Experiment 1.A – Insertion Sort

Code:

#include <stdio.h>

void insertion\_sort(int a[], int n){

int i,k,key,count=0;

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

key = a[k];

for(i=k-1; i>=0 && count++ && key<a[i]; --i){

a[i+1] = a[i];

}

a[i+1] = key;

printf("\nPass%d\n",k);

for(int j=0; j<=n-1; j++){

printf("%d ", a[j]);

}

}

printf("\nTotal Comparisons: %d",count);

printf("\nArray After Sorting: \n");

for(i=0; i<=n-1; i++){

printf("%d ", a[i]);

}

}

int main() {

int a[100],n;

printf("Tanmay Narkar C23@110\n\n");

printf("\nEnter Array Size: ");

scanf("%d",&n);

printf("\nEnter Elements of Array: ");

for(int i = 0; i<=n-1; i++){

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

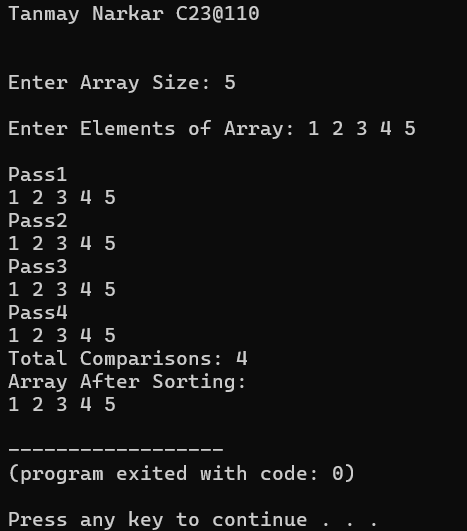
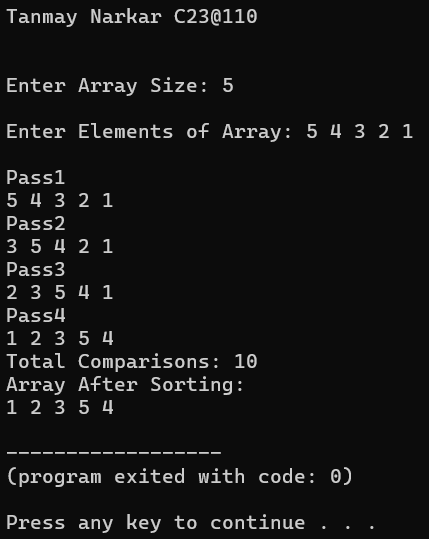
}

insertion\_sort(a,n);

return 0;

}

Output:



Experiment 1.B – Selection Sort

Code:

#include <stdio.h>

void selection\_sort(int a[], int n){

int i,j,min,pos,count=0;

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

min = a[i];

pos = i;

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

if(min>a[j]){

min = a[j];

pos = j; //Track Index

}

}

//swap Smallest and First

a[pos] = a[i];

a[i] = min;

printf("\nPass%d\n",i); //Array after every Pass

for(int k=0; k<=n-1; k++)

printf("%d ", a[k]);

}

}

printf("\nTotal Comparisons : %d",count)

printf("\nArray after Sorting: \n");

for(i=0; i<=n-1; i++){

printf("%d ", a[i]);

}

}

int main() {

int a[100],n;

printf("\nTanmay Narkar C23@110\n\n");

printf("\nEnter Array Size: ");

scanf("%d",&n);

printf("\nEnter Elements of Array: ");

for(int i = 0; i<=n-1; i++){

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

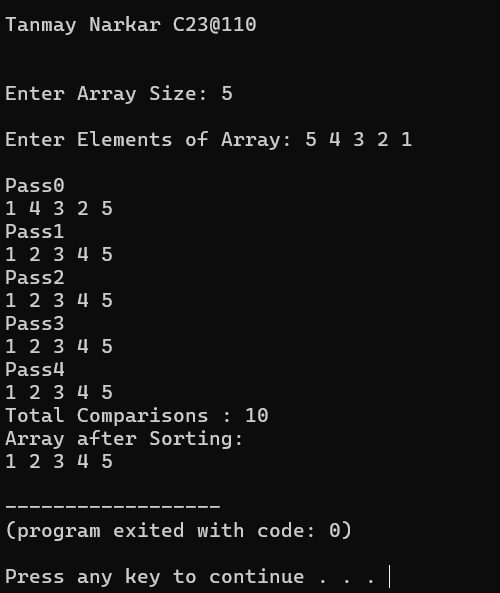
}

selection\_sort(a,n);

return 0;

