TO DEVELOP A DYNAMIC PROGRAMMING SOLUTION, WE CONSIDER THE GENERAL SUBPROBLEM OF FINDING AN OPTIMAL PARENTHESIZATION OF $A_i \cdots A_j$ WHERE $1 \le i \le j \le n$.

OBSERVE THAT AN OPTIMAL PARENTHESIZATION SPLITS $A_i \cdots A_j$ INTO

$$(A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j)$$

FOR SOME $k$ $(i \le k < j)$.

NOTE ALSO THAT IF $A_i \cdots A_j$ IS OPTIMALLY PARENTHESIZED, THEN SO ARE BOTH $(A_i \cdots A_k)$ AND $(A_{k+1} \cdots A_j)$

PROOF: IF THE PARENTHESIZATION OF $(A_i \cdots A_k)$ IS NOT OPTIMAL, THEN WE CAN REPLACE IT WITH AN OPTIMAL ONE YIELDING A PARENTHESIZATION OF $A_i \cdots A_k$ WITH FEWER SCALAR MULTIPLICATIONS. THIS CONTRADICTS THAT OUR ORIGINAL PARENTHESIZATION OF $A_i \cdots A_k$ WAS OPTIMAL. ///

THUS THE PRINCIPLE OF OPTIMALITY IS SATISFIED IN THIS PROBLEM.

$$\underset{(p_{i-1} \times p_k)}{\overset{\text{OPTIMAL}}{(A_i \cdots A_k)}} \cdot \underset{(p_k \times p_j)}{\overset{\text{OPTIMAL}}{(A_{k+1} \cdots A_j)}} \quad \text{OPTIMAL}$$

LET $m[i,j]$ DENOTE THE MINIMUM NUMBER OF SCALAR MULTIPLICATIONS NECESSARY TO COMPUTE $A_i \cdots A_j$. IF $i=j$ THEN $m[i,j]=0$ SINCE THE PRODUCT CONSISTS OF JUST ONE MATRIX.

IF $i \le k < j$, AND $k$ IS THE SPLIT POSITION OF AN OPTIMAL PARENTHESIZATION THEN

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$$

SINCE WE DON'T KNOW $k$, WE DEFINE IN GENERAL

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \left( m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & i < j \end{cases}$$

Ex. $n=5$; $P_0=10$, $P_1=20$, $P_2=30$, $P_3=10$, $P_4=40$, $P_5=10$

TABLE m

| $i \backslash j$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 6000 | 8000 | 12000 | 13000 |
| 2 | x | 0 | 6000 | 14000 | 12000 |
| 3 | x | x | 0 | 12000 | 7000 |
| 4 | x | x | x | 0 | 4000 |
| 5 | x | x | x | x | 0 |

FOR INSTANCE, TO COMPUTE $m[2,5]$:

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + P_1 P_2 P_5 = 13000 & (k=2) \\ m[2,3] + m[4,5] + P_1 P_3 P_5 = \boxed{12000} & (k=3) \\ m[2,4] + m[5,5] + P_1 P_4 P_5 = 22000 & (k=4) \end{cases}$$

OBSERVE THAT TO COMPUTE $m[2,5]$ ONE NEEDS ENTRIES $m[2, 2..4]$ AND $m[3..5, 5]$. THUS TO FILL THE TABLE, FIRST INITIALIZE THE MAIN DIAGONAL TO 0, THEN SUCCESSIVELY FILL EACH OFF DIAGONAL ABOVE THE MAIN.

i.e. $m[i,i]=0$ $(1 \le i \le n)$

THEN $m[i,i+1]$ $(1 \le i \le n-1)$

$m[i,i+2]$ $(1 \le i \le n-2)$

AND IN GENERAL

$m[i,i+l]$ $(1 \le i \le n-l)$

FOR $l = 1, 2, \ldots, n-1$

SEE P. 336 FOR PSEUDO-CODE.

FROM THIS TABLE WE CAN RE-CONSTRUCT THE VALUES K, WHICH GIVES THE SPLIT POINTS FOR EACH SUBPROBLEM.

A MORE EFFICIENT APPROACH IS TO STORE K VALUES IN A PARALLEL TABLE $S[i,j]$ AS WE CONSTRUCT $m[i,j]$. (P. 336).

EX. SAME AS ABOVE, WE SEE $S[2,5] = 3$, AND

TABLE S

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | X | 1 | 1 | 3 | 3 |
| 2 | X | X | 2 | 3 | 3 |
| 3 | X | X | X | 3 | 3 |
| 4 | X | X | X | X | 4 |
| 5 | X | X | X | X | X |

FROM THIS TABLE WE CAN CONSTRUCT THE OPTIMAL PARENTHESIZATION:

$$(A_1(A_2A_3))(A_4A_5).$$

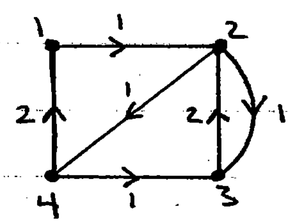
## All Pairs Shortest Paths (APSP) (25.2)

Consider a directed graph in which a weight (cost) is assigned to each directed edge.

We write $G = (V, E)$ where $V$ is the vertex set and $E$ is the set of directed edges. The <u>adjacency matrix</u> of $G$ is defined as $W = (w_{ij})$ where

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

<u>Ex.</u>




$V = \{1, 2, 3, 4\}$

$E = \{(1,2), (2,4), (4,1), (4,3), (3,2), (2,3)\}$

$$W = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & 1 \\ \infty & 2 & 0 & \infty \\ 2 & \infty & 1 & 0 \end{pmatrix}$$

THE <u>WEIGHT</u> OF A DIRECTED $i$-$j$ PATH ($i, j \in V$) IS THE SUM OF THE WEIGHTS OF EACH OF ITS DIRECTED EDGES.

