Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with 'A' Grade (CGPA : 3.18)

5D28

Continuous Assessment for Laboratory

Academic Year 2023-24

Name: Harsh Chitalya SAP ID: 60018220029

Course: Design and Analysis of Algorithm Laboratory Course Code: **DJS22ADL404**

Year: SY B.Tech (CSE(ICB)) **Sem: IV** **Batch: B1/B2**

Department: Artificial Intelligence and Data Science

Performance Indicators (Minimum 3 Indicators)	1	2	3	4	5	6	7	8	9	10	Assignment 1	Assignment 2
Course Outcome												
1. Knowledge (5) (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	4	4	4	4	4	4	4	4	4
2. Describe (5) (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5	5	5	4	4	4	4	5	5
3. Demonstration (10) (Factual/Conceptual/Procedural/ Metacognitive)	10	10	10	10	10	10	10	10	10	10	10	10
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–	–	–	–	–
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–	–	–	–	–
6. Attitude towards learning (5) (receiving, attending, responding, valuing, organizing, characterization by value)	5	5	5	5	5	5	5	5	5	5	5	5
Total (Out of 25)	23	24	24	24	24	24	23	23	23	23	24	24
Signature of the faculty member												

Sign of the Student: Harsh Chitalya

Signature of the Faculty member: Date 27/4

Signature of Head of the Department

Name of the Faculty member: Ms. Deepali Patil Date: 27/4/24

5028

Name: Harish Chitalya
GAP ID: 60018220029 , Roll no: 5028,
Branch: AIDS , Batch: A1

※ Design and Analysis of Algorithms (DAA)
Experiment - 1(a) ※

AIM: Implementation of Min Max algorithm using divide and conquer method. Also, derive its Complexity.

Problem Statement: Write a Program to find maximum and minimum temperature of greenhouse. Find Average temperature after getting max and min temperature. Also, find the range for plants growth of greenhouse.

Algorithm:

```
maxmin(i, j, max, min)
// a[1:n] is a global array, parameters i & j are integers,
   1<=i<=j<=n. The effect is to
// Set max & min to the largest & smallest value 5 in a
[i:j], respectively.
{
If (i=j) then max=min=a[i];
Else if (i=j-1) then
      {
          max=a[j];
          min=a[i];
      }
Else
```

Sundaram
FOR EDUCATIONAL USE

```
{
    max = a[i];
    min = a[j];
}
} Else
{
mid = (i+j)/2;
maxmin (I, mid, max, min);
maxmin (mid+1, j, max1, min1);
IF (max < max1) then max = max1;
IF (min > min1) then min = min1;
}}
```

The procedure is initially invoked by the statement, maxmin(1,n,x,y)

TIME COMPLEXITY:

The conventional algorithm (Naive Method) takes 2(n-1) comparisons in worst, best and average case.

maxmin does two comparisons to determine minimum and maximum element and creates two problems of size n/2, so the recurrence can be formulated as

$$T(n) = \begin{cases} 0 & , \text{if } n=1 \\ 1 & , \text{if } n=2 \\ 2T\left(\frac{n}{2}\right)+2 & , \text{if } n>2 \end{cases}$$

Let us solve this equation using Substitution Method.

$$T(n) = 2T\left(\frac{n}{2}\right) + 2 \quad \text{---} (1)$$

By Substituting n by (n/2) in eq^n -(1)

Sundaram FOR EDUCATIONAL USE

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + 2$ — (2)

$T(n) = 2\left[2T\left(\frac{n}{4}\right) + 2\right] + 2$

$= 4T\left(\frac{n}{4}\right) + 4 + 2$ ..... (3)

By substituting n by $\frac{n}{4}$ in eqn − (1)

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + 2$

Substituting in eqn − (3)

$\therefore T(n) = 4\left[2T\left(\frac{n}{8}\right) + 2\right] + 4 + 2$

$= 8T\left(\frac{n}{8}\right) + 8 + 4 + 2$

$= 2^3 T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2^1$

$\vdots$

After k-1 iterations

$\therefore T(n) = 2^{k-1} T\left(\frac{n}{2^{k-1}}\right) + 2^{k-1} + 2^{k-2} + \cdots + 2^3 + 2^2 + 2^1$

$= 2^{k-1} T\left(\frac{n}{2^{k-1}}\right) + \sum_{i=1}^{k-1} 2^i$

$= 2^{k-1} T\left(\frac{n}{2^{k-1}}\right) + (2^k - 2)$

let $n = 2^k \Rightarrow 2^{k-1} = \left(\frac{n}{2}\right)$

FOR EDUCATIONAL USE

$$\therefore T(n) = \left(\frac{n}{2}\right) T\left(\frac{2^n}{2^{n-1}}\right) + (n-2)$$

$$= \left(\frac{n}{2}\right) T(2) + (n-2)$$

We know, For $n=2$, $T(n)=1$

$$\therefore T(n) = \left(\frac{n}{2}\right) + (n-2)$$

$$T(n) = \frac{3n}{2} - 2 \quad \text{---- (Divide \& Conquer approach)}$$

For any random pattern, this algorithm takes the same number of comparisons.

If we consider the naive method as 100%, then the divide-and-conquer method is approximately 75% as efficient based on the constant factors in their time complexity expressions. However, it's important to note that big-O notations focuses on the growth rate, and in this case, both methods have the same linear growth rate.

The divide-and-conquer approach maybe more beneficial for larger datasets or in scenarios where optimization is crucial.

CONCLUSION: Thus, we had implemented MyMax algorithm

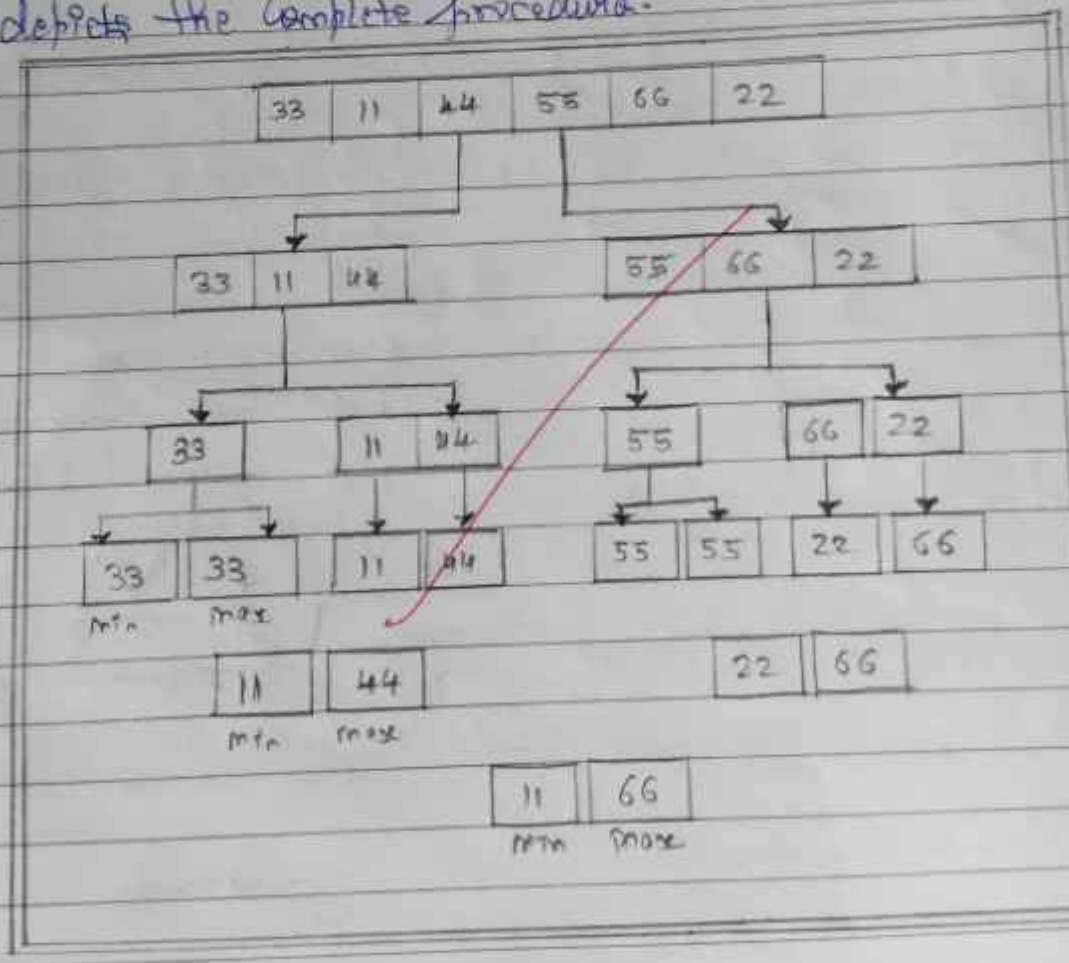
Sundaram
FOR EDUCATIONAL USE

EXAMPLE:

Find max and min from the sequence <33, 11, 44, 55, 66, 22> using divide and conquer approach.

Solution:

During the divide step, the algorithm divides the array until it reaches a size of one or two. Once the array size reaches the base case, we may get the maximum and minimum number from each array recursively. This method is continued until all the subarrays have been processed. The figure below depicts the complete procedure.



FOR EDUCATIONAL USE

Name: Harsh Chitalya
SAP ID: 60018210029, Roll no: SD2B, (SD2B)
Div: S, Branch: IT-DS, Batch: A1

Design and Analysis of Algorithm (DAA)

Experiment No 1(b)

- **Aim**: Implementation of Strassen's Matrix Multiplication using divide and conquer method. Also, desire its Complexity

- **Problem Statement**: Write a Program to Compute Matrix Multiplication using Strassen's and Naive method and Comment on its Time Complexity

- **Algorithm**:

1. Divide matrices A and B in 4 sub-matrices of size N/2 x N/2.

2. Calculate the 7 matrix multiplications recursively.

$P1 = a(f-h)$ $\quad P2 = (a+b)h$

$P3 = (c+d)e$ $\quad P4 = d(g-e)$

$P5 = (a+d)(e+h)$ $\quad P6 = (b-d)(g+h)$

$P7 = (a-c)(e+f)$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} P5+P4-P2+P6 & P1+P2 \\ P3+P4 & P1+P5-P3-P7 \end{bmatrix}$$

X Y C

3. Compute the submatrices of C

4. Combine these submatrices into our new matrix C

Sundaram
FOR EDUCATIONAL USE

Example:

Q) $A = \begin{bmatrix} -2 & 1 \\ 0 & 4 \end{bmatrix}$, $B = \begin{bmatrix} 6 & 5 \\ -7 & , \end{bmatrix}$

$p1 = -2(5-1) = -8$
$p2 = (-2+1)1 = -1$
$p3 = (0+4)6 = 24$
$p4 = (-7-6)4 = -52$
$p5 = (-2+1)(6+1) = -7$
$p6 = (1-4)(-7+5) = 6$
$p7 = (-2-0)(6+5) = -22$

$$A \times B = \begin{bmatrix} p5+p4-p2+p6 & p1+p2 \\ p3+p4 & p1+p5+p3-p7 \end{bmatrix}$$

$$A \times B = \begin{bmatrix} -19 & -3 \\ -28 & 4 \end{bmatrix}$$

Complexity Analysis:

The naive method has a time complexity of $O(n^3)$ because it involves three nested loops iterative over the elements of the matrices

Strassen's algorithm divides each matrix into four equal size submatrices of size $n/2 \times n/2$, resulting in seven recursive multiplications of size $n/2 \times n/2$. Each multiplication operation involves adding or subtracting matrices, which has a time complexity of $O(n^2)$.

Sundaram
FOR EDUCATIONAL USE

Therefore, the recurrence relation will be,

$$T(n) = 7T\left(\frac{n}{2}\right) + 18O(n^2)$$

By master's Theorem,
Compare with $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$a = 7, \; b = 2, \; f(n) = n^2$

Here

$$n^{\log_b a} = n^{\log_2 7} \quad \text{and} \quad f(n) = n^2$$

$$\therefore n^{\log_b a} > f(n) \Rightarrow n^{\log_2 7} > n^2$$

$$\therefore T(n) = O(n^{\log_2 7}) \simeq O(n^{2.807})$$

∴ For large enough values of n, Strassen's algorithm is theoretically more efficient than the naive method

Conclusion:

Thus, we had implemented Strassen's Matrix Multiplication using divide & Conquer.

FOR EDUCATIONAL USE

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2023-2024

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithms (DAA)	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

Experiment 01

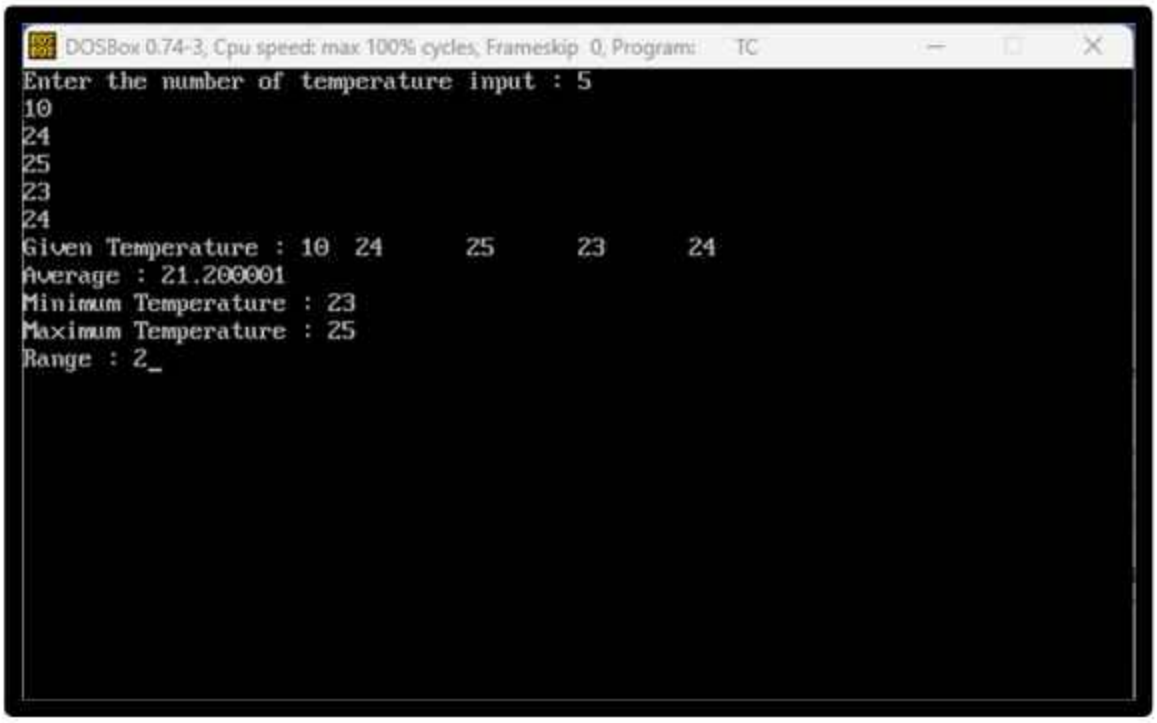
1. A) Min Max Problem Using Divide and Conquer Approach