}

Output:



Experiment 5 – Longest Common Subsequence

Code:

#include<stdio.h>

#include<string.h>

#define MAX 100

int max(int a, int b){

return (a>b) ? a : b;

}

void longestCommonSubs(char str1[], char str2[]){

int m,n;

m = strlen(str1);

n = strlen(str2);

int LCS[m+1][n+1]; //Make Matrix of 1 Extra R&C

for(int i=0; i<=m; i++){

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

if(i== 0 || j== 0)

LCS[i][j] = 0;

else if(str1[i-1] == str2[j-1])

LCS[i][j] = LCS[i-1][j-1] + 1;

else

LCS[i][j] = max(LCS[i-1][j],LCS[i][j-1]);

}

}

int i=m,j=n;

int index = LCS[m][n];

char lcs\_string[index+1];

lcs\_string[index] = '\0';

while(i>0 && j>0){

if(str1[i-1] == str2[j-1]){

lcs\_string[index-1] = str1[i-1];

i--;

j--;

index--;

}

else if(LCS[i-1] > LCS[j-1])

i--;

else

j--;

}

printf("\nLength of LCS: %d",LCS[m][n]);

printf("\nLongest Common Subsequence: %s",lcs\_string);

}

int main(){

char str1[MAX], str2[MAX];

printf("Tanmay Narkar C23@110\n\n");

printf("\nEnter 1st String : ");

scanf("%s",str1);

printf("\nEnter 2nd String : ");

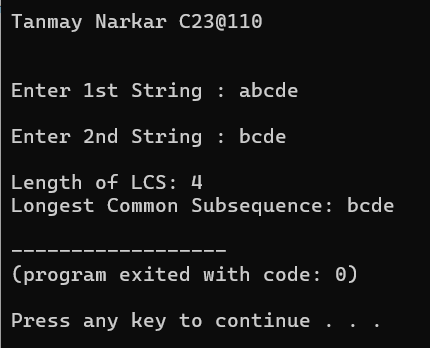
scanf("%s",str2);

longestCommonSubs(str1,str2);

return 0;

}

Output:



Experiment 2.A – Merge Sort

Code:

#include<stdio.h>

#define MAX 10

void merge(int arr[], int low, int mid, int high){

    int leftMax = mid - low + 1;

    int rightMax = high - mid;

    int leftArr[leftMax];

    int rightArr[rightMax];

    for(int i=0;i<leftMax;i++)

        leftArr[i] = arr[low+i];

    for(int j=0;j<rightMax;j++)

        rightArr[j] = arr[mid+1+j];

    int i = 0; // Left Array ka 1st index

    int j = 0; // Right Array ka 1st Index

    int k = low;    // Merged Array ka 1st Index

    while(i < leftMax && j < rightMax){

        if(leftArr[i] <= rightArr[j]){

            arr[k] = leftArr[i];

            i++;

        }

        else{

            arr[k] = rightArr[j];

            j++;

        }

        k++;

    }

    while(i < leftMax){

        arr[k] = leftArr[i];

        i++;

        k++;

    }

    while(i < rightMax){

        arr[k] = rightArr[j];

        j++;

        k++;

    }

}

void merge\_sort(int arr[], int low, int high){

    if(low<high){

        int mid = (low+high)/2;

        merge\_sort(arr,low,mid);

        merge\_sort(arr,mid+1,high);

        merge(arr,low,mid,high);

    }

}

int main(){

    int arr[MAX],n;

    printf("\nEnter Size of Array : ");

    scanf("%d",&n);

    printf("\nEnter Array Elements : ");

    for(int i=0;i<n;i++){

        scanf("%d",&arr[i]);

    }

    printf("Original Array: ");

    for(int i=0;i<n;i++){

        printf(" %d",arr[i]);

