

Apply algorithm design and analysis methods



Presenters: Huy Nguyen Dinh, Ngan Tran Kim Ngoc

Contents of the presentation



01

Review

02

**Computational
Thinking**

03

Experiments



01

Review



Review



Quiz

- Everyone will answer 10 questions. Your group's score will be the total score of the two of you



Active

- Every time our group asks a question, after thinking and discussing it. Top 3 will get 5, 4, 3 points respectively. The remaining groups get 1 point.

Prize Pool



Plus 4 points



Plus 5 points



Plus 3 points

READY FOR A

QUIZ?

Code: 350 752

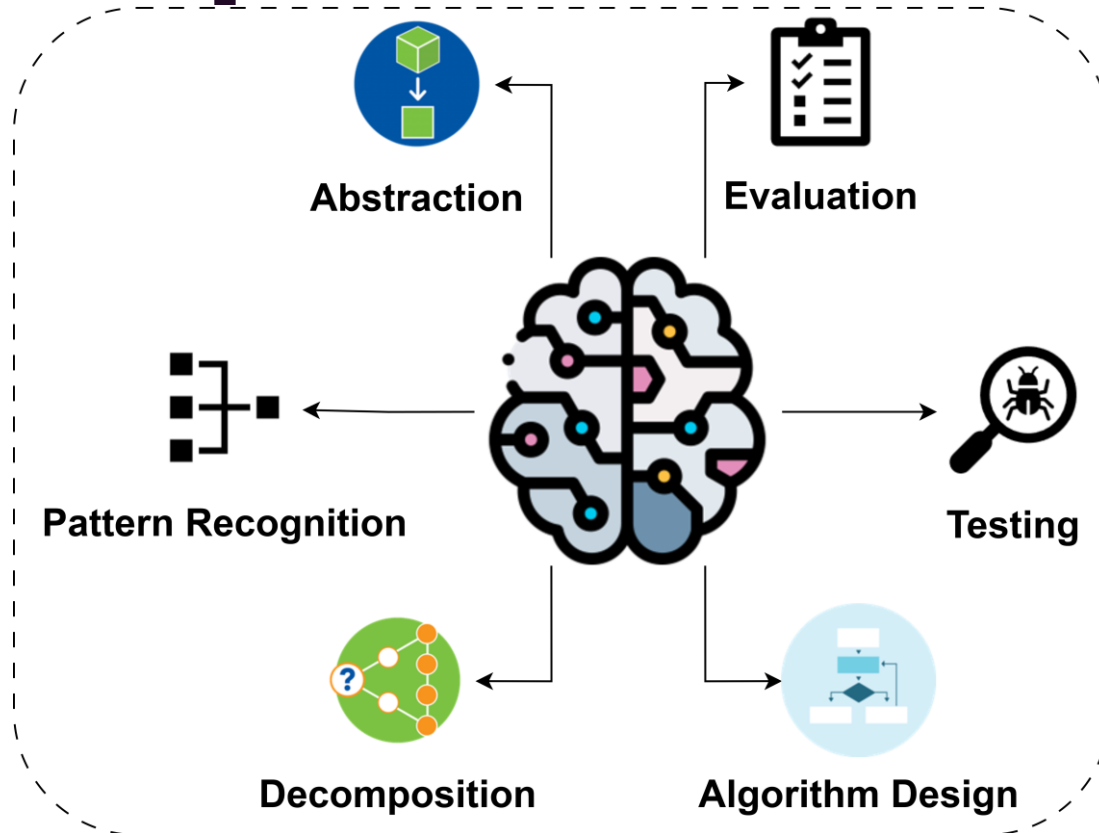
Nickname: **Gx_YourName**

02

Computational Thinking



Computational Thinking



Abstraction

Abstraction is "identifying and extracting relevant information to define main idea(s)".