Code:

```
#include<stdio.h>
//#include<stdlib.h>

int min = 1000;
int max = 0;

void DivideandConquer(int a[], int i, int j, int *min, int *max) {
   int mid, min1, max1;
   if (i == j) {
   *min = a[i];
   *max = a[i];
   } else if (i == j - 1) {
   if (a[i] > a[j]) {
   *min = a[j];
   *max = a[i];
     }
   } else {
   mid = (i + j) / 2;
   DivideandConquer(a, i, mid, min, max);
   min1 = max1 = a[mid + 1];
   DivideandConquer(a, mid + 1, j, &min1, &max1);
   if (*max > max1) {
    *max = *max;
   } else {
   *max = max1;
   }
   if (*min < min1) {
     *min = *min;
  } else {
   *min = min1;
     }
   }
}
```

```
int main() {
   clrscr();
   int i, j, n, range, sum = 0;
   float average;
   int a[100];
   printf("Enter the number of temperature input : ");
   scanf("%d", &n);

   for (i = 0; i < n; i++) {
   scanf("%d", &a[i]);
   sum = sum + a[i];
   }

   printf("Given Temperature : ");
   for (i = 0; i < n; i++) {
   printf("%d\t", a[i]);
   }

   average = sum / (float)n;
   printf("\nAverage : %f", average);
   i = 0;
   j = n - 1;

   DivideandConquer(a, i, j, &min, &max);
   printf("\nMinimum Temperature : %d", min);
   printf("\nMaximum Temperature : %d", max);
   range = max - min;
   printf("\nRange : %d", range);

   getch();
   return 0;
}
```

Output:



1. B) Strassen's Matrix Multiplication using Divide and Conquer Approach

Code:

```
#include<stdio.h>
#include<stdlib.h>
int i,j,k;

void multiply(int A[][2], int B[][2], int C[][2])
{
   for (int i = 0; i < 2; i++)
   {
      for (int j = 0; j < 2; j++)
      {
         C[i][j] = 0;
         for (int k = 0; k < 2; k++)
         {
            C[i][j] += A[i][k]*B[k][j];
         }
      }
   }
}

void optimize_multiply(int A[][2], int B[][2], int D[][2])
{
   int P1 = A[0][0] * (B[0][1] - B[1][1]);
   int P2 = (A[0][0] + A[0][1]) * B[1][1];
   int P3 = (A[1][0] + A[1][1]) * B[0][0];
   int P4 = A[1][1] * (B[1][0] - B[0][0]);
   int P5 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
   int P6 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);
   int P7 = (A[0][0] - A[1][0]) * (B[0][0] + B[0][1]);

   D[0][0] = P5 + P4 - P2 + P6;
   D[0][1] = P1 + P2;
   D[1][0] = P3 + P4;
   D[1][1] = P5 + P1 - P3 - P7;
}



int main()
{
   int A[2][2],B[2][2],C[2][2],D[2][2];
   printf("Enter the element in Matrix A: \n");
   for(i=0;i<2;i++)
   {
      for(j=0;j<2;j++){
         scanf("%d",&A[i][j]);
      }
   }
```

```
    printf("Enter the element in Matrix B: \n");
    for(i=0;i<2;i++)
    {
       for(j=0;j<2;j++){
          scanf("%d",&B[i][j]);
       }
    }
    printf("Entered Matrix A Is: \n");
    for(i=0;i<2;i++)
    {
       for(j=0;j<2;j++)
       {
          printf("%d \t",A[i][j]);
       }
      printf("\n");
    }
    printf("Entered Matrix B Is :\n");
    for(i=0;i<2;i++)
    {
       for(j=0;j<2;j++)
       {
          printf("%d \t",B[i][j]);
       }
       printf("\n");
    }
    multiply(A,B,C);
    printf("Resultant Matrix Is :\n");
    for(i=0;i<2;i++)
    {
       for(j=0;j<2;j++)
       {
          printf("%d \t",C[i][j]);
       }
       printf("\n");
    }
    optimize_multiply(A,B,D);
    printf("Resultant OPTIMIZED Matrix Is :\n");
    for(i=0;i<2;i++)
    {
       for(j=0;j<2;j++)
       {
          printf("%d \t",D[i][j]);
       }
       printf("\n");
    }
    return 0;
}
```

Output:

```
Enter the element in Matrix A:
12 23 34 45
Enter the element in Matrix B:
56 67 78 89
Entered Matrix A Is:
12      23
34      45
Entered Matrix B Is :
56      67
78      89
Resultant Matrix Is :
2466    2851
5414    6283
Resultant OPTIMIZED Matrix Is :
2466    2851
5414    6283
```

Name: Harish Chitalya
SAP ID: 60018220029
Roll no: 5028, Div: S
Branch: CSDS

Design & Analysis of Algorithms

5028

Experiment no 2

Aim / Output

Fractional Knapsack Problem

Q) Find optimal solution using Knapsack problem with n=7.
P = (4, 6, 7, 8, 1, 3, 2) & w = (8, 6, 3, 9, 4, 5), Let m = 15

Soln: i) Calculate p/w ratio

Wt_limit = m = 15

Item	1	2	3	4	5	6	7
Profit	4	6	7	8	1	3	2
Weight	8	6	3	9	2	4	5
P/W	0.5	1	2.33	0.88	0.5	0.75	0.4

ii) Arrange in decreasing order of p/w ratio

Item	3	2	4	6	1	5	7
Profit	7	6	8	3	4	1	2
Weight	3	6	9	4	8	2	5
P/W	2.33	1	0.88	0.75	0.5	0.5	0.4

iii) a) Item 3 wt[3] = 3
∵ 3 ≤ wt_limit (3 ≤ 15)
∴ Put Item 3 in bag
Set m = 3
wt_limit = 15 - 3 = 12

Sundaram
FOR EDUCATIONAL USE

b) Item 2 $wt[2] = 6$

$\because 6 \leq wt\_limit \quad (6 \leq 12)$

$\therefore$ Put item 2 in bag

Set $m_2 = 1$

$wt\_limit = 12 - 6 = 6$

c) Item 4 $wt[4] = 9$

$\because 9 \not\leq wt\_limit \quad (\because 3 \not< 6)$

But $wt\_limit = 6 > 0$

$\therefore$ Calculate fraction $= \frac{wt\_limit}{wt[4]} = \frac{6}{9}$

$= 0.6667$

Set $m_4 = 0.6667$

$wt\_limit = 0$

Since $wt\_limit = 0$ we cannot add any further item into bag

So, we can stop. Set m_i for rest item be zero

$\therefore$ Item 3, Item 2 & Item 4 will be put inside the bag

Maximum Profit $= \sum p_i m_i$

$= (4 \times 0) + (6 \times 1) + (7 \times 1) + (8 \times 0.6667) + (1 \times 0) + (3 \times 0) + (2 \times 0)$

$= 6 + 7 + 5.33$

Maximum Profit $= 18.33$

$\therefore$ We will get maximum profit of 18.333 if we put item 2, 3 & 4 in knapsack.

FOR EDUCATIONAL USE

Sundaram

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2023-2024

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithm	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

Experiment 02

AIM: Implementation of Fractional Knapsack using Greedy method.

Problem Statement: You are an adventurer exploring a desert oasis in search of valuable resources. Along your journey, you come across various types of fruits scattered around the oasis. Each fruit has a specific weight and nutritional value. Your goal is to determine the maximum total nutritional value you can carry in your backpack without exceeding its weight capacity. You have the flexibility to take fractional parts of fruits, optimizing your backpack's load for maximum nutritional value. Choose wisely to ensure you have enough energy to continue your exploration through the desert.

Algorithm:

The problem states-

Which items should be placed into the knapsack such that-

- The value or profit obtained by putting the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed

```
Fractional Knapsack (Array W, Array V, int M)

for i <- 1 to size (V)

        calculate cost[i] <- V[i] / W[i]

Sort-Descending (cost)

i ← 1

while (i <= size(V))

        if  W[i] <= M

                M ← M – W[i]

                total ← total + V[i];

        if  W[i] > M

                i ← i+1
```

Time Complexity:

- For one item there are two choices, either to select or reject. For 2 items we have four choices:
  - Select both items
  - Reject both items
  - Select first and reject second
  - Reject first and select second
- In general, for n items, knapsack has 2^n choices. So brute force approach runs in $O(2^n)$ time.
- We can improve performance by sorting items in advance. Using merge sort or heap sort, n items can be sorted in $O(n\log_2 n)$ time. Merge sort and heap sort are non-adaptive and their running time is the same in best, average and worst case.
- To select the items, we need one scan to this sorted list, which will take O(n) time.
- So the total time required is
  $T(n) = O(n\log_2 n) + O(n) = O(n\log_2 n)$.

Code:

```
#include<stdio.h>
#include<conio.h>
void knapsack(int n, float weight[], float profit[], float capacity)
{
  float x[20], tp = 0;
  int i, j, u;
  u = capacity;

  for (i = 0; i < n; i++)
    x[i] = 0.0;

  for (i = 0; i < n; i++) {
    if (weight[i] > u)
      break;
    else {
      x[i] = 1.0;
      tp = tp + profit[i];
      u = u - weight[i];
    }
  }

  if (i < n)
    x[i] = u / weight[i];
  tp = tp + (x[i] * profit[i]);
  printf("\nThe result vector is:- ");
  for (i = 0; i < n; i++)
    printf("%f\t", x[i]);
  printf("\nMaximum profit is:- %f", tp);
}
```

```
int main() {
  float weight[20], profit[20], capacity;
  int num, i, j;
  float ratio[20], temp;
  clrscr();

  printf("\nEnter the no. of objects:- ");
  scanf("%d", &num);


  for (i = 0; i < num; ++i) {
    printf("\nEnter the wts and profits of each object %d: " , i+1);
    scanf("%f %f", &weight[i], &profit[i]);
  }

  printf("\nEnter the capacity of knapsack:- ");
  scanf("%f", &capacity);

  for (i = 0; i < num; i++) {
    ratio[i] = profit[i] / weight[i];
  }

  for (i = 0; i < num; i++) {
    for (j = i + 1; j < num; j++) {
      if (ratio[i] < ratio[j]) {
        temp = ratio[j];
        ratio[j] = ratio[i];
        ratio[i] = temp;

        temp = weight[j];
        weight[j] = weight[i];
        weight[i] = temp;

        temp = profit[j];
        profit[j] = profit[i];
        profit[i] = temp;
      }
    }
  }

  knapsack(num, weight, profit, capacity);
  getch();
  return(0);
}
```

Output:

```
Enter the no. of objects:- 7

Enter the wts and profits of each object 1: 8 4

Enter the wts and profits of each object 2: 6 6

Enter the wts and profits of each object 3: 3 7

Enter the wts and profits of each object 4: 9 8

Enter the wts and profits of each object 5: 2 1

Enter the wts and profits of each object 6: 4 3

Enter the wts and profits of each object 7: 5 2

Enter the capacity of knapsack:- 15

The result vector is:- 1.000000 1.000000        0.666667        0.000000        0.000000        0.000000        0.000000
Maximum profit is:- 18.333334
```

```
Enter the no. of objects:- 7

Enter the wts and profits of each object 1: 2 10

Enter the wts and profits of each object 2: 3 5

Enter the wts and profits of each object 3: 5 15

Enter the wts and profits of each object 4: 7 7

Enter the wts and profits of each object 5: 1 6

Enter the wts and profits of each object 6: 4 18

Enter the wts and profits of each object 7: 1 3

Enter the capacity of knapsack:- 15

The result vector is:- 1.000000 1.000000        1.000000        1.000000        1.000000        0.666667        0.000000
Maximum profit is:- 55.333332
```

Conclusion: Thus, we have implemented Fractional Knapsack using Greedy method.

Name: Harsh Khitaliya
SAP ID: 60018230129,
Roll no: 5128, DM: S
Batch: A-DS

DAA

Experiment no 3

5128

Activity Selection

Q) Given following data, determine the optimal schedule using greedy approach A = <A1, A2, A3, A4, A5, A6>, S = <1, 2, 3, 4, 5, 6>, F = <3, 6, 4, 5, 7, 9>

Soln: Given activities:

	A1	A2	A3	A4	A5	A6
Start	1	2	3	4	5	6
finish	3	6	4	5	7	9

1) Sort in ascending order of finish time

	A1	A3	A4	A2	A5	A6
Start	1	3	4	2	5	6
finish	3	4	5	6	7	9

2) let finish time = 0

3) (a) A_1

Start = 1 ∵ start ≥ finish time (1 ≥ 0)

∴ Select the activity A_1

Set finish time = 3

(b) A_3

Start = 3 ∵ 3 ≥ finish time (3)

Select the Activity A_3

Set finish time = 4

Sundaram
FOR EDUCATIONAL USE

(e) A_4

Start = 4 $\because$ 4 ≥ finish time

$\therefore$ Select the activity A_4

Set finish time = 5

(d) A_2

Start = 2 $\because$ 2 ✗ finish time

$\therefore$ reject the activity A_2

(c) A_5

Start = 5 $\because$ 5 ≥ finish time

$\therefore$ Select the activity A_5

Set finish time = 7

(f) A_6

Start = 6 $\because$ 6 ✗ finish time

$\therefore$ reject the activity A_6

Therefore optimal schedule is $\Rightarrow (A_1 \Rightarrow A_3 \Rightarrow A_4 \Rightarrow A_5)$

Sundaram

FOR EDUCATIONAL USE

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2023-2024

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithm	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

Experiment 03

Activity Selection Algorithm

AIM: Implementation of activity selection problem using Greedy method.

Problem Statement: Given a set of tasks with their respective start and finish times, the task is to select the maximum number of non-conflicting tasks that can be performed by a single person or machine within a given time frame. Two tasks are said to be conflicting if they overlap with each other, i.e., if one task starts before the other finishes. The objective is to find an optimal solution that maximizes the number of tasks selected, ensuring that no two selected tasks conflict with each other.

