

LOVELY PROFESSIONAL UNIVERSITY, PHAGWARA



EIGHT WEEKS SUMMER TRAINING REPORT ON DATA STRUCTURE AND ALGORITHM (SELF PACED)

**Under the Guidance of Mr. Sandeep Jain, mentor in geeks for
geeks online learning platform.**

From (June - July,2023)

Submitted By:

Vishvajeet Singh
Chouhan

Reg No.:

DECLARATION

I hereby declare that I have completed my eight weeks summer training at GeeksforGeeks platform from June to July under the guidance of MR. Sandeep Jain. I have worked with full dedication during their 8 weeks of training and my learning outcomes fulfil the requirements of training for the award of degree of B.Tech. CSE, Lovely Professional University, Phagwara.

Reg. No.: 12200078

Vishvajeet Singh Chouhan

Vist github BLACKY - CODE50

ACKNOWLEDGEMENT

I would like to express my gratitude towards my University as well as Geeks for Geeks for providing me the golden opportunity to do this wonderful summer training regarding DSA, which also helped me in doing a lot of homework and learning. As a result, I came to know about so many new things. So, I am really thankful to them. Moreover, I would like to thank my friends who helped me a lot whenever I got stuck in some problem related to my course. I am really thankful to have such a good support of them as they always have my back whenever I need. Also, I would like to mention the support system and consideration of my parents who have always been there in my life to make me choose right thing and oppose the wrong. Without them I could never had learned and became a person who I am now. I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

CERTIFICATE



CERTIFICATE OF COURSE COMPLETION

THIS IS TO CERTIFY THAT

Vishwajeet Singh Chouhan

has successfully completed the course on DSA Self paced of
duration 8 weeks.

Sandeep Jain

Mr. Sandeep Jain

Founder & CEO, GeeksforGeeks

<https://media.geeksforgeeks.org/courses/certificates/f5cc575c7f213cfa308177abe66c72ec.pdf>

Vist github

TABLE OF CONTENTS

S.No.	Title	Page No.
1.	Introduction	6
2.	Technology Learnt	7-25
3.	Reason for choosing DSA	26
4.	8 Week plan	27-29
5.	Learning Outcome	30-31
6.	Project Description & code	32-39
7.	Conclusion	40
8.	Bibliography	31

INTRODUCTION

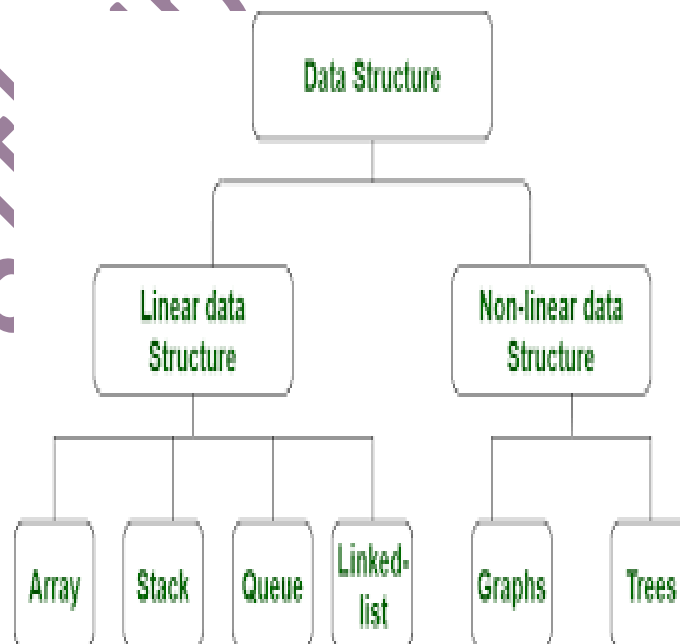
DSA self-paced course is a complete package that helped me to learn Data Structures and Algorithms from Basic to an Advance level. The course curriculum has been divided into 10 weeks, where I practiced questions and I have attempted the assessment tests accordingly. The course offers a wealth of programming challenges that helped me to learn all about DSA and making of an algorithm and how to solve problems and the logic behind the Algorithm. The course was Self-paced means I could join the course anytime and all the content will be available to me once I get enrolled. There were video lectures to learn from and multiple-choice questions to practice. I learned Algorithmic techniques for solving various problems with full flexibility of time as I was not time bounded. This course does not require any prior knowledge of Data Structure and Algorithms, but a basic knowledge of any programming language (C++) will be helpful. And as we all know Data Structure and Algorithm is a must skill in terms of Placement in any company because it helps us to increase our problem-solving skill.

TECHNOLOGY LEARNT

It had 24 units which was further divided into chapters and then topics.

Data Structure and Algorithms:

- Data structure is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data or a particular way of organizing data in a computer so that it can be used effectively. For example, we can store a list of items having the same data-type using the array data structure. This page contains detailed tutorials on different data structures (DS) with topic-wise problems.
- Algorithms algorithm is simply a set of steps used to complete a specific task. They're the building blocks for programming, and they allow things like computers, smartphones, and websites to function and make decisions. In addition to being used by technology, a lot of things we do on a daily basis are similar to algorithms.



Introduction to DSA

- Analysis of Algorithm: In this I learned about background analysis through a Program and its functions.
- Order of Growth: A mathematical explanation of the growth analysis through limits and functions. A direct way of calculating the order of growth.
- Asymptotic Notations: Best, Average and Worst-case explanation through a program.
- Big O Notation: Graphical and mathematical explanation. Calculation and Applications at Linear Search.
- Omega Notation: Graphical and mathematical explanation and Calculation.
- Theta Notation: Graphical and mathematical explanation and Calculation.
- Analysis of common loops: Single, multiple and nested loops.
- Analysis of Recursion: Various calculations through Recursion Tree method.
- Space Complexity: Basic Programs, Auxiliary Space ,Space Analysis of Recursion and Space Analysis of Fibonacci number.

