



Sign in to GeeksforGeeks with Google



Tengku Sembok

tmtsembok@gmail.com

Continue as Tengku



monisha_jegadeesan

Read

Discuss

Courses

Practice

Video

Grammar denotes the syntactical rules for conversation in natural language. But in the theory of formal language, grammar is defined as a set of rules that can generate strings. The set of all strings that can be generated from a grammar is called the language of the grammar.

Context Free Grammar:

We are given a **Context Free Grammar** $G = (V, X, R, S)$ and a string w , where:

- V is a finite set of variables or non-terminal symbols,
- X is a finite set of terminal symbols,
- R is a finite set of rules,
- S is the start symbol, a distinct element of V , and
- V and X are assumed to be disjoint sets.

The **Membership problem** is defined as: Grammar G generates a language $L(G)$. Is the given string a member of $L(G)$?

Chomsky Normal Form:

A Context Free Grammar G is in **Chomsky Normal Form (CNF)** if each rule of G is of the form:

- $A \rightarrow BC$, [with at most two non-terminal symbols on the RHS]
- $A \rightarrow a$, or [one terminal symbol on the RHS]
- $S \rightarrow \text{nullstring}$, [null string]

Cocke-Younger-Kasami Algorithm

It is used to solve the membership problem using a [dynamic programming](#) approach. The algorithm is based on the principle that the solution to problem $[i, j]$ can be constructed from solution to subproblem $[i, k]$ and solution to subproblem $[k, j]$. The algorithm requires the Grammar G to be in Chomsky Normal Form (CNF). Note that any Context-Free Grammar can be systematically [converted to CNF](#). This restriction is employed so that each problem can only be divided into two subproblems and not more – to bound the time complexity.

How does the CYK Algorithm work?

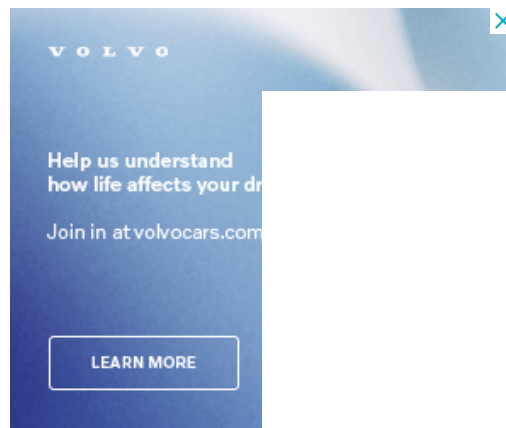
For a string of length N , construct a table T of size $N \times N$. Each cell in the table $T[i, j]$ is the set of all constituents that can produce the substring spanning from position i to j . The process involves filling the table with the solutions to the subproblems encountered in the bottom-up parsing process. Therefore, cells will be filled from left to right and bottom to top.

	1	2	3	4	5
1	[1, 1]	[1, 2]	[1, 3]	[1, 4]	[1, 5]
2		[2, 2]	[2, 3]	[2, 4]	[2, 5]
3			[3, 3]	[3, 4]	[3, 5]
4				[4, 4]	[4, 5]

Engineering Mathematics Discrete Mathematics Digital Logic and Design Computer Organization and Architecture C Progra

--	--	--	--	--	--

In $T[i, j]$, the row number i denotes the start index and the column number j denotes the end index.



$A \in T[i, j]$ if and only if $B \in T[i, k]$, $C \in T[k, j]$ and $A \rightarrow BC$ is a rule of G

The algorithm considers every possible subsequence of letters and adds K to $T[i, j]$ if the sequence of letters starting from i to j can be generated from the non-terminal K . For subsequences of length 2 and greater, it considers every possible partition of the subsequence into two parts, and checks if there is a rule of the form $A \rightarrow BC$ in the grammar where B and C can generate the two parts respectively, based on already existing entries in T . The sentence can be produced by the grammar only if the entire string is matched by the start symbol, i.e, if S is a member of $T[1, n]$.