Algorithm:

The problem statement for Activity Selection is that "Given a set of n activities with their start and finish times, we need to select maximum number of non-conflicting activities that can be performed by a single person, given that the person can handle only one activity at a time."

```
Activity-Selection(Activity, start, finish)
    Sort Activity by finish times stored in finish
    Selected = {Activity[1]}
    n = Activity.length
    j = 1
    for i = 2 to n:
       if start[i] ≥ finish[j]:
          Selected = Selected U {Activity[i]}
          j = i
return Selected
```

TIME COMPLEXITY:

When activities are sorted by their finish time: O(N)

When activities are not sorted by their finish time, the time complexity is O(NlogN)+O(N) =O(NlogN)due to complexity of sorting

Code:

```
#include<stdio.h>
 #include<stdlib.h>
  int start[100],finish[100];
 char task[100],activity[100];

 void swap(int a[], int j)
 {
    int temp1;
    temp1 = a[j];
    a[j] = a[j+1];
    a[j+1] = temp1;
 }

 void swap2(char a[],int j)
 {
    int temp1;
    temp1 = a[j];
    a[j] = a[j+1];
    a[j+1] = temp1;
 }

 int main()
 {
 int n,i=0,j,temp1,temp2,temp3;
 printf("Enter the number of task:");
 scanf("%d",&n);
 printf("Enter the start time of task : ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&start[i]);
      task[i] = i+1;
   }
 printf("Enter the finish time of task : ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&finish[i]);
   }


 for(i=0;i<n-1;i++)
 {
   for(j=0;j<n-i-1;j++)
   {
      if(finish[j] > finish[j+1])
      {
         swap(finish,j);
         swap(start,j);
         swap2(task,j);
      }
   }
 }

 activity[0]  = task[0];
```

```
i= 0;
int count = 1;
for(j=1;j<n;j++)
{
   if(start[j] >= finish[i])
      {
         activity[count++] = task[j];
         i=j;
      }
   }

printf("\nTasks that are selected : ");
for(i=0;i<count;i++)
   {
      printf("%d\t",activity[i]);
   }
}
```

Output:

```
Enter the number of task:8
Enter the start time of task : 1
0
1
4
2
5
3
4
Enter the finish time of task : 3
4
2
6
9
8
5
5

Tasks that are selected : 3     7       6
```

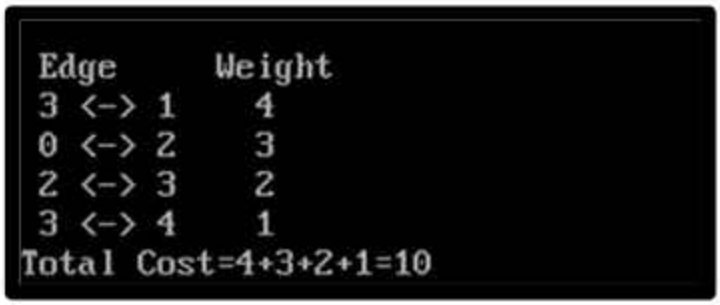
Conclusion: Thus, we have implemented activity selection problem using Greedy method.

Prim's Algorithm:

```
#include <stdio.h>
#include <limits.h>
#define vertices 5
int minimum_key(int k[], int mst[])
{
   int minimum  = INT_MAX, min,i;


   for (i = 0; i < vertices; i++)
      if (mst[i] == 0 && k[i] < minimum )
         minimum = k[i], min = i;
   return min;
}

void prim(int g[vertices][vertices])
{

   int parent[vertices];
   int k[vertices];
   int mst[vertices];
   int i, count,edge,v;
   int sum=0;
   for (i = 0; i < vertices; i++)
   {
      k[i] = INT_MAX;
      mst[i] = 0;
   }
   k[0] = 0;
   parent[0] = -1;
   for (count = 0; count < vertices-1; count++)
   {
      edge = minimum_key(k, mst);
      mst[edge] = 1;
      for (v = 0; v < vertices; v++)
      {
         if (g[edge][v] && mst[v] == 0 && g[edge][v] <  k[v])
         {
            parent[v]  = edge, k[v] = g[edge][v];
         }
      }
    }
    printf("\n Edge \t  Weight\n");
    for (i = 1; i < vertices; i++)
    printf(" %d <-> %d    %d \n", parent[i], i, g[i][parent[i]]);
    printf ("Total Cost=");
for (i = 1; i < vertices; i++) {
 sum = sum + (g[i][parent[i]]);
 if (i < vertices - 1) {
 printf ("%d+", g[i][parent[i]]);
} else {
 printf ("%d=", g[i][parent[i]]);
}
}
 printf ("%d", sum);
```

```
}
int main()
{
   int g[vertices][vertices] = {{0, 0, 3, 0, 0},
                        {0, 0, 10, 4, 0},
                        {3, 10, 0, 2, 6},
                        {0, 4, 2, 0, 1},
                        {0, 0, 6, 1, 0},
                        };
   prim(g);
   return 0;
}
```

OUTPUT:

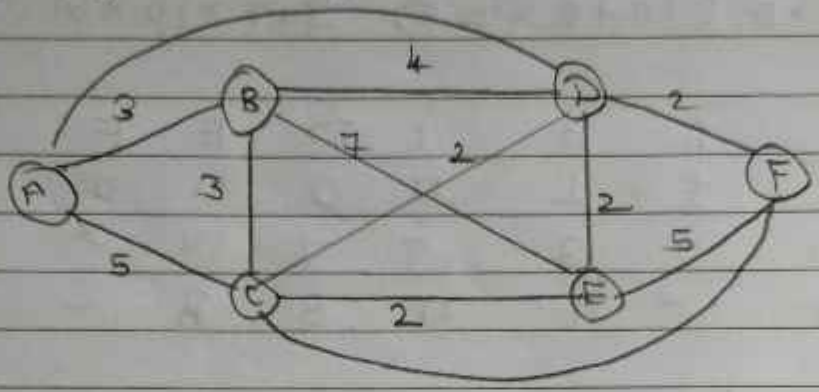


Name: Harsh Chitalya
SAP ID: 60018220029
Roll no: S028, Div: S
Branch: AIDS

DAA Experiment no 2

S1 24

Q) Find shortest path from source to all other node using Dijkstra's algorithm



Soln: 1) Initialize table
let source node be A

Node	A	B	C	D	E	F	G
Visited	0	0	0	0	0	0	0
distance	∞	∞	∞	∞	∞	∞	∞
parent	—	—	—	—	—	—	—

2) a) Our source node is A, let its distance be 0
Adjacent of A are A→B : 3 < ∞ ∴ Update its value
A→C : 5 < ∞ ∴ Update its value
A→D : 9 < ∞ ∴ Update its value

Node	A	B	C	D	E	F
Visited	1	0	0	0	0	0
distance	0	3	5	9	∞	∞
parent	—	A	A	A	—	—

Sundaram
FOR EDUCATIONAL USE

b) For node B,

The adjacent nodes are

B → C : (A → C via B) : 3+3 = 6 < 5 : don't update

B → D : (A → D via B) 3+4 = 7 < ∞ : update

B → E : (A → E via B) : 3+7 = 10 < ∞ : update

Node	A	B	C	D	E	F
Visited	1	1	0	0	0	0
Distance	0	3	5	7	10	∞
parent	-	A	A	B	B	-

c) For Node C,

Its adjacent node are

C → D : (A → D via C) : 5+2 = 7 < 7 : don't update

C → E : (A → E via C) : 5+6 = 11 < 10 : don't update

C → F : (A → F via C) : 5+8 = 13 < ∞ : update

Node	A	B	C	D	E	F
Visited	1	1	1	0	0	0
Distance	0	3	5	7	10	13
Parent	-	A	A	B	B	C

d) For node D

Its adjacent are

D → E : (A → E via D) : 7+2 = 9 < 10 : update

D → F : (A → F via D) : 7+2 = 9 < 13 : update

Node	A	B	C	D	E	F
Visited	1	1	1	1	0	0
distance	0	3	5	7	9	9
parent	-	A	A	B	D	D

Sundaram
FOR EDUCATIONAL USE

e) For Node E,
Its adjacent are
E→F : A→F via E : 9+5 = 14 ✗ 9

Also, F is the last node & it has no unvisited adjacent

Therefore final table will look like

Node	A	B	C	D	E	F
Visited	1	1	1	1	1	1
Distance	0	3	5	7	9	9
Parent	-	A	A	B	D	D

Therefore Shortest path from Source node A are :
A→A = 0
A→B : A→B = 3
A→C : A→C = 5
A→D : A→B→D = 7
A→E : A→B→D→E = 9
A→F : A→A→D→F = 9

Sundaram
FOR EDUCATIONAL USE

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2023-2024

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithm	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

Experiment 04

Dijkstra Algorithm

AIM: Implementation of Single source shortest path (Dijkstra's) problem using Greedy method.

Problem Statement: A new startup FastNetRoute wants to route information along a path in a communication network, represented as a graph. Each vertex represents a router and each edge a wire between routers. The wires are weighted by the maximum bandwidth they can support. FastNetRoute comes to you and asks you to develop an algorithm to find the path with maximum bandwidth from any source s to any destination t. As you would expect, the bandwidth of a path is the minimum of the bandwidths of the edges on that path; the minimum edge is the bottleneck. Show the results by implementing it.

Algorithm:

- Dijkstra Algorithm is a very famous greedy algorithm.
- It is used for solving the single source shortest path problem.
- It computes the shortest path from one particular source node to all other remaining nodes of the graph.
- **Conditions-**
- It is important to note the following points regarding Dijkstra Algorithm-
    - Dijkstra algorithm works only for connected graphs.
    - Dijkstra algorithm works only for those graphs that do not contain any negative weight edge.
    - The actual Dijkstra algorithm does not output the shortest paths.
    - It only provides the value or cost of the shortest paths.
    - By making minor modifications in the actual algorithm, the shortest paths can be easily obtained.
    - Dijkstra algorithm works for directed as well as undirected graphs.

dist[S] ← 0 // The distance to source vertex is set to 0

Π[S] ← **NIL** // The predecessor of source vertex is set as NIL

for all v ∈ V - {S} // For all other vertices

do dist[v] ← ∞ // All other distances are set to ∞

Π[v] ← **NIL** // The predecessor of all other vertices is set as NIL

S ← Ø // The set of vertices that have been visited 'S' is initially empty

Q ← V // The queue 'Q' initially contains all the vertices

while Q ≠ Ø // While loop executes till the queue is not empty

do u ← mindistance (Q, dist) // A vertex from Q with the least distance is selected

S ← S ∪ {u} // Vertex 'u' is added to 'S' list of vertices that have been visited

for all v ∈ neighbors[u] // For all the neighboring vertices of vertex 'u'

do if dist[v] > dist[u] + w(u,v) // if any new shortest path is discovered

then dist[v] ← dist[u] + w(u,v) // The new value of the shortest path is selected

return dist

TIME COMPLEXITY:

Case-01:

This case is valid when-The given graph G is represented as an adjacency matrix.Priority queue Q is represented as an unordered list.

Here,

A[i,j] stores the information about edge (i,j).

Time taken for selecting i with the smallest dist is O(V).

For each neighbor of i, time taken for updating dist[j] is O(1) and there will be maximum V neighbors.

Time taken for each iteration of the loop is O(V) and one vertex is deleted from Q.

Thus, total time complexity becomes $O(V^2)$.

Case-02:

This case is valid when-The given graph G is represented as an adjacency list.Priority queue Q is represented as a binary heap.

Here,

With adjacency list representation, all vertices of the graph can be traversed using BFS in O(V+E) time.

In min heap, operations like extract-min and decrease-key value takes O(logV) time.

So, overall time complexity becomes O(E+V) x O(logV) which is O((E + V) x logV) = O(ElogV)

This time complexity can be reduced to O(E+VlogV) using Fibonacci heap.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <conio.h>
#define MAX_VERTICES 100
int minDistance(int dist[], int visited[], int V) {
   int min = INT_MAX, min_index;
   int v;
   for (v = 0; v < V; v++) {
      if (visited[v] == 0 && dist[v] <= min) {
         min = dist[v];
         min_index = v;
      }
   }

   return min_index;
}


void printSolution(int dist[], int V) {
   int i;
   printf("Vertex   Distance from Source\n");
   for (i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}


void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int src, int V) {
   int dist[MAX_VERTICES];
   int visited[MAX_VERTICES];
   int i,  count ,v;
   for (i = 0; i < V; i++) {
      dist[i] = INT_MAX;
      visited[i] = 0;
   }
   dist[src] = 0;
   for (count = 0; count < V - 1; count++) {
      int u = minDistance(dist, visited, V);
      visited[u] = 1;
      for (v = 0; v < V; v++) {
         if (!visited[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) {
            dist[v] = dist[u] + graph[u][v];
         }
      }
   }
   printSolution(dist, V);
}
```

```
int main() {
   int V;
   int graph[MAX_VERTICES][MAX_VERTICES];
   int i,j;
   int source;
   clrscr();
   printf("Enter the number of vertices: ");
   scanf("%d", &V);
   printf("Enter the adjacency matrix (0 for no connection):\n");
   for (i = 0; i < V; i++) {
      for (j = 0; j < V; j++) {
         scanf("%d", &graph[i][j]);
      }
   }
   printf("Enter the source vertex: ");
   scanf("%d", &source);
   dijkstra(graph, source, V);
   getch();
   return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the adjacency matrix (0 for no connection):
0 2 6 12 15
0 0 7 0 3
0 0 0 5 0
0 0 0 0 3
0 0 0 0 0
Enter the source vertex: 0
Vertex   Distance from Source
0                0
1                2
2                6
3                11
4                5
```

Conclusion: Thus, we have implemented Single source shortest path (Dijkstra's) problem using Greedy method.

Name: Harish Chitalya
SAPID: 60018220029, Div: 5
RollNo: 5028, Batch: A1, Branch: ATDS

Design and Analysis of Algorithms

Experiment no 5

COIN CHANGE PROBLEM

Q.1] Create a coin change dynamic programming solution for following 4 coins denominations and amount as a sum

N=4, Sum=10, Coin = {2, 5, 3, 6}

Coins \ W	0	1	2	3	4	5	6	7	8	9	10
2	0	0	1	0	2	0	3	0	4	0	5
3	0	0	1	1	2	2	2	3	3	3	4
5	0	0	1	1	2	(1)	2	(2)	2	3	2
6	0	0	1	1	2	1	1	2	2	2	(2)

Therefore,
Minimum number of coins = 2
Denomination = {5, 5}

Sundaram
FOR EDUCATIONAL USE

Q.3) Create a coin change dynamic programming solution for following 5 coins denomination and amount as a sum.

Input = {1, 2, 3, 4, 5}
Sum = 16

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	0	1	1	2	2	3	3	4	4	5	5	6	6	6	7	7	7
3	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	5
4	0	1	1	1	(1)	2	2	2	2	3	3	3	3	4	4	4	4
5	0	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	(4)

Therefore,
Minimum no of coins = 4 coins
Denominations: {4, 4, 4, 4}

Sundaram
FOR EDUCATIONAL USE

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2023-2024

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithm (DAA)	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

Experiment 05

Aim: Implementation of Coin Change Problem using Dynamic Programming.

Problem Statement:
You are working on the software for a vending machine that accepts both bills and coins. The goal is to implement a dynamic programming solution to determine the minimum number of coins to be returned as change when a customer makes a purchase.

Input:

- The purchase amount.
- The amount paid by the customer.
- A set of available coin denominations in the vending machine.

Output:

- The minimum number of coins needed for change.
- The combination of coins that make up the minimum number.

Algorithm:

Given a set of Coins for example coins[] = {1, 2, 3} and total amount as sum, we need to find the number of ways the coins[] can be combined in order to get the sum, abiding the condition that the order of the coins doesn't matter.

Example:

coins[] = {1, 2, 3}

sum = 4

Possible changes: {1,1,1,1}, {2,2}, {1,3}, {1,1,2}.

Solutions: 4

Note: The order of coins does not matter – For example, {1,3} = {3,1}.

The Coin Change Problem can be solved in two ways –

Recursion – Naive Approach, Slow.

Dynamic Programming – Efficient Approach, Fast.

For Printing Table

