



Dhole Patil College of Engineering, Pune

Savitribai Phule Pune University (SPPU)
Fourth Year of Computer Engineering (2019 Course)

410246: Laboratory Practice III

Subject Teacher: - Prof. Suchitra Deokate(DAA)
Dr. Aarti Dandavate(ML)
Prof. Archana Priyadarshani(BT)

Term work: 50 Marks
Practical: 50 Marks
Design and Analysis of Algorithms (410241)
Machine Learning(410242)
Blockchain Technology(410243)

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group A

Assignment No: 1

Title of the Assignment: Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

Objective of the Assignment: Students should be able to perform non-recursive and recursive programs to calculate Fibonacci numbers and analyze their time and space complexity.

Prerequisite:

1. Basic of Python or Java Programming
2. Concept of Recursive and Non-recursive functions
3. Execution flow of calculate Fibonacci numbers
4. Basic of Time and Space complexity

Contents for Theory:

1. Introduction to Fibonacci numbers
2. Time and Space complexity

Introduction to Fibonacci numbers

- The Fibonacci series, named after Italian mathematician Leonardo Pisano Bogollo, later known as Fibonacci, is a series (sum) formed by Fibonacci numbers denoted as F_n . The numbers in Fibonacci sequence are given as: 0, 1, 1, 2, 3, 5, 8, 13, 21, 38, . . .
- In a Fibonacci series, every term is the sum of the preceding two terms, starting from 0 and 1 as first and second terms. In some old references, the term '0' might be omitted.

What is the Fibonacci Series?

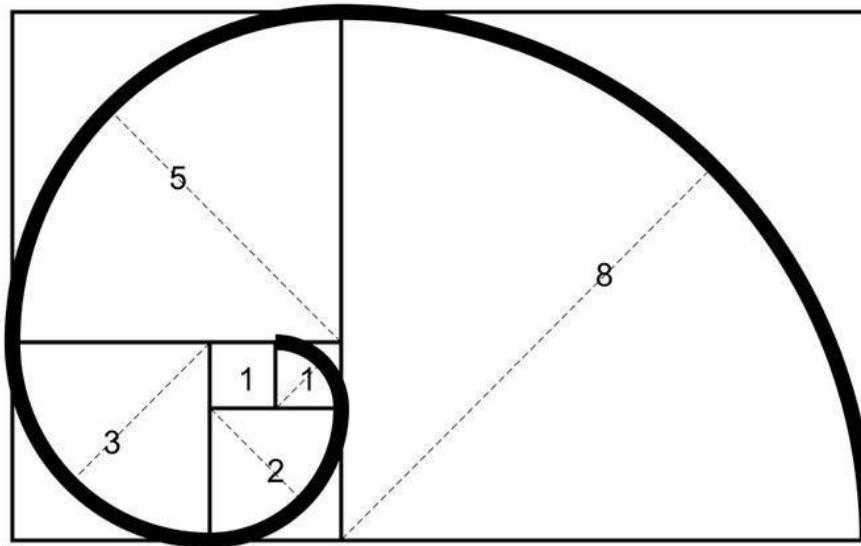
- The Fibonacci series is the sequence of numbers (also called Fibonacci numbers), where every number is the sum of the preceding two numbers, such that the first two terms are '0' and '1'.
- In some older versions of the series, the term '0' might be omitted. A Fibonacci series can thus be given as, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . It can be thus be observed that every term can be calculated by adding the two terms before it.
- Given the first term, F_0 and second term, F_1 as '0' and '1', the third term here can be given as, $F_2 = 0 + 1 = 1$

Similarly,

$$F_3 = 1 + 1 = 2$$

$$F_4 = 2 + 1 = 3$$

Given a number n , print n -th Fibonacci Number.



Fibonacci Sequence Formula

The Fibonacci sequence of numbers “Fn” is defined using the recursive relation with the seed values $F_0=0$ and $F_1=1$:

$$F_n = F_{n-1} + F_{n-2}$$

Here, the sequence is defined using two different parts, such as kick-off and recursive relation.

The kick-off part is $F_0=0$ and $F_1=1$.

The recursive relation part is $F_n = F_{n-1} + F_{n-2}$.

It is noted that the sequence starts with 0 rather than 1. So, F_5 should be the 6th term of the sequence.

Examples:

Input : $n = 2$

Output : 1

Input : $n = 9$

Output : 34

The list of Fibonacci numbers are calculated as follows:

F_n	Fibonacci Number
0	0
1	1
2	1
3	2
4	3
5	5

6	8
7	13
8	21
9	34
... and so on.	... and so on.

Method 1 (Use Non-recursion)

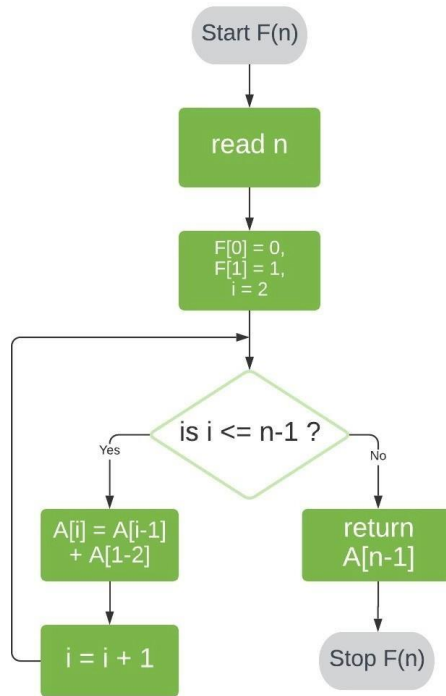
A simple method that is a direct recursive implementation of mathematical recurrence relation is given above.

First, we'll store 0 and 1 in $F[0]$ and $F[1]$, respectively.

Next, we'll iterate through array positions 2 to $n-1$. At each position i , we store the sum of the two preceding array values in $F[i]$.

Finally, we return the value of $F[n-1]$, giving us the number at position n in the sequence.

Here's a visual representation of this process:



Program to display the Fibonacci sequence up to n-th term

```
nterms = int(input("How many terms? "))
```

first two terms

```
n1, n2 = 0, 1
```

```
count = 0
```

check if the number of terms is valid

```
if nterms <= 0:
```

```
    print("Please enter a positive integer")
```

if there is only one term, return n1

```
elif nterms == 1:
```

```
    print("Fibonacci sequence upto", nterms, ":")
```

```
    print(n1)
```

generate fibonacci sequence

else:

```
print("Fibonacci sequence:")
```

```
while count < nterms:
```

```
    print(n1)
```

```
    nth = n1 + n2
```

```
    # update values
```

```
    n1 = n2
```

```
    n2 = nth
```

```
    count += 1
```

Output

How many terms? 7

Fibonacci sequence:

0

1

1

2

3

5

8

Time and Space Complexity of Space Optimized Method

- The time complexity of the Fibonacci series is **$T(N)$ i.e, linear**. We have to find the sum of two terms and it is repeated n times depending on the value of n .
- The space complexity of the Fibonacci series using dynamic programming is **$O(1)$** .

Time Complexity and Space Complexity of Dynamic Programming

- The time complexity of the above code is **$T(N)$ i.e, linear**. We have to find the sum of two terms and it is repeated n times depending on the value of n .

- The space complexity of the above code is $O(N)$.

Method 2 (Use Recursion)

Let's start by defining $F(n)$ as the function that returns the value of F_n .

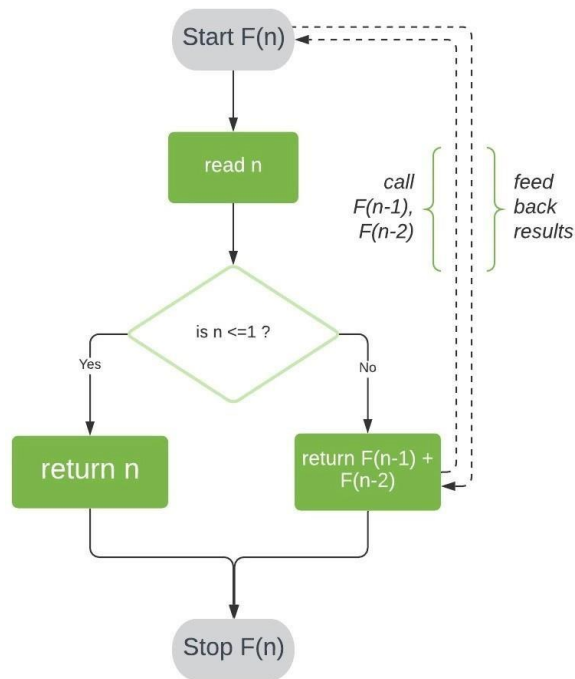
To evaluate $F(n)$ for $n > 1$, we can reduce our problem into two smaller problems of the same kind: $F(n-1)$ and $F(n-2)$. We can further reduce $F(n-1)$ and $F(n-2)$ to $F((n-1)-1)$ and $F((n-1)-2)$; and $F((n-2)-1)$ and $F((n-2)-2)$, respectively.

If we repeat this reduction, we'll eventually reach our known base cases and, thereby, obtain a solution to $F(n)$.

Employing this logic, our algorithm for $F(n)$ will have two steps:

1. Check if $n \leq 1$. If so, return n .
2. Check if $n > 1$. If so, call our function F with inputs $n-1$ and $n-2$, and return the sum of the two results.

Here's a visual representation of this algorithm:



Python program to display the Fibonacci sequence

```
def recur_fibo(n):  
    if n <= 1:  
        return n  
    else:  
        return(recur_fibo(n-1) + recur_fibo(n-2))  
  
nterms = 7  
  
# check if the number of terms is valid  
  
if nterms <= 0:  
    print("Plese enter a positive integer")  
else:  
    print("Fibonacci sequence:")  
    for i in range(nterms):  
        print(recur_fibo(i))
```

Output

Fibonacci sequence:

0
1
1
2
3
5
8

Time and Space Complexity

- The time complexity of the above code is $T(2^N)$ i.e, exponential.

- The Space complexity of the above code is **O(N) for a recursive series.**

Method	Time complexity	Space complexity
Using recursion	$T(n) = T(n-1) + T(n-2)$	O(n)
Using DP	O(n)	O(1)
Space optimization of DP	O(n)	O(1)
Using the power of matrix method	O(n)	O(1)
Optimized matrix method	O(log n)	O(log n)
Recursive method in O(log n) time	O(log n)	O(n)
Using direct formula	O(log n)	O(1)
DP using memoization	O(n)	O(1)

Applications of Fibonacci Series

The Fibonacci series finds application in different fields in our day-to-day lives. The different patterns found in a varied number of fields from nature, to music, and to the human body follow the Fibonacci series. Some of the applications of the series are given as,

- It is used in the grouping of numbers and used to study different other special mathematical sequences.
- It finds application in Coding (computer algorithms, distributed systems, etc). For example, Fibonacci series are important in the computational run-time analysis of Euclid's algorithm, used for determining the GCF of two integers.
- It is applied in numerous fields of science like quantum mechanics, cryptography, etc.
- In finance market trading, Fibonacci retracement levels are widely used in technical analysis.

Conclusion- In this way we have explored Concept of Fibonacci series using recursive and non recursive method and also learn time and space complexity

Assignment Question

1. What is the Fibonacci Sequence of numbers?
2. How do the Fibonacci work?
3. What is the Golden Ratio?
4. What is the Fibonacci Search technique?
5. What is the real application for Fibonacci series

Reference link

- <https://www.scaler.com/topics/fibonacci-series-in-c/>
- <https://www.baeldung.com/cs/fibonacci-computational-complexity>

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group A

Assignment No: 2

Title of the Assignment: Write a program to implement Huffman Encoding using a greedy strategy.

Objective of the Assignment: Students should be able to understand and solve Huffman Encoding using greedy method

Prerequisite:

1. Basic of Python or Java Programming
 2. Concept of Greedy method
 3. Huffman Encoding concept
-

Contents for Theory:

1. Greedy Method
 2. Huffman Encoding
 3. Example solved using huffman encoding
-

What is a Greedy Method?

- A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.
- The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.
- This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

Advantages of Greedy Approach

- The algorithm is **easier to describe**.
- This algorithm can **perform better** than other algorithms (but, not in all cases).

Drawback of Greedy Approach

- As mentioned earlier, the greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm
- For example, suppose we want to find the longest path in the graph below from root to leaf.

Greedy Algorithm

1. To begin with, the solution set (containing answers) is empty.
2. At each step, an item is added to the solution set until a solution is reached.
3. If the solution set is feasible, the current item is kept.
4. Else, the item is rejected and never considered again.

Huffman Encoding

- Huffman Coding is a technique of compressing data to reduce its size without losing any of the details. It was first developed by David Huffman.
- Huffman Coding is generally useful to compress the data in which there are frequently occurring

characters.

- Huffman Coding is a famous Greedy Algorithm.
- It is used for the lossless compression of data.
- It uses variable length encoding.
- It assigns variable length code to all the characters.
- The code length of a character depends on how frequently it occurs in the given text.
- The character which occurs most frequently gets the smallest code.
- The character which occurs least frequently gets the largest code.
- It is also known as **Huffman Encoding**.

Prefix Rule-

- Huffman Coding implements a rule known as a prefix rule.
- This is to prevent the ambiguities while decoding.
- It ensures that the code assigned to any character is not a prefix of the code assigned to any other character

Major Steps in Huffman Coding-

There are two major steps in Huffman Coding-

1. Building a Huffman Tree from the input characters.
2. Assigning code to the characters by traversing the Huffman Tree.

How does Huffman Coding work?

Suppose the string below is to be sent over a network.



- Each character occupies 8 bits. There are a total of 15 characters in the above string. Thus, a total of $8 * 15 = 120$ bits are required to send this string.
- Using the Huffman Coding technique, we can compress the string to a smaller size.

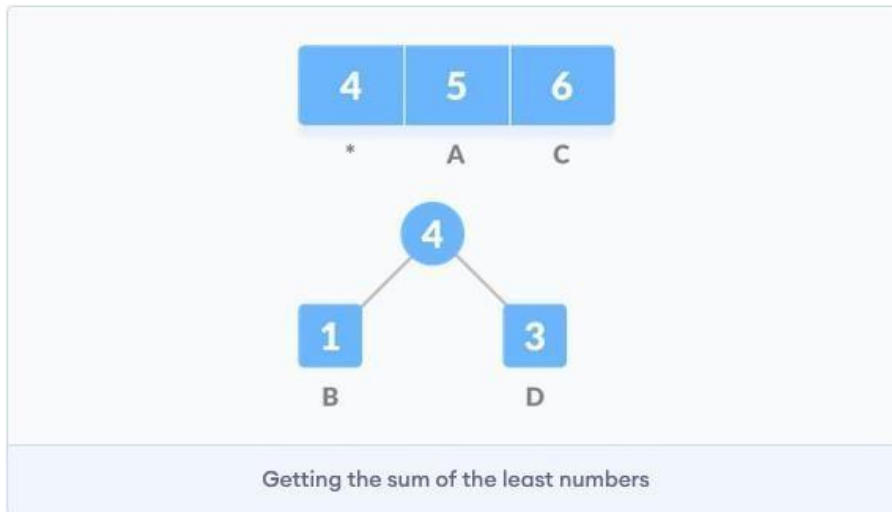
- Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.
 - Once the data is encoded, it has to be decoded. Decoding is done using the same tree.
 - Huffman Coding prevents any ambiguity in the decoding process using the concept of **prefix code** ie. a code associated with a character should not be present in the prefix of any other code. The tree created above helps in maintaining the property.
 - Huffman coding is done with the help of the following steps.
1. Calculate the frequency of each character in the string.



2. Sort the characters in increasing order of the frequency. These are stored in a priority queue Q.



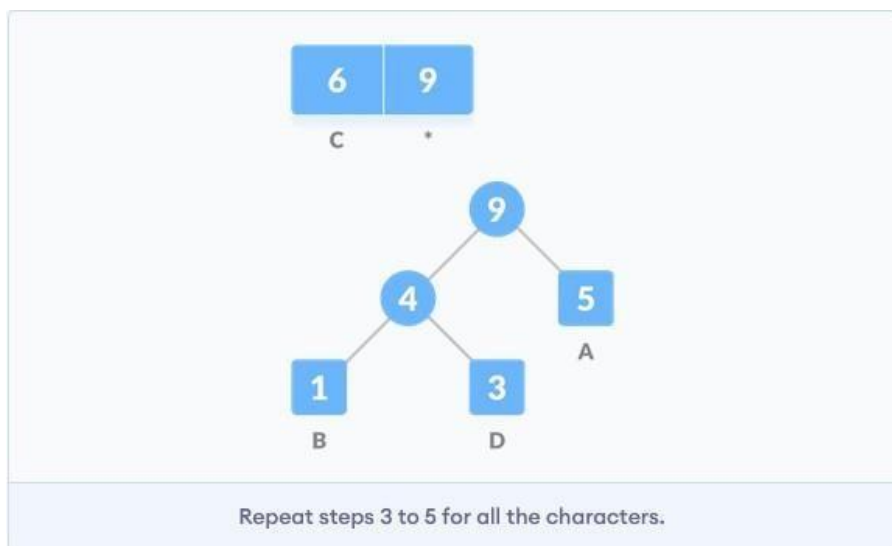
3. Make each unique character as a leaf node.
4. Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.

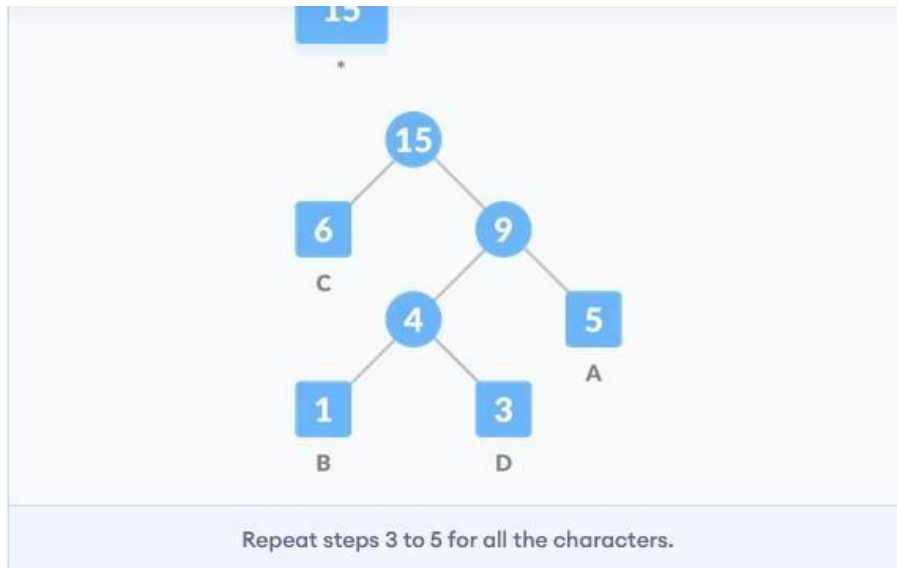


5. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (* denote the internal nodes in the figure above).

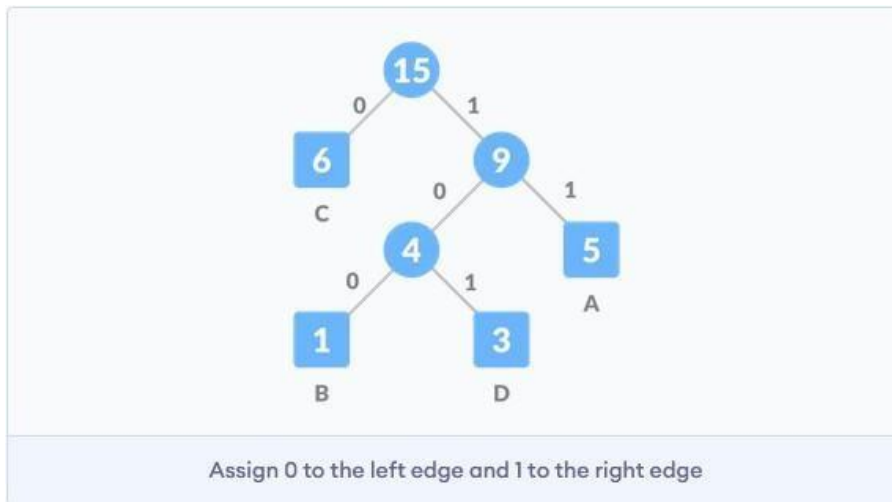
6. Insert node z into the tree.

