Dynamic Programming

- General method
- String edit

General method

- Dynamic programming is an algorithm design method for solving multi-stage decision making problems.
- For some problems greedy methods can be used to generate optimal solutions.
- For many other problems, they do yield optimal solution.
- One way of finding the global optimum is that all decision sequences are enumerated from which the best decision is picked.
- But the time and space requirement may be prohibitive.

Dynamic Programming technique

- In dynamic programming a collection of decision sequences are generated.
- Essential difference from greedy method
- In Greedy method only one sequence of decisions is generated. e.g.
  - Knapsack: We chose a sequence of objects fractions such that it optimizes the objective function.
  - Shortest Path: We chose a sequence of vertices such that it minimizes the path length.
- Solve smaller subproblems of the same type.

- Solution to problem is represented as a recurrence relation
- which links the solutions to the smaller problems into a solution to the whole problem.
- It employs principle of optimality: *An optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must produce an optimal decision sequence with regard to the state resulting from the first decision.*
- It reduces the number of possible solutions to be checked.
    - By eliminating redundant information (overlapping problems)
    - Removing cases that cannot be optimal

Floyd’s all pairs shortest path

String edit

Dynamic Programming

Warshall transitive closure

Traveling salesman

String edit

- In computational biology, need to measure similarity between two gene sequences.
- In natural language processing, need to automatically make corrections of optical character recognition (OCR) errors.
- Spell checkers software to identifying spelling mistakes.
- The objective is to find matches for short strings in many longer texts.
- Given two strings

  $X = x_1 x_2 \dots x_n$, $Y = y_1 y_2 \dots y_m$

  transform X into Y using a series of edit operation

- The basic edit operations and costs are defined
  1. Insert e.g. $abc \rightarrow abbc$
     cost associated is 1.
  2. Delete e.g. $abc \rightarrow ac$
     cost associated is 1
  3. Change (mutate), e.g, $abc \rightarrow aec$
     cost associated is 2.
- Example: How similar is $X=a, a, b, a, b$ to $Y=b, a, b, b$?
- Need to find a series of operations with smallest cost between two strings.
- What is the minimum-cost edit sequence in general ?

- Example: Transform $X = x_1 x_2 \ldots x_n = a, a, b, a, b$ into $Y = y_1 y_2 \ldots y_m = b, a, b, b$
  - Solution 1: Delete each x_i and Insert each y_j

    $\cancel{a}, \cancel{a}, \cancel{b}, \cancel{a}, \cancel{b}$ $b\ a\ b\ b$

    Total cost is 9

  - Solution 2: Delete $x_{1,}\ x_2$ and Insert y_4

    $\cancel{a}, \cancel{a}, b, a, b, b$

    Total cost of 3

- We use dynamic programming to find the optimal decision sequences

String edit – Decision sequence

- ε=*D**ICCICDDI***, is an optimal sequence of decisions for any arbitrary string
- It can be decomposed into an optimal first decision

  f*= *D(elete)

  followed by a sequence of optimal decisions

  ε'=***ICCICDDI***
- Let the resulting string of applying f to X be X'
- ε' is a minimum-cost edit sequence that transforms X' into Y
- Thus the principle of optimality applies. *An optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must produce an optimal decision sequence with regard to the state resulting from the first decision.*

- A dynamic programming solution is obtained as follows:
  - Define $cost(i, j)$ as the minimum-cost edit sequence for transforming $x_1, \ldots x_i$ to $y_1, \ldots y_j$ for $0 \le i \le n,\ 0 \le j \le m$.
  - $cost(n, m)$ is the cost of an optimal edit sequence
  - $cost(0,0) = 0$, since the two sequences are empty
  - Delete sequences:

    $$j = 0,\ i > 0,\ cost(i,0) = cost(i-1,0) + D(x_i)$$
  - Insert sequences:

    $$i = 0,\ j > 0,\ cost(0, j) = cost(0, j-1) + I(y_j)$$

- If $j > 0$, $i > 0$, $x_1, \dots x_i$ can be transformed to $y_1, \dots y_j$ in the following three ways:
    1. Transform $x_1, \dots x_{i-1}$ to $y_1, \dots y_j$ using a minimum cost edit and then delete x_i:
       $$cost(i, j) = cost(i-1, j) + D(x_i)$$
    2. Transform $x_1, \dots x_{i-1}$ to $y_1, \dots y_{j-1}$ using minimum cost edit and then change x_i to y_j :
       $$cost(i, j) = cost(i-1, j-1) + C(x_i, y_j)$$
    3. Transform $x_1, \dots x_i$ to $y_1, \dots y_{j-1}$ using a minimum cost edit and then insert y_j :
       $$cost(i, j) = cost(i, j-1) + I(y_j)$$

- $\text{cost}(i,j) = \begin{cases} 0 & \text{if} \quad i=j=0 \\ cost(i-1,0) + D(x_i) & \text{if} \quad i>0,\ j=0 \\ cost(0,j-1) + I(y_j) & \text{if} \quad j>0, i=0 \\ cost'(i,j) & \text{if} \quad j>0, i>0 \end{cases}$

where $\text{cost}'(i,j) = \min\{cost(i-1,j) + D(x_i),\ cost(i,j-1) + I(y_j),$
$cost(i-1,j-1) + C(y_j)\ \}$

- We have to calculate *cost(i, j)* for all $0 \le i \le n$, $0 \le j \le m$, filled by a dynamic programming matrix table of $(n+1)$ rows and $(m+1)$ columns.
- The zeroth row/ column can be firstly filled since they corresponds to a series of insertions/deletions.
- The whole algorithm takes $O(mn)$ time.

Example: Transform

$X = x_1 x_2 \ldots x_n = a, a, b, a, b$

into

$Y = y_1 y_2 \ldots y_m = b, a, b, b$

- We first create dynamic programming matrix and fill boundaries (zeroth row and columns)

Y

X						
			b	*a*	*b*	*b*
		0	*1*	*2*	*3*	*4*
	a	*1*				
	a	*2*				
	b	*3*				
	a	*4*				
	b	*5*				

- $\text{cost}(i,j) = \begin{cases} 0 & \text{if} \quad i = j = 0 \\ cost(i-1,0) + D(x_i) & \text{if} \quad i > 0,\ j = 0 \\ cost(0,j-1) + I(y_j) & \text{if} \quad j > 0, i = 0 \end{cases}$

- Calculate row entries one by one

- $cost(1,1) = \min\{cost(0,1) + D(x_1),$
  $cost(1,0) + I(y_1), cost(0,0) + C(x_1, y_1)\}$
  $= \min\{2,\ 2, 2\} = 2$

- $cost(1,2) = \min\{cost(0,2) + D(x_1),$
  $cost(1,1) + I(y_2),\ cost(0,1) + C(x_1, y_2)\}$
  $= \min\{3,\ 3, 1\} = 1$

- The rest of entries are computed similarly

Y

X

		b	*a*	*b*	*b*
	0	*1*	*2*	*3*	*4*
a	*1*	*2*	*1*		
a	*2*				
b	*3*				
a	*4*				
b	*5*				

- $cost(5,4) = 3$ is what we are looking for.
- The minimum edit sequence can be obtained by backward trace from $cost(n, m)$.
- This back-ward trace is enabled by recording which of the three options for $i > 0, j > 0$ yielded the minimum cost for each i and j.
- Backtrack from bottom right to top left following min cost path:

  delete x_1, delete x_2, insert y_4

Y

X

		b	*a*	*b*	*b*
	0	1	2	3	4
a	1	2	1	2	3
a	2	3	2	3	4
b	3	2	3	2	4
a	4	3	2	3	4
b	5	4	3	2	3

Dynamic Programming

- General method
- Floyd's all-Pairs Shortest Path

General method

- Dynamic programming is an algorithm design method for solving multi-stage decision making problems.
- For some problems greedy methods can be used to generate optimal solutions.
- For many other problems, they do yield optimal solution.
- One way of finding the global optimum is that all decision sequences are enumerated from which the best decision is picked.
- But the time and space requirement may be prohibitive.

