Data Structures and Algorithms
Lecture notes: Algorithm paradigms: Dynamic programming, Part 2

Lecturer: Michel Toulouse

Hanoi University of Science and Technology
michel.toulouse@soict.hust.edu.vn

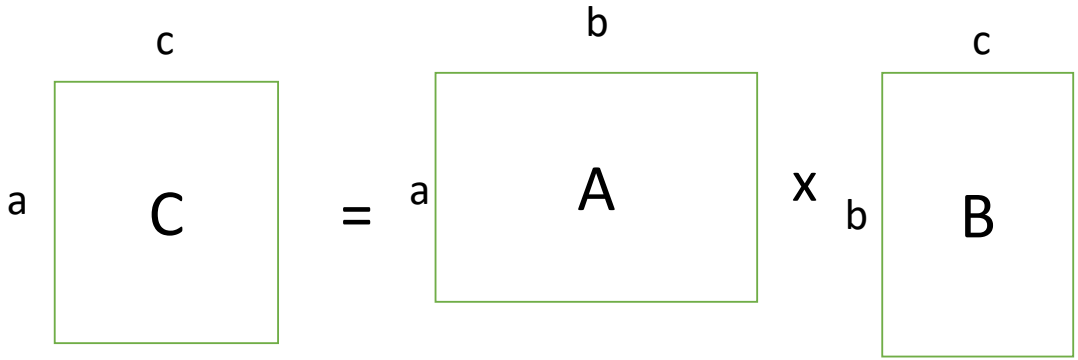
22 décembre 2021

Outline

Matrix-Chain Multiplication Problem (MCM)

Given a sequence of matrices $A_1, A_2, \ldots, A_n$, where matrix A_i has size $a \times b$, find a full parenthesization of the product $A_1 A_2 \cdots A_n$ such that the number of scalar multiplications required to compute $A_1 A_2 \cdots A_n$ is minimized.

Recall that to compute the product of an $a \times b$ matrix with a matrix $b \times c$ requires abc scalar multiplications.



MCM Example

We can express the sizes of the sequence of n matrices $a \times b, b \times c, c \times d, d \times e$ as a sequence $d_0, d_1, \ldots, d_n$ of $n+1$ positive integers, since the number of columns of A_i is the number of rows of A_{i+1} for each i.

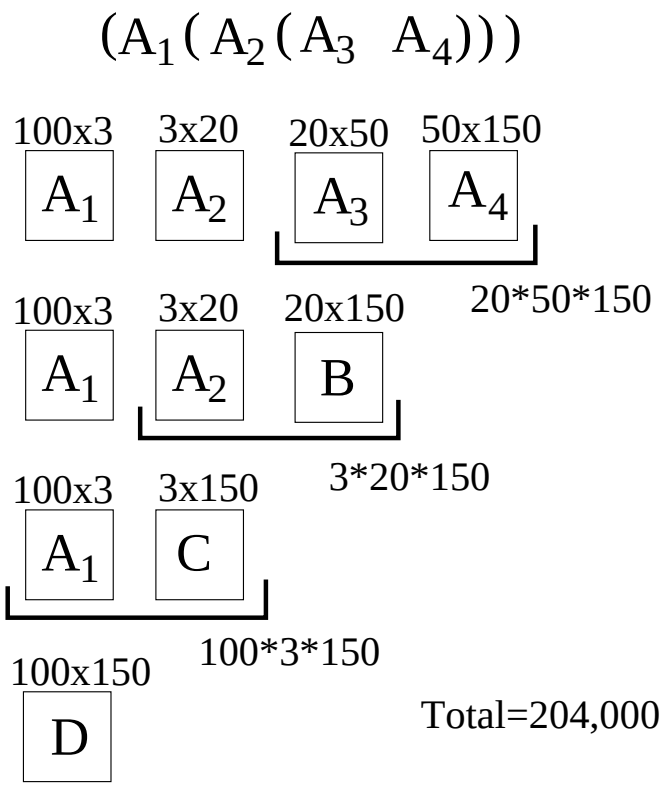
If we are given four matrices, A_1, A_2, A_3, and A_4 with sizes 100×3, 3×20, 20×50, 50×150, they can be represented by the sequence $100, 3, 20, 50, 150$.

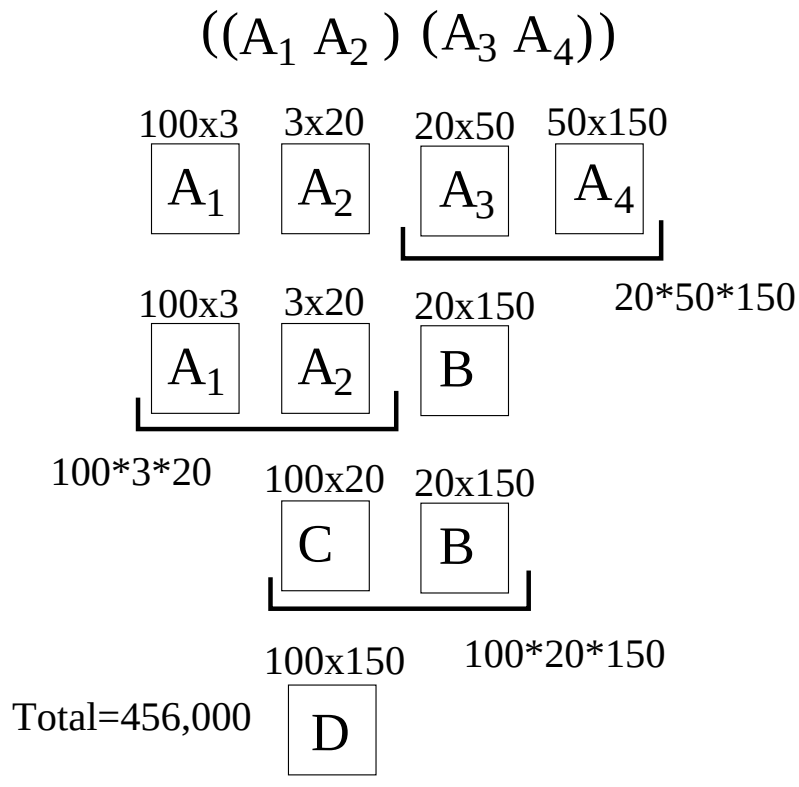
Assume we want to compute the product $A_1A_2A_3A_4$.

Two (of the 5) ways to compute this product are $(A_1(A_2(A_3A_4)))$ and $((A_1A_2)(A_3A_4))$.

MCM Example

Computing the product as $(A_1(A_2(A_3A_4)))$ requires 208,500 scalar multiplications, and computing it as $((A_1A_2)(A_3A_4))$ requires 456,000 scalar multiplications.





Solving MCM

Let $M[1, n]$ be the minimum number of scalar multiplications for parenthesizing the sequence of matrices $A_1 A_2 \cdots A_n$

The optimal solution $M[1..n]$ will look like $(A_1 \cdots A_k)(A_{k+1} \cdots A_n)$, where $1 \leq k < n$, and the two halves are parenthesized in some way.

Therefore, a divide-and-conquer approach for the matrix multiplication problem is to divide the problem $A_1 A_2 \cdots A_n$ into two subproblems $A_1 A_2 \cdots A_k$ and $A_{k+1} \cdots A_n$.

In the examples above,

- $(A_1(A_2(A_3 A_4)))$, $k = 1$ and the two subproblems are A_1 and $(A_2 A_3 A_4)$,
- $((A_1 A_2)(A_3 A_4))$, $k = 2$, and the 2 subproblems are $(A_1 A_2)$ and $(A_3 A_4)$.

Solving MCM

The cost of the optimal solution $M[1..n]$ is the cost of each of the two subsolutions $M[1..k]$ and $M[k+1..n]$ plus the cost of multiplying the final two matrices :

$$M[1..n] = M[1..k] + M[k+1..n] + d_0 \times d_k \times d_n$$

It must be that k partitions the sequence $A_1 A_2 \cdots A_n$ into subsequences such that $M[1..k]$ and $M[k+1..n]$ are two optimal solutions, i.e. $(A_1 \cdots A_k)(A_{k+1} \cdots A_n)$ is the optimal paranthetisation of $A_1 A_2 \cdots A_n$

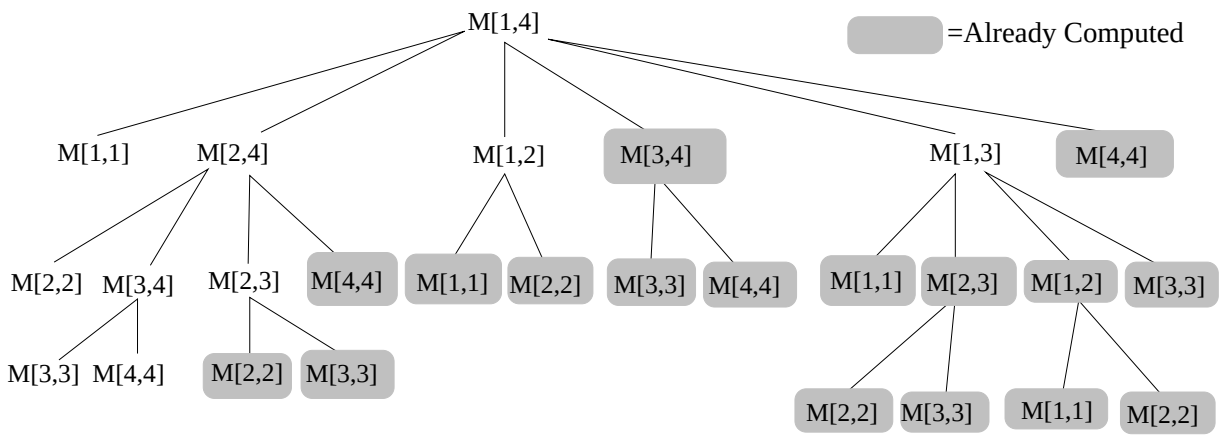
Thus

$$M[1,n] = \min_{1 \leq k < n} \{M[1,k] + M[k+1,n] + d_0 d_k d_n\}.$$

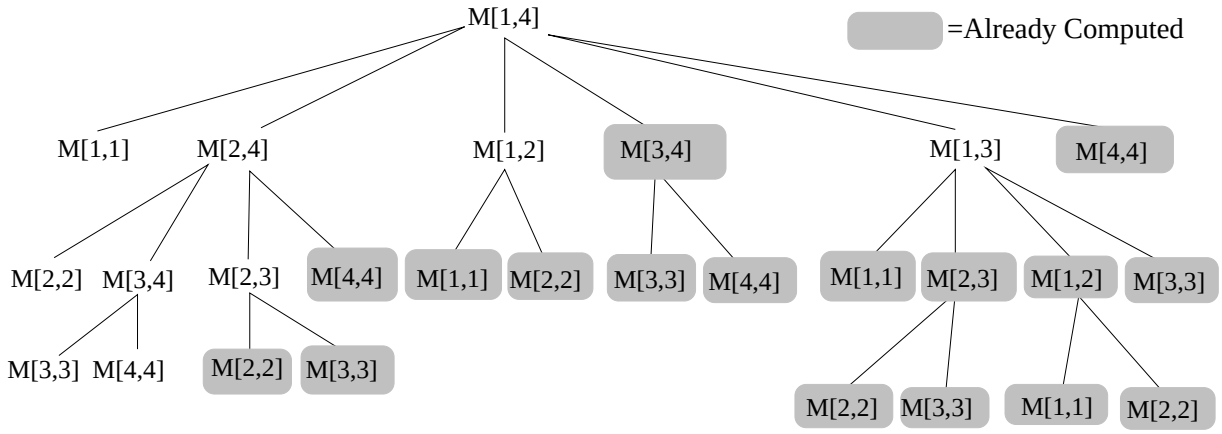
Divide-and-Conquer for MCM

int M(i, j)
  if $i == j$ **then return** (0);
  else
    return $min_{i \leq k < j}(\mathrm{M}(i,k) + \mathrm{M}(k+1,j) + d_{i-1} d_k d_j)$;

The recursion tree for the MCM problem with 4 matrices :



Divide-and-Conquer for MCM



- Notice that many of the calls are repeated (all the shaded boxes).
- The divide-and-conquer algorithm has the following recurrence

$$T(n) = \sum_{k=1}^{n-1} (T(k) + T(n-k)) + cn$$

with $T(0) = 0$ and $T(1) = 1$.

