Introduction to Dynamic Programming

Solving Optimization Problems

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#Algorithms-CSharp

Table of Contents

1. What is Dynamic Programming?
2. Fibonacci Sequence
3. Subset Sum
4. Move Down/Right Sum
5. Longest Common Subsequence

What is Dynamic Programming?

- "**Controlled**" brute force / exhaustive search
- Key ideas:
  - **Subproblems**: like original problem, but smaller
    - Write solution to one **subproblem** in terms of solutions to smaller acyclic subproblems
  - **Memoization**: remember the **solution** to subproblems we've already solved, and **re-use**
    - **Avoid** exponentials
  - **Guessing**: if you don't know something, **guess it!** (try all possibilities)

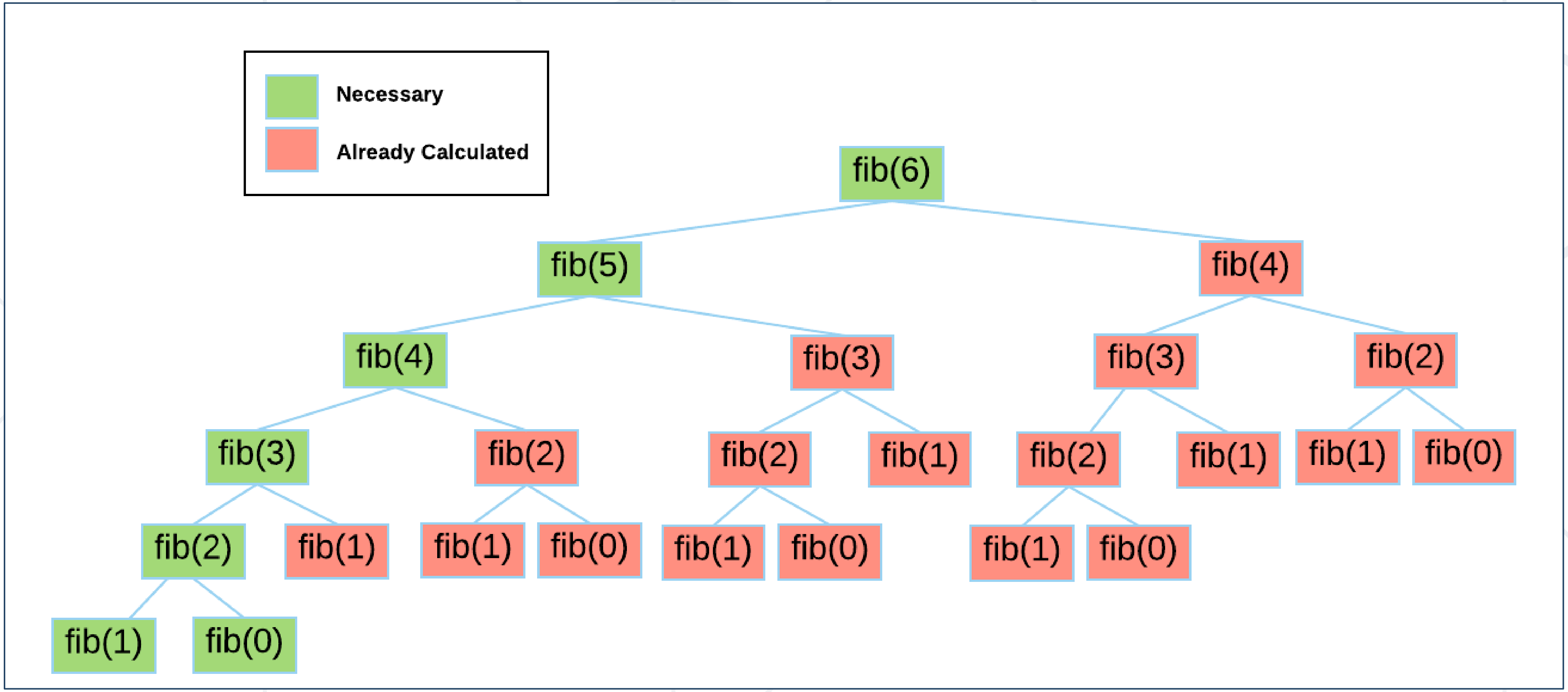
Fibonacci Sequence

Recursive Approach

Example: Fibonacci Sequence

- **The Fibonacci sequence** holds the following integers:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
  - The **first two** numbers are **0** and **1**
  - Each subsequent number is the sum of the previous two numbers
- Recursive mathematical formula:
  - F_0 = 0, F1 = 1
  - Fn = Fn-1 + Fn-2

Recursive Approach



Memoization

- DP → sub-problems **overlap**
- In order to **avoid solving** problems **multiple times**, memorize
  - **Memoization** → **save/cache** sub-problem solutions **for later use**
- Typically using an **array**, **matrix** or a **hash table**

Compare Fibonacci Solutions

- Recursive Fibonacci
  - **~ O(1.6^n)**
- Recursive Fibonacci (with memorization)
  - **~ O(n)**
- If we want to find the 36th Fibonacci number:
  - Recursive solution takes **48 315 633** steps
  - Iterative or recursive (with memorization) takes ~**36** steps

Subset Sum

Sum with Limited Coins

Subset Sum Problem and Its Variations

- **Subset sum problem** (zero subset sum problem)
  - Given a set of integers, find a non-empty **subset whose sum 0**
    - E.g. {8, **3**, -50, **1**, **-2**, -1, 15, **-2**} -> {3, 1, -2, -2}
  - Given a set of integers and an integer **S**, find a subset whose sum is **S**
    - E.g. {8, **3**, 2, **1**, **12**, 1}, S=16 -> {3, 1, 12}
- Given a set of integers, find all possible sums

Subset Sum Problem (No Repeats)

- Solving the subset sum problem:
  - **nums** = { 3, 5, 1, 4, 2 }, **targetSum** = 6
- Start with **possibleSums** = { 0 }
- Step 1: obtain all possible sums ending at { 3 }
  - **possibleSums** = { 0 } ∪ { 0+3 } = { 0, 3 }
- Step 2: obtain all possible sums ending at { 5 }
  - **possibleSums** = { 0, 3 } ∪ { 0+5, 3+5 } = { 0, 3, 5, 8 }
- Step 3: obtain all possible sums ending at { 1 }
  - **possibleSums** = { 0, 3, 5, 8 } ∪ { 0+1, 3+1, 5+1, 8+1 } = {0, 1, 3, 4, 5, 6, 8, 9}

Subset Sum Problem (No Repeats)

```
static ISet<int> CalcPossibleSumsSet(int[] nums)
{
  var possibleSums = new HashSet<int> { 0 };
  foreach (var num in nums) {
    var newSums = new HashSet<int>();
    foreach (var sum in possibleSums) {
      var newSum = sum + num;
      newSums.Add(newSum);
    }
    possibleSums.UnionWith(newSums);
  }
  return possibleSums;
}
```

Subset Sum: How to Recover the Subset?

- Keep for each obtained sum in `possibleSums` how it is obtained
- Use a dictionary instead of set:
  - `possibleSums[s]` -> `num`
  - The sum `s` is obtained by adding `num` to some previously obtained subset sum
    - `s` – `num` gives us the previous sum

Subset Sum (No Repeats + Subset Recovery)

```
static IDictionary<int, int> CalcPossibleSums(int[] nums)
{
  var possibleSums = new Dictionary<int, int> { { 0, 0 } };
  foreach (var num in nums) {
    var newSums = new Dictionary<int, int>();
    foreach (var sum in possibleSums.Keys) {
      var newSum = sum + num;
      if (!possibleSums.ContainsKey(newSum))
        newSums.Add(newSum, num);
    }
    foreach (var sum in newSums)
      possibleSums.Add(sum.Key, sum.Value);
  }
  return possibleSums;
}
```

Subset Sum (No Repeats): Subset Recovery

```
static List<int> FindSubset(
  int targetSum, IDictionary<int, int> possibleSums)
{
  var subset = new List<int>();
  while (targetSum > 0)
  {
    var lastNum = possibleSums[targetSum];
    subset.Add(lastNum);
    targetSum -= lastNum;
  }

  subset.Reverse();
  return subset;
}
```

Subset Sum Problem (with Repetition)

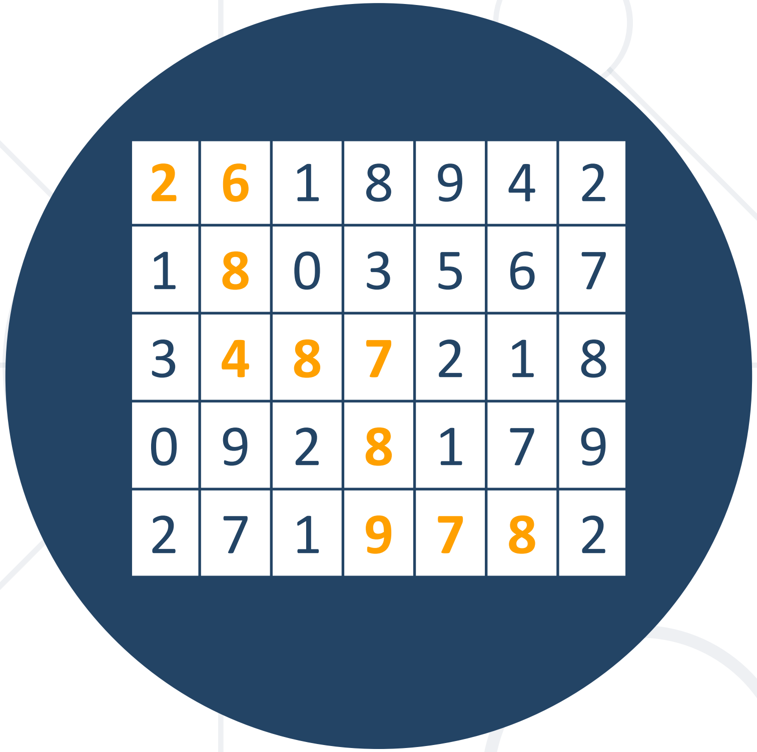
- Given a **set of integers** and an integer **S**, find a subset whose **sum is S**
  - Repetitions are allowed
  - E.g. {3, 5, 2}, S=17
    - {5, 5, 5, 2}
    - {3, 3, 3, 3, 3, 2}
    - {5, 5, 2, 2, 3}
    - …

