"Techno-Social Excellence"

Marathwada Mitra Mandal's

# Institute of Technology

# Lohgaon, Pune

***Accredited with 'A' Grade by NAAC***


**"Towards Ubiquitous Computing Technology"**

## Department of Computer Engineering

# Laboratory Practice III: 410246

## Group A: Design and Analysis of Algorithms

**Prepared By**

**Prof. Chaitanya S Bhosale**

**BE COMP 2019 Course**

**Semester I Academic Year 2022-23**

**Programme Outcomes: As prescribed by NBA**

**1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.

**2. Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

**4. Conduct investigations of complex problems:** The problems that cannot be solved by straightforward application of knowledge, theories and techniques applicable to the engineering discipline.

**5. Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

**6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**10. Communication**: Communicate effectively on complex engineering activities with the engineering community and with t h e society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**11. Project management and finance:** Demonstrate knowledge and understanding of t h e engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**12. Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**Course Objectives:**

- Learn effect of data preprocessing on the performance of machine learning algorithms
- Develop in depth understanding for implementation of the regression models.
- Implement and evaluate supervised and unsupervised machine learning algorithms.
- Analyze performance of an algorithm.
- Learn how to implement algorithms that follow algorithm design strategies namely divide and conquer, greedy, dynamic programming, backtracking, branch and bound.
- Understand and explore the working of Blockchain technology and its applications.

**Course Outcomes:**

After completion of the course, students will be able to

CO1: Apply preprocessing techniques on datasets.

CO2: Implement and evaluate linear regression and random forest regression models.

CO3: Apply and evaluate classification and clustering techniques.

CO4: Analyze performance of an algorithm.

CO5: Implement an algorithm that follows one of the following algorithm design strategies: divide and conquer, greedy, dynamic programming, backtracking, branch and bound.

CO6: Interpret the basic concepts in Blockchain technology and its applications

# BE Computer Engineering 2019 Course Curriculum structure

**Savitribai Phule Pune University**

**Final Year of Computer Engineering (2019 Course)**

**(With effect from Academic Year 2022-23)**

## Semester VII

| Course Code | Course Name | Teaching Scheme (Hours/week) | | | Examination Scheme and Marks | | | | | | Credit Scheme | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Lecture | Practical | Tutorial | Mid-Sem | End-Sem | Term work | Practical | Oral\Pre | Total | Lecture | Practical | Tutorial | Total |
| 410241 | Design and Analysis of Algorithms | 03 | - | - | 30 | 70 | - | - | - | 100 | 3 | - | - | 3 |
| 410242 | Machine Learning | 03 | - | - | 30 | 70 | - | - | - | 100 | 3 | - | - | 3 |
| 410243 | Blockchain Technology | 03 | - | - | 30 | 70 | - | - | - | 100 | 3 | - | - | 3 |
| 410244 | Elective III | 03 | - | - | 30 | 70 | - | - | - | 100 | 3 | - | - | 3 |
| 410245 | Elective IV | 03 | - | - | 30 | 70 | - | - | - | 100 | 3 | - | - | 3 |
| 410246 | Laboratory Practice III | - | 04 | - | - | - | 50 | 50 | - | 100 | - | 2 | - | 2 |
| 410247 | Laboratory Practice IV | - | 02 | - | - | - | 50 | - | - | 50 | - | 1 | - | 1 |
| 410248 | Project Stage I | - | 02 | - | - | - | 50 | - | - | 50 | - | 2 | - | 2 |
| | | | | | | | | | | **Total Credit** | **15** | **05** | **-** | **20** |
| | **Total** | **15** | **08** | **-** | **150** | **350** | **150** | **50** | **-** | **700** | **15** | **05** | **-** | **20** |
| 410249 | Audit Course 7 | | | | | | | | | | **Grade** | | | |

| **Elective III** | **Elective IV** |
|---|---|
| 410244(A) Pervasive Computing<br>410244(B) Multimedia Techniques<br>410244(C) Cyber Security and Digital Forensics<br>410244(D) Object Oriented Modeling and Design<br>410244(E) Digital Signal Processing | 410245(A) Information Retrieval<br>410245(B) GPU Programming and Architecture<br>410245(C) Mobile Computing<br>410245(D)Software Testing and Quality Assurance<br>410245(E) Compilers |
| **Laboratory Practice III:**<br>Laboratory assignments Courses- 410241, 410242, 410243 | **Laboratory Practice IV:**<br>Laboratory assignments Courses- 410244, 410245 |

## Savitribai Phule Pune University
## Fourth Year of Computer Engineering (2019 Course)410246: Laboratory Practice III

| Teaching Scheme: Practical: 04 Hours/Week | Credit02 | Examination Scheme: Term work: 50 Marks Practical: 50 Marks |
|---|---|---|

**Companion Course:** Design and Analysis of Algorithms (410241), Machine Learning(410242), Blockchain Technology(410243)

**Course Objectives:**

- Learn effect of data preprocessing on the performance of machine learning algorithms
- Develop in depth understanding for implementation of the regression models.
- Implement and evaluate supervised and unsupervised machine learning algorithms.
- Analyze performance of an algorithm.
- Learn how to implement algorithms that follow algorithm design strategies namely divide and conquer, greedy, dynamic programming, backtracking, branch and bound.
- Understand and explore the working of Blockchain technology and its applications.

**Course Outcomes:**

After completion of the course, students will be able to
CO1: Apply preprocessing techniques on datasets.
CO2: Implement and evaluate linear regression and random forest regression models.CO3: Apply and evaluate classification and clustering techniques.
CO4: Analyze performance of an algorithm.
CO5: Implement an algorithm that follows one of the following algorithm design strategies: divide and conquer, greedy, dynamic programming, backtracking, branch and bound.
CO6: Interpret the basic concepts in Blockchain technology and its applications

**Guidelines for Instructor's Manual**

The instructor's manual is to be developed as a reference and hands-on resource. It should include prologue (about University/program/ institute/ department/foreword/ preface), curriculum of the course, conduction and assessment guidelines, topics under consideration, concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references.

**Guidelines for Student's Laboratory Journal**

The laboratory assignments are to be submitted by students in the form of a journal. Journal consistsof Certificate, table of contents, and handwritten write-up of each assignment (Title, Date of Completion, Objectives, Problem Statement, Software and Hardware requirements, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm, flowchart, test cases, Test Data Set(if applicable), mathematical model (if applicable), conclusion/analysis. Program codes with sample output of all performed assignments are to be submitted as a softcopy. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to a journal must be avoided. Use of DVD containing student programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journalsmay be maintained with program prints in the Laboratory.

**Guidelines for Laboratory /Term Work Assessment**

Continuous assessment of laboratory work should be based on overall performance of Laboratory assignments by a student. Assessment of each Laboratory assignment will assign grade/marks based on parameters, such as timely completion, performance, innovation, efficient codes, punctuality, documentation and neatness.

**Guidelines for Practical Examination**

Problem statements must be decided jointly by the internal examiner and external examiner. During practical assessment, maximum weightage should be given to satisfactory implementation of the problem statement. Relevant questions may be asked at the time of evaluation to test the student's understanding of the fundamentals, effective and efficient implementation. This will encourage, transparent evaluation and fair approach, and hence will not create any uncertainty or doubt in the minds of the students. So, adhering to these principles will consummate our team efforts to the promising start of student's academics.

**Guidelines for Laboratory Conduction**

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy needs to address the average students and inclusive of an element to attract and promote the intelligent students. Use ofopen source software is encouraged. Based on the concepts learned. Instructors may also set one assignment or mini-project that is suitable to each branch beyond the scope of the syllabus.
Operating System recommended :- 64-bit Open source Linux or its derivative
Programming tools recommended: - C++, Java, Python, Solidity, etc.

