K. RAMAKRISHNAN COLLEGE OF ENGINEERING
(Autonomous)

SAMAYAPURAM, TRICHY – 621112.

DEPARTMENT OF COMPUTER SCIENCE AND BUSINESS SYSTEMS

PRACTICAL RECORD NOTE

Name: ……………………………………………………….........................

Register No: ………………………………………………..........................

Subject: ……………………………………………………..........................

Course: ……………………………………………………...........................

K. RAMAKRISHNAN COLLEGE OF ENGINEERING
(Autonomous)
SAMAYAPURAM, TRICHY – 621112.

CERTIFICATE

Certified that this is a bonafide record of work done by.. of **V** Semester in **UCB1511 -DESIGN AND ANALYSIS OF ALGORITHM LABORATORY** during the academic year 2023-2024.

His / Her University Register Number is …………...………….

Staff Incharge: Head of the Department

Date: Date:

Submitted for the Practical Examinations held on ……………………..……..

Internal Examiner **External Examiner**

Date: Date:

INDEX

EX. NO	DATE	TITLE	PAGE NO	SIGNATURE
1		Implementation of iterative and recursive algorithms for the given problem		
2		Empirical analysis of algorithms		
3		Implementation of divide-and-conquer sorting algorithms		
4		Implementation of closest-pairs algorithm		
5		Implementation of Huffman coding		
6		Implementation of Dijkstra’s and Prim’s algorithms		
7		Implementation of disjoint sets and Kruskal’s algorithm		
8		Implementation of dynamic programming algorithm for knapsack problem		
9		Implementation of backtracking to solve n-Queens and Hamilton circuits problems		
10		Implementation of iterative improvement strategy for stable marriage and maxflow problems		
11		Implementation of Branch and Bound technique to solve knapsack and TSP problems		
12		Implementation of approximation algorithms for knapsack and TSP problems		
		CONTENT BEYOND SYLLABUS		
13		Implementation of Stack Operations		
14		Implementation of Queue		

1.IMPLEMENTATION OF ITERATIVE AND RECURSIVE ALGORITHMS FOR THE GIVEN PROBLEM

SOURCE CODE:

```
#include <stdio.h>

int main() {
    int a[10], n, i, j, temp;
    int beg, end, mid, target;

    printf("Enter the total numbers: ");
    scanf("%d", &n);
    printf("Enter the array elements: ");

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

    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (a[j + 1] < a[j]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }printf("The sorted numbers are: ");for (i = 0; i < n; i++)
        printf("%4d", a[i]);

    beg = 0;
    end = n - 1;
    mid = (beg + end) / 2;
    printf("\nEnter the number to be searched: ");
    scanf("%d", &target);
    while (beg <= end && a[mid] != target) {
        if (target < a[mid])
            end = mid - 1;
        else
            beg = mid + 1;
        mid = (beg + end) / 2;
    }
    if (a[mid] == target) {
        printf("The number is found at position %d", mid+1);
    } else {
        printf("The number is not found.");
    }

    return 0;}
```

OUTPUT

Enter the total numbers: 6
Enter the array elements:
45
21
65
98
35
26
The sorted numbers are: 21 26 35 45 65 98
Enter the number to be searched: 45
The number is found at position 4

2. IMPLEMENTATION EMPIRICAL ANALYSIS OF ALGORITHMS

SOURCE CODE:

```
#include <stdio.h>

int fact_recursive(int n);
int fact_iterative(int n);

int main(void)
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    printf("%d! = %d\n", n, fact_iterative(n));

    return 0;
}

int fact_recursive(int n)
{
    // Base Case
    if (n == 0)
        return 1;

    return n * fact_recursive(n - 1);
}

int fact_iterative(int n)
{
    int i;
    int product = 1;

    for (i = 1; i <= n; i++)
    {
        product = product * i;
    }

    return product;
}
```

INPUT AND OUTPUT:

Enter a number: 5
5! = 120

3. IMPLEMENTATION MERGE SORT

SOURCE CODE:

```
#include <stdio.h>
int a[20];
void merge(int low, int mid, int high) {
    int b[20];
    int h = low, i = low, j = mid + 1, k;

    while (h <= mid && j <= high) {
        b[i++] = (a[h] <= a[j]) ? a[h++] : a[j++];
    }
    while (h <= mid) {
        b[i++] = a[h++];
    }
    while (j <= high) {
        b[i++] = a[j++];
    }
    for (k = low; k <= high; k++) {
        a[k] = b[k];
    }
}

void merge_sort(int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        merge_sort(low, mid);
        merge_sort(mid + 1, high);
        merge(low, mid, high);
    }
}

int main() {
    int num, i;

    printf("How many numbers do you want to sort (max 20)?\n");
```

```
    scanf("%d", &num);
    printf("Enter %d numbers:\n",num);

    for (i = 0; i < num; i++) { // Start the loop from 0
        scanf("%d", &a[i]);
    }merge_sort(0, num - 1);

    printf("\nSorted list (using MERGE SORT):\n");
    for (i = 0; i < num; i++) {
        printf("%d ", a[i]);
    }

    return 0;
}
```

INPUT AND OUTPUT:

```
How many numbers do you want to sort (max 20)?
8
Enter 8 numbers:
36
25
65
48
98
68
23
14

Sorted list (using MERGE SORT):
14 23 25 36 48 65 68 98
```

4.IMPLEMENTATION OF CLOSEST-PAIRS ALGORITHM

SOURCE CODE:

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

typedef struct point {
   int x, y;
}point;

int cmpX(point* p1, point* p2) {
   return (p1->x - p2->x);
}

int cmpY(point* p1, point* p2) {
   return (p1->y - p2->y);
}

float dist(point p1,point p2) {
   return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}

float findClosest( point xSorted[], point ySorted[], int n) {
   if (n <= 3) {
      float minDist = 9999.0;
      for (int i = 0; i < n; ++i) {
         for (int j = i + 1; j < n; ++j) {
            float distance = dist(xSorted[i], xSorted[j]);
            if (distance < minDist) {
               minDist = distance;
            }}}
      return minDist;
   }
   int mid = n / 2;
   point midPoint = xSorted[mid];
   point ySortedLeft[mid + 1];
   point ySortedRight[n - mid - 1];
   int leftIndex = 0, rightIndex = 0;
   for (int i = 0; i < n; i++) {
      if (ySorted[i].x <= midPoint.x) {
         ySortedLeft[leftIndex++] = ySorted[i];
      } else {
         ySortedRight[rightIndex++] = ySorted[i];
      }
   }
```

```
    float leftDist = findClosest(xSorted, ySortedLeft, mid);
    float rightDist = findClosest(ySorted + mid, ySortedRight, n - mid);
    float minDist = (leftDist < rightDist) ? leftDist : rightDist;

    point strip[n];
    int j = 0;

    for (int i = 0; i < n; i++) {
        if (abs(ySorted[i].x - midPoint.x) < minDist) {
            strip[j] = ySorted[i];
            j++;
        }
    }

    for (int i = 0; i < j; i++) {
        for (int k = i + 1; k < j && (strip[k].y - strip[i].y) < minDist; k++) {

            if (dist(strip[i], strip[k]) < minDist) {
                minDist = dist(strip[i], strip[k]);
            }}}
            return minDist;
}
float closestPair(point pts[], int n) {
    point xSorted[n];
    point ySorted[n];

    for (int i = 0; i < n; i++) {
        xSorted[i] = pts[i];
        ySorted[i] = pts[i];
    }
    qsort(xSorted, n, sizeof(struct point), cmpX);
    qsort(ySorted, n, sizeof(struct point), cmpY);