Dynamic Programming technique

- In dynamic programming a collection of decision sequences are generated.
- Essential difference from greedy method
- In Greedy method only one sequence of decisions is generated. e.g.
  - Knapsack: We chose a sequence of objects fractions such that it optimizes the objective function.
  - Shortest Path: We chose a sequence of vertices such that it minimizes the path length.
- Solve smaller subproblems of the same type.

- Solution to problem is represented as a recurrence relation
- which links the solutions to the smaller problems into a solution to the whole problem.
- It employs principle of optimality*: An optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must produce an optimal decision sequence with regard to the state resulting from the first decision.*
- It reduces the number of possible solutions to be checked.
  - By eliminating redundant information (overlapping problems)
  - Removing cases that cannot be optimal

Floyd’s all pairs shortest path

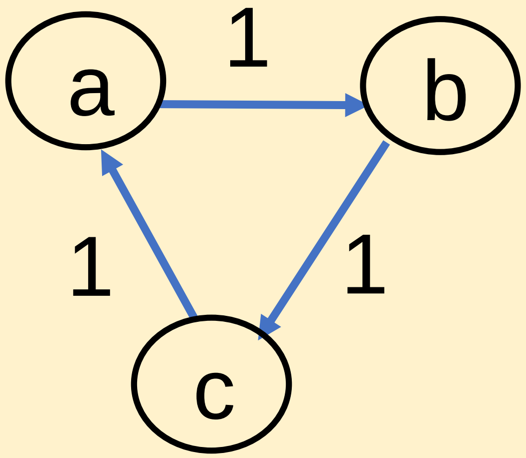
String edit

Dynamic Programming

Warshall transitive closure

Traveling salesman

Floyd's All-Pairs Shortest Path



- Given a weighted connected graph $G\ (V, E)$
- Find the distances (shortest paths) from each vertex to all other vertices.
- Solution can be represented as a distance matrix D, where $D(i, j)$ is the length of shortest path from vertex i to j

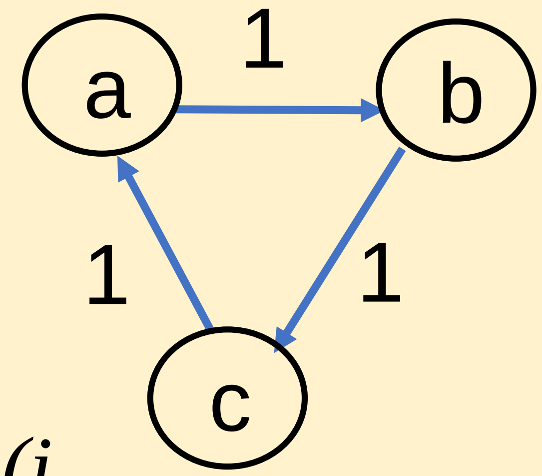
$$W = \begin{bmatrix} 0 & 1 & \infty \\ \infty & 0 & 1 \\ 1 & \infty & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

What is the difference between W and D?

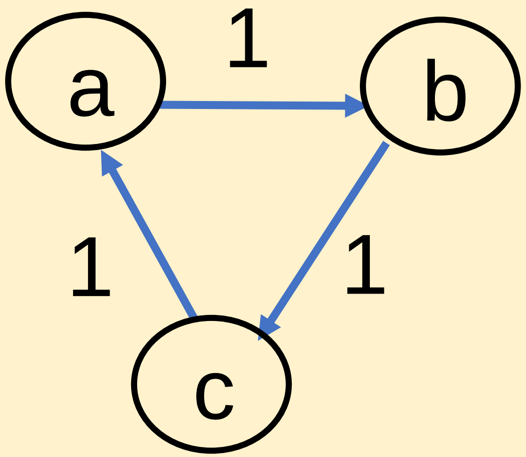
- There are three intermediate vertices **a, b, c**
- Denote the allowed number of intermediate vertex for any path as k.
- Increase k from zero to n