Analysis of the recurrence for MCM

If $n > 0$,

$$T(n) = 2\sum_{k=1}^{n-1} T(k) + cn$$

Therefore

$$\begin{aligned} T(n) - T(n-1) &= (2\textstyle\sum_{k=1}^{n-1} T(k) + cn) \\ &\quad - (2\textstyle\sum_{k=1}^{n-2} T(k) + c(n-1)) \\ &= 2T(n-1) + c \end{aligned}$$

That is

$$T(n) = 3T(n-1) + c$$

Analysis of the recurrence for MCM

using the iteration method :

$$
\begin{aligned}
T(n) &= 3T(n-1)+c \\
&= 3(3T(n-2)+c)+c \\
&= 9T(n-2)+3c+c \\
&= 9(3T(n-3)+c)+3c+c \\
&= 27T(n-3)+9c+3c+c \\
&= 3^k T(n-k)+c\sum_{l=0}^{k-1} 3^l \\
&= 3^n T(0)+c\sum_{l=0}^{n-1} 3^l \\
&= c3^n + c\frac{3^n-1}{2} \\
&= (c+\tfrac{c}{2})3^n - \tfrac{c}{2} \\
T(n) &\in \Theta(3^n).
\end{aligned}
$$

This recurrence can be solved using the recursion tree method. There are $n-1$ levels, each level i execute $3^i c$ operations, yielding the following summation : $3^0+3^1+3^2+\cdots+3^{n-1}$, the dominant term in this expression is $3^{n-1} \in \Theta(3^n)$

Optimal Substructure

MCM satisfies the requirement that optimal solutions combine solutions of two subproblems that are also optimal

The parenthesization of $A_1 \cdots A_k$ and $A_{k+1} \cdots A_n$ must be optimal

Otherwise there exist another parenthetisation P' of one of the two subproblems, for example $A_1 \cdots A_k$, such that $M'[1..k] < M[1..k]$ in which case $M[1..n] = M[1..k] + M[k+1..n] + d_0 \times d_k \times d_n$ is not an optimal solution

Dynamic programming solution

MCM is a candidate for DP :

1. The recursive algorithm solves some subproblems more than one time
2. It satisfies the optimal substructure condition : an optimal solution to $(A_1 \cdots A_k)(A_{k+1} \cdots A_n)$ is based on parenthesizations of $A_1 \cdots A_k$ and $A_{k+1} \cdots A_n$ that are optimal as well.

DP solution for MCM

The DP algorithm for MCM is as follow

- Compute the number of scalar multiplications for all sequences of length 1 (base case of the D&C), i.e. 0
- Compute the number of scalar multiplications for all sequences of length 2, here the solution to AB is sum of two sequences, A and B of length $1 + a \times b \times c$
- Compute the number of scalar multiplications for all sequences of length 3. A sequence of length 3 has 2 subsequences, one of length 1 and one of length 2, the solution to each of these two subsequences is already in the DP table
- Compute the number of scalar multiplications for all sequences of length 4
- until n

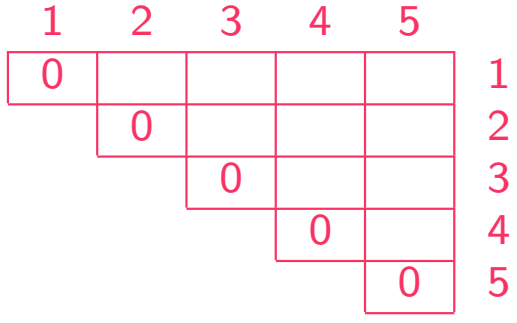
A dynamic programming solution to MCM

int M(i, j)
  if $i == j$ **then return** (0);
  else
    return $min_{i \leq k < j}(\mathrm{M}(i,k) + \mathrm{M}(k+1,j) + d_{i-1} d_k d_j)$;

We define a 2-dimensional array $M[n, n]$ to store solution of each subproblem (a sequence of matrices to multiply) :

- row i refers to the position of the first matrix in the sequence
- column j refers to the position of the last matrix in the sequence.

Given we first solve the base cases, i.e. sequences with only one matrix (where $i = j$), we first fill the entries $M[i, i]$ of the table :



A dynamic programming solution to MCM

```
int M(i, j)
  if i == j then return (0);
  else
    return min_{i≤k<j}(M(i, k) + M(k + 1, j) + d_{i−1} d_k d_j);
```

The next larger problem sizes to solve involve the product of two matrices.

We multiple each 2 consecutive matrices, i.e. $(1, 2), (2, 3), (n-1, n)$, and store the solution into $M[i, i+1]$

Here k can only take one value, $k = i$. According to the recursive algo $M[i, i+1] = M[i, i] + M[i+1, i+1] + d_{i-1} d_k d_j = d_{i-1} d_k d_j$

1	2	3	4	5	
0	$d_0 d_1 d_2$				1
	0	$d_1 d_2 d_3$			2
		0	$d_2 d_3 d_4$		3
			0	$d_3 d_4 d_5$	4
				0	5

A dynamic programming solution to MCM

```
int M(i, j)
  if i == j then return (0);
  else
    return min_{i≤k<j}(M(i, k) + M(k + 1, j) + d_{i−1} d_k d_j);
```

The next larger problem sizes to solve involve the product of three consecutive matrices.

We multiple each 3 consecutive matrices, i.e. $(1,3),(2,4),(n-2,n)$, and store the solution into $M[i, i+2]$

Here k can only take two values, $k = i$ and $k = i+1$. According to the recursive algo $M[i, i+2] = min_{k=i}^{k<j}(M[i,k] + M[k+1,j] + d_{i-1} d_k d_j)$

1	2	3	4	5	
0	$d_0 d_1 d_2$				1
	0	$d_1 d_2 d_3$			2
		0	$d_2 d_3 d_4$		3
			0	$d_3 d_4 d_5$	4
				0	5

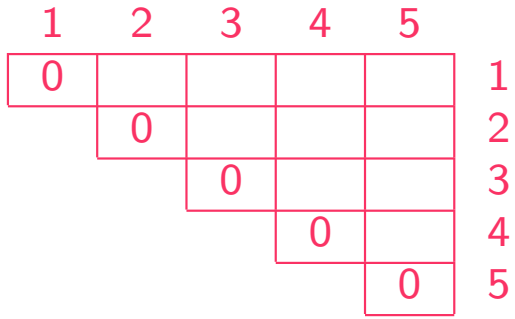
MCM dynamic programming Example

We are given the sequence $(4, 10, 3, 12, 20, 7)$.

The 5 matrices have sizes 4×10, 10×3, 3×12, 12×20, and 20×7.

We need to compute $M[i, j]$, $1 \leq i, j \leq 5$.

We know $M[i, i] = 0$ for all i.



MCM Example

We proceed, working away from the diagonal. We compute the optimal solutions for products of 2 matrices.
Given the sequence $(4, 10, 3, 12, 20, 7)$:

$M[1,2] = 4 \times 10 \times 3 = 120$,
$M[2,3] = 10 \times 3 \times 12 = 360$,
$M[3,4] = 3 \times 12 \times 20 = 720$,
$M[4,5] = 12 \times 20 \times 7 = 1680$

1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

MCM Example (continued)

There are two ways one can split a sequence of 3 matrices in two sub-sequences : (1) (2,3) or (1,2) (3). Given 5 matrices there are 3 sequences of 3 matrices. $(4, 10, 3, 12, 20, 7)$

$$M[1,3] = \min \begin{cases} M[1,2] + M[3,3] + d_0 d_2 d_3 = 120 + 0 + 4 \cdot 3 \cdot 12 = 264 \\ M[1,1] + M[2,3] + d_0 d_1 d_3 = 0 + 360 + 4 \cdot 10 \cdot 12 = 840 \end{cases}$$

$$M[2,4] = \min \begin{cases} M[2,3] + M[4,4] + d_1 d_3 d_4 = 360 + 0 + 10 \cdot 12 \cdot 20 = 2760 \\ M[2,2] + M[3,4] + d_1 d_2 d_4 = 0 + 720 + 10 \cdot 3 \cdot 20 = 1320 \end{cases}$$

$$M[3,5] = \min \begin{cases} M[3,4] + M[5,5] + d_2 d_4 d_5 = 720 + 0 + 3 \cdot 20 \cdot 7 = 1140 \\ M[3,3] + M[4,5] + d_2 d_3 d_5 = 0 + 1680 + 3 \cdot 12 \cdot 7 = 1932 \end{cases}$$

1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

$\Rightarrow$

1	2	3	4	5	
0	120	264			1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

MCM Example (continued again)

A sequence of 4 matrices can be split in 3 different ways (1) (2,4), or (1,2) (3,4), or (1,3) (4). Given 5 matrices, there are 2 sequences of 4 matrices. $(4, 10, 3, 12, 20, 7)$

$$M[1,4] = \min \begin{cases} M[1,3] + M[4,4] + d_0 d_3 d_4 = 264 + 0 + 4 \cdot 12 \cdot 20 = 1224 \\ M[1,2] + M[3,4] + d_0 d_2 d_4 = 120 + 720 + 4 \cdot 3 \cdot 20 = 1080 \\ M[1,1] + M[2,4] + d_0 d_1 d_4 = 0 + 1320 + 4 \cdot 10 \cdot 20 = 2120 \end{cases}$$

$$M[2,5] = \min \begin{cases} M[2,4] + M[5,5] + d_1 d_4 d_5 = 1320 + 0 + 10 \cdot 20 \cdot 7 = 2720 \\ M[2,3] + M[4,5] + d_1 d_3 d_5 = 360 + 1680 + 10 \cdot 12 \cdot 7 = 2880 \\ M[2,2] + M[3,5] + d_1 d_2 d_5 = 0 + 1140 + 10 \cdot 3 \cdot 7 = 1350 \end{cases}$$

1	2	3	4	5	
0	120	264			1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

$\Rightarrow$

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

MCM Example (continued *again*)

Now the product of 5 matrices. $(4, 10, 3, 12, 20, 7)$

$$M[1,5] = \min \begin{cases} M[1,4] + M[5,5] + d_0 d_4 d_5 = 1080 + 0 + 4 \cdot 20 \cdot 7 = 1640 \\ M[1,3] + M[4,5] + d_0 d_3 d_5 = 264 + 1680 + 4 \cdot 12 \cdot 7 = 2016 \\ M[1,2] + M[3,5] + d_0 d_2 d_5 = 120 + 1140 + 4 \cdot 3 \cdot 7 = 1344 \\ M[1,1] + M[2,5] + d_0 d_1 d_5 = 0 + 1350 + 4 \cdot 10 \cdot 7 = 1630 \end{cases}$$

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

$\Rightarrow$

1	2	3	4	5	
0	120	264	1080	1344	1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

MCM Example : Optimal Cost and Solution

The optimal cost is $M[1,5] = 1344$.

What is the optimal parenthesization ?

We didn't keep track of enough information to find that out.

The algorithm can be modified slightly to keep enough information to derive the optimal parenthesization.

Each time the optimal value for $M[i,j]$ is found, store also the value of k that was used.

Then we can just work backwards, partitioning the matrices according to the optimal split.

