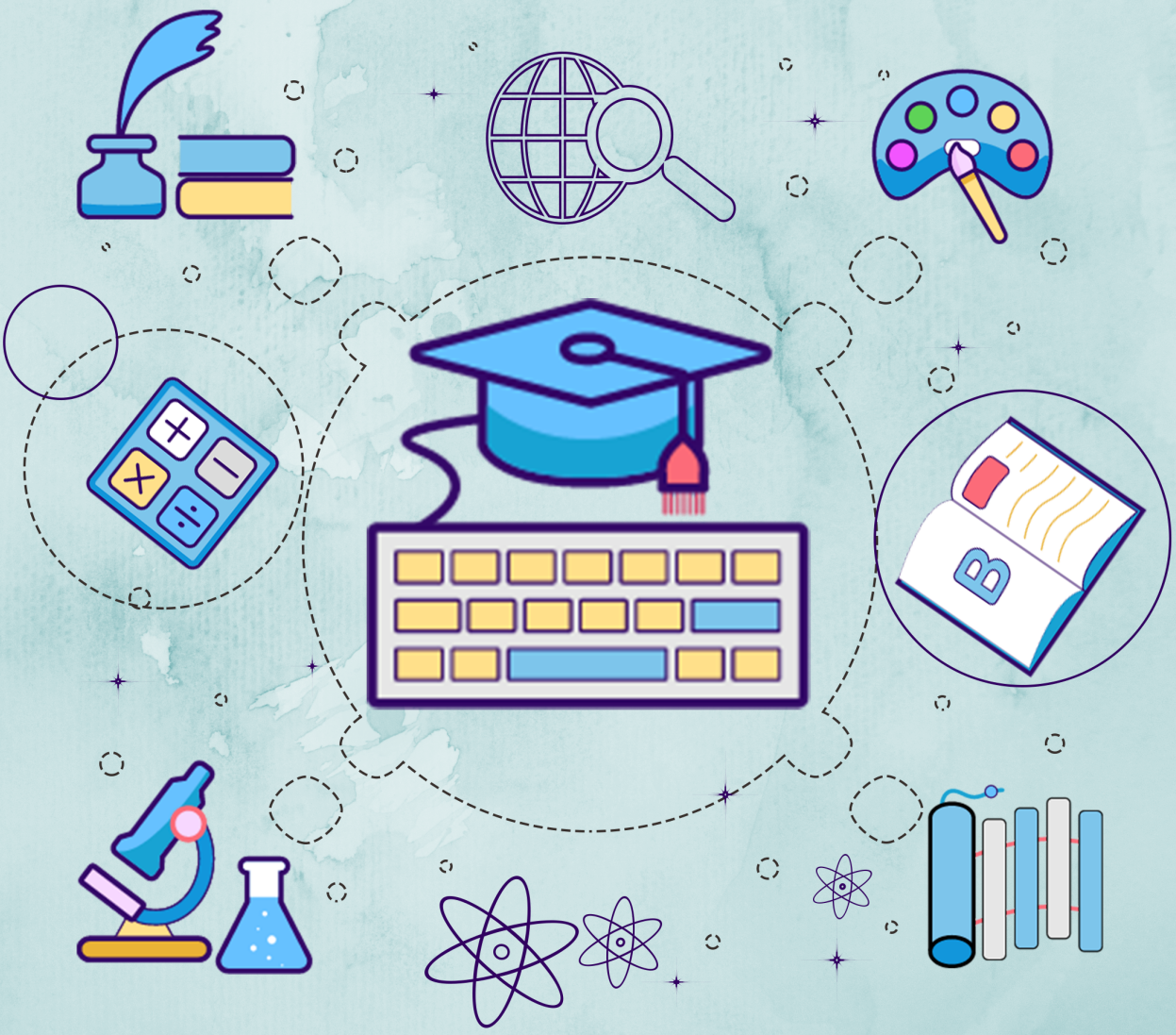
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KeralaNotes



SYLLABUS | STUDY MATERIALS | TEXTBOOK

PDF | SOLVED QUESTION PAPERS

www.keralanotes.com

KeralaNotes

KTU STUDY MATERIALS

ALGORITHM ANALYSIS AND DESIGN

CST 306

Module 4

Related Link :

- KTU S6 CSE NOTES 2019 SCHEME
- KTU S6 SYLLABUS CSE COMPUTER SCIENCE
- KTU PREVIOUS QUESTION BANK S6 CSE SOLVED
- KTU CSE TEXTBOOKS S6 B.TECH PDF DOWNLOAD
- KTU S6 CSE NOTES | SYLLABUS | QBANK | TEXTBOOKS DOWNLOAD

KeralaNotes

AAD

ALGORITHM ANALYSIS AND DESIGN - CST306

Module 4

Neethu Mathew , CSE Dept. EKCTC

keralanotes.com

Module-4 (Dynamic Programming, Back Tracking and Branch & Bound))

The Control Abstraction- The Optimality Principle- Matrix Chain Multiplication-Analysis, All Pairs Shortest Path Algorithm - Floyd-Warshall Algorithm-Analysis. The Control Abstraction of Back Tracking – The N Queen's Problem. Branch and Bound Algorithm for Travelling Salesman Problem.

For More Study Materials : www.keralanotes.com

Dynamic Programming

- Dynamic Programming is an algorithmic paradigm that solves a given complex problem by
  - breaking it into simpler subproblems ,
  - Solving each of those subproblems just once, and storing their solutions.
  - The next time the same subproblem occurs , instead of recomputing its solution ,one simply look up the previously computed solution (We stores the results of subproblems to avoid computing the same results again.)
- Dynamic programming is a technique for finding an optimal solution
- The technique of storing solutions to subproblems instead of recomputing them is called Memoization

keralanotes.com

Characteristics of Dynamic Programming

- Following are the two main properties of a problem that suggest that the given problem can be solved using Dynamic programming.
  - **Overlapping Subproblems**
  - **Optimal Substructure**

✓ Overlapping Subproblems:

- Like Divide and Conquer, Dynamic Programming combines solutions to sub-problems. Dynamic Programming is mainly used when solutions of same subproblems are needed again and again. The computed solutions to subproblems are stored in a table, so that these don't have to recomputed. Hence, this technique is needed where overlapping sub-problem exists.
- Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point storing the solutions if they are not needed again.

For More Study Materials : www.keralanotes.com

✓ **Optimal Sub-Structure**

- A given problem has Optimal Substructure Property, if the optimal solution of the given problem can be obtained by using optimal solutions of its sub-problems.
- For example, the Shortest Path problem has the following optimal substructure property −

: If a node **x** lies in the shortest path from a source node **u** to destination node **v**, then the shortest path from **u** to **v** is the combination of the shortest path from **u** to **x**, and the shortest path from **x** to **v**.

- The standard All Pair Shortest Path algorithms like Floyd - Warshall and Bellman-Ford are typical examples of Dynamic Programming.

keralanotes.com

The Principle of Optimality / The Optimality Principle

- To use dynamic programming , the problem must observe the principle of optimality
- **The principle of optimality states that an optimal sequence of decisions has the property that whatever the initial state and decisions are, the remaining states must constitute an optimal decision sequence with regard to the state resulting from the first decision.**
- Dynamic programming is a technique for finding an optimal solution

- In an optimal sequence of decisions/choices , each subsequence must also be optimal
- The principle of optimality applies if the optimal solution to a problem always contain optimal solutions to all sub-problems.

For More Study Materials : www.keralanotes.com

Control Abstraction - Dynamic programming

When developing a dynamic-programming algorithm, we follow a sequence of four steps:

Algorithm Dynamic programming ()

{

Characterize the structure of an optimal solution.

Recursively define the value of an optimal solution.

Compute the value of an optimal solution, typically in a bottom-up fashion.

