WORD SEARCH WITH DFS VISUALIZATION

A PROJECT REPORT

SUBMITTED BY

AMAN AHLAWAT(2K19/IT/012)

ARIHANT SHOKEEN(2K19/IT/031)

SUBMITTED TO

Mrs. Swati Sharda Mam

DEPARTMENT OF INFORMATION TECHNOLOGY

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our teacher Prof. Swati Sharda who gave us the golden opportunity to do this wonderful project on the topic: - Word Search Problem.

This project helped us in understanding graphs concepts better and we learnt about many new things. We would also like to thank our university, Delhi Technological University for giving us this opportunity to explore and research. We would also like to thank our peers and teacher for making this subject interesting and fun to learn. Thanking you,

Aman Ahlawat (2K19/IT/012)

Arihant Shokeen (2K19/IT/031)

CERTIFICATE

I hereby certify that the Project Dissertation titled “WORD SEARCH PROBLEM” which is submitted by Aman Ahlawat (2K19/IT/012) and Arihant Shokeen (2K19/IT/031); INFORMATION TECHNOLOGY, Delhi Technological University, Delhi, is a record of the project work carried out by the students under my supervision.

Place: Delhi

Date: 30-12-2020

ABSTRACT

Word Search Problem is a web development based project which allows people to learn and implement the concept of depth first search (dfs) algorithm by applying it to a problem of solving a word search. It demonstrates the

This project allows users to solve word search by themselves, displaying the solution to the puzzle, and to see how each word in the word search is found by applying our dfs algorithm.

The objective of this project is to show how much easy and interesting learning becomes when we find applications for our learned concepts in day to day life and visualize the things which we can apply as human brain tends to process visual information far more easily than written information.

Table of Contents

INTRODUCTION

A word search, word find, word seek, word sleuth or mystery word puzzle is a word game that consists of the letters of words placed in a grid, which usually has a rectangular or square shape. The objective of this puzzle is to find and mark all the words hidden inside the box. The words may be placed horizontally, vertically, or diagonally.

The problem statement becomes:

"Given a list of words and a grid of word search puzzle, finding all the words which are present in the list by using an algorithm.

We use DFS algorithm for solving our puzzle as ‘Depth first Search’ is a recursive algorithm for searching all the vertices of a graph or tree data structure.

We will use Trie data structure to build a tree with the list of words which we have to search in the puzzle as it will help us finding all the words in a single dfs run. It is data structure, a type of search tree used to store associative data structures.

PURPOSE

The Word Search problem is a popular word game that consists of the letters of words placed in a grid, which usually has a rectangular or square shape.

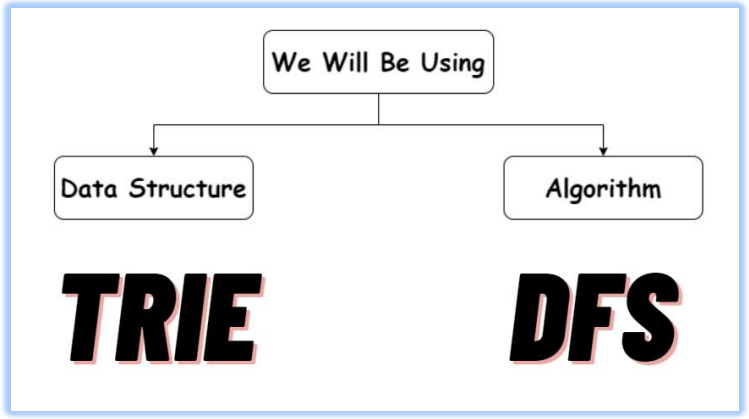
The main purpose and motive of coming up with this problem statement and project is all about visualization learning.

Why do we create visualizations?

