DAA PRACTICAL CODES

1)*Min Max Using Divide and Conquer*

Code:

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

int max, min;
int a[100];

// Function to find maximum and minimum elements in the array
void maxmin(int i, int j) {
   int max1, min1, mid;
   if(i == j) {                    // If only one element is present
      max = min = a[i];
   } else {
      if(i == j - 1) {             // If only two elements are present
         if(a[i] < a[j]) {
            max = a[j];
            min = a[i];
         } else {
            max = a[i];
            min = a[j];
         }
      } else {                      // If more than two elements are present
         mid = (i + j) / 2;
         maxmin(i, mid);          // Recursively find max and min in the first half
         max1 = max;
         min1 = min;
         maxmin(mid + 1, j);   // Recursively find max and min in the second half
         if(max < max1)
            max = max1;
         if(min > min1)
            min = min1;
      }
   }
}

int main() {
   int i, num;
   float average = 0;
   clrscr();
   printf("\nEnter the total number of numbers : ");
   scanf("%d", &num);

   printf("Enter the numbers : \n");
   for(i = 0; i < num; i++)
```

HARSH CHITALIYA

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

    max = a[0];
    min = a[0];

    maxmin(0, num - 1);  // Call the function to find max and min

    // Calculate average
    for(i = 0; i < num; i++)
        average += a[i];
    average /= num;

    // Output results
    printf("Minimum element in the array : %d\n", min);
    printf("Maximum element in the array : %d\n", max);
    printf("Average of the elements in the array: %.2f\n", average);
    printf("Range of the elements in the array: %d\n", max - min);

    getch();
    return 0;
}
```

Output:

```
Enter the total number of numbers : 5
Enter the numbers :
23
45
67
98
34
Minimum element in the array : 23
Maximum element in the array : 98
Average of the elements in the array: 53.40
Range of the elements in the array: 75
```

HARSHIT ALIYA

2) Stassen's matrix multiplication using divide and conquer

Code:

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

void multiply(int A[][2], int B[][2], int C[][2]) {
   int i, j, k;
   for (i = 0; i < 2; i++) {
      for (j = 0; j < 2; j++) {
         C[i][j] = 0;
         for (k = 0; k < 2; k++) {
            C[i][j] += A[i][k] * B[k][j];
         }
      }
   }
}
void optimize_multiply(int A[][2], int B[][2], int D[][2]) {
   int P1 = A[0][0] * (B[0][1] - B[1][1]);
   int P2 = (A[0][0] + A[0][1]) * B[1][1];
   int P3 = (A[1][0] + A[1][1]) * B[0][0];
   int P4 = A[1][1] * (B[1][0] - B[0][0]);
   int P5 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
   int P6 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);
   int P7 = (A[0][0] - A[1][0]) * (B[0][0] + B[0][1]);
   D[0][0] = P5 + P4 - P2 + P6;
   D[0][1] = P1 + P2;
   D[1][0] = P3 + P4;
   D[1][1] = P5 + P1 - P3 - P7;
}
int main() {
   int i,j;
   int A[2][2], B[2][2], C[2][2], D[2][2];
   clrscr();
   printf("Enter the elements of Matrix A:\n");
   for ( i = 0; i < 2; i++) {
      for ( j = 0; j < 2; j++) {
         scanf("%d", &A[i][j]);
      }
   }
   printf("Enter the elements of Matrix B:\n");
   for ( i = 0; i < 2; i++) {
      for ( j = 0; j < 2; j++) {
         scanf("%d", &B[i][j]);
      }
   }
```

HARSH CHIT ALIYA

```
    printf("Entered Matrix A is:\n");
    for ( i = 0; i < 2; i++) {
        for ( j = 0; j < 2; j++) {
            printf("%d\t", A[i][j]);
        }
        printf("\n");
    }
    printf("Entered Matrix B is:\n");
    for ( i = 0; i < 2; i++) {
        for ( j = 0; j < 2; j++) {
            printf("%d\t", B[i][j]);
        }
        printf("\n");
    }
    multiply(A, B, C);
    printf("Resultant Matrix is:\n");
    for ( i = 0; i < 2; i++) {
        for ( j = 0; j < 2; j++) {
            printf("%d\t", C[i][j]);
        }
        printf("\n");
    }
    optimize_multiply(A, B, D);
    printf("Optimized Resultant Matrix is:\n");
    for ( i = 0; i < 2; i++) {
        for ( j = 0; j < 2; j++) {
            printf("%d\t", D[i][j]);
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

Output:

```
Enter the elements of Matrix A:
12 23 34 45
Enter the elements of Matrix B:
56 67 70 89
Entered Matrix A is:
12      23
34      45
Entered Matrix B is:
56      67
70      89
Resultant Matrix is:
2466    2851
5414    6283
Optimized Resultant Matrix is:
2466    2851
5414    6283
```

ARSH CHITALIYA

3a) Quicksort using divide and conquer

Code:

```
#include <stdio.h>
#include<conio.h>
 // Function to swap two elements
void swap(int* a, int* b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

// Partition function
int partition(int arr[], int low, int high) {

   // initialize pivot to be the first element
   int pivot = arr[low];
   int i = low;
   int j = high;

   while (i < j) {

      // condition 1: find the first element greater than
      // the pivot (from starting)
      while (arr[i] <= pivot && i <= high - 1) {
         i++;
      }

      // condition 2: find the first element smaller than
      // the pivot (from last)
      while (arr[j] > pivot && j >= low + 1) {
         j--;
      }
      if (i < j) {
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[low], &arr[j]);
   return j;
}

// QuickSort function
void quickSort(int arr[], int low, int high) {
   if (low < high) {

      // call Partition function to find Partition Index
      int partitionIndex = partition(arr, low, high);
```

HARSH CHITALIYA

```
        // Recursively call quickSort() for left and right
        // half based on partition Index
        quickSort(arr, low, partitionIndex - 1);
        quickSort(arr, partitionIndex + 1, high);
    }
}
// driver code
int main() {
    int n,i ,arr[200];
    clrscr();
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Taking user input for the array
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    // printing the original array
    printf("Original array: ");
    for ( i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    // calling quickSort() to sort the given array
    quickSort(arr, 0, n - 1);

    // printing the sorted array
    printf("\nSorted array: ");
    for ( i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    getch();
    return 0;
}
```

Output:

```
Enter the number of elements: 5
Enter 5 elements:
34
78
12
23
98
Original array: 34 78 12 23 98
Sorted array: 12 23 34 78 98 _
```