Construct an optimal solution from computed information

}

Neethu Mathew , CSE Dept. EKCTC

keralanotes.com

DIVIDE AND CONQUER VERSUS DYNAMIC PROGRAMMING

Divide & Conquer Method	Dynamic Programming
• An algorithm that recursively breaks down a problem into 2 or more sub problems of the same or related type until it becomes simple enough to be solved directly	• An algorithm that helps to efficiently solve a class of problems that have overlapping subproblems and optimal substructure property
• It is Recursive.	• It is non Recursive.
• subproblems are independent of each other.	• subproblems are interdependent.
• More time consuming (as it solves each subproblem independently)	• Less time consuming (as it uses the answers of previous subproblems)
• Less efficient	• More efficient
• Top down approach	• Bottom-up approach
• Example Merge sort, binary search ,quicksort	• Example matrix chain multiplication ,optimal binary search tree

Neethu Mathew , CSE Dept. EKCTC

- Divide-and-conquer algorithms split a problem into separate subproblems, solve the subproblems, and combine the results for a solution to the original problem
- Divide-and-conquer algorithms can be thought of as top-down algorithms
- In contrast, a dynamic programming algorithm proceeds by solving small problems, then combining them to find the solution to larger problems
- Dynamic programming can be thought of as bottom-up

keralanotes.com

Optimal Matrix Multiplication / Matrix chain Multiplication

Neethu Mathew , CSE Dept. EKCTC

For More Study Materials : www.keralanotes.com

Matrix Multiplication

- Suppose, matrix A has p rows and q columns i.e., the dimension of **matrix A is p × q ,**

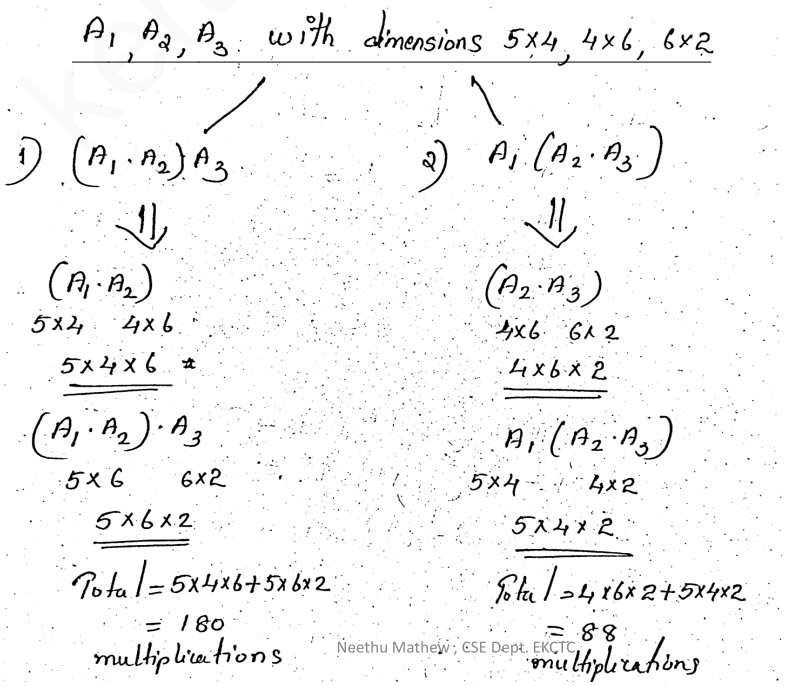
  matrix B is q × r, then result will be a **matrix C** with dimensions **p × r.**

  Order (p × q) (q × r) => (p × r)
- you can multiply 2 matrices if they are compatible: the number of columns of A must equal the number of rows of B
- Matrix multiplication is an associative but not a commutative operation.
- **To multiply a matrix of dimensions p × q with another a matrix of dimensions q × r, we need p × q × r scalar multiplications in total**

Neethu Mathew , CSE Dept. EKCTC

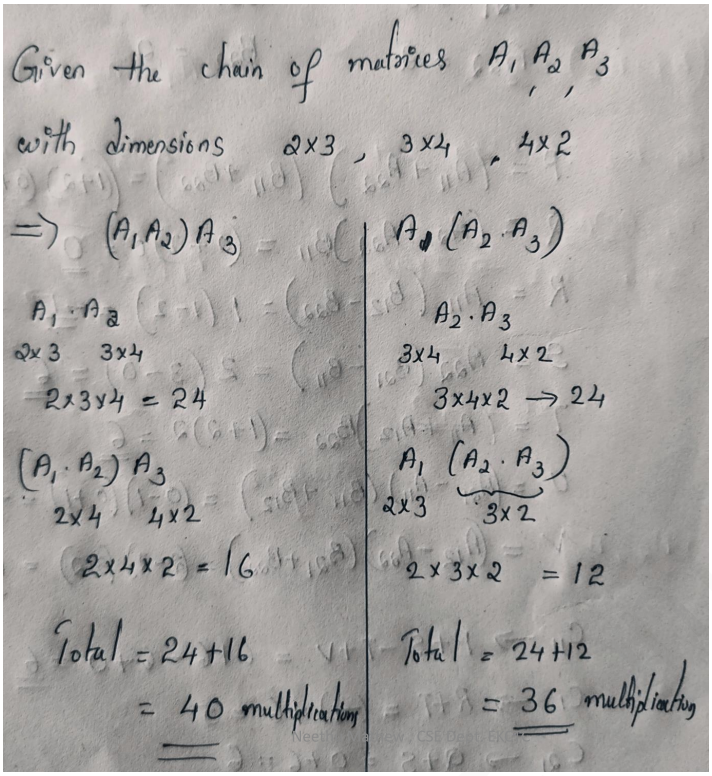
To multiply a matrix of dimensions p × q with another a matrix of q × r, we need total p × q × r scalar multiplications

Example 1:



Neethu Mathew , CSE Dept. EKCTC

Example 2 :



Optimal Matrix Multiplication / Matrix chain Multiplication

Suppose that our problem is to multiply a chain of n matrices $A_1, A_2 \ldots\ldots A_n$, and we wish to compute $A_1 . A_2 \ldots\ldots A_n$

- Given a sequence of n matrices $\boldsymbol{A_1, A_2 \ldots\ldots A_n}$, and their dimensions $\boldsymbol{P_0, P_1, P_2 \ldots\ldots P_n}$, where i = 1, 2, ..., n, and matrix $\boldsymbol{A_i}$ has dimension $\boldsymbol{P_{i-1} \times P_i}$, the optimal matrix chain multiplication problem determine the order of multiplication that minimizes the number of scalar multiplications.
- we determine the **multiplication sequence** that **minimize the number of scalar multiplications** in computing $A_1 . A_2 \ldots\ldots A_n$

  That is, determine how to parenthesize the multiplications

For More Study Materials : www.keralanotes.com

- **The first step of the dynamic programming paradigm is to characterize the structure of an optimal solution (in this case , a parenthesization).**

  The problem of determining the optimal sequence of multiplications is broken up into two questions:

  Question 1: How do we decide where to split the chain? (What is k?)

  Question 2: How do we parenthesize the sub chains $A_{1....k}$, $A_{k+1.....n}$?

- **The second step of the dynamic programming paradigm is to define the value of an optimal solution recursively in terms of the optimal solutions to subproblems.**

To help us keep track of solutions to subproblems, we will use a table, and build the table in a bottom-up manner.

Let , ***m[i, j]* be the minimum number of scalar multiplications needed to compute the $A_{i....j}$** where $1 \le i \le j \le n$

The optimum cost can be described by the following recursive formulation.

$$m[i,j] = \begin{cases} 0 & \text{if} \quad i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if} \quad i < j. \end{cases}$$

if $i = j$, then the sequence contains only one matrix, and so the cost is 0. (In other words, there is nothing to multiply.)

If $i < j$, then m[i,j] =The minimum cost for computing $A_{i....k}$ and $A_{k+1....j}$ + the cost of multiplying these two matrices.

Neethu Mathew , CSE Dept. EKCTC

To keep track of optimal sub solutions, we store the value of k in a table $s[i, j]$. Recall, k is the place at which we split the product $A_{i....j}$ to get an optimal parenthesization. That is,

$s[i, j] = k$ such that $m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} \cdot p_k \cdot p_j$

- **The third step of the dynamic programming paradigm is to Construct the value of an optimal solution, in a bottom-up fashion.**
- **The forth step of the dynamic programming paradigm is to Construct an optimal solution from computed information stored in table s**

Neethu Mathew , CSE Dept. EKCTC

For More Study Materials : www.keralanotes.com

Problem

Q)

Find out the optimal multiplication order for the matrix chain A_1, A_2, A_3, A_4 where P_0=5, P_1=4, P_2=6, P_3=2, P_4=7

0r

Q) Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is 5x4 , 4x6 , 6x2 , 2x7

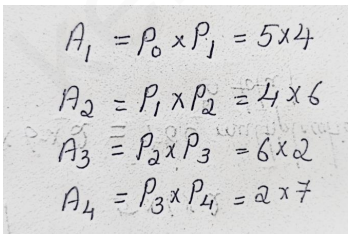
Q) Find out the optimal multiplication order for the matrix chain A_1, A_2, A_3, A_4 of order 5x4 , 4x6 , 6x2 , 2x7

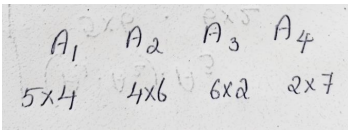
Neethu Mathew , CSE Dept. EKCTC

Ans)

A_i has dimension $P_{i-1} \times P_i$

Order of matrix is…….





