

Catalog

LinearSearch.c.....	1
binSearch.c.....	2
02Heapsort.c.....	3
03MergeSort.c.....	5
04selectionsort.c.....	7
05InsertionSort.c.....	8
06QuickSort.c.....	9
07KnapsackGreedy.c.....	11
08TravellingSalesman.c.....	13
09minSpaningTree.c.....	15
10N_Queen.c.....	17
110_1Knapsack.c.....	19
12Dijkstra.c.....	20
130_1Knapsack_DP_Greedy.c.....	22
14Kruskal.c.....	25
15a_Floyd.c.....	28
15b_TravellingSalesman_DP.c.....	30
16HamiltonianCycles.c.....	32

Exp_01 - LInear and Binary Search (Recursive)\LinearSearch.c

```
1 #include <stdio.h>
2
3 int linearSearch(int arr[], int size, int key, int index) {
4     if (index >= size)
5         return -1;
6
7     if (arr[index] == key)
8         return index;
9
10    return linearSearch(arr, size, key, index + 1);
11}
12
13 int main() {
14     int n, key, result;
15
16     printf("Enter number of elements: ");
17     scanf("%d", &n);
18
19     int arr[n];
20     printf("Enter %d elements: ", n);
21     for (int i = 0; i < n; i++) {
22         scanf("%d", &arr[i]);
23     }
24
25     printf("Enter element to search: ");
26     scanf("%d", &key);
27
28     result = linearSearch(arr, n, key, 0);
29
30     if (result != -1)
31         printf("Element found at index %d\n", result);
32     else
33         printf("Element not found\n");
34
35     return 0;
36 }
```

Exp_01 - LInear and Binary Search (Recursive)\binSearch.c

```
1 #include <stdio.h>
2
3 int binarySearch(int arr[], int left, int right, int x)
4 {
5     if (right >= left)
6     {
7         int mid = left + (right - left) / 2;
8
9         if (arr[mid] == x)
10            return mid;
11
12        if (arr[mid] > x)
13            return binarySearch(arr, left, mid - 1, x);
14
15        return binarySearch(arr, mid + 1, right, x);
16    }
17
18    return -1;
19}
20
21 int main()
22 {
23     int arr[] = {2, 3, 4, 10, 40, 50, 70, 80};
24     int n = sizeof(arr) / sizeof(arr[0]);
25     int x;
26
27     printf("Enter element to search: ");
28     scanf("%d", &x);
29
30     int result = binarySearch(arr, 0, n - 1, x);
31
32     if (result == -1)
33         printf("Element is not present in array\n");
34     else
35         printf("Element is present at index %d\n", result);
36
37     return 0;
38 }
```

Exp_02_Heap_Sort\Heapsort.c

```
1 #include <stdio.h>
2
3 void swap(int *a, int *b) {
4     int temp = *a;
5     *a = *b;
6     *b = temp;
7 }
8
9 void heapify(int arr[], int n, int i) {
10     int largest = i;
11     int left = 2 * i + 1;
12     int right = 2 * i + 2;
13
14     if (left < n && arr[left] > arr[largest])
15         largest = left;
16
17     if (right < n && arr[right] > arr[largest])
18         largest = right;
19
20     if (largest != i) {
21         swap(&arr[i], &arr[largest]);
22         heapify(arr, n, largest);
23     }
24 }
25
26 void heapSort(int arr[], int n) {
27     // Build max heap
28     for (int i = n / 2 - 1; i >= 0; i--)
29         heapify(arr, n, i);
30
31     // Extract elements from heap
32     for (int i = n - 1; i > 0; i--) {
33         swap(&arr[0], &arr[i]);
34         heapify(arr, i, 0);
35     }
36 }
37
38 void printArray(int arr[], int n) {
39     for (int i = 0; i < n; i++)
40         printf("%d ", arr[i]);
41     printf("\n");
42 }
43
44 int main() {
45     int n;
46     printf("Enter number of elements: ");
47     scanf("%d", &n);
48
49     int arr[n];
50     printf("Enter elements: ");
51     for (int i = 0; i < n; i++)
```

```
52     scanf("%d", &arr[i]);
53
54     printf("Original array: ");
55     printArray(arr, n);
56
57     heapSort(arr, n);
58
59     printf("Sorted array: ");
60     printArray(arr, n);
61
62     return 0;
63 }
```