Subset Sum (with Repetition)

```
static bool[] CalcPossibleSums(int[] nums, int targetSum) {
  var possible = new bool[targetSum + 1];
  possible[0] = true;
  for (int sum = 0; sum < possible.Length; sum++) {
    if (!possible[sum]) continue;
    foreach (var num in nums) {
      var newSum = sum + num;
      if (newSum <= targetSum)
        possible[newSum] = true;
    }
  }
  return possible;
}
```

Subset Sum (with Repetition): Recovery

```
static List<int> FindSubset(
  int[] nums, int targetSum, bool[] possibleSums) {
  var subset = new List<int>();
  while (targetSum > 0) {
    foreach (var num in nums) {
      var newSum = targetSum - num;
      if (newSum >= 0 && possibleSums[newSum]) {
        targetSum = newSum;
        subset.Add(num);
      }
    }
  }
  return subset;
}
```



2	**6**	1	8	9	4	2
1	**8**	0	3	5	6	7
3	**4**	**8**	**7**	2	1	8
0	9	2	**8**	1	7	9
2	7	1	**9**	**7**	**8**	2

Move Down/Right Sum

Largest Sum in Matrix of Numbers

"Move Down / Right Sum" Problem

- You are given a matrix of numbers
  - Find the **path with largest sum**
  - Start → top left
  - End → bottom right
  - Move only right/down
  - There won't be negative numbers

2	**6**	1	8	9	4	2
1	**8**	0	3	5	6	7
3	**4**	**8**	**7**	2	1	8
0	9	2	**8**	1	7	9
2	7	1	**9**	**7**	**8**	2
4	5	6	1	2	**5**	**6**
9	3	5	2	8	1	**9**
2	3	4	1	7	2	**8**

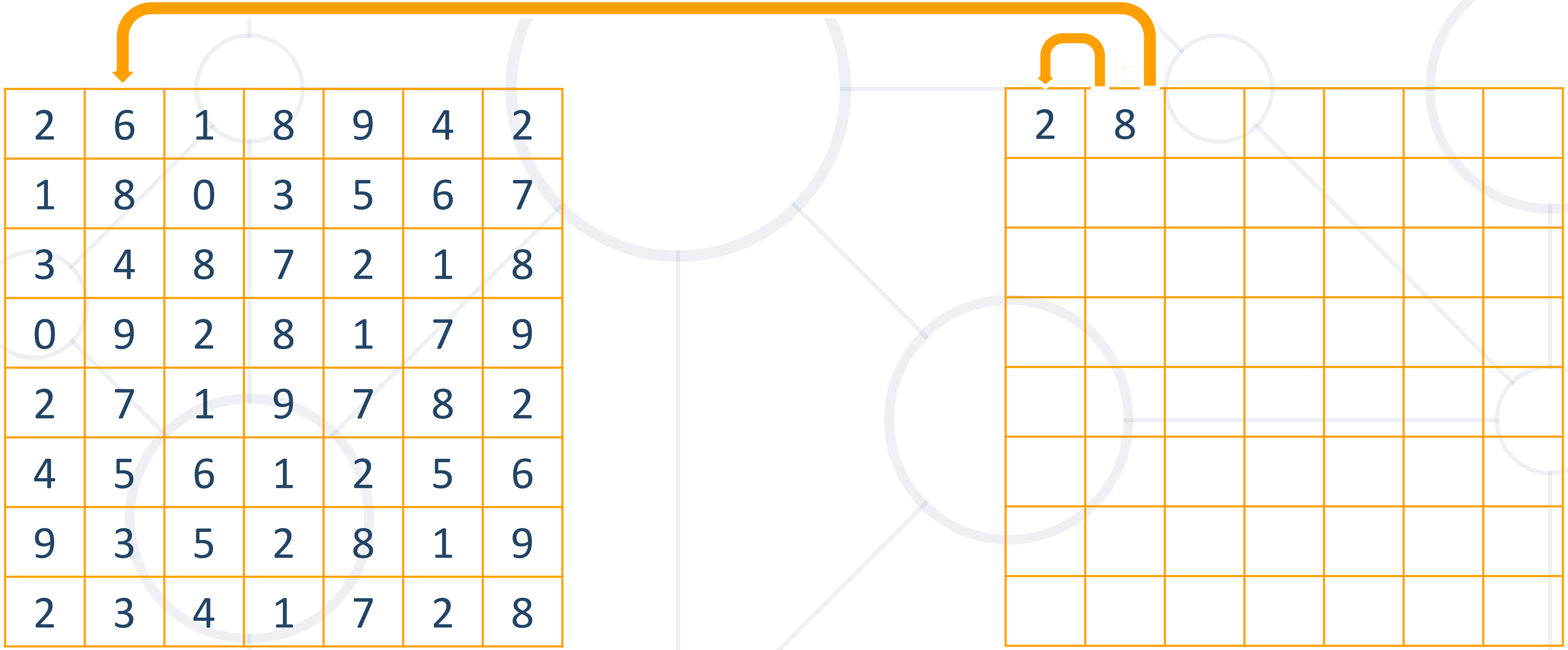
Building the DP Matrix

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

→

2						

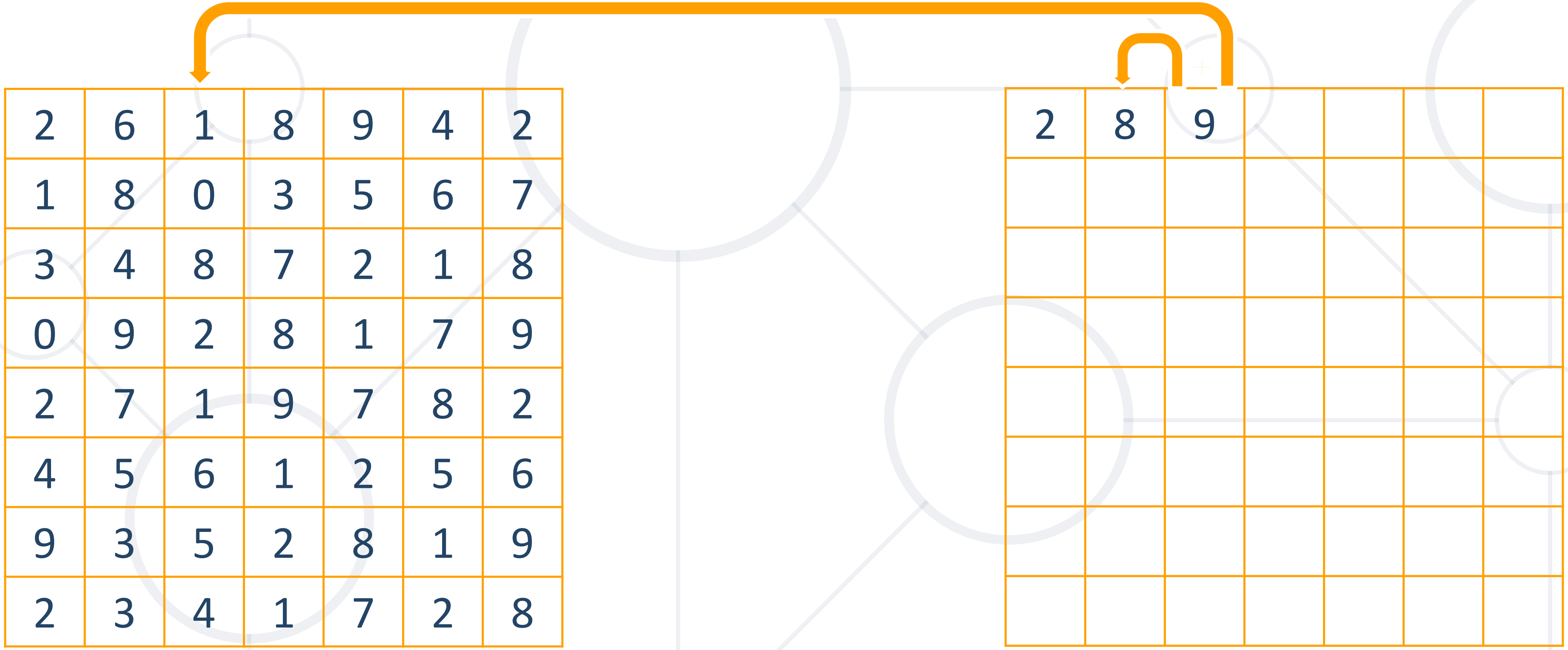
Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8					

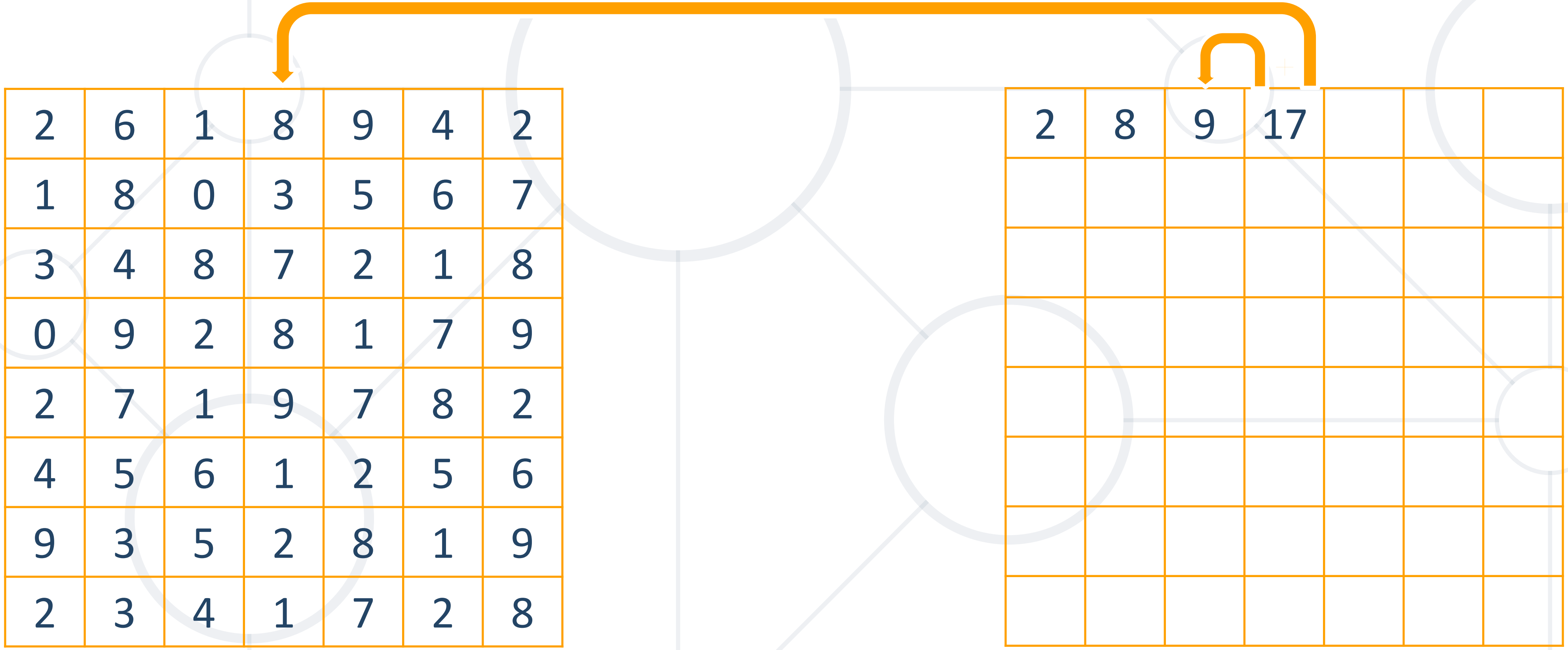
Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9				