7. Repeat steps 3 to 5 for all the characters.





8. For each non-leaf node, assign 0 to the left edge and 1 to the right edge



For sending the above string over a network, we have to send the tree as well as the above compressed-code. The total size is given by the table below.

Character	Frequency	Code	Size
A	5	11	$5 \times 2 = 10$
B	1	100	$1 \times 3 = 3$
C	6	0	$6 \times 1 = 6$
D	3	101	$3 \times 3 = 9$
$4 \times 8 = 32$ bits	15 bits		28 bits

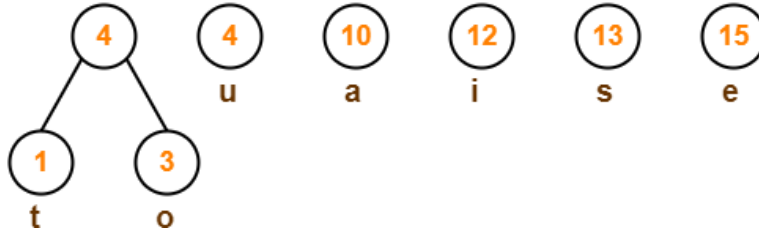
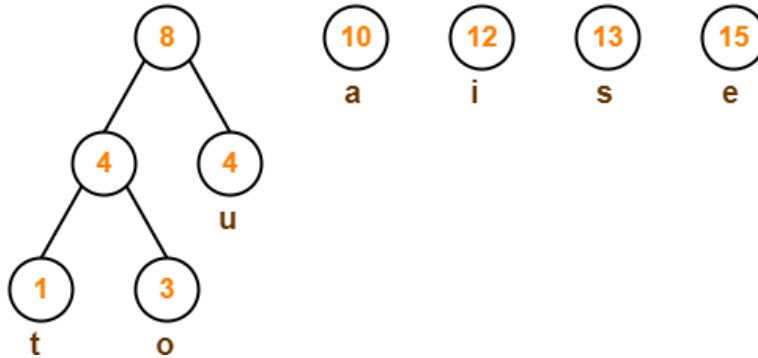
Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to $32 + 15 + 28 = 75$.

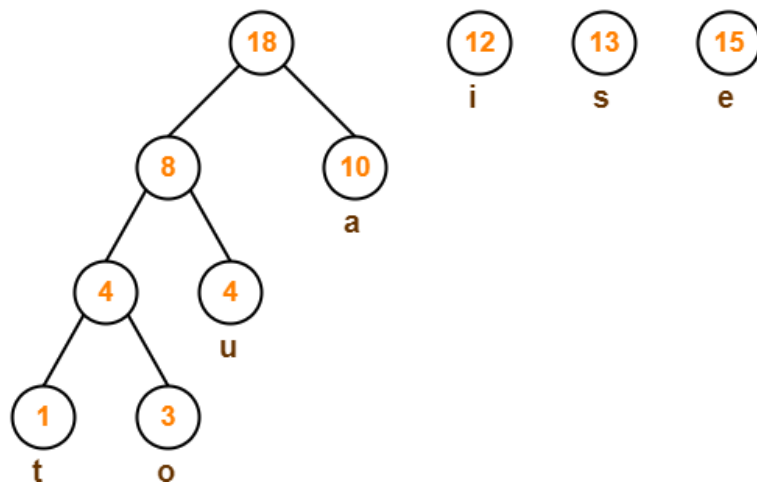
Example:

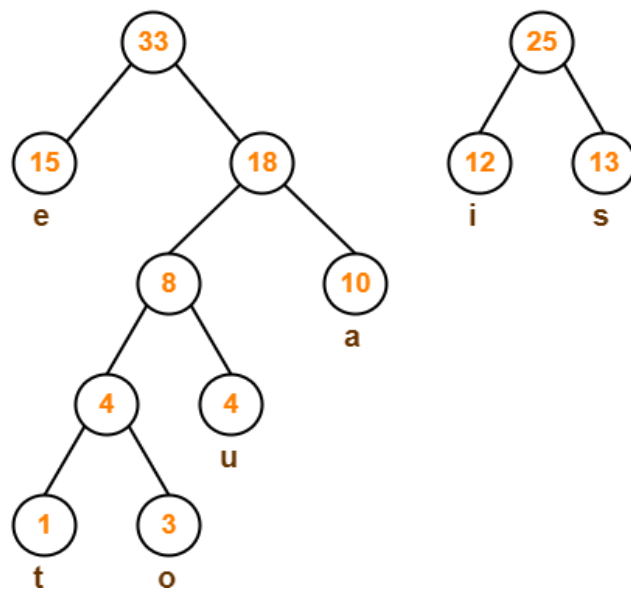
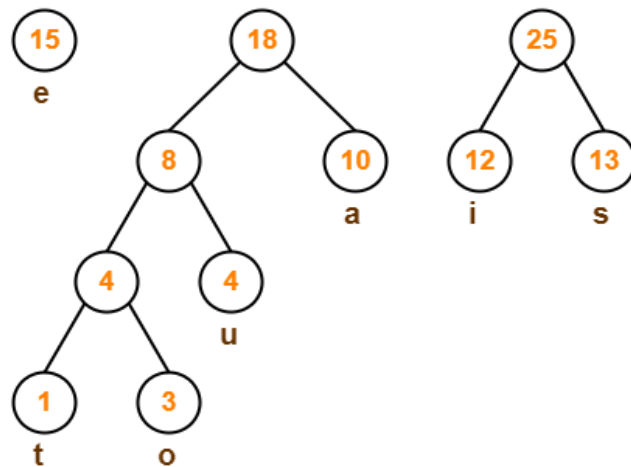
A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-

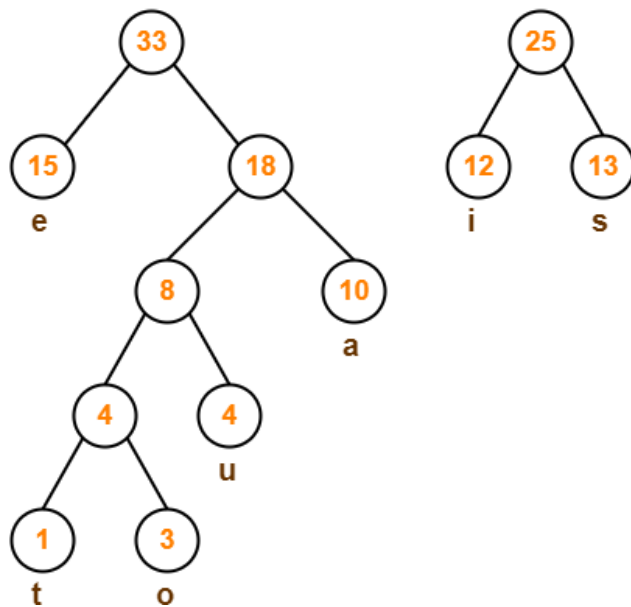
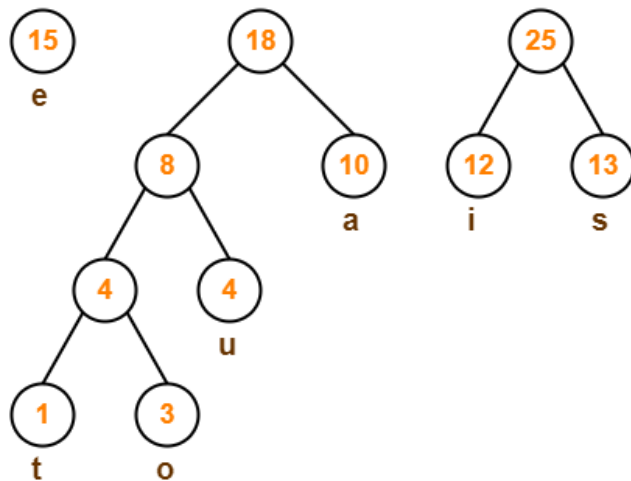
1. Huffman Code for each character
2. Average code length
3. Length of Huffman encoded message (in bits)

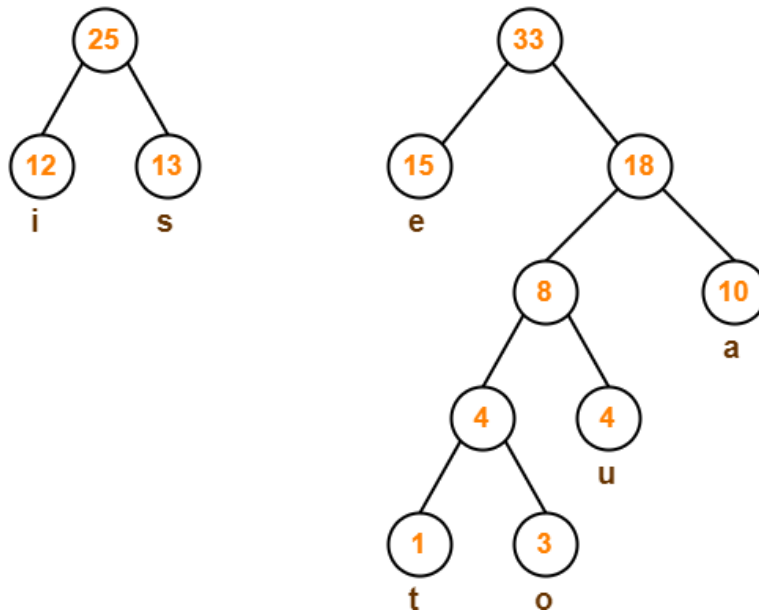
Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

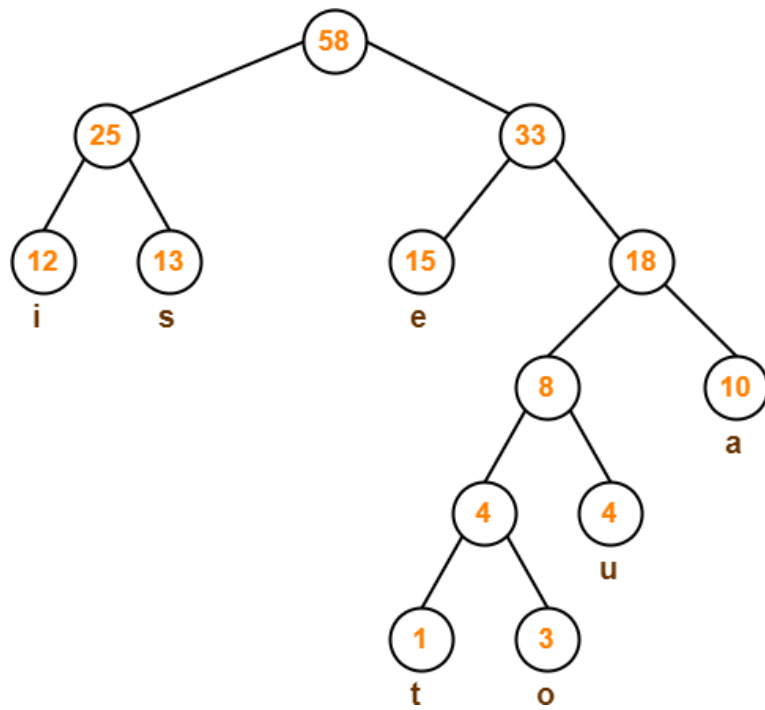
Step-01:**Step-02:****Step-03:**

Step-04:

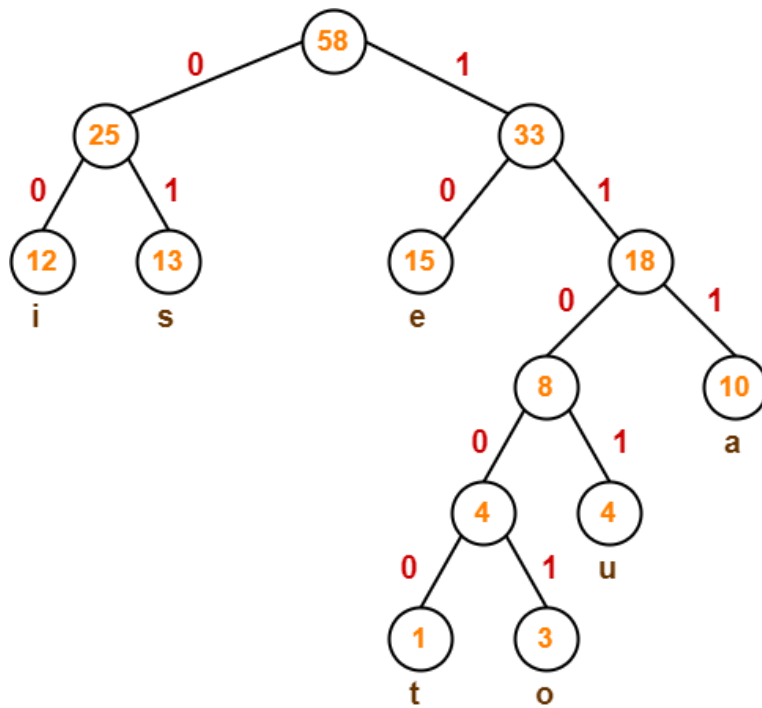
Step-06:

Step-06:

Step-07:



After assigning weight to all the edges, the modified Huffman Tree is-



Huffman Tree

To write Huffman Code for any character, traverse the Huffman Tree from root node to the leaf node of that character.

Following this rule, the Huffman Code for each character is-

a = 111

e = 10

i = 00

o = 11001

u = 1101

s = 01

t = 11000

Time Complexity-

The time complexity analysis of Huffman Coding is as follows-

- extractMin() is called $2 \times (n-1)$ times if there are n nodes.
- As extractMin() calls minHeapify(), it takes $O(\log n)$ time.

Thus, Overall time complexity of Huffman Coding becomes **$O(n \log n)$** .

Code :-

```
class Node:
    def __init__(self, prob, symbol, left=None, right=None):
        # probability of symbol
        self.prob = prob

        # symbol
        self.symbol = symbol

        # left node
        self.left = left

        # right node
        self.right = right

        # tree direction (0/1)
        self.code = ''

    """ A helper function to calculate the probabilities of symbols in given data """
    def Calculate_Probability(data):
        symbols = dict()
        for element in data:
            if symbols.get(element) == None:
                symbols[element] = 1
            else:
                symbols[element] += 1
        return symbols

    """ A helper function to obtain the encoded output """
    def Output_Encoded(data, coding):
        encoding_output = []
        for c in data:
            print(coding[c], end = '')
            encoding_output.append(coding[c])

        string = ''.join([str(item) for item in encoding_output])
        return string

    """ A helper function to calculate the space difference between compressed and non compressed """
    def Total_Gain(data, coding):
        before_compression = len(data) * 8 # total bit space to store the data before compression
        after_compression = 0
        symbols = coding.keys()
        for symbol in symbols:
            count = data.count(symbol)
            after_compression += count * len(coding[symbol]) # calculate how many bit is required for each symbol
        print("Space usage before compression (in bits):", before_compression)
        print("Space usage after compression (in bits):", after_compression)
```

```

def Huffman_Encoding(data):
    symbol_with_probs = Calculate_Probability(data)
    symbols = symbol_with_probs.keys()
    probabilities = symbol_with_probs.values()
    print("symbols: ", symbols)
    print("probabilities: ", probabilities)

    nodes = []

    # converting symbols and probabilities into huffman tree nodes
    for symbol in symbols:
        nodes.append(Node(symbol_with_probs.get(symbol), symbol))

    while len(nodes) > 1:
        # sort all the nodes in ascending order based on their probability
        nodes = sorted(nodes, key=lambda x: x.prob)
        # for node in nodes:
        #     print(node.symbol, node.prob)

        # pick 2 smallest nodes
        right = nodes[0]
        left = nodes[1]

        left.code = 0
        right.code = 1

        # combine the 2 smallest nodes to create new node
        newNode = Node(left.prob+right.prob, left.symbol+right.symbol, left, right)

        nodes.remove(left)
        nodes.remove(right)
        nodes.append(newNode)

    huffman_encoding = Calculate_Codes(nodes[0])
    print(huffman_encoding)
    Total_Gain(data, huffman_encoding)
    encoded_output = Output_Encoded(data, huffman_encoding)
    print("Encoded output:", encoded_output)
    return encoded_output, nodes[0]

```

Output

```

AAAAAABCCCCCDEEEEE
symbols: dict_keys(['A', 'B', 'C', 'D', 'E'])
probabilities: dict_values([7, 1, 6, 2, 5])
symbols with codes {'A': '00', 'C': '01', 'E': '10', 'D': '110', 'B': '111'}
Space usage before compression (in bits): 168
Space usage after compression (in bits): 45
Encoded output 0000000000000111010101010111011010101010

```

Conclusion- In this way we have explored Concept of Huffman Encoding using greedy method

Assignment Question

1. What is Huffman Encoding?
2. How many bits may be required for encoding the message 'mississippi'?
3. Which tree is used in Huffman encoding? Give one Example
4. Why Huffman coding is lossless compression?

Reference link

- <https://towardsdatascience.com/huffman-encoding-python-implementation-8448c3654328>
- <https://www.programiz.com/dsa/huffman-coding#cpp-code>
- <https://www.gatevidyalay.com/tag/huffman-coding-example-ppt/>

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group A

Assignment No: 3

Title of the Assignment: Write a program to solve a fractional Knapsack problem using a greedy method.

Objective of the Assignment: Students should be able to understand and solve fractional Knapsack problems using a greedy method.

Prerequisite:

1. Basic of Python or Java Programming
 2. Concept of Greedy method
 3. fractional Knapsack problem
-

Contents for Theory:

1. Greedy Method
 2. Fractional Knapsack problem
 3. Example solved using fractional Knapsack problem
-

What is a Greedy Method?

- A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.
- The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.
- This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

Advantages of Greedy Approach

- The algorithm is **easier to describe**.
- This algorithm can **perform better** than other algorithms (but, not in all cases).

Drawback of Greedy Approach

- As mentioned earlier, the greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm
- For example, suppose we want to find the longest path in the graph below from root to leaf.

Greedy Algorithm

1. To begin with, the solution set (containing answers) is empty.
2. At each step, an item is added to the solution set until a solution is reached.
3. If the solution set is feasible, the current item is kept.
4. Else, the item is rejected and never considered again.

Knapsack Problem

You are given the following-

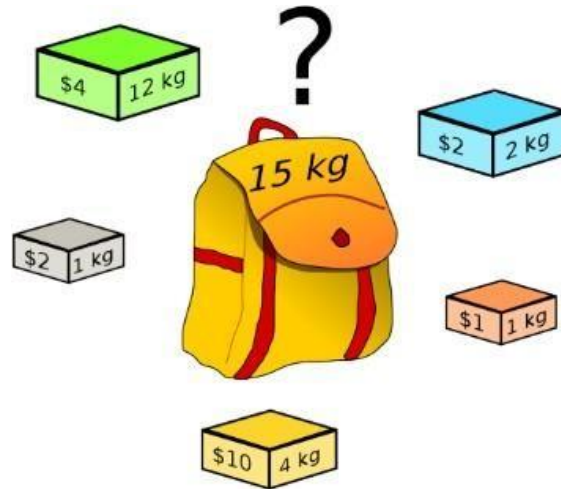
- A knapsack (kind of shoulder bag) with limited weight capacity.

- Few items each having some weight and value.

The problem states-

Which items should be placed into the knapsack such that-

- The value or profit obtained by putting the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed.



