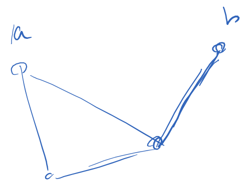
csi503-s21-lecture11

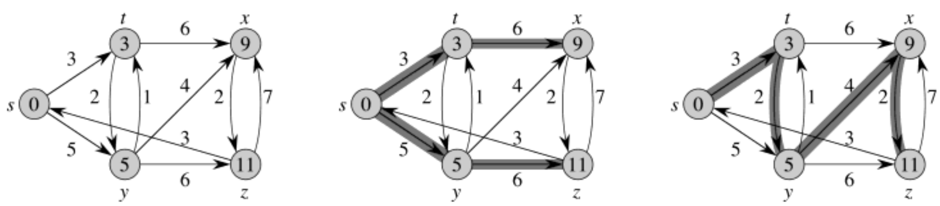
Thursday, April 8, 2021



path : $v_0 \to v_k$ $\quad p = \langle v_0, v_1, v_2, \dots, v_k \rangle$

weight of path $\quad w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$

min-weight path $\quad \delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path } u \leadsto v \\ \infty & \text{otherwise} \end{cases}$



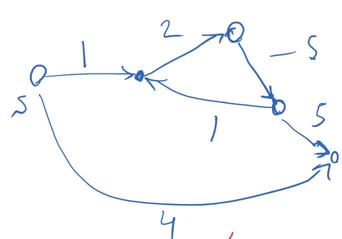
(a) (b) (c)

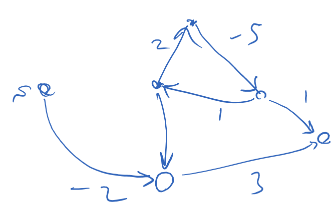
Variants of shortest path problem:

- single-source
- single-destination
- single-pair
- all-pairs

Can we have negative-weight edges?

yes as long as we don't encounter a negative-weight cycle.

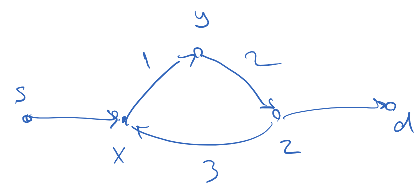




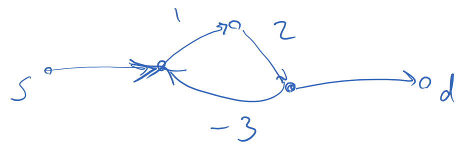
Lemma: Any subpath of a shortest path is itself a shortest path.

- shortest-paths cannot contain cycles:

- negative-weight cycle : X
- positive-weight cycle : must avoid the cycle



- zero-weight cycle : no need to contain it



$\delta(s, v)$: shortest path weight

$v.d \longrightarrow$ eventually $v.d = \delta(s, v)$

↳ initially $v.d = \infty$

↳ always $v.d \geq \delta(s, v)$

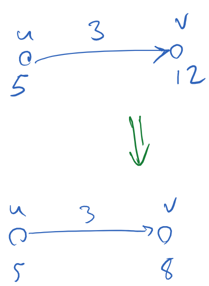
$v.\pi$: predecessor of v on shortest path $s \rightsquigarrow v$

↳ initially, $v.\pi = NIL$

```
INIT-SINGLE-SOURCE(G, s)
  for each v ∈ G.V
      v.d = ∞
      v.π = NIL
  s.d = 0
```

```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
      v.d = u.d + w(u, v)
      v.π = u
```

Relaxing (u,v) :



if $v.d > u.d + w(u,v)$:

$v.d = u.d + w(u,v)$

$v.\pi = u$

Shortest path properties:

- triangle inequality : $\delta(s,v) \leq \delta(s,u) + w(u,v)$

- upper-bound property : $v.d \geq \delta(s,v)$

- no-path property: if $\delta(s,v) = \infty$, then always $v.d = \infty$

- convergence property: if $s \leadsto u \to v$ is a shortest path

& $u.d = \delta(s,u)$

call Relax(u, v, w)