Consider a sample grammar in Chomsky Normal Form:

```

NP  --> Det | Nom
Nom --> AP | Nom
AP  --> Adv | A
Det --> a | an
Adv --> very | extremely
AP  --> heavy | orange | tall
A   --> heavy | orange | tall | muscular
Nom --> book | orange | man

```

Now consider the phrase, “**a very heavy orange book**”:

a(1) very(2) heavy (3) orange(4) book(5)

Let us start filling up the table from left to right and bottom to top, according to the rules described above:

	1 a	2 very	3 heavy	4 orange	5 book
1 a	Det	–	–	NP	NP
2 very		Adv	AP	Nom	Nom
3 heavy			A, AP	Nom	Nom
4 orange				Nom, A, AP	Nom
5 book					Nom

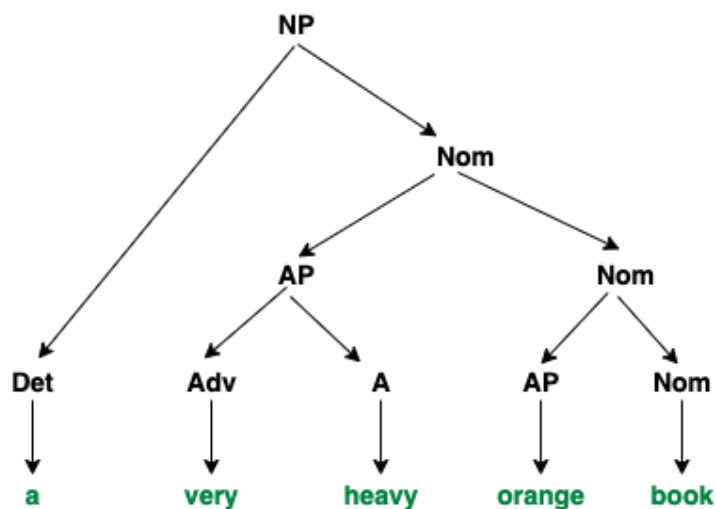
The table is filled in the following manner:

1. $T[1, 1] = \{\text{Det}\}$ as $\text{Det} \rightarrow a$ is one of the rules of the grammar.
2. $T[2, 2] = \{\text{Adv}\}$ as $\text{Adv} \rightarrow \text{very}$ is one of the rules of the grammar.
3. $T[1, 2] = \{\}$ as no matching rule is observed.
4. $T[3, 3] = \{A, AP\}$ as $A \rightarrow \text{very}$ and $AP \rightarrow \text{very}$ are rules of the grammar.
5. $T[2, 3] = \{AP\}$ as $AP \rightarrow \text{Adv } (T[2, 2]) \ A \ (T[3, 3])$ is a rule of the grammar.
6. $T[1, 3] = \{\}$ as no matching rule is observed.
7. $T[4, 4] = \{\text{Nom}, A, AP\}$ as $\text{Nom} \rightarrow \text{orange}$ and $A \rightarrow \text{orange}$ and $AP \rightarrow \text{orange}$ are rules

9. $T[2, 4] = \{\text{Nom}\}$ as $\text{Nom} \rightarrow \text{AP } (T[2, 3]) \text{ Nom } (T[4, 4])$ is a rule of the grammar.
10. $T[1, 4] = \{\text{NP}\}$ as $\text{NP} \rightarrow \text{Det } (T[1, 1]) \text{ Nom } (T[2, 4])$ is a rule of the grammar.
11. $T[5, 5] = \{\text{Nom}\}$ as $\text{Nom} \rightarrow \text{book}$ is a rule of the grammar.
12. $T[4, 5] = \{\text{Nom}\}$ as $\text{Nom} \rightarrow \text{AP } (T[4, 4]) \text{ Nom } (T[5, 5])$ is a rule of the grammar.
13. $T[3, 5] = \{\text{Nom}\}$ as $\text{Nom} \rightarrow \text{AP } (T[3, 3]) \text{ Nom } (T[4, 5])$ is a rule of the grammar.
14. $T[2, 5] = \{\text{Nom}\}$ as $\text{Nom} \rightarrow \text{AP } (T[2, 3]) \text{ Nom } (T[4, 5])$ is a rule of the grammar.
15. $T[1, 5] = \{\text{NP}\}$ as $\text{NP} \rightarrow \text{Det } (T[1, 1]) \text{ Nom } (T[2, 5])$ is a rule of the grammar.

We see that $T[1][5]$ has **NP**, the start symbol, which means that this phrase is a member of the language of the grammar G .

The parse tree of this phrase would look like this:



Let us look at another example phrase, “**a very tall extremely muscular man**”:

a(1) very(2) tall(3) extremely(4) muscular(5) man(6)

We will now use the CYK algorithm to find if this string is a member of the grammar G :

	1 a	2 very	3 tall	4 extremely	5 muscular	6 man
1 a	Det	–	–	–	–	NP
2 very		Adv	AP	–	–	Nom
3 tall			AP, A	–	–	Nom
4 extremely				Adv	AP	Nom
5 muscular					A	–
6 man						Nom

We see that $T[1][6]$ has **NP**, the start symbol, which means that this phrase is a member of the language of the grammar G.

