Design and Analysis of Algorithms CS218M

Dynamic Programming

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

Detecting Subsequence

Problem Given sequences (arrays) $X = \langle x_1, x_2, \ldots, x_m \rangle$ and sequence $Z = \langle z_1, \ldots, z_k \rangle$, determine whether Z is a subsequence of X, that is there exists a sequence of indices $\langle i_1, \ldots, i_k \rangle$ such that $Z_j = X_{i_j}$.

Detecting Subsequence

Problem Given sequences (arrays) $X = \langle x_1, x_2, \ldots, x_m \rangle$ and sequence $Z = \langle z_1, \ldots, z_k \rangle$, determine whether Z is a subsequence of X, that is there exists a sequence of indices $\langle i_1, \ldots, i_k \rangle$ such that $Z_j = X_{i_j}$.
Solution: Greedy Algorithm of Complexity $\Theta(m)$

Longest Common Subsequence (LCS)

Problem [CLRS Ch. 15] Given sequences (arrays) $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, determine the longest length sequence $Z = \langle z_1, \ldots, z_k \rangle$ which is a subsequence of both X and Y.

Problem [CLRS Ch. 15] Given sequences (arrays) $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, determine the longest length sequence $Z = \langle z_1, \ldots, z_k \rangle$ which is a subsequence of both X and Y.

Brute Force Solution

Systematically generate all subsequences Z of Y. For each check if Z is a subseqeunce of X. Also remember the maximum of the length of "yes" subsequences examined so far.

Complexity?

Optimal Substructure

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

X, Y X_z, Y_a

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-1} is LCS of X_{m-1}, Y_{n-1}.

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-1} is LCS of X_{m-1}, Y_{n-1}.
- If $x_m \neq y_n$ and $z_k \neq x_m$ then

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-1} is LCS of X_{m-1}, Y_{n-1}.
- If $x_m \neq y_n$ and $z_k \neq x_m$ then Z is LCS of X_{m-1}, Y.

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-1} is LCS of X_{m-1}, Y_{n-1}.
- If $x_m \neq y_n$ and $z_k \neq x_m$ then Z is LCS of X_{m-1}, Y.
- If $x_m \neq y_n$ and $z_k \neq y_n$ then Z is LCS of $X_{\cancel{m-1}}, Y_{n-1}$

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] \;=$

Recursive Solution

Let $c[i, j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i, j] \ =$

- 0 if $i = 0$ or $j = 0$

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] \;=$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j]\ =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1,j-1]+1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1,j], c[i,j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1, j], c[i, j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Designing the DP algorithm

- Memoization table $c[0..m, 0..n]$.

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] \ =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1, j], c[i, j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Designing the DP algorithm

- Memoization table $c[0..m, 0..n]$.
- Base case?, Order of computing $c[i,j]$?

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] \; =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1, j], c[i, j-1]]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Designing the DP algorithm

- Memoization table $c[0..m, 0..n]$.
- Base case?, Order of computing $c[i,j]$?
- Complexity?

Recursive Solution

Let $c[i, j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i, j] =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1, j], c[i, j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Designing the DP algorithm

- Memoization table $c[0..m, 0..n]$.
- Base case?, Order of computing $c[i, j]$?
- Complexity? $\Theta(m \cdot n)$

Bottom UP Procedure

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i,0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

$\Theta(m)$

$\Theta(n)$

$\Theta(1)$

$\Theta(mn)$

$\Theta(mn)$

Output of the Procedure

b3

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

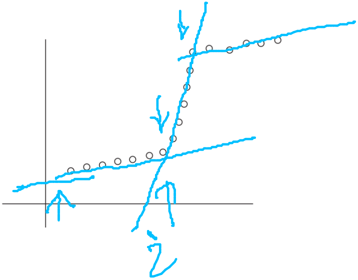
← X

2nd

C[6,7]

Segmented Least Square

Problem [KT6.3] Given a set of n points $(x_1, y_1), \ldots, (x_n, y_n)$ in x, y-plane in order $x_1 < x_2 < \ldots < x_n$, find a small set of line segments such that the soln. gives the least error squared.



Linear Regression and Error

- Given set of points P (as before) and a line L defined by $y = a \cdot x + b$, we have squared error: $ERR(L, P) = \Sigma_{i=1}^{n} \ (y_i - a \cdot x_i - b)^2$.

Linear Regression and Error

- Given set of points P (as before) and a line L defined by $y = a \cdot x + b$, we have squared error: $ERR(L, P) = \Sigma_{i=1}^{n} \ (y_i - a \cdot x_i - b)^2$.
- Line $a \cdot x + b$ giving least squared error is given by

$$a = \frac{n \sum_i x_i y_i - \left(\sum_i x_i\right) \left(\sum_i y_i\right)}{n \sum_i x_i^2 - \left(\sum_i x_i\right)^2} \quad \text{and} \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}.$$

Formulating the Problem

- if $p_i, p_{i+1}, \ldots, p_j$ belongs to a line segment then $e_{i,j}$ denotes the least squared error from these points after fitting the best line through them.
- Trade off between reducing error and reducing number of line segments.
- Each line segment incurs a cost of C.

