**K. J. Somaiya College of Engineering, Mumbai-77**

(Autonomous College Affiliated to University of Mumbai)

https://ssl.gstatic.com/ui/v1/icons/mail/images/cleardot.gif

**Batch: B1 Roll No.: 1711072**

**Experiment No. 7**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation Matrix Chain Multiplication of Dynamic Programming** |

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. [**http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf**](http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf)
4. [**http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/**](http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/)
5. [**http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf**](http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf)
6. [**https://class.coursera.org/algo2-2012-001/lecture/181**](https://class.coursera.org/algo2-2012-001/lecture/181)
7. [**http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-problem-using-Dynamic-programming**](http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-problem-using-Dynamic-programming)
8. [**www.cse.hcmut.edu.vn/~dtanh/download/Appendix\_B\_2.ppt**](http://www.cse.hcmut.edu.vn/~dtanh/download/Appendix_B_2.ppt)
9. **www.ms.unimelb.edu.au/~s620261/powerpoint/chapter9\_4.ppt‎**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

Dynamic Programming (DP) is used heavily in optimization problems (finding the maximum and the minimum of something). Applications range from financial models and operation research to biology and basic algorithm research. So the good news is that understanding DP is profitable. However, the bad news is that DP is not an algorithm or a data structure that you can memorize. It is a powerful algorithmic design technique.

**New Concepts to be learned:**

Application of algorithmic design strategy to any problem, dynamic Programming method of problem solving Vs other methods of problem solving, optimality of the solution, Optimal Binary Search Tree Problems and their applications

**Theory:**

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C, and D, we would have: A(BCD), (AB)(CD), etc. However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of dimension p[i-1] x p[i]. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain.

**Algorithm:**

The idea is to break the problem into a set of related subproblems which group the given matrix in such a way that yields the lowest total cost. Below is a recursive algorithm to find minimum cost:

1. Take the sequence of matrices and separate into two subsequences.
2. Find the minimum cost of multiplying out each susequence.
3. Add these costs together and then add the cost of muliplying the two resultant matrices.
4. Perform the same for each possible position at which the sequence of matrices can be split, and take the minimum over all of them.

**Example :**

Four matrices A,B,C and D can be multiplied as:

((AB)C)D = ((A(BC))D = (AB)(CD) = A((BC)D) = A(B(CD))

A: 10 x 30, B: 30 x 5, C: 5 x 60

(AB)C needs (10x30x5) + (10x5x60)=4500 operations.

A(BC) needs (30x5x60) + (10x3x60)=27000 operations.

First method is more efficient.

**Program:**

from prettytable import PrettyTable

pr1,pr2=PrettyTable(), PrettyTable()

p,m,b,j,minimum,q,n, name=[5,4,6,2,7], [[0 for i in range(5)] for j in range (5)], [[0 for i in range(5)] for j in range (5)],0,0,0,5, "A"

for d in range(1,n-1):

for i in range(1,n-d):

j=i+d

minimum=float('inf')

for k in range(i,j):

q=m[i][k]+m[k+1][j]+(p[i-1]\*p[k]\*p[j])

if q<minimum:

minimum,b[i][j]=q,k

m[i][j]=minimum

print("Minimum Cost is: ",m[1][n-1])

print("Cost Matrix: ")

pr1.field\_names=[i for i in range(5)]

for row in m:

pr1.add\_row(row)

print (pr1.get\_string(header=True, border=True))

print("Paranthesization Matrix: ")

pr2.field\_names=[i for i in range(5)]

for row in b:

pr2.add\_row(row)

print (pr2.get\_string(header=True, border=True))

for i in range(len(b[1])-1,2,-1):

print('Split occurs at matrix number: ', b[1][i])

def Parenthesize(i,j,n,b):

global name

if i==j:

print(chr(ord(name)),end='')

name=chr(ord(name)+1)

return

print("(",end='')