$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j) & i < j \end{cases}$$

$m[i, j]$= minimum number of scalar multiplications needed to compute the $A_{i \ldots j}$

(min cost for computing the product A_i to A_j)

i j

A_1, A_2, A_3, A_4

Using matrix m , we can find the min cost of multiplication(no. of multiplications needed)

Using matrix s, we can identify the optimal parenthesization or order matrix multiplication which yields min cost

$s[i, j]$= k , for which $m[i, j]$ is minimum

Neethu Mathew , CSE Dept. EKCTC

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j) & i < j \end{cases}$$

$A_1 . A_2 . A_3 . A_4$

➢ **Chain length 1 (initially consider single matrices)**

$m[1,1] = 0$

$m[2,2] = 0$

$m[3,3] = 0$

$m[4,4] = 0$

update

m

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

➢ **Chain length 2 (2 matrices)**

Take $A_1 . A_2$

$m[1,2] = \min_{1 \le k < 2} \{ m[1,1] + m[2,2] + p_0 p_1 p_2 \}$ (i j, $k=1$)

$= 0 + 0 + 5 \times 4 \times 6$

$= 120$

Neethu Mathew , CSE Dept. EKCTC

$m[i,j]$=? | $m[1,2]$=120

$s[i,j]$=k | $s[1,2]$=1

Update matrix m And s

Take $A_2 . A_3$

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j) & i < j \end{cases}$$

$m[2,3] = \min_{2 \le k < 3} \{ m[2,2] + m[3,3] + p_1 p_2 p_3 \}$ (i j, $k=2$)

$= 0 + 0 + 4 \times 6 \times 2 = 48$

$m[2,3]=48$

$s[2,3]=2$

Update matrix m And s

Take $A_3 . A_4$

$m[3,4] = \min_{3 \le k < 4} \{ m[3,3] + m[4,4] + p_2 p_3 p_4 \}$ ($k=3$)

$= 0 + 0 + 6 \times 2 \times 7 = 84$

$m[3,4]=84$

$s[3,4]=3$

Update matrix m And s

m

	1	2	3	4
1	0	120		
2		0	48	
3			0	84
4				0

S

	1	2	3	4
1		1		
2			2	
3				3
4				

Neethu Mathew , CSE Dept. EKCTC

keralanotes.com

$A_1.A_2.A_3.A_4$

➢ **Chain length 3**

Take $A_1.A_2.A_3$

A_1 to A_3
i=1
j=3

k = 1
$A_1(A_2.A_3)$
k=2
$(A_1.A_2).A_3$

$$m[1,3] = \min_{1 \le k < 3} \begin{cases} A_1(A_2 \cdot A_3) \\ k=1 \\ m[1,1] + m[2,3] + P_0 P_1 P_3 \\ (A_1 A_2) \cdot A_3, \quad k=2 \\ m[1,2] + m[3,3] + P_0 P_2 P_3 \end{cases}$$

$$= \min \begin{cases} k=1, \\ 0 + 48 + 5 \times 4 \times 2 = 88 \\ k=2 \\ 120 + 0 + 5 \times 6 \times 2 = 180 \end{cases}$$

m[1, 3]=88 (Take min cost to multiply A_1 **to** A_3)

s[1,3]=1 (splitting after A_1 gives min cost)

Update matrix m And s

Neethu Mathew , CSE Dept. EKCTC

Take A_2, A_3, A_4

$$m[2,4] = \min_{2 \le k < 4} \begin{cases} k=2, \\ m[2,2] + m[3,4] + P_1 P_2 P_4 \\ k=3, \\ m[2,3] + m[4,4] + P_1 P_3 P_4 \end{cases}$$

$A_2 (A_3.A_4)$

$(A_2 . A_3)A_4$

$$= \min \begin{cases} 0 + 84 + 4 \times 6 \times 7 \\ 48 + 0 + 4 \times 2 \times 7 \end{cases} = \min \begin{cases} k=2, 252 \\ k=3, 104 \end{cases}$$

m[2,4]=104
s[2,4]=3
Update matrix m And s

m

	1	2	3	4
1	0	120	88	
2		0	48	104
3			0	84
4				0

s

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

Neethu Mathew , CSE Dept. EKCTC

➢ **Chain length 4**

$A_1.A_2.A_3.A_4$

$$m[1,4] = \min \begin{cases} k=1, & m[1,1]+m[2,4]+P_0P_1P_4 \\ k=2, & m[1,2]+m[3,4]+P_0P_2P_4 \\ k=3, & m[1,3]+m[4,4]+P_0P_3P_4 \end{cases}$$

$$= \min \begin{cases} k=1, & 0+104+5\times4\times7 = 244 \\ k=2, & 120+84+5\times6\times7 = 414 \\ k=3, & 88+0+5\times2\times7 = 158 \end{cases}$$

$m[1,4]=158$
$s[1,4]=3$
Update matrix m And s

m

	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

S

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

parenthesization

minimum number of multiplications needed to compute the product $A_1.A_2.A_3.A_4$ is
m[1,4]=158

$A_1.A_2.A_3.A_4$
$(A_1.A_2.A_3)A_4$
$((\boldsymbol{A_1})(\boldsymbol{A_2}.\boldsymbol{A_3}))\boldsymbol{A_4}$

Neethu Mathew, CSE Dept. EKCTC

RUNNING TIME:

Recursive solution takes exponential time.

Matrix-chain order yields a running time of $O(n^3)$

For More Study Materials : www.keralanotes.com

All pairs shortest path problem

- It is a shortest path problem where the shortest path between every pair of vertices is computed.

Floyd-Warshall Algorithm

- It is an All Pairs Shortest Path Algorithm , used to find shortest path between all pairs of vertices
- negative edges are allowed

Algorithm :

```
Let V = number of vertices in graph
Let dist = V × V array of minimum distances.
for each vertex V
        dist [V][V] = 0
for each edge (u, v)
        dist[u][v] = weight(u, v)
for k from 1 to V
        for i from 1 to V
                for j from 1 to V
                        if dist[i, j] > dist[i, k] + dist[k, j]
                                dist[i, j] = dist[i, k] + dist[k, j]
                        end if
```

Neethu Mathew , CSE Dept. EKCTC

Analysis

Floyd Warshall Algorithm consists of three loops over all the nodes. Each loop has constant complexities. Hence, the time complexity of Floyd Warshall algorithm is $O(n^3)$. Here, n is the number of nodes in the given graph. Space complexity is $O(n^2)$.

- Consider the graph
- Write the initial distance matrix , D^0
  - It represents the distance between every pair of vertices in the form of given weights
  - For diagonal elements , distance value = 0.
  - For vertices having a direct edge between them, distance value = weight of that edge.
  - For vertices having no direct edge between them, distance value = ∞
- If 4 vertices , first construct D^0 from graph and then D^1 D^2 D^3 D^4
- Let D^k(i,j) be the weight of the shortest path between i and j

$$D^k[i,j] = \min\left\{ D^{k-1}[i,j],\ D^{k-1}(i,k) + D^{k-1}[k,j] \right\}$$

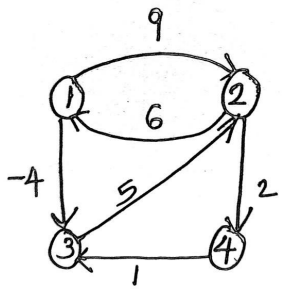
or

$$d_{ij}^{(k)} = \min\left[d_{ij}^{(k-1)},\ d_{ik}^{k-1} + d_{kj}^{(k-1)} \right]$$

Neethu Mathew , CSE Dept. EKCTC

keralanotes.com

Problem) Consider the following directed weighted graph. Using Floyd Warshall Algorithm, find the shortest path distance between every pair of vertices.