```
// Base case (If given value V is 0)
table[i][0] = 0;
 table[0][i] = 500;
    // Compute minimum coins required for all
   // values from 1 to V
   for (int i = 1; i &lt;= m; i++)
        // Go through all coins smaller than i
      for (int j = 1; j &lt;=V; j++)
          if (coins[i-1] &gt; j)
            table[i][j]=table[i-1][j];
          else
     table[i][j]=min(table[i-1][j],1+table[i][j-coins[i-1]])
```

For solution:

```
  i=m,j=V,p=0;
  while(j&gt;0)
     if(table[i][j]==table[i-1][j])
      i=i-1;
     else
        j=j-coins[i-1];
     sol[p]=coins[i-1];
     p++;
 print Sol
```

TIME COMPLEXITY:

The complexity of solving the coin change problem using recursive time and space will be:

Problems: Overlapping subproblems + Time complexity

$O(2^n)$ is the time complexity, where n is the number of coins

Time and space complexity will be reduced by using dynamic programming to solve the coin change problem:

- O(numberOfCoins*TotalAmount) time complexity
- O(numberOfCoins*TotalAmount) is the space complexity.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h> //This is used for infinity
#include <conio.h>
int minCoins(int coins[], int m, int amount, int result[]) {
    int* table = (int*)malloc((amount + 1) * sizeof(int));
    int i, j;
    int minCoinCount;
    for (i = 0; i <= amount; i++) {
        table[i] = INT_MAX;
        result[i] = -1;
    }
    table[0] = 0;
    for (i = 1; i <= amount; i++) {
        for (j = 0; j < m; j++) {
            if (coins[j] <= i) {
                int subResult = table[i - coins[j]];
                if (subResult != INT_MAX && subResult + 1 < table[i]) {
                    table[i] = subResult + 1;
                    result[i] = j;
                }
            }
        }
    }
    minCoinCount = table[amount];
    free(table);
    return minCoinCount;
}

void printCombination(int result[], int coins[], int amount) {
    int start = amount;
    if (start == -1) {
        printf("No valid combination found.");
        return;
    }
    printf("Coins used for the minimum number: ");
    while (start != 0) {
        int j = result[start];
        printf("%d ", coins[j]);
        start = start - coins[j];
    }
    printf("\n");
}
```

```
int main() {
   int i, m;
   int coins[100];
   int amount, paid;
   int change;
   int* result;
   int minCoinCount;
   //clrscr();
   printf("Enter the number of coin denominations: ");
   scanf("%d", &m);
   printf("Enter the coin denominations: ");
   for (i = 0; i < m; i++) {
      scanf("%d", &coins[i]);
   }
   printf("Enter the Total amount: ");
   scanf("%d", &amount);
   printf("Enter the amount paid by the customer: ");
   scanf("%d", &paid);
   change = amount - paid;
   printf("Reuqired Change Amount: %d \n",change);
   result = (int*)malloc((change + 1) * sizeof(int));
   minCoinCount = minCoins(coins, m, change, result);
   printf("Minimum number of coins needed for change: %d\n", minCoinCount);
   printCombination(result, coins, change);
   free(result);
   //getch();
   return 0;
}
```

Output:

```
Enter the number of coin denominations: 5
Enter the coin denominations: 1 2 3 4 5
Enter the Total amount: 30
Enter the amount paid by the customer: 14
Reuqired Change Amount: 16
Minimum number of coins needed for change: 4
Coins used for the minimum number: 1 5 5 5
```

```
Enter the number of coin denominations: 4
Enter the coin denominations: 2 3 5 6
Enter the Total amount: 20
Enter the amount paid by the customer: 10
Reuqired Change Amount: 10
Minimum number of coins needed for change: 2
Coins used for the minimum number: 5 5
```

Code:

```
#include <stdio.h>
int min(int a, int b) {
   return (a < b) ? a : b;
}

int minCoins(int coins[], int m, int V) {
   int table[m + 1][V + 1];

   for (int i = 0; i <= m; i++)
      table[i][0] = 0;
   for (int i = 1; i <= V; i++)
      table[0][i] = i;

   for (int i = 1; i <= m; i++) {
      for (int j = 1; j <= V; j++) {
         if (coins[i - 1] > j)
            table[i][j] = table[i - 1][j];
         else
            table[i][j] = min(table[i - 1][j], 1 + table[i][j - coins[i - 1]]);
      }
   }
   printf("Capacity/Coins\t\n");
   printf("\t\t");
   for (int i = 0; i <= V; i++) {
      printf("%d\t", i);
   }
   printf("\n");

   for (int i = 1; i <= m; i++) {
      printf("%d\t\t", coins[i - 1]);
      for (int j = 0; j <= V; j++) {
         printf("%d\t", table[i][j]);
      }
      printf("\n");
   }
   int i = m, j = V, p = 0;
   int sol[m + 1];

   while (j > 0) {
      if (table[i][j] == table[i - 1][j])
         i = i - 1;
      else {
         j = j - coins[i - 1];
         sol[p] = coins[i - 1];
         p++;
      }
   }
```

```
    printf("Coins used: ");
    for (int k = 0; k < p; k++) {
        printf("%d ", sol[k]);
    }
    printf("\n");

    return table[m][V];
}

int main() {
    int coins[100], m, V;

    printf("Enter the number of coins: ");
    scanf("%d", &m);
    printf("Enter the coins: ");
    for (int i = 0; i < m; i++) {
        scanf("%d", &coins[i]);
    }

    printf("Enter the total amount: ");
    scanf("%d", &V);

    int minCount = minCoins(coins, m, V);
    printf("Minimum number of coins required: %d\n", minCount);

    return 0;
}
```

Output:

```
Enter the number of coins: 4
Enter the coins: 1 3 5 9
Enter the total amount: 10
Capacity/Coins
               0       1       2       3       4       5       6       7       8       9       10
1              0       1       2       3       4       5       6       7       8       9       10
3              0       1       2       1       2       3       2       3       4       3       4
5              0       1       2       1       2       1       2       3       2       3       2
9              0       1       2       1       2       1       2       3       2       1       2
Coins used: 5 5
Minimum number of coins required: 2
```

Conclusion:

Thus, we have successfully implemented Dynamic coin change problem.

Name: Harsh Chitalya
SAPID: 60018220029, Roll No: 502B
Div: 5, Branch: AIDS, Batch: A1

502B

Experiment no 6

24/4/24

Design and Analysis of Algorithm

Aim: Implementation of Matrix chain multiplication using Dynamic programming

Solution:-

p = (4, 10, 5, 6, 2, 7) = (p0, p1, p2, p3, p4, p5)
1 1 2 3 4 5

0	200	320	240	346
	0	300	160	300
		0	60	130
			0	84
				0

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

$\therefore m[i,j] = \min\{m[i,k] + m[k+1] + (p_{i-1} \times p_k \times p_j)$

$m[1,2] = m[1,1] + m[2,2] + (4 \times 10 \times 5) = 0+0+200 = 200$
$m[2,3] = m[2,2] + m[3,3] + (10 \times 5 \times 6) = 0+0+300 = 300$
$m[3,4] = m[3,3] + m[4,4] + (5 \times 6 \times 2) = 0+0+60 = 60$
$m[4,5] = m[4,4] + m[5,5] + (6 \times 2 \times 7) = 0+0+84 = 84$

$m[1,3] = \min\{m[1,1] + m[2,3] + (4 \times 10 \times 6), m[1,2] + m[3,3] + (4 \times 5 \times 6)\}$
$= \min\{0+300+240, 200+0+120\} = 320$

$m[2,4] = \min\{m[2,2] + m[3,4] + (10 \times 5 \times 2), m[2,3] + m[4,4] + (10 \times 6 \times 2)\}$
$= \min\{0+60+100, 300+0+120\} = 160$

$m[3,5] = \min\{m[3,3] + m[4,5] + (5 \times 6 \times 7), m[3,4] + m[5,5] + (5 \times 2 \times 7)\}$
$= \min\{0+84+210, 60+0+70\} = 130$

Sundaram
FOR EDUCATIONAL USE

$m[1,4] = \min\{m[1,1] + m[2,4] + (4\times10\times2), m[1,2] + m[3,4] + (4\times5\times2), m[1,3] + m[4,4] + (4\times6\times2)\}$

$= \min\{0 + 160 + 80, 200 + 60 + 40, 320 + 0 + 48\}$

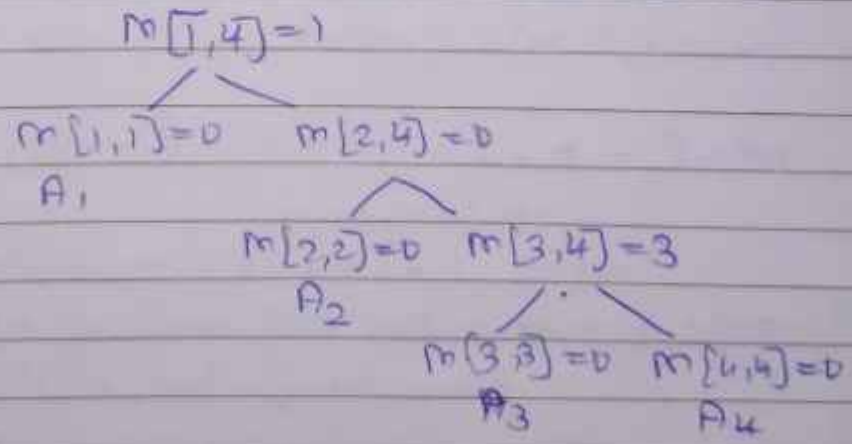
$= 240$

$m[2,5] = \min\{m[2,2] + m[3,5] + (10\times5\times7), m[2,3] + m[4,5] + (10\times6\times7), m[2,4] + m[5,5] + (10\times2\times7)\}$

$m[1,5] = \min\{m[1,1] + m[2,5] + (4\times10\times7), m[1,2] + m[3,5] + (4\times5\times7), m[1,3] + m[4,5] + (4\times6\times7), m[1,4] + m[5,5] + (4\times2\times7)\}$

$= \min\{0 + 340 + 280, 200 + 130 + 320 + 84 + 168, 240 + 0 + 56\}$

$= 296$



Optimum Sequence is given as

$(A_1 \times (A_2 \times (A_3 \times A_4)))$

Conclusion :- Thus, we have successfully implemented the Matrix Chain Multiplication.

FOR EDUCATIONAL USE

Name: Harsh Chhetriwal

DOMS | Page No.
Date / /

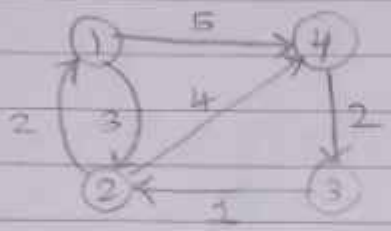
Design and Analysis of Algorithm

Experiment no. 7

Aim: Implementation of All Pair Shortest Path using Dynamic Programming

Question:

Apply all pair shortest path on the following graph



Solution:

At K = 0

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

Now, At k = 1

We know that $d^k[i,j] = \min(d^{k-1}[i,j], d^{k-1}[i,k] + d^{k-1}[k,j])$

$\therefore d^1[1,1] = \min(d^0[1,1], d^0[1,1] + d^0[1,1])$

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

$d^1[1,1] = 0$

$d^1[1,2] = \min(d^0[1,2], d^0[1,1] + d^0[1,2])$
$= \min(3, 0+3)$
$d^1[1,2] = 3$

$d^1[1,3] = \min(d^0[1,3], d^0[1,1] + d^1[1,3])$
$= \min(\infty, \infty)$
$d^1[1,3] = \infty$

$d^1[1,4] = \min[d^0[1,4], d^0[1,1] + d^0[1,4])$
$= \min(5,5)$
$d^1[1,4] = 5$

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

$d^1[2,3] = \min(\infty, 2+\infty) = \infty$

$d^1[2,4] = \min(4, 2+5) = 4$

$d^1[3,1] = \min[\infty, \infty+0) = \infty$

$d^1[3,2] = \min[\infty 1, \infty] = 1$

$d^1[3,4] = \min[\infty, \infty] = \infty$

$d^1[4,1] = \min[\infty, \infty] = \infty$
$d^1[4,2] = \min[\infty, \infty] = \infty$
$d^1[4,3] = \min[2, \infty] = 2$

DOMS | Page No.
Date / /

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

Now, at $k = 2$

We know that $d^k[i,j] = \min(d^{k-1}[i,j], d^{k-1}[i,k] + d^{k-1}[k,j])$

$d^2[1,2] = \min[3,3] = 3$

$d^2[1,3] = \min[\infty, 3+\infty] = \infty$
$d^2[1,4] = \min(5, 3+4) = 5$
$d^2[2,1] = \min(2, 0+2) = 2$
$d^2[2,3] = \min(\infty, 0+\infty) = \infty$
$d^2[2,4] = \min(4, 0+4) = 4$
$d^2[3,1] = \min(\infty, 1+2) = 3$
$d^2[3,2] = \min[1, 1+0) = 1$
$d^2[3,4] = \min(\infty, 1+4) = 5$
$d^2[4,1] = \min(\infty, \infty) = \infty$
$d^2[4,2] = \min(\infty, \infty) = \infty$
$d^2[4,3] = \min(2, \infty) = 2$

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

Now, at $K = 3$

We know, $d^k[i,j] = \min(d^{k-1}[i,j], d^{k-1}[i,j] + d^{k-1}[k,j])$

$d^3[1,2] = \min(3, \infty) = 3$
$d^3[1,3] = \min(\infty, \infty) = \infty$
$d^3[1,4] = \min(5, \infty) = 5$

x x x

$d^3[2,1] = \min(2, \infty) = 2$
$d^3[2,3] = \min(\infty, \infty) = \infty$
$d^3[2,4] = \min(4, \infty) = 4$
$d^3[3,1] = \min(3, 0+3) = 3$
$d^3[3,2] = \min(1, 0+1) = 1$
$d^3[3,4] = \min(5, 0+5) = 5$
$d^3[4,1] = \min(\infty, 2+3) = 5$
$d^3[4,2] = \min(\infty, 2+1) = 3$
$d^3[4,3] = \min(2, 2+0) = 2$

$\therefore D^3 =$

	1	2	3	4
1	0	3	∞	5
2	2	0	∞	4
3	3	1	0	5
4	5	3	2	0

Now, At $k=4$

We know, $d^k[i,j] = \min(d^{k-1}[i,j], d^{k-1}[i,k] + d^{k-1}[k,j])$

$d^4[1,2] = \min(3, 5+3) = 3$
$d^4[1,3] = \min(\infty, 5+2) = 7$
$d^4[1,4] = \min(5, 5+0) = 5$
$d^4[2,1] = \min(2, 4+5) = 2$
$d^4[2,3] = \min(\infty, 4+2) = 6$
$d^4[2,4] = \min(4, 4) = 4$
$d^4[3,1] = \min(3, 5+5) = 3$
$d^4[3,2] = \min(2, 5+3) = 2$
$d^4[3,4] = \min(5, 5+0) = 4$
$d^4[4,1] = \min(5, 0+5) = 5$
$d^4[4,2] = \min(3, 0+3) = 3$
$d^4[4,3] = \min(2, 0+2) = 2$

DOMS | Page No.
Date / /

$$\therefore D^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 7 & 5 \\ 2 & 0 & 6 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

∴ Shortest Path of the pairs are

1 → 1 = 0
1 → 2 = 3
1 → 3 = 7
1 → 4 = 5
2 → 1 = 2
2 → 2 = 0
2 → 3 = 6
2 → 4 = 4
3 → 1 = 3
3 → 2 = 1
3 → 3 = 0
3 → 4 = 5
4 → 1 = 5
4 → 2 = 3
4 → 3 = 2
4 → 4 = 0

Conclusion :- Thus, we have implemented All pair shortest Path using dynamic Programming.