Data Structure	Time Complexity							
	Average				Worst			
	Indexing	Searching	Insertion	Deletion	Indexing	Searching	Insertion	Deletion
Array	$O(1)$	$O(n)$			$O(1)$	$O(n)$		
Dynamic Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Singly Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Doubly Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Hash Table		$O(1)$	$O(1)$	$O(1)$		$O(n)$	$O(n)$	$O(n)$
Binary Search Tree	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Mathematics

- Finding the number of digits in a number.
- Arithmetic and Geometric Progressions.
- Quadratic Equations.
- Mean and Median.
- Prime Numbers.
- LCM and HCF
- Factorials
- Permutations and Combinations
- Modular Arithmetic

Bit magic

- Bitwise Operators in C++ : Operation of AND, OR, XOR operators .Operation of Left Shift, Right Shift and Bitwise Not.
- In C++, bitwise operators perform operations on integer data at the individual bit-level.
- Bitwise Operators in Java

Operation of AND, OR .Operation of Bitwise Not, Left Shift .Operation of Right Shift and unsigned Right Shift

- Problem (With Video Solutions): Check Kth bit is set or not.

Method 1: Using the left Shift.

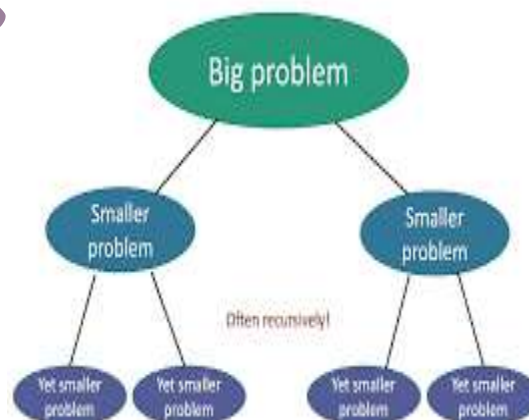
Method 2: Using the right shift.

Operators in C		
	Operator	Type
Unary operator	++, --, ~, !	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&, , ,	Logical operator
	&, , <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %	Assignment operator
Ternary operator	?:	Conditional operator

Recursion

• **Introduction to Recursion:** In recursion, a function or method has the ability to call itself to solve the problem. The process of recursion involves solving a problem by turning it into smaller varieties of itself. The process in which a function calls itself could happen directly as well as indirectly. Recursion work in c++ programming.

Idea: Break problem up into smaller (easier) subproblems



- **Applications of Recursion:**

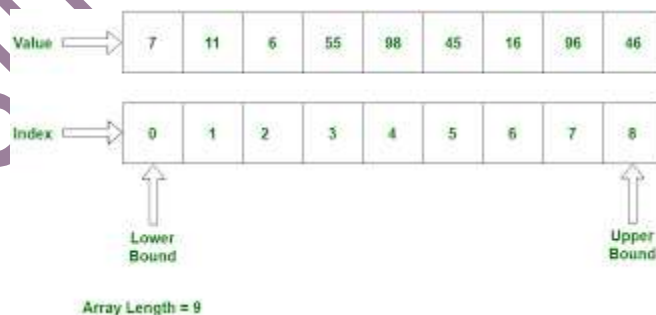
Application of Recursion

- So far, we have written recursive programs on integers: factorial, fibonacci, permutations, a^n
- Let us now consider a new application, grammars and parsing, that shows off the full power of recursion.
- Parsing has numerous applications: compilers, data retrieval, data mining,....

- **Writing base cases in Recursion:** Factorial and N-th Fibonacci number.

Arrays

- Introduction of Array: collection of fixed number of components (elements), wherein all of components have same data type.
- One-dimensional array: array in which components are arranged in list form.
- Multi-dimensional array: array in which components are arranged in tabular form (not covered).



Types of Arrays: -

1. Fixed-sized array
2. Dynamic-sized array

One-Dimensional Array				
Index	1	2	3	4
Value	12	25	88	245

Multi Dimensional Array			
Index	1	2	3
1	100	102	104
2	8	25	45
3	75	2	88

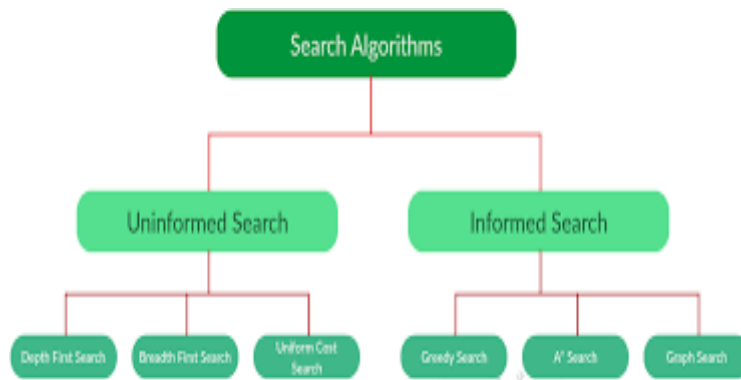
www.educba.com

Operations on Arrays: -

1. Searching
2. Insertions
3. Deletion
4. Arrays vs other DS
5. Reversing - Explanation with complexity

Searching

The process of finding the desired information from the set of items stored in the form of elements in the computer memory is referred to as 'searching in data structure'. These sets of items are in various forms, such as an array, tree, graph, or linked list.

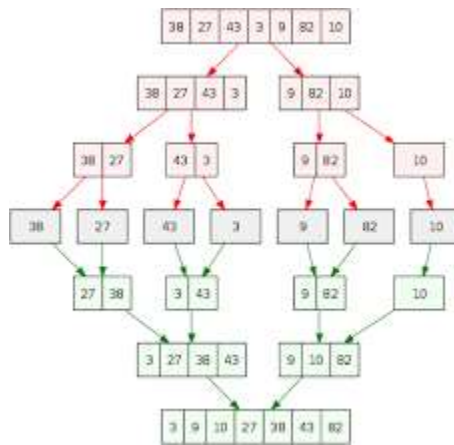


- Binary Search Iterative and Recursive
- Binary Search and various associated problems
- Two Pointer Approach Problems

Sorting

Sorting is the processing of arranging the data in ascending and descending order. There are several types of sorting in data structures namely – bubble sort, insertion sort, selection sort, bucket sort, heap sort, quick sort, radix sort etc.

- Implementation of C++ STL sort () function in Arrays and Vectors and Time Complexities
- Sorting in c++
- Arrays. Sort () in c++
- Collection. Sort () in c++
- Stability in Sorting Algorithms. Examples of Stable and Unstable Algos.
- Insertion Sort
- Merge Sort
- Quick Sort: Using Lomuto and Hoare, Time and Space analysis , Choice of Pivot and Worst case .
- Overview of Sorting Algorithms.

