ROHAN DAYAL
19C0060

# DAA ~~ASSIGN~~ EXPERIMENT 1

Page :
Date : / /

=> AIM :-

Write a program non-recursive & recursive program to calculate Fibonacci numbers & analyze their time & space complexity.

=> Outcome :-

Apply Preprocessing techniques Analyze performance of an algorithm.

=> System Requirement :-

- 64 bit Open Source Linux or its derivatives
- C++, Java, Python

=> THEORY :-

=> Time Complexity :-

i) The time complexity of an algorithm is the amount of computer time required by an algorithm to run to completion.

ii) It is difficult to compute the time complexity in term of physically clocked time.

iii) For instance in multiuser system, executing time depends on many factors -

- System load
- number of other programs running
- speed of underlying hardware.

iv) The time complexity is therefore given in terms of frequency count.

Page :
Date : / /

- Space Complexity -

>) The space complexity can be defined as amount of memory required by an algorithm to run.

>>) To compute the space complexity we use two factors : Constant & instance characteristics.

>>>) $S(p) = C + Sp$

$C \rightarrow$ constance

$S(p) \rightarrow$ space complexity

$Sp \rightarrow$ space dependent upon instance characteristics.

- Recursive Algorithm -

>) A method of solving a computational problems where the solution depends on solutions to smaller instance of the same problem.

>>) Recursion consist of 2 sub parts - recursive part in which a function is called recursively & end part in which an end statement is mentioned.

- Non-Recursive Algorithm -

>) Iterative algorithm is an algorithm which has at least one iterative component or loop. Hence the part of algorithmic statements will be executed for n number of times.

>>) Even small amount of time spent on execution of loop will directly affect the efficiency of overall algorithm. This situation can be worst if nested loops (a loop within another loop) is present in algorithm.

Page :
Date : / /

=> Conclusion :-

Hence a ~~program~~ non-recursive & recursive programs to calculate Fibonacci numbers have been implemented & their space & time complexities ~~are~~ have been analyzed.

ROHAN DAYAL
19C0060

Page:
Date: / /

# DAA EXPERIMENT 2

=> Aim :-

Write a program to implement Huffman Encoding using a greedy strategy.

=> Outcome :-

Implement an algorithm that follows a greedy algorithm design strategy.

=> System Requirement :-

- 64 bit Open Source Linux or its derivatives
- C++, Java, Python.

=> Theory :-

- Greedy Strategy :-

i) In an algorithmic strategy like Greedy, the decision of solution is taken based on the information available.

ii) The greedy method is a straightforward method. This method is popular for obtaining the optimized solutions.

iii) In Greedy technique, the solution is constructed through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.

Page :
Date : / /

iv) At each step the choice made should be -
- Feasible - It should satisfy the problem's constraints.
- locally optimal - Among all feasible solutions, the best choice is to be made.
- Irrevocable - Once the particular choice is made then it should not get changed on subsequent steps.

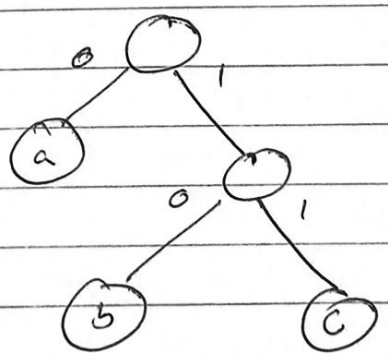
v) In short, while making a choice there should be a Greed for an optimal solution.

- Huffman Encoding :-

i) ~~Huffa~~ Huffman encoding is a lossless data compression algorithm.

ii) The idea is to assign variable-length codes to the input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.

iii) The most frequent character gets the smallest code & the least frequent characters~~es~~ gets the largest code.




a → 0
b → 10
c → 11

For a = 50 b = 30 c = 20

Page :
Date : / /

=> CONCLUSION :-

Hence a program to implement Huffman encoding using a greedy strategy has been implemented & analyzed.

Name.
Rohan Dayal
1910060

# DAA EXPERIMENT 3

Page:
Date:

=> AIM:

Write a program to solve a fractional Knapsack problem using a greedy method.

=> OUTCOME:

Implement an algorithm that follows a greedy algorithm design strategy.

=> SYSTEM REQUIREMENT:

- 64 bit Open Source Linux or Derivatives
- C++, Java, Python.

=> THEORY:

- Greedy Strategy:

i) In an algorithmic strategy like Greedy, the decision of solution is taken based on the information available.

ii) The greedy method is a straightforward method. It is used generally for obtaining the optimized solutions.

iii) In Greedy Technique, the solution is constructed through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.

Page :
Date : / /

iv) At each step the choice ~~we~~ made ~~should~~ should be -
- Feasible - It should satisfy the problem's constraint.
- Locally Optimal - Among all feasible solution, the best choice is to be made.
- Irrevocable - Once the particular choice is made then it should not get changed on subsequent steps.

v) In short, while making a choice there should be a Greed for an optimal solution.