HARSH CHITALIYA

3b) Merge sort using divide and conquer

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define max 10
int *a;    // Define 'a' as a pointer
int b[max + 1];     // Define 'b' as a fixed-size array
void merging(int low, int mid, int high) {
   int l1, l2, i;

   for (l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
      if (a[l1] <= a[l2])
         b[i] = a[l1++];
      else
         b[i] = a[l2++];
   }

   while (l1 <= mid)
      b[i++] = a[l1++];

   while (l2 <= high)
      b[i++] = a[l2++];

   for (i = low; i <= high; i++)
      a[i] = b[i];
}

void sort(int low, int high) {
   int mid;

   if (low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid + 1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}

int main() {
   int i, n;

   clrscr();
```

HARSH CHITALIYA

```
    printf("Enter the number of elements (maximum %d):\n", max + 1);
    scanf("%d", &n);

    // Check if the number of elements is valid
    if (n < 0 || n > max + 1) {
        printf("Invalid number of elements.\n");
        return 1;                    // Exit with error code 1
    }

    // Dynamically allocate memory for 'a'
    a = (int*)malloc((max + 1) * sizeof(int));

    // Prompt the user to enter elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    printf("List before sorting:\n");

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

    sort(0, n - 1);

    printf("\nList after sorting:\n");

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

    // Free dynamically allocated memory
    free(a);
    getch();
    return 0;
}
```

Output:

```
Enter the number of elements (maximum 11):
5
Enter 5 elements:
12
34
2
78
91
List before sorting:
12 34 2 78 91
List after sorting:
2 12 34 78 91
```

HARSH CHITALIYA

5) Single source shortest path Dijkstras

Code:

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

#define MAX_VERTICES 100

// Function to find the vertex with minimum distance value, from the set of vertices not yet included in the shortest path tree
int minDistance(int dist[], int visited[], int V) {
   int min = INT_MAX, min_index;
   int v;

   for (v = 0; v < V; v++) {
      if (visited[v] == 0 && dist[v] <= min) {
         min = dist[v];
         min_index = v;
      }
   }
   return min_index;
}

// Function to print the final shortest distances from the source vertex to all other vertices
void printSolution(int dist[], int V) {
   int i;
   printf("Vertex \t Distance from Source\n");
   for (i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}

// Dijkstra's algorithm for finding the shortest paths from a source vertex to all other vertices
void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int src, int V) {
   int dist[MAX_VERTICES]; // Array to store the shortest distance from src to i
   int visited[MAX_VERTICES]; // Array to keep track of visited vertices
   int i, count, v;

   // Initialize all distances as INFINITE and visited[] as false
   for (i = 0; i < V; i++) {
      dist[i] = INT_MAX;
      visited[i] = 0;
   }
   // Distance of source vertex from itself is always 0
   dist[src] = 0;
```

HARSH CHITALIYA

```
    // Find shortest path for all vertices
    for (count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not yet processed
        int u = minDistance(dist, visited, V);

        // Mark the picked vertex as visited
        visited[u] = 1;

        // Update dist value of the adjacent vertices of the picked vertex
        for (v = 0; v < V; v++) {
            // Update dist[v] only if it's not in visited, there's an edge from u to v, and total
weight of path from src to v through u is smaller than current value of dist[v]
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] <
dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Print the calculated shortest distances
    printSolution(dist, V);
}

int main() {
    int V;
    int graph[MAX_VERTICES][MAX_VERTICES];
    int i, j;
    int source;

    clrscr();
    printf("Enter the number of vertices: ");
    scanf("%d", &V);

    printf("Enter the adjacency matrix (0 for no connection):\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    printf("Enter the source vertex: ");
    scanf("%d", &source);

    // Call Dijkstra's algorithm function
    dijkstra(graph, source, V);
```

HARSH CHITALIYA

```
    getch();
    return 0;
}
```

Output:

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

HARSH

6) Activity selection problem

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int start[100], finish[100];
char task[100], activity[100];
// Function to swap two integers
void swap(int a[], int j) {
   int temp1;
   temp1 = a[j];
   a[j] = a[j+1];
   a[j+1] = temp1;
}
// Function to swap two characters
void swap2(char a[], int j) {
   char temp1;
   temp1 = a[j];
   a[j] = a[j+1];
   a[j+1] = temp1;
}
int main() {
   int n, i=0, j, temp1, temp2, temp3 , count;
   clrscr();
   printf("Enter the number of tasks: ");
   scanf("%d", &n);
   printf("Enter the start time of tasks: ");
   for(i=0; i<n; i++) {
```

HARSH CHITALIYA

```
    scanf("%d", &start[i]);
    task[i] = i+1;
}
printf("Enter the finish time of tasks: ");
for(i=0; i<n; i++) {
    scanf("%d", &finish[i]);
}
// Bubble sort based on finish times
for(i=0; i<n-1; i++) {
    for(j=0; j<n-i-1; j++) {
        if(finish[j] > finish[j+1]) {
            swap(finish, j);
            swap(start, j);
            swap2(task, j);
        }
    }
}
// Selecting compatible tasks
activity[0] = task[0];
i = 0;
count = 1;
for(j=1; j<n; j++) {
    if(start[j] >= finish[i]) {
        activity[count++] = task[j];
        i=j;
    }
}
```

HARSH CHITALIYA

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

Output:

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

Tasks that are selected : 3     7       6
```

7) Fractional knapsack

Code:

```
#include<stdio.h>
#include<conio.h>
// Function to solve the 0/1 knapsack problem
void knapsack(int n, float weight[], float profit[], float capacity) {
   float x[20], tp = 0;
   int i, j, u;
   u = capacity;

   // Initialize decision vector
   for (i = 0; i < n; i++)
      x[i] = 0.0;

   // Greedy algorithm to fill the knapsack
   for (i = 0; i < n; i++) {
      if (weight[i] > u)
         break;
      else {
         x[i] = 1.0;
         tp = tp + profit[i];
         u = u - weight[i];
      }
   }
   // Partially fill the knapsack if needed
   if (i < n)
      x[i] = u / weight[i];
   tp = tp + (x[i] * profit[i]);
```

HARSH CHITALIYA

```
    // Print the result vector
    printf("\nThe result vector is:- ");
    for (i = 0; i < n; i++)
        printf("%f\t", x[i]);

    // Print the maximum profit
    printf("\nMaximum profit is:- %f", tp);
}
int main() {
    float weight[20], profit[20], capacity;
    int num, i, j;
    float ratio[20], temp;
    clrscr();
        // Input the number of objects
    printf("\nEnter the no. of objects:- ");
    scanf("%d", &num);
    // Input the weights and profits of each object
    for (i = 0; i < num; ++i) {
        printf("\nEnter the weights and profits of each object %d: ", i+1);
        scanf("%f %f", &weight[i], &profit[i]);
    }
    // Input the capacity of the knapsack
    printf("\nEnter the capacity of knapsack:- ");
    scanf("%f", &capacity);
    // Calculate profit-to-weight ratio for each object
    for (i = 0; i < num; i++) {
        ratio[i] = profit[i] / weight[i];
    }
```