<u>PROBLEM</u>: (APSP)
FOR EACH PAIR $(i,j) \in V \times V$, DETERMINE AN $i$-$j$ PATH OF MINIMUM WEIGHT. (ALSO CALLED A SHORTEST PATH.)

AGAIN THERE ARE REALLY TWO PROBLEMS

- DETERMINE THE MINIMUM PATH WEIGHTS FOR EACH $(i,j)$
- DETERMINE SHORTEST $i$-$j$ PATHS.

WE CONCENTRATE ON THE FIRST PROBLEM, LEAVING THE SECOND AS AN EXCERCISE.

## <u>FLOYD-WARSHALL ALGORITHM</u>

AN <u>INTERMEDIATE VERTEX</u> OF A DIRECTED PATH $P = (v_1, v_2, \ldots, v_\ell)$ IS ANY VERTEX OTHER THAN $v_1$ OR $v_\ell$, i.e. ONE OF THE VERTICES $\{v_2, \ldots, v_{\ell-1}\}$.

Let $G=(V,E)$ be a directed graph with $V=\{1,2,\ldots,n\}$. Define subsets $V_k$ of $V$ as follows

$$V_k = \begin{cases} \emptyset & k=0 \\ \{1,2,\ldots,k\} & 1 \le k \le n \end{cases}$$

Let $(i,j) \in V \times V$ and $1 \le k \le n$. Let $p$ denote a minimum weight path amongst all $i$-$j$ paths with intermediate vertices in $V_k$.

Now observe that we have two alternatives

- $k$ is not an intermediate vertex of $p$. In this case $p$ is also of minimum weight amongst all $i$-$j$ paths with intermediate vertices in $V_{k-1}$.

- $k$ is an intermediate vertex of $p$. We can decompose $p$ into subpaths $p_1$ and $p_2$:

$$i \xrightarrow{p_1} k \xrightarrow{p_2} j$$

Note vertex $k$ is not intermediate to either $p_1$ or $p_2$.

THUS $P_1$ HAS MINIMUM WEIGHT AMONGST ALL $i$-$k$ PATHS WITH INTERMEDIATE VERTICES IN $V_{k-1}$, AND LIKEWISE $P_2$ HAS MINIMUM WEIGHT AMONGST ALL $k$-$j$ PATHS WITH INTERMEDIATE VERTICES IN $V_{k-1}$.

THESE OBSERVATIONS SHOW APSP EXHIBITS OPTIMAL SUBSTRUCTURE, NECESSARY FOR DYNAMIC PROGRAMMING.

LET $d_{ij}^{(k)}$ DENOTE THE WEIGHT OF A MINIMUM WEIGHT $i$-$j$ PATH WITH ALL INTERMEDIATE VERTICES IN $V_k$.

WHEN $k=0$, SUCH A PATH HAS NO INTERMEDIATE VERTICES, HENCE AT MOST ONE EDGE. THUS $d_{ij}^{(0)} = w_{ij}$.

THE ABOVE OBSERVATIONS SHOW THAT FOR $1 \le k \le n$ WE HAVE

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right).$$

LET $D^{(k)}$ DENOTE THE MATRIX $\left(d_{ij}^{(k)}\right)$. THEN WE SEEK $D^{(n)}$ GIVEN $D^{(0)} = W$.

## FLOYD-WARSHALL (W)

1.) $n \leftarrow$ Rows[W]
2.) $D^{(0)} \leftarrow W$
3.) for $k \leftarrow 1$ to $n$
4.) for $i \leftarrow 1$ to $n$
5.) for $j \leftarrow 1$ to $n$
6.) $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
7.) return $D^{(n)}$

SINCE (6) TAKES TIME $O(1)$, FLOYD-WARSHALL RUNS IN TIME $\Theta(n^3)$.

NOTE THE ABOVE ALGORITHM ALSO USES MEMORY $n^3$. IT IS POSSIBLE TO ACCOMPLISH THIS WITH JUST $n^2$ MEMORY (EXERCISE.)

TO CONSTRUCT SHORTEST PATHS WE COULD USE $D = D^{(n)}$ TO DETERMINE THE PREDECESSOR MATRIX $\Pi = (\pi_{ij})$, WHERE

$\pi_{ij}$ = PREDECESSOR OF $j$ ALONG A SHORTEST $i$-$j$ PATH

ALTERNATIVELY WE COULD DETERMINE INTERMEDIATE PREDECESSOR MATRICES $\Pi^{(k)} = (\pi_{ij}^{(k)})$ $(0 \le k \le n)$

$\pi_{ij}^{(k)}$ = PREDECESSOR OF $j$ ALONG A SHORTEST $i$-$j$ PATH AMONGST THOSE WITH INTERMEDIATE VERTICES IN $V_k$.

(SEE P.632 FOR DETAILS.)

EXCERCISE

- RUN FLOYD-WARSHALL ON THE WEIGHTED DIGRAPH IN PRECEEDING EXAMPLE
- WRITE AN ALGORITHM TO DETERMINE $\Pi$ FROM $D = D^{(n)}$
- ALTER FLOYD-WARSHALL TO BUILD $\Pi^{(k)}$ $(0 \le k \le n)$ AS YOU GO.
- WRITE AN ALGORITHM TO PRINT A SHORTEST $i$-$j$ PATH GIVEN $\Pi = \Pi^{(n)}$.

READ

- LONGEST COMMON SUBSEQUENCE (15.4)
- OPTIMAL BINARY SEARCH TREES (15.5)