- Answer a question
- Make decisions
- Analyze and discover

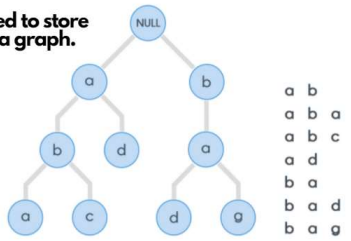
Visualization techniques can be used to enhance various activities during the learning process: finding and understanding educational resources, collaboration with learners and teachers, (self-) reflecting about learners' progress, and designing learning experiences. We illustrate our analysis with example tools and visualizations.

COMPONENTS



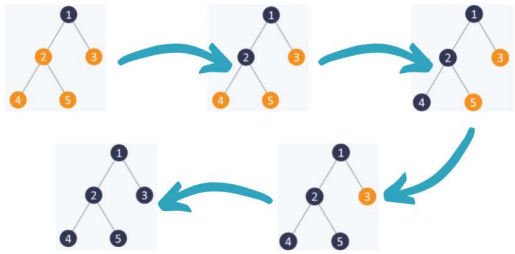
TRIE

A Trie is a special data structure used to store strings that can be visualized like a graph.



DFS (DEPTH FIRST SEARCH)

The DFS algorithm is a recursive algorithm that uses the idea of backtracking.



REQUIREMENT DESCRIPTION

TECHNOLOGIES USED :

JAVASCRIPT :

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions.JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles.
JavaScript can function as both a procedural and an object oriented language. Objects are created programmatically in JavaScript.

HTML:

HTML (Hypertext Markup Language) is a text-based approach to describing how content contained within an HTML file is structured. This markup tells a web browser how to display text, images and other forms of multimedia on a webpage.

CSS:

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, and enable multiple web pages to share formatting

TOOLS USED :

VS-CODE:

Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

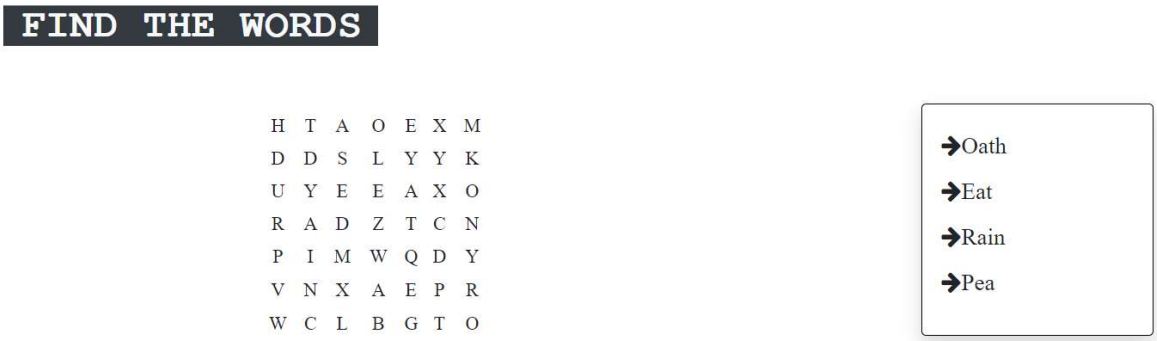
CHROME BROWSER:

Google Chrome is a cross-platform web browser developed by Google. It was first released in 2008 for Microsoft Windows, and was later ported to Linux, macOS, iOS, and Android where it is the default browser built into the OS.

APPLICATION SECTIONS

SOLVING WORD SEARCH :

The user can enjoy solving the word search puzzle provided in the first section of our application. Words list is present on the right hand side of the grid.



DISPLAY SOLUTION:

Clicking the 

button will display the solution to our puzzle which is backed by trie and dfs algorithm.

Before Click:

After Click:

SOLUTION

H	T	A	O	E	X	M
D	D	S	L	Y	Y	K
U	Y	E	E	A	X	O
R	A	D	Z	T	C	N
P	I	M	W	Q	D	Y
V	N	X	A	E	P	R
W	C	L	B	G	T	O

Show Solution

DISPLAY VISUALIZATION :

This section is the most important part of our project, where the user witnesses the visualization of each and every word in our puzzle.