The key to abstraction is to be able to identify and filter out or ignore the details not necessary to solve the problem

____?____ + ____?____ = ____?____

Abstraction

She eats Pie

Subject (person or thing)	+	Action/ Occurrence/ State of Being	+	Object (person or thing)
---------------------------------	---	--	---	--------------------------------

Noun	+	Verb	+	Noun
------	---	------	---	------

Abstraction

Listening English test

First you take your icicle then add a layer of dinosaur
before you pour on a hearty dose of maple syrup.

Next, press some tulips down into the noun before
covering with a sprinkle of noun.

That's how we make a noun !

Decompositional



"If you can't solve a problem, then there is an easier problem you can solve: find it."

George Pólya

Smaller subproblem + Smaller subproblem + Smaller subproblem =

**Big
Complex
Problem**

Example

- Take the example of criminal investigation by the police, how can the police apprehend criminals?.



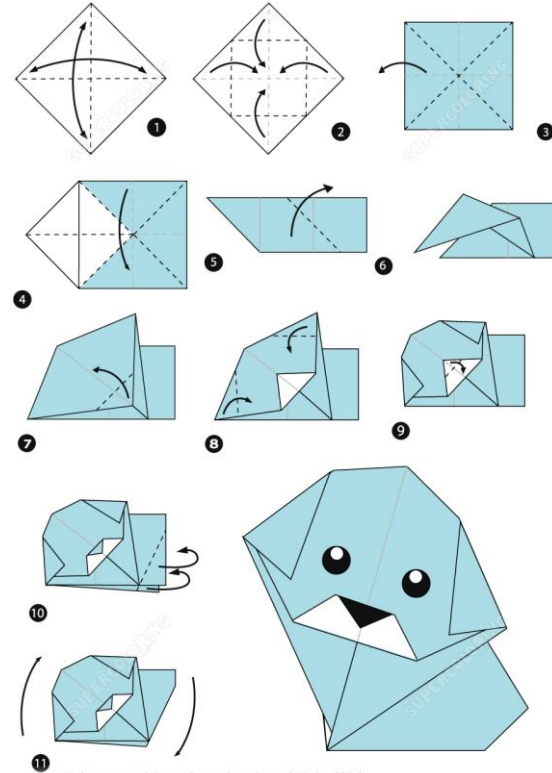
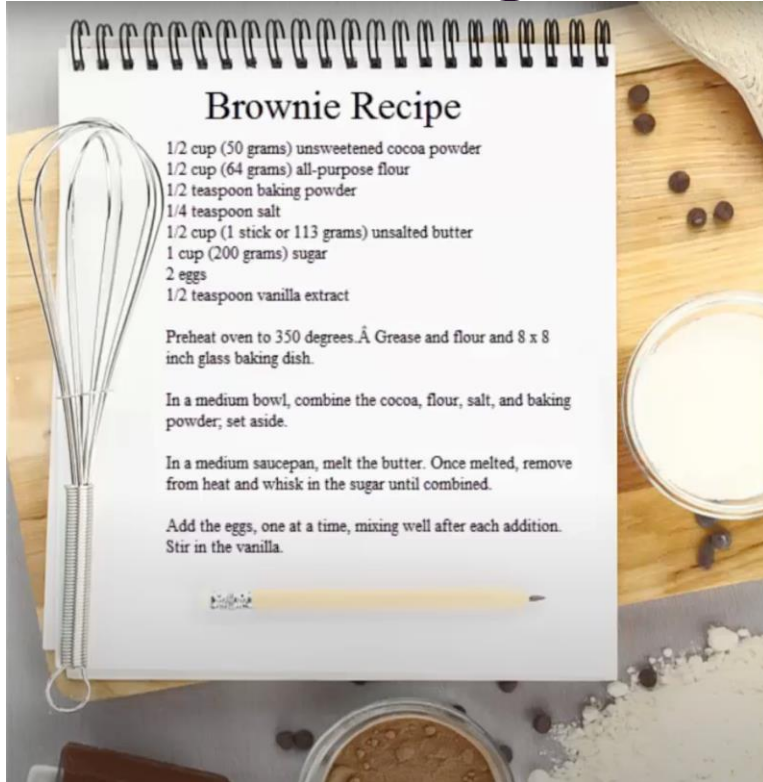
Smaller subproblem + Smaller subproblem + Smaller subproblem =