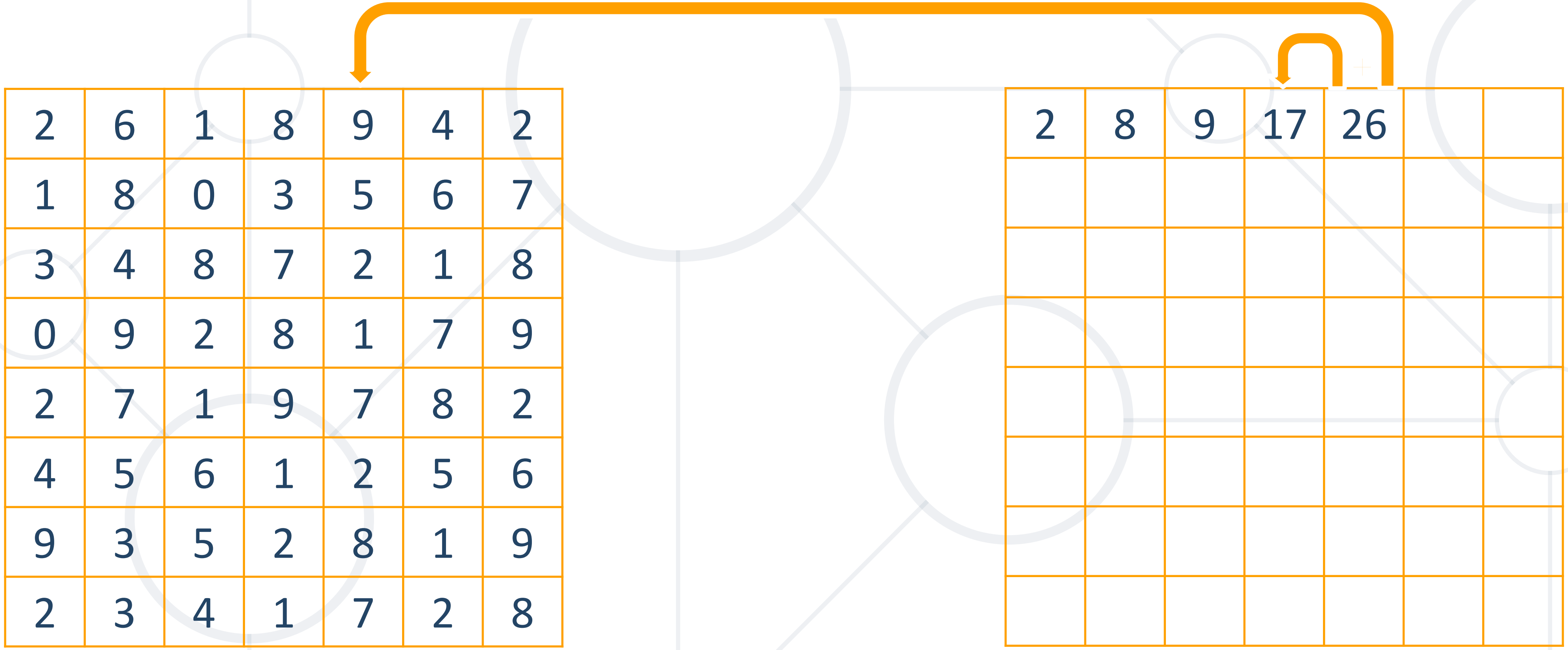
Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17			

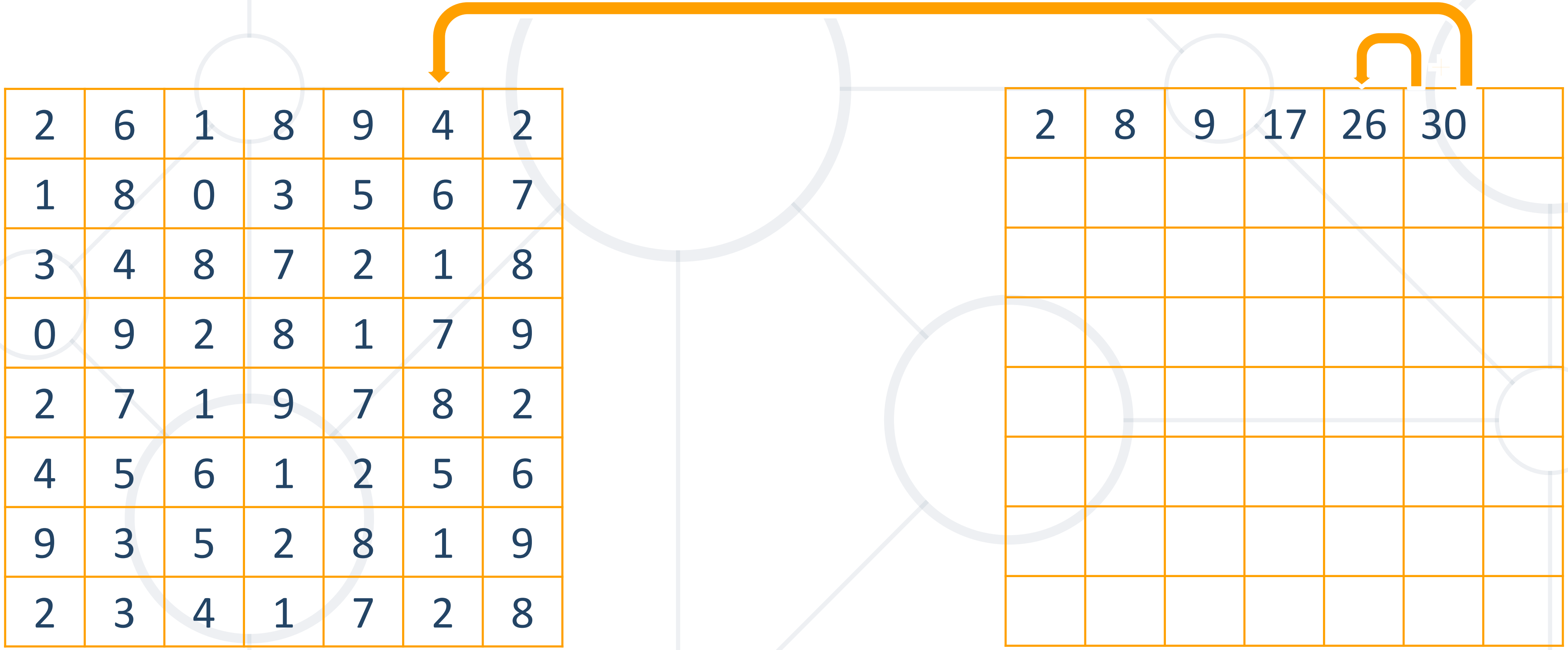
Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17	26		

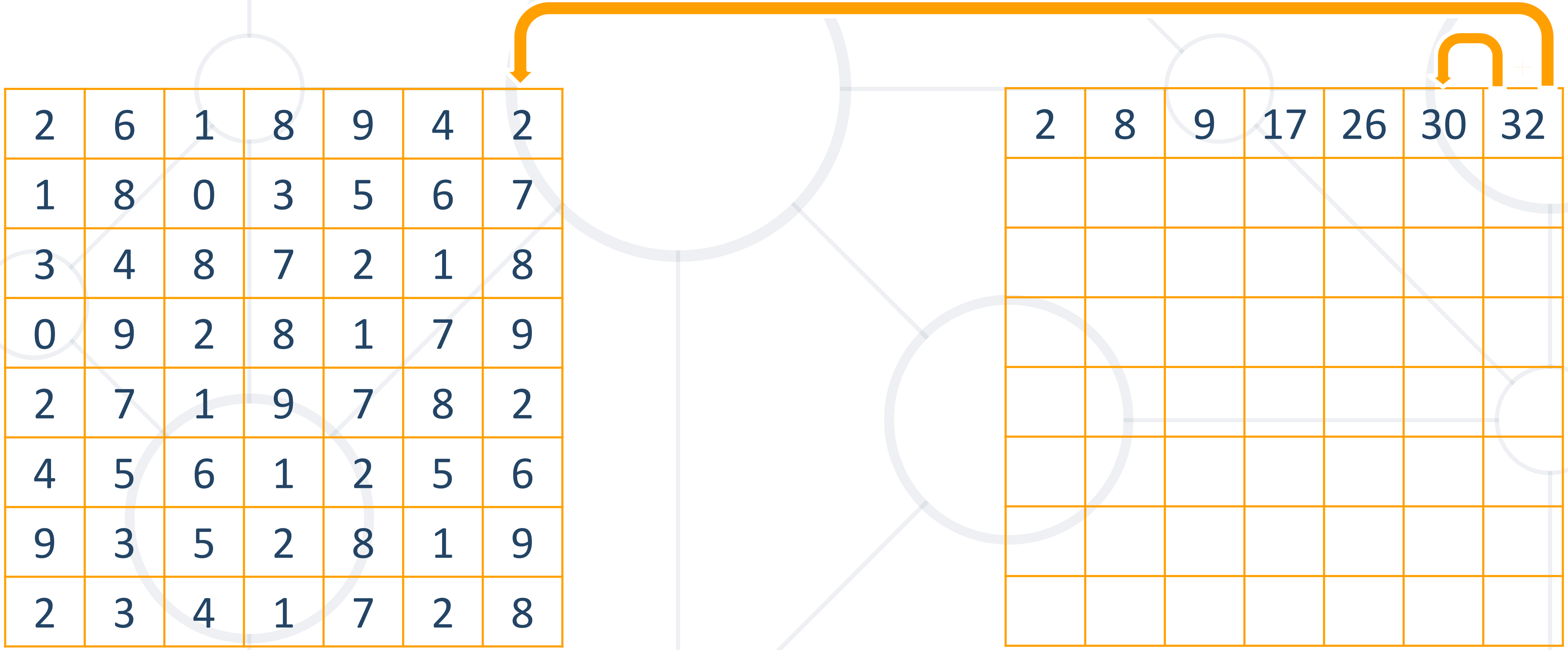
Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17	26	30	

Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17	26	30	32

Building the DP Matrix

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17	26	30	32
3						

Building the DP Matrix

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17	26	30	32
3						
6						

Building the DP Matrix

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

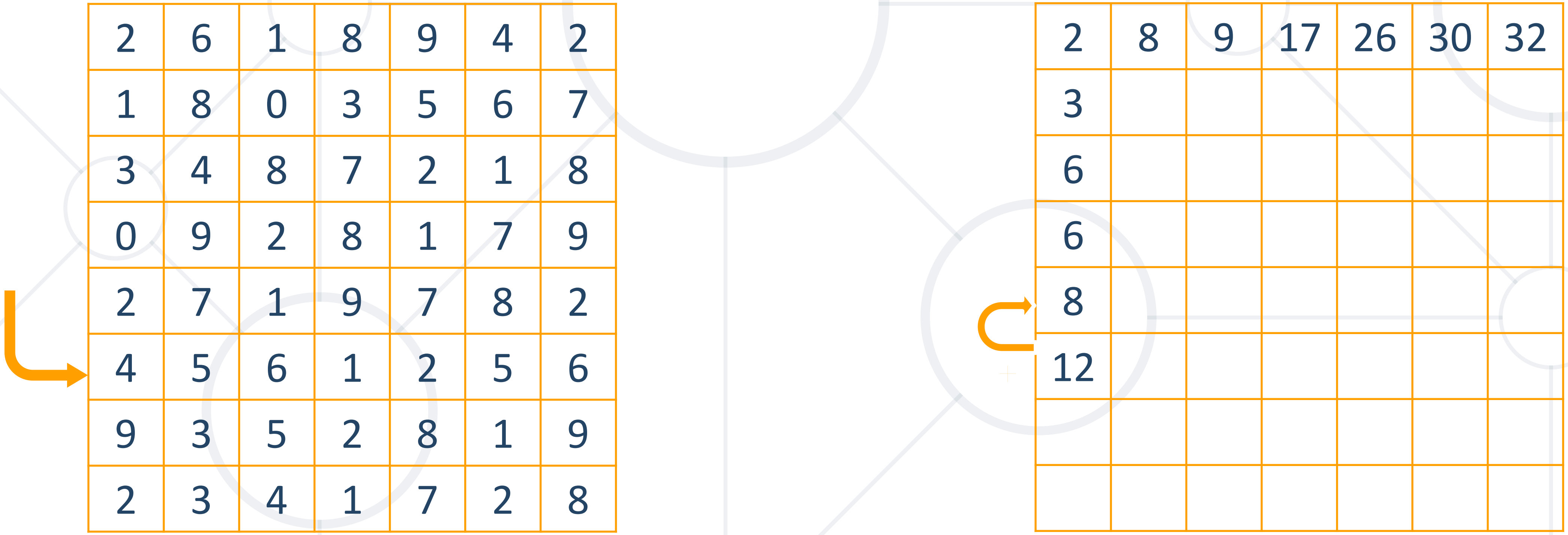
2	8	9	17	26	30	32
3						
6						
6						

Building the DP Matrix

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17	26	30	32
3						
6						
6						
8						

Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17	26	30	32
3						
6						
6						
8						
12						

Building the DP Matrix

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

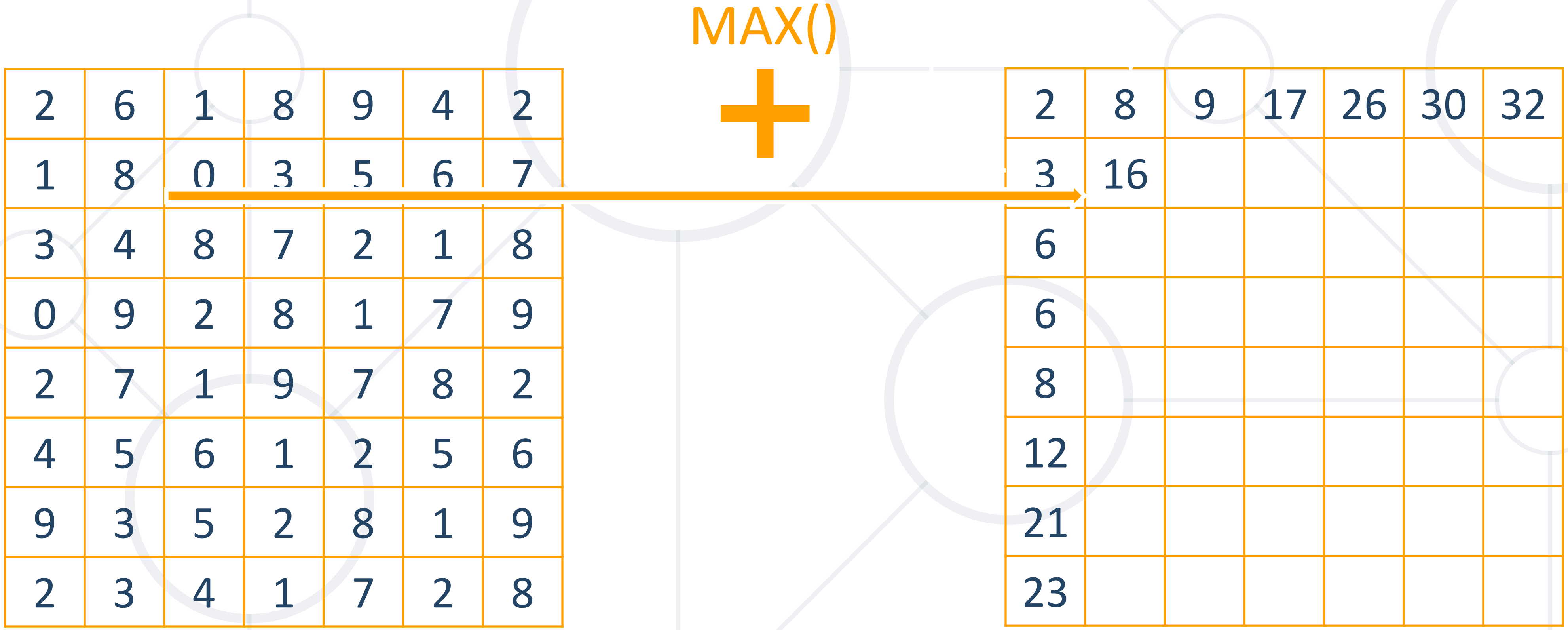
2	8	9	17	26	30	32
3						
6						
6						
8						
12						
21						

Building the DP Matrix

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

2	8	9	17	26	30	32
3						
6						
6						
8						
12						
21						
23						

Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

MAX()

+

2	8	9	17	26	30	32
3	16					
6						
6						
8						
12						
21						
23						

Building the DP Matrix

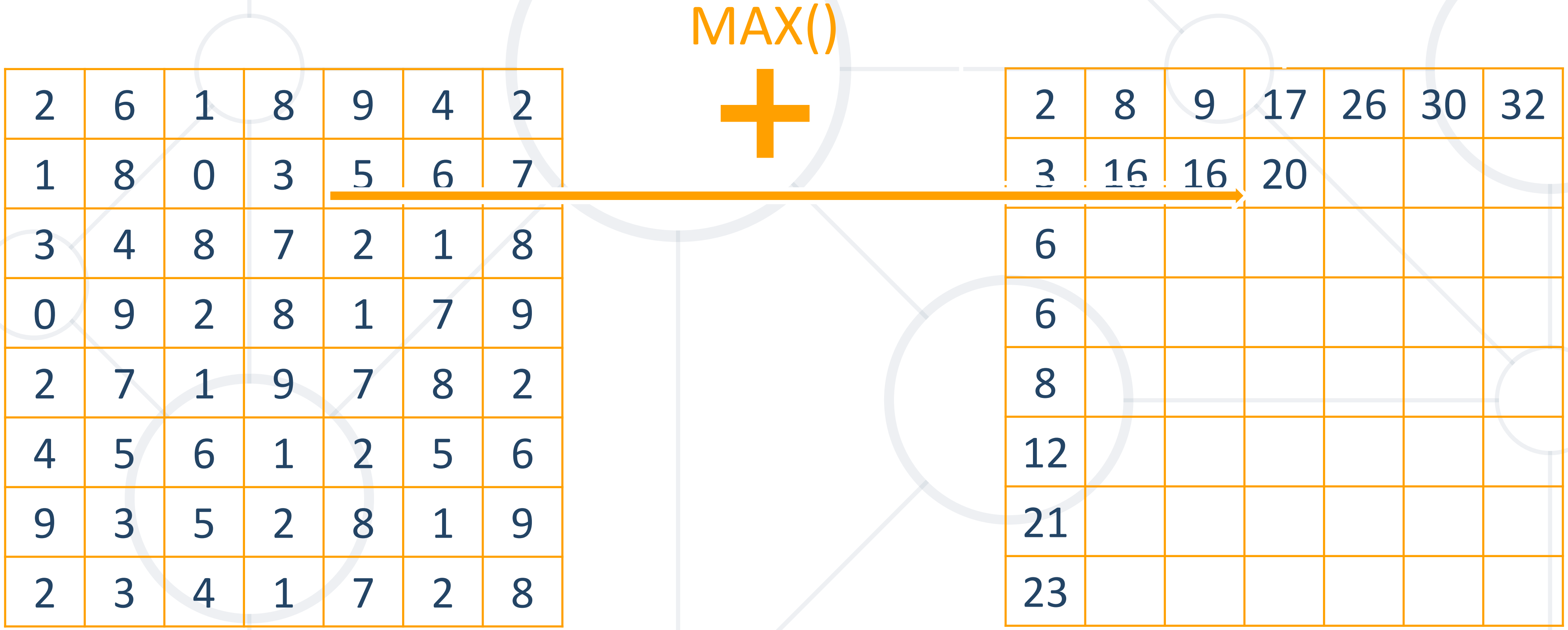
2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

