UNIT – IV
Dynamic Programming

Objective:

To get acquainted with algorithm design technique dynamic programming.

SYLLABUS:

Dynamic Programming: General method, applications-Matrix chain multiplication, Optimal binary search trees, 0/1 knapsack problem, Travelling sales person problem.

Outcomes:

Students will be able to:

- apply Dynamic Programming to solve the real time problems.
- find optimal sequence of Matrix multiplication
- determine optimal solution for 0/1 knapsack problem
- find the optimal tour for travelling salesman problem.

Learning Material

Introduction to Dynamic Programming

- A quicker solution to a problem can be achieved by breaking up the problem into a set of related sub problems.
- By solving sub problems once and reusing the solutions, the required run-time can drastically reduced.
- This concept is known as **dynamic programming.**
- Here, the word **Programming** stands for **planning** and it does not mean by computer programming.
- In this method, each sub problem is solved only once.

- The result of each sub problem is recorded in a table from which we can obtain a solution to the original problem.

General Method:

- Dynamic programming is typically applied to optimization problems. For each given problem, we may get any number of solutions.
- But we look for optimum solution.

Steps for Dynamic programming:

- Dynamic programming design involves 4 major steps:

  Step-1: Characterize the structure of optimal solution.

  Step-2: Recursively define the value of an optimal solution.

  Step-3: By using bottom-up technique, compute the value of optimal solution.

  Step-4: Compute an optimal solution from computed information.

Matrix chain multiplication

- Matrices play an active role in scientific calculations.
- Consider there are three matrices A, B and C.
- Chain matrix multiplication can be A (BC) or (AB)C.
- The chain matrix multiplication problem can be stated formally as follows:

Given a set of matrices {A_1, A2, A3,....A_n}, where A_i has dimensions P[i-1][i], Find the optimal order of matrices such that the cost of multiplication is optimal.

The possible orderings of the matrices are identified as follows:

$A_1(A_2,A_3,\ldots A_n)$ or

$(A_1, A2)(A_3,\ldots,A_n)$ or

$(A_1,A_2,A_3)(A_4\ldots. A_n)$ or

.

.

$(A_1, A2,A_3, A4\ldots., A_{n-1})(A_n)$

- Using the Dynamic programming concept, one can split the given matrices into two parts using the variable k as :

  $(A_1, A2,A_3\ldots.,A_k)(A_{k+1}\ldots A_n)$

The first partition can be parenthesized in k ways and the second partition in (n-k) ways.

Example:

Let us consider the following three matrices.
A: 2x3, B: 3x4, C: 4x3. What are the possible orderings? What is the optimal order?

Solution:

Three matrices are given. So, two possible orderings are possible.

They are:

((AB)C) and (A(BC)).

The cost of multiplying two matrices A(i,j) and B(j,k) is i*j*k.

So,

((AB)C) = (2*3*4)+(2*4*3) = 24+24 = 48

(A(BC)) = (2*3*3)+(3*4*3) = 18+36 = 54

Hence the optimal order is (A(BC))which gives minimum cost.

Dynamic Programming approach for solving Matrix-Chain Multiplication

The recursive relation of Matrix-Chain Multiplicationis:

M[i,j] =0, if i=j

M[i,j]= $\mathbf{min}_{i\le k\le j}\{\mathbf{M[i,k] + M[k+1,j] + P[i-1]*P[k]*P[j]}\}$ **, otherwise**

If i=j then there is only one matrix. So that M[i,i]=0.

If i is not equal to j and if i<j, then the product of A[1…j] is required to split into two parts as A[i…k] and A[k+1… j], where 1<= k<=j.

Algorithm for Matrix-Chain Multiplication is:

```
Algorithm MatrixChainMultiplication(p,n)
// p is the order and n is the chain of matrices.
{
    for i=1 to n do
    {
        M[i,i]=0
    }
    for diagonal=1 to n-1
    {
        for i=1 to n-diagonal
        {
            j=i+diagonal
            M[i,j]= ∞
            fork=1 to j-1 do
            {
                If M[i,j] < M[i,k] + M[k+1,j]+p[i-1]* p[k]* p[j]  then
                {
                    M[i,j]=M[i,k]+ M[k+1,j]+p[i-1]* p[k]* p[j]

                    R[i,j]= k
                }
                Else
                {
                    M[i,j]=M[i,j]
```

```
                        R[i,j] = k
                    }
                }
            }

        }
        return c[1,n]
} // end of algorithm
```

Example:

Consider four matrices and its orders. Perform matrix chain multiplication using dynamic programming approach.