MCM Example : Optimal Solution

If we did this for the example, we would get

1	2	3	4	5	
0	120/1	264/2	1080/2	1344/2	1
	0	360/2	1320/2	1350/2	2
		0	720/3	1140/4	3
			0	1680/4	4
				0	5

The k value for the solution is 2, so we have $(A_1A_2)(A_3A_4A_5)$. The first half is done.

MCM Example : Optimal Solution

1	2	3	4	5	
0	120/1	264/2	1080/2	1344/2	1
	0	360/2	1320/2	1350/2	2
		0	720/3	1140/4	3
			0	1680/4	4
				0	5

The optimal solution for the second half comes from entry $M[3,5]$.

The value of k here is 4, so now we have $(A_1A_2)((A_3A_4)A_5)$.

Thus the optimal solution is to parenthesize as $(A_1A_2)((A_3A_4)A_5)$.

MCM : Dynamic Programming Algo.

```
int MCM(int *MM, int n) {
  int M[n, n], min;
  for (i = 1; i ≤ n; i++) M[i, i] := 0;
  for (s = 1; s < n; s++)
    for (i = 1; i ≤ n − s; i++)
      min = ∞;
      for (k = i; k < i + s; k++)
        if (M[i, k] + M[k + 1, i + s] + d_{i−1} d_k d_{i+s} < min)
          min = M[i, k] + M[k + 1, i + s] + d_{i−1} d_k d_{i+s};
      M[i, i + s] = min;}
```

This algorithm has 3 nested loops. The summation is $\sum_{s=1}^{n} \sum_{i=1}^{n-1} \sum_{k=i}^{i+s} 1$. The complexity of dynamic programming MCM is $\Theta(n^3)$.

Exercises 6 and 7 on matrix-chain multiplications

6. Given the sequence 2, 3, 5, 2, 4, 3, how many matrices do we have and what is the dimension of each matrix. Using the previous dynamic programming algorithm for matrix-chain multiplication, compute the parenthetization of these matrices that minimize the number of scalar multiplications.
7. Given the sequence 5, 4, 6, 2, 7, how many matrices do we have and what is the dimension of each matrix. Using the previous dynamic programming algorithm for matrix-chain multiplication, compute the parenthetization of these matrices that minimize the number of scalar multiplications.

Solving exercise 6

We compute the optimal solutions for products of 2 matrices.
Given the sequence $(2, 3, 5, 2, 4, 3)$:

$M[1, 2] = 2 \times 3 \times 5 = 30$,
$M[2, 3] = 3 \times 5 \times 2 = 30$,
$M[3, 4] = 5 \times 2 \times 4 = 40$,
$M[4, 5] = 2 \times 4 \times 3 = 24$

1	2	3	4	5	
0	30				1
	0	30			2
		0	40		3
			0	24	4
				0	5

Solving exercise 6

There are two ways one can split a sequence of 3 matrices in two sub-sequences : (1) (2,3) or (1,2) (3). Given 5 matrices there are 3 sequences of 3 matrices. $(2, 3, 5, 2, 4, 3)$

$$M[1,3] = \min \begin{cases} M[1,2] + M[3,3] + d_0 d_2 d_3 = 30 + 0 + 2 \cdot 2 \cdot 2 = 50 \\ M[1,1] + M[2,3] + d_0 d_1 d_3 = 0 + 30 + 2 \cdot 3 \cdot 2 = 42 \end{cases}$$

$$M[2,4] = \min \begin{cases} M[2,3] + M[4,4] + d_1 d_3 d_4 = 30 + 0 + 3 \cdot 2 \cdot 4 = 54 \\ M[2,2] + M[3,4] + d_1 d_2 d_4 = 0 + 40 + 3 \cdot 5 \cdot 4 = 100 \end{cases}$$

$$M[3,5] = \min \begin{cases} M[3,4] + M[5,5] + d_2 d_4 d_5 = 40 + 0 + 5 \cdot 4 \cdot 3 = 100 \\ M[3,3] + M[4,5] + d_2 d_3 d_5 = 0 + 24 + 5 \cdot 2 \cdot 3 = 54 \end{cases}$$

1	2	3	4	5	
0	30/1	42/1			1
	0	30/2	54/3		2
		0	40/3	54/3	3
			0	24/4	4
				0	5

Solving exercise 6

A sequence of 4 matrices can be split in 3 different ways (1) (2,4), or (1,2) (3,4), or (1,3) (4). Given 5 matrices, there are 2 sequences of 4 matrices. $(2, 3, 5, 2, 4, 3)$

$$M[1,4] = \min \begin{cases} M[1,3] + M[4,4] + d_0 d_3 d_4 = 42 + 0 + 2 \cdot 2 \cdot 4 = 58 \\ M[1,2] + M[3,4] + d_0 d_2 d_4 = 30 + 40 + 2 \cdot 5 \cdot 4 = 120 \\ M[1,1] + M[2,4] + d_0 d_1 d_4 = 0 + 54 + 2 \cdot 3 \cdot 4 = 78 \end{cases}$$

$$M[2,5] = \min \begin{cases} M[2,4] + M[5,5] + d_1 d_4 d_5 = 54 + 0 + 3 \cdot 4 \cdot 3 = 90 \\ M[2,3] + M[4,5] + d_1 d_3 d_5 = 30 + 24 + 3 \cdot 2 \cdot 3 = 72 \\ M[2,2] + M[3,5] + d_1 d_2 d_5 = 0 + 54 + 3 \cdot 5 \cdot 3 = 99 \end{cases}$$

1	2	3	4	5	
0	30/1	42/1	58/3		1
	0	30/2	54/3	72/3	2
		0	40/3	54/3	3
			0	24/4	4
				0	5

Solving exercise 6

Now the product of 5 matrices. $(2, 3, 5, 2, 4, 3)$

$$M[1,5] = \min \begin{cases} M[1,4] + M[5,5] + d_0 d_4 d_5 = 58 + 0 + 2 \cdot 4 \cdot 3 = 82 \\ M[1,3] + M[4,5] + d_0 d_3 d_5 = 42 + 24 + 2 \cdot 2 \cdot 3 = 78 \\ M[1,2] + M[3,5] + d_0 d_2 d_5 = 30 + 54 + 2 \cdot 5 \cdot 3 = 114 \\ M[1,1] + M[2,5] + d_0 d_1 d_5 = 0 + 72 + 2 \cdot 3 \cdot 3 = 90 \end{cases}$$

1	2	3	4	5	
0	30/1	42/1	58/3	78/3	1
	0	30/2	54/3	72/3	2
		0	40/3	54/3	3
			0	24/4	4
				0	5

Solving exercise 6

The minimum number of scalar multiplications is stored in $M[1,5] = 78$.

The parenthesization that yields this solution is obtained by working backwards in the dynamic programming table the partitioning of each sequence into two subsequences of matrices.

We start with sequence $A_1A_2A_3A_4A_5$ which, according to the entry M[1,5] of the table, is decomposed into two subsequences between the third and the fourth matrix $(A_1A_2A_3)(A_4A_5)$

Then we move into the entry M[1,3] of the table to find the decomposition of the sequence $A_1A_2A_3$. According to the table this sequence is decomposed between the first and second matrix $(A_1)(A_2A_3)$

The solution is $(A_1)(A_2A_3)(A_4A_5)$.

Proof the solution to exercise 6 is correct

The dimensions of these 3 matrices in the solution $(A_1)(A_2A_3)(A_4A_5)$ are $A_1 = (2, 3), A_2A_3 = (3, 2), A_4A_5 = (2, 3)$.

Thus the number of scalar multiplications to multiply

- $A_1(A_2A_3) = 2 \times 3 \times 2 = 12$
- $(A_1A_2A_3)(A_4A_5) = 2 \times 2 \times 3 = 12$

There are 30 scalar multiplications to multiply $A_2 \times A_3$

There are 24 scalar multiplications to multiply $A_4 \times A_5$

Thus the total number of scalar multiplications using the parenthetization $(A_1)(A_2A_3)(A_4A_5)$ is $12 + 12 + 30 + 24 = 78$, as in table M[1,5]

Longest common subsequence (LCS)

A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, "abc" and "ace" are subsequences of "abcdef".

LCS problem : Given 2 sequences, $X = [x_1 \dots x_m]$ and $Y = [y_1 \dots y_n]$, find a subsequence common to both whose length is longest.

For example, "ADH" is a subsequence of length 3 of the input sequences "ABCDGH" and "AEDFHR".

Brute-force algorithm for LCS

Brute-force algorithm : For every subsequence of X, check whether it is a subsequence of Y. Time : $\Theta(n2^m)$

- $2^m - 1$ subsequences of X to check
- Each subsequence takes $\Theta(n)$ time to check : scan Y for first letter, from there scan for second, and so on.

Optimal substructure

Consider the input sequences be $X = [x_1 \dots x_m]$ and $Y = [y_1 \dots y_n]$ respectively of length m and n.

Let $lcs(X[1..m], Y[1..n])$ be the length of the LCS of X and Y. The length of the LCS is computed recursively as follow :

- if the last character of both sequences match, i.e. if $X[m] == Y[n]$ then $lcs(X[1..m], Y[1..n]) = 1 + lcs(X[1..m-1], Y[1..n-1])$
- if the last character of both sequences do not match, i.e. if $X[m] \neq Y[n]$ then $lcs(X[1..m], Y[1..n]) = max(lcs(X[1..m-1], Y[1..n]), lcs(X[1..m], Y[1..n-1]))$

Optimal substructure

Given sequences "AGGTAB" and "GXTXAYB", last characters match, so length of LCS can be written as :
$lcs("AGGTAB","GXTXAYB") = 1 + lcs("AGGTA","GXTXAY")$

For sequences "ABCDGH" and "AEDFHR", the last characters do not match, so length of LCS is : $lcs("ABCDGH","AEDFHR") = max(lcs("ABCDG","AEDFHR"), lcs("ABCDGH","AEDFH"))$

So the LCS problem has optimal substructure property as the optimal solution of the main problem can be solved using optimal solutions to subproblems.

A recursive algorithm for LCS

```
int lcs(char *X, char *Y, int i, int j )
  if (i == 0||j == 0) return 0;
  if (X[i] == Y[j])
    return 1 + lcs(X, Y, i − 1, j − 1);
  else
    return max(lcs(X, Y, i, j − 1), lcs(X, Y, i − 1, j));
```

Time complexity of this recursive algorithm is $O(2^n)$ in worst case and worst case happens when all characters of X and Y mismatch i.e., length of LCS is 0.

Building a partial call tree for sequences "AXYT" and "AYZX", it can be verified that the recursive algorithm solves the same subproblems several times. Soon you will observe that $lcs("AXY","AYZ")$ is being solved twice.

