# UNIT IV

Reference Books

- T.H Coreman "Introduction to Algorithms"
- S. Sahni "Fundamentals of Computer Algorithms"

## DYNAMIC PROGRAMMING

- Introduction
- Applications of Dynamic Prog$^{n}$
  - Knapsack Problem
  - Matrix chain Multiplication
  - Longest Common subsequence
  - All pair shortest Path Problem — Floyd Warshal Algo
  - Resource Allocation Problem.

## BACKTRACKING

- Introduction
- Applications of Backtracking
  - Hamiltonian Circuit Problem.
  - Sum of Subset Problem
  - N-Queens Problem
  - Graph Coloring Problem

## BRANCH & BOUND

- Introduction
- Applications of Branch & Bound
  - Travelling Saleman Problem
  - Knapsack Problem.

# DYNAMIC PROGRAMMING

- It is a designing technique of an algorithm
- Similar to Divide & Conquer Approach.
- Both techniques solve a problem by breaking it down into several sub-Problems that can be solved recursively.
- Difference between Divide & Conquer And Dynamic Prog$^{n}$

| DIVIDE AND CONQUER | DYNAMIC PROGRAMMING |
|---|---|
| - Splits a problem into separate sub-prob Solve sub-prob & combine the result for a solut$^{n}$ to the original prob. | - Splits a problem into sub-problem, some of which are common, solves the sub-prob & combine the result. |
| - follow TOP-DOWN approach | - BOTTOM UP Approach. |
| - sub-problems are independent | - Sub-problems are dependent |
| - Simple as compared to Dynamic | - Solution can often be complex & tricky. |
| - Can be used for any kind of prob | - Generally used for optimization prob. |
| - Only one decision sequence generated | - Many Decision sequences may be generated. |

- As Greedy approach, Dynamic Prog$^{n}$ is also typically applied to the optimization problems.
  - Solutions are computed by making choices in serial forward way in Greedy Technique while in Dynamic Prog$^{n}$ solutions are computed by using bottom up way.
  - No Backtracking & revision of choices in Greedy Technique. while Dynamic Prog$^{n}$ try out many possibilities before it arrives at optimal set of choices.

- **Development of Dynamic Prog$^{n}$ Algo**
  1. Characterize structure of an optimal solut$^{n}$
  2. Recursively define the value of the optimal solut$^{n}$.
  3. Compute the value of optimal solut$^{n}$ from bottom up i.e starting with the smallest sub-problem.
  4. Construct the optimal solution for the entire problem from the computed values of smaller sub-prob.

- Dynamic Prog$^{m}$ can be thought of as being the reverse of recursion. Recursion is a top-down mechanism.
- Dynamic Prog$^{m}$ is a bottom-up mechanism. - we solve all possible small problems & then combine them to obtain solut$^{n}$ for bigger problems.

✓ Dynamic Prog$^{m}$ works when a problem has following characteristics

➤ OPTIMAL SUBSTRUCTURE

- If an optimal solution contains optimal sub-solutions, then a problem exhibits optimal substructure.
- If a problem has optimal substructure, then we can recursively define an optimal solution.

➤ OVERLAPPING SUBPROBLEMS

- When a recursive algorithm would visit the same (sub-)problem repeatedly, then problem has overlapping sub-problem.
- If a problem has overlapping sub-problems, then we can improve on a recursive implementation by computing each sub-problem only once.

✓ Dynamic Prog$^{m}$ can also be implemented using MEMOIZATION

- With memoization, we implement the algorithm recursively, but we keep track of all the sub-solutions.
- If we encounter a sub-problem that we have seen, we look up the solution.
- If we encounter a sub-problem that we have not seen, we compute it, & add it to the list of sub-solutions we have seen.
- Each subsequent time that the sub-problem is encountered the value stored in table is simply looked up & returned.

- Memoization offers the efficiency. It maintains the top-down recursive strategy.

P.T.O →

## • APPLICATIONS OF DYNAMIC PROGⁿ

1. Knapsack Problem
2. Matrix chain Multiplication (MCM)
3. Longest Common Subsequence (LCS)
4. All Pair Shortest Path < Floyd-Warshall / DFS
5. Tower of Hanoi
6. Fibonacci Sequence
7. Travelling Salesman Person
8. Optimal Cost Binary search.

## ▪ KNAPSACK PROBLEM

Refer Notes of UNIT III Pg 13-18

## ▪ MATRIX CHAIN MULTIPLICATION

- Given following matrices

  { A1, A2, A3, ..... An }

We have to perform the matrix multiplication, it can be done by a series of matrix multiplicⁿ

A1 x A2 x A3 x .....An

- Matrix multiplicⁿ operation is Associative.
- We are free to parenthesize above multiplicⁿ

ex: $A_{5\times4}$ , $B_{4\times3}$ , $C_{3\times5}$

Possible Multiplication

[(AB)C] = (5x4x3) + (5x3x5) = 135 Best coz less cost

[A(BC)] = (5x4x5) + (4x3x5) = 160

NOTE: COST of operatⁿ changes in above two multiplicatⁿ

- No: of ways for parenthesizing the matrices.
  - If we have 2 or 3 matrices to which we have to compute the multiplication then the no: of possible ways for parenthesizing these matrices is not more than 3 to 4.
  - For parenthesizing the choicing are independent, thus if there are x ways for parenthesizing the left sequence and y ways for parenthesizing the right sequence, Then total will be x.y

- **Recurrence Relation**

$$MCM(i,j) = \begin{cases} 0 & \text{if } i=j \\ \min_{i \le k < j} \{ MCM(i,k) + MCM(k+1,j) + P_{i-1} P_k P_j \} & \text{if } i<j \end{cases}$$

(MCM(i,j): Min^m no: of multipl^n required to multiply i to j matrices)

- MCM(i,n) will generate n-level, n-array (upper bound)

  total fun^c Call $= (n-1)(n-2)(n-3)\ldots\ldots 1$

  $= (n-1)!$

- Time Complexity without Dynamic Prog^m $= O(n^n)$

  Space Complexity " " " $= O(n)$

In above Recurrence Relation we find many fun^c calls are repeating so we will go to dynamic Prog^n approach

Time Complexity Using Dynamic Prog^m $= O(n^3)$

Space Complexity Using Dynamic Prog^n $= \underline{O(n^2)}$

⇓

input $n$ + Extra Space

Extra Space: ⇓ Stack $n$ , Table $n^2$

$n + n + n^2 = n^2$

- **AIGO**

```
{
   Study from COREMANBOOK
}
```

- Given Two Matrices $A_{2\times3}$, $B_{3\times4}$

Result of multiplication of A * B is stored in Other matrix

$C_{2\times4} = A_{2\times3} * B_{3\times4}$

No: of element in C = 2×4 = 8

No: of Multiplicat$^{n}$ required in C = 2×4×3 = 24

- Given Three Matrices $A_{3\times4}$, $B_{4\times2}$, $C_{2\times5}$

Result of multiplicat$^{n}$ of A*B*C is stored in matrix D

Now multiplicat$^{n}$ can happen in two ways

We have to find which way is best.

$D_{3\times5} = ((A*B)*C)$ or $(A*(B*C))$

Cost of $(A*B)_{3\times2} = 4\times2\times3 = 24$

Cost of $((AB)*C)_{3\times5} = 3\times5\times2 = 30$

No: of Multiplications = 24 + 30 = 54

Cost of $(B*C)_{4\times5} = 4\times5\times2 = 40$

Cost of $(A*(B*C))_{3\times5} = 3\times5\times4 = 60$

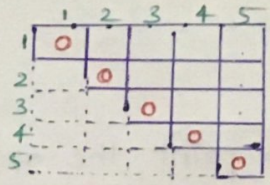
No: of Multiplicat$^{n}$ = 40 + 60 = 100

We will take this way of multiplicat$^{n}$ i.e ((A*B)*C) because less Multiplicat$^{n}$, Less time, Less cost.

Q1. Given matrices A1, A2, A3, A4, A5 of size 4x10, 10x3, 3x12, 12x20 and 20x7 respectively.
The sequence $P_0, P_1, P_2 \dots P_5$ is given as 4, 10, 3, 12, 20, 7
Find optimal Parenthesization for multiplying the matrices.

Sol^n We need to compute $M[i,j]$ ; $0 \le i$ and $j \le 5$

We know $M[i,j] = 0 \quad \forall\, i = j$




- First we compute the optimal solution for the products of two matrices.
Dynamic Prog^n will solve all possibility of multiplying two matrices & then find the optimal solution.