Knapsack Problem

Knapsack Problem Variants

Knapsack problem has the following two variants-

1. Fractional Knapsack Problem
2. 0/1 Knapsack Problem

Fractional Knapsack Problem-

In Fractional Knapsack Problem,

- As the name suggests, items are divisible here.
- We can even put the fraction of any item into the knapsack if taking the complete item is not

possible.

- It is solved using the Greedy Method.

Fractional Knapsack Problem Using Greedy Method-

Fractional knapsack problem is solved using greedy method in the following steps-

Step-01:

For each item, compute its value / weight ratio.

Step-02:

Arrange all the items in decreasing order of their value / weight ratio.

Step-03:

Start putting the items into the knapsack beginning from the item with the highest ratio.

Put as many items as you can into the knapsack.

Problem-

For the given set of items and knapsack capacity = 60 kg, find the optimal solution for the fractional knapsack problem making use of greedy approach.

Item	Weight	Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

$$n = 5$$

$$w = 60 \text{ kg}$$

$$(w_1, w_2, w_3, w_4, w_5) = (5, 10, 15, 22, 25)$$

$$(b_1, b_2, b_3, b_4, b_5) = (30, 40, 45, 77, 90)$$

Solution-**Step-01:**

Compute the value / weight ratio for each item-

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
3	15	45	3
4	22	77	3.5
5	25	90	3.6

Step-02:

Sort all the items in decreasing order of their value / weight ratio-

I1 I2 I5 I4 I3

(6) (4) (3.6) (3.5) (3)

Step-03:

Start filling the knapsack by putting the items into it one by one.

Knapsack Weight	Items in Knapsack	Cost
60	Ø	0
55	I1	30
45	I1, I2	70
20	I1, I2, I5	160

Now,

- Knapsack weight left to be filled is 20 kg but item-4 has a weight of 22 kg.
- Since in fractional knapsack problem, even the fraction of any item can be taken.
- So, knapsack will contain the following items-

< I1 , I2 , I5 , (20/22) I4 >

Total cost of the knapsack

$$= 160 + (20/22) \times 77$$

$$= 160 + 70$$

$$= 230 \text{ units}$$

Time Complexity-

- The main time taking step is the sorting of all items in decreasing order of their value / weight ratio.
- If the items are already arranged in the required order, then while loop takes $O(n)$ time.
- The average time complexity of Quick Sort is $O(n \log n)$.
- Therefore, total time taken including the sort is $O(n \log n)$.

Code:-

```
class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight

def fractionalKnapsack(W, arr):

    # Sorting Item on basis of ratio
    arr.sort(key=lambda x: (x.value/x.weight), reverse=True)

    # Result(value in Knapsack)
    finalvalue = 0.0

    # Looping through all Items
    for item in arr:

        # If adding Item won't overflow,
        # add it completely
        if item.weight <= W:
            W -= item.weight
            finalvalue += item.value

        # If we can't add current Item,
        # add fractional part of it
        else:
            finalvalue += item.value * W / item.weight
            break

    # Returning final value
    return finalvalue

# Driver Code
if __name__ == "__main__":

    W = 50
    arr = [Item(60, 10), Item(100, 20), Item(120, 30)]

    # Function call
    max_val = fractionalKnapsack(W, arr)
    print(max_val)
```

Output

Maximum value we can obtain = 24

Conclusion-In this way we have explored Concept of Fractional Knapsack using greedy method

Assignment Question

- 1. What is Greedy Approach?**
- 2. Explain concept of fractional knapsack**
- 3. Difference between Fractional and 0/1 Knapsack**
- 4. Solve one example based on Fractional knapsack(Other than Manual)**

Reference link

- <https://www.gatevidyalay.com/fractional-knapsack-problem-using-greedy-approach/>

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group A

Assignment No: 4

Title of the Assignment: Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

Objective of the Assignment: Students should be able to understand and solve 0-1 Knapsack problem using dynamic programming

Prerequisite:

1. Basic of Python or Java Programming
 2. Concept of Dynamic Programming
 3. 0/1 Knapsack problem
-

Contents for Theory:

1. Greedy Method
 2. 0/1 Knapsack problem
 3. Example solved using 0/1 Knapsack problem
-

What is Dynamic Programming?

- Dynamic Programming is also used in optimization problems. Like divide-and-conquer method, Dynamic Programming solves problems by combining the solutions of subproblems.
- Dynamic Programming algorithm solves each sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time.
- Two main properties of a problem suggest that the given problem can be solved using Dynamic Programming. These properties are **overlapping sub-problems and optimal substructure**.
- Dynamic Programming also combines solutions to sub-problems. It is mainly used where the solution of one sub-problem is needed repeatedly. The computed solutions are stored in a table, so that these don't have to be re-computed. Hence, this technique is needed where overlapping sub-problem exists.
- For example, Binary Search does not have overlapping sub-problem. Whereas recursive program of Fibonacci numbers have many overlapping sub-problems.

Steps of Dynamic Programming Approach

Dynamic Programming algorithm is designed using the following four steps –

- Characterize the structure of an optimal solution.
- Recursively define the value of an optimal solution.
- Compute the value of an optimal solution, typically in a bottom-up fashion.
- Construct an optimal solution from the computed information.

Applications of Dynamic Programming Approach

- Matrix Chain Multiplication
- Longest Common Subsequence
- Travelling Salesman Problem

Knapsack Problem

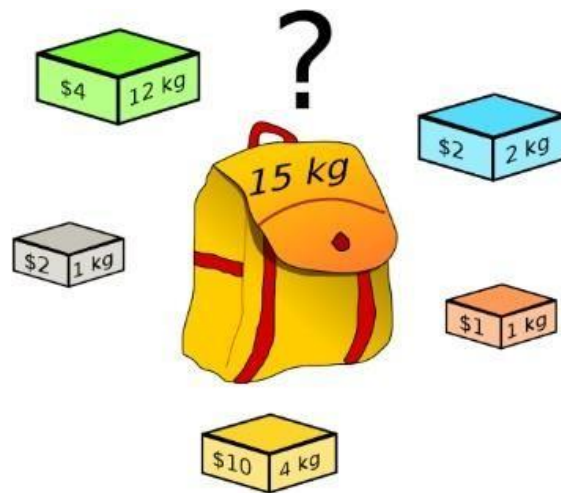
You are given the following-

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some weight and value.

The problem states-

Which items should be placed into the knapsack such that-

- The value or profit obtained by putting the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed.



Knapsack Problem

Knapsack Problem Variants

Knapsack problem has the following two variants-

1. Fractional Knapsack Problem
2. 0/1 Knapsack Problem

0/1 Knapsack Problem-

In 0/1 Knapsack Problem,

- As the name suggests, items are indivisible here.
- We can not take a fraction of any item.
- We have to either take an item completely or leave it completely.
- It is solved using a dynamic programming approach.

0/1 Knapsack Problem Using Greedy Method-

Consider-

- Knapsack weight capacity = w
- Number of items each having some weight and value = n

0/1 knapsack problem is solved using dynamic programming in the following steps-

Step-01:

- Draw a table say 'T' with $(n+1)$ number of rows and $(w+1)$ number of columns.
- Fill all the boxes of 0^{th} row and 0^{th} column with zeroes as shown-

	0	1	2	3	W
0	0	0	0	0	0
1	0					
2	0					
.....						
n	0					

T-Table

Step-02:

Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, j) = \max \{ T(i-1, j), \text{value}_i + T(i-1, j - \text{weight}_i) \}$$

Here, $T(i, j)$ = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.

- This step leads to completely filling the table.
- Then, value of the last box represents the maximum possible value that can be put into the knapsack.

Step-03:

- To identify the items that must be put into the knapsack to obtain that maximum profit,
- Consider the last column of the table.
- Start scanning the entries from bottom to top.
- On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
- After all the entries are scanned, the marked labels represent the items that must be put into the knapsack

Problem-.

For the given set of items and knapsack capacity = 5 kg, find the optimal solution for the 0/1 knapsack problem making use of a dynamic programming approach.

Item	Weight	Value
1	2	3
2	3	4
3	4	5
4	5	6

$$n = 4$$

$$w = 5 \text{ kg}$$

$$(w_1, w_2, w_3, w_4) = (2, 3, 4, 5)$$

$$(b_1, b_2, b_3, b_4) = (3, 4, 5, 6)$$

Solution-

Given

- Knapsack capacity (w) = 5 kg
- Number of items (n) = 4

Step-01:

- Draw a table say 'T' with $(n+1) = 4 + 1 = 5$ number of rows and $(w+1) = 5 + 1 = 6$ number of columns.
- Fill all the boxes of 0th row and 0th column with 0.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

T-Table

Step-02:

Start filling the table row wise top to bottom from left to right using the formula-

$$T(i, j) = \max \{ T(i-1, j), \text{value}_i + T(i-1, j - \text{weight}_i) \}$$

Finding T(1,1)-

We have,

- $i = 1$
- $j = 1$
- $(\text{value})_1 = (\text{value})_1 = 3$
- $(\text{weight})_1 = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,1) = \max \{ T(1-1, 1), 3 + T(1-1, 1-2) \}$$

$$T(1,1) = \max \{ T(0,1), 3 + T(0,-1) \}$$

$$T(1,1) = T(0,1) \{ \text{Ignore } T(0,-1) \}$$

$$T(1,1) = 0$$

Finding T(1,2)-

We have,

- $i = 1$
- $j = 2$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,2) = \max \{ T(1-1, 2), 3 + T(1-1, 2-2) \}$$

$$T(1,2) = \max \{ T(0,2), 3 + T(0,0) \}$$

$$T(1,2) = \max \{ 0, 3+0 \}$$

$$T(1,2) = 3$$

Finding T(1,3)-

We have,

- $i = 1$
- $j = 3$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,3) = \max \{ T(1-1, 3), 3 + T(1-1, 3-2) \}$$

$$T(1,3) = \max \{ T(0,3), 3 + T(0,1) \}$$

$$T(1,3) = \max \{ 0, 3+0 \}$$

$$T(1,3) = 3$$

Finding T(1,4)-

We have,

- $i = 1$
- $j = 4$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,4) = \max \{ T(1-1, 4), 3 + T(1-1, 4-2) \}$$

$$T(1,4) = \max \{ T(0,4), 3 + T(0,2) \}$$

$$T(1,4) = \max \{ 0, 3+0 \}$$

$$T(1,4) = 3$$

Finding T(1,5)-

We have,

- $i = 1$
- $j = 5$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,5) = \max \{ T(1-1, 5), 3 + T(1-1, 5-2) \}$$

$$T(1,5) = \max \{ T(0,5), 3 + T(0,3) \}$$

$$T(1,5) = \max \{ 0, 3+0 \}$$

$$T(1,5) = 3$$

Finding T(2,1)-

We have,

- $i = 2$
- $j = 1$
- $(\text{value})_i = (\text{value})_2 = 4$
- $(\text{weight})_i = (\text{weight})_2 = 3$

Substituting the values, we get-

$$T(2,1) = \max \{ T(2-1, 1), 4 + T(2-1, 1-3) \}$$

$$T(2,1) = \max \{ T(1,1), 4 + T(1,-2) \}$$

$$T(2,1) = T(1,1) \{ \text{Ignore } T(1,-2) \}$$

$$T(2,1) = 0$$

Finding T(2,2)-

We have,

- $i = 2$
- $j = 2$
- $(\text{value})_i = (\text{value})_2 = 4$
- $(\text{weight})_i = (\text{weight})_2 = 3$

Substituting the values, we get-

$$T(2,2) = \max \{ T(2-1, 2), 4 + T(2-1, 2-3) \}$$

$$T(2,2) = \max \{ T(1,2), 4 + T(1,-1) \}$$

$$T(2,2) = T(1,2) \{ \text{Ignore } T(1,-1) \}$$

$$T(2,2) = 3$$

Finding T(2,3)-

We have,

- $i = 2$
- $j = 3$
- $(\text{value})_i = (\text{value})_2 = 4$
- $(\text{weight})_i = (\text{weight})_2 = 3$

Substituting the values, we get-

$$T(2,3) = \max \{ T(2-1, 3), 4 + T(2-1, 3-3) \}$$

$$T(2,3) = \max \{ T(1,3), 4 + T(1,0) \}$$

$$T(2,3) = \max \{ 3, 4+0 \}$$

$$T(2,3) = 4$$

Similarly, compute all the entries.

After all the entries are computed and filled in the table, we get the following table-

	0	1	2	3	4	5
0	0	0	0	0	0	0
✓ 1	0	0	3	3	3	3
✓ 2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

T-Table

- The last entry represents the maximum possible value that can be put into the knapsack.
- So, maximum possible value that can be put into the knapsack = 7.

Identifying Items To Be Put Into Knapsack

Following Step-04,

- We mark the rows labelled “1” and “2”.
- Thus, items that must be put into the knapsack to obtain the maximum value 7 are-

Item-1 and Item-2

Time Complexity-

- Each entry of the table requires constant time $\theta(1)$ for its computation.
- It takes $\theta(nw)$ time to fill $(n+1)(w+1)$ table entries.
- It takes $\theta(n)$ time for tracing the solution since tracing process traces the n rows.
- Thus, overall $\theta(nw)$ time is taken to solve 0/1 knapsack problem using dynamic programming

Code :-

```
# code

# A Dynamic Programming based Python

# Program for 0-1 Knapsack problem

# Returns the maximum value that can

# be put in a knapsack of capacity W


def knapSack(W, wt, val, n):

    dp = [0 for i in range(W+1)] # Making the dp array


    for i in range(1, n+1): # taking first i elements

        for w in range(W, 0, -1): # starting from back,so that we also
            have data of

                                # previous computation when taking i-1
items

                if wt[i-1] <= w:

                    # finding the maximum value

                    dp[w] = max(dp[w], dp[w-wt[i-1]]+val[i-1])


    return dp[W] # returning the maximum value of knapsack


# Driver code

val = [60, 100, 120]

wt = [10, 20, 30]

W = 50

n = len(val)

print(knapSack(W, wt, val, n))
```

Output
220

Conclusion-In this way we have explored Concept of 0/1 Knapsack using Dynamic approach

Assignment Question

1. What is Dynamic Approach?
2. Explain concept of 0/1 knapsack
3. Difference between Dynamic and Branch and Bound Approach. Which is best?
4. Solve one example based on 0/1 knapsack (Other than Manual)

Reference link

- <https://www.gatevidyalay.com/0-1-knapsack-problem-using-dynamic-programming-approach/>
- <https://www.youtube.com/watch?v=mMhC9vuA-7o>
- https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_fractional_knapsack.htm

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group A

Assignment No: 5

Title of the Assignment: Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen's matrix.

Objective of the Assignment: Students should be able to understand and solve n-Queen Problem, and understand basics of Backtracking

Prerequisite:

1. Basic of Python or Java Programming
 2. Concept of backtracking method
 3. N-Queen Problem
-

Contents for Theory:

1. Introduction to Backtracking
 2. N-Queen Problem
-

Introduction to Backtracking

- Many problems are difficult to solve algorithmically. Backtracking makes it possible to solve at least some large instances of difficult combinatorial problems.

Suppose we have to make a series of decisions among various choices, where

- We don't have enough information to know what to choose
- Each decision leads to a new set of choices.
- Some sequence of choices (more than one choices) may be a solution to your problem.

What is backtracking?

Backtracking is finding the solution of a problem whereby the solution depends on the previous steps taken. For example, in a maze problem, the solution depends on all the steps you take one-by-one. If any of those steps is wrong, then it will not lead us to the solution. In a maze problem, we first choose a path and continue moving along it. But once we understand that the particular path is incorrect, then we just come back and change it. This is what backtracking basically is.

In backtracking, we first take a step and then we see if this step taken is correct or not i.e., whether it will give a correct answer or not. And if it doesn't, then we just come back and change our first step. In general, this is accomplished by recursion. Thus, in backtracking, we first start with a partial sub-solution of the problem (which may or may not lead us to the solution) and then check if we can proceed further with this sub-solution or not. If not, then we just come back and change it.

Thus, the general steps of backtracking are:

- start with a sub-solution
- check if this sub-solution will lead to the solution or not
- If not, then come back and change the sub-solution and continue again

Applications of Backtracking:

- N Queens Problem
- Sum of subsets problem

- Graph coloring
- Hamiltonian cycles.

N queens on NxN chessboard

One of the most common examples of the backtracking is to arrange N queens on an NxN chessboard such that no queen can strike down any other queen. A queen can attack horizontally, vertically, or diagonally. The solution to this problem is also attempted in a similar way. We first place the first queen anywhere arbitrarily and then place the next queen in any of the safe places. We continue this process until the number of unplaced queens becomes zero (a solution is found) or no safe place is left. If no safe place is left, then we change the position of the previously placed queen.

N-Queens Problem:

A classic combinational problem is to place n queens on a n*n chess board so that no two attack, i.e no two queens are on the same row, column or diagonal.

What is the N Queen Problem?

N Queen problem is the classical Example of backtracking. N-Queen problem is defined as, “given N x N chess board, arrange N queens in such a way that no two queens attack each other by being in the same row, column or diagonal”.

- For N = 1, this is a trivial case. For N = 2 and N = 3, a solution is not possible. So we start with N = 4 and we will generalize it for N queens.

If we take n=4 then the problem is called the 4 queens problem.

If we take n=8 then the problem is called the 8 queens problem.

Algorithm

- 1) Start in the leftmost column
- 2) If all queens are place return true
- 3) Try all rows in the current column.

Do following for every tried row.

- a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
- b) If placing the queen in [row, column] leads to a solution then return true.
- c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
- 4) If all rows have been tried and nothing worked, return false to trigger backtracking.

4- Queen Problem

Problem 1 : Given 4 x 4 chessboard, arrange four queens in a way, such that no two queens attack each other. That is, no two queens are placed in the same row, column, or diagonal.

	1	2	3	4
1				
2				
3				
4				

4 x 4 Chessboard

- We have to arrange four queens, Q1, Q2, Q3 and Q4 in 4 x 4 chess board. We will put with queen in ith row. Let us start with position (1, 1). Q1 is the only queen, so there is no issue. partial solution is <1>
- We cannot place Q2 at positions (2, 1) or (2, 2). Position (2, 3) is acceptable. the partial solution is <1, 3>.
- Next, Q3 cannot be placed in position (3, 1) as Q1 attacks her. And it cannot be placed at (3, 2), (3, 3) or (3, 4) as Q2 attacks her. There is no way to put Q3 in the third row. Hence, the algorithm backtracks and goes back to the previous solution and readjusts the position of queen Q2. Q2 is moved from positions (2, 3) to (2, 4). Partial solution is <1, 4>

- Now, Q3 can be placed at position (3, 2). Partial solution is $\langle 1, 4, 3 \rangle$.
- Queen Q4 cannot be placed anywhere in row four. So again, backtrack to the previous solution and readjust the position of Q3. Q3 cannot be placed on (3, 3) or (3, 4). So the algorithm backtracks even further.
- All possible choices for Q2 are already explored, hence the algorithm goes back to partial solution $\langle 1 \rangle$ and moves the queen Q1 from (1, 1) to (1, 2). And this process continues until a solution is found.