    }

    merge\_sort(arr, 0, n-1); // Passing first and last value

    printf("\nSorted Array: ");

    for(int i=0;i<n;i++){

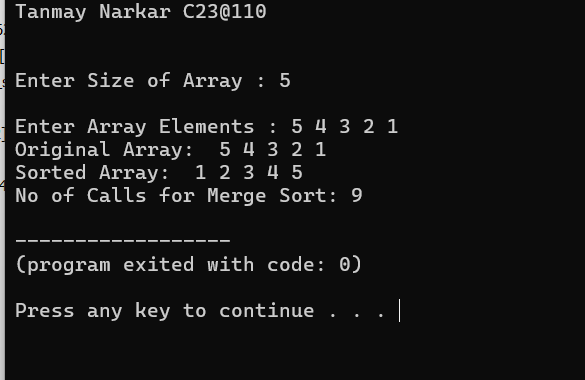
        printf(" %d",arr[i]);

    }

    return 0;

}

Output:



Experiment 2.B – Quick Sort

Code:

#include<stdio.h>

#define MAX 10

void swap(int\* a, int\* b){

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int partition(int arr[], int low, int high){

    int pivot = arr[high];

    int i = low - 1;

    for(int j=low; j<=high-1; j++){

        if(arr[j] < pivot){

            i++;

            swap(&arr[i],&arr[j]);

        }

    }

    swap(&arr[i+1],&arr[high]);

    return i+1;

}

void quick\_sort(int arr[], int low, int high){

        if(low < high){

            int q = partition(arr,low,high);

            quick\_sort(arr,low,q-1);

            quick\_sort(arr,q+1,high);

        }

}

int main(){

    int arr[MAX],n;

    printf("\nEnter Size of Array : ");

    scanf("%d",&n);

    printf("\nEnter Array Elements : ");

    for(int i=0;i<n;i++){

        scanf("%d",&arr[i]);

    }

    printf("Original Array: ");

    for(int i=0;i<n;i++){

        printf(" %d",arr[i]);

    }

    quick\_sort(arr, 0, n-1); // Passing first and last value

    printf("\nSorted Array: ");

    for(int i=0;i<n;i++){

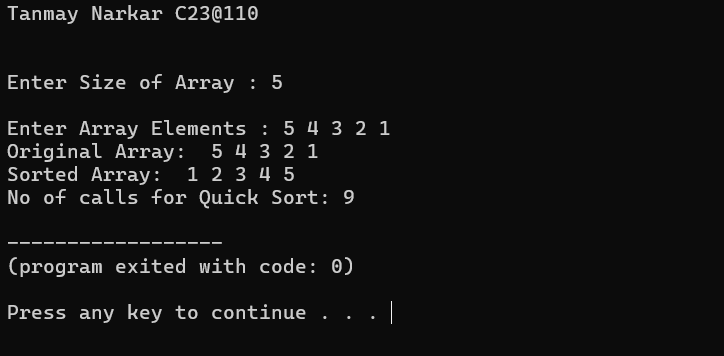
        printf(" %d",arr[i]);

    }

    return 0;

}

Output:



Experiment 7 – All Pair Shortest Path Algorithm (Floyd Warshall algorithm)

Code:

#include <stdio.h>

#define MAX 50

#define INF 99999

int distance[MAX][MAX];

int pred[MAX][MAX];

void initializeMat(int V){

for(int i=0; i<V; i++){

for(int j=0; j<V; j++){

if(i==j){

distance[i][j] = 0;

pred[i][j] = -1;

}

else if(distance[i][j] == INF)

pred[i][j] = -1;

else

pred[i][j] = i;

}

}

}

void printDistMat(int V){

printf("\nDistance Matrix: \n\n");

for(int i=0;i<V;i++){

for(int j=0;j<V; j++){

if(distance[i][j] == INF)

printf("%4s","INF");

else

printf("%4d",distance[i][j]);

}

printf("\n");

}

}

void printPredMat(int V){

printf("\nPredecessor Matrix: \n\n");

for(int i=0;i<V;i++){

for(int j=0;j<V; j++){

if(pred[i][j] == -1)

printf("%4s","NIL");

else

printf("%4d",pred[i][j] + 1);

}

printf("\n");

