# 0/1 Knapsack Visualization

**PROJECT REPORT**

# 4th semester, B. TECH COMPUTER SCIENCE ENGINEERING

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY**

**MAY,2024**

Submitted to –

**Dr. Diksha Hooda**

Submitted by-

**Vansh Seth (221030133) CS-45**

**Aditya Sharma (221030130) CS-45**

**Sanya Jain (221030048) CS-45**

**Aashi Gupta (221030140) CS-45**

# ABSTRACT



The Knapsack Visualizer is a software tool designed to aid in understanding and solving the Knapsack Problem, a classic combinatorial optimization problem. This visualizer provides an interactive and intuitive interface for users to explore different algorithms and strategies for solving the 0/1 Knapsack Problem. The tool supports various algorithms, including dynamic programming. By visualizing the process of filling the knapsack, the tool helps in comprehending the underlying mechanics of each algorithm. The Knapsack Visualizer is implemented in JS. This educational tool is particularly useful for students, educators, and anyone interested in algorithmic problem-solving and computer science concepts

# PROBLEM STATEMENT



The Knapsack Visualizer is designed to aid in understanding and solving the Knapsack Problem, specifically the 0/1 Knapsack Problem. This classic combinatorial optimization problem involves maximizing profit while adhering to a weight constraint. The tool provides an interactive interface for users to input item weights, profits, and the maximum weight the knapsack can carry. It then visualizes the process of filling the knapsack using dynamic programming algorithms, helping users comprehend the underlying mechanics of each approach.

# OBJECTIVES

The Knapsack Visualizer helps visualize the dynamic programming (DP) solution for the 0/1 Knapsack problem, allowing users to maximize profit based on given item weights, profits, and maximum carriable weight.

• To visualize the Binary knapsack algorithm.

• To allow users to interactively input data and witness the algorithm's step-by-step execution.

• To enhance understanding of the Binary knapsack problem through dynamic animations.

# WORKING/METHODOLOGY

**Steps:**

1. Input Fields:

* Users enter the weights and profits of items along with the maximum carriable weight..
* These inputs are gathered via HTML forms and processed using JavaScript.

1. Dynamic Programing table Creation:

* The visualizer constructs a DP table based on the input values.
* It uses the standard DP approach where the table is built iteratively to find the maximum profit without exceeding the weight limit

1. Visualization:

* The DP table construction is animated to help users understand each step.
* Users can control the speed of this animation for better comprehension.

1. Output Display:

* The maximum profit is displayed prominently.
* The completed DP table is shown, highlighting the chosen items that contribute to the maximum profi

# IMPLEMENTATION

## 

The project is built using HTML, CSS, and JavaScript to develop an interactive web application for visualizing the knapsack problem. Here are the key components:

* HTML: Defines the structure of the web page, including elements such as containers, buttons, and input fields.
* CSS: Styles the visual elements to enhance the user interface and improve readability.
* JavaScript: Implements the logic for dynamic data handling, animation, and interaction with the user interface. This includes algorithms for solving the knapsack problem and updating the visual representation based on user input.

**CODE**

**Index.html**

<!DOCTYPE html>

<html>

<head>

    <meta charset='utf-8'>

    <meta http-equiv='X-UA-Compatible' content='IE=edge'>

    <title>Knapsack Visualiser</title>

    <meta name='viewport' content='width=device-width, initial-scale=1'>

    <link rel='stylesheet' type='text/css' media='screen' href='index.css'>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

    <script src='index.js'></script>

</head>

<body>

    <!-- Header -->

    <div class="headarea">

        <h1>0/1 Knapsack Algorithm Visualiser</h1>

    </div><hr/>

    <!-- Input Section -->

    <div class="row g-5">

        <div class="col-1"></div>

        <div class="col-auto">

            <label class="form-label" for="profit">Enter Profit Array</label>

            <input class="form-control p-2 shadow rounded" placeholder="Ex: 3 2 4 5" type="text" id="profit">