Before Visualization

VISUALIZATION

Depth-First Search

H	T	A	O	E	X	M
D	D	S	L	Y	Y	K
U	Y	E	E	A	X	O
R	A	D	Z	T	C	N
P	I	M	W	Q	D	Y
V	N	X	A	E	P	R
W	C	L	B	G	T	O

Select the word and click play ▶

- Oath
- Eat
- Rain
- Pea

After Visualization

VISUALIZATION

Depth-First Search

H	T	A	O	E	X	M
D	D	S	L	Y	Y	K
U	Y	E	E	A	X	O
R	A	D	Z	T	C	N
P	I	M	W	Q	D	Y
V	N	X	A	E	P	R
W	C	L	B	G	T	O

Select the word and click play ▶

- Oath
- Eat
- Rain
- Pea

CODE

GITHUB LINK:

https://github.com/ahlawataman/Word-Search-with-DFS-Visualization

CODE SNIPPETS

CLASS NODE

```
class TrieNode {
    constructor(val){
        this.val = val;  // LETTER STORED IN NODE
        this.children = {};  // DICTIONARY OF NODE'S CHILDREN
        this.endshere = false; // BOOL VALUE TO CHECK WORD'S END
    }
}
```

METHOD TO INITIALIZE

```
class Trie {
    constructor(){
        this.root = new TrieNode(null);
    }  // CONSTRUCTOR TO INITIALIZE OUR TREE
    insert(word){
        var main = this.root;
        for(var i=0; i<word.length; i++)
        {
            if(Object.keys(main.children).indexOf(word[i])==-1)
            {
                main.children[word[i]] = new TrieNode(word[i]);
            }
            main = main.children[word[i]];
        }
        main.endshere = true;
    } // FUNCTION TO BUILD OUR TRIE(TREE)
}
```

DFS FUNCTION (SECTION 2)

```
function dfs(board, node, i, j, path, res, mydict, listCoordinate){
    if(node.endshere) {
        res.push(path)  // APPENDS THE WORD IF IT ENDS HERE
        mydict[path] = [];
        mydict[path].push.apply(mydict[path],listCoordinate);
        node.endshere = false;
    }  // CHECKS IF WORD ENDS HERE OR NOT

    // BASE CONDITIONS FOR RECURSION
    if(i < 0 || i >= board.length || j < 0 || j >= board[0].length) return ;

    var tmp = board[i][j];  // PICKING UP A LETTER
    if(Object.keys(node.children).indexOf(tmp)==-1) return;  // CHECKING CHILDREN TO ADVANCE THE SEARC
    node = node.children[tmp];  // MOVING THE POINTER TO CHILD NODE
    board[i][j] = '#'  // HASHING THE VISITED NODE
    listCoordinate.push([i, j]);
    dfs(board, node, i+1, j, path+tmp, res,
        mydict, listCoordinate);  // SEARCH BOTTOM
    dfs(board, node, i-1, j, path+tmp, res,
        mydict, listCoordinate);  // SEARCH TOP
    dfs(board, node, i, j+1, path+tmp, res,
        mydict, listCoordinate);  // SEARCH RIGHT
    dfs(board, node, i, j-1, path+tmp, res,
        mydict, listCoordinate);  // SEARCH LEFT
    listCoordinate.pop();
    board[i][j] = tmp;  // UNHASHING THE VISITED NODE
}
```