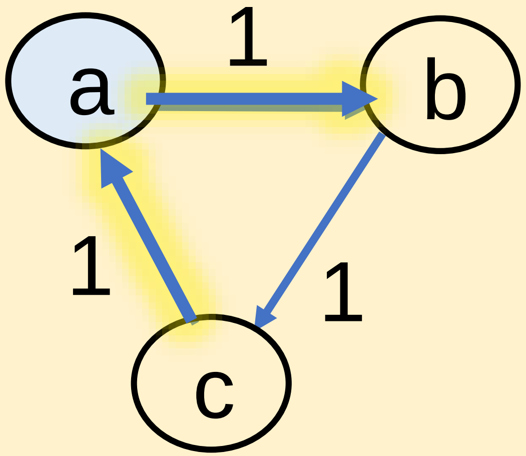
  (allowing **a, b, c** one at a time)
- We may regard the construction of a shortest (i, j) path as first requiring a decision as which is the shortest past path for k intermediate vertices.
- We need to find two shortest paths, one from i to k other from k to j, to see if this is shorter than before.



$$D = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

Example





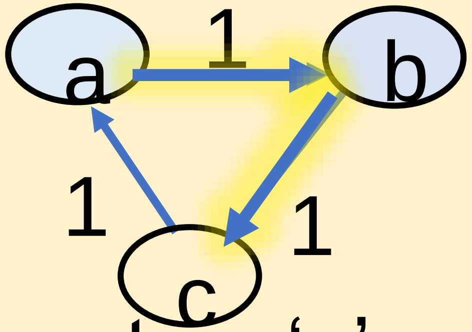
- Compute the distance matrix *'D'* through a series of *n-by-n* matrices $D^0, D^1, D^2, \ldots\ldots, D^n$

$$D^0 = \begin{bmatrix} 0 & 1 & \infty \\ \infty & 0 & 1 \\ 1 & \infty & 0 \end{bmatrix} \qquad D^1 = \begin{bmatrix} 0 & 1 & \infty \\ \infty & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

- With an intermediate vertex ‘a’, i.e. opens a new path via ‘a’

$$D^1(c, b) = \min(D^0(c,b), D^0(c,a) + D^0(a,b)) = 2$$

$$D^1 = \begin{bmatrix} 0 & 1 & \infty \\ \infty & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

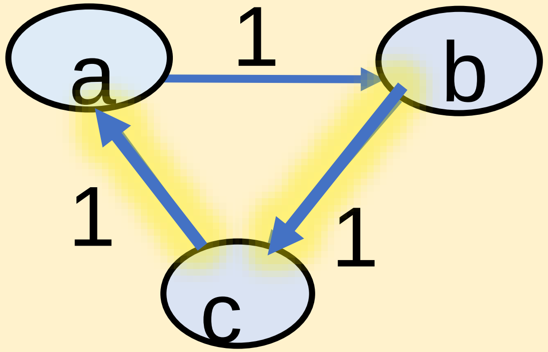


$$D^2 = \begin{bmatrix} 0 & 1 & 2 \\ \infty & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

- With two intermediate vertex ‘a’ and ‘b’, i.e. opens a new path via ‘b’

$$D^2(a, c) = \min\left(D^1(a,c)\,,\, D^1(a,b) + D^1(b,c)\right) = 2$$

$$D^2 = \begin{bmatrix} 0 & 1 & 2 \\ \infty & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$



$$D^3 = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

- With three intermediate vertex ‘a’ and ‘b’, and ‘c’, i.e. opens a new path via ‘c’

