FACULTY LABORATORY MANUAL

ALGORITHMS DESIGN AND ANALYSIS LAB

for

II Year II Sem. B.Tech. (IT)

Academic Year 2019-20

PREPARED BY

N S S ROJA RANI
Assistant Professor

P. V.V. S. D Nagendrudu
Assistant Professor

Department of Information Technology

Accredited by **NAAC** with **"A"** Grade
Recognised by **UGC** under section 2(f) &12(B)
Approved by **AICTE** - NEW Delhi
Permanently Affiliated to **JNTUK, SBTET**
Ranked as **"A" Grade** by Govt. of A.P.

SASI PUBLISHING HOUSE

CONTENT

Vision of the Institute:

Confect as a premier institute for professional education by creating technocrats who can address the society`s needs through inventions and innovations.

Mission of the Institute:

1. Partake in the national growth of technological, industrial arena with societal responsibilities.

2. Provide an environment that promotes productive research.

3. Meet stakeholders expectations through continued and sustained quality improvements.

Vision of the Program

To become recognized centre of excellence for quality IT Education and create professionals with ability to solve social needs.

Mission of the Program

1. To provide quality teaching learning environment that build necessary skills for employability and career development.
2. To conduct trainings/events for overall development of stakeholders with collaborations
3. To impart value education to students to serve society with high integrity and good character

PEOs & POs

Program Educational Objectives

This education is meant to prepare our students to thrive and to leadin their careers, our graduates

P1	Graduates will have strong knowledge about IT applications with leadership qualities
P2	Graduates will pursue successful career in IT and allied industries and provide solutions for global needs
P3	Graduates with life-long learning attitude and practice professional ethics.

Program Outcomes

Students in the Information Technology program should acquire the following outcomes during the time of their graduation

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

1. **Mobile & Web Application Development:** Ability to develop mobile & web applications using J2EE, Android and J2ME.
2. **Cloud Services:** Ability to develop virtualized and cloud based services in the organization

Preface

The Algorithms Design And Analysis Laboratory manual for II year II-Semester is strictly written as per SITE18 regulation of Sasi Institute of Technology and Engineering, Tadepalligudem .

For thorough understanding, viva questions are included at the end of each experiment. We hope that this manual will be useful for the students of Information Technology program of various universities for gaining deep knowledge in Design and Analysis of Algorithms Lab. It also would be useful for faculty as a ready reference.

The authors are thankful to the Management, Principal and Head of the IT department of Sasi Institute of Technology & Engineering, Tadepalligudem for their continuous encouragement in completing this manual.

Any suggestions for further improvement of the manual will be acknowledged and appreciated.

Authors

Lab Objectives:

1. Analyze the asymptotic performance of algorithms.
2. Write rigorous correctness proofs for algorithms.
3. Demonstrate a familiarity with major algorithms and data structures.
4. Apply important algorithmic design paradigms and methods of analysis.
5. Synthesize efficient algorithms in common engineering design situations

Lab Outcomes:

After the completion of this course student will be able

1. Apply the dynamic-programming paradigm when an algorithmic design situation calls for it.
2. Analyze worst-case running times of algorithms based on asymptotic analysis ,justify the correctness of algorithms.
3. Apply the greedy paradigm when an algorithmic design situation calls for it.
4. Apply the divide-and-conquer paradigm when an algorithmic design situation calls for it.
5. Illustrate the dynamic programming algorithms, and to determine its computational complexity.

	Course Outcomes to Program Outcomes Mapping:														
CO	**PO**												**PSO**		
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**
1	1	2	3	-	-	-	-	-	-	-	-	-	-	-	-
2	1	3	3	-	-	-	-	-	-	-	-	-	-	-	-
3	1	2	3	-	-	-	-	-	-	-	-	-	-	-	-
4	1	2		-	-	-	-	-	-	-	-	-	-	-	-
5	1	2	3	-	-	-	-	-	-	-	-	-	-	-	-
Course	**1**	**2**	3	-	-	-	-	-	-	-	-	-	-	-	-

SYSTEM REQUIREMENTS:

- Desktop PC with P-III or above, minimum of 166MHZ or faster processor with at least
- 64MB RAM and 100MB free disk space.
- Code blocks

GUIDELINES TO THE STUDENTS

The objective of the laboratory is on-hand experience and learning. The programs are designed to illustrate phenomena in different concepts of Algorithms Design And Analysis Lab . Perform the programs with interest and an attitude of learning which lay the path to business applications.

1. The students should bring the observation book, notes, record etc., to every lab session.
2. The students should come to the lab with a prior preparation to complete the programs within time.
3. Lab attendance will play a part of the internal assessment marks.
4. The programs and output must be first executed and write in the observation / manual book. The student should get acknowledged the same by the concerned teacher before leaving the lab. Later the programs should be recorded in the record sheets and again it should be signed by the concerned teacher. The student must submit the record in the following week for correction.
5. Any student failing to complete the programs to be evaluated in the same laboratory class will lose the Internal Assessment Marks for those programs.
6. The student shall be responsible for the system issued and take necessary safety measures in using it.
7. At the mid and end of the semester, lab exams will be conducted.
8. Strict discipline should be maintained inside the laboratory.
9. Do not sit idle with system if it is not under working condition. Report any problem with system to the Lab Instructor.
10. Shut down the system properly, turn-off the monitor, arrange the chair properly and then leave.

Academic Year 2019-20
II Year B.Tech–II Sem
Algorithms Design And Analysis Lab
List of experiments to be conducted

No. of Credits: 2

Experiment 1 (Dynamic Programming Technique)

a) Longest common Subsequence
b) Develop Optimal Binary search trees

Experiment 2 (Dynamic Programming Technique)

a) 0/1 Knap Sack Problem ,
b) The Traveling Salesperson Problem.

Experiment 3 (Greedy Methods)

a) Huffman codes
b) Knap Sack Problems

Experiment 4 (Greedy Methods)

a) Tree Vertex Splitting
b) Job Sequencing with Dead Lines

Experiment 5 (Back Tracking Techniques)

a) 8-Queens Problem
b) Sum of Sub sets

Experiment 6 (Back Tracking Techniques)

a) Graph Coloring.
b) Hamiltonian Cycles

Experiment 7 (Back Tracking Techniques)

a) 0/1 Knap Sack Problem

Experiment 8 (Branch and Bound)

a) 0/1 Knap Sack Problem
b) Traveling Sales Person Problem

Experiment 9 (Graph Algorithms)

a) Breadth First Search
b) Depth First Search

Experiment 10 (Graph Algorithms)

a) Kruskal`s Algorithm
b) Prim`s Algorithms

Experiment 11 (Graph Algorithms)

a) Bellman Ford Algorithm
b) Dijkstra`s Algorithm

Experiment 12 (Graph Algorithms)

a) Floyd- Warshall Algorithm.

sasi autonomous INSTITUTE OF TECHNOLOGY & ENGINEERING

Accredited by NAAC with "A" Grade
Recognised by UGC under section 2(f) &12(B)
Approved by AICTE - NEW Delhi
Permanently Affiliated to JNTUK, SBTET
Ranked as "A" Grade by Govt. of A.P.

Department of Information Technology

II B.Tech II Sem IT (ACADAMIC YEAR 2019-20)
Analysis Design and Algorithm Lab

<u>No of students per session</u>

S.No	CLASS/ SEC	Sessions	Student Roll No	No. of students per Session
1	II B. Tech II Sem IT	1st	18K61A1201-17K61A1226	25
2		2nd	18K61A1227-17K61A1250	24

<u>Lab Schedule for II-IT</u>

S.No	PROGRAMS/ EXPERIMENT	Session 1 (9:55 AM -12:50PM)		Session 2(9:55 AM – 12:50PM)	
		Tentative Date	Conducted Date	Tentative Date	Conducted Date
1	Introduction	02/12/2019		06/12/2019	
2	Experiment 1	09/12/2019		13/12/2019	
3	Experiment 2	16/12/2019		20/12/2019	
4	Experiment 3	23/12/2019		27/12/2019	
5	Experiment 4	30/12/2019		03/01/2020	
6	Experiment 5	06/01/2020		10/01/2020	
7	Experiment 6	13/01/2020		17/01/2020	
8	Experiment 7	20/01/2020		24/01/2020	
9	Experiment 8	03/02/2020		07/02/2020	
10	Experiment 9	10/02/2020		14/02/2020	
11	Experiment 10	17/02/2020		28/02/2020	
12	Experiment 11	24/02/2020		06/03/2020	
13	Experiment 12	02/03/2020		13/03/2020	
14	Mini Project	09/03/2020		20/03/2020	
15	Lab Internal	16/03/2020		27/03/2020	

LAB INCHARGE **HEAD OF THE DEPARTMENT**

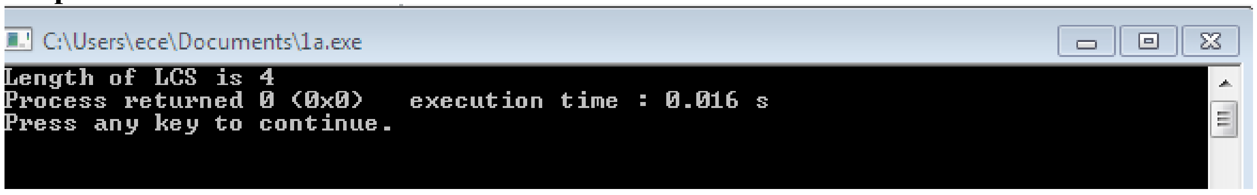
Week 1 (Dynamic Programming Technique)

a) Longest common Subsequence

Source code :

```
#include<stdio.h>
int max(int a, int b);
/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n )
{
   int L[m+1][n+1];
   int i, j;
   /* Following steps build L[m+1][n+1] in bottom up fashion. Note
 *       that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
   for (i=0; i<=m; i++)
   {
     for (j=0; j<=n; j++)
     {
       if (i == 0 || j == 0)
         L[i][j] = 0;
       else if (X[i-1] == Y[j-1])
         L[i][j] = L[i-1][j-1] + 1;
       else
         L[i][j] = max(L[i-1][j], L[i][j-1]);
     }
   }
   /* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] */
   return L[m][n];
}
int max(int a, int b)
{
    return (a > b)? a : b;
}
int main()
{
  char X[] = "AGGTAB";
  char Y[] = "GXTXAYB";
   int m = strlen(X);
  int n = strlen(Y);
  printf("Length of LCS is %d", lcs( X, Y, m, n ) );
  return 0;
}
```

Output :



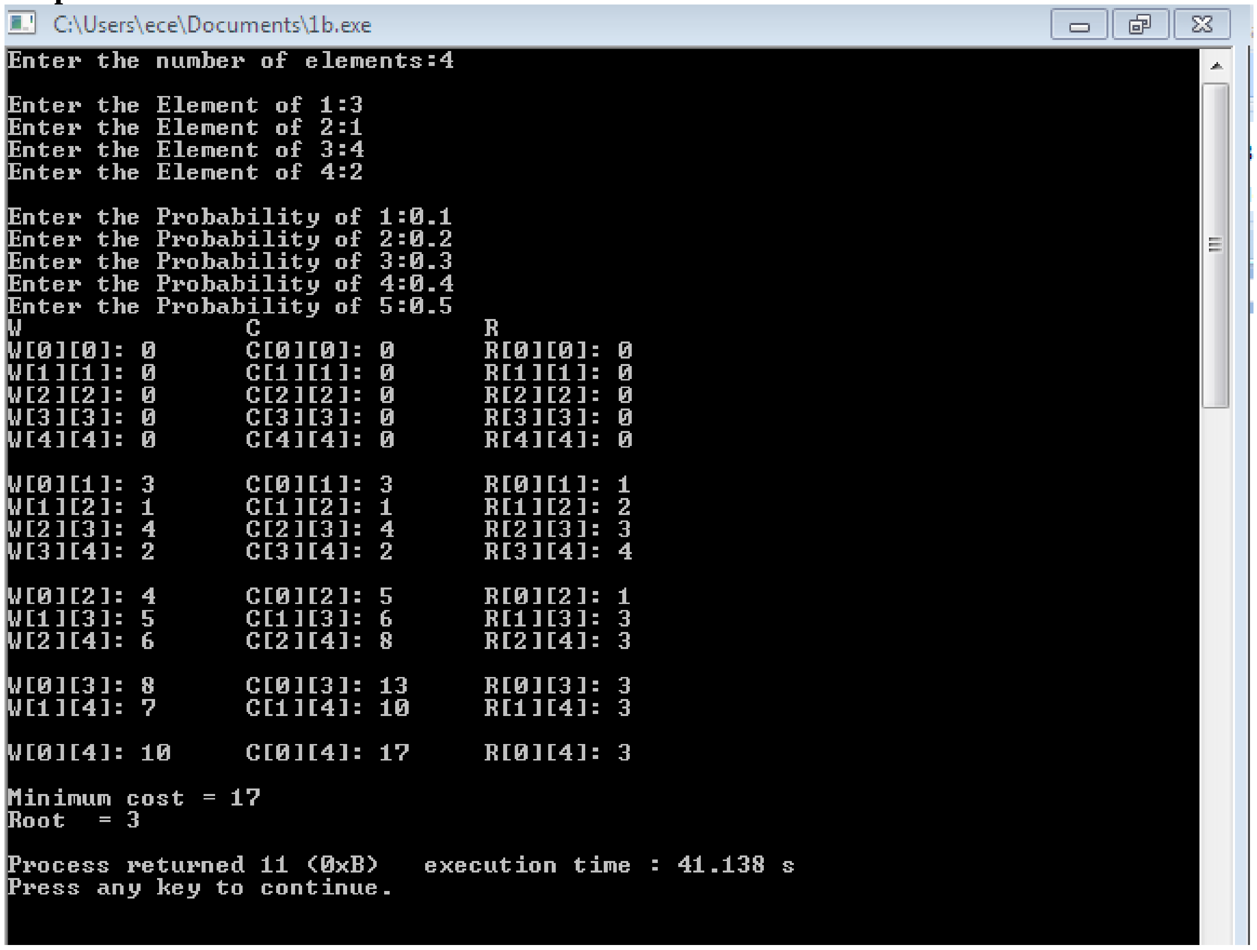
```
Length of LCS is 4
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

b) Develop Optimal Binary search trees
Source code