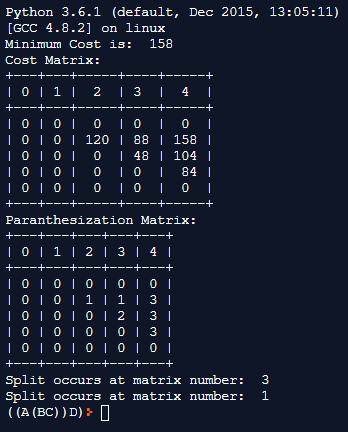
Parenthesize(i, b[i][j], n,b)

Parenthesize(b[i][j] + 1, j,n, b)

print(")",end='')

Parenthesize(1,n-1,n,b)

**Output Screen:**



**Analysis of algorithm:**

Since we are using three nested for loops, then for large number of inputs, the time complexity of program will be O(n3). The auxillary space complexity is O(n2).

**CONCLUSION: The program ran successfully for all test cases and the parenthesization was done accordingly.**

**Topic: Backtracking**

**Theory:** In many applications of the backtrack method, the desired solution is expressible as an n-tuple *(x1,...,Xn),* where the x*i* are chosen from some finite set Si. Often the problem to be solved calls for finding one vector that maximizes (or minimizes or satisfies) a *criterion function P(x1,…..* . , *xn). Sometime*s it seeks all vectors that satisfy *P.* For example, sorting the array of integers in. *a[1* : n] is a problem whose solution is expressible by an *n- tuple, w*here x*i* is the index in *a* of the ith smallest element. The criterion function P is the inequality *a[xi]* ≤ *a[xi+1]* for 1 ≤ i < *n.* The set *Si* is finite and includes the integers 1 through *n.* Though sorting is not usually one of the problems solved by backtracking, it is one example of a familiar problem whose solution can be formulated as an n-tuple.

**Control abstraction**:

void Backtrack( int k )

// This is a schema that describes the backtracking process //using recursion. On entering, the first k-1 values x[1], x[2], //…., x[k-1] of the solution vector x[1:n] have been //assigned. x[] and n are global.

{

for (each x[k] such that x[k] Є T(x[1], …, x[k-1])

{

if (Bk (x[1], x[2], …, x[k]))

{

if (x[1], x[2], …, x[k] is a path to an answer node)

output x[1:k];

if (k < n) Backtrack(k+1);

}

}

}

**Batch: B1 Roll No.: 1711072**

**Experiment No. 8**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of Backtracking Algorithm** |

**Objective:** To learn the Backtracking strategy of problem solving for 8-Queens problem

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.html**
4. [**http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf**](http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf)
5. [**http://www.slideshare.net/Tech\_MX/8-queens-problem-using-back-tracking**](http://www.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking)
6. [**http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html**](http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html)
7. [**http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**](http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/)
8. **http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

The **N-Queens puzzle** is the problem of placing N queens on an N×N chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

**New Concepts to be learned:**

Application of algorithmic design strategy to any problem, Backtracking method of problem solving Vs other methods of problem solving,8- Queens problem and its applications.

**Algorithm N Queens Problem:-**

void NQueens(int k, int n)

// Using backtracking, this procedure prints all possible placements of n queens on an n X n chessboard so that they are nonattacking.

{ for (int i=1; i<=n; i++)

{

if (Place(k, i))

{

x[k] = i;

if (k==n)

for (int j=1;j<=n;j++) Print x[j] ;

else NQueens(k+1, n);

}

}

}

Boolean Place(int k, int i)

// Returns true if a queen can be placed in kth row and ith column. Otherwise it returns false.

// x[] is a global array whose first (k-1) values have been set. abs(r) returns absolute value of r.