All possible solutions for 4-queen are shown in fig (a) & fig. (b)

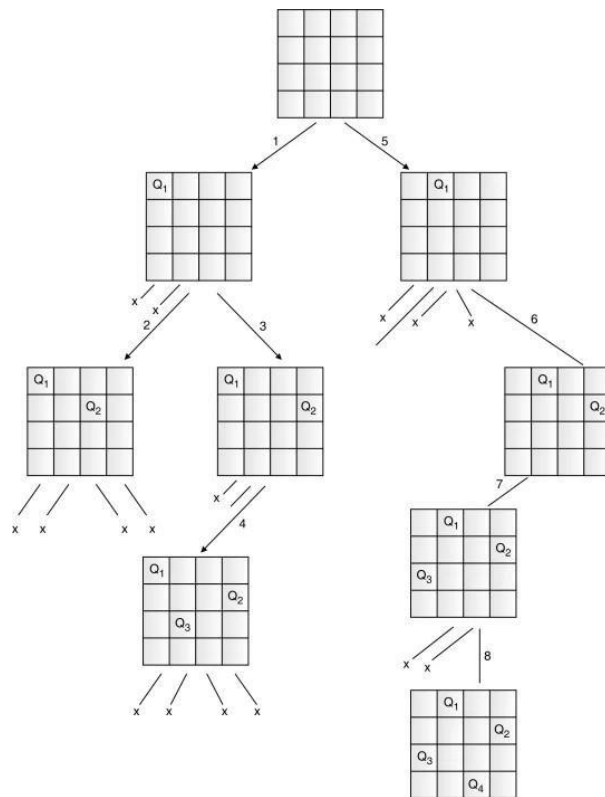
	1	2	3	4
1		Q ₁		
2				Q ₂
3	Q ₃			
4			Q ₄	

Fig. (a): Solution – 1

	1	2	3	4
1			Q ₁	
2	Q ₂			
3				Q ₃
4		Q ₄		

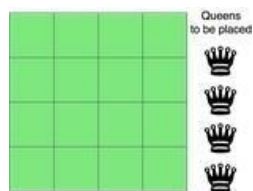
Fig. (b): Solution – 2

Fig. (d) describes the [backtracking](#) sequence for the 4-queen problem.

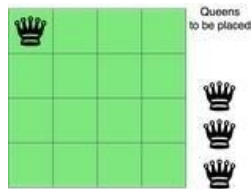


The solution of the 4-queen problem can be seen as four tuples (x_1, x_2, x_3, x_4) , where x_i represents the column number of queen Q_i . Two possible solutions for the 4-queen problem are $(2, 4, 1, 3)$ and $(3, 1, 4, 2)$.

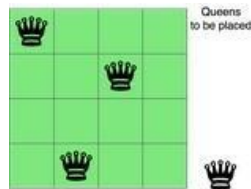
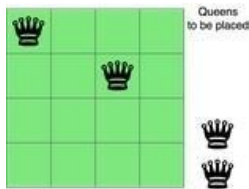
Explanation :



The above picture shows an $N \times N$ chessboard and we have to place N queens on it. So, we will start by placing the first queen.

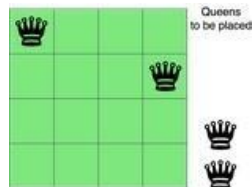


Now, the second step is to place the second queen in a safe position and then the third queen.

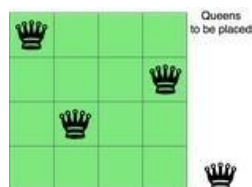


Now, you can see that there is no safe place where we can put the last queen. So, we will just change the position of the previous queen. And this is backtracking.

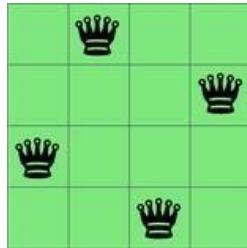
Also, there is no other position where we can place the third queen so we will go back one more step and change the position of the second queen.



And now we will place the third queen again in a safe position until we find a solution.

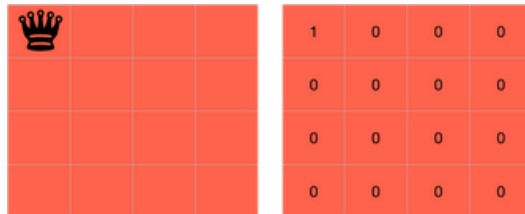


We will continue this process and finally, we will get the solution as shown below.

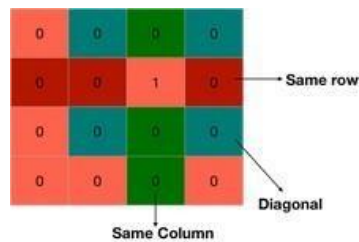


We need to check if a cell (i, j) is under attack or not. For that, we will pass these two in our function along with the chessboard and its size - IS-ATTACK(i, j, board, N).

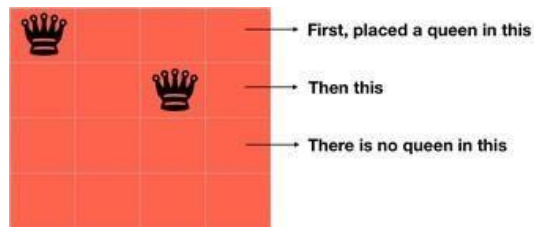
If there is a queen in a cell of the chessboard, then its value will be 1, otherwise, 0.



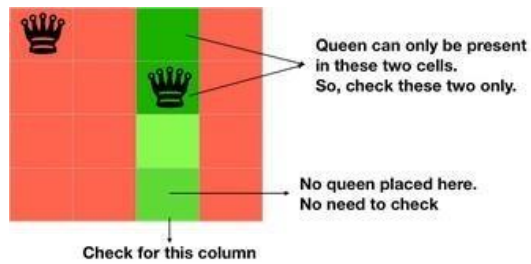
The cell (i,j) will be under attack in three condition - if there is any other queen in row i, if there is any other queen in the column j or if there is any queen in the diagonals.



We are already proceeding row-wise, so we know that all the rows above the current row(i) are filled but not the current row and thus, there is no need to check for row i.



We can check for the column j by changing k from 1 to $i-1$ in $\text{board}[k][j]$ because only the rows from 1 to $i-1$ are filled.



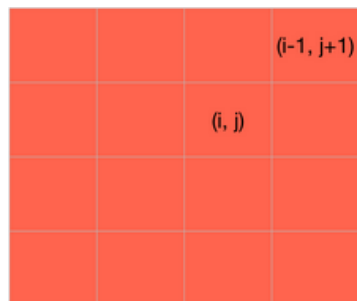
for k in 1 to $i-1$

if $\text{board}[k][j] == 1$

return TRUE

Now, we need to check for the diagonal. We know that all the rows below the row i are empty, so we need to check only for the diagonal elements which above the row i .

If we are on the cell (i, j) , then decreasing the value of i and increasing the value of j will make us traverse over the diagonal on the right side, above the row i .



```
k = i-1
```

```
l = j+1
```

```
while k >= 1 and l <= N
```

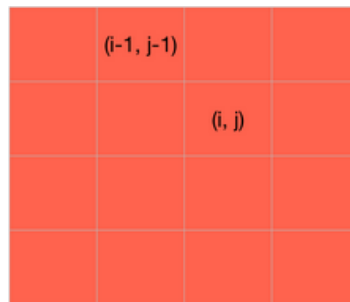
```
    if board[k][l] == 1
```

```
        return TRUE
```

```
    k = k-1
```

```
    l = l+1
```

Also if we reduce both the values of i and j of cell (i, j) by 1, we will traverse over the left diagonal, above the row i.



```
k = i-1
```

```
l = j-1
```

```
while k >= 1 and l >= 1
```

```
    if board[k][l] == 1
```

```
        return TRUE
```

```
    k = k-1
```

```
    l = l-1
```


At last, we will return false as it will be return true is not returned by the above statements and the cell (i,j) is safe.

We can write the entire code as:

```
IS-ATTACK(i,j, board, N)
```

```
// checking in the column j
```

```
for k in 1 to i-1
```

```
if board[k][j]==1
```

```
return TRUE
```

```
// checking upper right diagonal
```

```
k = i-1
```

```
l = j+1
```

```
while k>=1 and l<=N
```

```
if board[k][l] == 1
```

```
return TRUE
```

```
k=k+1
```

```
l=l+1
```

```
// checking upper left diagonal
```

```
k = i-1
```

```
l = j-1
```

```
while k>=1 and l>=1
```

```
if board[k][l] == 1
```

```
return TRUE
```

```
k=k-1
```

```
l=l-1
```

```
return FALSE
```

Now, let's write the real code involving backtracking to solve the N Queen problem.

Our function will take the row, number of queens, size of the board and the board itself - N-QUEEN(row, n, N, board).

If the number of queens is 0, then we have already placed all the queens.

```
if n==0
```

```
return TRUE
```

Otherwise, we will iterate over each cell of the board in the row passed to the function and for each cell, we will check if we can place the queen in that cell or not. We can't place the queen in a cell if it is under attack.

```
for j in 1 to N
```

```
if !IS-ATTACK(row, j, board, N)
```

```
board[row][j] = 1
```

After placing the queen in the cell, we will check if we are able to place the next queen with this arrangement or not. If not, then we will choose a different position for the current queen.

```
for j in 1 to N
```

```
...
```

```
if N-QUEEN(row+1, n-1, N, board)
```

```
    return TRUE
```

```
board[row][j] = 0
```

if N-QUEEN(row+1, n-1, N, board) - We are placing the rest of the queens with the current arrangement. Also, since all the rows up to 'row' are occupied, so we will start from 'row+1'. If this returns true, then we are successful in placing all the queen, if not, then we have to change the position of our current queen. So, we are leaving the current cell board[row][j] = 0 and then iteration will find another place for the queen and this is backtracking.

Take a note that we have already covered the base case - if n==0 → return TRUE. It means when all queens will be placed correctly, then N-QUEEN(row, 0, N, board) will be called and this will return true.

At last, if true is not returned, then we didn't find any way, so we will return false.

```
N-QUEEN(row, n, N, board)
```

```
...
```

```
return FALSE
```

```
N-QUEEN(row, n, N, board)
```

```
if n==0
```

```
    return TRUE
```

```
for j in 1 to N
```

```
    if !IS-ATTACK(row, j, board, N)
```

```
        board[row][j] = 1
```

```
if N-QUEEN(row+1, n-1, N, board)
```

```
return TRUE
```

```
board[row][j] = 0 //backtracking, changing current decision
```

```
return FALSE
```

Code :-

Python3 program to solve N Queen

Problem using backtracking

global N

N = 4

def printSolution(board):

for i in range(N):

for j in range(N):

print(board[i][j], end = " ")

print()

A utility function to check if a queen can

be placed on board[row][col]. Note that this

function is called when "col" queens are

already placed in columns from 0 to col -1.

So we need to check only left side for

attacking queens

def isSafe(board, row, col):

Check this row on left side

for i in range(col):

if board[row][i] == 1:

return False

Check upper diagonal on left side

for i, j in zip(range(row, -1, -1),

range(col, -1, -1)):

if board[i][j] == 1:

return False

Check lower diagonal on left side

for i, j in zip(range(row, N, 1),

range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solveNQUtil(board, col):

base case: If all queens are placed

then return true

if col >= N:

return True

Consider this column and try placing

this queen in all rows one by one

for i in range(N):

if isSafe(board, i, col):

```

# Place this queen in board[i][col]
board[i][col] = 1

# recur to place rest of the queens
if solveNQUtil(board, col + 1) == True:
    return True

# If placing queen in board[i][col]
# doesn't lead to a solution, then
# queen from board[i][col]
board[i][col] = 0

```

```

# if the queen can not be placed in any row in
# this column col then return false
return False

```

```

# This function solves the N Queen problem using
# Backtracking. It mainly uses solveNQUtil() to
# solve the problem. It returns false if queens
# cannot be placed, otherwise return true and
# placement of queens in the form of 1s.
# note that there may be more than one
# solutions, this function prints one of the
# feasible solutions.

```

```

def solveNQ():
    board = [ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

```

```

# Driver Code
solveNQ()

```

Output:-

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

Conclusion- In this way we have explored Concept of Backtracking method and solve n-Queen problem using backtracking method

Assignment Question

1. What is backtracking? Give the general Procedure.
2. Give the problem statement of the n-queens problem. Explain the solution
3. Write an algorithm for N-queens problem using backtracking?
4. Why it is applicable to N=4 and N

Reference link

- <https://www.codesdope.com/blog/article/backtracking-explanation-and-n-queens-problem/>
- <https://www.codesdope.com/course/algorithms-backtracking/>
- <https://codecrucks.com/n-queen-problem/>

Assignment No : 6

MINI PROJECT 1

Theory :-

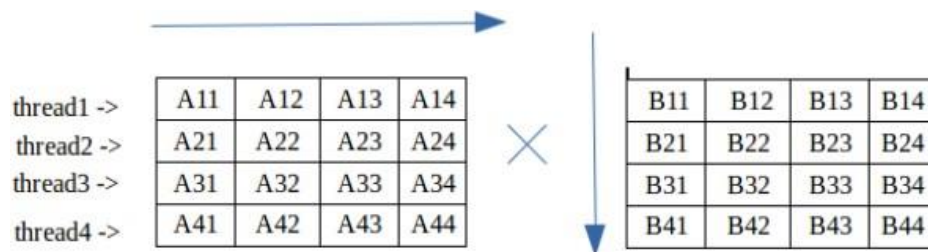
Multiplication of matrix does take time surely. Time complexity of matrix multiplication is $O(n^3)$ using normal matrix multiplication. And Strassen algorithm improves it and its time complexity is $O(n^{2.8074})$.

But, Is there any way to improve the performance of matrix multiplication using the normal method.

Multi-threading can be done to improve it. In multi-threading, instead of utilizing a single core of your processor, we utilize all or more core to solve the problem. We create different threads, each thread evaluating some part of matrix multiplication.

Depending upon the number of cores your processor has, you can create the number of threads required. Although you can create as many threads as you need, a better way is to create each thread for one core.

In second approach, we create a separate thread for each element in resultant matrix. Using `pthread_exit()` we return computed value from each thread which is collected by `pthread_join()`. This approach does not make use of any global variables.



Code :-

```
// CPP Program to multiply two matrix using pthreads
#include <bits/stdc++.h>
using namespace std;

// maximum size of matrix
#define MAX 4

// maximum number of threads
#define MAX_THREAD 4

int matA[MAX][MAX];
int matB[MAX][MAX];
int matC[MAX][MAX];
int step_i = 0;

void* multi(void* arg)
{
    int i = step_i++; //i denotes row number of resultant matC

    for (int j = 0; j < MAX; j++)
        for (int k = 0; k < MAX; k++)
            matC[i][j] += matA[i][k] * matB[k][j];
}

// Driver Code
int main()
{
    // Generating random values in matA and matB
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++) {
            matA[i][j] = rand() % 10;
            matB[i][j] = rand() % 10;
        }
    }
}
```

```
}

// Displaying matA
cout << endl
    << "Matrix A" << endl;
for (int i = 0; i < MAX; i++) {
    for (int j = 0; j < MAX; j++)
        cout << matA[i][j] << " ";
    cout << endl;
}

// Displaying matB
cout << endl
    << "Matrix B" << endl;
for (int i = 0; i < MAX; i++) {
    for (int j = 0; j < MAX; j++)
        cout << matB[i][j] << " ";
    cout << endl;
}

// declaring four threads
pthread_t threads[MAX_THREAD];

// Creating four threads, each evaluating its own part
for (int i = 0; i < MAX_THREAD; i++) {
    int* p;
    pthread_create(&threads[i], NULL, multi, (void*)(p));
}

// joining and waiting for all threads to complete
for (int i = 0; i < MAX_THREAD; i++)
    pthread_join(threads[i], NULL);

// Displaying the result matrix
cout << endl
    << "Multiplication of A and B" << endl;
for (int i = 0; i < MAX; i++) {
    for (int j = 0; j < MAX; j++)
        cout << matC[i][j] << " ";
    cout << endl;
}
return 0;
}
```

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group C

Assignment No : 13

Title of the Assignment: Installation of MetaMask and study spending Ether per transaction

Objective of the Assignment: Students should be able to learn new technology such as metamask. Its application and implementations

Prerequisite:

1. Basic knowledge of cryptocurrency
 2. Basic knowledge of distributed computing concept
 3. Working of blockchain
-

Contents for Theory:

1. Introduction Blockchain
 2. Cryptocurrency
 3. Transaction Wallets
 4. Ether transaction
 5. Installation Process of Metamask
-

Introduction to Blockchain

- Blockchain can be described as a data structure that holds transactional records and while ensuring security, transparency, and decentralization. You can also think of it as a chain of records stored in the forms of blocks which are controlled by no single authority.
- A blockchain is a distributed ledger that is completely open to any and everyone on the network. Once an information is stored on a blockchain, it is extremely difficult to change or alter it.
- Each transaction on a blockchain is secured with a digital signature that proves its authenticity. Due to the use of encryption and digital signatures, the data stored on the blockchain is tamper-proof and cannot be changed.
- Blockchain technology allows all the network participants to reach an agreement, commonly known as consensus. All the data stored on a blockchain is recorded digitally and has a common history which is available for all the network participants. This way, the chances of any fraudulent activity or duplication of transactions is eliminated without the need of a third-party.

Blockchain Features

The following features make the revolutionary technology of blockchain stand out:

- **Decentralized**

Blockchains are decentralized in nature meaning that no single person or group holds the authority of the overall network. While everybody in the network has the copy of the distributed ledger with them, no one can modify it on his or her own. This unique feature of blockchain allows transparency and security while giving power to the users.

- **Peer-to-Peer Network**

With the use of Blockchain, the interaction between two parties through a peer-to-peer model is easily accomplished without the requirement of any third party. Blockchain uses P2P protocol which allows all the network participants to hold an identical copy of transactions, enabling approval through a machine consensus. For example, if you wish to make any transaction from one part of the world to another, you can do that with blockchain all by yourself within a few seconds. Moreover, any interruptions or extra charges will not be deducted in the transfer.

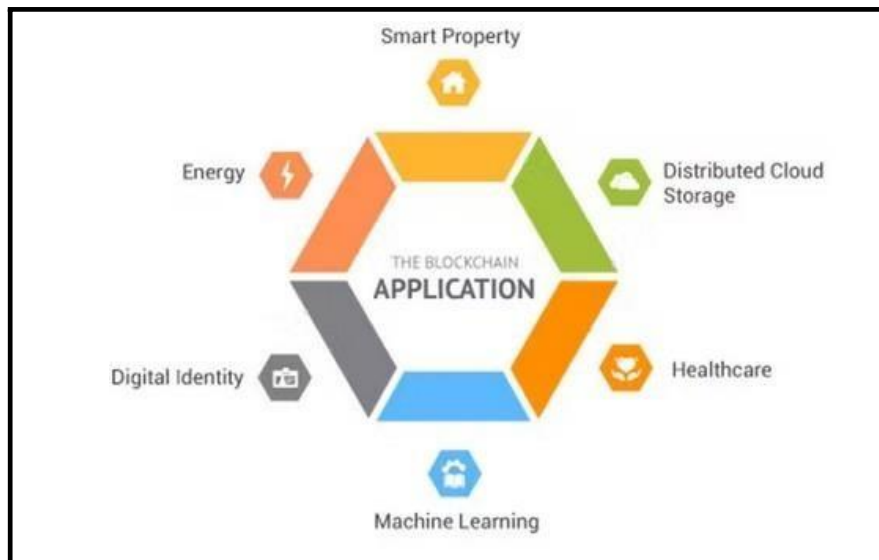
- **Immutable**

The immutability property of a blockchain refers to the fact that any data once written on the blockchain cannot be changed. To understand immutability, consider sending email as an example. Once you send an email to a bunch of people, you cannot take it back. In order to find a way around, you'll have to ask all the recipients to delete your email which is pretty tedious. This is how immutability works.

- **Tamper-Proof**

With the property of immutability embedded in blockchains, it becomes easier to detect tampering of any data. Blockchains are considered tamper-proof as any change in even one single block can be detected and addressed smoothly. There are two key ways of detecting tampering namely, hashes and blocks.

Popular Applications of Blockchain Technology



Benefits of Blockchain Technology:

- **Time-saving:** No central Authority verification needed for settlements making the process faster and cheaper.
- **Cost-saving:** A Blockchain network reduces expenses in several ways. No need for third-party verification. Participants can share assets directly. Intermediaries are reduced. Transaction efforts are minimized as every participant has a copy of shared ledger.
- **Tighter security:** No one can temper with Blockchain Data as it is shared among

millions of participants. The system is safe against cybercrimes and Fraud.