**Virtual Laboratory:**

- http://cse01-iiith.vlabs.ac.in/
- http://vlabs.iitb.ac.in/vlabs-dev/labs/blockchain/labs/index.php
- http://vlabs.iitb.ac.in/vlabs-dev/labs/machine_learning/labs/index.php

## Suggested List of Laboratory Experiments/Assignments.

**Assignments from all the Groups (A, B, C) are compulsory.**

## Course Contents

## Group A: Design and Analysis of Algorithms

Any 4 assignments and 1 mini project are mandatory.

| | |
|---|---|
| 1. | Write a program to calculate Fibonacci numbers and find its step count. |
| 2. | Implement job sequencing with deadlines using a greedy method. |
| 3. | Write a program to solve a fractional Knapsack problem using a greedy method. |
| 4. | Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch andbound strategy. |
| 5. | Write a program to generate binomial coefficients using dynamic programming. |
| 6. | Design 8-Queens matrix having first Queen placed. Use backtracking to place remainingQueens to generate the final 8-queen's matrix. |

| | |
|---|---|
| 7. | **Mini Project**<br>Write a program to implement matrix multiplication. Also implement multithreaded matrix multiplication with either one thread per row or one thread per cell. Analyze and compare theirperformance.<br>O<br>R<br>Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case.<br>O<br>R<br>Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.Observe difference in working of both the algorithms for the same input. |

## Group B: Machine Learning

Any 4 assignments and 1 Mini project are mandatory.

| | |
|---|---|
| 1. | Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:<br>1. Pre-process the dataset.<br>2. Identify outliers.<br>3. Check the correlation.<br>4. Implement linear regression and random forest regression models.<br>5. Evaluate the models and compare their respective scores like R2, RMSE, etc.Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset |

| | |
|---|---|
| 2. | Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.<br>Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv |
| 3. | Given a bank customer, build a neural network-based classifier that can determine whetherthey will leave or not in the next 6 months.<br>Dataset Description: The case study is from an open-source dataset from Kaggle.<br>The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc.<br>Link to the Kaggle project:<br>https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps:<br>1. Read the dataset.<br>2. Distinguish the feature and target set and divide the data set into training and test sets.<br>3. Normalize the train and test data.<br>4. Initialize and build the model. Identify the points of improvement and implement the same.<br>5. Print the accuracy score and confusion matrix (5 points). |
| 4. | Implement Gradient Descent Algorithm to find the local minima of a function.<br>For example, find the local minima of the function $y=(x+3)^2$ starting from the point x=2. |
| 5. | Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.<br><br>Dataset link : https://www.kaggle.com/datasets/abdallamahgoub/diabetes |

| | |
|---|---|
| 6. | Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method.<br><br>Dataset link : https://www.kaggle.com/datasets/kyanyoga/sample-sales-data |
| 7. | **Mini Project**<br><br>Use the following dataset to analyze ups and downs in the market and predict future stock price returns based on Indian Market data from 2000 to 2020.<br><br>Dataset Link: https://www.kaggle.com/datasets/sagara9595/stock-data<br>OR<br>Build a machine learning model that predicts the type of people who survived the Titanic shipwreck using passenger data (i.e. name, age, gender, socio-economic class, etc.).<br>Dataset Link: https://www.kaggle.com/competitions/titanic/data |
| **Group C: Blockchain Technology** | |
| Any 4 assignments and a Mini project are mandatory. | |
| 1. | Installation of Metamask and study spending Ether per transaction. |
| 2. | Create your own wallet using Metamask for crypto transactions. |
| 3. | Write a smart contract on a test network, for Bank account of a customer for following operations:<br>• Deposit money<br>• Withdraw Money<br>• Show balance |
| 4. | Write a program in solidity to create Student data. Use the following constructs:<br>• Structures<br>• Arrays<br>• Fallback<br>Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values. |
| 5. | Write a survey report on types of Blockchains and its real time use cases. |
| 6. | **Mini Project:** Create a dApp (de-centralized app) for e-voting system. |

**@The CO-PO Mapping Matrix**

| CO/ PO | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CO1 | 3 | 3 | 3 | 1 | 2 | 1 | - | 1 | 2 | - | 2 | 3 |
| CO2 | 3 | 3 | 3 | 2 | 2 | 1 | - | 1 | 2 | - | 2 | 3 |
| CO3 | 3 | 3 | 3 | 2 | 2 | 2 | - | 1 | 2 | - | 2 | 3 |
| CO4 | 3 | 2 | 2 | - | 1 | - | - | 1 | 2 | - | 2 | 2 |
| CO5 | 3 | 2 | 3 | - | 1 | - | - | 1 | 2 | - | - | 2 |
| CO6 | 3 | 3 | 2 | 2 | 2 | - | - | 1 | 2 | - | - | 2 |

# INDEX

| Sr. No. | Assignment No. | Title of Assignment | CO | PO |
|---|---|---|---|---|
| **1.** | **1.** | Write a program to calculate Fibonacci numbers and find its step count. | CO4 | |
| **2.** | **2.** | Implement job sequencing with deadlines using a greedy method. | CO5 | |
| **3.** | **3.** | Write a program to solve a fractional Knapsack problem using a greedy method. | CO5 | |
| **4.** | **4.** | Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch andbound strategy. | CO5 | |
| **5.** | **5.** | Write a program to generate binomial coefficients using dynamic programming. | CO5 | |
| **6.** | **6.** | Design 8-Queens matrix having first Queen placed. Use backtracking to place remainingQueens to generate the final 8-queen's matrix. | CO5 | |
| **7.** | **7.** | **Mini Project**<br>Write a program to implement matrix multiplication. Also implement multithreaded matrix multiplication with either one thread per row or one thread per cell. Analyze and compare their performance.<br>**OR**<br>Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case.<br>**OR**<br>Implement the Naive string-matching algorithm and Rabin-Karp algorithm for string matching. Observe difference in working of both the algorithms for the same input. | CO4 & CO5 | |

| Write-up | Correctness of Program | Documentation of Program | Viva | Timely Completion | Total | Dated Sign of Subject Teacher |
|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 20 | |
| | | | | | | |

**Date of Performance: .....** **Date of Completion: .....**

---

**GROUP: A**

**ASSIGNMENT NO: 1**

**Title of the Assignment:** Write a program to calculate Fibonacci numbers and find its step count.

**Prerequisites:** Basic C programming, If statement, For loop, While loop

**Objective of the Assignment:** To learn how to create the Fibonacci Series in Python/C++ using a loop, recursion, and dynamic programming.

**THEORY:**

In Mathematics, the Fibonacci Series is a sequence of numbers such that each number in the series is a sum of the preceding numbers. The series starts with 0 and 1.

Fibonacci Series is a pattern of numbers where each number results from adding the last two consecutive numbers. The first 2 numbers start with 0 and 1, and the third number in the sequence is 0+1=1. The 4th number is the addition of the 2nd and 3rd number, i.e., 1+1=2, and so on.

The Fibonacci Sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

## The logic of the Fibonacci Series

The following number is a sum of the two numbers before it.

The 3rd element is (1+0) = 1

The 4th element is (1+1) = 2

The 5th element is (2+1) = 3

## Fibonacci Series Formula

Hence, the formula for calculating the series is as follows:

xn = xn-1 + xn-2 ; where

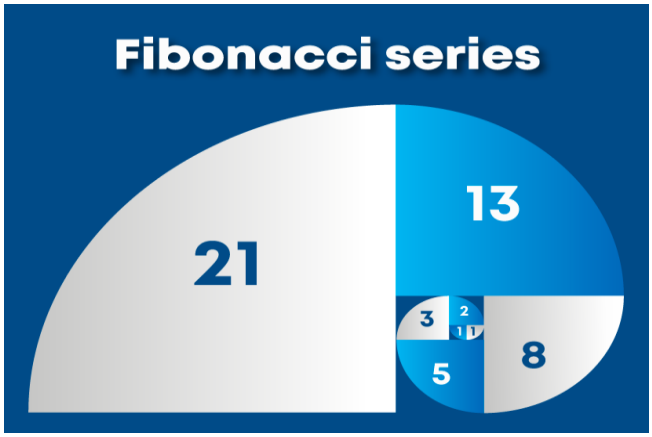
xn is the term number “n”

xn-1 is the previous term (n-1)

xn-2 is the term before that

## Fibonacci Spiral

An exciting property about these numbers is that we get a spiral when we make squares with these widths. A Fibonacci spiral is a pattern of quarter-circles connected inside a block of squares with Fibonacci numbers written in each of the blocks. The number in the giant square is a sum of the following 2 smaller squares. This is a perfect arrangement where each block is denoted a higher number than the previous two blocks. The main idea has been derived from the Logarithmic pattern, which also looks similar. These numbers are also related to the golden ratio.