Ans)

$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 9 & -4 & \infty \\ 6 & 0 & \infty & 2 \\ \infty & 5 & 0 & \infty \\ \infty & \infty & 1 & 0 \end{bmatrix} \end{array} \Rightarrow D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 9 & -4 & \infty \\ 6 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{array}$$

find $D^1(2,3)$
$D^1(2,4)$
$D^1(3,2)$
$D^1(3,4)$
$D^1(4,2)$
$D^1(4,3)$

Neethu Mathew , CSE Dept. EKCTC

$D^1(2,3) = ?$

$D^k(i,j) = \min\{D^{k-1}(i,j),\ D^{k-1}(i,k) + D^{k-1}(k,j)\}$

$D^1(2,3) = \min\{D^{k-1}(2,3),\ D^{1-1}(2,1) + D^{1-1}(1,3)\}$

(k=1)

$= \min\{D^0(2,3),\ D^0(2,1) + D^0(1,3)\}$

$= \min\{\infty, 2\} = 2$

$D^1(2,4) = \min[D^0(2,4),\ D^0(2,1) + D^0(1,4)]$

$= \min(2,\ 6+\infty) = 2$

$D^1(3,2) = \min(D^0(3,2),\ D^0(3,1) + D^0(1,2))$

$= \min(5,\ \infty+9) = 5$

$D^1(3,4) = \min(D^0(3,4),\ D^0(3,1) + D^0(1,4))$

$= \min(\infty,\ \infty+\infty) = \infty$

$D^1(4,2) = \min(D^0(4,2),\ D^0(4,1) + D^0(1,2))$

$= \min(\infty,\ \infty+0) = \infty$

$D^1(4,3) = \min(D^0(4,3),\ D^0(4,1) + D^0(1,3))$

$= \min(1,\ \infty+-4) = 1$

$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 9 & -4 & \infty \\ 6 & 0 & 2 & 2 \\ \infty & 5 & 0 & \infty \\ \infty & \infty & 1 & 0 \end{bmatrix} \end{array}$$

$$\downarrow$$

$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 9 & -4 & \infty \\ 6 & 0 & 2 & 2 \\ \infty & 5 & 0 & \infty \\ \infty & \infty & 1 & 0 \end{bmatrix} \end{array} \Rightarrow D^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 9 & & \\ 6 & 0 & 2 & 2 \\ & 5 & 0 & \\ & \infty & & 0 \end{bmatrix} \end{array}$$

Neethu Mathew , CSE Dept. EKCTC

For More Study Materials : www.keralanotes.com

$D^2(1,3) = \min\left(D^1(1,3),\ D^1(1,2) + D^1(2,3)\right)$

$(k=2,\ k-1=1) \quad = \min(-4,\ 9+2) = \min(-4, 11) = -4$

$D^2(1,4) = \min\left(D^1(1,4),\ D^1(1,2) + D^1(2,4)\right)$

$= \min(\infty,\ 9+2) = (\infty, 11) = 11$

$D^2(3,1) = \min\left(D^1(3,1),\ D^1(3,2) + D^1(2,1)\right)$

$= \min(\infty,\ 5+6) = 11$

$D^2(3,4) = \min\left(D^1(3,4),\ D^1(3,2) + D^1(2,4)\right)$

$= \min(\infty,\ 5+2) = \min(\infty, 7) = 7$

$D^2(4,1) = \min\left(D^1(4,1),\ D^1(4,2) + D^1(2,1)\right)$

$= \min(\infty,\ \infty+6) = \infty$

$D^2(4,3) = \min\left(D^1(4,3),\ D^1(4,2) + D^1(2,3)\right)$

$= \min(1,\ \infty+2) = 1$

$D^2 =$

	1	2	3	4
1	0	9	-4	11
2	6	0	2	2
3	11	5	0	7
4	∞	∞	1	0

$D^3 =$

	1	2	3	4
1	0		-4	
2		0	2	
3	11	5	0	7
4			1	0

$D^3(1,2) = \min\left(D^2(1,2),\ D^2(1,3) + D^2(3,2)\right)$

$(k=3,\ k-1=2) \quad = \min(9,\ -4+5) = \min(9, 1)$

$= 1$

$D^3(1,4) = \min\left(D^2(1,4),\ D^2(1,3) + D^2(3,4)\right)$

$= \min(11,\ -4+7) = \min(11, 3) = 3$

$D^3(2,1) = \min\left(D^2(2,1),\ D^2(2,3) + D^2(3,1)\right)$

$= \min(6,\ 2+11) = \min(6, 13)$

$= 6$

$D^3(2,4) = \min\left(D^2(2,4),\ D^2(2,3) + D^2(3,4)\right)$

$= \min(2,\ 2+7) = (2, 9)$

$= 2$

$D^3(4,1) = \min\left(D^2(4,1),\ D^2(4,3) + D^2(3,1)\right)$

$= \min(\infty,\ 1+11)$

$= \min(\infty, 12) = 12$

$D^3(4,2) = \min\left(D^2(4,2),\ D^2(4,3) + D^2(3,2)\right)$

$= \min(\infty,\ 1+5) = (\infty, 6)$

$= 6$

$D^3 =$

	1	2	3	4
1	0	1	-4	3
2	6	0	2	2
3	11	5	0	7
4	12	6	1	0

$D^4 =$

	1	2	3	4
1	0	1	-4	3
2	6	0	2	2
3	11	5	0	7
4	12	6	1	0

Shortest path will get from final vertex D^4

From 3 to 1 is 11
3 to 2 is 5
3 to 4 is 7

For More Study Materials : www.keralanotes.com

Back Tracking

- Backtracking is an algorithmic technique that considers searching in every possible combination for solving a computational problem.
- A backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output.
  - ✓ The Brute force approach tries out all the possible solutions and chooses the desired/best solutions.
- It is known for solving problems recursively one step at a time and removing those solutions that do not satisfy the problem constraints at any point of time.
- The backtracking approach is generally used in the cases where there are possibilities of multiple solutions.
- The term backtracking implies - if the current solution is not suitable, then eliminate that and **backtrack (go back)** and try other solutions.

keralanotes.com

Tree Organization : State Space Tree

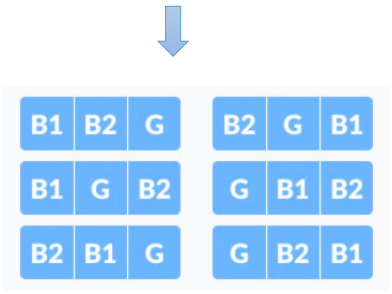
- The tree representation of a problem is called as a " **state space tree**" or solution tree
  It represents all possible states (solution or non-solution) of that given problem in the form of a tree
- In Backtracking, we search Depth first for solutions
- In Backtracking technique , we backtrack to the last valid path as soon as we hit a dead end

For More Study Materials : www.keralanotes.com

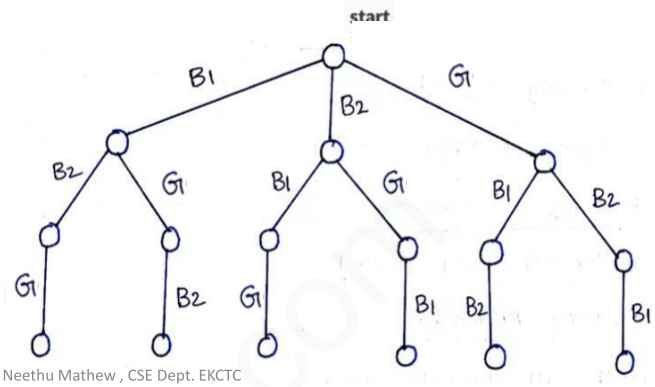
Example : Backtracking Approach

- **Problem:** You want to find all the possible ways of arranging 2 boys and 1 girl on 3 chairs.
- **Solution**: There are a total 6 possibilities. We will try all the possibilities and get the possible solutions. We recursively try all the possibilities.

All the possibilities are

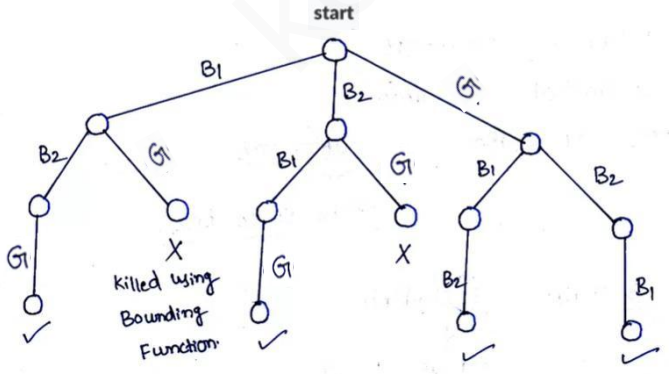


The following **state space tree** shows the all possible solutions:



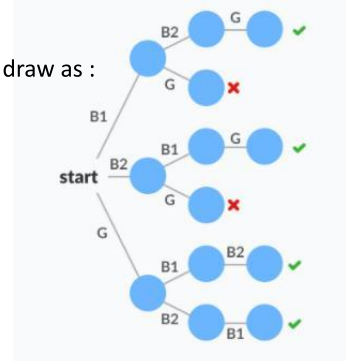
- Considering the **Constraint: Girl should not be on the middle chair**

The following state space tree shows the possible solutions:



Total 4 solutions

Or we can also draw as :



State tree with all the solutions

- The constraints applied to find the solution is called **bounding function**

For More Study Materials : www.keralanotes.com

Backtracking Algorithm – General Method

General Algorithm (Iterative)

```
Algorithm IBacktrack(n)
// This schema describes the backtracking process.
// All solutions are generated in x[1 : n] and printed
// as soon as they are determined.
{
    k := 1;
    while (k ≠ 0) do
    {
        if (there remains an untried x[k] ∈ T(x[1], x[2], ...,
            x[k − 1]) and B_k(x[1], ..., x[k]) is true) then
        {
                if (x[1], ..., x[k] is a path to an answer node)
                    then write (x[1 : k]);
                k := k + 1; // Consider the next set.
        }
        else k := k − 1; // Backtrack to the previous set.
    }
}
```