Exp_03_Merge_Sort\MergeSort.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void merge(int arr[], int left, int mid, int right) {
5     int n1 = mid - left + 1;
6     int n2 = right - mid;
7
8     int *L = (int*)malloc(n1 * sizeof(int));
9     int *R = (int*)malloc(n2 * sizeof(int));
10
11    for (int i = 0; i < n1; i++)
12        L[i] = arr[left + i];
13    for (int j = 0; j < n2; j++)
14        R[j] = arr[mid + 1 + j];
15
16    int i = 0, j = 0, k = left;
17
18    while (i < n1 && j < n2) {
19        if (L[i] <= R[j]) {
20            arr[k] = L[i];
21            i++;
22        } else {
23            arr[k] = R[j];
24            j++;
25        }
26        k++;
27    }
28
29    while (i < n1) {
30        arr[k] = L[i];
31        i++;
32        k++;
33    }
34
35    while (j < n2) {
36        arr[k] = R[j];
37        j++;
38        k++;
39    }
40
41    free(L);
42    free(R);
43 }
44
45 void mergeSort(int arr[], int left, int right) {
46     if (left < right) {
47         int mid = left + (right - left) / 2;
48
49         mergeSort(arr, left, mid);
50         mergeSort(arr, mid + 1, right);
51         merge(arr, left, mid, right);
```

```
52     }
53 }
54
55 void printArray(int arr[], int size) {
56     for (int i = 0; i < size; i++)
57         printf("%d ", arr[i]);
58     printf("\n");
59 }
60
61 int main() {
62     int n;
63
64     printf("Enter number of elements: ");
65     scanf("%d", &n);
66
67     int *arr = (int*)malloc(n * sizeof(int));
68
69     printf("Enter elements: ");
70     for (int i = 0; i < n; i++)
71         scanf("%d", &arr[i]);
72
73     printf("Original array: ");
74     printArray(arr, n);
75
76     mergeSort(arr, 0, n - 1);
77
78     printf("Sorted array: ");
79     printArray(arr, n);
80
81     free(arr);
82     return 0;
83 }
```

Exp_04_Selection Sort\selectionsort.c

```
1 #include <stdio.h>
2
3 void selectionSort(int arr[], int n) {
4     int i, j, minIdx, temp;
5
6     for (i = 0; i < n - 1; i++) {
7         minIdx = i;
8         for (j = i + 1; j < n; j++) {
9             if (arr[j] < arr[minIdx]) {
10                 minIdx = j;
11             }
12         }
13         if (minIdx != i) {
14             temp = arr[i];
15             arr[i] = arr[minIdx];
16             arr[minIdx] = temp;
17         }
18     }
19 }
20
21 void printArray(int arr[], int n) {
22     for (int i = 0; i < n; i++) {
23         printf("%d ", arr[i]);
24     }
25     printf("\n");
26 }
27
28 int main() {
29     int n;
30
31     printf("Enter number of elements: ");
32     scanf("%d", &n);
33
34     int arr[n];
35     printf("Enter elements: ");
36     for (int i = 0; i < n; i++) {
37         scanf("%d", &arr[i]);
38     }
39
40     printf("Original array: ");
41     printArray(arr, n);
42
43     selectionSort(arr, n);
44
45     printf("Sorted array: ");
46     printArray(arr, n);
47
48     return 0;
49 }
```

Exp_05_Insertion_Sort\InsertionSort.c

```
1 #include <stdio.h>
2
3 void insertionSort(int arr[], int n) {
4     for (int i = 1; i < n; i++) {
5         int key = arr[i];
6         int j = i - 1;
7
8         while (j >= 0 && arr[j] > key) {
9             arr[j + 1] = arr[j];
10            j--;
11        }
12        arr[j + 1] = key;
13    }
14}
15
16 void printArray(int arr[], int n) {
17     for (int i = 0; i < n; i++) {
18         printf("%d ", arr[i]);
19     }
20     printf("\n");
21 }
22
23 int main() {
24     int n;
25
26     printf("Enter number of elements: ");
27     scanf("%d", &n);
28
29     int arr[n];
30     printf("Enter %d elements: ", n);
31     for (int i = 0; i < n; i++) {
32         scanf("%d", &arr[i]);
33     }
34
35     printf("Original array: ");
36     printArray(arr, n);
37
38     insertionSort(arr, n);
39
40     printf("Sorted array: ");
41     printArray(arr, n);
42
43     return 0;
44 }
```

Exp_06_Quick_Sort\QuickSort.c