$v.d = \delta(s,v)$

- path relaxation property:
  - $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k
  - if we relax in order $(v_0, v_1), (v_1, v_2) \ldots, (v_{k-1}, v_k)$

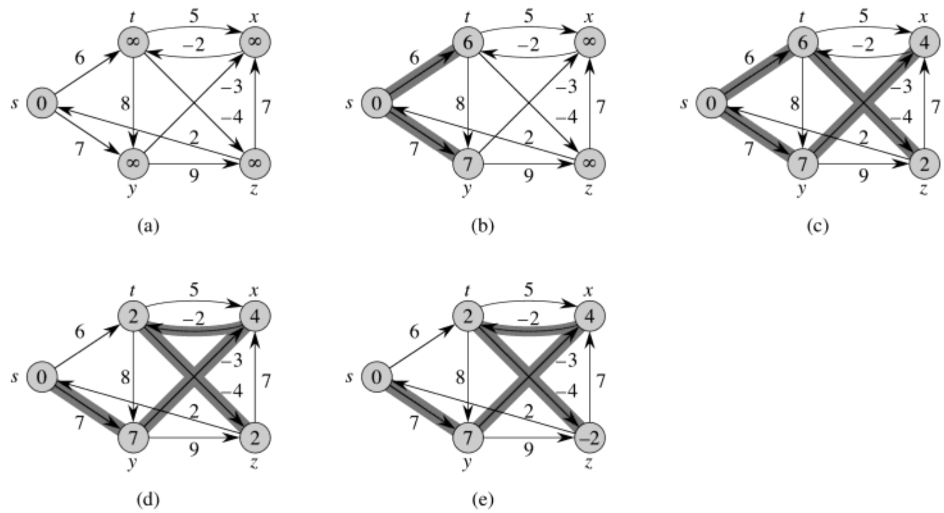
    then $v_k.d = \delta(s, v_k)$

```
BELLMAN-FORD(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| - 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

{ supports negative edges
{ return true if no negative-cycle reachable

$\Theta(VE)$

$u \to v$



Correctness of B.F Algorithm:

for reachable vertex v, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to $v = v_k$. p is acyclic $\Longrightarrow$ there will be $\leq |V|-1$ edges in p $\Rightarrow k \leq |V|-1$

Each iteration of the relaxation for loop, relaxes all edges.

- 1st iteration relaxes (v_0, v_1)
- 2nd iteration " (v_1, v_2)

  ⋮
- kth iteration " (v_{k-1}, v_k)

Based on path-relaxation property, $v.d = v_k.d = \delta(s, v)$

Also, check how to prove true/false return value

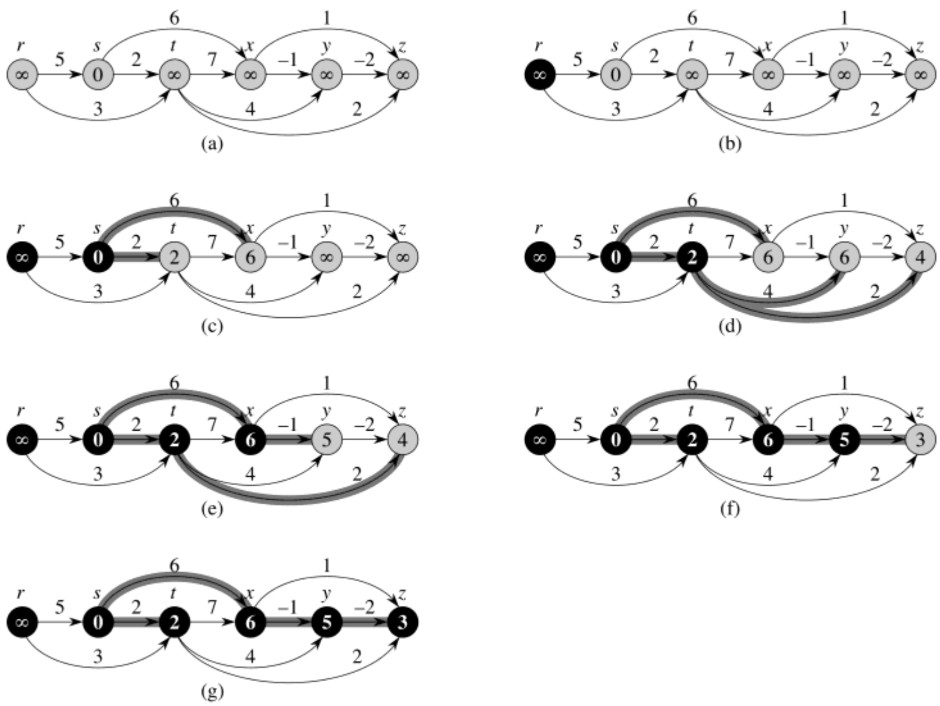
Single-source shortest path for DAGs:

```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices
  INIT-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

$\theta(V)$

$\theta(V+E)$

$\theta(V+E)$



Dijkstra's Algorithm

does not support negative edges

```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  for each vertex u ∈ G.V
      INSERT(Q, u)
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

S: set of vertices for which shortest path has been determined

Q: Priority queue for $V-S$

↳ use $v.d$

binary heap → $O(E \lg V)$

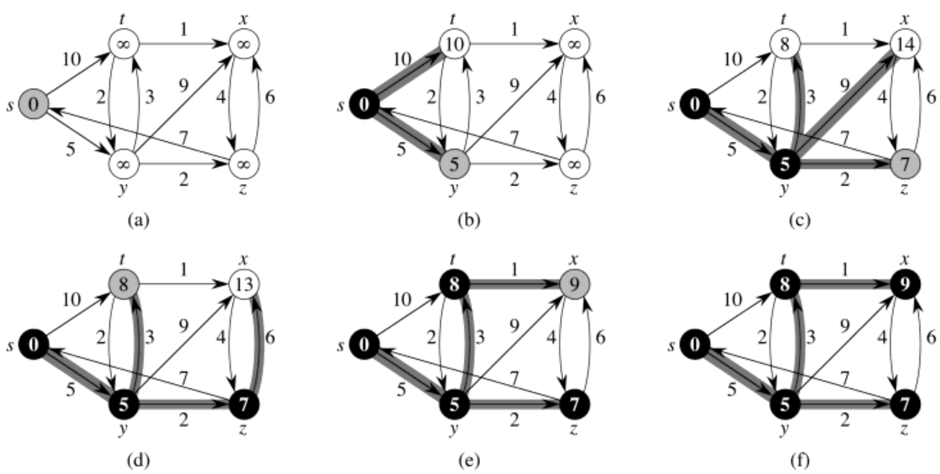
```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  for each vertex u ∈ G.V
      INSERT(Q, u)
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
          if v.d changed
              DECREASE-KEY(Q, v, v.d)
```

S: set of vertices whose final shortest path weights has been determined

Q: Priority queue for $V - S$

↳ use v.d

binary heap → $O(E \lg V)$



Feasibility problem / difference constraints

Constraints $x_j - x_i \leq b_k$ → constant; x_j, x_i → vars

example:

$$\begin{aligned} x_1 - x_2 &\leq 5 \\ x_1 - x_3 &\leq 6 \\ x_2 - x_4 &\leq -1 \\ x_3 - x_4 &\leq -2 \\ x_4 - x_1 &\leq -3 \end{aligned}$$

one solution: $x = (0, -4, -5, -3)$ with components x_1, x_2, x_3, x_4

↓ +3

also a solution: $x' = (3, -1, -2, 0)$

In general, $x + d$ is a feasible solution if x is a feasible solution

Solving it using Constraint graph:

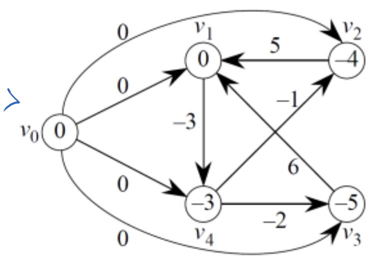
$V = \{v_0, v_1, \dots, v_n\} \to v_i$ corresponds to x_i

+ additional v_0

$E = \{(v_i, v_j) : x_j - x_i \leq b_k\} \cup \{(v_0, v_1), \dots, (v_0, v_n)\}$

$w(v_i, v_j) = b_k$ $\qquad$ $w(v_0, v_i) = 0$

parameters: $\begin{cases} n: \# x_s \\ m: \# b_s \end{cases}$



① Build constraint graph $\begin{cases} n+1 \text{ vertices} \\ m+n \text{ edges} \end{cases}$ $\theta(m+n)$

② need to run BF Alg.

↳ $O((n+1)(m+n)) = O(n^2 + nm)$

Complexity

Theorem: If G has no negative cycle then

$x = (\delta(v_0, v_1), \dots, \delta(v_0, v_n))$ is a feasible solution

Otherwise, there is no feasible solution

proof: if no negative cycle

Show: $x_j - x_i \leq b_k \to w(v_i, v_j)$

↳ $x_i = \delta(v_0, v_i)$