## Iterative Approach

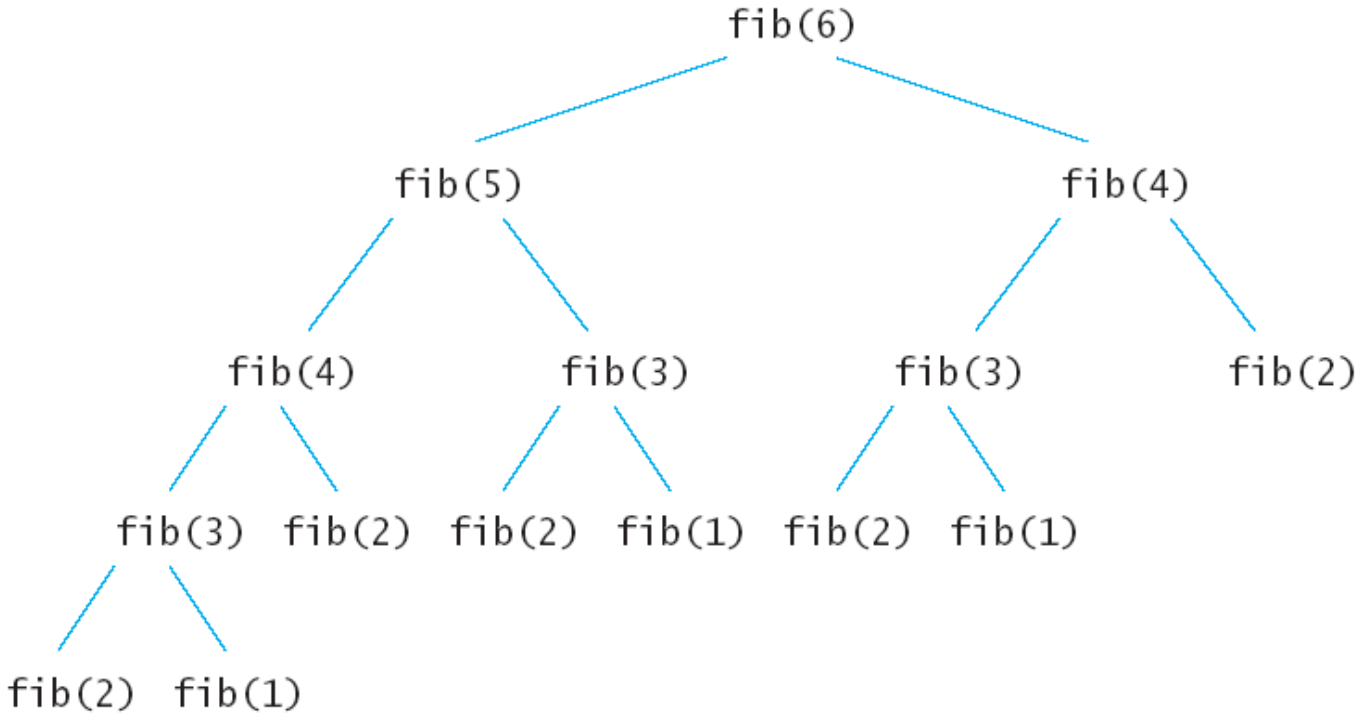
- Initialize variables a,b to 1
- Initialize for loop in range[1,n) # n exclusive
- Compute next number in series; total = a+b
- Store previous value in b
- Store total in a

## Recursive Approach

- If n equals 1 or 0; return 1
- Else return fib(n-1) + fib(n-2)

## Dynamic Programming Approach

- Initialize an array arr of size n to zeros
- If n equals 0 or 1; return 1 Else
- Initialize arr[0] and arr[1] to 1
- Run for loop in range[2,num]
- Compute the value arr[I]=arr[I-1] +arr[I-2]
- The array has the sequence computed till n




Hence, the solution would be to compute the value once and store it in an array from where it can be accessed the next time it is required. Therefore, we use dynamic programming in such cases. The conditions for implementing dynamic programming are
1. overlapping sub-problems
2. optimal substructure

## Iterative Approach

```
def fib_iter(n):
    a=1
    b=1
    if n==1:
        print('0')
    elif n==2:
        print('0','1')
    else:
        print("Iterative Approach: ", end=' ')
        print('0',a,b,end=' ')
        for i in range(n-3):
            total = a + b
            b=a
            a= total
            print(total,end=' ')
        print()
        return b

fib_iter(5)
```

## Recursive Approach

```
def fib_rec(n):
    if n == 1:
        return [0]
    elif n == 2:
        return [0,1]
    else:
        x = fib_rec(n-1)
        # the new element the sum of the last two elements
        x.append(sum(x[:-3:-1]))
        return x
x=fib_rec(5)
print(x)
```

## Dynamic Programming Approach

There **is** a slight modification to the iterative approach. We use an additional array.

```
def fib_dp(num):
    arr = [0,1]
    print("Dynamic Programming Approach: ",end= ' ')
    if num==1:
        print('0')
    elif num==2:
        print('[0,','1]')
    else:
        while(len(arr)<num):
            arr.append(0)
        if(num==0 or num==1):
            return 1
        else:
            arr[0]=0
            arr[1]=1
            for i in range(2,num):
                arr[i]=arr[i-1]+arr[i-2]
            print(arr)
            return arr[num-2]
fib_dp(5)
```

**CONCLUSION**: Hence we have studied to calculate Fibonacci numbers and find its step count.

## FAQs

**Q. What are the properties of the Fibonacci series?**
The Fibonacci series has several properties, including:
-Each number in the series is the sum of the two preceding numbers.
-The first two numbers in the series are 0 and 1.

**Q. What are some applications of the Fibonacci series?**
The Fibonacci series has several applications, including:
-It can be used to model the growth of populations of animals.
-It can be used to calculate the Golden Ratio, which is used in architecture and art.
-It can be used in computer programming to generate efficient algorithms.

**Q. What is the time complexity of generating the Fibonacci series?**
The time complexity of generating the Fibonacci series is O(n).
**Q. What is the space complexity of storing the Fibonacci series?**
The Fibonacci series is an infinite sequence, so the space complexity is infinite.

| Write-up | Correctness of Program | Documentation of Program | Viva | Timely Completion | Total | Dated Sign of Subject Teacher |
|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 20 | |
| | | | | | | |

**Date of Performance: .....** **Date of Completion: .....**

---

**GROUP: A**

**ASSIGNMENT NO: 2**

**Title of the Assignment:** Implement job sequencing with deadlines using a greedy method.

**Prerequisites:** Basic C, C++, Java programming, If statement, For loop, While loop

**Objective of the Assignment:** To learn what the Job Sequencing with Deadlines Problem is and how to solve the Job Sequencing Problem in C, C++, and Java. In job sequencing problem, the objective is to find a sequence of jobs, which is completed within their deadlines and gives maximum profit.

**THEORY:**

## What is Job Sequencing with Deadlines Problem?

Suppose you are a freelancer who has got 5 different jobs to do with the respective deadlines and profit/payment. Also, you are not sure whether you will be able to complete all the jobs before the respective deadlines.

So, which job will you start working on first?

- **The job which gives you maximum profit**: In this case, the job may have a long deadline and by doing this job you may miss some jobs with a short deadline. And it can be unprofitable to you if the sum of profits from missed jobs is greater than the profit of the job which you are doing.
- **The job which has a short deadline**: In this case, It may happen that you have completed multiple jobs within the given deadlines but there could be the possibility that you could have earned more if you would have chosen the job with max profit.

To solve the ambiguity of which job to choose, we use greedy approach.

## Job Sequencing with Deadlines Solution using Greedy Algorithm

1. Sort the jobs for their profit in descending order.
2. Choose the uncompleted job with high profit (i.e. first job in the array, since the array is sorted). Because it is not necessary to complete the job on the very first date, we will do/complete the job on the last day of the deadline (i.e. **Add the job to a new array/list at the index equal to its deadline day**).
3. Now again choose the next uncompleted high-profit job. Check its deadline. If its deadline is greater than the deadline of the previous chosen job, then we will do/complete this job on the last day of the deadline only if the day is empty, else will do the job on the previous empty day which is nearest to the deadline ( i.e. **Add the job to new array/list at the index equal or nearest to its deadline day, only if the array is empty at particular index**).
4. If you don't have the empty day before the job deadline (i.e. all the array index is occupied before the index=job. Deadline) then do not do the job.
5. Repeat the process for every job.
6. The final array or list will give the best jobs to do for max profit.

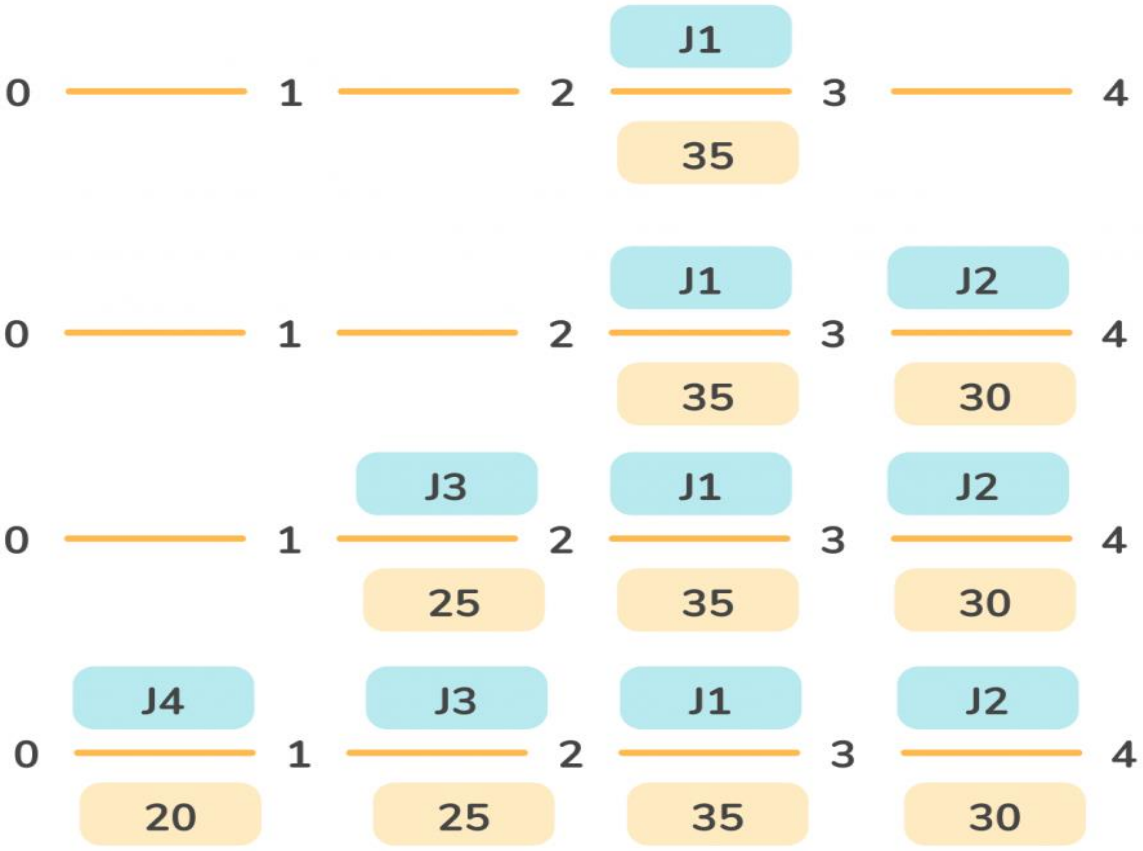
***In short, we are being greedy (choosing a job with max profit) and lazy (completing a job on the day of the deadline).***