```
1 #include <stdio.h>
2
3 void swap(int* a, int* b) {
4     int temp = *a;
5     *a = *b;
6     *b = temp;
7 }
8
9 int partition(int arr[], int low, int high) {
10    int pivot = arr[high];
11    int i = low - 1;
12
13    for (int j = low; j < high; j++) {
14        if (arr[j] < pivot) {
15            i++;
16            swap(&arr[i], &arr[j]);
17        }
18    }
19    swap(&arr[i + 1], &arr[high]);
20    return i + 1;
21 }
22
23 void quickSort(int arr[], int low, int high) {
24    if (low < high) {
25        int pi = partition(arr, low, high);
26        quickSort(arr, low, pi - 1);
27        quickSort(arr, pi + 1, high);
28    }
29 }
30
31 void printArray(int arr[], int size) {
32    for (int i = 0; i < size; i++) {
33        printf("%d ", arr[i]);
34    }
35    printf("\n");
36 }
37
38 int main() {
39    int n;
40
41    printf("Enter number of elements: ");
42    scanf("%d", &n);
43
44    int arr[n];
45    printf("Enter %d elements: ", n);
46    for (int i = 0; i < n; i++) {
47        scanf("%d", &arr[i]);
48    }
49
50    printf("Original array: ");
51    printArray(arr, n);
```

```
52  
53     quickSort(arr, 0, n - 1);  
54  
55     printf("Sorted array: ");  
56     printArray(arr, n);  
57  
58     return 0;  
59 }
```

Exp_07\KnapsackGreedy.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct {
5     int weight;
6     int value;
7     double ratio;
8     int index;
9 } Item;
10
11 int compare(const void *a, const void *b) {
12     Item *itemA = (Item *)a;
13     Item *itemB = (Item *)b;
14     if (itemB->ratio > itemA->ratio) return 1;
15     if (itemB->ratio < itemA->ratio) return -1;
16     return 0;
17 }
18
19 void knapsackGreedy(int weights[], int values[], int n, int capacity) {
20     Item items[n];
21
22     // Calculate value-to-weight ratio
23     for (int i = 0; i < n; i++) {
24         items[i].weight = weights[i];
25         items[i].value = values[i];
26         items[i].ratio = (double)values[i] / weights[i];
27         items[i].index = i + 1;
28     }
29
30     // Sort by ratio in descending order
31     qsort(items, n, sizeof(Item), compare);
32
33     double totalValue = 0.0;
34     int remainingCapacity = capacity;
35     double fractions[n];
36
37     for (int i = 0; i < n; i++) {
38         fractions[i] = 0.0;
39     }
40
41     printf("\nItem\tWeight\tValue\tRatio\tFraction\n");
42     printf("-----\n");
43
44     for (int i = 0; i < n; i++) {
45         if (remainingCapacity >= items[i].weight) {
46             // Take full item
47             fractions[items[i].index - 1] = 1.0;
48             totalValue += items[i].value;
49             remainingCapacity -= items[i].weight;
50             printf("%d\t%d\t%d\t%.2f\t%.2f\n", items[i].index, items[i].weight,
51                 items[i].value, items[i].ratio, 1.0);
```

```

52     } else if (remainingCapacity > 0) {
53         // Take fractional item
54         double fraction = (double)remainingCapacity / items[i].weight;
55         fractions[items[i].index - 1] = fraction;
56         totalValue += items[i].value * fraction;
57         printf("%d\t%d\t%d\t%.2f\t%.2f\n", items[i].index, items[i].weight,
58                items[i].value, items[i].ratio, fraction);
59         remainingCapacity = 0;
60         break;
61     }
62 }
63
64 printf("-----\n");
65 printf("Maximum value: %.2f\n", totalValue);
66 }
67
68 int main() {
69     int n, capacity;
70
71     printf("Enter number of items: ");
72     scanf("%d", &n);
73
74     int weights[n], values[n];
75
76     printf("Enter weights: ");
77     for (int i = 0; i < n; i++) {
78         scanf("%d", &weights[i]);
79     }
80
81     printf("Enter values: ");
82     for (int i = 0; i < n; i++) {
83         scanf("%d", &values[i]);
84     }
85
86     printf("Enter knapsack capacity: ");
87     scanf("%d", &capacity);
88
89     knapsackGreedy(weights, values, n, capacity);
90
91     return 0;
92 }
```