}

}

void floydWarshall(int V){

for(int k=0; k<V; k++){

for(int i=0; i<V; i++){

for(int j=0; j<V; j++){

if(distance[i][k]!=INF && distance[k][j]!=INF && (distance[i][k]+distance[k][j] < distance[i][j])){

distance[i][j] = distance[i][k]+distance[k][j];

pred[i][j] = pred[k][j];

}

}

}

}

}

void printPath(int i, int j){

if(i == j){

printf("%d",i+1);

return;

}

if(pred[i][j] == -1){

printf("No Path");

return;

}

printPath(i,pred[i][j]);

printf("-> %d",j+1);

}

int main(){

int V;

printf("Tanmay Narkar C23@110\n\n");

printf("\nEnter No of Vertices : ");

scanf("%d",&V);

printf("\nEnter Adjacency Matrix: "); //If no edge from vertex, enter -1

for(int i=0; i<V; i++){

for(int j=0; j<V; j++){

int weight;

scanf("%d",&weight);

if(weight==-1 && i!=j)

distance[i][j] = INF;

else

distance[i][j] = weight;

}

}

initializeMat(V);

floydWarshall(V);

printDistMat(V);

printPredMat(V);

printf("\nShortest Path for each Vertex : \n");

for(int i=0; i<V; i++){

for(int j=0; j<V; j++){

if(i!=j){

printf("Shortest path from %d to %d: ",i+1,j+1);

printPath(i,j);

printf("\n");

}

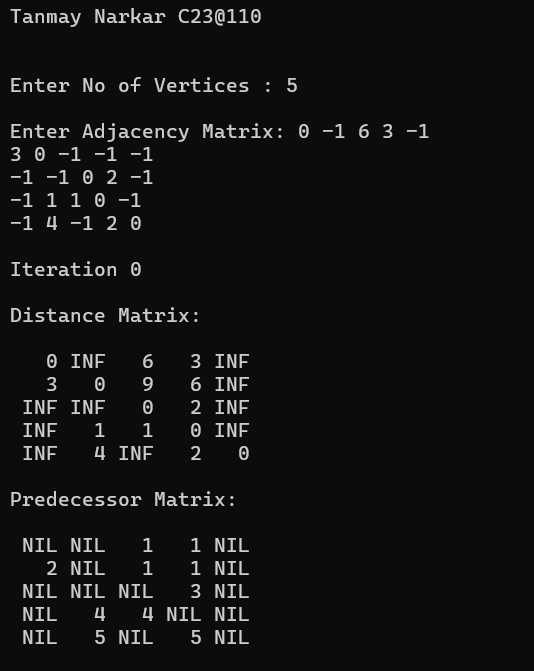
}

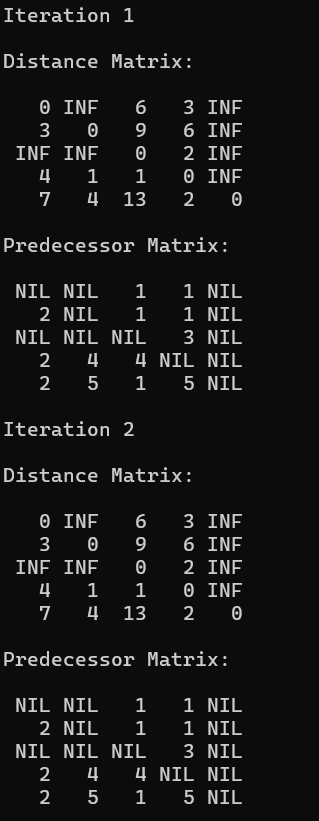
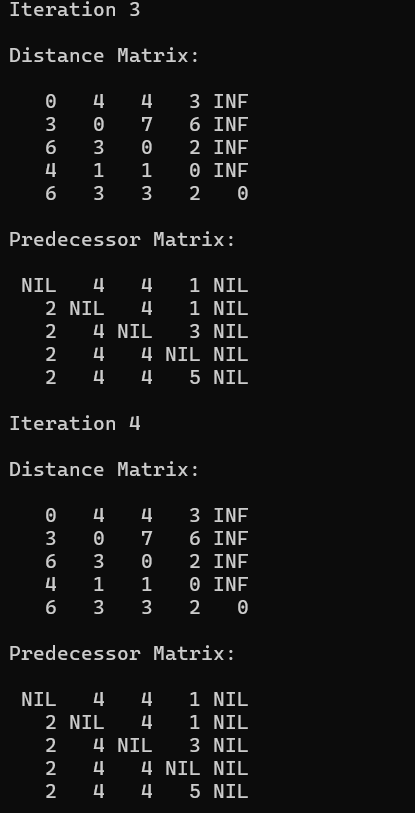
}