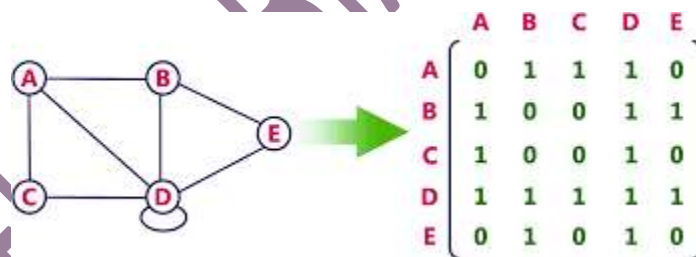


Matrix

Matrix is a way to store data in an organized form in the form of rows and columns. Matrices are usually used in computer graphics to project 3-dimensional space onto a 2-dimensional screen. Matrices in the form of arrays are used to store data in an organized form.

Uses: - To represent class hierarchy using Boolean square matrix

- For data encryption and decryption
- To represent traffic flow and plumbing in a network
- To implement graph theory of node representation

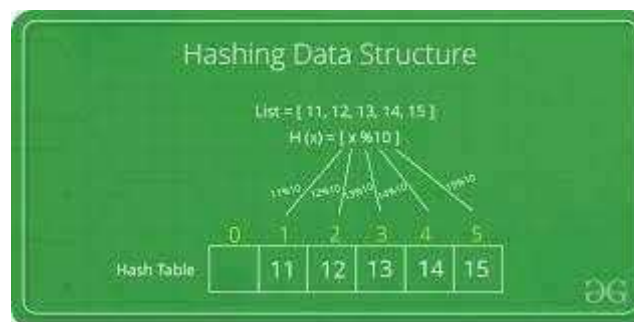


- Introduction to Matrix in C++
- Multidimensional Matrix
- Pass Matrix as Argument
- Printing matrix in a snake pattern
- Transposing a matrix
- Rotating a Matrix
- Check if the element is present in a row and column-wise sorted matrix.

- Boundary Traversal
- Spiral Traversal
- Matrix Multiplication
- Search in row-wise and column-wise Sorted Matrix

Hashing

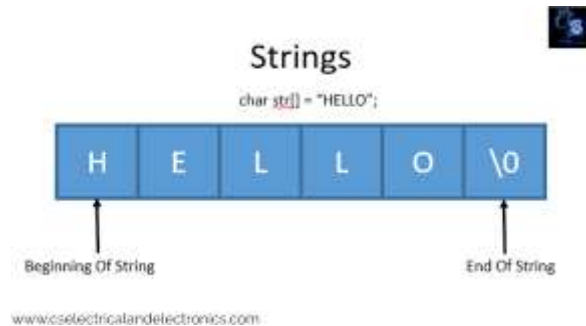
Hashing in the data structure is a technique of mapping a large chunk of data into small tables using a hashing function. It is also known as the message digest function. It is a technique that uniquely identifies a specific item from a collection of similar items.



- Introduction and Time complexity analysis
- Application of Hashing
- Discussion on Direct Address Table
- Working and examples on various Hash Functions
- Introduction and Various techniques on Collision Handling
- Chaining and its implementation
- Open Addressing and its Implementation
- Chaining V/S Open Addressing
- Double Hashing
- C++ Unordered Set Unordered Map
- Java HashSet HashMap

Strings

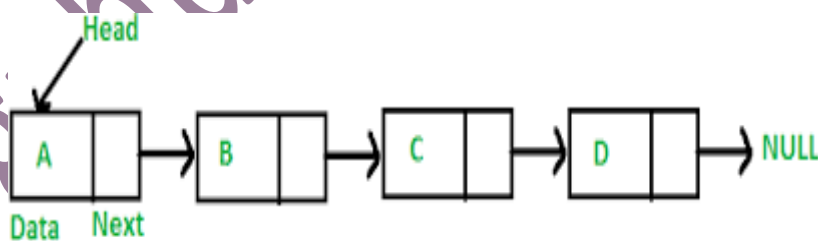
Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character `\0`.



- Discussion of String DS
- Strings in CPP
- Strings in Java
- Rabin Karp Algorithm
- KMP Algorithm

Linked list

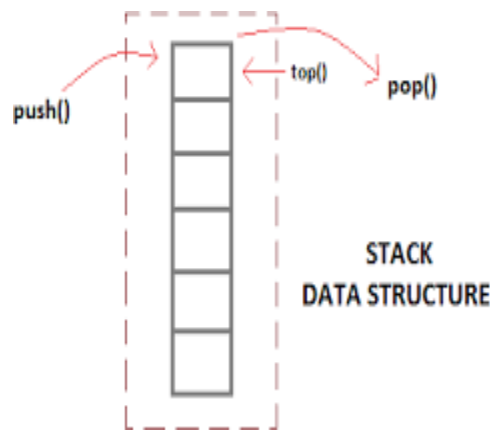
A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.



- Introduction Implementation in CPP. Comparison with Array DS.
- Doubly Linked List
- Circular Linked List
- Loop Problems, Detecting Loops , Detecting loops using Floyd cycle detection . Detecting and Removing Loops in Linked List

Stack

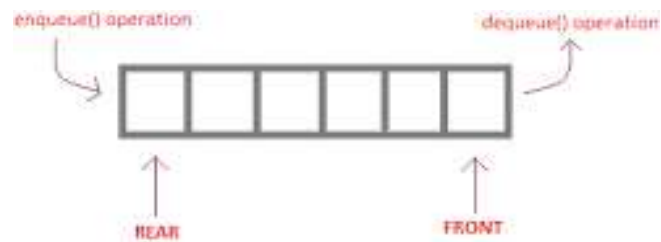
Stacks are a type of container adaptors with LIFO (Last in First Out) type of working, where a new element is added at one end (top) and an element is removed from that end only. Stack uses an encapsulated object of either vector or deque (by default) or list (sequential container class) as its underlying container, providing a specific set of member functions to access its elements.



- Understanding the Stack data structure
- Applications of Stack
- Implementation of Stack in Array and Linked List

Queue

Queue is a data structure designed to operate in FIFO (First in First out) context. In queue elements are inserted from rear end and get removed from front end. Queue class is container adapter. Container is an object that hold data of same type. Queue can be created from different sequence containers.