METHOD TO INITIALIZE

```
const drawPath = (renderer, point, width, height, map, color) => {
  // IF POINT[2] == 0 THEN DECOLOUR OUR CELL
  if (point[2] === 0) {
    renderer.ctx.clearRect(point[1] * width, point[0] * height, width, height);
    renderer.ctx.fillStyle = 'black';
    renderer.ctx.textAlign = 'center';
    renderer.ctx.font = '15px Arial';
    renderer.ctx.textBaseline = 'middle';
    renderer.ctx.fillText(
      map.data[point[0]][point[1]],
      (point[1] + 0.5) * width,
      (point[0] + 0.5) * height
    );
    renderer.ctx.strokeStyle = 'black';
    renderer.ctx.strokeRect(point[1] * width, point[0] * height, width, height);
  } else { // ELSE COLOUR OUR CELL
    renderer.ctx.strokeStyle = color;
    renderer.ctx.beginPath();
    renderer.ctx.arc(
      12.5 + point[1] * width,
      12.5 + point[0] * height,
      10,
      0,
      Math.PI * 2,
      true
    );
    renderer.ctx.stroke(); // DRAWS THE PATH
  }
};
```

RENDERING OUR GRID

```
const drawMap = (renderer, map) => {
  let ctx = renderer.ctx;
  let canvas = renderer.canvasEl;
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  // FILLING OUT DATA IN TABLE
  for (let y = 0; y < map.height; y++) {
    for (let x = 0; x < map.width; x++) {
      let cellChar = map.data[y][x];
      // DEFINING WHOLE TABLE PROPERTY
      ctx.textAlign = 'center';
      ctx.font = '15px Arial';
      ctx.textBaseline = 'middle';
      // FILLING CHARACTERS
      ctx.fillText(
        cellChar,
        (x + 0.5) * map.cellWidth,
        (y + 0.5) * map.cellHeight
      );
      ctx.strokeStyle = 'black';
      ctx.strokeRect(
        x * map.cellWidth,
        y * map.cellHeight,
        map.cellWidth,
        map.cellHeight
      );
    }
  }
};
```

RUNNING OUR DFS AND RETURNING TRAVERSED PATH

```
const makePath = (map, renderer) => {
  path = []; // PATH ACCOUNTS FOR ALL COORDINATES THAT WE HAVE TRAVERSED
  foundPath = []; // FOUND PATH IS FINAL PATH
  // OUR MAIN DFS FUNCTION
  function dfs(map, i, j, curr, vis) {
    // 'curr' ITERATOR OVER OUR WORD
    if (curr == word.length - 1 && map.data[i][j] == word[curr] && !vis[i][j]) {
      foundPath.push([i, j, 2]);
      path.push([i, j, 1]);
      return true;
    }
    path.push([i, j, 1]);
    foundPath.push([i, j, 2]);
    vis[i][j] = true;
    if (map.data[i][j] == word[curr]) {
      if (i > 0 && !vis[i - 1][j] && dfs(map, i - 1, j, curr + 1, vis))
        return true;
      if (
        i < map.height - 1 &&
        !vis[i + 1][j] &&
        dfs(map, i + 1, j, curr + 1, vis)
      )
        return true;
      if (j > 0 && !vis[i][j - 1] && dfs(map, i, j - 1, curr + 1, vis))
        return true;
      if (
        j < map.width - 1 &&
        !vis[i][j + 1] &&
        dfs(map, i, j + 1, curr + 1, vis)
      )
        return true;
    }
    foundPath.pop();
    vis[i][j] = false;
    path.push([i, j, 0]);
    return false;
  }
```

RUNNING VISUALIZATION

```
const runPath = (num, path, found, renderer, map) => {
  let pos = 0;
  function render() {
    if (pos < path.length) {
      drawPath(renderer, path[pos], map.cellWidth, map.cellHeight, map, 'red');
    } else {
      found.forEach((posi) => {
        drawPath(renderer, posi, map.cellWidth, map.cellHeight, map, 'blue');
        $('#reset').attr('disabled', false);
      });
      return;
    }
    pos += 1;
    setTimeout(render, num);
  }
  return render();
};
```

REFERENCES

- https://www.geeksforgeeks.org
- https://www.tutorialspoint.com
- https://www.w3schools.com