**Example:**
**Input:**

n=7

| Jobs | J1 | J2 | J3 | J4 | J5 | J6 | J7 |
|---|---|---|---|---|---|---|---|
| Profits | 35 | 30 | 25 | 20 | 15 | 12 | 5 |
| Deadlines | 3 | 4 | 4 | 2 | 3 | 1 | 2 |

**Output:** J4 J3 J1 J2
**Explanation:**




**Total profit** – 20 + 25 + 35 + 30 = 110

**Approach : Greedy Algorithm**

Since, the task is to get the maximum profit by scheduling the **jobs**, the idea is to approach this problem greedily.

**Algorithm**

- Sort the jobs based on decreasing order of profit.
- Iterate through the jobs and perform the following:
    - Choose a **Slot i** if:
        - **Slot i** isn't previously selected.
        - **I < deadline**

- **I** is maximum

If no such slot exists, ignore the job and continue.

**Dry Run with Example**

Given Jobs:

| Job ID | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Deadline | 2 | 3 | 2 | 1 | 3 |
| Profit | 20 | 38 | 16 | 10 | 30 |

Sort in decreasing order of profits:

| Job ID | 2 | 5 | 1 | 3 | 4 |
|---|---|---|---|---|---|
| Deadline | 3 | 3 | 2 | 2 | 1 |
| Profit | 38 | 30 | 20 | 16 | 10 |

1. The last empty slot available for Job 2 before deadline is slot 2-3


2. The last empty slot available for Job 5 before deadline is slot 1-2


2. The last empty slot available for Job 1 before deadline is slot 0-1


All the slots are full. So, the sequence of jobs is : 1 5 2

**Time Complexity: O($N^2$)where N is the size of the jobs array**
**Space Complexity: O(1), since no extra space is used.**

**CONCLUSION**: Hence we have studied the implementation of job sequencing with deadlines using a greedy method.

## FAQs

**Q. Do we need to sort the jobs array?**

**Q. What is the most efficient algorithm to solve the Job Sequencing problem?**

**Q. Solve the following job scheduling with deadlines problem using the greedy method.**
**Number of jobs N = 4. Profits associated with Jobs : (P1, P2, P3, P4) = (100, 10, 15, 27).**
**Deadlines associated with jobs (d1, d2, d3, d4) = (2, 1, 2, 1)**

| Write-up | Correctness of Program | Documentation of Program | Viva | Timely Completion | Total | Dated Sign of Subject Teacher |
|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 20 | |
| | | | | | | |

**Date of Performance: .....** **Date of Completion: .....**

---

**GROUP: A**

**ASSIGNMENT NO: 3**

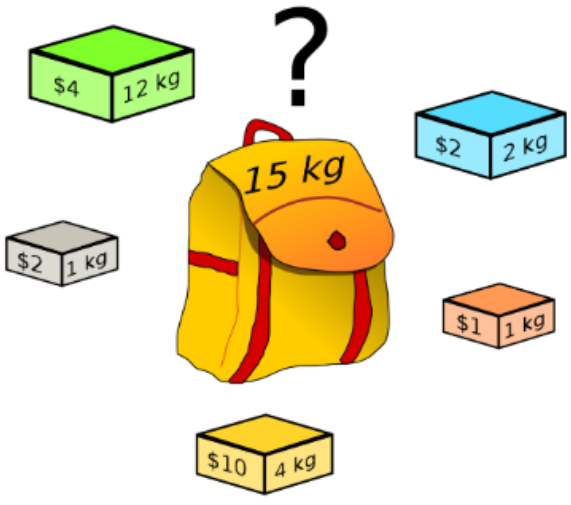
**Title of the Assignment:** Write a program to solve a fractional Knapsack problem using a greedy method.

**Prerequisites:** Basic C, C++, Java programming, If statement, For loop, While loop

**Objective of the Assignment:** To learn to solve a fractional Knapsack problem using a greedy method.

**THEORY:**

Fractional Knapsack problem is defined as, "Given a set of items having some weight and value/profit associated with it. The knapsack problem is to find the set of items such that the total weight is less than or equal to a given limit (size of knapsack) and the total value/profit earned is as large as possible."




Knapsack Problem

**Knapsack problem has two variants.**

- **Binary or 0/1 knapsack :** Item cannot be broken down into parts.
- **Fractional knapsack :** Item can be divided into parts.

Given a set of items, each having some weight and value/profit associated with it. The goal is to find the set of items such that the total weight is less than or equal to a given limit (size of knapsack) and the total value/profit earned is as large as possible.

- The knapsack is an optimization problem and it is useful in solving resource allocation problem.
- Let X = $<x_1, x_2, x_3, \ldots, x_n>$ is the set of n items. W = $<w_1, w_2, w_3, \ldots, w_n>$ and V = $<v_1, v_2, v_3, \ldots, v_n>$ are the set of weight and value associated with each items in x, respectively. Knapsack capacity is M.
- Select items one by one from the set of items x and fill the knapsack such that it would maximize the value. Knapsack problem has two variants. 0/1 knapsack does not allow breaking of items. Either add entire item in a knapsack or reject it. It is also known as a binary knapsack. Fractional knapsack allows the breaking of items. So profit will also be considered accordingly.
- Knapsack problem can be formulated as follow :

Maximize sum_{i=1}^{n} v_i x_i$sum_{i=1}^{n} v_i x_i$ subjected to sum_{i=1}^{n} v_i x_i le M$sum_{i=1}^{n} v_i x_i leq M$

x_i in (0, 1)$x_i in (0,1)$ for binary knapsack

x_i in [0, 1]$x_i in [0,1]$ for fractional knapsack

**Algorithm GREEDY_FRACTIONAL_KNAPSACK(X, V, W, M)**

// Description : Solve the knapsack problem using greedy approach

// Input:
X: An array of n items
V: An array of profit associated with each item
W: An array of weight associated with each item
M: Capacity of knapsack

// Output :
SW: Weight of selected items
SP: Profit of selected items
// Items are presorted in decreasing order of pi = vi / wi ratio

S ← Φ // Set of selected items, initially empty

```
SW ← 0    // weight of selected items
SP ← 0    // profit of selected items
i ← 1

while i ≤ n do
   if (SW + w[i]) ≤ M then
      S ← S ∪ X[i]
      SW ← SW + W[i]
      SP ← SP + V[i]
   else
      frac ← (M - SW) / W[i]
      S ← S ∪ X[i] * frac       // Add fraction of item X[i]
      SP ← SP + V[i] * frac     // Add fraction of profit
      SW ← SW + W[i] * frac     // Add fraction of weight
   end
   i ← i + 1
end
```

**Complexity Analysis**

For one item there are two choices, either to select or reject. For 2 items we have four choices:

- Select both items
- Reject both items
- Select first and reject second
- Reject first and select second
- In general, for n items, knapsack has $2^n$ choices. So brute force approach runs in $O(2^n)$ time.
- We can improve performance by sorting items in advance. Using merge sort or heap sort, n items can be sorted in $O(n\log_2 n)$ time. Merge sort and heap sort are non-adaptive and their running time is the same in best, average and worst case.
- To select the items, we need one scan to this sorted list, which will take O(n) time.
- So the total time required is
  $T(n) = O(n\log_2 n) + O(n) = O(n\log_2 n)$.