Exp_08\TravellingSalesman.c

```
1 #include <stdio.h>
2 #include <limits.h>
3 #include <stdbool.h>
4
5 #define MAX 20
6
7 int n;
8 int graph[MAX][MAX];
9 int dp[MAX][1 << MAX];
10
11 int min(int a, int b) {
12     return (a < b) ? a : b;
13 }
14
15 int tsp(int pos, int mask) {
16     if (mask == (1 << n) - 1) {
17         return graph[pos][0];
18     }
19
20     if (dp[pos][mask] != -1) {
21         return dp[pos][mask];
22     }
23
24     int ans = INT_MAX;
25
26     for (int city = 0; city < n; city++) {
27         if ((mask & (1 << city)) == 0) {
28             int newAns = graph[pos][city] + tsp(city, mask | (1 << city));
29             ans = min(ans, newAns);
30         }
31     }
32
33     return dp[pos][mask] = ans;
34 }
35
36 int main() {
37     printf("Enter number of cities: ");
38     scanf("%d", &n);
39
40     printf("Enter cost matrix:\n");
41     for (int i = 0; i < n; i++) {
42         for (int j = 0; j < n; j++) {
43             scanf("%d", &graph[i][j]);
44         }
45     }
46
47     for (int i = 0; i < MAX; i++) {
48         for (int j = 0; j < (1 << MAX); j++) {
49             dp[i][j] = -1;
50         }
51     }
52 }
```

```
52  
53     int result = tsp(0, 1);  
54  
55     printf("Minimum cost: %d\n", result);  
56  
57     return 0;  
58 }
```

Exp_09\minSpanningTree.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct {
5     int src, dest, weight;
6 } Edge;
7
8 typedef struct {
9     int parent, rank;
10} Subset;
11
12 int find(Subset subsets[], int i) {
13     if (subsets[i].parent != i)
14         subsets[i].parent = find(subsets, subsets[i].parent);
15     return subsets[i].parent;
16 }
17
18 void unionSets(Subset subsets[], int x, int y) {
19     int xroot = find(subsets, x);
20     int yroot = find(subsets, y);
21
22     if (subsets[xroot].rank < subsets[yroot].rank)
23         subsets[xroot].parent = yroot;
24     else if (subsets[xroot].rank > subsets[yroot].rank)
25         subsets[yroot].parent = xroot;
26     else {
27         subsets[yroot].parent = xroot;
28         subsets[xroot].rank++;
29     }
30 }
31
32 int compareEdges(const void* a, const void* b) {
33     return ((Edge*)a)->weight - ((Edge*)b)->weight;
34 }
35
36 void kruskalMST(Edge edges[], int V, int E) {
37     qsort(edges, E, sizeof(Edge), compareEdges);
38
39     Subset* subsets = (Subset*)malloc(V * sizeof(Subset));
40     for (int i = 0; i < V; i++) {
41         subsets[i].parent = i;
42         subsets[i].rank = 0;
43     }
44
45     Edge result[V];
46     int e = 0, i = 0;
47
48     printf("Edges in MST:\n");
49     while (e < V - 1 && i < E) {
50         Edge next_edge = edges[i++];
51         int x = find(subsets, next_edge.src);
```

```

52     int y = find(subsets, next_edge.dest);
53
54     if (x != y) {
55         result[e++] = next_edge;
56         printf("%d -- %d == %d\n", next_edge.src, next_edge.dest,
57               next_edge.weight);
58         unionSets(subsets, x, y);
59     }
60 }
61
62     int totalWeight = 0;
63     for (int i = 0; i < e; i++)
64         totalWeight += result[i].weight;
65     printf("Total weight of MST: %d\n", totalWeight);
66
67     free(subsets);
68 }
69
70 int main() {
71     int V, E;
72     printf("Enter number of vertices: ");
73     scanf("%d", &V);
74     printf("Enter number of edges: ");
75     scanf("%d", &E);
76
77     Edge* edges = (Edge*)malloc(E * sizeof(Edge));
78     printf("Enter edges (src dest weight):\n");
79     for (int i = 0; i < E; i++) {
80         scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
81     }
82
83     kruskalMST(edges, V, E);
84
85     free(edges);
86     return 0;
}

```

Exp_10\N_Queen.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4
5 bool isSafe(int** board, int row, int col, int n) {
6     for (int i = 0; i < col; i++)
7         if (board[row][i])
8             return false;
9
10    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
11        if (board[i][j])
12            return false;
13
14    for (int i = row, j = col; i < n && j >= 0; i++, j--)
15        if (board[i][j])
16            return false;
17
18    return true;
19}
20
21 bool solveNQueenUtil(int** board, int col, int n) {
22     if (col >= n)
23         return true;
24
25     for (int i = 0; i < n; i++) {
26         if (isSafe(board, i, col, n)) {
27             board[i][col] = 1;
28
29             if (solveNQueenUtil(board, col + 1, n))
30                 return true;
31
32             board[i][col] = 0;
33         }
34     }
35
36     return false;
37}
38
39 void printSolution(int** board, int n) {
40     for (int i = 0; i < n; i++) {
41         for (int j = 0; j < n; j++)
42             printf("%d ", board[i][j]);
43         printf("\n");
44     }
45}
46
47 bool solveNQueen(int n) {
48     int** board = (int**)malloc(n * sizeof(int*));
49     for (int i = 0; i < n; i++) {
50         board[i] = (int*)calloc(n, sizeof(int));
51     }
```

```
52
53     if (!solveNQueenUtil(board, 0, n)) {
54         printf("Solution does not exist\n");
55         for (int i = 0; i < n; i++)
56             free(board[i]);
57         free(board);
58         return false;
59     }
60
61     printSolution(board, n);
62
63     for (int i = 0; i < n; i++)
64         free(board[i]);
65     free(board);
66     return true;
67 }
68
69 int main() {
70     int n;
71     printf("Enter the value of N: ");
72     scanf("%d", &n);
73
74     solveNQueen(n);
75
76     return 0;
77 }
```