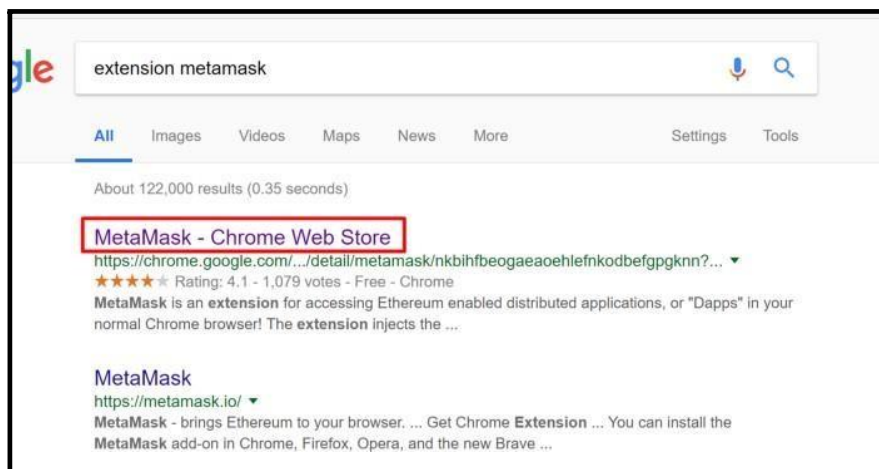
- In finance market trading, Fibonacci retracement levels are widely used in technical analysis.

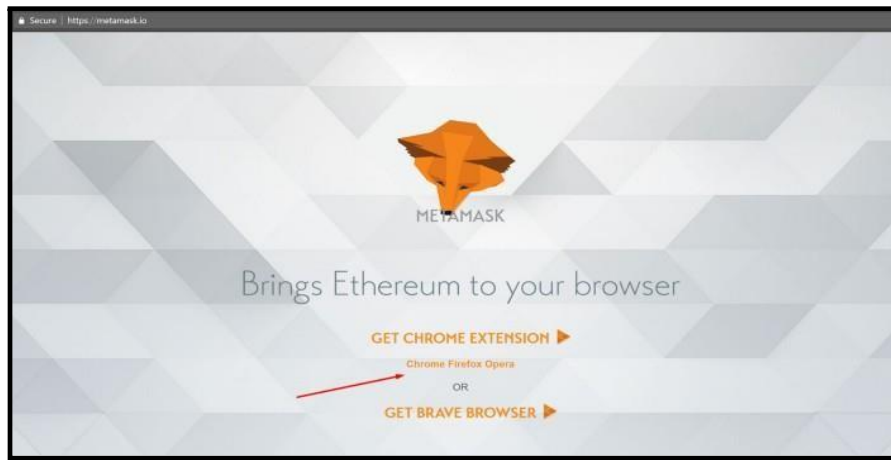
How to use MetaMask: A step by step guide

MetaMask is one of the most popular browser extensions that serves as a way of storing your Ethereum and other [ERC-20 Tokens](#). The extension is free and secure, allowing web applications to read and interact with Ethereum's blockchain.

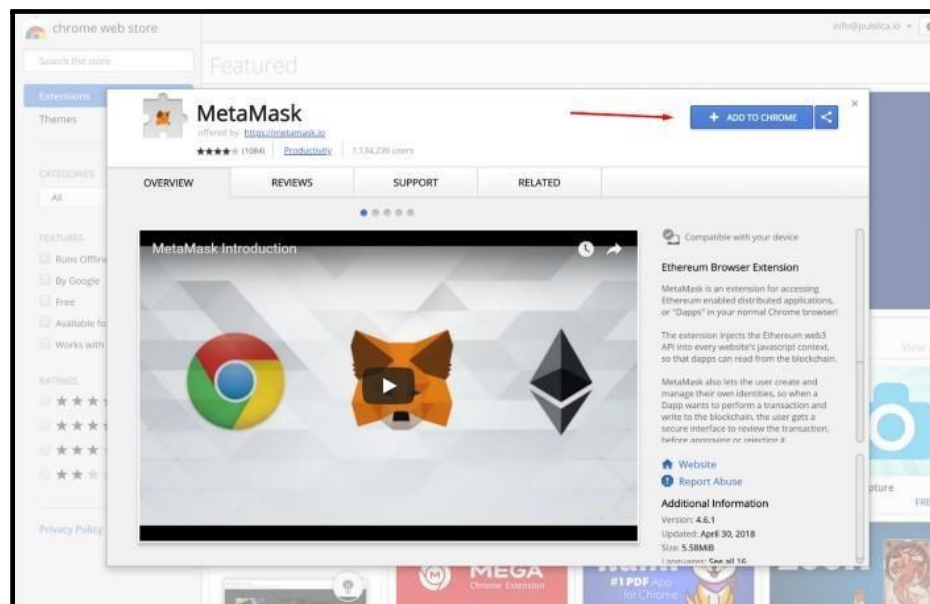
Step 1. Install MetaMask on your browser.

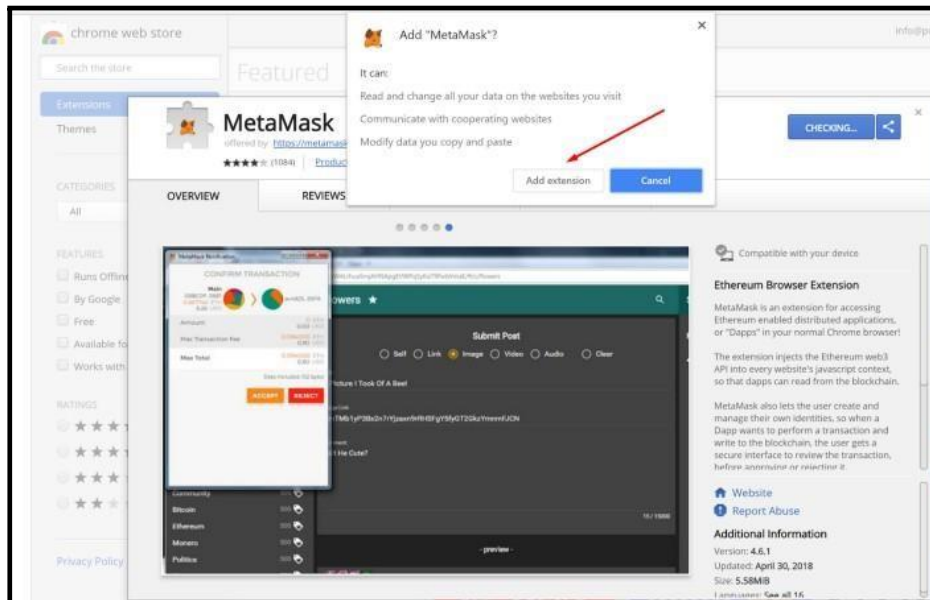
To create a new wallet, you have to install the extension first. Depending on your browser, there are different marketplaces to find it. Most browsers have MetaMask on their stores, so it's not that hard to see it, but either way, here they are [Chrome](#), [Firefox](#), and [Opera](#).





- Click on **Install MetaMask** as a Google Chrome extension.
- Click **Add to Chrome**.
- Click **Add Extension**.

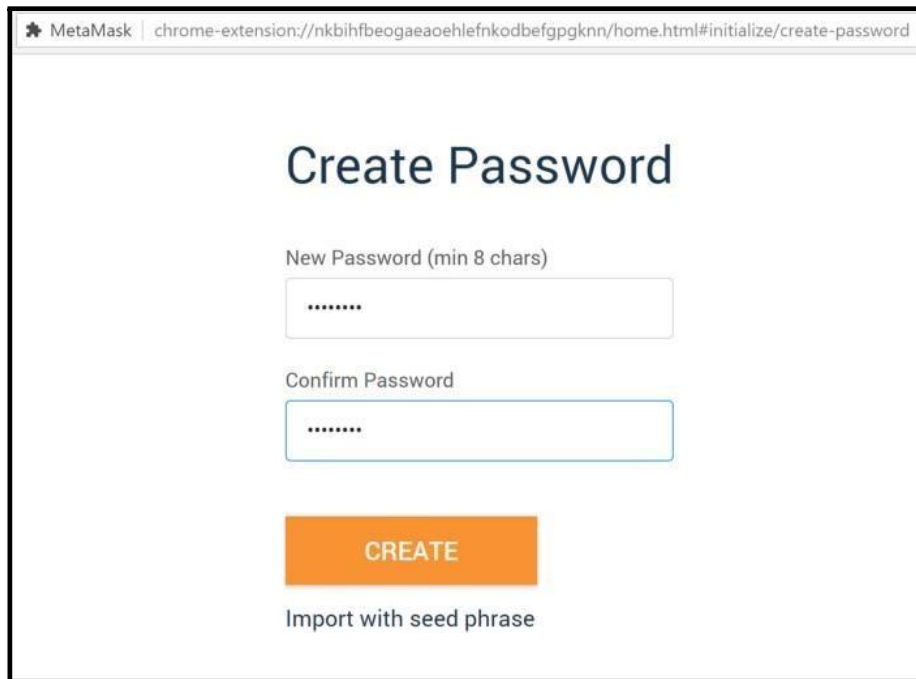




and it's as easy as that to install the extension on your browser, continue reading the next step to figure out how to create an account.

Step 2. Create an account.

- Click on the extension icon in the upper right corner to open MetaMask.
- To install the latest version and be up to date, **click Try it now.**
- **Click Continue.**
- You will be prompted to create a new password. **Click Create.**



MetaMask | chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/home.html#initialize/create-password

Create Password

New Password (min 8 chars)

.....

Confirm Password

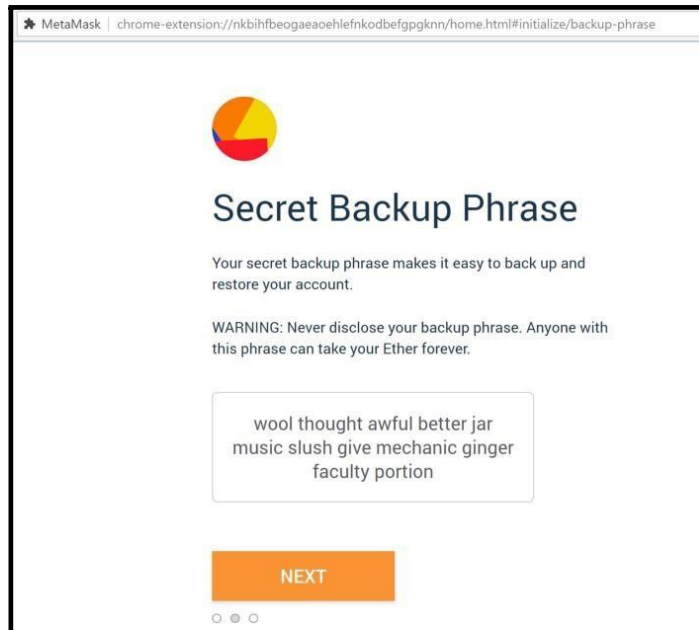
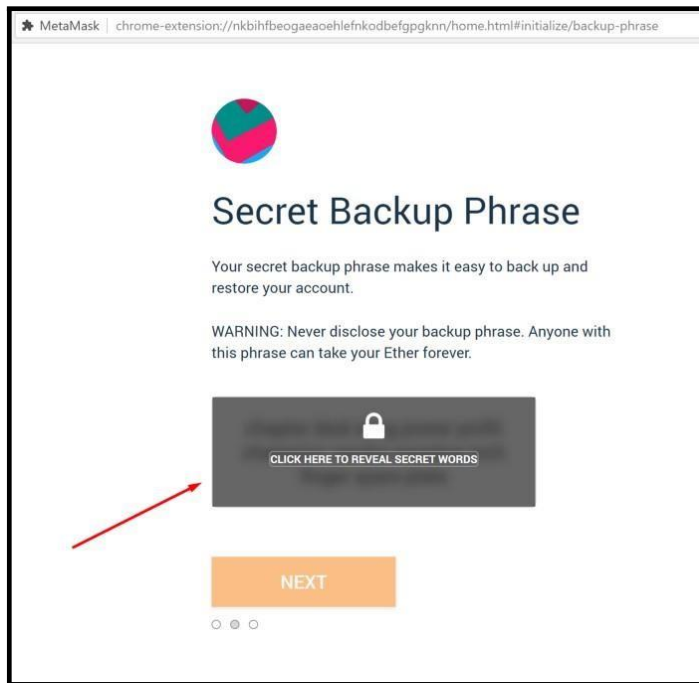
.....

CREATE

[Import with seed phrase](#)

- Proceed by **clicking Next** and accept the Terms of Use.

Click Reveal Secret Words. There you will see a 12 words seed phrase. This is really important and usually not a good idea to store digitally, so take your time and write it down



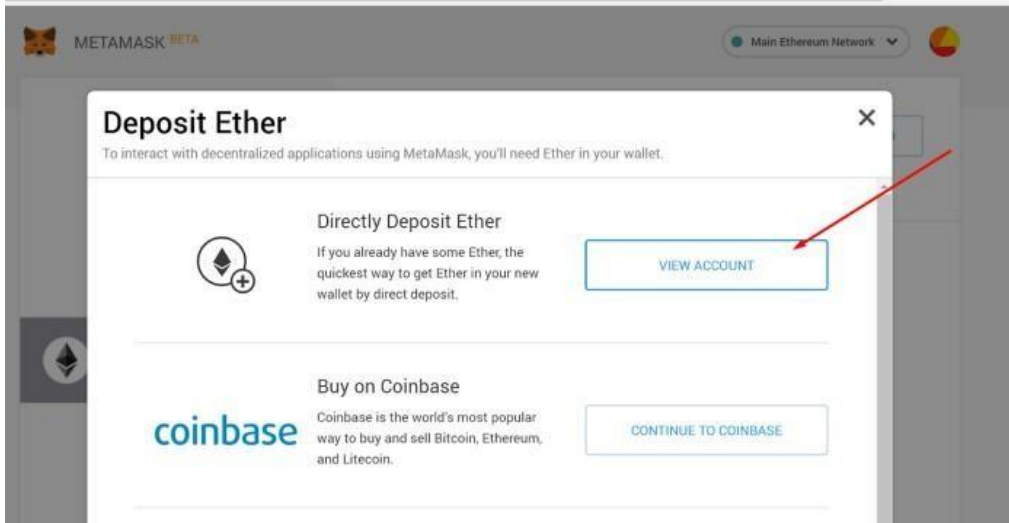
- Verify your secret phrase by selecting the previously generated phrase in order. **Click Confirm.**

And that's it; now you have created your MetaMask account successfully. A new Ethereum wallet

address has just been created for you. It's waiting for you to deposit funds, and if you want to learn how to do that, look at the next step below.

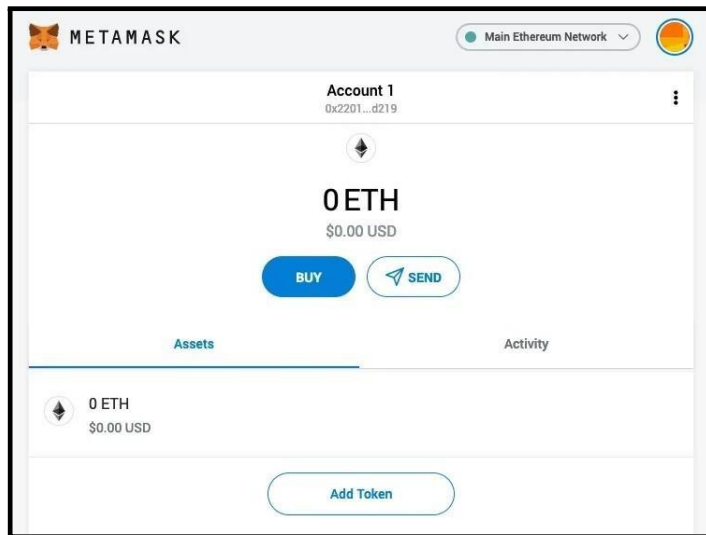
Step 3. Depositing funds.

- Click on **View Account**.



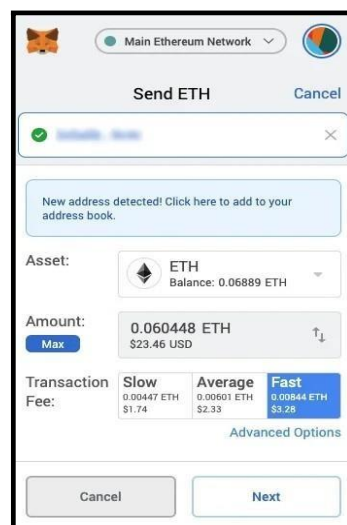
You can now see your public address and share it with other people. There are some methods to buy coins offered by MetaMask, but you can do it differently as well; you just need your address.

If you ever get logged out, you'll be able to log back in again by clicking the MetaMask icon, which will have been added to your web browser (usually found next to the URL bar).



You can now access your list of assets in the 'Assets' tab and view your transaction history in the 'Activity' tab.

- Sending crypto is as simple as clicking the 'Send' button, entering the recipient address and amount to send, and selecting a transaction fee. You can also manually adjust the transaction fee using the 'Advanced Options' button, using information from ETH Gas Station or similar platforms to choose a more acceptable gas price.
- After clicking 'Next', you will then be able to either confirm or reject the transaction on the subsequent page.



- To use MetaMask to interact with a dapp or [smart contract](#), you'll usually need to find a 'Connect to Wallet' button or similar element on the platform you are trying to use. After clicking this, you should then see a prompt asking whether you want to let the dapp connect to your wallet.

What advantages does MetaMask have?

- **Popular** - It is commonly used, so users only need one plugin to access a wide range of dapps.
- **Simple** - Instead of managing private keys, users just need to remember a list of words, and transactions are signed on their behalf.
- **Saves space** - Users don't have to download the Ethereum blockchain, as MetaMask sends requests to nodes outside of the user's computer.
- **Integrated** - Dapps are designed to work with MetaMask, so it becomes much easier to send Ether in and out.

Conclusion- In this way we have explored Concept Blockchain and metamask wallet for transaction of digital currency

Assignment Question

1. What Are the Different Types of Blockchain Technology?
2. What Are the Key Features/Properties of Blockchain?
3. What Type of Records You Can Keep in A Blockchain?
4. What is the difference between Ethereum and Bitcoin?
5. What are Merkle Trees? Explain their concept.
6. What is Double Spending in transaction operation
7. Give real-life use cases of blockchain.

Reference link

- <https://hackernoon.com/blockchain-technology-explained-introduction-meaning-and-applications-edbd6759a2b2>
- <https://levelup.gitconnected.com/how-to-use-metamask-a-step-by-step-guide-f380a3943fb1>
- <https://decrypt.co/resources/metamask>

SNJB's Late Sau.K.B.Jain College of Engineering,Chandwad

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Assignment No : 14

Title of the Assignment: Create your own wallet using Metamask for crypto transactions

Objective of the Assignment: Students should be able to learn about cryptocurrencies and learn how transaction done by using different digital currency

Prerequisite:

1. Basic knowledge of cryptocurrency
 2. Basic knowledge of distributed computing concept
 3. Working of blockchain
-

Contents for Theory:

1. Cryptocurrency
 2. Transaction Wallets
 3. Ether transaction
-

Introduction to Cryptocurrency

- Cryptocurrency is a digital payment system that doesn't rely on banks to verify transactions. It's a peer-to-peer system that can enable anyone anywhere to send and receive payments. Instead of being physical money carried around and exchanged in the real world, cryptocurrency payments exist purely as digital entries to an online database describing specific transactions. When you transfer cryptocurrency funds, the transactions are recorded in a public ledger. Cryptocurrency is stored in digital wallets.
- Cryptocurrency received its name because it uses encryption to verify transactions. This means advanced coding is involved in storing and transmitting cryptocurrency data between wallets and to public ledgers. The aim of encryption is to provide security and safety.
- The first cryptocurrency was Bitcoin, which was founded in 2009 and remains the best known today. Much of the interest in cryptocurrencies is to trade for profit, with speculators at times driving prices skyward.