    return findClosest(xSorted, ySorted, n);
}
int main() {
    struct point P[] = { {2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4} };
    int n = 6;
    printf("The minimum distance is %f\n", closestPair(P, n));
    return 0;
}
```

Output

The minimum distance is 1.414214

5.IMPLEMENTATION OF HUFFMAN CODING

SOURCE CODE:

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

struct MinHNode {
    char item;
    unsigned freq;
    struct MinHNode *left, *right;
};

struct MinHeap {
    unsigned size, capacity;
    struct MinHNode **array;
};

struct MinHNode *newNode(char item, unsigned freq) {
    struct MinHNode *temp = (struct MinHNode *)malloc(sizeof(struct MinHNode));
    temp->left = temp->right = NULL;
    temp->item = item;
    temp->freq = freq;
    return temp;
}

struct MinHeap *createMinH(unsigned capacity) {
    struct MinHeap *minHeap = (struct MinHeap *)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct MinHNode **)malloc(minHeap->capacity * sizeof(struct MinHNode *));
    return minHeap;
}

void swapMinHNode(struct MinHNode **a, struct MinHNode **b) {
    struct MinHNode *t = *a;
    *a = *b;
    *b = t;
}

void minHeapify(struct MinHeap *minHeap, int idx) {
    int smallest = idx, left = 2 * idx + 1, 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) {
        swapMinHNode(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}int checkSizeOne(struct MinHeap *minHeap) {return (minHeap->size == 1);
}

struct MinHNode *extractMin(struct MinHeap *minHeap) {
    struct MinHNode *temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}

void insertMinHeap(struct MinHeap *minHeap, struct MinHNode *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;
}

void buildMinHeap(struct MinHeap *minHeap) {
    int n = minHeap->size - 1;

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

int isLeaf(struct MinHNode *root) {
    return !(root->left) && !(root->right);
}

struct MinHeap *createAndBuildMinHeap(char item[], int freq[], int size) {
    struct MinHeap *minHeap = createMinH(size);

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

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

```
    return minHeap;
}

struct MinHNode *buildHuffmanTree(char item[], int freq[], int size) {
    struct MinHNode *left, *right, *top;
    struct MinHeap *minHeap = createAndBuildMinHeap(item, freq, size);

    while (!checkSizeOne(minHeap)) {left = extractMin(minHeap); right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }

    return extractMin(minHeap);
}

void printHCodes(struct MinHNode *root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printHCodes(root->left, arr, top + 1);
    }

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

    if (isLeaf(root)) {
        printf(" %c | ", root->item);
        for (int i = 0; i < top; ++i)
            printf("%d", arr[i]);
        printf("\n");
    }
}

void HuffmanCodes(char item[], int freq[], int size) {
    struct MinHNode *root = buildHuffmanTree(item, freq, size);
    int arr[50], top = 0;

    printf(" Char | Huffman code ");
    printf("\n.............................\n");

    printHCodes(root, arr, top);
}

int main() {
    char arr[] = {'A', 'B', 'C', 'D'};
    int freq[] = {5, 1, 6, 3};
```

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

    HuffmanCodes(arr, freq, size);
}
```

INPUT AND OUTPUT:

```
Char | Huffman code
-----------------------
 C | 0
 B  | 100
 D  | 101
 A | 11
```

6(a). IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

```
#include <limits.h>
#include <stdio.h>
#include <stdbool.h>
#define V 9

int minDistance(int dist[], bool sptSet[])
{
// Initialize min value
int min = INT_MAX, min_index;
for (int v = 0; v < V; v++)
if (sptSet[v] == false && dist[v] <= min) min = dist[v], min_index = v;
return min_index;
}

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

void dijkstra(int graph[V][V], int src)
{
int dist[V];

bool sptSet[V];

for (int i = 0; i < V; i++)
dist[i] = INT_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
dist[v] = dist[u] + graph[u][v];
}

// print the constructed distance array
printSolution(dist, V);
```

```
}

// driver program to test above function
int main()
{

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },
{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },
{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },
{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },
{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },
{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },
{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },
{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
dijkstra(graph, 0);

return 0;
}
```