**Example of Fractional Knapsack**

**Problem: Consider the following instances of the fractional knapsack problem: n = 3, M = 20, V = (24, 25, 15) and W = (18, 15, 20) find the feasible solutions.**

**Solution:**

Let us arrange items by decreasing order of profit density. Assume that items are labeled as X = ($I_1$, $I_2$, $I_3$), have profit V = {24, 25, 15} and weight W = {18, 15, 20}.

| **Item ($x_i$)** | **Value ($v_i$)** | **Weight ($w_i$)** | $p_i = v_i / w_i$ |
|---|---|---|---|
| $I_2$ | 25 | 15 | 1.67 |
| $I_1$ | 24 | 18 | 1.33 |
| $I_3$ | 15 | 20 | 0.75 |

We shall select one by one item from Table. If the inclusion of an item does not cross the knapsack capacity, then add it. Otherwise, break the current item and select only the portion of item equivalent to remaining knapsack capacity. Select the profit accordingly. We should stop when knapsack is full or all items are scanned.

Initialize, Weight of selected items, SW = 0,

Profit of selected items, SP = 0,

Set of selected items, S = { },

Here, Knapsack capacity M = 20.

**Iteration 1 :** SW= (SW + $w_2$) = 0 + 15 = 15
SW ≤ M, so select $I_2$
S = { $I_2$ }, SW = 15, SP = 0 + 25 = 25
**Iteration 2 :** SW + $w_1$ > M, so break down item $I_1$.
The remaining capacity of the knapsack is 5 unit, so select only 5 units of item $I_1$.
*frac* = (M – SW) / W[i] = (20 – 15) / 18 = 5 / 18
S = { $I_2$, $I_1$ * 5/18 }
SP = SP + $v_1$ * *frac* = 25 + (24 * (5/18)) = 25 + 6.67 = 31.67
SW = SW + $w_1$ * *frac* = 15 + (18 * (5/18)) = 15 + 5 = 20
**The knapsack is full. Fractional <u>Greedy algorithm</u> selects items { $I_2$, $I_1$ * 5/18 }, and it gives a profit of 31.67 units.**

**CONCLUSION**: Hence we have studied to solve a fractional Knapsack problem using a greedy method.

# FAQs:

**Q. What are the applications of the fractional knapsack problem using greedy method?**

**Q. Why is fractional knapsack problem using greedy method?**

**Q. How do you solve fractional knapsack problem using greedy method?**

**Q. Find the optimal solution for the fractional knapsack problem making use of greedy approach.**

**Consider-**

**n = 5**

**w = 60 kg**

**(w1, w2, w3, w4, w5) = (5, 10, 15, 22, 25)**

**(b1, b2, b3, b4, b5) = (30, 40, 45, 77, 90)**

| Write-up | Correctness of Program | Documentation of Program | Viva | Timely Completion | Total | Dated Sign of Subject Teacher |
|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 20 | |
| | | | | | | |

**Date of Performance: .....** **Date of Completion: .....**

---

**GROUP: A**

**ASSIGNMENT NO: 4**

**Title of the Assignment:** Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

**Prerequisites:** Basic C, C++, Java programming, If statement, For loop, While loop

**Objective of the Assignment:** To learn to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

**THEORY:**

## Problem Statement

We are a given a set of *n* objects which have each have a value $v_i$ and a weight $w_i$. The objective of the 0/1 Knapsack problem is to find a subset of objects such that the total value is maximized, and the sum of weights of the objects does not exceed a given threshold *W*. An important condition here is that one can either take the entire object or leave it. It is not possible to take a fraction of the object.
Consider an example where $n = 4$, and the values are given by {10, 12, 12, 18}and the weights given by {2, 4, 6, 9}. The maximum weight is given by $W = 15$. Here, the solution to the problem will be including the first, third and the fourth objects.

## Approaches to solve this problem

The first idea that comes to mind as soon as we look at the problem would be to look at all possible combinations of objects, calculate their total weight, and if the total weight is less than the threshold, to calculate the total value (This approach is known as the *Brute Force*. Although this approach would give us the solution, it is of exponential time complexity. Hence, we look at the other possible methods. We can use the *Dynamic Programming* approach to solve this problem as well. Although this method is far more efficient than the Brute Force method, it does not work in scenarios where the item weights are non-integer values.

*Backtracking* can also be used to solve this problem. However, this would mean exploring all possible branches until the solution is invalid, then going back a step and exploring other possibilities. As was in the case of the brute force method, this method also has exponential time complexity. Since this is a combinatorial problem, once can use the *Branch and Bound* method to solve this problem. We shall explore this way of solving this problem in detail.

## Branch and Bound Method

In solving this problem, we shall use the Least Cost- Branch and Bound method, since this shall help us eliminate exploring certain branches of the tree. We shall also be using the fixed-size solution here. Another thing to be noted here is that this problem is a maximization problem, whereas the Branch and Bound method is for minimization problems. Hence, the values will be multiplied by -1 so that this problem gets converted into a minimization problem.

Now, consider the 0/1 knapsack problem with *n* objects and total weight W. We define the upper bound(U) to be the summation of $v_i x_i$ (where $v_i$ denotes the value of that objects, and $x_i$ is a binary value, which indicates whether the object is to be included or not), such that the total weights of the included objects is less than *W*. The initial value of U is calculated at the initial position, where objects are added in order until the initial position is filled.

We define the cost function to be the summation of $v_i f_i$, such that the total value is the maximum that can be obtained which is less than or equal to *W*. Here $f_i$ indicates the fraction of the object that is to be included. Although we use fractions here, it is not included in the final solution.
Here, the procedure to solve the problem is as follows are:

- Calculate the cost function and the Upper bound for the two children of each node. Here, the $(i + 1)^{th}$ level indicates whether the $i^{th}$ object is to be included or not.
- If the cost function for a given node is greater than the upper bound, then the node need not be explored further. Hence, we can kill this node. Otherwise, calculate the upper bound for this node. If this value is less than *U*, then replace the value of *U* with this value. Then, kill all unexplored nodes which have cost function greater than this value.
- The next node to be checked after reaching all nodes in a particular level will be the one with the least cost function value among the unexplored nodes.
- While including an object, one needs to check whether the adding the object crossed the threshold. If it does, one has reached the terminal point in that branch, and all the succeeding objects will not be included.

In this manner, we shall find a value of U at the end which eliminates all other possibilities. The path to this node will determine the solution to this problem.

**Algorithm for Knapsack Problem using Branch and Bound**

```
Algorithm BB_KNAPSACK(cp, cw, k)
// Description : Solve knapsack problem using branch and bound
// Input:
cp: Current profit, initially 0
cw: Current weight, initially 0
k: Index of item being processed, initially 1

// Output:
fp: Final profit
Fw: Final weight
X: Solution tuple

cp ← 0
cw ← 0
k ← 1

if (cw + w[k] ≤ M) then
  Y[k]  ← 1
  if (k < n) then
    BB_KNAPSACK(cp + p[k], cw + w[k], k + 1)
  end
  if (cp + p[k] > fp) && (k == n) then
    fp ← cp + c[k]
    fw ← cw + w[k]
    X ← Y
  end
end

if BOUND(cp, cw, k) ≥ fp then
  Y[k] ← 0
  if (k < n) then
    BB_KNAPSACK(cp, cw, k + 1)
  end
  if( cp>fp ) && ( k == n ) then
    fp ← cp
    fw ← cw
    X ← Y
  end
end
```

Upper bound is computed as follow :

```
Function BOUND(cp, cw, k)

b ← cp
c ← cw
for i ← k + 1 to n do
  if (c + w[i] ≤ M) then
    c ← c + w[i]
    b ← b – p[i]
  end
end
return b
```

BB_KNAPSACK(0, 0, 1) would be the first call.

**LCBB for Knapsack**

LC branch and bound solution for knapsack problem is derived as follows :

- Derive state space tree.
- Compute lower bound hat{c}(.)*hatc*(.) and upper bound u(.)*u*(.) for each node in state space tree.
- If lower bound is greater than upper bound than kill that node.
- Else select node with minimum lower bound as E-node.
- Repeat step 3 and 4 until all nodes are examined.
- The node with minimum lower bound value hat{c}(.)*hatc*(.) is the answer node. Trace the path from leaf to root in the backward direction to find the solution tuple.

**Variation:**

The bounding function is a heuristic computation. For the same problem, there may be different bounding functions. Apart from the above-discussed bounding function, another very popular bounding function for knapsack is,

$$ub = v + (W - w) * (v_{i+1} / w_{i+1})$$

where,

v is value/profit associated with selected items from the first i items.