MAX()

+

2	8	9	17	26	30	32
3	16	16				
6						
6						
8						
12						
21						
23						

Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

MAX()

+

2	8	9	17	26	30	32
3	16	16	20			
6						
6						
8						
12						
21						
23						

Building the DP Matrix

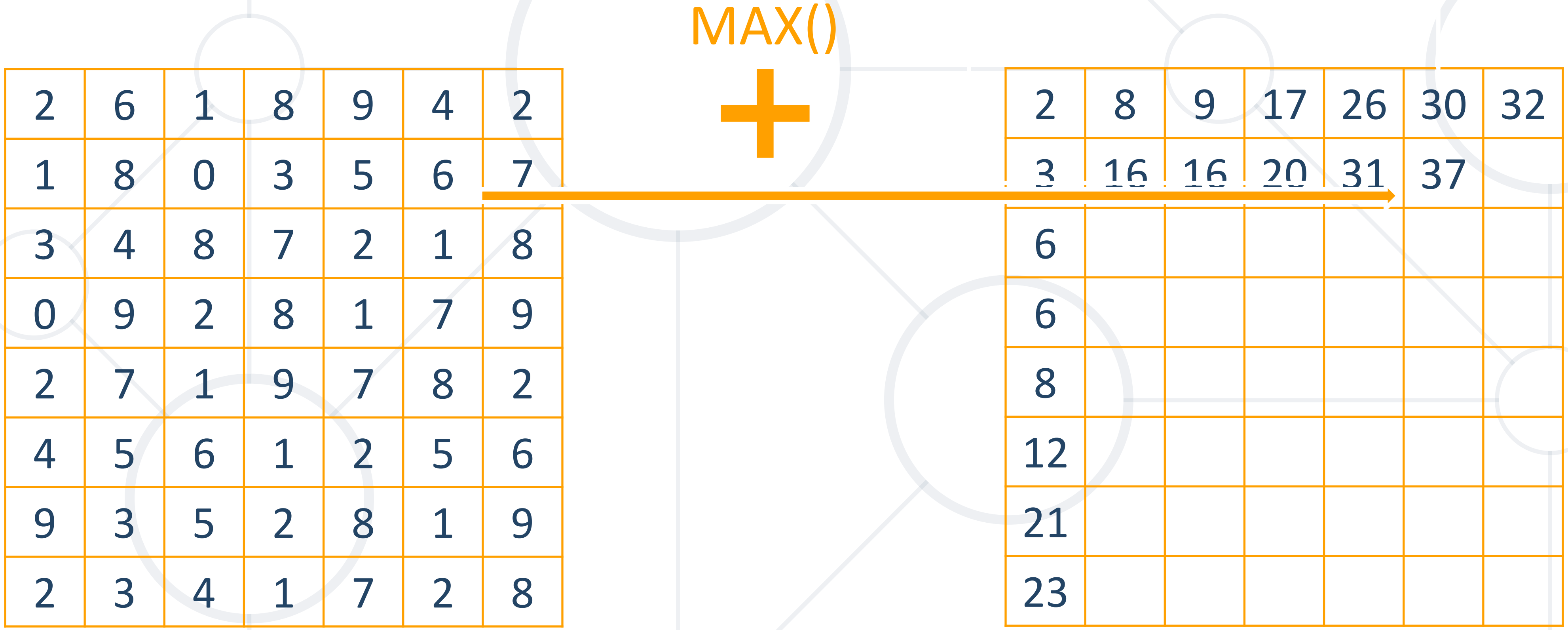
2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

MAX()

+

2	8	9	17	26	30	32
3	16	16	20	31		
6						
6						
8						
12						
21						
23						

Building the DP Matrix



2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

MAX()

+

2	8	9	17	26	30	32
3	16	16	20	31	37	
6						
6						
8						
12						
21						
23						

Building the DP Matrix

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

MAX()

+

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6						
6						
8						
12						
21						
23						

Building the DP Matrix

Start

2	6	1	8	9	4	2
1	8	0	3	5	6	7
3	4	8	7	2	1	8
0	9	2	8	1	7	9
2	7	1	9	7	8	2
4	5	6	1	2	5	6
9	3	5	2	8	1	9
2	3	4	1	7	2	8

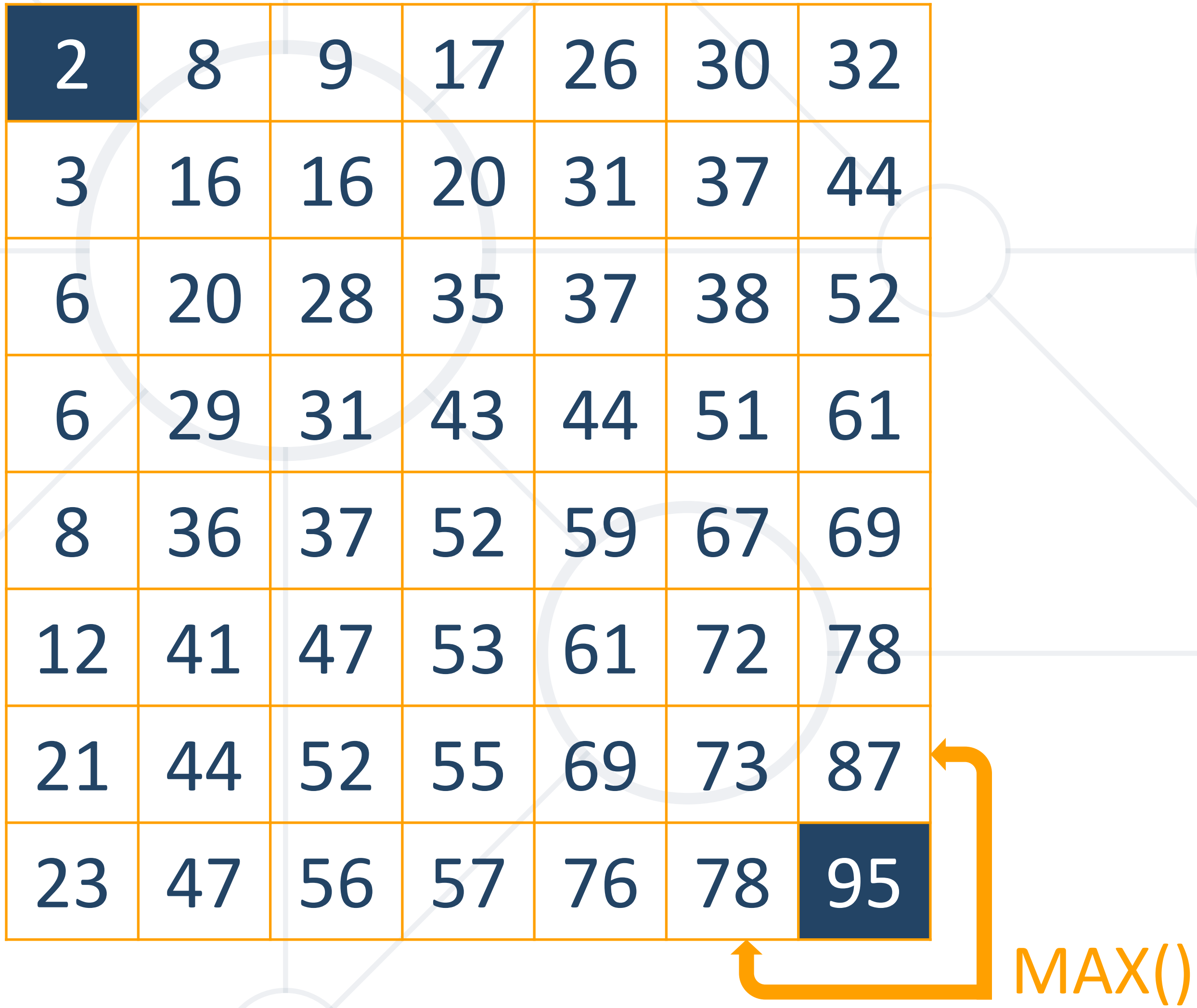
End

Start

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	43	44	51	61
8	36	37	52	59	67	69
12	41	47	53	61	72	78
21	44	52	55	69	73	87
23	47	56	57	76	78	95