Exp_11\0_1Knapsack.c

```
1 #include <stdio.h>
2
3 int max(int a, int b) {
4     return (a > b) ? a : b;
5 }
6
7 int knapsack(int W, int wt[], int val[], int n) {
8     int dp[n + 1][W + 1];
9
10    for (int i = 0; i <= n; i++) {
11        for (int w = 0; w <= W; w++) {
12            if (i == 0 || w == 0)
13                dp[i][w] = 0;
14            else if (wt[i - 1] <= w)
15                dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1]
16 [w]);
17            else
18                dp[i][w] = dp[i - 1][w];
19        }
20    }
21
22    return dp[n][W];
23}
24
25 int main() {
26     int n, W;
27
28     printf("Enter number of items: ");
29     scanf("%d", &n);
30
31     int val[n], wt[n];
32
33     printf("Enter values: ");
34     for (int i = 0; i < n; i++)
35         scanf("%d", &val[i]);
36
37     printf("Enter weights: ");
38     for (int i = 0; i < n; i++)
39         scanf("%d", &wt[i]);
40
41     printf("Enter knapsack capacity: ");
42     scanf("%d", &W);
43
44     int maxValue = knapsack(W, wt, val, n);
45     printf("Maximum value: %d\n", maxValue);
46
47 }
```

Exp_12\Dijskstra.c

```
1 #include <stdio.h>
2 #include <limits.h>
3 #include <stdbool.h>
4
5 #define MAX 100
6 #define INF INT_MAX
7
8 int minDistance(int dist[], bool visited[], int n) {
9     int min = INF, minIndex;
10    for (int i = 0; i < n; i++) {
11        if (!visited[i] && dist[i] < min) {
12            min = dist[i];
13            minIndex = i;
14        }
15    }
16    return minIndex;
17 }
18
19 void dijkstra(int graph[MAX][MAX], int n, int src) {
20     int dist[MAX];
21     bool visited[MAX];
22
23     for (int i = 0; i < n; i++) {
24         dist[i] = INF;
25         visited[i] = false;
26     }
27
28     dist[src] = 0;
29
30     for (int count = 0; count < n - 1; count++) {
31         int u = minDistance(dist, visited, n);
32         visited[u] = true;
33
34         for (int v = 0; v < n; v++) {
35             if (!visited[v] && graph[u][v] && dist[u] != INF &&
36                 dist[u] + graph[u][v] < dist[v]) {
37                 dist[v] = dist[u] + graph[u][v];
38             }
39         }
40     }
41
42     printf("Vertex\t\tDistance from Source (%d)\n", src);
43     for (int i = 0; i < n; i++) {
44         printf("%d\t\t", i);
45         if (dist[i] == INF)
46             printf("INF\n");
47         else
48             printf("%d\n", dist[i]);
49     }
50 }
```

```
52 int main() {
53     int n, src;
54     int graph[MAX][MAX];
55
56     printf("Enter number of vertices: ");
57     scanf("%d", &n);
58
59     printf("Enter adjacency matrix (0 for no edge):\n");
60     for (int i = 0; i < n; i++) {
61         for (int j = 0; j < n; j++) {
62             scanf("%d", &graph[i][j]);
63         }
64     }
65
66     printf("Enter source vertex: ");
67     scanf("%d", &src);
68
69     dijkstra(graph, n, src);
70
71     return 0;
72 }
```