Formulating the Problem

- if $p_i, p_{i+1}, \ldots, p_j$ belongs to a line segment then $e_{i,j}$ denotes the least squared error from these points after fitting the best line through them.
- Trade off between reducing error and reducing number of line segments.
- Each line segment incurs a cost of C.

Formulating the Problem

- if $p_i, p_{i+1}, \ldots, p_j$ belongs to a line segment then $e_{i,j}$ denotes the least squared error from these points after fitting the best line through them.
- Trade off between reducing error and reducing number of line segments.
- Each line segment incurs a cost of C.

Optimal Substructure (Recurrence 6.7)

For the subproblem $p_1, \ldots, p_j$

k

Formulating the Problem

- if $p_i, p_{i+1}, \ldots, p_j$ belongs to a line segment then $e_{i,j}$ denotes the least squared error from these points after fitting the best line through them.
- Trade off between reducing error and reducing number of line segments.
- Each line segment incurs a cost of C.

Optimal Substructure (Recurrence 6.7)

For the subproblem $p_1, \ldots, p_j$

$$opt(j) \;=\; min_{1 \leq i \leq j} \;\; (e_{i,j} + C + opt(i-1))$$

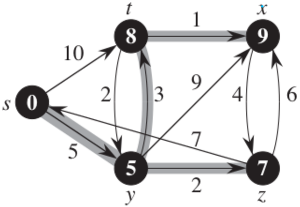
Bottom UP Procedure

```
Segmented-Least-Squares(n)
  Array M[0 ... n]
  Set M[0] = 0
  For all pairs i ≤ j
    Compute the least squares error e_{i,j} for the segment p_i, ..., p_j
  Endfor
  For j = 1, 2, ..., n
    Use the recurrence (6.7) to compute M[j]
  Endfor
  Return M[n]
```

Problem [CLRS Ch 15] Given weighted directed graph $G = (V, E), w)$ with edge-weigths $w : E \to \Re$ and a start vertex s, the aim is to find for every vertex t a shortest path from s to t (as shortest path tree π) along with weight $d.t$ of the shortest path.

$t.d = 8$

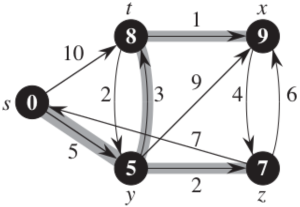
$x.d = 9$
$x.\pi = t$



Single Source Shortest Paths

Problem [CLRS Ch 15] Given weighted directed graph $G = (V, E), w)$ with edge-weigths $w : E \rightarrow \Re$ and a start vertex s, the aim is to find for every vertex t a shortest path from s to t (as shortest path tree π) along with weight $d.t$ of the shortest path.

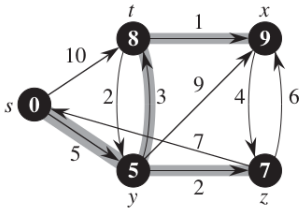
- A path $p = \langle v_0, v_1, \ldots v_k \rangle$ has weight $w(p) \;=\; \Sigma_{i=0}^{k-1}\; w(v_i, v_{i+1})$.



Single Source Shortest Paths

Problem [CLRS Ch 15] Given weighted directed graph $G = (V, E), w)$ with edge-weigths $w : E \rightarrow \Re$ and a start vertex s, the aim is to find for every vertex t a shortest path from s to t (as shortest path tree π) along with weight $d.t$ of the shortest path.

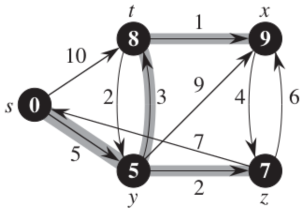
- A path $p = \langle v_0, v_1, \ldots v_k \rangle$ has weight $w(p) = \Sigma_{i=0}^{k-1} w(v_i, v_{i+1})$.
- Let $\delta(u, v) = min\ \{w(p) \mid u \xrightarrow{p} v\}$. Here $min(\emptyset) = \infty$ and $\delta(u, u) = 0$.



Single Source Shortest Paths

Problem [CLRS Ch 15] Given weighted directed graph $G = (V, E), w)$ with edge-weigths $w : E \rightarrow \Re$ and a start vertex s, the aim is to find for every vertex t a shortest path from s to t (as shortest path tree π) along with weight $d.t$ of the shortest path.