$$D^3(b, a) = \min\left(D^2(b,a)\,,\, D^2(b,c) + D^2(c,a)\right) = 2$$

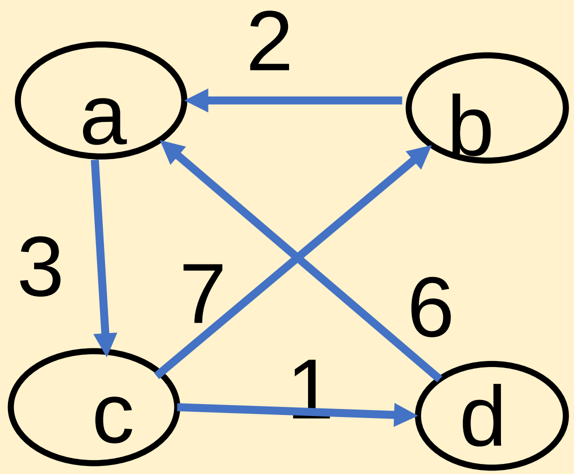
- *Input: Weighted graph W of size n*
- *Output: Distance matrix of all pair shortest paths*
- *Copy weight matrix W to D*
- *Elements of each matrix D^k can be computed from its immediate predecessor D^{k-1} by recurrence*

```
Algorithm Floyd(Weighted Graph W){
    for(i= 1, i <=n, i+ +)
      for(j= 1, j <=n, j+ +)
        D[i, j] =W[i, j];
    for(k= 1, k <=n, k+ +)
        f or(i= 1, i <=n, i+ +)
          for(j= 1;j <=n;j+ +)
      D[i, j] = min(D[i, j],  D[i, k] +D[k; j])
}
```

Elements of each matrix D^k can be computed from its immediate predecessor D^{k-1} with the following recurrence:

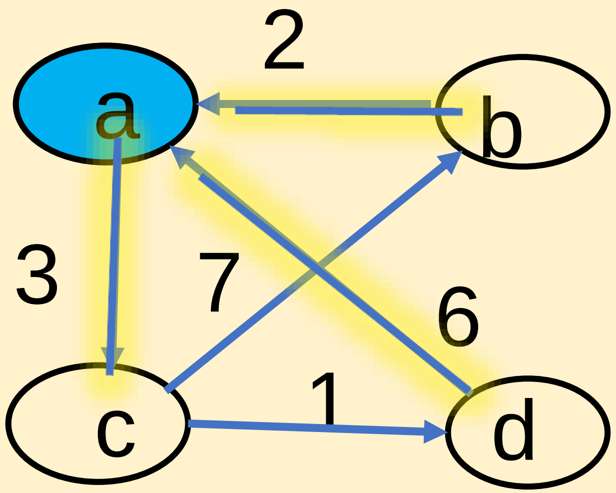
- $d_{i,j}^0 = W_{i,j}^0, \quad d_{i,j}^k = \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\},$

Example: Find all pair shortest paths for the graph below



$$W = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix}$$

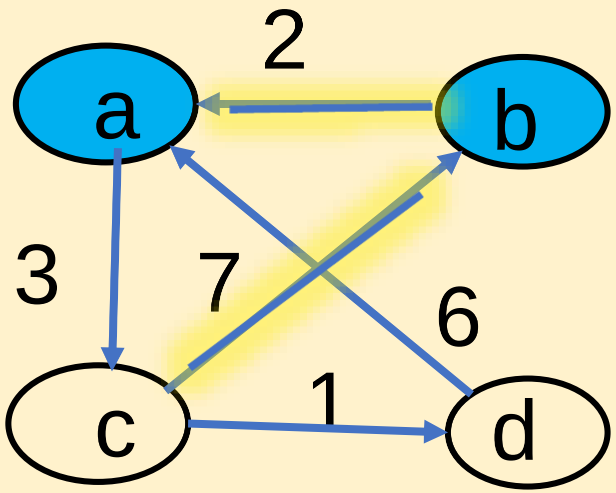
$$D^0 = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix}$$