{

for (int j=1; j < k; j++)

if ((x[j] == i) // Two in the same column

|| (abs(x[j]-i) == abs(j-k))) // or in the same diagonal

return(false);

return(true);

}

**Solution Using Backtracking Approach:**

global N,k

k=0

N=int(input("Enter board size: "))

def printSolution(board):

  for i in range(N):

    for j in range(N):

      print ("Q" if board[i][j]==1 else "-", end=' ')

    print()

def isSafe(board, row, col):

  for i in range(col): #checking if pos already occupied

    if board[row][i] == 1:

      return False

#upper diagonal left side

  for i,j in zip(range(row,-1,-1), range(col,-1,-1)):

    if board[i][j] == 1:

      return False

  #lower diagonal left side

  for i,j in zip(range(row,N,1), range(col,-1,-1)):

    if board[i][j] == 1:

      return False

  return True

def PlaceQueens(board, col):

global k

if col == N:

print('Solution ',k+1,':\n')

k=k+1

printSolution(board)

return True

for i in range(N):

res=False

if isSafe(board, i, col):

board[i][col] = 1

res = PlaceQueens(board, col + 1) or res

board[i][col] = 0

return res

def NQueens():

board = [[0 for i in range(N)] for i in range(N)]

if PlaceQueens(board, 0) == False and k==0:

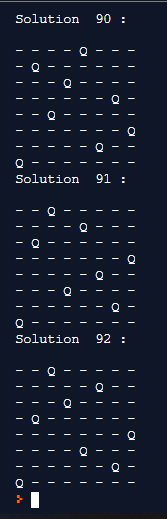
print("Solution does not exist")

return False

return True

NQueens()

**Screenshot:**



**For chess board size input of 8, we got 92 solutions.**

**Analysis of Backtracking solution for 8-Queens Problem:**

Instead of generating one optimal solution, we are generating all possible and feasible solutions for a given chess board size. The time complexity of the program is:

**T(n) = O(nn)**

Where n is the size of the chessboard. The complexity is approximately n! for higher values of n.

**CONCLUSION: The program for determining solution to N-Queen problem was solved using backtracking and all feasible solutions were found.**

**Batch: B1 Roll No.: 1711072**

**Experiment No. 9**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of Backtracking Algorithm** |

**Objective:** To learn the Backtracking strategy of problem solving for Graph Colouring problem

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.html**
4. [**http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf**](http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf)
5. [**http://www.slideshare.net/Tech\_MX/8-queens-problem-using-back-tracking**](http://www.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking)
6. [**http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html**](http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html)
7. [**http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**](http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/)
8. **http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

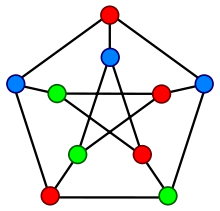
Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with same color. Here coloring of a graph means assignment of colors to all vertices.

**Input:**

1) A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph.

**Output:**

An array color[V] that should have numbers from 1 to m. color[i] should represent the color assigned to the ith vertex. The code should also return false if the graph cannot be colored with m colors.

Following is an example graph can be colored with 3 colors.  
[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/graph_col.png)

**New Concepts to be learned:**

Application of algorithmic design strategy to any problem, Backtracking method of problem solving Vs other methods of problem solving problem graph colouring and its applications.

**Algorithm Graph colouring Problem:-**

****

**Code:**

states={0:"Maharashtra", 1:"Goa", 2:"Karnataka", 3:"Gujarat", 4:"Chattisgarh", 5:"Odisha", 6:"Bengal"}

graph={0:[1,2,3,4], 1:[0,2], 2:[0,1], 3:[0], 4:[0,5], 5:[4,6], 6:[5]}

colors=["Red", "Green", "Blue"]

col\_graph,j={},1

def check(state, colour):

  global graph

  global col\_graph

  for i in graph[state]:

    if i in col\_graph and col\_graph[i]==colour:

      return False

  return True

def assign(state, colour):

  global col\_graph

  col\_graph[state]=colour

def solve(vertex):

i=0

global j

if vertex==7:

print('Solution ',j,':\n')

for key, value in col\_graph.items():

print(states[key] + " : " + colors[value])

if(states[key]=='Bengal'):

print('\n')

j=j+1

return False

for i in range(len(colors)):

if check(vertex,i)==True:

assign(vertex,i)

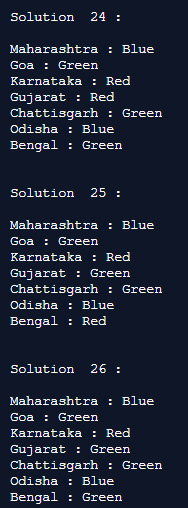
if solve(vertex+1)==True:

return True

assign(vertex,0)

solve(0)

**Screenshots:**



**Analysis of Backtracking solution for Graph Colouring Problem:**

In the main logic of the program, we are comparing every vertex with its adjacent edge to check for colours. So, every such comparison yields a time complexity of

**T(n) = O(V2+E)**

where V is number of vertices and E is number of edges. We are displaying all possible solutions instead of showing one optimal solution with minimum colours. If optimal solution was to be shown, the time complexity would have been around **O(V2).**

**CONCLUSION: The graph colouring problem was solved using backtracking and all possible solutions for a given graph were generated.**

**Batch: Roll No.:**

**Experiment No.\_\_\_\_\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation Of String Matching Algorithm** |

**Objective:** To compute longest common subsequence for the given two strings.

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. [**http://en.wikipedia.org/wiki/Longest\_common\_subsequence\_problem**](http://en.wikipedia.org/wiki/Longest_common_subsequence_problem)
4. [**http://www.columbia.edu/~cs2035/courses/csor4231.F11/lcs.pdf**](http://www.columbia.edu/~cs2035/courses/csor4231.F11/lcs.pdf)
5. **http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node4.html**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

Given 2 sequences, *X* = *x*1 *, ..., xm*  and *Y* = *y*1 *, ... , yn* , find a subsequence common to both whose length is longest. A subsequence doesn’t have to be consecutive, but it has to be in order.

**New Concepts to be learned:**

String matching algorithm, Dynamic programming approach for LCS, Applications of LCS

**Recursive Formulation:**

Define *c*[*i, j* ] = length of LCS of *Xi* and *Y j* .

Final answer will be computed with *c*[*m, n*].

c[i, j]= 0 if i=0 or j=0.

c[i, j]= c[i − 1, j − 1] + 1 if i,j>0 and xi=yj

c[i, j]= max(c[i − 1, j ], c[i, j − 1]) if i, j > 0 and xi <> y j

**Algorithm: Longest Common Subsequence**

**Compute length of optimal solution-**

**LCS-LENGTH** *( X , Y, m, n)*

**for** *i* ← 1 **to** *m*

**do** *c*[*i,* 0] ← 0

**for** *j* ← 0 **to** *n*

**do** *c*[0*, j* ] ← 0

**for** *i* ← 1 **to** *m*

**do for** *j* ← 1 **to** *n*

**do if** *xi* = *y j*

**then** *c*[*i, j* ] ← *c*[*i* − 1*, j* − 1] + 1

*b*[*i, j* ] ← “≈”

**else if** *c*[*i* − 1*, j* ] ≥ *c*[*i, j* − 1]

**then** *c*[*i, j* ] ← *c*[*i* − 1*, j* ]

*b*[*i, j* ] ← “↑”

**else** *c*[*i, j* ] ← *c*[*i, j* − 1]

*b*[*i, j* ] ← “←”

**return** *c* and *b*

**Print the solution-**

**PRINT-LCS*(b, X , i, j )***

**if** *i* = 0 or *j* = 0

**then return**

**if** *b*[*i, j* ] = “≈”

**then** PRINT-LCS*(b, X , i* − 1*, j* − 1*)*

print *xi*

**elseif** *b*[*i, j* ] = “↑”

**then** PRINT-LCS*(b, X , i* − 1*, j )*

**else** PRINT-LCS*(b, X , i, j* − 1*)*

•Initial call is PRINT-LCS*(b, X , m, n)*.

•*b*[*i, j* ] points to table entry whose subproblem we used in solving LCS of *Xi*

and *Y j* .

• When *b*[*i, j* ] = ≈, we have extended LCS by one character. So longest com- mon subsequence = entries with ≈ in them.

**Example: LCS computation**

**LCS computation**

**CONCLUSION:**