HARSH CHITALIYA

```
    // Sort objects based on profit-to-weight ratio in non-increasing order
    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;
            }
        }
    }
    // Call the knapsack function
    knapsack(num, weight, profit, capacity);
    getch();
    return(0);
}
```

HARSH CHITALIYA

Output:

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

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

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

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

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

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

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

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

Enter the capacity of knapsack:- 15

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

HARSH CHITADA

8) Prims algorithm

Code:

```
#include <stdio.h>
#include <limits.h>
#include <conio.h>
#define vertices 5
// Function to find the vertex with minimum key value
int minimum_key(int k[], int mst[]) {
   int minimum = INT_MAX, min, i;
   for (i = 0; i < vertices; i++) {
      if (mst[i] == 0 && k[i] < minimum) {
         minimum = k[i];
         min = i;
      }
   }
   return min;
}
// Function to perform Prim's algorithm
void prim(int g[vertices][vertices]) {
   int parent[vertices];  // Array to store the parent node of each vertex in MST
   int k[vertices];       // Array to store key values used to pick minimum weight edge
   int mst[vertices];     // Array to mark vertices included in MST
   int i, count, edge, v; // Loop variables
   int sum = 0;           // Total cost of the minimum spanning tree
   // Initialize key values, MST set, and parent array
   for (i = 0; i < vertices; i++) {
      k[i] = INT_MAX;
      mst[i] = 0;
   }
```

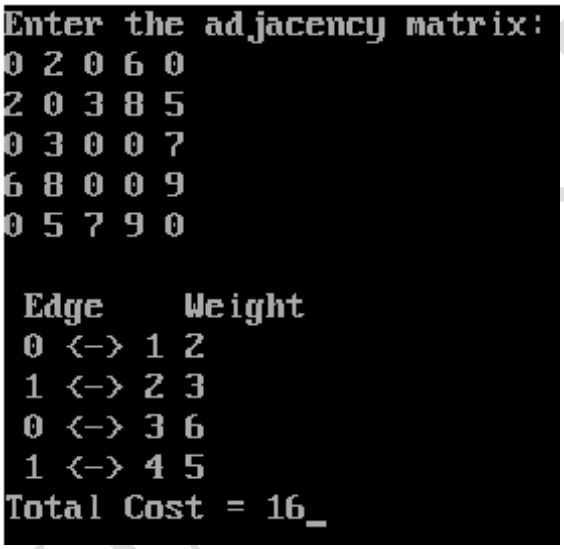
HARSH CHITALIYA

```
    k[0] = 0;       // Start with the first vertex
    parent[0] = -1;  // First vertex is root of MST
    // Construct MST
    for (count = 0; count < vertices - 1; count++) {
        // Pick the minimum key vertex from the set of vertices not yet included in MST
        edge = minimum_key(k, mst);
        mst[edge] = 1;  // Add the picked vertex to MST set
        // Update key value and parent index of adjacent vertices of the picked vertex
        for (v = 0; v < vertices; v++) {
            if (g[edge][v] && mst[v] == 0 && g[edge][v] < k[v]) {
                parent[v] = edge;
                k[v] = g[edge][v];
            }
        }
    }
    // Print the edges of the MST and calculate total cost
    printf("\n Edge \t Weight\n");
    for (i = 1; i < vertices; i++) {
        printf(" %d <-> %d %d \n", parent[i], i, g[i][parent[i]]);
        sum += g[i][parent[i]];
    }
    // Print the total cost of the MST
    printf("Total Cost = %d", sum);
}
int main() {
    int g[vertices][vertices];
    int i, j;
    clrscr();  // Clear the screen
```

HARSH CHITALIYA

```
    // Prompt the user to enter the adjacency matrix
    printf("Enter the adjacency matrix:\n");
    for (i = 0; i < vertices; i++) {
        for (j = 0; j < vertices; j++) {
            scanf("%d", &g[i][j]);
        }
    }
    // Perform Prim's algorithm and display the minimum spanning tree
    prim(g);
    getch();  // Wait for a key press before exiting
    return 0;
}
```

Output:



HARSHIT ALIYA

9) Job sequencing with deadline

Code:

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

// Structure to represent a job
typedef struct {
   int id;        // Job ID
   int profit;    // Profit from the job
   int deadline;  // Deadline of the job
} Job;

// Function to compare jobs based on profit (used in qsort)
int compare_jobs(const void* a, const void* b) {
   return ((Job*)b)->profit - ((Job*)a)->profit;
}

// Function to find the maximum deadline among all jobs
int max_deadline(Job* jobs, int n) {
   int i;
   int max = 0;
   for (i = 0; i < n; i++) {
      if (jobs[i].deadline > max) {
         max = jobs[i].deadline;
      }
   }
   return max;
}

// Function to schedule jobs with deadlines
void schedule_with_deadline(Job* jobs, int n) {
   int i, j;
   int max_dl = max_deadline(jobs, n);
   int* schedule = (int*)malloc((max_dl + 1) * sizeof(int));
   int* profit_set = (int*)malloc((max_dl + 1) * sizeof(int));
   int total_profit = 0;

   // Sort jobs based on profit in non-decreasing order
   qsort(jobs, n, sizeof(Job), compare_jobs);

   // Initialize schedule and profit set arrays
   for (i = 0; i <= max_dl; i++) {
```

HARSH CHITALIYA

```
        schedule[i] = -1;     // Initialize schedule with -1 (empty slot)
        profit_set[i] = 0;    // Initialize profit set with 0
    }

    // Schedule jobs and calculate profit set
    for (i = 0; i < n; i++) {
        for (j = jobs[i].deadline; j > 0; j--) {
            if (schedule[j] == -1) {
                schedule[j] = jobs[i].id;
                profit_set[j] = jobs[i].profit;
                total_profit += jobs[i].profit;
                break;
            }
        }
    }

    // Print the schedule
    printf("Optimal schedule J: ");
    for (i = 1; i <= max_dl; i++) {
        if (schedule[i] != -1) {
            printf("%d ", schedule[i]);
        }
    }
    printf("\n");

    // Print the profit set
    printf("Profit set after job scheduling: ");
    for (i = 1; i <= max_dl; i++) {
        printf("%d ", profit_set[i]);
    }
    printf("\n");

    // Print total profit
    printf("Total profit: %d\n", total_profit);