Below is the implementation of the above algorithm:

C++

```
// C++ implementation for the
// CYK Algorithm

#include<bits/stdc++.h>
using namespace std;

// Non-terminals symbols
vector<string> terminals, non_terminals;
```

```

// function to perform the CYK Algorithm
void cykParse(vector<string> w)
{
    int n = (int)w.size();

    // Initialize the table
    map<int, map<int, set<string>>> T;

    // Filling in the table
    for(int j=0; j<n; j++)
    {

        // Iterate over the rules
        for(auto x:R)
        {
            string lhs = x.first;
            vector<vector<string>> rule = x.second;

            for(auto rhs:rule)
            {

                // If a terminal is found
                if(rhs.size() == 1 && rhs[0] == w[j])
                    T[j][j].insert(lhs);
            }
        }

        for(int i=j; i>=0; i--)
        {

            // Iterate over the range from i to j
            for(int k = i; k<=j; k++)
            {

                // Iterate over the rules
                for(auto x:R)
                {
                    string lhs = x.first;
                    vector<vector<string>> rule = x.second;

                    for(auto rhs:rule)
                    {
                        // If a terminal is found
                        if(rhs.size()==2 && T[i][k].find(rhs[0])!=T[i][k].end() && T[k+1][j].find(rhs[1])!=T[k+1][j].end())
                            T[i][j].insert(lhs);
                    }
                }
            }
        }
    }
}

```

```

// of given grammar
if(T[0][n-1].size() != 0)
    cout << "True\n";
else
    cout << "False\n";
}

// Driver Code
int main()
{
    // terminal symbols
    terminals = {"book", "orange", "man",
                "tall", "heavy",
                "very", "muscular"};

    // non terminal symbols
    non_terminals = {"NP", "Nom", "Det", "AP",
                    "Adv", "A"};

    // Rules
    R["NP"] = {{ "Det", "Nom" }};
    R["Nom"] = {{ "AP", "Nom", {"book"},
                  {"orange"}, {"man"} }};
    R["AP"] = {{ "Adv", "A", {"heavy"},
                  {"orange"}, {"tall"} }};
    R["Det"] = {{ "a" }};
    R["Adv"] = {{ "very", {"extremely"} }};
    R["A"] = {{ "heavy", {"orange"}, {"tall"},
                {"muscular"} }};

    // Given String
    vector<string> w = {"a", "very", "heavy", "orange", "book"};

    // Function Call
    cykParse(w);

    return 0;
}

```

Java

```

import java.util.*;

class GFG
{
    // Non-terminals symbols
    static List<String> terminals = new ArrayList<>();
    static List<String> non_terminals = new ArrayList<>();
}

```



```

    = new HashMap<>();

// function to perform the CYK Algorithm
static void cykParse(List<String> w)
{
    int n = w.size();

    // Initialize the table
    Map<Integer, Map<Integer, Set<String> > > T
    = new HashMap<>();

    // Filling in the table
    for (int j = 0; j < n; j++) {

        // Iterate over the rules
        for (Map.Entry<String, List<List<String> > > x :
            R.entrySet()) {
            String lhs = x.getKey();
            List<List<String> > rule = x.getValue();

            for (List<String> rhs : rule) {

                // If a terminal is found
                if (rhs.size() == 1
                    && rhs.get(0).equals(w.get(j))) {
                    if (T.get(j) == null)
                        T.put(j, new HashMap<>());
                    T.get(j)
                        .computeIfAbsent(
                            j, k -> new HashSet<>())
                        .add(lhs);
                }
            }
        }
    }
    for (int i = j; i >= 0; i--) {

        // Iterate over the range from i to j
        for (int k = i; k <= j; k++) {

            // Iterate over the rules
            for (Map.Entry<String,
                List<List<String> > > x :
                R.entrySet()) {
                String lhs = x.getKey();
                List<List<String> > rule
                = x.getValue();

                for (List<String> rhs : rule) {
                    // If a terminal is found
                    if (rhs.size() == 2
                        && T.get(i) != null
                        && T.get(i).get(k) != null