INPUT AND OUTPUT:

```
Vertex Distance from Source
0        0
1        4
2        12
3        19
4        21
5        11
6        9
7        8
8        14
```

6(b). IMPLEMENTATION OF PRIM'S ALGORITHMS

SOURCE CODE :

```
#include <stdio.h>
#include <stdbool.h>
#define INF 9999999
#define V 5

int G[V][V] = {
    {0, 9, 75, 0, 0},
    {9, 0, 95, 19, 42},
    {75, 95, 0, 51, 66},
    {0, 19, 51, 0, 31},
    {0, 42, 66, 31, 0}
};

int main() {
    int no_edge; // number of edges
    int selected[V];
    for (int i = 0; i < V; i++) {
        selected[i] = false;
    }

    no_edge = 0;
    selected[0] = true;
    int x; // row number
    int y; // col number

    printf("Edge : Weight\n");

    while (no_edge < V - 1) {
        int min = INF;
        x = 0;
        y = 0;
        for (int i = 0; i < V; i++) {
            if (selected[i]) {
                for (int j = 0; j < V; j++) {
                    if (!selected[j] && G[i][j]) {
                        if (min > G[i][j]) {
                            min = G[i][j];
                            x = i;
                            y = j;
                        }
                    }
                }
            }
        }
```

```
      printf("%d - %d : %d\n", x, y, G[x][y]);
      selected[y] = true;
      no_edge++;
    }
    return 0;
}
```

INPUT AND OUTPUT:

Edge : Weight
0 - 1 : 9
1 - 3 : 19
3 - 4 : 31
3 - 2 : 51

7.IMPLEMENTATION OF KRUSKAL'S ALGORITHM

SOURCE CODE :

```
#include <stdio.h>

#define MAX 30

typedef struct edge {
   int u, v, w;
} edge;

typedef struct edge_list {
   edge data[MAX];
   int n;
} edge_list;

edge_list elist;
int Graph[MAX][MAX], n;
edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();

void kruskalAlgo() {
   int belongs[MAX], i, j, cno1, cno2;
   elist.n = 0;

   for (i = 1; i < n; i++) {
      for (j = 0; j < i; j++) {
         if (Graph[i][j] != 0) {
            elist.data[elist.n].u = i;
            elist.data[elist.n].v = j;
            elist.data[elist.n].w = Graph[i][j];
            elist.n++;
         }
      }
   }

   sort();

   for (i = 0; i < n; i++) {
      belongs[i] = i;
   }
```

```
    spanlist.n = 0;

    for (i = 0; i < elist.n; i++) {
        cno1 = find(belongs, elist.data[i].u);
        cno2 = find(belongs, elist.data[i].v);
        if (cno1 != cno2) {
            spanlist.data[spanlist.n] = elist.data[i];
            spanlist.n = spanlist.n + 1;
            applyUnion(belongs, cno1, cno2);
        }
    }
}

int find(int belongs[], int vertexno) {
    return (belongs[vertexno]);
}

void applyUnion(int belongs[], int c1, int c2) {
    int i;

    for (i = 0; i < n; i++) {
        if (belongs[i] == c2) {
            belongs[i] = c1;
        }
    }
}

void sort() {
    int i, j;
    edge temp;

    for (i = 1; i < elist.n; i++) {
        for (j = 0; j < elist.n - 1; j++) {
            if (elist.data[j].w > elist.data[j + 1].w) {
                temp = elist.data[j];
                elist.data[j] = elist.data[j + 1];
                elist.data[j + 1] = temp;
            }
        }
    }
}

void print()  {
    int i, cost = 0;

    for (i = 0; i < spanlist.n; i++) {
        printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
        cost = cost + spanlist.data[i].w;
```

```
    }

    printf("\nSpanning tree cost: %d\n", cost);
}

int main() {
    int i, j, total_cost;
    n = 6;
    // Define the
    graph int
    graph[6][6] = {
        {0, 4, 4, 0, 0, 0},
        {4, 0, 2, 0, 0, 0},
        {4, 2, 0, 3, 4, 0},
        {0, 0, 3, 0, 3, 0},
        {0, 0, 4, 3, 0, 0},
        {0, 0, 2, 0, 3, 0}
    };

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            Graph[i][j] = graph[i][j];
        }
    }

    kruskalAlgo();
    print();

    return 0;
```

