Objectives

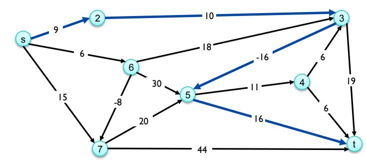
- Dynamic Programming
  - Improving Shortest Path
- Network Flow

Shortest Paths

- **Problem**: Given a directed graph G = (V, E), with edge weights c_{vw}, find shortest path from node *s* to node *t*

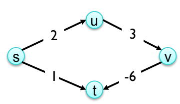
allow negative weights

- Allows modeling other phenomena



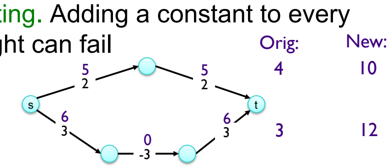
Shortest Paths: Failed Attempts

- Dijkstra. Can fail if negative edge costs

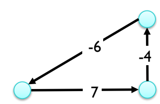


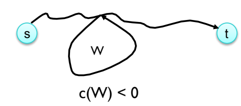
- Re-weighting. Adding a constant to every edge weight can fail

Why?



Shortest Paths: Negative Cost Cycles



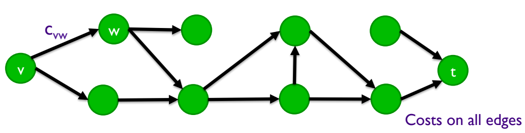


- If some path from *s* to *t* contains a negative cost cycle, there does **not** exist a shortest *s-t* path
- Otherwise, there exists one that is *simple* (i.e., does not repeat nodes)
  - Path has *at most n-1* edges, where *n* is # of nodes in graph

Towards a Recurrence

- **OPT(*i*,*v*)**: minimum cost of a ***v***-*t* path *P* using ***at most i*** edges
  - This formulation eases later discussion
- Original problem is OPT(n-1, s)

Break down into subproblems based on *i* and *v*



Shortest Paths: Dynamic Programming

- OPT(i, v) = minimum cost of a ***v***-*t* path *P* using at most ***i*** edges
  - Case 1: *P* uses at most *i-1* edges
    - OPT(i, v) = OPT(i-1, v)
  - Case 2: *P* uses exactly *i* edges
    - if (*v, w*) is first edge, then OPT uses (*v, w*), and then selects best *w-t* path using at most *i-1* edges
    - Cost: cost of chosen edge

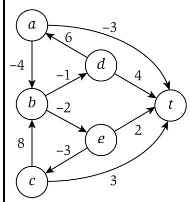
$$OPT(i,v) = \begin{cases} 0 & \text{if } i = 0 \\ \min\left\{ OPT(i-1,\ v),\ \min_{(v,w) \in E} \left\{ OPT(i-1,\ w) + c_{vw} \right\} \right\} & \text{otherwise} \end{cases}$$

Shortest Paths: Analysis

```
Shortest-Path(G, t)
  n = number of nodes in G
  foreach node v ∈ V
    M[0, v] = ∞     # infinite cost to reach all nodes
  M[0, t] = 0     # no cost to reach destination from dest

  for i = 1 to n-1     O(n)
    foreach node v ∈ V
      M[i, v] = M[i-1, v]  # at most cost of 1 less
      foreach edge (v, w) ∈ E
         M[i, v] = min(M[i, v], M[i-1, w] + c_vw )
```
O(m)

Time: $O(n^3)$, $\Theta(mn)$
Space: $\Theta(n^2)$

Example

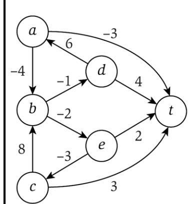


Number of edges in path

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞					
b	∞					
c	∞					
d	∞					
e	∞					

Review: trying to get to t
Looking towards improved implementation,
how to find the *solution*, not just the value of the solution

Example

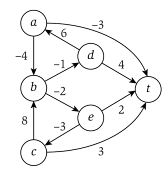


Number of edges in path

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞					
b	∞					
c	∞					
d	∞					
e	∞					

What edges do we need to look at for each node?

Example

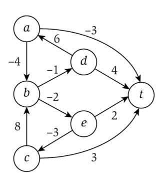


Edges
b, t
d, e
b, t
a, t
c, t

Number of edges in path

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞					
b	∞					
c	∞					
d	∞					
e	∞					

Example

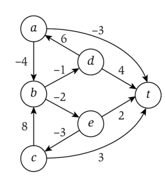


Edges
b, t
d, e
b, t
a, t
c, t

Number of edges in path

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3				
b	∞	∞				
c	∞	3				
d	∞	4				
e	∞	2				

Example



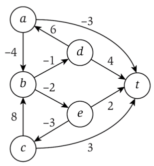
Edges
b, t
d, e
b, t
a, t
c, t

Number of edges in path

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3			
b	∞	∞	0			
c	∞	3	3			
d	∞	4	3			
e	∞	2	0			

Example

Number of edges in path

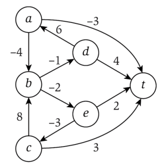


Edges
b, t
d, e
b, t
a, t
c, t

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4		
b	∞	∞	0	-2		
c	∞	3	3	3		
d	∞	4	3	3		
e	∞	2	0	0		

Example

Number of edges in path

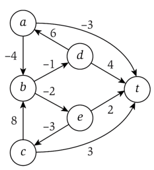


Edges
b, t
d, e
b, t
a, t
c, t

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4	-6	
b	∞	∞	0	-2	-2	
c	∞	3	3	3	3	
d	∞	4	3	3	2	
e	∞	2	0	0	0	