```
#include<stdio.h>
#define MAX 10
void main()
{
 char ele[MAX][MAX];
 int w[MAX][MAX], c[MAX][MAX], r[MAX][MAX], p[MAX], q[MAX];
 int temp=0, root, min, min1, n;
 int i,j,k,b;
 printf("Enter the number of elements:");
 scanf("%d",&n);
 printf("\n");
 for(i=1; i <= n; i++)
 {
  printf("Enter the Element of %d:",i);
  scanf("%d",&p[i]);
 }
        printf("\n");
 for(i=0; i <= n; i++)
 {
  printf("Enter the Probability of %d:",i);
  scanf("%d",&q[i]);
 }
 printf("W\t\tC\t\tR\n");
 for(i=0; i <= n; i++)
 {
 for(j=0; j <= n; j++)
 {
  if(i == j)
   {
  w[i][j] = q[i];
  c[i][j] = 0;
  r[i][j] = 0;
  printf("W[%d][%d]: %d\tC[%d][%d]: %d\tR[%d][%d]:
%d\n",i,j,w[i][j],i,j,c[i][j],i,j,r[i][j]);
}
 }
}
 printf("\n");
 for(b=0; b < n; b++)
 {
  for(i=0,j=b+1; j < n+1 && i < n+1; j++,i++)
   {
  if(i!=j && i < j)
   {
    w[i][j] = p[j] + q[j] + w[i][j-1];
    min = 30000;
    for(k = i+1; k <= j; k++)
```

```
          {
           min1 = c[i][k-1] + c[k][j] + w[i][j];
           if(min > min1)
           {
            min = min1;
            temp = k;
           }
          }
          c[i][j] = min;
          r[i][j] = temp;
         }
         printf("W[%d][%d]: %d\tC[%d][%d]: %d\tR[%d][%d]:
        %d\n",i,j,w[i][j],i,j,c[i][j],i,j,r[i][j]);
                  }
         printf("\n");
        }
        printf("Minimum cost = %d\n",c[0][n]);
        root = r[0][n];
        printf("Root  = %d \n",root);
       return 0;
       }
```

Output



```
Enter the number of elements:4

Enter the Element of 1:3
Enter the Element of 2:1
Enter the Element of 3:4
Enter the Element of 4:2

Enter the Probability of 1:0.1
Enter the Probability of 2:0.2
Enter the Probability of 3:0.3
Enter the Probability of 4:0.4
Enter the Probability of 5:0.5
W                C                R
W[0][0]: 0       C[0][0]: 0       R[0][0]: 0
W[1][1]: 0       C[1][1]: 0       R[1][1]: 0
W[2][2]: 0       C[2][2]: 0       R[2][2]: 0
W[3][3]: 0       C[3][3]: 0       R[3][3]: 0
W[4][4]: 0       C[4][4]: 0       R[4][4]: 0

W[0][1]: 3       C[0][1]: 3       R[0][1]: 1
W[1][2]: 1       C[1][2]: 1       R[1][2]: 2
W[2][3]: 4       C[2][3]: 4       R[2][3]: 3
W[3][4]: 2       C[3][4]: 2       R[3][4]: 4

W[0][2]: 4       C[0][2]: 5       R[0][2]: 1
W[1][3]: 5       C[1][3]: 6       R[1][3]: 3
W[2][4]: 6       C[2][4]: 8       R[2][4]: 3

W[0][3]: 8       C[0][3]: 13      R[0][3]: 3
W[1][4]: 7       C[1][4]: 10      R[1][4]: 3

W[0][4]: 10      C[0][4]: 17      R[0][4]: 3

Minimum cost = 17
Root   = 3

Process returned 11 (0xB)   execution time : 41.138 s
Press any key to continue.
```

viva questions :

1. Define optimal binary search tree with an example
2. What are the conditions for an optimal binary search tree and what is its advantage?
3. Define longest common subsequence problem.
4. Consider the strings “PQRSTPQRS” and “PRATPBRQRPS”. What is the length of the longest common subsequence?
5. What is the time complexity of the brute force algorithm used to find the longest common subsequence?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor

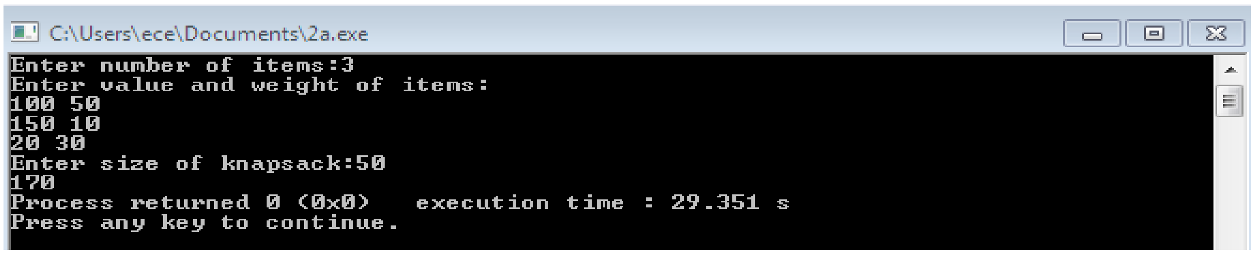
Week 2 (Dynamic Programming Technique)

a) **0/1 Knap Sack Problem**

Source code

```
#include<stdio.h>
int max(int a, int b) { return (a > b)? a : b; }
int knapSack(int W, int wt[], int val[], int n)
{
  int i, w;
  int K[n+1][W+1];
  for (i = 0; i <= n; i++)
  {
     for (w = 0; w <= W; w++)
     {
        if (i==0 || w==0)
           K[i][w] = 0;
        else if (wt[i-1] <= w)
             K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
        else
             K[i][w] = K[i-1][w];
     }
  }
  return K[n][W];
}
int main()
{
   int i, n, val[20], wt[20], W;
   printf("Enter number of items:");
   scanf("%d", &n);

   printf("Enter value and weight of items:\n");
   for(i = 0;i < n; ++i){
      scanf("%d%d", &val[i], &wt[i]);
   }
   printf("Enter size of knapsack:");
scanf("%d", &W);
   printf("%d", knapSack(W, wt, val, n));
   return 0;
}
```

Output :



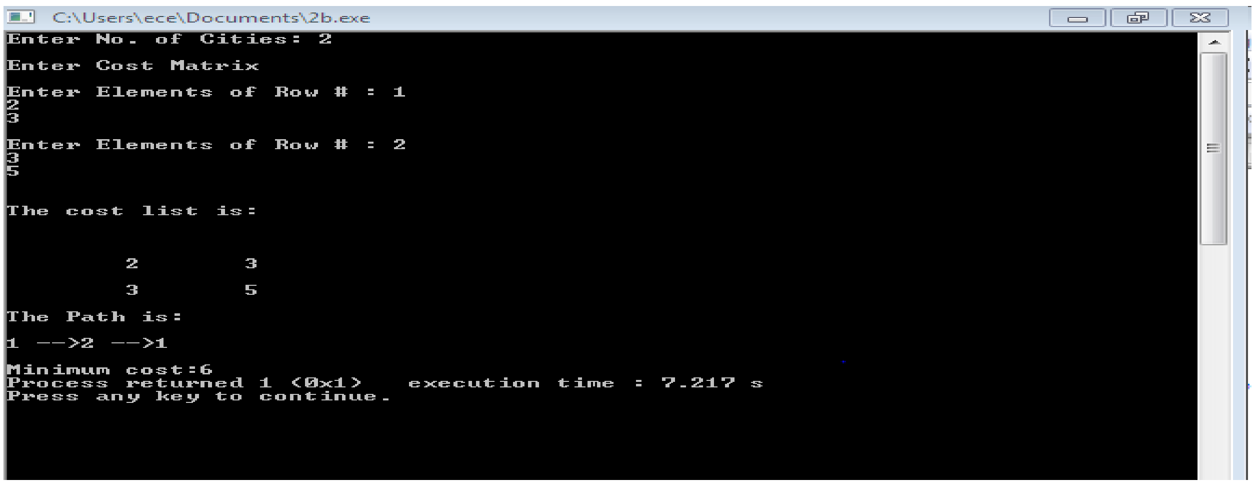
C:\Users\ece\Documents\2a.exe

```
Enter number of items:3
Enter value and weight of items:
100 50
150 10
20 30
Enter size of knapsack:50
170
Process returned 0 (0x0)   execution time : 29.351 s
Press any key to continue.
```

b) **<u>The Traveling Salesperson Problem.</u>**
<u>Source code</u>

```
#include<stdio.h>
#include<conio.h>
int a[10][10],visited[10],n,cost=0;

void get()
{
        int i,j;
        printf("Enter No. of Cities: ");
        scanf("%d",&n);
        printf("\nEnter Cost Matrix\n");
        for(i=0;i < n;i++)
        {
                printf("\nEnter Elements of Row # : %d\n",i+1);
                for( j=0;j < n;j++)
                        scanf("%d",&a[i][j]);
                visited[i]=0;
        }
        printf("\n\nThe cost list is:\n\n");
        for( i=0;i < n;i++)
        {
                printf("\n\n");
                for(j=0;j < n;j++)
                        printf("\t%d",a[i][j]);
        }
}

void mincost(int city)
{
        int i,ncity;
        visited[city]=1;
        printf("%d -->",city+1);
        ncity=least(city);
        if(ncity==999)
        {
                ncity=0;
                printf("%d",ncity+1);
                cost+=a[city][ncity];
                return;
        }
        mincost(ncity);
}

int least(int c)
{
        int i,nc=999;
        int min=999,kmin;
        for(i=0;i < n;i++)
        {
                if((a[c][i]!=0)&&(visited[i]==0))
                        if(a[c][i] < min)
                        {
                                min=a[i][0]+a[c][i];
                                kmin=a[c][i];
                                          nc=i;
                                }
          }
          if(min!=999)
                    cost+=kmin;
          return nc;
}

void put()
{
          printf("\n\nMinimum cost:");
          printf("%d",cost);
}
void main()
{
          clrscr();
          get();
          printf("\n\nThe Path is:\n\n");
          mincost(0);
          put();
          getch();
}
```

Out put



C:\Users\ece\Documents\2b.exe

```
Enter No. of Cities: 2

Enter Cost Matrix

Enter Elements of Row # : 1
2
3

Enter Elements of Row # : 2
3
5


The cost list is:


        2        3

        3        5
The Path is:

1 -->2 -->1

Minimum cost:6
Process returned 1 (0x1)   execution time : 7.217 s
Press any key to continue.
```

Viva questions:

1. What is 0/1 knapsack problem?
2. What is the objective of the knapsack problem?
3. You are given a knapsack that can carry a maximum weight of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. What is the maximum value of the items you can carry using the knapsack?
4. Define TSP ?
5. What is Travelling salesman problem and how is it modeled as a graph problem?
6. What are the applications of Travelling salesman problem?
7. What is the time complexity of Travelling salesman problem?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor

Week 3 (Greedy Methods)

a) **Huffman codes**

Source code :

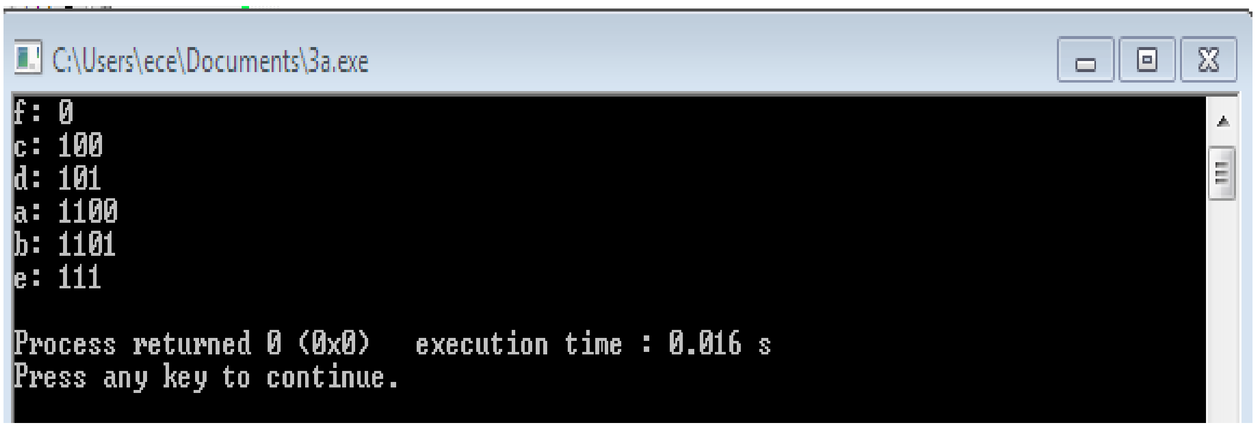
```
#include <stdio.h>
#include <stdlib.h>
  // This constant can be avoided by explicitly  calculating height of Huffman Tree
#define MAX_TREE_HT 100
  // A Huffman tree node
struct MinHeapNode {
      // One of the input characters
    char data;
      // Frequency of the character     unsigned freq;

    // Left and right child of this node
    struct MinHeapNode *left, *right;
};

// A Min Heap:  Collection of
// min-heap (or Huffman tree) nodes
struct MinHeap {

    // Current size of min heap
    unsigned size;

    // capacity of min heap
    unsigned capacity;

    // Array of minheap node pointers
    struct MinHeapNode** array;
};

// A utility function allocate a new
// min heap node with given character
// and frequency of the character
struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp
        = (struct MinHeapNode*)malloc
(sizeof(struct MinHeapNode));

    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;

    return temp;
}

// A utility function to create
// a min heap of given capacity
struct MinHeap* createMinHeap(unsigned capacity)

{

   struct MinHeap* minHeap
      = (struct MinHeap*)malloc(sizeof(struct MinHeap));

   // current size is 0
   minHeap->size = 0;

   minHeap->capacity = capacity;

   minHeap->array
      = (struct MinHeapNode**)malloc(minHeap->
capacity * sizeof(struct MinHeapNode*));
   return minHeap;
}

// A utility function to
// swap two min heap nodes
void swapMinHeapNode(struct MinHeapNode** a,
                    struct MinHeapNode** b)

{

   struct MinHeapNode* t = *a;
   *a = *b;
   *b = t;
}

// The standard minHeapify function.
void minHeapify(struct MinHeap* minHeap, int idx)

{

   int smallest = idx;
   int left = 2 * idx + 1;
   int right = 2 * idx + 2;

   if (left < minHeap->size && minHeap->array[left]->
freq < minHeap->array[smallest]->freq)
      smallest = left;

   if (right < minHeap->size && minHeap->array[right]->
freq < minHeap->array[smallest]->freq)
      smallest = right;

   if (smallest != idx) {
      swapMinHeapNode(&minHeap->array[smallest],
                    &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

