Dynamic Programming

Solving Optimization Problems with Dynamic Programming Part 2

SoftUni Foundation

SoftUni Team

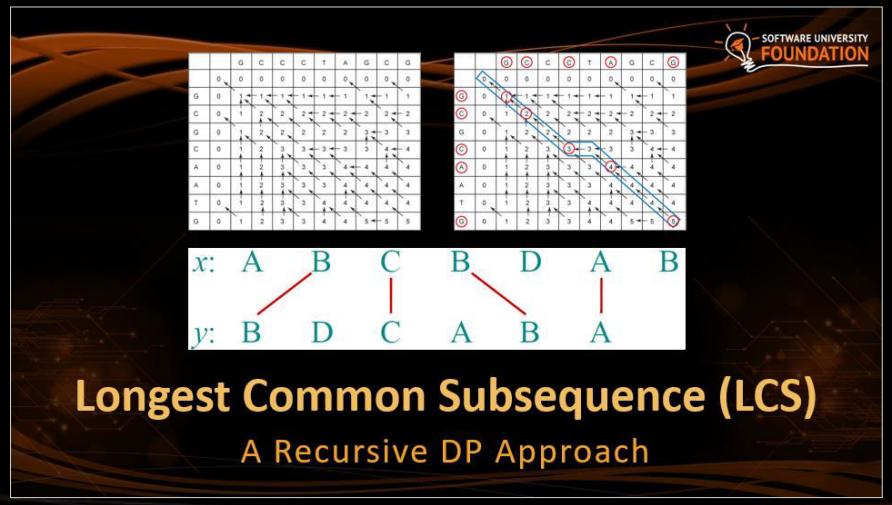
Technical Trainers

Software University

http://softuni.bg

Table of Contents

SoftUni Foundation



Have a Question?

SoftUni Foundation

sli.do

#DsAlgo

SoftUni Foundation

0/1 Knapsack Problem

Take Best Items of a Collection

The "Knapsack" Problem

SoftUni Foundation

- We have n items
  - Each item has price c_i and weight w_i
- We have a knapsack of given capacity (maximum weight)
  - We want to take maximal priced items within the knapsack capacity
- Take a subset of the items, so that:
  - The total price is maximal
  - The total weigth ≤ capacity

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

	0	1	2	3	4
No Items					
					
					
					

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

	0	1	2	3	4
No Items	0	0	0	0	0
					
					
					

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

	0	1	2	3	4
No Items	0	0	0	0	0
	0				
	0				
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
max(excluding, including)

	0	1	2	3	4
No Items	0	0	0	0	0
	0				
	0				
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:

```
excluding = 0
```

Including current item value:

```
including = 0
```

Weight > Capacity

Choosing if we are going to take the item

```
max(excluding, including)
```

	0	1	2	3	4
No Items	0	0	0	0	0
	0				
	0				
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0			
	0				
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3		
	0				
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3	3	
	0				
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3	3	3
	0				
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3	3	3
	0	0			
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3	3	3
	0	0	3		
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3	3	3
	0	0	3	3	
	0				

The "Knapsack" Problem – Example


Value: 3
Weight: 2


Value: 1
Weight: 2


Value: 3
Weight: 1


Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3	3	3
	0	0	3	3	4
	0				

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3	3	3
	0	0	3	3	4
	0	3			

The "Knapsack" Problem – Example

Value: 3
Weight: 2

Value: 1
Weight: 2

Value: 3
Weight: 1

Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0	3	3	3
	0	0	3	3	4
	0	3	3		

The "Knapsack" Problem – Example


Value: 3
Weight: 2


Value: 1
Weight: 2


Value: 3
Weight: 1


Max Capacity: 4

Excluding current item value:
matrix[index -1][capacity]

Including current item value:
value + matrix[index - 1][capacity - **weight**]

Choosing if we are going to take the item
`max(excluding, including)`

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0 ✗	3 ✓	3 ✓	3 ✓
	0	0 ✗	3 ✗	3 ✗	4 ✓
	0	3 ✓	3 ✗	6 ✓	○

The "Knapsack" Problem – Example


Value: 3
Weight: 2


Value: 1
Weight: 2


Value: 3
Weight: 1

Maximum Value: 6


Max Capacity: 4

	0	1	2	3	4
No Items	0	0	0	0	0
	0	0 ✗	3 ✓	3 ✓	3 ✓
	0	0 ✗	3 ✗	3 ✗	4 ✓
	0	3 ✓	3 ✗	6 ✓	6 ✓