INPUT AND OUTPUT:

```
2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14
```

8.IMPLEMENTATION OF DYNAMIC PROGRAMMING ALGORITHM FOR KNAPSACK 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 val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);

    printf("Maximum value = %d\n", knapsack(W, wt, val, n));
    return 0;
}
```

INPUT AND OUTPUT:

Maximum value = 220

9.IMPLEMENTATION OF BACKTRACKING TO SOLVE N-QUEENS

SOURCE CODE :

```
#include <stdio.h>

int is_attack(int i, int j, int board[5][5], int N) {
    int k, l;

    // checking for column j
    for (k = 1; k <= i - 1; k++) {
        if (board[k][j] == 1)
            return 1;
    }

    // checking upper right diagonal
    k = i - 1;
    l = j + 1;
    while (k >= 1 && l <= N) {
        if (board[k][l] == 1)
            return 1;
        k = k - 1;
        l = l + 1;
    }

    // checking upper left diagonal
    k = i - 1;
    l = j - 1;
    while (k >= 1 && l >= 1) {
        if (board[k][l] == 1)
            return 1;
        k = k - 1;
        l = l - 1;
    }

    return 0;
}

int n_queen(int row, int n, int N, int board[5][5]) {
    if (n == 0)
        return 1;

    int j;
    for (j = 1; j <= N; j++) {
        if (!is_attack(row, j, board, N)) {
            board[row][j] = 1;
            if (n_queen(row + 1, n - 1, N, board))
```

```
            return 1;
          board[row][j] = 0; // backtracking
        }
    }
    return 0;
}
int main() {
    int board[5][5];
    int i, j;

    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            board[i][j] = 0;
        }
    }

    n_queen(1, 4, 4, board);

    // printing the matrix
    for (i = 1; i <= 4; i++) {
        for (j = 1; j <= 4; j++) {
            printf("%d\t", board[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