**Big
Complex
Problem**

Decomposition



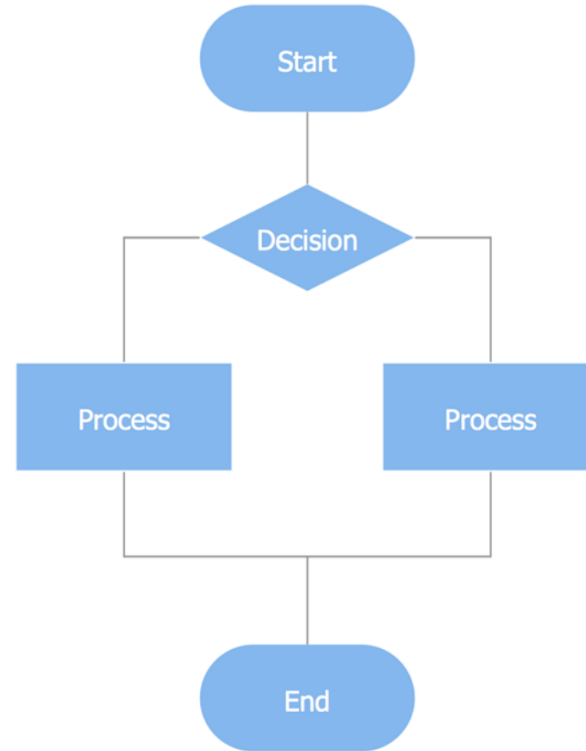
Algorithm Design



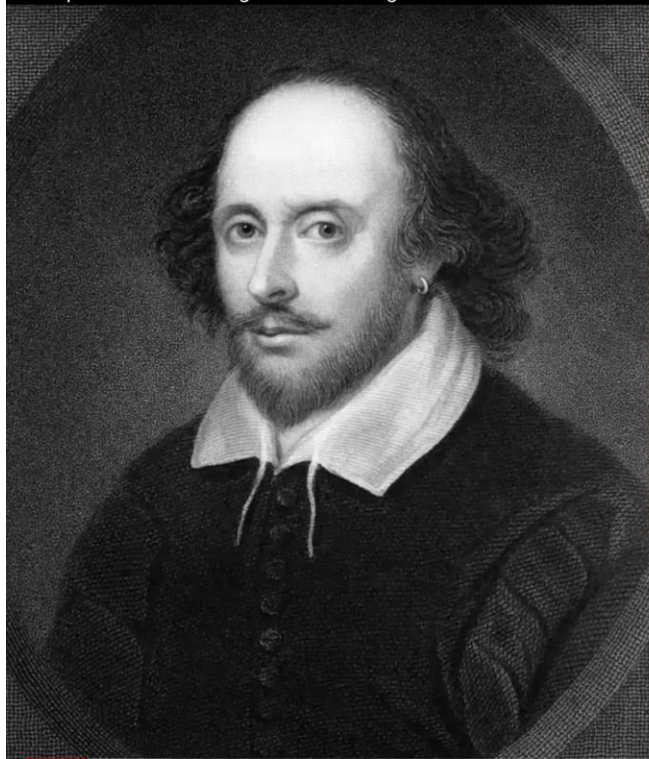
Algorithmic

Algorithmic thinking is about

- Planning,
- Detailing each step
- Creating a process
- Creating a flowchart,...