Name: Harsh Chitaliya

Name: Harsh Chitaliya
SAP: 60018230079, Div: 5
Batch: A1D5, Batch: A1
Roll no: 5428

DOMS | Page No. 28
Date / /

Design and Analysis of Algorithm

Experiment No: 8

Aim: Implementation of N-Queen problem using backtracking.

Post lab questions :-

Solve 4-queen problem stepwise and derive its time complexity.

The constraints are,
We cannot place any two queens together in,
(1) Same row
(2) Same column
(3) diagonally

Let us have, the 4×4 board

(i)
Let us place our first queen at shown position
Also we cannot place other queen at 'x' place due to three constraints

Q1 x x x
x x
x x
x x

(ii) let us have second queen at first possible place in queen at column 2

Q1 x x x
x x x
x Q2 x x
x x x

Since there is no possible place in Column 3 for Q_3 We will have to backtrack and place Q_2 at next position

Let us place Q_3 in Column 3,

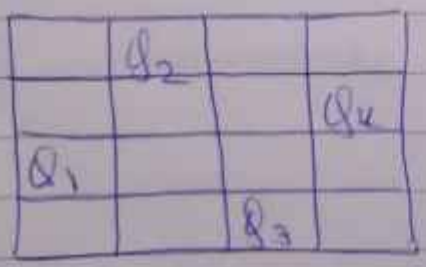
$\times \times Q_3 \times$
$Q_1 \times \times \times$
$\times \times \times$
$\times Q_2 \times \times$

④ Let us place Q_4 in possible place of Column 4

$\times \times Q_3 \times$
$Q_1 \times \times \times$
$\times \times \times Q_4$
$\times Q_2 \times \times$

The final position of Solution is

However, there can be more than one possible solution



DOMS | Page No.
Date / /

Time Complexity:

Let, n → number of queen
k → Current row being processed
T(n) → time complexity of algorithm

- At each row, there are n possible positions to place queen
- For each placement, the boolean function iterates through almost k-1 times previously placed queen.
- Recursion depth is n.

It can be expressed as,

$$T(n) = n(T(n-1) + O(n))$$

Expanding this recurrence relation given,

$$T(n) = n(n-1)\cdot(T(n-2) + O(n-1)) + n \cdot O(n)$$
$$= n(n-1)(n-2)(T(n-3) + O(n-2)) + n \cdot O(n-1) + nO(n)$$

On solving this relation we get,

$$T(n) = n!$$

Time complexity of N queen problem is O(n!) where n is no of queen

Conclusion: Thus, we successfully studied and implemented N queen problem

Name: Harsh Chitalya

Name: Harsh Chitalya
SAP ID: 60018220029
Roll No: 5028, Div: 5
Batch: A2, Batch: A2

DOMS | 5028
Date

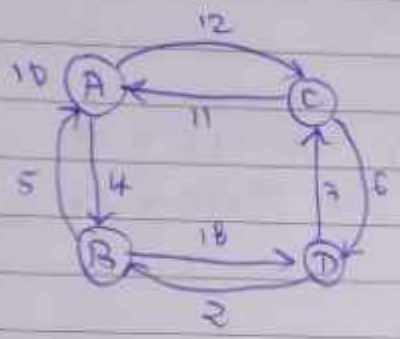
Design and Analysis of Algorithm

Experiment No: 9

Aim: To implement travelling Salesman problem

Post Lab questions

Q1]



Solve the travelling salesman problem starting Vertex A

Adjacency matrix is given by

C_{ij} =	A	B	C	D
A	0	4	12	7
B	5	0	∞	18
C	11	∞	0	6
D	10	2	3	0

1) $|S| = 0$ ... Empty Set

formula: $g(i, \phi) = C_{iA}$

$g(A, \phi) = C_{AA} = 0$

$g(B, \phi) = C_{BA} = 5$

$g(C, \phi) = C_{CA} = 11$

$g(D, \phi) = C_{DA} = 10$

2) $|S| = 1$

$g[i, S]$ function can be solved using following formula:

$$g[i, S] = \min_{j \in S} [C_{ij} + g[j, S - j]]$$

Calculation:

i) $i = B$

$g[B, \{C\}] = C_{BC} + g[C, \phi] = 20 + 11 = 31$

$g[B, \{D\}] = C_{BD} + g[D, \phi] = 18 + 10 = 28$

ii) $i = C$

$g[C, \{B\}] = C_{CB} + g[B, \phi] = 20 + 5 = 25$

$g[C, \{D\}] = C_{CD} + g[D, \phi] = 6 + 10 = 16$

iii) $i = D$

$g[D, \{B\}] = C_{DB} + g[B, \phi] = 2 + 5 = 7$

$g[D, \{C\}] = C_{DC} + g[C, \phi] = 3 + 11 = 14$

3) $|S| = 2$

i) $i = B$

$g[B, \{C, D\}] = \min \{ C_{BC} + g[C, \{D\}] = 20 + 16 = 36,\ C_{BD} + g[D, \{C\}] = 18 + 14 = 32 \}$

$\therefore g[B, \{C, D\}] = 32$

ii) $i = C$

$g[C, \{B, D\}] = \min \{ C_{CB} + g[B, \{D\}] = 20 + 28 = 48,\ C_{CD} + g[D, \{B\}] = 6 + 7 = 13 \}$

$g[C, \{B, D\}] = 13$

iii) $i = D$

$$g(D, \{B, C\}) = \min \begin{cases} C_{DB} + g(B, \{C\}) = 2 + \infty = \infty \\ C_{DC} + g(C, \{B\}) = 3 + \infty = \infty \end{cases}$$

$\therefore g(D, \{B, C\}) = \infty$

4) $|S| = 3$

$$g(A, \{B, C, D\}) = \min \begin{cases} C_{AB} + g(B, \{C, D\}), \\ C_{AC} + g(C, \{B, D\}), \\ C_{AD} + g(D, \{C, B\}). \end{cases}$$

$$= \min(36, 25, \infty)$$

$\therefore \boxed{g(A, \{B, C, D\}) = 25}$

Path is given by:

$g(A, \{B, C, D\}) \Rightarrow C_{AC} + g(C, \{B, D\})$

$\Rightarrow C_{CD} + g(D, \{B\})$

$\Rightarrow C_{DB} + g(B, \phi)$

$\Rightarrow C_{BA}$

$\therefore$ **Optimal path is $A \to C \to D \to B \to A$**

Name: Harsh Vitaliya
SAP ID: 60018220029, Roll no: 5028
D.Y.S, Branch: AIDS
Batch: B1

DOMS | Page No. 502B | Date / /

Design and Analysis of Algo
Experiment no 10

Aim: Implementation of Rabin Karp using string matching

Question:

Given a string 'str' and pattern 'pat', you have to find all occurences of the pattern in the string, you have to print the starting positions of all the occurences of pattern in the string. Also calculate number of spurious hits.

Example:

'str' = "hashkey"
'pat' = "key"

Step 1:
n = len(str) = 8
m = len(pat) = 3
hashkey = 17

Step 2:
h = 1
e = 2
y = 3
i = 4

Step 3

hashPattern = 123 % 17 = 4

Step 4 (Check for str)

i=0 hash('hey') = 123 % 17

found at index 0

countfound = 1

i=1 hash('eyh') = 231 % 17 = 10

hashPattern ≠ hash('eyh')

SpuriousHit = 0

i=2 hash('yhi') = 314 % 17 = 8

hashPattern ≠ hash(yhi)

i=3 hash('hih') = 141 % 17 = 5

hashPattern ≠ hash(hih)

i=4 hash('ihe') = 412 % 17 = 4

hashPattern = hash(hih)

∴ Spurious hit found at index 4

Spuroushit = 1

i=5 hash(hey) = 123 % 17 = 4

hashPattern = hash(Rey)

found at index 5

Count found = 2

DOMS | Page No.
Date / /

: Pattern found at index 0 and index 5
Spurious hit occured once at index 4
Pattern found twice.

Conclusion: Thus, we have successfully implemented the Robin Karp algorithm using string matching technique.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2023-2024

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithm	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

AIM: Implementation of Matrix chain multiplication using Dynamic Programming.

Problem Statement: Find the least cost of matrix multiplication using dynamic programming.

Algorithm:

Given a sequence of matrices, the goal is to find the most efficient way to multiply these matrices. The problem is not actually to perform the multiplications, but merely to decide the sequence of the matrix multiplications involved.

In Matrix Chain Multiplication Problem, we are given some matrices and are asked to multiply them in such a way that the total number of multiplication is minimum.

```
n  = length[p]-1
for i ← 1 to n
        do m [i, i] ← 0
for l ← 2 to n    // l is the chain length
      do for i ← 1 to n-l + 1
        do j ← i+ l -1
           m[i,j] ←  ∞
           for k  ← i to j-1
                do q  ← m [i, k] + m [k + 1, j] + pi-1 pk pj
                   If q < m [i,j]
                     then m [i,j] ← q
                          s [i,j] ← k
        return m and s.
```

TIME COMPLEXITY:

TimeComplexity: $O(N^3)$
Auxiliary Space: $O(N^2)$ ignoring recursion stack space

Code:

```
#include <bits/stdc++.h>
using namespace std;
vector<int> A(1000);
int dp[100][100];
int s[100][100];

int rec(int i, int j){
   if(dp[i][j] != 1000000){
      return dp[i][j];
   }
   if(i==j)return dp[i][j]=0;
   for(int k = i; k < j; k++){
      int cost = rec(i, k) + rec(k+1, j) + (A[i-1]*A[k]*A[j]);
      if(cost<dp[i][j]){
         dp[i][j] = cost;
         s[i][j]=k;
      }
   }
   return dp[i][j];
}

int main(){

   int n;
   cout<<"Enter the number of dimensions of matrix:";
   cin >> n;

   cout<<"Enter the dimensions of matrix:";
   for(int i = 0; i < n; i++)cin >> A[i];
   for(int i = 0; i <= n; i++){
      for(int j = 0; j <= n; j++){
         dp[i][j]=1000000;
         s[i][j]=-1;
      }
   }
   cout<<"Final Answer is:";
   cout << rec(1, n-1) << endl;
   for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
         if(dp[i][j]==1000000)
         {
            cout << "0 ";
         }
         else
         {
         cout << dp[i][j] << " ";
         }
      }
      cout << endl;
   }
   cout << endl;
   cout << "Values of K are as follows:";
   cout << endl;
   for(int i = 0; i < n; i++){
```

```
        for(int j = 0; j < n; j++){
            cout << s[i][j] << " ";
        }
        cout << endl;
    }

    return 0;

}
```

Output:

```
Enter the number of dimensions of matrix:4
Enter the dimensions of matrix:1 2 3 4
Final Answer is:18
0 0 0 0
0 0 6 18
0 0 0 24
0 0 0 0

Values of K are as follows:
-1 -1 -1 -1
-1 -1 1 2
-1 -1 -1 2
-1 -1 -1 -1
```

CONCLUSION: Thus, we have implemented Matrix chain multiplication using dynamic programming.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2023-2024

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design Analysis and Algorithm	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

Experiment 07

Aim: Implementation of All Pair Shortest Path using Dynamic Programming.