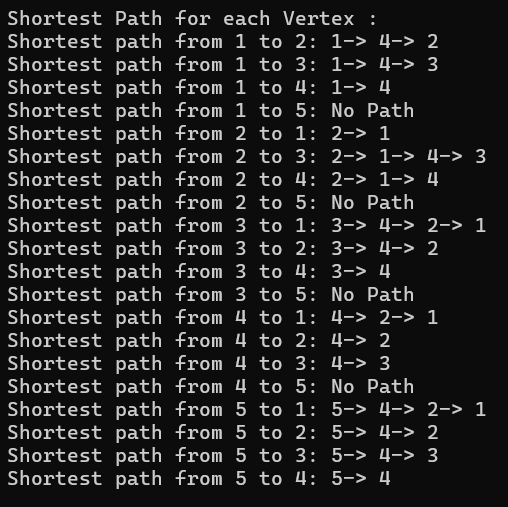
return 0;

}

Output:





Experiment 6 – 0/1 Knapsack Problem

Code:

#include <stdio.h>

#define MAX\_ITEMS 100

// Function to calculate the maximum value using Dynamic Programming

int knapsack(int capacity, int weights[], int values[], int n, int selected[]) {

int dp[n + 1][capacity + 1];

// Initialize the dp table

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

for (int w = 0; w <= capacity; w++) {

if (i == 0 || w == 0)

dp[i][w] = 0;

else if (weights[i - 1] <= w)

dp[i][w] = (values[i - 1] + dp[i - 1][w - weights[i - 1]] > dp[i - 1][w]) ?

values[i - 1] + dp[i - 1][w - weights[i - 1]] : dp[i - 1][w];

else

dp[i][w] = dp[i - 1][w];

}

}

// Trace back to find which items are included

int w = capacity;

for (int i = n; i > 0; i--) {

if (dp[i][w] != dp[i - 1][w]) {

selected[i - 1] = 1; // Item i-1 is included

w -= weights[i - 1];

} else {

selected[i - 1] = 0; // Item i-1 is not included

}

}

printf("\nMatrix:\n\n");

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

for (int w = 0; w <= capacity; w++) {

printf("%d ",dp[i][w]);

}

printf("\n");

}

// Return the maximum value that can be obtained with the given capacity

return dp[n][capacity];

}

int main() {

int n, capacity;

printf("Tanmay Narkar C23@110\n\n");

printf("Enter the number of items: ");

scanf("%d", &n);

printf("Enter the capacity of the knapsack: ");

scanf("%d", &capacity);

int values[n], weights[n];

int selected[n];

printf("Enter the values of the items: ");

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

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

}

printf("Enter the weights of the items: ");

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

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

}

int max\_value = knapsack(capacity, weights, values, n, selected);

printf("\nThe maximum value that can be obtained is: %d\n", max\_value);

printf("\nSelected items (Objects) -> (weights): \n");

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

if (selected[i] == 1) {

printf("%d -> %d \n", i+1,weights[i]);