$$D^0 = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix}$$

Introduction of an intermediate vertex ***a*** gives two detours

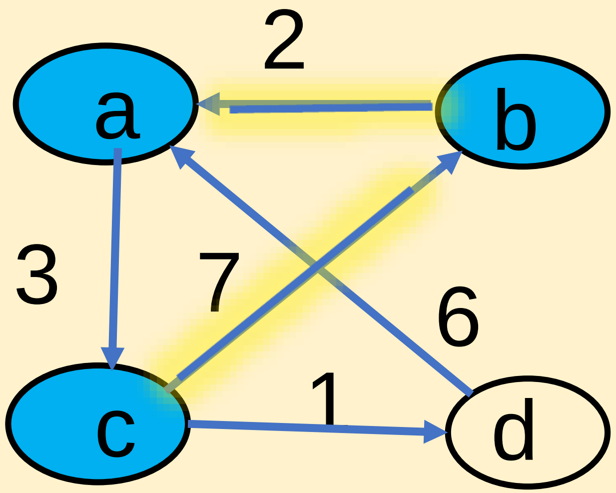
$$D^1 = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \min(2+3, \infty) & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \min(6+3, \infty) & 0 \end{bmatrix} = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$



$$D^1 = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$

Introduction of an intermediate vertex ***b*** gives one detours

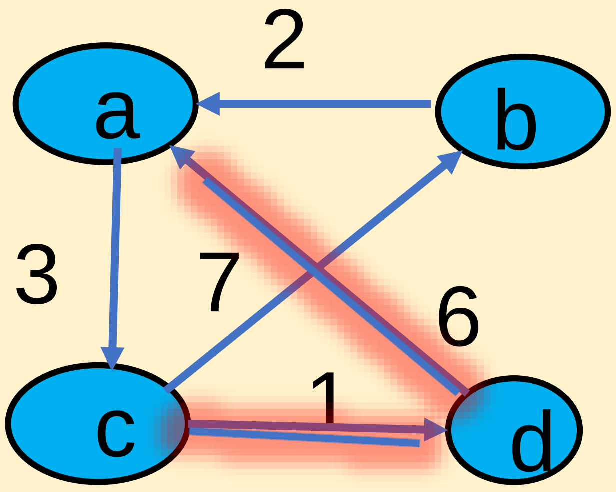
$$D^2 = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \min(7+2, \infty) & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$



$$D^2 = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$

Introduction of an intermediate vertex ***c*** gives two detours

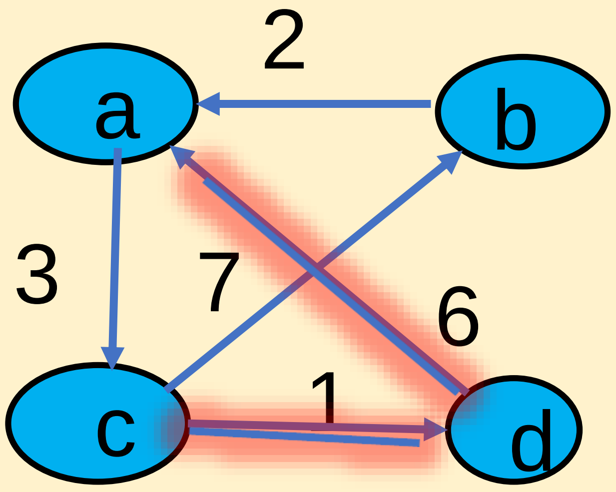
$$D^3 = \begin{bmatrix} 0 & \min(3+7,\infty) & 3 & \min(3+1,\infty) \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$



$$D^3 = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$

Introduction of an intermediate vertex ***d*** gives new detour

$$D^4 = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & \infty \\ \min(1+6, 9) & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & \infty \\ 7 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$



$$D^4 = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & \infty \\ 7 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$

Introduction of an intermediate vertex ***a and c*** give two detours

$$D^5 = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & \min(2+3+1, \infty) \\ 7 & 7 & 0 & 1 \\ 6 & \min(6+3+7, \infty) & 9 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix}$$