// A utility function to check
// if size of heap is 1 or not
int isSizeOne(struct MinHeap* minHeap)
{

    return (minHeap->size == 1);
}

// A standard function to extract
// minimum value node from heap
struct MinHeapNode* extractMin(struct MinHeap* minHeap)

{

    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0]
        = minHeap->array[minHeap->size - 1];

    --minHeap->size;
    minHeapify(minHeap, 0);

    return temp;
}

// A utility function to insert
// a new node to Min Heap
void insertMinHeap(struct MinHeap* minHeap,
                   struct MinHeapNode* minHeapNode)

{

    ++minHeap->size;
    int i = minHeap->size - 1;

    while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {

        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }

    minHeap->array[i] = minHeapNode;
}

// A standard function to build min heap
void buildMinHeap(struct MinHeap* minHeap)
```

```
{

   int n = minHeap->size - 1;
   int i;

   for (i = (n - 1) / 2; i >= 0; --i)
       minHeapify(minHeap, i);
}

// A utility function to print an array of size n
void printArr(int arr[], int n)
{
   int i;
   for (i = 0; i < n; ++i)
       printf("%d", arr[i]);

   printf("\n");
}

// Utility function to check if this node is leaf
int isLeaf(struct MinHeapNode* root)

{

   return !(root->left) && !(root->right);
}

// Creates a min heap of capacity
// equal to size and inserts all character of
// data[] in min heap. Initially size of
// min heap is equal to capacity
struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size)

{

   struct MinHeap* minHeap = createMinHeap(size);

   for (int i = 0; i < size; ++i)
       minHeap->array[i] = newNode(data[i], freq[i]);

   minHeap->size = size;
   buildMinHeap(minHeap);

   return minHeap;
}

// The main function that builds Huffman tree
struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size)
```

```
{
   struct MinHeapNode *left, *right, *top;

   // Step 1: Create a min heap of capacity
   // equal to size.  Initially, there are
   // modes equal to size.
   struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);

   // Iterate while size of heap doesn't become 1
   while (!isSizeOne(minHeap)) {

      // Step 2: Extract the two minimum
      // freq items from min heap
      left = extractMin(minHeap);
      right = extractMin(minHeap);

      // Step 3:  Create a new internal
      // node with frequency equal to the
      // sum of the two nodes frequencies.
      // Make the two extracted node as
      // left and right children of this new node.
      // Add this node to the min heap
      // '$' is a special value for internal nodes, not used
      top = newNode('$', left->freq + right->freq);

      top->left = left;
      top->right = right;

      insertMinHeap(minHeap, top);
   }

   // Step 4: The remaining node is the
   // root node and the tree is complete.
   return extractMin(minHeap);
}

// Prints huffman codes from the root of Huffman Tree.
// It uses arr[] to store codes
void printCodes(struct MinHeapNode* root, int arr[], int top)

{

   // Assign 0 to left edge and recur
   if (root->left) {

      arr[top] = 0;
      printCodes(root->left, arr, top + 1);
   }

   // Assign 1 to right edge and recur
```

```
    if (root->right) {

        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    // If this is a leaf node, then
    // it contains one of the input
    // characters, print the character
    // and its code from arr[]
    if (isLeaf(root)) {

        printf("%c: ", root->data);
        printArr(arr, top);
    }
}

// The main function that builds a
// Huffman Tree and print codes by traversing
// the built Huffman Tree
void HuffmanCodes(char data[], int freq[], int size)

{
    // Construct Huffman Tree
    struct MinHeapNode* root
        = buildHuffmanTree(data, freq, size);

    // Print Huffman codes using
    // the Huffman tree built above
    int arr[MAX_TREE_HT], top = 0;

    printCodes(root, arr, top);
}

// Driver program to test above functions
int main()
{

    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int freq[] = { 5, 9, 12, 13, 16, 45 };

    int size = sizeof(arr) / sizeof(arr[0]);

    HuffmanCodes(arr, freq, size);

    return 0;
}
```

Output :


C:\Users\ece\Documents\3a.exe

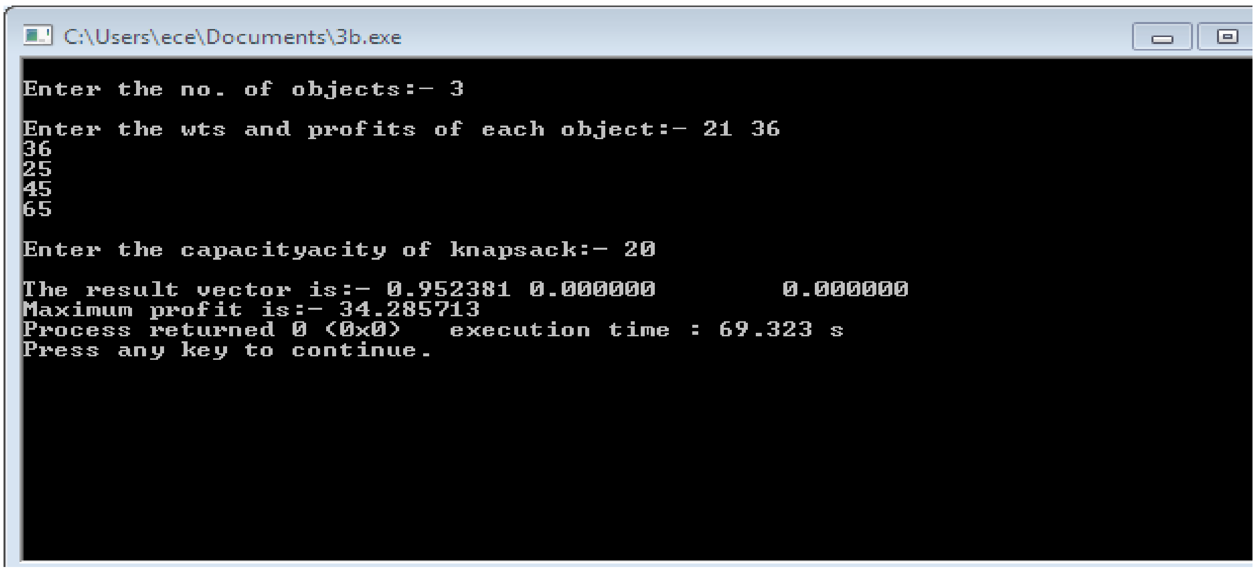
```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

b) <u>Knap Sack Problems</u>
<u>Source code :</u>

```
# include<stdio.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;

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

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

  printf("\nEnter the capacityacity 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);
  return(0) ;
}
```

Output :



```
Enter the no. of objects:- 3

Enter the wts and profits of each object:- 21 36
36
25
45
65

Enter the capacityacity of knapsack:- 20

The result vector is:- 0.952381 0.000000        0.000000
Maximum profit is:- 34.285713
Process returned 0 (0x0)   execution time : 69.323 s
Press any key to continue.
```

viva questions :

1. Which algorithm is the best approach for solving Huffman codes?
2. The type of encoding where no character code is the prefix of another character code is called?
3. What is meant by greedy method.
4. How do you make a Huffman code?
5. What is Huffman coding algorithm?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

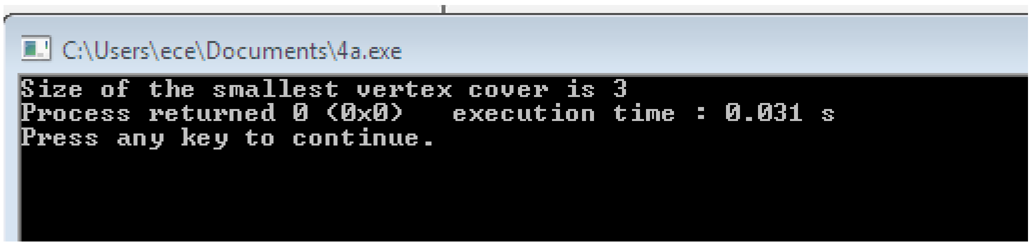
Signature of the Instructor

Week 4 (Greedy Methods)

a) **Tree Vertex Splitting**
Source code

```
#include <stdio.h>

#include <stdlib.h>

// A utility function to find min of two integers

int min(int x, int y) { return (x < y)? x: y; }

/* A binary tree node has data, pointer to left child and a pointer to
   right child */
struct node
{
    int data;
    struct node *left, *right;
};

// The function returns size of the minimum vertex cover
int vCover(struct node *root)
{
    // The size of minimum vertex cover is zero if tree is empty or there
    // is only one node
    if (root == NULL)
        return 0;
    if (root->left == NULL && root->right == NULL)
        return 0;

    // Calculate size of vertex cover when root is part of it
    int size_incl = 1 + vCover(root->left) + vCover(root->right);

    // Calculate size of vertex cover when root is not part of it
    int size_excl = 0;
    if (root->left)
      size_excl += 1 + vCover(root->left->left) + vCover(root->left->right);
    if (root->right)
      size_excl += 1 + vCover(root->right->left) + vCover(root->right->right);

    // Return the minimum of two sizes
    return min(size_incl, size_excl);
}

// A utility function to create a node
struct node* newNode( int data )
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    // Let us construct the tree given in the above diagram
    struct node *root         = newNode(20);
    root->left               = newNode(8);
    root->left->left         = newNode(4);
    root->left->right        = newNode(12);
    root->left->right->left   = newNode(10);
    root->left->right->right  = newNode(14);
    root->right              = newNode(22);
    root->right->right       = newNode(25);
    printf ("Size of the smallest vertex cover is %d ", vCover(root));
    return 0;
}
```

<u>Output</u>



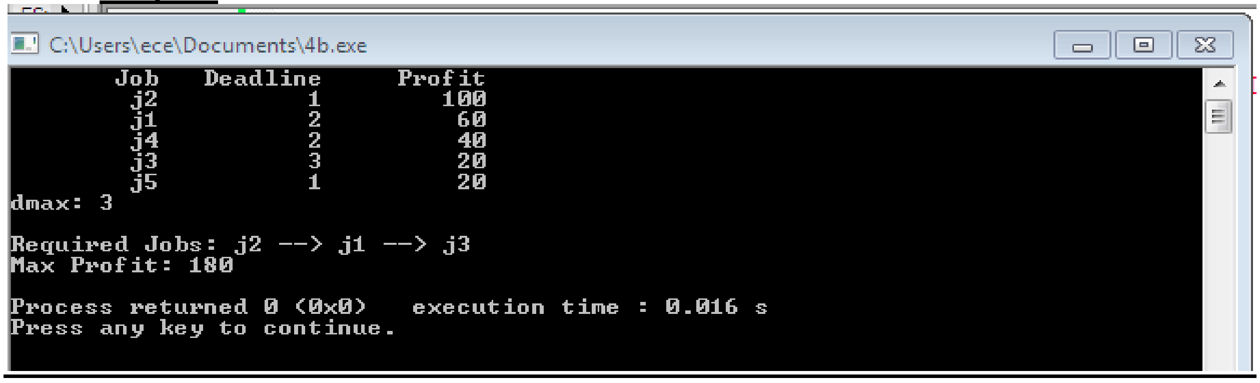
```
C:\Users\ece\Documents\4a.exe
Size of the smallest vertex cover is 3
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

b) <u>Job Sequencing with Dead Lines</u>
<u>Source code</u>

```
#include <stdio.h>
#define MAX 100
typedef struct Job {
  char id[5];
  int deadline;
  int profit;
} Job;
void jobSequencingWithDeadline(Job jobs[], int n);
int minValue(int x, int y) {
  if(x < y) return x;
  return y;
}
int main(void) {
  int i, j;
  Job jobs[5] = {
    {"j1", 2,  60},
    {"j2", 1, 100},
    {"j3", 3,  20},
    {"j4", 2,  40},
    {"j5", 1,  20},
  };
    Job temp;
    int n = 5;
  for(i = 1; i < n; i++) {
    for(j = 0; j < n - i; j++) {
      if(jobs[j+1].profit > jobs[j].profit) {
        temp = jobs[j+1];
        jobs[j+1] = jobs[j];
        jobs[j] = temp;
      }
    }
  }
  printf("%10s %10s %10s\n", "Job", "Deadline", "Profit");
  for(i = 0; i < n; i++) {
    printf("%10s %10i %10i\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);
  }
  jobSequencingWithDeadline(jobs, n);
  return 0;
}
void jobSequencingWithDeadline(Job jobs[], int n) {
  int i, j, k, maxprofit;
  int timeslot[MAX];
  int filledTimeSlot = 0;
  int dmax = 0;
  for(i = 0; i < n; i++) {
    if(jobs[i].deadline > dmax) {
      dmax = jobs[i].deadline;
    }
  }
  for(i = 1; i <= dmax; i++) {
    timeslot[i] = -1;
  }
  printf("dmax: %d\n", dmax);
  for(i = 1; i <= n; i++) {
    k = minValue(dmax, jobs[i - 1].deadline);
    while(k >= 1) {
      if(timeslot[k] == -1) {
        timeslot[k] = i-1;
        filledTimeSlot++;
        break;
      }
      k--;
    }
    if(filledTimeSlot == dmax) {
      break;
    }  }

  printf("\nRequired Jobs: ");
```

```
for(i = 1; i <= dmax; i++) {

        printf("%s", jobs[timeslot[i]].id);
        if(i < dmax) {
          printf(" --> ");
        }  }
      maxprofit = 0;
      for(i = 1; i <= dmax; i++) {
        maxprofit += jobs[timeslot[i]].profit;
      }
      printf("\nMax Profit: %d\n", maxprofit);
    }
```

Output :



```
C:\Users\ece\Documents\4b.exe
        Job    Deadline      Profit
         j2           1         100
         j1           2          60
         j4           2          40
         j3           3          20
         j5           1          20
dmax: 3

Required Jobs: j2 --> j1 --> j3
Max Profit: 180

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

viva questions :

1. What is job sequencing with deadlines?
2. What is tree vertex splitting
3. Let us consider a set of given jobs as shown in the following table. We have to find a sequence of jobs, which will be completed within their deadlines and will give maximum profit. Each job is associated with a deadline and profit.