    // Free dynamically allocated memory
    free(schedule);
    free(profit_set);
}

int main() {
    int n, i;
    Job* jobs;      // Declare the jobs array
```

HARSH CHITALIYA

```
    clrscr();
    printf("Enter the number of jobs: ");
    scanf("%d", &n);
    jobs = (Job*)malloc(n * sizeof(Job));       // Allocate memory for jobs array

    // Input job details
    for (i = 0; i < n; i++) {
        printf("Enter job ID, profit, and deadline for job %d: ", i + 1);
        scanf("%d %d %d", &jobs[i].id, &jobs[i].profit, &jobs[i].deadline);
    }

    // Schedule jobs
    schedule_with_deadline(jobs, n);

    // Free dynamically allocated memory
    free(jobs);
    getch();
    return 0;
}
```

Output:

```
Enter the number of jobs: 4
Enter job ID, profit, and deadline for job 1: 1 100 2
Enter job ID, profit, and deadline for job 2: 2 10 1
Enter job ID, profit, and deadline for job 3: 3 15 2
Enter job ID, profit, and deadline for job 4: 4 27 1
Optimal schedule J: 4 1
Profit set after job scheduling: 27 100
Total profit: 127
```

HARSHIT ALIYA

10) Matrix chain multiplication

Code:

```
#include <stdio.h>
#include <stdlib.h>                    // Include stdlib for using malloc
#include<conio.h>
int memo[10][10][2] = {0};       // Stores the minimum cost and split point

// Recursive function to compute minimum cost of matrix chain multiplication
int MatrixChainOrder(int p[], int i, int j) {
   int k;
   int min = 9999;
   int count;
   int k1;
   if (i == j) {
      return 0;
   }
   if (memo[i][j][0] != 0) {
      return memo[i][j][1];
   }
   for (k = i; k < j; k++) {
      count = MatrixChainOrder(p, i, k) + MatrixChainOrder(p, k + 1, j) + p[i - 1] * p[k] * p[j];
      if (count < min) {
         min = count;
         k1 = k;
      }
   }
   memo[i][j][0] = k1;
   memo[i][j][1] = min;
   return min;
}

// Function to print the optimal parenthesization
void printOptimalParens(int i, int j) {
   if (i == j) {
      printf("A%d", i);
   } else {
      printf("(");
      printOptimalParens(i, memo[i][j][0]);
      printOptimalParens(memo[i][j][0] + 1, j);
      printf(")");
   }
}
```

HARSH CHITALIYA

```
int main() {
    int n, i, j;
    int *p = malloc((n + 1) * sizeof(int));        // Dynamically allocate memory for dimensions
    clrscr();
    printf("Enter the number of matrices: ");
    scanf("%d", &n);

    if (p == NULL) {
        printf("Memory allocation failed.\n");
        return 1;                                  // Exit if memory allocation fails
    }

    printf("Enter the dimensions of matrices (including dimensions of result matrix):\n");
    for (i = 0; i <= n; i++) {
        printf("p[%d]: ", i);
        scanf("%d", &p[i]);
    }

    // Calculating minimum cost of matrix multiplication
    printf("The minimum cost matrix is: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (j >= i) {
                printf("%d ", MatrixChainOrder(p, i + 1, j + 1));
            } else {
                printf("0 ");
            }
        }
        printf("\n");
    }

    // Displaying the split points
    printf("\nThe split points (k-values) matrix is:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (j >= i) {
                printf("%d ", memo[i+1][j+1][0]);
            } else {
                printf("0 ");
            }
        }
        printf("\n");
    }
```

HARSH CHITALIYA

```
    // Printing the optimal parenthesization
    printf("\nOptimal parenthesization is: ");
    printOptimalParens(1, n);
    printf("\n");

    free(p);        // to free the allocated memory
    getch();
    return 0;
}
```

Output:

```
Enter the number of matrices: 4
Enter the dimensions of matrices (including dimensions of result matrix):
p[0]: 5
p[1]: 10
p[2]: 15
p[3]: 20
p[4]: 25
The minimum cost matrix is:
0 750 2250 4750
0 0 3000 8000
0 0 0 7500
0 0 0 0

The split points (k-values) matrix is:
0 1 2 3
0 0 2 3
0 0 0 3
0 0 0 0

Optimal parenthesization is: (((A1A2)A3)A4)
```

HARIYA

11) . All pair shortest path

Code:

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

void floydWarshall(int **graph, int n) {
   int i, j, k;
   for (k = 0; k < n; k++) {
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            if (graph[i][k] != INT_MAX && graph[k][j] != INT_MAX && graph[i][j] >
graph[i][k] + graph[k][j]) {
               graph[i][j] = graph[i][k] + graph[k][j];
            }
         }
      }
   }
}

int main() {
   int n, i, j;
   int **graph;
   clrscr();
   printf("Enter the number of vertices: ");
   scanf("%d", &n);

   // Allocate memory for the graph
   graph = (int **)malloc(n * sizeof(int *));
   if (graph == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   for (i = 0; i < n; i++) {
      graph[i] = (int *)malloc(n * sizeof(int));
      if (graph[i] == NULL) {
         printf("Memory allocation failed\n");
         exit(1);
      }
   }
```

HARSH CHITALIYA

```
// Initialize the graph with default values
for (i = 0; i < n; i++) {
   for (j = 0; j < n; j++) {
      if (i == j) {
         graph[i][j] = 0;
      } else {
         graph[i][j] = INT_MAX;
      }
   }
}

// Input edge weights
printf("Enter the edges: \n");
for (i = 0; i < n; i++) {
   for (j = 0; j < n; j++) {
      if (i != j) {
         printf("Enter weight for edge [%d][%d], or %d for no edge: ", i, j, INT_MAX);
         scanf("%d", &graph[i][j]);
         if (graph[i][j] == INT_MAX) { // Handle case where user input INT_MAX explicitly
            graph[i][j] = INT_MAX;
         }
      }
   }
}

// Display the original graph
printf("The original graph is:\n");
for (i = 0; i < n; i++) {
   for (j = 0; j < n; j++) {
      if (graph[i][j] == INT_MAX)
         printf(" INF ");
      else
         printf("%4d ", graph[i][j]);
   }
   printf("\n");
}

// Perform Floyd-Warshall algorithm to find shortest paths
floydWarshall(graph, n);

// Display the shortest path matrix
printf("The shortest path matrix is:\n");
for (i = 0; i < n; i++) {
```

HARSH CHITALIYA

```
        for (j = 0; j < n; j++) {
            if (graph[i][j] == INT_MAX)
                printf(" INF ");
            else
                printf("%4d ", graph[i][j]);
        }
        printf("\n");
    }

    // Free dynamically allocated memory
    for (i = 0; i < n; i++) {
        free(graph[i]);
    }
    free(graph);

    getch();
    return 0;
}
```