A_{4x5} B_{5x3} C_{3x2} D_{2x7}

Solution:

As per the algorithm, entries of matrix M are initialized as follows.

M[1,1]=0

M[2,2]=0

M[3,3]=0

M[4,4]=0

The initial table is:

	1	2	3	4
1	0			
2		0		
3			0	
4				0

Now, compute the first super diagonal as follows:

M[i,j]=M[i,k]+ M[k+1,j]+p[i-1]* p[k]* p[j]

M[1,2] = M[1,1] + M[2,2]+ $p_0 * p_1 * p_2$
= 0 + 0+ 4*5*3
= 60

M[2,3] = M[2,2] + M[3,3]+ $p_1 * p_2 * p_3$
= 0 + 0+ 5*3*2
= 30

M[3,4] = M[3,3] + M[4,4]+ $p_2 * p_3 * p_4$
= 0 + 0+ 3*2*7
= 42

The resultant table after first diagonal is:

	1	2	3	4
1	0	60		
2		0	30	
3			0	42
4				0

Next, the second super diagonal needs to be computed.

This implies that M[1…3] needs to be computed. Two splits are possible, with k=1 and k=2.

The resulting computation is as follows.

M[i,j] = M[i,k]+ M[k+1,j] + p[i-1] * p[k]* p[j]

M[1,3] splits at k=1 and k=2

First, k=1

M[1,3]	=	M[1,1] + M[2,3]+ p_0* p_1 * p_3
	=	0 + 30+ 4*5*2
	=	70 (when k=1)

Second, k=2

M[1,3]	=	M[1,2] + M[3,3]+ p_0* p_2 * p_3
	=	60 + 0+ 4*3*2
	=	84 (when k=2)

The minimum is 70 when k=1. Therefore, this must be noted in another table R. Thus table R records k that gives the minimum cost.

This process is repeated for other possibilities.

M[2,4]	=	M[2,2] + M[3,4] + p1 * p2 * p4
	=	0+42+5*3*7
	=	147 (when k=2)
M[2,4]	=	M[2,3]+M[4,4]+p1 *p3*p4
	=	30+0+5*2*7
	=	100 (when k=2)

The minimum is 100 when k=3.

The resultant matrix now appears as shown in table.

	1	2	3	4
1	0	60	70	
2		0	30	100
3			0	42
4				0

Now, calculate M[1,4].

There are three possible splits at k.

M[1,4]	=	M[1,1]+M[2,4]+p0 * p1 * p4
	=	0+100+4*5*7
	=	240 (when k=1)
M[1,4]	=	M[1,2]+M[3,4]+p0 * p2 * p4
	=	60+42+4*3*7
	=	186 (when k=2)
M[1,4]	=	M[1,3]+M[4,4]+p0 * p3 * p4
	=	70+0+4*2*7
	=	126 (when k=3)

Here, the minimum cost is 126. And this happens when k=3.

The resultant matrix is given in the below table.

	1	2	3	4
1	0	60	70	126
2		0	30	100
3			0	42
4				0

	1	2	3	4
1	0	60 k=1	70 k=1	126 k=3
2		0	30 k=2	100 k=3
3			0	42 k=3
4				0

It can be observed that R(1,4) is 3. Hence, Split the multiplication at k=3.

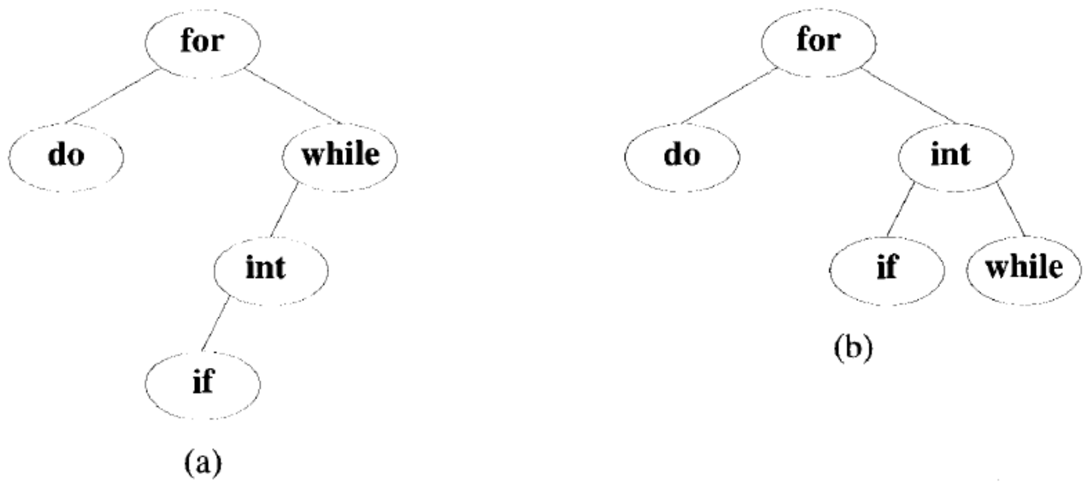
This gives the order ((ABC)D).