}

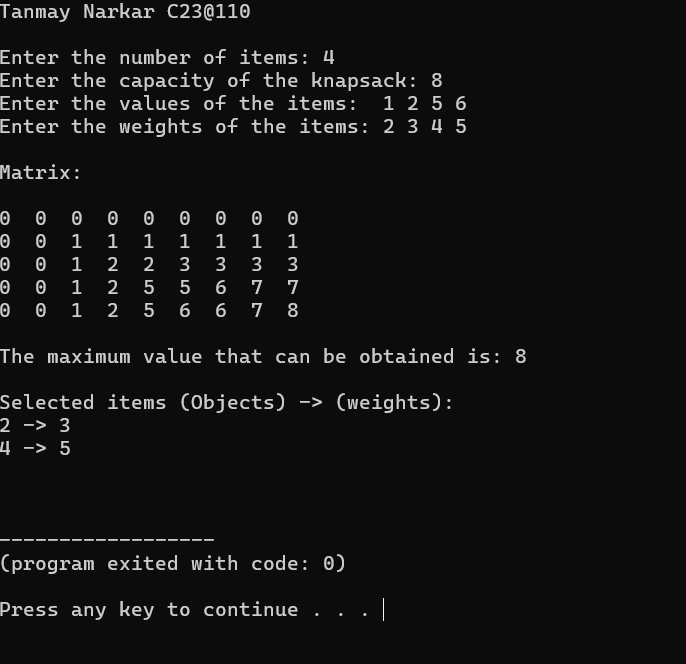
}

printf("\n");

return 0;

}

Output:



Experiment 3 – Minimum Spanning Tree – Prim’s Algorithm

Code:

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define MAX 100

// Minimum Value Find Function

int minKey(int key[], bool visited[], int V) {

int min = INT\_MAX, minIndex;

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

if (!visited[v] && key[v] < min) {

min = key[v];

minIndex = v;

}

return minIndex;

}

void printMST(int parent[], int graph[MAX][MAX], int V) {

printf("\nMinimum Spanning Tree:\n");

printf("Edge \tWeight\n");

for (int i = 1; i < V; i++)

printf("%d - %d \t%d\n", parent[i] + 1, i + 1, graph[i][parent[i]]);

}

void primMST(int graph[MAX][MAX], int V) {

int parent[V];

int key[V];

bool visited[V];

for (int i = 0; i < V; i++) {

key[i] = INT\_MAX;

visited[i] = false;

}

key[0] = 0;

parent[0] = -1;

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

int u = minKey(key, visited, V);

visited[u] = true;

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

if (graph[u][v] && !visited[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

printMST(parent, graph, V);

}

int main() {

int V;

int graph[MAX][MAX];

printf("Tanmay Narkar C23@110\n\n");

printf("Enter number of vertices: ");

scanf("%d", &V);

printf("Enter the adjacency matrix (enter 0 if no edge):\n");

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

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

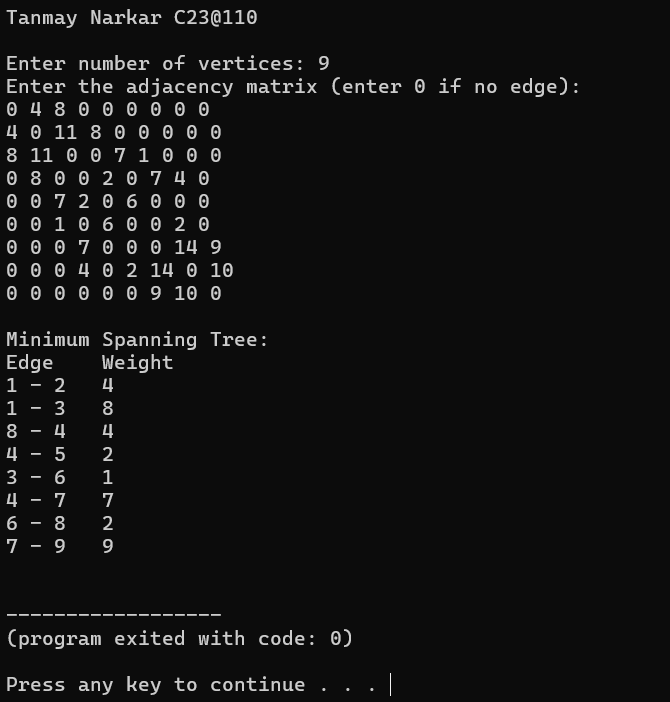
scanf("%d", &graph[i][j]);

primMST(graph, V);

return 0;

}

Output:



Experiment 4 – Greedy Method Approach - Dijkstra algorithm

Code:

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define MAX 100

// Min Val Fn

int minDistance(int dist[], bool visited[], int V) {

int min = INT\_MAX, minIndex;

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

if (!visited[v] && dist[v] < min) {

min = dist[v];

minIndex = v;

}

return minIndex;

