**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. 6**

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

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

|  |
| --- |
| **Title: Implementation of Knapsack Problem using Dynamic Programming** |

**Objective** To learn the Dynamic Programming using Knapsack Problemalgorithm

**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. **en.wikipedia.org/wiki/Knapsack\_problem**
4. **www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf**
5. **cse.unl.edu/~ylu/raik283/notes/0-1-knapsack.ppt**
6. **www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf**
7. **cse.unl.edu/~ylu/raik283/notes/0-1-knapsack.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.

**Principle of Optimality:** It states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

The solution to the knapsack problem can be viewed as the result of a sequence of decisions. We have to decide the values of xi, 1<= i<+n. First we make a decision on xi , then on x2 . then on x3 ,etc. An optimal sequence of decisions maximizes the objective function ∑pixi.

**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,

**Algorithm**

// Input:

// Values (stored in array v)

// Weights (stored in array w)

// Number of distinct items (n)

// Knapsack capacity (W)

for j from 0 to W do

m[0, j] := 0

end for

for i from 1 to n do

for j from 0 to W do

if w[i] <= j then

m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])

else

m[i, j] := m[i-1, j]

end if

end for

end for

**Implementation:**

pw,cap,n=[[1,2],[2,3],[5,4],[6,5]],8,4

table=[[0 for x in range(cap+1)] for x in range(n+1)]

for i in range(1,n+1):

for j in range(cap+1):

if pw[i-1][1]<=j:

table[i][j]=max(table[i-1][j], table[i-1][j-pw[i-1][1]]+pw[i-1][0])

else:

table[i][j]=table[i-1][j]

for i in table:

print(\*i, sep=' ')

i,j,sack=n,cap,[]

while i>0 and j>0:

if table[i][j]!=table[i-1][j]:

sack.append(i)

i,j=i-1,j-pw[i-1][1]

else:

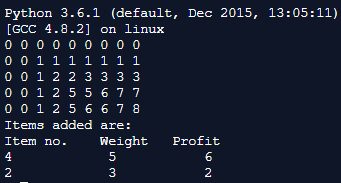
i=i-1

print('Items added are: \nItem no.\tWeight\t Profit')

for i in sack:

print(i,'\t\t\t',pw[i-1][1],'\t\t\t',pw[i-1][0])

**Output Screen:**



**Analysis of 0/1 Knapsack algorithm:**

The 0/1 Knapsack problem has two parts: one for finding the number of items that can be inserted and its profit, and second for finding which items are added to the knapsack. If the capacity of the knapsack is W and number of items considered is N, then the main algorithm consists of two for loops. The time complexity of the algorithm is given by:

**T(n)=O(NW)**

**CONCLUSION: The 0/1 Knapsack problem was solved using dynamic programming technique and the program ran successfully for all test cases.**

**Batch: Roll No.:**

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

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

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

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

**Objective:**

**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:**

**Algorithm:**

**Example :**

**Solution for the example :**

**Analysis of algorithm:**

**CONCLUSION:**

**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: Roll No.:**

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

**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);

}

**Example 8-Queens Problem:**

**Solution Using Backtracking Approach:**

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

**CONCLUSION:**

**Batch: Roll No.:**

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

**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:**

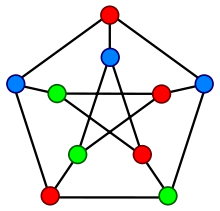
Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with same color. Here coloring of a graph means assignment of colors to all vertices.

**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:-**

****

**Example Graph Colouring Problem:**

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

**CONCLUSION:**

**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:**