OUTPUT:

```
0   1   0   0
0   0   0   1
1   0   0   0
0   0   1   0
```

10.IMPLEMENTATION OF ITERATIVE IMPROVEMENT STRATEGYFOR STABLE MARRIAGE AND MAXFLOW PROBLEMS

SOURCE CODE :

```
#include <stdio.h>
#define A 0
#define B 1
#define C 2
#define MAX_NODES 1000
#define O 1000000000

int n; int e;
int capacity[MAX_NODES][MAX_NODES]; int flow[MAX_NODES][MAX_NODES];
int color[MAX_NODES]; int pred[MAX_NODES];
int min(int x, int y) { return x < y ? x : y;
}

int head, tail;
int q[MAX_NODES + 2];
void enqueue(int x) { q[tail] = x;
tail++; color[x] = B;
}
int dequeue() { int x = q[head]; head++; color[x] = C; return x;
}

// Using BFS as a searching algorithm
int bfs(int start, int target) {
int u, v;
for (u = 0; u < n; u++) { color[u] = A;
}
head = tail = 0; enqueue(start); pred[start] = -1; while (head != tail) { u = dequeue();

for (v = 0; v < n; v++) {
if (color[v] == A && capacity[u][v] - flow[u][v] > 0) { enqueue(v);
pred[v] = u;
}
}
}
return color[target] == C;
}
// Applying fordfulkerson algorithm
int fordFulkerson(int source, int sink) { int i, j, u;
int max_flow = 0;
for (i = 0; i < n; i++) { for (j = 0; j < n; j++) { flow[i][j] = 0;
}
}
// Updating the residual values of edges
```

```
while (bfs(source, sink)) {
int increment = O;
for (u = n - 1; pred[u] >= 0; u = pred[u]) {
increment = min(increment, capacity[pred[u]][u] - flow[pred[u]][u]);
}
for (u = n - 1; pred[u] >= 0; u = pred[u]) { flow[pred[u]][u] += increment; flow[u][pred[u]] -=increment;
}
// Adding the path flows
max_flow += increment;
}
return max_flow;
}
int main() {
for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { capacity[i][j] = 0;
}
}
n = 6;
e = 7;
capacity[0][1] = 8;
capacity[0][4] = 3;
capacity[1][2] = 9;
capacity[2][4] = 7;
capacity[2][5] = 2;
capacity[3][5] = 5;
capacity[4][2] = 7;
capacity[4][3] = 4; int s = 0, t = 5;
printf("Max Flow : %d\n", fordFulkerson(s, t));
return 0;
}
```

OUTPUT

Max Flow : 6

11.IMPLEMENTATION OF BRANCH AND BOUND TECHNIQUE TO SOLVE KNAPSACK AND TSP PROBLEMS

SOURCE CODE :

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

int a[10][10], visited[10], n, cost = 0;
int least(int c);
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("Enter 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;
    }
    cost += a[city][ncity];
    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[c][i];
                kmin = a[c][i];
                nc = i;
            }
        }
    }
    if (min != 999)
        return nc;
    return 999;
}

void put() {
    printf("\n\nMinimum cost: %d\n", cost);
}

int main() {
    get();
    printf("\n\nThe Path is:\n");
    mincost(0);
    put();
    getch();
    return 0;
}
```

OUTPUT
Enter No. of Cities: 4
Enter Cost Matrix:
Enter Elements of Row # 1:
0 10 15 20
Enter Elements of Row # 2:
10 0 35 25
Enter Elements of Row # 3:
15 35 0 30
Enter Elements of Row # 4:
20 25 30 0

The cost list is:

0	10	15	20
10	0	35	25
15	35	0	30
20	25	30	0

The Path is:
1 --> 2 --> 4 --> 3 --> 1

Minimum cost: 80

12.IMPLEMENTATION OF APPROXIMATIONALGORITHMS FOR KNAPSACK AND TSP PROBLEMS

SOURCE CODE :

```
#include <stdio.h>
int least(int c);
int  ary[10][10];
int completed[10];
int n;
int cost = 0;

void takeInput() {
   int i, j;
   printf("Enter the number of villages: ");
   scanf("%d", &n);
   printf("\nEnter the Cost Matrix\n");

   for (i = 0; i < n; i++) {
      printf("\nEnter Costs from Village %d to all other Villages:\n", i + 1);
      for (j = 0; j < n; j++) {
         scanf("%d", &ary[i][j]);
      }
      completed[i] = 0;
   }

   printf("\n\nThe cost list is:\n");
   for (i = 0; i < n; i++) {
      printf("\n");
      for (j = 0; j < n; j++) {
         printf("\t%d", ary[i][j]);
      }
   }
}

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

```
    cost += ary[city][ncity];
    mincost(ncity);
}
int least(int c) { int i, nc = 999;int min = 999;int kmin;
    for (i = 0; i < n; i++) {
        if (ary[c][i] != 0 && completed[i] == 0) {
            if (ary[c][i] + ary[i][c] < min) {
                min = ary[c][i] + ary[i][c];
                kmin = ary[c][i];
                nc = i;
            }
        }
    }
    if (min != 999) {
        cost += kmin;
    }
    return nc;
}

int main() {
    takeInput();
    printf("\n\nThe Path is:\n");
    mincost(0); // starting from Village 1
    printf("\n\nMinimum cost is %d\n", cost);
    return 0;
}
```