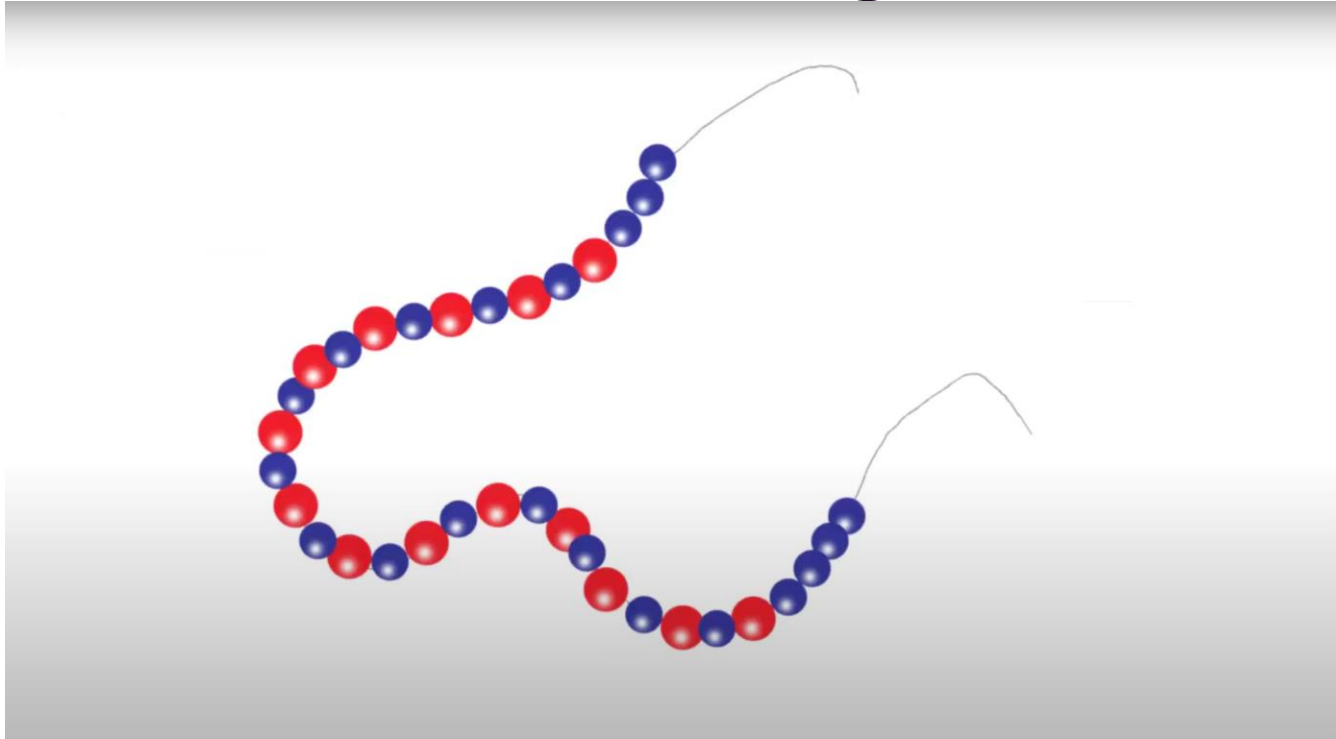


Pattern Recognition



Shall I / com **PARE**/ thee **TO** / a **SUM** / mer's **DAY**?
Thou **ART** / more **LOVE** / ly **AND** / more **TEM** / per **ATE**

