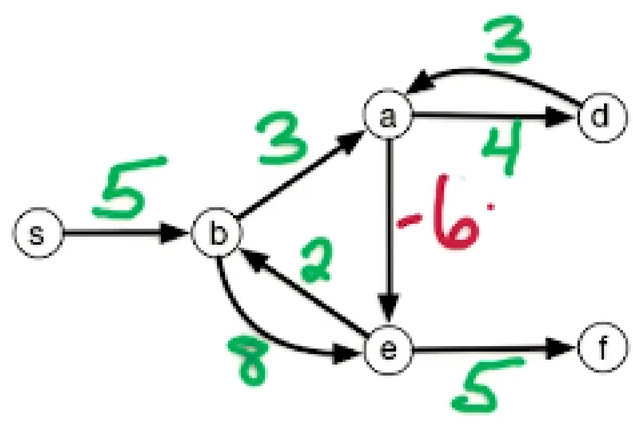
1. Directed graph G = (V, E) with edge weights w(e)
2. Can encode undirected graph as a directed graph through use of antiparallel edges (A -> D, D -> A)
    - Directed graph problem is more general than undirected
3. For z in V, dist(z) = length of shortest path from s to z
    - dist(s) = 0
    - dist(b) = 5
    - dist(a) = 8
    - dist(e) = 6
    - dist(d) = 12
    - dist(f) = 11
4. Dijkstra's algorithm:
    - Given G and s in V, finds dist(z) for all z in V
    - Similar to BFS, explore graph in a layered approach
    - O((n+m) * log(n)) time
        - n + m for vertices vs edges
        - Additional log(n) because BFS uses a priority queue where operations take log(n) time
    - Dijkstra's algorithm requires positive edge weights



Directed Graph

Negative Weights Cycles

1. Negative weight cycles can cause dist to be -inf
    - Path: Can only visit a vertex once
    - Walk: Can visit a vertex multiple times
    - a -> e -> b is a negative weight cycle
    - Negative weight cycles make the shortest path problem not well-defined
2. Given G with w(e) and s in V
    - Find a negative weight cycle
    - Else: find dist(z) for all z in V



Directed Graph with Negative Cycle

Single Source: Subproblem

1. Given G with edge weights and s in V
   - Assume no negative weight cycles
   - Shortest path from s to z visits every vertex <= 1
   - |P| <= n - 1 edges
2. DP idea: Use i = 0 -> n - 1 edges on the paths
   - For 0 <= i <= n - 1 and z in V
   - Let D(i,z) = length of shortest path from s to z using <= i edges

Single Source: Recurrence

1. For 0 <= i <= n - 1 and z in V
   - Let D(i,z) = length of shortest path from s to z using <= i edges
2. Base case: D(0,s) = 0 and for all z != s, D(0,z) = inf
3. For i >= 1: look at shortest path s -> z using i edges
   - D(i,z) = min{D(i-1,y) + w(y,z)} for yz in E
   - Length of shortest path from S to Z that goes through Y
     - D(i-1,y) + w(y,z)
4. For i >= 1: look at shortest path s -> z using <= i edges
   - D(i,z) = min(min{D(i-1,y) + w(y,z)} for yz in E, D(i-1,z))

Single Source: Summary

1. For 0 <= i <= n - 1 and z in V
   - Let D(i,z) = length of shortest path from s to z using <= i edges
2. Base case: D(0,s) = 0 and for all z != s, D(0,z) = inf
3. For i >= 1: D(i,z) = min(min{D(i-1,y) + w(y,z)} for yz in E, D(i-1,z))

Single Source: Pseudocode

```
# G = graph, S = source w = weights
def BellmanFord(G, S, W):
    for all z in V:
        D(0,z) = inf
    D(0,s) = 0
    for i = 1 -> n-1:
        for all z in V:
```

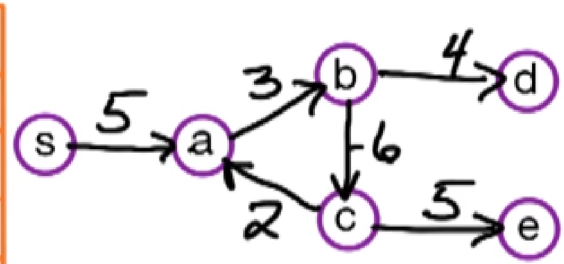
```
        D(i,z) = D(i-1,z)
        for all yz in E:
            if (Di,z) > D(i-1,y) + w(y,z):
                D(i,z) = D(i-1,y) + w(y,z)
return D(n-1,:)
```

1. Subtlety: We want to look at all of the edges from y into z
    - Typically when we use adjacency lists, it contains all edges leaving a node
    - Instead, look at the adjacency list for the reverse graph
        - Takes O(n+m) time to compute
2. Time complexity:
    - O(n) for outer loop
    - O(m) for inner loop
    - Combined, this takes O(nm) time

Finding Negative Weight Cycle 1

1. How to find a negative weight cycle?
    - Negative weight cycle will continue to update graph
    - Check if D(n,:) == D(n-1,:) for some z in V

	s	a	b	c	d	e
i=0	0	∞	∞	∞	∞	∞
i=1	0	5	∞	∞	∞	∞
2	0	5	8	∞	∞	∞
3	0	5	8	2	12	∞
4	0	4	8	2	12	7
5	0	4	7	2	12	7
6	0	4	7	1	11	7