Exp_13\0_1Knapsack_DP_Greedy.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Structure to represent an item
5 typedef struct {
6     int weight;
7     int value;
8     double ratio; // value-to-weight ratio for greedy approach
9 } Item;
10
11 // Function to find maximum of two integers
12 int max(int a, int b) {
13     return (a > b) ? a : b;
14 }
15
16 // Dynamic Programming approach for 0/1 Knapsack
17 int knapsackDP(int weights[], int values[], int n, int capacity) {
18     int dp[n + 1][capacity + 1];
19
20     // Build table dp[][] in bottom-up manner
21     for (int i = 0; i <= n; i++) {
22         for (int w = 0; w <= capacity; w++) {
23             if (i == 0 || w == 0)
24                 dp[i][w] = 0;
25             else if (weights[i - 1] <= w)
26                 dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]],
27                                 dp[i - 1][w]);
28             else
29                 dp[i][w] = dp[i - 1][w];
30         }
31     }
32
33     return dp[n][capacity];
34 }
35
36 // Comparison function for qsort (descending order of ratio)
37 int compare(const void *a, const void *b) {
38     Item *item1 = (Item *)a;
39     Item *item2 = (Item *)b;
40     if (item2->ratio > item1->ratio)
41         return 1;
42     else if (item2->ratio < item1->ratio)
43         return -1;
44     return 0;
45 }
46
47 // Greedy approach for 0/1 Knapsack (approximation)
48 int knapsackGreedy(int weights[], int values[], int n, int capacity) {
49     Item items[n];
50
51     // Create items array with value-to-weight ratio
```

```

52     for (int i = 0; i < n; i++) {
53         items[i].weight = weights[i];
54         items[i].value = values[i];
55         items[i].ratio = (double)values[i] / weights[i];
56     }
57
58     // Sort items by ratio in descending order
59     qsort(items, n, sizeof(Item), compare);
60
61     int totalValue = 0;
62     int currentWeight = 0;
63
64     // Pick items based on greedy choice
65     for (int i = 0; i < n; i++) {
66         if (currentWeight + items[i].weight <= capacity) {
67             currentWeight += items[i].weight;
68             totalValue += items[i].value;
69         }
70     }
71
72     return totalValue;
73 }
74
75 int main() {
76     int n, capacity;
77
78     printf("Enter the number of items: ");
79     scanf("%d", &n);
80
81     int weights[n], values[n];
82
83     printf("Enter the weights of items:\n");
84     for (int i = 0; i < n; i++) {
85         printf("Item %d: ", i + 1);
86         scanf("%d", &weights[i]);
87     }
88
89     printf("Enter the values of items:\n");
90     for (int i = 0; i < n; i++) {
91         printf("Item %d: ", i + 1);
92         scanf("%d", &values[i]);
93     }
94
95     printf("Enter the knapsack capacity: ");
96     scanf("%d", &capacity);
97
98     // Dynamic Programming approach
99     int dpResult = knapsackDP(weights, values, n, capacity);
100    printf("\n--- Dynamic Programming Approach ---\n");
101    printf("Maximum value (DP): %d\n", dpResult);
102
103    // Greedy approach
104    int greedyResult = knapsackGreedy(weights, values, n, capacity);
105    printf("\n--- Greedy Approach ---\n");

```