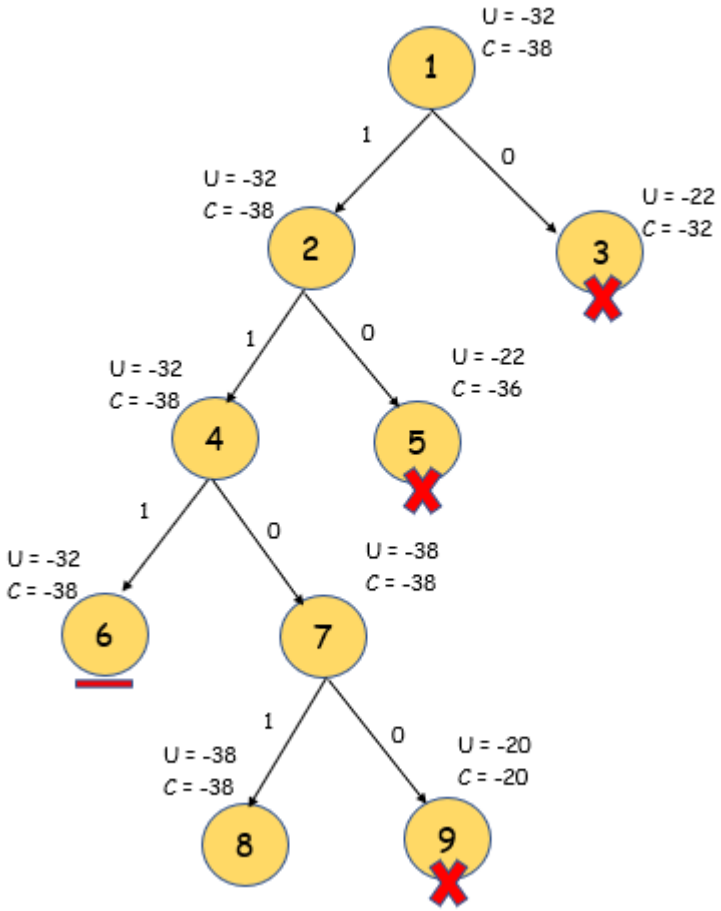
W is the capacity of the knapsack.

w is the weight of selected items from first i items

**Solving an Example**

Consider the problem with n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15. Here, we calculate the initital upper bound to be U = 10 + 10 + 12 = 32. Note that the 4th object cannot be included here, since that would exceed W. For the cost, we add $3/9^{th}$ of the final value, and hence the cost function is 38. Remember to negate the values after calculation before comparison.

After calculating the cost at each node, kill nodes that do not need exploring. Hence, the final state space tree will be as follows (Here, the number of the node denotes the order in which the state space tree was explored):




Note here that node 3 and node 5 have been killed after updating U at node 7. Also, node 6 is not explored further, since adding any more weight exceeds the threshold. At the end, only nodes 6 and 8 remain. Since the value of U is less for node 8, we select this node. Hence the solution is {1, 1, 0, 1}, and we can see here that the total weight is exactly equal to the threshold value in this case.

We can see that the branch and bound method will give the solution to the problem by exploring just 1 path in the best case. Although the worst case will be of exponential time complexity, this method will perform better than the other methods in most scenarios, since multiple branches get eliminated in each iteration.

## 0/1 Knapsack Problem Using Dynamic Programming

0/1 knapsack problem is solved using dynamic programming in the following steps-

**Step-01:**

- Draw a table say ‘T’ with (n+1) number of rows and (w+1) number of columns.
- Fill all the boxes of $0^{th}$ row and $0^{th}$ column with zeroes as shown-

| | 0 | 1 | 2 | 3 | ..... | W |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | ..... | 0 |
| 1 | 0 | | | | | |
| 2 | 0 | | | | | |
| | ..... | | | | | |
| n | 0 | | | | | |

T-Table

## Step-02:

Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, j) = \max \{ T(i-1, j), value_i + T(i-1, j - weight_i) \}$$

Here, T(i , j) = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.

- This step leads to completely filling the table.
- Then, value of the last box represents the maximum possible value that can be put into the knapsack.

## Step-03:

To identify the items that must be put into the knapsack to obtain that maximum profit,

- Consider the last column of the table.
- Start scanning the entries from bottom to top.
- On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.

- After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

**Time Complexity-**

- Each entry of the table requires constant time $\theta(1)$ for its computation.
- It takes $\theta(nw)$ time to fill $(n+1)(w+1)$ table entries.
- It takes $\theta(n)$ time for tracing the solution since tracing process traces the n rows.
- Thus, overall $\theta(nw)$ time is taken to solve 0/1 knapsack problem using dynamic programming.

**CONCLUSION**: Hence we have studied to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

# FAQs:

**Q. Can we solve the 0/1 Knapsack Problem using Backtracking?**

Yes, the recursive DP approach itself is the backtracking approach for 0/1 knapsack.

**Q. What is the Time Complexity of 0/1 Knapsack Problem?**

Time complexity for 0/1 Knapsack problem solved using DP is O(N*W) where N denotes number of items available and W denotes the capacity of the knapsack.

**Q. Can we solve the 0/1 Knapsack Problem using Greedy Algorithm?**

No, 0/1 Knapsack Problem cannot be solved using a greedy approach.

| Write-up | Correctness of Program | Documentation of Program | Viva | Timely Completion | Total | Dated Sign of Subject Teacher |
|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 20 | |
| | | | | | | |

**Date of Performance: .....** **Date of Completion: .....**

---

**GROUP: A**

**ASSIGNMENT NO: 5**

**Title of the Assignment:** Write a program to generate binomial coefficients using dynamic programming.

**Prerequisites:** Basic C, C++, Java programming, If statement, For loop, While loop

**Objective of the Assignment:** To learn to generate binomial coefficients using dynamic programming

**THEORY:**

Computing binomial coefficient is very fundamental problem of mathematics and computer science. Binomial coefficient C(n, k) defines coefficient of the term $x^n$ in the expansion of $(1 + x)^n$. C(n, k) also defines the number of ways to select any k items out of n items. Mathematically it is defined as,

$$C(n, k) = \frac{n!}{(n-k)!\,k!} \quad (0 \le k \le n)$$

C(n, k) can be found in different ways. In this article we will discuss divide and conquer as well as dynamic programing approach.

## Binomial Coefficient using Dynamic Programming

- Many sub problems are called again and again, since they have an overlapping sub problems property. Re-computations of the same sub problems is avoided by storing their results in the temporary array C[i, j] in a bottom up manner.
- The optimal substructure for using dynamic programming is stated as,
- $$C[i, j] = \begin{cases} 1 & , \text{if } i = j \text{ or } j = 0 \\ C[i-1, j-1], + C[i-1, j] & \text{otherwise} \end{cases}$$

In Table, index i indicates row and index j indicates column

| n\k | 0 | 1 | 2 | 3 | 4 | . | . | . | (k – 1) | k |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | | | | | | | | | |
| **1** | 1 | 1 | | | | | | | | |
| **2** | 1 | 2 | 1 | | | | | | | |
| **3** | 1 | 3 | 3 | 1 | | | | | | |
| **4** | 1 | 4 | 6 | 4 | 1 | | | | | |
| **.** | . | | | | | | | | | |
| **.** | . | | | | | | | | | |
| **k** | 1 | | | | | | | | | 1 |
| **.** | . | | | | | | | | | |
| **.** | . | | | | | | | | | |
| **n – 1** | 1 | | | | | | | | C(n – 1, k – 1) | C(n – 1, k) |
| **n** | 1 | | | | | | | | | C(n, k) |

- This tabular representation of binomial coefficients is also known as Pascal's triangle.
- Algorithm to solve this problem using dynamic programming is shown below

```
Algorithm BINOMIAL_DC (n, k)
// n is total number of items
// k is the number of items to be selected from n

if k == 0 or k == n then
   return 1
else
   return DC_BINOMIAL(n – 1, k – 1) + DC_BINOMIAL(n – 1, k)
end
```

**Algorithm**

- **Step 1** : Get the two inputs, the positive value of n and the non-positive value of k which denotes the k-th binomial coefficient in the Binomial Expansion.
- **Step 2** : Allocate the array of size k + 1 with the value of 1 at 0-th index and rest with value 0.
- **Step 3** : Next, generating the sequence of pascal's triangle, with the first row containing single element valued 1 which was already created in step 2.
- **Step 4** : Further next consecutive rows of pascal's triangle are computed from the previous row by adding the two consecutive elements, but step 4 is to be carried out upto k-times, for enclosing n-value times.
- **Step 5** : Stop.