Bellman-Ford on a Graph with Negative Cycles

All Pairs Shortest Path

1. Bellman-Ford looks at a single source
    - Now, look at all pairs
2. Given G = (V,E) with edge weights w(e)
    - For y,z in V, let dist(y,z) = length of shortest path y -> z
        - Goal: Find dist(y,z) for all y,z in V
3. Easy: Run Bellman-Ford for all s in V

Naive Approach

1. What is the running time of the naive all-pairs algorithm?
    - O(n^2 * m)
2. Better approach: Floyd-Warshall

- $O(n^3)$
- This is better because m can be up to n^2

All Pairs: Subproblem

1. Bellman-Ford idea: Condition on number of edges
2. New idea: let V = {1, 2, . . . , n}
   - Number the vertices
   - Condition on intermediate vertices -> Use prefix of V
3. For 0 <= i <= n and 1 <= s,t <= n (start and end vertices)
   - Let D(i,s,t) = length of shortest path s -> t using a subset of {1, . . . , i} as intermediate vertices

All Pairs: Base Case

1. For 0 <= i <= n and 1 <= s,t <= n (start and end vertices)
   - Let D(i,s,t) = length of shortest path s -> t using a subset of {1, . . . , i} as intermediate vertices
2. Base case:
   - D(0,s,t) = {w(s,t) if st in E, inf otherwise}

All Pairs: Recurrence

1. Base case:
   - D(0,s,t) = {w(s,t) if st in E, inf otherwise}
2. For i >= 1: Look at shortest path P s -> t using {1, . . . , i}

Case: i not on path

1. For i >= 1: Look at shortest path P s -> t using {1, . . . , i}
   - if i not on path: D(i,s,t) = D(i-1,s,t)

Case: i is on path

1. For i >= 1: Look at shortest path P s -> t using {1, . . . , i}
   - if i is on path:
     - We use a subset of the intermediate nodes (maybe all, maybe none)
     - Break up path into four sections:
       * s to subset
       * subset to i
       * i to subset
       * subset to t

Recurrence: i is on path

1. If i is on path:
   - D(i,s,t) = D(i-1,s,i) + D(i-1,i,t)

Recurrence: Summary

1. D(i,s,t) = min{D(i-1,s,t), D(i-1,s,i) + D(i-1,i,t)} i = 0 -> n

All Pairs: Pseudocode

```
def FloydWarshall(G,w):
    for s = 1 -> n:
        for t = 1 -> n:
```

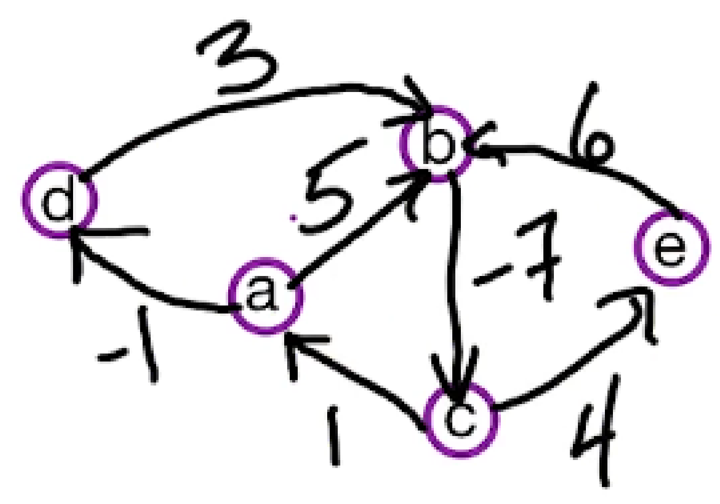
```
        if st in E:
            D(0,s,t) = w(s,t)
        else:
            D(0,s,t) = inf

for i = 1 -> n:
    for s = 1 -> n:
        for t = 1 -> n:
            D(i,s,t) = min{D(i-1,s,t), D(i-1,s,i) + D(i-1,i,t)}

return D(n,:,:)
```

All Pairs: Running Time

1. What is the worst-case running time of the Floyd-Warshall all pairs shortest path algorithm?
   - $O(n^3)$

Finding Negative Weight Cycle 2

1. As written, algorithm assums no negative weight cycles
2. How to detect a negative weight cycle?
   - If D(n,a,a) < 0, there is a path to itself with negative weight
   - Check all diagonal entries:
     - if D(n,y,y) < 0 for some y in V



Directed Graph with a Negative Weight Cycle

Comparing Algorithms

1. Bellman-Ford
   - Only finds negative weight cycles reachable from the start vertex s
2. Floyd-Warshall
   - Finds negative weight cycle anywhere in the graph

DP3 Practice Problems

1. DPV 4.21 (currency exchange)

- Dollar -> Yen -> Pound -> Dollar
  - Look for opportunities where this is greater than 0
  - Arbitrage opportunity
- Reduce this problem to a negative weight cycle algorithm