}

void printSolution(int dist[], int pred[], int V, int src) {

printf("\nVertex\tShortest Distance\tPredecessor\n");

for (int i = 0; i < V; i++) {

if (i == src)

printf("%d\t%d\t\t\t-\n", i + 1, dist[i]); // Source node has no predecessor

else if (dist[i] == INT\_MAX)

printf("%d\tINF\t\t\tNIL\n", i + 1); // Unreachable node

else

printf("%d\t%d\t\t\t%d\n", i + 1, dist[i], pred[i] + 1);

}

}

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

int dist[V];

bool visited[V];

int pred[V];

for (int i = 0; i < V; i++) {

dist[i] = INT\_MAX;

visited[i] = false;

pred[i] = -1;

}

dist[src] = 0;

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

int u = minDistance(dist, visited, V);

visited[u] = true;

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

if (!visited[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];

pred[v] = u;

}

}

printSolution(dist, pred, V, src);

}

int main() {

int V, src;

int graph[MAX][MAX];

printf("Tanmay Narkar C23@110\n\n");

printf("Enter number of vertices: ");

scanf("%d", &V);

printf("Enter the adjacency matrix (enter 0 if no edge):\n");

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

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

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

printf("Enter the source vertex (1-%d): ", V);

scanf("%d", &src);

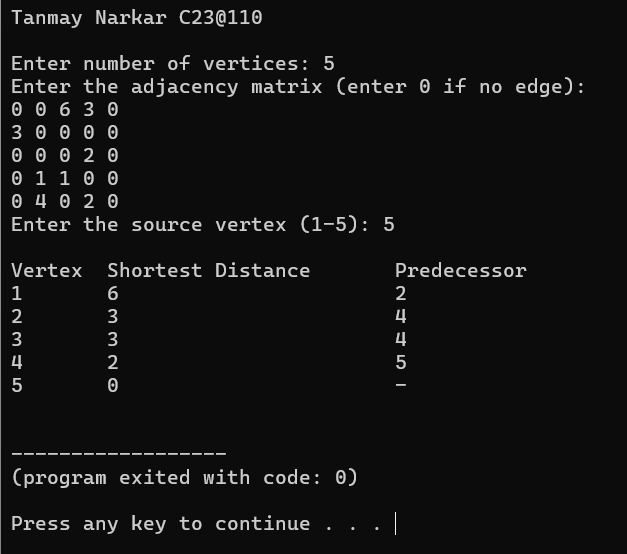
src--; // Adjust for 0-based index

dijkstra(graph, V, src);

return 0;

}

Output:



Experiment 8 – Backtracking Approach – NQueens Problem

Code:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int x[10]; // x[i] will store the column position of the queen in row i

int c = 0;

int place(int k, int i) {

int j;

for (j = 1; j <= k - 1; j++) {

if (x[j] == i || abs(x[j] - i) == abs(j - k)) //same col or same diagonal

return 0;

}

return 1;

}

void nqueen(int k, int n) {

int i, row, col;

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

if (place(k, i)) {

x[k] = i;

if (k == n) {

c++;

printf("\nSolution %d:\n", c);

printf("x[] =\t");

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

printf("%d\t", x[row]);

printf("\n\n");

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

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

if (x[row] == col)

printf("Q\t");

else

printf("--\t");

}

printf("\n");

}

} else {

nqueen(k + 1, n);

}

}

}

}

int main() {

int n;

printf("Tanmay Narkar C23@110\n\n");

printf("\nEnter number of queens: ");

scanf("%d", &n);

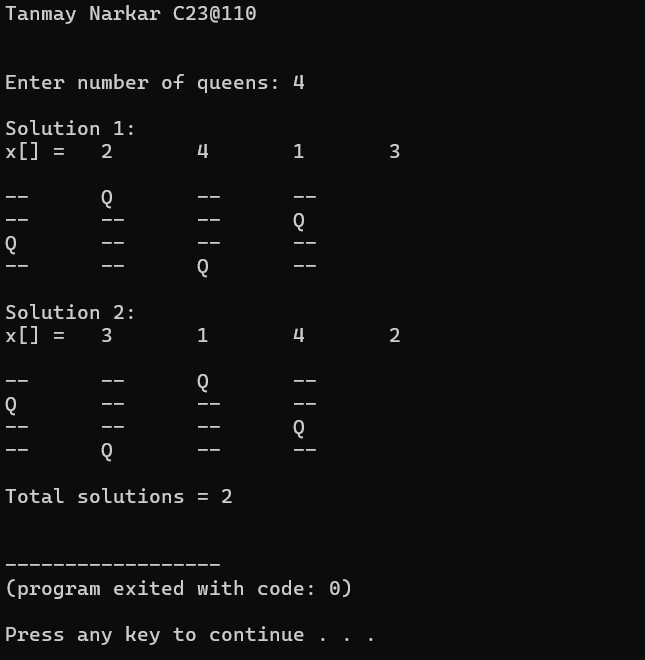
nqueen(1, n);

printf("\nTotal solutions = %d\n", c);

return 0;