The "Knapsack" Problem – Example


Weight: 2


Weight: 2


Weight: 1


Max Capacity: 4

Maximum Value: 6

	0	1	2	3	4
No Items					
		✗	✓	✓	✓
		✗	✗	✗	✓
		✓	✗	✓	✓

The "Knapsack" Problem – Example

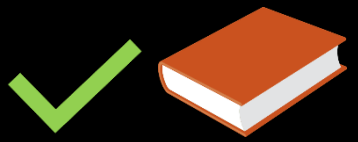

Weight: 2


Weight: 2


Weight: 1


Max Capacity: 4

Maximum Value: 6

	0	1	2	3	4
No Items					
		✗	✓	✓	✓
		✗	✗	✗	✓
		✓	✗	✓	◯

The "Knapsack" Problem – Example


Weight: 2


Weight: 2


Weight: 1


Max Capacity: 4

Maximum Value: 6

	0	1	2	3	4
No Items					
		✗	✓	✓	✓
		✗	✗	✗	✓
✓ 		✓	✗	✓	

The "Knapsack" Problem – Example


Weight: 2


Weight: 2


Weight: 1


Max Capacity: 4

Maximum Value: 6

	0	1	2	3	4
No Items					
		✗	✓	✓	✓
✗ 		✗	✗		✓
✓ 		✓	✗	✓	

The "Knapsack" Problem – Example


Weight: 2


Weight: 2


Weight: 1


Max Capacity: 4

Maximum Value: 6

	0	1	2	3	4
No Items					
✓ 		✗	✓		✓
✗ 		✗	✗		✓
✓ 		✓	✗	✓	

The "Knapsack" Problem – Example

Max Capacity: 4

Maximum Value: 6
With Items:

Total Capacity: 3

The "Knapsack" Problem – Variants

SoftUni Foundation

- Many variants are available for the "Knapsack" problem
  - E.g. take each item at most once / use each item multiple times
  - Integer knapsack problem
    - Items have integer weights
    - Efficient dynamic programming solution exists O(**n*****c**) where
      - **n** – number of items
      - **c** – the knapsack capacity
  - Non-integer knapsack problem → no efficient solution
  - Bin packing problem – m knapsacks of capacities (c_1, …, c_m)

0/1 Knapsack – Iterative Implementation

SoftUni Foundation

```
public static List<Item> FillKnapsack(Item[] items, int capacity)
{
  int[,] maxValues = new int[items.Length + 1, capacity + 1];
  bool[,] itemIncluded = new bool[items.Length + 1, capacity + 1];

  // Calculate maxValues

  // Return the items in the knapsack
}
```

Knapsack – Iterative Implementation (2)

SoftUni Foundation

```
for (int i = 0; i < items.Length; i++)
{
  for (int currCapacity = 1; currCapacity <= capacity; currCapacity++)
  {
    if (items[i].Weight > currCapacity) continue;
    int valueIncluded = items[i].Value
          + maxValues[i, currCapacity - items[i].Weight];
    if (valueIncluded > maxValues[i, currCapacity])
    {
      maxValues[i + 1, currCapacity] = valueIncluded;
      itemIncluded[i + 1, currCapacity] = true;
    }
    else
      maxValues[i + 1, currCapacity] = maxValues[i, currCapacity];
  }
}
```

Knapsack – Iterative Implementation (3)

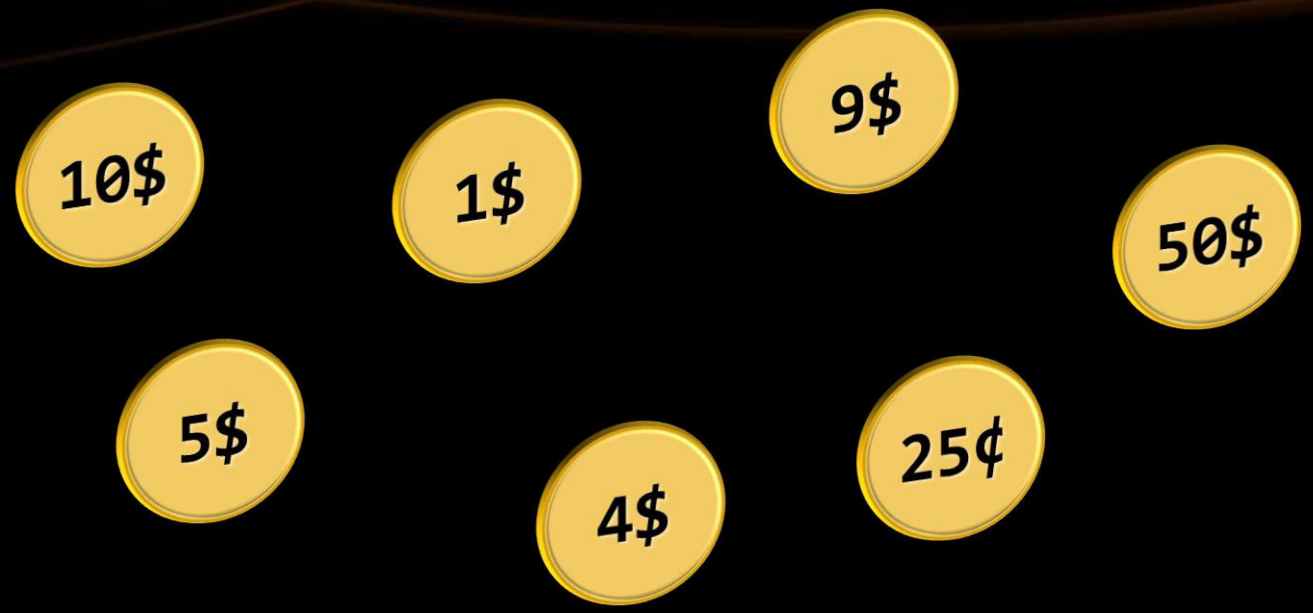
SoftUni Foundation

```
// return the taken items in the knapsack
List<Item> takenItems = new List<Item>();
for (int i = items.Length; i > 0; i--)
{
  if (!itemIncluded[i, capacity]) continue;

  Item item = items[i - 1];
  takenItems.Add(item);

  capacity -= item.Weight;
}

return takenItems;
```

SoftUni Foundation

The 0/1 Knapsack Problem

Live Coding (Lab)



Subset Sum

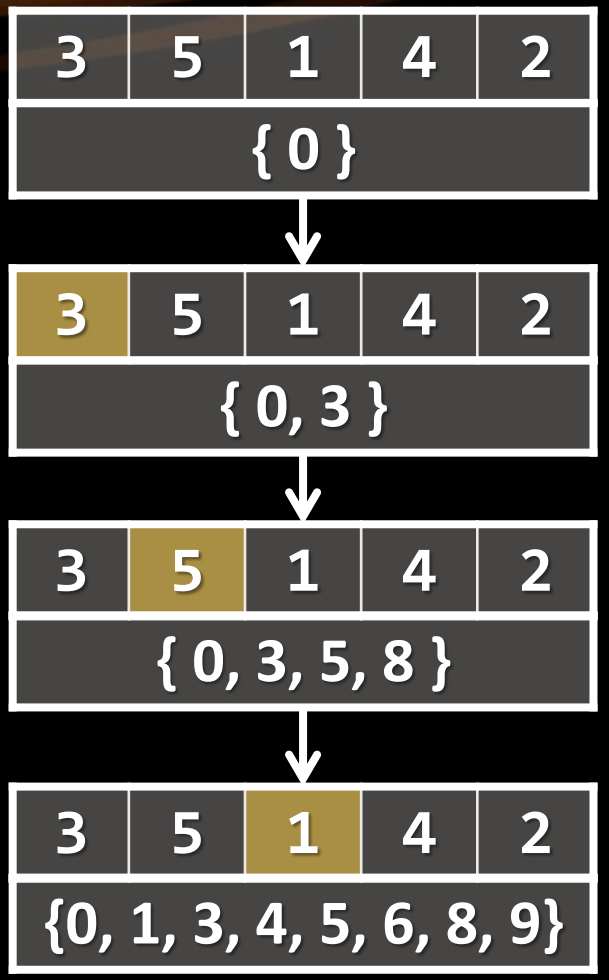
Sum with Limited Coins

Subset Sum Problem and Its Variations

SoftUni Foundation

- 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
- Can you split given sum into set of coins?

Subset Sum Problem (No Repeats)

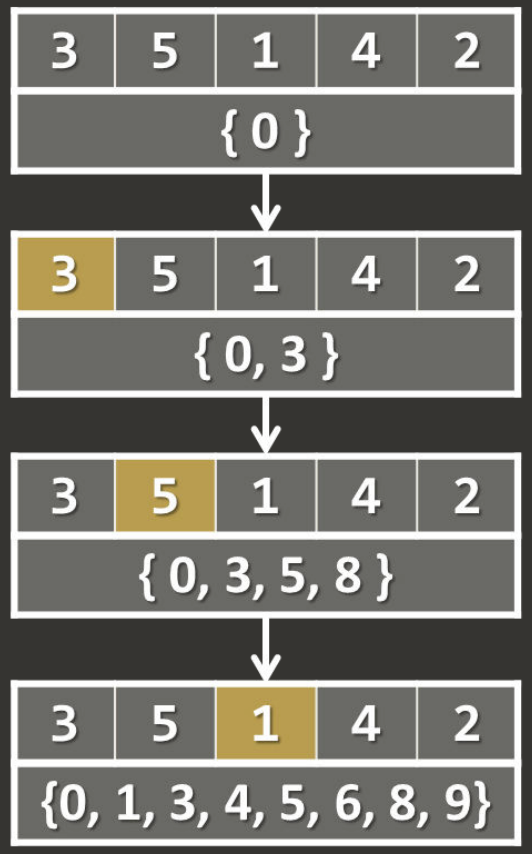


- Solving the subset sum problem:
  - `nums` = { 3, 5, 1, 4, 2 }, `targetSum` = 6
- Start with `possibleSums` = { 0 }
- Step 1: obtain all possible sums ending at { 3 }
  - `possibleSums` = { 0 } ∪ { 0+3 } = { 0, 3 }
- Step 2: obtain all possible sums ending at { 5 }
  - `possibleSums` = { 0, 3 } ∪ {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

SoftUni Foundation

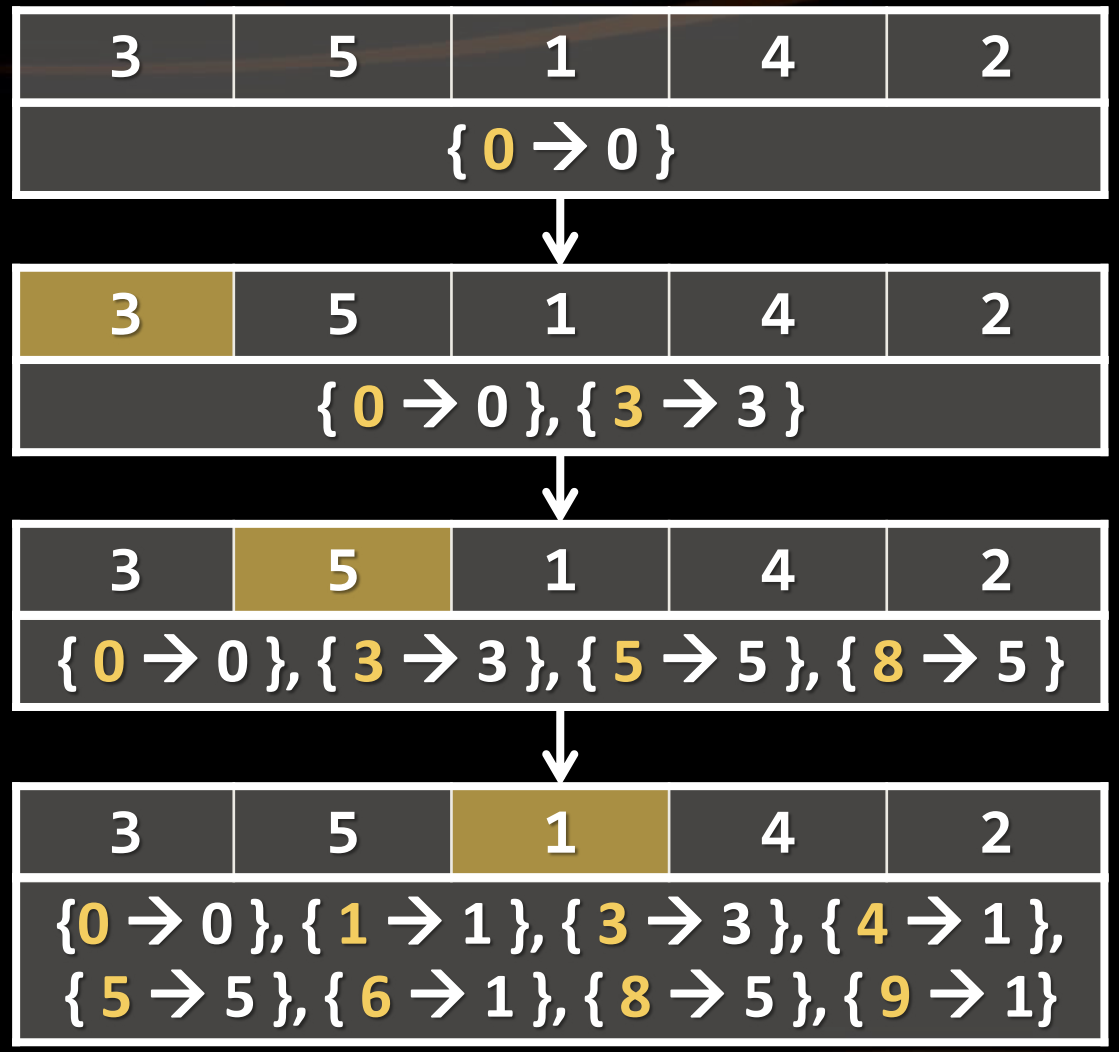
```
static ISet<int> CalcPossibleSumsSet(int[] nums, int targetSum)
{
  var possibleSums = new HashSet<int>() { 0 };
  for (int i = 0; i < nums.Length; i++)
  {
    var newSums = new HashSet<int>();
    foreach (var sum in possibleSums)
    {
      int newSum = sum + nums[i];
      newSums.Add(newSum);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



Subset Sum: How to Recover the Subset?

SoftUni Foundation

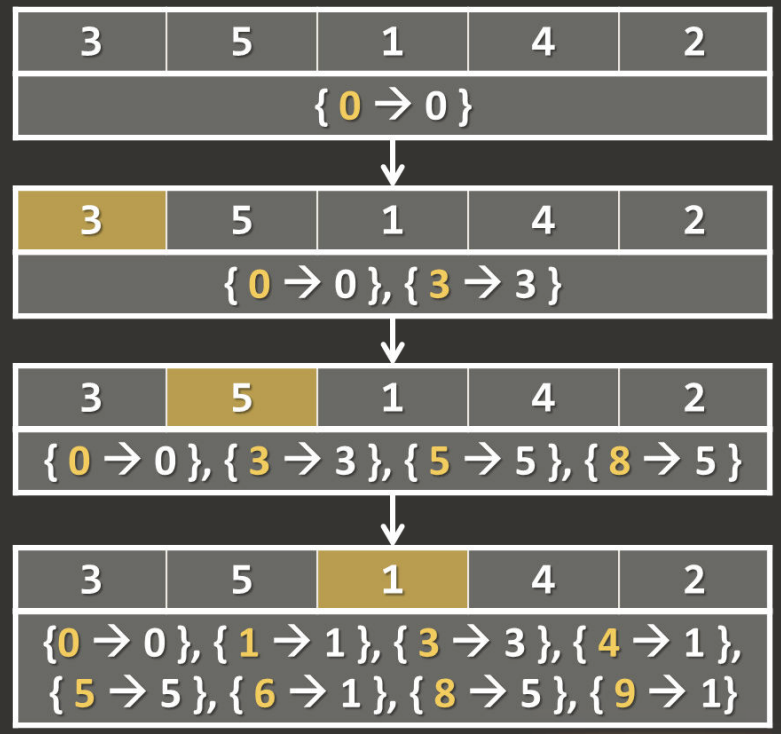
- Keep for each obtained sum in **possibleSums** how it is obtained
- Use a dictionary instead of set:
  - **possibleSums[*s*]** → ***num***
  - The sum ***s*** is obtained by adding ***num*** to some previously obtained subset sum
    - ***s*** – ***num*** gives us the previous sum



Subset Sum (No Repeats + Subset Recovery)

SoftUni Foundation

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



Subset Sum (No Repeats): Subset Recovery

SoftUni Foundation

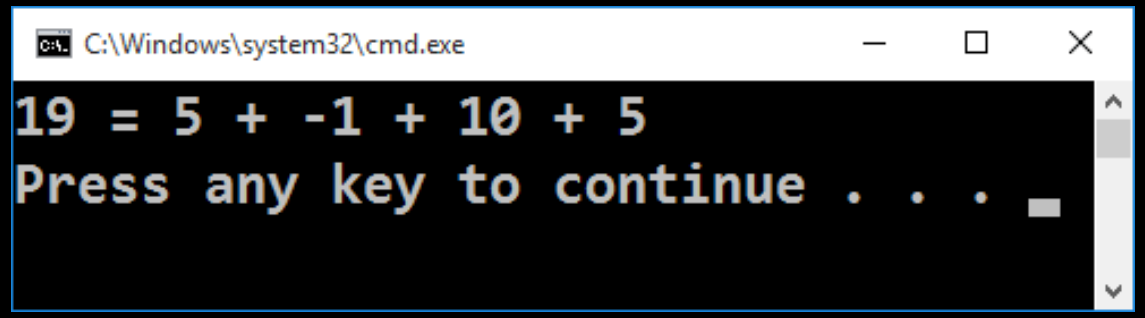
```
static List<int> FindSubset(int[] nums, int targetSum,
  IDictionary<int, int> possibleSums)
{
  var subset = new List<int>();
  while (targetSum > 0)
  {
    var lastNum = possibleSums[targetSum];
    subset.Add(lastNum);
    targetSum -= lastNum;
  }

  subset.Reverse();

  return subset;
}
```

{ 9 → 1} → { 8 → 5 } → { 3 → 3 } → 0

SoftUni Foundation

```
int[] nums = { 3, 5, -1, 10, 5, 7 };
int targetSum = 19;

var possibleSums = CalcPossibleSums(nums, targetSum);
```

```
var subset = FindSubset(nums, targetSum, possibleSums);
Console.Write(targetSum + " = ");
Console.WriteLine(String.Join(" + ", subset));
```



Subset Sum (with No Repeats)

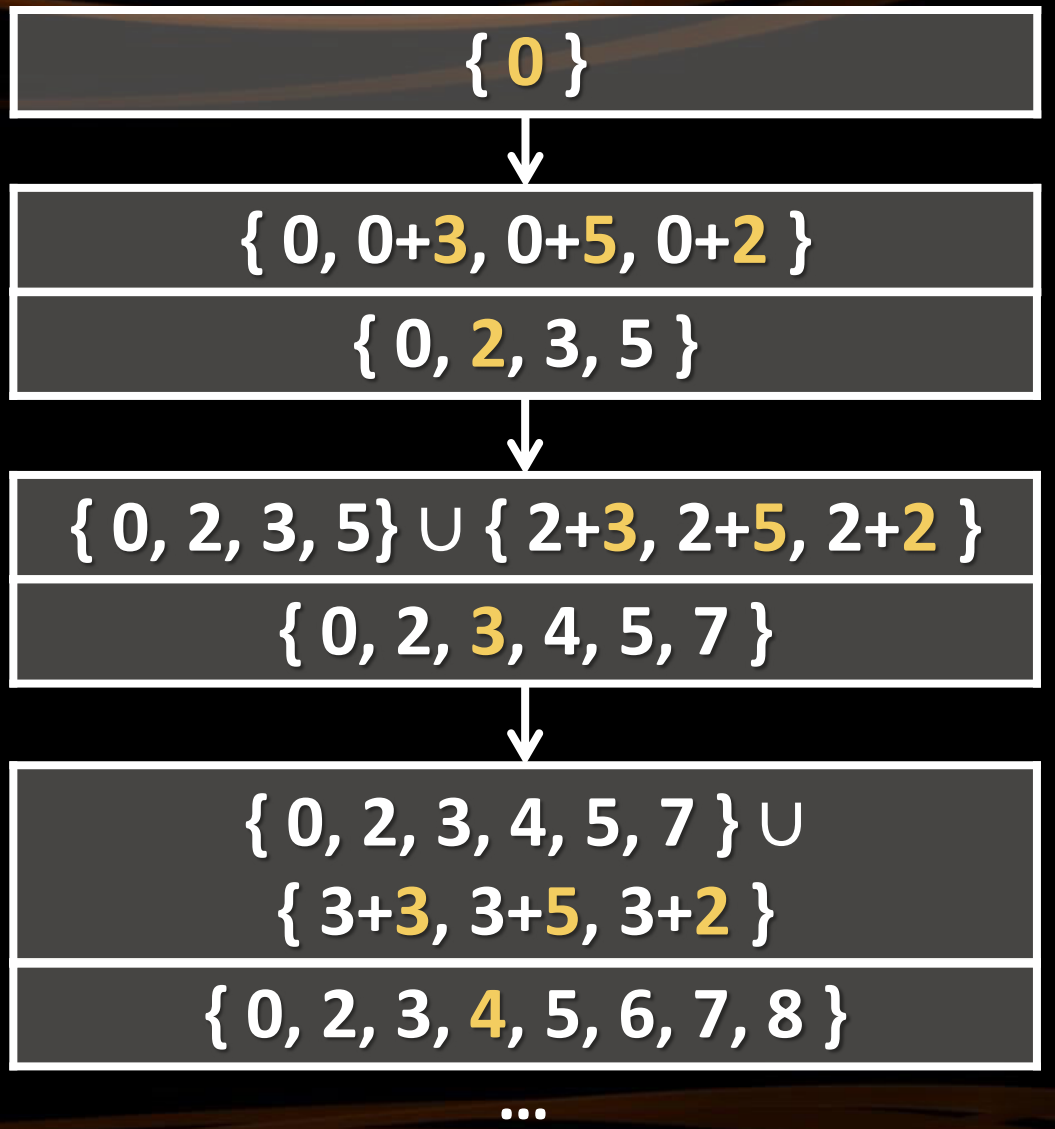
Live Demo

Subset Sum Problem (With Repeats)

SoftUni Foundation

- Solving the subset sum problem (BFS-based algorithm):
  - **nums** = { 3, 5, 2 }, **targetSum** = 6

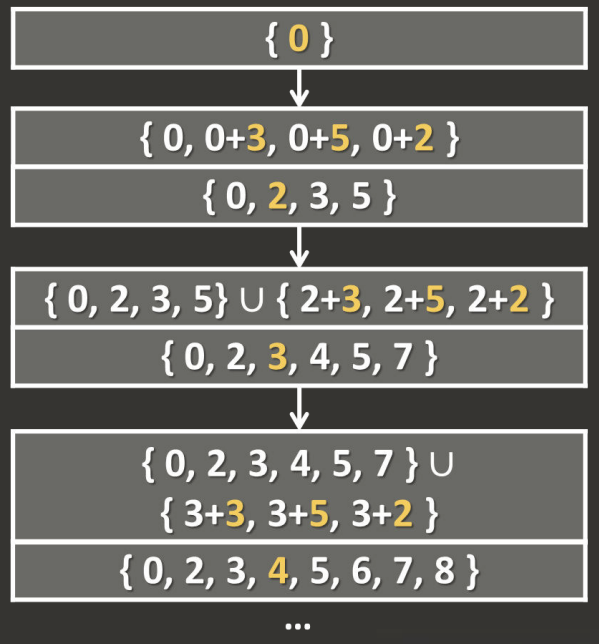
```
possibleSums = { 0 };
for i = 0 … targetSum
{
    Append to possibleSums[i]
    all numbers from nums[]
}
```



Subset Sum (With Repeats)

SoftUni Foundation

```
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])
      for (int i = 0; i < nums.Length; i++)
      {
        int newSum = sum + nums[i];
        if (newSum <= targetSum)
          possible[newSum] = true;
      }
  return possible;
}
```



Subset Sum (With Repeats): Recovery

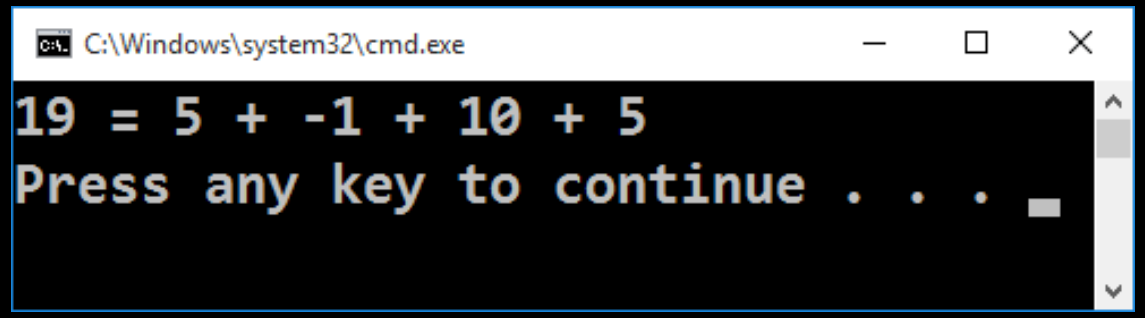
SoftUni Foundation

```
private static IEnumerable<int> FindSubset(
  int[] nums, int targetSum, bool[] possibleSums)
{
  var subset = new List<int>();
  while (targetSum > 0)
    for (int i = 0; i < nums.Length; i++)
    {
      int newSum = targetSum - nums[i];
      if (newSum >= 0 && possibleSums[newSum])
      {
        targetSum = newSum;
        subset.Add(nums[i]);
      }
    }
  return subset;
}
```

SoftUni Foundation

```
int[] nums = { 3, 5, -1, 10, 5, 7 };
int targetSum = 19;

var possibleSums = CalcPossibleSums(nums, targetSum);
```

```
var subset = FindSubset(nums, targetSum, possibleSums);
Console.Write(targetSum + " = ");
Console.WriteLine(String.Join(" + ", subset));
```



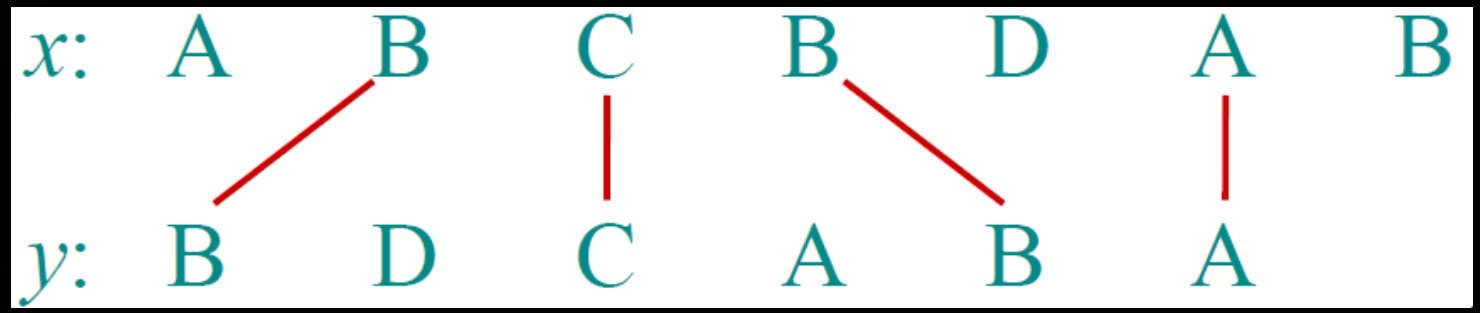
Subset Sum (With Repeats)

Live Demo

SoftUni Foundation

		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1
C	0	1	2	2	2	2	2	2	2	2
G	0	1	2	2	2	2	2	3	3	3
C	0	1	2	3	3	3	3	3	4	4
A	0	1	2	3	3	3	4	4	4	4
A	0	1	2	3	3	3	4	4	4	4
T	0	1	2	3	3	4	4	4	4	4
G	0	1	2	3	3	4	4	5	5	5

		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1
C	0	1	2	2	2	2	2	2	2	2
G	0	1	2	2	2	2	2	3	3	3
C	0	1	2	3	3	3	3	3	4	4
A	0	1	2	3	3	3	4	4	4	4
A	0	1	2	3	3	3	4	4	4	4
T	0	1	2	3	3	4	4	4	4	4
G	0	1	2	3	3	4	4	5	5	5



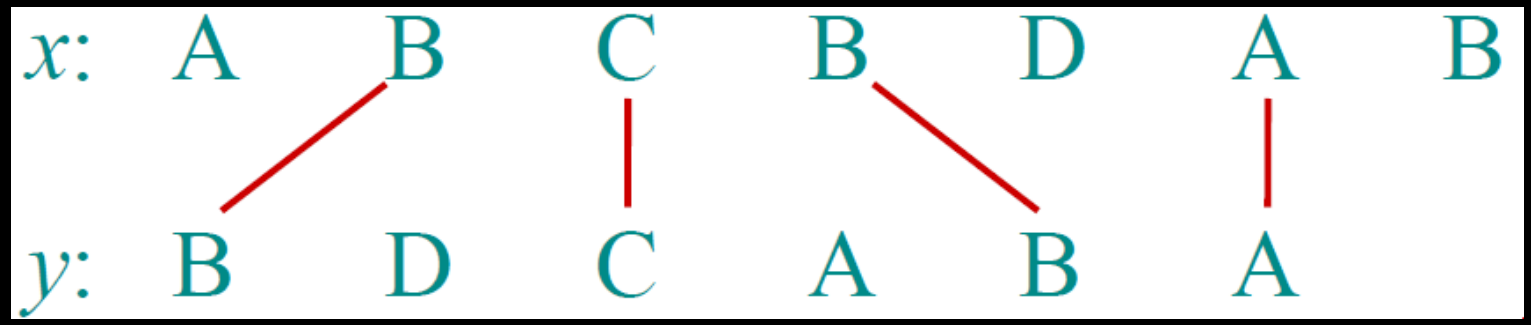
Longest Common Subsequence (LCS)

A Recursive DP Approach

Longest Common Subsequence (LCS)

SoftUni Foundation

- 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**C**A**BA**"
  - LCS = "**BCBA**"



LCS – Recursive Approach

SoftUni Foundation

- 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

SoftUni Foundation

- Let `lcs[x, y]` be the longest common subsequence of $S_1[0 \dots x]$ and $S_2[0 \dots y]$
- 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]
)
```

		G	C	C	C	T	A	G	C	G
	(0)	0	0	0	0	0	0	0	0	0
G	0	(1)	(1)	(1)	1	1	1	1	1	1
C	0	1	2	2	(2)	(2)	(2)	2	2	2
G	0	1	2	2	2	2	2	(3)	3	3
C	0	1	2	3	3	3	3	3	(4)	4
A	0	1	2	3	3	3	4	4	(4)	4
A	0	1	2	3	3	3	4	4	(4)	4
T	0	1	2	3	3	4	4	4	(4)	4
G	0	1	2	3	3	4	4	5	5	(5)

Calculating the LCS Table

SoftUni Foundation

```
static int CalcLCS(int x, int y) {
  if (x < 0 || y < 0)
    return 0;
  if (lcs[x, y] == NOT_CALCULATED)  {
    int lcsFirstMinusOne = CalcLCS(x - 1, y);
    int lcsSecondMinusOne = CalcLCS(x, y - 1);
    lcs[x, y] = Math.Max(
      lcsFirstMinusOne, lcsSecondMinusOne);
    if (firstStr[x] == secondStr[y]) {
      int lcsBothMinusOne = 1 + CalcLCS(x - 1, y - 1);
      lcs[x, y] = Math.Max(lcs[x, y], lcsBothMinusOne);
    }
  }
  return lcs[x, y];
}
```

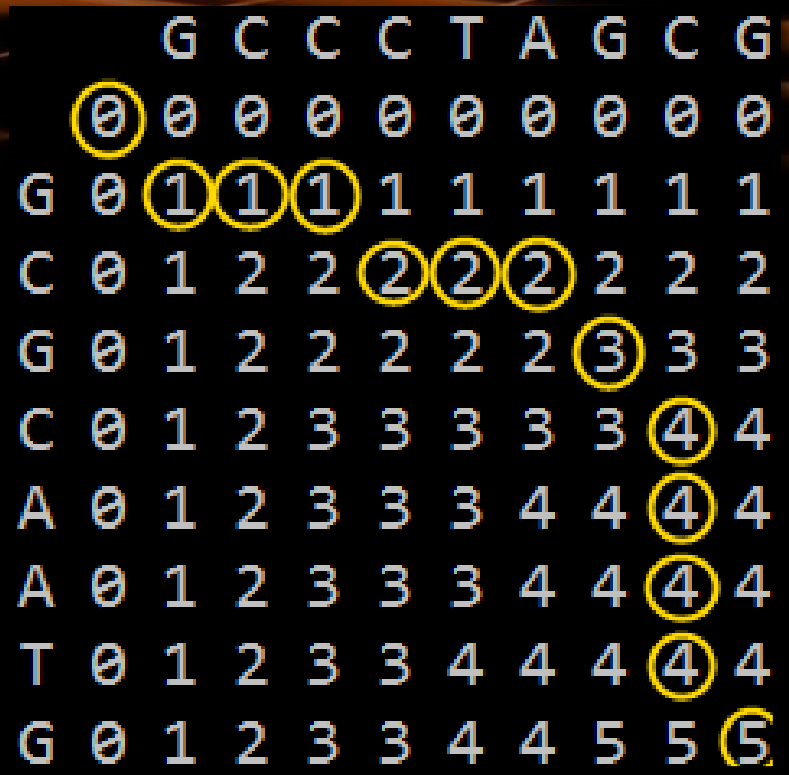
		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1
C	0	1	2	2	2	2	2	2	2	2
G	0	1	2	2	2	2	2	3	3	3
C	0	1	2	3	3	3	3	3	4	4
A	0	1	2	3	3	3	4	4	4	4
A	0	1	2	3	3	3	4	4	4	4
T	0	1	2	3	3	4	4	4	4	4
G	0	1	2	3	3	4	4	5	5	5

Reconstructing the LCS Sequence

SoftUni Foundation

```
static string PrintLCS(int x, int y) {
  var lcsLetters = new List<char>();
  while (x >= 0 && y >= 0)
    if ((firstStr[x] == secondStr[y]) &&
      (CalcLCS(x - 1, y - 1) + 1 == lcs[x, y])) {
      lcsLetters.Add(firstStr[x]);
      x--;
      y--;
    }
    else if (CalcLCS(x - 1, y) == lcs[x, y])
      x--;
    else
      y--;
  lcsLetters.Reverse();
  return string.Join("", lcsLetters);
}
```

		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1
C	0	1	2	2	2	2	2	2	2	2
G	0	1	2	2	2	2	2	3	3	3
C	0	1	2	3	3	3	3	3	4	4
A	0	1	2	3	3	3	4	4	4	4
A	0	1	2	3	3	3	4	4	4	4
T	0	1	2	3	3	4	4	4	4	4
G	0	1	2	3	3	4	4	5	5	5



Longest Common Subsequence (LCS)

Live Coding (Lab)

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
- **Top down** approach – **Recursive**
- **Bottom up** approach – **Iterative**

Dynamic Programming

SoftUni Foundation

Questions?

https://softuni.bg/opencourses/algorithms

License

SoftUni Foundation

- This course (slides, examples, labs, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Fundamentals of Computer Programming with C#" book by Svetlin Nakov & Co. under CC-BY-SA license
  - "Data Structures and Algorithms" course by Telerik Academy under CC-BY-NC-SA license

Trainings @ Software University (SoftUni)

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

Software University

SoftUni Foundation

FORUM