- A path $p = \langle v_0, v_1, \ldots v_k \rangle$ has weight $w(p) = \Sigma_{i=0}^{k-1} w(v_i, v_{i+1})$.
- Let $\delta(u, v) = min \{w(p) \mid u \xrightarrow{p} v\}$. Here $min(\emptyset) = \infty$ and $\delta(u, u) = 0$.
- Graph may have negative edge weights.



Properties of Shortest Paths

- If the graph has a reachable negative weight cycle, then there is no shortest path possible.

Properties of Shortest Paths

- If the graph has a reachable negative weight cycle, then there is no shortest path possible.
- If $\langle v_0, v_1, \ldots v_k \rangle$ is a shortest path from v_0 to v_k, then $\forall 1 \leq i \leq j \leq k$ the subpath $\langle v_i, \ldots v_j \rangle$ is the shortest path from v_i to v_j.
  Also, $\delta(v_0, v_k) = \delta(v_0, v_i) + \delta(v_i, v_k)$.

Properties of Shortest Paths

- If the graph has a reachable negative weight cycle, then there is no shortest path possible.
- If $\langle v_0, v_1, \ldots v_k \rangle$ is a shortest path from v_0 to v_k, then $\forall 1 \leq i \leq j \leq k$ the subpath $\langle v_i, \ldots v_j \rangle$ is the shortest path from v_i to v_j.
  Also, $\delta(v_0, v_k) = \delta(v_0, v_i) + \delta(v_i, v_k)$.
- Triangle inequality: For any edge (u, v) we have
  $$\delta(s, v) \leq \delta(s, u) + w(u, v).$$

Constraint Propagation Strategy

We will over-approximate $\delta(s, v)$ maintaining invariant $\delta(s, v) \leq v.d$.

Constraint Propagation Strategy

We will over-approximate $\delta(s, v)$ maintaining invariant $\delta(s, v) \leq v.d$.

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

Constraint Propagation Strategy

We will over-approximate $\delta(s, v)$ maintaining invariant $\delta(s, v) \leq v.d$.

INITIALIZE-SINGLE-SOURCE(G, s)

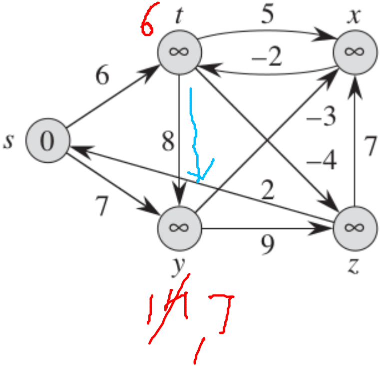
```
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

Relaxation

RELAX(u, v, w)

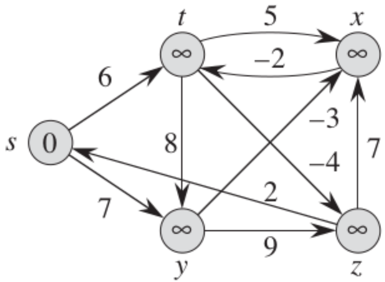
```
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

Exploring Relaxation Schedules

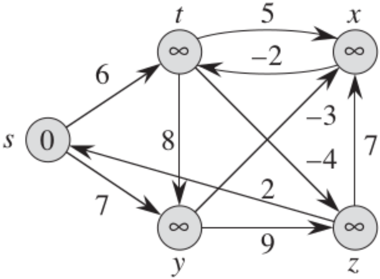


(t,y) (t,x)
(z,x) (s,t)
(t,y) (s,y)

Exploring Relaxation Schedules



Exploring Relaxation Schedules



.

Properties of Relaxation Schedules

Convergence property (Lemma 24.14)
If $s \rightsquigarrow u \rightarrow v$ is a shortest path in G for some $u, v \in V$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all times afterward.

Path-relaxation property (Lemma 24.15)
If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-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
```

Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-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
```

- Correctness?

Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-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
```

← $\Theta(V)$

$\Theta(E)$

$\Theta(VE)$

- Correctness?
- Complexity?

$\Theta(VE)$

Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-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
```

- Correctness?
- Complexity? $O(V \cdot E)$

SSP: Directed Acyclic Graphs

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

SSP: Directed Acyclic Graphs

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

- Correctness?

SSP: Directed Acyclic Graphs

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

- Correctness?
- Complexity?

SSP: Directed Acyclic Graphs

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

- Correctness?
- Complexity? $O(V + E)$

All Pairs Shortest Paths

Given directed acyclic graph G (with some negative edge weights but no negative weight cycles) where nodes are numbered $1 \ldots n$, for all $i, j \in (1 \ldots n)^2$, compute matrix D giving $d_{i,j} = \delta(i, j)$.

