BRUTE FORCE

LEAGUE

by Karl Apolonio

Pretim

OUTPUT PORTFOLIO

A real world problem about Brute Force Algorithm

LIST OF CONTENTS

DESCRIPTION OF THE PROBLEM

The problem involves creating a sports league schedule that avoids any game overlaps, ensures teams have 1-2 days of rest between games, avoid scheduling two teams to play on the same day, while also accommodating specific rules for conference and interconference matchups.

ALGORITHM

A brute force algorithm solves a problem by trying all possible solutions without employing any shortcuts or heuristic methods. It is straightforward but often inefficient, especially for large problem spaces.

```
function generateMatchups(conferenceTeams) {
    let matchups = [];
    for (let i = 0; i < conferenceTeams.length; i++) {
        for (let j = i + 1; j < conferenceTeams.length; j++) {
            matchups.push([conferenceTeams[i], conferenceTeams[j]],
                [conferenceTeams[i], conferenceTeams[j]]);
            matchups.push([conferenceTeams[j], conferenceTeams[i]],
                [conferenceTeams[j], conferenceTeams[i]]);
        }
    }
    return matchups;
}
```

CODE IMPLEMENTATION (JS)

```
function shuffle(array) {
    for (let i = array.length - 1; i > 0; i--) {
        const j = Math.floor(Math.random() * (i + 1));
        [array[i], array[j]] = [array[j], array[i]];
    }
}

function generateMatchups(conferenceTeams) {
    let matchups = [];
    for (let i = 0; i < conferenceTeams.length; i++) {
        for (let j = i + 1; j < conferenceTeams.length; j++) {
            matchups.push([conferenceTeams[i], conferenceTeams[j]],
                [conferenceTeams[i], conferenceTeams[j]]);
            matchups.push([conferenceTeams[j], conferenceTeams[i]],
                [conferenceTeams[j], conferenceTeams[i]]);
        }
    }
    return matchups;
}
```

```
function crossConferenceMatchups(east, west)
{
    let matchups = [];
    for (let e of east) {
        for (let w of west) {
            matchups.push([e, w], [w, e]);
        }
    }
    return matchups;
}
```

CODE IMPLEMENTATION

```
function createSchedule(east, west, teams) {
    let schedule = [];
    const eastMatchups = generateMatchups(east);
    const westMatchups = generateMatchups(west);
    const crossMatchups = crossConferenceMatchups(east, west);

    let allGames = eastMatchups.concat(westMatchups).concat(crossMatchups);
    shuffle(allGames);

    let scheduleDate = new Date();
    const gamesPerDay = 5;
    let restDaysTeam = {};
    teams.forEach(team => restDaysTeam[team] =
        new Date(scheduleDate.getTime() - 86400000));
    // Set to 'yesterday'
```

CODE IMPLEMENTATION

```
while (allGames.length > 0) {
    let dayGames = [];
    for (let i = 0; i < gamesPerDay && allGames.length > 0; i++) {
        allGames = allGames.filter(game => {
            const lastPlayedHome = restDaysTeam[game[0]];
            const lastPlayedAway = restDaysTeam[game[1]];
            if (scheduleDate > new Date(lastPlayedHome.getTime() + 86400000)
            && scheduleDate > new Date(lastPlayedAway.getTime() + 86400000))
            {
                dayGames.push(game);
                restDaysTeam[game[0]] = new Date(scheduleDate);
                restDaysTeam[game[1]] = new Date(scheduleDate);
                return false;
            }
            return true;
        });
        if (dayGames.length === gamesPerDay) break;
    }
```

```
if (dayGames.length > 0) {
    dayGames.forEach(game =>
        schedule.push({
    "date": new Date(scheduleDate),
    "matchup": game }));
}
scheduleDate = new Date(
    scheduleDate.getTime()
+ 86400000); // Increment day
    }
return schedule;
}
```