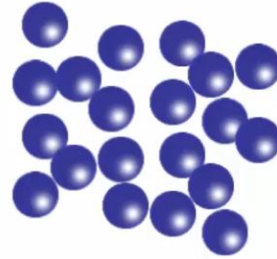
Pattern Recognition



Pattern Recognition



Red beads
\$0.33 each



Blue beads
\$0.22 each

Thread \$0.14
per inch

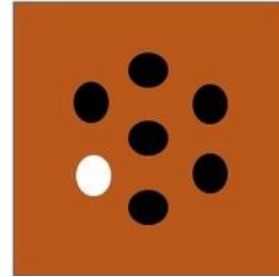
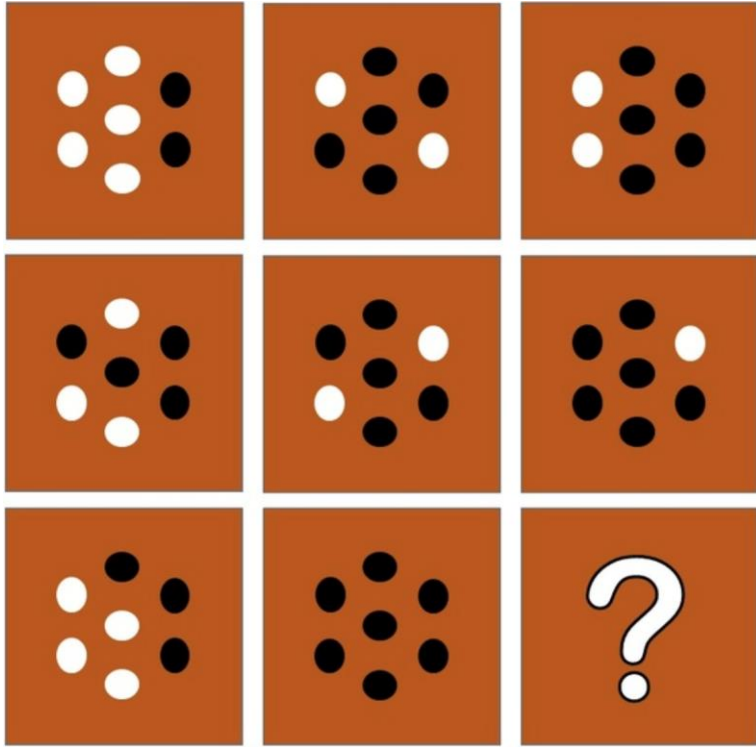


Pattern Recognition

a	. —	k	— . —	u	. . —
b	— . . .	l	. — . .	v	. . . —
c	— . — .	m	— —	w	. — —
d	— . .	n	— .	x	— . . —
e	. —	o	— — —	y	— . — —
f	. . — .	p	. — — .	z	— — . .
g	— — .	q	— — . —	ch	— — — —
h	r	. — .	â æ	. — . —
i	. . — —	s	. . .	ö œ	— — — .
j	. — — —	t	—	ü	. . — —

1	. — — — —	6	—
2	. . — — —	7	— — . . .
3	. . . — —	8	— — — . .
4 —	9	— — — — .
5	10	— — — — —

Pattern Recognition



03

Experiments



Problem 1



Problem 1

In an escape puzzle game, you are trapped in a locked room with a keyhole. Inside the room, there is only a table, a piece of paper, a pen, and sequences of numbers and characters written all over the room. On the wall, you discover an important clue for your escape:

3

6 3 1 2 5

The password is 4

To escape, you need to write the password on the piece of paper. As a super sharp detective, you immediately recognize the pattern of the characters. The first line contains only one number, N , and the second line contains $2N - 1$ positive integers. The password is the smallest number that, when added to the sequence, allows for at least one way to pair the numbers so that the sum of each pair is equal.



Problem 1

In the given example, there exists $[6, 3, 1, 2, 5]$. You can separate it into $[6, 1]$, $[5, 2]$, $[3, x]$. You need to fill in the number 4 so that $6+1 = 5+2 = 3+4$. If you cannot find such a number, write -1 on the piece of paper, and you will be set free.

Input format:

$$1 \leq K \leq 70.$$

$$1 \leq N \leq 3 * 10^5.$$

$$1 \leq A_i \leq 10^9 (A_i \text{ is any number}).$$



Solution

- **Abstraction**

To find the smallest number 'a' ($a \geq 1$) to add to an array of size $2N - 1$ such that the sum of the array ($S/2N$) is a positive integer

- **Decomposition & Pattern Recognition**

We divide it into three cases: adding at the beginning of the array, in the middle of the array, and at the end of the array.

$S1 = a[0] + a[n-2]$ if inserted at the beginning of the array,

$S2 = a[0] + a[n-1]$ if inserted in the middle of the array,

$S3 = a[1] + a[n-1]$ if inserted at the end of the array.