How does cryptocurrency work?

- Cryptocurrencies run on a distributed public ledger called blockchain, a record of all transactions updated and held by currency holders.
- Units of cryptocurrency are created through a process called mining, which involves using computer power to solve complicated mathematical problems that generate coins. Users can also buy the currencies from brokers, then store and spend them using cryptographic wallets.
- If you own cryptocurrency, you don't own anything tangible. What you own is a key that allows you to move a record or a unit of measure from one person to another without a trusted third party.
- Although Bitcoin has been around since 2009, cryptocurrencies and applications of blockchain technology are still emerging in financial terms, and more uses are expected in the future. Transactions including bonds, stocks, and other financial assets could eventually be traded using the technology.

Cryptocurrency examples

There are thousands of cryptocurrencies. Some of the best known include:

- **Bitcoin:**

Founded in 2009, Bitcoin was the first cryptocurrency and is still the most commonly traded. The currency was developed by Satoshi Nakamoto – widely believed to be a pseudonym for an individual or group of people whose precise identity remains unknown.

- **Ethereum:**

Developed in 2015, Ethereum is a blockchain platform with its own cryptocurrency, called Ether (ETH) or Ethereum. It is the most popular cryptocurrency after Bitcoin.

- **Litecoin:**

This currency is most similar to bitcoin but has moved more quickly to develop new innovations, including faster payments and processes to allow more transactions.

- **Ripple:**

Ripple is a distributed ledger system that was founded in 2012. Ripple can be used to track different kinds of transactions, not just cryptocurrency. The company behind it has worked with various banks and financial institutions.

- Non-Bitcoin cryptocurrencies are collectively known as “altcoins” to distinguish them from the original.

How to store cryptocurrency

- Once you have purchased cryptocurrency, you need to store it safely to protect it from hacks or theft. Usually, cryptocurrency is stored in crypto wallets, which are physical devices or online software used to store the private keys to your cryptocurrencies securely. Some exchanges provide wallet services, making it easy for you to store directly through the platform. However, not all exchanges or brokers automatically provide wallet services for you.
- There are different wallet providers to choose from. The terms “hot wallet” and “cold wallet” are used:
- **Hot wallet storage:** "hot wallets" refer to crypto storage that uses online software to protect the private keys to your assets.
- **Cold wallet storage:** Unlike hot wallets, cold wallets (also known as hardware wallets) rely on offline electronic devices to securely store your private keys.

Conclusion- In this way we have explored Concept Cryptocurrency and learn how transactions are done using digital currency

Assignment Question

1. What is Bitcoin?
2. What Are the biggest Four common cryptocurrency scams
3. Explain How safe are money e-transfers?
4. What is cryptojacking and how does it work?

Reference link

- <https://www.kaspersky.com/resource-center/definitions/what-is-cryptocurrency>

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Assignment No : 15

Title of the Assignment: Write a smart contract on a test network, for Bank account of a customer for following operations:

- Deposit money
- Withdraw Money
- Show balance

Objective of the Assignment: Students should be able to learn new technology such as metamask. Its application and implementations

Prerequisite:

1. Basic knowledge of cryptocurrency
2. Basic knowledge of distributed computing concept
3. Working of blockchain.

Contents for Theory:

The contract will allow deposits from any account, and can be trusted to allow withdrawals only by accounts that have sufficient funds to cover the requested withdrawal.

This post assumes that you are comfortable with the ether-handling concepts introduced in our post, [Writing a Contract That Handles Ether](#).

That post demonstrated how to restrict ether withdrawals to an “owner’s” account. It did this by persistently storing the owner account’s address, and then comparing it to the msg.sender value for any withdrawal attempt. Here’s a slightly simplified version of that smart contract, which allows anybody to deposit money, but only allows the owner to make withdrawals:

```
pragma solidity ^0.4.19;
```

```
contract TipJar {

    address owner; // current owner of the contract

    function TipJar() public {
        owner = msg.sender;
    }

    function withdraw() public {
        require(owner == msg.sender);
        msg.sender.transfer(address(this).balance);
    }

    function deposit(uint256 amount) public payable {
        require(msg.value == amount);
    }

    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }
}
```

I am going to generalize this contract to keep track of ether deposits based on the account address of the depositor, and then only allow that same account to make withdrawals of that ether. To do this, we need a way keep track of account balances for each depositing account—a mapping from accounts to balances. Fortunately, Solidity provides a ready-made mapping data type that can map account addresses to integers,

which will make this bookkeeping job quite simple. (This mapping structure is much more general key/value mapping than just addresses to integers, but that's all we need here.)

Here's the code to accept deposits and track account balances:

```
pragma solidity ^0.4.19;
```

```
contract Bank {

    mapping(address => uint256) public balanceOf; // balances, indexed by addresses

    function deposit(uint256 amount) public payable {
        require(msg.value == amount);

        balanceOf[msg.sender] += amount; // adjust the account's balance
    }
}
```

Here are the new concepts in the code above:

- `mapping(address => uint256) public balanceOf;` declares a persistent public variable, `balanceOf`, that is a mapping from account addresses to 256-bit unsigned integers. Those integers will represent the current balance of ether stored by the contract on behalf of the corresponding address.
- Mappings can be indexed just like arrays/lists/dictionaries/tables in most modern programming languages.
- The value of a missing mapping value is 0. Therefore, we can trust that the beginning balance for all account addresses will effectively be zero prior to the first deposit.

It's important to note that `balanceOf` keeps track of the ether balances assigned to each account, but it does not actually move any ether anywhere. The bank contract's ether balance is the sum of all the balances of all accounts—only `balanceOf` tracks how much of that is assigned to each account.

Note also that this contract doesn't need a constructor. There is no persistent state to initialize other than the `balanceOf` mapping, which already provides default values of 0.

Given the `balanceOf` mapping from account addresses to ether amounts, the remaining code for a fully-functional bank contract is pretty small. I'll simply add a withdrawal function:

bank.sol

```
pragma solidity ^0.4.19;
```

```
contract Bank {

    mapping(address => uint256) public balanceOf; // balances, indexed by addresses

    function deposit(uint256 amount) public payable {
        require(msg.value == amount);
        balanceOf[msg.sender] += amount; // adjust the account's balance
    }

    function withdraw(uint256 amount) public {
        require(amount <= balanceOf[msg.sender]);
        balanceOf[msg.sender] -= amount;
        msg.sender.transfer(amount);
    }
}
```

The code above demonstrates the following:

- The `require(amount <= balanceOf[msg.sender])` checks to make sure the sender has sufficient funds to cover the requested withdrawal. If not, then the transaction aborts without making any state changes or ether transfers.
- The `balanceOf` mapping must be updated to reflect the lowered residual amount after the withdrawal.
- The funds must be sent to the sender requesting the withdrawal.

In the `withdraw()` function above, it is very important to adjust `balanceOf[msg.sender]` **before** transferring ether to avoid an exploitable vulnerability. The reason is specific to smart contracts and the fact that a transfer to a smart contract executes code in that smart contract. (The essentials of Ethereum transactions are discussed in [How Ethereum Transactions Work](#).)

Now, suppose that the code in `withdraw()` did not adjust `balanceOf[msg.sender]` before making the transfer *and* suppose that `msg.sender` was a malicious smart contract. Upon receiving the transfer—handled by `msg.sender`'s fallback function—that malicious contract could initiate *another* withdrawal from the banking contract. When the banking contract handles this second withdrawal request, it would have already transferred ether for the original withdrawal, but it would not have an updated balance, so it would allow this second withdrawal!

This vulnerability is called a “reentrancy” bug because it happens when a smart contract invokes code in a different smart contract that then calls back into the original, thereby reentering the exploitable contract. For this reason, it's essential to always make sure a contract's internal state is fully updated before it potentially invokes code in another smart contract. (And, it's essential to remember that every transfer to a smart contract executes that contract's code.)

To avoid this sort of reentrancy bug, follow the “Checks-Effects-Interactions pattern” as [described in the Solidity documentation](#). The `withdraw()` function above is an example of implementing this pattern

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Assignment No : 16

Title of the Assignment: Write a survey report on types of Blockchains and its real time use cases.

Objective of the Assignment: Students should be able to learn new technology such as metamask. Its application and implementations

Prerequisite:

1. Basic knowledge of cryptocurrency
2. Basic knowledge of distributed computing concept
3. Working of blockchain

Contents for Theory:

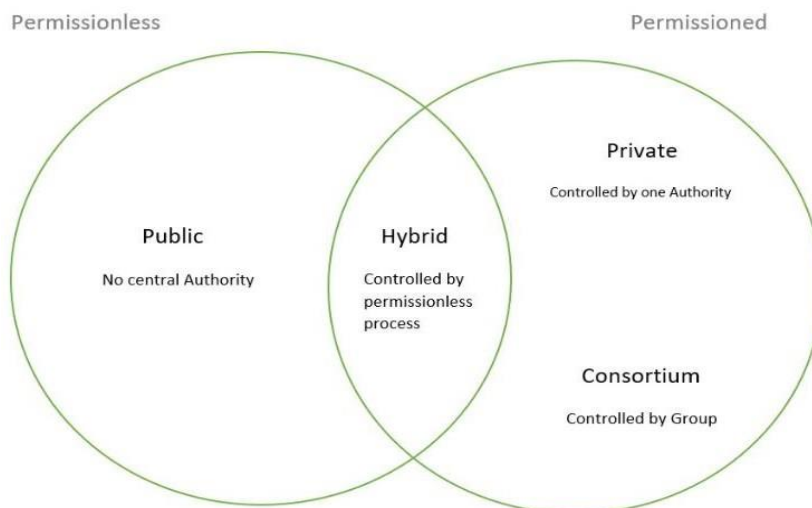
There are 4 types of blockchain:

Public Blockchain.

Private Blockchain.

Hybrid Blockchain.

Consortium Blockchain



1. Public Blockchain

These blockchains are completely open to following the idea of decentralization. They don't have any restrictions, anyone having a computer and internet can participate in the network.

As the name is public this blockchain is open to the public, which means it is not owned by anyone. Anyone having internet and a computer with good hardware can participate in this public blockchain. All the computer in the network hold the copy of other nodes or block present in the network. In this public blockchain, we can also perform verification of transactions or records.

Advantages:

Trustable: There are algorithms to detect no fraud. Participants need not worry about the other nodes in the network.

Secure: This blockchain is large in size as it is open to the public. In a large size, there is greater distribution of records.

Anonymous Nature: It is a secure platform to make your transaction properly at the same time, you are not required to reveal your name and identity in order to participate.

Decentralized: There is no single platform that maintains the network, instead every user has a copy of the ledger.

Disadvantages:

Processing: The rate of the transaction process is very slow, due to its large size. Verification of each node is a very time-consuming process.

Energy Consumption: Proof of work is high energy-consuming. It requires good computer hardware to participate in the network.

Acceptance: No central authority is there so governments are facing the issue to implement the technology faster.

Use Cases: Public Blockchain is secured with proof of work or proof of stake they can be used to displace traditional financial systems. The more advanced side of this blockchain is the smart contract that enabled this blockchain to support decentralization. Examples of public blockchain are Bitcoin, Ethereum.

2. Private Blockchain

These blockchains are not as decentralized as the public blockchain only selected nodes can participate in the process, making it more secure than the others.

These are not as open as a public blockchain.

They are open to some authorized users only.

These blockchains are operated in a closed network.

In this few people are allowed to participate in a network within a company/organization.

Advantages:

Speed: The rate of the transaction is high, due to its small size. Verification of each node is less time-consuming.

Scalability: We can modify the scalability. The size of the network can be decided manually.

Privacy: It has increased the level of privacy for confidentiality reasons as the businesses required.

Balanced: It is more balanced as only some user has the access to the transaction which improves the performance of the network.

Disadvantages:

Security- The number of nodes in this type is limited so chances of manipulation are there. These blockchains are more vulnerable.

Centralized- Trust building is one of the main disadvantages due to its central nature. Organizations can use this for malpractices.

Count- Since there are few nodes if nodes go offline the entire system of blockchain can be endangered.

Use Cases: With proper security and maintenance, this blockchain is a great asset to secure information without exposing it to the public eye. Therefore companies use them for internal auditing, voting, and asset management. An example of private blockchains is Hyperledger, Corda.

3. Hybrid Blockchain

It is the mixed content of the private and public blockchain, where some part is controlled by some organization and other makes are made visible as a public blockchain.

It is a combination of both public and private blockchain. Permission-based and permissionless systems are used. User access information via smart contracts.

Even a primary entity owns a hybrid blockchain it cannot alter the transaction.

Advantages:

Ecosystem: Most advantageous thing about this blockchain is its hybrid nature. It cannot be hacked as 51% of users don't have access to the network.

Cost: Transactions are cheap as only a few nodes verify the transaction. All the nodes don't carry the verification hence less computational cost.

Architecture: It is highly customizable and still maintains integrity, security, and transparency.

Operations: It can choose the participants in the blockchain and decide which transaction can be made public.

Disadvantages:

Efficiency: Not everyone is in the position to implement a hybrid Blockchain. The organization also faces some difficulty in terms of efficiency in maintenance.

Transparency: There is a possibility that someone can hide information from the user. If someone wants to get access through a hybrid blockchain it depends on the organization whether they will give or not.

Ecosystem: Due to its closed ecosystem this blockchain lacks the incentives for network participation.

Use Case: It provides a greater solution to the health care industry, government, real estate, and financial companies. It provides a remedy where data is to be accessed publicly but needs to be shielded privately.

Examples of Hybrid Blockchain are Ripple network and XRP token.

4. Consortium Blockchain

It is a creative approach that solves the needs of the organization. This blockchain validates the transaction and also initiates or receives transactions.

Also known as Federated Blockchain.

This is an innovative method to solve the organization's needs.

Some part is public and some part is private.

In this type, more than one organization manages the blockchain.

Advantages:

Speed: A limited number of users make verification fast. The high speed makes this more usable for organizations.

Authority: Multiple organizations can take part and make it decentralized at every level. Decentralized authority, makes it more secure.

Privacy: The information of the checked blocks is unknown to the public view. but any member belonging to the blockchain can access it.

Flexible: There is much divergence in the flexibility of the blockchain. Since it is not a very large decision can be taken faster.

Disadvantages:

Approval: All the members approve the protocol making it less flexible. Since one or more organizations are involved there can be differences in the vision of interest.

Transparency: It can be hacked if the organization becomes corrupt. Organizations may hide information from the users.

Vulnerability: If few nodes are getting compromised there is a greater chance of vulnerability in this blockchain

Use Cases: It has high potential in businesses, banks, and other payment processors. Food tracking of the organizations frequently collaborates with their sectors making it a federated solution ideal for their use.

Examples of consortium Blockchain are Tendermint and Multichain.

Conclusion-In this way we have explored types of blockchain and its applications in real time

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Assignment No : 17

Title of the Assignment: Write a program to create a Business Network using Hyperledger.

Objective of the Assignment: Students should be able to learn hyperledger .Its application and implementations

Prerequisite:

1. Basic knowledge of cryptocurrency
 2. Basic knowledge of distributed computing concept
 3. Working of blockchain
-

Contents for Theory:

Hyperledger Composer is an extensive, open development toolset and framework to make developing blockchain applications easier. The primary goal is to accelerate time to value, and make it easier to integrate your blockchain applications with the existing business systems.

- You can use Composer to rapidly develop use cases and deploy a blockchain solution in days.
- Composer allows you to model your business network and integrate existing systems and data with your blockchain applications.
- Hyperledger Composer supports the existing [Hyperledger Fabric blockchain](#) infrastructure and runtime.
- Hyperleder Composer generate business network archive (bna) file which you can deploy on existing Hyperledger Fabric network

You can use Hyperledger Composer to model business network, containing your existing assets and the transactions related to them

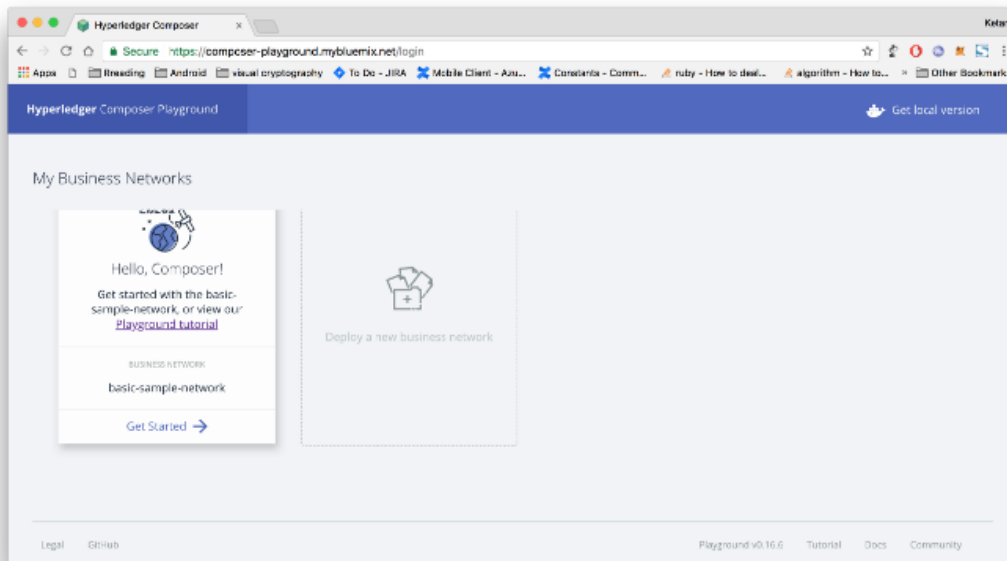
Key Concepts of Hyperledger Composer

1. Blockchain State Storage: It stores all transaction that happens in your hyperledger composer application. It stores transaction in Hyperledger fabric network.

2. **Connection Profiles:** Connection Profiles to configuration JSON file which help composer to connect to Hyperledger Fabric. You can find Connection Profile JSON file in user's home directory.
3. **Assets:** Assets are tangible or intangible goods, services, or property, and are stored in registries. Assets can represent almost anything in a business network, for example, a house for sale, the sale listing, the land registry certificate for that house. Assets must have a unique identifier, but other than that, they can contain whatever properties you define.
4. **Participants:** Participants are members of a business network. They may own assets and submit transactions. Participant must have an identifier and can have any other properties.
5. **Identities and ID cards:** Participants can be associated with an identity. ID cards are a combination of an identity, a connection profile, and metadata. ID cards simplify the process of connecting to a business network.
6. **Transactions:** Transactions are the mechanism by which participants interact with assets. Transaction processing logic you can define in JavaScript and you can also emit event for transaction.
7. **Queries:** Queries are used to return data about the blockchain world-state. Queries are defined within a business network, and can include variable parameters for simple customisation. By using queries, data can be easily extracted from your blockchain network. Queries are sent by using the Hyperledger Composer API.
8. **Events:** Events are defined in the model file. Once events have been defined, they can be emitted by transaction processor functions to indicate to external systems that something of importance has happened to the ledger.
9. **Access Control:** Hyperledger is enterprise blockchain and access control is core feature of any business blockchain. Using Access Control rules you can define who can do what in Business networks. The access control language is rich enough to capture sophisticated conditions.
10. **Historian registry:** The historian is a specialised registry which records successful transactions, including the participants and identities that submitted them. The historian stores transactions as HistorianRecord assets, which are defined in the Hyperledger Composer system namespace.