CODE IMPLEMENTATION

```
function displaySchedule(schedule) {
    const scheduleDiv = document.getElementById("scheduleDiv");
    scheduleDiv.innerHTML = ""; // Clear existing content

    // Create a table and a header row
    const table = document.createElement("table");
    table.className = "schedule-table"; // Optional: for styling
    const headerRow = document.createElement("tr");
    const headers = ["DATE", "HOME", "VS", "AWAY"];
    headers.forEach(headerText => {
        const headerCell = document.createElement("th");
        headerCell.textContent = headerText;
        headerRow.appendChild(headerCell);
    });
    table.appendChild(headerRow);
```

```
schedule.forEach(game => {
    const dateStr = game.date.toISOString().substring(0, 10);
    const row = document.createElement("tr");

    // Date cell
    const dateCell = document.createElement("td");
    dateCell.textContent = dateStr;
    row.appendChild(dateCell);

    // Home team logo cell
    const homeLogoCell = document.createElement("td");
    const homeLogo = document.createElement("img");
    homeLogo.src = `img/${game.matchup[0].toLowerCase()}-logo.png`;
    homeLogo.alt = game.matchup[0];
    homeLogo.className = "team__logo";
    homeLogoCell.appendChild(homeLogo);
    row.appendChild(homeLogoCell);
```

CODE IMPLEMENTATION

```
const vsCell = document.createElement("td");
vsCell.textContent = "VS";
row.appendChild(vsCell);

// Away team logo cell
const awayLogoCell = document.createElement("td");
const awayLogo = document.createElement("img");
awayLogo.src = `img/${game.matchup[1].toLowerCase()}-logo.png`;
awayLogo.alt = game.matchup[1];
awayLogo.className = "team__logo";
awayLogoCell.appendChild(awayLogo);
row.appendChild(awayLogoCell);

// Append the row to the table
table.appendChild(row);
});

// Append the table to the scheduleDiv
scheduleDiv.appendChild(table);
}
```

CODE IMPLEMENTATION

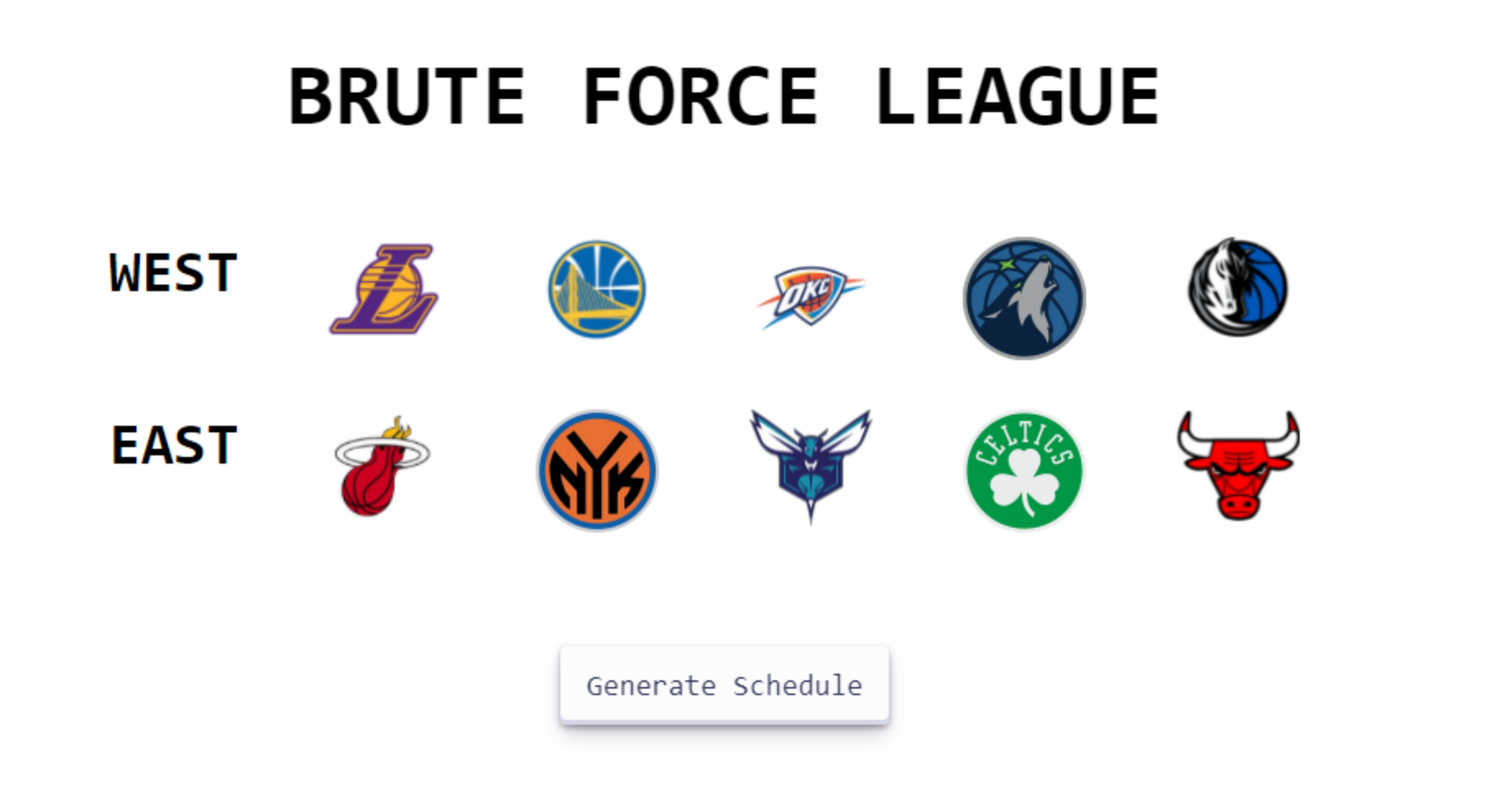
```
function generateAndDisplaySchedule() {
    const east = ["Celtics", "Heat", "Knicks", "Bulls", "Hornets"];
    const west = ["Lakers", "Mavericks", "OKC", "Warriors", "Wolves"];
    const teams = east.concat(west);
    const schedule = createSchedule(east, west, teams);
    displaySchedule(schedule);
}

document.getElementById("generateSchedule").addEventListener("click", generateAndDisplaySchedule);
```