Top Down DP algorithm for LCS

```
int TD − DP − lcs(char *X, char *Y, int i, int j)
  if (i == 0||j == 0)
    table[i, j] = 0; /* re-calculate each time */
    return table[i, j];
  if (X[i] == Y[j])
    if (table[i, j] == −1)
      table[i, j] = 1 + TD − DP − lcs(X, Y, i − 1, j − 1);
      return table[i, j]
    else return table[i, j];
  if (X[i] ≠ Y[j])
    if (table[i, j] == −1)
      table[i, j] = max(TD − DP − lcs(X, Y, i, j − 1),
                        TD − DP − lcs(X, Y, i − 1, j));
      return table[i, j]
    else return table[i, j];
```

Bottom up DP for LCS

LCS-Length(X,Y,m,n)
let $b[1..m, 1..n]$ and $c[0..m, 0..n]$
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 $X[i] == Y[j]$
      $c[i, j] = c[i - 1, j - 1] + 1$
      $b[i, j] =$"$\nwarrow$"
    else if $c[i - 1, j] \geq c[i, j - 1]$
      $c[i, j] = c[i - 1, j]$
      $b[i, j] =$"$\uparrow$"
    else $c[i, j] = c[i, j - 1]$
      $b[i, j] =$"$\leftarrow$"
return c and b

Computational complexity is $\Theta(mn)$

Example

LCS-Length(X,Y,m,n)
let $b[1..m, 1..n]$ and $c[0..m, 0..n]$
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 $X[i] == Y[j]$
      $c[i, j] = c[i - 1, j - 1] + 1$
      $b[i, j] =$"$\nwarrow$"
    else if $c[i - 1, j] \geq c[i, j - 1]$
      $c[i, j] = c[i - 1, j]$
      $b[i, j] =$"$\uparrow$"
    else $c[i, j] = c[i, j - 1]$
      $b[i, j] =$"$\leftarrow$"
return c and b

	j	0	1	2	3	4	5	6
i		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

$X = [ABCBDAB]$
$Y = [BDCABA]$
$m = 7$; $n = 6$
LCS is $[BCBA]$

Printing the LCS

```
Print-LCS(b,X,i,j)
if i == 0 or j == 0 return
if b[i,j] == "↖"
   Print-LCS(b,X,i-1,j-1)
   print x_i
else if b[i,j] == "↑"
   Print-LCS(b,X,i-1,j)
else if b[i,j] == "←"
   Print-LCS(b,X,i,j-1)
```

Computational complexity is $\Theta(m+n)$

Printing the LCS

- Initial call is Print-LCS(b,X,m,n)
- $b[i,j]$ points to table entry whose subproblem we used in solving LCS of X_i and Y_j.
- When $b[i,j] = \nwarrow$, we have extended LCS by one character. So longest common subsequence = entries with $\nwarrow$ in them

Exercises : longest common subsequences

8. Find the LCS for $X = [ABAZDC]$ and $Y = [BACBAD]$
   Solution :

	B	A	C	B	A	D
A	0	1	1	1	1	1
B	1	1	1	2	2	2
A	1	2	2	2	3	3
Z	1	2	2	2	3	3
D	1	2	2	2	3	4
C	1	2	3	3	3	4

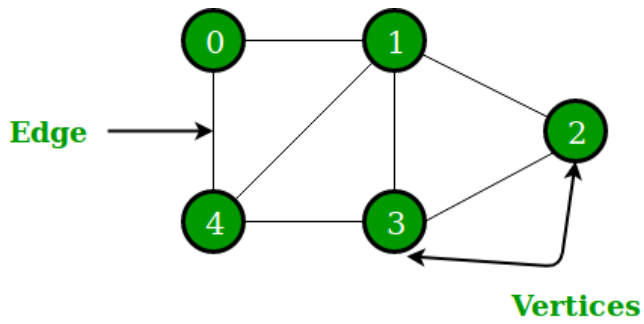
9. Prove that the LCS for the input sequences $X = [ABCDGH]$ and $Y = [AEDFHR]$ is "ADH" of length 3.

Graphs

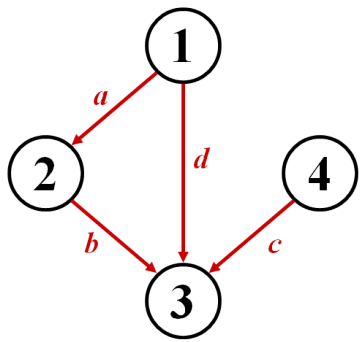
A graph $G = (V, E)$ is a set V of vertices (nodes) and a set E of edges connecting vertices

Types of graphs :

- *Undirected graph* : Edge $(u, v) =$ edge (v, u)
- Edges (0,4), (2,3), (1,2) is a subset of edges of this graph



- *Directed graph* : Edge (u,v) goes from vertex u to vertex v, notated $u \rightarrow v$
- *Weighted graph* : associates weights with the edges. E.g., a road map : weighted edges are distances

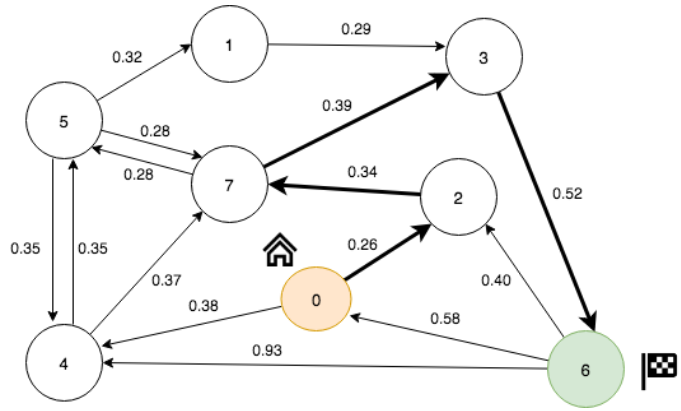


Shortest path problem

A path between a pair of vertices a, b in a weighted graph is a sequence of consecutive edges that connect a, b

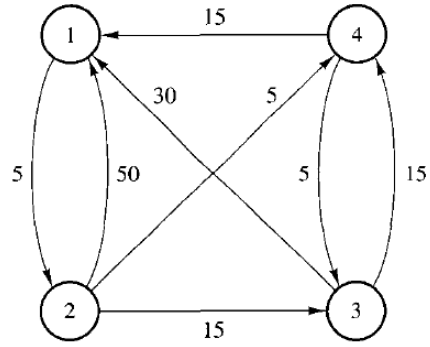
In the graph below, the sequence (1,3,6,2,7) s a path connecting vertices 1 and 7

The shortest path problem consists to find the path of minimum weight between a pair of vertices



In this graph there are several paths between vertices 1,7 beside the one we found above : (1,3,6,0,2,7), (1,3,6,4,7), (1,3,6,0,4,5,7), etc.

The All-Pairs Shortest-Path Problem



$$L = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 15 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

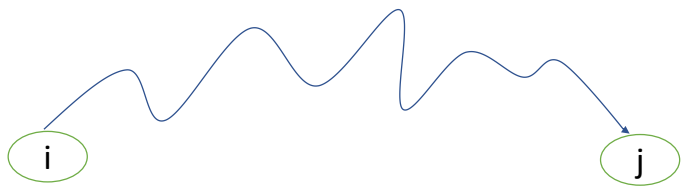
Problem definition :

- $G = \{V, E\}$ is a connected "directed" graph, V is the set of nodes ($|V| = n$) and E is the set of edges.
- Each edge has an associated nonnegative length. A distance matrix $L[i, j]$ gives the length of each edge :
  - $L[i, i] = 0$, $L[i, j] \geq 0$ if $i \neq j$,
  - $L[i, j] = \infty$ if the edge (i, j) does not exist.
- *Find the shortest path distance between each pair of vertices in the graph*

Reasoning to find a D-&-C algorithm

The n nodes in the graph are numbered consecutively from 1 to n

The shortest path between a pair of nodes i, j passes through some intermediary nodes in the set $\{1, 2, \ldots, n\} \setminus \{i, j\}$.

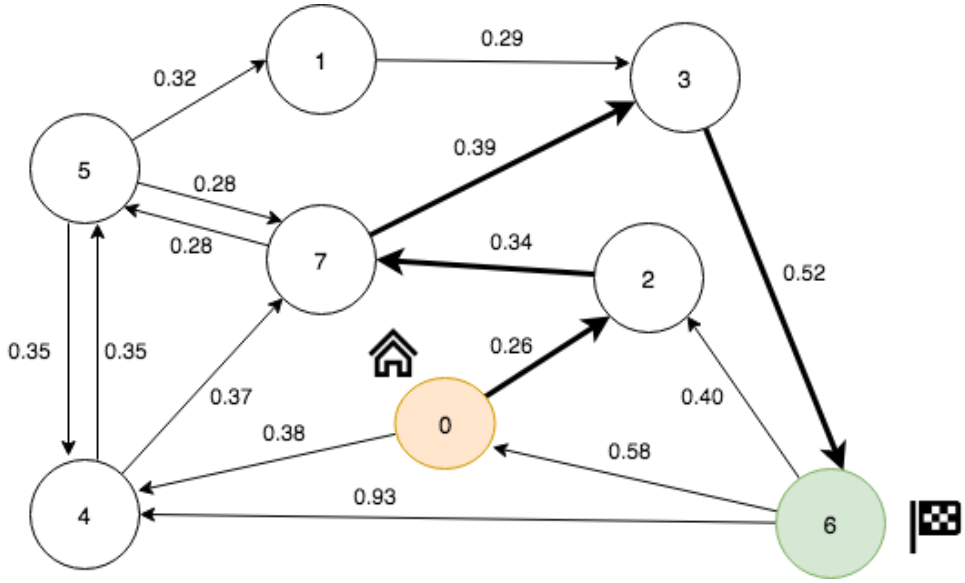


One way to reduce the problem size for i, j is to consider a subset of the nodes $\{1, 2, \ldots, n\} \setminus \{i, j\}$ from which the shortest path can be built, such as for example $\{1, 2, \ldots, n\} \setminus \{i, j, n\}$

The problem is now to find the shortest distance between i, j going through some intermediary nodes in the set $\{1, 2, \ldots, n-1\}$

Example

What is the shortest path between vertices 1, 7 if we are only allowed to use vertex 3 as set of intermediate vertices



Solution : The length of the shortest path $\{3\} \setminus \{0, 1, 2, 4, 5, 6, 7\} = \infty$

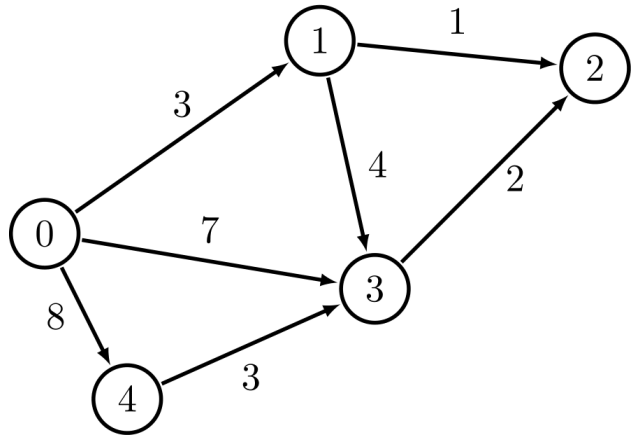
Example : recursive solution

As always with recursive solutions, with start with the original problem instance and seek a way to reduce its size until we find a base case

Assume we want to find the shortest path between nodes 0 and 2 using intermediary nodes 1,3,4.

There are several paths to consider

One way to make the problem easier to solve consists to reduce the size of the intermediary set of vertices, for example considering only vertices 1 and 3

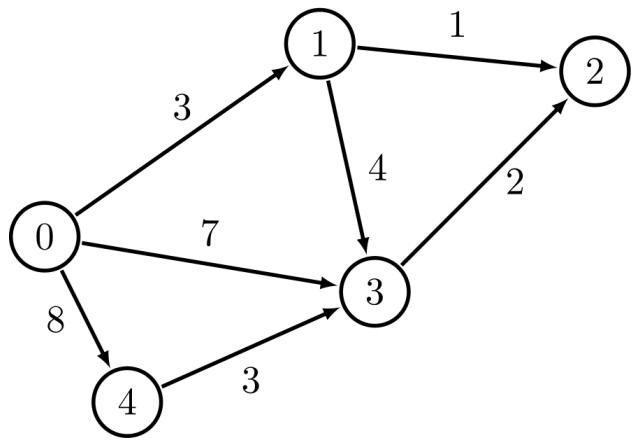


Recursive solution : base case

If recursively we keep removing vertices from the intermediary set, this set eventually is empty, this is the base case

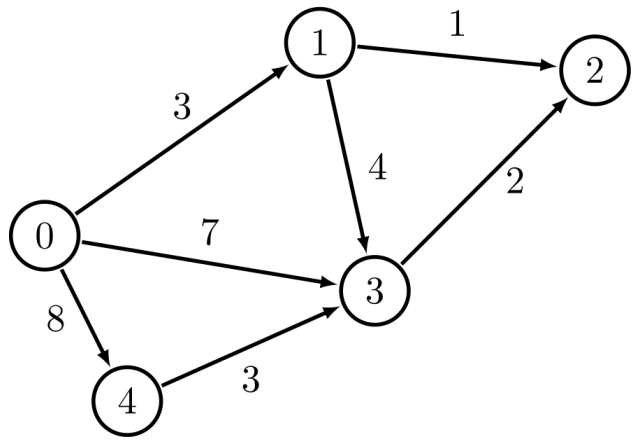
If we are not allowed to use any intermediary node to construct the shortest path between a pair of nodes a, b, then this path is equal to the length of the direct edge connecting a,b.