```
106     printf("Maximum value (Greedy): %d\n", greedyResult);
107
108     printf("\nNote: Greedy approach may not always give optimal solution for 0/1
109     Knapsack.\n");
110
111     return 0;
112 }
```

Exp_14\Kruskal.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 30
5
6 typedef struct edge {
7     int u, v, w;
8 } edge;
9
10 typedef struct edge_list {
11     edge data[MAX];
12     int n;
13 } edge_list;
14
15 edge_list elist;
16 int Graph[MAX][MAX], n;
17 edge_list spanlist;
18
19 void kruskal();
20 int find(int belongs[], int vertexno);
21 void union1(int belongs[], int c1, int c2);
22 void sort();
23 void print();
24
25 void kruskal() {
26     int belongs[MAX], i, j, cno1, cno2;
27     elist.n = 0;
28
29     for (i = 1; i < n; i++)
30         for (j = 0; j < i; j++) {
31             if (Graph[i][j] != 0) {
32                 elist.data[elist.n].u = i;
33                 elist.data[elist.n].v = j;
34                 elist.data[elist.n].w = Graph[i][j];
35                 elist.n++;
36             }
37         }
38
39     sort();
40
41     for (i = 0; i < n; i++)
42         belongs[i] = i;
43
44     spanlist.n = 0;
45
46     for (i = 0; i < elist.n; i++) {
47         cno1 = find(belongs, elist.data[i].u);
48         cno2 = find(belongs, elist.data[i].v);
49
50         if (cno1 != cno2) {
51             spanlist.data[spanlist.n] = elist.data[i];
```

```

52         spanlist.n++;
53         union1(belongs, cno1, cno2);
54     }
55 }
56
57
58 int find(int belongs[], int vertexno) {
59     return belongs[vertexno];
60 }
61
62 void union1(int belongs[], int c1, int c2) {
63     int i;
64     for (i = 0; i < n; i++)
65         if (belongs[i] == c2)
66             belongs[i] = c1;
67 }
68
69 void sort() {
70     int i, j;
71     edge temp;
72
73     for (i = 1; i < elist.n; i++)
74         for (j = 0; j < elist.n - 1; j++)
75             if (elist.data[j].w > elist.data[j + 1].w) {
76                 temp = elist.data[j];
77                 elist.data[j] = elist.data[j + 1];
78                 elist.data[j + 1] = temp;
79             }
80 }
81
82 void print() {
83     int i, cost = 0;
84     printf("\nMinimum Spanning Tree Edges:\n");
85     for (i = 0; i < spanlist.n; i++) {
86         printf("\nEdge %d: (%d, %d) with weight %d",
87                i + 1, spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
88         cost += spanlist.data[i].w;
89     }
90     printf("\n\nTotal Cost of Spanning Tree: %d\n", cost);
91 }
92
93 int main() {
94     int i, j, total_cost;
95
96     printf("Enter number of vertices: ");
97     scanf("%d", &n);
98
99     printf("\nEnter the adjacency matrix:\n");
100    for (i = 0; i < n; i++)
101        for (j = 0; j < n; j++)
102            scanf("%d", &Graph[i][j]);
103
104    kruskal();
105    print();

```

```
106 |
107 |     return 0;
108 }
```

Exp_15\ a_Floyd.c

```
1 #include <stdio.h>
2 #include <limits.h>
3
4 #define V 4
5 #define INF 99999
6
7 void printSolution(int dist[][V]);
8
9 void floydWarshall(int graph[][V]) {
10     int dist[V][V], i, j, k;
11
12     // Initialize the solution matrix same as input graph matrix
13     for (i = 0; i < V; i++) {
14         for (j = 0; j < V; j++) {
15             dist[i][j] = graph[i][j];
16         }
17     }
18
19     // Add all vertices one by one to the set of intermediate vertices
20     for (k = 0; k < V; k++) {
21         // Pick all vertices as source one by one
22         for (i = 0; i < V; i++) {
23             // Pick all vertices as destination for the above picked source
24             for (j = 0; j < V; j++) {
25                 // If vertex k is on the shortest path from i to j,
26                 // then update the value of dist[i][j]
27                 if (dist[i][k] != INF && dist[k][j] != INF &&
28                     dist[i][k] + dist[k][j] < dist[i][j]) {
29                     dist[i][j] = dist[i][k] + dist[k][j];
30                 }
31             }
32         }
33     }
34
35     printSolution(dist);
36 }
37
38 void printSolution(int dist[][V]) {
39     printf("Following matrix shows the shortest distances between every pair of
vertices:\n");
40     for (int i = 0; i < V; i++) {
41         for (int j = 0; j < V; j++) {
42             if (dist[i][j] == INF)
43                 printf("%7s", "INF");
44             else
45                 printf("%7d", dist[i][j]);
46         }
47         printf("\n");
48     }
49 }
50
51 int main() {
```

```
52     int graph[V][V] = {  
53         {0, 5, INF, 10},  
54         {INF, 0, 3, INF},  
55         {INF, INF, 0, 1},  
56         {INF, INF, INF, 0}  
57     };  
58  
59     floydWarshall(graph);  
60  
61     return 0;  
62 }
```