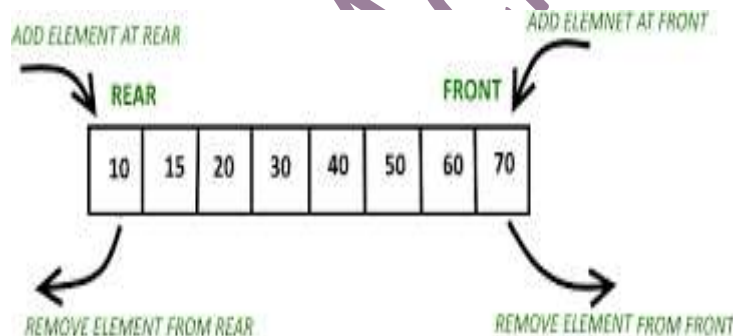
enqueue() is the operation for adding an element into Queue.
dequeue() is the operation for removing an element from Queue.

QUEUE DATA STRUCTURE

- Introduction and Application
- Implementation of the queue using array and LinkedList

Deque

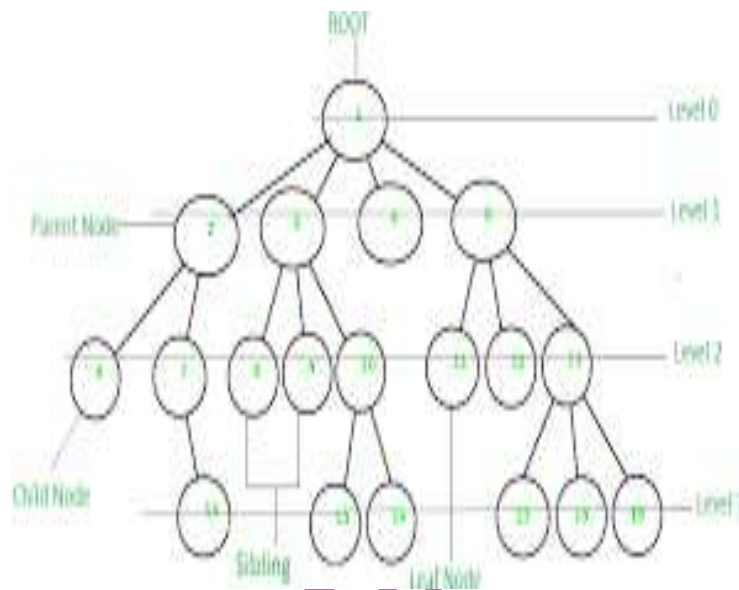
Deque (usually pronounced like "deck") is an irregular acronym of double-ended queue. Double-ended queues are sequence containers with dynamic sizes that can be expanded or contracted on both ends (either its front or its back).



- Introduction and Application
- Implementation o In C++ STL
- Problems (With Video Solutions) o Maximums of all subarrays of size k . Array Deque in java. Design a DS with min max operations.

Tree

A tree is a nonlinear data structure, compared to arrays, linked lists, stacks and queues which are linear data structures. A tree can be empty with no nodes or a tree is a structure consisting of one node called the root and zero or one or more subtrees. Binary Search Tree is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item. Heap is a tree data structure which is implemented using arrays and used to implement priority queues. B-Tree and B+ Tree They are used to implement indexing in databases.



Binary search tree

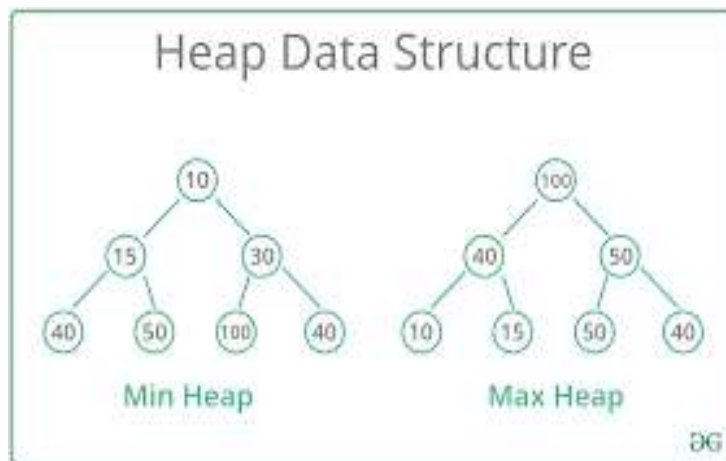


- Background, Introduction and Application
- Implementation of Search in BST
- Insertion in BST o In CPP
- Deletion in BST o In CPP
- Floor in BST and In CPP
- Self Balancing BST
- AVL Tree
- Red Black Tree
- Set in C++ STL
- Map in C++ STL.

Heap

A heap is a tree-based data structure in which all the nodes of the tree are in a specific order. For example, if is the parent node of, then the value of follows a

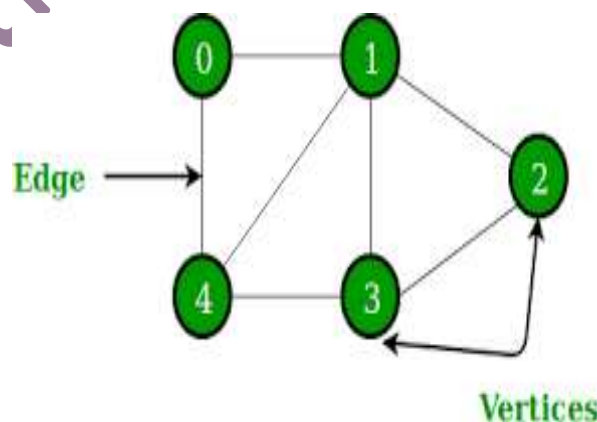
specific, order with respect to the value of and the same order will be followed across the tree.



- Introduction & Implementation
- Binary Heap o Insertion o Heapify and Extract o Decrease Key, Delete and Build Heap Sort
- Priority Queue in C++
- Priority Queue in Java

Graph

A Graph in the data structure can be termed as a data structure consisting of data that is stored among many groups of edges(paths) and vertices (nodes), which are interconnected. Graph data structure (N, E) is structured with a collection of Nodes and Edges. Both nodes and vertices need to be finite.