Code:

```
#include <stdio.h>
#define V 4
#define INF 99999
void printSolution(int dist[][V]);
void floydWarshall(int dist[][V])
{
   int i, j, k;
   for (k = 0; k < V; k++) {
      for (i = 0; i < V; i++) {
         for (j = 0; j < V; j++) {
            if (dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
         }
      }
   }
   printSolution(dist);
}
void printSolution(int dist[][V])
{
   printf(
      "The following matrix shows the shortest distances"
      " between every pair of vertices \n");
```

```
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == 99999)
                printf("%7s", "99999");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}
int main()
{
    printf("SUM1\n");
    int graph[V][V] = { { 0, 5, 99999, 2 },
                        { 99999, 0, 99999, 7 },
                        { 10, 99999, 0, 15 },
                        { 99999, 99999, 12 ,0 } };
    floydWarshall(graph);

    printf("SUM2\n");
    int graphs[V][V] = { { 0, 5, 99999, 10 },
                        { 99999, 0, 3, 99999 },
                        { 99999, 99999, 0, 1 },
                        { 99999, 99999, 99999, 0 } };

    // Function call
    floydWarshall(graphs);

    return 0;
}
```

Output:

```
SUM1
The following matrix shows the shortest distances between every pair of vertices
      0       5      14       2
     29       0      19       7
     10      15       0      12
     22      27      12       0
SUM2
The following matrix shows the shortest distances between every pair of vertices
      0       5       8       9
  99999       0       3       4
  99999   99999       0       1
  99999   99999   99999       0
```

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithm	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

EXPERIMENT NO. 8

AIM: Implementation of N-Queen Problem using backtracking

Problem Statement:

Implementation of N-Queen Problem using backtracking

Algorithm:

```
Nqueen(int k)
for(int i=1;i<=n;i++)
  if(place(k,i))       // calling place function
    {
     x[k]=i;
    if(k==n)
      {
          for(m=1;m<=n;m++)
        System.out.print(x[m]+"  ");

      System.out.println();
}
    else
    Nqueen(k+1);

boolean place(int k,int i)
{
for(int j=1;j<k;j++)
  {
```

```
        if(x[j]==i || Math.abs(x[j]-i)==Math.abs(k-j))
        return false;
    }
    return true;
}
```

}TIME COMPLEXITY:

TimeComplexity: $O(N^3)$

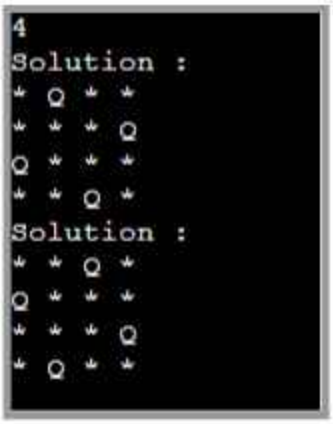
Code:

```
#include <bits/stdc++.h>
using namespace std;
int n;
void nqueen(int i, bool table[100][100], vector<pair<int, int> > v){
    if(i==n){
        char ans[n][n];
        for(int a = 0; a < n; a++){
            for(int b = 0; b < n; b++){
                ans[a][b] = '*';
            }
        }
        for(int p = 0; p < v.size(); p++){
            ans[v[p].first][v[p].second] = 'Q';
        }
        cout << "Solution : " << endl;
        for(int a = 0; a < n; a++){
            for(int b = 0; b < n; b++){
                cout << ans[a][b] << " ";
            }
            cout << endl;
        }
```

```
} else {
    for(int j = 0; j < n; j++){
        if(table[i][j]){
            bool copy[100][100];
            for(int i = 0; i < n; i++){
                for(int j = 0; j < n; j++){
                    copy[i][j] = table[i][j];
                }
            }
            for(int k = i+1; k<n; k++){
                copy[k][j] = false;
            }
            int l = i+1, m = j-1;
            while(l<n && m>=0){
                copy[l][m] = false;
                l++;
                m--;
            }
            l = i+1, m = j+1;
            while(l<n && m<n){
                copy[l][m] = false;
                l++;
                m++;
            }
            vector<pair<int, int> > v2 = v;
            v2.push_back({i, j});
            nqueen(i+1, copy, v2);
        }
    }
```

```
    }
}

int main(){
      cin >> n;
    bool table[100][100];
    vector<pair<int, int> > v;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            table[i][j] = true;
        }
    }
    nqueen(0, table, v);
    return 0;
  }
```

Output:



CONCLUSION: Thus, we have N-Queen Problem using backtracking.

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithm	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

EXPERIMENT NO. 9

AIM: Implementation of Traveling Salesperson Problem using branch and bound.

Problem Statement:

In a world of interconnected cities, a traveling salesperson is tasked with visiting a set of cities and returning to the starting city, aiming to minimize the total distance traveled. Each city is connected to some or all other cities by direct routes, and the distances between cities are known.
The goal is to find the shortest possible route that visits each city exactly once and returns to the starting city.

Algorithm:

The term *Branch and Bound* refer to all state-space search methods in which all the children of an E−node are generated before any other live node can become the E−node. E−node is the node, which is being expended. State−space tree can be expended in any method, i.e., BFS or DFS. Both start with the root node and generate other nodes. A node that has been generated and whose children are not yet been expanded is called a live-node. A node is called a dead node, which has been generated, but it cannot be expanded further. In this method, we expand the most promising node, which means the node which promises that expanding or choosing it will give us the optimal solution. So, we prepare the tree starting from the root and then expand it.

- Branch and bound is an effective way to find better, if not best, solution in quick time by pruning some of the unnecessary branches of search tree.
- It works as follow :

Consider directed weighted graph G = (V, E, W), where node represents cities and weighted directed edges represents direction and distance between two cities.

1. Initially, graph is represented by cost matrix C, where

   C_{ij} = cost of edge, if there is a direct path from city i to city j

   C_{ij} = ∞, if there is no direct path from city i to city j.

2. Convert cost matrix to reduced matrix by subtracting minimum values from appropriate rows and columns, such that each row and column contains at least one zero entry.

3. Find cost of reduced matrix. Cost is given by summation of subtracted amount from the cost matrix to convert it in to reduce matrix.

4. Prepare state space tree for the reduce matrix

5. Find least cost valued node A (i.e. E-node), by computing reduced cost node matrix with every remaining node.

6. If <i, j> edge is to be included, then do following :

(a) Set all values in row i and all values in column j of A to ∞

(b) Set A[j, 1] = ∞

(c) Reduce A again, except rows and columns having all ∞ entries.

7. Compute the cost of newly created reduced matrix as,

Cost = L + Cost(i, j) + r

Where, L is cost of original reduced cost matrix and r is A[i, j].

8. If all nodes are not visited then go to step 4.

Reduction procedure is described below :

Raw Reduction:

Matrix M is called reduced matrix if each of its row and column has at least one zero entry or entire row or entire column has ∞ value. Let M represents the distance matrix of 5 cities. M can be reduced as follow:

$M_{RowRed} = \{M_{ij} - \min \{M_{ij} \mid 1 \le j \le n, \text{ and } M_{ij} < \infty \}\}$

Raw Reduction:

Matrix M is called reduced matrix if each of its row and column has at least one zero entry or entire row or entire column has ∞ value. Let M represents the distance matrix of 5 cities. M can be reduced as follow:

$M_{RowRed} = \{M_{ij} - \min \{M_{ij} \mid 1 \le j \le n, \text{ and } M_{ij} < \infty \}\}$

Consider the following distance matrix:

M =

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

Find the minimum element from each row and subtract it from each cell of matrix.

$=$

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

Reduced matrix would be:

$M_{RowRed} =$

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

Row reduction cost is the summation of all the values subtracted from each rows:

Row reduction cost (M) = 10 + 2 + 2 + 3 + 4 = 21

Column reduction:

Matrix M_{RowRed} is row reduced but not the column reduced. Matrix is called column reduced if each of its column has at least one zero entry or all ∞ entries.

$M_{ColRed} = \{M_{ji} - \min \{M_{ji} \mid 1 \leq j \leq n, \text{ and } M_{ji} < \infty \}\}$

To reduced above matrix, we will find the minimum element from each column and subtract it from each cell of matrix.

$M_{RowRed} =$

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

Column reduced matrix M_{ColRed} would be:

M_{ColRed} =

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

Each row and column of M_{ColRed} has at least one zero entry, so this matrix is reduced matrix.

Column reduction cost (M) = 1 + 0 + 3 + 0 + 0 = 4

TIME COMPLEXITY:

TimeComplexity: Now this thing is tricky and need a deeper understanding of what we are doing. We are actually creating all the possible extenstions of E-nodes in terms of tree nodes. Which is nothing but a **permutation**. Suppose we have **N** cities, then we need to generate all the permutations of the **(N-1)** cities, excluding the root city. Hence the time complexity for generating the permutation is **O((n-1)!)**, which is equal to **O(2^(n-1))**.

Hence the final time complexity of the algorithm can be **O(n^2 * 2^n)**.

CODE:

```
#include <stdio.h>

#include <limits.h>

#include <math.h>  // Include the math library for pow function


#define MAX 9999

int n = 4;

int distan[20][20] = {

  { 0, 12, 18, 24 },

  { 12, 0, 36, 28 },

  { 18, 36, 0, 32 },

  { 24, 28, 32, 0 }};

int DP[16][4];  // Change the size of DP array to match the new approach
```

```
int TSP(int mark, int position) {
   int completed_visit = pow(2, n) - 1;  // Use the pow function to calculate 2 raised to the power of n
  if (mark == completed_visit) {
     return distan[position][0];
  }
  if (DP[mark][position] != -1) {
     return DP[mark][position];
  }
  int answer = MAX;
  for (int city = 0; city < n; city++) {
     if ((mark & (int)pow(2, city)) == 0) {  // Use the pow function to calculate 2 raised to the power of city
       int newAnswer = distan[position][city] + TSP(mark | (int)pow(2, city), city);  // Use the pow function to calculate 2 raised to the power of city
       answer = (answer < newAnswer) ? answer : newAnswer;
     }
  }
  return DP[mark][position] = answer;
}

void printPath(int mark, int position) {
   int completed_visit = pow(2, n) - 1;  // Use the pow function to calculate 2 raised to the power of n
  if (mark == completed_visit) {
     printf("%d ", position + 1);
     return;
  }
  int nextMark, city;
  int minDist = MAX;
  for (city = 0; city < n; city++) {
```

```
    if ((mark & (int)pow(2, city)) == 0) {  // Use the pow function to calculate 2 raised to the power of city
        int newMark = mark | (int)pow(2, city);  // Use the pow function to calculate 2 raised to the power of city
      int newDist = distan[position][city] + TSP(newMark, city);
      if (newDist == DP[mark][position]) {
        nextMark = newMark;
        break;
      }
    }
  }
  printf("%d ", position + 1);
  printPath(nextMark, city);
}

int main() {
   for (int i = 0; i < pow(2, n); i++) {  // Use the pow function to calculate 2 raised to the power of n
    for (int j = 0; j < n; j++) {
      DP[i][j] = -1;
    }
  }
  printf("Minimum Distance Travelled -> %d\n", TSP(1, 0));
  printf("Shortest Path Travelled ->  ");
  printPath(1, 0);
  printf("1");
  return 0;
}
```

Output:

Input Matrix:

{ 0, 12, 18, 24 },

{ 12, 0, 36, 28 },

{ 18, 36, 0, 32 },

{ 24, 28, 32, 0 }};

```
Minimum Distance Travelled -> 90
Shortest Path Travelled ->  1 2 4 3 1
```

CONCLUSION: Thus, we have implemented Traveling Salesperson Problem using branch and bound.

Name: Harsh Chitaliya	**Sap Id:** 60018220029
Course: Design and Analysis of Algorithm	**Roll No:** S028
Branch: Artificial Intelligence and Data Science	**Div:** S

EXPERIMENT NO. 10

AIM: Implementation of Rabin Karp using String matching.

Problem Statement:

You're given two strings, ***'text'*** of length ***'n'*** and ***'pattern'*** of length ***'m'***, consisting of lowercase characters.

Find all the occurrences of the string 'pattern' in 'text'.

For each occurrence, print the index from where it starts in the string 'text' (1 - indexed).

Example:

Input: 'text' = "cxyzghxyzvjkxyz" and 'pattern' = "xyz"

Output: 2 7 13

Algorithm:

In the Naive String Matching algorithm, we check whether every substring of the text of the pattern's size is equal to the pattern or not one by one.

Like the Naive Algorithm, the Rabin-Karp algorithm also check every substring. But unlike the Naive algorithm, the Rabin Karp algorithm matches the hash value of the pattern with the hash value of the current substring of text, and if the hash values match then only it starts matching individual characters. So Rabin Karp algorithm needs to calculate hash values for the following strings.

- Pattern itself
- All the substrings of the text of length m which is the size of pattern.

Algorithm RabinKarp(P, T)

```
%% Input: Pattern P and text T
%% Output: Occurrence of P in T
Begin
  m = length(P)
  n = length(T)
  d = 10
  q = 13
  p = 0
  t = 0

for j = 1 to m do
       p = (d * p + p[1]) mod q
      t = (d * t + t[i]) mod q
End for
hashP = p
hashT = t                                        %% Initial fingerprint
i = 1
while (i <= n - m + 1) do
  if (hashP = hashT) then
      if (P[1 .. m] = T(i + 1 .. i + m) then
      return(i)
      End if
   hashT = Compute fingerprint  fi+1 from fi using the rolling hash function
   end if
  i = i + 1
End while
End
```

Code:

```
#include<stdio.h>
#include<string.h>
#include<math.h>

#define d 256

int hashToDigit(long long int hash, int q)
{
   return hash % q;
}

void search(char *pattern, char *text, int q)
{
   int M = strlen(pattern);
   int N = strlen(text);
   int i, j;
   int p = 0;
   int t = 0;
   int h = 1;
   int spuriousHits = 0;
   int patternCount = 0;

   for (i = 0; i < M - 1; i++)
      h = (h * d) % q;

   for (i = 0; i < M; i++)
   {
      p = (d * p + pattern[i]) % q;
      t = (d * t + text[i]) % q;
   }
```

```
    for (i = 0; i <= N - M; i++)
    {
        if (p == t)
        {
            for (j = 0; j < M; j++)
            {
                if (text[i + j] != pattern[j])
                    break;
            }
            if (j == M)
            {
                printf("Pattern found at index %d\n", i);
                patternCount++;
            }
            else
            {
                printf("Spurious hit at index %d\n", i);
                spuriousHits++;
            }
        }

        if (i < N - M)
        {
            t = (d * (t - text[i] * h) + text[i + M]) % q;
            if (t < 0)
                t = (t + q);
        }
    }

    printf("Number of spurious hits: %d\n", spuriousHits);
    printf("Number of pattern occurrences found: %d\n", patternCount);
}

int main()
{
    char text[100];
    char pattern[100];
    int q;

    printf("Enter the text: ");
    scanf("%s", text);
    printf("Enter the pattern: ");
    scanf("%s", pattern);
    printf("Enter the value of q: ");
    scanf("%d", &q);

    int patternHashDigit = hashToDigit(123456789, q);

    printf("Hash value of pattern converted into digit: %d\n", patternHashDigit);

    search(pattern, text, q);

    return 0;
}
```

Output:

```
Enter the text: newwne
Enter the pattern: wn
Enter the value of q: 3
Hash value of pattern converted into digit: 0
Spurious hit at index 0
Spurious hit at index 1
Spurious hit at index 2
Pattern found at index 3
Spurious hit at index 4
Number of spurious hits: 4
Number of pattern occurrences found: 1
```

CONCLUSION: Thus, we have implemented Rabin Karp using String matching.

Name: Harsh Chitaliya
SAP ID: 60018220029
Roll no: 5028
Div: 5, Batch: A3
5028

DAA Assignment 1

Q1) Solve the following Occurrence Relation using Master's Theorem

$$T(n) = 2T\left(\frac{n}{4}\right) + n^{0.51}$$

Sol^n: Comparing with $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, we get

$$a=2, b=4, f(n) = n^{0.51}$$

Now,

$$n^{\log_b a} = n^{\log_4 2} = n^{0.5}$$

Here,

$$f(n) > n^{\log_b a} \quad \because [n^{0.51} > n^{0.5}]$$

$$\therefore T(n) = \Theta(f(n))$$

$$\therefore \boxed{T(n) = \Theta(n^{0.51})}$$

Q2) A machine needs a minimum of 200 sec to sort 1000 elements by quicksort The minimum time needed to sort 200 elements will be

→ Best case time complexity of Quick Sort is given by $O(n \log n)$

For 1000 elements

$$n \log_2 n = 1000 \times \log_2 1000$$
$$= 9965 \times 7842$$
$$\simeq 9966$$

So, 9966 comparisons are required to sort 1000 elements which takes 200 sec.