Exp_15\b_TravellingSalesman_DP.c

```
1 #include <stdio.h>
2 #include <limits.h>
3
4 #define N 4 // Number of cities
5 #define INF INT_MAX
6
7 int dist[N][N]; // Distance matrix
8 int dp[1 << N][N]; // DP table: dp[mask][i] = min cost to visit cities in mask
9 ending at city i
10
11 int min(int a, int b) {
12     return (a < b) ? a : b;
13 }
14
15 // Function to solve TSP using Dynamic Programming
16 int tsp(int mask, int pos) {
17     // If all cities have been visited
18     if (mask == (1 << N) - 1) {
19         return dist[pos][0]; // Return to starting city
20     }
21
22     // If already computed
23     if (dp[mask][pos] != -1) {
24         return dp[mask][pos];
25     }
26
27     int ans = INF;
28
29     // Try visiting all unvisited cities
30     for (int city = 0; city < N; city++) {
31         // If city is not visited and there is a path
32         if ((mask & (1 << city)) == 0 && dist[pos][city] != INF) {
33             int newAns = dist[pos][city] + tsp(mask | (1 << city), city);
34             ans = min(ans, newAns);
35         }
36     }
37
38     return dp[mask][pos] = ans;
39 }
40
41 int main() {
42     // Initialize distance matrix
43     printf("Enter the distance matrix (%d x %d):\n", N, N);
44     printf("(Enter %d for no direct path)\n", INF);
45
46     for (int i = 0; i < N; i++) {
47         for (int j = 0; j < N; j++) {
48             scanf("%d", &dist[i][j]);
49         }
50     }
51     // Initialize DP table with -1
```

```
52     for (int i = 0; i < (1 << N); i++) {
53         for (int j = 0; j < N; j++) {
54             dp[i][j] = -1;
55         }
56     }
57
58     // Start from city 0 with only city 0 visited
59     int minCost = tsp(1, 0);
60
61     if (minCost >= INF) {
62         printf("\nNo valid tour exists!\n");
63     } else {
64         printf("\nMinimum cost of TSP tour: %d\n", minCost);
65     }
66
67     return 0;
68 }
```

Exp_16\HamiltonianCycles.c

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 #define MAX 20
5
6 int graph[MAX][MAX];
7 int path[MAX];
8 int n;
9 int cycleCount = 0;
10
11 void printPath() {
12     printf("Hamiltonian Cycle %d: ", ++cycleCount);
13     for (int i = 0; i < n; i++) {
14         printf("%d ", path[i]);
15     }
16     printf("%d\n", path[0]); // Return to starting vertex
17 }
18
19 bool isSafe(int v, int pos) {
20     // Check if vertex v is adjacent to the previous vertex
21     if (graph[path[pos - 1]][v] == 1)
22         return false;
23
24     // Check if vertex v is already in path
25     for (int i = 0; i < pos; i++) {
26         if (path[i] == v)
27             return false;
28     }
29
30     return true;
31 }
32
33 bool hamiltonianCycleUtil(int pos) {
34     // Base case: if all vertices are included in path
35     if (pos == n) {
36         // Check if there is an edge from the last vertex to the first vertex
37         if (graph[path[pos - 1]][path[0]] == 1) {
38             printPath();
39             return true; // Continue to find more cycles
40         }
41         return false;
42     }
43
44     // Try different vertices as the next candidate
45     for (int v = 1; v < n; v++) {
46         if (isSafe(v, pos)) {
47             path[pos] = v;
48
49             // Recur to construct rest of the path
50             hamiltonianCycleUtil(pos + 1);
51         }
52     }
53 }
```

```

52         // Backtrack
53         path[pos] = -1;
54     }
55 }
56
57 return false;
58 }
59
60 void findHamiltonianCycles() {
61     // Initialize path array
62     for (int i = 0; i < n; i++)
63         path[i] = -1;
64
65     // Start from vertex 0
66     path[0] = 0;
67     cycleCount = 0;
68
69     printf("\nFinding all Hamiltonian Cycles:\n");
70     hamiltonianCycleUtil(1);
71
72     if (cycleCount == 0) {
73         printf("No Hamiltonian Cycle exists in the given graph.\n");
74     } else {
75         printf("\nTotal number of Hamiltonian Cycles: %d\n", cycleCount);
76     }
77 }
78
79 int main() {
80     printf("Enter the number of vertices: ");
81     scanf("%d", &n);
82
83     printf("\nEnter the adjacency matrix:\n");
84     for (int i = 0; i < n; i++) {
85         for (int j = 0; j < n; j++) {
86             scanf("%d", &graph[i][j]);
87         }
88     }
89
90     printf("\nAdjacency Matrix:\n");
91     for (int i = 0; i < n; i++) {
92         for (int j = 0; j < n; j++) {
93             printf("%d ", graph[i][j]);
94         }
95         printf("\n");
96     }
97
98     findHamiltonianCycles();
99
100    return 0;
101 }
```