Job	**J_1**	**J_2**	**J_3**	**J_4**	**J_5**
Deadline	2	1	3	2	1
Profit	60	100	20	40	20

4. How do you find the vertex cover of a graph?
5. What is feasible solution in greedy method?

0: Not Completed ☐ 5: Late Complete/ In complete ☐ 10: Complete ☐

Viva Voce										

Signature of the Instructor

Week 5 (Back Tracking Techniques)

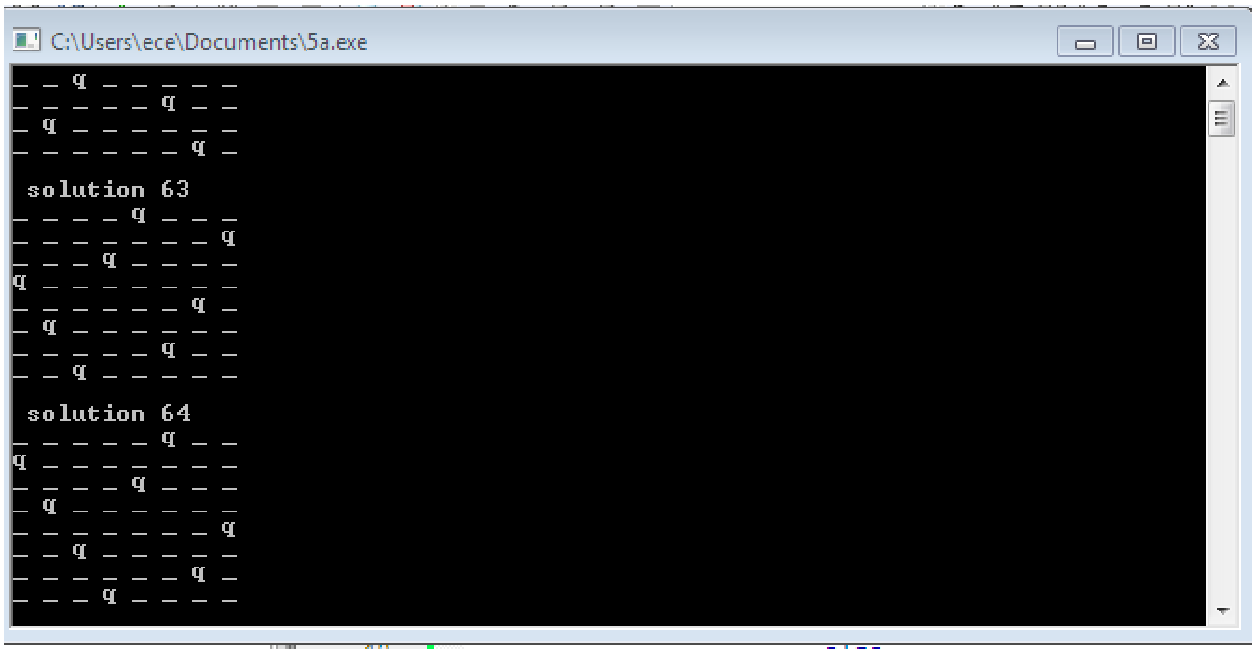
a) **8-Queens Problem**

Source code

```
#include<stdio.h>
 #include<stdlib.h>
 int t[8] = {-1};
 int sol = 1;
 void printsol()
 {
 int i,j;
 char crossboard[8][8];
 for(i=0;i<8;i++)
 {
 for(j=0;j<8;j++)
 {
 crossboard[i][j]='_';
 }
 }
 for(i=0;i<8;i++)
 {
 crossboard[i][t[i]]='q';
 }
 for(i=0;i<8;i++)
 {
 for(j=0;j<8;j++)
 {
 printf("%c ",crossboard[i][j]);
 }
 printf("\n");
 }
 }
 int empty(int i)
 {
 int j=0;
 while((t[i]!=t[j])&&(abs(t[i]-t[j])!=(i-j))&&j<8)j++;
 return i==j?1:0;
 }
 void queens(int i)
 {
 for(t[i] = 0;t[i]<8;t[i]++)
 {
 if(empty(i))
 {
 if(i==7){
```

```
printsol();
printf("\n solution %d\n",sol++);
}
else
queens(i+1);
}
}
}
int main()
{
queens(0);
return 0;
}
```

Output :



```
_ _ q _ _ _ _ _
_ _ _ _ _ q _ _
_ q _ _ _ _ _ _
_ _ _ _ _ _ q _

 solution 63
_ _ _ _ q _ _ _
_ _ _ _ _ _ _ q
_ _ _ q _ _ _ _
q _ _ _ _ _ _ _
_ _ _ _ _ _ q _
_ q _ _ _ _ _ _
_ _ _ _ _ q _ _
_ _ q _ _ _ _ _

 solution 64
_ _ _ _ _ q _ _
q _ _ _ _ _ _ _
_ _ _ _ q _ _ _
_ q _ _ _ _ _ _
_ _ _ _ _ _ _ q
_ _ q _ _ _ _ _
_ _ _ _ _ _ q _
_ _ _ q _ _ _ _
```

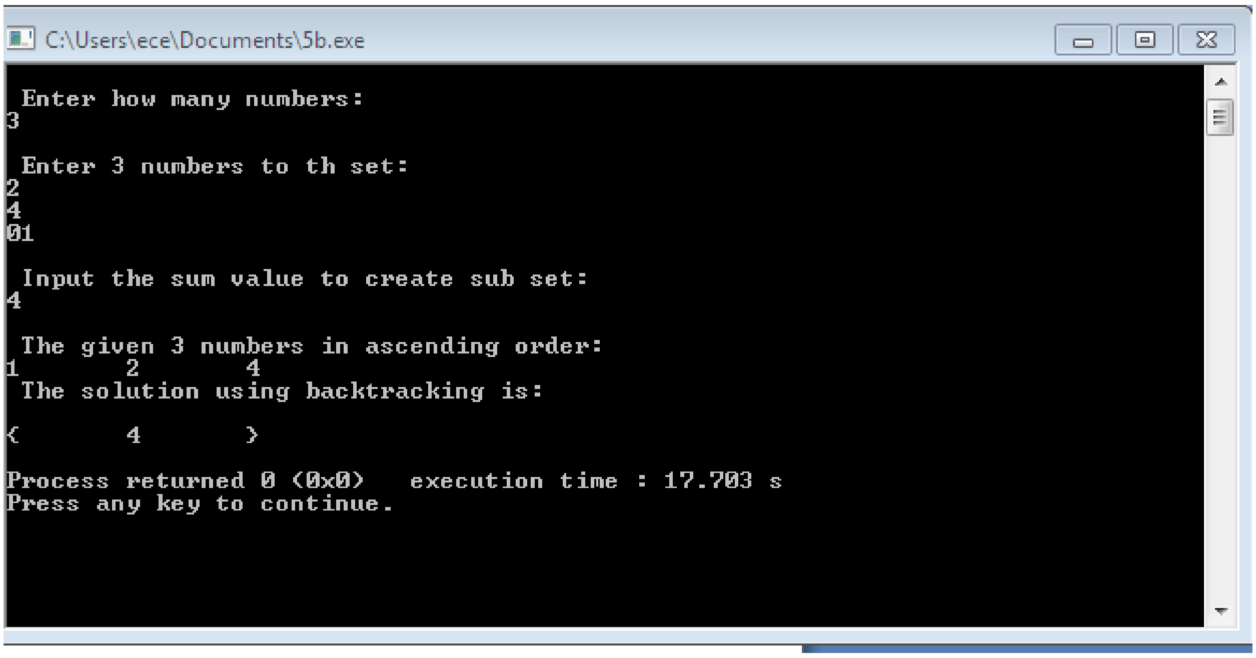
b) <u>Sum of Sub sets</u>

<u>Source code</u>

```
#include<stdio.h>
#define TRUE 1
#define FALSE 0
int inc[50],w[50],sum,n;
int promising(int i,int wt,int total) {
        return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));
}
```

```
void main() {
        int i,j,n,temp,total=0;
//      clrscr();
        printf("\n Enter how many numbers:\n");
        scanf("%d",&n);
        printf("\n Enter %d numbers to th set:\n",n);
        for (i=0;i<n;i++) {
                scanf("%d",&w[i]);
                total+=w[i];
        }
        printf("\n Input the sum value to create sub set:\n");
        scanf("%d",&sum);
        for (i=0;i<=n;i++)
          for (j=0;j<n-1;j++)
           if(w[j]>w[j+1]) {
                temp=w[j];
                w[j]=w[j+1];
                w[j+1]=temp;
        }
        printf("\n The given %d numbers in ascending order:\n",n);
        for (i=0;i<n;i++)
          printf("%d \t",w[i]);
        if((total<sum))
          printf("\n Subset construction is not possible"); else {
                for (i=0;i<n;i++)
                   inc[i]=0;
                printf("\n The solution using backtracking is:\n");
                sumset(-1,0,total);
        }
return (0);
}
void sumset(int i,int wt,int total) {
        int j;
        if(promising(i,wt,total)) {
                if(wt==sum) {
                        printf("\n{\t");
                        for (j=0;j<=i;j++)
                            if(inc[j])
                             printf("%d\t",w[j]);
printf("}\n");
                } else {
                        inc[i+1]=TRUE;
                        sumset(i+1,wt+w[i+1],total-w[i+1]);
                        inc[i+1]=FALSE;
                        sumset(i+1,wt,total-w[i+1]);
                }
        }
}
```

Output :



```
 Enter how many numbers:
3

 Enter 3 numbers to th set:
2
4
01

 Input the sum value to create sub set:
4

 The given 3 numbers in ascending order:
1       2       4
 The solution using backtracking is:

{       4       }

Process returned 0 (0x0)   execution time : 17.703 s
Press any key to continue.
```

viva questions :

1. What is backtracking technique?
2. What happens when the backtracking algorithm reaches a complete solution?
3. What is 8 queen problem
4. Which type of algorithm is used to solve the 8 queens problem?
5. What is a subset sum problem?
6. Give control abstraction for Dynamic Programming.
7. What is the Time complexity of Sum of Subsets problem?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor

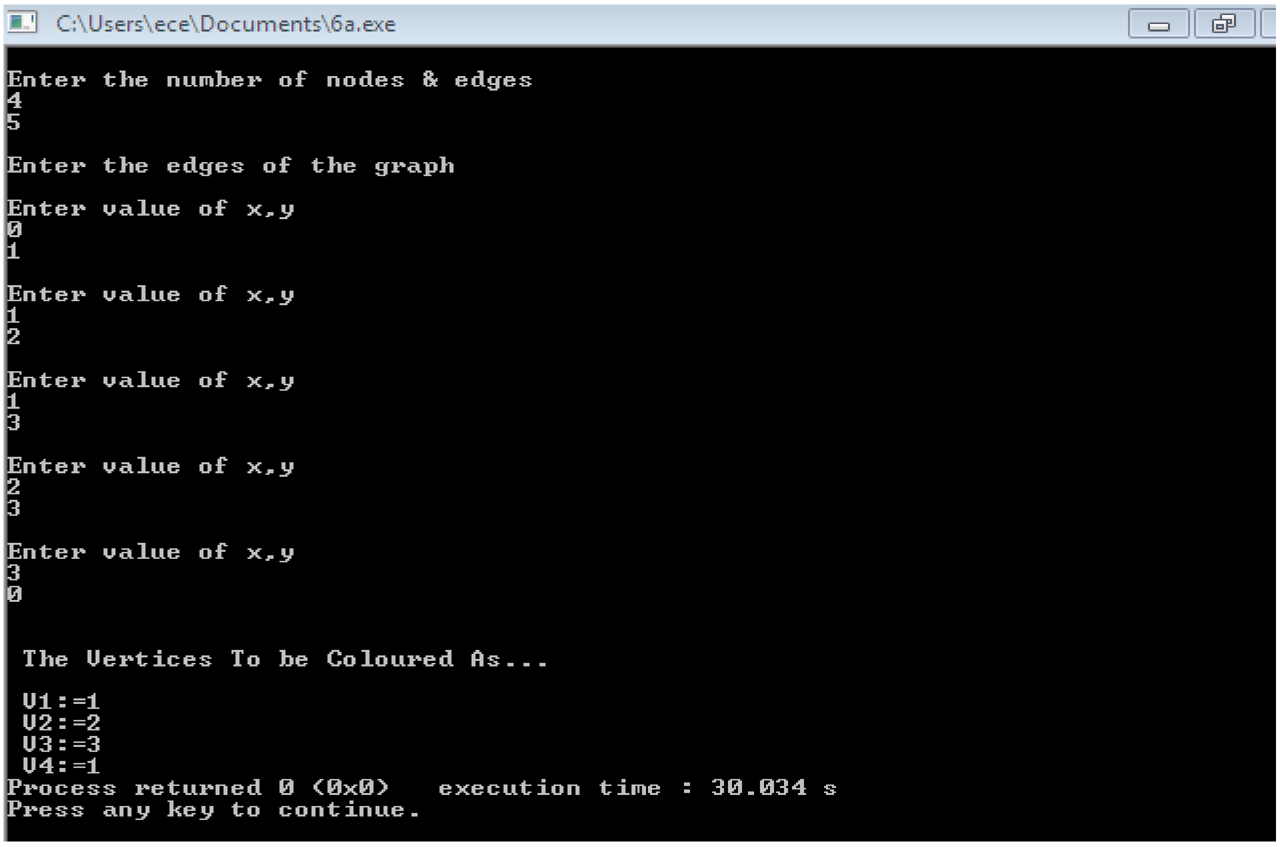
Week 6 (Back Tracking Techniques)

a) **Graph Coloring.**
Source code

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

```
int G[10][10],m,edges,color_tab[10],v1,v2,i,j,n,a,b;
void Gen_Col_Value(int,int);
void Gr_coloring(int,int);
int main()
{
 printf("\nEnter the number of nodes & edges\n");
 scanf("%d%d",&n,&edges);
 m=n-1;
 printf("\nEnter the edges of the graph\n\n");
 for (i=1;i<=edges; i++)
 {
  printf("Enter value of x,y\n");
  scanf("%d%d",&v1,&v2);
   G[v1][v2] = G[v2][v1] = 1;
   printf("\n");
 }
 Gr_coloring(1,n);
 printf("\n The Vertices To be Coloured As...\n");
  for(i=1;i<=n;i++)
    printf("\n V%d:=%d",i,color_tab[i]);
 return 0;
}
void Gen_Col_Value(int k,int n)
{
 while(1)
 {
  a=color_tab[k]+1;
  b=m+1;
  color_tab[k] = a%b;
  if(color_tab[k]==0) return;
  for(j=1;j<=n;j++)
  {
   if(G[k][j] && color_tab[k]==color_tab[j])
     break;
  }
  if(j==n+1) return;
   }
}
void Gr_coloring(int k,int n)
{
  Gen_Col_Value(k,n);
  if(color_tab[k]==0) return;
  if(k==n)  return;
  else Gr_coloring(k+1,n);
}
```