In the graph below, the shortest path between $(0, 1) = 3$, $(0, 2) = \infty$, $(0, 3) = 7$, and $(0, 4) = 8$



Intermediary set equal $\{1\}$

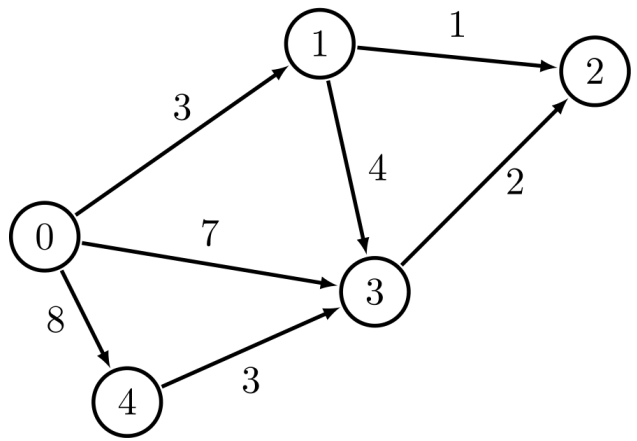
In the graph below, the shortest path between $(0, 1) = 3$, $(0, 2) = 4$, $(0, 3) = 7$, and $(0, 4) = 8$



Example : recursive solution

Let denote the shortest path problem between nodes 0 and 2 using all the other nodes as intermediary nodes as $Path(0, 2, n)$.

Denote shortest path problem between nodes 0 and 2 using only nodes 1 and 3 as intermediary nodes as $Path(0, 2, n-1)$.



A D-&-C algo for all-pairs shortest path

As nodes are numbered consecutively, a sub-problem for finding the shortest path between i, j is *finding the shortest path between* i, j *using only nodes in the set* $\{1, 2, \ldots, k\}$ *where* $k \leq n$.

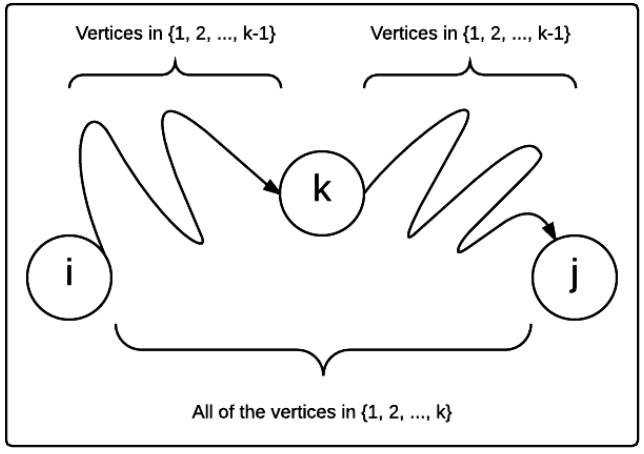
Let denote the function that returns the length of the shortest path between nodes i, j using only intermediary nodes from the set $\{1, 2, \ldots, k\}$ as $Path(i, j, k)$

Base case : the value returns by $Path(i, j, 0)$, the length of the shortest path between i, j with no intermediate node, is the cost of the direct edge from i to j in the graph, i.e. $Path(i, j, 0) = L[i, j]$

Shortest path D&C

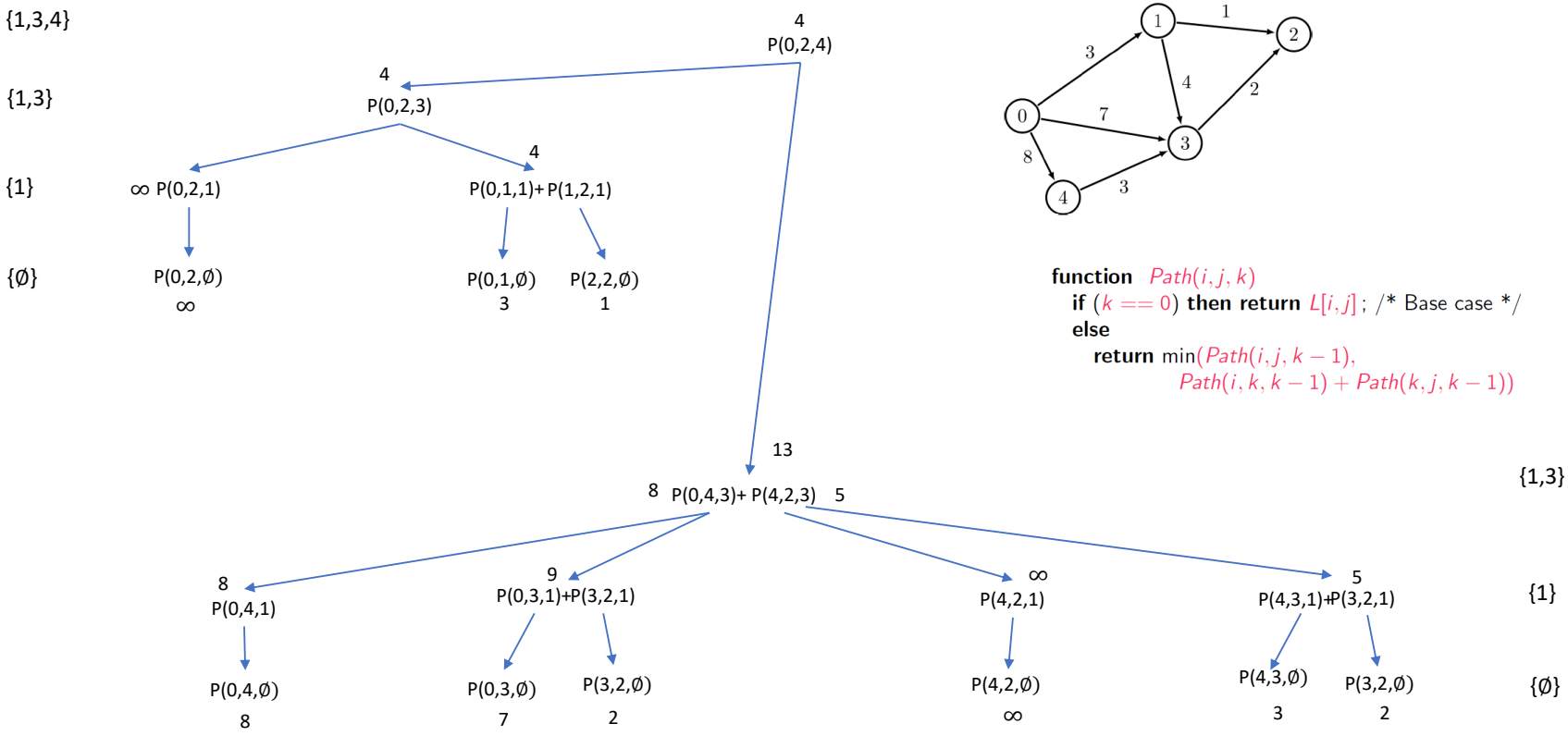
There are two possible ways one can compute the value of $Path(i, j, k)$

1. one may decide not to include node k in the computation of the shortest path between i, j (only uses nodes in the set $\{1, 2, \ldots, k-1\}$, in which case $Path(i, j, k) = Path(i, j, k-1)$
2. one may decide to select node k, from i to k then from k to j using only intermediate nodes $\{1, 2, \ldots, k-1\}$, then $Path(i, j, k) = Path(i, k, k-1) + Path(k, j, k-1)$



The length of the path from i to k to j is the concatenation of the shortest path from i to k and the shortest path from k to j, each one only using intermediate vertices in $\{1, 2, \ldots, k-1\}$.

Call tree for D-&-C shortest path between nodes 0 and 2

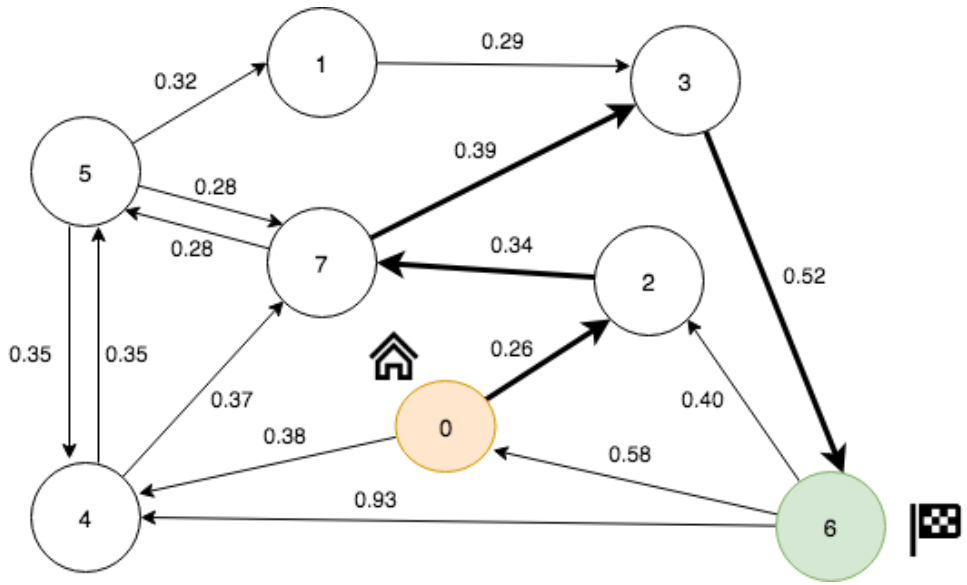



```
function Path(i, j, k)
  if (k == 0) then return L[i, j] ; /* Base case */
  else
    return min(Path(i, j, k − 1),
               Path(i, k, k − 1) + Path(k, j, k − 1))
```

Recursive computation of $Path(i, j, k)$

There are two possible ways one can compute the value of $Path(1, 7, k = 6)$

1. one may decide not to include node $k = 6$ in the computation of the shortest path between $1, 7$ (only uses nodes in the set $\{0, 1, 2, \ldots, 5\}$, in which case $Path(1, 7, 6) = Path(1, 7, 5)$
2. one may decide to select node 6, from 1 to 6 then from 6 to 7 using only intermediate nodes $\{0, 1, 2, \ldots, 5\}$, then $Path(1, 7, 6) = Path(1, 6, 5) + Path(6, 7, 5)$



Recursive computation of $Path(i, j, k)$

Then we can define $Path(i, j, k)$ as a recursive function to compute the shortest path between nodes i and j

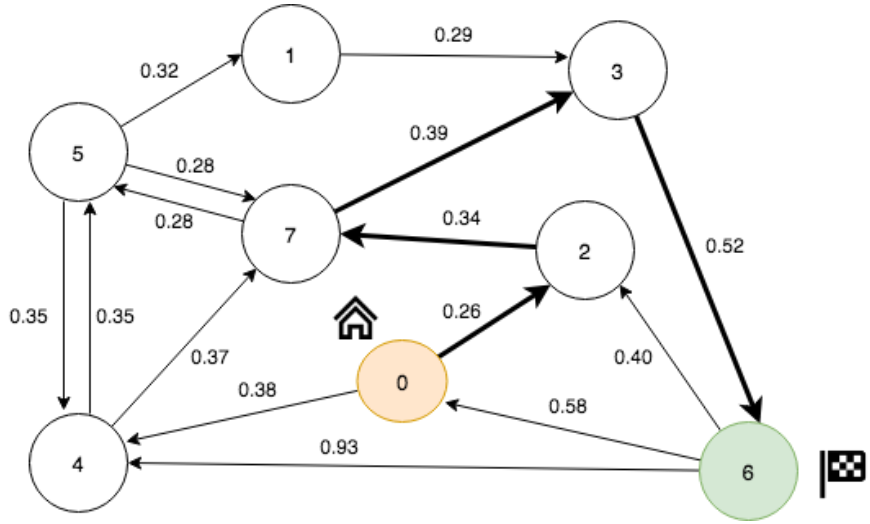
```
function Path(i, j, k)
  if (k == 0) then return L[i, j] ; /* Base case */
  else
    return min(Path(i, j, k − 1),
               Path(i, k, k − 1) + Path(k, j, k − 1))
```

The initial call is $Path(i, j, n)$. This function is run for each pair of node i and j

The above recursive algorithm is used to design the Floyd-Warshall dynamic programming algorithm that solves the all pairs of shortest paths problem.