End

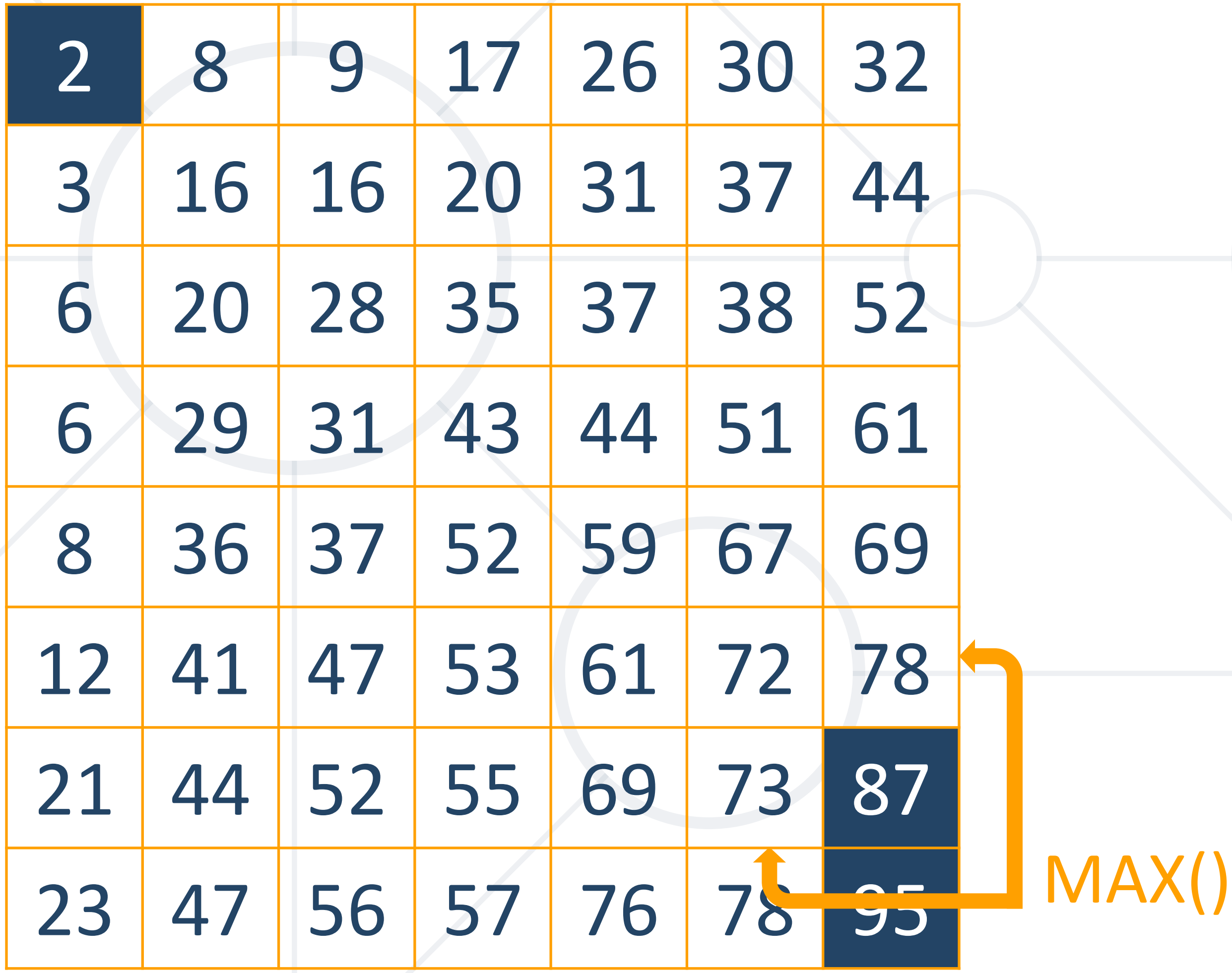
Finding the Path



2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	43	44	51	61
8	36	37	52	59	67	69
12	41	47	53	61	72	78
21	44	52	55	69	73	87
23	47	56	57	76	78	95

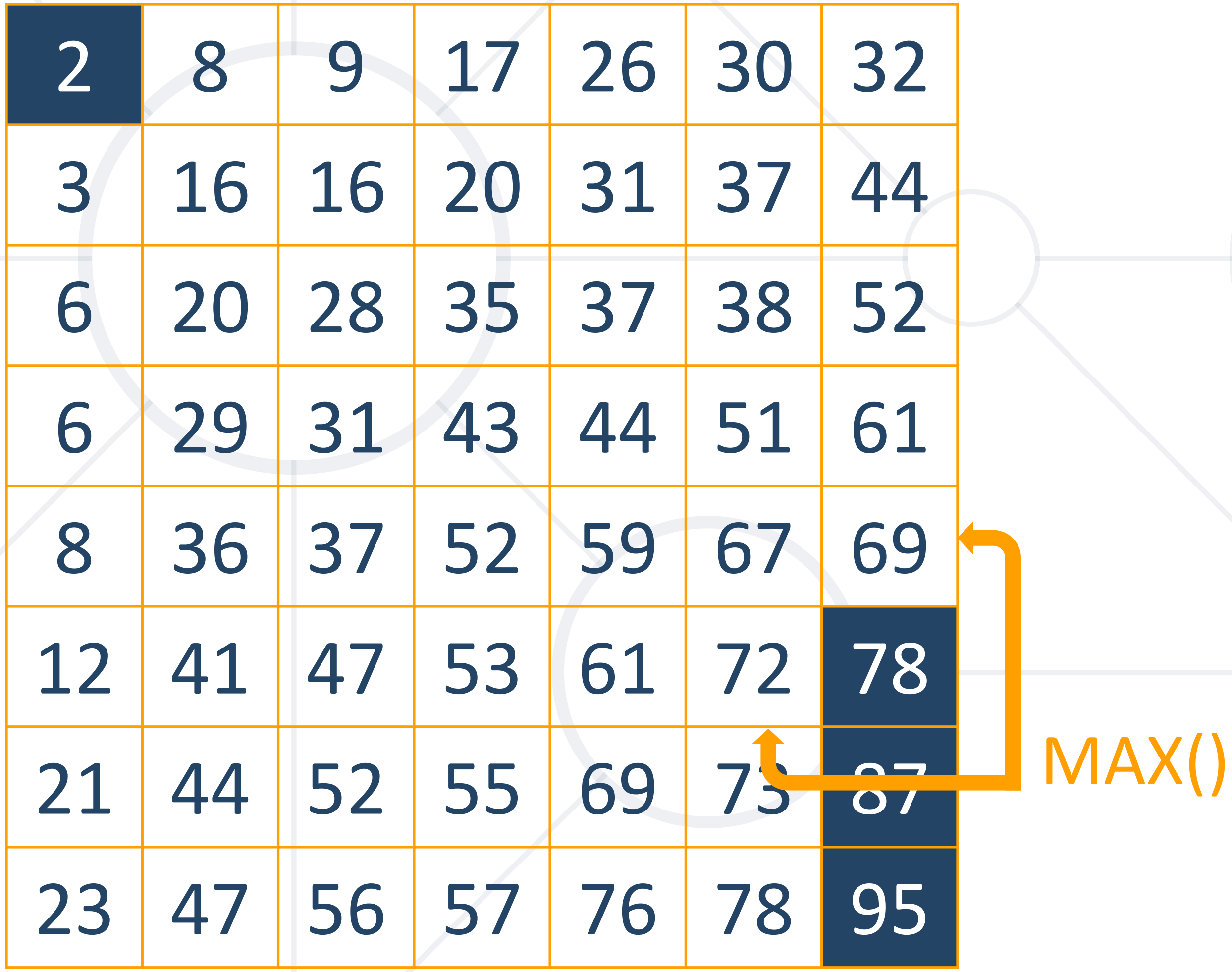
MAX()

Finding the Path



2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	43	44	51	61
8	36	37	52	59	67	69
12	41	47	53	61	72	78
21	44	52	55	69	73	87
23	47	56	57	76	78	95

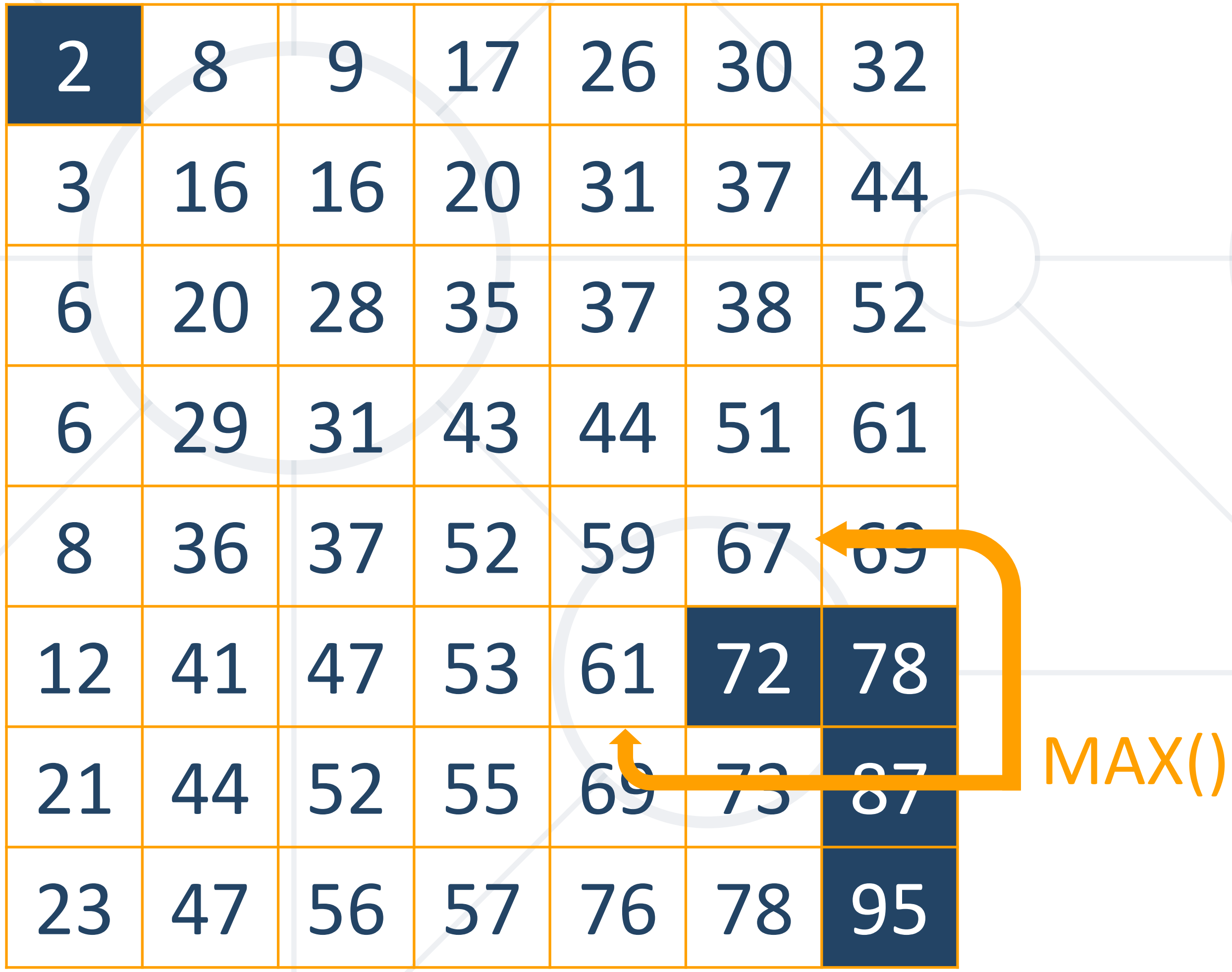
Finding the Path



2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	43	44	51	61
8	36	37	52	59	67	69
12	41	47	53	61	72	78
21	44	52	55	69	73	87
23	47	56	57	76	78	95

MAX()

Finding the Path



2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	43	44	51	61
8	36	37	52	59	67	69
12	41	47	53	61	72	78
21	44	52	55	69	73	87
23	47	56	57	76	78	95

Finding the Path

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	43	44	51	61
8	36	37	52	59	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

Finding the Path

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	43	44	51	61
8	36	37	52	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

Finding the Path

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	43	44	51	61
8	36	37	**52**	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

Finding the Path

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	35	37	38	52
6	29	31	**43**	44	51	61
8	36	37	**52**	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

Finding the Path

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	28	**35**	37	38	52
6	29	31	**43**	44	51	61
8	36	37	**52**	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

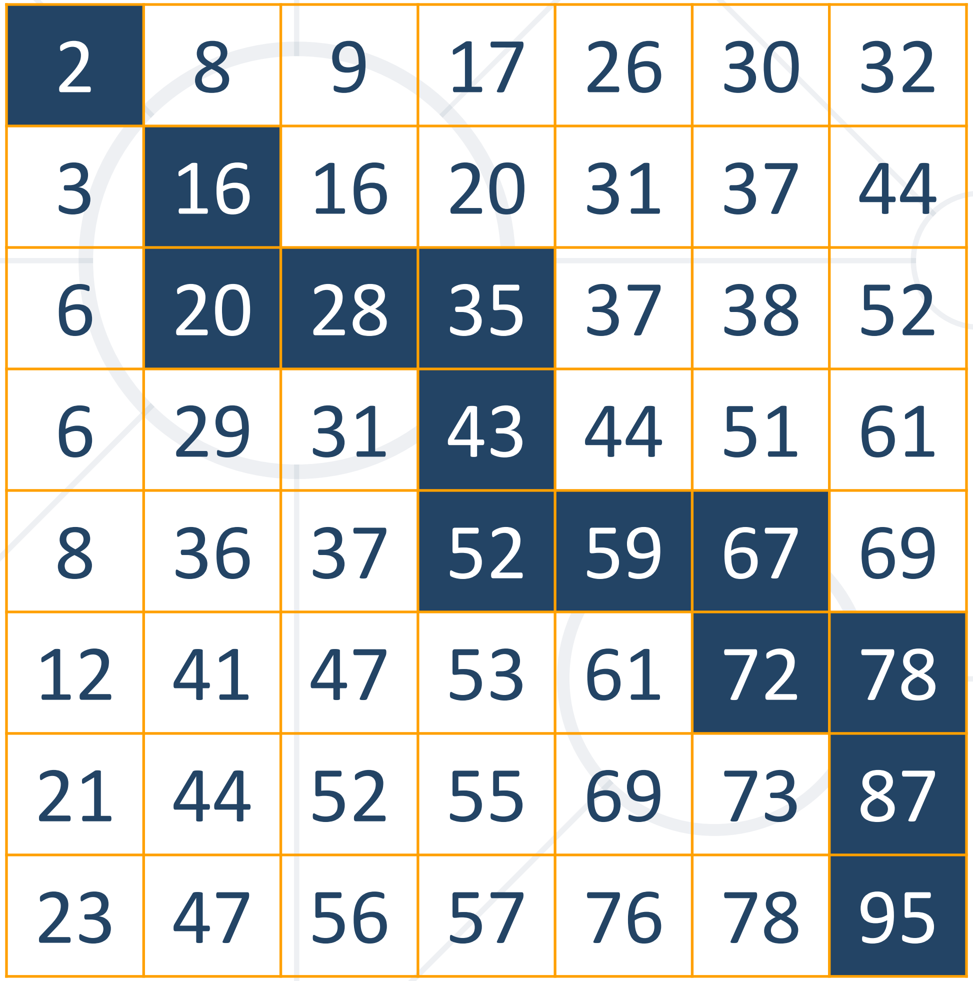
Finding the Path

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	20	**28**	**35**	37	38	52
6	29	31	**43**	44	51	61
8	36	37	**52**	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

Finding the Path

2	8	9	17	26	30	32
3	16	16	20	31	37	44
6	**20**	**28**	**35**	37	38	52
6	29	31	**43**	44	51	61
8	36	37	**52**	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

Finding the Path

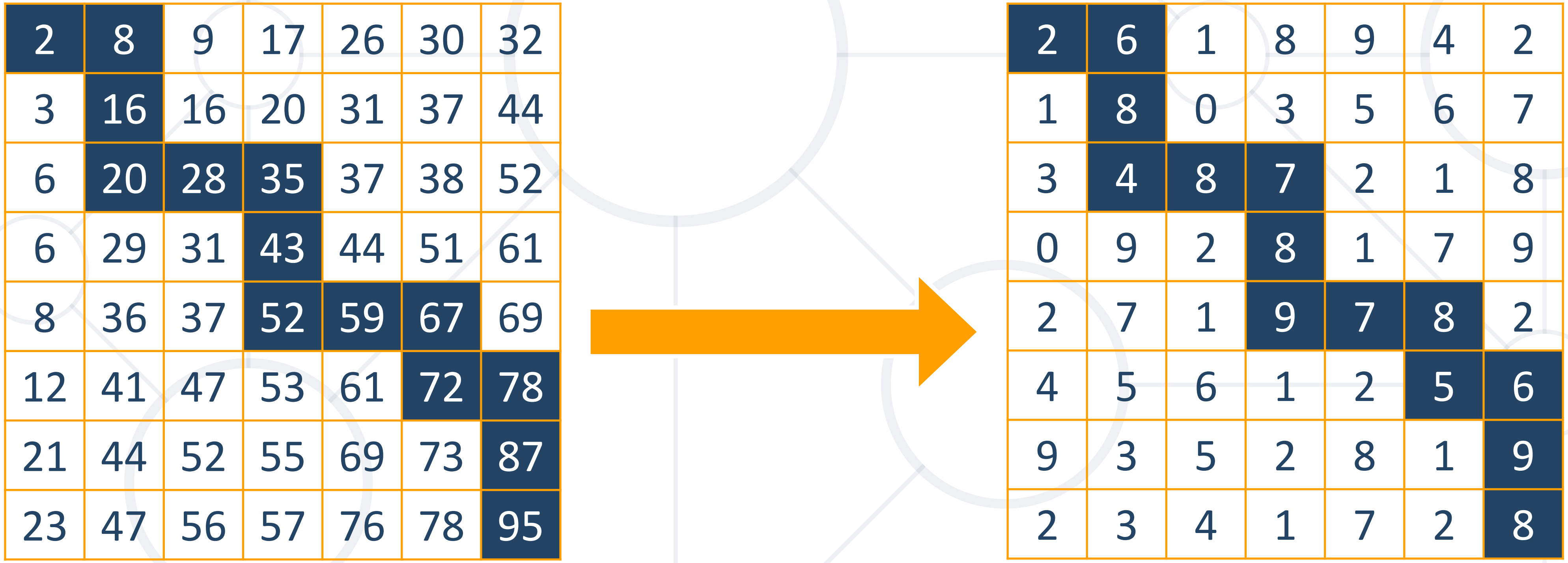


2	8	9	17	26	30	32
3	**16**	16	20	31	37	44
6	**20**	**28**	**35**	37	38	52
6	29	31	**43**	44	51	61
8	36	37	**52**	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

Finding the Path

2	**8**	9	17	26	30	32
3	**16**	16	20	31	37	44
6	**20**	**28**	**35**	37	38	52
6	29	31	**43**	44	51	61
8	36	37	**52**	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

Finding the Path

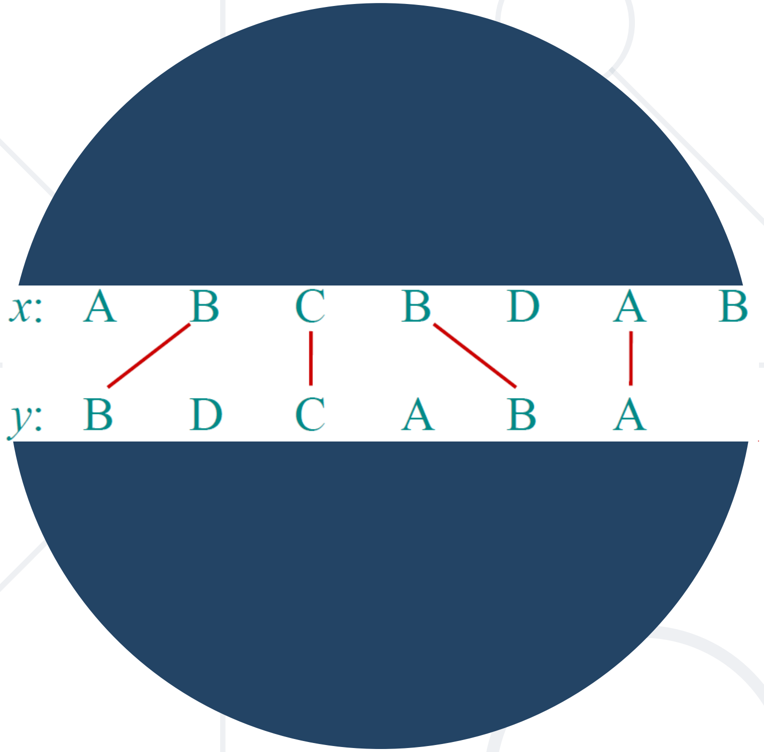


2	**8**	9	17	26	30	32
3	**16**	16	20	31	37	44
6	**20**	**28**	**35**	37	38	52
6	29	31	**43**	44	51	61
8	36	37	**52**	**59**	**67**	69
12	41	47	53	61	**72**	**78**
21	44	52	55	69	73	**87**
23	47	56	57	76	78	**95**

2	**6**	1	8	9	4	2
1	**8**	0	3	5	6	7
3	**4**	**8**	**7**	2	1	8
0	9	2	**8**	1	7	9
2	7	1	**9**	**7**	**8**	2
4	5	6	1	2	**5**	**6**
9	3	5	2	8	1	**9**
2	3	4	1	7	2	**8**

"Move Down / Right Sum" – Solution

```
for (int row = 0; row < rowsCount; row++) {
  for (int col = 0; col < colsCount; col++) {
    long maxPrevCell = long.MinValue;
    if (col > 0 && sum[row, col - 1] > maxPrevCell)
      maxPrevCell = sum[row, col - 1];
    if (row > 0 && sum[row - 1, col] > maxPrevCell)
      maxPrevCell = sum[row - 1, col];
    sum[row, col] = cells[row, col];
    if (maxPrevCell != long.MinValue)
      sum[row, col] += maxPrevCell;
  }
}
```