Solution

- **Algorithm Design**

Sort the array

Sort the cases S_i (S_1, S_2, S_3) in ascending order

Perform the loop

Taking (S_i - the corresponding array element) = Result.

- **Evaluation**

To sort the array with a time complexity of $O(n \log n)$ and perform the $O(n)$ loop, resulting in a total time complexity of $O(n \log n)$

Solution

```
def find_password(N, nums):
    if (N==1):
        return 1
    nums.sort()
    K_candidates = set()
    # Case 1: Remove the first element
    K1 = nums[0] + nums[-2]
    K_candidates.add(K1)
    # Case 2: Remove the last element
    K2 = nums[-1] + nums[1]
    K_candidates.add(K2)
    # Case 3: Remove an element in the middle
    K3 = nums[0] + nums[-1]
    K_candidates.add(K3)

    #Sort Answer
    for K in sorted(K_candidates):
        i = 0
        j = N - 1
        skips = 0
        mark = 0
        while i < j:
            if nums[i] + nums[j] < K:
                mark = i
                skips += 1
                i += 1
            elif nums[i] + nums[j] > K:
                mark = j
                skips += 1
                j -= 1
            else:
                i += 1
                j -= 1

        if (skips <= 1 ):
            if (skips == 0 and K-nums[i]>0):
                return(K-nums[i])
            if (K-nums[mark]>0):
                return(K-nums[mark])

    return -1
```

Problem 2 – Cashier Problem

Problem 1: Coin Problems: Minimum.

Given a set of coin values coins

$\{c_1, c_2, \dots, c_k\}$ and a target sum of money m ,
what's the minimum number of coins that
form the sum m ?

Constraints :

$$1 \leq k \leq 10$$

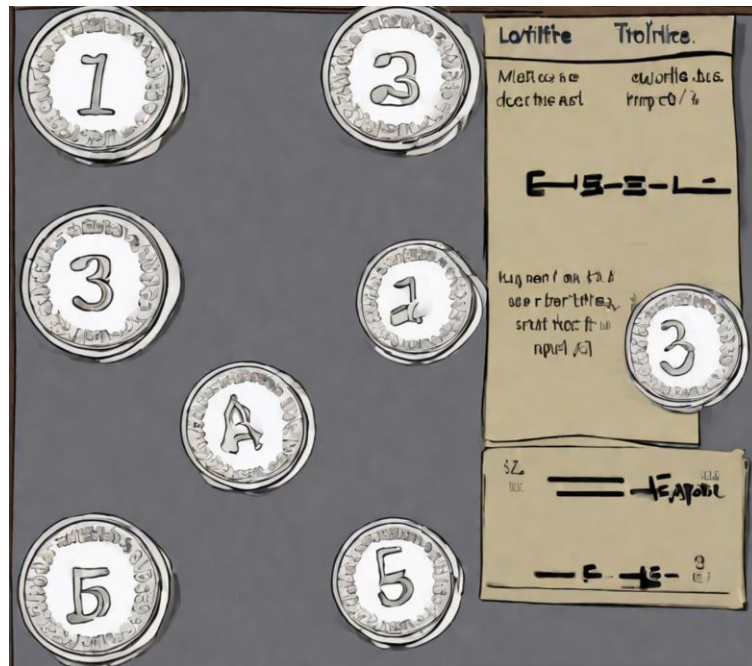
$$1 \leq m \leq 10^5$$

$$1 \leq c_i \leq 500$$

Example:

Coins = $\{200, 100, 50, 20, 10, 5, 2, 1\}$

Target: 734



Problem 2

❑ Greedy Solution: $3\{200\} + 1\{100\} + 1\{20\} + 1\{10\} + 2\{2\}$. Sum=8 .

❑ Recursion Solution:

We will try calling `minimum_coins(coins, m)` as the function that returns the desired result.

`minimum_coins(coins, 0) = 0` is the base case.

`minimum_coins(coins, m) = ?` is the problem that needs to be solved.