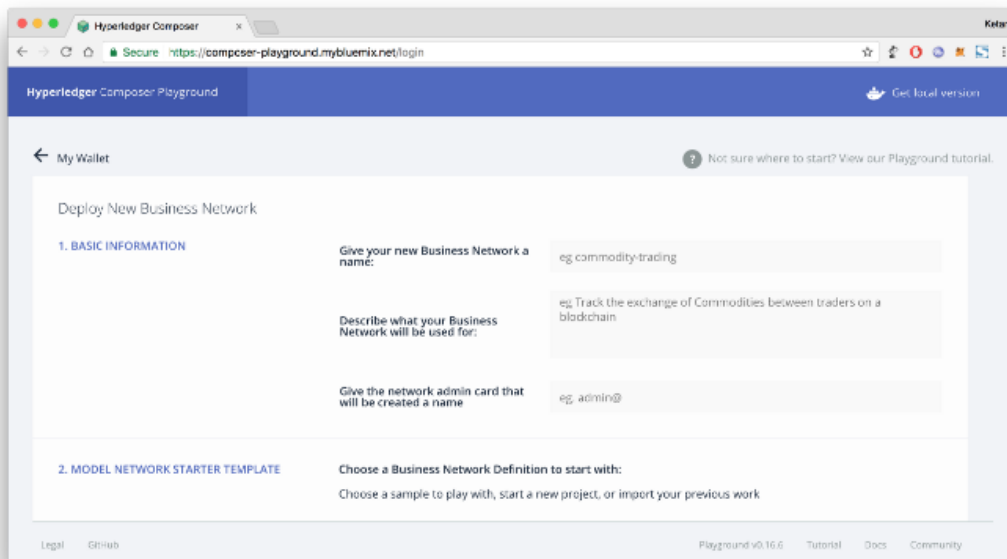
**Let's create first
Hyperledger
Composer Application**

Step 1: Start Hyperledger Composer Online version of Local. Click on Deploy a new business network

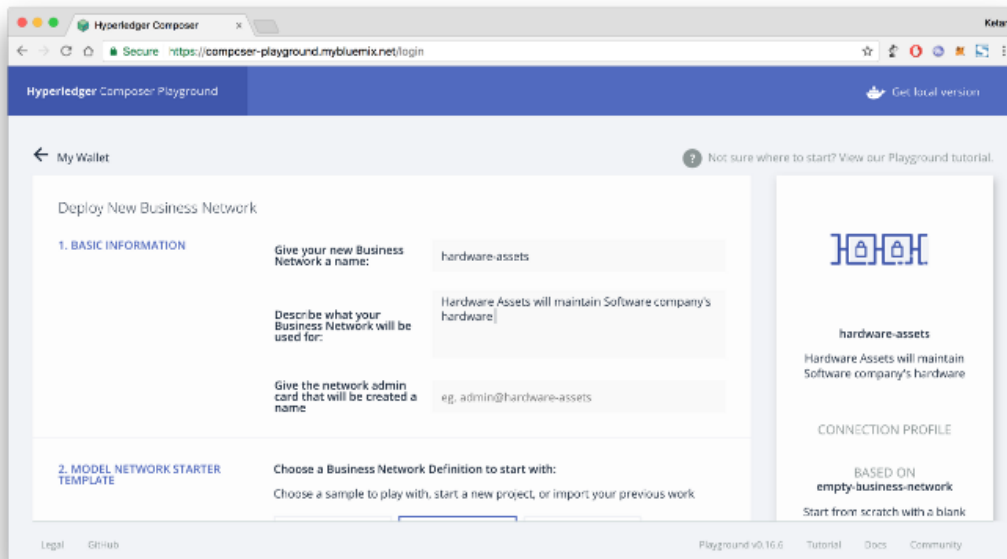


Hyperledger Composer Playground Online version

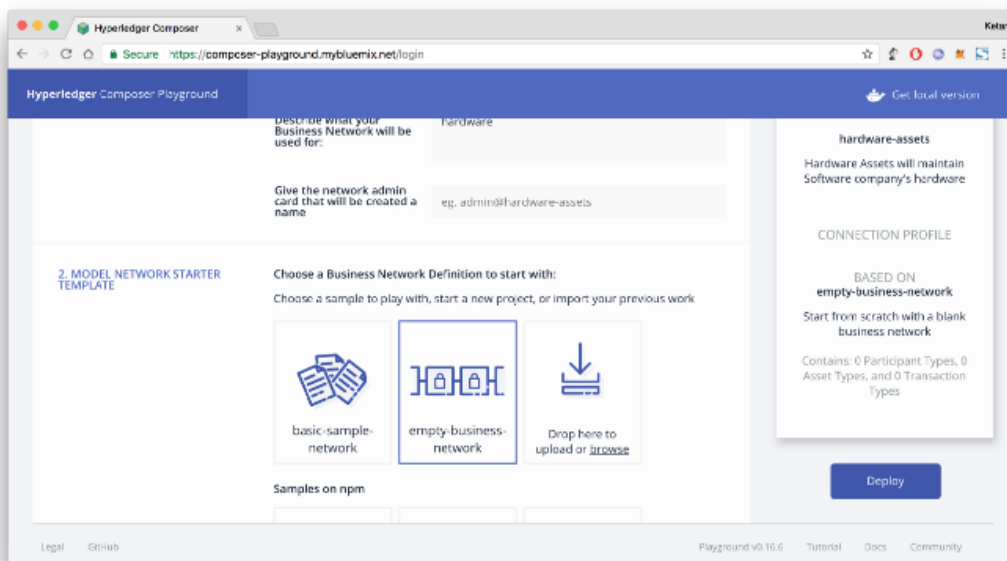
Step 2: Select empty business network



Step 3: Fill basic information, select empty business network and click “deploy” button from right pannel

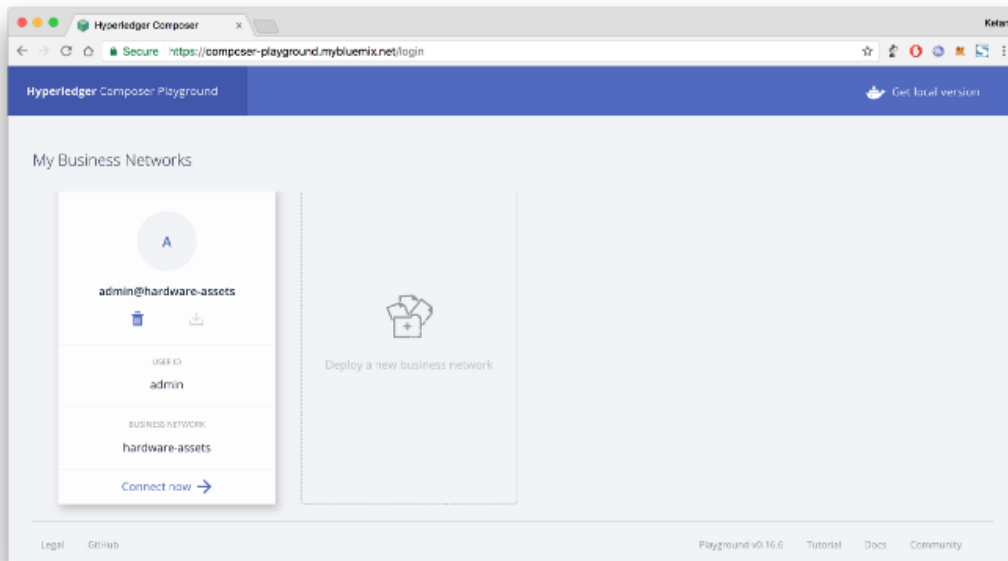


Fill basic information

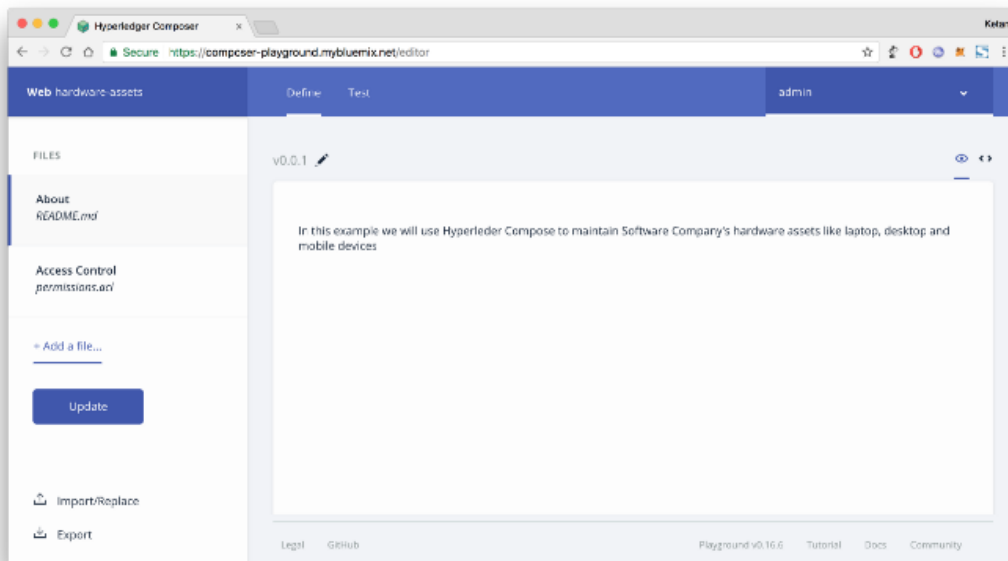


select empty business network

Step 4: Connect to "hardware-assets" business network that we have just deployed

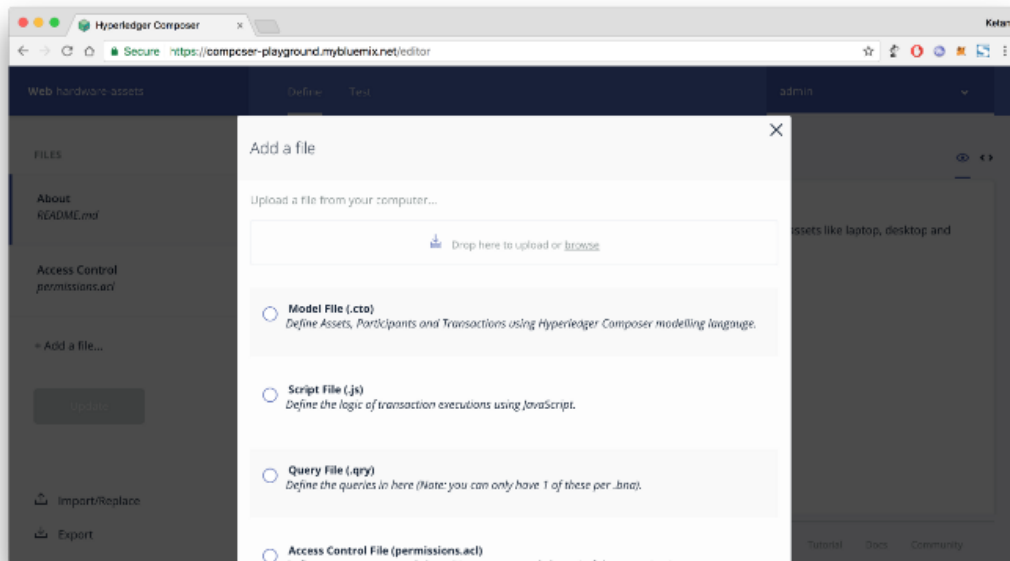


click on “connect now” button



Inside hardware-assets business network

Step 5: Click on “+Add a file...” from left panel and select “model file (.cto)”



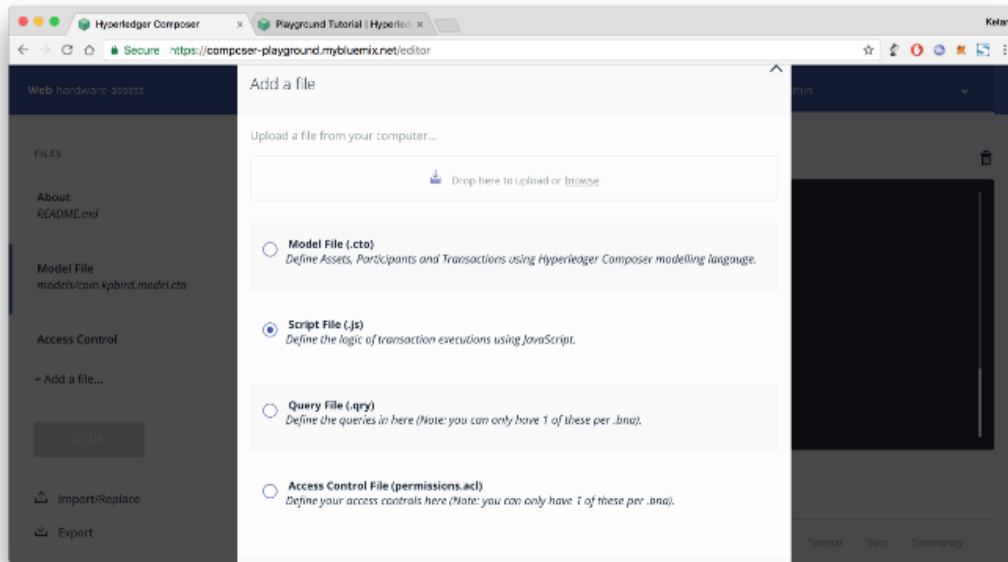
Write following code in model file. Model file contain asset in our case it's hardware, participant in our case participants are employee of organisation and transaction as Allocate hardware to employee. Each model has extra properties. Make sure you have proper and unique namespace. In this example I am using "com.kpbird" as namespace. You can access all models using this namespace i.e. com.kpbird.Hardware, com.kpbird.Employee

```
/**
 * Hardware model
 */
namespace com.kpbirdasset
Hardware identified by hardwareid {
  o String hardwareid
  o String name
  o String type
  o String description
  o Double quantity
  → Employee owner
}
participant Employee identified by employeeid {
  o String employeeid
  o String firstName
  o String lastName
}
transaction Allocate {
  → Hardware hardware
  → Employee newOwner
}
```

Hyperledger modeling language

reference: https://hyperledger.github.io/composer/reference/cto_language.html

Step 6: Click on "+Add a file..." from left panel and select "script file (*.js)"



Write following code in Script File. In Script we can define transaction processing logic. In our case we want to allocate hardware to the employee so, we will update owner of hardware. Make sure about annotation above functions @params and @transaction

```
/**
 * Track the trade of a commodity from one trader to another
 * @param {com.kpbird.Allocate} trade – the trade to be processed
 * @transaction
 */
function allocateHardware(allocate) {
  allocate.hardware.owner = allocate.newOwner;
  return getAssetRegistry('com.kpbird.Hardware')
    .then(function (assetRegistry) {
      return assetRegistry.update(allocate.hardware);
    });
}
```

Hyperledger Composer Script file reference: https://hyperledger.github.io/composer/reference/js_scripts.html

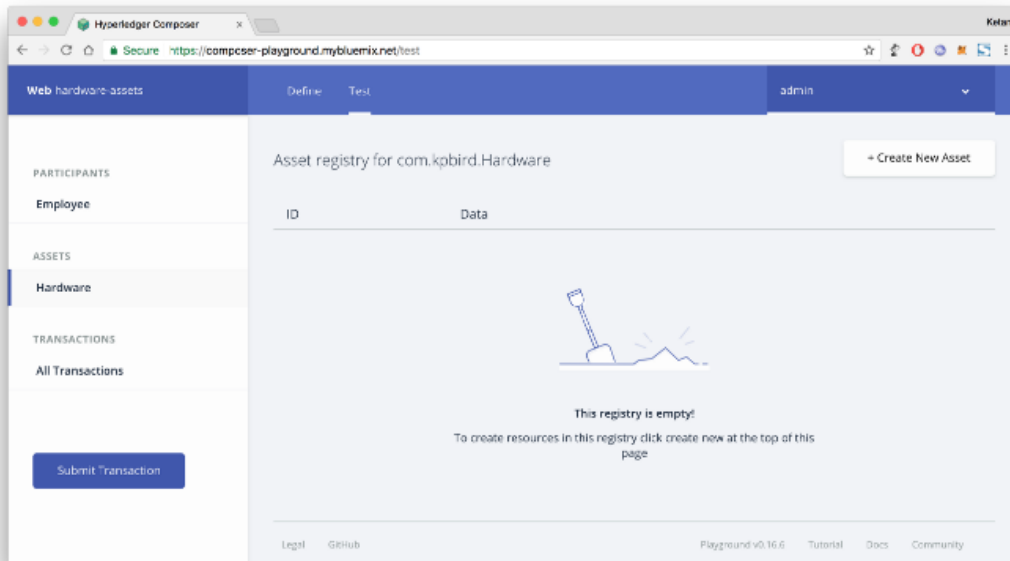
Step 7: permissions.acl file sample is already available, Add following code in permissions.acl file.

```
/**
 * New access control file
 */
rule AllAccess {
  description: "AllAccess – grant everything to everybody."
  participant: "ANY"
  operation: ALL
  resource: "com.kpbird.***"
  action: ALLOW
}
rule SystemACL {
  description: "System ACL to permit all access"
  participant: "org.hyperledger.composer.system.Participant"
  operation: ALL
  resource: "org.hyperledger.composer.system.***"
  action: ALLOW
}
```

Hyperledger Composer Access Control Language

reference: https://hyperledger.github.io/composer/reference/acl_language.html

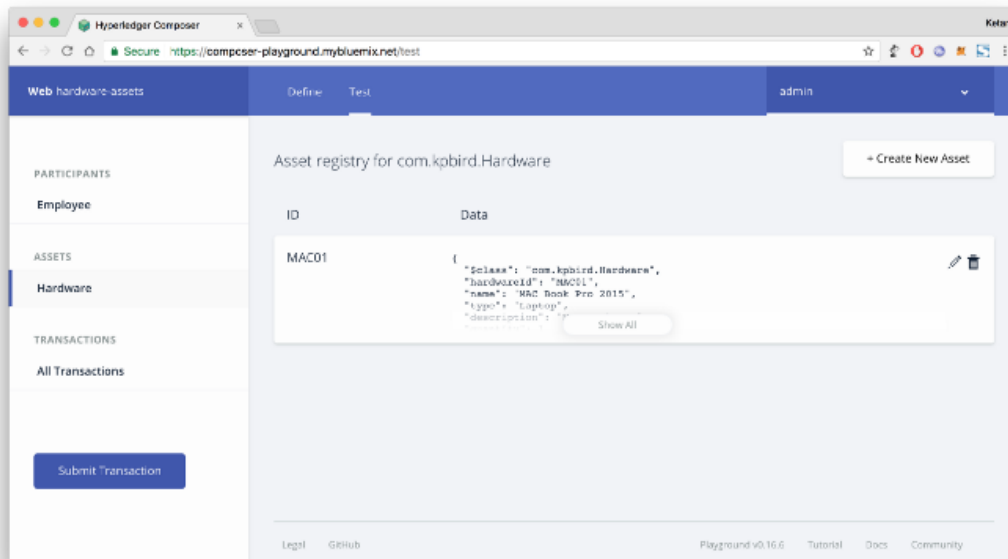
Step 8: Now, It's time to test our hardware assets business network. Hyperledger composer gives "Test" facility from composer panel it self. Click on "Test" tab from top panel



Test feature of Hyperledger Composer

Step 9: Create Assets. Click on "Hardware" from left panel and click "+ Create New Assets" from right top corner and add following code. We will create Employee#01 in next step. Click on "Create New" button

```
{
  "$class": "com.kpbird.Hardware",
  "hardwareId": "MAC01",
  "name": "MAC Book Pro 2015",
  "type": "Laptop",
  "description": "Mac Book Pro",
  "quantity": 1,
  "owner": "resource:com.kpbird.Employee#01"
}
```