**Complexity**

The Time and Space Complexities of code by using Dynamic Programming Approach are :

- Time complexity: **O(n*k)**
- Space complexity: **k**

T(n, k) = sum for upper triangle + sum for the lower rectangle

$$T(n,k) = \sum_{i=1}^{k} \sum_{j=1}^{i-1} 1 + \sum_{i=k+1}^{n} \sum_{j=1}^{k} 1 = \sum_{i=1}^{k} (i-1) + \sum_{i=k+1}^{k} k$$

$$= (1 + 2 + 3 + \ldots + k - 1) + k \sum_{i=k+1}^{n} 1 = \frac{(k-1)\,k}{2} + k\,(n-k)$$

$$= \frac{k^2 - k}{2} + nk - k^2 = \frac{k^2 - k + 2nk - 2k^2}{2} = nk - \frac{k^2}{2} - \frac{k}{2}$$

$$= nk \qquad (k \ll n)\ldots$$

$$T(n,k) = O(nk)$$

**CONCLUSION:** Hence we have studied to solve a generate binomial coefficients using dynamic programming

FAQs:

Q. How do you find the binomial coefficient in dynamic programming?

| Write-up | Correctness of Program | Documentation of Program | Viva | Timely Completion | Total | Dated Sign of Subject Teacher |
|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 20 | |
| | | | | | | |

**Date of Performance: .....** **Date of Completion: .....**

---

**GROUP: A**

**ASSIGNMENT NO: 6**

**Title of the Assignment:** Design 8-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final 8-queen's matrix

**Prerequisites:** Basic C, C++, Java programming, If statement, For loop, While loop

**Objective of the Assignment:** To learn design of 8-Queens matrix

**THEORY:**

**What is Backtracking?**

In backtracking, we start with one possible move out of many available moves. We then try to solve the problem.

If we are able to solve the problem with the selected move then we will print the solution. Else we will backtrack and select some other move and try to solve it.

If none of the moves works out we claim that there is no solution for the problem.

**What is the N-Queens Problem?**

This problem is commonly seen for N=4 and N=8.

Let's look at an example where N=4

Before solving the problem, you need to know about the movement of the queen in chess.

A queen can move any number of steps in any direction. The only constraint is that it can't change its direction while it's moving.

One thing that is clear by looking at the queen's movement is that no two queens can be in the same row or column.

That allows us to place only one queen in each row and each column.

When N=4, the solution looks like :

Given an 8 x 8 chessboard, arrange 8 queens in a way such that no two queens attack each other.

Two queens are attacking each other if they are in the same row, column, or diagonal. Cells attacked by queen Q are shown in fig..

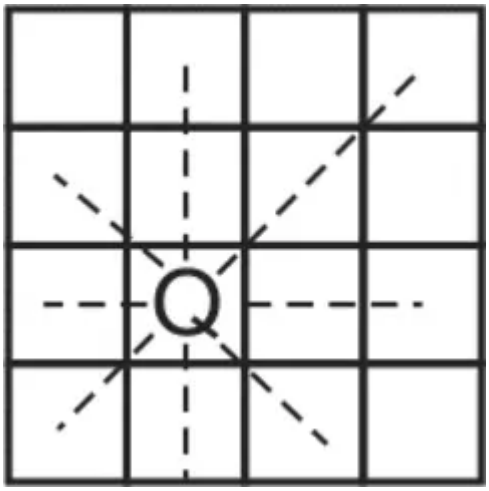



**Fig. 1 Attacked cells by queen Q**

8 queen problem has 64C8= 4,42,61,65,368 different arrangements. Of these, only 92 arrangements are valid solutions. Out of which, only 12 are the fundamental solutions. The remaining 80 solutions can be generated using reflection and rotation.

The 2-queen problem is not feasible. The minimum problem size for which a solution can be found is 4. Let us understand the workings of backtracking on the 4-queen problem.

For simplicity, a partial state space tree is shown in fig. (f). Queen 1 is placed in the first column in the first row. All the positions are crossed in which Queen 1 is attacking. In the next level, queen 2 is placed in a 3rd column in row 2 and all cells that are crossed are attacked by already placed queens 1 and 2. As can be seen from fig (f), no place is left to place the next queen in row 3, so queen 2 backtracks to the next possible position and the process continues. In a similar way, if (1, 1) position is not feasible for queen 1,

then the algorithm backtracks and puts the first queen in cell (1, 2), and repeats the procedure. For simplicity, only a few nodes are shown in fig.

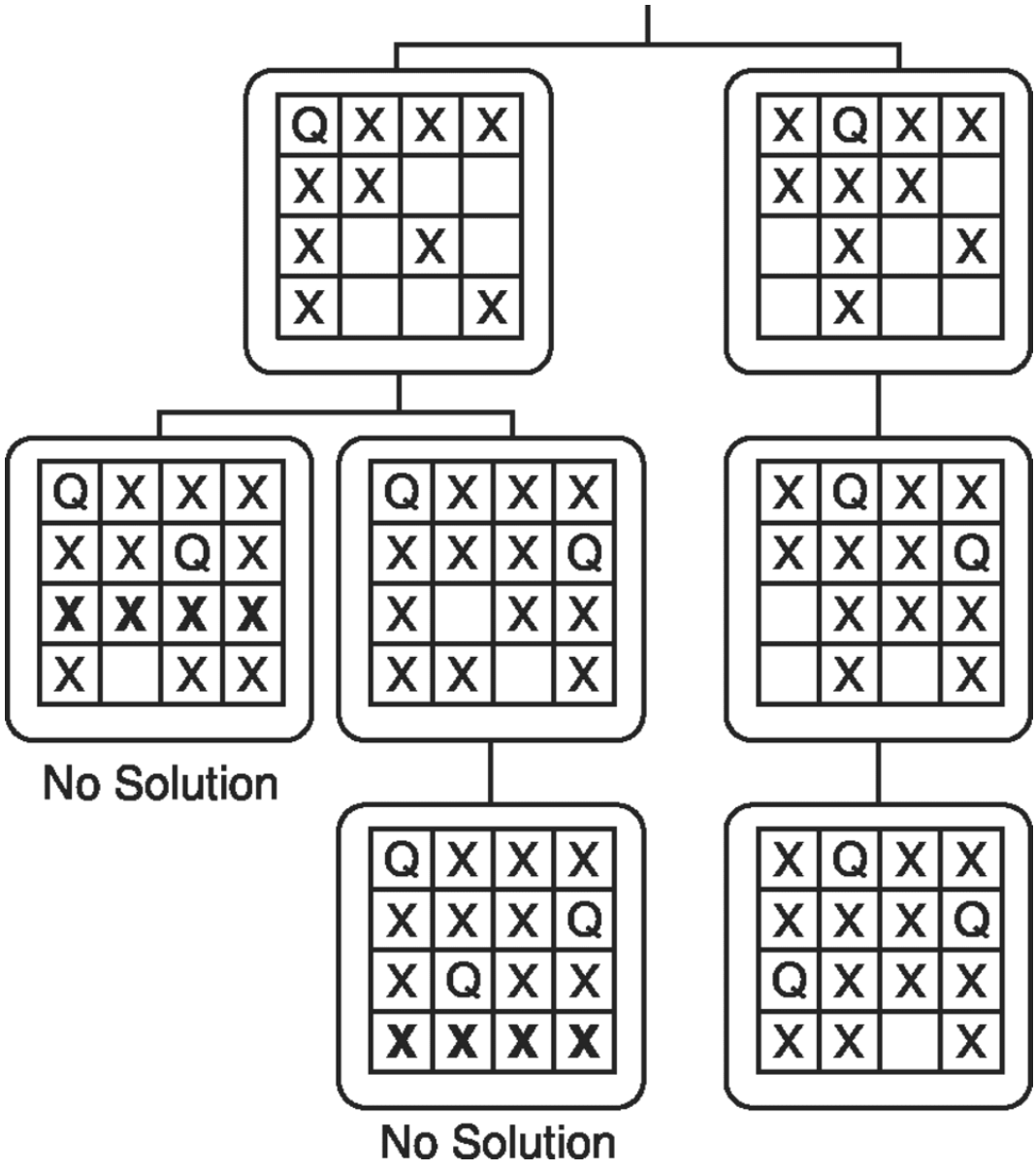



Fig. 2 : Snapshot of backtracking procedure of 4-Queen problem

- A complete state space tree for the 4-queen problem is shown in fig. (g)
- The number within the circle indicates the order in which the node gets explored. The height of the e from the t indicates row and label, besides the arc indicating that the Q is placed in an ith column. Out of all the possible states, only a few are the answer states.