Two levels recursive calls

function $Path(1, 7, 6)$
    return min$(Path(1, 7, 5),$
        $Path(1, 6, 5) + Path(6, 7, 5))$



function $Path(1, 6, 5)$
    return min$(Path(1, 6, 4),$
        $Path(1, 4, 4) + Path(4, 7, 4))$

function $Path(6, 7, 5)$
    return min$(Path(6, 7, 4),$
        $Path(6, 4, 4) + Path(4, 7, 4))$

Floyd-Warshall Algorithm

It is a bottom up dynamic programming algorithm. It starts by computing the shortest path for all pairs of nodes for $k = 0$. Then it considers $k = 1$, $k = 2$, until $k = n$.

A matrix D gives the length of the shortest path between each pair of nodes

$D_0 = L$, the direct distances between nodes.

Algorithm Floyd$(L[n, n])$
  $D = L$
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        $D[i, j] = min(D[i, j], D[i, k] + D[k, j])$
  return D

After each iteration k, D_k contains the length of the shortest paths that only use nodes in $\{1, 2, \ldots, k\}$ as intermediate nodes.

Floyd Algorithm

Algorithm Floyd$(L[n,n])$
  $D = L$
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        $D[i,j] = min(D[i,j], D[i,k] + D[k,j])$
  return D

At iteration k, the algo checks each pair of nodes (i,j) whether or not there exists a path from i to j passing through node k that is better than the present optimal path passing only through nodes in $\{1, 2, \ldots, k-1\}$.

$$D[i,j] = min(D[i,j], D[i,k] + D[k,j])$$

Execution of Floyd's algorithm

Algorithm Floyd$(L[n,n])$
  $D = L$
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        $D[i,j] = min(D[i,j], D[i,k] + D[k,j])$
  return D

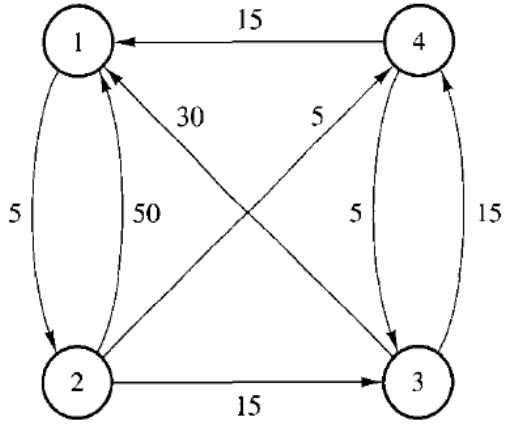
Base case $D_0 = L$, the smallest problem instances

$$D_0 = L = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

Execution of Floyd's algorithm

Algorithm Floyd$(L[n, n])$
  $D = L$
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        $D[i, j] = min(D[i, j], D[i, k] + D[k, j])$
  return D

For $k = 1$, compute the shortest path between each pair of nodes (i, j) when the path is allowed to pass through node 1.



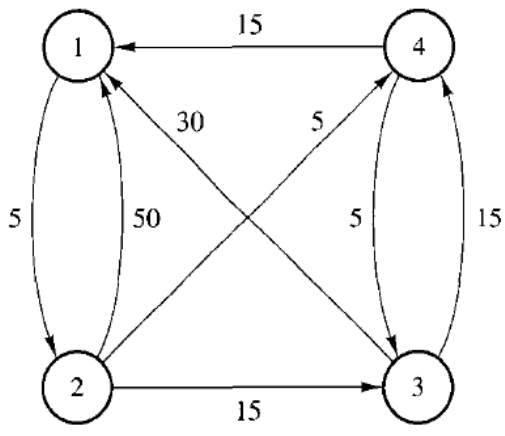
$$\begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Execution of Floyd's algorithm

Algorithm Floyd$(L[n, n])$
  $D = L$
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        $D[i, j] = min(D[i, j], D[i, k] + D[k, j])$
  return D

For $k = 2$, compute the shortest path between each pair of nodes (i, j) when the path is allowed to pass through nodes 1 and 2.



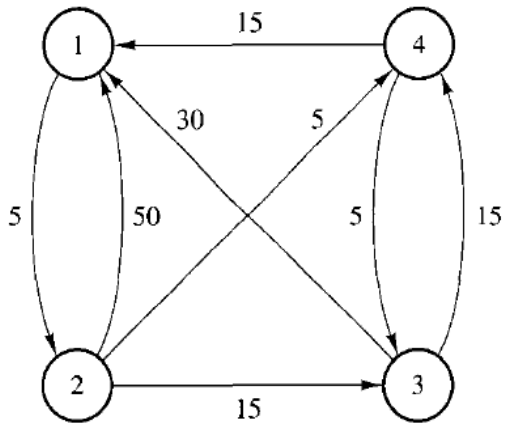
$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Execution of Floyd's algorithm

Algorithm Floyd$(L[n, n])$
  $D = L$
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        $D[i, j] = min(D[i, j], D[i, k] + D[k, j])$
  return D

For $k = 3$, compute the shortest path between each pair of nodes (i, j) when the path is allowed to pass through nodes $\{1, 2, 3\}$.



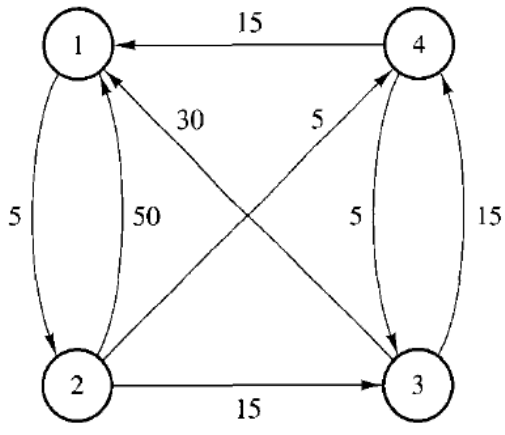
$$D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Execution of Floyd's algorithm

Algorithm Floyd$(L[n, n])$
  $D = L$
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        $D[i, j] = min(D[i, j], D[i, k] + D[k, j])$
  return D

For $k = 4$, solution, compute the shortest path between each pair of nodes (i, j) when the path is allowed to pass through any nodes.



$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 5 & 15 & 10 \\ 20 & 0 & 10 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Computing the shortest paths

We want to know the shortest paths, not just their length.

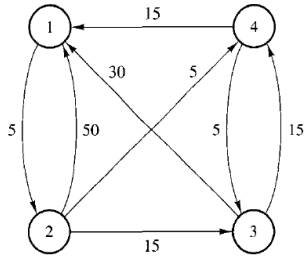
For that we create a new matrix P of size $n \times n$.

Then use the following algorithm in place of the previous one :

Algorithm Floyd$(D[n,n])$
Input : An array D of shortest path lenghts
Output : The shortest path between every pair of nodes
$P[n,n]$ an $n \times n$ array initialized to 0
for $(k = 1; k \leq n; k++)$
  for $(i = 1; i \leq n; i++)$
    for $(j = 1; j \leq n; j++)$
      if $D[i,k] + D[k,j] < D[i,j]$ **then**
        $D[i,j] = D[i,k] + D[k,j]$
        $P[i,j] = k$;

Computing the shortest pathsK $k = 1$

Algorithm Floyd$(D[n, n])$
  Input : An array D of shortest path lenghts
  Output : The shortest path between every pair of nodes
  $P[n, n]$ an $n \times n$ array initialized to 0
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        if $D[i, k] + D[k, j] < D[i, j]$ **then**
          $D[i, j] = D[i, k] + D[k, j]$
          $P[i, j] = k$;



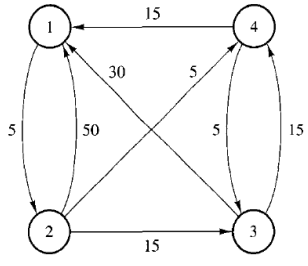
$$\begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

0	0	0	0
0	0	0	0
0	1	0	0
0	1	0	0

Computing the shortest paths : $k = 2$

Algorithm Floyd$(D[n, n])$
  Input : An array D of shortest path lenghts
  Output : The shortest path between every pair of nodes
  $P[n, n]$ an $n \times n$ array initialized to 0
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        if $D[i, k] + D[k, j] < D[i, j]$ **then**
          $D[i, j] = D[i, k] + D[k, j]$
          $P[i, j] = k$;



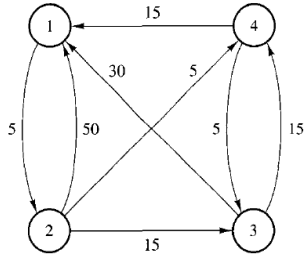
$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

0	0	2	2
0	0	0	0
0	1	0	0
0	1	0	0

Computing the shortest paths : $k = 3$

Algorithm Floyd$(D[n, n])$
  Input : An array D of shortest path lenghts
  Output : The shortest path between every pair of nodes
  $P[n, n]$ an $n \times n$ array initialized to 0
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        if $D[i, k] + D[k, j] < D[i, j]$ **then**
          $D[i, j] = D[i, k] + D[k, j]$
          $P[i, j] = k$;



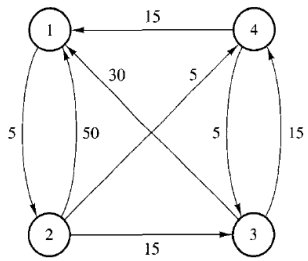
$$D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

0	0	2	2
3	0	0	0
0	1	0	0
0	1	0	0

Computing the shortest paths : $k = 4$

Algorithm Floyd$(D[n, n])$
  Input : An array D of shortest path lenghts
  Output : The shortest path between every pair of nodes
  $P[n, n]$ an $n \times n$ array initialized to 0
  for $(k = 1; k \leq n; k++)$
    for $(i = 1; i \leq n; i++)$
      for $(j = 1; j \leq n; j++)$
        if $D[i, k] + D[k, j] < D[i, j]$ **then**
          $D[i, j] = D[i, k] + D[k, j]$
          $P[i, j] = k$;



$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 5 & 15 & 10 \\ 20 & 0 & 10 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

0	0	4	2
4	0	4	0
0	1	0	0
0	1	0	0

Computing the shortest paths

- The matrix P is initialized to 0.
- When the previous algorithm stops, $P[i,j]$ contains the number of the last iteration that caused a change in $D[i,j]$.

$$P = \begin{pmatrix} 0 & 0 & 4 & 2 \\ 4 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

- If $P[i,j] = 0$, then the shortest path between i and j is directly along the edge (i,j).
- If $P[i,j] = k$, the shortest path from i to j goes through k.