Out put



C:\Users\ece\Documents\6a.exe

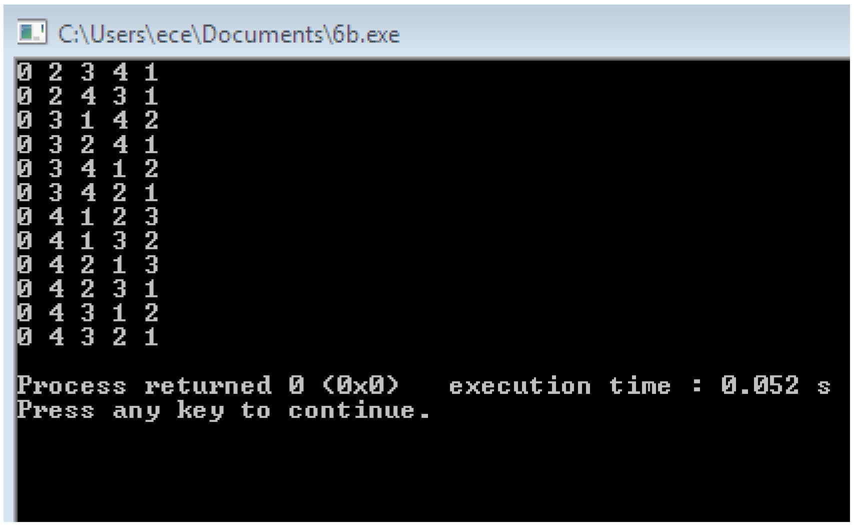
```
Enter the number of nodes & edges
4
5

Enter the edges of the graph

Enter value of x,y
0
1

Enter value of x,y
1
2

Enter value of x,y
1
3

Enter value of x,y
2
3

Enter value of x,y
3
0


 The Vertices To be Coloured As...

 V1:=1
 V2:=2
 V3:=3
 V4:=1
Process returned 0 (0x0)   execution time : 30.034 s
Press any key to continue.
```

b) <u>Hamiltonian Cycles</u>
<u>Source code</u>

```
#include <stdlib.h>

static unsigned int circuit_contains(const unsigned int *circuit, unsigned int c, unsigned int v)

{

   unsigned int i;

   unsigned int contains = 0;

   for (i = 0; i < c && !contains; i++) {

      contains = circuit[i] == v;

   }

   return contains;

}
```

```
typedef void (*circuitfn)(const unsigned int *, size_t);

static void hamiltonian_circuits_recursive(unsigned int **adjmat, size_t n, unsigned int
*circuit,

       unsigned int c, circuitfn fun)

{

    if (c == n) {

        /* Found a circuit */

        fun(circuit, n);

    }    else {

        unsigned int v;

        for (v = 1; v < n; v++) {

            if (!circuit_contains(circuit, c, v) /* Vertex is not in the circuit already */

                && adjmat[circuit[ c - 1]][v] == 1 /* Vertex is adjacent to the previous vertex */

                && (c < n - 1 || (adjmat[0][v] == 1 /* Last vertex is adjacent to the first */

                    && v < circuit[1]))) /* Last vertex is less than the second */

            {

                circuit[ c] = v;

                hamiltonian_circuits_recursive(adjmat, n, circuit, c + 1, fun);

            }        }    } }

void hamiltonian_circuits(unsigned int **adjmat, size_t n, circuitfn fun)

{

    unsigned int *circuit;

    circuit = malloc(n * sizeof(unsigned int));

    if (circuit == NULL) {

        return;

    }

    circuit[0] = 0;
    hamiltonian_circuits_recursive(adjmat, n, circuit, 1, fun);

    free(circuit);

}

static void print_circuit(const unsigned int *circuit, size_t len)

{

    unsigned int v;

    for (v = 0; v < len; v++) {

        printf("%d ", circuit[v]);

    }

    putchar('\n');

}

int main(void)

{

    unsigned int **adjmat;

    const size_t n = 5;

    unsigned int i, j;

    /* Create a complete graph on 5 vertices */

    adjmat = malloc(n * sizeof(unsigned int *));

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

        adjmat[i] = malloc(n * sizeof(unsigned int));

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

            adjmat[i][j] = 1;

        }    }

    hamiltonian_circuits(adjmat, n, print_circuit);

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

        free(adjmat[i]);

    }   free(adjmat);
    return 0;

}
```

OUTPUT:



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

Process returned 0 (0x0)   execution time : 0.052 s
Press any key to continue.
```

Viva Questions

1. What is Hamiltonian Cycle?Can a graph have more than one Hamiltonian Cycle.
2. What do you mean by Coloring a Graph?
3. Define state space Tree
4. Give the rules for colouring a Graph.
5. Which problem is very similar to Hamiltonian Cycle?
6. What is Inclusion Exclusion Principle?
7. What is the time complexity for finding a Hamiltonian path for a graph having N verticesusing permutation)?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

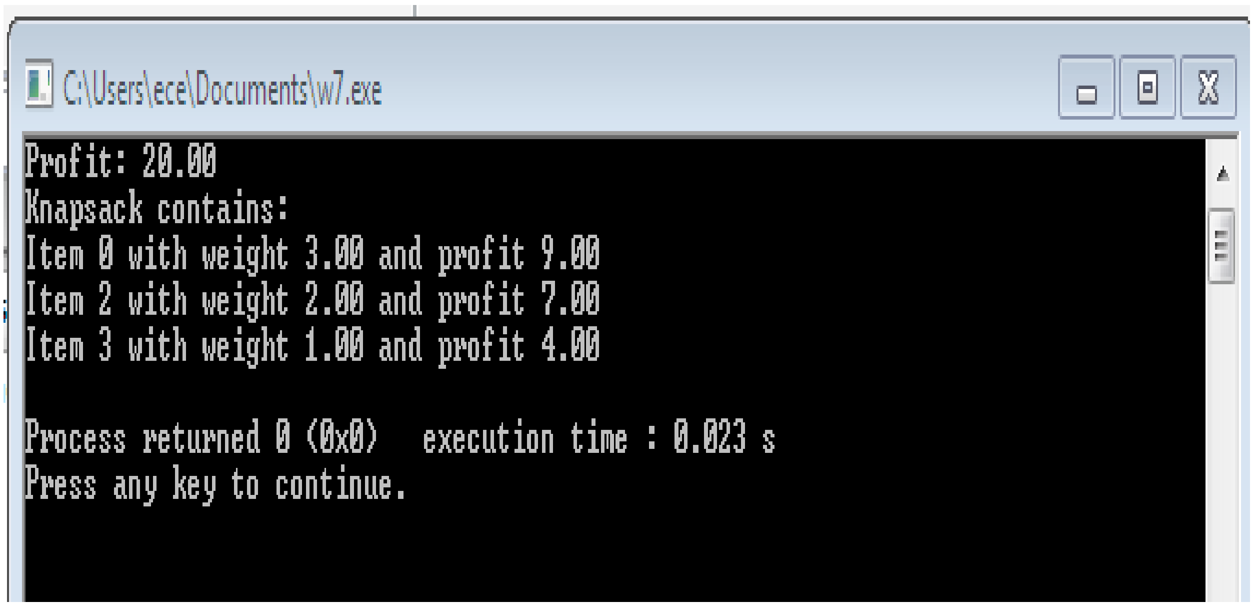
Signature of the Instructor

Week 7 (Back Tracking Techniques)

a) **0/1 Knap Sack Problem**
   Source code

```
#include <stdlib.h>
#include <string.h>

/* A knapsack item */
typedef struct {
   unsigned int id;
   double weight;
   double profit;
   double profit_density;
} item;

/* Compare items by lesser profit density */
static int compare_items(const item *item1, const item *item2)
{
   if (item1->profit_density > item2->profit_density) {
      return -1;
   }
   if (item1->profit_density < item2->profit_density) {
      return 1;
   }
   return 0;
}

/* Bounding function */
static double profit_bound(const item *items, size_t n, double capacity,
      double current_weight, double current_profit,
      unsigned int level)
{
   double remaining_capacity = capacity - current_weight;
   double bound = current_profit;
   unsigned int lvl = level;
   /* Fill in order of decreasing profit density */
   while (lvl < n &&
      items[lvl].weight <= remaining_capacity)
   {
      remaining_capacity -= items[lvl].weight;
      bound += items[lvl].profit;
      lvl++;
   }
   /* Pretend we can take a fraction of the next object */
   if (lvl < n) {
      bound += items[lvl].profit_density
         * remaining_capacity;
   }
   return bound;
}

static void knapsack_recursive(const item *items, size_t n, double capacity,
      unsigned int *current_knapsack, double *current_weight, double *current_profit,
      unsigned int *max_knapsack, double *max_profit, unsigned int level)
{
   if (level == n) {
      /* Found a new max knapsack */
      *max_profit = *current_profit;
      memcpy(max_knapsack, current_knapsack, n * sizeof(unsigned int));
      return;
   }
   if (*current_weight + items[level].weight <= capacity)
   {   /* Try adding this item */
      *current_weight += items[level].weight;
      *current_profit += items[level].profit;
      current_knapsack[items[level].id] = 1;
      knapsack_recursive(items, n, capacity, current_knapsack, current_weight,
              current_profit, max_knapsack, max_profit, level + 1);
      *current_weight -= items[level].weight;
      *current_profit -= items[level].profit;
      current_knapsack[items[level].id] = 0;
   }
   if (profit_bound(items, n, capacity, *current_weight,
              *current_profit, level + 1) > *max_profit) {
      /* Still promising */
      knapsack_recursive(items, n, capacity, current_knapsack, current_weight,
              current_profit, max_knapsack, max_profit, level + 1);
   }}
double knapsack(const double *weights, const double *profits,
      size_t n, double capacity, unsigned int **max_knapsack)
{
   double current_weight = 0.0;
   double current_profit = 0.0;
   double max_profit = 0.0;
   unsigned int i;
   item *items  = malloc(n * sizeof(item));
   unsigned int *current_knapsack = calloc(n, sizeof(unsigned int));
   *max_knapsack = malloc(n * sizeof(unsigned int));
   if (!(items && current_knapsack && *max_knapsack)) {
      free(items);
      free(current_knapsack);
      free(*max_knapsack);
      *max_knapsack = NULL;
      return 0;
   }
   /* Populate the array of items */
   for (i = 0; i < n; i++) {
      items[i].id = i;
      items[i].weight = weights[i];
      items[i].profit = profits[i];
      items[i].profit_density = profits[i] / weights[i];
   }
   /* Sort into decreasing density order */
   qsort(items, n, sizeof(item),
         (int (*)(const void *, const void *))compare_items);
   knapsack_recursive(items, n, capacity, current_knapsack, &current_weight,
         &current_profit, *max_knapsack, &max_profit, 0);
   free(items);
   free(current_knapsack);
   return max_profit;
}

int main(void)
{
   double weights[] = {3, 5, 2, 1};
   double profits[] = {9, 10, 7, 4};
   const size_t n = sizeof(profits) / sizeof(profits[0]);
   const double capacity = 7;
   unsigned int *max_knapsack;
   double max_profit = knapsack(weights, profits, n, capacity, &max_knapsack);
   unsigned int i;
   printf("Profit: %.2f\n", max_profit);
   printf("Knapsack contains:\n");
   for (i = 0; i < n; i++) {
      if (max_knapsack[i] == 1) {
         printf("Item %u with weight %.2f and profit %.2f\n", i, weights[i], profits[i]);
      }    }
   free(max_knapsack);
   return 0;
}
```

Out put



```
Profit: 20.00
Knapsack contains:
Item 0 with weight 3.00 and profit 9.00
Item 2 with weight 2.00 and profit 7.00
Item 3 with weight 1.00 and profit 4.00

Process returned 0 (0x0)   execution time : 0.023 s
Press any key to continue.
```

Viva Questions

1. What is Backtracking.How it is usefull to give solution for a problem?
2. Explain 0/1 Knapsack problem with an example?
3. Give control abstraction for BackTracking?
4. List out some of the applications on BackTracking Technique?
5. What happens when the backtracking algorithm reaches a complete solution?
6. In general, backtracking can be used to solve?
7. The problem of finding a list of integers in a given specific range that meets certain conditions is called?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor

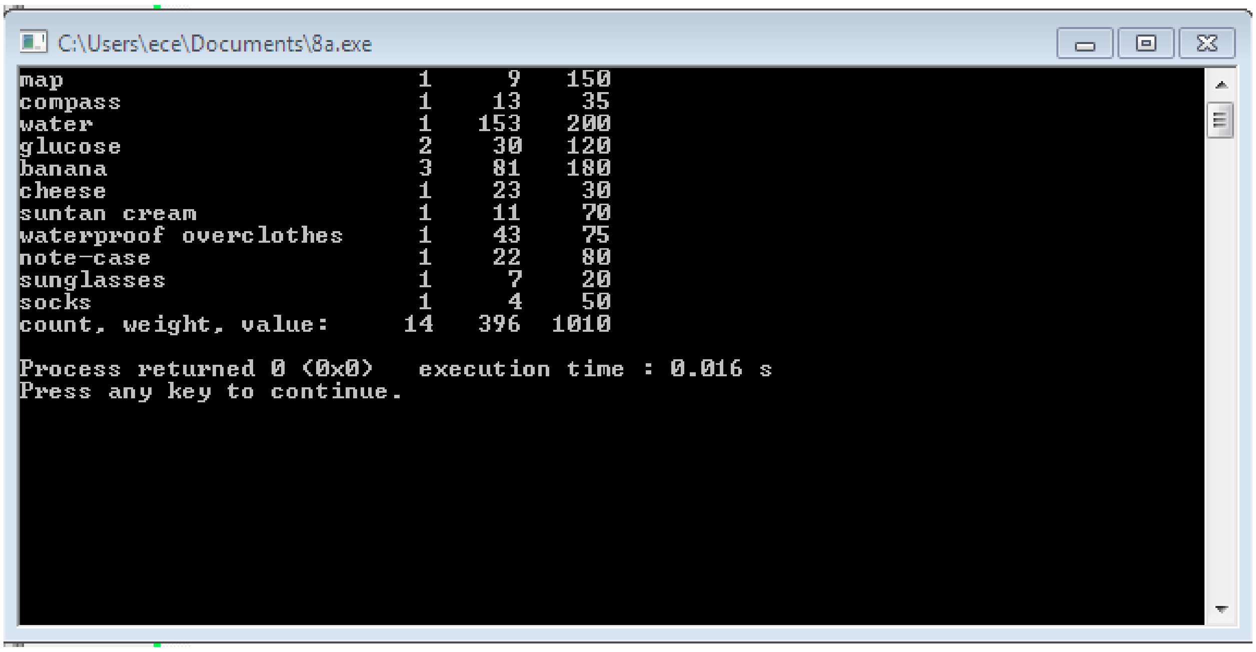
Week 8 (Branch and Bound)

a) **0/1 Knap Sack Problem**
Source code