- Introduction to Graph
- Graph Representation, Adjacency Matrix, Adjacency List in CPP. Adjacency Matrix VS List

- Breadth-First Search and Applications
- Depth First Search o Applications
- Shortest Path in Directed Acyclic Graph
- Prim's Algorithm/Minimum Spanning Tree , Implementation in CPP and Implementation in Java
- Dijkstra's Shortest Path Algorithm

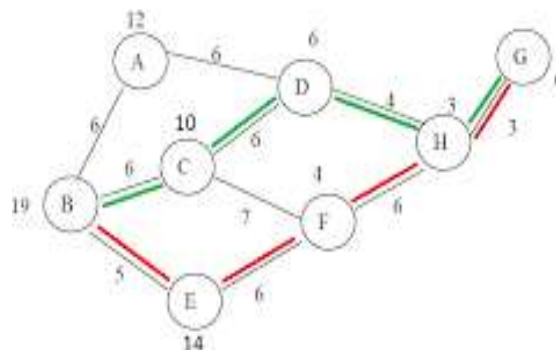
Implementation in CPP and Implementation in Java

- Bellman-Ford Shortest Path Algorithm
- Kisarazu's Algorithm
- Articulation Point
- Bridges in Graph
- Tarjan's Algorithm

Greedy

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy.

Figure 1:



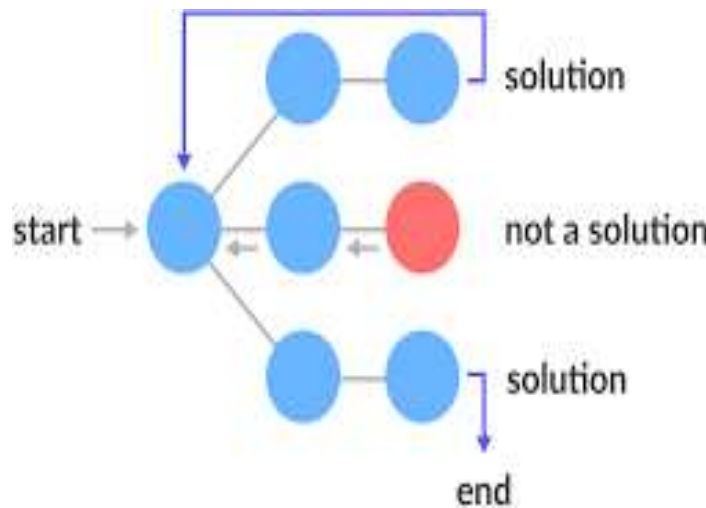
For example, consider the Fractional Knapsack Problem

- Activity Selection Problem

- Job Sequencing Problem

Backtracking

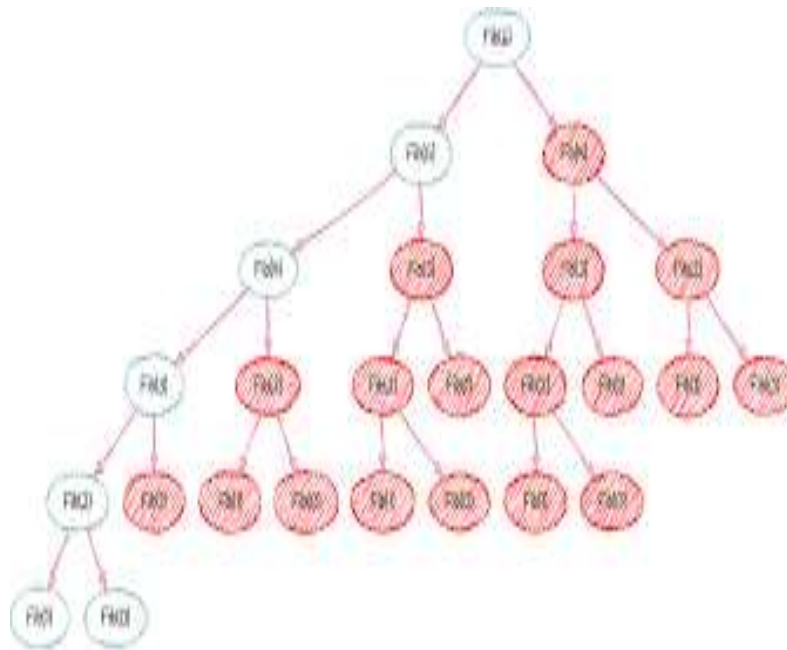
Backtracking is an algorithmic technique where the goal is to get all solutions to a problem using the brute force approach. It consists of building a set of all the solutions incrementally. Since a problem would have constraints, the solutions that fail to satisfy them will be removed.



- Concepts of Backtracking
- Rat in a Maze
- N Queen Problem

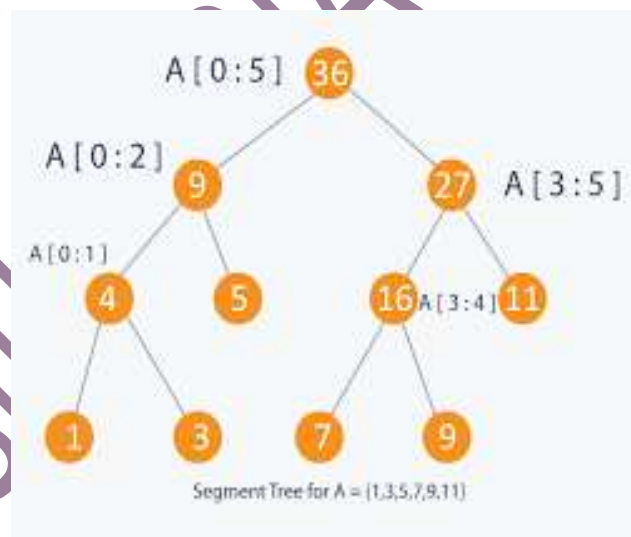
Dynamic programming

Dynamic programming is both a mathematical optimization method and a computer programming method. Likewise, in computer science, if a problem can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems, then it is said to have optimal substructure.



- Dynamic Programming
- Memorization
- Tabulation

Segment tree



A segment tree is a data structure used to store information about array segments and answer segment queries efficiently. There are two main operations performed on a segment tree: Both range (i, j) and update take $\log(n) \log(n) \log(n)$ time, where n is the number of elements in the segment tree.

A Segment Tree can be built using recursion (bottom-up approach). Start with the leaves and go up to the root and update the corresponding changes in the

nodes that are in the path from leaves to root. Leaves represent a single element.