To split ABC, check R(1,3). And R(1,3) is 1.

Therefore, the final chain of matrix represented as (A(BC)D).

Optimal Binary Search Trees

- We may expect different binary search trees for the same identifiers to have different performance characteristics.



Two possible binary search trees

- Consider above binary search trees. In the worst case, BST (a) requires four comparisons to find an identifier, where as the tree of (b) requires only three.

- On the average the two trees need12/5and 11/5comparisons respectively.

- Let us assumethat the given setof identifiers is $\{a_1,a_2,\ldots a_n.\}$ with $a_i<a_2<\ldots.<a_n$. Let p(i) be the probability with which we search for a_i. Let q(i) be the probability that the identifier x being searched for is such that $a_i<x < a_{i+1}, 0 <i <n$

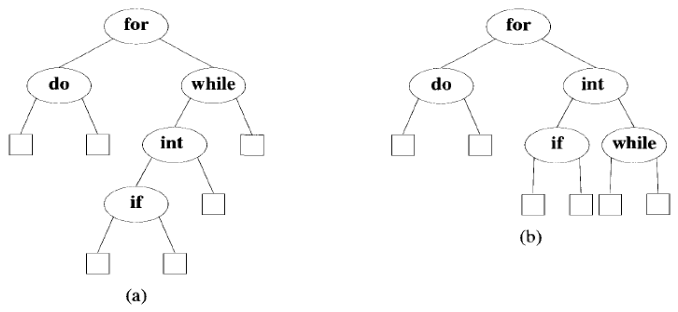
  Then, $\sum_{0\le i\le n} q(i)$ is the probability of unsuccessful search.

Then,

$$\sum_{0\le i\le n} p(i) + \sum_{0\le i\le n} q(i) = 1$$

- If a binary search tree represents n identifiers, then there will be exactly n internal nodes and n +1 (fictitious) external node.

- Every internal node represents a point where a successful search may terminate.
- Every external node represents a point where an unsuccessful search may terminate.



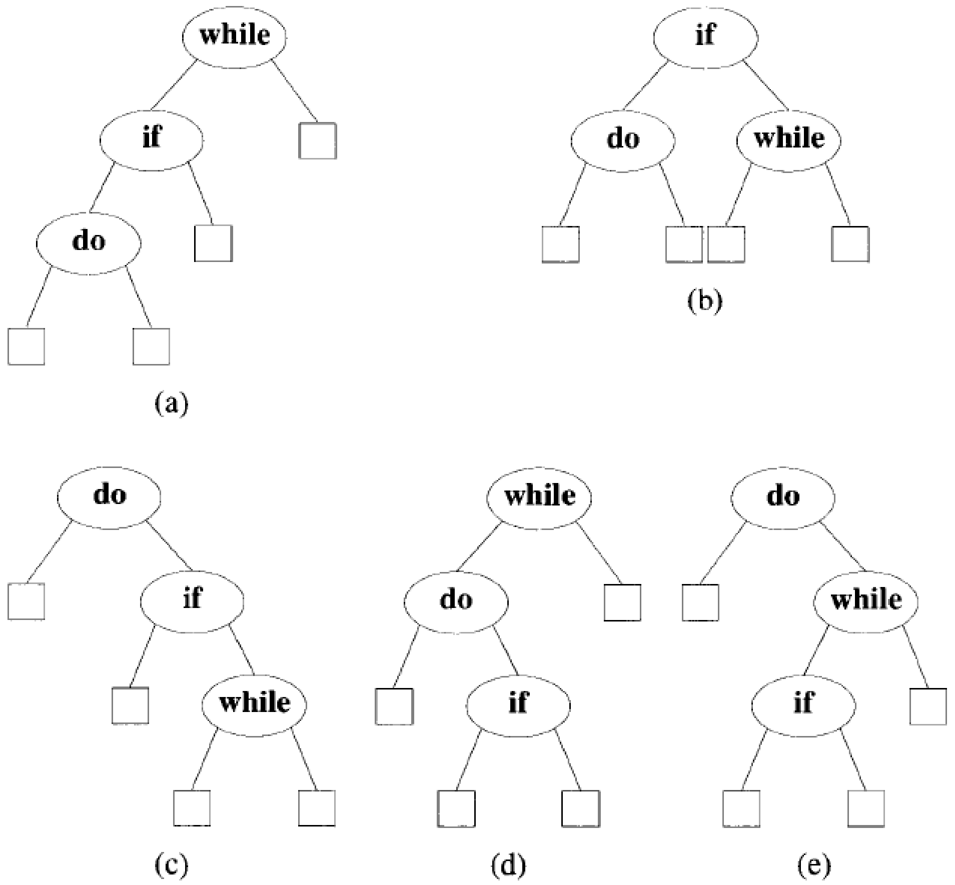
Binary search trees with external nodes added