All Pairs Shortest Paths

Given directed acyclic graph G (with some negative edge weights but no negative weight cycles) where nodes are numbered $1 \ldots n$, for all $i, j \in (1 \ldots n)^2$, compute matrix D giving $d_{i,j} = \delta(i, j)$.

- Input graph is given as adjecency matrix W where $w_{i,j}$ gives weight of edge (i, j).

All Pairs Shortest Paths

Given directed acyclic graph G (with some negative edge weights but no negative weight cycles) where nodes are numbered $1 \dots n$, for all $i, j \in (1 \dots n)^2$, compute matrix D giving $d_{i,j} = \delta(i, j)$.

- Input graph is given as adjecency matrix W where $w_{i,j}$ gives weight of edge (i, j).
- If $\langle v_0, v_1, \dots v_k \rangle$ is a shortest path from v_0 to v_k, then $\forall 1 \leq i \leq j \leq k$ the subpath $\langle v_i, \dots v_j \rangle$ is the shortest path from v_i to v_j.
  Also, $\delta(v_0, v_k) = \delta(v_0, v_i) + \delta(v_i, v_k)$.

All Pairs Shortest Paths

Given directed acyclic graph G (with some negative edge weights but no negative weight cycles) where nodes are numbered $1 \ldots n$, for all $i, j \in (1 \ldots n)^2$, compute matrix D giving $d_{i,j} = \delta(i,j)$.

- Input graph is given as adjecency matrix W where $w_{i,j}$ gives weight of edge (i,j).
- If $\langle v_0, v_1, \ldots v_k \rangle$ is a shortest path from v_0 to v_k, then $\forall 1 \leq i \leq j \leq k$ the subpath $\langle v_i, \ldots v_j \rangle$ is the shortest path from v_i to v_j.
  Also, $\delta(v_0, v_k) = \delta(v_0, v_i) + \delta(v_i, v_k)$.
- For path $p = \langle v_1, v_2, \ldots, v_n \rangle$ vertices $\{v_2, \ldots, v_{n-1}\}$ are intermediate vertices.

Recursive Optimal Substructure

- We focus on paths where intermediate vertices are in set $\{1, \ldots, k\}$. Let $PATHS^{(k)}[i, j]$ denote simple paths from vertex i to j with intermediate vertices in $1 \ldots k$.

Recursive Optimal Substructure

- We focus on paths where intermediate vertices are in set $\{1, \ldots, k\}$. Let $PATHS^{(k)}[i, j]$ denote simple paths from vertex i to j with intermediate vertices in $1 \ldots k$.
- $$PATHS^{(k)}[i,j] = PATHS^{(k-1)}[i,j] \cup PATHS^{(k-1)}[i,k] \cdot PATHS^{(k-1)}[i,k]$$

Recursive Optimal Substructure

- We focus on paths where intermediate vertices are in set $\{1, \ldots, k\}$. Let $PATHS^{(k)}[i,j]$ denote simple paths from vertex i to j with intermediate vertices in $1 \ldots k$.
- $$PATHS^{(k)}[i,j] = PATHS^{(k-1)}[i,j] \cup PATHS^{(k-1)}[i,k] \cdot PATHS^{(k-1)}[i,k]$$
- Let $d^{(k)}{}_{i,j}$ denote the weigth of shortest path in $PATHS^{(k)}[i,j]$. Then, $d_{i,j}^{(n)} = \delta(i,j)$.

Recursive Optimal Substructure

- We focus on paths where intermediate vertices are in set $\{1, \dots, k\}$. Let $PATHS^{(k)}[i,j]$ denote simple paths from vertex i to j with intermediate vertices in $1 \dots k$.
- $PATHS^{(k)}[i,j] = PATHS^{(k-1)}[i,j]$
  $\cup \; PATHS^{(k-1)}[i,k] \cdot PATHS^{(k-1)}[i,k]$
- Let $d^{(k)}{}_{i,j}$ denote the weigth of shortest path in $PATHS^{(k)}[i,j]$. Then, $d^{(n)}_{i,j} \; = \; \delta(i,j)$.
- Optimal Substructure

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

Floyd-Warshall All-Pairs SP Algorithm

FLOYD-WARSHALL(W)

$n = W.rows$
$D^{(0)} = W$
for $k = 1$ **to** n
    let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ 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 $D^{(n)}$

Floyd-Warshall All-Pairs SP Algorithm

FLOYD-WARSHALL(W)

$n = W.rows$
$D^{(0)} = W$
for $k = 1$ **to** n
    let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ 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 $D^{(n)}$

- Complexity?

Floyd-Warshall All-Pairs SP Algorithm

FLOYD-WARSHALL(W)

$n = W.rows$
$D^{(0)} = W$
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 $D^{(n)}$

- Complexity? $O(n^3)$