Neethu Mathew , CSE Dept. EKCTC

keralanotes.com

- In many applications of backtrack method , the desired solution is expressible as an n-tuple (x_1, x_2 ..... x_n), where x_i are chosen from some finite set S_i ($x_i \in S_i$)
- The x_i values must satisfies (maximizes or minimizes) a criterion function P(x_1, x_2..... x_n)
- Suppose m_i is the size of set S_i . Then there are m = m_1, m_2..... m_n n-tuples, which are possible candidates for satisfying the function P. The backtrack algorithm, on an average, requires far less than m trials.
- The basic idea is to Buildup the vector, one component (x_1, x_2 ..... x_n) at a time and use modified criterion functions P_i (x_1, x_2 ..... x_n) (also called bounding functions) to test whether the vector formed has any chance of success

Neethu Mathew , CSE Dept. EKCTC

Applications of Backtracking:

- **N Queen problem**.
- 0/1 knapsack problem
- Sum of subsets
- Graph coloring
- Maze solving problem.
- Hamiltonian Paths

Neethu Mathew , CSE Dept. EKCTC

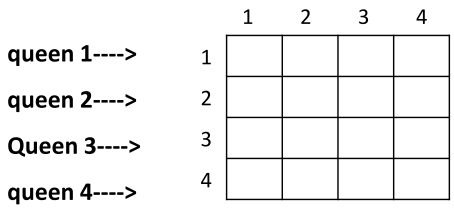
keralanotes.com

N Queen's Problem

✓ **Problem** :

- Given an **n x n** chessboard, How to place **n** queens on an ***n x n*** chessboard such that no queens may attack each other , i.e., no two queens are in same row or same column or same diagonal.
- It can be seen that for n = 1, the problem has a trivial solution, and no solution exist for n = 2 and n =3.
- So let us consider the **4-queens problem**

n=4



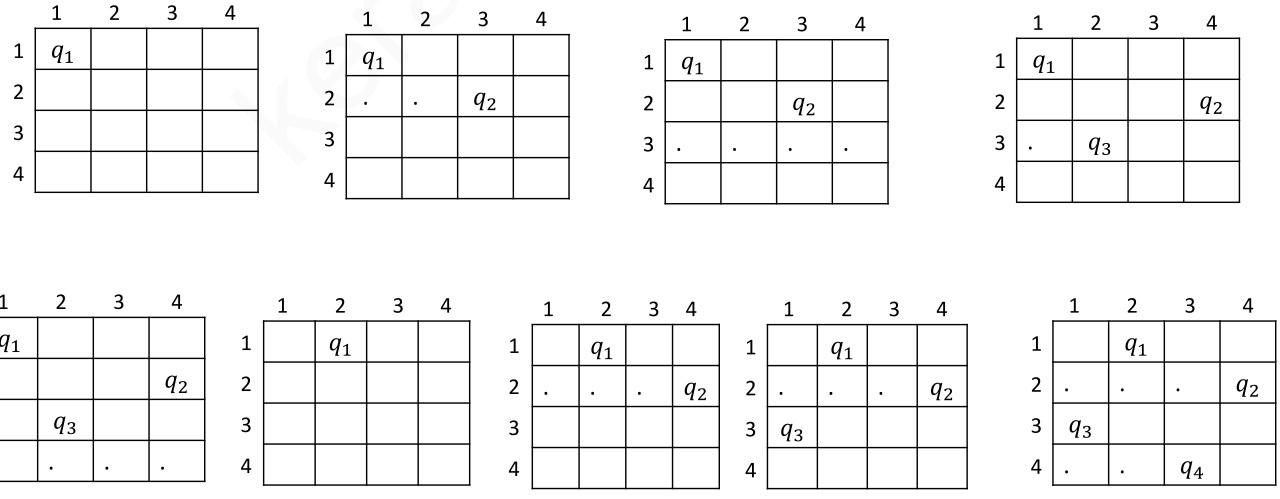
4x4 chessboard

Neethu Mathew , CSE Dept. EKCTC

- Since we have to place 4 queens such as q_1, q_2, q_3, q_4 on a chessboard such that no 2 queens attack each other. In such a condition each queen must be placed on a different row ie, we place queen i on row i
- Now we place queen 1 , q_1 in the first possible position of its row (ie, (1,1) - column 1 of row 1)
- Next we place queen 2 (q_2) so that both these queens do not attack each other. We find that If we place q_2 in columns 1 and 2 then the dead end is encountered. Thus the first acceptable position is (2,3),ie, in row 2 and column 3,
- But then no position is left for placing queen q_3 safely. So we backtrack one step and place the queen q_2 in next possible position at (2,4).
- Then queen q_3 is placed at (3,2). But later this position also leads to a dead end and no place is found where q_4 can be placed safely. Then we have to backtracks all the way to queen 1 and place it to (1,2) and then all other queens are placed safely by moving q_2 to (2,4) , q_3 to (3,1), and q_4 to (4,3)
- Ie, we get a solution < 2,4,1,3 > ,this is one of the possible solution for 4 queens problem

Neethu Mathew , CSE Dept. EKCTC

keralanotes.com



solution is (2, 4, 1, 3)

Neethu Mathew , CSE Dept. EKCTC

- **solution 1 is < 2,4,1,3 >**

	1	2	3	4
1		q_1		
2				q_2
3	q_3			
4			q_4	

q_1 in 1st row 2nd column
q_2 in 2nd row 4th column
q_3 in 3rd row 1st column
q_4 in 4th row 3rd column

- The other solution is < 3,1,4,2 >

	1	2	3	4
1			q_1	
2	q_2			
3				q_3
4		q_4		

All the solution to 4 queen problem can be represented as 4 tuples (x_1, x_2, x_3, x_4)

Where x_i represent the column on which queen q_i is placed

4-Queen Problem

STEP 1: Placed 1st queen Q1 in the 1st column.

Q1			

STEP 2: After placing 1st queen in the 1st column , we cannot place 2nd queen in the 1st or 2nd column(diagonally). So, we place Q2in the 3rd column.

Q1			
		Q2	

- **STEP 3:** After placing the 1st and 2nd queen we cannot place **Q3** anymore then the dead end is encountered .

Q1			
		Q2	
.	.	.	.

- So, we **backtrack** one step and place the 2nd queen in the 4th column.

Q1			
		backtrack →	Q2

- **STEP 4:** After placing the 3rd queen in the 2nd column, we cannot place Q4 queen any where then **dead end** is encountered .

Q1			
			Q2
	Q3		
	.	.	.

For More Study Materials : www.keralanotes.com

- **STEP 5:** From step 4 we notice that for the placement of Q4 position of Q1,Q2 and Q3 cannot be changed. Hence , now we **backtrack** and start with the placement of queen Q1in the 2nd column.

backtrack → Q1		

- **STEP 6:**Having placed the queen Q1 in the 2nd column, we can place Q2 in the 4th column.

backtrack → Q1		
		Q2

- **STEP 7:** After placed queen Q2, we can queen Q3 placed only in the 1st column.

	Q1		
			Q2
Q3			

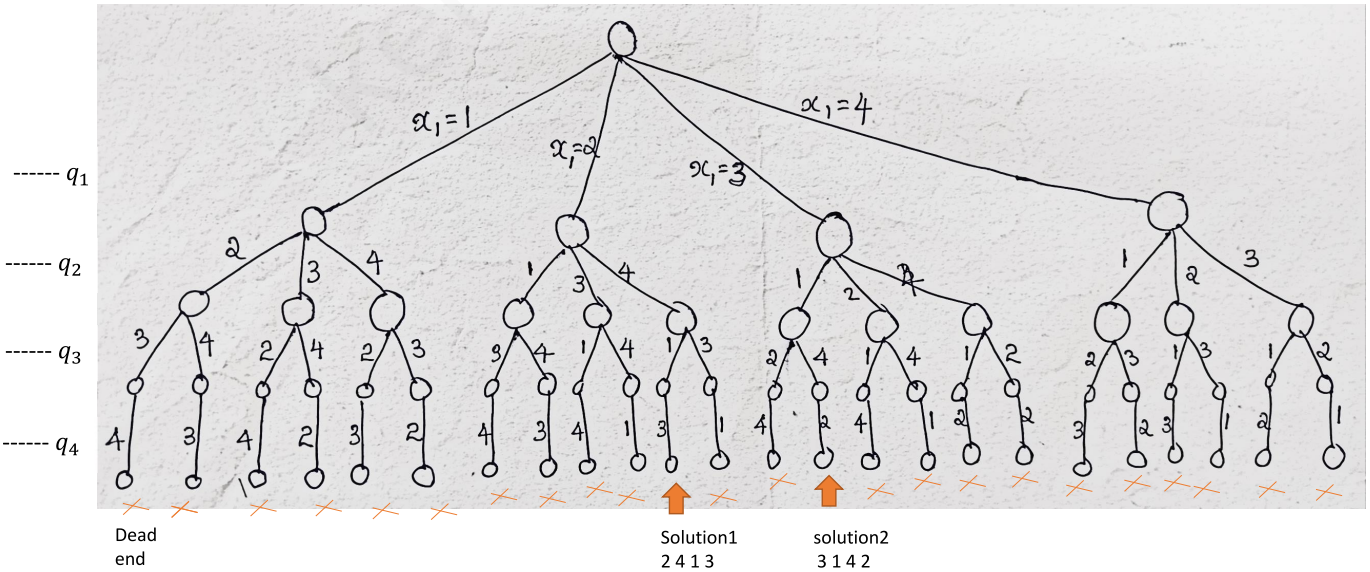
- **STEP 8:** Now after placing queens Q1,Q2 and Q3, we can queen Q4 place only in the 3rd column.