        </div>

        <div class="col-auto">

            <label class="form-label" for="weight">Enter Weight Array</label>

            <input class="form-control p-2 shadow rounded" placeholder="Ex: 4 1 2 6" type="text" id="weight">

        </div>

        <div class="col-auto">

            <label class="form-label" for="maxw">Enter Max Weight we can carry.</label>

            <input class="form-control p-2 shadow rounded" placeholder="Ex: 12" type="number" id="maxw">

        </div>

        <div class="col-1"></div>

        <div class="col-2 mx-5">

            <label class="form-label" for="speed">Animation Speed:</label>

            <input class="form-range shadow" type="range" min=250 max=1000 id="speed">

        </div>

    </div>

    <!-- Data Area -->

    <div id="data">

        <h3 id="ans"></h3>

    </div>

    <div id="data2">

        <table id="table"></table>

    </div>

    <div class="customB">

        <button class="btn-lg btn-primary px-4" type="submit" onclick="start()">Start</button>

    </div>

</body>

</html>

**Index.css**

body{

    background-image: url("./images/cover.png");

    background-size: cover;

    overflow-x: hidden;

    scroll-behavior: smooth;

}

.headarea h1{

    margin: 10px 0px;

    text-align: center;

}

#speed{

    transform: rotateY(180deg);

    background-color: white;

    margin-top: 10px;

    padding: 0px 6px;

    border-radius: 15px;

}

#ans{

    background-color: rgba(44, 86, 47, 0.8);

    color: white;

    width: max-content;

    height: max-content;

    padding: 10px;

    border-radius: 10px;

}

#data, #data2{

    width: max-content;

    height: max-content;

    margin: auto;

    margin-top: 20px;

    transition-duration: 1s;

}

#data2 table tr td{

    background-color: rgba(153, 255, 153, 0.95);

    text-align: center;

    font-size: large;

    font-weight: bold;

    height: 60px;

    width: 60px;

    border: 2px solid black;

}

.customB{

    margin: auto;

    margin-top: 15px;

    width: fit-content;

}

.cola{

    background-color: rgba(44, 86, 47, 1) !important;

    color: white;

}

.colb{

    background-color: rgba(255, 223, 13, 0.95) !important;

}

.hide{

    visibility: hidden;

}

**Index.js**

function start(){

    let p=document.getElementById("profit").value;

    let w=document.getElementById("weight").value;

    let maxw=document.getElementById("maxw").value;

    if(p && w && maxw){

        maxw=parseInt(maxw);

        p=p.split(" ");

        w=w.split(" ");

        for (let i = 0; i < w.length; i++) {

            w[i] = parseInt(w[i]);

            p[i] = parseInt(p[i]);

        }

        let n=w.length;

        knapsack(n,w,p,maxw);

        setTimeout(() => {

            animate(n,maxw,w)

        }, 1000);

    }

    else{

        alert("Please Fill All Fields!");

    }

}

function knapsack(n, w, p, maxw){

    let dp=new Array(n+1);

    for (let i = 0; i < n+1; i++) {

        dp[i] = new Array(maxw+1);

        dp[i].fill(0);

    }

    for(let i = 1; i < n+1; i++){

        for(let j = 1; j < maxw+1; j++){

            if(w[i-1] <= j){

                dp[i][j] = Math.max(dp[i-1][j], p[i-1] + dp[i-1][j-w[i-1]]);

            }

            else{

                dp[i][j] = dp[i-1][j];

            }

        }

    }

    createTable(dp, n, maxw);

    let str="Max Profit: "+dp[n][maxw]+" | Included Item ";

    let temp=dp[n][maxw];

    for (let i = n; i > 0; i--){

        if(temp == dp[i-1][maxw]){

            continue;

        }

        else{

            str += i + " ";

            maxw -= w[i-1];

            temp -= p[i-1];