Computing the shortest paths

$$P = \begin{pmatrix} 0 & 0 & 4 & 2 \\ 4 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

- Look recursively at $P[i,k]$ and $P[k,j]$ to find other intermediate vertex along the shortest path.
- In the table above, since, $P[1,3] = 4$, the shortest path from 1 to 3 goes through 4. If we look recursively at $P[1,4]$ we find that the path between 1 and 4 goes through 2. Recursively again, if we look at $P[1,2]$ and $P[2,4]$ we find direct edge.
- Similarly if we look recursively to $P[4,3]$ we find a direct edge (because $P[4,3] = 0$). Then the shortest path from 1 to 3 is 1,2,4,3.

Exercises : All pairs shortest paths

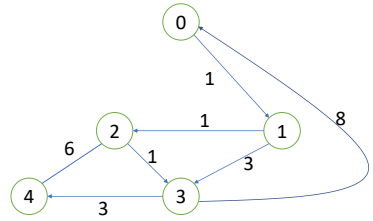
10. Compute the all pairs of shortest paths for the following oriented graph.

$$L = \begin{pmatrix} 0 & 5 & 10 & 3 \\ \infty & 0 & 1 & 4 \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 2 & 0 \end{pmatrix}$$

11. Compute the all pairs of shortest paths for the following oriented graph.

$$L = \begin{pmatrix} 0 & 3 & 8 & \infty & 4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & 5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D_0 = L = \begin{pmatrix} 0 & 1 & \infty & \infty & \infty \\ \infty & 0 & 1 & 3 & \infty \\ \infty & \infty & 0 & 1 & 6 \\ 8 & \infty & \infty & 0 & 3 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$



Here the set of intermediate nodes is $\{0\}$

$$D_1 = \begin{pmatrix} 0 & 1 & \infty & \infty & \infty \\ \infty & 0 & 1 & 3 & \infty \\ \infty & \infty & 0 & 1 & 6 \\ 8 & 9 & \infty & 0 & 3 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

$D_1[0,0] = 0$;

$D_1[0,1] = \min D_0[0,1], D_0[0,0] + D_0[0,1] = 1$
$D_1[0,2] = \min D_0[0,2], D_0[0,0] + D_0[0,2] = \infty$
$D_1[0,3] = \min D_0[0,3], D_0[0,0] + D_0[0,3] = \infty$
$D_1[0,4] = \min D_0[0,4], D_0[0,0] + D_0[0,4] = \infty$

$D_1[1,0] = \min D_0[1,0], D_0[1,0] + D_0[0,0] = \infty$;
$D_1[1,1] = 0$
$D_1[1,2] = \min D_0[1,2], D_0[1,0] + D_0[0,2] = 1$
$D_1[1,3] = \min D_0[1,3], D_0[1,0] + D_0[0,3] = 3$
$D_1[1,4] = \min D_0[1,4], D_0[1,0] + D_0[0,4] = \infty$

$D_1[2,0] = \min D_0[2,0], D_0[2,0] + D_0[0,0] = \infty$;

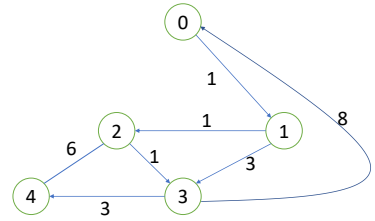
$D_1[2,1] = \min D_0[2,1], D_0[2,0] + D_0[0,1] = \infty$
$D_1[2,2] = 0$
$D_1[2,3] = \min D_0[2,3], D_0[2,0] + D_0[0,3] = 1$
$D_1[2,4] = \min D_0[4,4], D_0[2,0] + D_0[0,4] = 6$

$D_1[3,0] = \min D_0[3,0], D_0[3,0] + D_0[0,0] = 8$;
$D_1[3,1] = \min D_0[3,1], D_0[3,0] + D_0[0,1] = 9$;
$D_1[3,2] = \min D_0[3,2], D_0[3,0] + D_0[0,2] = \infty$
$D_1[3,3] = 0$
$D_1[3,4] = \min D_0[3,4], D_0[3,0] + D_0[0,4] = 3$

$D_1[4,0] = \min D_0[4,0], D_0[4,0] + D_0[0,0] = \infty$;

$D_1[4,1] = \min D_0[4,1], D_0[4,0] + D_0[0,1] = \infty$
$D_1[4,2] = \min D_0[4,3], D_0[4,0] + D_0[0,2] = \infty$
$D_1[4,3] = \min D_0[4,3], D_0[4,0] + D_0[0,3] = \infty$
$D_1[4,4] = 0$

$$D_1 = \begin{pmatrix} 0 & 1 & \infty & \infty & \infty \\ \infty & 0 & 1 & 3 & \infty \\ \infty & \infty & 0 & 1 & 6 \\ 8 & 9 & \infty & 0 & 3 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$



Here the set of intermediate nodes is $\{0, 1\}$

$$D_2 = \begin{pmatrix} 0 & 1 & 2 & 4 & \infty \\ \infty & 0 & 1 & 3 & \infty \\ \infty & \infty & 0 & 1 & 6 \\ 8 & 9 & 10 & 0 & 3 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

$D_1[0,0] = 0$;

$D_1[0,1] = \min D_1[0,1], D_1[0,1] + D_1[1,1] = 1$
$D_1[0,2] = \min D_1[0,2], D_1[0,1] + D_1[1,2] = 2$
$D_1[0,3] = \min D_1[0,3], D_1[0,1] + D_1[1,3] = 4$
$D_1[0,4] = \min D_1[0,4], D_1[0,1] + D_1[1,4] = \infty$

$D_1[1,0] = \min D_1[1,0], D_1[1,1] + D_1[1,0] = \infty$;
$D_1[1,1] = 0$
$D_1[1,2] = \min D_1[1,2], D_1[1,1] + D_1[1,2] = 1$
$D_1[1,3] = \min D_1[1,3], D_1[1,1] + D_1[1,3] = 3$
$D_1[1,4] = \min D_1[1,4], D_1[1,1] + D_1[1,4] = \infty$

$D_1[2,0] = \min D_1[2,0], D_1[2,1] + D_1[1,0] = \infty$;

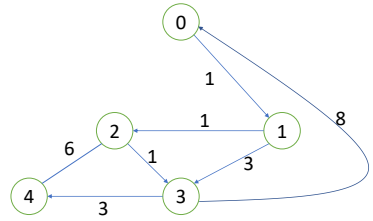
$D_1[2,1] = \min D_1[2,1], D_1[2,1] + D_1[1,1] = \infty$
$D_1[2,2] = 0$
$D_1[2,3] = \min D_1[2,3], D_1[2,1] + D_1[1,3] = 1$
$D_1[2,4] = \min D_1[2,4], D_1[2,1] + D_1[1,4] = 6$

$D_1[3,0] = \min D_1[3,0], D_1[3,1] + D_1[1,0] = 8$;
$D_1[3,1] = \min D_1[3,1], D_1[3,1] + D_1[1,1] = 9$;
$D_1[3,2] = \min D_1[3,2], D_1[3,1] + D_1[1,2] = 10$
$D_1[3,3] = 0$
$D_1[3,4] = \min D_1[3,4], D_1[3,1] + D_1[1,4] = 3$

$D_1[4,0] = \min D_1[4,0], D_1[4,1] + D_1[1,0] = \infty$;

$D_1[4,1] = \min D_1[4,1], D_1[4,1] + D_1[1,1] = \infty$
$D_1[4,2] = \min D_1[4,3], D_1[4,1] + D_1[1,2] = \infty$
$D_1[4,3] = \min D_1[4,3], D_1[4,1] + D_1[1,3] = \infty$
$D_1[4,4] = 0$

$$D_2 = \begin{pmatrix} 0 & 1 & 2 & 4 & \infty \\ \infty & 0 & 1 & 3 & \infty \\ \infty & \infty & 0 & 1 & 6 \\ 8 & 9 & 10 & 0 & 3 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$



Here the set of intermediate nodes is $\{0, 1, 2\}$

$$D_3 = \begin{pmatrix} 0 & 1 & 2 & 3 & 8 \\ \infty & 0 & 1 & 2 & 7 \\ \infty & \infty & 0 & 1 & 6 \\ 8 & 9 & 10 & 0 & 3 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

$D_1[0,0] = 0$;

$D_1[0,1] = \min D_2[0,1], D_2[0,2] + D_2[2,1] = 1$
$D_1[0,2] = \min D_2[0,2], D_2[0,2] + D_2[2,2] = 2$
$D_1[0,3] = \min D_2[0,3], D_2[0,2] + D_2[2,3] = 3$
$D_1[0,4] = \min D_2[0,4], D_2[0,2] + D_2[2,4] = 8$

$D_1[1,0] = \min D_2[1,0], D_2[1,2] + D_2[2,0] = \infty$;
$D_1[1,1] = 0$
$D_1[1,2] = \min D_2[1,2], D_2[1,2] + D_2[2,2] = 1$
$D_1[1,3] = \min D_2[1,3], D_2[1,2] + D_2[2,3] = 2$
$D_1[1,4] = \min D_2[1,4], D_2[1,2] + D_2[2,4] = 7$

$D_1[2,0] = \min D_2[2,0], D_2[2,2] + D_2[2,0] = \infty$;

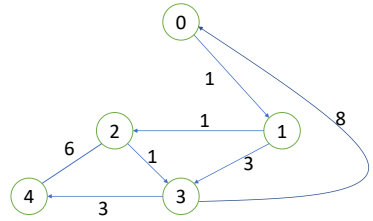
$D_1[2,1] = \min D_2[2,1], D_2[2,2] + D_2[2,1] = \infty$
$D_1[2,2] = 0$
$D_1[2,3] = \min D_2[2,3], D_2[2,2] + D_2[2,3] = 1$
$D_1[2,4] = \min D_2[4,4], D_2[2,2] + D_2[2,4] = 6$

$D_1[3,0] = \min D_2[3,0], D_2[3,2] + D_2[2,0] = 8$;
$D_1[3,1] = \min D_2[3,1], D_2[3,2] + D_2[2,1] = 9$;
$D_1[3,2] = \min D_2[3,2], D_2[3,2] + D_2[2,2] = 10$
$D_1[3,3] = 0$
$D_1[3,4] = \min D_2[3,4], D_2[3,2] + D_2[2,4] = 3$

$D_1[4,0] = \min D_2[4,0], D_2[4,2] + D_2[2,0] = \infty$;

$D_1[4,1] = \min D_2[4,1], D_2[4,2] + D_2[2,1] = \infty$
$D_1[4,2] = \min D_2[4,3], D_2[4,2] + D_2[2,2] = \infty$
$D_1[4,3] = \min D_2[4,3], D_2[4,2] + D_2[2,3] = \infty$
$D_1[4,4] = 0$

$$D_3 = \begin{pmatrix} 0 & 1 & 2 & 3 & 8 \\ \infty & 0 & 1 & 2 & 7 \\ \infty & \infty & 0 & 1 & 6 \\ 8 & 9 & 10 & 0 & 3 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$



Here the set of intermediate nodes is $\{0, 1, 2, 3\}$

$$D_4 = \begin{pmatrix} 0 & 1 & 2 & 3 & 6 \\ 10 & 0 & 1 & 2 & 5 \\ 9 & 10 & 0 & 1 & 4 \\ 8 & 9 & 10 & 0 & 3 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

$D_1[0,0] = 0$;