	Q1		
			Q2
Q3			
		Q4	

- **Therefore this is one possible solution vector for 4 queens problem is (2,4,1,3).**

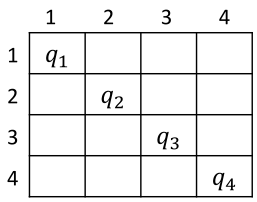
Neethu Mathew , CSE Dept. EKCTC

State space tree



Neethu Mathew , CSE Dept. EKCTC

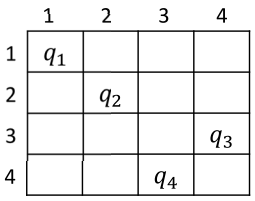
Check Path 1 2 3 4



q_1 in 1st row 1st column
q_2 in 2nd row 2nd column
q_3 in 3rd column
q_4 in 4th column

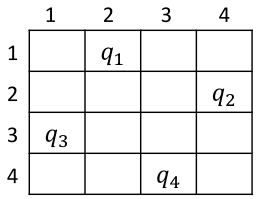
Not possible

Path 1 2 4 3

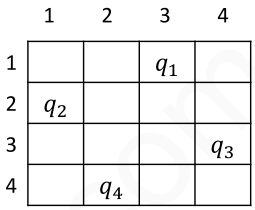


Not possible
(dead end)

Check other ways and we get 2 solutions



Solution 1
2 4 1 3



Solution 2
3 1 4 2

N Queen's Algorithm

```
N - Queens (k, n)
{
  // Prints all possible placements of n queens on an nxn chess board so that none of them are in attacking position
  for i=1 to n do
      if Place (k, i) then
      {
        x [k] = i;
        if (k ==n) then
          write (x [1....n));
        else
          N - Queens (k + 1, n);
      }
}
```


For More Study Materials : www.keralanotes.com

Place (k, i)

```
{
    // Returns true if a queen can be placed in the k^th row and i^th column ,Otherwise return false.
    //x [ ] is a global array
   for j = 1 to k -1 do
    if ((x[j] == i) or (abs (x[j]-i) == abs (j-k) )          //same column or same diagonal
     {
      return false;
      }
    return true;
}
```


keralanotes.com

If two queens are placed at position (i, j) and (k, l).

Then they are on same diagonal only if (i - j) = k - l or i + j = k + l.

The first equation implies that j - l = i - k.

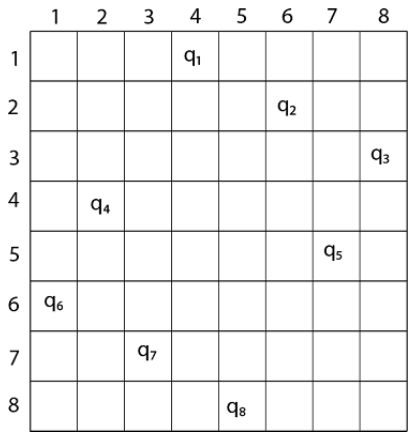
The second equation implies that j - l = k - i.

Therefore, two queens lie on the same diagonal if and only if |j-l|=|i-k|

Suppose that two queens are placed at (i, j) and (k, l) positions.
They are on the same diagonal iff
$i - j = k - l$ or $i + j = k + l$
i.e., $j - l = i - k$ or $j - l = k - i$
$|j - l| = |i - k|$

For More Study Materials : www.keralanotes.com

❑ One possible solution for **8 queens problem** is shown in fig:



Thus, the solution for 8 -queen problem for (4, 6, 8, 2, 7, 1, 3, 5).

keralanotes.com

Branch and Bound

- Branch and bound (B&B) is an algorithmic design strategy for solving optimization problems
- B&B is a rather general optimization technique that applies where the greedy method and dynamic programming fail.
- Can Use state space tree
- It is a state space search method in which all the children of a node are generated before expanding any of its children
- Two general method of Branch and bound : a BFS like state space search in which the live nodes are maintained using a queue is called FIFO , And a D-Search like state space search in which the live nodes are maintained using a stack is called LIFO.
- Bounding functions are used to avoid the generation of subtrees that do not contain an answer node.
- A cost function can be used to derive a bounding function

For More Study Materials : www.keralanotes.com

- **Search strategies in Branch and bound**
  - FIFO (first in first out) search

    -----> it's a BFS

    children of E node or (live nodes) are inserted in a queue
  - LIFO (Last in first out) search

    -----> LIFO branch and bound is a D-Search

    children of E node or (live nodes) are inserted in a stack
  - LC (Least Cost) search

    -----> It pick the one with least cost. E node is the live node with the best cost value.

- **Terminologies**
  - **Live node:** A node which has been generated and all of whose children are not yet been generated/expanded is called a live node.
  - **E node (Expanded node):** A live node whose children are currently been generated is called an E-node (node currently being expanded).
  - **Dead node**: Node which can not be expanded further.
  - The constraints applied to find the solution is called **bounding function** used to kill live nodes without generating all their children)

keralanotes.com

General Branch and Bound Algorithm

- Each solution is assumed to be expressible as an array X[1:n]
- A predictor, called an approximate cost function C, is assumed to have been defined.

❑ **Procedure B&B()**

```
begin
E: nodepointer;
E := new(node);          // this is the root node which is the dummy start node
H: heap;                 // A heap for all the live nodes .
                         // H is a min-heap for minimization problems, & a max-heap for maximization problems.
while (true) do
   if (E is a final leaf) then        //  E is an optimal solution
     print out the path from E to the root;
     return;
   endif
   Expand(E);
```


For More Study Materials : www.keralanotes.com

```
    if (H is empty) then
        report that there is no solution;
        return;
    endif
    E := delete-top(H);
  end while
end
```

❑ **Procedure Expand(E)**

```
begin
  - Generate all the children of E;
  - Compute the approximate cost value C of each child;
  - Insert each child into the heap H;
end
```

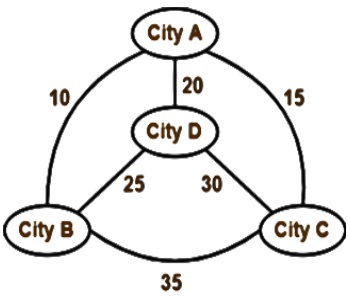

keralanotes.com

Travelling Salesman Problem(TSP)

Using Branch And Bound

"*Given a set of cities and distance between every pair of cities, the problem is to find the* ***shortest possible route*** *that* ***visits every city exactly once*** *and returns to the starting point.*"

Example - The following graph shows a set of cities and distance between every pair of cities-



If salesman starting city is A, then a TSP tour in the graph is-

A → B → D → C → A

Tsp chose the path with min distance
Cost of the tour = 10 + 25 + 30 + 15 = 80 units

For More Study Materials : www.keralanotes.com

Steps of TSP

Basic steps

Let G=(V,E) be a direct graph defining an instance of the TSP.

1. This graph is first represented by a *cost matrix* where
   c_{ij} = the cost of edge , if there is a path between vertex i and vertex j
   $c_{ij} = \infty$, if there is no path
2. Convert cost matrix into *reduced matrix* i.e.,every row and column should contain at least one zero entry.
3. Cost of the reduced matrix is the sum of elements that are subtracted from rows and columns of cost matrix to make it reduced.
4. Make the *state space tree* for reduced matrix.
5. To find the next E-node, find the *least cost valued node* by calculating the reduced cost matrix with every node.
6. If (i, j) edge is to be included, then there are three conditions to accomplish this task:
   1. Change all entries in row i and column j of A to ∞ .
   2. set A [j, 1] =∞
   3. Reduce all rows and columns in resulting matrix except for rows and columns containing ∞.
7. Calculate the cost of the matrix where
   cost = L + cost (i, j) + r
   where L = cost of original reduced cost matrix and
   r= new reduced cost matrix
8. Repeat the above steps for all the nodes until all the nodes are generated and we get a path.