↳ $x_j = \delta(v_0, v_j)$

triangle inequality: $\delta(v_0, v_j) \leq \delta(v_0, v_i) + w(v_i, v_j)$

$x_j \leq x_i + b_k$

$x_j - x_i \leq b_k$ ∎

You can also show that if we have negative cycle there is no feasible solution.

All-pairs Shortest Paths

Goal: calculate $\delta(i, j)$ for all i & j

Use Bellman-Ford: run it for each vertex (as source)
$O(VE)$

$\to O(V^2 E) \xrightarrow{\text{dense graph}} O(V^4)$

Use Bellman-Ford: run for every vertex

$O(VE)$ $\longrightarrow$ $O(V^2E)$ $\xrightarrow[V=O(E^2)]{\text{dense graph}}$ $O(V^4)$

Use Dijkstra: run for every vertex

$O(E\lg V)$ $\longrightarrow$ $O(VE\lg V)$ $\xrightarrow{\text{dense graph}}$ $O(V^3\lg V)$

Can we do better? $(O(V^3))$

Use dynamic programming for all-pairs shortest-path

$W = (w_{ij})$ $\quad$ $w_{ij} = \begin{cases} 0 & \text{if } i=j \\ \text{weight of } (i,j) & \text{if } i \neq j,\ (i,j) \in E \\ \infty & \text{if } i \neq j,\ (i,j) \notin E \end{cases}$

$l_{ij}^{(m)}$ = weight of shortest path between i and j of length $\leq m$

$m=0 \longrightarrow l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i=j \\ \infty & \text{if } i \neq j \end{cases}$

$m \geq 1 \longrightarrow l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1\leq k\leq n}\{l_{ik}^{(m-1)} + w_{kj}\}\right)$

length $\leq m-1$

$i \rightsquigarrow j$

$i \rightsquigarrow k \rightarrow j$

length $\leq m-1$ $\quad$ predecessor

$= \min_{1\leq k\leq n}\{l_{ik}^{(m-1)} + w_{kj}\}$ $\longleftarrow$ when $k=j$

$l_{ij}^{(1)} = \min_{1\leq k\leq n}\{l_{ik}^{(0)} + w_{kj}\}$

$= l_{ii}^{(0)} + w_{ij}$ $\Longrightarrow$ $(l_{ij}^{(1)})$ is the weight matrix

$= w_{ij}$

What is the max length of any shortest path?

~~$|E|$~~

~~$|E|-1$~~

simple path that has no cycle

$\Downarrow$

$\cancel{|E|-1}$

$\cancel{\text{if } |E| > |V|}$

$\Downarrow$

$|V|-1$

$\delta(i,j) = l_{ij}^{(n-1)} \quad \left(= l_{ij}^{(n)} = l_{ij}^{(n+1)} \ldots \right)$

```
EXTEND(L, W, n)
  let L' = (l'_ij) be a new n × n matrix
  for i = 1 to n
      for j = 1 to n
          l'_ij = ∞
          for k = 1 to n
              l'_ij = min(l'_ij, l_ik + w_kj)
  return L'
```

$\rightarrow \Theta(n^3)$