Sundaram FOR EDUCATIONAL USE

For 200 elements

$n\log_2 n = 200 \times \log_2 200$

$= 1528.77$

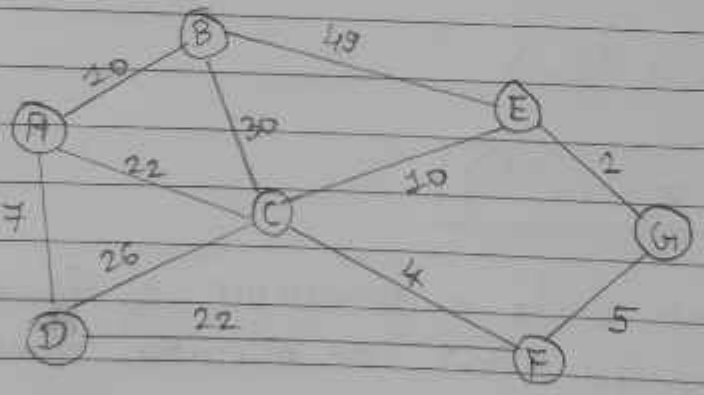
$= 1529$

So, 1529 comparisons are required to sort 200 elements

So time taken for 200 elements $= 200 \times \frac{1529}{9966}$

Time Taken for 200 elements = 30.6843 seconds.

Q3) Consider the undirected graph below Using Prim's algorithm find the order in which edges would be added to construct MST



Soln: Let A be source node

1. Initialize key-value as infinity for all vertices (except source) and an empty parent node as well

Vertex	A	B	C	D	E	F	G
Key value	0	∞	∞	∞	∞	∞	∞
Parent	–	–	–	–	–	–	–

FOR EDUCATIONAL USE

2. i) Selected Vertex : A

adjacent A → B : 10 ∴ 10 < ∞
A → C : 22 ∴ 22 < ∞
A → D : 7 ∴ 7 < ∞
} Update values

Vertex	A	B	C	D	E	F	G
Key value	0	10	22	7	∞	∞	∞
Parent	-	A	A	A	-	-	-

Selecting D ∴ 7 is the minimum

ii) Selecting node : D

adjacent arc : D → C : 26 ~~∴ 26 < 22~~
D → F : 22 22 < ∞ ∴ update it

Vertex	A	B	C	D	E	F	G
Key value	0	10	22	7	∞	22	∞
parent	-	A	A	A	-	D	-

Selecting F as it is minimum

iii) Selected node : F

Adjacent arc : F → C : 4 ∴ 4 < 22
F → G : 5 5 < ∞
} Update these values

Vertex	A	B	C	D	E	F	G
Key value	0	10	4	7	∞	22	5
Parent	-	A	F	A	-	D	F

Selecting C as it is minimum

Sundaram
FOR EDUCATIONAL USE

3] Selected node : C

Adjacent are : C → A : 22 ✗ 0

C → B : 30 ✗ 10

C → E : 10 < ∞ : update it.

Vertex	A	B	C	D	E	F	G
Key value	0	10	4	7	10	22	5
Parent	-	A	F	A	C	D	~~F~~

Selecting E as it is Minimum

4] Selected node : E

Adjacent are : E → G 2 < 5 .. update it

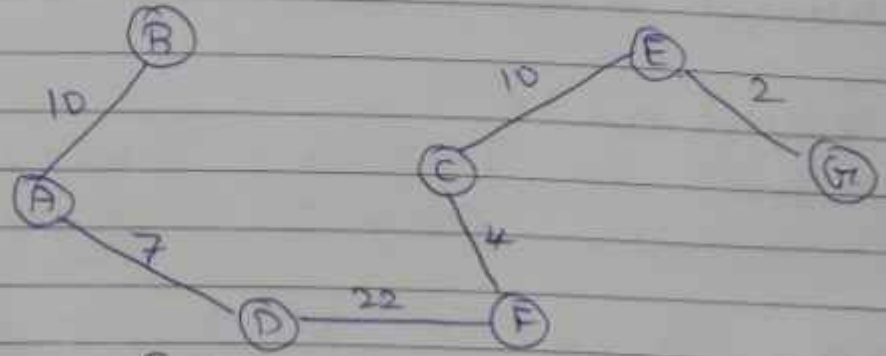
E → G 10 ✗ 4

Vertex	A	B	C	D	E	F	G
Key value	0	10	4	7	10	22	2
Parent	-	A	F	A	C	D	E

: There are no unVisited nodes from here

: We stop here

Find MST is



: Minimum Cost is : 10+7+22+4+10+2 = 55

FOR EDUCATIONAL USE

Q 4] Difference Between Greedy and Dynamic Approach

Ans

Dynamic	Greedy
1. There is guaranteed optimal solution as DP considered all possible cases and then choose best among them.	1. Provide no guarantee of getting optimum approach
2. It requires a table or cache for remembering and this increases its memory complexity	2. More memory efficient as it never looks back or revises its previous choices
3. Generally slower due to considering all possible cases and then possible choosing best among them	3. Generally faster
4. No feasible solution created	4. We get feasible solution at each step
5. Based on recurrent formula that uses some previously calculated states	5. Solves problem using heuristic values of mainly locally optimal choice
6. Ex: 0/1 knapsack, Bellman Ford.	6. Jobsequencing, fractional knapsack

Sundaram
FOR EDUCATIONAL USE

Q.5] Explain how 0/1 knapsack problem is used to maximize profit using set Method.

Ans
- As the name suggest items are indivisible here
- We cannot take fraction of any item
- We have to either take an item completely or leave it completely
- It is solved using dynamic programming.

Consider,
- Knapsack weight capacity = w
- Number of items each having some weight & value = n

Example

m = 8 .... Maximum weight of bag
n = 4
P = {1, 2, 5, 6}
w = {2, 3, 4, 5}

1) Creating set of (P, w)

S = {(0, 0)}
S_1^0 = {(1, 2)}

S = {(0, 0), (1, 2)}
S_1^1 = {(2, 3), (3, 5)}

S^2 = {(0, 0), (1, 2), (2, 3), (3, 5)}
S_1^2 = {(5, 4), (6, 6), (7, 7), (8, 9)}

FOR EDUCATIONAL USE

$S^3 = \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6), (7,7)\}$

$S_1^3 = \{(6,5), (7,7), (8,8)\}$

$S^4 = \{(0,6), (1,2), (2,3), (5,4), (6,5), (7,7), (8,8)\}$

Now find the item which will get deleted

① $(8,8) \in S^4$

but $(8,8) \in S^3$ $\therefore x_4 = 1$

$(8-6, 8-5) = (2,3)$

② $(2,3) \in S^3$, & $(2,3) \in S^2$

③ $(2,3) \in S^2$ but $(2,3) \notin S^1$

④ $(2-2, 3-3) = (0,0)$

$(0,0) \in S^1$ and $(0,0) \notin S^0$ $x_1 = 0$

Therefore deleted item are $\{(2,3), (6,5)\}$

Maximum profit $= 6+2 = 8$

FOR EDUCATIONAL USE

Name: Harsh Chitelya
SAP ID: 60018220023
Div: 5; Batch: A1
Branch: MPSTME
Div: 9

Design and Analysis of Algorithm

(5028)

DOMS | Page No.
Date

Q.1) n = 4

Let nodes = { a_1, a_2, a_3, a_4 }

P_i = { 4, 4, 2, 2 } ⇒ Successful probability

q_i = { 3, 4, 2, 2, 2 } ⇒ Unsuccessful probability

Table:

j →	0	1	2	3	4
0	$w_{00}=3$ $c_{00}=0$ $r_{00}=0$	$w_{11}=4$ $c_{11}=0$ $w_{11}=0$	$w_{22}=2$ $c_{22}=0$ $r_{22}=0$	$w_{33}=2$ $c_{33}=0$ $r_{33}=0$	$w_{44}=2$ $c_{44}=0$ $r_{44}=0$
1	$w_{01}=11$ $c_{01}=11$ $r_{01}=a_1$	$w_{12}=10$ $c_{12}=10$ $r_{12}=a_2$	$w_{23}=6$ $c_{23}=6$ $r_{23}=a_3$	$w_{34}=6$ $c_{34}=6$ $r_{34}=a_4$	
2	$w_{02}=17$ $c_{02}=27$ $r_{02}=a_1$	$w_{13}=14$ $c_{13}=24$ $r_{13}=a_2$	$w_{24}=10$ $c_{24}=16$ $r_{24}=a_3/a_4$		
3	$w_{03}=21$ $c_{03}=31$ $r_{03}=a_2$	$w_{14}=18$ $c_{14}=34$ $r_{14}=a_2/a_3$			
4	$w_{04}=25$ $c_{05}=32$ $r_{04}=a_2$				

⇒ Step 1: i=0 to 4 j = i+0

$w_{00} = q_0 = 3$, $w_{11} = q_1 = 4$, $w_{22} = q_2 = 2$, $w_{33} = q_3 = 2$,
$c_{00} = 2$ $c_{11} = 0$ $c_{22} = 0$
$w_4 = q_4 = 2$

Step 2: i=0 to 3 j = i+1

1) i = 0 ; j = 1

$w[0,1] = w[0,0] + P_i + q_i = 3+4+4 = 11$

$c[0,1] = \min_{0<k\le 1} \{ c[0,0] + c[1,1] \} + w[0,1]$

$= 0 + 0 + 11$

$= 11$

$r[0,1] = a_k = a_i$

ii) $i=1;\ j=2$

$w[1,2] = w[1,1] + P_2 + q_2 = 4+4+2 = 10$

$c[1,2] = \min_{1<k\le 2} \{ c[1,1] + c[2,2] \} + w[1,2]$

$= 0 + 0 + 10$

$= 10$

$r[1,2] = a_k = a_2$

iii) $i=2, j=3$

$w[2,3] = w[2,2] + P_3 + q_3 = 2+2+2 = 6$

$c[2,3] = c[2,2] + c[3,3] + w[2,3] = 0+0+6 = 6$

$r[2,23] = a_k = a_3$

iv) $i=3, j=4$

$w[3,4] = w[3,3] + P_4 + q_4 = 2+2+2 = 6$

$c[3,4] = c[3,3] + c[4,4] + w[3,4] = 0+0+6 = 6$

$r[3,4] = a_k = a_4$

Step 3: $i = 0$ to 2 $\quad j = i+2$

i) $i=0,\ j=2$

$w[0,2] = w[0,1] + P_2 + q_2 = 1+4+4+2 = 17$

$c[0,2] = \min_{0<k\le 2} \begin{cases} k=1 \\ c[0,0] + c[1,2] = 10 \\ k=2 \\ c[0,1] + c[2,2] = 11 \end{cases} + w[0,2] = 10 + 17 = 27$

$r[1,2] = q_k = q_1$

2) $i=1,\ j=3$

$$C[1,3] = \min_{1<k\le 3} \begin{cases} k=2 \\ C[1,1) + C[2,3) = 0 \\ k=3 \\ C[1,2) + C[3,3] = 20 \end{cases} + w[1,3) = 6 + 14 = 27$$

$r[1,2] = q_k = q_2$

3) $i=2,\ j=4$

$w[2,4] = w[2,3) + q_3 + p_4 = 6 + 2 + 2 = 10$

$$C[2,4] = \min_{0<k\le 3} \begin{cases} k=1 \\ C[0,0) + C[1,3) = 20 \\ k=2 \\ C[0,1) + C[2,3) = 17 \\ k=3 \\ C[0,2) + C[3,3] = 27 \end{cases} + w[0,3) = 38$$

$r[0,3) = q_k = q_2$

Step 4: $i=0$ to 2, $j = i+3$

1) $i=0,\ j=3$

$w[0,3) = w[0,2) + p_3 + q_3 = 17 + 2 + 2 = 21$

$$C[0,3) = \min_{0<k\le 3} \begin{cases} k=1 \\ C[0,0) + C[1,3) = 20 \\ k=2 \\ C[0,1) + C[2,3) = 17 \\ k=3 \\ C[0,2) + C[3,3] = 27 \end{cases} + w[0,3) = 38$$

DOMS Page
Date / /

Step 5: $i=0, j=4$

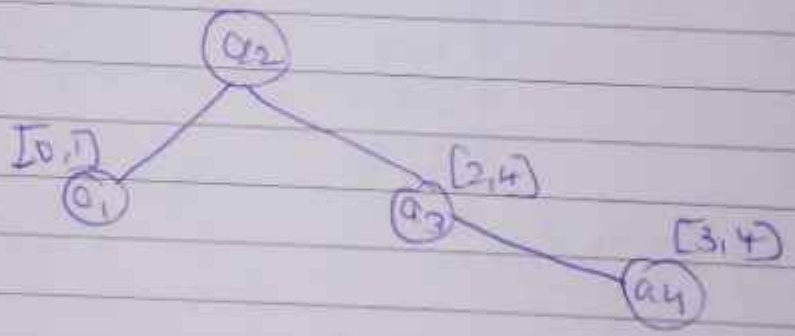
$w(0,4) = w[0,3] + p_4 + q_4 = 21 + 2 + 2 = 25$

$$C[0,4) = \min_{0<k\leq 4} \begin{cases} k=1 \Rightarrow C[0,0) + C[1,4) = 34 \\ k=2 \Rightarrow C[0,1) + C[2,4) = 27 \\ k=3 \Rightarrow C[0,2) + C[3,4) = 33 \\ k=4 \Rightarrow C[0,3) + C[4,4) = 33 \end{cases} + w[0,4]$$

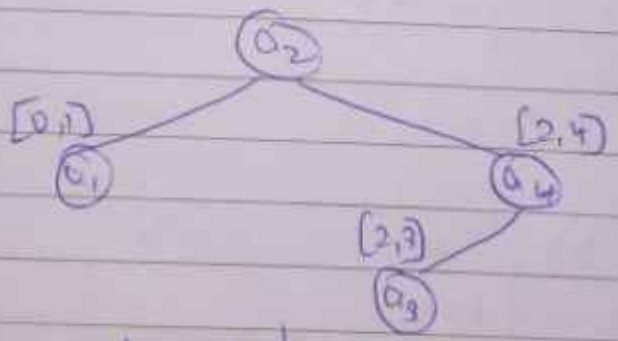
$= 52$

Our final optimal binary search tree will look like

1)



2)



We got two optimal trees because there were two node for a_2 which would give minimum values $\{C_{24}\}$

DOMS | Page No.
Date / /

Q3] A polynomial verifiable time problem, often abbreviated as NP refers to a class of decision problems in Computer Science

Characteristics

(i) Easy Verification: While the problem itself might be different to solve efficiently, the solution can be verified quickly

(ii) Decision Problems: NP problems are typically "yes" or no questions. They require determining whether a given solution exists or not

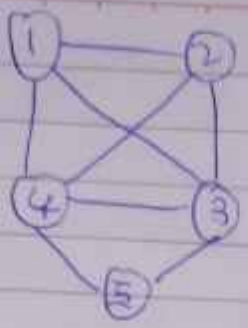
Example: 3-Coloring Problem

Imagine a map with countries sharing borders. We want to color the map with 3 colors such that no two adjacent countries share the same color

Solving: Finding the 3 coloring might be difficult especially for complex maps.