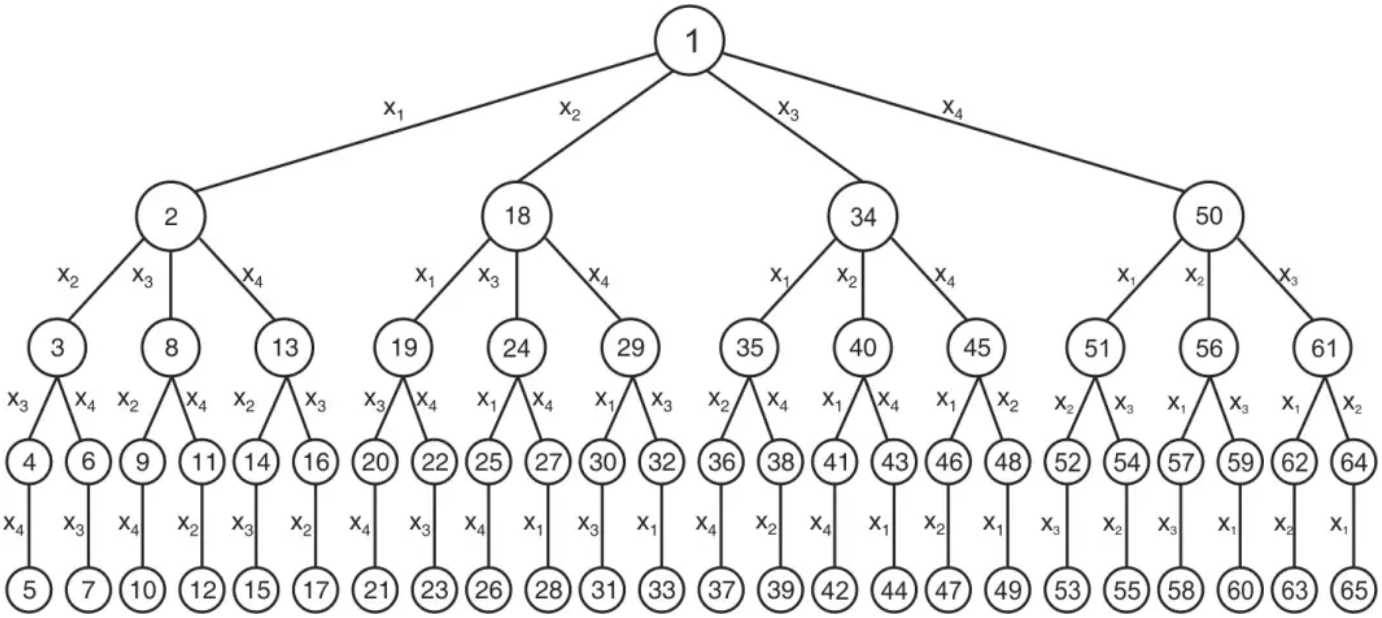



**Fig. 3: Solution of 8-queen problem**

- Solution tuple for the solution shown in fig (h) is defined as <4, 6, 8, 2, 7, 1, 3, 5>.

From observations, two queens placed at (i, j) and (k, l) positions, can be in same diagonal only if,

$$(i - j) = (k - l) \text{ or}$$

$$(i + j) = (k + l)$$

From first equality, $j - l = i - k$

From second equality, $j - l = k - i$

So queens can be in diagonal only if $|j - l| = |i - k|$.

The arrangement shown in fig. (i) leads to failure. As it can be seen from fig. (i), Queen Q6 cannot be placed anywhere in the 6th row. So the position of Q5 is backtracked and it is placed in another feasible cell. This process is repeated until the solution is found.

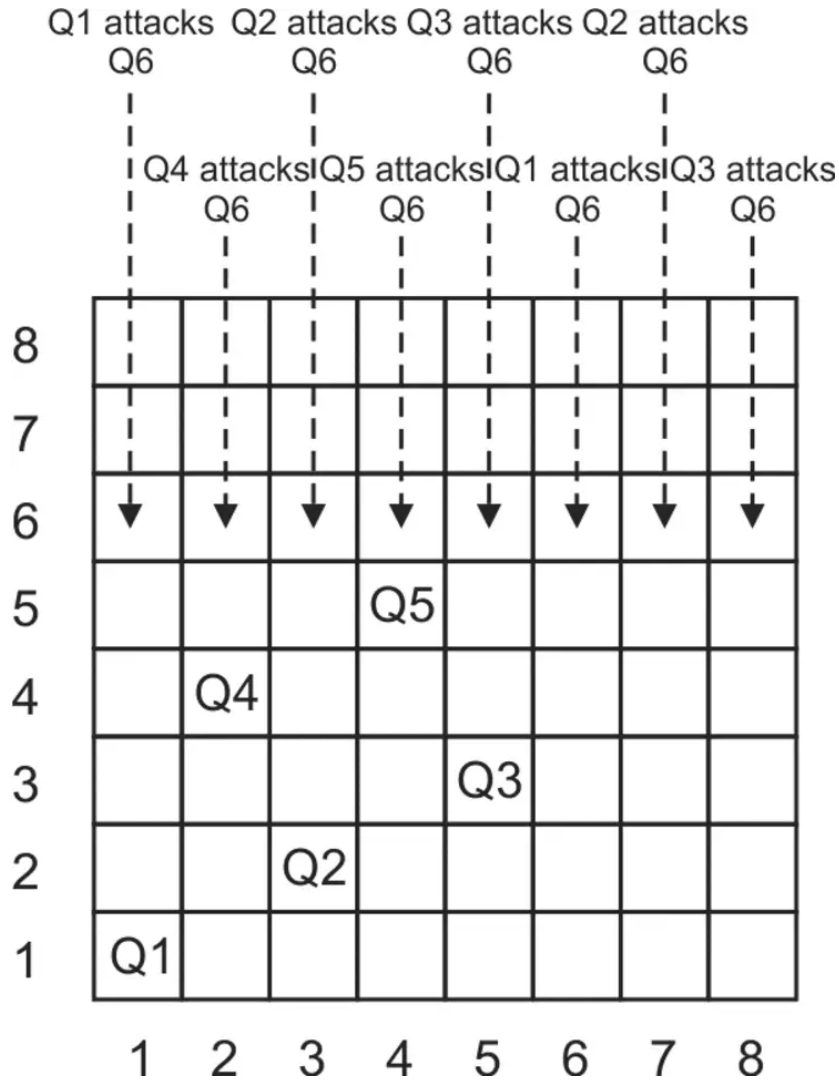


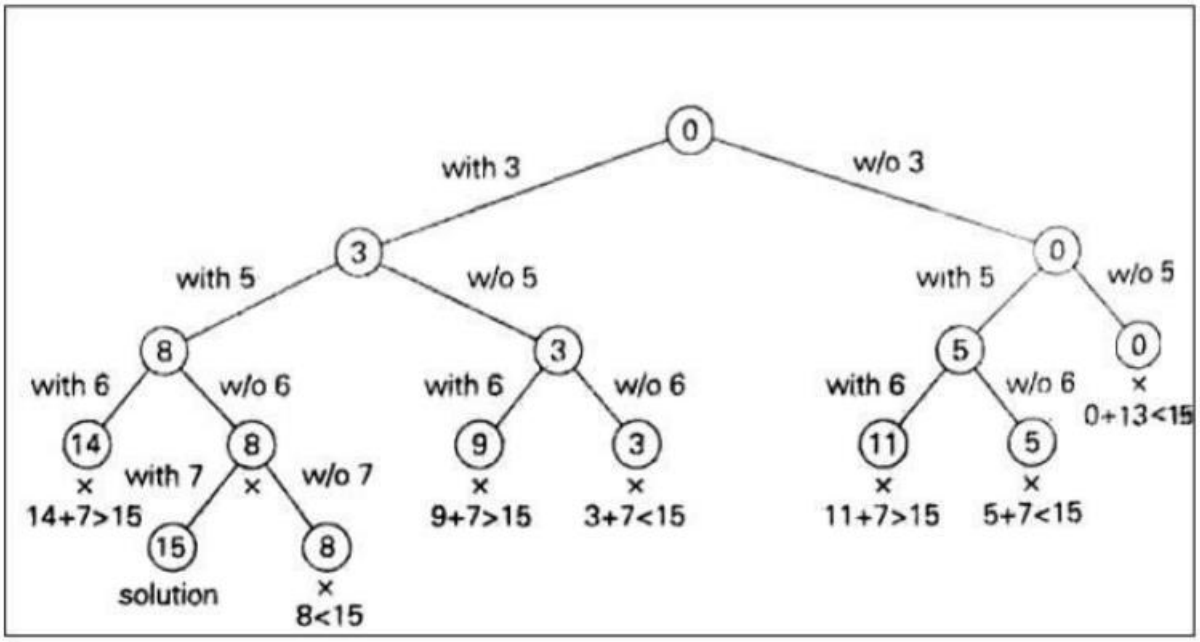

Fig: Non-feasible state of the 8-queen problem

**Algorithm**

```
N_QUEEN (k, n)
// Description : To find the solution of n x n queen problem using
backtracking
// Input :
n: Number of queen
k: Number of the queen being processed currently, initially set to 1.
```

```
// Output : n x 1 Solution tuple
```

Example: S = {3,5,6,7} and d = 15, Find the sum of subsets by using backtracking




**Solution {3,5,7}**

**CONCLUSION:** Hence we have studied design of 8-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final 8-queen's matrix

**FAQ**

**Q. What is the most efficient approach to solve the N queens problem?**
The backtracking approach is the most efficient approach since it takes O(N!) time.

**Q. How does the backtrack function work?**
The backtrack function explores all paths possible by placing the queens on the rows and the columns. It keeps a visiting set to keep track of the rows and columns which have been already used.