- Fractional Knapsack Problem :-

i) Suppose there are n ~~to~~ objects, each object i has some positive weight $w_i$ & some profit $p_i$. This knapsack will carry at most weight W.

ii) While solving above mentioned knapsack problem, we have the capacity constraint. ~~While we~~

iii) While we try to solve this problem ~~that~~ using Greedy approach our goal is,
i) Choose only those objects that give maximum profit
ii) The total weight of those object should be $\leq W$.

iv) In other words,

$$\text{Maximized} \sum_{n}^{i} p_i x_i \text{ subject to } \sum_{n}^{i} w_i x_i \leq W$$

where knapsack can carry the fraction $x_i$ of an object i such that $0 \leq x_i \leq 1$ & $1 \leq i \leq n$.

ROHAN DAYAL

Page :
Date : / /

=> CONCLUSION:-

Hence, a program to solve a fractional Knapsack problem using a greedy method has been implemented and analyzed.

ROHAN DAYAL
ROHAN DAYAL
19CO060

Page :
Date : / /

# DAA EXPERIMENT 4

=> AIM :-

Write a program to solve a 0-1 knapsack problem using dynamic programming or branch & bound strategy.

=> OUTCOME :-

Implement an algorithm that follows the dynamic programming & branch & bound ~~strategy~~ algorithm design strategies.

=> SYSTEM REQUIREMENT :-

- 64 bit Open Source linux or its derivatives
- C++, Java, & Python.

=> THEORY :-

- Dynamic Programming :-

:) Dynamic Programming is a technique for solving problems with ~~of~~ overlapping subproblems.

:) In this method, each subproblem is solved only once. The result of each subproblem is recorded in a table from which we can obtain a solution to original problems.

:) For each given problem, we may get any number of solution we seek for optimum solution. And such solution becomes the solution to the given problem.

ROHAN DAHAL

Page:
Date: / /

## Branch & Bound Strategy:-

i) Branch and bound is a general algorithm method for finding optimal solutions of various optimization problem.

ii) Branch & bounding method is a general optimization technique that applies where the greedy method & dynamic programming fail. However it is much slower.

iii) It often leads to exponential time complexities in worst case. On the other hand, if applied carefully, it can lead to algorithms that run reasonably fast on average.

## 0/1 Knapsack Problem:-

i) If we are given n objects & a knapsack in which the object i that has weight $w_i$ is to be placed. The knapsack has a capacity of W. The the profit can be earned is $p_i x_i$. The objective is to get maximum profit.

ii) maximized $\sum_{1}^{n} p_i x_i$ subject to constraint $\sum_{1}^{n} w_i x_i \leq W$
where $1 \leq i \leq n$ & $x_i = 0$ or $1$.

iii) To solve this, we will write the recurrence relation:

$$table[i,j] = \text{maximum}\{table[i-1,j],\ v_i + table[i-1, j-w_i]\} \quad \text{if } j \geq w_i$$

or

$$table[i-1,j] \quad \text{if } j < w_i$$

Page :
Date : / /

=> CONCLUSION :-

Hence a program to solve 0-1 knapsack problem using dynamic programming or branch and bound strategy has been implemented and analyzed.

ROHAN DAYAL
19CO060

# DAA EXPERIMENT 5

Page:
Date: / /

=> Aim:

Design n-queens matrix having first queen placed. Use backtracking to place remaining queens to generate the final n-queen's matrix.

=> Outcome:

Implement an algorithm that follows ~~one of the~~ Backtracking algorithm design strategy.

=> System Requirement:

- 64 bit Open Source Linux or its derivatives
- C++, Java, Python.

=> THEORY:

- Backtracking:

i) In the backtracking method -
   a) The desired solution is expressible as an n tuple $(x_1 \dots x_n)$ where $x_i$ is chosen from some finite set $S_i$.
   b) The solution maximizes or minimizes or satisfies a criterion function $C(x_1 \dots x_n)$

ii) The problem can be categorized into three categories -
   a) Decision problem.
   b) Optimization problem
   c) Enumeration Problem.

iii) The basic idea of backtracking is to build up a vector, one component at a time & to test whether the vector being formed has any chance of success.

Page:
Date: / /

iv) The major advantage of backtracking is that we can realize the fact that the partial vector gaurantees does not lead to an optimal solution. In such a situation, that vector can be ignored.

v) Backtracking determines the solution by systematically searching the solution space for the given problem.

- **n-Queens Problem -**

i) Consider a chess board of $n \times n$. The problem is to place n Queens on this board such that, no two queens can attack each other. That means, no two queens can be placed on the same row, column or diagonal.

ii) This n Queen's problem is solved by applying implicit & explicit constraints.

iii) The explicit constraints shows that the solution space $S_i$ must be $\{1, 2 \dots n\}$ with $1 \le i \le n$. Hence solution space consists of $n^n$ n-tuples.

iv) The implicit constraints are -
a) No two $x_i$ will be same. That means ~~queen bee~~ all queens must be in different column.
b) No two queens can be on the same row, column or diagonal.

v) Hence the solution of an 8-queen problem can be represented as $\{4, 6, 8, 2, 7, 1, 3, 5\}$

Page :
Date : / /

=> CONCLUSION :

Hence a a backtracking algorithm to solve an n-queens problem given the location of the first queen has been implemented & analyzed.