Output:

```
Enter the number of vertices: 4
Enter the edges:
Enter weight for edge [0][1], or 32767 for no edge: 32767
Enter weight for edge [0][2], or 32767 for no edge: -2
Enter weight for edge [0][3], or 32767 for no edge: 32767
Enter weight for edge [1][0], or 32767 for no edge: 4
Enter weight for edge [1][2], or 32767 for no edge: 3
Enter weight for edge [1][3], or 32767 for no edge: 32767
Enter weight for edge [2][0], or 32767 for no edge: 32767
Enter weight for edge [2][1], or 32767 for no edge: 32767
Enter weight for edge [2][3], or 32767 for no edge: 2
Enter weight for edge [3][0], or 32767 for no edge: 32767
Enter weight for edge [3][1], or 32767 for no edge: -1
Enter weight for edge [3][2], or 32767 for no edge: 32767
The original graph is:
   0  INF   -2  INF
   4    0    3  INF
 INF  INF    0    2
 INF   -1  INF    0
The shortest path matrix is:
   0   -1   -2    0
   4    0    2    4
   5    1    0    2
   3   -1    1    0
```

CHITALIYA

12) 0/1 knapsack problem

Code:

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

void fractional(int cap, int w[], int p[], int n);
int max(int a, int b)
{
   return (a > b) ? a : b;
}

void main()
{
   int cap, n, i;
   int w[101], p[101];
   int temp_w, temp_p, j;
   clrscr();
   printf("Enter the number of items: ");
   scanf("%d", &n);
   printf("Enter the capacities of the items:\n");
   for(i = 0; i < n; i++)
   {
      printf("Weight %d: ", i+1);
      scanf("%d", &w[i]);
      printf("Profit %d: ", i+1);
      scanf("%d", &p[i]);
   }

   printf("Enter the capacity of the knapsack: ");
   scanf("%d", &cap);

   // Sorting them according to the weight in ascending order
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n-i-1; j++)
      {
         if(w[j] > w[j+1])
         {
            temp_w = w[j];
            w[j] = w[j+1];
            w[j+1] = temp_w;
            temp_p = p[j];
            p[j] = p[j+1];
```

HARSH CHITALIYA

```
            p[j+1] = temp_p;
        }
      }
   }

   fractional(cap, w, p, n);
   getch();
}

void fractional(int cap, int w[], int p[], int n)
{
   int table[101][101];
   int i, j;

   // Initialize the first column and first row to 0
   for(i = 0; i <= n; i++)
   {
      table[i][0] = 0;
   }

   for(j = 0; j <= cap; j++)
   {
      table[0][j] = 0;
   }

   // Applying the formula
   for(i = 1; i <= n; i++)
   {
      for(j = 1; j <= cap; j++)
      {
         if(w[i-1] > j)
         {
            table[i][j] = table[i-1][j];
         }
         else
         {
            table[i][j] = max(table[i-1][j], p[i-1] + table[i-1][j-w[i-1]]);
         }
      }
   }
   // For table display
   printf("DP Table:\n");
   for (i = 0; i <= n; i++)
```

HARSH CHITALIYA

```
    {
        for(j = 0; j <= cap; j++)
        {
            printf("%d ", table[i][j]);
        }
        printf("\n");
    }
    printf("The maximum profit is %d\n", table[n][cap]);
}
```

Output:

```
Enter the number of items: 3
Enter the capacities of the items:
Weight 1: 4
Profit 1: 10
Weight 2: 6
Profit 2: 12
Weight 3: 8
Profit 3: 15
Enter the capacity of the knapsack: 10
DP Table:
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 10 10 10 10 10 10 10
0 0 0 0 10 10 12 12 12 12 22
0 0 0 0 10 10 12 12 15 15 22
The maximum profit is 22
```

HARSH PALIYA

13a) Coin change problem using dynamic

Code :

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

int min(int a, int b) {
   return (a < b) ? a : b;
}

int minCoins(int coins[], int m, int V) {
   int i, j, p;
   static int table[101][101];
   int sol[101];

   /* Initialize table */
   for (i = 0; i <= m; i++) {
      table[i][0] = 0;
   }
   for (i = 1; i <= V; i++) {
      table[0][i] = V + 1;    // Filling this with V+1 instead of i to signify unreachable with 0 coins
   }

   /* Fill the table */
   for (i = 1; i <= m; i++) {
      for (j = 1; j <= V; j++) {
         if (coins[i - 1] > j) {
            table[i][j] = table[i - 1][j];
         } else {
            table[i][j] = min(table[i - 1][j], 1 + table[i][j - coins[i - 1]]);
         }
      }
   }

   /* Print the table */
   printf("Capacity/Coins\t\n");
   printf("\t");
   for (i = 0; i <= V; i++) {
      printf("%4d", i);
   }
   printf("\n");
   for (i = 1; i <= m; i++) {
      printf("%3d\t", coins[i - 1]);
      for (j = 0; j <= V; j++) {
```

HARSH CHITALIYA

```
            if (table[i][j] >= V + 1)
                printf("  INF");
            else
                printf("%4d", table[i][j]);
        }
        printf("\n");
    }

    /* Determine the coins used */
    i = m, j = V, p = 0;
    while (j > 0 && i > 0) {
        if (table[i][j] == table[i - 1][j]) {
            i--;
        } else {
            j -= coins[i - 1];
            sol[p++] = coins[i - 1];
        }
    }

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

    return table[m][V];
}

int main() {
    int coins[100], m, V, i;
    int minCount;
    clrscr();
    printf("Enter the number of coins: ");
    scanf("%d", &m);

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

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

HARSH CHITALIYA

```
    minCount = minCoins(coins, m, V);
    printf("Minimum number of coins required: %d\n", minCount);
    getch();
    return 0;
}
```

Output:

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

HARSH CHITRANSHYA

13b) Coin change problem using greedy

Code:

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

void sortCoins(int coins[], int m) {
   int i ;
   // A simple insertion sort for sorting coins in descending order
   for ( i = 1; i < m; i++) {
      int key = coins[i];
      int j = i - 1;
      while (j >= 0 && coins[j] < key) {
         coins[j + 1] = coins[j];
         j = j - 1;
      }
      coins[j + 1] = key;
   }
}

int minCoinsGreedy(int coins[], int m, int V) {
   int count , i ;
   sortCoins(coins, m);         // Sort coins in decreasing order
   count = 0;
   printf("Coins used: ");
   for ( i = 0; i < m; i++) {
      while (V >= coins[i]) {
         V -= coins[i];
         printf("%d ", coins[i]);          // Print the coin used
         count++;                          // Increase the count of coins used
      }
      if (V == 0) break;              // If the exact change has been made, stop
   }
   printf("\n");
   return count;      // Return the total number of coins used
}

int main() {
   int coins[100], m, V, i;
   int minCount;
   clrscr();
   printf("Enter the number of coins: ");
   scanf("%d", &m);
```