Recursion Solution

We will solve the subproblem

Coins = {1, 4, 5} and target = 13.

First, let's consider the brute force approach:
With the brute force approach, we can choose any coin.

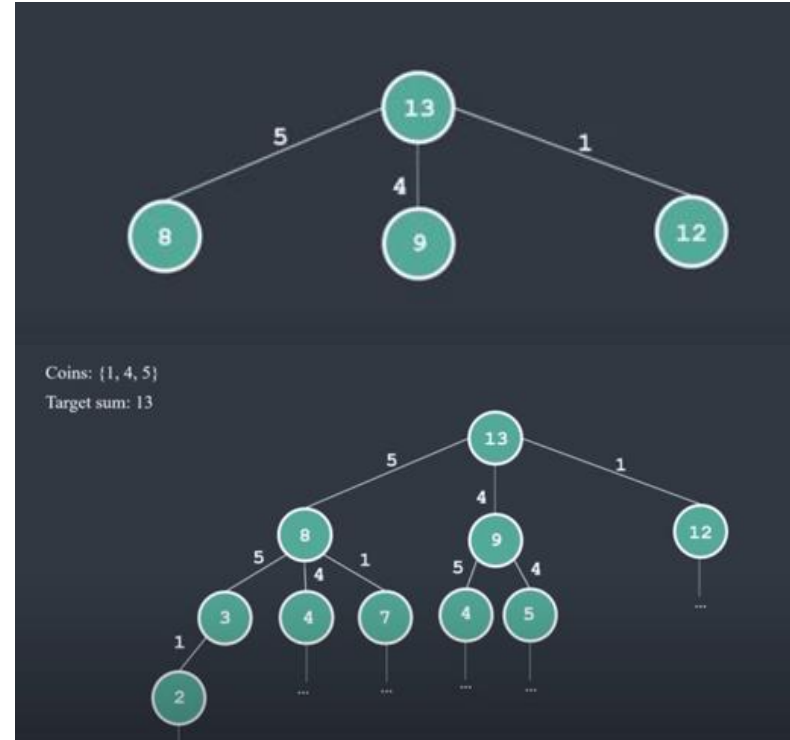
If we choose 5, we will need to solve the subproblem with

Coins = {1, 4, 5} and target = 8.

If we choose 4, we will need to solve the subproblem with

Coins = {1, 4, 5} and target = 9.

If we choose 1, we will need to solve the same problem with target = 12.



Solution

1. Abstraction

Minimum_coins(coins,m) = ?

2. Decomposition

Minimum_coins(coins,0) = 0

Minimum_coins(coins,x) = ? ($0 \leq x \leq m$)

3. Pattern Recognition

Recursion solution

Solution

4. Algorithm Design

```
def min_ignore_none(a, b):  
    if a is None:  
        return b  
    if b is None:  
        return a  
    return min(a, b)  
  
def minimum_coins(m, coins):  
    if m == 0:  
        answer = 0  
    else:  
        answer = None  
        for coin in coins:  
            subproblem = m - coin  
            if subproblem < 0:  
                # Skip solutions where we try to reach [m]  
                # from a negative subproblem.  
                continue  
            answer = min_ignore_none(  
                answer,  
                minimum_coins(subproblem, coins) + 1)  
    return answer  
  
print(minimum_coins(13, [1, 4, 5]))  
Output: 3
```

Solution

5. Evaluation : $O(k^m)$

6. Update ?

```
print(minimum_coins(150, [1, 4, 5]))
Output: 30

memo = {}

def minimum_coins(m, coins):
    if m in memo:
        return memo[m]

    if m == 0:
        answer = 0
    else:
        answer = None
        for coin in coins:
            subproblem = m - coin
            if subproblem < 0:
                # Skip solutions where we try to reach [m]
                # from a negative subproblem.
                continue
            answer = min_ignore_none(
                answer,
                minimum_coins(subproblem, coins) + 1)
        memo[m] = answer
    return answer

print(minimum_coins(13, [1, 4, 5]))
Output: 3

print(minimum_coins(150, [1, 4, 5]))
Output: 30
```

ReSolution

1. Abstraction

Minimum_coins(coins,m) = ?