Example

Number of edges in path



Edges
b, t
d, e
b, t
a, t
c, t

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4	-6	-6
b	∞	∞	0	-2	-2	-2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

Shortest Paths: Implementation

```
Shortest-Path(G, t)
  n = number of nodes in G
  foreach node v ∈ V
    M[0, v] = ∞    # infinite cost to reach all nodes
  M[0, t] = 0    # no cost to reach destination from dest

  for i = 1 to n-1
    foreach node v ∈ V
      M[i, v] = M[i-1, v]  # at most cost of 1 less
      foreach edge (v, w) ∈ E
         M[i, v] = min(M[i, v], M[i-1, w] + c_vw )
```

- Shortest path length is M[n-1, s]

Discussion

- How can we find the shortest *path*?
  - What information do we need?
- Based on experience from example, what could we do to improve the algorithm's runtime and space requirements?

Shortest Paths: Practical Improvements

- To find the shortest paths, maintain a successor for each node
- Practical improvements
  - Maintain only one array M[v] = shortest *v*-t path length that we have found so far
  - No need to check edges of the form (*v*, *w*) ***unless*** M[w] changed in previous iteration
- Theorem. Throughout algorithm, M[v] is length of some *v*-*t* path.
  - After *i* rounds of updates, the value M[v] is no larger than the length of shortest *v*-*t* path using ≤ *i* edges

Bellman-Ford: Efficient Implementation

```
Push-Based-Shortest-Path(G, s, t)
   foreach node v ∈ V
      M[v] = ∞
      successor[v] = ϕ

   M[t] = 0
   for i = 1 to n-1
      foreach node w ∈ V
         if M[w] has been updated in previous iteration
            foreach node v such that (v, w) ∈ E
               if M[v] > M[w] + c_vw
                  M[v] = M[w] + c_vw
                  successor[v] = w

        if no M[w] value changed in iteration i, stop.
```

Analysis of running time, space?

Bellman-Ford: Efficient Implementation

```
Push-Based-Shortest-Path(G, s, t)
   foreach node v ∈ V
      M[v] = ∞
      successor[v] = ϕ

   M[t] = 0
   for i = 1 to n-1
      foreach node w ∈ V
         if M[w] has been updated in previous iteration
            foreach node v such that (v, w) ∈ E
               if M[v] > M[w] + c_vw
                  M[v] = M[w] + c_vw
                  successor[v] = w

        if no M[w] value changed in iteration i, stop.
```

Space: O(m + n)
Running time: O(mn) worst case but substantially faster in practice

Bellman-Ford: Efficient Implementation

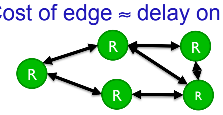
```
Push-Based-Shortest-Path(G, s, t)
   foreach node v ∈ V
      M[v] = ∞
      successor[v] = ϕ

   M[t] = 0
   for i = 1 to n-1
      foreach node w ∈ V
         if M[w] has been updated in previous iteration
            foreach node v such that (v, w) ∈ E
               if M[v] > M[w] + c_vw
                  M[v] = M[w] + c_vw
                  successor[v] = w

        if no M[w] value changed in iteration i, stop.
```

How do we get the solution if only have O(m+n) space?

DISTANCE VECTOR PROTOCOL

Application of Shortest-Path Problem

- Routers in communication network need to find most efficient path to destination
- Model of communication network
  - Nodes ≈ routers
  - Edge ≈ direct communication link
  - Cost of edge ≈ delay on link



- Possible solution: Dijkstra's algorithm
  - Why?

Distance Vector Protocol

- Model of communication network
  - Nodes ≈ routers
  - Edge ≈ direct communication link
  - Cost of edge ≈ delay on link ⟵ *Naturally non-negative*
- However, Dijkstra's algorithm requires *global* information of network
  - Create whole paths from node
- Better: use only local information

Distance Vector Protocol

- Model of communication network
  - Nodes ≈ routers
  - Edge ≈ direct communication link
  - Cost of edge ≈ delay on link ← *Naturally non-negative*
- Bellman-Ford uses only *local* knowledge of neighboring nodes
  - **Distribute** algorithm: each node *v* maintains its value M[v]
  - Updates its value after getting neighbor's values:
    - $\min_{w \in V} (c_{vw} + M[w])$

Distance Vector Protocol

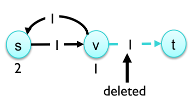
- Each router maintains vector

Node	Shortest Path Length	First Hop (direction)
…	…	…

- Algorithm: each router performs *n* computations, 1 for each potential destination node
  - Periodically gets updates from neighbors
- Synchronization issues
  - Routers don't run in lockstep
  - Order `foreach` loop executes is not important
  - Algorithm still converges even if updates are asynchronous
- "Routing by rumor"
  - Reliance on neighbors

Issues with Distance Vector Protocol

- Original algorithm developed for one central machine
  - Costs known in advance, didn't change
- Edge costs may change during algorithm (or fail completely)



"counting to infinity"

Path Vector Protocols

- Link state routing
  - Each router stores *entire path*
    - Not just the distance and the first hop
  - Based on Dijkstra's algorithm
  - Avoids "counting-to-infinity" problem and related difficulties
  - Tradeoff: requires significantly more storage
- Ex. Border Gateway Protocol (BGP), Open Shortest Path First (OSPF)

Milestone: Page 300 in book

Next Week

- Wiki - Wednesday
  - Finish reading Chapter 6: 6.4-6.8
- Problem Set 8 due Friday
  - Implementing pretty printing