HARSH CHITALIYA

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

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

    minCount = minCoinsGreedy(coins, m, V);
    printf("Minimum number of coins required using greedy approach: %d\n", minCount);
    getch();
    return 0;
}
```

Output:

```
Enter the number of coins: 4
Enter the coins: 1 3 5 9
Enter the total amount: 10
Coins used: 9 1
Minimum number of coins required using greedy approach: 2
```

HARSH CHAUHAN

14) LCS

Code:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
// Function to find the maximum of two integers
int max(int a, int b) {
   return (a > b) ? a : b;
}
// Function to compute and print the Longest Common Subsequence (LCS) of strings x and y
void LCS_computation_and_print(char *x, char *y) {
   int m = strlen(x);
   int n = strlen(y);
   int index;
   // Initialize LCS matrix
   int LCS[101][101];
   int i, j;
   char lcs[201];
   for (i = 0; i <= m; i++) {
      for (j = 0; j <= n; j++) {
         if (i == 0 || j == 0)
            LCS[i][j] = 0;
         else if (x[i - 1] == y[j - 1])
            LCS[i][j] = 1 + LCS[i - 1][j - 1];
         else
            LCS[i][j] = max(LCS[i - 1][j], LCS[i][j - 1]);
      }
   }

   // Following code is used to print LCS
   index = LCS[m][n];
   lcs[index] = '\0'; // Set the terminating character

   // Start from the right-most-bottom-most corner and
   // one by one store characters in lcs[]
   i = m, j = n;
   while (i > 0 && j > 0) {
      // If current character in x[] and y[] are same, then
      // current character is part of LCS
      if (x[i - 1] == y[j - 1]) {
         lcs[index - 1] = x[i - 1];          // Put current character in result
         i--; j--; index--;                  // reduce values of i, j and index
```

HARSH CHITALIYA

```
        }

        // If not same, then find the larger of two and
        // go in the direction of larger value
        else if (LCS[i - 1][j] > LCS[i][j - 1])
            i--;
        else
            j--;
    }

    // Print the LCS
    printf("LCS of \"%s\" and \"%s\" is \"%s\"\n", x, y, lcs);
    printf("Length of LCS: %d\n", LCS[m][n]);
}

int main() {
    char x[1000], y[1000];      // Declare strings to hold the input
    clrscr();

    printf("Enter first string: ");
    fgets(x, sizeof(x), stdin);              // Read the first string
    x[strcspn(x, "\n")] = 0;                 // Remove the newline character if any

    printf("Enter second string: ");
    fgets(y, sizeof(y), stdin);             // Read the second string
    y[strcspn(y, "\n")] = 0;                 // Remove the newline character if any

    LCS_computation_and_print(x, y);
    getch();
    return 0;
}
```

Output:

```
Enter first string: EXAMPLE
Enter second string: APE
LCS of "EXAMPLE" and "APE" is "APE"
Length of LCS: 3
```

HARSH CHITALIYA

15) . Bell man Ford algortihm

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include<conio.h>
struct Edge {
   int source, destination, weight;
};

struct Graph {
   int V;
   int E;
   struct Edge* edge;
};

struct Graph* createGraph(int V, int E) {
   struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
   graph->V = V;
   graph->E = E;
   graph->edge = (struct Edge*)malloc(E * sizeof(struct Edge));
   return graph;
}

void BellmanFord(struct Graph* graph, int source) {
   int V = graph->V;
   int E = graph->E;
   int distance[1000];
   int i, j, u, v, weight;

   for (i = 0; i < V; i++)
      distance[i] = INT_MAX;
   distance[source] = 0;

   for (i = 1; i <= V - 1; i++) {
      for (j = 0; j < E; j++) {
         u = graph->edge[j].source;
         v = graph->edge[j].destination;
         weight = graph->edge[j].weight;
         if (distance[u] != INT_MAX && distance[u] + weight < distance[v])
            distance[v] = distance[u] + weight;
      }
   }
```

HARSH CHITALIYA

```
    for (j = 0; j < E; j++) {
        u = graph->edge[j].source;
        v = graph->edge[j].destination;
        weight = graph->edge[j].weight;
        if (distance[u] != INT_MAX && distance[u] + weight < distance[v]) {
            printf("Graph contains negative weight cycle");
            return;
        }
    }

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

int main() {
    int V, E, i, source;
    struct Graph* graph;
    clrscr();
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &V, &E);

    graph = createGraph(V, E);

    printf("Enter source, destination, and weight for each edge:\n");
    for (i = 0; i < E; i++) {
        scanf("%d %d %d", &graph->edge[i].source, &graph->edge[i].destination, &graph-
>edge[i].weight);
    }

    printf("Enter source vertex: ");
    scanf("%d", &source);

    BellmanFord(graph, source);
    getch();
    return 0;
}
```

HARSH CHITALIYA

Output:

```
Enter number of vertices and edges: 5 9
Enter source, destination, and weight for each edge:
0 1 4
0 2 2
1 2 3
2 1 1
1 3 2
4 3 -5
2 4 5
1 4 3
2 3 4
Enter source vertex: 0
Vertex   Distance from Source
0                0
1                3
2                2
3                1
4                6
```

HARSH CH

16) OBST

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <conio.h>
int **memo;

int sum(int freq[], int i, int j) {
   int s = 0,k;
   for (k = i; k <= j; k++)
      s += freq[k];
   return s;
}

int OBST(int freq[], int i, int j) {
   int min,fsum,r;
   if (j < i)                    // Base case
      return 0;
   if (j == i)                    // One element
      return freq[i];

   if (memo[i][j] != INT_MAX)
      return memo[i][j];

   fsum = sum(freq, i, j);
   min = INT_MAX;

   for ( r = i; r <= j; r++) {
      int cost = OBST(freq, i, r - 1) + OBST(freq, r + 1, j);
      if (cost < min)
         min = cost;
   }

   memo[i][j] = min + fsum;
   return memo[i][j];
}

int main() {
   int n, *keys, *freq, i, j,cost;
   clrscr();
   printf("Enter the number of keys: ");
   scanf("%d", &n);
```