```

```

        && T.get(k + 1) != null
        && T.get(k + 1).get(j)
        != null
        && T.get(k + 1)
        .get(j)
        .contains(
            rhs.get(1))) {
    if (T.get(i) == null)
        T.put(i,
            new HashMap<>());
    if (T.get(i).get(j) == null)
        T.get(i).put(
            j, new HashSet<>());
    T.get(i).get(j).add(lhs);
}
}
}
}
}

// If word can be formed by rules
// of given grammar
if (T.get(0) != null && T.get(0).get(n - 1) != null
    && T.get(0).get(n - 1).size() != 0)
    System.out.println("True");
else
    System.out.println("False");
}

// Driver Code
public static void main(String[] args)
{
    // terminal symbols
    terminals
        = Arrays.asList("book", "orange", "man", "tall",
            "heavy", "very", "muscular");

    // non terminal symbols
    non_terminals = Arrays.asList("NP", "Nom", "Det",
        "AP", "Adv", "A");

    // Rules
    R.put("NP",
        Arrays.asList(Arrays.asList("Det", "Nom")));
    R.put("Nom",
        Arrays.asList(Arrays.asList("AP", "Nom"),
            Arrays.asList("book"),
            Arrays.asList("orange"),
            Arrays.asList("man")));
    R.put("AP", Arrays.asList(Arrays.asList("Adv", "A"),
        Arrays.asList("heavv")));

```

```

R.put("Det", Arrays.asList(Arrays.asList("a")));
R.put("Adv",
      Arrays.asList(Arrays.asList("very"),
                    Arrays.asList("extremely")));
R.put("A",
      Arrays.asList(Arrays.asList("heavy"),
                    Arrays.asList("orange"),
                    Arrays.asList("tall"),
                    Arrays.asList("muscular")));

// Given String
List<String> w = Arrays.asList("a", "very", "heavy",
                              "orange", "book");

// Function Call
cykParse(w);
}
}

// This code is contributed by lokeshpotta20

```

Python3

```

# Python implementation for the
# CYK Algorithm

# Non-terminal symbols
non_terminals = ["NP", "Nom", "Det", "AP",
                 "Adv", "A"]
terminals = ["book", "orange", "man",
             "tall", "heavy",
             "very", "muscular"]

# Rules of the grammar
R = {
    "NP": [["Det", "Nom"]],
    "Nom": [["AP", "Nom"], ["book",
                          "orange"], ["man"]],
    "AP": [["Adv", "A"], ["heavy",
                          "orange"], ["tall"]],
    "Det": [["a"]],
    "Adv": [["very"], ["extremely"]],
    "A": [["heavy"], ["orange"], ["tall"],
          ["muscular"]]
}

# Function to perform the CYK Algorithm
def cykParse(w):
    n = len(w)

```

```

# Filling in the table
for j in range(0, n):

    # Iterate over the rules
    for lhs, rule in R.items():
        for rhs in rule:

            # If a terminal is found
            if len(rhs) == 1 and \
               rhs[0] == w[j]:
                T[j][j].add(lhs)

    for i in range(j, -1, -1):

        # Iterate over the range i to j + 1
        for k in range(i, j + 1):

            # Iterate over the rules
            for lhs, rule in R.items():
                for rhs in rule:

                    # If a terminal is found
                    if len(rhs) == 2 and \
                       rhs[0] in T[i][k] and \
                       rhs[1] in T[k + 1][j]:
                        T[i][j].add(lhs)

# If word can be formed by rules
# of given grammar
if len(T[0][n-1]) != 0:
    print("True")
else:
    print("False")

# Driver Code

# Given string
w = "a very heavy orange book".split()

# Function Call
cykParse(w)

```

C#

```

// C# program to implement above approach
using System;
using System.Collections;
using System.Collections.Generic;

class GFG

```

```

// static List<string> terminals = new List<string>();
// static List<string> non_terminals = new List<string>();

// Rules of the grammar
static Dictionary<string, List<List<string>>> R = new Dictionary<string, List<List<string>>>

// function to perform the CYK Algorithm
static void cykParse(List<string> w)
{
    int n = w.Count;

    // Initialize the table
    SortedDictionary<int, SortedDictionary<int, SortedSet<string>>> T = new SortedDicti

    // Filling in the table
    for (int j = 0 ; j < n ; j++)
    {

        // Iterate over the rules
        foreach (KeyValuePair<string, List<List<string>>> x in R)
        {
            string lhs = x.Key;
            List<List<string>> rule = x.Value;

            foreach (List<string> rhs in rule)
            {

                // If a terminal is found
                if(rhs.Count == 1 && rhs[0] == w[j]){
                    if(!T.ContainsKey(j)){
                        T.Add(j, new SortedDictionary<int, SortedSet<string>>());
                    }
                    if(!T[j].ContainsKey(j)){
                        T[j].Add(j, new SortedSet<string>());
                    }
                    T[j][j].Add(lhs);
                }
            }
        }

        for(int i = j ; i >= 0 ; i--)
        {

            // Iterate over the range from i to j
            for(int k = i ; k <= j ; k++)
            {

                // Iterate over the rules
                foreach (KeyValuePair<string, List<List<string>>> x in R)
                {
                    string lhs = x.Key;
                    List<List<string>> rule = x.Value:

```

```

    {
        // If a terminal is found
        if(rhs.Count == 2 &&
            T.ContainsKey(i) &&
            T[i].ContainsKey(k) &&
            T[i][k].Contains(rhs[0]) &&
            T.ContainsKey(k + 1) &&
            T[k + 1].ContainsKey(j) &&
            T[k + 1][j].Contains(rhs[1]))
        {
            if(!T.ContainsKey(i)){
                T.Add(i, new SortedDictionary<int, SortedSet<string>>());
            }
            if(!T[i].ContainsKey(j)){
                T[i].Add(j, new SortedSet<string>());
            }
            T[i][j].Add(lhs);
        }
    }
}

// If word can be formed by rules
// of given grammar
if(T.ContainsKey(0) && T[0].ContainsKey(n - 1) && T[0][n - 1].Count != 0){
    Console.WriteLine("True\n");
}else{
    Console.WriteLine("False\n");
}
}

// Driver code
public static void Main(string[] args){

    // terminal symbols
    // terminals = new List<string>{
    //     "book",
    //     "orange", "man",
    //     "tall", "heavy",
    //     "very", "muscular"
    // };

    // non terminal symbols
    // non_terminals = new List<string>{
    //     "NP", "Nom", "Det",
    //     "AP", "Adv", "A"
    // };

```

```

        new List<string>{"Det", "Nom"}
    });

    R["Nom"] = new List<List<string>>{
        new List<string>{"AP", "Nom"},
        new List<string>{"book"},
        new List<string>{"orange"},
        new List<string>{"man"}
    };

    R["AP"] = new List<List<string>>{
        new List<string>{"Adv", "A"},
        new List<string>{"heavy"},
        new List<string>{"orange"},
        new List<string>{"tall"}
    };

    R["Det"] = new List<List<string>>{
        new List<string>{"a"}
    };

    R["Adv"] = new List<List<string>>{
        new List<string>{"very"},
        new List<string>{"extremely"}
    };

    R["A"] = new List<List<string>>{
        new List<string>{"heavy"},
        new List<string>{"orange"},
        new List<string>{"tall"},
        new List<string>{"muscular"}
    };

    // Given String
    List<string> w = new List<string>{"a", "very", "heavy", "orange", "book"};

    // Function Call
    cykParse(w);
}
}

// This code is contributed by subhamgoyal2014.

```

Javascript

```

// CYK Algorithm

// Non-terminal symbols

```

```

// Rules of the grammar
const R = {
  "NP": [["Det", "Nom"]],
  "Nom": [["AP", "Nom"], ["book"],
    ["orange"], ["man"]],
  "AP": [["Adv", "A"], ["heavy"],
    ["orange"], ["tall"]],
  "Det": [["a"]],
  "Adv": [["very"], ["extremely"]],
  "A": [["heavy"], ["orange"], ["tall"],
    ["muscular"]]
};

// function to perform the CYK Algorithm
function cykParse(w) {
  let n = w.length;

  // Initialize the table
  let T = [];
  for (let i = 0; i < n; i++) {
    T[i] = [];
    for (let j = 0; j < n; j++) {
      T[i][j] = new Set();
    }
  }

  // Filling in the table
  for (let j = 0; j < n; j++) {
    // Iterate over the rules
    for (let lhs in R) {
      let rule = R[lhs];
      for (let rhs of rule) {
        // If a terminal is found
        if (rhs.length == 1 && rhs[0] == w[j]) {
          T[j][j].add(lhs);
        }
      }
    }
  }
  for (let i = j; i >= 0; i--) {
    // Iterate over the range from i to j
    for (let k = i; k <= j; k++) {
      // Iterate over the rules
      for (let lhs in R) {
        let rule = R[lhs];
        for (let rhs of rule) {
          // If a terminal is found
          if (rhs.length == 2 && T[i][k].has(rhs[0]) && T[k + 1][j].has(rhs[1])) {
            T[i][j].add(lhs);
          }
        }
      }
    }
  }
}