```
#include <stdio.h>

#include <stdlib.h>


typedef struct {

    char *name;

    int weight;

    int value;

    int count;

} item_t;


item_t items[] = {

    {"map",                    9,   150,   1},

    {"compass",               13,    35,   1},

    {"water",                153,   200,   2},

    {"sandwich",              50,    60,   2},

    {"glucose",               15,    60,   2},

    {"tin",                   68,    45,   3},

    {"banana",                27,    60,   3},

    {"apple",                 39,    40,   3},

    {"cheese",                23,    30,   1},

    {"beer",                  52,    10,   3},

    {"suntan cream",          11,    70,   1},

    {"camera",                32,    30,   1},

    {"T-shirt",               24,    15,   2},

    {"trousers",              48,    10,   2},
    {"umbrella",                73,    40,   1},
    {"waterproof trousers",     42,    70,   1},
    {"waterproof overclothes",  43,    75,   1},
    {"note-case",               22,    80,   1},
    {"sunglasses",               7,    20,   1},
    {"towel",                   18,    12,   2},
    {"socks",                    4,    50,   1},
    {"book",                    30,    10,   2},
};

int n = sizeof (items) / sizeof (item_t);

int *knapsack (int w) {
    int i, j, k, v, *mm, **m, *s;
    mm = calloc((n + 1) * (w + 1), sizeof (int));
    m = malloc((n + 1) * sizeof (int *));
    m[0] = mm;
    for (i = 1; i <= n; i++) {
        m[i] = &mm[i * (w + 1)];
        for (j = 0; j <= w; j++) {
            m[i][j] = m[i - 1][j];
            for (k = 1; k <= items[i - 1].count; k++) {
                if (k * items[i - 1].weight > j) {
                    break;
                }
                v = m[i - 1][j - k * items[i - 1].weight] + k * items[i - 1].value;
                if (v > m[i][j]) {
```

```
                m[i][j] = v;
            }
        }
    }
}
s = calloc(n, sizeof (int));
for (i = n, j = w; i > 0; i--) {
    int v = m[i][j];
    for (k = 0; v != m[i - 1][j] + k * items[i - 1].value; k++) {
        s[i - 1]++;
        j -= items[i - 1].weight;
    }
}
free(mm);
free(m);
return s;
}


int main () {
    int i, tc = 0, tw = 0, tv = 0, *s;
    s = knapsack(400);
    for (i = 0; i < n; i++) {
        if (s[i]) {
            printf("%-22s %5d %5d %5d\n", items[i].name, s[i], s[i] * items[i].weight, s[i] *
items[i].value);
            tc += s[i];
            tw += s[i] * items[i].weight;
            tv += s[i] * items[i].value;
        }
    }
    printf("%-22s %5d %5d %5d\n", "count, weight, value:", tc, tw, tv);
    return 0;
}
```

Output :



C:\Users\ece\Documents\8a.exe

```
map                            1     9   150
compass                        1    13    35
water                          1   153   200
glucose                        2    30   120
banana                         3    81   180
cheese                         1    23    30
suntan cream                   1    11    70
waterproof overclothes         1    43    75
note-case                      1    22    80
sunglasses                     1     7    20
socks                          1     4    50
count, weight, value:         14   396  1010

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

b) **Traveling Sales Person Problem**
Source code :

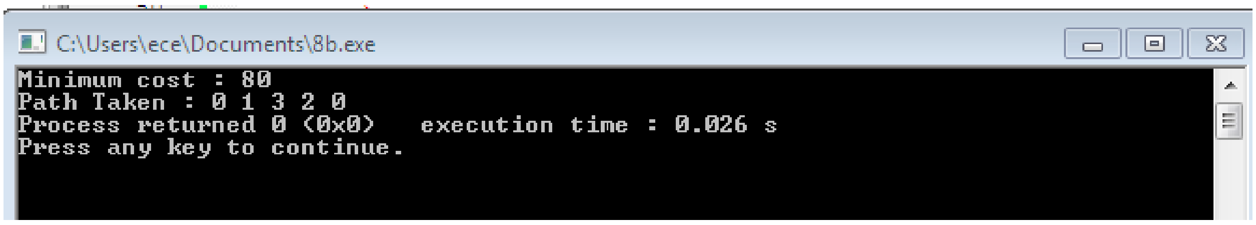
```
#include <bits/stdc++.h>
using namespace std;
const int N = 4;

// final_path[] stores the final solution ie, the
// path of the salesman.
int final_path[N+1];

// visited[] keeps track of the already visited nodes
// in a particular path
bool visited[N];

// Stores the final minimum weight of shortest tour.
int final_res = INT_MAX;

// Function to copy temporary solution to
// the final solution
void copyToFinal(int curr_path[])
{
   for (int i=0; i<N; i++)
      final_path[i] = curr_path[i];
   final_path[N] = curr_path[0];
}

// Function to find the minimum edge cost
// having an end at the vertex i
int firstMin(int adj[N][N], int i)
{
   int min = INT_MAX;
   for (int k=0; k<N; k++)
      if (adj[i][k]<min && i != k)
         min = adj[i][k];
   return min;
}

// function to find the second minimum edge cost
// having an end at the vertex i
int secondMin(int adj[N][N], int i)
{
   int first = INT_MAX, second = INT_MAX;
   for (int j=0; j<N; j++)
   {
      if (i == j)
         continue;

      if (adj[i][j] <= first)
      {
         second = first;
         first = adj[i][j];
      }
      else if (adj[i][j] <= second &&
               adj[i][j] != first)
         second = adj[i][j];
   }
   return second;
}

// function that takes as arguments:
// curr_bound -> lower bound of the root node
// curr_weight-> stores the weight of the path so far
// level-> current level while moving in the search
//        space tree
// curr_path[] -> where the solution is being stored which
//               would later be copied to final_path[]
void TSPRec(int adj[N][N], int curr_bound, int curr_weight,
         int level, int curr_path[])
{
   // base case is when we have reached level N which
   // means we have covered all the nodes once
   if (level==N)
   {
      // check if there is an edge from last vertex in
      // path back to the first vertex
      if (adj[curr_path[level-1]][curr_path[0]] != 0)
```

```
    {
        // curr_res has the total weight of the
        // solution we got
        int curr_res = curr_weight +
                adj[curr_path[level-1]][curr_path[0]];

        // Update final result and final path if
        // current result is better.
        if (curr_res < final_res)
        {
            copyToFinal(curr_path);
            final_res = curr_res;
        }
    }
    return;
}

// for any other level iterate for all vertices to
// build the search space tree recursively
for (int i=0; i<N; i++)
{
    // Consider next vertex if it is not same (diagonal
    // entry in adjacency matrix and not visited
    // already)
    if (adj[curr_path[level-1]][i] != 0 &&
        visited[i] == false)
    {
        int temp = curr_bound;
        curr_weight += adj[curr_path[level-1]][i];

        // different computation of curr_bound for
        // level 2 from the other levels
        if (level==1)
          curr_bound -= ((firstMin(adj, curr_path[level-1]) +
                         firstMin(adj, i))/2);
        else
          curr_bound -= ((secondMin(adj, curr_path[level-1]) +
                         firstMin(adj, i))/2);

        // curr_bound + curr_weight is the actual lower bound
        // for the node that we have arrived on
        // If current lower bound < final_res, we need to explore
        // the node further
        if (curr_bound + curr_weight < final_res)
        {
            curr_path[level] = i;
            visited[i] = true;

            // call TSPRec for the next level
            TSPRec(adj, curr_bound, curr_weight, level+1,
                   curr_path);
        }

        // Else we have to prune the node by resetting
        // all changes to curr_weight and curr_bound
        curr_weight -= adj[curr_path[level-1]][i];
        curr_bound = temp;
```

```
            // Also reset the visited array
            memset(visited, false, sizeof(visited));
            for (int j=0; j<=level-1; j++)
                visited[curr_path[j]] = true;
        }
    }
}

// This function sets up final_path[]
void TSP(int adj[N][N])
{
    int curr_path[N+1];

    // Calculate initial lower bound for the root node
    // using the formula 1/2 * (sum of first min +
    // second min) for all edges.
    // Also initialize the curr_path and visited array
    int curr_bound = 0;
    memset(curr_path, -1, sizeof(curr_path));
    memset(visited, 0, sizeof(curr_path));

    // Compute initial bound
    for (int i=0; i<N; i++)
        curr_bound += (firstMin(adj, i) +
                       secondMin(adj, i));

    // Rounding off the lower bound to an integer
    curr_bound = (curr_bound&1)? curr_bound/2 + 1 :
                                 curr_bound/2;

    // We start at vertex 1 so the first vertex
    // in curr_path[] is 0
    visited[0] = true;
    curr_path[0] = 0;

    // Call to TSPRec for curr_weight equal to
    // 0 and level 1
    TSPRec(adj, curr_bound, 0, 1, curr_path);
}

// Driver code
int main()
{
    //Adjacency matrix for the given graph
    int adj[N][N] = { {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    TSP(adj);

    printf("Minimum cost : %d\n", final_res);
    printf("Path Taken : ");
    for (int i=0; i<=N; i++)
        printf("%d ", final_path[i]);

    return 0;
```

}

Output :



C:\Users\ece\Documents\8b.exe

```
Minimum cost : 80
Path Taken : 0 1 3 2 0
Process returned 0 (0x0)   execution time : 0.026 s
Press any key to continue.
```

Viva Questions

1. What is Knapsack Problem,on what strategies it can be applied?

2. Differences between Knapsack and 0/1 Knapsack problem?
3. Give control abstraction for Branch and Bound Strategy.
4. What are the different kinds of strategies involved under Branch and Bound Technique?
5. Define LIFO?
6. What is state space tree?
7. Explain Branch and Bound Technique?
8. Explain TSP Problem?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor

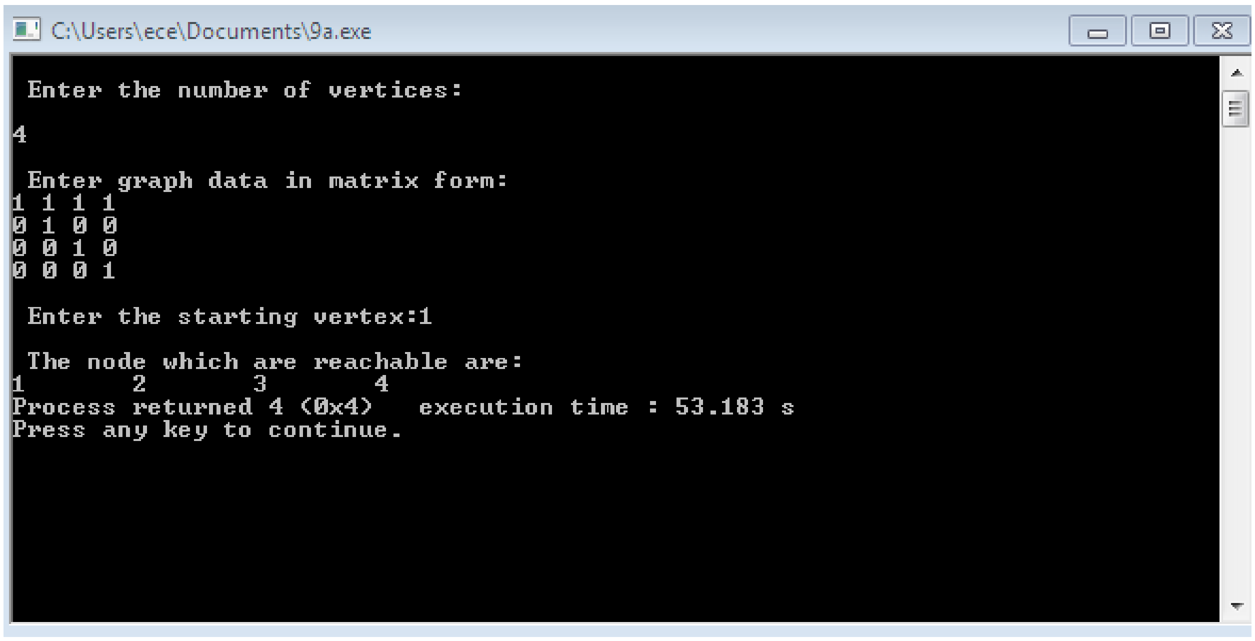
Week 9 (Graph Algorithms)

a) <u>Breadth First Search</u>

<u>Source code :</u>

```
#include<stdio.h>

int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v) {

 for(i = 1; i <= n; i++)

 if(a[v][i] && !visited[i])

 q[++r] = i;

 if(f <= r) {

 visited[q[f]] = 1;

 bfs(q[f++]);

 }

}

void main() {

 int v;

 printf("\n Enter the number of vertices:");

 scanf("%d", &n);

 for(i=1; i <= n; i++) {

 q[i] = 0;

 visited[i] = 0;

 }

 printf("\n Enter graph data in matrix form:\n");

 for(i=1; i<=n; i++) {

 for(j=1;j<=n;j++) {

 scanf("%d", &a[i][j]);

 }
}

printf("\n Enter the starting vertex:");
scanf("%d", &v);
bfs(v);
printf("\n The node which are reachable are:\n");
for(i=1; i <= n; i++) {
if(visited[i])
printf("%d\t", i);
else {
printf("\n Bfs is not possible. Not all nodes are reachable");
break;
}
}
}
```

Output :