Example The possible binary search trees for the identifier set (a_1, a_2, a_3) = (**do**, **if**, **while**) are given if Figure 5.14. With equal probabilities $p(i) = q(i) = 1/7$ for all i, we have

$$\begin{array}{lllllll} cost(\text{tree a}) & = & 15/7 & & cost(\text{tree b}) & = & 13/7 \\ cost(\text{tree c}) & = & 15/7 & & cost(\text{tree d}) & = & 15/7 \\ cost(\text{tree e}) & = & 15/7 & & & & \end{array}$$

As expected, tree b is optimal. With $p(1) = .5$, $p(2) = .1$, $p(3) = .05$, $q(0) = .15$, $q(1) = .1$, $q(2) = .05$ and $q(3) = .05$ we have

$$\begin{array}{lllllll} cost(\text{tree a}) & = & 2.65 & & cost(\text{tree b}) & = & 1.9 \\ cost(\text{tree c}) & = & 1.5 & & cost(\text{tree d}) & = & 2.05 \\ cost(\text{tree e}) & = & 1.6 & & & & \end{array}$$



Possible binary searchtrees for the identifier set {do,if,while}

Using $w(i,j)$ to represent the sum $q(i) + \sum_{l=i+1}^{j}(q(l)+p(l))$, we obtain the following as the expected cost of the search tree

$$p(k) + cost(l) + cost(r) + w(0,k-1) + w(k,n)$$

If the tree is optimal, then must be minimum. Hence, $cost(l)$ must be minimum over all binary search trees containing $a_1, a_2, \ldots, a_{k-1}$ and $E_0, E_1, \ldots, E_{k-1}$. Similarly $cost(r)$ must be minimum. If we use $c(i,j)$ to represent the cost of an optimal binary search tree t_{ij} containing $a_{i+1}, \ldots, a_j$ and $E_i, \ldots, E_j$, then for the tree to be optimal, we must have $cost(l) = c(0,k-1)$ and $cost(r) = c(k,n)$. In addition, k must be chosen such that

$$p(k) + c(0,k-1) + c(k,n) + w(0,k-1) + w(k,n)$$

is minimum. Hence, for $c(0,n)$ we obtain

$$c(0,n) = \min_{1 \le k \le n}\{c(0,k-1) + c(k,n) + p(k) + w(0,k-1) + w(k,n)\}$$

We can generalize to obtain for any $c(i,j)$

$$c(i,j) = \min_{i<k\le j}\{c(i,k-1) + c(k,j) + p(k) + w(i,k-1) + w(k,j)\}$$

<u>0/1 knapsack problem</u>

- There are two versions of the problem:
    - **0/1 knapsack problem-** Items are indivisible; you either take an item or not. Solved with dynamic programming
    - **Fractional knapsack problem -**Items are divisible: you can take any fraction of an item. Solved with a greedy algorithm.

- In 0/1 knapsack problem, our goal is to maximize the value of a knapsack that can hold at most W units worth of goods from a list of items I_0, I_1, … I_{n-1}.
- Each item has two attributes:

1) Value - let this be v_i for item I_i. (Benefit value)

2) Weight - let this be w_i for item I_i.

- Now, instead of being able to take a certain weight of an item, you can only either take the item or not take the item.
- The way to solve this problem is to cycle through all 2^n subsets of the n items and pick the subset with a legal weight that maximizes the value of the knapsack.

- But, we can find a dynamic programming algorithm that will USUALLY do better than this brute force technique.
- Our first attempt might be to characterize a sub-problem as follows:
    - Let S_k be the optimal subset of elements from $\{I_0, I_1,\ldots I_k\}$.
    - But what we find is that the optimal subset from the elements $\{I_0, I_1,\ldots I_{k+1}\}$ may not correspond to the optimal subset of elements from $\{I_0, I_1,\ldots I_k\}$ in any regular pattern.
    - Basically, the solution to the optimization problem for S_{k+1} might NOT contain the optimal solution from problem S_k.