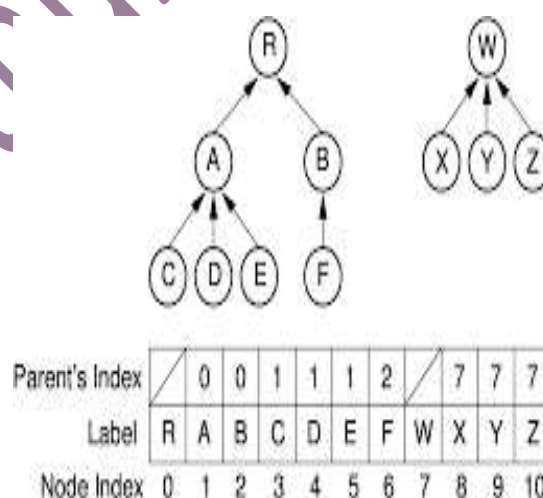
- Construction
- Range Query
- Update Query

Disjoint set

A disjoint-set data structure, also called a union–find data structure or merge–find set, is a data structure that stores a collection of disjoint (non-overlapping) sets. Equivalently, it stores partition of a set into disjoint subsets. It provides operations for adding new sets, merging sets (replacing them by their union), and finding a representative member of a set. The last operation allows to find out efficiently if any two elements are in the same or different sets.

- Find and Union Operations
- Union by Rank
- Path Compression

Kruskal's Algorithm



REASON FOR CHOOSING DATA STRUCTURE AND ALGORITHM (DSA)

All of the above was part of my training during my summer break I specially choose the DSA

by Geeks for Geeks for reasons stated below:

- I was interested in Problem Solving and Algorithms since my third semester.
- Data structure is a thing you need to know no matter in which language do you code.
- One need to learn how to make algorithm of a real life problem he/she is facing.
- It had video lectures of all the topics from which one can easily learn. I prefer learning from video rather than books and notes. I know books and notes and thesis have their own significance but still video lecture or face to face lectures make it easy to understand faster as we are involved Practically.
- It had 200+ algorithmic coding problems with video explained solutions.
- It had track based learning and weekly assessment to test my skills.
- It was a great opportunity for me to invest my time in learning instead of wasting it here and there during my summer break in this Covid-19 pandemic.
- It contained a lot of knowledge for such a reasonable price.
- The course was in two programming languages C++.
- This was a life time accessible course which I can use to learn even after my training whenever I want to revise.
- Along with all these reasons one of the reasons was the Geeks for Geeks platform which is offering the course because Geeks for Geeks is one of the best platforms for Computer Science Students.

8 WEEK COURSE PLAN

Week 1:

Day 1: Introduction- Analysis of Algorithms, Asymptotic Notation, Time Complexity, Space Complexity.

Day 2,3,4: Mathematics- Prime Numbers and Sieve of Eratosthenes, Multiples and Numbers (GCD, LCM, Factorials), Combinatorics, Modular Arithmetic, Practice Questions.

Day 5,6: Bit Manipulation- Bitwise operators, Set Bits, Count Bits, Power of 2 and Power sets, one odd occurring, 2 odd occurring, Practice Questions.

Day 7: Recursion- Application, Tail Recursion, Base Conditions, Classic Recursion Problems including Tower of Hanoi and Josephus Problem, Practice Questions.

Week 2:

Day 1: Continuation of Week 1 Day 7 topics.

Day 2,3,4: Arrays- Introduction and simple problems, Shifting and Rotation, Stock buy and sell, Trapping Rainwater, Kadane's Algo, Prefix Sum, Sliding window, Practice Questions.

Day 5,6: Searching- Linear, Binary, Ternary Search, Modified Binary Search to find floor, ceil, index of 1st and last occurrence, 2 pointers, median in sorted array, Find Square/Cube root, Search in rotated Sorted Array, Allocate Minimum pages, Practice Questions.

Day 7: Sorting- All types of sorting with best, avg, worst time complexity, Problems using merge sort and Quick Sort implementation, Comparator Function in Cpp/Java inbuilt sort function, Practice Questions.

Week 3:

Day 1,2: Continuation of Week 2 Day 7 topics.

Day 3,4: Matrix- Terminologies and Properties, Transpose, Rotation, Spiral, Multiplication of matrices, Search in Sorted matrix, Median in row sorted matrix, Practice Questions.

Day 5,6: Hashing- Concept of hashing, Separate chaining and open addressing,

implementation of hashing in Cpp/Java, Practice Questions.

Day 7: Strings- Introduction and basic questions, String Matching Algorithms, Lexicographic rank of string, Longest substring with distant characters, Practice Questions.

Week 4:

Day 1,2: Continuation of Week 3 Day 7 topics.

Day 3,4,5: Linked List- Types (Single LL, Double LL, XOR LL, Circular LL), Advantages, implementation and operations, Practice questions.

Day 6,7: Stacks- Introduction, implementation and operations using array and linked list, Practice Questions.

Week 5:

Day 1: Continuation of Week 4 Day 7 topics.

Day 2,3: Queue and Deque- - Introduction, implementation and operations using array and linked list, few Questions, implement stack using queues and vice versa, Practice Questions.

Day 4,5,6: Trees- Introduction and types of trees, Creation, Deletion, Insertion in a tree, Binary trees and Practice Questions.

Day 7: Binary Search Trees- Trees- Introduction and properties of BST, Creation, Searching, Deletion, Insertion in a BST, implementation of BST and Practice Questions.

Week 6:

Day 1,: Continuation of Week 5 Day 7 topics.

Day 2,3: Heaps- Introduction to Heap, Creation, Insertion, Deletion, Heapify, Priority Queues and related application in questions. Practice Questions.

Day 4,5,6,7: Graphs- Introduction to Graphs, properties, and representations, traverse the graphs using BFS and DFS, detect cycles, find shortest paths, find strongly connected components, etc. in a graph. Practice Questions on BFS, DFS and all given algorithms.

Week 7:

Day 1,2: Greedy - Practice Questions related to different Greedy approaches given.

Day 3,4: Backtracking- Introduction, Rat in a maze, N Queen Problem, Suduko Problem. Practice questions.

Day 5,6,7: Dynamic Programming- Introduction to Dynamic Programming, overlapping subproblems, and optimal substructures, Top-down and bottom-up approaches to solving a DP problem, All different types of DP problems and their variations. Practice Questions.

Week 8:

Day 1: Continuation of Week 7 Day 7 topics.

Day 2,3: Tries- Introduction to Tries, insert, search, delete, and updating in Tries, Practice Questions.

Day 4,5: Segment Trees (Basic Level)- Introduction to Segment Trees and when to use them, applying segment-trees in questions, Practice Questions.

Day 6,7: Disjoint Set- Introduction to DSU, Union by Rank and Path Compression, Kruskal's algorithm, Practice Questions.