C:\Users\ece\Documents\9a.exe

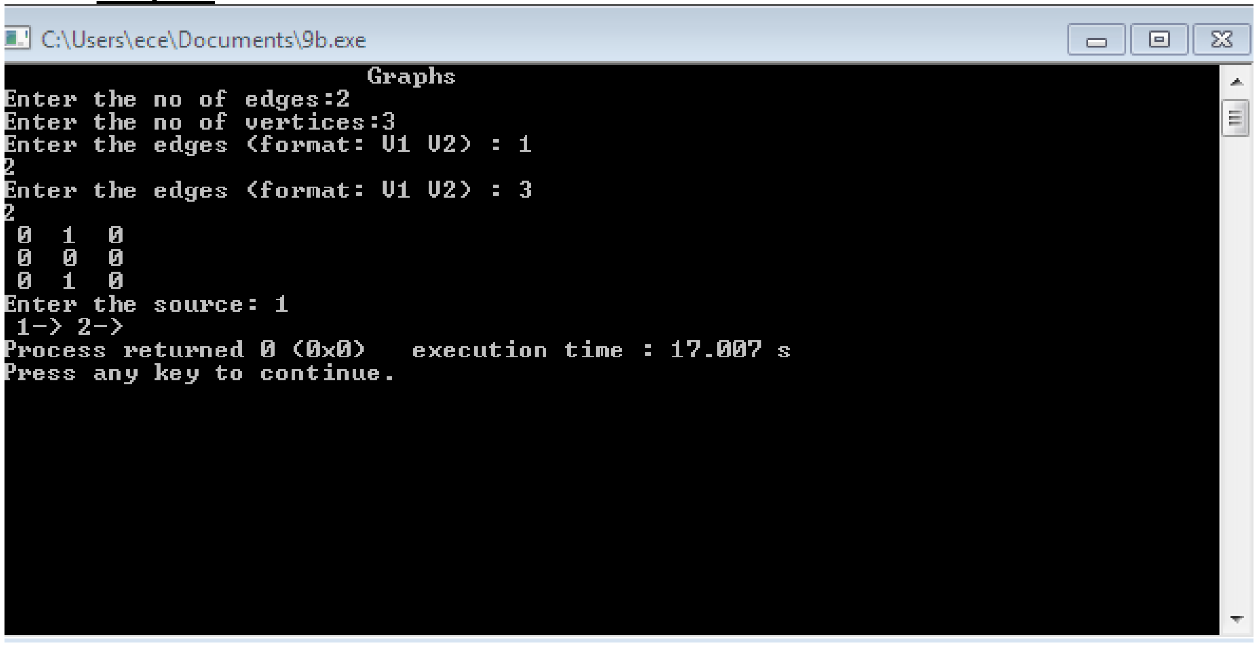
```
 Enter the number of vertices:

4

 Enter graph data in matrix form:
1 1 1 1
0 1 0 0
0 0 1 0
0 0 0 1

 Enter the starting vertex:1

 The node which are reachable are:
1       2       3       4
Process returned 4 (0x4)   execution time : 53.183 s
Press any key to continue.
```

b) <u>Depth First Search</u>

<u>Source code :</u>

```
#include <stdio.h>
#include <stdlib.h>
/*          ADJACENCY MATRIX                            */
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
   int j;
   visited[i]=1;
   printf(" %d->",i+1);
   for(j=0;j<V;j++)
   {
      if(G[i][j]==1&&visited[j]==0)
         DFS(j);
   }
}
int main()
{
   int i,j,v1,v2;
   printf("\t\t\tGraphs\n");
   printf("Enter the no of edges:");
   scanf("%d",&E);
   printf("Enter the no of vertices:");
   scanf("%d",&V);
   for(i=0;i<V;i++)
   {
      for(j=0;j<V;j++)
         G[i][j]=0;
   }
   /*    creating edges :P    */
   for(i=0;i<E;i++)
   {
      printf("Enter the edges (format: V1 V2) : ");
      scanf("%d%d",&v1,&v2);
      G[v1-1][v2-1]=1;

   }

   for(i=0;i<V;i++)
   {
      for(j=0;j<V;j++)
         printf(" %d ",G[i][j]);
      printf("\n");
   }
   printf("Enter the source: ");
   scanf("%d",&source);
      DFS(source-1);
   return 0;
}
```

Output :



```
                                        Graphs
Enter the no of edges:2
Enter the no of vertices:3
Enter the edges (format: V1 V2) : 1
2
Enter the edges (format: V1 V2) : 3
2
 0  1  0
 0  0  0
 0  1  0
Enter the source: 1
 1-> 2->
Process returned 0 (0x0)   execution time : 17.007 s
Press any key to continue.
```

Viva Questions

1. Define Graph with an example?
2. List out the Graph Traversal Algorithms.
3. What are the Datastructures does BFS and DFS use?
4. Give applications of BFS.
5. Define transitive closure of graph using DFA?
6. Give applications of DFS.
7. Breadth First Search is equivalent to which of the traversal in the Binary Trees?
8. A person wants to visit some places. He starts from a vertex and then wants to visit every place connected to this vertex and so on. What algorithm he should use?
9. What is the timecomplexity of BFS and DFS?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor

Week 10 Graph Algorithms

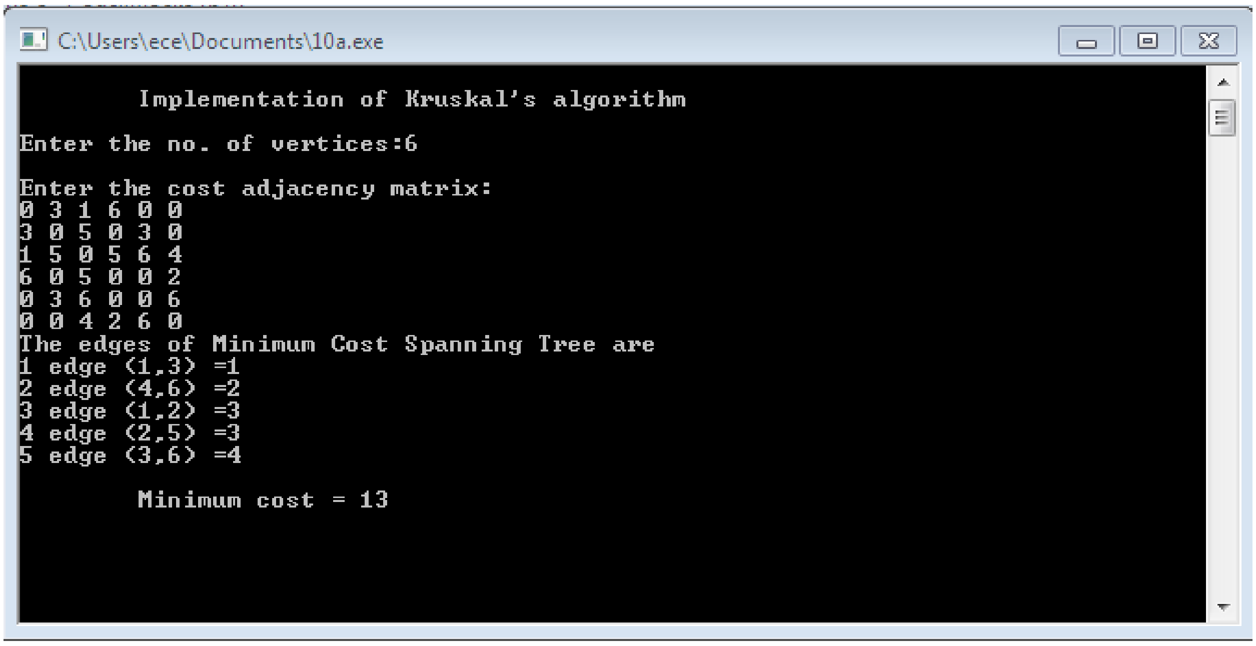
a) <u>Kruskal`s Algorithm</u>

<u>Source code</u>

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void main()

{       printf("\n\tImplementation of Kruskal's algorithm\n");

        printf("\nEnter the no. of vertices:");

        scanf("%d",&n);

        printf("\nEnter the cost adjacency matrix:\n");

        for(i=1;i<=n;i++)

        {

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

}printf("The edges of Minimum Cost Spanning Tree are\n");

        while(ne < n)

        {

                for(i=1,min=999;i<=n;i++)

                {
                        for(j=1;j <= n;j++)
                        {
                                if(cost[i][j] < min)
                                {
                                        min=cost[i][j];
                                        a=u=i;
                                        b=v=j;
                                }
                        }
                }
                u=find(u);
                v=find(v);
                if(uni(u,v))
                {
                        printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                        mincost +=min;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
        getch();
}
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
```

```
}
int uni(int i,int j)
{
	if(i!=j)
	{
		parent[j]=i;
		return 1;
	}
	return 0;
}
```

Output



C:\Users\ece\Documents\10a.exe

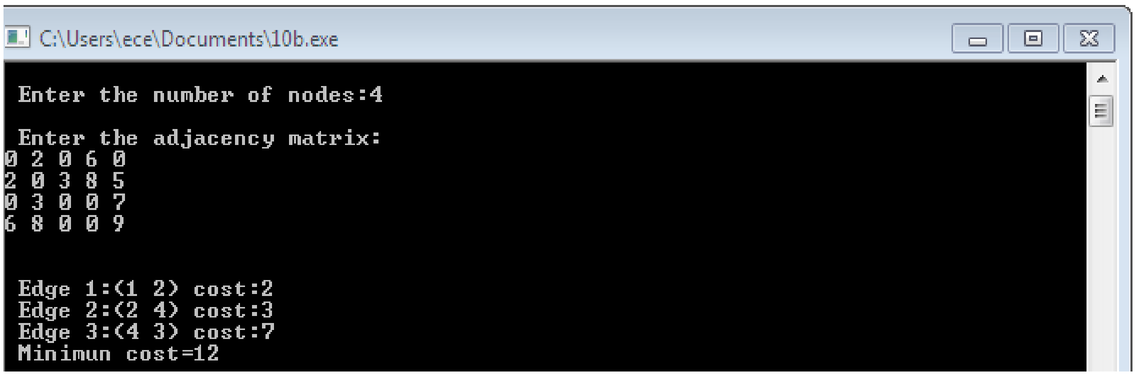
```
        Implementation of Kruskal's algorithm
Enter the no. of vertices:6

Enter the cost adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (4,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (3,6) =4

        Minimum cost = 13
```

b) Prims algorithm

Source code

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {   0}
,min,mincost=0,cost[10][10];
void main() {
    printf("\n Enter the number of nodes:");
    scanf("%d",&n);
    printf("\n Enter the adjacency matrix:\n");
    for (i=1;i<=n;i++)
      for (j=1;j<=n;j++) {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
               cost[i][j]=999;
    }
    visited[1]=1;
    printf("\n");
    while(ne<n) {
            for (i=1,min=999;i<=n;i++)
              for (j=1;j<=n;j++)
               if(cost[i][j]<min)
                if(visited[i]!=0) {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
            }
            if(visited[u]==0 || visited[v]==0) {
                    printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                    mincost+=min;
                    visited[b]=1;
            }
            cost[a][b]=cost[b][a]=999;
    }
    printf("\n Minimun cost=%d",mincost);
    getch();}
```

output :



C:\Users\ece\Documents\10b.exe

```
 Enter the number of nodes:4

 Enter the adjacency matrix:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9


 Edge 1:(1 2) cost:2
 Edge 2:(2 4) cost:3
 Edge 3:(4 3) cost:7
 Minimun cost=12
```

Viva Questions

1. What strategy does Prim’s and Kruskals Follow?
2. List out some of the Greedy Algorithms?
3. Prim’s Algorithm can also be called as?
4. What is the Time complexity of Prim’s and Kruskals Algorithm?
5. Give the best,Average,Worst case Time complexities of kruskal’s Algorithm?
6. State the differences between Prims and Kruskals Algorithms?
7. Give the best,Average,Worst case Time complexities of Prims Algorithm?
8. Does Kruskal’s algorithm is best suited for the sparse graphs than the prim’s algorithm.

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor

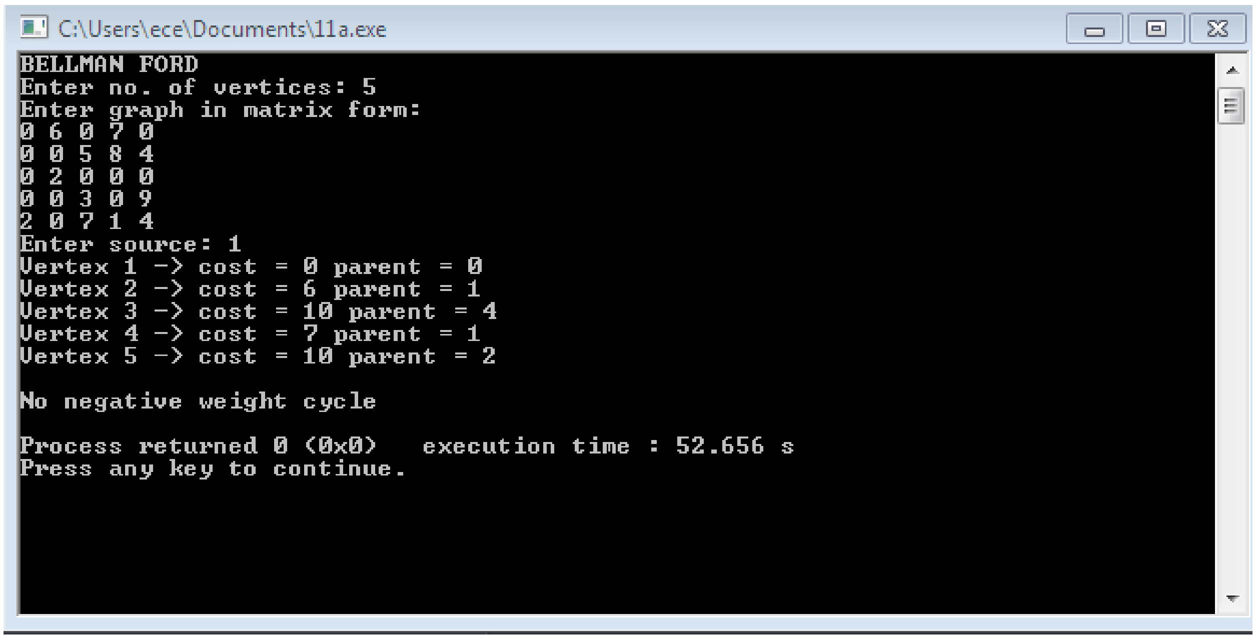
Week 11 (Graph Algorithms)

a) **Bellman Ford Algorithm**

Source code

```
#include<stdio.h>
#include<stdlib.h>
int Bellman_Ford(int G[20][20] , int V, int E, int edge[20][2])
{
   int i,u,v,k,distance[20],parent[20],S,flag=1;
   for(i=0;i<V;i++)
      distance[i] = 1000 , parent[i] = -1 ;
      printf("Enter source: ");
      scanf("%d",&S);
      distance[S-1]=0 ;
   for(i=0;i<V-1;i++)
   {
      for(k=0;k<E;k++)
      {
         u = edge[k][0] , v = edge[k][1] ;
         if(distance[u]+G[u][v] < distance[v])
            distance[v] = distance[u] + G[u][v] , parent[v]=u ;
      }
   }
   for(k=0;k<E;k++)
      {
         u = edge[k][0] , v = edge[k][1] ;
         if(distance[u]+G[u][v] < distance[v])
            flag = 0 ;
      }
      if(flag)
         for(i=0;i<V;i++)
            printf("Vertex %d -> cost = %d parent =
%d\n",i+1,distance[i],parent[i]+1);

      return flag;
}
int main()
{
   int V,edge[20][2],G[20][20],i,j,k=0;
   printf("BELLMAN FORD\n");
   printf("Enter no. of vertices: ");
   scanf("%d",&V);
   printf("Enter graph in matrix form:\n");
   for(i=0;i<V;i++)
      for(j=0;j<V;j++)
      {
         scanf("%d",&G[i][j]);
         if(G[i][j]!=0)
```

```
                edge[k][0]=i,edge[k++][1]=j;
            }

        if(Bellman_Ford(G,V,k,edge))
            printf("\nNo negative weight cycle\n");
        else printf("\nNegative weight cycle exists\n");
        return 0;
    }
```