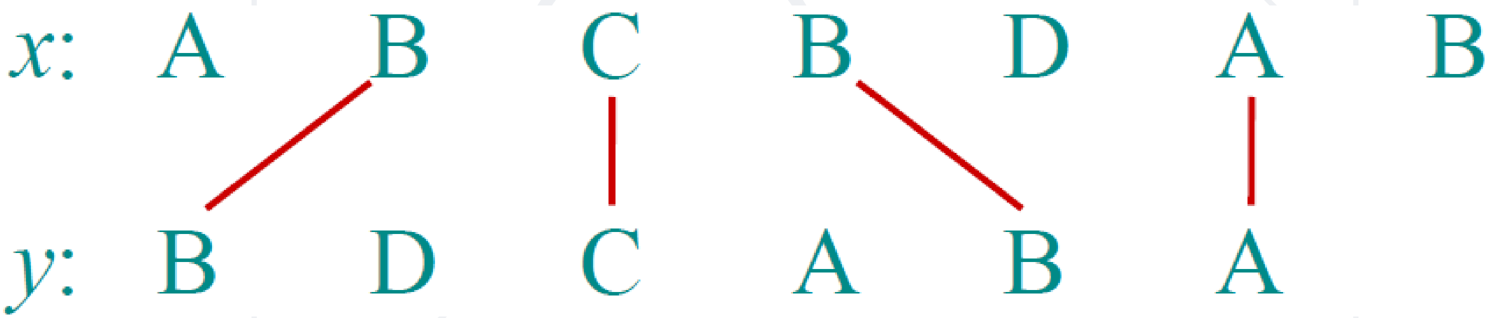


Longest Common Subsequence (LCS)

A Recursive DP Approach

Longest Common Subsequence (LCS)

- Longest common subsequence (LCS) problem:
  - Given two sequences **x[1 … m]** and **y[1 … n]**
  - Find a longest common subsequence (LCS) to them both
- Example:
  - x = "A**BCB**D**A**B"
  - y = "**B**D**CABA**"
  - LCS = "**BCBA**"



LCS – Recursive Approach

- S_1 = **GCCCTAGCG**, S_2 = **GCGCAATG**
  - Let C_1 = the right-most character of S_1 (C_1 = G)
  - Let C_2 = the right-most character of S_2 (C_2 = G)
  - Let S_1' = S_1 with C_1 "chopped-off" (S_1' = GCCCTAGC)
  - Let S_2' = S_2 with C_2 "chopped-off" (S_2' = GCGCAAT)
- There are three recursive sub-problems:
  - $L_1 = LCS(S_1', S_2)$
  - $L_2 = LCS(S_1, S_2')$
  - $L_3 = LCS(S_1', S_2')$

LCS – Recursive Formula

- Let lcs[x][y] be the longest common subsequence of S1[0 … x] and S_2[0 … y]
- LCS has the following recursive properties:

```
lcs[-1][y] = 0
lcs[x][-1] = 0
lcs[x][y] = max(
  lcs[x-1][y],
  lcs[x][y-1],
  lcs[x-1][y-1]+1 when S1[x] == S2[y])
```

Calculating the LCS Table

```
var str1 = Console.ReadLine();
var str2 = Console.ReadLine();
var lcs = new int[str1.Length + 1, str2.Length + 1];
for (int r = 1; r < lcs.GetLength(0); r++)
{
  for (int c = 1; c < lcs.GetLength(1); c++)
  {
    if (str1[r - 1] == str2[c - 1])
      lcs[r, c] = lcs[r - 1, c - 1] + 1;
    else
      lcs[r, c] = Math.Max(lcs[r, c - 1], lcs[r - 1, c]);
  }
}
```

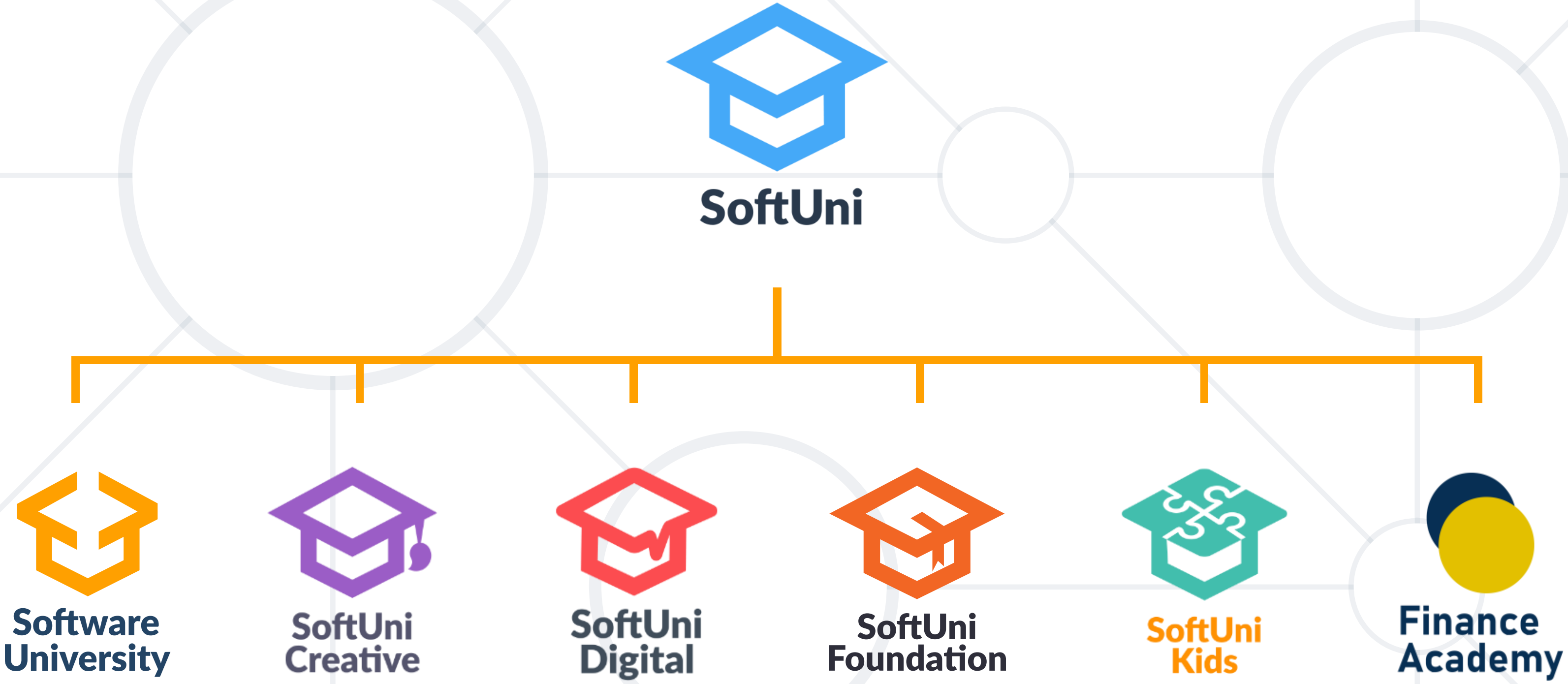
Reconstructing the LCS Sequence

```
static string PrintLCS(
  int row, int col, string str1, string str2, int[][] lcs) {
  var lcsLetters = new Stack<char>();
  while (row >= 0 && col >= 0) {
    if (str1[row] == str2[col]) {
      lcsLetters.Push(str1[row]);
      row--;
      col--;
    } else if (lcs[row - 1][col] > lcs[row][col - 1]) { row--; }
      else { col--; }
  }
  return string.Join("", lcsLetters);
}
```

Summary

- **DP** → Solve a problem by **solving overlapping subproblems**
- **Memoization** → **Save** subproblem **solutions** for later use
- **Optimal Substructure**
  - **Subproblems** should have **optimal solutions**
  - Combine optimal solutions for subproblems
  - Get optimal solution for original problem

Questions?


SoftUni
Software University
SoftUni Creative
SoftUni Digital
SoftUni Foundation
SoftUni Kids
Finance Academy

SoftUni Diamond Partners

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

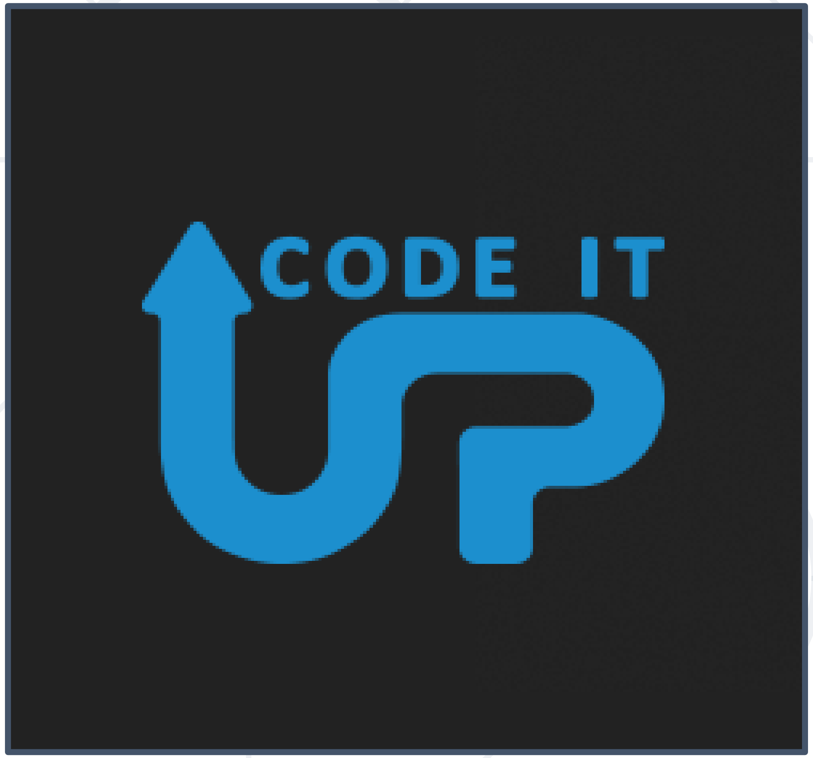
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



License

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

FORUM