LEARNING OUTCOMES FROM TRAINING/TECHNOLOGY

LEARNT

A lot of beginners and experienced programmers avoid learning Data Structures and Algorithms because it's complicated and they think that there is no use of all the above stuff in real life but there is a lot of implementations of DSA in daily life. For example, if we have to search our roll number in 2000 pages of Document how would we do that?

- If we try to search it randomly or in sequence it will take too much time.
- We can try another method in which we can directly go to page no. 1000 and we can see if our roll no. is there or not if not we can move ahead and by repeating this and eliminating we can search our roll no. in no time. And this is called Binary Search Algorithm.

Two reasons to Learn Data Structure and Algorithms -

- If you want to crack the interviews and get into the product-based companies.
- If you love to solve the real-world complex problems.

When we work in IT sector (Software or Programming part to be specific) we need to solve the problems and make programs write tons of code which will help us with the given problem and to write a program one need to make different algorithms. Many algorithms combine to make a program. Now, algorithm are written in some languages but they are not dependent ton them, one need to make a plan and algo first then write it into any language whether it is C++ or JAVA or C or any other programming language. Algorithm is based on data structure and its implementation and working. So, basically one need to have a good grip on DSA to work in programming sector. Software developers also have to make the right decisions when it comes to solving the problems of these companies. Knowledge of data structures like Hash Tables, Trees, Tries, Graphs, and various algorithms goes a long way in solving these problems efficiently and the interviewers are more interested in seeing how candidates use these tools to solve a problem. I learned about how to break a problem into pieces and then find the solution then how to make the desired algorithm which will help me to solve my respective problem.

What I Learned from the course precisely:

- I Learned Data Structures and Algorithms from basic to advanced level.
- Learned Topic-wise implementation of different Data Structures & Algorithms.
- Improved my problem-solving skills to become a stronger developer.
- Developed my analytical skills on Data Structures and use them efficiently.
- Solved problems asked in product-based companies' interviews.
- Solved problems in contests similar to coding round for SDE role.

Vist github BLACKY_CODE50

Project Description

Introduction: This project presents a Tic Tac Toe game, developed as part of the GFG self-paced Data Structures and Algorithms course using C++.

Overview: The Tic Tac Toe game is a classic 2-player game played on a 3x3 grid. The game involves players taking turns to place 'X' and 'O' markers to achieve a winning combination of three in a row, column, or diagonal.

Design and Features:

- The game features a text-based user interface.
- The game state is managed using a 2D array to represent the board.
- The game includes a minimax algorithm to enable optimal moves for the computer player.
- Valid input checks ensure player moves are within bounds and unoccupied cells.

Implementation:

- Functions are organized for board display, initialization, win checking, minimax algorithm, and player input handling.
- The minimax algorithm evaluates all possible moves, aiding computer player decisions.
- Debugging ensured accurate win, draw, and occupied cell handling.

User Interaction:

- Players input positions as numbers corresponding to the displayed grid.
- The game updates the board after each move and declares a win, loss, draw, or invalid input.

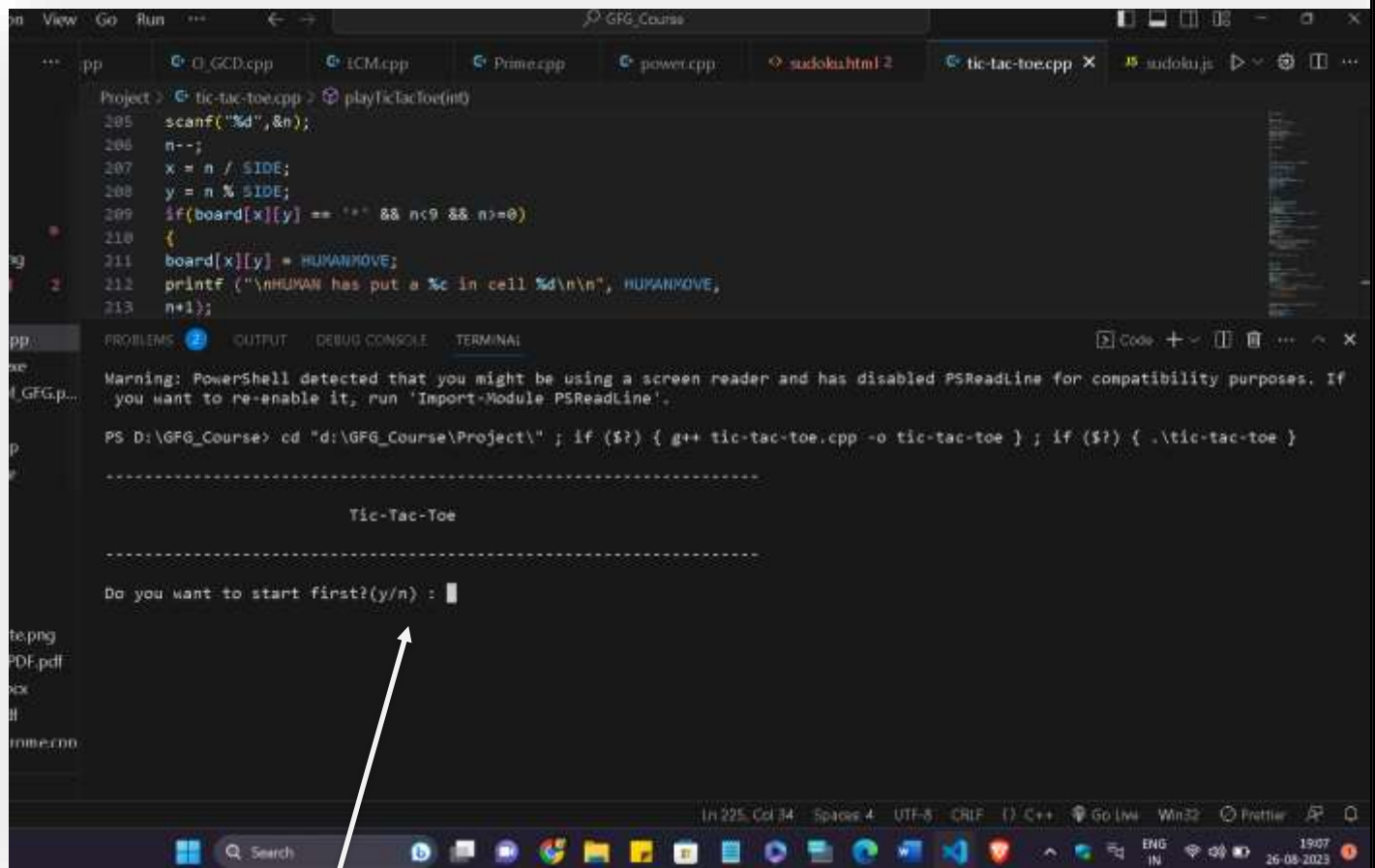
Learning Outcome:

- Implementing the minimax algorithm provided insight into AI decision-making.
- Handling user input and optimizing moves strengthened problem-solving skills.