C:\Users\ece\Documents\11a.exe

```
BELLMAN FORD
Enter no. of vertices: 5
Enter graph in matrix form:
0 6 0 7 0
0 0 5 8 4
0 2 0 0 0
0 0 3 0 9
2 0 7 1 4
Enter source: 1
Vertex 1 -> cost = 0 parent = 0
Vertex 2 -> cost = 6 parent = 1
Vertex 3 -> cost = 10 parent = 4
Vertex 4 -> cost = 7 parent = 1
Vertex 5 -> cost = 10 parent = 2

No negative weight cycle

Process returned 0 (0x0)   execution time : 52.656 s
Press any key to continue.
```

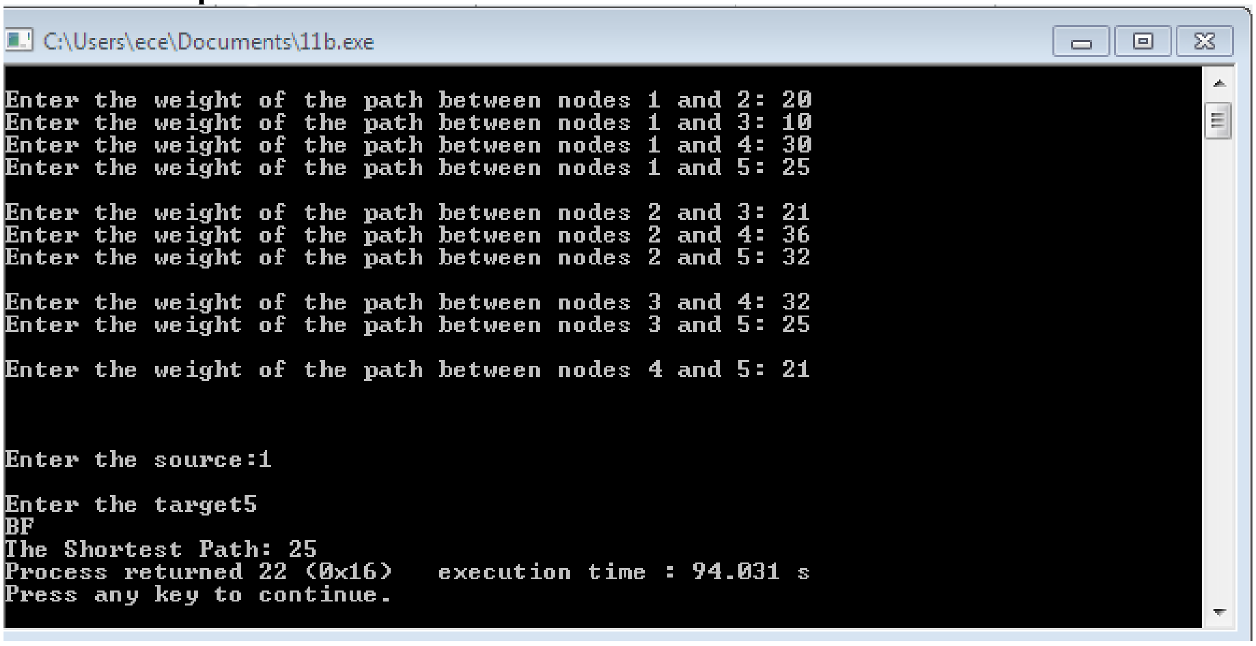
b) <u>Dijkstras algorithm</u>

<u>Source code</u>

```
#include<stdio.h>

#include<conio.h>

#include<process.h>

#include<string.h>

#include<math.h>

#define IN 99

#define N 6

int dijkstra(int cost[][N], int source, int target);

int main()

{

   int cost[N][N],i,j,w,ch,co;
   int source, target,x,y;

   printf("\t The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C \n\n");

   for(i=1;i< N;i++)

   for(j=1;j< N;j++)

   cost[i][j] = IN;

   for(x=1;x< N;x++)

   {

      for(y=x+1;y< N;y++)

      {

         printf("Enter the weight of the path between nodes %d and %d: ",x,y);

         scanf("%d",&w);
         cost [x][y] = cost[y][x] = w;
      }
      printf("\n");
   }
   printf("\nEnter the source:");
   scanf("%d", &source);
   printf("\nEnter the target");
   scanf("%d", &target);
   co = dijsktra(cost,source,target);
   printf("\nThe Shortest Path: %d",co);
}
int dijsktra(int cost[][N],int source,int target)
{
   int dist[N],prev[N],selected[N]={0},i,m,min,start,d,j;
   char path[N];
   for(i=1;i< N;i++)
   {
      dist[i] = IN;
      prev[i] = -1;
   }
   start = source;
   selected[start]=1;
   dist[start] = 0;
   while(selected[target] ==0)
   {
      min = IN;
      m = 0;
      for(i=1;i< N;i++)
      {
         d = dist[start] +cost[start][i];
```

```
            if(d< dist[i]&&selected[i]==0)
            {
               dist[i] = d;
               prev[i] = start;
            }
            if(min>dist[i] && selected[i]==0)
            {
               min = dist[i];
               m = i;
            }
         }
         start = m;
         selected[start] = 1;
      }
      start = target;
      j = 0;
      while(start != -1)
      {
         path[j++] = start+65;
         start = prev[start];
      }
      path[j]='\0';
      strrev(path);
      printf("%s", path);
      return dist[target];
}
```

Output :



```
Enter the weight of the path between nodes 1 and 2: 20
Enter the weight of the path between nodes 1 and 3: 10
Enter the weight of the path between nodes 1 and 4: 30
Enter the weight of the path between nodes 1 and 5: 25

Enter the weight of the path between nodes 2 and 3: 21
Enter the weight of the path between nodes 2 and 4: 36
Enter the weight of the path between nodes 2 and 5: 32

Enter the weight of the path between nodes 3 and 4: 32
Enter the weight of the path between nodes 3 and 5: 25

Enter the weight of the path between nodes 4 and 5: 21



Enter the source:1

Enter the target5
BF
The Shortest Path: 25
Process returned 22 (0x16)   execution time : 94.031 s
Press any key to continue.
```

Viva Questions

1. Define Shortest path?
2. List out some of the algorithms for calculating Shortest paths
3. Which kind of problems can be solved using Dijkstra’s Algorithm?
4. Which is the most commonly used data structure for implementing Dijkstra’s Algorithm?
5. Define priority Queue. How many priority queue operations are involved in Dijkstra’s Algorithm?
6. What is running time of Dijkstra’s algorithm using Binary min- heap method?
7. What is negative cycle?
8. When a graph is said to have a negative weight cycle?
9. What is the first step in the naïve greedy algorithm?
10. Under what condition can a vertex combine and distribute flow in any manner?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor

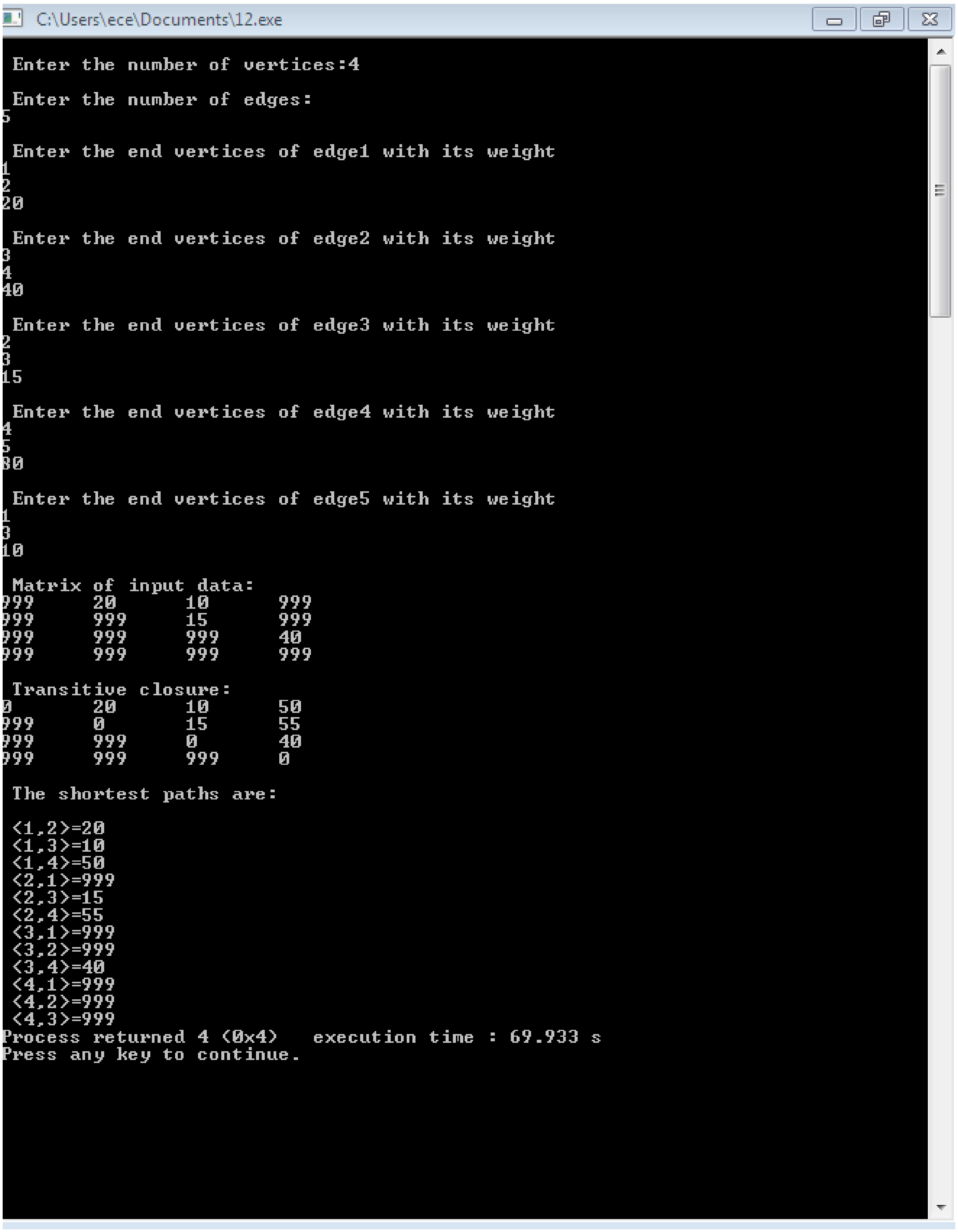
Week 12 (Graph Algorithms)

b) Floyd- Warshall Algorithm.

Source code

```
#include<stdio.h>

int min(int,int);

void floyds(int p[10][10],int n)

{

 int i,j,k;

 for(k=1;k<=n;k++)

  for(i=1;i<=n;i++)

   for(j=1;j<=n;j++)

    if(i==j)

     p[i][j]=0;

    else

     p[i][j]=min(p[i][j],p[i][k]+p[k][j]);

}

int min(int a,int b)

{

 if(a<b)

  return(a);

 else

  return(b);

}

void main()

{

 int p[10][10],w,n,e,u,v,i,j;;

 printf("\n Enter the number of vertices:");
```

```
scanf("%d",&n);
printf("\n Enter the number of edges:\n");
scanf("%d",&e);
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
  p[i][j]=999;
}
for(i=1;i<=e;i++)
{
 printf("\n Enter the end vertices of edge%d with its weight \n",i);
 scanf("%d%d%d",&u,&v,&w);
 p[u][v]=w;
}
printf("\n Matrix of input data:\n");
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
  printf("%d \t",p[i][j]);
 printf("\n");
}
floyds(p,n);
printf("\n Transitive closure:\n");
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
  printf("%d \t",p[i][j]);
  printf("\n");
 }
 printf("\n The shortest paths are:\n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {
   if(i!=j)
    printf("\n <%d,%d>=%d",i,j,p[i][j]);
  }
}
```



```
 Enter the number of vertices:4

 Enter the number of edges:
5

 Enter the end vertices of edge1 with its weight
1
2
20

 Enter the end vertices of edge2 with its weight
3
4
40

 Enter the end vertices of edge3 with its weight
2
3
15

 Enter the end vertices of edge4 with its weight
4
5
30

 Enter the end vertices of edge5 with its weight
1
3
10

 Matrix of input data:
999     20      10      999
999     999     15      999
999     999     999     40
999     999     999     999

 Transitive closure:
0       20      10      50
999     0       15      55
999     999     0       40
999     999     999     0

 The shortest paths are:

 <1,2>=20
 <1,3>=10
 <1,4>=50
 <2,1>=999
 <2,3>=15
 <2,4>=55
 <3,1>=999
 <3,2>=999
 <3,4>=40
 <4,1>=999
 <4,2>=999
 <4,3>=999
Process returned 4 (0x4)   execution time : 69.933 s
Press any key to continue.
```

Viva Questions

1. What is the running time of the Floyd Warshall Algorithm?
2. What approach is being followed in Floyd Warshall Algorithm?
3. What procedure is being followed in Floyd Warshall Algorithm?
4. Floyd- Warshall algorithm was proposed by?
5. What are the differences between Trees and Graphs?
6. State the differences between Floyd’s and Warshall Algorithm?
7. What is the time complexity of Floyd–Warshall algorithm to calculate all pair shortest path in a graph with **n** vertices?

8. What are the differences between Dijkstra’s Algorithm and Floyd Warshall Algorithm?

0: Not Completed		5: Late Complete/ In complete		10: Complete	

Viva Voce										

Signature of the Instructor