Neethu Mathew , CSE Dept. EKCTC

Analysis : -The worst case complexity of TSP is $O(n^2 2^n)$

keralanotes.com

Example:

Solve Travelling Salesman problem (TSP) for the following graph using Branch and Bound Technique /

Find out the shortest tour for the Travelling Salesman Problem for the cost matrix given

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Neethu Mathew , CSE Dept. EKCTC

Apply row reduction

Reducing the matrix

We find Minimum values of each row

	1	2	3	4	5	min
1	∞	20	30	10	11	→ 10
2	15	∞	16	4	2	→ 2
3	3	5	∞	2	4	→ 2
4	19	6	18	∞	3	→ 3
5	16	4	7	16	∞	→ 4

Reduce 10 from row 1
Reduce 2 from row 2
Reduce 2 from row 3
Reduce 3 from row 4
Reduce 4 from row 5

After Reducing the Minimum values from all values of each row ,

we get the row wise reduced cost matrix =

	1	2	3	4	5
1	∞	10	20	0	1
2	13	∞	14	2	0
3	1	3	∞	0	2
4	16	3	15	∞	0
5	12	0	3	12	∞

row wise reduction sum = 10+2+2+3+4 = **21**

Neethu Mathew , CSE Dept. EKCTC

Apply column reduction

Find the Minimum values from each column

	1	2	3	4	5
1	∞	10	20	0	1
2	13	∞	14	2	0
3	1	3	∞	0	2
4	16	3	15	∞	0
5	12	0	3	12	∞
	↓	↓	↓	↓	↓
	1	0	3	0	0

Reduce 1 from column 1
Reduce 3 from column 3

After Reducing the Minimum values from all values of each column ,

we get the **reduced cost matrix** =>

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

column wise reduction sum = 1+3 = **4**

Total sum=reduced cost = row wise reduction sum +column wise reduction sum

= 21+4 = **25**

25 is the cost of **root** ie, **node 1** . Because this is the **initially reduced cost matrix**

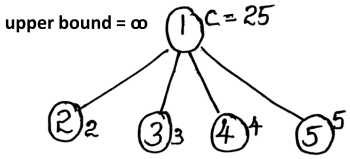
Lower bound for node is 25 and upper bound for node is ∞

Neethu Mathew , CSE Dept. EKCTC

keralanotes.com

Starting from node 1. we can next visit vertices **2,3,4 and 5**. so consider to explore the path (1,2) (1,3) (1,4) and (1,5)

Tree organization up to this point is as follows :



To find the cost of node 2 , Consider the path **(1,2) :**

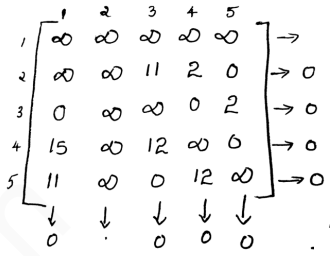
- change all entries of row 1 and column 2 to ∞
- Also set (2,1) to ∞ (once we travelled from 1 to 2 ,no path go back from 2 to 1. so)
- Apply row and column reduction (find min value and reduce)

we get, reduction cost= row wise + column wise reduction sum = 0

- Cost of node 2 = Cost of (1,2)+ cost of reduction (ie cost of parent node)+ new cost of reduction

= C(1,2)+ r + $r^{\wedge}$

= 10+25+0 =**35**

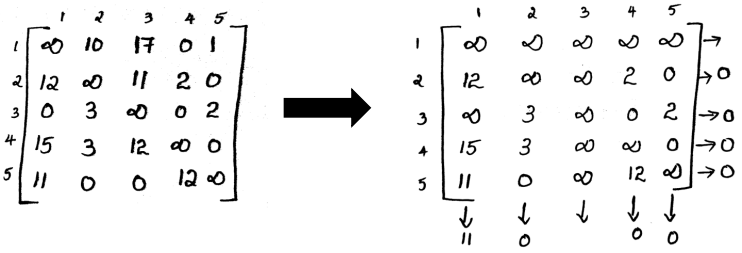


To find the cost of node 3 ,

Consider the path **(1,3) :**

change all entries of row 1 and column 3 to ∞

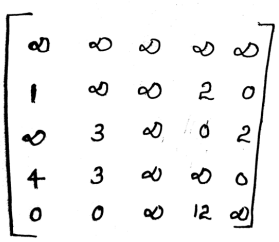
Also set (3,1) to ∞



Apply row and column reduction,

we get, row wise and column wise reduction sum as 0+11 =11

After Reducing the Minimum values reduced matrix =



cost of node 3 =C(1,3)+ r + $r^{\wedge}$ = 17+25+11 =**53**

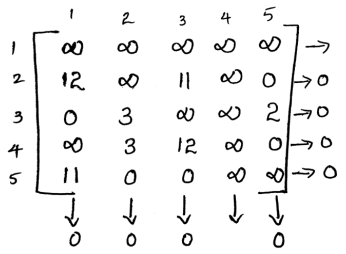
keralanotes.com

To find the cost of node 4 ,

Consider the path **(1,4) :**

change all entries of row 1 and column 4 to ∞

Also set (4,1) to ∞



Apply row and column reduction,

we get, row wise and column wise reduction sum as 0

cost of node 4 = C(1,4)+ r + $r^{\wedge}$ = 0+25+0 =25

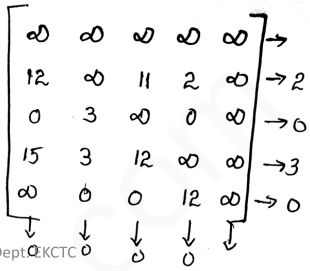
To find the cost of node 5 ,

Consider the path **(1,5) :**

change all entries of row 1 and column 5 to ∞

Also set (5,1) to ∞

Apply row and column reduction



Neethu Mathew , CSE Dept. EKCTC

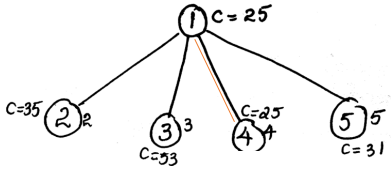
we get, row wise and column wise reduction sum =5

reduced cost matrix =

∞	∞	∞	∞	∞
10	∞	9	0	∞
0	3	∞	0	∞
12	0	9	∞	∞
∞	0	0	12	∞

cost of node 5 = C(1,5)+ r + $r^{\wedge}$ = 1+25+5 =**31**

Tree organization up to this point is as follows :

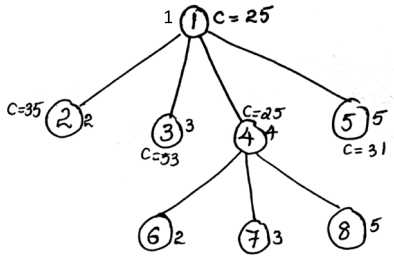


Neethu Mathew , CSE Dept. EKCTC

The cost of the path between (1,2) = 35 , (1,3) = 53 , (1,4) =25 , and (1,5) =31

Cost of the path between (1,4) is minimum

Hence the matrix obtained for path (1,4) is considered as reduced cost matrix



The new possible paths are (4,2) , (4,3) , (4,5)

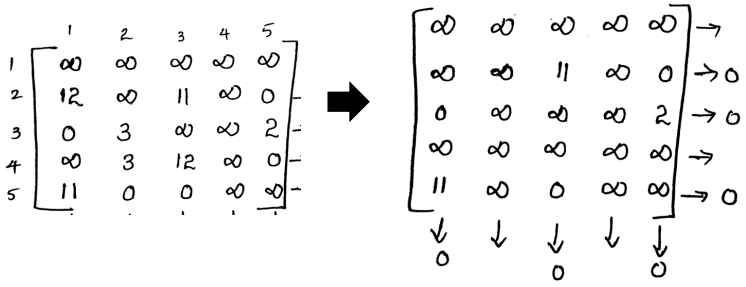
Neethu Mathew , CSE Dept. EKCTC

Consider the path **(4,2) :**

((parent—consider reduced matrix of path (1,4)))

change all entries of row 4 and column 2 to ∞

Also set (2,1) to ∞ (should not go back from 2 to 1)