```
SLOW-APSP(W, n)
  L^(1) = W
  for m = 2 to n − 1
      let L^(m) be a new n × n matrix
      L^(m) = EXTEND(L^(m−1), W, n)
  return L^(n−1)
```

$\rightarrow \Theta(n^4)$

$L^{(1)} \rightarrow L^{(2)} \rightarrow L^{(3)} \rightarrow \cdots \rightarrow L^{(n-1)}$

pseudocode for matrix multiplication $\quad C = A \times B$

for $i = 1$ to n

for $j = 1$ to n

$c_{ij} = 0 \rightarrow \infty$

for $k = 1$ to n

$c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$

$+ \rightarrow$ min $\quad \cdot \rightarrow +$

Extend is similar to matrix multiplication

$A^{50} = A \times A^{49} = \cdots$

$A^2 = A \times A \rightarrow A^4 = A^2 \times A^2 \rightarrow A^8 = A^4 \times A^4 \rightarrow A^{16} \rightarrow A^{32} \rightarrow A^{48} \rightarrow A^{50}$

$L^{(1)} \rightarrow L^{(2)} \rightarrow L^{(3)} \rightarrow \cdots \rightarrow L^{(n-1)}$

$\underbrace{L^{(1)} \rightarrow L^{(2)} \rightarrow L^{(4)} \rightarrow L^{(8)} \rightarrow \cdots}_{\text{how many steps?}}$

if $(n-1) = 50 \longrightarrow$ looking for $L^{(50)}$

if we perform this 6 times $\longrightarrow L^{(64)}$

```
FASTER-APSP(W, n)
  L^(1) = W
  m = 1
  while m < n − 1
      let L^(2m) be a new n × n matrix
      L^(2m) = EXTEND(L^(m), L^(m), n)
      m = 2m
  return L^(m)
```

loop $\longrightarrow \theta(\lg n)$

$\longrightarrow$ Totals $\theta(n^3 \lg n)$

Floyd-Warshall algorithm

$d_{ij}^{(k)}$ = shortest path weight of any path $i \rightsquigarrow j$ with all intermediate vertices in $\{1, \ldots, k\}$

shortest path p : $i \xrightarrow{p} j$ with all intermediate nodes in $\{1, \ldots, k\}$

$\longrightarrow$
- p does not pass through k
  - $\Rightarrow$ all intermediate vertices are in $\{1, \ldots, k-1\}$
- p passes through k
  - $\Rightarrow$ $i \rightsquigarrow k \rightsquigarrow j$

intermediate v.s of sub-shortest path are in $\{1, .. k-1\}$

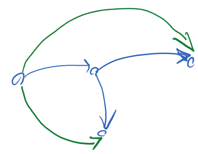
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\{d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{otherwise} \end{cases}$$

goal: $D^{(n)} = (d_{ij}^{(n)})$

FLOYD-WARSHALL(W, n)
$D^{(0)} = W$
for $k = 1$ **to** n
  let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
  for $i = 1$ **to** n
    for $j = 1$ **to** n
      $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

$\theta(n^3) = \theta(V^3)$

Transitive Closure



$G = (V, E)$

$G^* = (V, E^*)$ $E^* = \{ (u,v) : \text{there is a path } u \rightsquigarrow v \text{ in } G \}$

Calculating E^*:

- assign weight 1 to all edges
- run Floyd_Warshall
  - if $d_{uv}^{(n)} < n$ then $(u,v) \in E^*$
  - otherwise $d_{uv}^{(n)} = \infty$ and $(u,v) \notin E^*$

we can calculate it using simpler operations (using binary logic):

TRANSITIVE-CLOSURE(G, n)
let $T^{(0)} = (t_{ij}^{(0)})$ be a new $n \times n$ matrix
for $i = 1$ **to** n
  for $j = 1$ **to** n
    if $i == j$ or $(i, j) \in G.E$
      $t_{ij}^{(0)} = 1$
    else $t_{ij}^{(0)} = 0$
for $k = 1$ **to** n
  let $T^{(k)} = (t_{ij}^{(k)})$ be a new $n \times n$ matrix
  for $i = 1$ **to** n
    for $j = 1$ **to** n
      $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
return $T^{(n)}$

} initialization

binary ops are much cheaper

(min) (+)