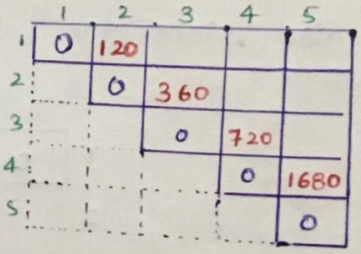
$M[1,2] = P_0 P_1 P_2 = 4 \times 10 \times 3 = 120$

$M[2,3] = P_1 P_2 P_3 = 10 \times 3 \times 12 = 360$

$M[3,4] = P_2 P_3 P_4 = 3 \times 12 \times 20 = 720$

$M[4,5] = P_3 P_4 P_5 = 12 \times 20 \times 7 = 1680$

↳ gives cost of multiplying A4, A5 matrices

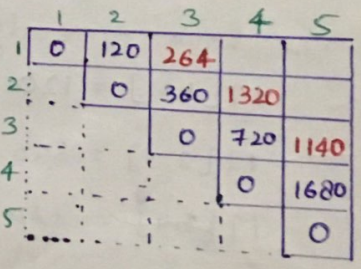



- Now Compute optimal solution for the products of three matrices

$M[1,3] = \min \begin{cases} M[1,2] + M[3,3] + P_0 P_2 P_3 = \mathbf{264} \\ M[1,1] + M[2,3] + P_0 P_1 P_3 = 840 \end{cases}$

$M[2,4] = \min \begin{cases} M[2,3] + M[4,4] + P_1 P_3 P_4 = 2760 \\ M[2,2] + M[3,4] + P_1 P_2 P_4 = \mathbf{1320} \end{cases}$

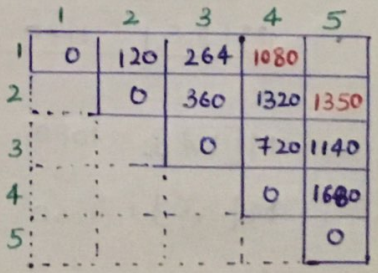
$M[3,5] = \min \begin{cases} M[3,4] + M[5,5] + P_2 P_4 P_5 = \mathbf{1140} \\ M[3,3] + M[4,5] + P_2 P_3 P_5 = 1932 \end{cases}$




- Now Compute optimal solution for the products of Four matrices.

$M[1,4] = \min \begin{cases} M[1,3] + M[4,4] + P_0 P_3 P_4 = 1224 \\ M[1,2] + M[3,4] + P_0 P_2 P_4 = \mathbf{1080} \\ M[1,1] + M[2,4] + P_0 P_1 P_4 = 2120 \end{cases}$

$M[2,5] = \min \begin{cases} M[2,4] + M[5,5] + P_1 P_4 P_5 = 2720 \\ M[2,3] + M[4,5] + P_1 P_3 P_5 = 2880 \\ M[2,2] + M[3,5] + P_1 P_2 P_5 = \mathbf{1350} \end{cases}$




P.T.O

- Compute optimal solution for the product of Five matrices.

$$M[1,5] = \min \begin{cases} M[1,4] + M[5,5] + P_0 P_4 P_5 = 1544 \\ M[1,3] + M[4,5] + P_0 P_3 P_5 = 2016 \\ M[1,2] + M[3,5] + P_0 P_2 P_5 = \mathbf{1344} \\ M[1,1] + M[2,5] + P_0 P_1 P_5 = 1630 \end{cases}$$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120 | 264 | 1080 | 1344 |
| 2 | | 0 | 360 | 1320 | 1350 |
| 3 | | | 0 | 720 | 1140 |
| 4 | | | | 0 | 1680 |
| 5 | | | | | 0 |

- To print the optimal parenthesization, we use the procedure PRINT-OPTIMAL-PARENS(s, i, j)

- For each time we find the optimal value for M[i,j] we also store the value of **K** that we used

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 120/**1** | 264/**2** | 1080/**2** | 1344/**2** |
| 2 | | 0 | 360/**2** | 1320/**2** | 1350/**2** |
| 3 | | | 0 | 720/3 | 1140/4 |
| 4 | | | | 0 | 1680/4 |
| 5 | | | | | 0 |

- We have already calculated below costs of multiplying matrices

M[1,2] = 120 K = 1

M[2,3] = 360 K = 2

M[3,4] = 720 K = 3

M[4,5] = 1680 K = 4

M[1,3] = 264 K = 2

M[2,4] = 1320 K = 2

M[3,5] = 1140 K = 4

M[1,4] = 1080 K = 2

M[2,5] = 1350 K = 2

M[1,5] = 1344 K = 2

- The k value for the solut$^n$ is 2, so we have

((A1*A2)(A3 A4 A5))

- The optimal solution for the second half comes from entry M[3,5].

The k value of k is 4, so we have

((A1 A2)(A3 A4) A5)).

∴ Optimal solution of give Matrix Chain Multiplication is

((A1*A2)*((A3*A4)*A5))

# LONGEST COMMON SUBSEQUENCE

- A subsequence of given sequence is just the given sequence with some elements left out.
- Given two sequence X & Y, we say that a sequence Z is a common sequence of X & Y if Z is a subsequence of both

X = (A, B, B, A, B, B)  Y = (B, A, A, B, A, A)

Subsequence Length = 2

Z1 = (A, B)

Z2 = (B, A)

Subsequence Length = 3

Z3 = (A, B, A)

Z4 = (A, B, B)

Z5 = (B, B, A)

Z6 = (B, A, B)

Find Longest Common Subsequence to X & Y.

X = (A, B, A, B, A, B)

Y = (B, A, A, B, A, A)

Sol$^{n}$

$Z_0$ = ( )  // subsequence = 0

$Z_1$ = (A)  // subsequence = 1

$Z_2$ = (A, B), (B, A), (A, A), (B, B)  // subsequence = 2

$Z_3$ = (A, B, A), (B, A, B), (B, A, A)  // subsequence = 3

$Z_4$ = (B, A, B, A), (A, B, A, A)  // subsequence = 4

We get Longest Common subsequence (LCS) = 4

How to find LCS of two sequence X and Y.

LCS(m,n) : The length of two sequence X & Y is given
X contains m symbols.
Y contains n symbols.

ex: LCS(4,4)

1 + LCS(3, 3)
⇓
1 + LCS(2,2)
⇓
1 + LCS(1,1)
⇓
1 + LCS(0,0)
⇓
0

$$LCS(m,n) = 0 \quad \text{if} \quad m = n = 0$$

DMP

## • Recurrence Relation

$$LCS(m,n) = \begin{cases} 0 & ; m=0 \text{ or } n=0 \\ 1+LCS(m-1,n-1) & ; x_m == y_n \\ \max(LCS(m,n-1), LCS(m-1,n)) & ; x_m \neq y_n \end{cases}$$

## • ALGO

```
LCS(m, n)
{   if(m==0 || n==0)
      then return 0

    else
    if(X[m]==Y[n])
      then return(1+LCS(m-1, n-1))   // Print "↖"

    else
    { a = LCS(m, n-1)                // if(a≤b) Print "↑"
      b = LCS(m-1, n)                // if(a>b) Print "←"
      c = max(a, b);
      return c
    }
}
```

## • Time Complexity

| | Time Complexity | Space Complexity |
|---|---|---|
| Using Greedy | $= O(2^{m+n})$ | $O(m+n)$ |
| DMP Using Dynamic Prog$^n$ | $= O(m*n)$ | $O(m*n)$ |

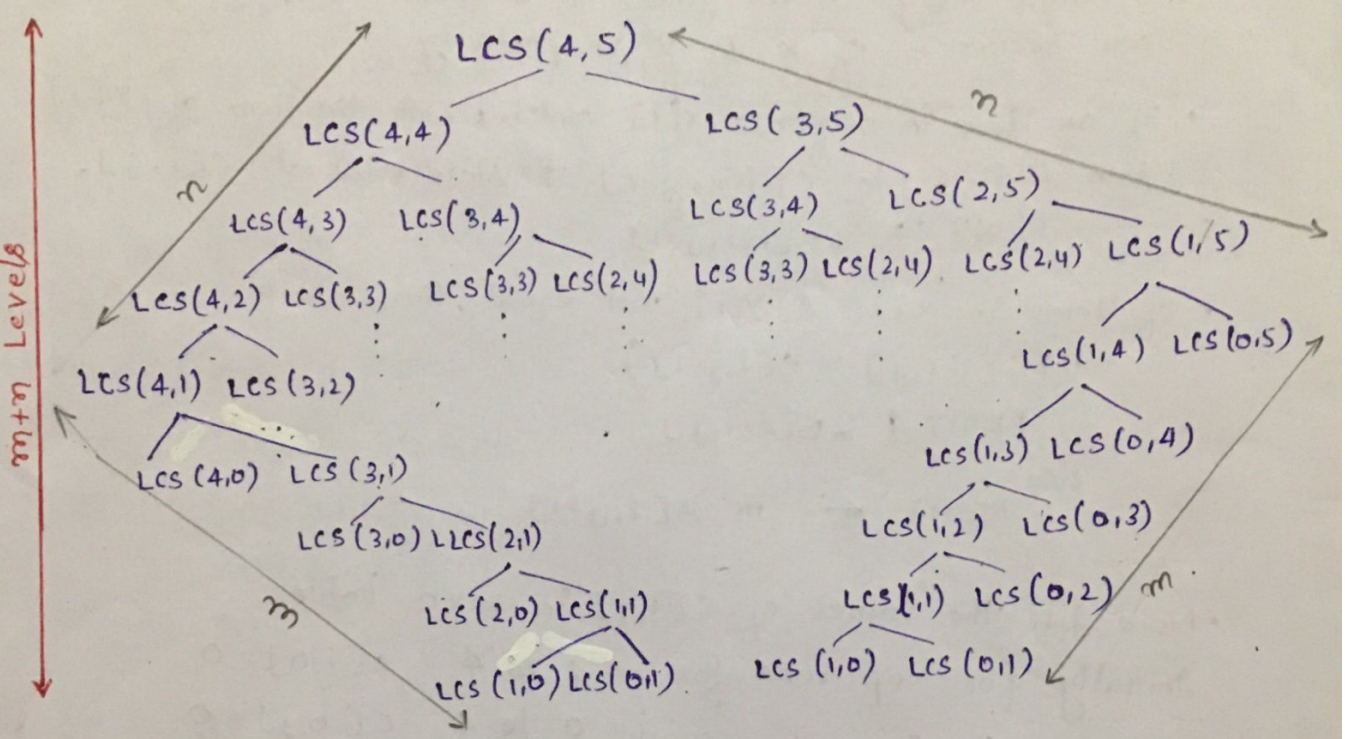
# • Recursive Tree

Let two sequence be $X = (A, A, A, A)$ ; $Y = (B, B, B, B, B)$

Length of $X = m = 4$

Length of $Y = n = 5$

Recursive Function calls for LCS(4,5) is given below




In funct$^n$ LCS(m,n), we see

(m+n) Level Complete Binary Tree (upper Bound)

* No: of Nodes $= 2^{level} = 2^{m+n}$
* No: of fun$^l$ Calls = No: of Nodes in Recursive tree $= 2^{m+n}$

Dynamic Prog$^m$ uses a table to store values so that it solves only distinct fun$^l$ call.

ex: Using Greedy LCS(4,5) has $2^{4+5} = 2^9$ fun$^l$ calls

Using Dynamic LCS(4,5) has $4 * 5 = 20$ fun$^l$ calls.