```



```
}

// If word can be formed by rules
// of given grammar
if (T[0][n - 1].size !== 0) {
  console.log("True");
} else {
  console.log("False");
}
}

// Given String
const w = ["a", "very", "heavy", "orange", "book"];

// Function Call
cykParse(w);
```

Output:

True

Time Complexity: $O(N^3)$

Auxiliary Space: $O(N^2)$

Last Updated : 13 Feb, 2023

Similar Reads

1. Painting Fence Algorithm
2. Iterative algorithm for a forward data-flow problem
3. Welsh Powell Graph colouring Algorithm
4. Algorithm for non recursive Predictive Parsing
5. Count of occurrences of each prefix in a string using modified KMP algorithm
6. Finding shortest path between any two nodes using Floyd Warshall Algorithm

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

8. Iterative algorithm for a backward data flow problem

9. Bellman–Ford Algorithm | DP-23

10. Bellman Ford Algorithm (Simple Implementation)

Related Tutorials

1. Learn Data Structures with Javascript | DSA Tutorial

2. Introduction to Max-Heap – Data Structure and Algorithm Tutorials

3. Introduction to Set – Data Structure and Algorithm Tutorials

4. Introduction to Map – Data Structure and Algorithm Tutorials

5. What is Dijkstra’s Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm

Previous

Minimum number of edges required to be removed from an Undirected Graph to make it acyclic

Next

Applications of String Matching Algorithms

Article Contributed By :



monisha_jegadeesan

monisha_jegadeesan

Vote for difficulty

Current difficulty : Hard

Easy

Normal

Medium

Hard

Expert

Improved By : [lokeshpotta20](#), [pradeepkumarppk2003](#), [subhamgoyal2014](#), [ashutoshsinghgeeksforgeeks](#)

Article Tags : [Algorithms](#), [Compiler Design](#), [DSA](#), [Dynamic Programming](#)

[Improve Article](#)[Report Issue](#)

A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

[About Us](#)[Careers](#)[In Media](#)[Contact Us](#)[Terms and Conditions](#)[Privacy Policy](#)[Copyright Policy](#)[Third-Party Copyright Notices](#)[Advertise with us](#)

Data Structures

[Array](#)[String](#)[Linked List](#)[Stack](#)[Queue](#)[Tree](#)[Graph](#)

Web Development

[HTML](#)[CSS](#)[JavaScript](#)[Bootstrap](#)

Languages

[Python](#)[Java](#)[C++](#)[GoLang](#)[SQL](#)[R Language](#)[Android Tutorial](#)

Algorithms

[Sorting](#)[Searching](#)[Greedy](#)[Dynamic Programming](#)[Pattern Searching](#)[Recursion](#)[Backtracking](#)

Write & Earn

[Write an Article](#)[Improve an Article](#)[Pick Topics to Write](#)[Write Interview Experience](#)

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

[AngularJS](#)[Video Internship](#)[NodeJS](#)

Computer Science

[GATE CS Notes](#)[Operating Systems](#)[Computer Network](#)[Database Management System](#)[Software Engineering](#)[Digital Logic Design](#)[Engineering Maths](#)

Interview Corner

[Company Preparation](#)[Preparation for SDE](#)[Company Interview Corner](#)[Experienced Interview](#)[Internship Interview](#)[Competitive Programming](#)[Aptitude](#)

GfG School

[CBSE Notes for Class 8](#)[CBSE Notes for Class 9](#)[CBSE Notes for Class 10](#)[CBSE Notes for Class 11](#)[CBSE Notes for Class 12](#)[English Grammar](#)

Data Science & ML

[Data Science With Python](#)[Data Science For Beginner](#)[Machine Learning Tutorial](#)[Maths For Machine Learning](#)[Pandas Tutorial](#)[NumPy Tutorial](#)[NLP Tutorial](#)

Python

[Python Tutorial](#)[Python Programming Examples](#)[Django Tutorial](#)[Python Projects](#)[Python Tkinter](#)[OpenCV Python Tutorial](#)

UPSC/SSC/ BANKING

[SSC CGL Syllabus](#)[SBI PO Syllabus](#)[IBPS PO Syllabus](#)[UPSC Ethics Notes](#)[UPSC Economics Notes](#)[UPSC History Notes](#)

@geeksforgeeks , Some rights reserved