$D_1[0,1] = \min D_3[0,1], D_3[0,3] + D_3[3,1] = 1$
$D_1[0,2] = \min D_3[0,2], D_3[0,3] + D_3[3,2] = 2$
$D_1[0,3] = \min D_3[0,3], D_3[0,3] + D_3[3,3] = 3$
$D_1[0,4] = \min D_3[0,4], D_3[0,3] + D_3[3,4] = 6$

$D_1[1,0] = \min D_3[1,0], D_3[1,3] + D_3[3,0] = 10$;
$D_1[1,1] = 0$
$D_1[1,2] = \min D_3[1,2], D_3[1,3] + D_3[3,2] = 1$
$D_1[1,3] = \min D_3[1,3], D_3[1,3] + D_3[3,3] = 2$
$D_1[1,4] = \min D_3[1,4], D_3[1,3] + D_3[3,4] = 5$

$D_1[2,0] = \min D_3[2,0], D_3[2,3] + D_3[3,0] = 9$;

$D_1[2,1] = \min D_3[2,1], D_3[2,3] + D_3[3,1] = 10$
$D_1[2,2] = 0$
$D_1[2,3] = \min D_3[2,3], D_3[2,3] + D_3[3,3] = 1$
$D_1[2,4] = \min D_3[4,4], D_3[2,3] + D_3[3,4] = 4$

$D_1[3,0] = \min D_3[3,0], D_3[3,3] + D_3[3,0] = 8$;
$D_1[3,1] = \min D_3[3,1], D_3[3,3] + D_3[3,1] = 9$;
$D_1[3,2] = \min D_3[3,2], D_3[3,3] + D_3[3,2] = 10$
$D_1[3,3] = 0$
$D_1[3,4] = \min D_3[3,4], D_3[3,3] + D_3[3,4] = 3$

$D_1[4,0] = \min D_3[4,0], D_3[4,3] + D_3[3,0] = \infty$;

$D_1[4,1] = \min D_3[4,1], D_3[4,3] + D_3[3,1] = \infty$
$D_1[4,2] = \min D_3[4,3], D_3[4,3] + D_3[3,2] = \infty$
$D_1[4,3] = \min D_3[4,3], D_3[4,3] + D_3[3,3] = \infty$
$D_1[4,4] = 0$

Computing the P matrix

$$P = \begin{pmatrix} -1 & -1 & 1 & 2 & 3 \\ 3 & -1 & -1 & 2 & 3 \\ 3 & 3 & 0-1 & -1 & 3 \\ -1 & 0 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

-1 means the shortest path between nodes x and y is the directed edge in the graph. Shortest path between :

- $(0,2) = (0,1)(1,2) = (0,1,2)$
- $(0,3) = (0,2)(2,3)$ where $(0,2) = (0,1)(1,2) = (0,1,2,3)$
- $(0,4) = (0,3)(3,4) = (0,1,2,3,4)$
- $(1,0) = (1,3)(3,0)$ where $(1,3) = (1,2)(2,3) = (1,2,3) = (1,2,3,0)$
- $(1,3) = (1,2)(2,3) = (1,2,3)$
- $(1,4) = (1,3)(3,4) = (1,2,3,4)$
- $(2,0) = (2,3)(3,0) = (2,3,0)$
- $(2,1) = (2,3)(3,1) = (2,3,1)$
- $(2,4) = (2,3)(3,4) = (2,3,4)$
- $(3,1) = (3,0)(0,1) = (3,0,1)$
- $(3,2) = (3,1)(1,2)$ where $(3,1) = (3,0)(0,1) = (3,0,1) = (3,0,1,2)$

Cost of dynamic programs

Calculate how many table or list entries you fill in (sometimes you don't use the entire table, just all entries under the diagonal or something like that).

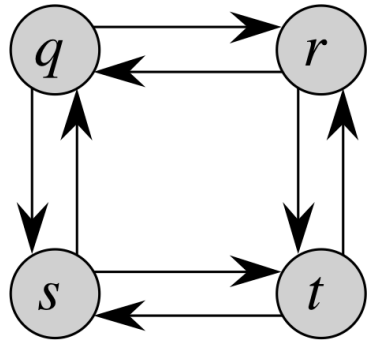
Calculate how much work it takes to compute each entry.

Multiply the two numbers.

Then add the cost of retrieving the answer from the table.

Example : problem fails the optimal substructure test

Unweighted longest simple path : *Find a "simple" path from u to v consisting of the most edges.*
We may assume that the problem of finding an unweighted longest simple path exhibits optimal substructure, i.e. if we decompose the longest simple path from u to v into subpaths $p_1 = u \rightarrow w$ and $p_2 = w \rightarrow v$ then p_1 and p_2 must also be the longest subpaths between $u - w$ and $w - v$. The figure below shows that this is not the case :



Consider the path $q \rightarrow r \rightarrow t$, the longest simple path from q to t. $q \rightarrow r$ is not the longest, rather $q \rightarrow s \rightarrow t \rightarrow r$ is.

Dynamic Programming : Conclusion

As we have seen, dynamic programming is a technique that can be used to find optimal solutions to certain problems.

Dynamic programming works when a problem has the following characteristics :

- **Optimal Substructure :** If an optimal solution contains optimal subsolutions, then a problem exhibits *optimal substructure*.
- **Overlapping subproblems :** When a recursive algorithm would visit the same subproblems repeatedly, then a problem has *overlapping subproblems*.

Dynamic programming takes advantage of these facts to solve problems more efficiently.

It works well only for problems that exhibit *both* properties.

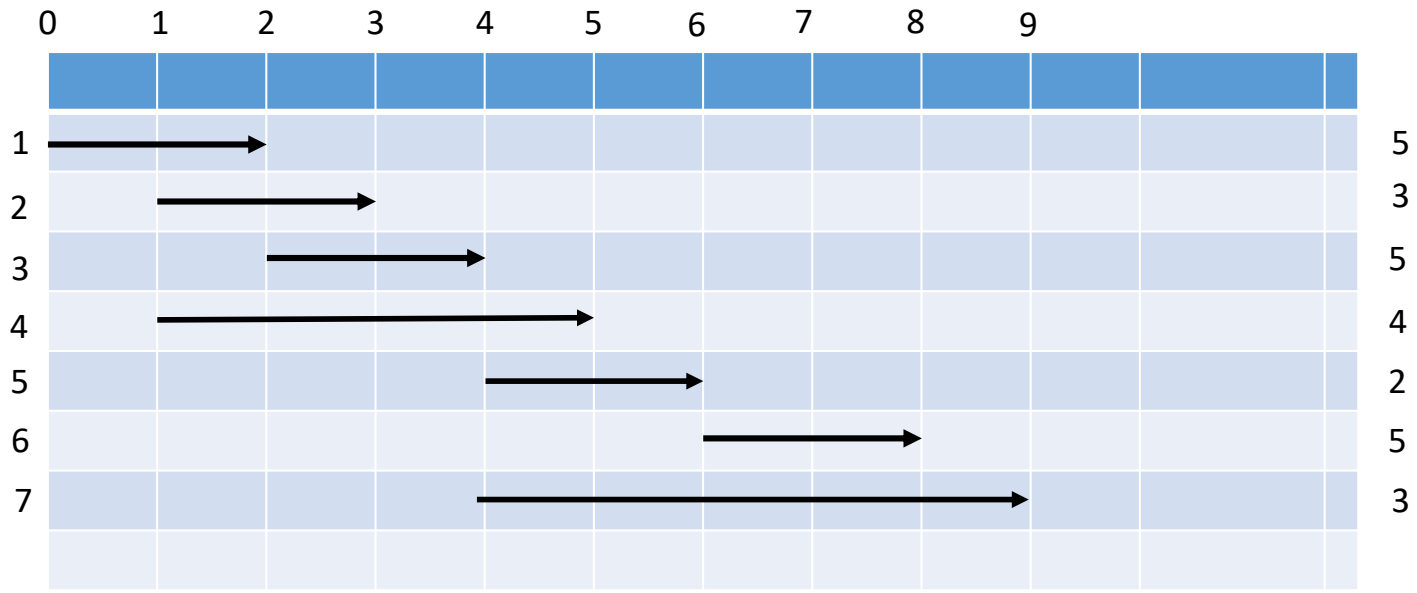
Exercise 12 : write a dynamic programming algorithm

You have a scheduling problem in which the input consists of a list L of requested activities to be executed where each request i consists of the following elements :

- a time interval $[s_i, f_i]$ where s_i is the start time when the activity should begin and f_i is the moment when the activity should finish

- a benefit b_i which is an indicator of the benefit for performing this activity.

Below is a list of 7 requests that need to be scheduled over a period of 9 time units. On the right side of the requests table is the benefit associated to each request. For example, request 3 has to run during the interval $[2, 4]$ for a benefit of $b_3 = 5$.



Given a list L of activity requests, the optimization problem is to schedule the requests in a non-conflicting way such to maximize the total benefit of the activities that are included in the schedule (requests i and j conflict if the time interval $[s_i, f_i]$ intersects with the time interval $[s_j, f_j]$).
Write a dynamic programming algorithm for this scheduling problem. The recursive algorithm is as follow. In the request table above, the list L of requests is ordered in increasing order of the requests finishing time. We use this ordering to decompose the problem into sub-problems. Let

$B_i =$ the maximum benefit that can be achieved using the first i requests in L

So B_i is the value of the optimal solution while considering the subproblem consisting the i first requests. The base case is $B_0 = 0$. We are given a

predecessor function $pred(i)$ which for each request i is the largest index $j < i$ such that request i and j don't conflict. If there is no such index, then $pred(i) = 0$. For example, $pred(3) = 1$ while $pred(2) = 0$.
We can now define a recursive function that we call $HST(L, i)$:

- If the optimal schedule achieving the benefit B_i includes request i, then $B_i = B_{pred(i)} + b_i$
- If the optimal schedule achieving the benefit B_i does not include request i, then $B_i = B_{i-1}$

$HST(L, i)$
  if $(i == 0)$ $B_i = 0$;
  else
    $B_i = max\{HST(L, i-1), HST(L, pred(i)) + b_i\}$

Answer the following questions :

13. Write a bottom up dynamic programming algorithm corresponding to the above HST recursive algorithm

14. For the problem instance described in the request table above, construct the table of all the subproblems considered by your dynamic programming algorithm and fill it with the optimal benefit for each subproblem

Exercise 15 : Longest decreasing subsequence

Write a dynamic programming algorithm to solve the longest decreasing subsequence problem (LDS) : Given a sequence of integers $s_1, \ldots, s_n$ find a subsequence $s_{i_1} > s_{i_2} > \ldots > s_{i_k}$ with $i_1 < \ldots < i_k$ so that k is the largest possible. For example, for the sequence 15, 27, 14, 38, 63, 55, 46, 65, 85, the length of the LDS is 3 and that sequence is 63, 55, 46. In the case of the following sequence 50, 3, 10, 7, 40, 80, the LDS is 50, 10, 7 of length 3.

1. Give a recursive function for computing this problem
2. Using your recurrence, give a dynamic programming algorithm
3. Test your algorithm on this simple sequence [5,0,3,2,1,8]

Dynamic Programming

1. Give a recurrence relation for this problem

   int M(i, j)
   if $j == 1$ **then return** 1 ;
   else
   return 1 + $max_{1 \leq k < j}\{\text{M}(1, k)$ and $s_k > s_j\}$;

   $L[j] = 1 + max\{L[i] : i < j$ and $s_i > s_j\}$ (where max equal 1 if $j == 1$)

2. Using your recurrence relation, give a dynamic programming algorithm

   function LDS(s)
   for j=1 to n do
   L[j] = 1
   P[j] = 0
   for i=1 to j-1 do
   if ($s_i > s_j \& L[i] + 1 > L[j]$) then
   P[j] = i
   L[j] = L[i]+1

3. Test your algorithm on this simple sequence [5,0,3,2,1,8]