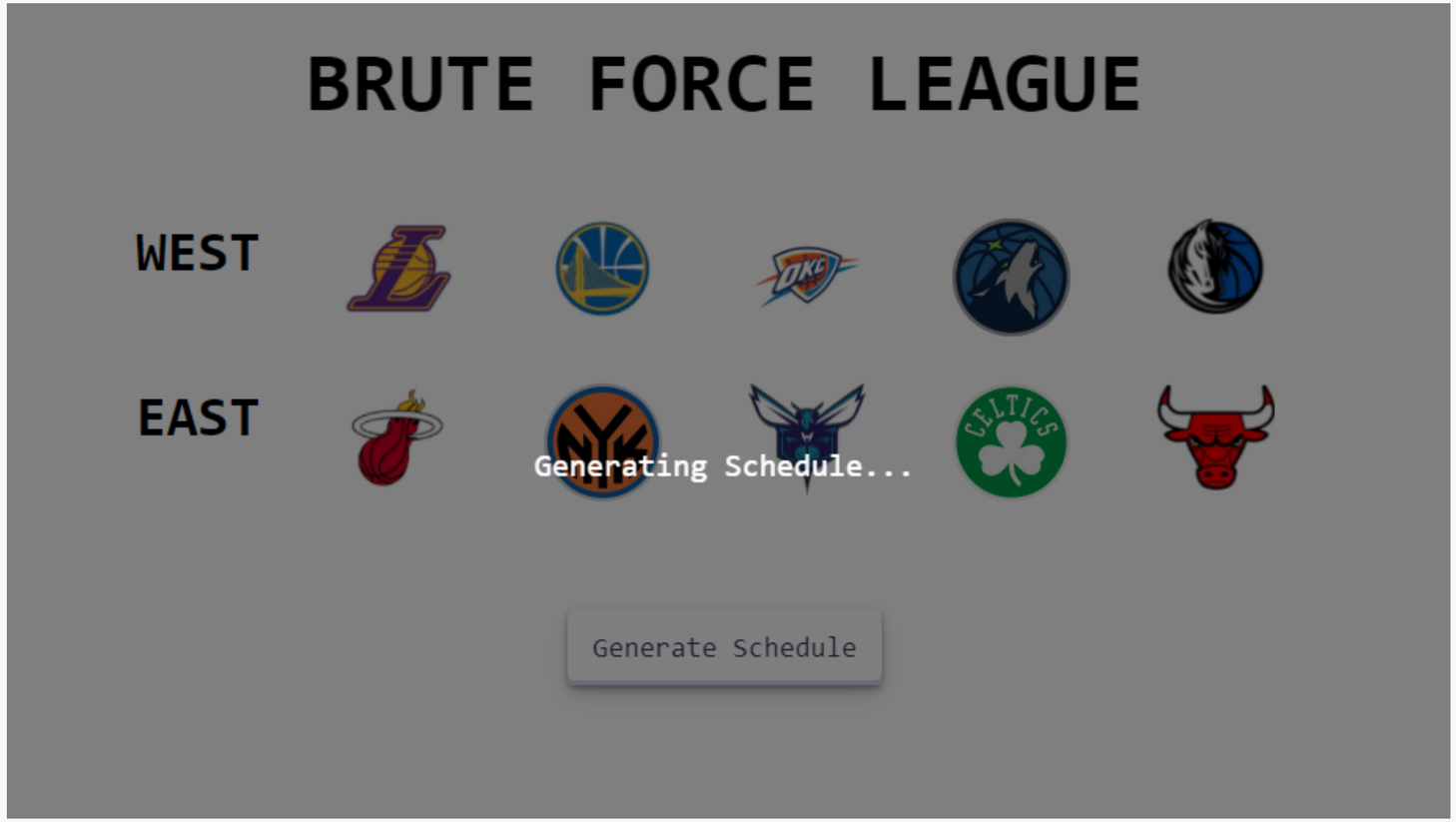
CODE EXPLANATION

This JavaScript code includes functions for creating and displaying a basketball schedule for a league with two conferences (East and West) and several teams in each. The **generateMatchups** method generates pairings inside a conference, whereas **crossConferenceMatchups** pairs teams from separate conferences. The **createSchedule** function creates a schedule with given games per day using a round-robin approach that takes into account teams' rest days between games. Finally, the **displaySchedule** function dynamically constructs an HTML table to visually present the schedule, while the **generateAndDisplaySchedule** function configures an event listener to trigger schedule generation and display when a button is clicked.

SCREENSHOT OF THE OUTPUT



SCREENSHOT OF THE OUTPUT



SCREENSHOT OF THE OUTPUT

BRUTE FORCE SCHEDULE

Optimized Scheduling Algorithm Features:

- **Conflict-Free Scheduling**: Meticulously designed to prevent scheduling overlaps.
- **Guaranteed Rest Periods**: Each team enjoys a break of **1-2 days** between games, ensuring optimal performance.
- **Unique Game Days**: No two teams will play on the same day, promoting fairness and viewer engagement.
- **Conference Matchups**:
  - Teams within the same conference compete **four times**.
  - Interconference matchups are limited to **two games** per team.
- **Refresh** to generate again.

DATE	HOME	VS	AWAY
2024-03-09		VS	
2024-03-09		VS	

SCREENSHOT OF THE OUTPUT

2024-03-17		VS	
2024-03-17		VS	
2024-03-17		VS	
2024-03-19		VS	
2024-03-19		VS	

TIME COMPLEXITY EXPLANATION

Generating Matchups:

- The generateMatchups function creates all possible pairings within each conference, resulting in **O(n^2)** matchups for each conference, where **n** is the number of teams in the conference.
- The crossConferenceMatchups function generates matchups between teams from different conferences, resulting in **O(n^2)** matchups if there are n teams in each conference.

TIME COMPLEXITY EXPLANATION

Combining Matchups:

- The matchups from each conference and the cross-conference matchups are combined into a single array, resulting in **O(n^2)** total matchups if **n** is the number of teams in each conference.

Shuffling Matchups:

- The shuffle function randomizes the order of the matchups. The shuffling process takes **O(n)** time.

TIME COMPLEXITY EXPLANATION

Scheduling Games:

- The createSchedule function iterates through the shuffled matchups, attempting to schedule games while considering rest day constraints.
- In the worst case, the filtering process within the loop can approach **O(m^2)**, where **m** is the total number of games. This is due to the need to check rest day conditions for each game and the possibility of revisiting previously scheduled games.

Displaying Schedule:

- The displaySchedule function iterates through the generated schedule and performs constant-time operations for each game, resulting in **O(k)** time complexity, where **k** is the number of games in the schedule.

TIME COMPLEXITY EXPLANATION

The **Brute Force Algorithm** has a time complexity dominated by the scheduling phase, which is **O(m^2)** in the worst case where **m** depends on the number of teams in each conference. The total time complexity is influenced by the number of teams **(n)** and how many matchups are generated **(O(n^2))**, making it computationally expensive as the number of teams increases. The efficiency can be improved by exploring more optimized scheduling algorithms, especially for larger leagues.

LEARNING REFLECTION

During my preliminary period, I focused on the study of complex algorithms, such as the brute force approach and the traveling salesman problem, which first tested my problem-solving abilities due to their computational requirements and intricate logic.

To address these challenges, I engaged in hands-on practice and explored resources outside of the classroom, such as tutorials which improved my comprehension and application abilities.

The key takeaway from this period has been the importance of perseverance and resourcefulness when dealing with complex topics and a skill set that is essential in many aspects of computer science.