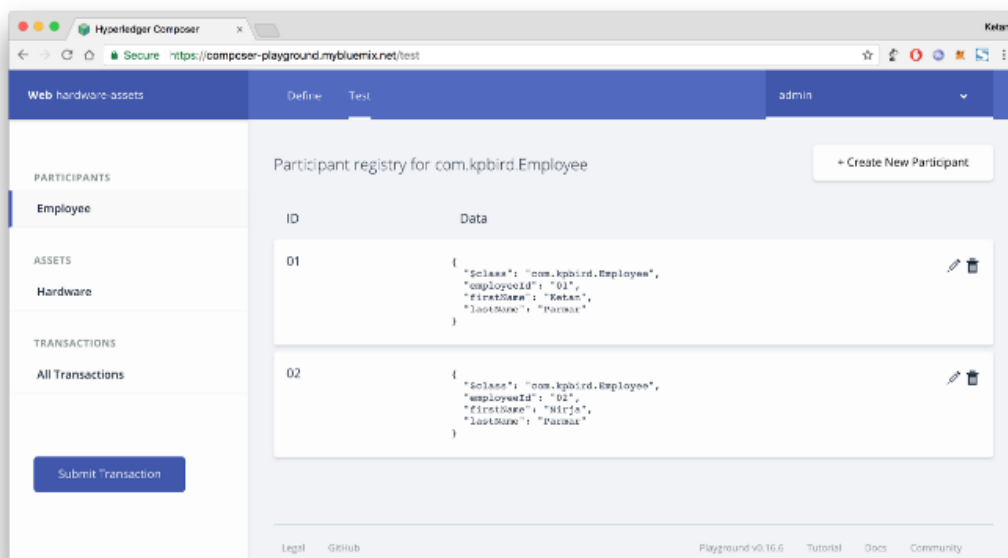
After adding Hardware assets

Steps 10: Let's create participants. Click "Employee" and click "+ Create New Participants" and add following code. We will add two employees

```
{  
  "$class": "com.kpbird.Employee",  
  "employeeId": "01",  
  "firstName": "Ketan",  
  "lastName": "Parmar"  
}
```

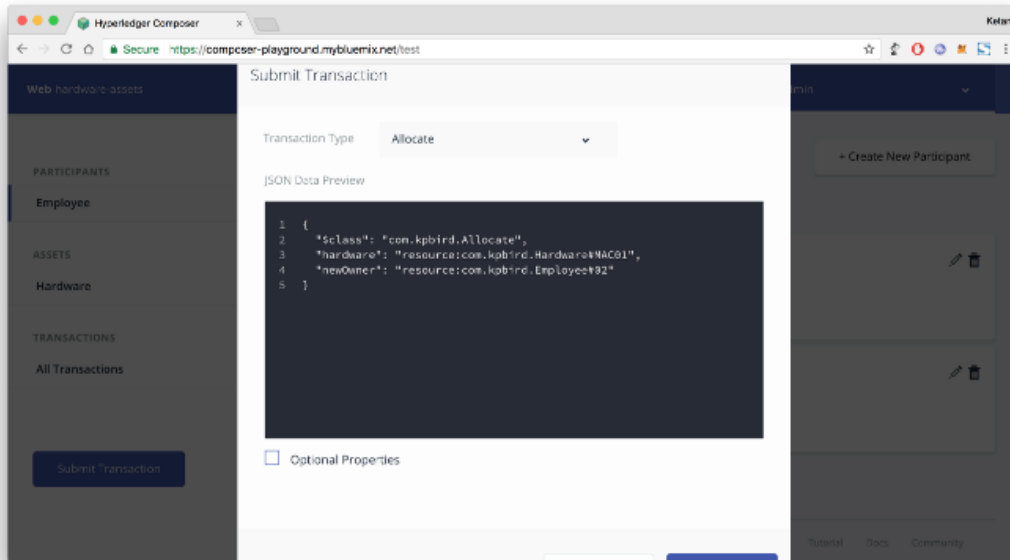
Click on "Create New" on dialog

```
{  
  "$class": "com.kpbird.Employee",  
  "employeeId": "02",  
  "firstName": "Nirja",  
  "lastName": "Parmar"  
}
```



We have two employees

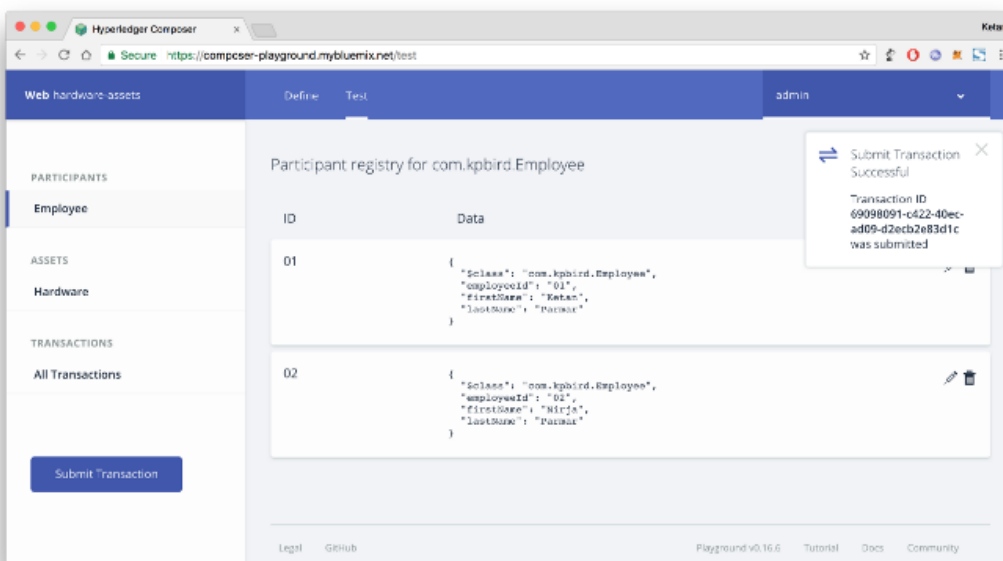
Step 11: It's time to do transaction, We will allocate Macbook Pro from Ketan (Employee#01) to Nirja (Employee#02). Click on "Submit Transaction" button from left panel. In Transaction dialog, We can see all transaction functions on top "Transaction Type" dropdown.



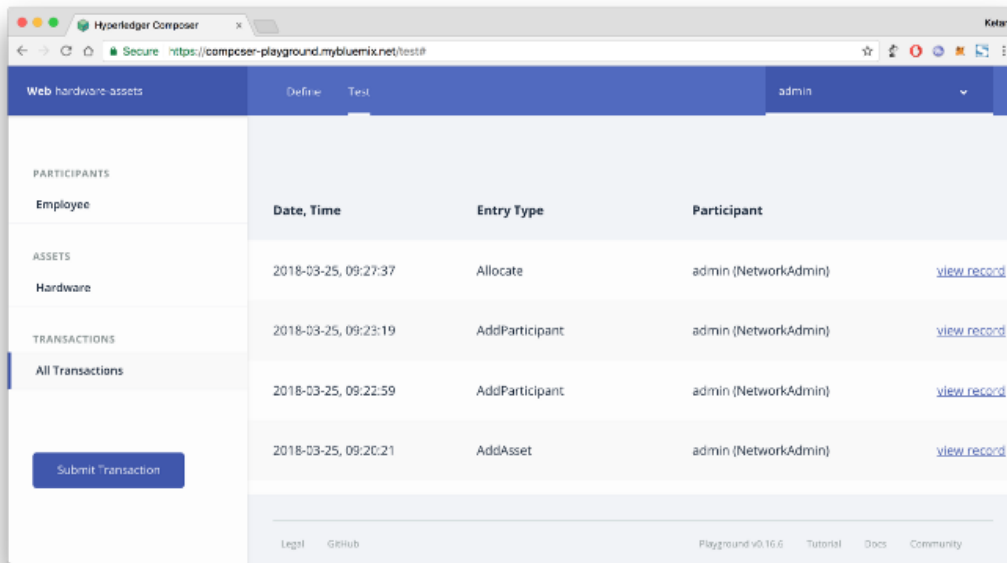
Submit Transaction Dialog

```
{
  "$class": "com.kpbird.Allocate",
  "hardware": "resource:com.kpbird.Hardware#MAC01",
  "newOwner": "resource:com.kpbird.Employee#02"
}
```

Now, We are allocating Mac01 to Employee 02. Click Submit button after update above JSON in Transaction Dialog. As soon as you hit submit button. Transaction processed and Transaction Id will generate.

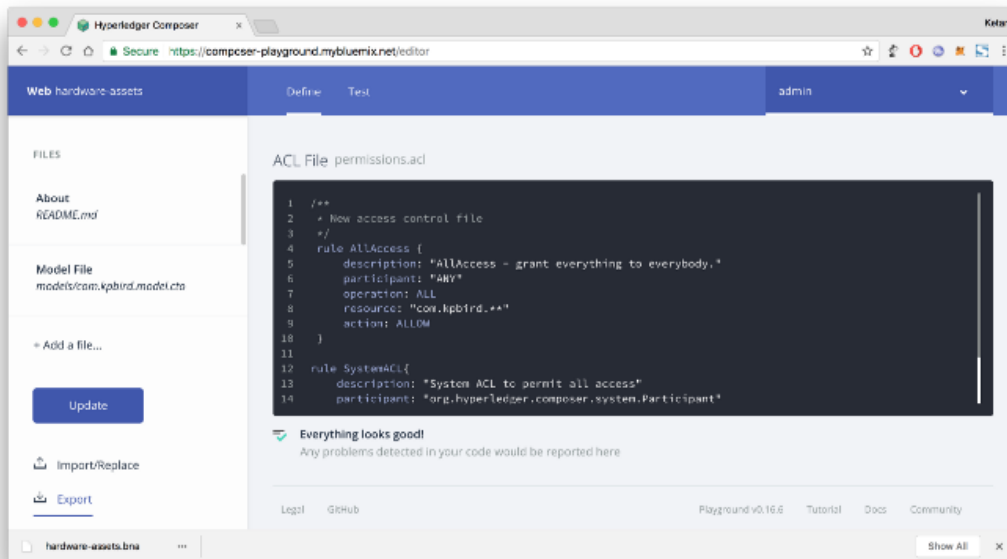


Step 12: Click on “All Transactions” from left panel to verify all transactions. In following screenshots you can see add assets, ass participants and allocation all operation are consider as transactions. “view records” will give us more information about transaction.



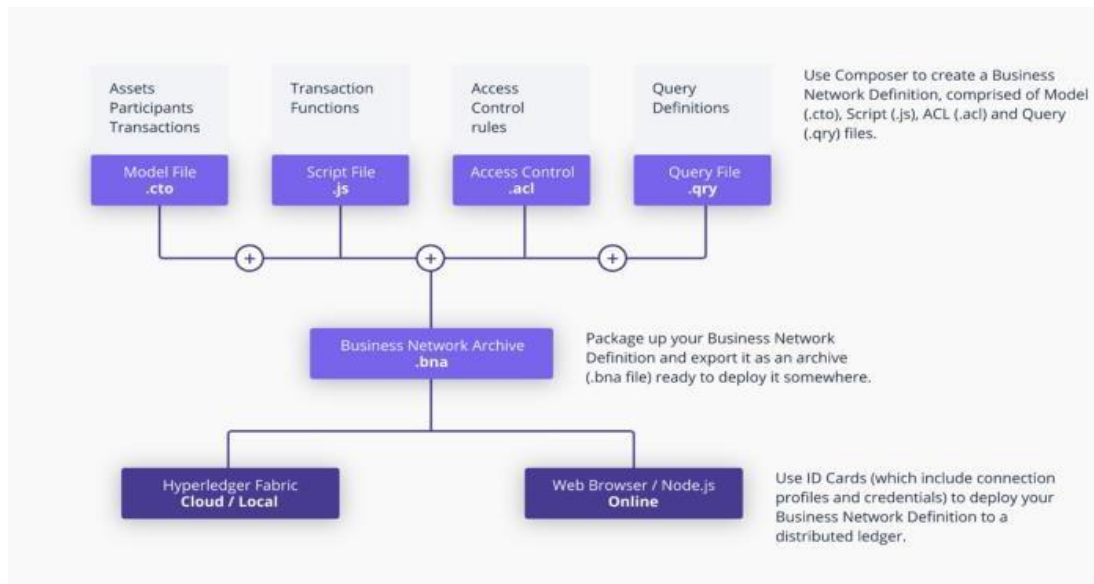
All Transactions

Step 13: Now, It's time to deploy “hardware-assets” business network to Hyperledger Fabric. Click on “Define” tab from top panel and click “Export” button from left panel. Export will create hardware-assets.bna file.



Download hardware-assets.bna file

.bna is Business Network Archive file which contains model, script, network access and query file



source: <https://hyperledger.github.io/composer/introduction/introduction>

Step 14: Start Docker and run following commands from ~/fabric-tools directory

Install business network to Hyperledger Fabric, If business network is already installed you can use “update” instead of “install”

```
$composer runtime install -c PeerAdmin@hlfv1 -n hardware-assets
```

```
Ketan-Parmar:~$ composer runtime install -c PeerAdmin@hlfv1 -n hardware-assets
```

```
✓ Installing runtime for business network hardware-assets. This may take a minute...
```

```
Command succeeded
```

Following command will deploy and start hardware-assets.bna file. Change hardware-assets.bna file before you execute following command. networkadmin.card file will generate in ~/fabric-tools directory from previous command.

```
$composer network start --card PeerAdmin@hlfv1 --networkAdmin admin --networkAdminEnrollSecret adminpw --archiveFile /Users/ketan/Downloads/hardware-assets.bna --file networkadmin.card
```

```
Ketan-Parmar:~$ composer network start --card PeerAdmin@hlfv1 --networkAdmin admin --networkAdminEnrollSecret adminpw --archiveFile /Users/ketan/Downloads/hardware-assets.bna --file networkadmin.card
```

```
Starting business network from archive: /Users/ketan/Downloads/hardware-assets.bna
```

```
Business network definition:
```

```
Identifier: hardware-assets@0.0.1
```

```
Description: Hardware Assets will maintain Software company's hardware
```

```
Processing these Network Admins:
```

```
username: admin
```

```
✓ Starting business network definition. This may take a minute...
```

```
Successfully created business network card:
```

```
Filename: networkadmin.card
```

```
Command succeeded
```

To connect business network you need connection card. so we can import networkadmin.card using following command

```
$composer card import -f networkadmin.card
```

To make sure networkadmin.card successfully install you can list cards using following command

```
$composer card list
```

Ketan-Parmar:fabric-tools ketan\$ composer card list
 The following Business Network Cards are available:

Connection Profile: hlfv1

Card Name	UserId	Business Network
admin@hardware-assets	admin	hardware-assets
PeerAdmin@trade-network	PeerAdmin	trade-network
admin@trade-network	admin	trade-network
PeerAdmin@hlfv1	PeerAdmin	

Issue `composer card list --name <Card Name>` to get details a specific card

Command succeeded

Following command will make sure that our hardware-assets business network is successfully running in Hyperledger Fabric.

```
$composer network ping --card admin@hardware-assets
```

```
Ketan-Parmar:fabric-tools ketan$ composer network ping --card admin@hardware-assets
The connection to the network was successfully tested: hardware-assets
version: 0.16.0
participant: org.hyperledger.composer.system.NetworkAdmin#admin
```

Command succeeded

Now It's time to interact with REST API. To develop Web or Mobile Application we require REST API. you can run following command to generate REST API for hardware-assets business network.

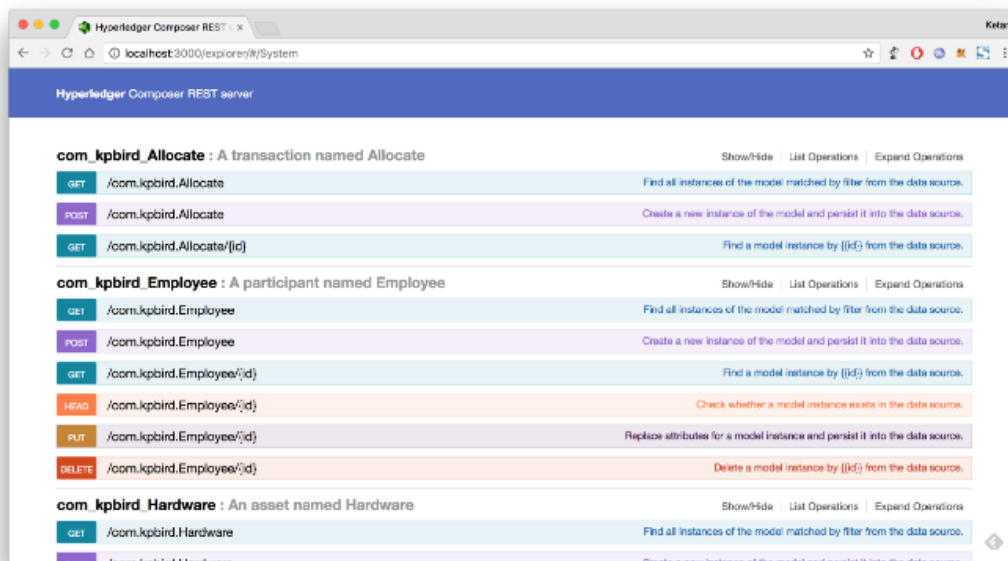
```
$composer-rest-server
```

```
Ketan-Parmar:fabric-tools ketan$ composer-rest-server
? Enter the name of the business network card to use: admin@hardware-assets
? Specify if you want namespaces in the generated REST API: always use namespaces
? Specify if you want to enable authentication for the REST API using Passport: No
? Specify if you want to enable event publication over WebSockets: Yes
? Specify if you want to enable TLS security for the REST API: No
```

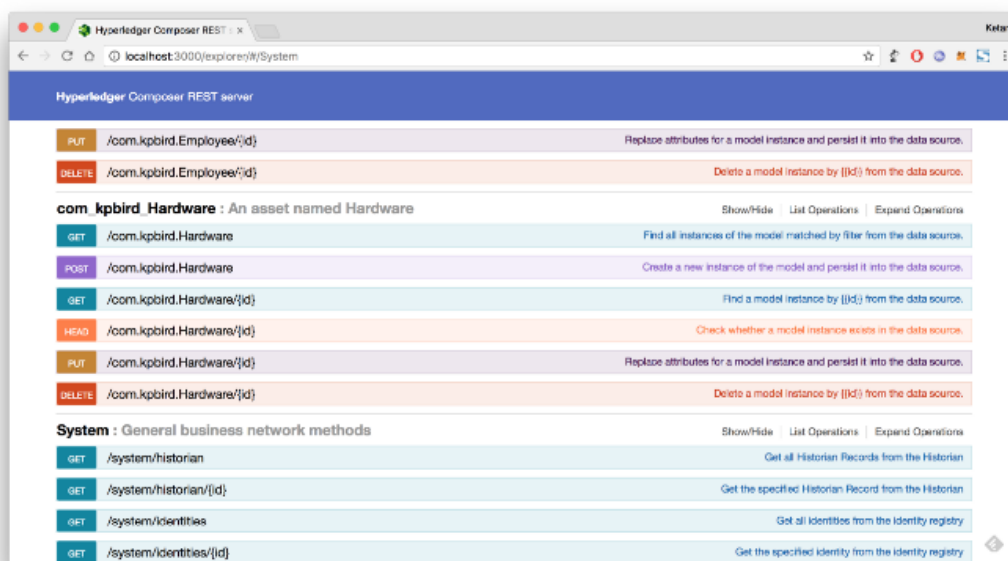
To restart the REST server using the same options, issue the following command:
`composer-rest-server -c admin@hardware-assets -n always -w true`

```
Discovering types from business network definition ...
Discovered types from business network definition
Generating schemas for all types in business network definition ...
Generated schemas for all types in business network definition
Adding schemas for all types to Loopback ...
Added schemas for all types to Loopback
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

rest server will ask few basic information before generate rest api



REST API for our hardware assets



REST API methods for all operations

Conclusion: In this way we have learnt about hyperledger and its use case in business world.

Assignment No : 18

MINI PROJECT 3

Code :- <https://github.com/sherwyn11/E-Voting-App>