To illustrate this, consider the following example:

Item	**Weight**	**Value**
I_0	3	10
I_1	8	4
I_2	9	9
I_3	8	11

The maximum weight the knapsack can hold is 20.

- The best set of items from $\{I_0, I_1, I_2\}$ is $\{I_0, I_1, I_2\}$ but the best set of items from $\{I_0, I_1, I_2, I_3\}$ is $\{I_0, I_2, I_3\}$.
- In this example, note that this optimal solution, $\{I_0, I_2, I_3\}$, does NOT build upon the previous optimal solution, $\{I_0, I_1, I_2\}$. (Instead it build's upon the solution, $\{I_0, I_2\}$, which is really the optimal subset of $\{I_0, I_1, I_2\}$ with weight 12 or less.)

So, now, we must rework our example. In particular, after trial and error we may come up with the following idea:

- Let B[k, w] represent the maximum total value of a subset S_k with weight w. Our goal is to find B[n, W], where n is the total number of items and W is the maximal weight the knapsack can carry.

  Using this definition, we have B[0, w] = w_0, if w >= w_0.

  = 0, otherwise

  Now, we can derive the following relationship that B[k, w] obeys:

  B[k, w] = B[k - 1,w], if $w_k > w$

  = max { B[k - 1,w], B[k - 1,w - w_k] + v_k}

  here is what this is saying:

- The maximum value of a knapsack with a subset of items from {I_0, I_1, ...I_k} with weight w is the same as the maximum value of a knapsack with a subset of items from {I_0, I_1, ... I_{k-1}} with weight w, if item k weighs greater than w.
- Basically, you can NOT increase the value of your knapsack with weight w if the new item you are considering weighs more than w – because it WON'T fit.
- The maximum value of a knapsack with a subset of items from {I_0, I_1, ... I_k} with weight w could be the same as the maximum value of a knapsack with a subset of items from {I_1, I_2, ... I_{k-1}} with weight w, if item k should not be added into the knapsack.
- The maximum value of a knapsack with a subset of items from {I_0, I_1, ... I_k} with weight w could be the same as the maximum value of a knapsack with a subset of items from {I_0, I_1, ... I_{k-1}} with weight w-w_k, plus item k.

- You need to compare the values of knapsacks in both case 2 and 3 and take the maximal one.
- Recursively, we will STILL have an $O(2^n)$ algorithm. But, using dynamic programming, we simply have to do a double loop - one loop running n times and the other loop running W times.
- Here is a dynamic programming algorithm to solve the 0-1 Knapsack problem:

Input: S, a set of n items as described earlier, W the total weight of the knapsack. (Assume that the weights and values are stored in separate arrays named w and v, respectively.)

Output: The maximal value of items in a valid knapsack.

```
    int w, k;

    for (w=0; w <= W; w++)

        B[w] = 0

    for (k=0; k<n; k++) {

            for (w = W; w>= w[k]; w--) {

                    if (B[w – w[k]] + v[k]> B[w])

                        B[w] = B[w – w[k]] + v[k]

            }

}
```

- **Note on run time:** Clearly the run time of this algorithm is O(nW), based on the nested loop structure and the simple operation inside of both loops.
- When comparing this with the previous $O(2^n)$, we find that depending on W, either the dynamic programming algorithm is more efficient or the brute force algorithm could be more efficient.

(For example, for n=5, W=100000, brute force is preferable, but for n=30 and W=1000, the dynamic programming solution is preferable.)

Let's run through an example:

I	Item	w_i	v_i
0	I_0	4	6
1	I_1	2	4
2	I_2	3	5
3	I_3	1	3
4	I_4	6	9
5	I_5	4	7

W = 10

Item	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	6	6	6	6	6	6	6
1	0	0	4	4	6	6	10	10	10	10	10
2	0	0	4	5	6	9	10	11	11	15	15
3	0	3	4	7	8	9	12	13	14	15	18
4	0	3	4	7	8	9	12	13	14	16	18
5	0	3	4	7	8	10	12	14	15	16	19

Traveling Salesperson Problem

- The traveling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one and returning to the same city.
- The challenge of the problem is that the traveling salesman wants to minimize the total length of the trip.

- Travelling salesman problem is a permutation problem. Permutation problems usually are much harder to solve than subset problems as there are n! different permutations of n objects whereas there are only 2^n different subsets of n objects($n! > 2^n$).

Problem:

- Let G = (V, E) be a directed graph with edge costs C_{ij}. The variable C_{ij} is defined such that $C_{ij} > 0$ for all i and j and $C_{ij} = \infty$ if $(i,j) \neq E$.
- Let $|V| = n$ and assume $n > 1$.A tour of G is a directed simple cycle that includes every vertex in V. The cost of a tour is the sum of the cost of the edges on the tour. The travelling sales person problem is to find a tour of minimum cost.

Examples:

Here are some of the examples that use TSP:

- **Example 1:** We have to route a postal van to pick up mail from mail boxes located at n different sites. An n+1vertex graph can be used to represent the situation. One vertex represents the post office from which the postal van starts and to which it must return. Edge (i, j) is assigned a cost equal to the distance from site i to site j. The route taken by the postal van is a tour, and we are interested in finding a tour of minimum length.

- **Example 2:** Suppose we wish to use a robot arm to tighten the nuts on some piece of machinery on an assembly line. The arm will start from its initial position (which is over the first nut to be tightened), successively move to each of the remaining nuts, and return to the initial position. The path of the arm is clearly a tour on a graph in which vertices represent the nuts. A minimum-cost tour will minimize the time needed for the arm to complete its task.

- **Example 3:** Consider a production environment in which several commodities are manufactured on the same set of machines. The manufacture proceeds in cycles. In each production cycle, n different commodities are produced. When the machines are changed from production of commodity i to commodity j, a change over cost C_{ij} is incurred. It is desired to find a sequence in which to manufacture these commodities. This sequence should minimize the sum of change over costs (the remaining production costs are sequence independent).Since the manufacture proceeds cyclically, it is necessary to include the cost of starting the next cycle. This is just the change over cost from the last to the first commodity. Hence, this problem can be regarded as a travelling sales person problem on an n vertex graph with edge cost C_{ij}'s being the change over cost from commodity i to commodity j.

Analysis:

- Consider a tour to be a simple path that starts and ends at vertex 1. Every tour consists of an edge (l, k) for some k € V- {1} and a path from vertex k to vertex 1.

- The path from vertex k to vertex 1 goes through each vertex in V - {1,k} exactly once.

- If the tour is optimal, then the path from k to 1 must be a shortest k to 1path going through all vertices in V - {l, k}.

- Let g(i, S)be the length of a shortest path starting at vertex i, going through all vertices in S, and terminating at vertex1.Thefunction g(l, V - {1}) is the length of an optimal sales person tour.

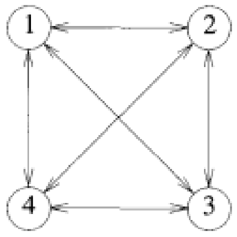
- From the principal of optimality it follows that:

$$g(1, V - \{1\}) = \min_{2 \le k \le n} \{c_{1k} + g(k, V - \{1, k\})\}$$

- After generalizing the above equation, we get :

$$g(i, S) = \min_{j \in S}\{c_{ij} + g(j, S - \{j\})\}$$

- Consider the directed graph .The edge lengths are given by matrix c :



(a)

$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

(b)

- We can solve g(l, V - {1}) if we know g(k, V -{1,k}) for all choices of k. The g values can be obtained by using (b).

- Hence, we can use the above figure to obtain g(i,S) for all S of size1.Then we can obtain g(i, S) for S with |S|= 2, and so on. When

Thus $g(2, \phi) = c_{21} = 5, g(3, \phi) = c_{31} = 6$, and $g(4, \phi) = c_{41} = 8$. Using (5.21), we obtain

$$\begin{array}{lllllll} g(2,\{3\}) & = & c_{23} + g(3,\phi) & = & 15 & g(2,\{4\}) & = & 18 \\ g(3,\{2\}) & = & 18 & & & g(3,\{4\}) & = & 20 \\ g(4,\{2\}) & = & 13 & & & g(4,\{3\}) & = & 15 \end{array}$$

Next, we compute $g(i, S)$ with $|S| = 2$, $i \neq 1$, $1 \notin S$ and $i \notin S$.

$$\begin{array}{lllll} g(2,\{3,4\}) & = & \min\ \{c_{23} + g(3,\{4\}), c_{24} + g(4,\{3\})\} & = & 25 \\ g(3,\{2,4\}) & = & \min\ \{c_{32} + g(2,\{4\}), c_{34} + g(4,\{2\})\} & = & 25 \\ g(4,\{2,3\}) & = & \min\ \{c_{42} + g(2,\{3\}), c_{43} + g(3,\{2\})\} & = & 23 \end{array}$$

|S|<n-1, the values of i and S for which g(i,S) is needed are such that i≠1 1≠S,and i ≠S.