Apply row and column reduction,

we get, row wise and column wise reduction sum as 0

$C(4,2) + r + r^{\wedge} = 3+25+0 = \mathbf{28}$

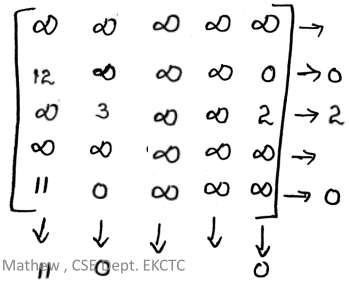
cost of vertex 4

Consider the path **(4,3) :**

change all entries of row 4 and column 3 to ∞

Also set (3,1) to ∞

Apply row and column reduction,



Neethu Mathew , CSE Dept. EKCTC

we get, row wise and column wise reduction sum as 2+11=13

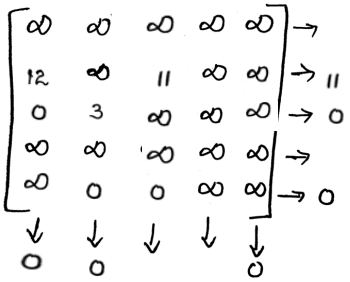
reduced matrix =

C(4,3)+ r + $r^{\wedge}$ = 12+25+13 = **50**

Consider the path **(4,5) :**

change all entries of row 4 and column 5 to ∞

Also set (5,1) to ∞



Apply row and column reduction,

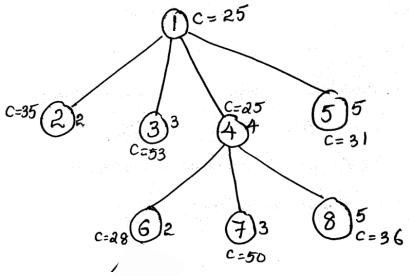
we get, row wise and column wise reduction sum as 0+11=11

reduced matrix =

C(4,5)+ r + $r^{\wedge}$ = 0+25+11 =**36**

For More Study Materials : www.keralanotes.com

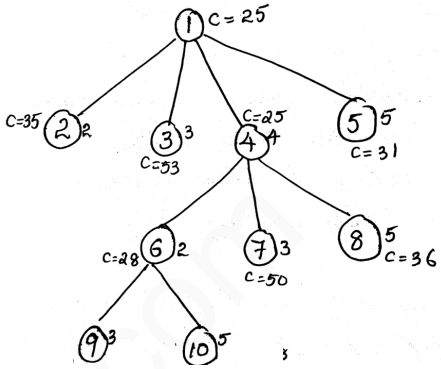
Tree organization up to this point is as follows :



Cost of the path between (4,2) = 28 , (4,3) = 50 and (4,5) =36

Cost of the path between (4,2) is minimum. Hence the matrix obtained for path (4,2) is considered as reduced cost matrix

The new possible paths are (2,3) , (2,5)



Consider the path **(2,3) :**

change all entries of row 2 and column 3 to ∞

Also set (3,1) to ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix}$$

Apply row and column reduction,

we get, row wise and column wise reduction sum as 2+11=13

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

C(2,3)+ r + $r^{\wedge}$ = 11+28+13 =**52**

keralanotes.com

For More Study Materials : www.keralanotes.com

Consider the path **(2,5) :**

change all entries of row 2 and column 5 to ∞

Also set (5,1) to ∞

Apply row and column reduction,

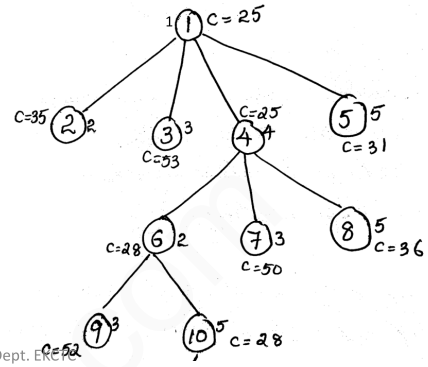
we get, row wise and column wise reduction sum 0

C(2,5)+ r + $r^{\wedge}$ = 0+28+0 =**28**

Tree organization up to this point is as follows :

Cost of the path between (2,3) = 52 and (2,5) =28

(2,5) is minimum , so new possible path is (5,3)

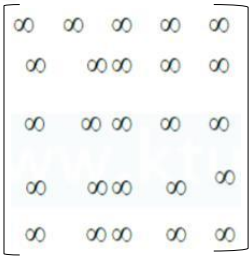


Neethu Mathew , CSE Dept. EKCTC

Consider the path **(5,3) :**

change all entries of row 5 and column 3 to ∞

Also set (3,1) to ∞



Apply row and column reduction,

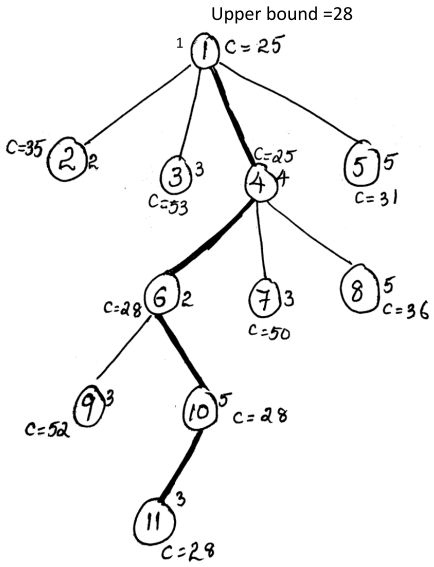
we get, row wise and column wise reduction sum 0

C(5,3)+ r + $r^{\wedge}$ = 0+28+0 =**28**

Neethu Mathew , CSE Dept. EKCTC

For More Study Materials : www.keralanotes.com

Tree organization is as follows :

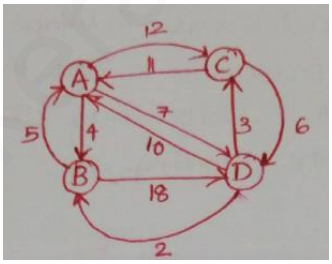


The path of traveling sale person problem is:

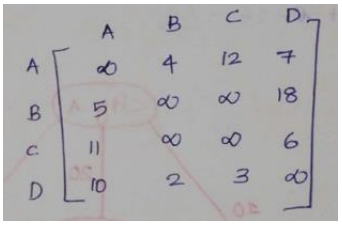
1 → 4 → 2 → 5 → 3 → 1

The minimum cost of the path is: 10 + 6 +2+ 7 + 3 = 28.

Q) To Solve Travelling Salesman problem for the following graph using Branch and Bound Technique,



first we write the cost matrix



then we reduce it and Solve step by step . And finally we get the path A-C-D-B-A With cost of 25

For More Study Materials : www.keralanotes.com

OR

Steps of TSP **Procedure for solving traveling salesman person problem**

1. Reduce given cost matrix. A matrix is reduced if every row and column is reduced. A row (column) is said to be reduced if it contain at least one zero and all-remaining entries are non-negative.
   a) Row reduction: Take the minimum element from first row, subtract it from all elements of first row, next take minimum element from the second row and subtract it from second row. Similarly apply the same procedure for all rows.
   b) Find the sum of elements, which were subtracted from rows.
   c) Apply column reductions for the matrix obtained after row reduction.

      Column reduction: Take the minimum element from first column, subtract it from all elements of first column, next take minimum element from the second column and subtract it from second column. Similarly apply the same procedure for all columns.
   d) Find the sum of elements, which were subtracted from columns.
   e) Obtain the cumulative sum of row wise reduction and column wise reduction.

      Cumulative reduced sum = Row wise reduction sum + column wise reduction sum.

      Associate the cumulative reduced sum to the starting state as lower bound and ∞ as upper bound.

Neethu Mathew , CSE Dept. EKCTC

2. Calculate the reduced cost matrix for every node R. Let A is the reduced cost matrix for node R. Let S be a child of R such that the tree edge (R, S) corresponds to including edge <i, j> in the tour. If S is not a leaf node, then the reduced cost matrix for S may be obtained as follows:
   a) Change all entries in row i and column j of A to ∞.
   b) Set A (j, 1) to ∞.
   c) Reduce all rows and columns in the resulting matrix except for rows and column containing only ∞.
   d) Find $c(S) = c(R) + A(i, j) + r$, where "r" is the total amount subtracted to reduce the matrix, *c(R) indicates the lower bound of the* i^{th} node in (i, j) path and *c(S) is called the cost function.*
3. Repeat step 2 until all nodes are visited.

Neethu Mathew , CSE Dept. EKCTC

Difference between Backtracking and Branch-and-Bound

	Backtracking	*Branch-and-Bound (BB)*
1.	It is used to find all possible solutions available to the problem.	It is used to solve optimization problem.
2.	It traverse tree by DFS (Depth First Search).	It may traverse the tree in any manner, DFS or BFS.
3.	It realizes that it has made a bad choice and undoes the last choice by backing up.	It realizes that it already has a better optimal solution that the pre-solution leads to, so it abandons that pre-solution.
4.	It search the state space tree until it found a solution.	It completely searches the state space tree to get optimal solution.
5.	It involves feasibility function.	In involves bounding function.

keralanotes.com

For More Study Materials : www.keralanotes.com