HARSH CHITALIYA

```
    keys = (int*)malloc(n * sizeof(int));
    freq = (int*)malloc(n * sizeof(int));
    memo = (int**)malloc(n * sizeof(int*));

    for (i = 0; i < n; i++) {
        memo[i] = (int*)malloc(n * sizeof(int));
        for (j = 0; j < n; j++) {
            memo[i][j] = INT_MAX;              // Initialize memoization matrix
        }
    }

    printf("Enter the keys: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &keys[i]);              // Read keys
    }

    printf("Enter the frequencies: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &freq[i]);          // Read frequencies
    }

    cost = OBST(freq, 0, n - 1);
    printf("Cost of Optimal BST is %d\n", cost);

    // Clean up memory
    for (i = 0; i < n; i++) {
        free(memo[i]);
    }
    free(memo);
    free(keys);
    free(freq);
    getch();
    return 0;
}
```

Output:

```
Enter the number of keys: 4
Enter the keys: 10 20 30 40
Enter the frequencies: 2 4 6 3
Cost of Optimal BST is 26
```

HARSH CHITALIYA

17) Hamiltonion cycle using backtracking

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAX_V 10

int V;                                  // Global variable for number of vertices
int graph[MAX_V][MAX_V];        // Adjacency matrix with a fixed size

void printSolution(int path[]) {
   int i ;
   printf("Hamiltonian Cycle: \n");
   for ( i = 0; i < V; i++)
      printf("%d ", path[i]);
   printf("%d\n", path[0]);                    // Show the full cycle back to the starting node
}

int isSafe(int v, int path[], int pos) {
   int i;
   if (graph[path[pos - 1]][v] == 0)
      return 0;                     // Check if the current vertex is connected to the previous vertex

   for ( i = 0; i < pos; i++)
      if (path[i] == v)
         return 0;                     // Check if the vertex has already been included

   return 1;
}

int hamCycleUtil(int path[], int pos) {
   int count,v;
   if (pos == V) {
      if (graph[path[pos - 1]][path[0]] == 1) {
         printSolution(path);                          // Print the found Hamiltonian cycle
         return 1;                                      // Return 1 to indicate a cycle was found
      }
      return 0;
   }

    count = 0;                                 // Count of Hamiltonian cycles found
   for ( v = 1; v < V; v++) {
      if (isSafe(v, path, pos)) {
```

HARSH CHITALIYA

```
            path[pos] = v;
            count += hamCycleUtil(path, pos + 1);          // Continue to construct the path

            path[pos] = -1;          // Backtrack
        }
    }
    return count;
}

void hamCycle() {
    int i;
    int cycleCount;
    int path[MAX_V];
    for (i = 0; i < V; i++)
        path[i] = -1;

    path[0] = 0; // Start from the first vertex
     cycleCount = hamCycleUtil(path, 1);

    if (cycleCount == 0)
        printf("No Hamiltonian Cycle exists\n");
    else
        printf("Total Hamiltonian Cycles found: %d\n", cycleCount);
}

int main() {
    int i , j;
    clrscr();
    printf("Enter the number of vertices: ");
    scanf("%d", &V);

    printf("Enter the adjacency matrix:\n");
    for ( i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    hamCycle();
    getch();
    return 0;
}
```

HARSH CHITALIYA

Output:

```
Enter the number of vertices: 5
Enter the adjacency matrix:
0 1 1 0 1
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
1 1 0 1 0
Hamiltonian Cycle:
0 1 2 3 4 0
Hamiltonian Cycle:
0 1 4 3 2 0
Hamiltonian Cycle:
0 2 1 3 4 0
Hamiltonian Cycle:
0 2 3 1 4 0
Hamiltonian Cycle:
0 2 3 4 1 0
Hamiltonian Cycle:
0 4 1 3 2 0
Hamiltonian Cycle:
0 4 3 1 2 0
Hamiltonian Cycle:
0 4 3 2 1 0
Total Hamiltonian Cycles found: 8
```

HARSH CHITALIYA

18) Graph coloring using backtracking

Code:

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

#define MAX_VERTICES 100

int adjacency[MAX_VERTICES][MAX_VERTICES];      // Adjacency matrix of the graph
int colors[MAX_VERTICES];                       // Array to store colors of vertices
int numVertices, numColors;                     // Number of vertices and colors

/* Function prototypes */
int promising_colouring(int v);
void colouring(int v);

/* Function to check if coloring of vertex 'v' is valid */
int promising_colouring(int v) {
   int i;
   for (i = 0; i < v; i++) {
      if (adjacency[v][i] && colors[i] == colors[v]) {
         return 0; // If adjacent vertices have same color, return false
      }
   }
   return 1;
}

/* Recursive function to color vertices of the graph */
void colouring(int v) {
   int i, c;
   if (v == numVertices) {                      // If all vertices are colored
      // Print the coloring
      printf("Vertex Colors: ");
      for (i = 0; i < numVertices; i++) {
         printf("%d ", colors[i]);
      }
      printf("\n");
      return;
   }

   // Try assigning colors to vertex 'v'
   for (c = 1; c <= numColors; c++) {
      colors[v] = c; // Assign color 'c' to vertex 'v'
      if (promising_colouring(v)) { // If coloring is promising
```

HARSH CHITALIYA

```
            colouring(v + 1);              // Recur for next vertex
        }
        colors[v] = 0;                     // Backtrack: reset color of vertex 'v'
    }
}

int main() {
    int i, j;
    clrscr();
    // Input the number of vertices and colors
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of colors: ");
    scanf("%d", &numColors);

    // Input the adjacency matrix of the graph
    printf("Enter the adjacency matrix (%d x %d):\n", numVertices, numVertices);
    for (i = 0; i < numVertices; i++) {
        for (j = 0; j < numVertices; j++) {
            scanf("%d", &adjacency[i][j]);
        }
    }

    // Initialize colors of all vertices to 0 (unassigned)
    for (i = 0; i < numVertices; i++) {
        colors[i] = 0;
    }

    // Start coloring from vertex 0
    colouring(0);
    getch();
    return 0;
}
```

HARSH CHITALIYA

Output:

```
Enter the number of colors: 3
Enter the adjacency matrix (4 x 4):
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
Vertex Colors: 1 2 1 2
Vertex Colors: 1 2 1 3
Vertex Colors: 1 2 3 2
Vertex Colors: 1 3 1 2
Vertex Colors: 1 3 1 3
Vertex Colors: 1 3 2 3
Vertex Colors: 2 1 2 1
Vertex Colors: 2 1 2 3
Vertex Colors: 2 1 3 1
Vertex Colors: 2 3 1 3
Vertex Colors: 2 3 2 1
Vertex Colors: 2 3 2 3
Vertex Colors: 3 1 2 1
Vertex Colors: 3 1 3 1
Vertex Colors: 3 1 3 2
Vertex Colors: 3 2 1 2
Vertex Colors: 3 2 3 1
Vertex Colors: 3 2 3 2
```