Future Enhancements:

- Potential enhancements include graphical interface implementation and advanced AI strategies.

Screenshots of the Project



```
Project > G: tic-tac-toe.cpp > playTicTacToe(int)
205 scanf("%d",&n);
206 n--;
207 x = n / SIDE;
208 y = n % SIDE;
209 if(board[x][y] == '*' && n<9 && n>=0)
210 {
211 board[x][y] = HUMANMOVE;
212 printf ("\nHUMAN has put a %c in cell %d\n", HUMANMOVE,
213 n+1);
```

```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadline for compatibility purposes. If
you want to re-enable it, run 'Import-Module PSReadline'.

PS D:\GFG_Course> cd "d:\GFG_Course\Project\" ; if ($?) { g++ tic-tac-toe.cpp -o tic-tac-toe } ; if ($?) { .\tic-tac-toe }

-----
Tic-Tac-Toe
-----

Do you want to start first?(y/n) : 
```

This is the starting of the game and the normal terminal based interface that will take the user input

```
Project > tic-tac-toe.cpp > playTicTacToe(int)
205 scanf("%d",&n);
206 n++;
207 x = n / SIDE;
208 y = n % SIDE;
209 if(board[x][y] == '*' && n<9 && n>=0)
210 {
211 board[x][y] = HUMANMOVE;
212 printf("\nHUMAN has put a %c in cell %d\n\n", HUMANMOVE,
213 n+1);
```

1 | 2 | 3

4 | 5 | 6

7 | 8 | 9

You can insert in the following positions : You can insert in the following positions : 1 2 3 4 5 6 7 8 9

Enter the position = 2

HUMAN has put a X in cell 2

```
+ | x | +  
-----  
+ | * | +  
-----  
+ | * | +
```

COMPUTER has put a O in cell 1

```
o | x | +  
-----
```

- now after that user has to choose the another cell for input.
- Same game is going on till the diagonal,vertical,or horizontal choice will match of same user (either user or computer).

```
Project > tic-tac-toe.cpp > playTicTacToe(int)
205 scanf("%d",&n);
206 n--;
207 x = n / SIDE;
208 y = n % SIDE;
209 if(board[x][y] == '+' && n<9 && n>=0)
210 {
211 board[x][y] = HUMANMOVE;
212 printf("\nHUMAN has put a %c in cell %d\n\n", HUMANMOVE,
213 n+1);
```

Enter the position = 6

HUMAN has put a X in cell 6

```

  0 | X | +
  ---
  + | 0 | X
  ---
  + | + | X

```

COMPUTER has put a 0 in cell 3

```

  0 | X | 0
  ---
  + | 0 | X
  ---
  + | + | X

```

You can insert in the following positions : 4 7 8

Enter the position =

- User has to enter the index or specified cell number for choosing the cell.
- After that computer will choose the cell and make his decision.

```
Project > tic-tac-toe.cpp > playTicTacToe(int)
205 scanf("%d",&n);
206 n--;
207 x = n / SIDE;
208 y = n % SIDE;
209 if(board[x][y] == '*' && n<9 && n>=0)
210 {
211 board[x][y] = HUMANMOVE;
212 printf ("\nHUMAN has put a %c in cell %d\n\n", HUMANMOVE,
213 n+1);
```

Enter the position = 8

HUMAN has put a X in cell 8

0	X	0
*	0	X
*	X	X

COMPUTER has put a 0 in cell 7

0	X	0
*	0	X
0	X	X

COMPUTER has won

Do you want to quit(y/n) :

- As seen in the screenshot the computer has won the game.
- User choosen the wrong choice so it will lose the game.


```

if (board[0][0] == '-')
{
    board[0][0] = 'x';
    score = minMax(board, depth = 1, true);
    board[0][0] = '-';
    if (score < bestScore)
    {
        bestScore = score;
    }
}

return bestScore;
}

//min
return 0;

//max
return 0;

//minMax(board, board[0][0], 0, minMax)
{
    int x = 0, y = 0;
    int score = 0, bestScore = -100;
    for (int i = 0; i < 100; i++)
    {
        for (int j = 0; j < 100; j++)
        {
            if (board[i][j] == '-')
            {
                board[i][j] = 'o';
                score = minMax(board, bestScore, false);
                board[i][j] = '-';
                if (score > bestScore)
                {
                    bestScore = score;
                    x = i;
                    y = j;
                }
            }
        }
    }

    return x*y;
}

// is function to play the two
// min player's turn's move
{
    char board[100][100];
    int minScore = 0, x = 0, y = 0;
    minMax(board);
    minMax(board);
    while (gameOver(board) == false && minScore != 100*100)
    {
        int x;
        if (minScore == 100*100)
        {
            x = bestScore(board, minScore);
            x = x / 100;
            board[x][0] = 'o';
            print("It's now o's turn to play. It's o's turn to play.");
        }
    }
}

```

© 2006 The Authors
Journal compilation © 2006 Blackwell Publishing Ltd

CONCLUSION

A data structure is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data, i.e., it is algebraic expression about data. Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer—a bit string, representing a memory address, that can be itself stored in memory and manipulated by the program. Thus, the array and record data structures are based on computing the addresses of data items with arithmetic operations, while the linked data structures are based on storing addresses of data items within the structure itself. The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analysed separately from those operations. This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost). The course was Self placed means I could join the course anytime and all the content will be available to me once I get enrolled. There were video lectures to learn from and multiple-choice questions to practice. I learned Algorithmic techniques for solving various problems with full flexibility of time as I was not time bounded. This course does not require any prior knowledge of Data Structure and Algorithms, but a basic knowledge of any programming language (C++) will be helpful.

BIBLIOGRAPHY

- Geeks for Geeks website
- Geeks for Geeks Course
 - YouTube

Vist github BLACKY_CODE50

THANK YOU

Vist github BLACKY - CODE50