2. Decomposition

Minimum_coins(coins,0) = 0 Minimum_coins(coins,x) = ?
($0 \leq x \leq m$)

3. Pattern Recognition

Dynamic Programming

ReSolution

4. Algorithms Design

5. Evaluation $O(K * M)$

```
def minimum_coins(m, coins):  
    memo = {}  
  
    memo[0] = 0  
    for i in range(1, m + 1):  
        for coin in coins:  
            subproblem = i - coin  
            if subproblem < 0:  
                continue  
  
            memo[i] = min_ignore_none(memo.get(i), memo.get(subproblem) + 1)  
  
    return memo[m]
```

Problem 3

Given a set of coin values coins
 $\{c_1, c_2, \dots, c_k\}$ and a target sum of money m .
But how many ways can we form the
sum

m using these coins?

Coins: $\{1, 4, 5\}$

Target sum: 5

There are 4 ways in total:

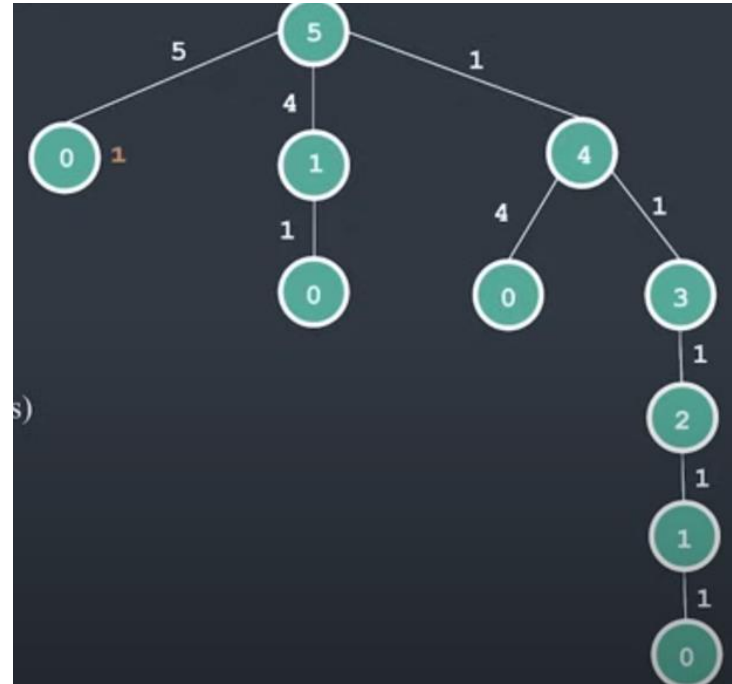
$1+1+1+1+1$ $1+4$ $4+1$ 5

Constraints :

$1 \leq k \leq 10$ (number of coins)

$1 \leq m \leq 10^5$

$1 \leq c_i \leq 500$



Solution

Given a set of coin values $\text{coins}=\{c_1, c_2, \dots, c_k\}$ and a target sum of money m , in how many ways can we form the sum m using these coins?

```
from collections import defaultdict
```

```
def how_many_ways(m, coins):  
    memo = defaultdict(lambda _: 0)
```

```
    memo[0] = 1  
    for i in range(1, m + 1):  
        memo[i] = 0
```

```
        for coin in coins:  
            subproblem = i - coin  
            if subproblem < 0:  
                continue
```

```
            memo[i] = memo[i] + memo[subproblem]
```

```
    return memo[m]
```

```
print(how_many_ways(5, [1, 4, 5]))
```

Output: 4

```
print(how_many_ways(87, [1, 4, 5, 8]))
```

Output: 3306149332861088

$$\text{solution}(0) = 1$$

$$\text{solution}(m) = \sum_{c \in \text{coins}} \text{solution}(m - c)$$

*Thank
You*