HARSH KALIYA

19) . N queen using backtracking

Code:

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

#define MAX_N 20

int board[MAX_N][MAX_N];
int N;

int isSafe(int row, int col) {
   int i, j;
   for (i = 0; i < col; i++)
      if (board[row][i])
         return 0;
   for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
      if (board[i][j])
         return 0;
   for (i = row, j = col; i < N && j >= 0; i++, j--)
      if (board[i][j])
         return 0;
   return 1;
}

void printSolution() {
   int i, j;
   for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++)
         printf("%d ", board[i][j]);
      printf("\n");
   }
}

int solveNQUtil(int col) {
   int res = 0, i;
   if (col >= N) {
      printSolution();
      printf("\n");
      return 1;  // Return 1 to count this solution
   }

   for (i = 0; i < N; i++) {
```

HARSH CHITALIYA

```
        if (isSafe(i, col)) {
            board[i][col] = 1;
            res += solveNQUtil(col + 1);            // Sum up all solutions
            board[i][col] = 0;
        }
    }
    return res;
}

void solveNQ() {
    int totalSolutions = solveNQUtil(0);            // Receives the total number of solutions
    if (totalSolutions == 0) {
        printf("No solution exists\n");
    } else {
        printf("Total number of solutions: %d\n", totalSolutions);
    }
}

int main() {
    clrscr();
    printf("Enter the number of queens (Max %d): ", MAX_N);
    scanf("%d", &N);
    if (N > MAX_N) {
        printf("Number of queens is too large!\n");
        getch();
        return 0;
    }
    solveNQ();
    getch();
    return 0;
}
```

Output:

```
Enter the number of queens (Max 20): 4
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0

Total number of solutions: 2
```

ARSH CHITALIYA

20) Rabin karp

Code:

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

#define d 256        // A prime number

// Function to search for a pattern in a given text using Rabin-Karp algorithm
void rabinKarp(char pattern[], char text[], int q) {
   int M = strlen(pattern);
   int N = strlen(text);
   int i, j;
   int p = 0;  // hash value for pattern
   int t = 0;  // hash value for text
   int h = 1;

   // Calculate hash value of pattern and the first window of text
   for (i = 0; i < M - 1; i++)
      h = (h * d) % q;

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

   // Slide the pattern over text one by one
   for (i = 0; i <= N - M; i++) {
      // Check the hash values of current window of text and pattern
      if (p == t) {
         // Check for characters one by one
         for (j = 0; j < M; j++) {
            if (text[i + j] != pattern[j])
               break;
         }
         if (j == M)
            printf("Pattern found at index %d\n", i);
      }

      // Calculate hash value for next window of text: Remove leading digit, add trailing
digit
      if (i < N - M) {
         t = (d * (t - text[i] * h) + text[i + M]) % q;
```

HARSH CHITALIYA

```
            // Make sure t is positive
            if (t < 0)
                t = (t + q);
        }
    }
}

int main() {
    char text[1024];            // Increase buffer size as needed
    char pattern[256];
    int q = 101;  // A prime number

    printf("Enter the text: ");
    fgets(text, sizeof(text), stdin);              // Read the full line of text
    text[strcspn(text, "\n")] = 0;                 // Remove newline character if present

    printf("Enter the pattern to search: ");
    fgets(pattern, sizeof(pattern), stdin);              // Read the full line of pattern
    pattern[strcspn(pattern, "\n")] = 0;                 // Remove newline character if present

    rabinKarp(pattern, text, q);
    return 0;
}
```

Output:

```
Enter the text: CCACCAACDAB
Enter the pattern to search: DAB
Pattern found at index 8
```

HARSH CHITALIYA

21) Naive string matching

Code :

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
void naiveStringMatch(char *text, char *pattern) {
   int textLen = strlen(text);
   int patternLen = strlen(pattern);
   int i, j;

   for (i = 0; i <= textLen - patternLen; i++) {
      j = 0;
      while (j < patternLen && text[i + j] == pattern[j]) {
         j++;
      }
      if (j == patternLen) {
         printf("Pattern found at index %d\n", i);
      }
   }
}

int main() {
   char text[256];
   char pattern[256];
   clrscr();
   printf("Enter the text: ");
   gets(text);
   printf("Enter the pattern: ");
   gets(pattern);

   printf("Text: %s\n", text);
   printf("Pattern: %s\n", pattern);

   printf("Pattern found at following indices:\n");
   naiveStringMatch(text, pattern);
   getch();
   return 0;
}
```

HARSH CHITALIYA

Output:

```
Enter the text: AABAACAADAABAAABAA
Enter the pattern: AABA
Text: AABAACAADAABAAABAA
Pattern: AABA
Pattern found at following indices:
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
```

HARSH CHITALIYA

22) KMP aalgorithm

Code:

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

void KMP_prefix(char* P, int m, int* pi) {
   int i = 1, j = 0;
   pi[0] = -1;                    // Start with a base value for the prefix table

   while (i < m) {
      while (j >= 0 && P[i] != P[j]) {                    // If characters do not match
         j = pi[j];                                          // Fallback in the prefix table
      }
      i++;
      j++;
      pi[i] = j;  // Set the length of the current longest prefix which is also suffix
   }
}

void KMP_match(char* P, char* T, int m, int n, int* pi) {
   int i = 0, j = 0;

   while (i < n) {
      while (j >= 0 && T[i] != P[j]) {                  // If there's a mismatch
         j = pi[j];                                       // Fall back in the prefix table
      }
      i++;
      j++;

      if (j == m) {                                  // If a full match of the pattern is found
         printf("Pattern found at index %d\n", i - j);
         j = pi[j];                                     // Continue to look for more matches
      }
   }

   if (j == m) {
      printf("Pattern found at index %d\n", i - j);
   } else {
      printf("Pattern not found in text/Pattern Ended\n");
   }
}
```

HARSH CHITALIYA

```
int main() {
    char text[1024];  // Text string
    char pattern[256];  // Pattern string
    int m, n;
    int pi[101];
    clrscr();
    printf("Enter the text: ");
    gets(text);  // Use gets to read string input
    printf("Enter the pattern: ");
    gets(pattern);  // Use gets to read string input

    m = strlen(pattern);  // Length of pattern
    n = strlen(text);  // Length of text

    KMP_prefix(pattern, m, pi);  // Compute prefix table
    KMP_match(pattern, text, m, n, pi);  // Perform KMP matching

    getch();
    return 0;
}
```

Output:

```
Enter the text: AABAACAADAABAAABAA
Enter the pattern: AABA
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
Pattern not found in text/Pattern Ended
```

HARSHIT ALIYA