        }

    }

    const data=document.getElementById("data");

    data.removeChild(document.getElementById("ans"));

    let ans = document.createElement("h3");

    let text=document.createTextNode(str);

    ans.id="ans";

    ans.appendChild(text);

    data.appendChild(ans);

}

function createTable(dp, n, maxw){

    const data=document.getElementById("data2");

    if(document.getElementById("table")){

        data.style.opacity=0;

        setTimeout(() => {

            data.removeChild(document.getElementById("table"));

        }, 1000);

    }

    let table=document.createElement("table");

    for (let i = 0; i < n+1; i++) {

        const row = document.createElement("tr");

        for (let j = 0; j < maxw+1; j++) {

            const cell = document.createElement("td");

            const val = document.createTextNode(dp[i][j]);

            cell.appendChild(val);

            cell.classList.add("hide");

            row.appendChild(cell);

        }

        table.appendChild(row);

    }

    table.id="table";

    setTimeout(() => {

        data.appendChild(table);

        data.style.opacity=1;

    }, 1000);

}

function animate(n,maxw,w){

    let table = document.getElementById("table");

    let speed = parseInt(document.getElementById("speed").value);

    for(let i=0;i<n+1;i++){

        let cell = table.getElementsByTagName("tr")[i].getElementsByTagName("td")[0];

        cell.classList.remove("hide");

    }

    for(let j=0;j<maxw+1;j++){

        let cell = table.getElementsByTagName("tr")[0].getElementsByTagName("td")[j];

        cell.classList.remove("hide");

    }

    for (let i = 1; i < n+1; i++) {

        for (let j = 1; j < maxw+1; j++) {

            if(w[i-1]<=j){

                let currentCell = table.getElementsByTagName("tr")[i].getElementsByTagName("td")[j];

                let cell1 = table.getElementsByTagName("tr")[i-1].getElementsByTagName("td")[j];

                let cell2 = table.getElementsByTagName("tr")[i-1].getElementsByTagName("td")[j-w[i-1]];

                setTimeout(() =>{

                    currentCell.classList.add("cola");

                    cell1.classList.add("colb");

                    cell2.classList.add("colb");

                    currentCell.classList.remove("hide");

                }, (speed\*(maxw+1)\*(i-1))+speed\*(j-1));

                setTimeout(() =>{

                    currentCell.classList.remove("cola");

                    cell1.classList.remove("colb");

                    cell2.classList.remove("colb");

                }, (speed\*(maxw+1)\*(i-1))+speed\*(j-1) + speed);

            }

            else{

                let currentCell = table.getElementsByTagName("tr")[i].getElementsByTagName("td")[j];

                let cell = table.getElementsByTagName("tr")[i-1].getElementsByTagName("td")[j];

                setTimeout(() =>{

                    currentCell.classList.add("cola");

                    cell.classList.add("colb");

                    currentCell.classList.remove("hide");

                }, (speed\*(maxw+1)\*(i-1))+speed\*(j-1));

                setTimeout(() =>{

                    currentCell.classList.remove("cola");

                    cell.classList.remove("colb");

                }, (speed\*(maxw+1)\*(i-1))+speed\*(j-1) + speed);

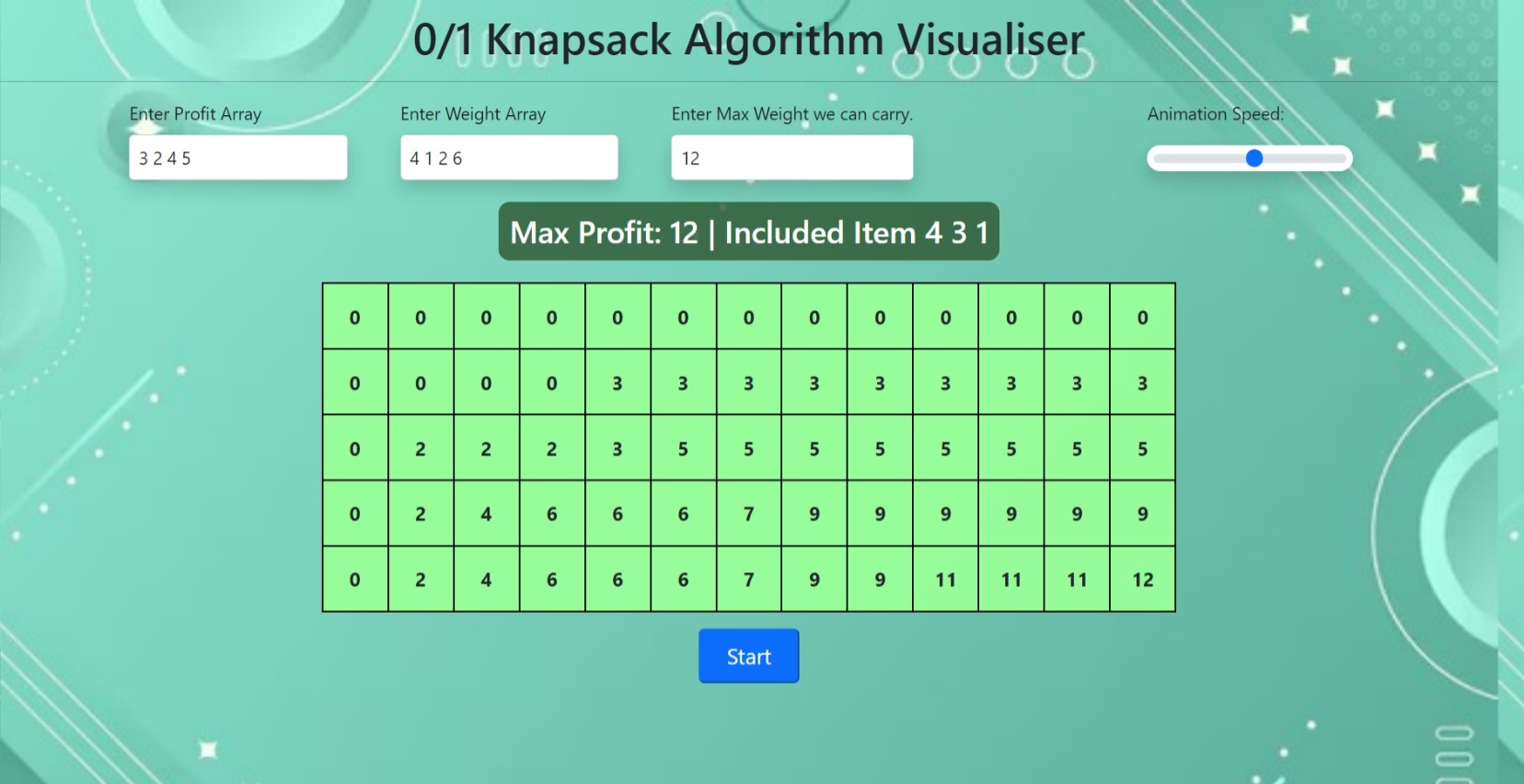
            }

        }

    }

}

**OUTPUT**

****

**TIME AND SPACE ANALYSIS**



* Time Complexity: The time complexity of the dynamic programming solution for the 0/1 Knapsack Problem is O(n \* W), where n is the number of items and W is the maximum weight the knapsack can carry. This is because the algorithm fills up a DP table of size (n+1) \* (W+1) iteratively.
* Space Complexity: The space complexity of the dynamic programming solution is also O(n \* W) as it requires storing a DP table of the same size.

**RESULT ANALYSIS**



The application successfully animates the process of the fractional knapsack algorithm. Users can:

* Input custom data.
* Choose sorting algorithms.
* Observe the algorithm's behavior through animation

**CONCLUSION**



The "Animation Code for Fractional Knapsack" project effectively demonstrates the fractional knapsack problem through an interactive and educational web application. It serves as a useful tool for students and educators to visualize and understand this classic algorithm.