- From the above figure , we obtain:

$$\begin{array}{lll} g(1,\{2,3,4\}) & = & \min\{c_{12} + g(2,\{3,4\}), c_{13} + g(3,\{2,4\}), c_{14} + g(4,\{2,3\})\} \\ & = & \min\{35, 40, 43\} \\ & = & 35 \end{array}$$

- An optimal tour of the graph has length 35. A tour of this length can be constructed if we retain with each g(i,S) the value of j that minimizes J(i,S).
- Then, J (l,{2,3,4}) = 2. Thus the tour starts from 1and goes to 2. The remaining tour can be obtained from g(2,{3,4}). So J(2,{3,4})= 4.
- Thus the next edge is (2,4).The remaining tour is for g(4, {3}).So J(4,{3})= 3.**The optimal tour is 1,2,4, 3, 1.**
- For each value of |S| there are n - 1 choices for i. The number of distinct sets S of size k not including 1and i is K^{n-2}. Hence:

$$N = \sum_{k=0}^{n-2} (n-1)\binom{n-2}{k} = (n-1)2^{n-2}$$

- An algorithm that proceeds to find an optimal tour will require □(n^2 2") time as the computation of g(i,S) with |S|= k requires k - 1comparisons.

UNIT – IV

Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. We use dynamic programming approach when [ ]
   A) It provides optimal solution
   B) The solution has optimal substructure
   C) The given problem can be reduced to sub problems
   D) It's faster than Greedy
2. Dynamic programming divides problems into a number of [ ]
   A) Conflicting objective functions.
   B) Policies.
   C) Unrelated constraints.
   D) Decision stages.
3. In dynamic programming, the output to stage n becomes the input to [ ]
   A) Stage n-1. B) Stage n itself.
   C) Stage n+1. D) Stage n-2.
4. Which of the following statement(s) is/are the characteristic(s) of dynamic programming? [ ]
   i) The decision at one stage transforms one state into a state in the next.
   ii) The problem can't be divided into a finite number of stages.
   iii) The final stage must be solvable by itself.
   A) i& ii are correct B) i& iii are correct
   C) ii & iii are correct D)i , ii & iii are correct
5. Dynamic programming is based on ___________.
6. Matrix chain multiplication technique is used to [ ]
   A) multiply the given matrices.
   B) find total number of elements in all matrices.
   C) write recurrence relations

D) determine the optimal parenthesization of a product of matrices

7. Which of the following is **true** about optimal binary search tree(OBST)? [ ]
   A) OBST minimizes the leaf nodes.
   B) OBST maximizes expected search cost
   C) OBST minimizes expected search cost
   D) OBST focuses on reducing the height of binary search tree.
8. Which of the following is **false** about optimal binary search tree(OBST)? [ ]
   A) OBST may not have smallest height.
   B) OBST minimizes expected search cost
   C) OBST gives O(n^2) computation time
   D) OBST always holds highest-probability key at root.
9. **Travelling Salesman Problem** is to find [ ]
   A) the shortest possible route that visits every city not exactly once and returns to the starting point.
   B) the shortest possible route that visits every city exactly once and returns to the starting point.
   C) the shortest possible route that visits every city exactly once and doesn't return to the starting point.
   D) None of the above
10. What is the computing time of optimal binary search tree? [ ]
    A) O(n) B) O(n^2) C) O(n logn) D) O(n^3)
11. Let A1, A2, A3, and A4 be four matrices of dimensions 10x5, 5x20, 20x10, and 10x5, respectively. The minimum number of scalar multiplications required to find the product A1A2A3A4 using the basic matrix multiplication method is [ ]
    A)1500 B)2000 C)500 D)100

12. Let A1, A2, A3, and A4 be four matrices of dimensions 10x5, 5x20, 20x10, and 10x5, respectively. The minimum number of scalar multiplications required to find the product A1A2A3A4 using dynamic programming is [ ]

    A)1500 B)2000 C)500 D)100

13. Consider a 0/1 knapsack with capacity, w=20. The weights and values of five items are given below. [ ]

Item(I)	I1	I2	I3	I4	I5
W_i	3	4	7	8	9
P_i	4	5	10	11	13

    What is the maximum value of knapsack subject to its capacity?