Verification: However, if someone presents a 3-coloring, we can easily check if each adjacent country has a different color. This verification takes a constant amount of time regardless of the map size.

Q4]



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

Least element in every rows is:

1 : 10
2 : 0
3 : 2
4 : 3
5 : 4 $\quad \Sigma r = 19$

	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	∞
least	1	-	3	-	-

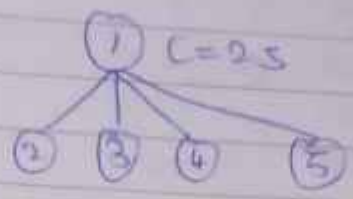
$\Sigma c = 4$

After Column Reduction

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix} \end{array}$$

Sum of row and column reduction $= C = 21+4 = 25$

Cost of 1st node $= C(1) = 25$



ii) Consider for cost of (1,2), set row & column of (1,2) to ∞

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	-
2	∞	∞	11	2	0	-
3	0	∞	∞	0	2	-
4	15	∞	12	∞	0	-
5	11	∞	0	12	∞	-
least	-	-	-	-	-	

Cost of node 2 =

$\hat{c}(2)$ = Sum of row & column reduction + $A[1,2]$ + $\hat{c}(1)$

$= 0 + 10 + 25$

$\hat{c}(2) = 35$

iii) For (1,3)

Make row 1 & 3rd column infinity & Set (3,1) to

	1	2	3	4	5	least
1	∞	∞	∞	∞	∞	-
2	12	∞	∞	2	0	-
3	∞	3	∞	0	2	-
4	16	3	∞	∞	0	-
5	11	0	∞	12	∞	-
least	11	-	-	-	-	

After column & row reduction

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	∞	2	0
3	∞	3	∞	0	2
4	4	3	∞	∞	0
5	0	0	∞	12	∞

Cost for $c^{\wedge}(3)$ = $C(3) + A(1,3) + c^{\wedge}(1)$

= 11 + 17 + 25

$c^{\wedge}(3) = 53$

For (1,4)

					least
∞	∞	∞	∞	∞	-
12	∞	11	∞	0	-
0	3	∞	∞	2	-
∞	3	12	∞	0	-
11	0	0	∞	∞	-
least	-	-	-	-	-

Cost of node 4 = $c^{\wedge}(4)$ = $C(4) + A(1,4) + c^{\wedge}(1)$

= 0 + 0 + 25

$c^{\wedge}(4) = 25$

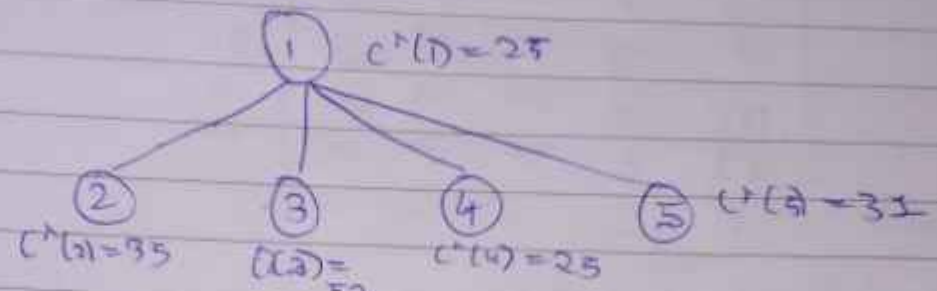
For (1,5)

					least
∞	∞	∞	∞	∞	-
12	∞	11	2	∞	2
0	3	∞	0	∞	-
15	3	12	∞	∞	3
∞	0	0	12	∞	-
least	-	-	-	-	-

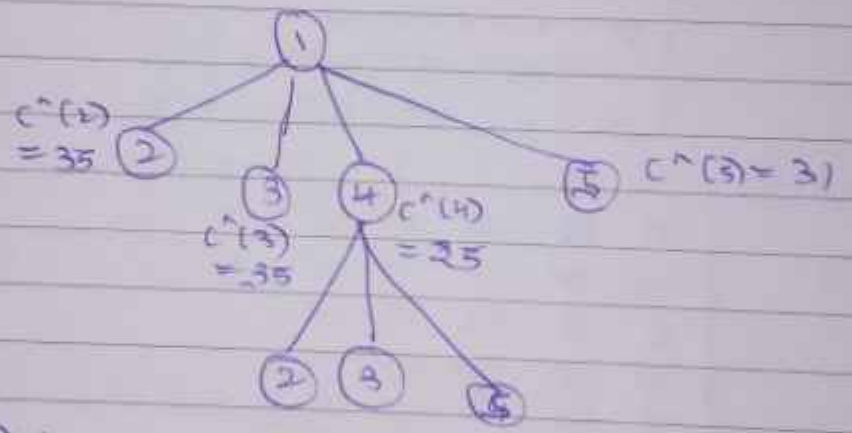
Cost of node 5 = $c'(5) = r(5) + A(1,5) + c'(1)$

$= 5 + 1 + 25$

$c'(5) = 31$



Now, out of all four child nodes, the cost of node 4 is minimum so will expand node 4



For (4,2):

					least
∞	∞	∞	∞	∞	–
∞	∞	11	∞	0	–
0	∞	∞	∞	2	–
∞	∞	∞	∞	∞	–
11	∞	0	∞	∞	–
least	–	–	–	–	–

Cost of $(4,2) = c'(4,2) = r(4) + A[4,2] + c'(2)$

$= 0 + 3 + 25$

$c'(4,2) = 28$

For (4,3)

					least
∞	10	∞	∞	∞	–
12	∞	∞	∞	0	–
∞	3	∞	∞	2	2
10	∞	∞	∞	∞	–
11	0	∞	∞	∞	–
least	11	–	–	–	–

After column Reduction

∞	10	∞	∞	∞	–
1	∞	∞	∞	0	–
∞	1	∞	∞	0	–
∞	∞	∞	∞	∞	–
0	0	∞	∞	∞	–

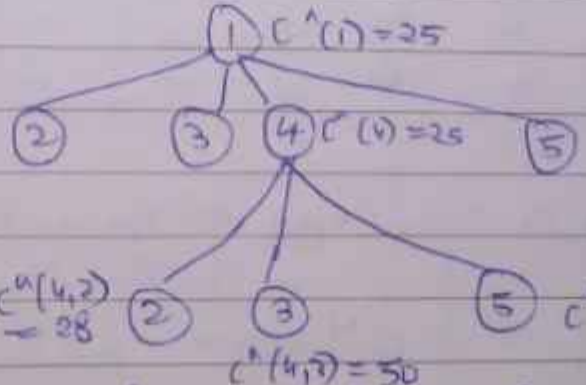
For (4,5)

					least
∞	10	∞	∞	∞	–
12	∞	11	∞	∞	2
0	3	∞	∞	∞	–
∞	∞	∞	∞	∞	–
∞	0	0	∞	∞	–
least	–	–	–	–	–

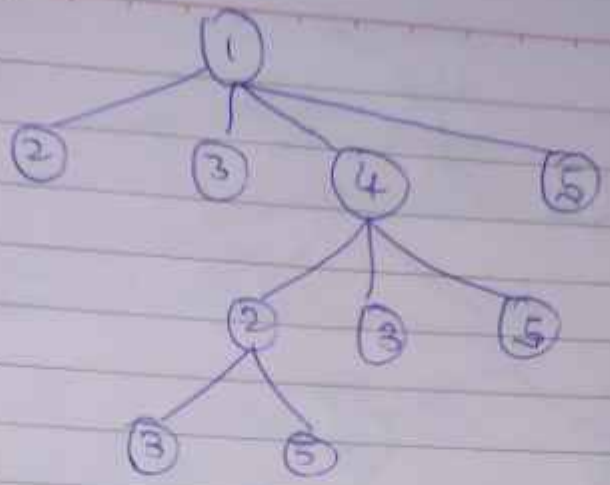
Cost of (4,5) = $c^*(4,5) = \Sigma L + A(4,5) + c^*(4)$

$= 11 + 0 + 25$

$c^*(4,5) = 36$



Now among all live nodes node 2 has minimum cost so we will extend it.



For (2,3)

Now, for this, we will use matrix of (4,2) Make 2nd row & 3rd column infinity along with (3,1)

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

$$11 \quad - \quad - \quad - \quad -$$

Cost of C(2,3) = $\sum C + A(4,2) + C'(2)$

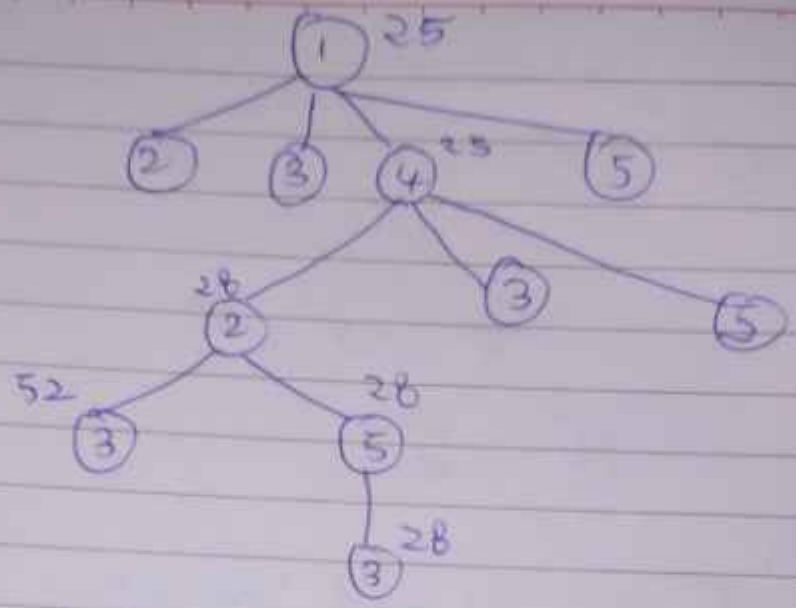
$= 13 + 11 + 28$

$C(2,3) = 52$

For (2,5)

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

Cost of $\bar{(2,5)} = C\bar{(2,5)} = 0 + 0 + 28 = 28$



Now, 28 is the minimum cost among the live nodes, which is of node 5, & we will extend till it.
We will take the matrix of (2,5)
Make 5th row & 3rd column infinity

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

Cost of $(5,3) = C'(5,3) = 0+0+28 = 28$

Path: $1 \to 4 \to 2 \to 5 \to 3 \to 1$

Total Cost $= 25+25+28+28+28 = \underline{130}$

DOMS | Page No.
Date / /

Q.5) Given a string T and pattern 'P' as follows
T: bacbaba babacaca
P: ababaca
Execute KMP algorithm

⇒ let n = length of String T = 15
m = length of string P = 7
Constructing Prefix function table for P

index	1	2	3	4	5	6	7
letter	a	b	a	b	a	c	a
π	0	0	1	2	3	0	1

Applying KMP algorithm:
let i=1, j=0

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	b	a	c	b	a	b	a	b	a	b	a	c	a	c	a

P	a	b	a	b	a	c	a
j=0	1	2	3	4	5	6	7

(1) i=1, j=0
T[i] ≠ P[j] & j=0
∴ i++

(2) i=2, j=0
T[i] = P[j]
∴ i++, j++

(3) $i=3, j=1$
$T[3] \neq P[2]$
$\therefore i=4, j=0$

(4) $i=4, j=0$
$T[4] \neq P[1]$ & $j=0$
$i=5$ ($\because i++$)

(5) $i=5, j=0$
$T[5] = P[1]$ & $j=0$
$i++, j++$

(6) $i=6, j=1$
$T[6] = P[2]$
$i++, j++$

(7) $i=7, j=2$
$T[7] = P[3]$
$i++, j++$

(8) $i=8, j=3$
$T[8] = P[4]$
$i++, j++$

(9) $i=9, j=4$
$T[9] = P[5]$
$i++, j++$

(10) $i=10, j=5$
$T[10] \neq P[6]$
$j = \pi[5]$

(11) $i=10, j=3$
$T[10] = P[4]$
$i++, j++$

(12) $i=11, j=4$
$T[11] = P[5]$
$i++, j++$

(13) $i=12, j=5$
$T[12] = P[6]$
$i++, j++$

(14) $i=13, j=6$
$T[13] = P[7]$
$i++, j++$
$\therefore j = 7-m$ $\therefore$ Pattern found at index $= i-m$
$= 13-7+1=0$
also $j=0$

(15) $i=14, j=0$
$T[14] \neq P[1]$
$i++$

(16) $i=15, j=0$
$T[15] = P[1]$ & $j=0$
$i++, j++$
$\therefore i=16 < 1 > 15$

We stop algorithm here.

Result: Yes, 'P' occur in 'T' at position 0

DOMS | Page No.
Date / /

Q3]

→ To prove that the knapsack problem is NP-hard, we'll use the reduction technique we'll reduce a known NP-hard problem to the knapsack problem demonstrating that if we could efficiently solve the knapsack problem, we could also efficiently solve the other NP-hard problem

We'll choose the subset sum problem, which is a classic NP-hard problem The subset sum problem can be noted as follows.

Given a set of integers and a target sum, can we find a subset of integers that adds up to the target sum?

1) Subset sum to knapsack reduction

Given an instance of the subset problem with set of integers $S = \{a_1, a_2, \ldots a_n\}$ and a target T, we will construct an instance of the knapsack problem

For each integer a_i in the set S, create an item, with weight a_i and value a_i

Set the capacity of knapsack of T

2) Explanation:

If there exits a subset of integers in S that sums up to T, then we can select those corresponding item in the knapsack to fill it exactly to T

Conversely, if we can fill the knapsack to its capacity exactly with items from S, then the total weight of those items will be equal to T Therefore, the exists a subset of S that sums up to T

3) Complexity Analysis:

The reduction from subset sum to knapsack can be done in polynomial time, as it involves simply constructing a new instance with some Modifications

4) Conclusion:

By Reducing the Subset Sum problem, which is NP hard, to the Knapsack problem in polynomial time, we have demonstrated that if we could solve the Knapsack problem efficiently, we could also solve the Subset Sum problem efficiently.

Therefore, the Knapsack problem is NP-hard.