Q1. Given two sequence $X[1 \ldots n]$ & $Y[1 \ldots n]$. Find Longest Common Subseq. to both. $X : \langle A, B, B, A \rangle$ ($x_1, x_2, x_3, x_4$) $Y : \langle B, A, A, B \rangle$ ($y_1, y_2, y_3, y_4$)

Sol$^n$
- We have length $[X] = m = 4$ & Length $[Y] = n = 4$
- There could be more than one subsequence of longest length, the algo will get all of them.
- Dynamic Prog$^n$ stores values in a table $c[i, j]$ where $i$ & $j$ are sequence of $x$ & $y$ respectively.
- If an item in sequence $x[i]$ matches with the item of $y[j]$ then ADD 1 to $c[i-1, j-1]$ & store value at $c[i, j]$.

  PRINT "↖" in $B[i-1, j-1]$
- If items in $x[i]$ & $y[i]$ not equal, then check

  if $(c[i-1, j] \geq c[i, j-1])$

  PRINT "↑" in $B[i-1, j]$

  else

  PRINT "←" in $B[i, j-1]$
- Now fill the values of $c[i, j]$ in $m \times n$ table.

  Initially for loop runs for $i = 1$ to 4 $c[i, 0] = 0$

  " " " " $j = 1$ to 4 $c[0, j] = 0$

| i ↓ | j → | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| | | $y_j$ | B | A | A | B |
| 0 | $x_i$ | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | | | | |
| 2 | B | 0 | | | | |
| 3 | B | 0 | | | | |
| 4 | A | 0 | | | | |

- for $i = 1$ $j = 1$

  We get $x[1] \neq y[1]$ i.e $A \neq B$

  $c[i-1, j] = c[0, 1] = 0$

  $c[i, j-1] = c[1, 0] = 0$ } $c[1, 1] = 0$

  PRINT ↑ in $B[1, 1]$

  $i = 1$ $j = 2$

  $x[1] = y[2]$ i.e $A = A$

  $c[1, 2] = c[1-1, 2-1] + 1$

  $= c[0, 1] + 1$

  $= 0 + 1 = 1$

  PRINT "↖" in $B[1, 2]$

P. T. O.

$i = 1 \quad j = 3$

$X[1] = Y[3]$ ie $A = A$

$C[1,3] = C[0,2] + 1$

$= 0 + 1 = 1$

PRINT "↖"

$i = 1 \quad j = 4$

$X[1] \neq Y[4]$ ie $A \neq B$

$C[0,4] = 0$
$C[1,3] = 1$ } Max = $C[1,4] = 1$

PRINT ↑

| i j → | | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| | | $Y_j$ | B | A | A | B |
| 0 | $X_i$ | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | ↑ 0 | ↖ 1 | ↖ 1 | ↑ 1 |
| 2 | B | 0 | | | | |
| 3 | B | 0 | | | | |
| 4 | A | 0 | | | | |

- Now $i = 2 \quad j = 1$

$X[2] = Y[1]$ ie $B = B$

$C[2,1] = C[1,0] + 1$

$= 0 + 1 = 1$

PRINT ↖

$i = 2 \quad j = 2$

$X[2] \neq Y[2]$ ie $B \neq A$

$C[1,2] = 1$
$C[2,1] = 1$ } Max = $C[2,2] = 1$

PRINT ↑

$i = 2 \quad j = 3$

$X[2] \neq Y[3]$ ie $B \neq A$

$C[1,3] = 1$
$C[2,2] = 1$ } Max = $C[2,3] = 1$

PRINT ↑

$i = 2 \quad j = 4$

$X[2] = Y[4]$ ie $B = B$

$C[2,4] = C[1,3] + 1$

$= 1 + 1 = 2$

PRINT ↖

| i j → | | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| | | $Y_j$ | B | A | A | B |
| 0 | $X_i$ | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | ↑ 0 | ↖ 1 | ↖ 1 | ↑ 1 |
| 2 | B | 0 | ↖ 1 | ↑ 1 | ↑ 1 | ↖ 2 |
| 3 | B | 0 | | | | |
| 4 | A | 0 | | | | |

P.T.O

- Now $i=3$ $j=1$

$X[3] = Y[1]$ ie $B=B$

$C[3,1] = [2,0] + 1$

$= 0 + 1 = 1$

PRINT "↖"

$i=3$ $j=2$

$X[3] \neq Y[2]$ ie $B \neq A$

$C[2,2] = 1$
$C[3,1] = 1$
} Max = $C[3,2] = 1$

PRINT ↑

$i=3$ $j=3$

$X[3] \neq Y[3]$ ie $B \neq A$

$C[2,3] = 1$
$C[3,2] = 1$
} Max = $C[3,3] = 1$

PRINT ↑

$i=3$ $j=4$

$X[3] = Y[4]$ ie $B = B$

$C[3,4] = C[2,3] + 1$

$= 1 + 1 = 2$

PRINT ↖

| i ↓ / j → | | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| | $Y_j$ | | B | A | A | B |
| 0 | $X_i$ | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | ↑ 0 | ↖ 1 | ↖ 1 | ↑ 1 |
| 2 | B | 0 | ↖ 1 | ↑ 1 | ↑ 1 | ↖ 2 |
| 3 | B | 0 | ↖ 1 | ↑ 1 | ↑ 1 | ↖ 2 |
| 4 | A | 0 | | | | |

Similarly, we fill all the values of $C[i,j]$ & we get

| i ↓ / j → | | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| | $Y_j$ | | B | (A) | A | (B) |
| 0 | $X_i$ | 0 | 0 | 0 | 0 | 0 |
| 1 | (A) | 0 | ↑ 0 | ↖ (1) | ↖ 1 | ← 1 |
| 2 | B | 0 | ↖ 1 | ↑ 1 | ↑ 1 | ↖ 2 |
| 3 | (B) | 0 | ↖ 1 | ↑ 1 | ↑ 1 | ↖ (2) |
| 4 | A | 0 | ↑ 1 | ↖ 2 | ↖ 2 | ↑ 2 |

We see entry 2 in the table $C[4,4]$ is the length of the LCS $< A, B >$
$< B, B >$
$< A, A >$
$< B, A >$

NOTE: Each "↖" on path corresponds to an entry for which $x_i = y_j$ is a member of LCS.

# ALL PAIR SHORTEST PATH

- The Problem is to find shortest distance between every pair of vertices in a given edge weighted directed Graph.
- Floyd-Warshall is one of the solution to this problem.

## FLOYD WARSHALL ALGO

- Works for +ve, -ve and unweighted edge
- It is Graph Analysis algo for finding shortest path between all pairs of vertices
- Time Complexity = $\theta(V^3)$
- The algo considers the intermediate vertices of a shortest path, where an intermediate vertex of a simple path $P = (V_1\ V_2\ V_3 \dots V_m)$ is any vertex of P other than $V_1$ or $V_m$ i.e any vertex in set $\{V_2, V_3 \dots V_{m-1}\}$
- ALGO

{
1.
2.
:
7.
}

Study ALGO from CORETHAN BOOK

- Recursive Fun$^c$

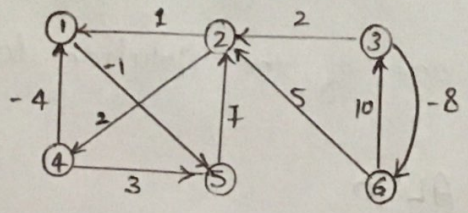
$$(D_{ij})^k = \begin{cases} W_{ij} & ; K=0 \\ \min\left[(D_{ij})^{K-1}, (D_{ik})^{K-1} + (D_{kj})^{K-1}\right] & ; K \geq 1 \end{cases}$$

✓ The recursive fun$^c$ will generate n - level Complete Binary Tre
⇒ No: of Node = No of fun$^c$ calls = $2^n$ T.C w/o dynamic

✓ Total No: of fun$^c$ calls = $n * n * n = n^3$ using dynamic

✓ Space Complexity using Dynamic = $O(n^3)$ which can be reduced to $O(n^2)$ by doing some modificat$^n$.

Q1. Apply Floyd Warshall algo for constructing shortest path. Show $(D)^k$ that results each iteration.