    A) 27 B) 26 C) 28 D) none of the above

14. What is the optimal TSP tour for the following distance matrix?

$$\begin{pmatrix} 0 & 2 & 9 & 10 \\ 1 & 0 & 6 & 4 \\ 15 & 7 & 0 & 8 \\ 6 & 3 & 12 & 0 \end{pmatrix}$$

    A) 1→3→2→4→1 B) 1→3→4→2→1 [ ]

    C) 1→3→4→2→1 D) 1→2→4→3→1

15. Find the optimal tour for the following distance matrix? [ ]

0	5	6	8
10	0	13	8
15	9	0	9
20	10	12	0

    A) 1→3→2→4→1 B) 1→3→4→2→1

    C) 1→4→2→3→1 D)1→2→4→3→1

16. Find out the correction solution for the given 0/1 Knapsack problem using Dynamic Programming. P=(11,21,31,33), W= (2,11,22,15), c=40, n=4. [ ]

    A){1,0,1,1} B) {1,1,0,1} C) {1,1,1,0} D) {1,0,1,0}

17. Find optimal solution for 0/1 Knapsack problem (w1, w2, w3, w4)=(10,15,6,9), (p1,p2,p3,p4)=(2,5,8,1) and m=30. [ ]

    A) (0,1,1,1) B) (1,0,0,0) C) (1,1,0,0) D) (1,0,1,0)

SECTION-B

SUBJECTIVE QUESTIONS

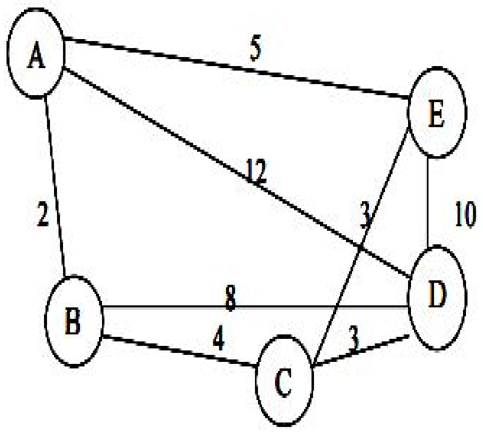
1. Distinguish between Dynamic Programming and Greedy method.
2. Write an algorithm for finding optimal binary search tree and analyze its time complexity.
3. Write an algorithm for matrix chain multiplication problem using dynamic programming.
4. What is 0/1 knapsack problem? Explain with suitable example how it is solved using dynamic programming.
5. Write an algorithm for 0/1 knapsack problem using dynamic programming.
6. Describe the Travelling salesman problem & discuss how to solve it using dynamic programming.
7. Find the minimum no of operations required for the following chain matrix multiplication using dynamic programming A(5,3) * B(3,4) * C(4,2) * D(2,6)
8. Find the minimum no of operations required for the following chain matrix multiplication using dynamic programming A(30,40) * B(40,5) * C(5, 15) * D(15, 6)
9. Construct the OBST with the identifier set(a1,a2,a3,a4) = (end, goto, print, stop) with (P1,P2,P3,P4) = (0.5,0.1,0.02,0.5) & (Q1,Q2,Q3,Q4,Q5)= (0.5,0.15,0.2,0.1,0.2) Also compute the cost of the tree.
10. Use the function OBST to compute w(i,j), r(i,j), and c(i,j), 0 <=i< j <= 4, for the identifier set (a1,a2,a3,a4)=(do, if, int, while) with p(1:4)=(3,3,1,1)

and q(0:4)=(2,3,1,1,1). Using the r(i,j)'s construct the optimal binary search tree.

11. Find an optimal solution for the dynamic programming 0/1 knapsack instance for n=3, m=6,profits are (p1, p2,p3) = (1,2,5), weights are (w1,w2,w3)=(2,3,4).
12. Solve the following 0/1 Knapsack problem using dynamic programming (p1, p2,…,p4) =(1,2,5,6), (w1, w2, …, w4) =(2,3,4,5), m=8, n=4.
13. Construct an optimal travelling sales person tour using Dynamic Programming.

$$\begin{pmatrix} 0 & 10 & 9 & 3 \\ 5 & 0 & 6 & 2 \\ 9 & 6 & 0 & 7 \\ 7 & 3 & 5 & 0 \end{pmatrix}$$

14. Consider the following set of cities and find the optimal tour and minimum cost by applying TSP approach.



SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. Four matrices M_1, M_2, M_3, M_4 of dimensions pxq,qxr,rxs and sxt respectively can be multiplied in several ways with different number of total scalar multiplications. For example when multiplied as ((M_1 x M2)x(M_3x M_4)), the total number of scalar multiplications is pqr+rst+prt. When multiplied as (((M_1 x M2)xM_3)x M_4), the total number of scalar

multiplications is pqr + prs +pst.If p=10, q=100, r=20, s=5, and t=80, then the minimum number of scalar multiplications needed is

(GATE 2011) [ ]

A) 248000 B) 44000 C)19000 D)25000