INPUT AND OUTPUT

Enter the number of

villages: 4 Enter the Cost

Matrix

Enter Costs from Village 1 to all other Villages:
0 4 1 3

Enter Costs from Village 2 to all other Villages:
4 0 2 1

Enter Costs from Village 3 to all other Villages:
1 2 0 5

Enter Costs from Village 4 to all other Villages:
3 1 5 0

The cost list is:

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

The Path is:
1--->3--->2--->4--->1

Minimum cost is 11

13.Implementation of Stack Operations

PROGRAM:

```
#include
<stdio.h>
#include
<conio.h>
#define max 5
static int
stack[max]; inttop
= -1;
void push(int x)
{
stack[++top] = x;
}
int pop()
{
return (stack[top--]);
}
void view()
{
int i;
if (top < 0)
printf("\n Stack Empty
\n"); else
{
printf("\n Top-->");
for(i=top; i>=0; i--)
{
printf("%4d", stack[i]);
}
printf("\n");
}
}
main()
{
int ch=0, val;
clrscr();while(ch
!= 4)
{
printf("\n STACK OPERATION
\n");printf("1.PUSH ");
printf("2.POP ");
printf("3.VIEW ");
printf("4.QUIT \n");
printf("Enter Choice
: ");scanf("%d",
```

```
&ch); switch(ch)
{
case 1:
if(top < max-1)
{
printf("\nEnter Stack element : ");
scanf("%d", &val);push(val);
}
else
printf("\n Stack Overflow
\n");break;
case 2:
if(top < 0)
printf("\n Stack Underflow
\n");else
{
val = pop();
printf("\n Popped element is %d\n", val);
}
break; case 3:
view(); break;
case 4:exit(0);
default:
printf("\n Invalid Choice \n");
}
}
}
```

OUTPUT

STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element
: 12STACK
OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element
: 23STACK
OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element
: 34STACK
OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element
: 45STACK
OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice :
3 Top--> 45 34
23 12
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 2
Popped element is
45 STACK
OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter
Choice : 3
Top--> 34
23 12
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 4

14.Implementation of Queue

Program

```
#include  <stdio.h>
#include <conio.h>
#define max 5
static int queue[max]; int front = -1;
int rear = -1; void insert(int x)
{
queue[++rear] = x; if (front == -1) front = 0;
}
int remove()
{
int val;
val = queue[front];
if (front==rear && rear==max-1) front = rear = -1;
else front ++;
return (val);
}
void view()
{
int i;
if (front == -1)
printf("\n Queue Empty \n"); else
{
printf("\n Front-->"); for(i=front; i<=rear; i++)
printf("%4d", queue[i]);
printf(" <--Rear\n");
}
}
main()
{
int ch= 0,val; clrscr();
while(ch != 4)
{
printf("\n QUEUE OPERATION \n");
printf("1.INSERT ");
printf("2.DELETE ");
printf("3.VIEW ");
printf("4.QUIT\n"); printf("Enter Choice : ");
scanf("%d", &ch); switch(ch)
{
case 1:
if(rear < max-1)
{
printf("\n Enter element to be inserted : "); scanf("%d",
&val);
insert(val);
```

```
}
else
printf("\n Queue Full \n"); break;
case 2:
if(front == -1)
printf("\n Queue Empty \n"); else
{
val = remove();
printf("\n Element deleted : %d \n", val);
}
break; case 3: view();
break; case 4:
exit(0); default:
printf("\n Invalid Choice \n");
}
}
}
```

OUTPUT:

```
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be
inserted : 12
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be
inserted : 23
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be
inserted : 34
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be
inserted : 45
 QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be
inserted : 56
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Queue Full
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 3
Front--> 12 23 34 45 56 <--Rear
```