Sol$^n$ We know $(D_{ij})^k = \min\{(D_{ij})^{k-1}, (D_{ik})^{k-1} + (D_{kj})^{k-1}\}$

$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

// This matrix shows direct distance between all pair of vertices

$$D^1 = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix}$$

↓ It gives distance b/w all pair of vertices via vertex (1)

$$D^2 = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}$$

↓ gives distance b/w all pair of vertices via vertex (2)

$$D^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{bmatrix} \end{array}$$

$$D^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix} \end{array}$$

$$D^5 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix} \end{array}$$

$$D^6 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix} \end{array}$$

To find the distance between all pair of vertices via Vertex 1, we have

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

$$= \min\{ \infty , 1+\infty \}$$

$$= \infty$$

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

$$= \min\{ 2, 1+\infty \}$$

$$= 2$$

$$D^1(2,5) = \min\{ D^0(2,5) , D^0(2,1) + D^0(1,5) \}$$

$$= \min\{ \infty , 1 + (-1) \}$$

$$= 0$$

$$D^1(2,6) = \min\{ D^0(2,6) , D^0(2,1) + D^0(1,6) \}$$

$$= \min\{ \infty , 1+\infty \}$$

$$= \infty$$

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

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

$$= 2$$

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

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

$$= \infty$$

P.T.O →

$D^1(3,5) = \min\{D^0(3,5), D^0(3,1)+D^0(1,5)\}$
$= \min\{\infty, \infty+(-1)\}$
$= \infty$

$D^1(3,6) = \min\{D^0(3,6), D^0(3,1)+D^0(1,6)\}$
$= \min\{-8, \infty+\infty\}$
$= -8$

$D^1(4,2) = \min\{D^0(4,2), D^0(4,1)+D^0(1,2)\}$
$= \min\{\infty, -4+\infty\}$
$= \infty$

$D^1(4,3) = \min\{D^0(4,3), D^0(4,1)+D^0(1,4)\}$
$= \min\{\infty, -4+\infty\}$
$= \infty$

$D^1(4,5) = \min\{D^0(4,5), D^0(4,1)+D^0(1,5)\}$
$= \min\{3, -4+(-1)\}$
$= -5$

$D^1(4,6) = \min\{D^0(4,6), D^0(4,1)+D^0(1,6)\} = \min\{\infty, -4+\infty\} = \infty$
$D^1(5,2) = \min\{D^0(5,2), D^0(5,1)+D^0(1,2)\} = \min\{7, \infty+\infty\} = 7$
$D^1(5,3) = \min\{D^0(5,3), D^0(5,1)+D^0(1,3)\} = \min\{\infty, \infty+\infty\} = \infty$
$D^1(5,4) = \min\{D^0(5,4), D^0(5,1)+D^0(1,4)\} = \min\{\infty, \infty+\infty\} = \infty$
$D^1(5,6) = \min\{D^0(5,6), D^0(5,1)+D^0(1,6)\} = \min\{\infty, \infty+\infty\} = \infty$
$D^1(6,2) = \min\{D^0(6,2), D^0(6,1)+D^0(1,2)\} = \min\{5, \infty+\infty\} = 5$
$D^1(6,3) = \min\{D^0(6,3), D^0(6,1)+D^0(1,3)\} = \min\{10, \infty+\infty\} = 10$
$D^1(6,4) = \min\{D^0(6,4), D^0(6,1)+D^0(1,4)\} = \min\{\infty, \infty+\infty\} = \infty$
$D^1(6,5) = \min\{D^0(6,5), D^0(6,1)+D^0(1,5)\} = \min\{\infty, \infty+(-1)\} = \infty$

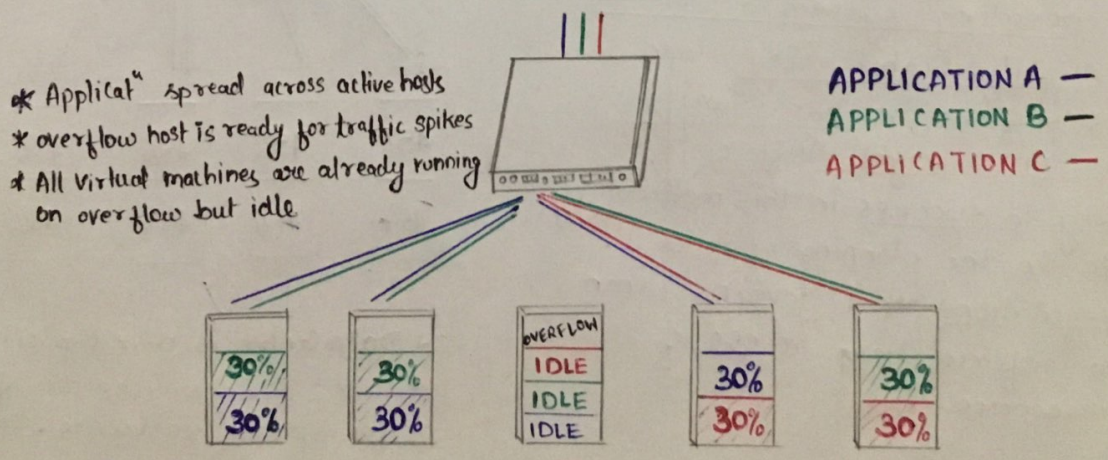
Similarly find distance b/w all pair of vertices via vertex 2, 3, 4, 5 and 6.

# RESOURCE ALLOCATION PROBLEM

- Process of allocating scarce resource among the various Projects or business unit.
- There are a no: of approaches to solve resource allocat" prob.
  ex: Resources can be allocated using manual approach. or an algorithmic approach.
- There are two types of dynamic resource allocation

| ① Overflow | ② Tiering |
|---|---|
| - All resources are pre-provisioned and ready | - All resource are pre-provisioned and ready |
| - We have standby computer resource ready for on-demand access | - We dynamically reprioritize an application's access to the computer resources. |

① Suppose we have virtual Application and services provisioned across hardware by an application delivery controller. The are five local traffic manager. In middle we have an overflow host that is copy of all applications and services provisioned and running but in idle state.
We can see application A's resource requirements have increased pushing the utilization of some host upto 75%.
The coordinating device commits application A to access its instances running on overflow host, the service utilization continues to increase but with no risk to end user experience.



- To summarize there was no change in the infrastructure, no IP Addressing, Hostname allocation or introduction of Operating System and services.

Consequently Low risk and we achieve relatively low gain as far as dynamic infrastrure is concerned due to the need to have overflow host but still an improvement over no consolidation.

(2) In Tiering, Suppose we have a data center that has been heavily consolidated and is now dangerously near capacity at peak times.

The problem is that we have low priority services getting equal share of computer resources as high priority services.

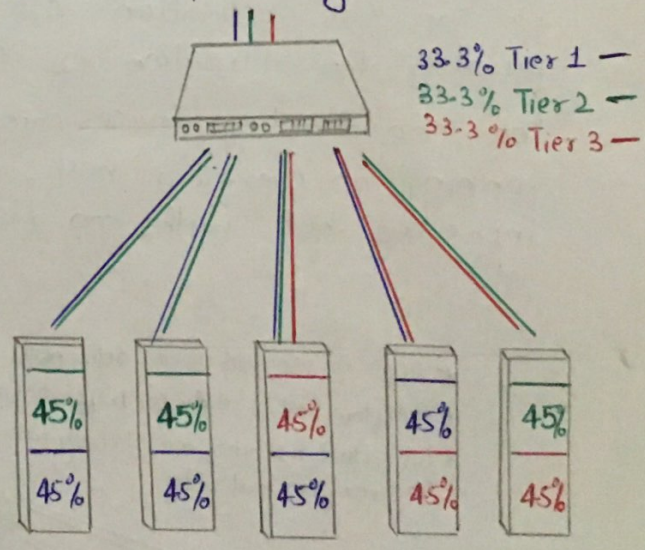
Tier 1, 2 & 3 all have equal share of resources.

With dynamic tiered approach we the coordinating device that enforce reprioritization of each tier of service, first by instructing the application delivery controller to reduce the connectivity to tier 2 & 3 services, second by instructing hypervisors to reduce tier 2 & 3 acess to underlying computer resource.

As a result tier 1 apps and services are able to flex their requirements by sacrificing the available resource to the lower tiers. A customer facing web service might be granted preference over development & test service's priority i.e end user experience is protected.

To summarize, there was no change in the infrastructure, no IP addresing, host names, operating system or services. Slightly higher risks than overflow as low priority services might suffer.

Key to success in this methodology is in the planning of rules & regulations that governs when to enforce tier access of resources.




* Data Centre is near capacity
* Tier 2 & 3 services are fighting for sparse resources with Tier 1.

# Resource Allocation Problem Using Multistage Graph

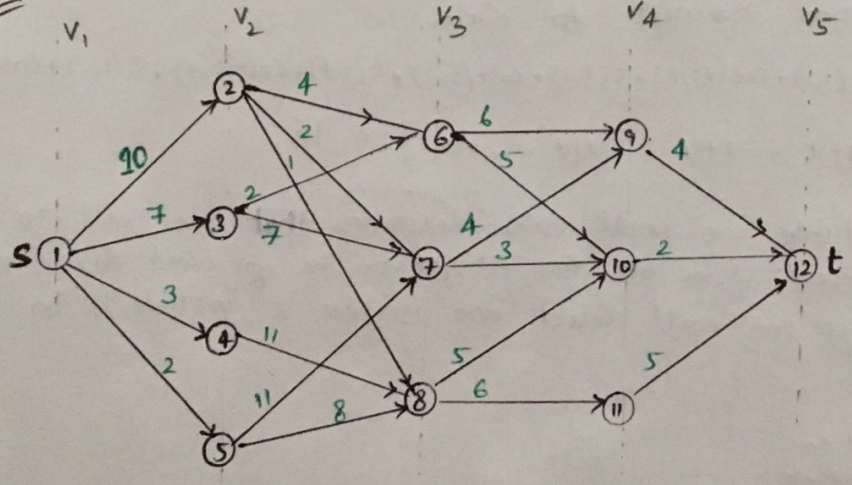
- A multistage graph $G=(V,E)$ is a directed Graph where the vertices are partitioned into k no: of disjoint subset $S=\{S_1, S_2, \ldots S_k\}$ ; $k>1$ such that edge $(u,v)$ is in E, then $u \in S_i$ & $v \in S_{i+1}$ for some subset in the partition and $|S_1| = |S_k| = 1$.
- Vertex $s \in S_1$ is called Source Vertex.
  Vertex $t \in S_k$ is called Sink Vertex.
- G is assumed to be a weighted graph.
  Cost of an edge $(i,j)$ is represented by $c(i,j)$
  Cost of path from source 's' to sink 't' = Sum of cost of each edge in this path
- Multistage graph problem is to find the path with minimum cost from source 's' to the sink t.

$$cost(i,j) = \min \{ c(j,l) + cost(i+1, l) \}$$

i → stage, j → node/vertex
$c(j,l)$ → cost of an edge b/w vertex j & l
$cost(i+1, l)$ → cost of vertex l at stage i+1

- Multistage Graph is used for resource allocation.
- It holds principle of optimality.

Q. find optimal solution for a given graph.




Sol" for finding the cost start from sink vertex because it makes calculation easy. We use tabulation method to fill data & find cost.

P. T. O →

| | Stage 1 | Stage 2 vertices | | | | Stage 3 vertices | | | Stage 4 vertices | | | Stage 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| V | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Cost | 16 | 7 | 9 | 18 | 15 | 7 | 5 | 7 | 4 | 2 | 5 | 0 |
| d | 3 | 7 | 6 | 8 | 8 | 10 | 10 | 10 | 12 | 12 | 12 | 12 |

- cost of 5th stage Vertex 12 i.e $Cost(5,12) = 0$
- find cost of each vertex in stage 4

  $cost(4,9) = 4$ $cost(4,10) = 2$ $cost(4,11) = 5$

  By taking vertices 9, 10 & 11 we are reaching to vertex 12
- Now find cost of each vertex in stage 3

  $cost(3,6) = \min\{c(6,9) + cost(4,9), c(6,10) + cost(4,10)\}$ ↳ vertex 10 has given us min^m result so d = 10

  (stage, vertex)

  $= \min\{6+4, 5+2\} = 7$

  $cost(3,7) = \min\{c(7,9) + cost(4,9), c(7,10) + cost(4,10)\} = \min\{4+4, 3+2\} = 5$

  $cost(3,8) = \min\{c(8,10) + cost(4,10), c(8,11) + cost(4,11)\} = \min\{5+2, 6+5\} = 7$
- Now find cost of each vertex in stage 2

  $cost(2,2) = \min\{c(2,6) + cost(3,6), c(2,7) + cost(3,7), c(2,8) + cost(3,8)\} = \min\{4+7, 2+5, 1+7\} = 7$

  $cost(2,3) = \min\{c(3,6) + cost(3,6), c(3,7) + cost(3,7)\} = \min\{2+7, 7+5\} = 9$

  $cost(2,4) = \min\{c(4,8) + cost(3,8)\} = 11 + 7 = 18$

  $cost(2,5) = \min\{c(5,7) + cost(3,7), c(5,8) + cost(3,8)\} = \min\{11+5, 8+7\} = 15$
- find minimum cost for starting vertex i.e source which will give us minimum cost from source to sink.

  $cost(1,1) = \min\{c(1,2) + cost(2,2), c(1,3) + cost(2,3), c(1,4) + cost(2,4), c(1,5) + cost(2,5)\}$

  $= \min\{10+7, 7+9, 3+18, 2+15\} = 16$
- After filling the table we will take sequence of decisions i.e we will go to vertex l from vertex v in all the stages in the forward direction. from all the stages we will select one vertex & include it in our optimal path cost.

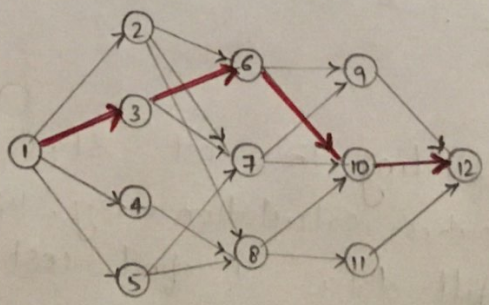
  $d(1,1) = 3$ ← we got this from table

  $d(2,3) = 6$

  $d(3,6) = 10$

  $d(4,10) = 12$

  (Stage, vertex)



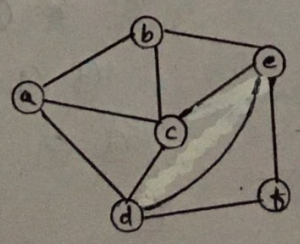
# BACKTRACKING

- It is a design technique of algorithm.
- It try out each possibility until finds the right one.
- Backtracking is a methodical way of trying out various sequences of decisions, until we find one that "works".
- Applications of Backtracking
  1. Hamiltonian Circuit Prob.
  2. N-Queen Prob
  3. Sum of Subset Prob
  4. Graph Coloring.

## HAMILTONIAN CIRCUIT PROBLEM

- Given a graph $G=(V,E)$, we have to find the Hamiltonian Circuit using Backtracking approach.
- We start our search from any arbitrary vertex, say 'A'. This vertex A becomes the root of our implicit tree. The first element of our partial solution is the first intermediate vertex of the Hamiltonian cycle that is to be constructed.
- The next adjacent vertex is selected on the basis of alphabetical or numerical order.
- If at any stage any arbitrary vertex makes a cycle with any vertex other than vertex A then we say that dead end is reached.
- In this case we backtrack one step, and again the search begins by selecting another vertex & backtrack the element from the partial solution must be removed.
- The search using backtracking is successful if a Hamiltonian cycle is obtained.

ex:



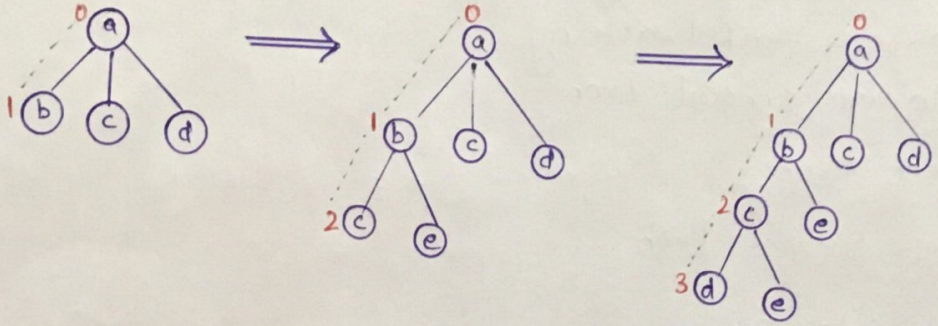

Consider a graph $G=(V,E)$ in fig shown we have to find Hamiltonian circuit using Backtracking method.

P.T.O.

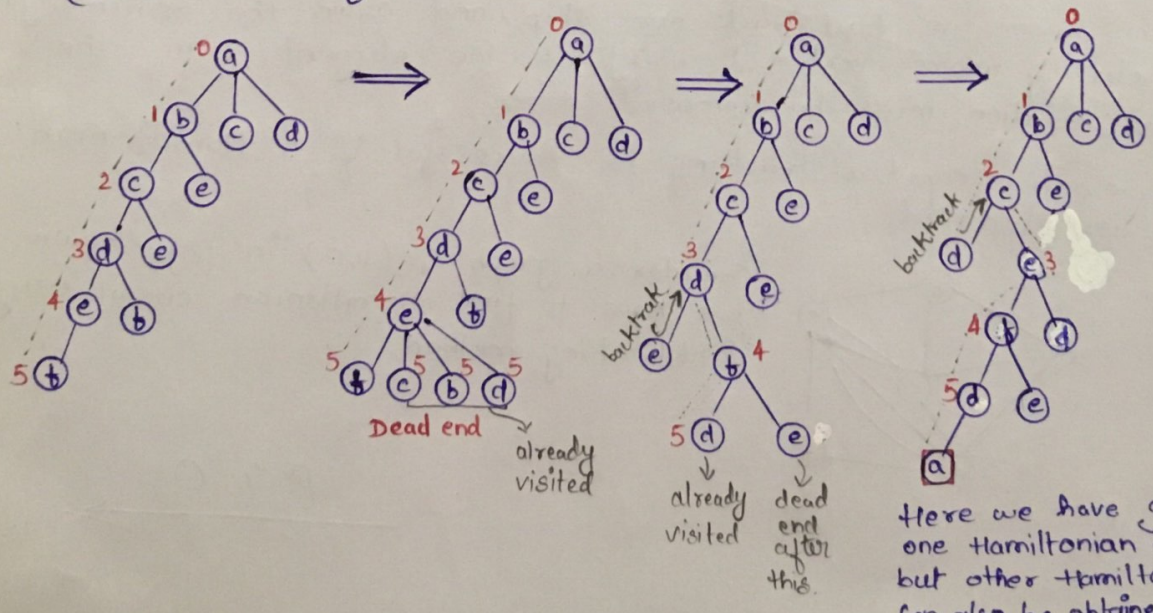
- we start our search with vertex 'a'. this vertex becomes the root of our implicit tree.

(a)

- we choose vertex 'b' adjacent to a as it comes first in lexographical order (b, c, d). Next we select 'c' adjacent to 'b'. Then select 'd' adjacent to c



- Now select 'e' adjacent to d. Next we select 'f' adjacent to e. The vertex adjacent to 'f' are 'd' & 'e' but they have already visited. Thus, we get dead end & we back track one step & remove the vertex 'f' from partial solution.
From back tracking, the vertex adjacent to 'e' are b, c, d, f. We have already checked 'f' & b, c, d are already visited.
So again we backtrack one step.
Now vertex adjacent to d are e, f from which 'e' has already been checked & adjacent of 'f' are d & e. If vertex 'e' visited again then we get dead state. So again backtrack one step.
Now adjacent to c is 'e'. Here we get Hamiltonian cycle as all the vertex other than start vertex is visited only once.
(a-b-c-e-f-d-a).




Here we have generated one Hamiltonian circuit but other Hamiltonian circuit can also be obtained by considering other vertex.

# SUM OF SUBSET PROBLEM

- In the sum of subset problem we have to find a subset $s'$ of the given set $S = \langle S_1, S_2, S_3 \ldots S_n \rangle$ where the elements of the set $S$ are $n$ positive integers in such a manner that $s' \in S$ and sum of the elements of subset $s'$ is equal to some positive integer $X$.

- Sum of subset problem can be solved by using the backtracking approach. In this implicit tree is a binary tree. The root of the tree is selected in such a way that it represents that no decision is yet taken on any input.
We assume that the elements of the given set are arranged in an increasing order.

$$S1 \leq S2 \leq S3 \ldots \leq Sn$$

Left child of the root node indicates that we have to include '$S_i$' from the set $S'$ & the Right child of the root node indicates that we have to exclude '$S_i$'.

Each node stores the sum of the partial solution elements.
If at any stage the no: equals to 'X' then search is successful and terminates.
The dead end in the tree occurs only when either of the two inequalities exists:

- The sum of $s'$ is too large i.e

$$s' + s_i + 1 > X$$

- The sum of $s'$ is too small i.e

$$s' + \sum_{j=i+1}^{n} s_j < X$$

P.T.O.

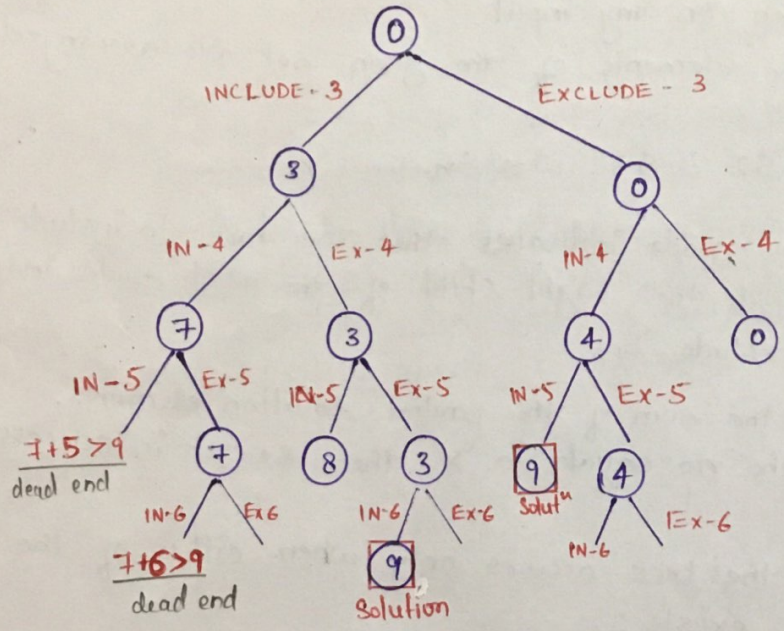
Q. Given a set $S = \langle 3, 4, 5, 6 \rangle$ & $X = 9$
= obtain the sum of set using Backtracking Approach.

Sol$^n$ Initially $S = \langle 3, 4, 5, 6 \rangle$

$X = 9$

$S' = \phi$

The implicit Binary Tree for "sum of subset problem is shown




The no: inside a node is the sum of partial solut$^n$ element at particular level.

Thus, if our partial solut$^n$ elements sum is equal to the positive no: 'X' then at that time search will terminate or it continues if all the possible solutions need to be obtained.

# N-QUEEN PROBLEM

- N-Queen Problem is to place n-queens in such a manner on **N x N** chessboard that no two Queens attack each other by being in the same row, column or diagonal.

- NOTE : for $n = 1$, the problem has a trivial solution

  $n = 2$ or $3$, No solution exists

  So we will consider 4-Queens problem & then generalize it to n-Queens problem.

- Given 4x4 chessboard and number of rows & column of the chessboard 1 through 4.

  Since we have to place 4 Queens such as q1, q2, q3 & q4 on a chessboard such that no two queens attack each other. In such condition each queen must be placed on different row i.e we place queen 'i' in row 'i'.

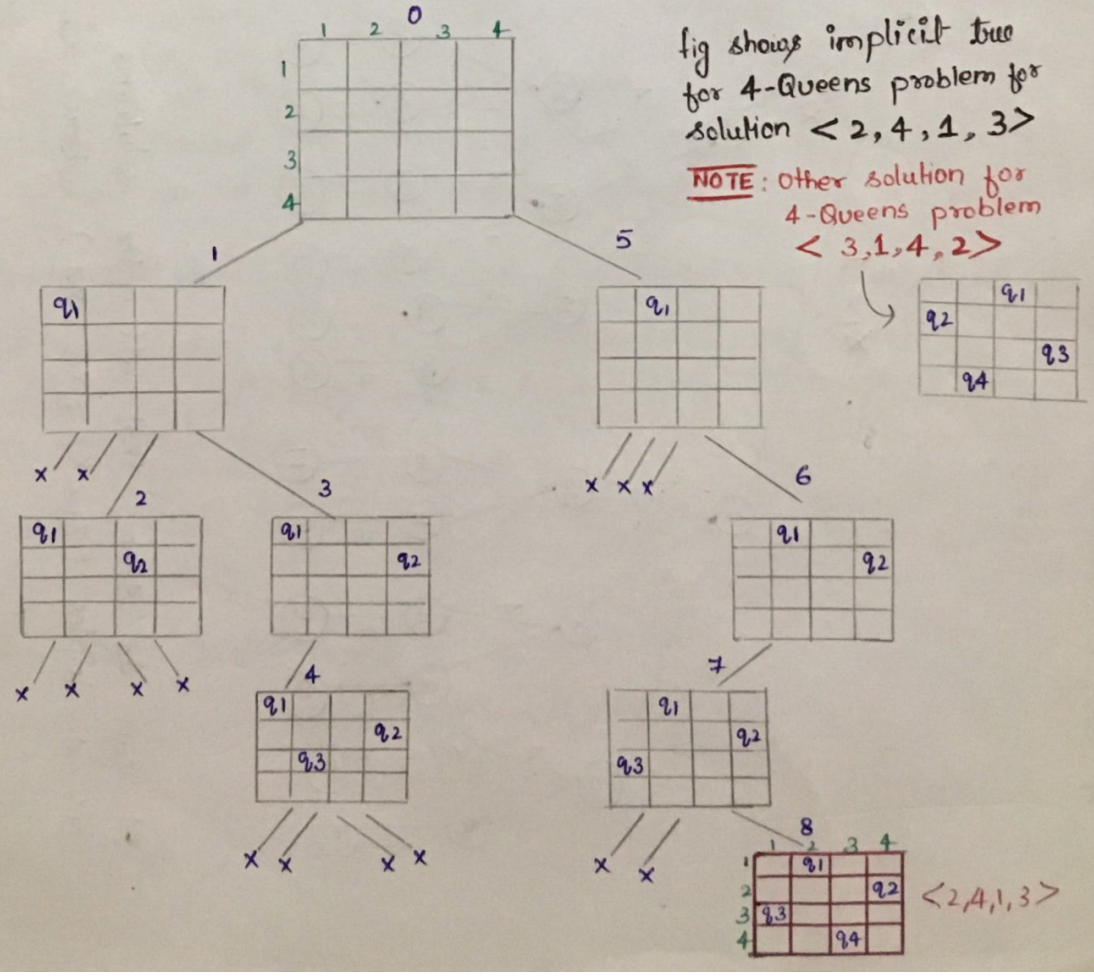
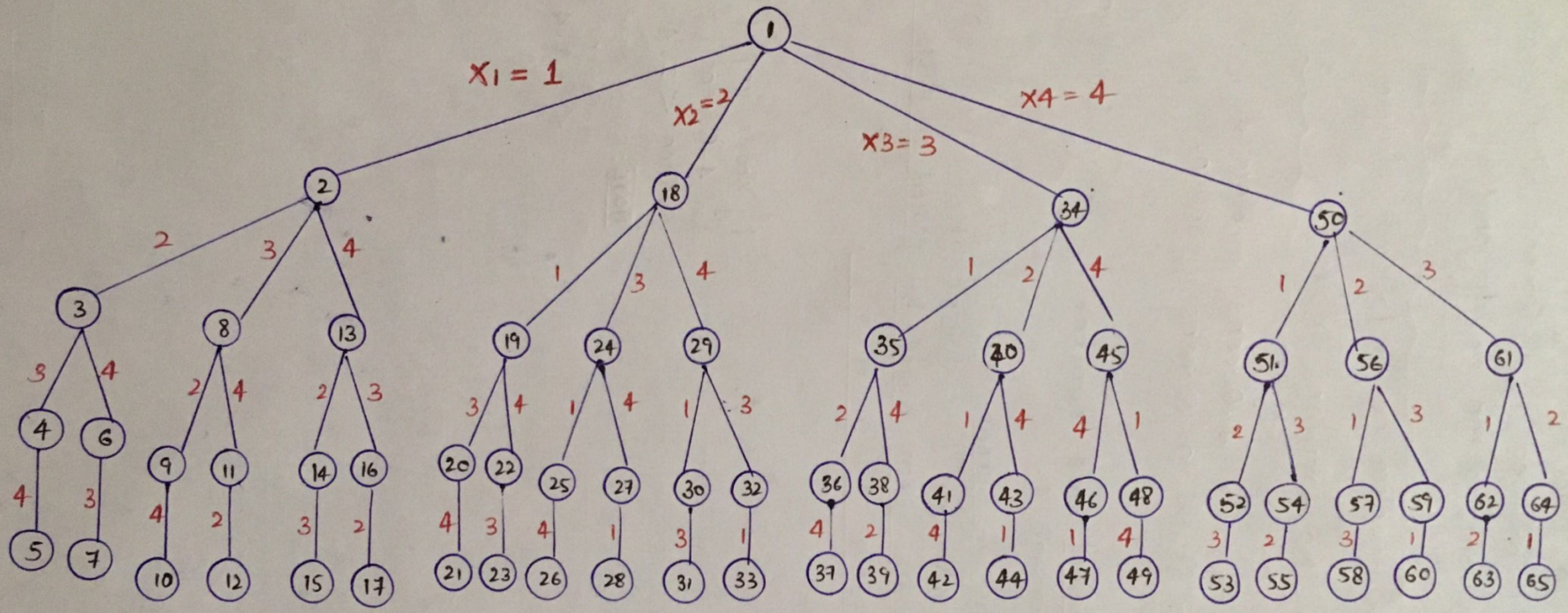



fig shows implicit tree for 4-Queens problem for solution <2, 4, 1, 3>

NOTE : other solution for 4-Queens problem < 3, 1, 4, 2>

# 4-Queens solution space with nodes numbered in DFS




It can be seen that all the solutions to the 4-Queens problem can be represented as 4-tuples $(x_1, x_2, x_3, x_4)$ where $x_i$ represents the column on which "$q_i$" is placed.

# GRAPH COLORING

Graph coloring problem is to assign colors to certain elements of a graph subject to certain constraints.

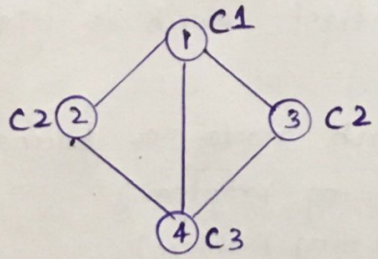
Vertex coloring is the most common graph coloring problem.

The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using same color.

In graph theory, Graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called colors to elements of a graph subject to certain constraints.

In Graph coloring problem we require minimum no: of color to make the graph colorful such that no two adjacent vertices will contain same color.

e.g:

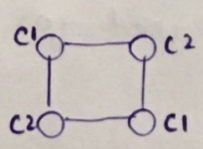



Minimum no: of color required = 3

NOTE * Minimum no: of colors required to color the graph is called CHROMATIC NUMBER.

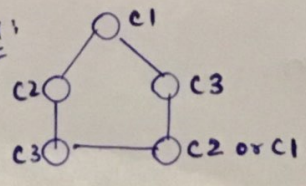
* Cycle Graph with even no: of vertices has chromatic number = 2

ex:




* Cycle Graph with odd no: of vertices has chromatic number = 3

ex:



# BRANCH & BOUND

- Systematic method for solving optimization problems.
- General optimization technique that applies where the greedy method & Dynamic programming fail.
- It is much slower, indeed, it often leads to exponential time complexities in the worst case.
- On the other hand, if applied carefully, it can lead to faster execut$^{n}$ in some cases.
- Branch & Bound technique like Branching explores the implicit graph & deals with the optimal solution to a given problem.
- In this technique at each stage we calculate the bound for a particular node and check whether this bound will be able to give solution or not.
- If we find that at any node the solution so obtained is appropriate but the remaining solution is not leading to a best case then we leave this node without exploring.
- Depth-First Search or Breadth-first Search is used for calculating the bound.
- At each stage we have for each node a bound
  - Lower Bound (for minimizing problem)
  - Upper Bound (for maximizing Problem).
- for each node, bound is calculated by means of partial solut$^{n}$.
- The calculated bound is checked with previous best result & bound with best solut$^{n}$ is selected & leave other part of solution without exploring it further.
- Branch and Bound Approach is used for no: of NP-hard problems such as
  1. Knapsack Problem
  2. Travelling Salesman Problem (TSP)
  3. Non-Linear Programming
  4. The 15-puzzle problem.

# TRAVELING SALESMAN PROBLEM

- TSP includes a salesman who has to visit a no: of cities during a tour and the condition is to visit all the cities exactly once and return back to same city where man started.

- ALGO

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

① This graph is first represented by cost matrix where
$C_{ij}$ = cost of edge ; if there is a path b/w vertex i & j
$C_{ij} = \infty$ ; if there is no path

② Convert cost of matrix into reduced matrix. i.e every row & column should contain atleast one zero entry

③ Cost of the reduced matrix is the sum of elements that are subtracted from rows & columns of cost matrix to make it reduced.

④ Make state space tree for reduced matrix.

⑤ To find the next node, find the least cost valued node by calculating the reduced cost matrix with every node.

⑥ If $\langle i, j \rangle$ edge is to be included, then there are 3 conditions to accomplish this task.

a.) change all entries in row i & col j of A to $\infty$

b) set $A[j, 1] = \infty$

c) Reduce all rows & col in resulting matrix except for rows & columns containing $\infty$

⑦ Calculate the cost of the matrix where
$cost = L + cost(i, j) + r$
L: Cost of original reduced cost matrix
r: new reduced cost matrix

⑧ Repeat above steps for all the nodes until all the nodes are generated & we get a path.

Q1 Apply Branch & Bound technique to solve TSP for the graph whose cost matrix is given below

$$\begin{bmatrix} \infty & 7 & 3 & 12 & 8 \\ 3 & \infty & 6 & 14 & 9 \\ 5 & 8 & \infty & 6 & 18 \\ 9 & 3 & 5 & \infty & 11 \\ 18 & 14 & 9 & 8 & \infty \end{bmatrix}$$

Sol$^{n}$ For finding reduced cost matrix, we try to reduce its rows.

We subtract minimum cost from corresponding row

subtract 3 from first row
" 3 " 2nd row
" 5 " 3rd row
" 3 " 4th row
" 8 " 5th row

Resulting maxtrix is

$$A = \begin{bmatrix} \infty & 4 & 0 & 9 & 5 \\ 0 & \infty & 3 & 11 & 6 \\ 0 & 3 & \infty & 1 & 13 \\ 6 & 0 & 2 & \infty & 8 \\ 10 & 6 & 1 & 0 & \infty \end{bmatrix}$$

Now check resulting matrix. All of its rows & columns should have atleast one zero value.
We see that each row in resulting matrix has zero entry
Also all columns have zero entry except last column.
So subtract 5 from last column.

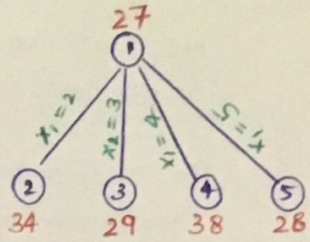
The Resulting matrix is

$$A = \begin{bmatrix} \infty & 4 & 0 & 9 & 5 \\ 0 & \infty & 3 & 11 & 6 \\ 0 & 3 & \infty & 1 & 13 \\ 6 & 0 & 2 & \infty & 8 \\ 10 & 6 & 1 & 0 & \infty \end{bmatrix} \quad \text{--- eq}^{n}(1)$$

Total cost of above matrix is the sum of total cost reduced from original matrix i.e $3+3+5+3+8+5 = 27$

P.T.O →

We will find path for (1,4) & cost = 38
also for path (1,5) we will find the cost = 28.




We will continue with node of minimum cost
i.e node 5 of cost 28.

Possible values of $x_2$ are 2, 3 & 4.
So we need to find the cost for paths (1,5,2) (1,5,3) & (1,5,4).
We again follow the same steps i.e

- set all entries in row 5 & col 2 to ∞, entry (2,1) to ∞
- Reduce the matrix
- find the cost of reduced matrix.

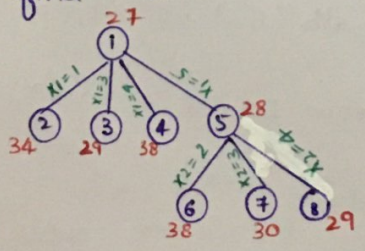
for path (1,5) reduced matrix is represented by eq$^n$ ②. We will now consider that matrix as root matrix. So

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 2 & 11 & \infty \\ 0 & \infty & \infty & 1 & \infty \\ 6 & \infty & 1 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{subtract 2 from second row}} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 9 & \infty \\ 0 & \infty & \infty & 1 & \infty \\ 5 & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{Subtract 1 from col 4}} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 8 & \infty \\ 0 & \infty & \infty & 0 & \infty \\ 5 & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

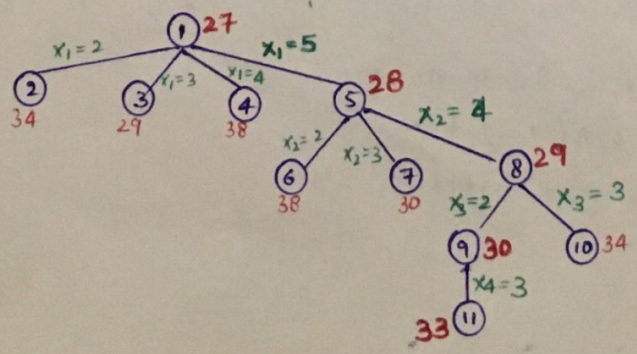
Total cost is given by

$$C(6) = C(5) + A(5,2) + r$$
$$= 28 + 6 + (2+1+1) = 38$$

Similarly we will find that minim$^m$ cost = 29 for path (1,5,4)

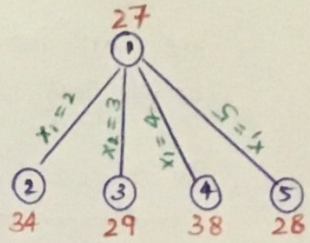



Again we select minimum cost from the possible path (1,5,4,2) and path (1,5,4,3). We find cost for path (1,5,4,2) = 30 and for the path (1,5,4,3) = 34.
Now we will find cost for path (1,5,4,2,3) = 33




We know that in TSP initial & final position are same so path is (1,5,4,2,3,1) with minimum cost as **33**

We will find path for (1,4) & cost = 38
also for path (1,5) we will find the cost = 28.




We will continue with node of minimum cost
i.e node 5 of cost 28.

Possible values of $x_2$ are 2, 3 & 4.
So we need to find the cost for paths (1,5,2) (1,5,3) & (1,5,4).
We again follow the same steps i.e

- set all entries in row 5 & col 2 to ∞, entry (2,1) to ∞
- Reduce the matrix
- find the cost of reduced matrix.

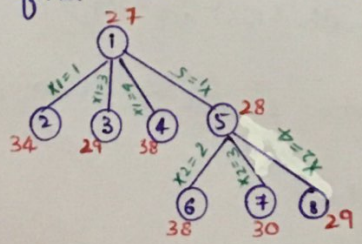
for path (1,5) reduced matrix is represented by eq$^n$ ②. We will now consider that matrix as root matrix. So

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 2 & 11 & \infty \\ 0 & \infty & \infty & 1 & \infty \\ 6 & \infty & 1 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{subtract 2 from second row}} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 9 & \infty \\ 0 & \infty & \infty & 1 & \infty \\ 5 & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \xrightarrow{\text{Subtract 1 from col 4}} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 8 & \infty \\ 0 & \infty & \infty & 0 & \infty \\ 5 & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

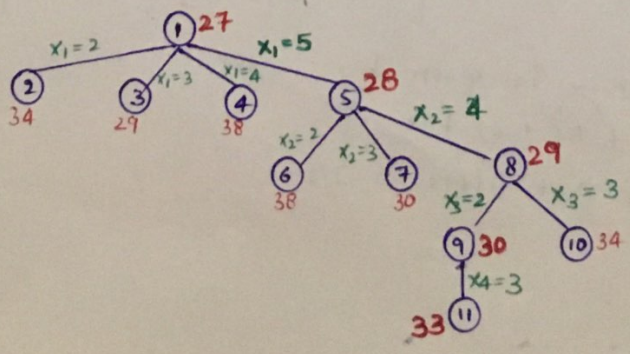
Total cost is given by

$$C(6) = C(5) + A(5,2) + r$$
$$= 28 + 6 + (2+1+1) = 38$$

Similarly we will find that minim$^m$ cost = 29 for path (1,5,4)




Again we select minimum cost from the possible path (1,5,4,2) and path (1,5,4,3). We find cost for path (1,5,4,2) = 30 and for the path (1,5,4,3) = 34.
Now we will find cost for path (1,5,4,2,3) = 33




We know that in TSP initial & final position are same so path is (1,5,4,2,3,1) with minimum cost as **33**

# 0/1 Knapsack Problem (Solution using Branch & Bound)

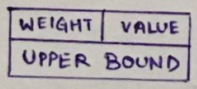
In knapsack problem we have to fill the knapsack of capacity W, with a given set of items $I_1, I_2, I_3 \ldots I_n$ having weight $w_1, w_2 \ldots w_n$ in such a manner that the total weight of the items should not exceed the knapsack capacity and the maximum possible value can be obtained.

We have a bound that none of the items can have total sum more than capacity of knapsack & must give the maximum possible value.

Implicit tree for this problem is constructed as a binary tree, where left branch signifies the inclusion of the item & the right branch signifies exclusion.

Node structure contains three parts :

first part indicates the total weight of the item
Second " " the value of current item
Third " " the upper bound for the node.




The upper bound (ub) of the node can be computed as

$$ub = v + \frac{(W - w)}{(V_{i+1}/W_{i+1})}$$

v: value of current node
W: knapsack capacity
w: weight of current node.

Q. Consider three items along with respective weights & value as

| | weight | value | value/weight |
|---|---|---|---|
| $I_1$ | 5 | 6 | 6/5 = 1.2 |
| $I_2$ | 4 | 5 | 5/4 = 1.25 |
| $I_3$ | 3 | 4 | 4/3 = 1.3 |

knapsack capacity W = 7

Solve knapsack problem using the branch & bound technique so as to give maximum possible value.

Sol$^n$ Calculate value per weight ratio. Now, we arrange the item's value per weight ratio in descending order.

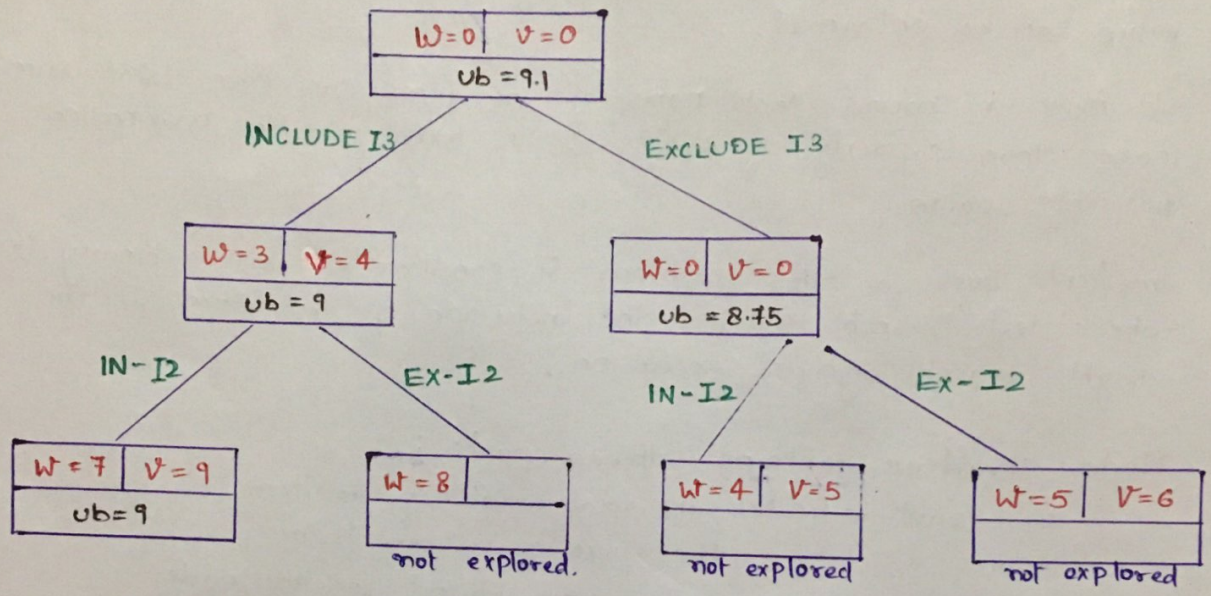
$I_3 > I_2 > I_1$

P.T.O.

upper bound = 0 + (7-0) * 1.3 = 9.1

| W=0 | V=0 |
|---|---|
| ub = 9.1 | |

Next we include item I3 which is indicated by the left branch & exclude item I3 which is indicated by right branch.




At every level we compute the upper bound & explore the node while selecting the item.

finally the node with maximum upper bound is selected as an optimum solution.

In this example node with I3 & I2 gives the optimum solution i.e maximum value = 9.