}

Output:



Experiment 9 – Backtracking Approach – Sum of Subsets Problem

Code:

#include <stdio.h>

int w[10], x[10], n, W;

void sumOfSubsets(int k, int currentSum, int r) {

int i;

x[k] = 1; // Include w[k]

// Check if adding w[k] gives the target sum

if (currentSum + w[k] == W) {

printf("\nSubset: ");

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

if (x[i] == 1)

printf("%d ", w[i]);

}

}

// If it’s promising to go further

else if (currentSum + w[k] + w[k+1] <= W) {

sumOfSubsets(k + 1, currentSum + w[k], r - w[k]);

}

// Backtrack: don’t include w[k]

if ((currentSum + r - w[k]) >= W && (currentSum + w[k+1]) <= W) {

x[k] = 0;

sumOfSubsets(k + 1, currentSum, r - w[k]);

}

}

int main() {

int i, total = 0;

printf("Tanmay Narkar C23@110\n\n");

printf("\nEnter number of elements: ");

scanf("%d", &n);

printf("\nEnter elements in increasing order: ");

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

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

total += w[i];

}

printf("\nEnter target sum: ");

scanf("%d", &W);

if (total < W) {

printf("\nNo subset possible.");

} else {

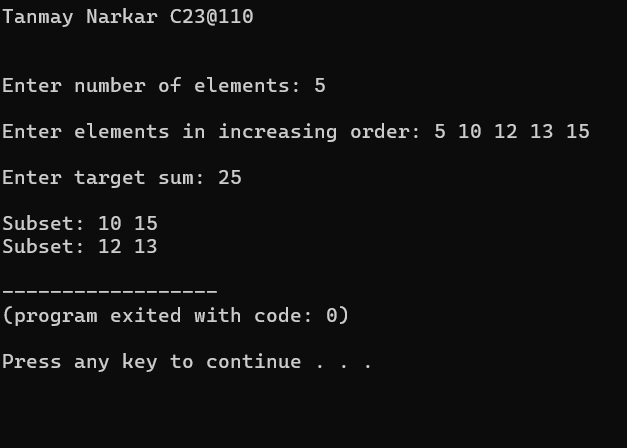
sumOfSubsets(0, 0, total);

}

return 0;

}

Output:



Experiment 10 – String Matching Algorithm – Knutt – Morris – Pratt Algorithm

Code:

#include <stdio.h>

#include <string.h>

void computeLPS(char\* pattern, int m, int\* lps) {

int len = 0;

lps[0] = 0;

int i = 1;

while (i < m) {

if (pattern[i] == pattern[len]) {

len++;

lps[i] = len;

i++;

} else {

if (len != 0) {

len = lps[len - 1];

} else {

lps[i] = 0;

i++;

}

}

}

}

void KMPsearch(char\* text, char\* pattern) {

int n = strlen(text);

int m = strlen(pattern);

int lps[m];

computeLPS(pattern, m, lps);

int i = 0, j = 0;

while (i < n) {

if (pattern[j] == text[i]) {

i++;

j++;

}

if (j == m) {

printf("\nPattern found at index %d\n", i - j);

j = lps[j - 1];

} else if (i < n && pattern[j] != text[i]) {

if (j != 0)

j = lps[j - 1];

else

i++;

}

}

}

int main() {

char text[100], pattern[100];

printf("Tanmay Narkar C23@110\n\n");

printf("\nEnter the text: ");

scanf(" %[^\n]", text); // reads until newline, includes spaces

printf("\nEnter the pattern: ");

scanf(" %[^\n]", pattern);

KMPsearch(text, pattern);

return 0;

}

Output:

